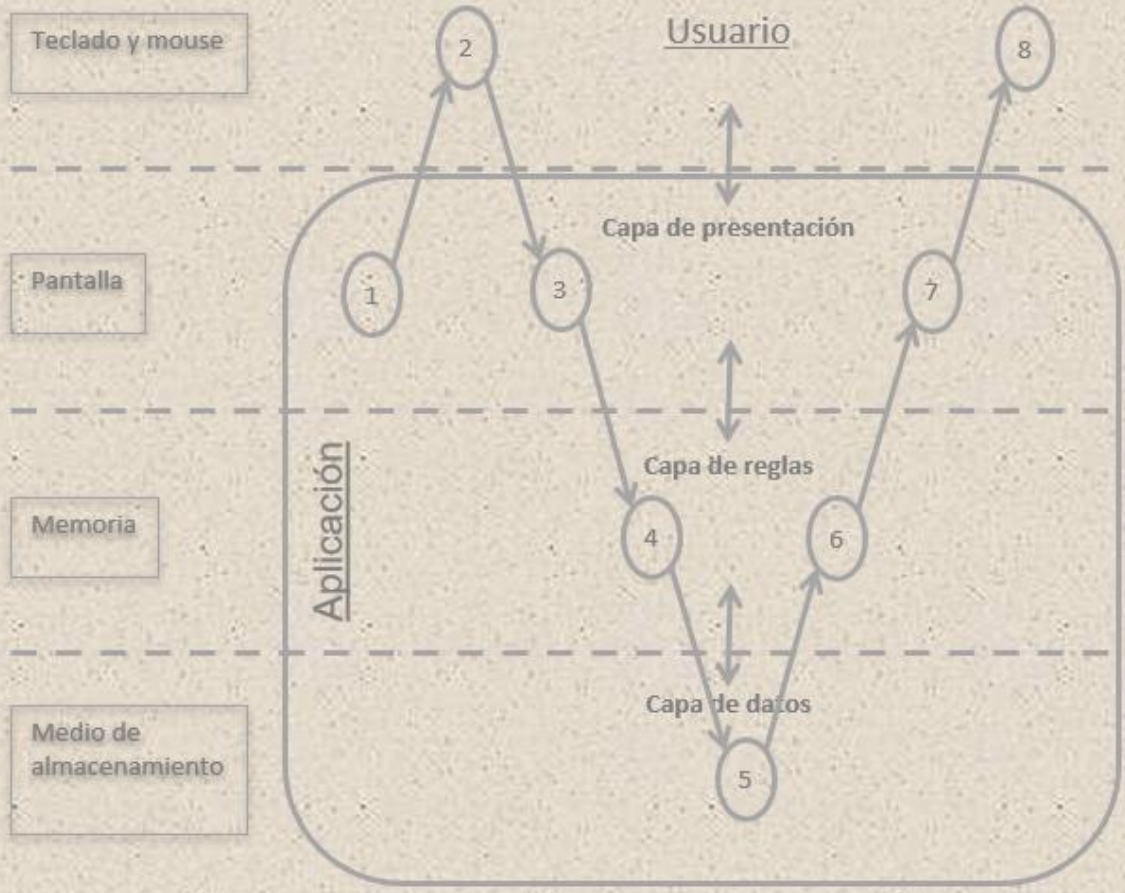
Programación Orientada a Objetos en Arquitectura de Capas

Propuesta metodológica apoyada en Prototipado Evolutivo



Ricardo Vicente Jaime Vivas
Carlos Andrés Palma Suárez
Laura Cristina Villamizar Vecino
Roy Hernando Llamas Muñoz

UDI UNIVERSIDAD DE INVESTIGACIÓN Y DESARROLLO
Unidad de Publicaciones
ISBN: 978-958-8796-19-2

Programación Orientada a Objetos en Arquitectura de Capas

Propuesta metodológica apoyada en Prototipado Evolutivo

Programación Orientada a Objetos en Arquitectura de Capas

Propuesta metodológica apoyada en Prototipado Evolutivo

Ricardo Vicente Jaime Vivas
Carlos Andrés Palma Suárez
Laura Cristina Villamizar Vecino
Roy Hernando Llamas Muñoz

Grupo de Investigación en nuevas tecnologías aplicadas a la Educación - GIDSAW
Universidad de Investigación y Desarrollo - UDI

Programación Orientada a Objetos en Arquitectura de Capas – Propuesta metodológica apoyada en Prototipado Evolutivo

Autores: Ricardo Vicente Jaime Vivas ricardojaime@udi.edu.co
Carlos Andrés Palma Suárez carlos.palma@udi.edu.co
Laura Cristina Villamizar Vecino lvillamizar6@udi.edu.co
Roy Hernando Llamas Muñoz rllamas1@udi.edu.co

ISBN: 978-958-8796-19-2
Primera edición: Diciembre de 2020

La información contenida en esta obra es resultado del trabajo de los autores como integrantes del Grupo de Investigación en nuevas tecnologías aplicadas a la Educación – GIDSAW, de la Universidad de Investigación y Desarrollo – UDI.

Calle 9 # 23-55. Bucaramanga, Santander, Colombia.
Tel.: (+57-7) 6352525
Correo electrónico: investigaciones@udi.edu.co
Sitio web: www.udi.edu.co/investigaciones/142-publicaciones

Catalogación de la publicación en la Universidad de Investigación y Desarrollo UDI

Jaime, Ricardo; Palma, Carlos; Villamizar, Laura; Llamas, Roy.
Programación Orientada a Objetos en Arquitectura de Capas – Propuesta metodológica apoyada en Prototipado Evolutivo.
Primera edición.
Bucaramanga, Unidad de Publicaciones Universidad de Investigación y Desarrollo – UDI, 2020
316 páginas: ilustraciones; 21 x 28 cm.

ISBN: Pendiente

5 - Programación. Programas. Datos de computadores
UMZW - Ingeniería del software orientado a objetos

Contenido

Prólogo

Ricardo Jaime

¿Programar es fácil? Esta ha sido una pregunta recurrente que, quienes ejercemos la docencia en Ingeniería de Sistemas, escuchamos cada vez que una nueva cohorte llega para iniciar sus estudios universitarios. Mediante esta pregunta algunos estudiantes expresan su temor por el mundo desconocido en que se han metido, tal vez con la esperanza de que una respuesta afirmativa les asegure que el desafío es asequible. Lo preguntan con timidez, porque junto a ellos hay otros, que han llegado convencidos de que los diez semestres típicos del plan de estudios son un mero trámite administrativo, pues lo que se supone que van a aprender, ya ellos lo saben y lo dominan. *¡Programar es fácil!;* no preguntan, sino que lo afirman. *¡Profe, yo desde el colegio prácticamente pienso en C!*, dijo con toda sobradez alguno de los otros, quizás veinte años atrás.

Apenas saliendo de nuestra primera sesión de clase de *Introducción a los Computadores*, en contraste con la voz de la profesora Martha Vitalia Corredor, por momentos casi inaudible, alguien más dijo a voz en cuello: *¡yo esto lo termino en tres años!* Luego se supo que ella, al mismo tiempo que orientaba nuestro primer curso de la universidad, empezaba a enfrentar el rigor de su propia etapa de formación a nivel de doctorado, consciente de lo mucho que hace falta por saber en cualquier campo del conocimiento. En contraste, quien -al igual que yo- apenas era *primíparo* en la Universidad Industrial de Santander (UIS), ya se tenía la confianza suficiente para concluir que su dominio del lenguaje de programación *Basic*, en el computador portátil *Casio PB-700* -aparato al que los más ignorantes seguíamos llamando calculadora-, ya le hacía acreedor de una de las esquinas del diploma. Él solo sabía que ya todo lo sabía.

Pareciera un recordatorio hostil con aquel excompañero de estudios, tardío y resentido por demás, si se piensa que ya han pasado treinta y tantos años. Se dice que resultó siendo un profesional competente, aunque graduarse le tomó, eso sí, bastante más de los tres años que él mismo había calculado. Esta anécdota es solo un medio para señalar que, al menos durante cuarenta años, ha tenido eco la idea de que conocer las sentencias de un lenguaje de programación es equivalente a saber programar.

Claro que programar es fácil… programar mal es facilísimo… mucha gente sabe hacerlo… yo mismo lo he hecho. Esta ha sido la primera parte de mi respuesta recurrente en semestres recientes. Con otros docentes solemos comentar las dificultades que observamos en los estudiantes en cuanto al aprendizaje de la programación, matizando con autocrítica sobre nuestros propios trabajos, los que presentamos a quienes fueron nuestros profesores, que se nos antojan deficientes ahora que también somos evaluadores. De algunos de nuestros trabajos de grado, vistos con criterios de hoy, quizás lo único que no nos da un poco de vergüenza es la

dedicatoria. Como atenuante, valga decir que para entonces la Ingeniería del Software no había llegado al currículo, y que la Programación Orientada a Objetos era algo que sucedía en tierras muy lejanas, de lo que apenas alcanzábamos a tener una reseña por cuenta del profesor Ricardo Llamosa, sobre el lenguaje *SmallTalk*.

Programar computadores bien es difícil. Bastante difícil. Así suelo completar la respuesta iniciada en el párrafo anterior. Para quienes llegan convencidos de su autosuficiencia, el primer paso sugerido para programar bien es abstruso: alejarse del teclado. Pero es necesario, si por programar bien se entiende hacerlo con el rigor a que se compromete quien estudia ingeniería: analizar y diseñar, antes de desarrollar e implementar; asumir que así un producto funcione, si no está respaldado por la traza de dicha secuencia metódica, no es de ingeniería. Se analiza para no hacer las cosas como primero se nos ocurren o como siempre las hemos hecho, sino para determinar, entre varias alternativas, cuál es la forma más conveniente de hacer las cosas en el marco de ciertas restricciones técnicas y de recursos. Se diseña para poder someter a prueba lo que se va a construir, y para solo construirlo cuando el diseño ha superado una rigurosa evaluación que garantice su viabilidad. Se analiza, se diseña, y se deja documentación de esto, para que el desarrollo y la implementación no dependan de que las mismas personas continúen a cargo, para que la obra pueda ser luego mantenida o escalada convenientemente, incluso por otros. El lenguaje de programación es solo una herramienta, y la labor de codificar un programa es solo parte de un todo mucho mayor, al que se puede denominar proyecto de ingeniería.

El proyecto detrás de este libro, ejecutado por el Grupo de Investigación en Nuevas Tecnologías aplicadas a la Educación (Gidsaw), ha involucrado también al programa de Ingeniería de Sistemas. Ambos, grupo de investigación y programa académico, han conformado un tándem ya de larga data, con una tradición que reconoce a la docencia como la función primaria de la Universidad de Investigación y Desarrollo (UDI), con la investigación al servicio de aquella y de los estudiantes. En este escenario, la era reciente del grupo de investigación ha girado en torno a proponer y probar métodos para mejorar la educación en ingeniería, y las aulas han sido el escenario de experimentación.

Además del esfuerzo académico se ha requerido el trabajo político, desde el comité curricular, para proponer reformas al plan de estudios, exigirse a fondo en la deliberación interna y en la labor de convencer a la institución, y para demostrar la coherencia de la reforma ante los pares académicos, durante el proceso que condujo a la reciente renovación del registro calificado del programa. Lo propio se hizo ante los pares de acreditación de alta calidad, solicitud que actualmente está en fase de decisión. En dicho esfuerzo académico y político se estableció que la formación en Ingeniería de Sistemas ofrecida por la UDI es una propuesta híbrida entre Ingeniería del Software, Sistemas de Información y Tecnologías de la Información, tres de las disciplinas incluidas en las recomendaciones formuladas conjuntamente por la *Association for Computing Machinery (ACM)* y el *Institute of Electrical and Electronics Engineers (IEEE)*, los dos máximos referentes internacionales para el currículo en computación.

Cuando se propuso el proyecto la pregunta de investigación estaba asociada a la posibilidad de utilizar la Programación Orientada a Objetos como paradigma de inicio del plan de estudios, en un curso denominado *Fundamentos de Programación*. La experiencia se relata de forma más

amplia en el primer capítulo pero, en resumen, aquel primer experimento generó más efectos adversos que ventajas, y debieron pasar varios semestres hasta configurar un escenario más propicio en otro curso, cuando ya la pregunta de investigación no era sobre el efecto de abordar primero un paradigma u otro, sino acerca de la ventaja que pudiera tener, y lo viable que pudiera ser, adelantar a los cursos iniciales de programación un conjunto de conceptos, metodologías y buenas prácticas, que hasta ahora solo eran abordadas en los cursos posteriores de *Ingeniería del Software*. Los resultados favorables de este nuevo enfoque influyeron en la decisión de realizar esta publicación.

Este libro no aporta nada, o casi nada, a quien esté interesado en aprender sobre sentencias, funciones, métodos, sobre las palabras específicas de un lenguaje. Para eso es más recomendable recurrir a otro tipo de textos, a manuales de referencia, o a sitios en internet. El público objetivo de este libro está conformado por los docentes de programación en el ámbito de la Ingeniería del Software. Contiene una propuesta metodológica con la que, en el escenario de experimentación, se ha conseguido que los cursos de programación dejen de estar asociados a determinados lenguajes y, en cambio, estén orientados por la confluencia de paradigmas de programación, patrones de arquitectura, y metodologías de desarrollo de software: Programación Orientada a Objetos, Arquitectura de Capas y Prototipado Evolutivo, respectivamente, en el caso del curso de *Programación I*, que es posterior al de *Fundamentos de Programación*, que continúa bajo el enfoque algorítmico.

En consecuencia, en este libro no se proponen problemas diseñados para incorporar determinadas instrucciones del lenguaje. Se optó por un único caso de estudio que, a medida que se hace más complejo, pone en evidencia la ventaja de asumir la disciplina y el rigor de la ingeniería desde las primeras jornadas de formación: análisis y diseño antes del desarrollo. El caso de estudio se extrae, además, de una situación del orden social, con lo que, no sobra mencionarlo, se ha querido establecer también un vínculo, desde el comienzo, de los estudiantes con la sociedad a través de las competencias ciudadanas. Dejamos pues, en manos de los docentes, esta propuesta metodológica, con la confianza de que replicarla con sus propios casos los ayudará a ellos, y por su intervención también a los estudiantes, en la etapa inicial de su formación en esa interesante disciplina que es la Ingeniería de Sistemas.

Agradecimientos

No queremos los autores mencionar solo a las instituciones, sin manifestar nuestra gratitud también hacia las personas que en un momento dado las conforman. Hacemos expreso reconocimiento a los docentes Alexandra Soraya Beltrán Castro, Jonnathan Alfredo Ramos Chaux, Martín Pérez Jaimes, Rafael Ricardo Mantilla Güiza, Danith Patricia Solórzano Escobar, Carlos Humberto Carreño Díaz, Ruy Fernán Ruiz Mojica, Juan Carlos García Ojeda y Wilfredo Ariel Gómez Bueno. También a la ingeniera Lina Margarita Henao, la persona y la profesional, en la Dirección de Investigaciones.

1 Contexto de la investigación

Este libro es el resultado final del proyecto de investigación ***Orientación a Objetos en Arquitectura de Capas***. Sin embargo, como muestra la Figura 1.1, este que aparece con el número 7 en la secuencia es en realidad una segunda mirada a otro proyecto anterior denominado ***Orientación a Objetos en Arquitectura de Capas en cursos de programación a nivel de pregrado***, el número 3 de la secuencia, y sobre todo es el resultado de la confluencia entre diversas áreas que el Grupo de Investigación en Nuevas Tecnologías aplicadas a la Educación (Gidsaw), asociado al programa de pregrado de Ingeniería de Sistemas de la Universidad de Investigación y Desarrollo UDI, ha abordado en una línea de investigación sobre Educación en Ingeniería que alcanza ya una tradición de 12 años.

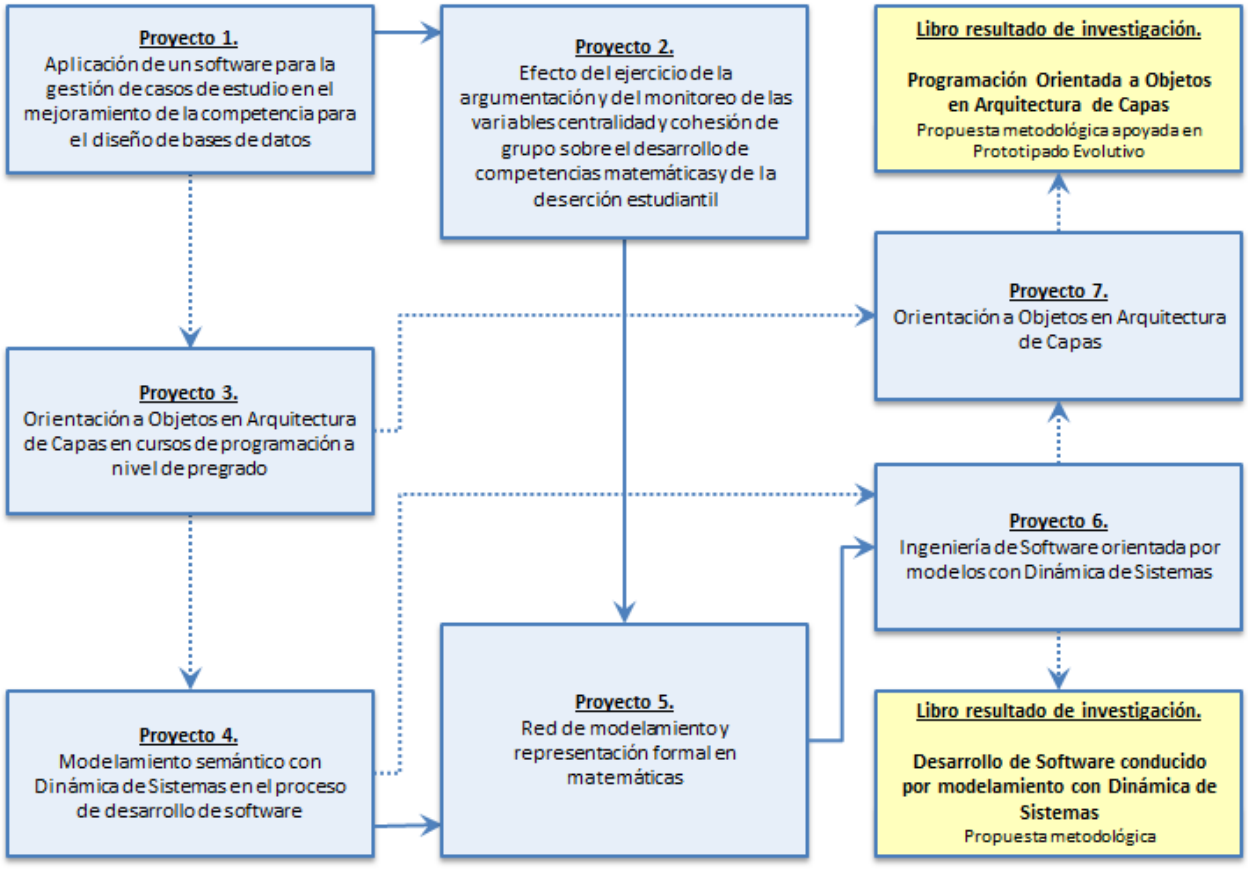


Figura 1.1 Traza de proyectos de investigación que preceden a este libro

1.1 Abordaje de casos soportado en herramientas simplificadas

Uno de los primeros problemas de investigación abordados por el grupo GIDSAW estuvo relacionado con el bajo rendimiento académico en algunos cursos del plan de estudios del programa de Ingeniería de Sistemas, además una de las principales causas de deserción. Uno de los casos más representativos de esta situación era el curso de Bases de Datos I. Se formuló entonces el proyecto ***Aplicación de un software para la gestión de casos de estudio en el mejoramiento de la competencia para el diseño de bases de datos***, desarrollado entre los años 2008 y 2010 a partir de la siguiente pregunta de investigación: ¿El uso de herramientas simplificadas tiene efecto favorable en el desarrollo de competencias de modelamiento de bases de datos relacionales?

Ya el aprendizaje del diseño de bases de datos como problema que suscitaba interés en las investigaciones sobre Educación en Ingeniería, era reportado como un campo en que las competencias de los profesionales en Ciencias de la Computación eran con frecuencia deficientes Connolly & Begg (2006), algo que entre otras causas se atribuía a una doble carga cognitiva para el estudiante: aprender a diseñar bases de datos y al mismo tiempo aprender a utilizar software propio del ámbito profesional como Oracle, PostgreSQL e incluso Microsoft Access (Bogdanovic et al., 2008). Frente a esto algunos autores recomendaban la inclusión de actividades de implementación de los diseños para efectos de comprobación por parte de los estudiantes (Abdullat, 2001), la utilización de software educativo desarrollado a partir de requisitos didácticos específicos del curso (Bogdanovic et al., 2008; Wals Zurita, 2005) y el modelamiento con datos reales sobre temas significativos para los estudiantes durante el trabajo de diseño (Chen, 2000; Chen & Ray, 2004).

Para este proyecto de investigación se desarrollaron tres versiones de un software para el modelamiento de bases de datos relacionales denominado ***MBD***, y una metodología basada en casos. Como resultado de este primer proyecto, se encontraron avances en varios aspectos: los estudiantes mejoraron su competencia para utilizar internet como fuente de información, exploraron grandes cantidades de datos, ganaron destreza en las técnicas de diseño de bases de datos relacionales y pudieron formular consultas con varias semanas de anticipación con respecto al curso tradicional (Jaime, 2009).

Desde luego, también surgieron dificultades, siendo la más significativa la dependencia que los estudiantes desarrollaron con respecto a la herramienta didáctica, por lo que durante la replicación del experimento en periodos posteriores se buscó un equilibrio entre la cantidad y complejidad de casos desarrollados con el software didáctico, y los desarrollados con los manejadores usuales en la práctica profesional, tales como Oracle o PostgreSQL.

Aunque su objetivo directo era el curso académico de Bases de Datos I, este proyecto generó conclusiones importantes acerca de la favorabilidad del aprendizaje basado en casos de estudio, de la utilización de herramientas informáticas simplificadas, y de no restringir el acceso a Internet

durante las sesiones de clase y las evaluaciones, sino elaborar estrategias para incorporarlo a la dinámica del proceso de aprendizaje incluso durante la evaluación.

El abordaje temprano de la construcción de consultas en lenguaje SQL, generó preguntas de tipo práctico por parte de los estudiantes, acerca de la manera como se implementan estas consultas ya no en una herramienta didáctica ni en un manejador de bases de datos, sino a nivel de desarrollo de aplicaciones en un lenguaje de programación; es decir, los estudiantes empezaron a requerir la integración entre lo que aprendían en el curso de Bases de Datos I y lo que estudiaban en los cursos de programación.

Estas nuevas inquietudes de los estudiantes contribuyeron a que se buscara la posibilidad de incluir nociones básicas de programación con acceso a bases de datos. Pero un obstáculo en este propósito era que las bases de datos se suelen estudiar aparte de la programación, de manera que cursos que se veían simultáneos, como Estructuras de Datos y Bases de Datos, eran percibidos como pertenecientes a líneas de estudio diferentes.

Ante la imposibilidad de destinar horas del curso de Bases de Datos I a repasar conceptos de programación, para poder introducir la conexión a bases de datos se formuló la conveniencia de que previo a este curso los estudiantes alcanzaran a trabajar en la implementación de persistencia de datos, un tema tradicionalmente tardío en los cursos de programación. Se buscó una estrategia para anticipar el aprendizaje de manejo de archivos planos en los cursos de fundamentos de programación, como precursores para que en el curso de Bases de Datos I se pudiera incluir el componente de programación con acceso a bases de datos. Una manera posible de conseguirlo era incorporar fundamentos de Programación Orientada a Objetos en Arquitectura de Capas desde el primer año, de manera que para los estudiantes fuera claro que el acceso a una base de datos es una función especializada de las clases que conforman la capa de datos, separada de las capas de reglas y de presentación.

1.2 Abordaje colectivo argumentado de casos de estudio

En el año 2010 el Ministerio de Educación Nacional, la Red Nacional Académica de Tecnología Avanzada (Renata) y el Centro de Investigación de las Telecomunicaciones (Cintel), abrieron una convocatoria para conformar un banco de proyectos elegibles para apoyar la investigación, el desarrollo y la innovación educativa.

La experiencia obtenida en el abordaje de casos de estudio le permitió al grupo Gidsaw de la UDI, en asocio con los grupos Tecnice y Tecnimat, de la Universidad Central, y el grupo Kishurim, de la Universidad Hebrea de Jerusalén, presentar un proyecto titulado ***Efecto del ejercicio de la argumentación y del monitoreo de las variables centralidad y cohesión de grupo sobre el desarrollo de competencias matemáticas y de la deserción estudiantil***. Este proyecto hizo parte de los elegibles al cierre de la convocatoria.

El objetivo general era evaluar una estrategia pedagógica para el desarrollo de competencias mediante procesos argumentativos en la solución de casos matemáticos apoyados por computador. La experimentación se orientó al desarrollo de sesiones de solución de problemas

en contexto, de forma colaborativa en línea, en grupos compuestos por estudiantes de primer semestre de ingeniería. Se introdujo el uso de categorías ontológicas en las sesiones de discusión, y se analizó la relación entre centralidad, cohesión de grupo, categoría argumentativa y el desarrollo de competencias matemáticas.

Según la definición de Gruber (1993), una ontología constituye una conceptualización formal y explícita de un conocimiento compartido. En esta definición, la conceptualización se refiere a un modelo abstracto de algún fenómeno del mundo del que se identifican los conceptos que son relevantes; explícito hace referencia a la necesidad de especificar de forma consciente los distintos conceptos que conforman una ontología; formal implica que la especificación debe representarse por medio de un lenguaje de representación formalizado; y compartido determina que una ontología debe, en el mejor de los casos, dar cuenta de conocimiento aceptado como mínimo, por el grupo de personas que deben usarla (Maldonado et al., 2012).

Aunque la población objetivo de este proyecto era distinta, la experiencia desarrollada en el proyecto ***Aplicación de un software para la gestión de casos de estudio en el mejoramiento de la competencia para el diseño de bases de datos***, en cuanto al desarrollo de actividades de aprendizaje basadas en casos de estudio, fue importante para el diseño y ejecución de la fase experimental de este nuevo proyecto, y el salto cualitativo estaba dado por el hecho de que ya el interés del ejercicio pedagógico no se circunscribía a la comparación entre las respuestas que cada individuo realizó con respecto a un caso de estudio, sino al proceso de construcción colectiva de soluciones a los casos, y a los argumentos y dinámicas mediante los cuales cada uno de los estudiantes trata de convencer a los otros durante la construcción de la solución.

La experimentación en la UDI se efectuó con dos grupos de estudiantes del curso de matemáticas de primer semestre de Ingeniería. Un grupo participó en las sesiones de discusión de casos; el otro grupo fue de control.

Los resultados finales del proyecto fueron un tanto paradójicos. Por una parte, entre la media de las notas del grupo experimental y del grupo de control no hubo diferencia significativa; pero, por otra parte, el número de sesiones en que los estudiantes participaban, y la cantidad de vínculos directos que establecían durante las mismas, resultaron ser muy buenos predictores de las notas finalmente obtenidas. Con base en esto se concluyó que, en términos generales, el ejercicio de la argumentación en el abordaje de casos en un contexto de aprendizaje colaborativo soportado por computador tiene una incidencia favorable en el éxito académico y el desarrollo de competencias matemáticas. Pero se dejó establecido también que varios elementos del diseño metodológico, relacionados con el tipo de software a utilizar, el tipo y número de las categorías ontológicas, y la metodología de acompañamiento por parte de los docentes como moderadores de las sesiones de argumentación, requerirían de nuevos experimentos para poder afirmar o negar su impacto favorable en el aprendizaje (Jaime et al., 2012).

1.3 Programación Orientada a Objetos en Arquitectura de Capas

A la par con el proyecto sobre abordaje colectivo de casos de estudio que se desarrolló en colaboración con grupos de investigación de otras instituciones y financiación externa, un nuevo

proyecto realizado con recursos propios de la UDI, denominado ***Orientación a Objetos en Arquitectura de Capas en cursos de programación a nivel de pregrado***, formuló una pregunta de investigación para retomar las cuestiones que sobre la enseñanza de la programación había generado el proyecto sobre bases de datos: ¿Se puede utilizar Programación Orientada a Objetos y Arquitectura de Capas para mejorar la competencia de implementación de bases de datos relacionales?

Introducir Programación Orientada a Objetos en el curso de Fundamentos de Programación, sin pasar por la etapa de programación secuencial o estructurada, era ya romper con una tradición arraigada. Proponer nociones de Arquitectura de Capas suponía además reflexionar si era prudente adelantar en casi 2 años el estudio de un tema hasta ahora reservado para los cursos de Ingeniería del Software, y se corría el riesgo de sobrecargar a los estudiantes que apenas iniciaban su plan de estudios.

Acerca de la conveniencia de introducir técnicas y conceptos de Ingeniería del Software en los cursos básicos de programación, Towhidnejad & Salimi (1996) reportan resultados satisfactorios al introducir técnicas de *Personal Software Process (PSP)* en los dos primeros cursos de programación, en un intento por incorporar métodos de desarrollo disciplinado de software desde el inicio de la carrera de Ciencias de la Computación.

Westin & Nordström (2004) afirman que no se cuenta con textos apropiados, y que los entornos de desarrollo usuales en lenguajes como Java son a su vez demasiado complejos para principiantes; una problemática similar a la ya comentada con respecto al campo de las Bases de Datos.

Este proyecto alcanzó un grado de dificultad superior al que se había estimado, pues implicaba cambios de enfoque de programación estructurada a Programación Orientada a Objetos, cambio de textos, de entornos de desarrollo, y sobre todo dejar de lado la aproximación algorítmica a la programación por la concepción de arquitectura de software, más propio de los cursos avanzados de Ingeniería del Software.

1.4 Modelamiento con Dinámica de Sistemas

La Dinámica de Sistemas consiste en la formulación, prueba y depuración de explicaciones de las causas internas del comportamiento de un sistema, para el desarrollo de políticas de manejo y toma de decisiones, haciendo uso de mapas informales, modelos formales y simulaciones computarizadas que representan el origen endógeno del comportamiento del sistema (Richardson, 2011).

Esta disciplina ha tenido una presencia tradicional en el plan de estudios de Ingeniería de Sistemas en la UDI, como metodología de modelamiento para la simulación. Sin embargo, por el perfil del egresado y el tipo de labores que caracteriza a esta profesión en el área de influencia de la institución, la simulación es un área que ha perdido interés.

El proyecto anterior dejó abierta una pregunta: ¿Se puede utilizar la Dinámica de Sistemas como técnica de modelamiento en el diseño de clases, en un contexto de Programación Orientada a Objetos y Arquitectura de Capas?

Esta pregunta, junto al interés por encontrar una nueva aplicación de la Dinámica de Sistemas diferente a la simulación, generó el contexto propicio para un nuevo proyecto denominado ***Modelamiento semántico con Dinámica de Sistemas en el proceso de desarrollo de software***, en el que se optaba por una tendencia menos ortodoxa de modelamiento cualitativo, en la que los diagramas propios de la metodología tradicional se utilizan para ayudar a las personas a externalizar y enriquecer sus modelos mentales (Wolstenholme, 1999), orientar discusiones, explicar el comportamiento de los sistemas a partir de estructuras cíclicas, ampliar el contexto de abordaje de un problema, y facilitar el paso entre el lenguaje natural y el lenguaje matemático (Coyle, 2000).

Partiendo de los conceptos de atributo y método, fundamentales en orientación a objetos, siendo los primeros características de los objetos y los segundos eventos de cambio de dichas características, se exploró la posibilidad de hacer corresponder los conceptos atributo y método con los de variable de estado y variable de flujo respectivamente, que son fundamentales en el uso clásico de la Dinámica de Sistemas para la simulación, y luego con los conceptos de variable y derivada propios del cálculo diferencial, que hace parte de la base matemática de la Dinámica de Sistemas, para finalmente hacerlos corresponder también con los conceptos de sustantivo y verbo, no en el sentido gramatical sino en sentido estricto matemático. Con esto se estableció un puente entre la orientación a objetos y el lenguaje natural, que ahora habría que recorrer en sentido inverso, mediante una metodología resultado del proyecto de investigación (Jaime, 2012).

La metodología resultante fue incorporada a los cursos de Dinámica de Sistemas que pasaron de estar orientados a la simulación por computador, y se convirtieron en cursos de matemáticas para la Ingeniería de Sistemas, con el propósito de impactar favorablemente el desarrollo de competencias en diseño y desarrollo de software, experiencia a partir de la cual el curso fue seleccionado para integrarse a un nuevo proyecto de investigación.

1.5 Modelamiento en matemáticas

Finalmente, en este punto convergen el interés por mejorar el diseño metodológico del proyecto anterior, el indicio de la incidencia favorable del trabajo colaborativo soportado en computador, la necesidad de seguir buscando soluciones a los problemas de logro académico en el área de las matemáticas, y la expectativa de transformar el curso de Dinámica de Sistemas de estar orientado a las matemáticas para la simulación, a enfocar su interés en el modelamiento matemático para la mejora de las competencias de diseño en el desarrollo de software orientado a objetos.

Llegando al cierre del proyecto sobre argumentación, en colaboración con grupos externos, que coincidió también con el cierre del proyecto con Dinámica de Sistemas, al equipo del proyecto anterior se unió el grupo de investigación Cognitek, de la Universidad Pedagógica Nacional, y de

nuevo se obtuvo un resultado favorable en la convocatoria de Renata y Cintel del año 2011. Esta vez el proyecto se denominó ***Red de modelamiento y representación formal en matemáticas***.

En cuanto a resultados, esta vez el grupo experimental aventajó al grupo de control, siendo las medias de las notas del último examen del semestre 4,07 y 3,28 respectivamente, lo que en términos cuantitativos es una diferencia del 24% a favor del grupo de control. Dado que la única diferencia entre el grupo experimental y el de control fue la mediación del software, se pudo atribuir un efecto favorable de la utilización de software para la mediación en el abordaje de casos de estudio en trabajo colaborativo, en el logro de competencias de modelamiento matemático con Dinámica de Sistemas (Jaime & Lizcano, 2015).

1.6 Ingeniería del Software orientada por modelos

Dos circunstancias de la dinámica en el programa de Ingeniería de Sistemas en la UDI propiciaron el contexto para la fase final de elaboración de los dos libros que se muestran como resultados del proceso de investigación en la **¡Error! No se encuentra el origen de la referencia.**.

Por una parte, la Ingeniería de Sistemas, como sucede en general con las disciplinas profesionales del área de Tecnologías de la Información, atraviesa por un periodo crítico de baja demanda de admisión y deserción académica más elevada que la media de los programas universitarios de pregrado en Colombia. Por otra parte, considerando el contexto regional de actuación de sus egresados, la UDI ofrece el programa de Ingeniería de Sistemas como un híbrido entre 3 de las 5 disciplinas que conforman el currículo propuesto por la *Association for Computing Machinery ACM* y el *Institute of Electrical and Electronics Engineers IEEE*, las dos instituciones internacionales de referencia en cuanto a currículo en el campo de la computación: Ingeniería del software, Sistemas de información y Tecnología de la información. De la evolución reciente de estas tres disciplinas ha surgido la necesidad de que el plan de estudios migre del énfasis en la programación al énfasis en las arquitecturas implicadas en un proyecto de desarrollo de software: arquitectura de datos, de software, del negocio y arquitectura organizacional.

Por iniciativa de los docentes del programa se abordó la formulación de una propuesta de reforma curricular, de la cual resultó beneficiado el proyecto sobre Dinámica de Sistemas, pues de la discusión curricular emergió la propuesta de formalizar el método de obtención de diagramas de clases y diagramas relacionales, para depurarla y presentarla como una metodología de transformación de los modelos de influencias construidos con Dinámica de Sistemas, en el marco de lo que se conoce como Desarrollo de Software orientado por Modelos MDD (*Model Driven Development*). La metodología debía tener la capacidad de definir transformaciones de modelos de alto nivel de abstracción del sistema siguiendo un conjunto de reglas de transformación (Herrero & Carmona, 2013).

Se decidió entonces que, en paralelo con el estudio de la reforma curricular, el grupo GIDSAW llevara a cabo un proyecto de investigación que se denominó ***Ingeniería de Software orientada por modelos con Dinámica de Sistemas***, cuyo objetivo era formalizar la metodología de transformación de modelos con Dinámica de Sistemas en diagramas de análisis y diseño de software; de manera que los diagramas de clases y diagramas relacionales que conforman el

análisis y diseño de los sistemas, que luego mediante técnicas de Ingeniería del Software se depuraran en diagramas de diseño de software, heredaran el rigor matemático con que se validan los diagramas de influencias a partir de los cuales son obtenidos mediante un proceso sistemático de transformación.

Comúnmente el proceso de análisis y diseño de software se lleva a cabo en ambiente de deliberación y negociación entre usuarios y desarrolladores; estos últimos utilizan Lenguaje Unificado de Modelamiento UML para construir las representaciones necesarias del sistema. Diversos autores, entre ellos Tignor (2004), señalan que ninguno de los modelos de UML representa la estructura ni el comportamiento de un sistema, por lo que la Dinámica de Sistemas es más conveniente para el modelamiento del problema. A lo anterior se agrega que los diagramas UML, como el diagrama de clases o el diagrama entidad relación, carecen de recursos matemáticos para la validación de su consistencia. La propuesta metodológica busca que los modelos UML puedan ser considerados matemáticamente consistentes, en razón de ser obtenidos mediante transformación sistemática de modelos en Dinámica de Sistemas matemáticamente validados (Jaime, 2015).

Es preciso mencionar que el proceso de modelamiento dinámico sistémico tiene un valor adicional como recurso conversacional, con potenciales consecuencias favorables en el resultado de la discusión entre usuarios y desarrolladores de software. Partiendo del hecho de que el desarrollo de software está asociado a la intención de intervenir en un sistema problemático, para optimizarlo o controlarlo, el modelamiento con Dinámica de Sistemas permite ayudar a los grupos a construir conocimiento compartido acerca del problema o situación a intervenir, a ampliar gradualmente la complejidad del sistema considerada en la discusión, a enfocarse en la forma en que los procesos internos del sistema funcionan juntos (Peterson, 2010). Estas características proveen una estructuración que con frecuencia hace falta en la interacción entre las partes involucradas en el proceso de desarrollo de software. La conclusión resultante de todo este proceso fue publicada bajo el título ***Desarrollo de Software conducido por modelamiento con Dinámica de Sistemas*** (Jaime Vivas et al., 2018).

1.7 Orientación a Objetos en Arquitectura de Capas

Aún dentro del contexto de la reforma curricular mencionada en el apartado anterior, y de la cohesión entre el grupo Gidsaw y el programa de Ingeniería de Sistemas UDI, se revisó metodológicamente el proyecto inicial, reformulado ahora como ***Orientación a Objetos en Arquitectura de Capas***, con una nueva fase experimental llevada a cabo entre 2014 y 2015.

Del proyecto anterior se habían obtenido conclusiones que, si bien no descartaban de forma absoluta la conveniencia de llevar la orientación a objetos al primer curso, el de Fundamentos de Programación, dejaban ver que, en el contexto del programa de Ingeniería de Sistemas de la UDI, lo obtenido en orientación a objetos tenía un alto costo en cuanto al desarrollo de la lógica de programación por parte de los estudiantes. Este costo quizás pudiera ser aceptable desde la perspectiva de investigación, pero no desde la del proyecto educativo del programa, lo que resultaba en una necesaria discusión política al interior del comité curricular, en cuyo interior se llegó a un acuerdo: mantener el enfoque algorítmico en el curso de Fundamentos de

Programación, pero retomar la investigación acerca de la incorporación de la Orientación a Objetos, con la Arquitectura de Capas ya no solo como recurso metodológico sino como pauta de diseño de objetos a partir de la especialización de roles de las clases en el software.

El propósito de los cursos introductorios a la programación, como parte de la formación de pregrado en computación - Ingeniería de Sistemas para el caso del proyecto que da origen a este libro -, ha sido objeto de permanente debate. Si bien durante décadas dominó un modelo basado en algoritmia y programación estructurada, la Programación Orientada a Objetos pasó de ser un tópico especializado a un paradigma cada vez con mayor presencia en el contexto académico, impulsado por la consolidación del lenguaje de programación Java como uno de los más utilizados en la industria de software (Hu, 2004), tal como había sucedido años atrás con la programación estructurada y el auge del lenguaje Pascal (Sajaniemi & Hu, 2006).

Como consecuencia de esta activa discusión académica, incluso otras tendencias han reclamado la conveniencia de su incorporación como estrategia de implementación de los cursos introductorios a la programación. En *Computing Curricula 2001* se reconocían 6 estrategias que ya habían alcanzado algún grado de éxito y madurez, cada una acompañada de una secuencia de dos o tres cursos que ya había sido probada para su implementación que se muestra en la Tabla 1 1 (ACM & IEEE, 2001).

Antes que conducir a un consenso para que una de las tendencias se imponga sobre las otras en los cursos introductorios de programación, ACM e IEEE en su última revisión del currículo en computación enriquecen el debate con otros elementos importantes en la formación, tales como el proceso de desarrollo de software, la calidad del software, el desempeño del profesional, la seguridad de las aplicaciones, el profesionalismo, entre otros. Es de destacar que lejos de estandarizar el currículo o forzar dicho consenso, ACM e IEEE formulan criterios de selección del paradigma para los cursos introductorios dependiendo de los propósitos del plan de estudio y los perfiles de egreso.

Una razón para hacer énfasis inicial en programación es que esta es una habilidad esencial para cualquier profesional en computación, y que solo después de adquirirla puede definir una línea de desempeño.

La tradición de *Algortihms-first* e *Imperative-first* son similares en cuanto proponen el estudio de expresiones, estructuras de control, procedimientos y funciones, como temas iniciales en la formación del profesional en computación, diferenciándose en que la primera se mantiene al margen de cualquier lenguaje o paradigma de programación, mientras la segunda suele inclinarse por la programación estructurada; pero ya en *Computing Curricula 2001* se señalaba como una de las debilidades de estas tendencias el aplazamiento para un segundo o tercer curso del plan de estudios, de los conceptos de orientación a objetos (ACM & IEEE, 2001), algo inconveniente teniendo en cuenta la preponderancia cada vez mayor de este paradigma tanto en la academia como en la industria (Donchev & Todorova, 2008; Hu, 2004; Johnson & Moses, 2008; Sajaniemi & Hu, 2006).

Tabla 1-1 Estrategias de introducción a la computación

Estrategia	Secuencias de cursos sugerida en (ACM & IEEE, 2001)	
Imperative-first	Fundamentos de Programación	Conceptos de programación estructurada, tipos de datos, estructuras de control, funciones, arreglos, archivos, ejecución, pruebas y depuración
	Paradigma Orientado a Objetos	Introducción de conceptos de Programación Orientada a Objetos a partir de programación estructurada. Definición y uso de clases, fundamentos de diseño orientado a objetos
	Estructuras de Datos y Algoritmos	Recursividad, estructuras de datos fundamentales (colas, pilas, listas enlazadas, hash tables, árboles, grafos), análisis de algoritmos
Objects-first	Introducción a la Programación Orientada a Objetos	Conceptos fundamentales de programación desde la perspectiva de orientación a objetos. Tipos de datos, estructuras de control, arreglos, algoritmos, principios y buenas prácticas de Ingeniería del Software
	Objetos y Abstracción de Datos	Diseño de objetos, interfaces hombre-computador, gráficas, ingeniería del software
	Algoritmos y Estructuras de Datos	Algoritmos, estructuras de datos, ingeniería del software
Functional-first	Introducción a la Programación Funcional	Abstracción procedural, recursividad, abstracción de datos, algoritmos y solución de problemas, estrategias algorítmicas, teoría de la computación, generalidades de lenguajes de programación
	Objetos y Algoritmos	Constructos básicos de programación, programación orientada a objetos, estructuras de datos fundamentales, programación concurrente y orientada a eventos, uso de API
Breadth-first	Introducción a las Ciencias de la Computación	Generalidades de matemáticas discretas, constructos fundamentales de programación, algoritmos y solución de problemas, estructuras de datos fundamentales, recursividad, análisis básico de algoritmos, algoritmos fundamentales en computación
	Algoritmos y Técnicas de Programación	Matemáticas discretas, lógica básica, técnicas de prueba. constructos fundamentales de programación, estructuras de datos fundamentales, programación orientada a objetos, algoritmos fundamentales en computación, lenguajes de programación
	Principios de Diseño Orientado a Objetos	Programación orientada a objetos con énfasis en estructuras de datos, interacción entre algoritmos y programación, principios de diseño orientado a objetos.
Algorithms-first	Introducción a los Algoritmos y las Aplicaciones	Algoritmos y solución de problemas, recursividad, constructos fundamentales de programación, estructuras fundamentales de datos, introducción a la POO, algoritmos fundamentales en computación, análisis básico de algoritmos
	Metodología de la Programación	Programación Orientada a Objetos, programación conducida por eventos, estrategias algorítmicas, algoritmos fundamentales
Hardware-first	Introducción a los computadores	Representación de datos a nivel de máquina, lógica digital, organización de computadores, algoritmos y solución de problemas, constructos fundamentales de programación
	Técnicas de Programación Orientada a Objetos	Algoritmos y solución de problemas, programación orientada a objetos, estructuras de datos, programación concurrente y conducida por eventos, uso de API, análisis básico de algoritmos

Por otra parte persisten críticas como: el ocultamiento inicial de otros tópicos y el refuerzo de una asociación común entre computación y programación como sinónimos; el aplazamiento de

conceptos teóricos hasta fases avanzadas de los planes de estudio en las que ya el estudiante llega a considerarlos innecesarios; simplificación excesiva del proceso de programación para hacerlo accesible a los novatos, en desmedro de las competencias de análisis, diseño y validación; generación de la sensación de desventaja insuperable por parte de los novatos que no han tenido experiencias previas en computación, e ilusión de seguridad entre quienes sí la han tenido y que los lleva a dar continuidad a malos hábitos ya adquiridos (ACM & IEEE, 2013).

Permanece sin cerrar la discusión académica acerca del lugar apropiado para programación en el nivel introductorio de las carreras en computación; desde las recomendaciones curriculares de 1978 se incluyó el curso de introducción a la programación, como alternativa al de introducción a la computación que se había definido en las recomendaciones del año 1968.

Con respecto a las estrategias de introducción a la computación, las recomendaciones curriculares actualizadas en 2013 dejan espacio abierto a la investigación, la diversificación, e incluso a desenfocar el curso de la programación para dar cobertura a tópicos como calidad, profesionalismo, desarrollo de software, entre otros, necesarios para generar un contexto más propicio para la programación entre los novatos, ya que es notorio entre ellos que una inmersión prematura de la programación, ha llevado a una visión muy estrecha de su importancia en el escenario completo de la computación (ACM & IEEE, 2013).

1.8 Estructura del libro

Concluida esta introducción, los capítulos restantes se orientan por la secuencia recomendada de conceptos, y luego por la complejidad creciente tenida en cuenta para la determinación de los prototipos.

El capítulo 2, denominado *El software como producto de ingeniería*, tiene como propósito establecer como escenario del resto del libro, y de los cursos de programación que acojan la metodología, que la visión tradicional sobre programación ha dado paso a la visión del software, pero sobre todo al abordaje desde la disciplina de la ingeniería, que supone seguir rigurosamente las fases de análisis y diseño, antes de proceder a la fase de implementación, es decir, codificación, que es con la que suele estar familiarizado el estudiante novato.

En el capítulo 3, denominado *Solución Monolítica y Solución Distribuida*, se establece una comparación entre estos dos enfoques para establecer que, como en toda ingeniería, existen diversas formas de resolver los mismos problemas, y que, si se considera un proyecto de desarrollo de cierta complejidad, en el ámbito de la Ingeniería del Software el desarrollo de soluciones distribuidas se acerca a las características de escalabilidad. Con esto se pretende la introducción de conceptos y buenas prácticas de Ingeniería del Software en los cursos iniciales de programación.

En el capítulo 4 se denomina *Introducción a la Arquitectura de Capas*. Su propósito es ofrecer una primera aproximación desde la perspectiva del usuario del a las diferentes funciones que se desarrollan durante la ejecución de un software, y la distribución que puede hacerse entre las funciones de procesamiento, presentación y persistencia. Se presenta como ejemplo una

aplicación real disponible en el portal web de una institución gubernamental, la que luego se replica en dos pequeñas aplicaciones de escritorio diferenciadas en su capa de presentación, acompañados de diagramas para la identificación de las funciones y su asociación con las capas de reglas, presentación y datos, para las cuales se formulan las primeras definiciones.

Este no es un libro texto, sino una propuesta metodológica. En efecto, en los siguientes capítulos no se resuelven muchos ni muy novedosos problemas. El capítulo 5, denominado *Formulación de un caso de estudio*, plantea una situación de referencia para proponer la idea de que a partir de una situación organizacional se puede formular no solo un escenario continuo de requisitos de programación, sino también una oportunidad de abordar la formación en competencias ciudadanas en los estudiantes de Ingeniería de Sistemas. Como ya se dijo, este no es un libro texto para un curso de programación, ni tampoco un manual de referencia del lenguaje de programación. Como propuesta metodológica, pretende que los docentes formulen sus propios casos de estudio y la gradualidad para desarrollar en sus cursos.

El capítulo 6, denominado *Punto de partida*, propone la identificación de una situación de requerimientos mínimos dentro del caso de estudio, para un primer prototipo del software. En este sentido, la metodología de prototipado evolutivo, una de las de uso frecuente en la Ingeniería del Software, confiere también ventajas pedagógicas como la de establecer niveles cada vez más complejos de requerimientos, con los que se pueda ofrecer al estudiante continuamente la situación de terminación de un producto funcional. Esta idea de gradualidad y complejidad creciente se continúa con los restantes capítulos: el número 7, denominado *Mejoras básicas*; el octavo, llamado *Persistencia de datos*, con el que se completa la arquitectura de 3 capas; el noveno, denominado *Consistencia de los datos*, con el que ya no se avanza en aspectos de funcionalidad sino de seguridad; y el capítulo final de esta entrega, llamado *Arreglos de datos*, con el cual se establece como meta ya no el procesamiento de datos de objetos individuales, sino el análisis de conjuntos de datos y alguna leve introducción al concepto de sistemas de información.

Referencias

Abdullat, A. A. (2001). Teaching A Database Systems Design Course : Is It Theory Or Practice? *2001 Information Systems Educators Conference Proceedings*.

Alagic, S. (2017). *Software Engineering: Specification, Implementation, Verification* (1st ed.). Springer International Publishing.

Asociación Colombiana de Ingeniería Sísmica. (2014). *Norma Colombiana de Diseño de Puentes CCP14*.

Association for Computing Machinery ACM, & Institute of Electric and Electronic Engineers - Computer Society IEEE-CS. (2001). *Computing Curricula 2001 Computer Science*. https://doi.org/10.1145/384274.384275

Association for Computing Machinery ACM, & Institute of Electric and Electronic Engineers - Computer Society IEEE-CS. (2013). *CS 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. https://doi.org/10.1145/2534860

Bogdanovic, M., Stanimirovic, A., Davidovic, N., & Stoimenov, L. (2008). The development and usage of a Relational Database design tool for educational purposes. *Proceedings of the Informing Science & IT Education Conference 2008*.

Chemuturi, M. (2013). *Requirements Engineering and Management for Software Development Projects* (1st ed.). Springer.

Chen, C. (2000). Using realistic business data in teaching business problem solving. *Information Technology, Learning and Performance Journal*, *18*(2), 41–50.

Chen, C., & Ray, C. (2004). The systematic approach in teaching Database Applications: is there transfer when solving realistic business problems ? *Information Technology, Learning and Performance Journal*, *22*(1), 9–21.

Connolly, T. M., & Begg, C. E. (2006). A Constructivist-Based approach to teaching Database Analysis and Design. *Journal of Information Systems Education*, *17*(1), 43–53.

Coyle, G. (2000). Qualitative and quantitative modelling in system dynamics : some research questions. *System Dynamics Review*, *16*(3), 225–244.

Donchev, I., & Todorova, E. (2008). Object-Oriented Programming in Bulgarian universities' Informatics and Computer Science curricula. *Informatics in Education*, *7*(2), 159–172. http://www.scopus.com/inward/record.url?eid=2-s2.0-60249087790&partnerID=40&md5=e2bd39a20c7680c56022c0313182905a

García Gamallo, A. M. (1997). *La evolución de las cimentaciones en la historia de la arquitectura, desde la prehistoria hasta la primera Revolución Industrial* (1st ed.). Universidad Politécnica de Madrid.

Godoy, P., & Sánchez, C. (2009). *Diseño y construcción de una Máquina Automática para la fabricación de prefabricados de hormigón* [Escuela Superior Politécnica de Chimborazo]. http://dspace.espoch.edu.ec/bitstream/123456789/40/1/15T00415.pdf

Gruber, T. R. (1993). A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, *5*(2), 199–220.

Herrero, J., & Carmona, P. (2013). A model-driven approach to develop high performance web applications. *The Journal of Systems & Software*, *86*(12), 3013–3023.

Hu, C. (2004). Rethinking of teaching Objects-First. *Education and Information Technologies*, *9*(3), 209–218. https://doi.org/10.1023/B:EAIT.0000042040.90232.88

International Organization for Standardization (ISO). (2011). *ISO/IEC 25010:2011 Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models*.

Jaime, R. (2009). An Educational Software Supported Constructivist Methodology for Data Modeling and Database Design Skills Improvement. *Proceedings of 2009 Information Systems Education Conference Isecon*, *26*(Washington DC), 1–7. http://proc.isecon.org/2009/4144/ISECON.2009.Jaime.pdf

Jaime, R. (2012). Modelamiento Semántico con Dinámica de Sistemas en el proceso de desarrollo de software. *RISTI Revista Ibérica de Sistemas y Tecnologías de La Información*, *10*, 19–34.

Jaime, R. (2015). System Dynamics for a mathematical modeling approach in Software Engineering and Information Systems. *33rd International Conference of the System Dynamics Society*, 1494–1507.

Jaime, R., & Lizcano, A. (2015). ICT mediated collaborative work in system dynamics learning. *DYNA*, *82*(189), 59–67.

Jaime, R., Lizcano, A., Muñoz, O., & Gutiérrez, J. (2012). Análisis de redes y desarrollo de competencias matemáticas. *Memorias Del 11° Congreso Iberoamericano de Informática Educativa*.

Jaime Vivas, R. V., Palma Suárez, C. A., Beltrán Castro, A. S., & Solórzano Escobar, D. P. (2018). *Desarrollo de Software conducido por modelamiento con Dinámica de Sistemas* (1st ed.). Universidad de Investigación y Desarrollo UDI.

Johnson, R. A., & Moses, D. R. (2008). Objects-First Vs. Structures-First approaches to OO Programming education: an empirical study. *Academy of Information & Management Sciences Journal*, *11*(2), 95–102. http://search.ebscohost.com/login.aspx?direct=true&profile=ehost&scope=site&authtype=crawler&jrnl=15325806&AN=41336973&h=3gRlWDpQBv/2fNGZ0MlbW2ypiy+80RcctN15qCi2CPHHN/xLGU/j0ChH+eiK5U1Z3TLVae8383OXLd+Jr+Bg4g==&crl=c

Joyanes Aguilar, L. (2008). *Fundamentos de programación. Algoritmos, estructura de datos y objetos* (1st ed.). McGraw-Hill/Interamericana de España, S. A. U.

Kniberg, K. (2007). *Scrum y XP desde las trincheras. Cómo hacemos Scrum.* (1st ed.). C4Media Inc.

Maldonado, L. F., Serrano Iglesias, E., Rodríguez, M., & Jaime, R. (2012). Gestión de redes de aprendizaje: revisión de investigaciones. *Revista de Ciencia, Educación, Innovacion y Cultura Apoyadas Por Redes de Tecnología Avanzada*, *2*(3), 145–163.

Peterson, S. (2010). Systems thinking for anyone: practices to consider. In J. Richmond, L. Stuntz, K. Richmond, & J. Egner (Eds.), *Tracing connections - Voices of Systems Thinkers* (1st ed., pp. 30–51). ISEE Systems & The Creative Learning Exchange.

Pressman, R. (2012). *Ingeniería del Software* (5th ed.). McGraw Hill.

Richardson, G. P. (2011). *The field of System Dynamics*. System Dynamics Society - What Is SD? http://www.systemdynamics.org/what-is-s/

Rincón Bermúdez, R. D. (2014). *Aseguramiento de la calidad en el diseño del software*. Universidad EAFIT.

Sajaniemi, J., & Hu, C. (2006). Teaching Programming: going beyond "Objects First." *18th Workshop of the Psychology of Programming Interest Group*, 255–265. https://doi.org/10.1145/1026487.1008014

Tignor, W. (2004). System Engineering and System Dynamics models. In M. Kennedy, G. Winch, R. Langer, J. Rowe, & J. Yanni (Eds.), *Proceedings of the 22nd International Conference of the System Dynamics Society* (Vol. 34, Issue 12, p. 110). System Dynamics Society. https://doi.org/10.1093/carcin/bgt388

Towhidnejad, M., & Salimi, A. (1996). Incorporating a disciplined software development process into

introductory Computer Science Programming courses: initial results. *Proceedings of Frontiers in Education Conference*, 497–500.

Wals Zurita, I. (2005). *Herramienta Web para la enseñanza de Bases de Datos* (1st ed.). Escuela Técnica Superior de Ingeniería Informática. Universidad de Sevilla.

Westin, L. K., & Nordström, M. (2004). Teaching OO Concepts - A new approach. *Proceedings of Frontiers in Education Conference*, 6–11.

Wolstenholme, E. (1999). Qualitative vs Quantitative Modelling : the evolving balance. *The Journal of the Operational Research Society*, *50*(4), 422–428.

2 El software como producto de ingeniería

Desde hace más de 3000 años, el hombre comenzó a crear máquinas de toda clase y emplearlas en diferentes actividades para realizar labores con menor esfuerzo y mayor eficacia o rapidez. Las máquinas fueron utilizadas en diversos campos como la manufactura, la construcción y la guerra, aplicando conocimientos técnicos para inventarlas, desarrollarlas y mejorarlas por medio del aprovechamiento de diferentes recursos. Esto inició la aplicación de la ingeniería[1] como un campo de estudio muy importante para el desarrollo social y económico de la humanidad.

No solo se ha aplicado ingeniería en la fabricación de máquinas, sino también en la construcción de estructuras civiles, para estudiar terrenos antes de su emplazamiento (García Gamallo, 1997) o para diseñar y construir dichas estructuras que antes eran realizadas bajo procedimientos empíricos y que ahora, con el nacimiento de las escuelas de ingeniería y otros factores, se ha impulsado una evolución en los procesos de análisis, diseño y construcción de estas estructuras (Asociación Colombiana de Ingeniería Sísmica, 2014).

El ingeniero usa los conceptos de otras áreas de conocimiento como las matemáticas, la física, la química, entre otras ciencias exactas para desarrollar nuevas tecnologías, aplicar con eficiencia las existentes, y manejar los recursos con los que puede contar. Entre estas nuevas tecnologías, los programas de computador (que en conjunto son conocidos como software) se han vuelto indispensables para los negocios, las ciencias y como gran apoyo en los otros campos de ingeniería, y han permitido la creación de tecnologías nuevas o la ampliación de las existentes. Además, el software se ha vuelto tan importante que está cambiando la manera en que las diferentes organizaciones operan sus procesos de negocio, y logran así mejoras importantes dado que automatizan sus operaciones, proporcionan información de apoyo para una toma de decisiones adecuada y facilitan la consecución de ventajas competitivas. Pero a diferencia de los mecanismos, máquinas, edificaciones, circuitos y muchos otros dispositivos físicos, la naturaleza lógica del software presenta un desafío intelectual a la gente que lo desarrolla.

Al igual que en la fabricación de dispositivos físicos, con el software se aplican conocimientos técnicos para desarrollar código con el fin de mejorar los procesos de ingreso, almacenamiento, procesamiento, transporte y presentación de la información, aprovechando los diferentes recursos con los que se cuenta durante su desarrollo. Sin embargo, el software no se basa en las leyes cuantitativas de la física (longitud, voltaje, masa, velocidad, temperatura...), y este desafío intelectual junto con la necesidad de gestionar adecuadamente los aspectos ligados al desarrollo de un proyecto (tiempos de desarrollo, costos, comunicación, riesgos, calidad, entre otros), se

[1] Las palabras españolas *ingenio* e *ingeniería* y la palabra americana *motor* (*engine*) provienen de la misma raíz latina *ingenium*, que tiene dos significados: talento natural y máquina o dispositivo

han vuelto las razones más importantes para implementar procesos de ingeniería en el desarrollo de software.

El software se caracteriza por ser un elemento de un sistema lógico, por lo tanto, no se puede tratar como un elemento físico, pero ello no implica que no puede aplicarse ingeniería en su desarrollo. Por eso se deben aclarar algunos aspectos que lo diferencian de los elementos físicos:

- El software evoluciona desde la comprensión de la necesidad del usuario hasta la entrega de un producto, por esto el software se desarrolla o se modifica con intelecto; no se fabrica o manufactura en sentido estricto.
- Los elementos físicos se deterioran con el uso. El software no se deteriora, pero sí se "desgasta", porque durante cada modificación se podrían generar defectos que disminuyen la calidad del código, pudiendo generar posibles fallos durante su ejecución.

2.1 Productos de ingeniería

Los productos resultantes finales que se entregan al usuario (máquinas, edificaciones, dispositivos electrónicos, software…) no son los únicos elementos que se generan en la aplicación de un proceso de ingeniería adecuado. Por ejemplo, un motor no llega a ser un motor sin una concepción inicial, un análisis de la necesidad de los futuros usuarios, unas métricas y cálculos, un plano o conjunto de planos, una determinación de parámetros de calidad, un plan de pruebas para determinar su eficiencia, y por supuesto, la máquina ensamblada adecuadamente. En todo proceso de ingeniería se aplica una serie de actividades que conllevan a la generación de un conjunto de entregables, llamados *productos de ingeniería*.

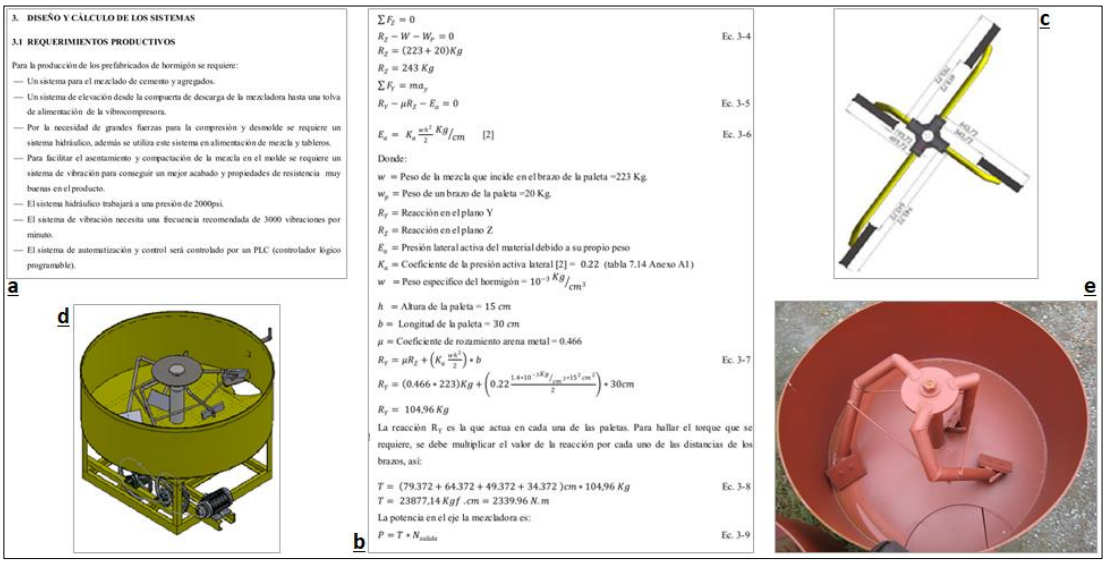


Figura 2.1 Productos de diversas etapas de un proceso de ingeniería mecánica

Para construir, por ejemplo, una máquina para producción de prefabricados de hormigón (Godoy & Sánchez, 2009) se requiere, como se muestra en la Figura 2.1, una lista de los requerimientos a tener en cuenta para su fabricación (a), un conjunto de cálculos para determinar

aspectos de tamaño, capacidad, fuerza, entre otros, de cada una de las piezas que lo conformarían (b), un plano del diseño de cada pieza de la máquina (c), un plano general de ensamble (d), un protocolo de pruebas, entre otros. Todos estos elementos son considerados productos de ingeniería, que son generados con el fin de fabricar la máquina bajo parámetros de calidad acordes con los requerimientos solicitados desde el inicio.

La importancia de los productos de ingeniería radica fundamentalmente en la determinación y verificación, no solo de las funcionalidades esperadas, sino también de los aspectos de calidad necesarios para tener en cuenta en la construcción del resultado final. Esto significa que todo producto de ingeniería debe estar enmarcado bajo parámetros de calidad establecidos incluso desde su ideación. Así como los productos tangibles o físicos cuentan con atributos de calidad como eficiencia, robustez, resistencia, velocidad, duración, capacidad, entre otros, el software también cuenta con atributos tales como la confiabilidad, la mantenibilidad, la seguridad, la portabilidad, entre muchos otros (International Organization for Standardization (ISO), 2011; Rincón Bermúdez, 2014).

Todo proceso de ingeniería se ejecuta en una serie de etapas o fases[2] que, de manera muy generalizada, comprenden: una fase inicial (prediseño o análisis) donde se identifican y se especifican las necesidades de los usuarios o interesados en el resultado final y se establece si es posible fabricarse y qué alcance tendrá; una fase intermedia (diseño) donde se bosqueja y se detalla el producto a través de la generación de modelos, planos, cálculos y la determinación de parámetros de cumplimiento; y una fase final (implementación) donde se fabrica el producto, se prueba para verificar su calidad y se entrega al usuario final. A continuación, se presentarán, en orden inverso, dichas fases del proceso de ingeniería.

2.2 Implementación

Esta es la fase final de la aplicación de un proceso de ingeniería. En esta fase se adquieren, producen, ensamblan y prueban los componentes del resultado requerido. Aquí el resultado final toma una forma tangible o verificable que puede ser presentada al usuario final mientras se construye y/o cuando se tenga ensamblado en su totalidad. Por ejemplo, en el proceso de implementación de una máquina (Ver Figura 2.2), hay que tener en cuenta que, para llegar a la fase de producto terminado listo para entregar (a), los fabricantes debieron tener claro cómo se va a ensamblar (b), con base en una serie de componentes previamente elaborados (c) con base en un diseño.

[2] En el proceso de ingeniería del software, aparecen también estas fases, y en conjunto se conocen como *ciclo de vida del desarrollo de software.* Estas son *el análisis, el diseño, la implementación y las pruebas* y son aplicadas antes de entregar el software al cliente (*implantación del software*).

Figura 2.2 Proceso de implementación de una máquina

En el desarrollo del software sucede el mismo proceso. La producción y ensamblaje de los componentes del software implica un proceso de programación - actividades que permiten codificar, depurar y organizar el código fuente del software - y pruebas - actividades que permitan corroborar que el software cumple con las expectativas y necesidades planteadas por los usuarios (Pressman, 2012). Los resultados finales toman una forma verificable que puede ser presentada al usuario mientras se construye (a través de prototipos) y/o cuando se tenga programado en su totalidad (aplicación terminada).

Figura 2.3 Equipo de desarrollo y software final desarrollado

La programación sigue básicamente un comportamiento algorítmico, es decir, se rige por ciertas reglas y expresiones que permiten establecer una secuencia ordenada y entendible -al menos para los mismos computadores- de órdenes o instrucciones precisas, definidas y finitas (Joyanes Aguilar, 2008). La programación de un algoritmo con pocas funciones no es problema para su desarrollo si se sigue un enfoque tradicional estructurado en el que todo el algoritmo se organiza en un bloque de código. Sin embargo, al volverse más complejo el software y requerir más funciones, es mucho más tedioso poder realizarle una mejora o mantenimiento.

main.py

```
import os
os.system("clear")
#
# Primera sección: Inicialización y captura de datos
#
i = 0
continuar = "S"
sufragantes = []
while (continuar == "S"):
    departamento = []
    departamento.append(
        int(
            input("Potencial de sufragantes departamento # " + str(i + 1) +
                  ": ")))
    departamento.append(
        int(input("Total de sufragantes departamento # " + str(i + 1) + ": ")))
    sufragantes.append(departamento)
    i = i + 1
    continuar = input("Desea digitar datos de otro departamento (S/N)? ")
cantidadDepartamentos = i
#
# Segunda sección: Cálculo del potencial de sufragantes acumulado, el total de sufragantes
# acumulado y el indice de participación total, por proceso de acumulación
#
potencialSufragantesAcumulado = 0
totalSufragantesAcumulado = 0
for i in range(0, cantidadDepartamentos):
    potencialSufragantesAcumulado = potencialSufragantesAcumulado + sufragantes[
        i][0]
    totalSufragantesAcumulado = totalSufragantesAcumulado + sufragantes[i][1]
indiceParticipacionTotal = totalSufragantesAcumulado / potencialSufragantesAcumulado
#
# Tercera sección: Determinación del indice de participación minimo y del indice de
# participación maximo por proceso de selección
#
for i in range(0, cantidadDepartamentos):
    if (i == 0):
        minimoIndiceParticipacion = sufragantes[i][1] / sufragantes[i][0]
        maximoIndiceParticipacion = sufragantes[i][1] / sufragantes[i][0]
    else:
        minimoIndiceParticipacion = min(minimoIndiceParticipacion,sufragantes[i][1] / sufragantes[i][0])
        maximoIndiceParticipacion = max(maximoIndiceParticipacion,sufragantes[i][1] / sufragantes[i][0])
#
# Cuarta sección: Ordenamiento
#
for i in range(0, cantidadDepartamentos - 1):
    for j in range(i + 1, cantidadDepartamentos):
        if (sufragantes[j][1] < sufragantes[i][1]):
            temporal = sufragantes[i][0]
            sufragantes[i][0] = sufragantes[j][0]
            sufragantes[j][0] = temporal
            temporal = sufragantes[i][1]
            sufragantes[i][1] = sufragantes[j][1]
            sufragantes[j][1] = temporal
#
# Quinta sección: Presentación de resultados
#
os.system("clear")
for i in range(0, cantidadDepartamentos):
    print(f"Potencial de sufragantes departamento # " + str(i + 1) + ": " +
          str(sufragantes[i][0]))
    print(f"Total de sufragantes departamento # " + str(i + 1) + ": " +
          str(sufragantes[i][1]))
print(f"Potencial de sufragantes acumulado: {potencialSufragantesAcumulado}")
print(f"Total de sufragantes acumulado: {totalSufragantesAcumulado}")
print(f"Indice de participacion total: {indiceParticipacionTotal}")
```

Figura 2.4 Fragmento de código fuente mediante programación estructurada en lenguaje Python

A diferencia de la programación estructurada tradicional en la que se presenta todo un bloque de líneas de código, se presentará en este libro un enfoque particular de programación en el que se establecen varias unidades de código, separadas y organizadas dependiendo de sus distintas características, comportamientos y responsabilidades. Estas unidades de código se les conoce como ***clases*** y se plantean para que el *software* funcione y se comporte de una forma previamente determinada por su diseño, y permitiendo su desarrollo con una mejor calidad. Este enfoque de trabajo con estas unidades de código es el que se conoce como Programación Orientada a Objetos. Al igual que en los productos tangibles, los desarrolladores deben tener claro qué objetos se deben crear y cómo se va a organizar el código, y para ello deben construir algunos documentos de diseño antes de iniciar.

2.3 Diseño

Como se habló previamente, antes de fabricar un producto hay que tener en cuenta que los fabricantes deben tener claro qué y cómo se va a construir, y deben basarse en algunos documentos que describan los productos a fabricar.

Figura 2.5 Edificio de Oficinas Tamedia en Zúrich (Suiza), diseñado por Shigeru Ban Architects

Estos documentos presentan diferentes representaciones del producto desde varios puntos de vista. A dichas representaciones iniciales del producto real se le conocen como “modelos”, y se observan en todas las ramas de ingeniería -maquetas y planos en ingeniería civil, circuitos y prototipos en ingeniería electrónica, modelos 3D y dibujos en ingeniería mecánica, entre otros-. Todo diseño busca agrupar y aplicar principios, conceptos y prácticas que conlleven al desarrollo de productos de alta calidad, y siempre se debe basar el diseño en las necesidades que han presentado previamente los interesados en el producto.

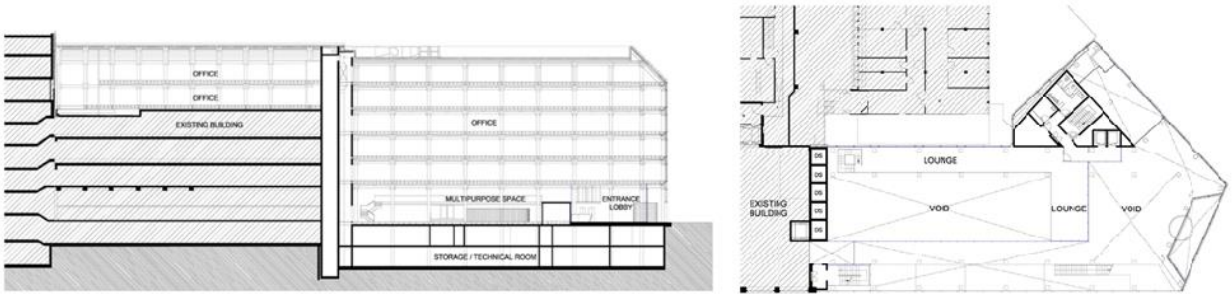


Figura 2.6 Planos de diversas perspectivas de la estructura del edificio

En el desarrollo de software se trabaja de manera similar el uso de algunos planos o modelos para su implementación. El diseño busca traducir los requisitos o características esperadas del

software en planos detallados para construirlo. La organización de las unidades de código o clases parte desde aquí, justamente determinando las características, comportamientos y responsabilidades en cada uno en los diferentes modelos. Este enfoque de diseño es el que se conoce como "diseño orientado a objetos". El software es "dibujado" usando diversas notaciones para la presentación de estos modelos. Se basan particularmente en un lenguaje de modelamiento llamado lenguaje de modelado unificado - UML (Booch, Rumbaugh, & Jacobson, 1999). No obstante, es posible encontrar otros tipos de diseño que no siempre se desarrollan usando UML sino cualquier otro lenguaje de modelamiento; incluso, es posible encontrar modelos basados únicamente en texto.

Tres de los modelos tradicionales más usados en el desarrollo de software son el diagrama de clases, que muestra la estructura de las unidades de código presentando sus características, comportamientos y responsabilidades de comunicación, el diagrama de flujo que presenta el comportamiento interno de cada uno de los métodos incluidos en las clases, y los casos de uso que presentan y especifican la forma de interacción del programa con los usuarios finales. Estos son solo algunos de los modelos que pueden desarrollarse de acuerdo con la necesidad particular del desarrollo del software.

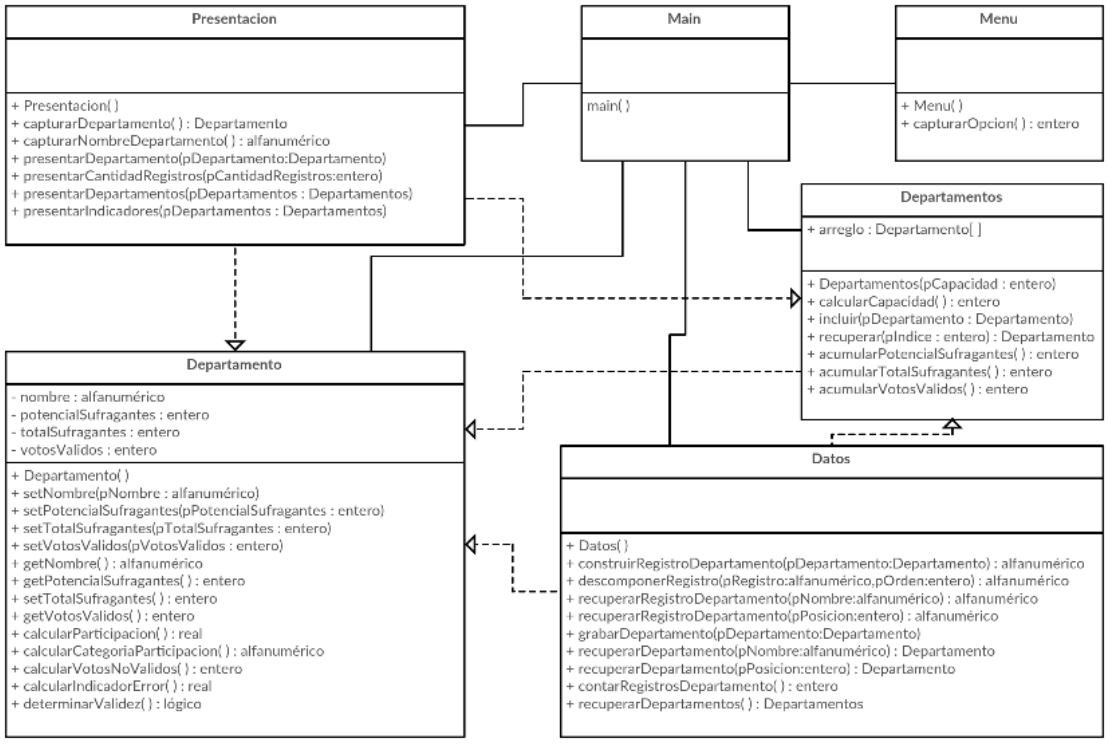


Figura 2.1 Modelo UML: Diagrama de clases

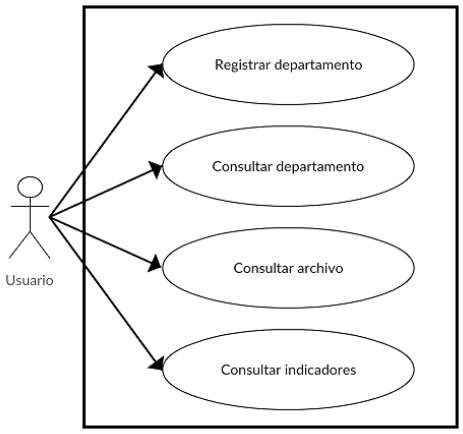


Figura 2.2 Modelo UML: Diagrama de casos de uso

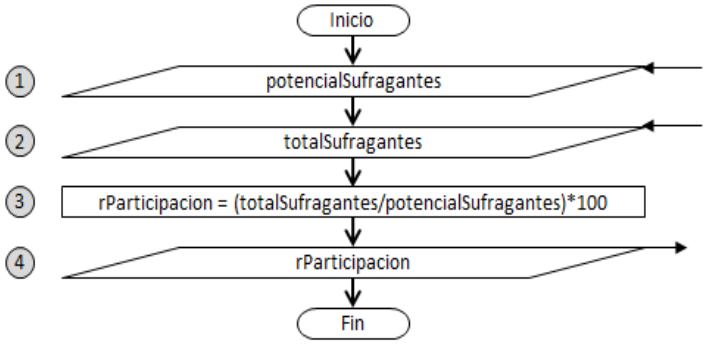


Figura 2.3 Modelo UML: Diagrama de Flujo

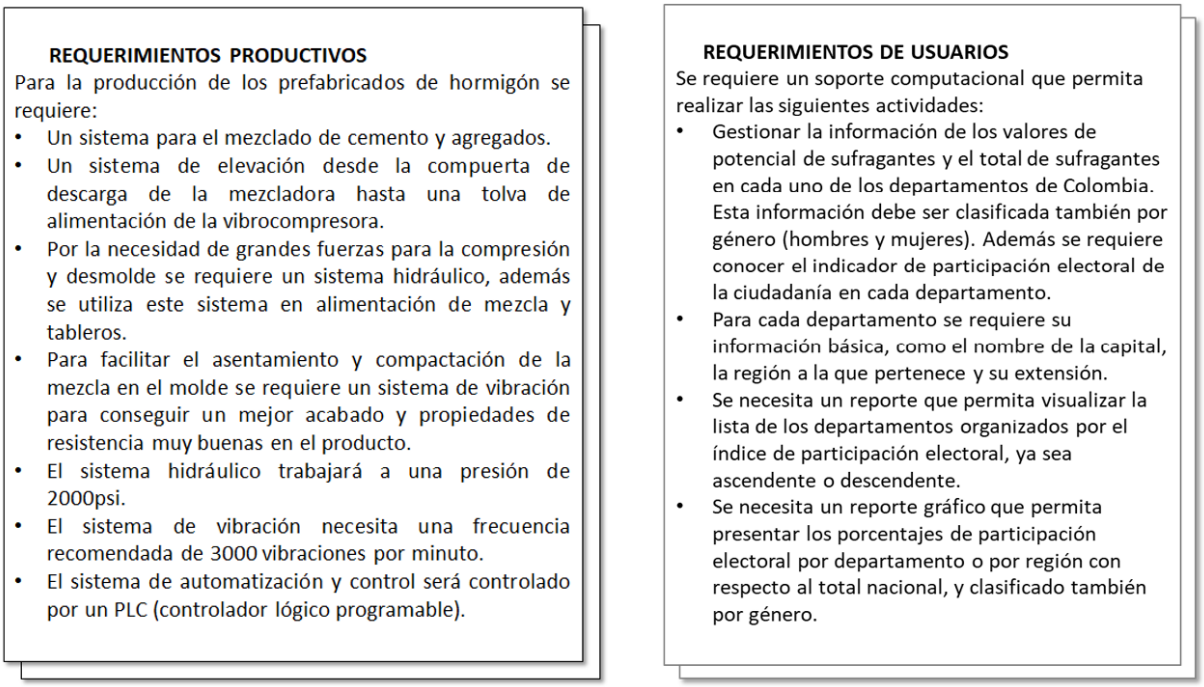
Mientras se diseña, surge una organización general en la cual se determina la estructura de todo el software para poder darle integridad. Esto es lo que se conoce como el diseño arquitectónico del software, presentando la forma como se organizará el sistema como conjuntos de componentes en continua comunicación. Una de estas formas o patrones de arquitectura es conocida como “arquitectura en capas”, la cual se abordará de manera particular en este libro.

2.4 Análisis

Esta es la primera fase en la fabricación de un producto resultado de ingeniería. A pesar de que todas las otras fases son significativas, ésta podría ser considerada la más importante pues busca comprender exactamente qué requieren los usuarios y los interesados en el producto final, identificando los problemas o las oportunidades de mejora, generando una visión general del producto a construir y determinando sus características funcionales y atributos de calidad antes de ser diseñado.

Durante esta fase es muy común encontrar productos de ingeniería basados únicamente en texto, tales como documentos con información sobre normativas, reglamentos, contratos, leyes, requerimientos, aspectos técnicos, operativos, tecnológicos, metodológicos, económicos,

financieros, entre otros, que no solo conciben la idea para el desarrollo del producto sino que también envuelven todo el proceso de ingeniería a aplicar. Entre estos documentos, uno de los más importantes es el documento de determinación de requerimientos, el cual debe presentar un consolidado de las necesidades de los usuarios, así como los criterios que se tendrán en cuenta para la aceptación del resultado final por parte del cliente o usuario.



REQUERIMIENTOS PRODUCTIVOS

Para la producción de los prefabricados de hormigón se requiere:

- Un sistema para el mezclado de cemento y agregados.
- Un sistema de elevación desde la compuerta de descarga de la mezcladora hasta una tolva de alimentación de la vibrocompresora.
- Por la necesidad de grandes fuerzas para la compresión y desmolde se requiere un sistema hidráulico, además se utiliza este sistema en alimentación de mezcla y tableros.
- Para facilitar el asentamiento y compactación de la mezcla en el molde se requiere un sistema de vibración para conseguir un mejor acabado y propiedades de resistencia muy buenas en el producto.
- El sistema hidráulico trabajará a una presión de 2000psi.
- El sistema de vibración necesita una frecuencia recomendada de 3000 vibraciones por minuto.
- El sistema de automatización y control será controlado por un PLC (controlador lógico programable).

REQUERIMIENTOS DE USUARIOS

Se requiere un soporte computacional que permita realizar las siguientes actividades:

- Gestionar la información de los valores de potencial de sufragantes y el total de sufragantes en cada uno de los departamentos de Colombia. Esta información debe ser clasificada también por género (hombres y mujeres). Además se requiere conocer el indicador de participación electoral de la ciudadanía en cada departamento.
- Para cada departamento se requiere su información básica, como el nombre de la capital, la región a la que pertenece y su extensión.
- Se necesita un reporte que permita visualizar la lista de los departamentos organizados por el índice de participación electoral, ya sea ascendente o descendente.
- Se necesita un reporte gráfico que permita presentar los porcentajes de participación electoral por departamento o por región con respecto al total nacional, y clasificado también por género.

Figura 2.4 Comparativo entre un documento de requerimientos para la fabricación de una máquina y los requerimientos para el desarrollo de un software

La información contenida en estos documentos se toma como guía base para la especificación de las funciones y los parámetros de diseño de los productos finales. Al igual que los documentos de requerimientos para la construcción de elementos físicos, los documentos de requerimientos en ingeniería de sistemas reflejan las necesidades encontradas de gestión de información en una organización. Es por ello por lo que en esta fase es muy importante la comprensión del contexto en el que se pretende trabajar, es decir, se busca “comprender el negocio”, así se pueden identificar necesidades más específicas y generar una visión y un alcance general del sistema a implementar.

Otro conjunto de documentos representativos en esta fase lo conforman guías basadas en normativas de instituciones importantes (IPC, ISO, IEEE, ASTM, ICONTEC…) que permiten presentar de forma organizada y detallada las descripciones o características de los productos finales de ingeniería a fabricar; a estos se les conocen como documentos de especificación de

requisitos[3]. Estos documentos de especificación pueden ser descritos desde el punto de vista del mismo sistema -como un requisito formal- (IEEE, 1998) o desde el punto de vista del usuario -como una historia de usuario-. Esta última forma se utiliza de manera frecuente en los marcos de trabajo ágiles para indicar las características que el cliente quiere del sistema, descritas usando la terminología del cliente (Kniberg, 2007).

Todo este conjunto de actividades, aplicado de una forma adecuada desde el entendimiento de los procesos de negocio de una organización hasta la entrega de un producto que cumpla con las expectativas de los usuarios, es lo que se conoce como la aplicación de un proceso de ingeniería, el cual busca hacer un esfuerzo concertado para entender el problema en un contexto antes de desarrollar una solución que satisfaga la necesidad planteada; estudiar de manera cuidadosa las diferentes interacciones de los componentes de la solución que se plantea para poder desarrollarla, adaptarla y mejorarla; y garantizar el correcto funcionamiento de la solución. Para lograrlo, los diferentes componentes a construir deben ser modelados, verificados y validados con el fin de corroborar sus parámetros de funcionalidad y calidad.

2.5 Buenas prácticas de ingeniería

Presentada aquí la importancia de la aplicación de técnicas apropiadas bajo criterios propios de la ingeniería de sistemas, se busca en esta propuesta metodológica incluir algunas de dichas técnicas con el fin de habituar al estudiante a su aplicación desde una etapa temprana de aprendizaje de la programación. Es por ello por lo que en la metodología se incluirán las siguientes técnicas:

1) Una técnica particularmente usada en el desarrollo de software es la construcción de pequeñas soluciones basadas en prototipos, con el fin de validar características o requisitos del software que cumplan con los objetivos o necesidades de personas interesadas. Es eficaz también su utilización cuando no se conoce desde el inicio todo el alcance del software y por consiguiente de todos sus requisitos. Cada uno de los prototipos incluidos en este libro presentará inicialmente un alcance, una visión general u objetivo que se busca lograr con la versión a generar. Esta visión refleja una pequeña descripción de las necesidades de los usuarios finales, los cuales se comportarán como directrices esenciales del prototipo a realizar.
2) A partir de allí se presentarán los requisitos o características que debe tener el prototipo para satisfacer el alcance planteado. Los requisitos se presentan en dos grupos: en cuanto a la interacción esperada con el usuario (presentado de una forma general), y en cuanto a la funcionalidad interna (operaciones matemáticas, establecimiento de reglas que generen validaciones, condiciones o iteraciones, requisitos de almacenamiento para persistencia de información, entre otros).

[3] Estos documentos de requisitos en su versión en inglés contienen en su título el nombre de "*requirements*", y casi siempre se basan en normas internacionales. Como dato curioso, un requerimiento (*request* en inglés) es diferente a un requisito (*requirement* en inglés). Un requerimiento es sinónimo de necesidad, carencia o falta de algo. Por otra parte, un requisito es una circunstancia, condición o característica necesaria para algo.

3) Posteriormente se presentará un modelamiento basado en las características presentadas del prototipo. Este modelamiento presentará el escenario planteado (es decir, la interacción esperada entre el usuario y el sistema presentada de forma más específica, como la presentación de la posible interfaz y el flujo de eventos durante la entrada y de salida de datos), las entidades y atributos (es decir, la organización de los datos más importantes con los que se va a interactuar), la estructura (es decir, los diferentes componentes como clases, con sus atributos definidos y la lista de las funcionalidades y métodos a programar), y el comportamiento (es decir, los algoritmos de cada método a implementar).
4) Se presentará también una versión de código implementado en el lenguaje de programación Java, con el fin de incorporar los conceptos de orientación a objetos y la separación de responsabilidades bajo una estructura general de código (arquitectura de software) llamada arquitectura en capas.

Referencias

Asociación Colombiana de Ingeniería Sísmica. (2014). Norma Colombiana de Diseño de Puentes CCP14.

García Gamallo, A. M. (1997). La evolución de las cimentaciones en la historia de la arquitectura, desde la prehistoria hasta la primera Revolución Industrial (1st ed.). Universidad Politécnica de Madrid.

Godoy, P., & Sánchez, C. (2009). Diseño y construcción de una Máquina Automática para la fabricación de prefabricados de hormigón [Escuela Superior Politécnica de Chimborazo]. http://dspace.espoch.edu.ec/bitstream/123456789/40/1/15T00415.pdf

International Organization for Standardization (ISO). (2011). ISO/IEC 25010:2011 Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models.

Joyanes Aguilar, L. (2008). Fundamentos de programación. Algoritmos, estructura de datos y objetos (1st ed.). McGraw-Hill/Interamericana de España, S. A. U.

Kniberg, K. (2007). Scrum y XP desde las trincheras. Cómo hacemos Scrum. (1st ed.). C4Media Inc.

Pressman, R. (2012). Ingeniería del Software (5th ed.). McGraw Hill.

Rincón Bermúdez, R. D. (2014). Aseguramiento de la calidad en el diseño del software. Universidad EAFIT.

3 Solución monolítica y solución distribuida

La hipótesis inicial de la investigación que dio origen a este libro consistía en que el estudio de la Programación Orientada a Objetos – POO desde el primer curso de introducción a la programación de computadores tiene un efecto favorable en el logro académico de los estudiantes de Ingeniería de Sistemas. Para comprobarlo, durante dos semestres se impartió el curso a grupos experimentales con el enfoque *Objects first*, en tanto que los grupos de control siguieron el abordaje tradicional *Algorithms first*.

La evidencia recolectada condujo a descartar la hipótesis inicial. No se notaron ventajas en la comprensión de la POO por parte de los estudiantes de los grupos experimentales, comparada con la que tuvieron los estudiantes de los grupos de control luego de que estos últimos abordaron la POO en su segundo curso. En cambio, sí fue notorio un déficit del grupo experimental en la capacidad de proponer la lógica de programación para solucionar problemas complejos.

Descartada la hipótesis, el curso de Fundamentos de Programación retornó a su orientación algorítmica tradicional. No obstante, a partir de algunos indicios de mejor comprensión por parte del grupo experimental, se decidió formular un nuevo proyecto con la hipótesis de que la introducción de elementos de arquitectura de software tiene un impacto favorable en la comprensión de la POO en estudiantes que previamente han estudiado programación de computadores con el enfoque *Algorithms first*.

Se decidió incorporar principios de Arquitectura de Capas al segundo curso de programación del plan de estudios, que se centra en POO. La arquitectura de software suele ser objeto de cursos posteriores de Ingeniería del Software. Pero para la realización de este nuevo proyecto se consideró que la especialización de las capas en funciones de presentación, procesamiento y persistencia permitiría proponer una secuencia metodológica que facilitara la incorporación gradual de conceptos de orientación a objetos.

Este capítulo presenta la Arquitectura de Capas en ese contexto, es decir, como recurso metodológico para introducir a estudiantes que acaban de superar un curso de fundamentos de programación con enfoque algorítmico, a situaciones complejas que los persuadan de cambiar a un enfoque de programación más complejo. En las siguientes secciones se muestra cómo el tamaño de un programa va creciendo a medida que se generan nuevos requisitos para el mismo. Se busca demostrar la conveniencia de considerar desde un principio aspectos de arquitectura de software, a pesar del grado adicional de complejidad que implica, y de lo poco usual que resulta su inclusión temprana en el plan de estudios.

3.1 Un proyecto simple

Para comenzar el ejemplo de este capítulo, supongamos una empresa que lleva a cabo un estudio demográfico, para lo cual requiere un software que a partir de la cantidad de hombres y de mujeres que habitan un departamento calcule la población total y el porcentaje de hombres y de mujeres. Se aplican las siguientes fórmulas:

$$población = hombres + mujeres$$
$$porcentajeHombres = (hombres/población) * 100$$
$$porcentajeMujeres = (mujeres/población) * 100$$

Este es un problema típico de los que pueden solucionar estudiantes que han aprobado un curso de fundamentos de programación. La Figura 3.1 muestra el diagrama de flujo propuesto para la solución, denominado ***Algoritmo 1***.

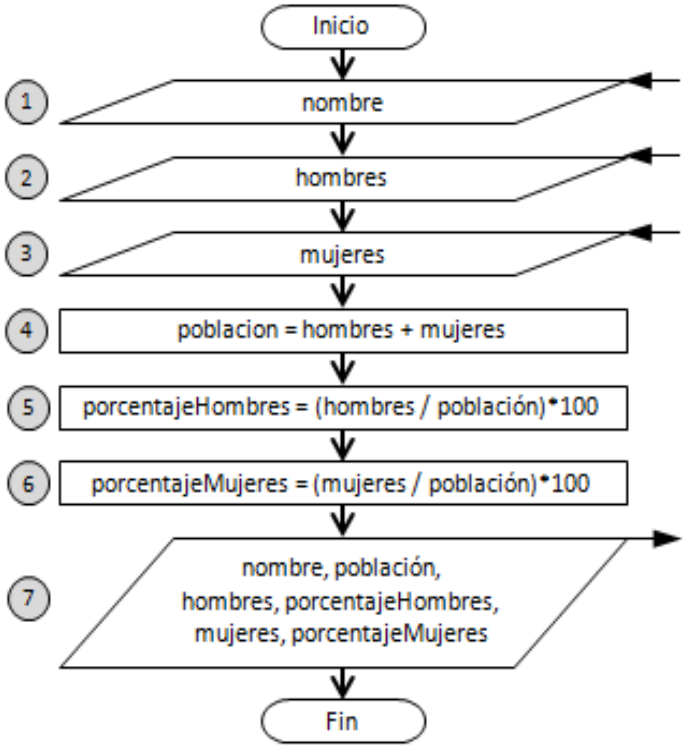


Figura 3.1 Diagrama de flujo de Algoritmo 1.

La Tabla 3.1 describe el funcionamiento del algoritmo representado mediante diagrama de flujo en la Figura 3.1. Los números entre óvalos no son constitutivos del diagrama, sino elementos auxiliares para orientar al lector en las tablas siguientes.

Tabla 3.1 Descripción de *Algoritmo 1*

Paso	Descripción
1	Pedir como dato de entrada el nombre del departamento y almacenarlo en la variable *nombre*.
2	Pedir como dato de entrada la cantidad de hombres que viven en el departamento y almacenarla en la variable *hombres.*
3	Pedir como dato de entrada la cantidad de hombres que viven en el departamento y almacenarla en la variable *mujeres.*
4	Sumar los valores de las variables *hombres* y *mujeres*, y almacenarlo el resultado en la variable *poblacion*.
5	Dividir el valor de la variable *hombres* entre el valor de la variable *poblacion*; multiplicar el resultado por 100 y almacenarlo en la variable *porcentajeHombres*.
6	Dividir el valor de la variable *mujeres* entre el valor de la variable *poblacion*; multiplicar el resultado por 100 y almacenarlo en la variable *porcentajeMujeres*.
7	Presentar como datos de salida los valores de las variables *nombre*, *poblacion*, *hombres, porcentajeHombres, mujeres, porcentajeMujeres*.

Luego en la Tabla 3.2 está codificado el ***Algoritmo 1*** en lenguaje de programación Python. La última columna contiene la numeración correspondiente a cada paso en el diagrama de flujo de la Figura 3.1. El diseño es independiente del lenguaje de programación, en tanto que la sintaxis de cada lenguaje y el estilo de cada programador influyen en la cantidad de líneas requeridas.

Tabla 3.2 Codificación del Algoritmo 1 en lenguaje de programación Python

Línea	Instrucción en Python	Paso
1	nombre=input("Departamento: ")	1
2	hombres=int(input("Cantidad de hombres: "))	2
3	mujeres=int(input("Cantidad de mujeres: "))	3
4	poblacion=hombres+mujeres	4
5	porcentajeHombres=(hombres/poblacion)*100	5
6	porcentajeMujeres=(mujeres/poblacion)*100	6
7	print("\nDepartamento: ",nombre,	7
8	"\nPoblación: ",poblacion," - Categoría: ",categoria,	7
9	"\nHombres: ",hombres," - ",porcentajeHombres," %",	7
10	"\nMujeres: ",mujeres," - ",porcentajeMujeres," %")	7

Por el propósito de este ejemplo se escogió el lenguaje de programación Python, dada su sintaxis simple y las pocas líneas de codificación que por consiguiente requiere. La Figura 3.2 presenta la codificación del programa ***Programa 1***, basado en ***Algoritmo 1***, utilizando el entorno integrado de desarrollo y aprendizaje Python 3.7.

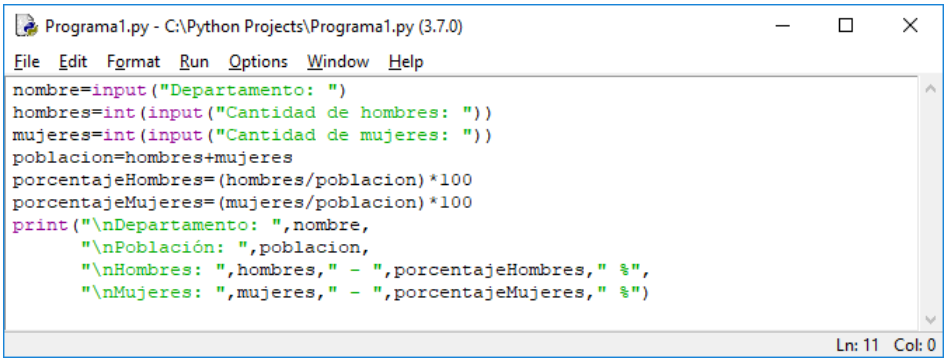


Figura 3.2 Implementación de Programa 1 en lenguaje Python

La Figura 3.3 muestra la primera prueba de ejecución de ***Programa 1***. El usuario digitó ***Santander*** como nombre del departamento, con ***939.672*** como cantidad de hombres y ***973.772*** como cantidad de mujeres. En respuesta el programa presentó para el departamento de ***Santander*** una población total de ***1'913.444*** personas. El porcentaje de hombres es de ***49,10%*** y el de mujeres ***50,89%***.

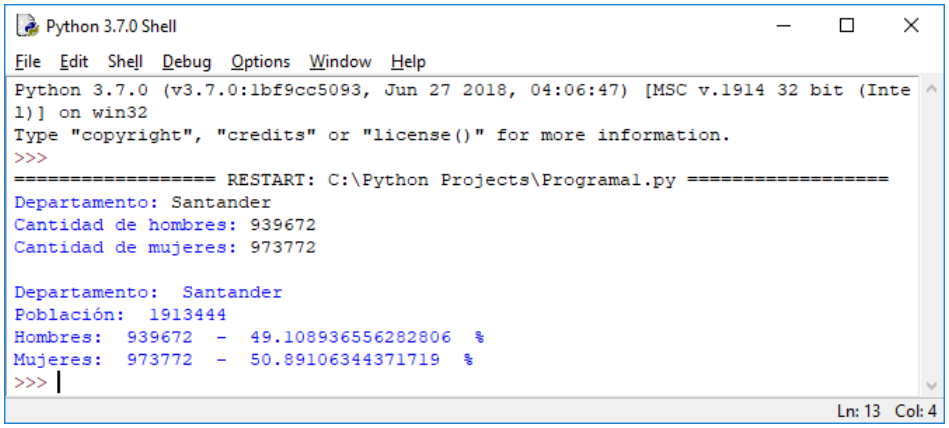


Figura 3.3 Primera prueba de ejecución de Programa 1

La Figura 3.4 muestra la segunda prueba de ejecución de ***Programa 1***, esta vez para el departamento de ***Vichada***. Además del nombre el usuario digitó ***23.306*** para cantidad de hombres y ***21.286*** para cantidad de mujeres. En respuesta el programa presentó para el departamento de ***Vichada*** una población total de ***44.592*** habitantes. Los hombres son el ***52,26%*** de la población, y las mujeres el ***47,73%*** restante.

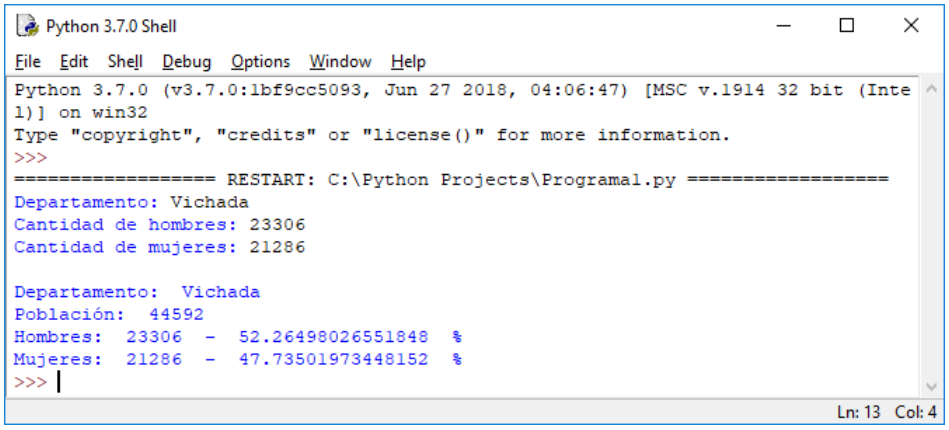


Figura 3.4 Segunda prueba de ejecución de Programa 1

3.2 Nuevos requerimientos

Supongamos que la empresa, satisfecha con los resultados de ***Programa 1***, ahora requiere para el mismo estudio demográfico la aplicación de la Ley 617 del año 2000 en cuanto a la clasificación de los departamentos en categorías según su población: ***Cuarta*** cuando la población sea menor o igual a 100.000 habitantes; ***Tercera*** cuando sea mayor que 100.000 pero menor o igual a 390.000 habitantes; ***Segunda*** cuando supere los 390.000 pero sin sobrepasar los 700.000 habitantes; ***Primera*** si es mayor que 700.000 pero menor o igual a 2'000.000; y ***Especial*** cuando la población sea mayor a 2'000.000 de habitantes.

Al algoritmo para solucionar el problema planteado se le ha denominado ***Algoritmo 2***. Su diagrama de flujo se muestra en la Figura 3.5, con la numeración atenuada en los pasos que se conservan intactos del algoritmo anterior. El algoritmo está compuesto ahora por 16 pasos, es decir, 9 más que el primero.

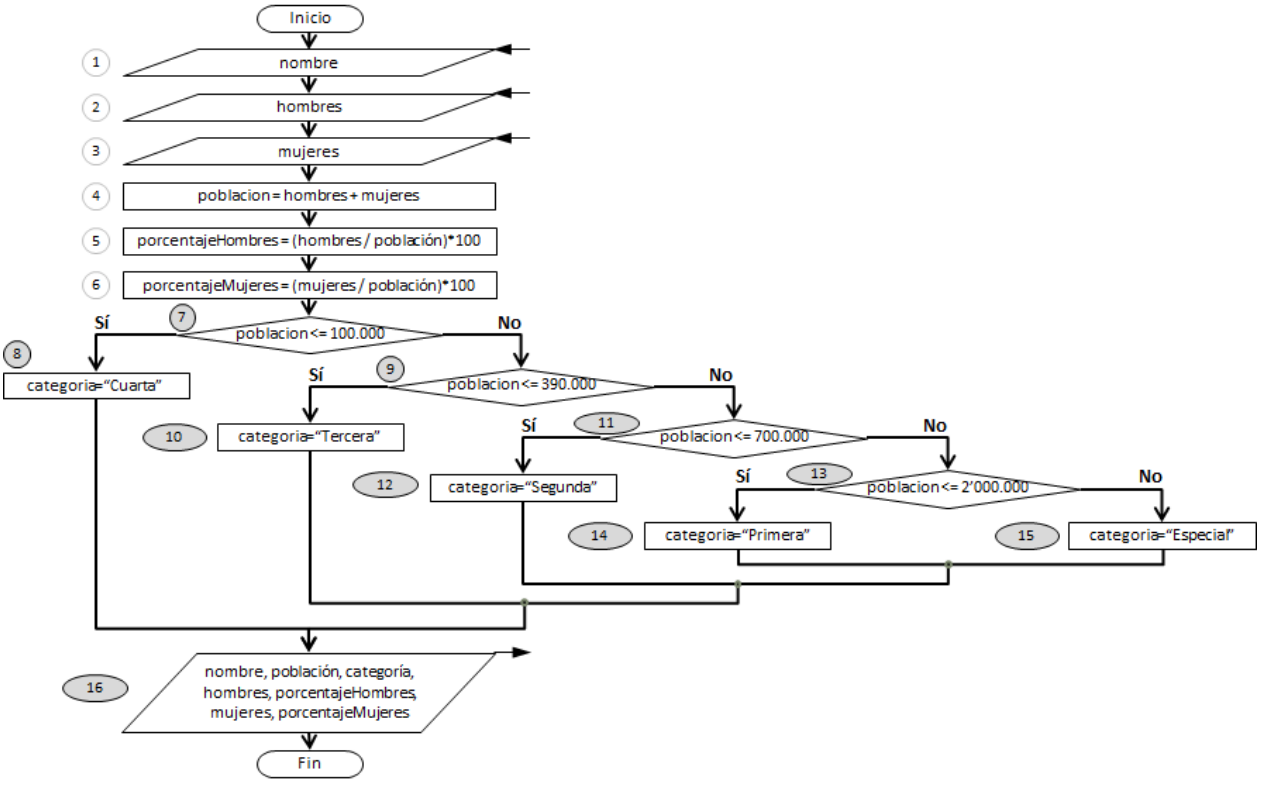


Figura 3.5 Diagrama de flujo de Algoritmo 2

La Tabla 3.3 describe el funcionamiento del algoritmo, codificado posteriormente como ***Programa 2*** en lenguaje de programación Python, según se muestra en la Tabla 3.4, que incluye los números tanto de las líneas como de los pasos para facilitar la comparación entre el código de programación y el algoritmo. La Figura 3.6 presenta la misma codificación utilizando lenguaje Python.

Tabla 3.3 Descripción de *Algoritmo 2*

Paso	Descripción
1	Pedir como dato de entrada el nombre del departamento y almacenarlo en la variable *nombre*.
2	Pedir como dato de entrada la cantidad de hombres que viven en el departamento y almacenarla en la variable *hombres.*
3	Pedir como dato de entrada la cantidad de hombres que viven en el departamento y almacenarla en la variable *mujeres.*
4	Sumar los valores de las variables *hombres* y *mujeres*, almacenando el resultado en la variable *poblacion*.
5	Dividir el valor de la variable *hombres* entre el valor de la variable *poblacion*; multiplicar el resultado por 100 y almacenarlo en la variable *porcentajeHombres*.
6	Dividir el valor de la variable *mujeres* entre el valor de la variable *poblacion*; multiplicar el resultado por 100 y almacenarlo en la variable *porcentajeMujeres*.
7	Evaluar si el valor de la variable *poblacion* es menor o igual que 100.000
8	Si la respuesta en el paso 7 es afirmativa, asignar a la variable *categoria* el valor “Cuarta”.
9	Si la respuesta en el paso 7 es negativa, evaluar si el valor de la variable *poblacion* es menor o igual que 390.000
10	Si la respuesta en el paso 9 es afirmativa, asignar a la variable *categoria* el valor “Tercera”.
11	Si la respuesta en el paso 9 es negativa, evaluar si el valor de la variable *poblacion* es menor o igual que 700.000
12	Si la respuesta en el paso 11 es afirmativa, asignar a la variable *categoria* el valor “Segunda”.
13	Si la respuesta en el paso 11 es negativa, evaluar si el valor de la variable *densidad* es menor o igual que 2’000.000
14	Si la respuesta en el paso 13 es afirmativa, asignar a la variable *categoria* el valor “Primera”.
15	Si la respuesta en el paso 13 es negativa, asignar a la variable *categoria* el valor “Especial”.
16	Presentar como datos de salida los valores de las variables *nombre*, *poblacion*, *categoria*, *hombres*, *porcentajeHombres*, *mujeres* y *porcentajeMujeres*.

Tabla 3.4 Codificación de Algoritmo 2 en lenguaje de programación Python

Línea	Instrucción en Python	Paso
1	nombre=input("Departamento: ")	1
2	hombres=int(input("Cantidad de hombres: "))	2
3	mujeres=int(input("Cantidad de mujeres: "))	3
4	poblacion=hombres+mujeres	4
5	porcentajeHombres=(hombres/poblacion)*100	5
6	porcentajeMujeres=(mujeres/poblacion)*100	6
7	if poblacion<=100000:	7
8	categoria="Cuarta"	8
9	else:	9
10	if poblacion<=390000:	9
11	categoria="Tercera"	10
12	else:	11
13	if poblacion<=700000:	11
14	categoria="Segunda"	12
15	else:	13
16	if poblacion<=2000000:	13
17	categoria="Primera"	14
18	else:	15
19	categoria="Especial"	15
20	print("\nDepartamento: ",nombre,	16
21	"\nPoblación: ",poblacion," Categoría: ",categoria,	16
22	"\nHombres: ",hombres," - ",porcentajeHombres," %",	16
23	"\nMujeres: ",mujeres," - ",porcentajeMujeres," %")	16

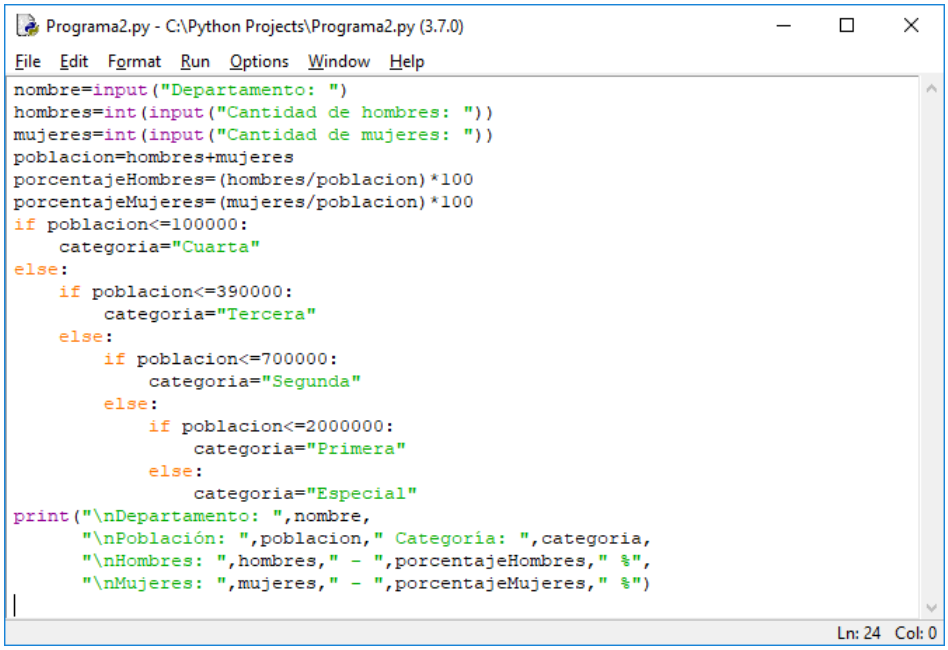


Figura 3.6 Implementación de Programa 2 en lenguaje Python

La Figura 3.7 muestra la primera prueba de ejecución de ***Programa 2***. Para el departamento de ***Santander*** el usuario digitó los mismos datos de la prueba anterior. En respuesta el programa calculó una población de ***1'913.444*** personas, clasificado en la categoría de población ***Primera***. El porcentaje de hombres es de ***49,10%*** y el de mujeres ***50,89%***.

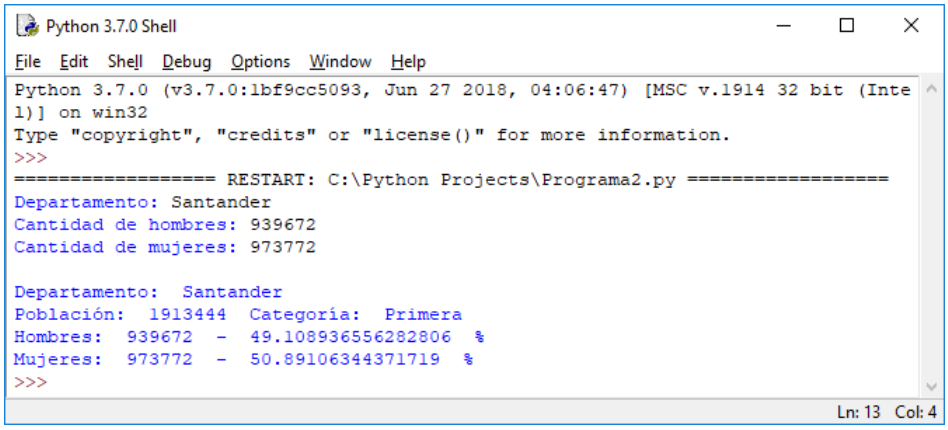


Figura 3.7 Primera prueba de ejecución de Programa 2

La Figura 3.8 muestra la segunda prueba de ejecución de ***Programa 2*** para el departamento de ***Vichada***, con los mismos datos de la prueba anterior. En respuesta el programa calculó una población total de ***44.592*** habitantes, que corresponde a la categoría de población ***Cuarta***. Los hombres son el ***52,26%*** de la población, y las mujeres el ***47,73%*** restante.

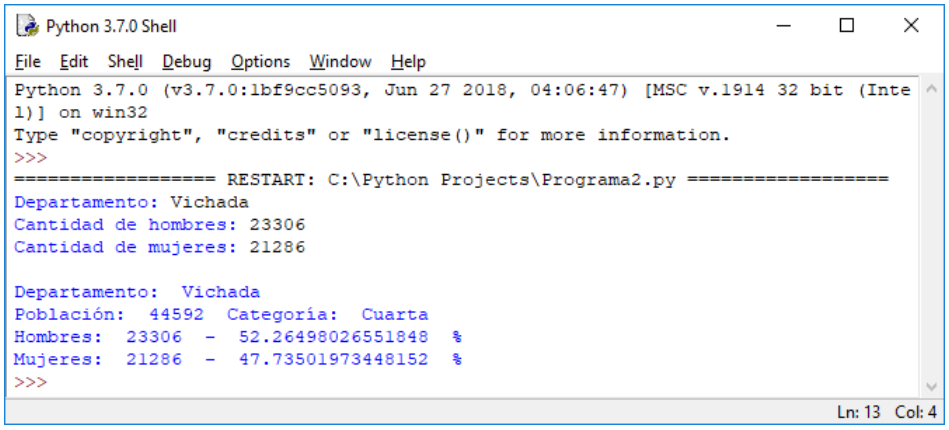


Figura 3.8 Segunda prueba de ejecución de Programa 2

3.3 Un cálculo más

Para llevar el ejemplo un poco más allá, supongamos ahora que la empresa requiere calcular también la densidad poblacional, expresada en habitantes por kilómetro cuadrado, con base en la extensión del departamento aplicando la siguiente fórmula:

$$densidadPoblacional = población/extensión$$

Al algoritmo que incluye este nuevo cálculo para solucionar el problema se le ha denominado ***Algoritmo 3***. Su diagrama de flujo se muestra en la **¡Error! No se encuentra el origen de la referencia.**, también con numeración atenuada de los pasos que se conservan de *Algoritmo 2*. Al igual que con los algoritmos previos, la Tabla 3.5 describe su funcionamiento y la Tabla 3.6 contiene la codificación en lenguaje Python, que luego se muestra también implementada como ***Programa 3*** en el entorno de desarrollo en la **¡Error! No se encuentra el origen de la referencia.**.

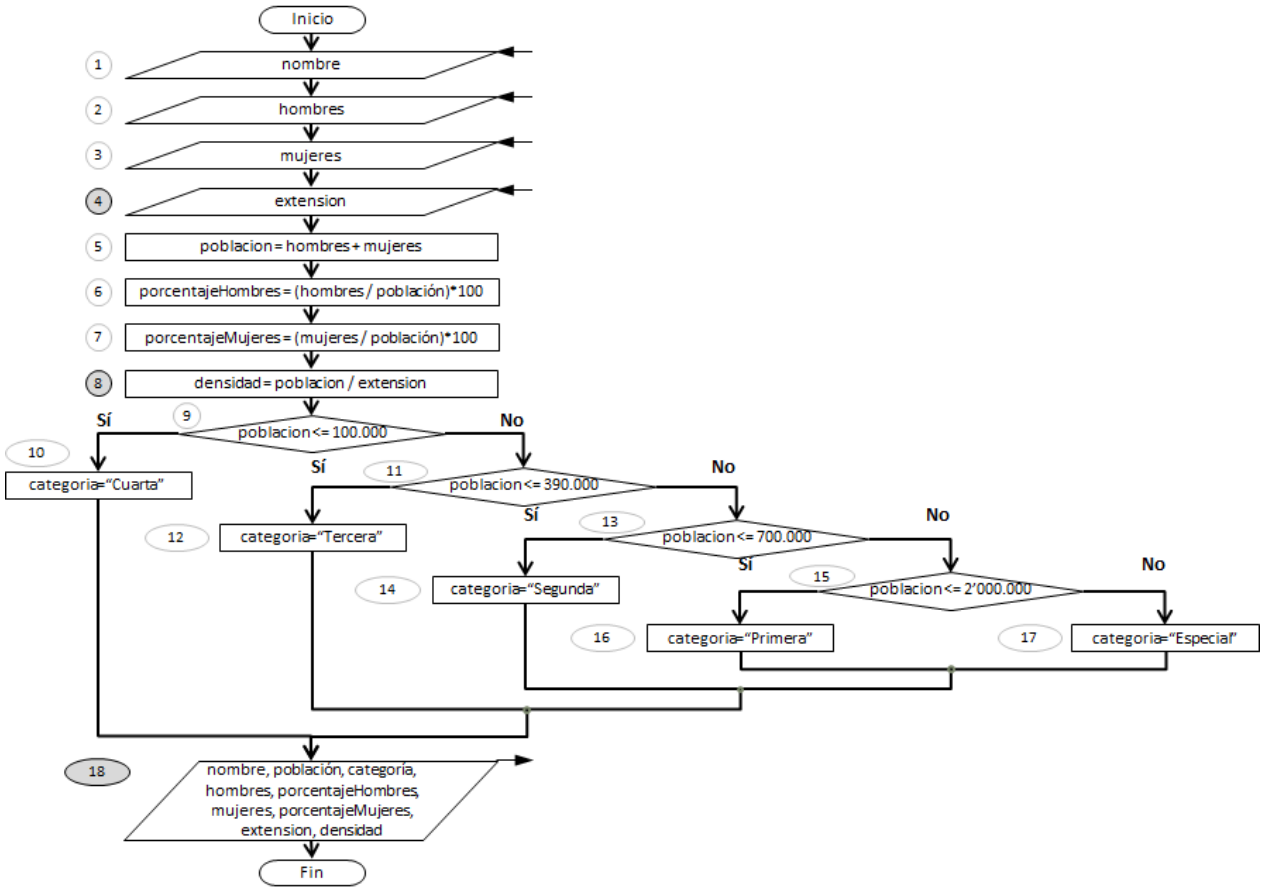


Figura 3.9 Diagrama de flujo de Algoritmo 3

Tabla 3.5 Descripción de *Algoritmo 3*

Paso	Descripción
1	Pedir como dato de entrada el nombre del departamento y almacenarlo en la variable *nombre*.
2	Pedir como dato de entrada la cantidad de hombres que viven en el departamento y almacenarla en la variable *hombres.*
3	Pedir como dato de entrada la cantidad de mujeres que viven en el departamento y almacenarla en la variable *mujeres.*
4	Pedir como dato de entrada la extensión del departamento y almacenarla en la variable *extension.*
5	Sumar los valores de las variables *hombres* y *mujeres*, almacenando el resultado en la variable *poblacion*.
6	Dividir el valor de la variable *hombres* entre el valor de la variable *poblacion*; multiplicar el resultado por 100 y almacenarlo en la variable *porcentajeHombres*.
7	Dividir el valor de la variable *mujeres* entre el valor de la variable *poblacion*; multiplicar el resultado por 100 y almacenarlo en la variable *porcentajeMujeres*.
8	Dividir el valor de la variable *población* entre el valor de la variable *extension*, almacenando el resultado en la variable *densidad*.
9	Evaluar si el valor de la variable *poblacion* es menor o igual que 100.000
10	Si la respuesta en el paso 9 es afirmativa, asignar a la variable *categoria* el valor “Cuarta”.

11	Si la respuesta en el paso 9 es negativa, evaluar si el valor de la variable *poblacion* es menor o igual que 390.000
12	Si la respuesta en el paso 11 es afirmativa, asignar a la variable *categoria* el valor “Tercera”.
13	Si la respuesta en el paso 11 es negativa, evaluar si el valor de la variable *poblacion* es menor o igual que 700.000
14	Si la respuesta en el paso 13 es afirmativa, asignar a la variable *categoria* el valor “Segunda”.
15	Si la respuesta en el paso 13 es negativa, evaluar si el valor de la variable *densidad* es menor o igual que 2’000.000
16	Si la respuesta en el paso 15 es afirmativa, asignar a la variable *categoria* el valor “Primera”.
17	Si la respuesta en el paso 15 es negativa, asignar a la variable *categoria* el valor “Especial”.
18	Presentar como datos de salida los valores de las variables *nombre*, *poblacion*, *categoria*, *hombres*, *porcentajeHombres*, *mujeres*, *porcentajeMujeres*, *extension* y *densidad*.

Tabla 3.6 Codificación de *Algoritmo 3* en lenguaje de programación Python

Línea	Instrucción en Python	Paso
1	nombre=input("Departamento: ")	1
2	hombres=int(input("Cantidad de hombres: "))	2
3	mujeres=int(input("Cantidad de mujeres: "))	3
4	extension=int(input("Extensión [Km2]: "))	4
5	poblacion=hombres+mujeres	5
6	porcentajeHombres=(hombres/poblacion)*100	6
7	porcentajeMujeres=(mujeres/poblacion)*100	7
8	densidad=poblacion/extension	8
9	if poblacion<=100000:	9
10	categoria="Cuarta"	10
11	else:	11
12	if poblacion<=390000:	11
13	categoria="Tercera"	12
14	else:	13
15	if poblacion<=700000:	13
16	categoria="Segunda"	14
17	else:	15
18	if poblacion<=2000000:	15
19	categoria="Primera"	16
20	else:	17
21	categoria="Especial"	17
22	print("\nDepartamento: ",nombre,	18
23	"\nPoblación: ",poblacion," Categoría: ",categoria,	18
24	"\nHombres: ",hombres," - ",porcentajeHombres," %",	18
25	"\nMujeres: ",mujeres," - ",porcentajeMujeres," %",	18
26	"\nExtensión: ",extension," Km2",	18
27	"\nDensidad poblacional: ",densidad," hab/Km2")	18

La Figura 3.11 muestra la primera prueba de ejecución de ***Programa 3***. Para el departamento de ***Santander*** el usuario digitó los mismos datos de la prueba anterior, más la extensión de ***30.534*** Km2. En respuesta el programa presentó una población de ***1’913.444*** personas, clasificado en la categoría de población ***Primera***. El porcentaje de hombres es de ***49,10%*** y el de mujeres ***50,89%***. La densidad poblacional es de ***62,66*** habitantes por kilómetro cuadrado.

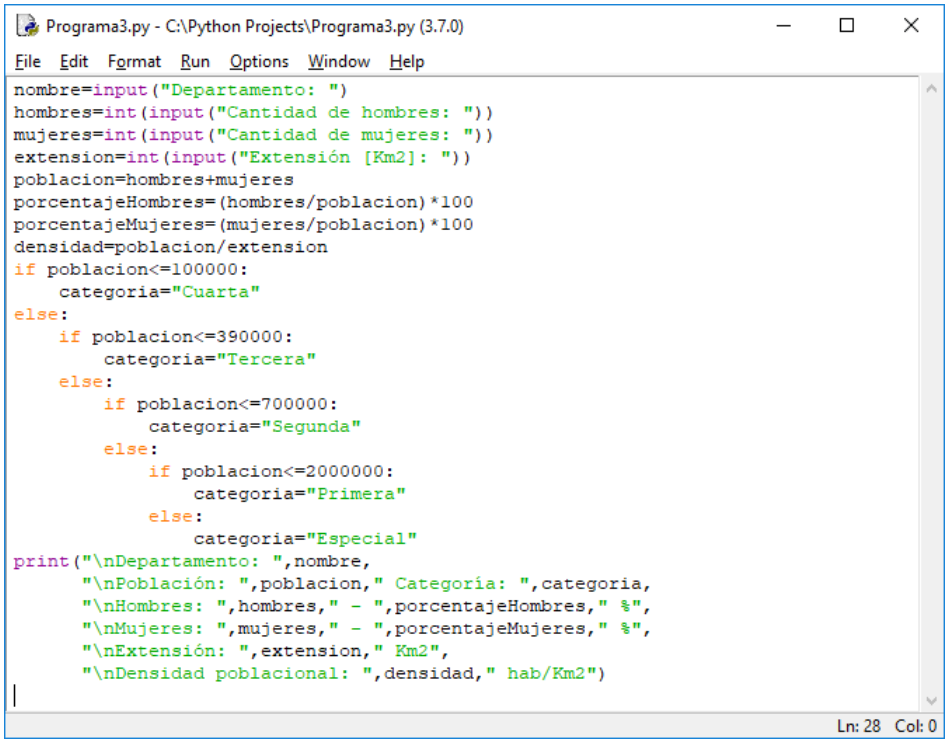


Figura 3.10 Implementación de Programa 3 en lenguaje Python

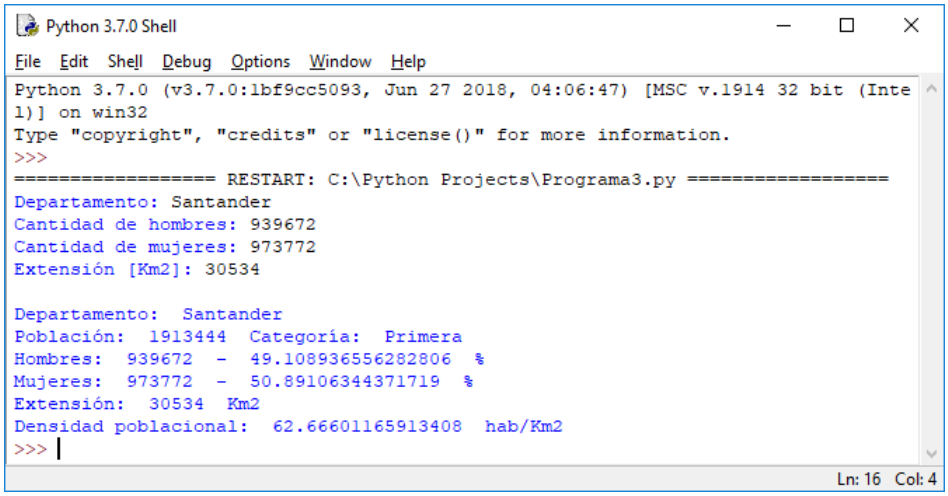


Figura 3.11 Primera prueba de ejecución de Programa 3

La Figura 3.12 muestra la segunda prueba de ejecución de ***Programa 3*** para el departamento de ***Vichada***. Además de los mismos datos de la prueba anterior, el usuario digitó ***100.242*** como la extensión del departamento en Km^2. En respuesta el programa presentó una población total

de ***44.592*** habitantes, que corresponde a la categoría de población ***Cuarta***. Los hombres son el ***52,26%*** de la población y las mujeres el ***47,73%*** restante. La densidad poblacional es de ***0,44*** habitantes por kilómetro cuadrado.

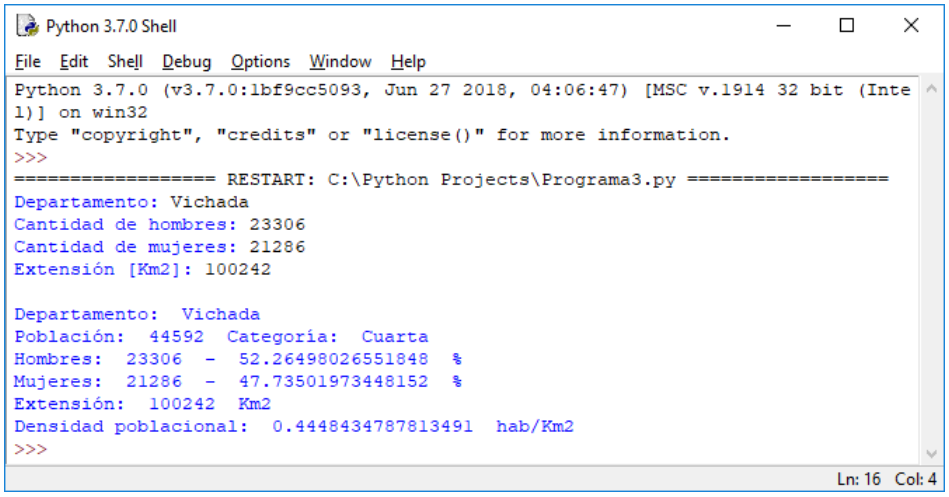


Figura 3.12 Segunda prueba de ejecución de Programa 3

3.4 Escalabilidad

Que el cliente formule nuevos requerimientos una vez ha recibido el software terminado no es una situación extraña. Desde luego, en este ejemplo es una situación artificial generada como recurso didáctico. Pero en proyectos reales sucede con frecuencia, ya sea porque desde el comienzo el cliente no tenía claras todas las necesidades para las que consideró contratar el desarrollo del software, o porque los desarrolladores no asumieron con todo rigor su responsabilidad de acompañar y asesorar al cliente en una determinación exhaustiva de los requerimientos, a lo que están obligados si fungen como Ingenieros de Sistemas o de Software.

En algunos casos el problema del cliente está tan claramente definido, que es posible con la asesoría del equipo de desarrollo hacer una determinación completa de requerimientos desde el comienzo del proyecto. Sin embargo, no es raro que en el transcurso del desarrollo o luego de cada entrega parcial del producto se decida incluir nuevos detalles. La Ingeniería del Software provee metodologías adecuadas para cada caso.

Este ejemplo se propuso para hacer una comparación entre los tres algoritmos y programas, y señalar las implicaciones en el diseño e implementación derivadas de la modificación de los requerimientos. La Figura 3.13 muestra juntos los algoritmos 1, 2 y 3. En cada uno se atenúa la numeración de los pasos que se conservan del algoritmo precedente, para resaltar los nuevos y dimensionar la magnitud del cambio.

Según la Figura 3.13 el segundo algoritmo tiene 9 pasos más que el primero, numerados del 7 al 15. El paso 7 que se ha desplazado hasta la posición 16 no es nuevo, pero incluye más datos de salida.

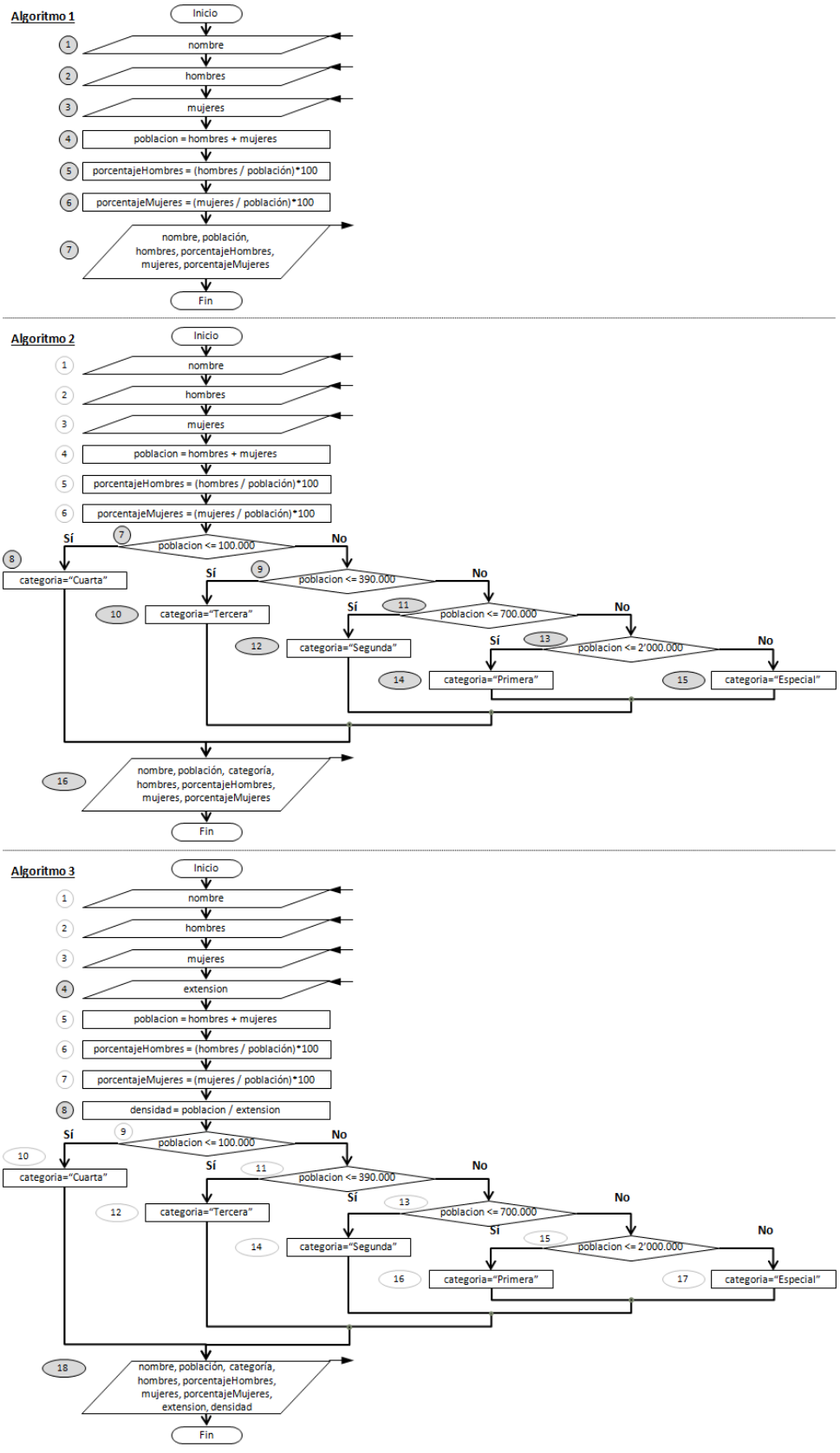


Figura 3.13 Comparación entre Algoritmos 1, 2 y 3

Por otra parte, la estructura del algoritmo deja de ser estrictamente secuencial, haciéndose más compleja en tanto que para determinar la categoría de población se requieren condicionales anidados.

La misma figura permite ver que el tercer algoritmo sube el número de pasos de 16 a 18, siendo nuevos los pasos 4 y 8; es decir, además del incremento en la cantidad de pasos necesarios, los nuevos ahora no se insertan en lugares adyacentes.

A falta de una definición más técnica que se dará posteriormente, la escalabilidad hace referencia a la característica del software de estar hecho de forma que admita mejoras importantes en la funcionalidad sin que se altere de forma traumática su estructura. En el ejemplo esta escalabilidad es baja. Además de la cantidad de líneas adicionales, para la codificación del ***Programa 2*** fue necesario desplazar bloques de código del primero en dos sitios distintos: a partir de la línea 4 y de la línea 7, además de las líneas agregadas al final después de la 10.

Tabla 3.7 Comparación entre la codificación de Programas 1, 2 y 3.

Línea Programa 1	Línea Programa 2	Línea Programa 3	Instrucción en Python
1	1	1	nombre=input("Departamento: ")
2	2	2	hombres=int(input("Cantidad de hombres: "))
3	3	3	mujeres=int(input("Cantidad de mujeres: "))
		4	extension=int(input("Extensión [Km2]: "))
4	4	5	poblacion=hombres+mujeres
5	5	6	porcentajeHombres=(hombres/poblacion)*100
6	6	7	porcentajeMujeres=(mujeres/poblacion)*100
		8	densidad=poblacion/extensión
	7	9	if poblacion<=100000:
	8	10	categoria="Cuarta"
	9	11	else:
	10	12	if poblacion<=390000:
	11	13	categoria="Tercera"
	12	14	else:
	13	15	if poblacion<=700000:
	14	16	categoria="Segunda"
	15	17	else:
	16	18	if poblacion<=2000000:
	17	19	categoria="Primera"
	18	20	else:
	19	21	categoria="Especial"
7	20	22	print("\nDepartamento: ",nombre,
8	21	23	"\nPoblación: ",poblacion," Categoría: ",categoria,
9	22	24	"\nHombres: ",hombres," - ",porcentajeHombres," %",
10	23	25	"\nMujeres: ",mujeres," - ",porcentajeMujeres," %",
		26	"\nExtensión: ",extension," Km2",
		27	"\nDensidad poblacional: ",densidad," hab/Km2")

Estando frente a un programa que apenas alcanza las 27 líneas de código, una cantidad mínima, la inserción en tres lugares distintos puede considerarse una alteración importante de la

estructura. Se puede hacer el ejercicio mental de imaginar lo complejo que puede resultar la modificación de un programa, ya no de dos o tres decenas de líneas, sino de miles e incluso de millones de ellas; asimismo lo difícil que habrá sido antes de eso la elaboración y modificación del diagrama de flujos para representar el algoritmo.

Los inconvenientes señalados en el párrafo anterior son atribuibles a una característica del programa, seguramente usual en las soluciones estudiadas en el curso precedente de introducción a la programación, abordado desde el enfoque algorítmico. Hasta ahora los estudiantes han aprendido a construir algoritmos y programas monolíticos, es decir, que en un solo algoritmo que luego da paso a un único programa solucionan todo el problema.

Para superar este inconveniente, en proyectos de mayor envergadura la solución se distribuye en varios algoritmos, y consecuentemente, en varios bloques de programación.

3.5 Solución distribuida

La Figura 3.14 muestra de nuevo, en la parte superior, la prueba de funcionamiento de ***Programa 1***, y en la parte inferior la prueba de ***Programa 1B***, un programa equivalente al anterior desde la perspectiva del usuario, pero que desde la perspectiva del desarrollador ha sido implementado en lenguaje de programación Java siguiendo un esquema de programación distribuido. Se puede observar que funcionan exactamente igual, con los mismos datos de entrada para el departamento de ***Santander***.

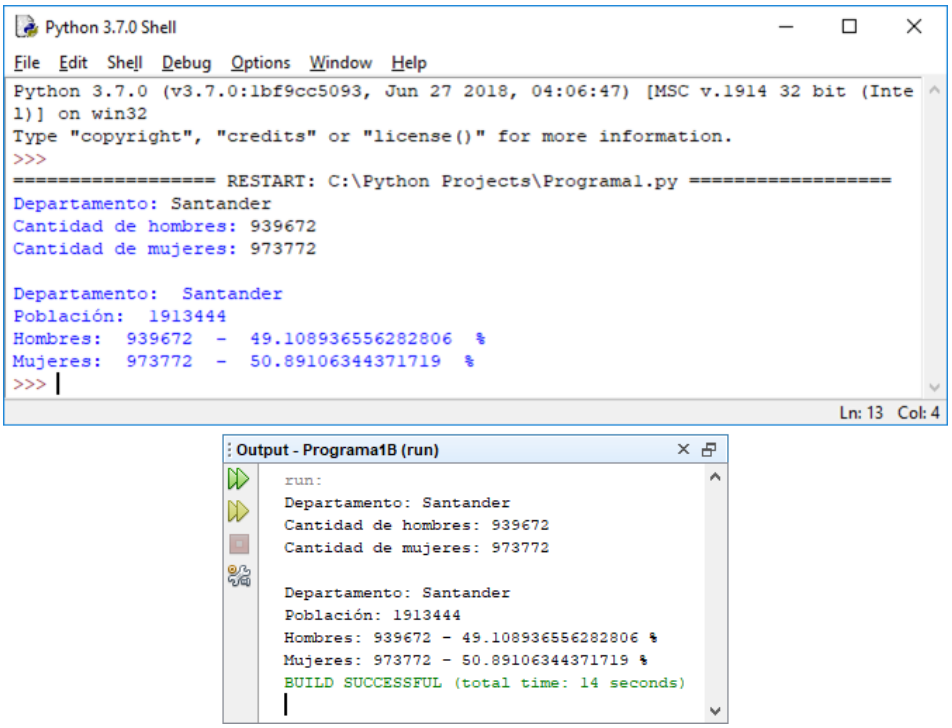


Figura 3.14 Ejecución comparada de Programa 1 y Programa 1B

Más allá del lenguaje en que se ha codificado cada programa, lo que para el usuario resulta aparentemente idéntico, en realidad tiene diferencias de fondo desde la perspectiva del desarrollador. A diferencia de ***Programa 1***, el ***Programa 1B*** consta de tres bloques de código denominadas clases (***Departamento***, ***Presentacion*** y ***Main***) que interactúan entre sí como se representa en la Figura 3.15. La falta de la tilde en el nombre Presentacion no obedece a un error ortográfico, sino a una convención por la cual no se utilizan caracteres especiales, como la tilde, en la nomenclatura de las clases.

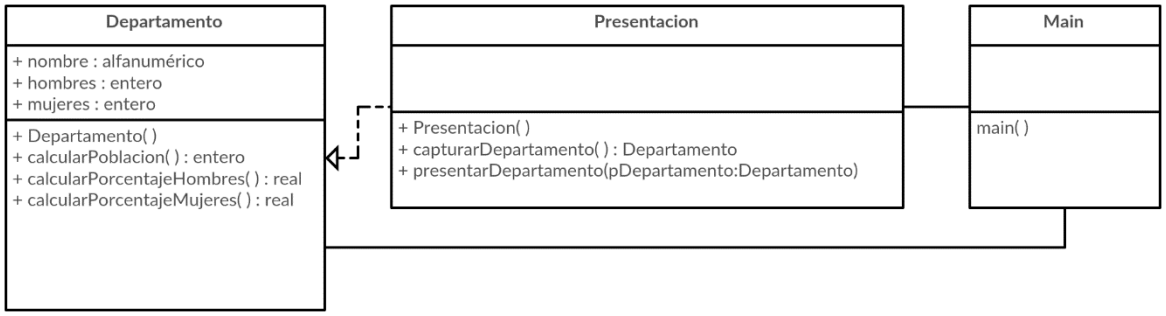


Figura 3.15 Diagrama de clases de Programa 1B

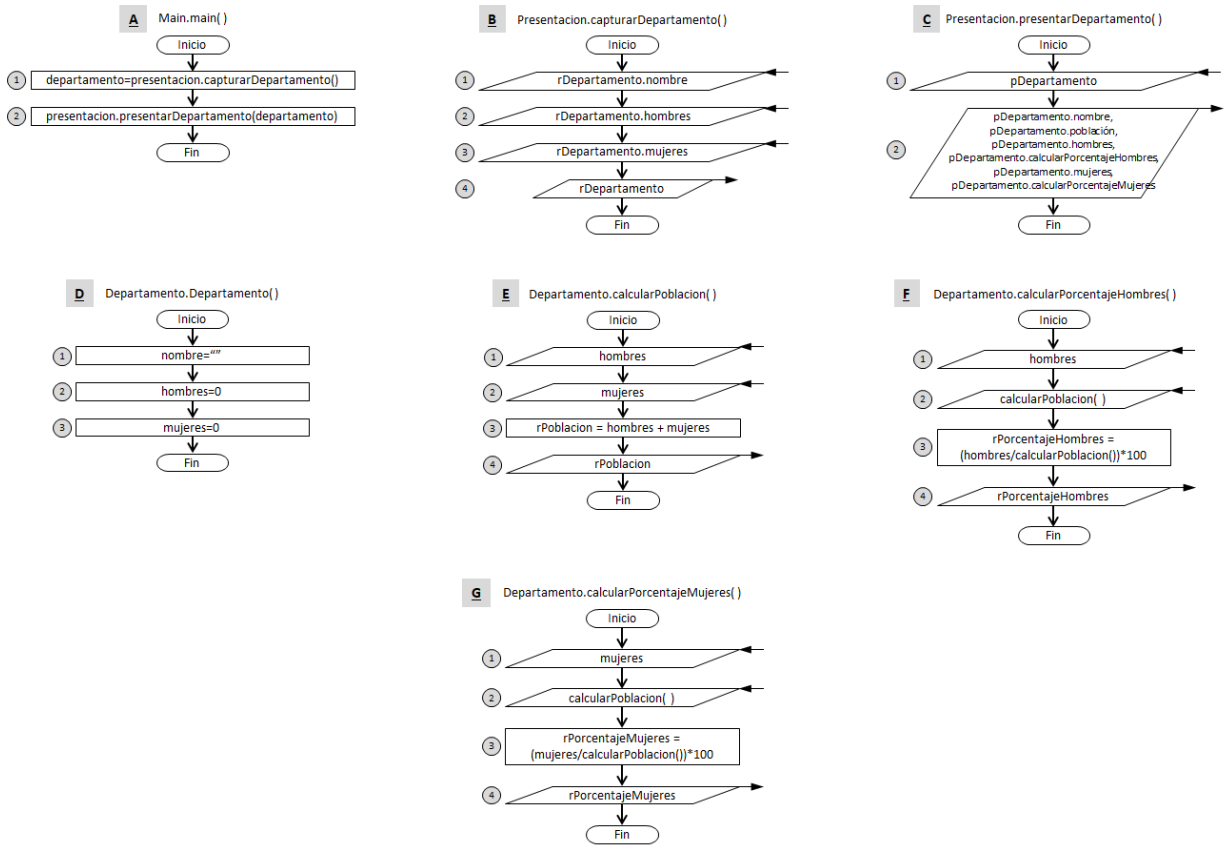


Figura 3.16 Diagramas de flujo de los algoritmos que conforman Programa 1B

En lugar de un único algoritmo el diseño de este nuevo programa consta de tres partes, ***Departamento***, ***Presentacion*** y ***Main***, que conforman el diseño estructural de la Figura 3.15. Cada una de estas partes (clases) está provista internamente de uno o varios algoritmos (métodos), en general mucho más sencillos que el único algoritmo de la solución monolítica, y no destinados al abordaje total del problema sino al cumplimiento de pequeñas tareas que luego juntas conforman la solución. En la Figura 3.16 se muestra dicho conjunto de algoritmos, cada uno con un rótulo que permite identificar a qué clase pertenece. Por ejemplo, el algoritmo con el rótulo ***E*** corresponde al método ***calcularPoblacion*** de la clase ***Departamento***.

Si bien cada algoritmo es más sencillo que el de la solución inicial, la suma de todos ellos al menos en cuanto al número de pasos es mayor. ¿Entonces, qué sentido tiene esta descomposición del algoritmo? La respuesta requiere observar lo que pasa con este enfoque cuando se formulan nuevos requerimientos, lo que se verá en la próxima sección.

Tabla 3.8 Codificación en lenguaje de programación Java de la clase *Departamento* en *Programa 1B*

Línea	Instrucción en Java
1	public class Departamento {
2	public String nombre;
3	public int hombres;
4	public int mujeres;
5	
6	public Departamento(){
7	nombre="";
8	hombres=0;
9	mujeres=0;
10	}
11	
12	public int calcularPoblacion(){
13	int rPoblacion;
14	rPoblacion=hombres+mujeres;
15	return rPoblacion;
16	}
17	
18	public Double calcularPorcentajeHombres(){
19	Double rPorcentajeHombres;
20	rPorcentajeHombres=((double)(hombres)/calcularPoblacion())*100;
21	return rPorcentajeHombres;
22	}
23	
24	public Double calcularPorcentajeMujeres(){
25	Double rPorcentajeMujeres;
26	rPorcentajeMujeres=((double)(mujeres)/calcularPoblacion())*100;
27	return rPorcentajeMujeres;
28	}
29	}

Las Tabla 3.8, Tabla 3.9 y Tabla 3.10 muestran la codificación en lenguaje de programación Java de las tres clases que conforman ***Programa 1B***. Como ya se anotó con respecto a los pasos en los algoritmos, también las líneas de código son más numerosas. Esto se debe a dos aspectos: primero, a la sintaxis propia del lenguaje Java, en general más extensa que la de Python; y

segundo, a que estando las clases divididas en secciones, se hace uso frecuente de líneas en blanco para ayudar a la legibilidad de cada archivo.

Tabla 3.9 Codificación en lenguaje de programación Java de la clase *Presentacion* en *Programa 1B*

Línea	Instrucción en Java
1	public class Presentacion {
2	
3	public Presentacion(){
4	
5	}
6	
7	public Departamento capturarDepartamento(){
8	Scanner scanner=new Scanner(System.in);
9	Departamento rDepartamento;
10	rDepartamento=new Departamento();
11	System.out.print("Departamento: ");
12	rDepartamento.nombre=scanner.nextLine();
13	System.out.print("Cantidad de hombres: ");
14	rDepartamento.hombres=Integer.parseInt(scanner.nextLine());
15	System.out.print("Cantidad de mujeres: ");
16	rDepartamento.mujeres=Integer.parseInt(scanner.nextLine());
17	return rDepartamento;
18	}
19	
20	public void presentarDepartamento(Departamento pDepartamento){
21	System.out.print("\nDepartamento: "+pDepartamento.nombre+
22	"\nPoblación: "+pDepartamento.calcularPoblacion()+
23	"\nHombres: "+pDepartamento.hombres+
24	" - "+pDepartamento.calcularPorcentajeHombres()+" %"+
25	"\nMujeres: "+pDepartamento.mujeres+
26	" - "+pDepartamento.calcularPorcentajeMujeres()+" %\n");
27	}
28	}

Tabla 3.10 Codificación en lenguaje de programación Java de la clase *Main* en *Programa 1B*

Línea	Instrucción en Java
1	public class Main {
2	
3	public static void main(String[] args) {
4	Departamento departamento;
5	Presentacion presentacion;
6	presentacion=new Presentacion();
7	departamento=presentacion.capturarDepartamento();
8	presentacion.presentarDepartamento(departamento);
9	}
10	}

3.6 Característica nueva, algoritmo nuevo

Para emular el ***Programa 2*** que incluye la determinación de la categoría del departamento según la Ley 617 del año 2000, se desarrolló el ***Programa 2B*** cuya prueba de ejecución se muestra en la Figura 3.17. Es evidente que los resultados son idénticos para los mismos datos de entrada del departamento de ***Santander***.

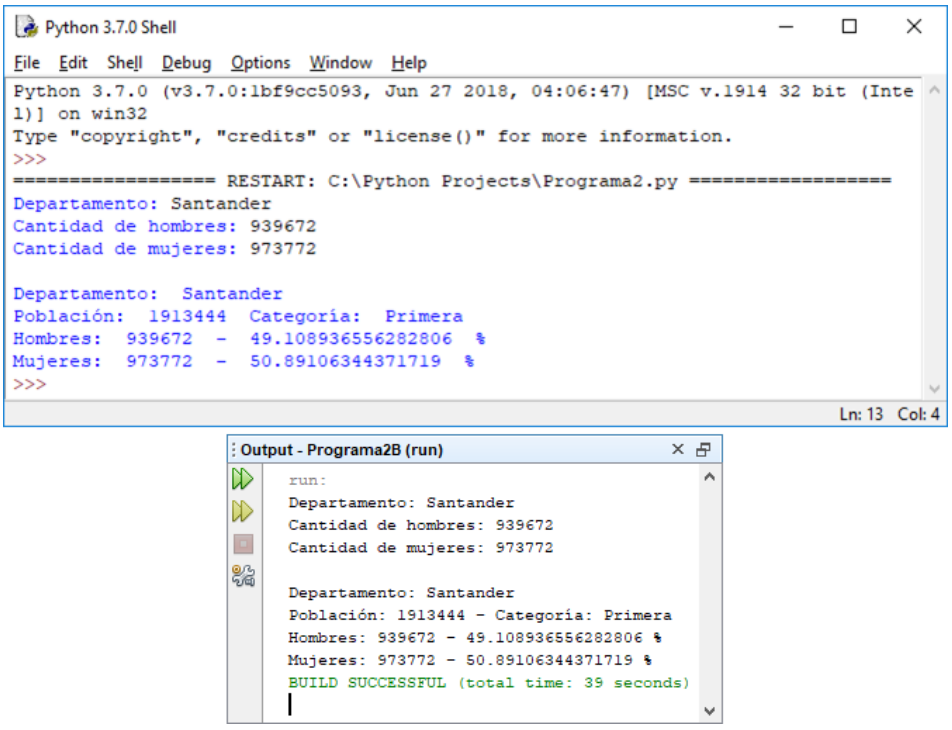


Figura 3.17 Ejecución comparada de Programa 2 y Programa 2B

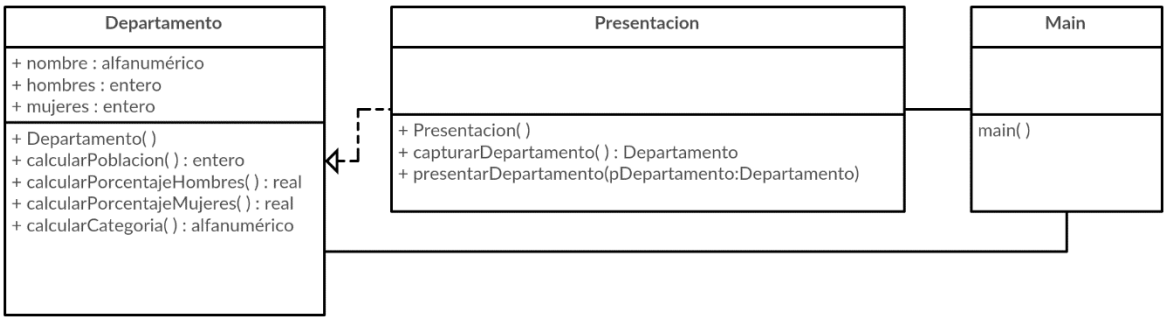


Figura 3.18 Diagrama de clases de Programa 2B

A pesar de que, como resulta obvio, hay que incorporar un algoritmo de cierta complejidad para el cálculo o determinación de la categoría en que se clasifica el departamento, se puede observar en la Figura 3.18 que la estructura del programa sigue siendo la misma, con excepción de la inserción de un nuevo algoritmo (método) denominado ***calcularCategoria()*** en la clase ***Departamento***, pero cuyos pasos no se entrelazan con los del resto de algoritmos, sino que ocupa su propio lugar, delimitado y aparte de los demás. La clase ***Presentacion*** sufre una variación menor en su algoritmo ***presentarDepartamento()*** que requiere una ampliación en el paso 2. En la Figura 3.19 están resaltados los rótulos así como la numeración de los pasos de los algoritmos nuevos o modificados, para facilitar su comparación con los de la Figura 3.16.

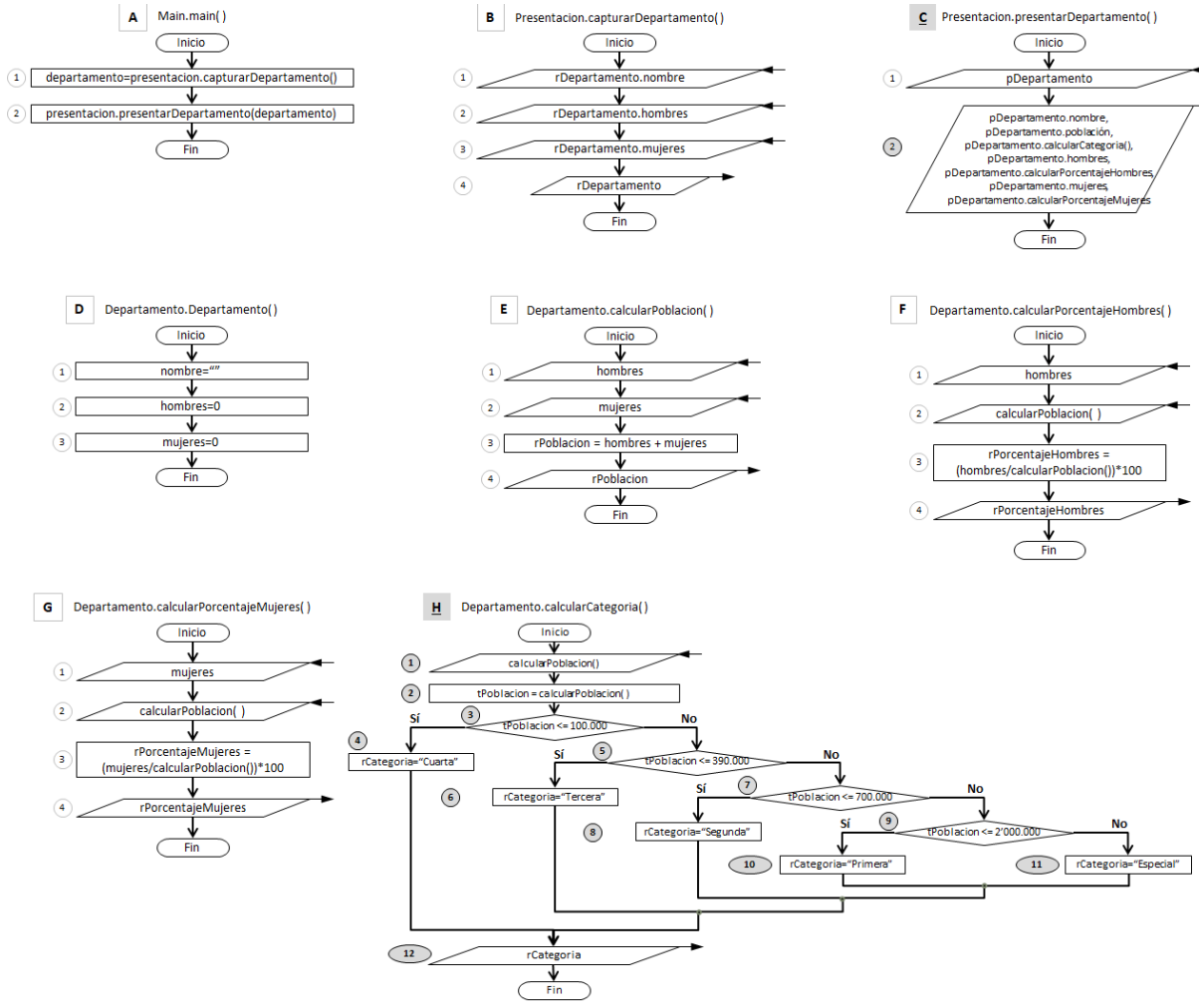


Figura 3.19 Diagramas de flujo de los algoritmos que conforman Programa 2B

Como se evidencia en la Tabla 3.11, toda la codificación del algoritmo nuevo ***calcularCategoria()*** ocupa un solo bloque de programación a partir de la línea 30, con lo que se evita el desplazamiento de otros bloques en distintos lugares del archivo. También aquí se resaltan los números de las líneas nuevas para facilitar la comparación con la Tabla 3.8.

Tabla 3.11 Codificación en lenguaje de programación Java de la clase *Departamento* en *Programa 2B*

Línea	Instrucción en Java
1	public class Departamento {
2	public String nombre;
3	public int hombres;
4	public int mujeres;
5	
6	public Departamento(){
7	nombre="";
8	hombres=0;
9	mujeres=0;
10	}
11	
12	public int calcularPoblacion(){
13	int rPoblacion;
14	rPoblacion=hombres+mujeres;
15	return rPoblacion;
16	}
17	
18	public Double calcularPorcentajeHombres(){
19	Double rPorcentajeHombres;
20	rPorcentajeHombres=((double)(hombres)/calcularPoblacion())*100;
21	return rPorcentajeHombres;
22	}
23	
24	public Double calcularPorcentajeMujeres(){
25	Double rPorcentajeMujeres;
26	rPorcentajeMujeres=((double)(mujeres)/calcularPoblacion())*100;
27	return rPorcentajeMujeres;
28	}
29	
30	public String calcularCategoria(){
31	String rCategoria;
32	if (this.calcularPoblacion()<=100000){
33	rCategoria="Cuarta";
34	}
35	else{
36	if (this.calcularPoblacion()<=390000){
37	rCategoria="Tercera";
38	}
39	else{
40	if (this.calcularPoblacion()<=700000){
41	rCategoria="Segunda";
42	}
43	else{
44	if (this.calcularPoblacion()<=2000000){
45	rCategoria="Primera";
46	}
47	else{
48	rCategoria="Especial";
49	}
50	}
51	}

Línea	Instrucción en Java
52	} return rCategoria;
53	}
54	}
55	

En términos de escalabilidad, como se definió antes, este esquema hace posible que la implementación de nuevas características se haga con una menor interferencia del código ya existente, es decir, con mínimas variaciones de la estructura.

Tabla 3.12 Codificación en lenguaje de programación Java de la clase *Presentacion* en *Programa 2B*

Línea	Instrucción en Java
1	public class Presentacion {
2	
3	public Presentacion(){
4	
5	}
6	
7	public Departamento capturarDepartamento(){
8	Scanner scanner=new Scanner(System.in);
9	Departamento rDepartamento;
10	rDepartamento=new Departamento();
11	System.out.print("Departamento: ");
12	rDepartamento.nombre=scanner.nextLine();
13	System.out.print("Cantidad de hombres: ");
14	rDepartamento.hombres=Integer.parseInt(scanner.nextLine());
15	System.out.print("Cantidad de mujeres: ");
16	rDepartamento.mujeres=Integer.parseInt(scanner.nextLine());
17	return rDepartamento;
18	}
19	
20	public void presentarDepartamento(Departamento pDepartamento){
21	System.out.print("\nDepartamento: "+pDepartamento.nombre+
22	"\nPoblación: "+pDepartamento.calcularPoblacion()+
23	" - Categoría: "+pDepartamento.calcularCategoria()+
24	"\nHombres: "+pDepartamento.hombres+
25	" - "+pDepartamento.calcularPorcentajeHombres()+" %"+
26	"\nMujeres: "+pDepartamento.mujeres+
27	" - "+pDepartamento.calcularPorcentajeMujeres()+" %\n");
28	}
29	}

3.7 Un cálculo adicional y mejoras en la interacción

Además de emular el ***Programa 3*** en el cálculo de la densidad poblacional, el ***Programa 3B*** como se muestra la Figura 3.20, brinda al usuario un modo en el que van apareciendo ventanas sucesivas para que el usuario digite los datos y pulse un botón ***Aceptar***; toda la información, incluyendo los cálculos, se presenta en una última ventana.

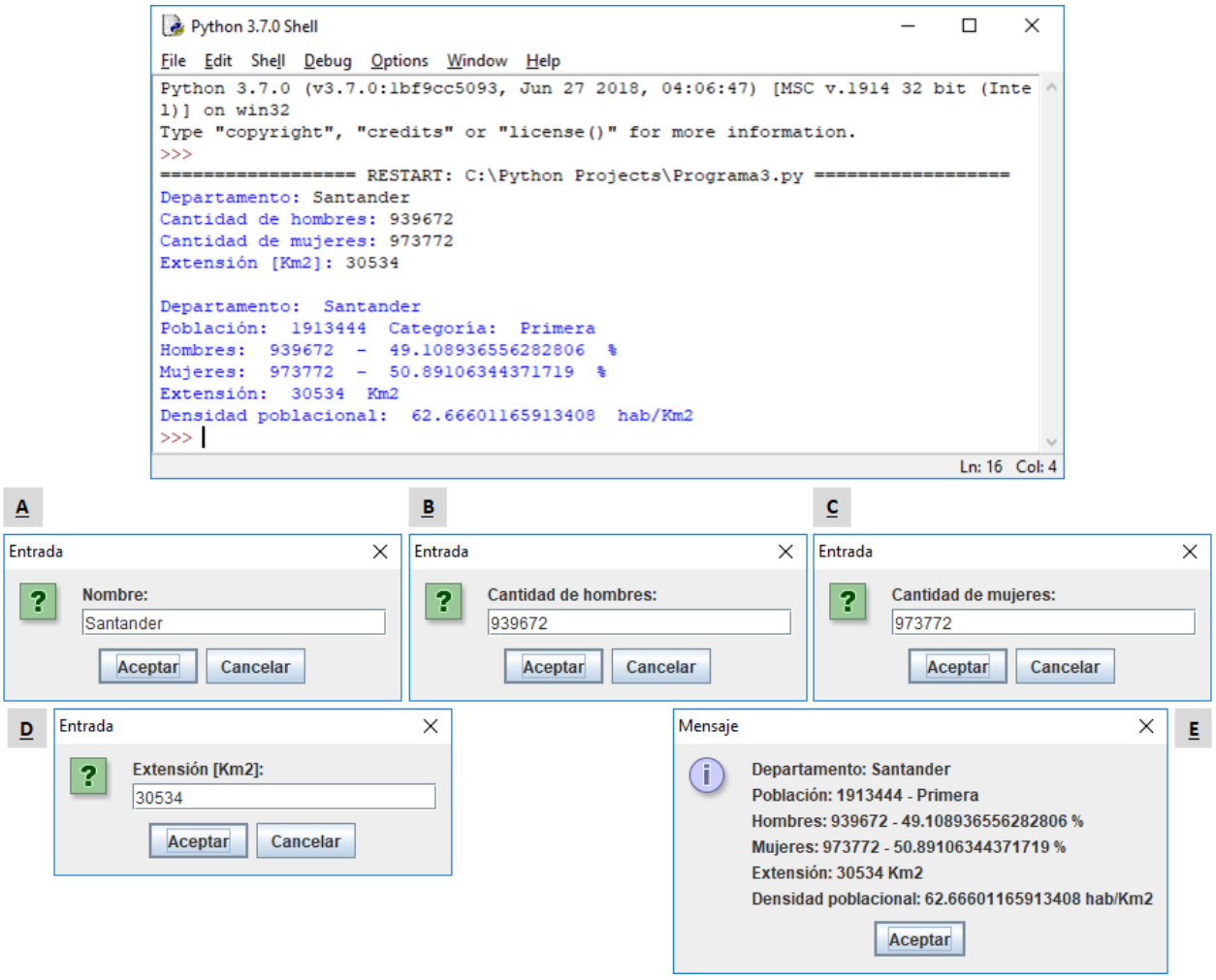


Figura 3.20 Ejecución comparada de Programa 3 y Programa 3B

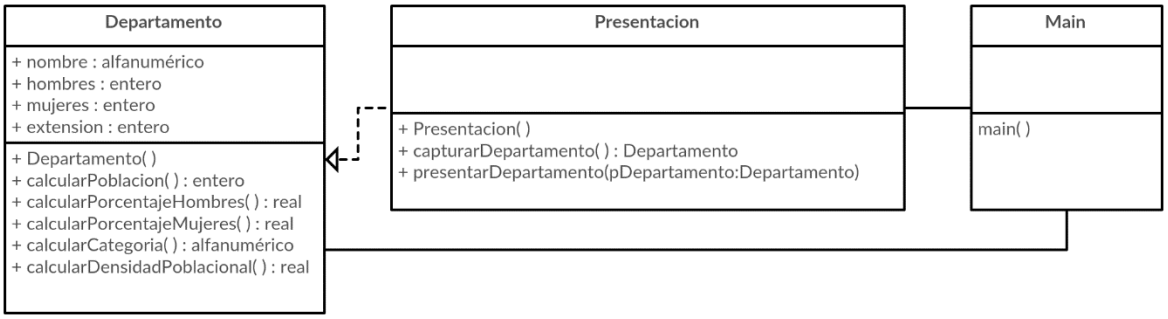


Figura 3.21 Diagrama de clases de Programa 3B

A la clase ***Departamento*** se ha agregado un nuevo atributo ***extension***, que se puede observar en la franja intermedia de dicha clase en la Figura 3.21. Además esta clase cuenta ahora con el método ***calcularDensidadPoblacional()***, que se comporta de acuerdo con el algoritmo con el rótulo ***I*** en la Figura 3.22, en donde también se muestra que por la necesidad de manejar un nuevo dato y el cálculo adicional se han modificado los algoritmos de la clase ***Presentacion***, con los rótulos ***B*** y ***C***.

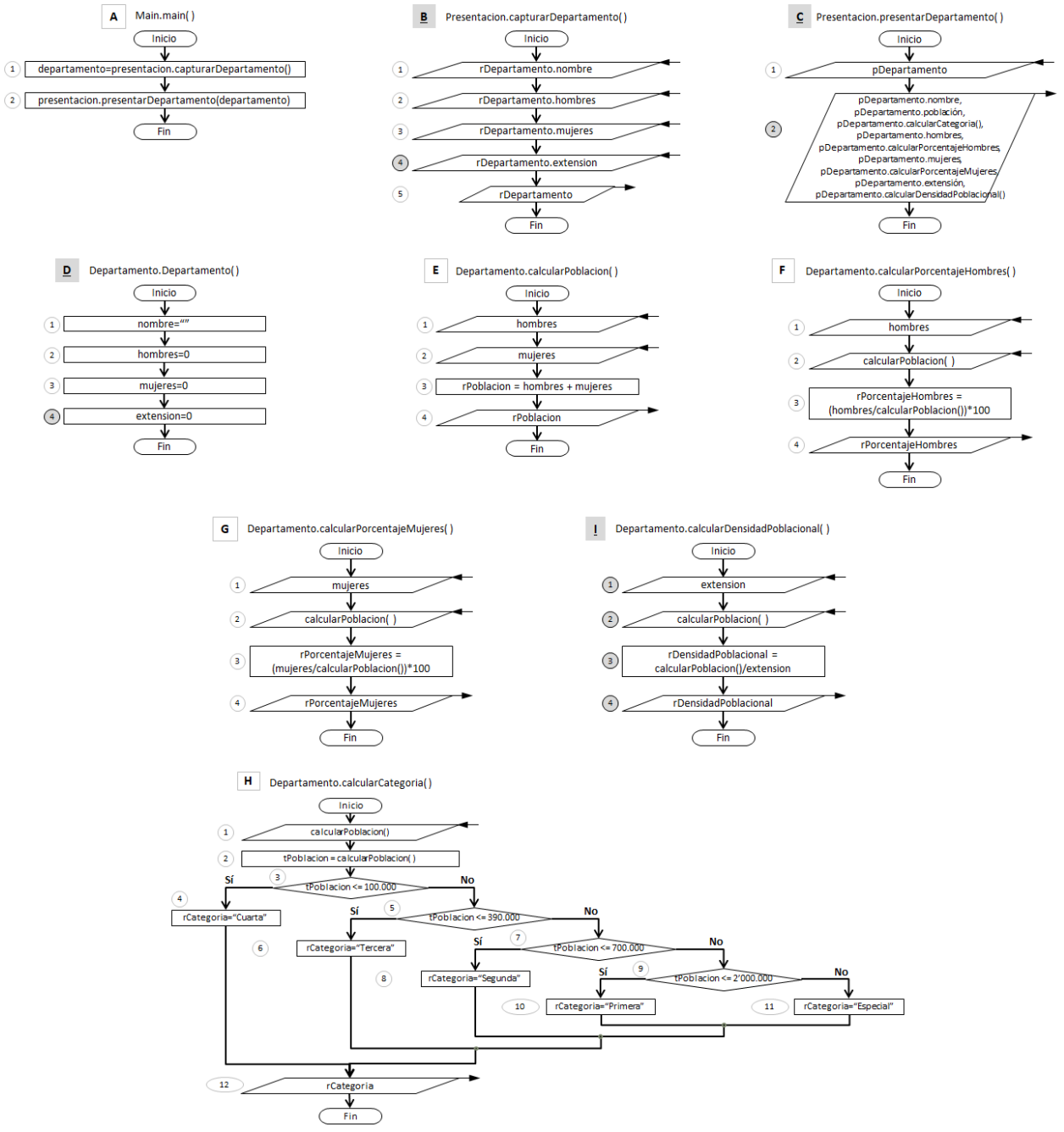


Figura 3.22 Algoritmos de Programa 3

Tabla 3.13 Codificación en lenguaje de programación Java de la clase Departamento en *Programa 3B*

Línea	Instrucción en Java
1	public class Departamento {
2	public String nombre;
3	public int hombres;
4	public int mujeres;
5	public int extension;
6	
7	public Departamento(){
8	nombre="";
9	hombres=0;
10	mujeres=0;
11	extension=0;
12	}
13	
14	public int calcularPoblacion(){
15	int rPoblacion;
16	rPoblacion=hombres+mujeres;
17	return rPoblacion;
18	}
19	
20	public Double calcularPorcentajeHombres(){
21	Double rPorcentajeHombres;
22	rPorcentajeHombres=((double)(hombres)/calcularPoblacion())*100;
23	return rPorcentajeHombres;
24	}
25	
26	public Double calcularPorcentajeMujeres(){
27	Double rPorcentajeMujeres;
28	rPorcentajeMujeres=((double)(mujeres)/calcularPoblacion())*100;
29	return rPorcentajeMujeres;
30	}
31	
32	public String calcularCategoria(){
33	String rCategoria;
34	if (this.calcularPoblacion()<=100000){
35	rCategoria="Cuarta";
36	}
37	else{
38	if (this.calcularPoblacion()<=390000){
39	rCategoria="Tercera";
40	}
41	else{
42	if (this.calcularPoblacion()<=700000){
43	rCategoria="Segunda";
44	}
45	else{
46	if (this.calcularPoblacion()<=2000000){
47	rCategoria="Primera";
48	}
49	else{
50	rCategoria="Especial";
51	}

Línea	Instrucción en Java
52	} }
53	}
54	return rCategoria;
55	}
56	
57	public Double calcularDdensidadPoblacional(){
58	Double rDensidadPoblacional;
59	rDensidadPoblacional=(double)(this.calcularPoblacion())/this.extension;
60	return rDensidadPoblacional;
61	}
62	}
63	

Las líneas nuevas o modificadas tienen la numeración subrayada para facilitar su ubicación. En términos de escalabilidad, como se definió antes, este esquema hace posible que la implementación de nuevas características se haga con una menor interferencia del código ya existente, es decir, con mínimas variaciones de la estructura.

Tabla 3.14 Codificación en lenguaje de programación Java de la clase *Presentacion* en *Programa 3B*

Línea	Instrucción en Java
1	public class Presentacion {
2	
3	public Presentacion(){
4	
5	}
6	
7	public Departamento capturarDepartamento(){
8	Departamento rDepartamento;
9	rDepartamento=new Departamento();
10	rDepartamento.nombre=JOptionPane.showInputDialog("Nombre: ");
11	rDepartamento.hombres=Integer.parseInt(JOptionPane.showInputDialog("Cantidad de hombres: "));
12	rDepartamento.mujeres=Integer.parseInt(JOptionPane.showInputDialog("Cantidad de mujeres: "));
13	rDepartamento.extension=Integer.parseInt(JOptionPane.showInputDialog("Extensión [Km2]: "));
14	return rDepartamento;
15	}
16	
17	public void presentarDepartamento(Departamento pDepartamento){
18	String rMensaje;
19	JOptionPane.showMessageDialog(null,
20	"Departamento: "+pDepartamento.nombre+
21	"\nPoblación: "+pDepartamento.calcularPoblacion()+
22	" - "+pDepartamento.calcularCategoria()+
23	"\nHombres: "+pDepartamento.hombres+
24	" - "+pDepartamento.calcularPorcentajeHombres()+" %"+
25	"\nMujeres: "+pDepartamento.mujeres+
26	" - "+pDepartamento.calcularPorcentajeMujeres()+" %"+
27	"\nExtensión: "+pDepartamento.extension+" Km2"+
28	"\nDensidad poblacional: "+pDepartamento.calcularDensidadPoblacional()+" hab/Km2");
29	}
30	}

3.8 Comparación entre desarrollo monolítico y distribuido

En las secciones anteriores se ha presentado dos enfoques de solución de un problema. En el enfoque de ***solución monolítica*** un solo algoritmo se encarga de resolver todo el problema. Cuando se requiere que el programa tenga características adicionales se hacen las modificaciones pertinentes en el mismo algoritmo. Por ese camino fueron construidos el ***Algoritmo 1***, el ***Algoritmo 2*** y el ***Algoritmo 3***, compuestos por 7, 16 y 18 pasos respectivamente, codificados luego en lenguaje Python para dar origen a ***Programa 1***, ***Programa 2*** y ***Programa 3***.

El mismo problema fue abordado mediante el enfoque de ***solución distribuida***. A tres clases diferentes se les dotó de un conjunto de algoritmos más sencillos, y con las tres se implementaron las versiones alternativas ***Programa 1B***, ***Programa 2B*** y ***Programa 3B***.

¿Qué diferencias, ventajas o desventajas, supone cada enfoque frente al otro? La respuesta a esta pregunta puede darse desde diferentes perspectivas.

La Figura 3.23 compara el número de pasos que conforman el algoritmo único en la solución analítica, y la suma del número de pasos de todos los algoritmos en la solución distribuida, contados en los respectivos diagramas de flujo.

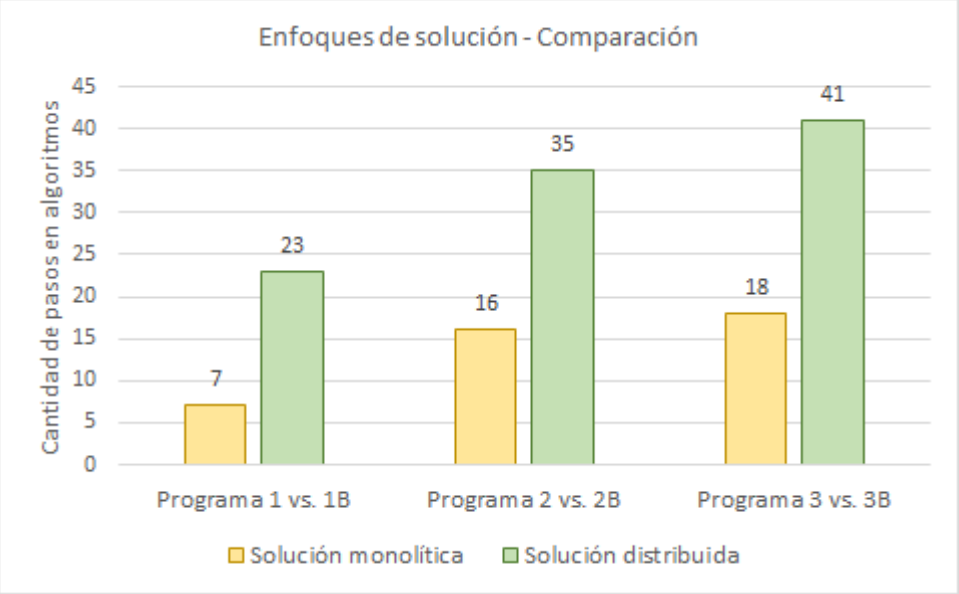


Figura 3.23 Enfoques de solución - Comparación por número de pasos

En la solución analítica la primera versión (Programa 1) requirió un algoritmo de 7 pasos, que se incrementaron hasta 16 en la segunda versión y luego a 18 en la tercera. Por su parte en la solución distribuida la primera versión ya tenía 23 pasos, la segunda 35, y la tercera versión requirió un total de 41 pasos para los 9 algoritmos que conforman la base del ***Programa 3B***.

Una conclusión inmediata es que incluso la primera versión de la solución distribuida ya es más extensa que la tercera de la versión monolítica. A pesar de esta afirmación, los restantes capítulos

de este libro están dedicados a presentar una metodología para el estudio de la Programación Orientada a Objetos apoyada en Arquitectura de Capas, que es una de varias alternativas posibles de solución distribuida. Hay que asumir que al comienzo este enfoque implica un esfuerzo de diseño mayor que el requerido en la solución monolítica, quedando pendiente demostrar qué justifica incurrir en esta aparente desventaja inicial.

Una justificación posible se puede construir si en lugar de comparar la cantidad absoluta de pasos requeridos en ambos enfoques, se compara la tasa a que dicha cantidad crece a medida que se incrementa la complejidad del problema. Para esto en la Figura 3.24 se hace una comparación normalizada, es decir, tomando como base la cantidad de pasos requeridos en la primera versión de la solución en cada enfoque, y expresando los demás datos en proporción a los primeros.

En ambos casos de la primera versión, ***Programa 1*** y ***Programa 1B***, el dato presentado es 1. Como el algoritmo que sustenta el ***Programa 2*** requiere 16 pasos, frente a los 7 del primer algoritmo, el dato de 2,29 corresponde a la división de 16 entre 7, e indica que el segundo algoritmo tiene 2,29 veces la cantidad de pasos que el primero. En cuanto a la solución distribuida los algoritmos con que se construyó el ***Programa 2B*** requirieron 1,57 veces la cantidad de pasos que la versión inicial, resultado de dividir 35 entre 23.

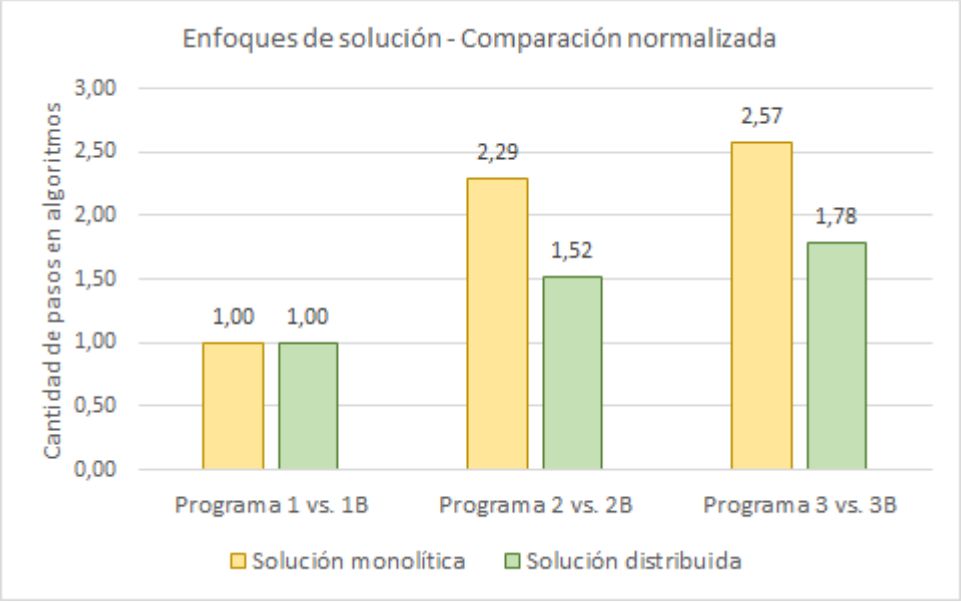


Figura 3.24 Enfoques de solución - Comparación normalizada por número de pasos

La tercera versión de la solución monolítica tiene 2,57 veces la cantidad de pasos de la primera versión. Esta proporción en el caso de la solución distribuida es de solo 1,78 veces. En conclusión, con los datos de los programas desarrollados como ejemplo en este capítulo, aunque la solución distribuida requiere un mayor esfuerzo inicial en cuanto al número de pasos de los algoritmos, a medida que la complejidad del problema se incrementa, la cantidad de pasos requerida crece menos que en la solución monolítica. Está documentado en la literatura técnica en computación

que, como se representa en la Figura 3.25, a partir de cierto grado de complejidad el enfoque de solución distribuida se hace cada más eficiente que el de la solución monolítica, aspecto denominado escalabilidad de la solución.

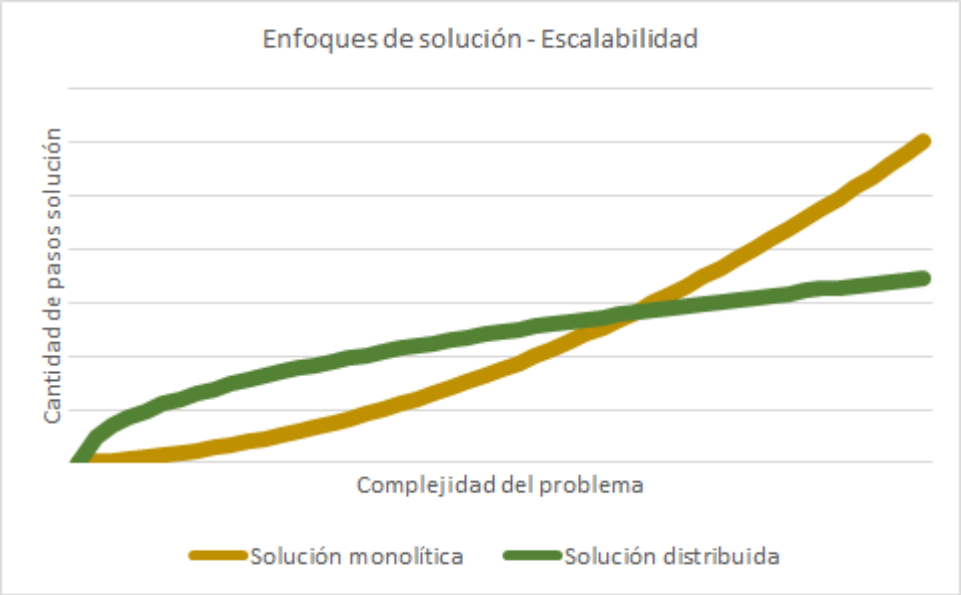


Figura 3.25 Enfoques de solución – Comparación de escalabilidad

Como segunda conclusión, el enfoque de solución distribuida cuesta un mayor esfuerzo inicial, pero se hace más eficiente a medida que se incrementa la complejidad del problema. En consecuencia, los ejemplos propuestos para el desarrollo de la metodología propuesta en este libro están pensados en el contexto de un nivel de complejidad superior al de un ejercicio típico de nivel introductorio a la programación. En efecto, en el capítulo siguiente se propone un caso de aplicación de una complejidad significativa, cuya solución se abordará de forma gradual, esperando demostrar a medida que avanzan los capítulos restantes las ventajas del enfoque en cuanto a la escalabilidad.

La implementación de la solución consiste en la codificación de los algoritmos en un lenguaje de programación. Para el enfoque de solución monolítica se empleó el lenguaje Python, en tanto que para la solución distribuida se utilizó el lenguaje Java. La Figura 3.26 muestra la comparación del número de líneas de código que requirieron las 3 versiones en los dos enfoques de solución.

Para la primera versión, ***Programa 1*** consta de 10 líneas de código en Python. Por su parte ***Programa 1B*** está conformado por 67 líneas de código en Java. Para la segunda versión el número de líneas de código se incrementó hasta 23 y 94 respectivamente. En la versión final, ***Programa 3*** consta de 27 líneas en tanto que ***Programa 3B*** tiene 103.

Es notorio que las diferencias son aún más grandes que en la comparación por cantidad de pasos de los algoritmos. Una causa obvia es la cantidad de pasos que conforman los algoritmos como se acaba de mostrar en los párrafos anteriores en esta misma sección.

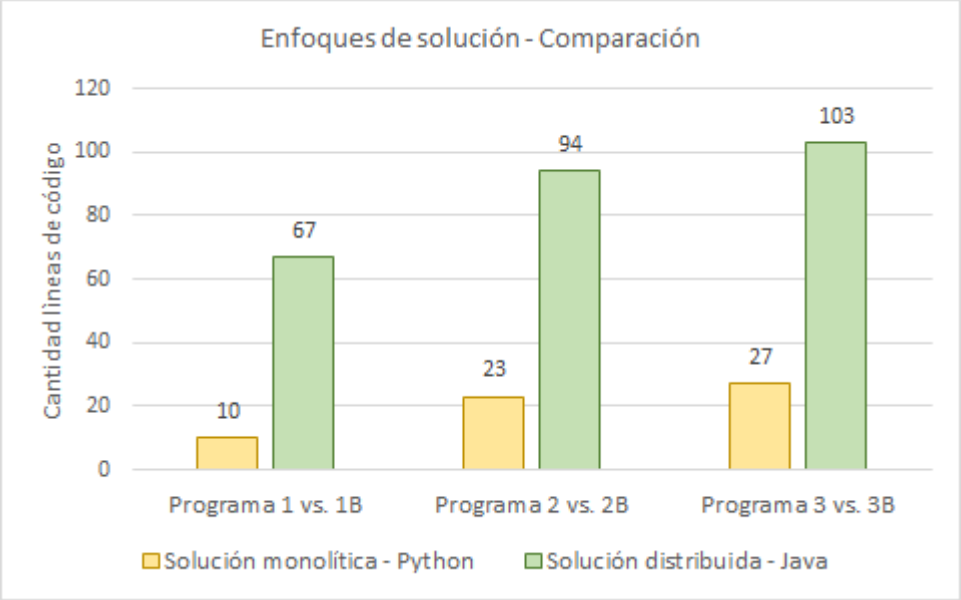


Figura 3.26 Enfoques de solución - Comparación por cantidad de líneas de código

Otro aspecto está relacionado con la sintaxis de los lenguajes de programación y con una serie de prácticas consideradas convenientes. En la Tabla 3.7 se puede observar que el código en lenguaje Python es bastante compacto; en contraste, en la Tabla 3.13 se puede observar el uso frecuente de líneas en blanco para separar diferentes secciones, así como líneas que tan solo contienen llaves de cierre "*}*" para delimitar algunos bloques. Tanto las líneas en blanco como las que tienen llaves de cierre constituyen una fracción significativa del total de líneas. Aunque su uso no es estrictamente obligatorio, su inclusión ayuda a que el archivo de código resulte más legible para el programador, ya sea que se trate del autor del código o de un analista diferente.

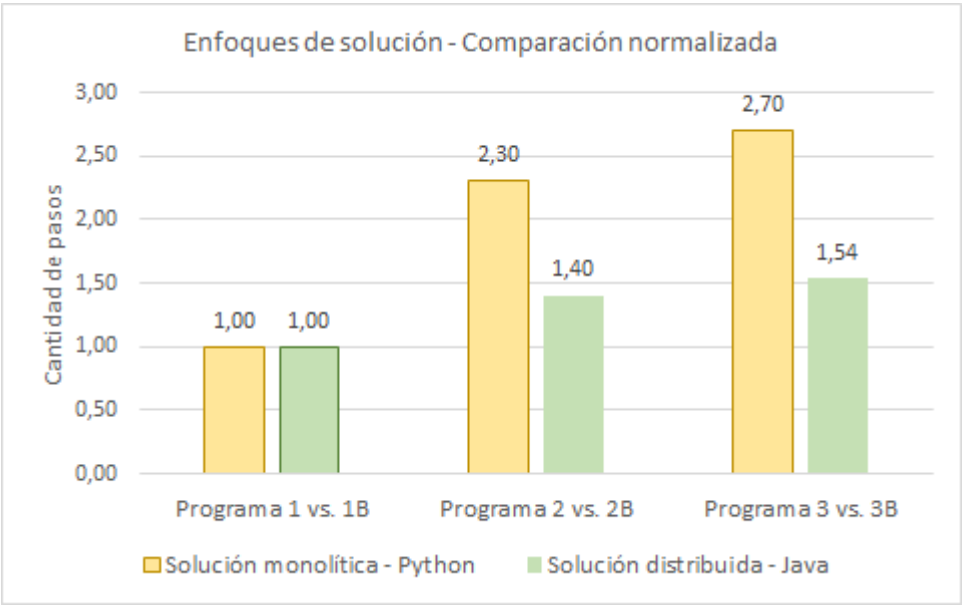


Figura 3.1 Enfoques de solución - Comparación normalizada por cantidad de líneas de código

Como resulta evidente la adopción un enfoque de solución distribuida, como la Arquitectura de Capas, no persigue una meta de eficiencia entendida como economía de líneas de programación. En cambio, se busca que el proyecto completo, y desde luego las líneas de código, sean legibles y estén distribuidas en secciones que faciliten su revisión y documentación, así como la identificación del lugar en que se deban realizar las correcciones o actualizaciones que lleguen a ser requeridas, característica conocida en Ingeniería del Software como mantenibilidad.

4 Introducción a la Arquitectura de Capas

Si bien este proyecto promueve el aprendizaje inicial de la programación orientada a objetos, incorpora en la primera etapa del proceso la introducción a la Arquitectura de Capas, como contexto para la comprensión de la complejidad potencial de un desarrollo de software. Aunque la arquitectura de software como concepto suele dejarse para cursos más avanzados en el currículo de Ingeniería de Sistemas, durante la formulación y planeación de este proyecto fue considerada propicia para que los estudiantes pudieran construir competencias de diseño de aplicaciones y hábitos rigurosos de documentación, al igual que para fomentar la apreciación del software como proyecto complejo, que además de la programación requiere operaciones futuras de mantenimiento y actualización (Yongbin & Jiayin, 2013).

Desde una perspectiva pedagógica, esperar a que los estudiantes desarrollen un nivel suficiente de experiencia en programación, para luego orientarlos al uso de una arquitectura de software, resulta en una carga considerable de trabajo en los cursos avanzados, sólo para sacarlos de su zona de confort en la que toda la lógica del software está contenida en un solo programa. Los estudiantes de fundamentos de programación usualmente no cuentan con los saberes necesarios para comprender la Arquitectura de Capas desde la perspectiva del desarrollo de software, pero el objetivo de la metodología aquí propuesta es la incorporación de técnicas y prácticas de Ingeniería del Software desde los cursos introductorios; en concordancia se hará a continuación una presentación de la arquitectura desde el punto de vista del usuario del software.

4.1 Caso de estudio: la aplicación “Geoportal DANE”

El Departamento Administrativo Nacional de Estadística DANE tiene como objetivos garantizar la producción, disponibilidad y calidad de la información estadística estratégica, y dirigir, planear, ejecutar, coordinar, regular y evaluar la producción y difusión de información oficial básica. Sus funciones están orientadas a: producción de estadísticas estratégicas; síntesis de cuentas nacionales; producción y difusión de información oficial básica; difusión y cultura estadística (Presidencia de la República, 2004).

En la dirección web www.dane.gov.co el DANE, mediante un portal web, pone a disposición del usuario información estadística de diversa índole, de acuerdo con los objetivos mencionados en el párrafo anterior. Como se indica en la Figura 4.1, entre los servicios al ciudadano se ofrecen variados servicios de información, entre los cuales está “Geoportal”.

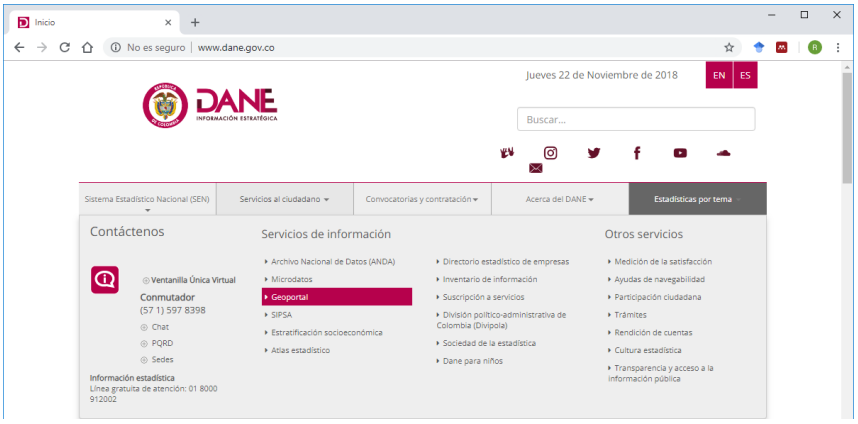


Figura 4.1 Fragmento de captura de pantalla de portal web del DANE

La opción del menú indicada en la figura anterior conduce a un nuevo sitio denominado Geoportal DANE, por el que se accede al sistema de información geográfica de la institución, que integra la información estadística georreferenciada del país. Existe una enorme cantidad de información; para el caso de estudio propuesto en este capítulo se utilizará la información accesible en la opción “Estimaciones y proyecciones de población” en la opción “Geovisualización de Datos” del menú de este sitio, como se muestra en la Figura 4.2.

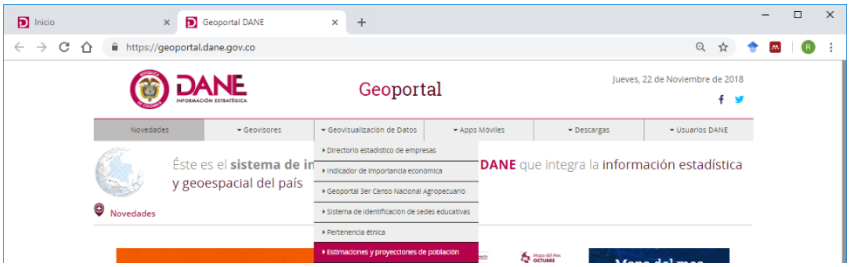


Figura 4.2 Fragmento de captura de pantalla de Geoportal DANE

Esta opción dirige al usuario a otro sitio donde encuentra información general del proyecto “Geoportal” y sus opciones de visualización, entre las cuales se encuentra la “Distribución de la población total, según departamento”, a la que corresponde la captura de pantalla de la Figura 4.3.

Figura 4.3 Fragmento de captura de pantalla Geoportal DANE – Estimaciones y proyecciones de población

Pulsando el botón “Ver más” de la parte inferior derecha de la sección mostrada en la Figura 4.3, se llega a la aplicación propuesta como caso de estudio en este capítulo: la geovisualización de las estimaciones y proyecciones de población del DANE, que tiene el aspecto mostrado en la Figura 4.4.

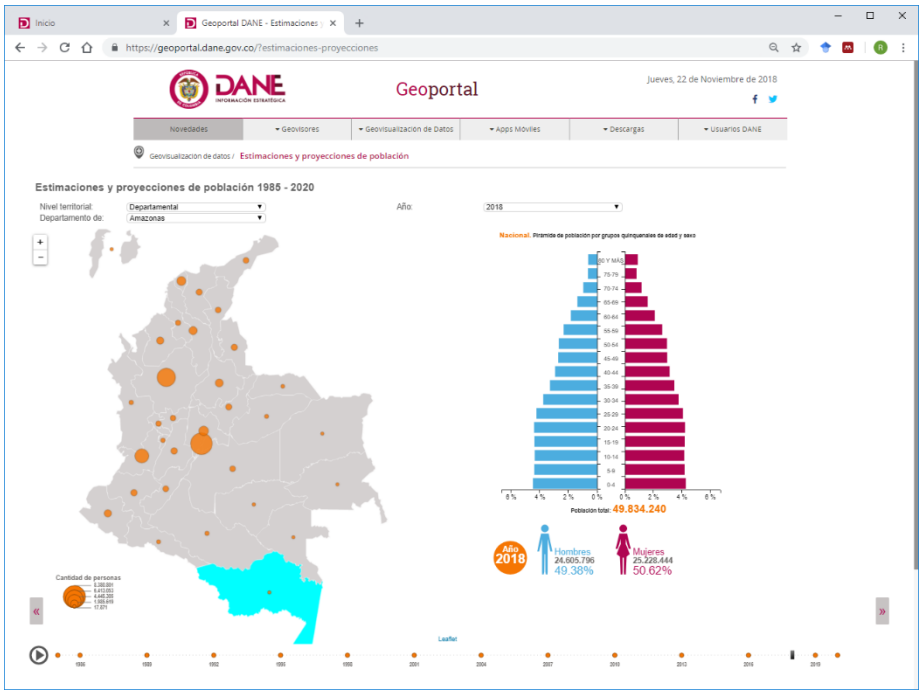


Figura 4.4 Captura de pantalla de geovisualización de datos en Geoportal DANE

En la parte superior izquierda se puede seleccionar entre las opciones “Departamental” y “Municipal” para el nivel territorial. Luego se puede seleccionar en una lista el departamento sobre el que se requiere la consulta de datos. En la parte superior derecha se puede escoger el año a que corresponde la consulta, que para lo que resta de este capítulo será el año 2018. En la parte derecha se muestra la información a nivel nacional, es decir, la cantidad de hombres y de mujeres que, según las proyecciones del DANE, conforman la población total de Colombia. Se presentan barras horizontales para indicar la cantidad de hombres y mujeres por rangos de edad, pero esta no será tenida en cuenta para el presente caso de estudio.

Los resultados preliminares del Censo Nacional de Población y Vivienda 2018 publicados para la época en que este capítulo ha tenido su última revisión, han arrojado diferencias significativas con respecto a las proyecciones. Pero ese no es un problema concerniente al propósito del capítulo ni del libro. Para efectos del caso de estudio se siguen utilizando las proyecciones presentadas en Geoportal DANE, como las que aparecen en la captura de pantalla de la Figura 4.4, según las cuales la población colombiana está compuesta por 24’605.796 hombres y 25’228.444 mujeres, correspondientes al 49,38% y 50,62% respectivamente, del total de 49’834.240 personas que son la población total proyectada del país en el año 2018.

4.2 Aproximación a la Arquitectura de Capas desde la funcionalidad

El estudiante que apenas inicia su aprendizaje en programación de computadores carece de las bases para construir el concepto de Arquitectura de Capas desde la perspectiva del desarrollo de software. No obstante, es muy probable que tenga ya bastante experiencia utilizando aplicaciones, que pueda aprovecharse para construir el concepto desde la perspectiva del usuario, y que luego pueda utilizarlo como esquema para el diseño de aplicaciones.

Una forma de aproximarse al concepto de arquitectura de capas es identificar la secuencia de operaciones que se ejecutan durante el funcionamiento de una aplicación, desde el punto de vista del usuario. A continuación, se dará un vistazo a la interacción del usuario con el caso de estudio propuesto: la aplicación Geoportal DANE.

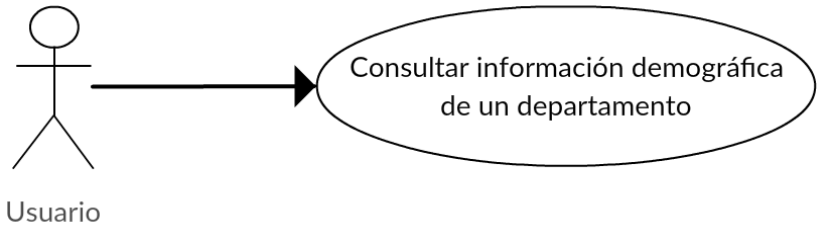


Figura 4.5 Diagrama de caso de uso de la aplicación Colombia en cifras

La interacción entre una persona y el computador puede representarse mediante diagramas de casos de uso, como el que se muestra en la Figura 4.5. En este diagrama se presenta el usuario y el propósito que tiene al utilizar la aplicación Geoportal DANE. En la Tabla 4.1 se mencionan acciones efectuadas tanto por el usuario como por la aplicación, descritas en pasos específicos.

Tabla 4.1 Especificación del caso de uso *Consultar información demográfica de un departamento*.

1	El usuario selecciona la opción “Departamental” en la lista “Nivel territorial”.
2	El usuario selecciona el departamento a consultar en la lista “Departamento”.
3	El usuario selecciona en la lista “Año” el que corresponda a la consulta requerida.
4	La aplicación muestra en pantalla los datos demográficos correspondientes al departamento y año seleccionados: cantidad de hombres, cantidad de mujeres, porcentaje de hombres, porcentaje de mujeres y población total del departamento.
5	El usuario pulsa el botón *Aceptar* para terminar la ejecución de la aplicación

Se puede realizar la consulta de información demográfica siguiendo los pasos de la especificación del caso de uso, presentados en la tabla anterior. Para el departamento de Santander y el año 2018, los resultados son los que aparecen en la Figura 4.6.

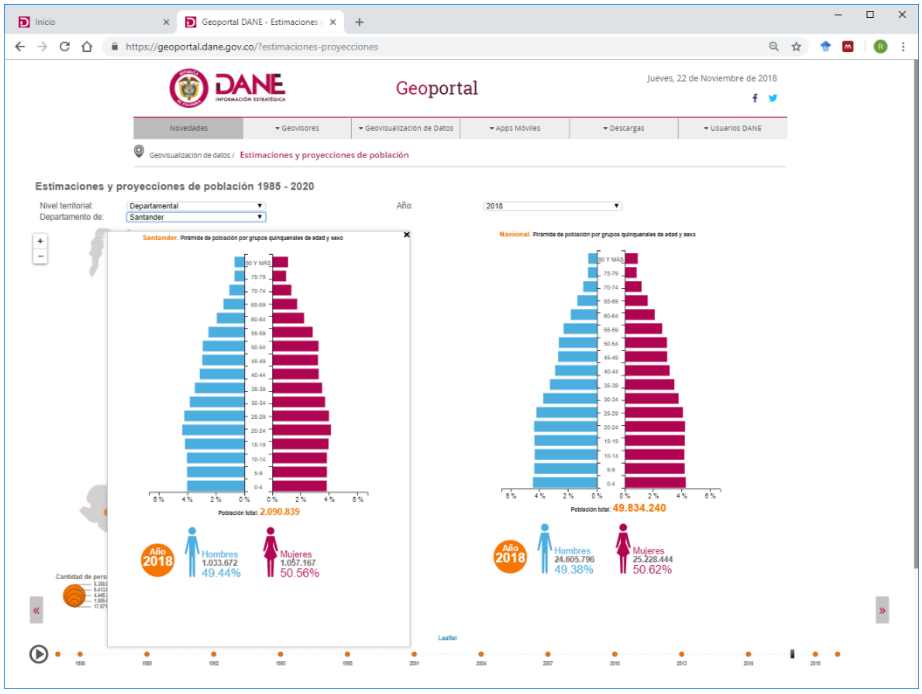


Figura 4.6 Pantalla de Geoportal DANE con datos del departamento de Santander para el año 2018

4.3 Secuencia de operaciones de la aplicación

Una forma de aproximarse al concepto de arquitectura de capas es identificar la secuencia de operaciones que se efectúan en el proceso mostrado anteriormente, desde el punto de vista del usuario de la aplicación.

Tabla 4.2 Secuencia de operaciones en el uso de la aplicación Geoportal DANE.

	Aplicación	Usuario
1	Presentación en pantalla de los elementos necesarios para que el usuario seleccione el departamento y año de los cuales requiere información	
2		Utilización del *mouse* o del teclado para seleccionar el departamento y el año cuyos datos requiere.
3	Captura del nombre y el año sobre los cuales el usuario ha pulsado con el *mouse*	
4	Preparación de los recursos necesarios para procesar datos de un departamento	
5	Búsqueda y recuperación de los datos del departamento en un medio de almacenamiento	
6	Procesamiento de los datos recuperados del medio de almacenamiento	
7	Presentación en pantalla de los datos del departamento solicitado por el usuario	
8		Revisión de los datos y utilización del *mouse* o del teclado para iniciar el proceso nuevamente con otro departamento, o para cerrar la aplicación

Los pasos 1, 2, 3, 7 y 8 son evidentes para el usuario, por cuanto implican su participación directa observando el monitor o utilizando dispositivos como el *mouse* y el teclado.

En cuanto al paso 5, el usuario debe suponer que para que los datos del departamento y del año que solicitó pudieran ser presentados, previamente alguien tuvo que haberlos registrado o grabado en un medio de almacenamiento, de una forma comparable a como el mismo usuario graba por ejemplo un trabajo escrito mediante un editor de texto, el cual queda disponible para cuando quiera recuperarlo de nuevo con el editor, y es además visible en el explorador de archivos del sistema operativo, por lo que puede ver en qué parte del disco está almacenado. Previamente algunos funcionarios del DANE debieron grabar los datos demográficos de todos los departamentos, y cuando el usuario lo requiere la aplicación en este paso 5 los busca y recupera.

Los pasos intermedios 4 y 6 posiblemente no sean tan evidentes para el usuario; haría falta explicar entonces que entre los pasos que suceden en monitor, teclado y *mouse* con los que el usuario interactúa directamente, y los que se efectúan en el medio de almacenamiento del lado de la máquina, deben realizarse algunas funciones de preparación y procesamiento que suceden en la memoria del computador, y que corresponden a dichos pasos 4 y 6 de la secuencia. En la memoria del computador, la aplicación prepara el esquema correspondiente a un departamento, para recibir la solicitud del usuario y trasladarla en forma apropiada como solicitud de recuperación de datos, y en donde luego los datos recuperados son procesados para presentarlos como información relativa a un departamento.

La Figura 4.7 muestra esta misma secuencia, asociada a los dispositivos con que se realiza cada uno de los pasos.

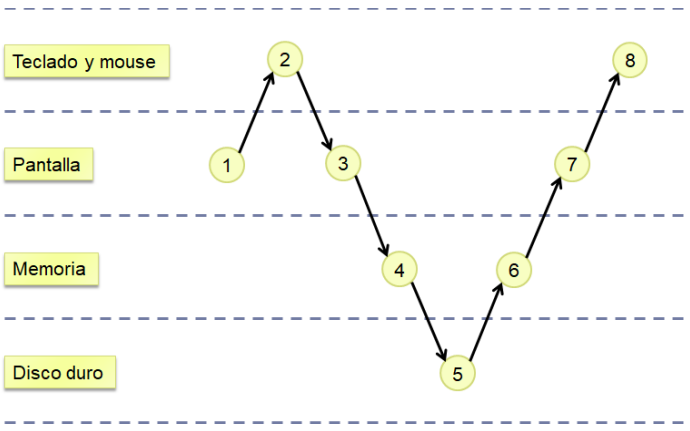


Figura 4.7 Secuencia de operaciones y dispositivos con que se realizan

Con base en la Figura 4.7 se puede apreciar que las operaciones se realizan en niveles a diferentes distancias con respecto al usuario: el nivel superficial lo conforman los dispositivos con los que el usuario interactúa directamente, tales como monitor, teclado, mouse e impresoras; el nivel más profundo lo conforman los medios de almacenamiento de datos, como por ejemplo el disco duro, a los cuales el usuario no tiene acceso físico directo; y un nivel intermedio en el que se encuentra la memoria del computador, que hace un puente entre las dos, pero que como se verá más adelante es mucho más compleja que eso.

4.4 Primera aproximación a las capas en arquitectura de software

Cada nivel de profundidad mencionado en el apartado anterior corresponde a lo que en arquitectura de software se denomina capa. Dichas capas, en orden de menor a mayor profundidad con respecto al usuario, reciben los nombres de capa de presentación, capa de reglas y capa de datos, cada una especializada en cierto tipo de operaciones, como se muestra en la Figura 2.11.

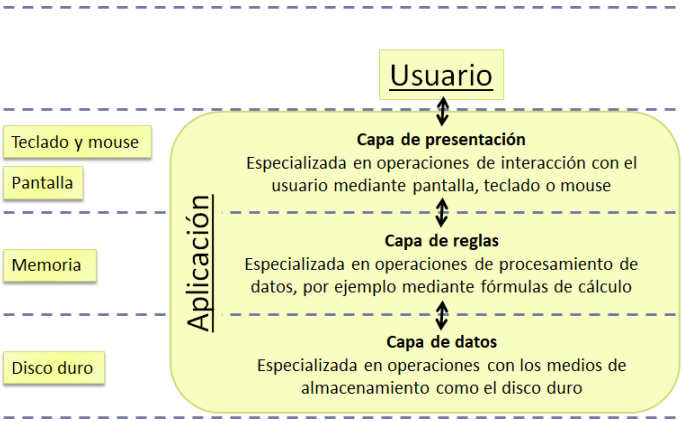


Figura 4.8 Esquema básico de una aplicación en Arquitectura de Capas

Dado esto, es pertinente para los estudiantes de los cursos introductorios a la programación de computadores en el contexto de la Ingeniería de Sistemas o carreras afines, asumir que su trabajo a partir de ahora es pensar ya no como usuarios sino como productores de software, que existen unos principios a seguir cuando se desarrolla una aplicación, como con cualquier otra obra de ingeniería, ya sea un edificio, una máquina, una red eléctrica o de comunicaciones.

Como desarrolladores de software, si se opta por la arquitectura de capas, las aplicaciones se diseñan e implementan teniendo en cuenta principios de especialización de operaciones, lo que determina una serie de requisitos para el diseño y la construcción de las aplicaciones.

Todo este análisis no implica que Colombia en cifras haya sido desarrollada siguiendo la arquitectura de capas, porque existen otras arquitecturas, aunque probablemente haya sido así. Lo que se sugiere hasta ahora es que una aplicación similar se podría desarrollar en arquitectura de capas, que es el enfoque propuesto para este libro.

4.5 “Demografía Colombiana”, una réplica de “Geoportal DANE”

Se presentará ahora una aplicación que replica parcialmente la funcionalidad que se ha mostrado de “Geoportal DANE”, pero diseñada en arquitectura de capas. Se llamará en adelante Demografía Colombiana, e incluye también las herramientas para el ingreso y grabación de datos, indispensable para que se puedan luego ejecutar las consultas. Para mantener la simplicidad del ejemplo, los datos demográficos de cada departamento se reducen a la cantidad de hombres y de mujeres; con base en estos la aplicación calcula la población total. La Figura 4.9 presenta un diagrama de la aplicación réplica, con dos casos de uso: el primero representa al usuario en

función de registrar los datos demográficos de cada departamento; el segundo representa la función de realizar la consulta de información demográfica a nivel de departamento.

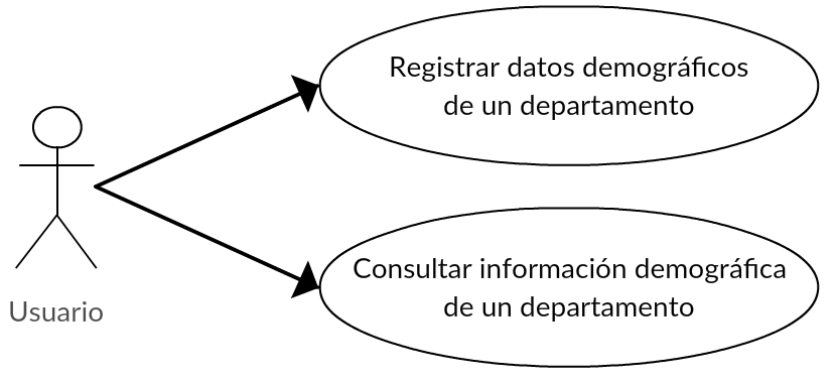


Figura 4.9 Diagrama de casos de uso de la aplicación Demografía Colombiana

4.5.1 Caso de uso "Registrar datos demográficos"

El caso de uso para registro de datos demográficos consta de la serie de pasos detallados en la Tabla 4.3.

Tabla 4.3 Especificación del caso de uso *Registrar datos demográficos de un departamento*.

1	El usuario ingresa a la aplicación y selecciona la opción "Registrar departamento".
2	La aplicación presenta una pantalla con el rótulo "Nombre del departamento" y una caja de texto para que el usuario lo digite.
3	El usuario digita el nombre del departamento y pulsa el botón "Aceptar".
4	La aplicación presenta una pantalla con el rótulo "Cantidad de hombres" y a continuación el nombre del departamento del que se están registrando los datos. También se presenta una caja de texto para que el usuario digite el dato correspondiente.
5	El usuario digita la cantidad de hombres del departamento y pulsa el botón "Aceptar".
6	La aplicación presenta una pantalla con el rótulo "Cantidad de mujeres" y a continuación del nombre del departamento del que se están registrando los datos. También se presenta una caja de texto para que el usuario digite el dato correspondiente.
7	El usuario digita la cantidad de mujeres del departamento y pulsa el botón "Aceptar".
8	La aplicación muestra una pantalla con los datos completos: el nombre del departamento, su cantidad de hombres y de mujeres, y la población total.
9	El usuario verifica los datos y pulsa el botón "Aceptar".
10	La aplicación graba los datos demográficos del departamento en el medio de almacenamiento.

La aplicación Demografía Colombiana muestra una ventana inicial como la de la Figura 4.10, que contiene las opciones disponibles para el usuario, un campo de edición para que el usuario digite el número correspondiente. Para registrar los datos demográficos de un departamento, el usuario debe digitar el número 1 y pulsar el botón Aceptar.

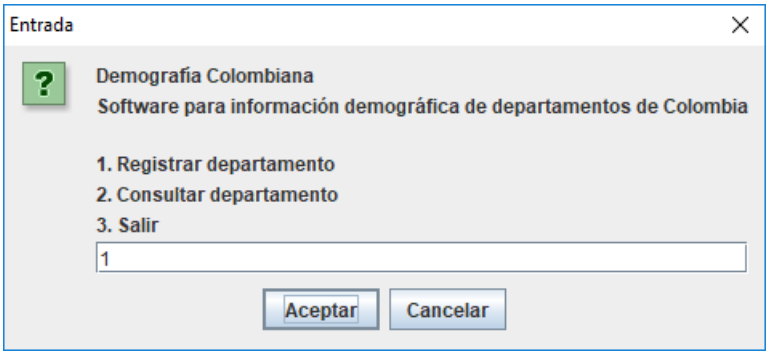


Figura 4.10 Ventana con opciones de la aplicación Demografía Colombiana

A continuación la aplicación muestra una serie de ventanas solicitando cada uno de los datos del departamento, empezando por la de ingreso del nombre del departamento, que se muestra en la Figura 4.11. El usuario autorizado debe digitar el dato correspondiente y pulsar el botón Aceptar.

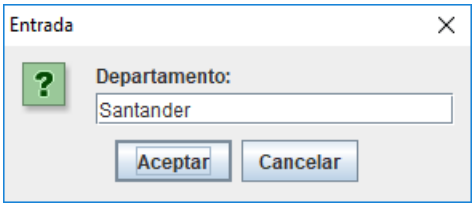


Figura 4.11 Ventana de registro del nombre del departamento

Una vez aceptado el nombre, la aplicación presenta una ventana para la cantidad de hombres del departamento, como se muestra en la Figura 4.12.

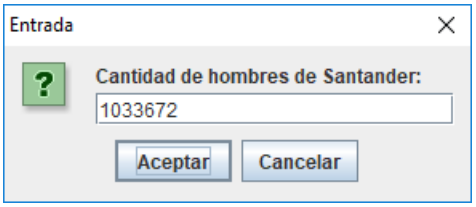


Figura 4.12 Ventana de registro de la cantidad de hombres del departamento

Asimismo, la aplicación presenta una ventana para la cantidad de mujeres que viven en el departamento, como se muestra en la Figura 4.13.

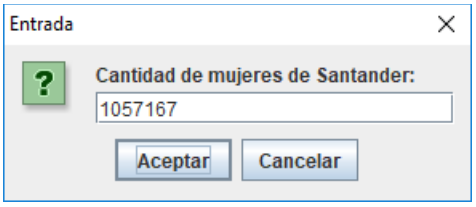


Figura 4.13 Ventana de registro de la cantidad de mujeres del departamento

Luego la aplicación calcula la población total y presenta todos los datos registrados, en una ventana como la que muestra la Figura 4.14.

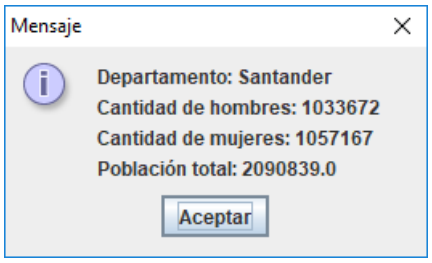


Figura 4.14 Ventana de presentación de la información demográfica del departamento

Sobre esta última ventana, el usuario verifica que los datos coinciden con los que digitó, y pulsa el botón Aceptar, luego de lo cual la aplicación realiza la grabación de estos en el medio de almacenamiento.

4.5.2 Caso de uso "Consultar información demográfica"

El caso de uso para consultar información demográfica en Demografía Colombiana está compuesto por los pasos detallados en la Tabla 4.4.

Tabla 4.4 Especificación del caso de uso *Consultar información demográfica de un departamento*.

1	El usuario ingresa a la aplicación y selecciona la opción "Consultar departamento".
2	La aplicación presenta una pantalla con el rótulo "Nombre del departamento a buscar" y una caja de texto para que el usuario lo digite.
3	El usuario digita el nombre del departamento a buscar y pulsa el botón "Aceptar".
4	La aplicación recupera los datos del medio de almacenamiento.
5	La aplicación muestra una pantalla con los datos completos: el nombre del departamento, su cantidad de hombres y de mujeres, y la población total.
6	El usuario verifica los datos y pulsa el botón "Aceptar".

En la aplicación Demografía Colombiana, que replica parcialmente a "Geoportal DANE", para consultar la información demográfica de un departamento se debe seleccionar la opción 2 del menú y luego el botón Aceptar, como muestra la Figura 4.15.

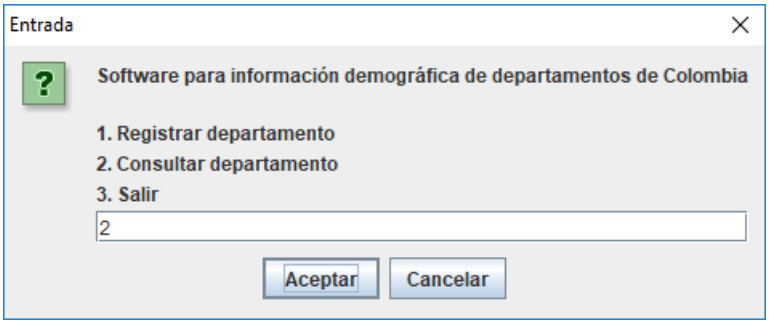


Figura 4.15 Ventana con opciones de la aplicación Demografía Colombiana

En seguida, como se muestra en la Figura 4.16, la aplicación presenta una ventana solicitando el nombre del departamento cuyos datos desea consultar el usuario, quien debe digitar el nombre correspondiente y pulsar el botón Aceptar.

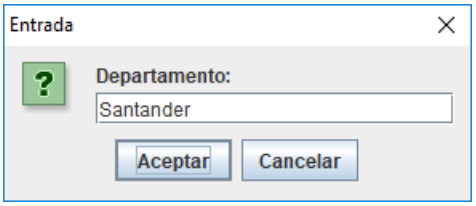


Figura 4.16 Ventana de ingreso del nombre del departamento a consultar

Una vez aceptado el nombre, la aplicación busca en el medio de almacenamiento, y si encuentra una coincidencia con el nombre digitado por el usuario, recupera los datos registrados, realiza los cálculos pertinentes, en este caso el de la población total, y en una nueva ventana presenta la información demográfica como se ve en la Figura 4.17.

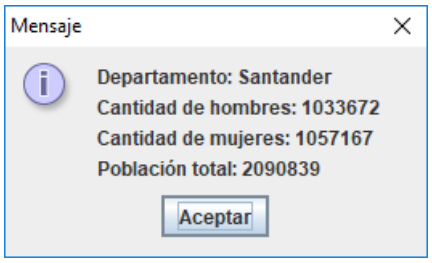


Figura 4.17 Ventana de consulta de información demográfica de un departamento

El modo en que el usuario interactúa con esta aplicación es poco utilizado en la actualidad. Durante la secuencia de ejecución aparecen pantallas diferentes, cada una de las cuales recibe un dato y procede al siguiente paso. Es efectiva pero poco eficiente, pues fragmenta el proceso para el usuario; sin embargo, es una forma relativamente fácil de implementar para el desarrollador, en consideración con la complejidad de las ventanas a las que un usuario actual está acostumbrado.

4.6 “Demografía Colombiana” versión 2

Con el propósito de dar mayores argumentos para el uso de la arquitectura de capas en este libro, se ofrece una segunda versión de la aplicación Demografía Colombiana, que implementa los mismos casos de uso pero con una especificación diferente de los pasos a seguir por parte del usuario, implementada con un tipo de interacción más eficiente.

4.6.1 Caso de uso "Registrar datos demográficos"

El registro de datos demográficos en Demografía Colombiana versión 2 consta de la serie de pasos detallados en la Tabla 4..

Tabla 4.5 Especificación del caso de uso *Registrar datos demográficos de un departamento* en la versión 2.

1	El usuario ingresa a la aplicación y selecciona la opción "Registro".
2	La aplicación presenta una pantalla con los rótulos y las cajas de edición para los tres datos de cada departamento: nombre, cantidad de hombres y cantidad de mujeres.
3	El usuario digita los datos correspondientes en cada caja de edición y pulsa el botón "Grabar".
4	La aplicación graba los datos demográficos del departamento en el medio de almacenamiento.
5	La aplicación muestra una pantalla con la información completa: el nombre del departamento, su cantidad de hombres y de mujeres, y la población total.

A diferencia de la versión 1, la segunda versión de la aplicación Demografía Colombiana ofrece en una sola ventana los elementos necesarios para el registro de los datos demográficos completos para un departamento, como se muestra en la Figura 4.18. El usuario debe digitar cada dato en la casilla correspondiente y pulsar el botón Grabar.

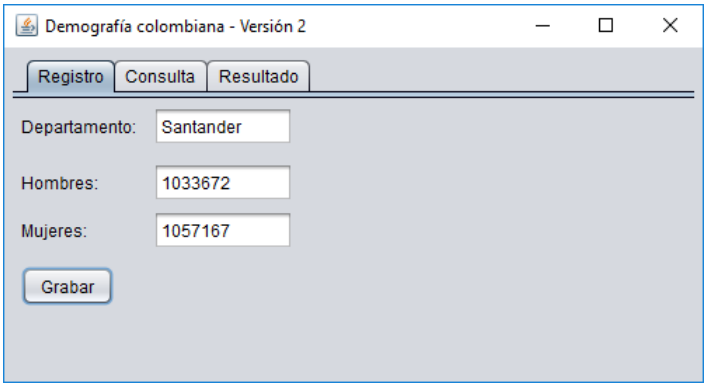


Figura 4.18 Pantalla de registro de datos demográficos en Demografía Colombiana versión 2

A continuación la aplicación presenta una ventana, como la de la Figura 4.19, con los datos recién grabados, para verificación por parte del usuario.

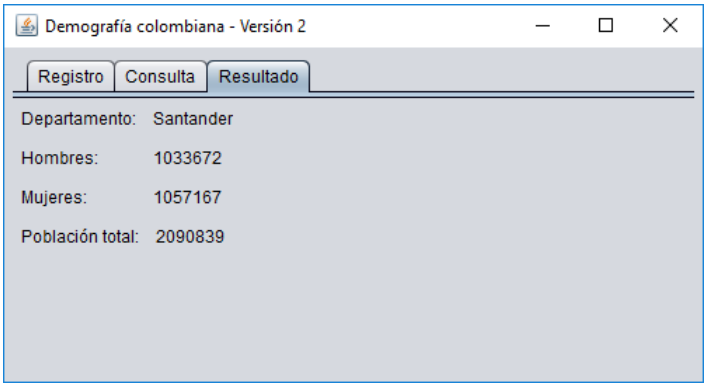


Figura 4.19 Ventana de presentación de la información demográfica en Demografía Colombiana versión 2

4.6.2 Caso de uso "Consultar información demográfica"

En cuanto al caso de uso para consulta de información demográfica, en la segunda versión de Demografía Colombiana consta de la serie de pasos especificados en la Tabla 4.5.

Tabla 4.5 Especificación del caso de uso *Consultar información demográfica de un departamento* en la versión 2.

1	El usuario ingresa a la aplicación y selecciona la opción "Consulta".
2	La aplicación presenta una pantalla con el rótulo "Departamento" y una caja de texto para que el usuario digite el nombre a buscar.
3	El usuario digita el nombre del departamento y pulsa el botón "Buscar".
4	La aplicación recupera los datos del medio de almacenamiento.
5	La aplicación muestra una pantalla con la información completa: el nombre del departamento, su cantidad de hombres y de mujeres, y la población total.

Como se muestra en la Figura 4.20, el usuario puede seleccionar la sección Consulta, y digitar el nombre de un departamento para realizar la consulta; luego debe pulsar el botón Buscar.

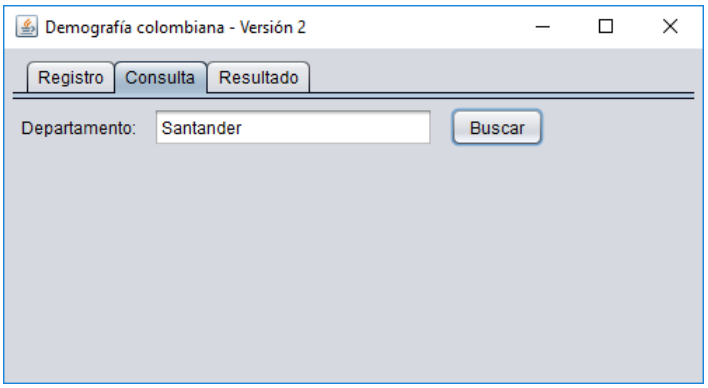


Figura 4.20 Pantalla de consulta de información demográfica en Demografía Colombiana versión 2

En respuesta, la aplicación busca en el medio de almacenamiento, y al encontrar una coincidencia con el nombre dado por el usuario, recupera los datos y realiza los cálculos, para presentar en la sección Resultado la información demográfica correspondiente, según se muestra en la Figura 4.21.

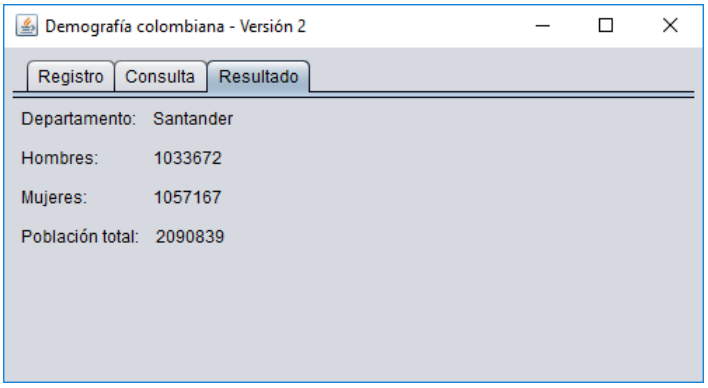


Figura 4.21 Pantalla de presentación de resultados de consulta en Demografía Colombiana versión 2

4.7 Análisis de "Demografía Colombiana" desde la arquitectura de capas

Aunque resulta obvio, es pertinente afirmar que las dos versiones de la aplicación Demografía Colombiana hacen lo mismo, aunque tengan ciertas diferencias, no de fondo sino de forma.

Como se resumió en el esquema básico de una aplicación en Arquitectura de Capas, mostrado en la Figura 4.8, la capa de datos se especializa en operaciones que involucran medios de almacenamiento. Las dos versiones de Demografía Colombiana son idénticas en la forma en que almacenan los datos de los departamentos, en archivos secuenciales; para quien sea novato en el asunto, bastaría por ahora explicar que es el tipo de archivo que se puede generar con una herramienta básica, presente en todos los equipos con sistema operativo Windows: el bloc de notas. En la Figura 4.22 se muestra un fragmento del explorador de archivos, con el archivo departamentos.txt ubicado en la carpeta correspondiente a la aplicación. En dicho archivo, abierto mediante el bloc de notas, se puede observar que contiene los datos de dos departamentos, uno de ellos el utilizado durante el seguimiento del ejemplo.

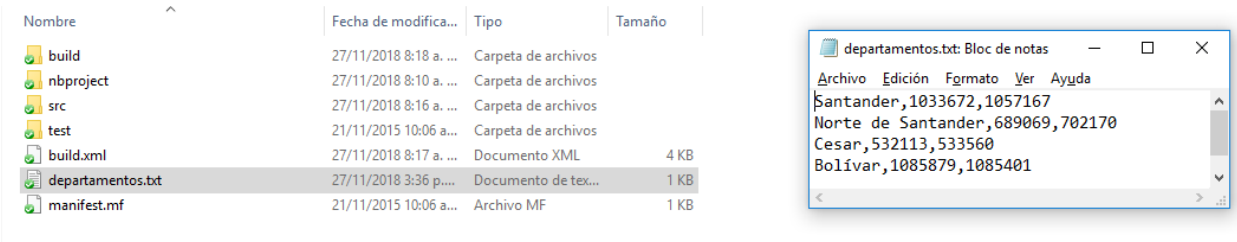


Figura 4.22 Almacenamiento de datos de la aplicación Demografía Colombiana

En cuanto a las operaciones de interacción con el usuario, que constituyen la especialidad de la capa de reglas, hay notorias diferencias entre las dos versiones de la aplicación.

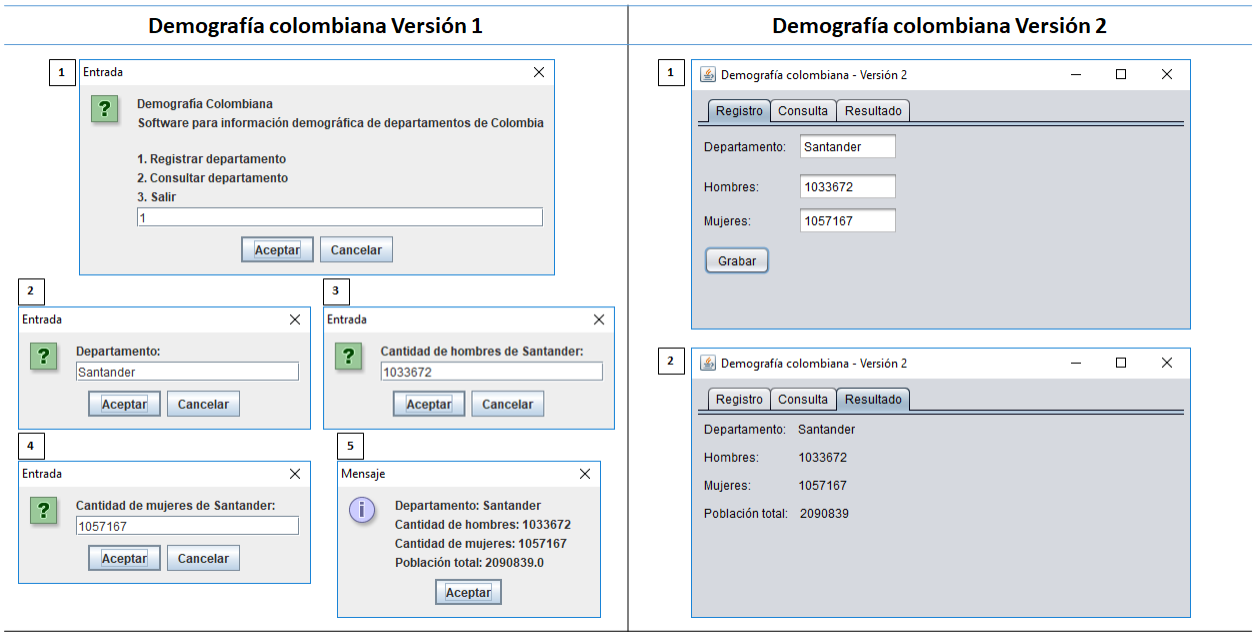


Figura 4.23 Diferencias de interacción con el usuario entre versiones de Demografía Colombiana

Mientras la primera se basa en pantallas secuenciales de ingreso de datos y habilitación del paso siguiente, la segunda ofrece una única ventana subdividida en secciones, que ofrece una visión más integral del proceso y requiere menos pasos de interacción. Para ilustrar estas diferencias entre versiones, en la Figura 4.23 se han incluido las secuencias de interacción correspondientes a cada versión, para el caso de uso de registro de datos demográficos.

Un aspecto sobre el cual se debe prestar atención es que en las dos versiones de la aplicación, al usuario que hace el registro se le solicitan 3 datos: el nombre del departamento, la cantidad de hombres y la cantidad de mujeres que viven en él, como se puede constatar en la Figura 4.23, los mismos que se graban en el medio de almacenamiento, es decir, en el archivo departamentos.txt como se puede verificar en la Figura 4.22. Sin embargo, volviendo a la Figura 4.23, la información demográfica completa de un departamento incluye la población total.

La capa de reglas se definió como la que se especializa en operaciones de procesamiento de datos, que puede entenderse como la ejecución de fórmulas de cálculo. En un sentido más completo, en la capa de reglas se define, para el caso de la aplicación Demografía Colombiana, que un departamento es una unidad, caracterizada por tres datos, el nombre, la cantidad de hombres y la cantidad de mujeres, pero también dotada de la capacidad para calcular la población total, sumando los dos últimos datos mencionados.

Puede afirmarse, luego de este análisis, que las dos versiones de Demografía Colombiana no son tan diferentes como su funcionalidad lo sugiere; por el contrario, tienen en común sus componentes de las capas de datos y de reglas, y sólo se diferencian en su capa de presentación, como se esquematiza en la Figura 4.24.

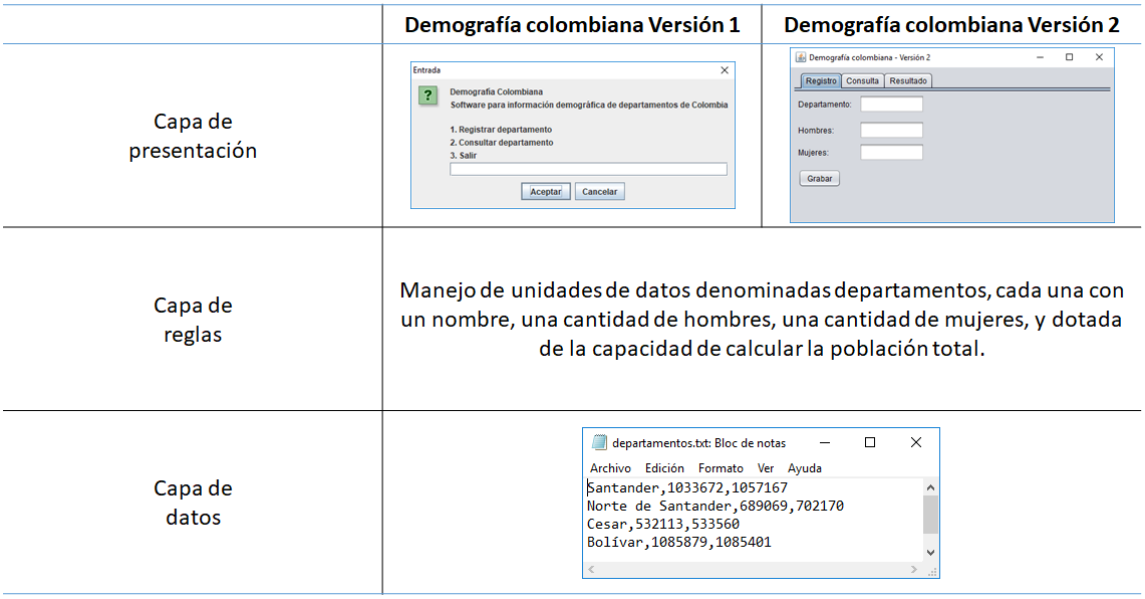


Figura 4.24 Comparación entre las dos versiones de Demografía Colombiana

Más allá de cuál de estas dos opciones es más conveniente para el usuario, lo que resulta pertinente para la formación inicial en programación de computadores es que los aspectos de interacción entre el usuario y la aplicación son solo una parte de la complejidad de esta, y que

pueden ser cambiadas en forma relativamente fácil de una versión a otra, y que el desarrollo de una aplicación puede pensarse desde la perspectiva de capas especializadas por funciones.

4.8 Comparación entre "Geoportal DANE" y "Demografía Colombiana" versión 2

Desde luego, la aplicación "Geoportal DANE" es más compleja de lo que aquí se ha presentado, y en cambio Demografía Colombiana es quizás un nombre excesivamente pretencioso en relación con su limitada funcionalidad. Pero manteniendo esta salvedad, las aplicaciones hacen lo mismo, y la comparación que se hace a continuación sigue basándose en la suposición de que Colombia en cifras haya sido desarrollada en arquitectura de capas.

Con respecto a la capa de presentación, las diferencias son evidentes. La aplicación del DANE está disponible en un sitio web, y ofrece a los usuarios elementos visuales que le permiten una navegación intuitiva, seleccionando los departamentos por su ubicación en el mapa o en un listado. Con la segunda versión de Demografía Colombiana, el usuario todavía debe digitar los datos y pulsar un botón para continuar.

En cuanto a la capa de datos, con toda seguridad los de "Geoportal DANE" residen en una base de datos y no en un archivo en formato .txt como en Demografía Colombiana. Estos archivos son útiles dentro de ciertas condiciones, pero cuando la cantidad o la complejidad de los datos se incrementan, y aspectos como la seguridad, disponibilidad y tiempo de respuesta en el acceso adquieren importancia, las bases de datos son una opción más robusta.

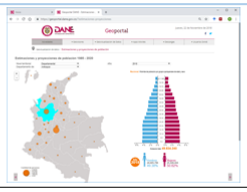
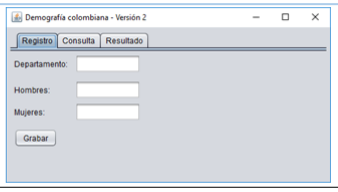
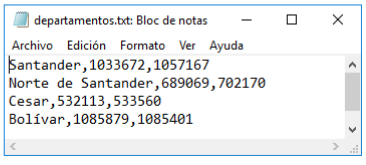
<table>
<tr><th></th><th>Geoportal DANE</th><th>Demografía colombiana Versión 2</th></tr>
<tr><td>Capa de presentación</td><td>  </td><td> 
</td></tr>
<tr><td>Capa de reglas</td><td colspan="2">Manejo de unidades de datos denominadas departamentos, cada una con un nombre, una cantidad de hombres, una cantidad de mujeres, y dotada de la capacidad de calcular la población total.</td></tr>
<tr><td>Capa de datos</td><td>Probablemente una base de datos</td><td> 
</td></tr>
</table>

Figura 4.25 Comparación entre Geoportal DANE y Demografía colombiana versión 2

Pero en medio de estas dos capas, es decir, en la capa de reglas, a pesar de que esta comparación se basa en la suposición, no comprobable pero sí plausible, del uso de capas por parte de los desarrolladores de la aplicación del DANE, hay una buena probabilidad de que las

aplicaciones tengan en común la lógica que les permite el manejo de departamentos con los datos y operaciones ya antes mencionadas, o que, al menos, sería posible hacer una tercera versión de Demografía Colombiana, conservando la capa de reglas de las dos anteriores y modificando las capas de datos y de presentación para que la réplica sea completa. El asunto no es trivial: implica que una parte importante del esfuerzo que se realiza para desarrollar una aplicación, la capa de reglas, no requiere de cambios para adaptarla a diferentes formas de interacción con el usuario y también con diferentes formas de almacenamiento de los datos.

4.9 Abstracción de una aplicación en arquitectura de capas

Desde la perspectiva del usuario, el ingreso a una aplicación puede ser comparado con el ingreso a una oficina o dependencia en donde requiere que se le brinde un servicio.

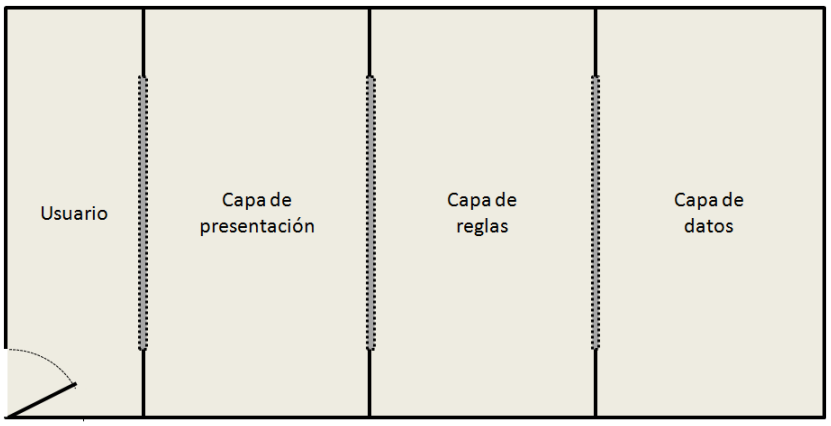


Figura 4.26 Metáfora de oficina para la arquitectura de capas

Se propone esta metáfora suponiendo que el estudiante de un curso apenas introductorio a la programación quizás esté más familiarizado con la arquitectura de una edificación que con la arquitectura de un software, en tanto que implica algo de su entorno físico tangible.

Para el caso de la arquitectura de capas, la aplicación se asemeja a una oficina como la representada en la Figura 4.26. Existe un área de acceso a la que el usuario puede ingresar, y desde donde solo tiene comunicación directa con la capa de presentación; cada subdivisión de la oficina sólo está comunicada con las que le son adyacentes.

Antes de resumir lo visto en el capítulo, vale la pena agregar una precisión sobre lo que se busca en los siguientes. En el enfoque tradicional de estudio de la programación, al estudiante se le formulan problemas que debe resolver con un programa: un programa que calcule el área de un círculo de radio conocido, un programa que determine el mayor en una secuencia de n números enteros, entre otros problemas típicos. Uno de los retos para esta metodología es que el estudiante asuma que, en el momento actual de evolución de la informática, con la complejidad y la masificación que ha alcanzado, posiblemente utilizar el “programa” para referirse al producto no sea lo más conveniente; al menos en el contexto de la arquitectura de capas en este libro se habla de la “aplicación” como unidad, en la que las responsabilidades o funciones están

distribuidas entre varios programas; una aplicación entonces viene a ser un sistema de programas que juntos solucionan un problema.

En el próximo capítulo se aborda el diseño de aplicaciones en arquitectura de capas, para lo cual el estudiante debe poder pasar de la visión del software desde la perspectiva del usuario con la que empezó este capítulo, a la perspectiva de desarrollador. Esta sección resume lo que debe tener claro para empezar dicho trabajo de diseño.

La arquitectura de capas tiene como propósito eludir la aparición de problemas en el desarrollo de software, especialmente cuando se trata de proyectos complejos. Una arquitectura de tres capas (hay propuestas con una cantidad mayor de capas) se constituye de capa de presentación, capa de datos y capa de reglas, nombres estos que se utilizan en este texto por su simplicidad y para uso en una etapa introductoria de la formación de los estudiantes; las mismas capas son conocidas en el ámbito académico por nombres como capa de interfaz de usuario, capa de acceso a datos y capa de lógica de negocios, respectivamente.

La capa de presentación es la capa más externa del software, es decir, la capa con la que el usuario entra en contacto directo, y se especializa en proveer a los usuarios los datos y los métodos de interacción con la aplicación (Yongbin & Jiayin, 2013). En la capa de reglas se diseñan e implementan las consideraciones centrales de la solución de un problema, como por ejemplo los datos requeridos y las fórmulas de cálculo, o la lógica de procesamiento. En la capa de datos se implementan las operaciones con los medios de almacenamiento del computador, como su disco duro, por ejemplo. Esta capa se encarga de guardar y/o buscar datos en archivos, o sea, de comunicar a la capa de datos con los dispositivos de almacenamiento.

Referencias

Presidencia de la República de Colombia. Decreto 262 de 2004, (2004).

Yongbin, Q., & Jiayin, W. (2013). The solution of enterprise ERP based on six-tier architecture. *Proceedings - 2013 International Conference on Computational and Information Sciences, ICCIS 2013*, 616–618. https://doi.org/10.1109/ICCIS.2013.169

5 Formulación del caso de estudio

En el capítulo anterior se utilizó la aplicación "Geoportal DANE" como caso de estudio para introducir la Arquitectura de Capas. Para el propósito de este libro no es práctico hacer una réplica total de su funcionalidad. La aplicación "Demografía Colombiana" en sus dos versiones fue delimitada en su alcance a replicar solo una parte de la funcionalidad, permitiendo la captura y almacenamiento de tres datos por cada departamento (nombre, cantidad de hombres y cantidad de mujeres), y el cálculo de la población total como complemento a la información presentada al usuario.

Con la salvedad de la delimitación ya descrita, se evidenció que la misma funcionalidad pero bajo una apariencia más sencilla, incluso pobre teniendo en cuenta que en "Geoportal DANE" el usuario dispone de un mapa para seleccionar el departamento y la presentación de los datos está enriquecida con gráficos, el aspecto central de la aplicación que es en efecto la definición de los datos y cálculos a manejar para cada departamento, así como el almacenamiento de los mismos, se puede obtener de una aplicación en Arquitectura de Capas, es decir, distribuyendo los elementos componentes de la aplicación de acuerdo a tres especialidades: interacción con el usuario mediante una capa de presentación, definición de datos y cálculos en una capa de reglas, e interacción con el medio de almacenamiento mediante la capa de datos. Con la segunda versión de "Demografía Colombiana" se aportó evidencia de la conveniencia de este enfoque, en tanto que se puede mejorar la interacción con el usuario sin tener que hacer un desarrollo nuevo, sino reutilizando las capas de reglas y datos y remplazando la capa de presentación.

A partir del siguiente capítulo se presenta la metodología propuesta en el marco del proyecto de investigación en el que se origina este libro, expresada en forma de una secuencia didáctica recomendada para el estudio de la Programación Orientada a Objetos aprovechando los principios de la Arquitectura de Capas. En este capítulo se formula a nivel general el caso de estudio sobre el que se basa la presentación de la secuencia didáctica mencionada, caso de estudio formulado a partir de una problemática de interés general, que asimismo compromete la construcción de competencias ciudadanas, y para el que se cuenta con una solución de referencia a gran escala, la cual va a ser delimitada con propósito didáctico.

5.1 Participación electoral en Colombia

Los ciudadanos colombianos participan periódicamente en la elección, mediante voto popular directo, del presidente de la República, los congresistas, y las autoridades de los poderes ejecutivo y legislativo a nivel municipal y departamental.

El costo total de las elecciones del año 2018 fue estimado en 1,2 billones de pesos (Referencia pendiente proyecto de ley 057/2017 Senado, Ley 1873 de 2017, Artículo El Tiempo). En cada

proceso electoral nacional llevado a cabo en el año 2018 estuvieron habilitados para votar aproximadamente 37’700.000 personas. La votación fue de 17’818.185 en la elección de Senado, 19’636.714 en la primera vuelta presidencial, y 19’511.168 en la segunda vuelta presidencial, para un total acumulado de 56’966.067 votos en los tres comicios, de un potencial de más de 113’000.000.

Esto implica que el Estado gastó más de $21.065 por cada voto, lo cual a juicio de amplios sectores de opinión es demasiado costoso, y debería ser tenido en cuenta para cambiar el sistema actual de urna física por uno de voto electrónico, lo que ahorraría el costo de producción, transporte y seguridad de los tarjetones electorales que en su mayoría no son utilizados y terminan siendo destruidos. Para otros sectores el problema más grave no es el costo mismo de las elecciones, sino la abstención, es decir, la poca participación de la ciudadanía. Para soportar este argumento, se han impulsado estudios para determinar la participación de los votantes y la validez de los votos en los distintos departamentos, tomando como base segunda vuelta electoral para Presidencia de la República, llevada a cabo el 17 de junio de 2018.

5.2 Aplicación de resultados electorales Registraduría Nacional del Estado Civil

Todos los datos de la sección anterior han sido consultados en la aplicación que sirve como referencia al caso de estudio a desarrollar en los siguientes capítulos: la aplicación de presentación de resultados electorales de la Registraduría Nacional del Estado Civil, de la que se muestra una imagen en la Figura 5.1.

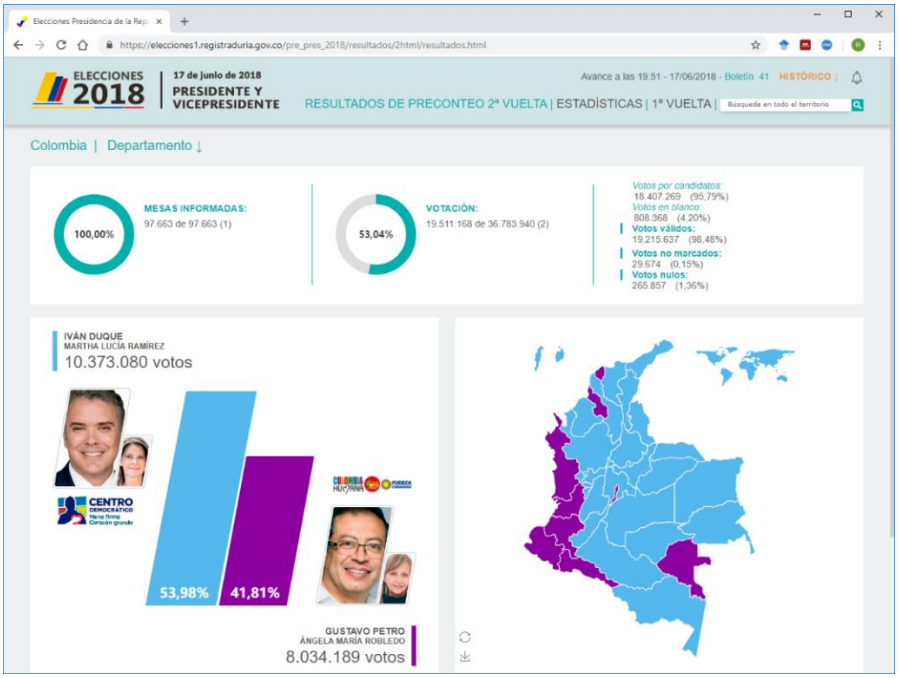


Figura 5.1 Aplicación de resultados electorales Registraduría Nacional del Estado Civil

La aplicación presentada en la figura anterior permite consultar los resultados electorales a nivel departamental, pulsando con el *mouse* sobre el mapa. La Figura 5.2 muestra los resultados de la segunda vuelta de las elecciones presidenciales 2018 en el departamento de Santander.

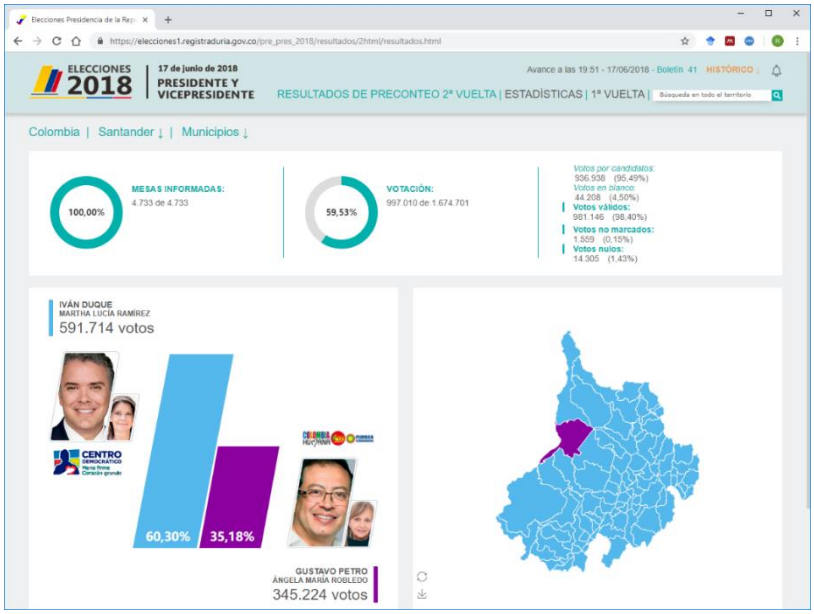


Figura 5.2 Resultados de la segunda vuelta electoral presidencial 2018 en Santander

La consulta indica que en Santander estaban habilitados para votar 1’674.701 personas, pero solo lo hicieron 997.010 de ellas, dando como índice de votación el 59,53%. Se reportaron 981.146 votos válidos, que representan el 98,40% del total de votos.

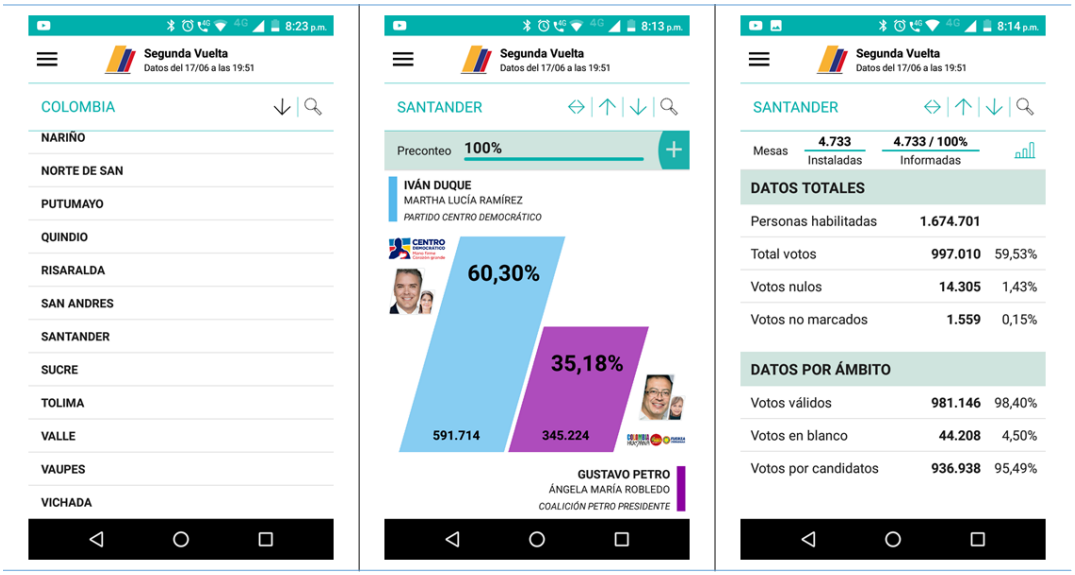


Figura 5.3 Consulta desde la aplicación para móviles

Los mismos resultados son accesibles desde dispositivos móviles mediante otra aplicación. La Figura 5.3 muestra tres capturas de pantalla desde un teléfono móvil. De izquierda a derecha, la

primera presenta una lista de departamentos para selección por parte del usuario; la segunda presenta la cantidad de votos obtenidos por cada candidato; y la tercera los datos de participación de interés para el presente caso.

Dado que el caso que se propone aborda la participación electoral, el resultado en términos de candidato ganador o perdedor de la contienda no es relevante. El mapa del departamento muestra los municipios que lo conforman, pero este nivel tampoco se considera pertinente.

5.3 Propuesta de aplicación para el análisis electoral

Se propone como caso de estudio el desarrollo de una aplicación para el análisis de la participación electoral en Colombia a nivel departamental, a partir de que el usuario registre en la aplicación los datos provistos por la Registraduría Nacional del Estado Civil acerca de la segunda vuelta de la elección presidencial del año 2018. Como marco para el desarrollo de la aplicación se utilizan las siguientes definiciones:

- **Departamento**: Cada una de las circunscripciones territoriales de primer nivel en que se divide el país para propósitos electorales.
- **Potencial de sufragantes**: Cantidad de personas habilitadas en un departamento para votar en un proceso electoral.
- **Total de sufragantes**: Cantidad de personas que acudieron a votar en un departamento en un proceso electoral.
- **Votos válidos**: Cantidad de tarjetones marcados de manera que indica, sin lugar a ninguna duda, la voluntad del sufragante de apoyar a alguno de los candidatos o de votar en blanco.
- **Indicador de participación electoral**: Indicador consistente en el total de sufragantes expresado como porcentaje del potencial de sufragantes.
- **Categoría de participación electoral**: Los departamentos se clasifican en categorías de acuerdo con el rango en que se ubique su indicador de participación electoral.
- **Votos no válidos**: Por definición un voto no válido es aquel que no indica de forma clara la voluntad del sufragante.
- **Indicador de error en la votación**: Indicador resultante de dividir el total de votos no válidos entre el total de sufragantes.

Para las definiciones de la lista anterior que impliquen la realización de cálculos se utilizan las siguientes fórmulas:

$$Indicador\ de\ participación\ electoral = \frac{Total\ de\ sufragantes}{Potencial\ de\ sufragantes} * 100$$

$$Votos\ válidos = Total\ de\ sufragantes - Votos\ válidos$$

$$Indicador\ de\ error\ en\ la\ votación = \frac{Votos\ no\ válidos}{Total\ de\ sufragantes}$$

En cuanto a las definiciones de la lista anterior que impliquen la ubicación en rangos se utilizan las siguientes tablas:

Tabla 5.1 Rangos y categorías de participación electoral.

Indicador de participación electoral	Categoría de participación electoral
Menor que 40%	Baja
Mayor que 40% y menor que 50%	Media
Mayor que 50%	Alta

Para que un departamento sea incluido en el análisis debe cumplir las siguientes condiciones de validación:

- El nombre del departamento no puede quedar vacío.
- El potencial de sufragantes debe ser mayor que cero.
- El total de sufragantes debe ser mayor que cero, pero menor o igual al potencial de sufragantes.
- La cantidad de votos válidos debe ser mayor que cero, pero menor o igual al total de sufragantes.

A medida que se registran los datos de varios departamentos la aplicación debe permitirle al usuario consultar los mismos indicadores ya mencionados pero a nivel nacional, y efectuar comparaciones entre departamentos según su participación electoral.

5.4 Caso propuesto

Se propone el desarrollo de una aplicación para el análisis de la participación electoral en Colombia, la cual será utilizada con los datos de los resultados electorales de la segunda vuelta presidencial del año 2018. Esta aplicación se denominará "*Aplicación para análisis de participación electoral*".

Se establecen dos restricciones en este desarrollo: la aplicación debe desarrollarse en el paradigma de Programación Orientada a Objetos y aplicando la Arquitectura de Capas.

Para efectos didácticos se establece que el desarrollo de la aplicación se haga mediante la metodología de prototipado evolutivo, con incrementos pequeños entre cada prototipo y el siguiente, de manera que se facilite la comprensión de cada cambio propuesto y sus implicaciones en el análisis, diseño e implementación.

Ya se ha mencionado en capítulos previos que una obra de ingeniería atraviesa durante su desarrollo por varias fases, dentro de las cuales están el diseño y el análisis. No se empieza a construir un puente, una red eléctrica ni una máquina, como obras de ingeniería, sin que antes

se hayan trazado sus planos, los cuales luego son sometidos a diversos criterios de validación hasta que se comprueba su consistencia. Pero tampoco se inicia el diseño sin haber hecho una especificación del problema y de lo que se requiere de la obra de ingeniería para la solución de este.

Por la masificación actual del uso de la informática, y la formación en programación de computadores que algunos estudiantes alcanzan a tener en sus cursos de educación básica secundaria y media, generalmente fuera del contexto de la ingeniería, se ha hecho frecuente la tendencia de empezar a digitar código de programación inmediatamente apenas se ha planteado el problema. Uno de los propósitos de este libro metodológico es contrarrestar esta tendencia.

Del mismo modo, cuando una aplicación software se va a construir como obra de ingeniería, lo que en Colombia se aprende en programas universitarios de Ingeniería de Sistemas y afines, también es necesario establecer claramente los requerimientos, determinar las características del software y luego elaborar una serie de planos que se convierten en la base del trabajo de implementación. En la metodología que se propone en este libro, el estudiante de Ingeniería de Sistemas debe aprender desde su formación básica que en el desarrollo de una aplicación informática la labor de programación se sustenta en todo un trabajo previo de análisis, de diseño y de implementación que debe ser reflejado en un proceso adecuado de documentación.

6 Punto de partida

La Programación Orientada a Objetos en Arquitectura de Capas (POOAC), como enfoque de solución distribuida, requiere un esfuerzo significativo en el diseño y la implementación aún en proyectos básicos. Para diferir dicho esfuerzo significativo en una cantidad de cuotas que puedan ser gradualmente asimiladas por el lector, se decidió que la *"Aplicación para análisis de participación electoral"* fuera desarrollada mediante metodología de prototipado evolutivo.

Este capítulo está limitado sólo al primer prototipo de solución al caso planteado en el capítulo anterior, para establecer las condiciones mínimas en que se desarrolla un proyecto básico de POOAC, en tres de las fases inherentes a todo proceso de ingeniería: análisis, diseño e implementación.

Tabla 6-1 Objetivos de aprendizaje del capítulo.

Desarrollo de software	Conceptos	Instrumentos
Análisis Diseño Implementación Abstracción Requisito	Clase: atributo, método, objeto Algoritmo: dato de entrada, proceso, dato de salida, estructura secuencial Tipo de dato	Estándar de desarrollo Modelo UML: Diagrama de casos de uso Especificación de casos de uso Diagrama de clases Diagrama de flujo

6.1 Un cálculo sencillo – Prototipo 1

6.1.1 Análisis

En un proyecto de desarrollo de software el análisis comprende la identificación de las actividades en las que el producto (el software) dará soporte a los usuarios, haciendo mención explícita de los requisitos para el producto y las restricciones en que dichos usuarios llevan a cabo sus actividades (Alagic, 2017). Esta visión conceptual es lo que se proponen esta sección y sus secciones equivalentes en los demás prototipos que conforman la secuencia de solución del caso planteado en el capítulo anterior.

El primer elemento por definir es el *alcance*, una descripción breve y general de las actividades en que el producto, es decir el software, dará soporte computacional al usuario, incluidas algunas restricciones.

Dado que el alcance se formula en términos generales, luego se presentan las *definiciones*, elementos del ámbito del problema, del campo del conocimiento en que este problema se manifiesta, que permiten entender y sobre todo precisar plenamente el alcance del producto. Es importante que en este proceso el desarrollador se asegure de utilizar y entender las mismas definiciones que su cliente o usuario, así parezcan triviales.

El análisis también comprende la especificación de requisitos del software, entendidos como necesidades, expectativas y restricciones de los usuarios que deben ser cumplidos por el producto de software (Chemuturi, 2013). La determinación de requisitos es fundamental como guía de desarrollo y como documento para la verificación del buen funcionamiento del software y su posterior entrega a satisfacción al cliente. En este libro se presentan dos tipos de requisitos para cada prototipo: requisitos de interfaz de usuario y requisitos funcionales. Los primeros determinan el comportamiento externo esperado de la aplicación respecto al usuario; los segundos se refieren a la descripción de las funciones internas de la aplicación, como procesos de cálculo, establecimiento de reglas, y demás aspectos que puedan ser programados.

6.1.1.1 Alcance

El prototipo 01 de la aplicación para análisis de participación electoral debe dar soporte computacional para que mediante el *potencial de sufragantes* y el *total de sufragantes* se calcule el *indicador de participación electoral* de la ciudadanía en un departamento.

No se contempla que este prototipo cuente con almacenamiento de datos. En consecuencia, el usuario deberá repetir la digitación cada vez que quiera calcular el indicador de participación de cualquier departamento.

En cuanto a interacción con el usuario el prototipo 01 de la aplicación debe solicitar los datos de un departamento, para luego mostrarlos de nuevo al usuario agregando el indicador de participación electoral calculado.

6.1.1.2 Definiciones

Como ya se mencionó antes, en ocasiones puede parecer trivial exponer una definición, cuando se trata de cosas que aparentemente "*todo el mundo sabe*", o que "*se dan por entendidas*". ¿Qué docente o estudiante de programación de computadores en una universidad colombiana no sabría qué es un departamento?

Pues bien, si se da por sentada la trivialidad de la definición de departamento, es probable que se excluya a Bogotá D.C. de la lista. Si bien desde el punto de vista de la división político-administrativa de Colombia es la capital del departamento de Cundinamarca, en cuanto a propósitos electorales y para que su gran cantidad de población no perjudique la representatividad del resto de Cundinamarca, Bogotá D.C. se toma como una circunscripción de primer nivel, es decir, del mismo orden de los departamentos.

Asimismo, los consulados de Colombia en el exterior, no tanto por su cantidad de electores como por su altísima dispersión geográfica, también son asimilados a una circunscripción

departamental y en ella votan todos los ciudadanos colombianos residentes en el exterior que se hayan inscrito para ejercer este derecho. Estos dos ejemplos se ilustran en la Figura 6.1.

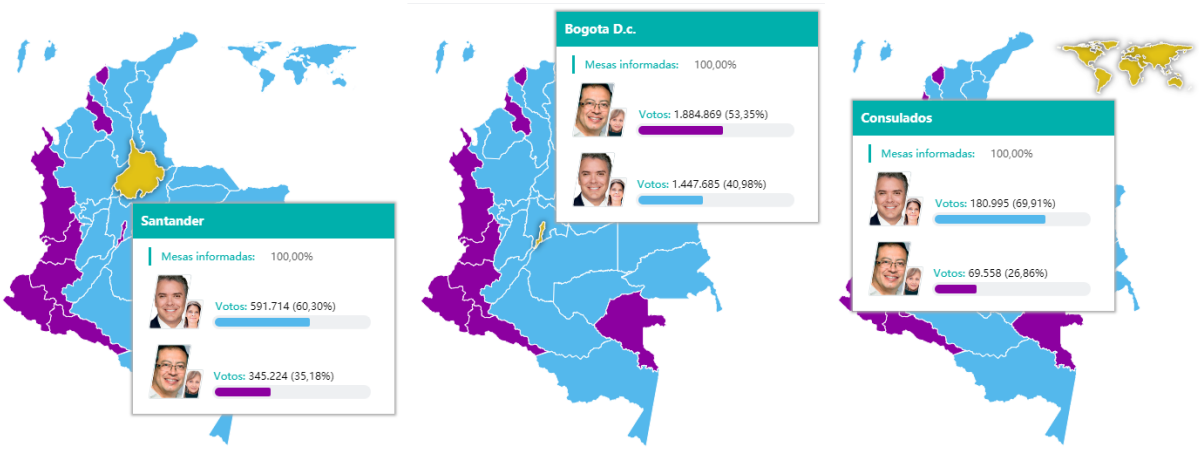


Figura 6.1 Ejemplos de departamentos definidos como circunscripciones electorales (Tomada de portal web Registraduría Nacional del Estado Civil)

En precisión de todo lo anterior, para el desarrollo del prototipo 01 de la aplicación para análisis de participación electoral se utilizan las siguientes definiciones.

Tabla 6-2 Definiciones aplicables en desarrollo del prototipo 01.

Departamento	Cada una de las circunscripciones de primer nivel en que se divide el país para propósitos electorales
Potencial de sufragantes	Cantidad de personas habilitadas en un departamento para votar en un proceso electoral
Total de sufragantes	Cantidad de personas que acudieron a votar en un departamento en un proceso electoral
Indicador de participación electoral	Indicador consistente en el total de sufragantes expresado como porcentaje del potencial de sufragantes $Indicador\ de\ participación\ electoral = \frac{Total\ de\ sufragantes}{Potencial\ de\ sufragantes} * 100$

Para complementar las definiciones, no está de más contar con una lista de ejemplos que permita a desarrolladores y usuarios comprobar que hay acuerdo en las definiciones. En la Tabla 6-3 se muestran los mismos ejemplos presentados antes en la Figura 6.1, con datos de los resultados de la segunda vuelta de la elección presidencial del año 2018, tomados del portal web de la Registraduría Nacional del Estado Civil.

Tabla 6-3 Ejemplos de las definiciones aplicables en desarrollo del prototipo 01.

Departamento	Potencial de sufragantes	Total de sufragantes	Indicador de participación electoral
Santander	1.674.701	997.010	59,53 %
Bogotá D.C.	5.702.805	3.572.698	62,64 %
Consulados	1.376.071	259.539	18,86 %

6.1.1.3 Requisitos

6.1.1.3.1 Requisitos de interfaz de usuario

6.1.1.3.1.1 Ingreso de información en casillas de edición

Para la interacción con el usuario, la aplicación debe presentar ventanas con casillas de edición en donde pueda digitar los datos solicitados. A su vez, la aplicación presentará la información en ventanas que el usuario podrá controlar mediante un botón de Aceptar.

6.1.1.3.2 Requisitos funcionales

6.1.1.3.2.1 Calcular el indicador de participación electoral de un departamento

La aplicación debe permitir al usuario introducir los siguientes datos básicos de un departamento: el nombre, el potencial de sufragantes y el total de sufragantes. Una vez digitados los datos básicos, la aplicación debe mostrar por pantalla los mismos datos, y además presentar el indicador de participación electoral.

6.1.2 Diseño

Diseñar un software es elaborar una especificación técnica de un producto que pueda cumplir satisfactoriamente los requisitos formulados durante la fase de análisis (Alagic, 2017). Como resultado del diseño se generan diferentes representaciones de la aplicación desde varios puntos de vista. A dichas representaciones iniciales del producto real se les conoce como *modelos*, dicho de otra forma, los *planos* detallados para construirlo, tomando este término prestado de la Ingeniería Civil. Para el diseño en Ingeniería del Software se usa un lenguaje gráfico llamado *Lenguaje Unificado de Modelado (UML).*

6.1.2.1 Diseño de escenarios

Algunos autores ubican el diseño de escenarios como una de las actividades de análisis. Más allá de esta controversia, los escenarios son las diferentes posibilidades esperadas de comunicación entre un usuario y la aplicación a desarrollar. Cada posibilidad de comunicación busca un fin particular que se conoce como *caso de uso*. Se describe como una frase que inicia con un verbo en infinitivo, y se representa mediante un *diagrama de caso de uso*. Dado que la funcionalidad del prototipo 01 es casi la mínima que una aplicación pudiera tener, en su diseño se considera un único caso de uso: calcular el indicador de participación electoral de un departamento, como se muestra en la Figura 6.2.

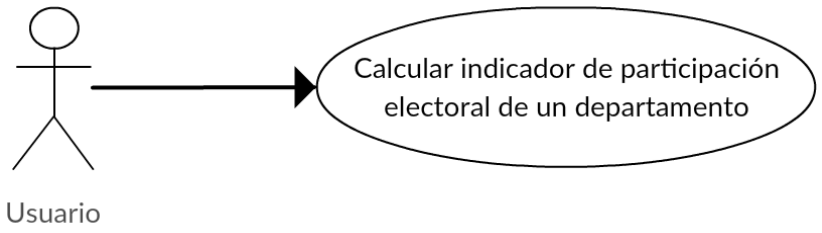


Figura 6.2 Diagrama de casos de uso para el Prototipo 01

Cada caso de uso debe contar con una descripción de la interacción exacta esperada entre el usuario y la aplicación, paso a paso. Esto se conoce como *especificación de caso de uso,* y ayuda a cumplir y validar lo que se describió anteriormente en los requisitos de interfaz de usuario (Ver Tabla 6-4).

Tabla 6-4 Especificación del caso de uso Calcular indicador de participación electoral de un departamento.

1	El usuario ingresa a la aplicación.
2	La aplicación presenta una pantalla con una caja de texto
3	El usuario digita el nombre del departamento y pulsa el botón *Aceptar*
4	La aplicación presenta una pantalla con una caja de texto
5	El usuario digita el potencial de sufragantes del departamento y pulsa el botón *Aceptar*
6	La aplicación presenta una pantalla con una caja de texto
7	El usuario digita el total de sufragantes del departamento y pulsa el botón *Aceptar*
8	La aplicación muestra en pantalla el nombre del departamento
9	El usuario pulsa el botón *Aceptar* para pasar al siguiente dato
10	La aplicación muestra en pantalla el potencial de sufragantes del departamento
11	El usuario pulsa el botón *Aceptar* para pasar al siguiente dato
12	La aplicación muestra en pantalla el total de sufragantes del departamento
13	El usuario pulsa el botón *Aceptar* para pasar al siguiente dato
14	La aplicación muestra en pantalla el indicador de participación electoral del departamento
15	El usuario pulsa el botón *Aceptar* para terminar la ejecución de la aplicación

Es de notar que la especificación de caso de uso no indica la forma en que la aplicación construye la información que brinda al usuario. Por ejemplo, en la Tabla 6-4 no se menciona la fórmula para calcular el indicador de participación electoral del departamento, ni se señala el momento en que dicho cálculo se lleva a cabo. En resumen, el caso de uso y su especificación sólo representan la interacción entre el usuario y la aplicación, con lo cual se determinan los datos de entrada y de salida.

6.1.2.2 Diseño de la estructura

Apartarse del enfoque de solución monolítica implica dejar la pretensión de solucionar todo el problema mediante un solo algoritmo, codificado en un único programa. Cuando se acoge el enfoque POOAC, en lugar de un único algoritmo el diseño de este software consta de varias partes

que conforman su diseño estructural, especializadas de acuerdo con la separación de funciones que subyace a la Arquitectura de Capas: una parte, la *capa de presentación* está encargada de la interacción con el usuario, mientras otra, la *capa de reglas*, se encarga de los cálculos y operaciones. En cada una de estas capas se ubican las *clases*, abstracciones conformadas por datos y procedimientos íntimamente vinculados a partir de la separación de funciones ya mencionada.

El Diagrama de Clases es el elemento para expresar el diseño de la estructura de una aplicación en POOAC. La Figura 6.3 presenta una primera aproximación a la estructura del prototipo 01.

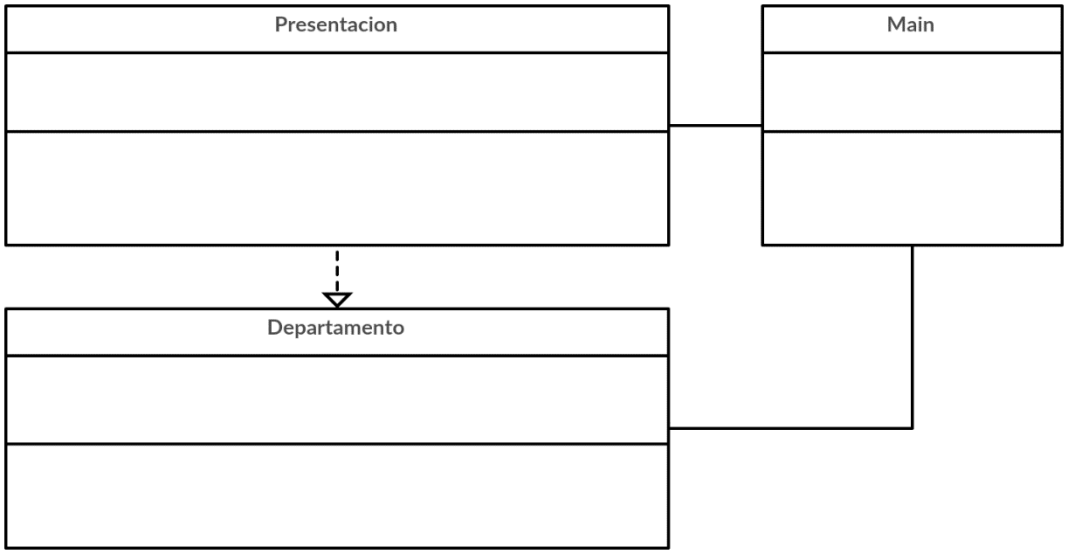


Figura 6.3 Primera aproximación al diagrama de clases del prototipo 01

La clase *Departamento* corresponde a la capa de reglas, y se encargará de la gestión de los datos de los departamentos y el cálculo del indicador de participación. En la capa de presentación se ubica la clase *Presentacion*, y tendrá como responsabilidad la interacción con el usuario. La clase *Main* es una especie de coordinadora de las acciones de las demás clases. La línea punteada de *Presentacion* a *Departamento* indica dependencia de la primera con respecto a la segunda. Las líneas continuas desde *Main* a las otras dos clases establecen una relación.

Cada clase se representa mediante un rectángulo dividido en tres secciones: la superior para el nombre de la clase; la intermedia para los atributos; la inferior para los métodos.

Los atributos son datos básicos o características de la clase. En la capa de reglas, la clase *Departamento* tiene como atributos los datos básicos de un departamento (nombre, potencial de sufragantes y total de sufragantes), y como métodos las operaciones que se hacen con dichos datos, en este caso el cálculo del indicador de participación.

Cada una de estas partes, se llama **clase**, y cuenta con un conjunto de características, llamadas **atributos.** Además, deben estar provistas internamente de uno o varios algoritmos, estos **algoritmos** deberán ser organizados en funciones internas que se llaman **métodos**. La distribución de los métodos dependerá exclusivamente de *la responsabilidad* que tendrá cada clase.

Adicionalmente, el análisis de los objetos que aparecen en el modelo de "negocio" permite también determinar posibles clases. Cada **objeto** debe poder crearse a través de una clase, es decir, la clase es un molde que se usa para poder crear un objeto, y a pesar de que la clase tendrá implementados (programados en código fuente) los atributos y los métodos, no se pueden ejecutar los métodos ni acceder a los valores de sus atributos a menos que se cree (se **instancie**) un objeto de dicha clase.

En este ejemplo de prototipo, al referirse a *responsabilidades por cada clase*, la arquitectura en capas establece que una clase que pertenezca a una capa se debe encargar de la comunicación con el usuario del sistema, mientras que una clase de otra capa se encargará del manejo de la información a través de almacenamiento de los datos en sus atributos y los cálculos que requiera realizar con sus propios datos. En síntesis, se cuenta entonces con esta información para determinar las posibles clases a implementar:

- De acuerdo con el **análisis**, se debe tener una clase llamada Departamento que permita crear un objeto departamento, y que debe tener como atributos el nombre, el potencial de sufragantes y el total de sufragantes.
- Debe implementarse una funcionalidad que es el cálculo del indicador de participación electoral. Dado que los datos para calcular pertenecen al departamento, y que el indicador de participación electoral también pertenece al departamento, se debe agregar esta funcionalidad a la clase Departamento. El indicador de participación electoral podría tomarse o no como un atributo adicional, pero debe tenerse claro desde el inicio que este valor no será solicitado, sino calculado.
- De acuerdo con el **diseño** establecido por la arquitectura en capas, se debe tener una clase que se encargue del proceso de comunicación con el usuario. Esta clase deberá tener las dos funcionalidades que se plantearon en el diseño: la solicitud de información y la presentación de información. Cada funcionalidad deberá ser programada a través de un algoritmo, dentro de un método. Esta clase se podrá llamar Presentación.
- Debe haber una clase principal que se encargue de coordinar las acciones de las dos clases. Podría llamarse *Main*.

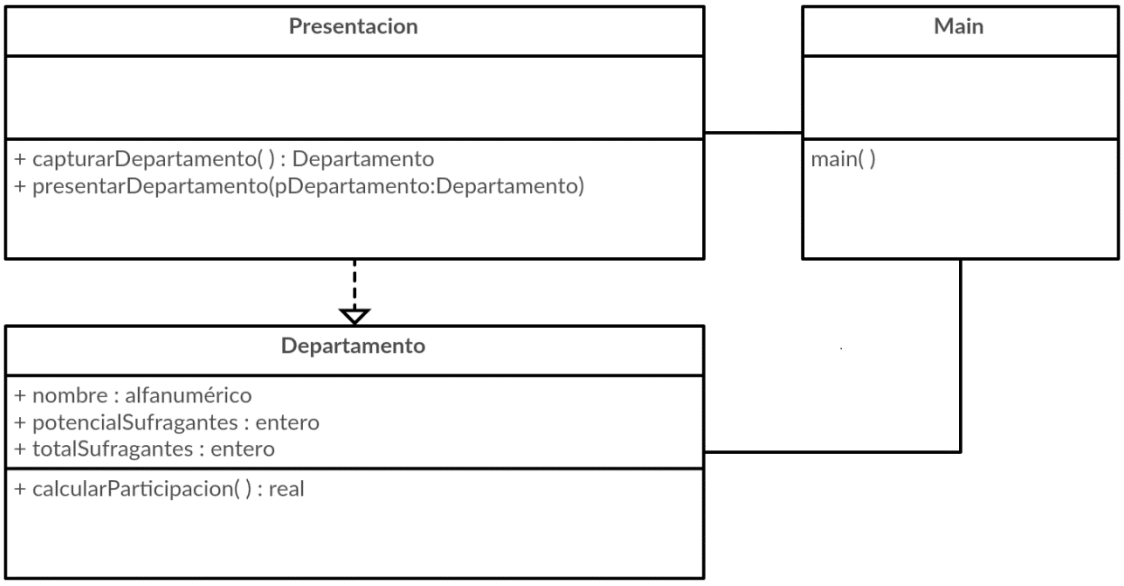


Figura 6.4 Diagrama de clases para el Prototipo 1

El conjunto de clases que se deberá implementar se presenta listado en la siguiente tabla, indicando la capa a la cual pertenece, dependiendo de la responsabilidad a la cual se hace cargo.

Tabla 6-5 Organización de la estructura de la aplicación en capas

Capa	Clase	Responsabilidad
Presentación	Presentación	Encargada del proceso de comunicación con el usuario. Cuenta con funcionalidades de solicitud y presentación de información.
Reglas de negocio	Departamento	Plantilla para crear un objeto de tipo *departamento*, que contenga los atributos y métodos requeridos.
Reglas de negocio	Main	Coordina las acciones de las otras clases
Persistencia de datos	-	-

Para tener en cuenta en el diseño de clases, cada clase debe indicar su conjunto de **atributos** y su conjunto de **métodos**, si los tienen. En orientación a objetos es común utilizar una normatividad sencilla en cuanto a las nomenclaturas, el nombramiento de las variables y la organización de código. A esto se le conoce como **estándar de desarrollo**, y es aplicado como buena práctica en la programación. En la síntesis de conceptos encontrará algunas de las normas aplicables para el estándar de desarrollo usado en Java.

Hay que tener en cuenta el contexto del problema para determinar adecuadamente las características de los **objetos** a los que se refiere y sus restricciones. Por ejemplo, en este caso particular se habla de departamentos: un departamento es una unidad política administrativa, geográfica y social en la que se divide el país. Si se habla en un contexto geográfico, se puede

hablar de características como su extensión, o su ubicación en una región. Incluso se sabe que en Colombia se cuenta con 32 departamentos. Sin embargo, en un contexto político administrativo, las características para tener en cuenta tienden más a conocer la cantidad de personas o la cantidad de sufragantes, incluso se conoce bajo este contexto que, por ejemplo, Bogotá D.C. se consideraría departamento. A este proceso de identificación de **objetos** y sus características se le conoce como **abstracción**. Adicionalmente es requerido que las características tengan definido un **tipo de dato** particular (texto, número entero, número real, lógico, etc.). Las definiciones de los objetos que se encuentran en el análisis del problema y sus características se presentan en la Para el prototipo 03 aparece una nueva definición en la lógica del negocio, llamado categoría de participación electoral. Este nuevo elemento se incorpora a la lista de definiciones. De acuerdo con el proceso de abstracción, este elemento pertenece al objeto departamento, pero depende particularmente de otro atributo del mismo objeto. Al igual que el indicador de participación electoral, la categoría de participación podría tomarse o no como un atributo adicional, pero debe tenerse claro que este valor no será solicitado, sino determinado dependiendo del indicador de participación.

Respecto a los **atributos**, en cada uno de ellos se debe indicar los **tipos de datos** a utilizar. Existen tres **tipos de datos** básicos: numéricos (enteros o reales), alfanuméricos (caracter único o cadena de caracteres) y lógicos. En un sentido riguroso, se podría hablar de **tipos de datos** primitivos y tipos de datos provistos por los diferentes lenguajes de programación. Su abordaje en este punto podría generar una carga conceptual incompatible con el propósito del capítulo que se centra en el diseño. En lo que concierne a este capítulo, se consideran como tipos de datos los ya mencionados, y sólo haría falta agregar que los datos numéricos pueden ser de tipo real o entero, según requieran cifras decimales o no, respectivamente, y que los datos alfanuméricos pueden ser de un solo carácter o de un conjunto de caracteres.

Tabla 6-6 Definiciones de los objetos en el análisis del problema

Objeto	Atributo	Tipo de Dato
Departamento	nombre	Texto
	potencialSufragantes	Número Entero
	totalSufragantes	Número Entero
	indiceParticipacion	Número Real

Respecto a los **métodos**, cada uno de los métodos representa un **algoritmo** que realiza un conjunto de tareas a ejecutar por el sistema. El método de la clase *Departamento*, según el diseño, debe tener el algoritmo que se encargue del cálculo del indicador de participación electoral. Los métodos de la clase *Presentación* son los que se encargan del proceso de comunicación con el usuario, es decir, la clase tendrá implementados dos algoritmos que ejecutarán las siguientes actividades:

- Capturar la información de un departamento: este método deberá pedir los datos de un departamento y armar un objeto de la clase Departamento con los valores solicitados para ingresarlos en sus atributos.
- Presentar la información de un departamento: este método deberá recibir un objeto de la clase Departamento y mostrar los valores de sus atributos en pantalla.

6.1.2.3 Diseño de algoritmos

6.1.2.3.1 Clase Departamento –calcularParticipacion( ): real

Como método de cálculo, aplica una fórmula para obtener como resultado el indicador de participación electoral con base en el potencial y el total de sufragantes del departamento. Es un algoritmo secuencial, con **datos de entrada** a requerir, **procesos de cálculo** y **datos de salida**. Debe devolver un número real.

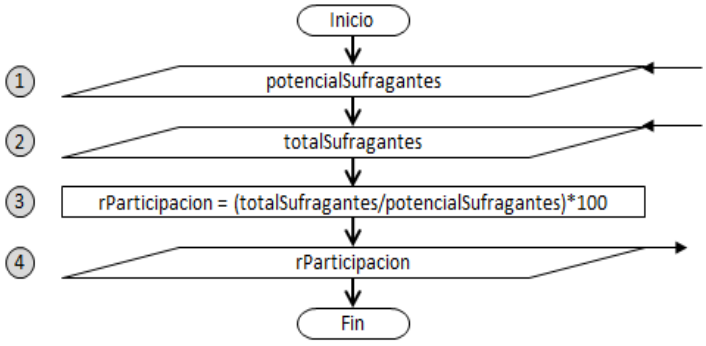


Figura 6.5 Diagrama de flujo para el método calcularParticipacion()

Tabla 6-7 Descripción del algoritmo para el método calcularParticipacion()

1	Se requiere como dato de entrada *potencialSufragantes,* se cumple por tener este dato como atributo propio.
2	Se requiere como dato de entrada *totalSufragantes,* se cumple por tener este dato como atributo propio.
3	Divide *totalSufragantes* entre *potencialSufragantes* y multiplica por cien (100). Almacena el resultado de la operación en la variable *rParticipacion*.
4	Se genera como dato de salida el valor de la variable *rParticipacion*. Para ello se realiza como respuesta de este método.

6.1.2.3.2 Clase Presentacion - capturarDepartamento( ):Departamento

Obtiene del usuario cada uno de los valores para los atributos, y entrega como respuesta el objeto de la clase Departamento con los valores de los atributos asignados. Como se requiere trabajar con los datos de la clase Departamento, se debe **crear un nuevo objeto** de tipo Departamento. Tratándose del método de captura, como resultado debe devolver el objeto de la clase Departamento.

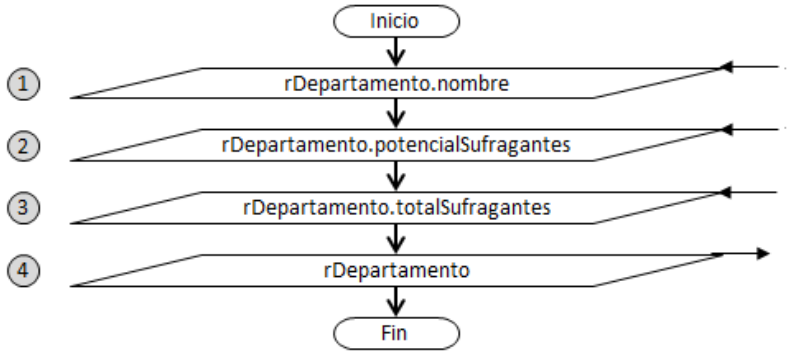


Figura 6.6 Diagrama de flujo para el método capturarDepartamento()

Tabla 6-8 Descripción del algoritmo para el método capturarDepartamento()

1	Se requiere como dato de entrada el nombre del departamento para almacenar en el objeto de tipo Departamento. Para cada dato de entrada, la aplicación debe presentar una pantalla con una caja de texto.
2	Se requiere como dato de entrada el potencial de sufragantes del departamento para almacenar en el objeto
3	Se requiere como dato de entrada el total de sufragantes del departamento para almacenar en el objeto
4	Se genera como dato de salida el objeto *rDepartamento*, de la clase *Departamento*. Para ello se realiza como respuesta de este método.

6.1.2.3.3 Clase Presentacion - presentarDepartamento(pDepartamento:Departamento)

Este método recibe un objeto de la clase Departamento como parámetro. Presenta en pantalla cada uno de los valores de los atributos del objeto recibido, así como el resultado del método de cálculo del indicador de participación electoral.

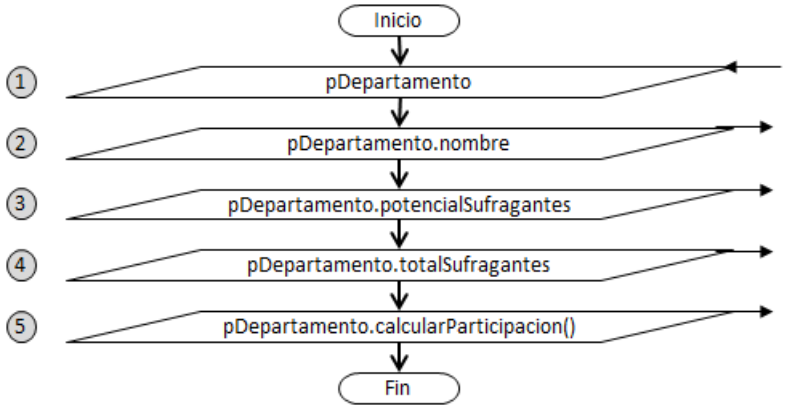


Figura 6.7 Diagrama de flujo para el método presentarDepartamento()

Tabla 6-9 Descripción del algoritmo para el método presentarDepartamento()

1	Se requiere como dato de entrada el objeto *pDepartamento*, de la clase *Departamento*. En este caso se recibe el objeto como mensaje.
2	Se genera como dato de salida el nombre del departamento almacenado en el objeto. Dado que este dato de salida es para la visualización por parte del usuario, se utiliza la forma de óvalo con punta izquierda.
3	Se genera como dato de salida el potencial de sufragantes del departamento almacenado en el objeto
4	Se genera como dato de salida el total de sufragantes del departamento almacenado en el objeto
5	Se genera como dato de salida la respuesta del método calcularParticipacion() del objeto.

6.1.2.3.4 Clase Main – main( )

La clase Main coordina las acciones de las demás clases. Su método main() solicita a un objeto de la clase Presentacion que lleve a cabo la captura de los datos de un objeto de la clase Departamento, cuyos datos almacena luego en el objeto departamento. A continuación, solicita al objeto de la clase Presentacion que lleve a cabo el método de presentación tomando como parámetro el objeto de la clase Departamento. Como desde Main se requiere usar las funciones de la clase Presentación, se debe **crear un nuevo objeto** de tipo Presentación. Sin embargo, como se requiere usar las funciones de un objeto de tipo Departamento, no se requiere crear un objeto nuevo de tipo Departamento, sino que se usaría el que le devuelve el objeto Presentación con el método capturarDepartamento.

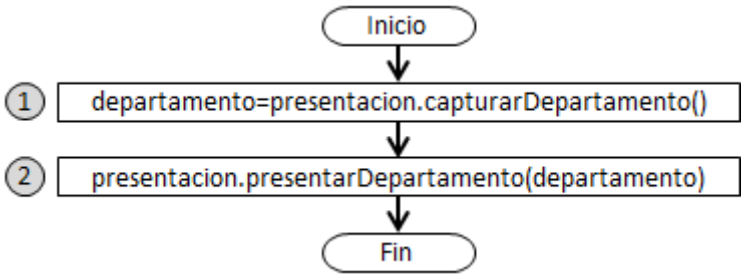


Figura 6.8 Diagrama de flujo para el método main()

Tabla 6-10 Descripción del algoritmo para el método main()

1	Ordena al objeto *presentacion* que ejecute su método *capturarDepartamento()* y entregue el resultado al objeto *departamento*. El objeto *presentacion* debe ser de la clase *Presentacion*, y el objeto *departamento* de la clase *Departamento*.
2	Ordena al objeto *presentacion* que ejecute su método *presentarDepartamento()*, tomando como parámetro el objeto *departamento*.

6.1.3 Implementación

El **análisis** y el **diseño** son fases ineludibles de todo proceso de ingeniería, incluida la Ingeniería del Software. Una mala práctica frecuente es la de empezar a trabajar directamente digitando líneas en un lenguaje de programación, lo que por lo general conduce a desarrollos de mala

calidad y de difícil mantenimiento y corrección. Como se demuestra en esta sección, la implementación o codificación en un lenguaje de programación no es en sí misma el acto creativo de ingeniería, sino una especie de traducción del **diseño** a un lenguaje de programación específico.

Para la implementación se toma como referencia el **diseño** de cada una de las clases. Al crear un proyecto, se crea una carpeta donde se generarán los archivos de cada una de las clases. En este caso, la carpeta donde se almacenará todo el proyecto se llamará *proyecto*. Esa carpeta es un empaquetador (*package*) de las clases del proyecto, por eso se debe indicar en cada clase el nombre del *package* donde estarán ubicadas las clases que se usarán dentro del código.

Con base en el diagrama de clase de la Figura 6.4 se establece que la codificación del prototipo 1 debe distribuirse en tres archivos: Departamento, Presentacion y Main.

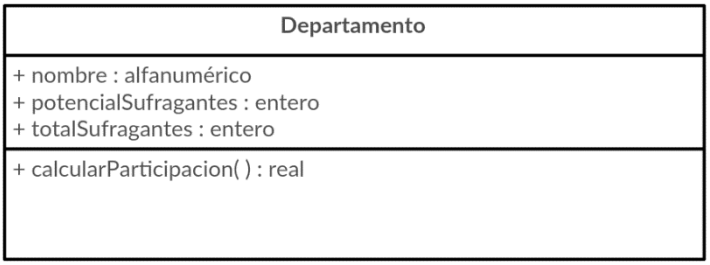


Figura 6.9 Clase Departamento para el Prototipo 1

La división superior del rectángulo que representa a una clase contiene su nombre. El archivo correspondiente requiere una línea de declaración de la clase, que consta de tres partes: la declaración de visibilidad, que por lo pronto se deja como *public* hasta tanto se explique este concepto; la palabra reservada *class*, para especificar que se está declarando una clase; y por último el nombre de la clase. Para el presente caso se trata de la línea 3 en la Tabla 6-11.

Es de mucha importancia en los lenguajes de programación la delimitación de bloques, algo que suele causar confusión y problemas de desorden a los programadores novatos. El lenguaje de programación Java utiliza las llaves "{" y "}" como delimitadores de apertura y de cierre de bloque respectivamente. Es de notar entonces que toda la codificación de la clase Departamento queda delimitada entre la llave de apertura al final de la línea 3 y la de cierre en la línea 13; todo debe quedar entre estas dos llaves. La línea 1 no hace parte de la codificación de la clase, sino que corresponde a la definición del empaquetador en que está contenido el archivo.

Tabla 6-11 Codificación de la clase Departamento - Archivo "Departamento.java"

1	package proyecto;
2	
3	public class Departamento {
4	public String nombre;

```
5     public int potencialSufragantes;
6     public int totalSufragantes;
7
8     public double calcularParticipacion(){
9        double rParticipacion;
10       rParticipacion = (totalSufragantes / potencialSufragantes) * 100;
11       return rParticipacion;
12    }
13 }
```

La sección media contiene los **atributos**, cada uno de los cuales requiere una línea de declaración de atributo, que consta de tres partes: la declaración de visibilidad, que por lo pronto se deja como *public* hasta tanto se explique este concepto; el **tipo de dato**; y por último el nombre del atributo. Los atributos de la clase Departamento se declaran en las líneas 4, 5 y 6 de la Tabla 6-11.

La sección inferior del rectángulo contiene los **métodos** de la clase. Cada método requiere su propia línea de declaración de método, que consta de tres partes: la declaración de visibilidad, que por lo pronto se deja como *public* hasta tanto se explique este concepto; el **tipo de dato** que el método entrega como respuesta; y por último el nombre del método y el paréntesis para los parámetros que se explicarán posteriormente con algún método que los requiera. El único método de la clase Departamento se declara en la línea 8 de la Tabla 6-11, que termina con la llave de apertura que delimita un bloque hasta la correspondiente llave de cierre en la línea 12.

Además de la declaración se codifica el algoritmo, que para este método es el diagrama de flujo de la Figura 6.5, el cual representa una **estructura secuencial** de órdenes. Los pasos 2 y 3 de dicha figura no requieren codificación ya que los datos de entrada del algoritmo son atributos de la misma clase y como atributos ya están disponibles sin que se declaren como datos de entrada. El paso 4 se codifica en la línea 10, y el paso 5 en la línea 11. Java es un lenguaje fuertemente tipado, por lo que antes de utilizar una variable se debe declarar su tipo de dato; para que no se genere un error en la línea 10, la variable *rParticipacion* es declarada previamente en la línea 9.

Es preciso señalar la importancia de las indentaciones, o sea, los desplazamientos a la derecha de las líneas que permiten visualizar a qué bloque pertenecen. Asimismo, el uso de líneas en blanco para separar los bloques entre sí. No contribuye a la eficiencia, pero sí mejora la legibilidad, propósito ya comentado en el capítulo anterior.

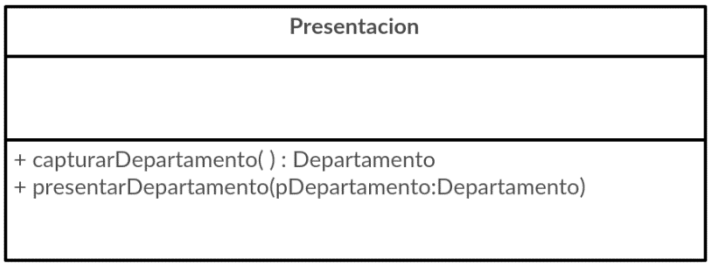


Figura 6.10 Clase Presentacion para el Prototipo 1

De forma similar se ha codificado la clase *Presentacion*. Se declara la clase en la línea 4 de la Tabla 6-12. El método *capturarDepartamento()* se declara en la línea 6 y el método *presentarDepartamento()* en la línea 15.

La codificación del método capturarDepartamento() corresponde al algoritmo presentado en la . El paso 1 se implementa en la línea 9, el paso 2 en la línea 10, el paso 3 en la línea 11, y el paso 4 en la línea 12.

Como se mencionó antes, Java es un lenguaje fuertemente tipado, por lo que el objeto rDepartamento que se entrega como respuesta debió ser previamente declarado en la línea 7, lo cual consiste en indicar el nombre de la clase (Departamento) y el nombre del objeto dentro del método (rDepartamento). Asimismo, el objeto debe ser inicializado mediante la sentencia ***new*** seguida del nombre de la clase como muestra la línea 8.

Tabla 6-12 Codificación de la clase Presentacion - Archivo "Presentacion.java"

```
1  package proyecto;
2  import javax.swing.JOptionPane;
3
4  public class Presentacion {
5
6     public Departamento capturarDepartamento(){
7        Departamento rDepartamento;
8        rDepartamento=new Departamento();
9        rDepartamento.nombre=JOptionPane.showInputDialog(null);
10       rDepartamento.potencialSufragantes=Integer.parseInt(JOptionPane.showInputDialog(null));
11       rDepartamento.totalSufragantes=Integer.parseInt(JOptionPane.showInputDialog(null));
12       return rDepartamento;
13    }
14
15    public void presentarDepartamento(Departamento pDepartamento){
16       JOptionPane.showMessageDialog(null, pDepartamento.nombre);
17       JOptionPane.showMessageDialog(null, pDepartamento.potencialSufragantes);
18       JOptionPane.showMessageDialog(null, pDepartamento.totalSufragantes);
19       JOptionPane.showMessageDialog(null, pDepartamento.calcularParticipacion());
20    }
21 }
```

La línea 2 requiere una mención adicional. Siendo Java un lenguaje orientado a objetos, no solo permite al usuario crear sus propias clases, sino que cuenta con clases nativas que pone a disposición del usuario para diversos fines. En este caso la clase JOptionPane se usa para la interacción con el usuario, y es la encargada de presentar en pantalla y recibir por teclado lo que sea pertinente. Para poder usar la clase JOptionPane se debe indicar la importación de una librería adicional, que es justo lo que se hace en dicha línea 2.

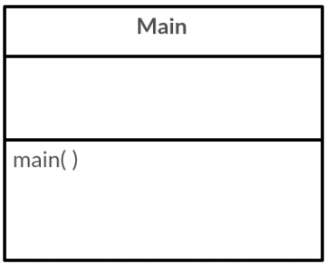


Figura 6.11 Clase Main para el Prototipo 1

La clase *Main* es declarada en la línea 3 de la Tabla 6-13. Su único método main() es a su vez declarado en la línea 5. Como coordinadora de las demás clases esta no suele manejar datos y su método main() requiere ser declarado como *static void*, característica que solo será explicada en un capítulo posterior luego de la presentación de algunos conceptos más avanzados.

El bloque de la línea 6 a la 10 es la codificación del algoritmo de la Figura 6.8. Los pasos 1 y 2 corresponden a las líneas 9 y 10 respectivamente. El objeto *departamento* debió ser previamente declarado en la línea 6. El objeto *presentacion* fue declarado en la línea 7 e inicializado en la 8 mediante la sentencia *new*. El objeto *departamento* no requiere inicialización, pues el método *capturarDepartamento()* se encarga de hacerlo.

Tabla 6-13 Codificación de la clase Main - Archivo "Main.java"

```
1   package proyecto;
2
3   public class Main {
4
5      public static void main(String[] args) {
6         Departamento departamento;
7         Presentacion presentacion;
8         presentacion=new Presentacion();
9         departamento=presentacion.capturarDepartamento();
10        presentacion.presentarDepartamento(departamento);
11     }
12  }
```

6.2 Síntesis de conceptos

6.2.1 Orientación a objetos

A diferencia de la programación estructurada tradicional en la que se presenta todo un bloque de líneas de código, existe un enfoque particular de programación en el que se establecen varias "unidades de código" separadas y organizadas dependiendo de sus distintas características, comportamientos y responsabilidades. Estas unidades de código se les conocen como **clases**.

Las **clases** son los elementos que definen la estructura de un programa (Villalobos & Casallas, 2006). Son creadas como resultado de un proceso de abstracción o entendimiento de la naturaleza de los elementos que se pueden encontrar en un contexto real. Estos elementos se conocen con el nombre de **objetos.** El propósito de un objeto es la comprensión del mundo real y así mismo proporcionar una base practica para la implementación por computadora. Estos objetos del mundo real podrían ser asignados a los objetos definidos en el software y esto hace que un problema se divida en subproblemas o problemas más simples (Houben, 2014).

Una clase también está compuesta por métodos y atributos. Los **métodos** son aquellos algoritmos que van a determinar el comportamiento u operaciones de los objetos. Algunos de los diferentes usos de los métodos que se identifican son de consulta, de operaciones, modificadores, entre otros . Los **atributos** son valores de un dato que se almacenan en los objetos de una clase, y se definen dentro de la clase de la misma forma que una variable (Rumbaugh et al., 1995).

La estructura de una clase se puede observar en la siguiente figura:

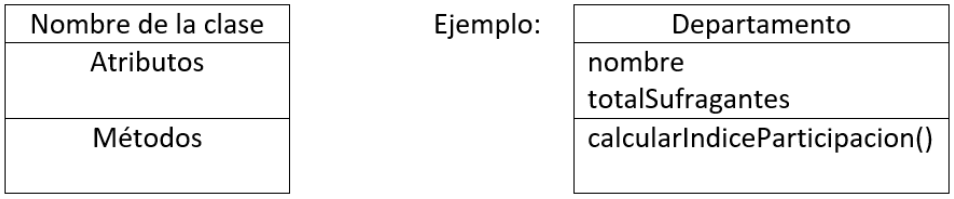


Figura 6.12 Ejemplo Clase Departamento para el Prototipo 1

Una clase se crea con el fin de poder crear (instanciar) objetos, comportándose como un "modelo", un "molde" o una "plantilla" para la creación de diferentes objetos de la misma clase. Las clases no ejecutarán su programación a menos que se haya invocado un método de una clase estática o de un objeto instanciado; esto significa que técnicamente, el objeto es la unidad que realiza las tareas del programa en tiempo de ejecución. Los objetos instanciados adquieren las mismas características que se programan en la clase, es decir, contendrán una colección de elementos de datos (los atributos), junto con las funciones o algoritmos utilizados para operar sobre esos datos (los métodos). La potencia real de los objetos reside en dos características muy importantes: los objetos pueden crear otros objetos y los objetos pueden comunicarse (interactuar) con otros objetos.

El conjunto de todos los valores de los atributos de un objeto se le conoce como estado, y el conjunto de todos los métodos de un objeto se le llama comportamiento. Todos los objetos se

consideran únicos, por lo que cada uno posee una identidad, pudiendo diferenciarse según su estado específico.

Los objetos pueden ser concretos (objetos físicos, lugares, personas, archivos, sistemas de archivos…), como también conceptuales (procesos, políticas o normas, eventos, elementos abstractos…) (Rumbaugh et al., 1995). Para establecer una clase adecuadamente, se debe realizar un proceso mental por el cual, a partir de una situación o problema planteado, se captan elementos con características y comportamientos esenciales y particulares, permitiendo que estos elementos puedan ser descritos en forma general (qué tienen –atributos– y qué hacen –métodos–), y que también puedan ser aislados del resto de los otros elementos. Este proceso mental se le conoce como **abstracción**. Esta descripción y aislamiento permite que algunos elementos puedan ser agrupados para llegar a un nivel de generalización óptimo. Cuando se identifican estos elementos (u objetos), se dice que se está generando una abstracción de las clases que las comprenden.

Cuando se tiene una abstracción inicial de las clases que deben estar en el sistema, se plasma en un diagrama y se le asignan características adicionales a los atributos y métodos de acuerdo con sus características en el contexto en el que se esté trabajando. Es muy posible que aparezcan más atributos y más métodos después, sin embargo, la ventaja radica en que la programación orientada a objetos hace que se deba preocupar solo de la parte que debe modificarse o crearse sin dañar las demás partes del programa. Se recomienda sin embargo que cuando genere la abstracción del problema, las clases que deba crear y sus atributos mantengan un nombre muy similar al del objeto y sus características que han sido identificados en el contexto real.

6.2.2 UML

El modelado es una técnica que permite generar representaciones abstractas de los elementos que conforman cualquier sistema, donde se identifican los aspectos importantes de lo que se va a desarrollar. Estas representaciones son conocidas como **modelos**, los cuales permiten ver y comprender claramente comportamientos, diseños, tomas de decisiones, fenómenos, sistemas o procesos a fin de describirlos, explicarlos, simularlos y predecirlos. Un solo modelo no puede presentar todos los aspectos de un sistema, por ello se generan diversos modelos omitiendo aspectos que no son relevantes, reduciendo la complejidad en su presentación y así poder abstraer, organizar, analizar, y corregir la información pertinente de un sistema. Los modelos, no son usados únicamente en la ingeniería, también en arquitectura y los campos creativos (Rumbaugh et al., 2000).

Cuando se habla de **UML** se refiere al Lenguaje Unificado de Modelado, un lenguaje que permite realizar un modelado visual donde se especifica, construye y documenta artefactos de un software. Sus principales funciones son capturar y conocer todas las características de los sistemas que se deben construir, es decir que lo que se pretende es entender completamente el diseño, configuración y control de la información de un sistema que se desarrollará. Para la aplicación de este lenguaje se deben tener en cuenta técnicas de modelado, que incluyen conceptos semánticos y notaciones con reglas particulares. Es importante aclarar que este no es un lenguaje de programación, es una herramienta que capta la información de la estructura de

un sistema, tanto estática como el comportamiento dinámico, donde se identifica una colección de componentes que interactúan para cumplir un objetivo que brindará un servicio o beneficio a un usuario externo. UML permite realizar modelos que se puedan agrupar en diferentes puntos de vista, lo cual facilita a los equipos de software realzar una división de grandes sistemas en partes pequeñas de trabajo (Rumbaugh et al., 2000).

Uno de los diagramas que se pueden modelar utilizando UML es el **diagrama de casos de uso**, por medio de este se puede capturar e identificar el comportamiento de un sistema, un subsistema o de una clase, y asignar la funcionalidad del sistema en acciones, teniendo en cuenta que sea significativo para los actores u usuarios del sistema. Este diagrama debe complementarse con una lista de las actividades secuenciales que se espera obtener como resultado de la interacción entre el usuario y el sistema. Esta descripción secuencial se le conoce como **especificación de caso de uso**.

En la siguiente figura se puede observar un ejemplo de un diagrama de casos de uso.

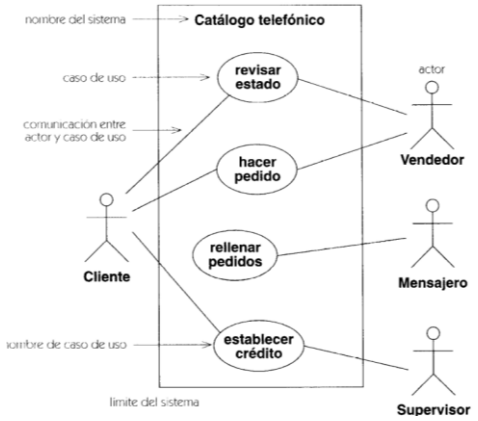


Figura 6.13 Ejemplo Caso de Uso

Para comprender completamente el concepto de diagrama de casos de uso es importante definir cada una de las partes que lo componen, estas partes son:

- Actor: hace referencia a una idealización de una persona o cosa que interactúa con un proceso de manera externa, es decir son aquellos que realizan los casos de uso (Fowler & Scott, 2000).
- Caso de uso: Hace referencia a cada una de las funcionalidades del sistema, que reflejan la interacción entre un usuario (actor) y un sistema, y que se expresa por medio de una serie de secuencias de mensajes (Rumbaugh et al., 2000).
- Sistema: Se refiere a una colección de componentes interconectados entre sí para cumplir un propósito. Normalmente ese conjunto de componentes brinda las funcionalidades expresadas en los casos de uso ubicados dentro de él.

Otro diagrama que se pueden modelar usando UML es el **diagrama de clases,** el cual se usa para describir los tipos de objetos presentes en un sistema, las diferentes clases y sus relaciones. Las clases contienen atributos y operaciones (métodos) y se representan gráficamente en un rectángulo, donde los atributos y métodos se muestran en espacios separados. Las relaciones son representadas con líneas que conectan los rectángulos de las clases.

En la siguiente figura se puede observar un ejemplo de un diagrama de clases.

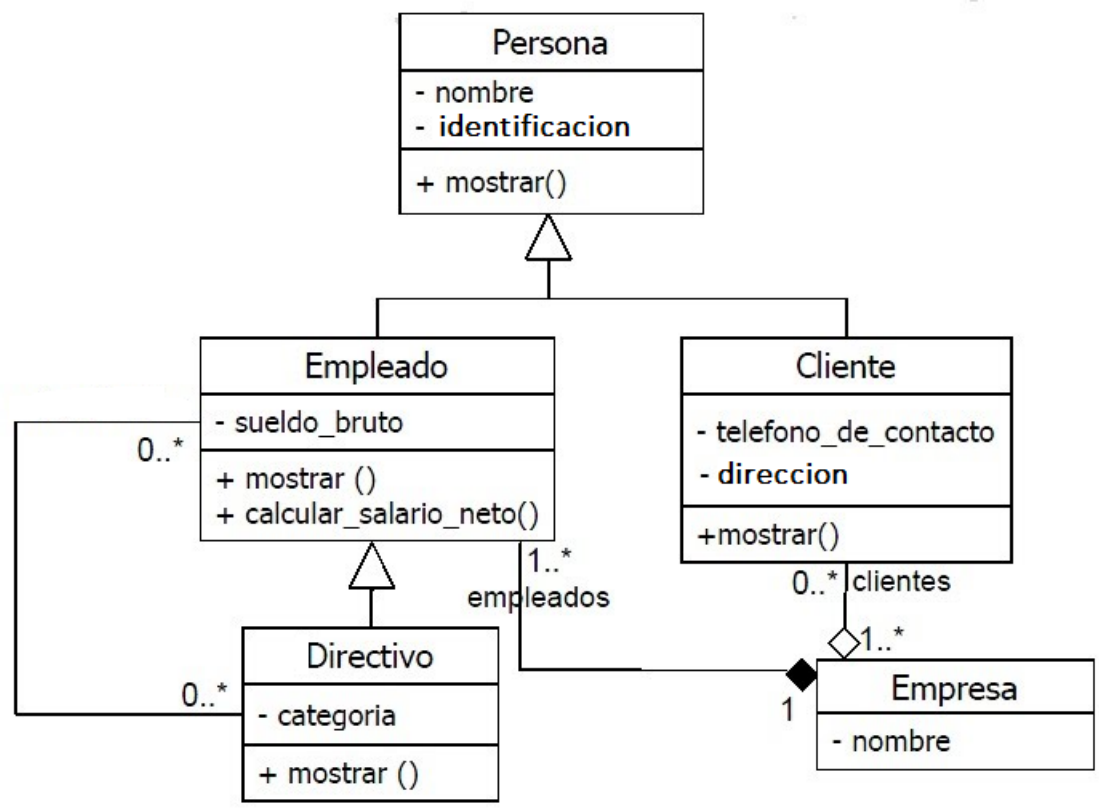


Figura 6.14 Ejemplo Diagrama de Clases

6.2.3 Programación en Java – Buenas prácticas

Un truco muy importante para que un programa en Java sea de buena calidad radica en la capacidad de orden que tiene el programador. Si un programador es muy desordenado o no sigue unas reglas base, es muy difícil que un programa le salga sin errores. También es muy difícil que cualquier programador aparte pueda colaborarle en su labor. Existen unas normas básicas que debe cumplir todo programador Java en varios aspectos, y que deben convertirse en su **estándar de desarrollo**:

- **Creación de clases**

Toda clase debe ir con mayúscula inicial, y debe comprender una palabra (sustantivo en singular) o un par de palabras pegadas (sustantivos o sustantivo + adjetivo en singular) sin tildes, que expresen realmente que significa esa clase, que explique sólo con la palabra el propósito para

el cual fue creada. Se pide en mayúscula inicial porque es el estándar mantenido por Java, en sustantivo y en singular, porque con solo una clase se pueden crear muchos objetos similares o simplemente uno solo. Todas las clases que permitan crear objetos deben ser públicas, por lo que debe comenzar con la palabra *public*. La "fórmula" o sintaxis para crear una clase es la siguiente:

public class NombreClase

Al finalizar la orden completa, no se coloca punto y coma. Además debe abrirse y cerrarse llaves { }.

Se puede escribir así:

Tabla 6-14 Codificación de una Clase en Java

```
1  package proyecto;
2
3  public class Departamento {
4
5  }
```

El nombre de la clase con la primera letra en Mayúscula

O así:

Tabla 6-15 Codificación de una Clase en Java

```
1  package proyecto;
2
3  public class Departamento
4  {
5
6  }
```

Todo lo que esté dentro de estas llaves, pertenecerá exclusivamente a dicha clase. Solo deben ir fuera de la clase (y antes de definir la clase), los comandos de importación (*import…*) y las indicaciones de pertenencia a un paquete (*package*).

Todas las órdenes que se coloquen dentro de un par de llaves deben ir corridas a la derecha, un espacio de 4 caracteres en blanco ó 1 caracter que genere el tabulador [TAB]. Esto se le conoce como indentación.

- **Creación de atributos**

Todos los atributos de la clase deben ser creados al principio de la clase. Además, a todos los atributos se les especifica el tipo de dato que se desea que maneje. Cada vez que se cree un atributo, debe terminar con punto y coma. Los nombres de los atributos deben ser un sustantivo o dos o más palabras pegadas (sustantivos o sustantivo + adjetivo) que se deben escriban en letra minúscula. Solo se deben usar mayúsculas entre la palabra únicamente en caso de que se desee mejorar la legibilidad del nombre (como, por ejemplo: nombreEstudiante, o areaProgramada, o notaParcial). También se pueden usar símbolos como guiones (-) o guiones bajos (_), pero no

abuse de ellos (por ejemplo: nombre_estudiante, o area_programada, o nota_parcial). También puede usar números, pero no puede usarlos al principio del nombre. En cuanto a los tipos de datos, hay que recordar que son en minúsculas (*byte, short, int, long, float, double, char*) excepto *String*. Un ejemplo sería este:

Tabla 6-16 Tabla 6.15 Declaración de Tipos de Datos en Java

```
1  package proyecto;
2
3  public class Departamento {
4     public String nombre;
5     public int potencialSufragantes;
6     public int totalSufragantes;
7  }
```

La declaración de la clase NO termina en punto y coma

Se inicia la indentación con los atributos y se mantiene en toda la clase

- **Creación de objetos como atributos**

Para los atributos de tipo objeto (los que se crean usando las clases), el tipo de dato debe ser el nombre de la clase (es decir, en mayúscula inicial). Si se desea crear un objeto nuevo se debe asignarle de una vez el espacio en memoria y es posible hacerlo de una vez en la misma orden de creación. Cabe recordar que se asigna el espacio en memoria con el comando *new*, seguido del nombre de la clase (con mayúscula inicial) y un par de paréntesis. Si se desea crear un objeto solo como referencia para que reciban datos de otro objeto similar, no necesitan ser nuevos, y cuando son asignados, se consideran como si fuera el mismo objeto del cual reciben los datos.

Tabla 6-17 Creación de un Objeto

```
1  package proyecto;
2
3  public class Main {
4
5     public static void main(String[] args) {
6        Departamento departamento;
7        Presentacion presentacion;
8        presentacion=new Presentacion();
9        departamento=presentacion.capturarDepartamento();
10       presentacion.presentarDepartamento(departamento);
11    }
12 }
```

*Aquí se usa un NUEVO objeto llamado **presentacion** para usar sus funciones, debido a que no existe previamente.*

*NO se crea un objeto nuevo llamado **departamento** porque estará siendo creado desde el método de la clase **presentacion***

- **Creación de métodos**

Todos los métodos de la clase deben ser creados luego de la definición de los atributos de la clase, y deben ir en forma pública los métodos que deban ser llamados desde el exterior. Todo método debe ir con minúscula, y debe comprender una palabra que indique una acción (verbo) o un par de palabras pegadas que indiquen acción (verbo + sustantivo) sin tildes, que expresen realmente que significa el método a implementar, que explique sólo con la palabra el propósito para el cual fue creada. Solo se deben usar mayúsculas entre la palabra únicamente en caso de

que se desee mejorar la legibilidad del nombre (como por ejemplo: cargarDatos, o guardarRegistro, o mostrarInformacion). También se pueden usar símbolos como guiones (-) o guiones bajos (_), pero no abuse de ellos (por ejemplo: cargar_datos, o guardar_registro, o mostrar_informacion). No se recomienda incluir números en el nombre del método.

Es necesario recordar que los métodos deben ser la única forma que tienen los objetos de comunicarse entre sí. La forma de transmitir valores a través de sus métodos se puede hacer en dos vías: recibiendo 0 ó varios datos a través de sus parámetros, y/o, enviando 1 o ningún dato de retorno. La "fórmula" (o sintaxis) para crear un método es la siguiente:

public tipo_de_dato_retorno nombreMetodo (listado_de_parámetros)

En el tipo de dato de retorno se coloca el tipo de dato que se espera retornar o la palabra *void* si no retornará dato. El método puede retornar una sola variable de cualquier tipo de dato Java (*byte, short, int, long, float, double, char, String*), o puede retornar un tipo de dato de clase. El listado de parámetros es una o un conjunto de variables declaradas cada una con el tipo de dato seguido de su propio nombre. Dentro de método se codifica el algoritmo que se requiera, colocando siempre punto y coma por cada orden que sea secuencial (definiciones de variables, asignaciones de variables, presentaciones de datos, solicitudes de datos). Al igual que las clases, para todos los métodos deben abrirse y cerrarse llaves, y colocar las órdenes de dicho método dentro de las llaves. Además, los métodos que requieran retornar un valor DEBEN llevar dentro de su definición y solamente al final una orden de retorno return, indicando el valor o la variable que retornará. Los métodos que NO retornan valor se les llama *void*, y no debe tener la orden de retorno al final del método.

Tabla 6-18 Codificación completa

```
1  package proyecto;
2  import javax.swing.JOptionPane;
3
4  public class Presentacion {
5
6     public Departamento capturarDepartamento(){
7        Departamento rDepartamento;
8        rDepartamento=new Departamento();
9        rDepartamento.nombre=JOptionPane.showInputDialog(null);
10       rDepartamento.potencialSufragantes=Integer.parseInt(JOptionPane.showInputDialog(null));
11       rDepartamento.totalSufragantes=Integer.parseInt(JOptionPane.showInputDialog(null));
12       return rDepartamento;
13    }
14
15    public void presentarDepartamento(Departamento pDepartamento){
16       JOptionPane.showMessageDialog(null, pDepartamento.nombre);
17       JOptionPane.showMessageDialog(null, pDepartamento.potencialSufragantes);
18       JOptionPane.showMessageDialog(null, pDepartamento.totalSufragantes);
19       JOptionPane.showMessageDialog(null, pDepartamento.calcularParticipacion());
20    }
21 }
```

Referencias

Alagic, S. (2017). *Software Engineering: Specification, Implementation, Verification* (1st ed.). Springer International Publishing.

Chemuturi, M. (2013). *Requirements Engineering and Management for Software Development Projects* (1st ed.). Springer.

Fowler, M., & Scott, K. (2000). *UML gota a gota* (1st ed.). Addison-Wesley.

Houben, C. (2014). Fostering Functional Safety of Real-time Systems with Concepts of Object Orientation. *19th Methods and Models in Automation and Robotics*.

Rumbaugh, J., Blaha, M., Premerlani, W., Frederick, E., & Lorensen, W. (1995). *Modelado y diseño orientados a objetos* (1st ed.). Prentice-Hall.

Rumbaugh, J., Jacobson, I., & Booch, G. (2000). *El Lenguaje Unificado de Modelado. Manual de referencia* (1st ed.). Addison-Wesley.

Sánchez, J., Huecas, G., Fernández, B., & Moreno, P. (2005). *Java 2: Iniciación y referencia* (2nd ed.). MacGraw-Hill.

Villalobos, J., & Casallas, R. (2006). *Fundamentos de programación: aprendizaje activo basado en casos : un enfoque moderno usando Java, UML, Objetos y Eclipse* (1st ed.). Pearson Education.

7 Mejoras básicas

En el capítulo anterior se dejó lista la base inicial para un proyecto de software en Programación Orientada a Objetos en Arquitectura de Capas. El primer prototipo abordó como meta general que el estudiante se empiece a familiarizar con las soluciones distribuidas, en este caso en el ámbito de la Arquitectura de Capas, y siguiendo un conjunto de buenas prácticas como fundamento de la Ingeniería del Software. Ahora, mediante dos prototipos adicionales, se empezará a aprovechar las ventajas de este enfoque en cuanto a la incorporación de nuevas características sin tener que recurrir a intervenciones o modificaciones complicadas de la estructura.

De aquí en adelante, se presentará el proceso de desarrollo de cada prototipo con la misma metodología, abordando las 3 fases: análisis, diseño e implementación. La Tabla 7.1 muestra las metas de aprendizaje para estos prototipos.

Tabla 7.1 Objetivos de aprendizaje del capítulo.

Desarrollo de software	Conceptos	Instrumentos
Análisis Diseño Implementación Tipos de requisitos Usabilidad	Clase: método constructor, paso de mensajes. Algoritmo: operadores matemáticos, tipos de datos, estructuras condicionales, conversión (*parsing*) de datos, *casting* de datos, cuadros de diálogo. Arquitectura en capas	UML: Diagrama de casos de uso Especificación de casos de uso Diagrama de clases Diagrama de flujo

7.1 Requerimientos de los prototipos 02 y 03

Con el prototipo inicial el usuario no recibe ninguna orientación con respecto a los datos que digita o consulta. Además, la información se le presenta dispersa. Por último, la presentación del indicador de participación electoral siempre presenta un valor entero, a pesar de que el tipo de dato se ha indicado como de tipo *double*.

El prototipo 02 busca mejoras en la interacción con el usuario, para que este reciba una orientación sobre el dato que debe digitar o que está observando. Asimismo, se busca integración de la presentación de datos y el cálculo correcto del indicador de participación electoral. Algunas mejoras se obtienen mediante cambios en la codificación, en tanto otras requieren cambios en el

diseño de algoritmos. La funcionalidad del prototipo 03 de la aplicación es la misma del prototipo 02, pero además busca determinar la categoría del departamento con base en su indicador de participación. De acuerdo con este indicador, los departamentos se clasifican en categorías de acuerdo con la siguiente tabla:

Tabla 7.2 Categorías de participación electoral.

Indicador de participación electoral	Categoría de participación electoral
Mayor o igual que 50%	Alta
Menor de 50% pero mayor o igual de 40%	Media
Menor que 40%	Baja

7.1.1 Análisis

7.1.1.1 Alcance

En el prototipo 01 el usuario no se le brinda ninguna orientación con respecto a los datos que se le solicitan durante la captura, lo cual no es conveniente dado que es posible que el usuario ingrese la información de forma incorrecta, por ejemplo, si no conoce el orden. Asimismo, durante la presentación de información, cada dato se muestra sin que al usuario se le indique qué dato está observando, y cada uno se le presenta en una ventana nueva.

La funcionalidad del prototipo 02 de la aplicación es la misma del prototipo 01, pero con mejoras en la interacción con el usuario. Cada ventana de captura de datos debe mostrar la orientación suficiente sobre el dato que solicita. En cuanto a la presentación de datos se debe hacer en una sola ventana o **cuadro de diálogo**, con los rótulos necesarios para que el usuario entienda a qué corresponde cada uno. Finalmente, a pesar de que la fórmula del cálculo del indicador de participación electoral está correcta, el prototipo anterior no presentó un dato de tipo real sino un valor entero terminando con .0 (punto cero). En este prototipo se explicará la causa de esta presentación y su solución. Con esto se mejora la interacción con el usuario, dicho en otras palabras, se mejoró la usabilidad del software. Además, se corrigieron errores en el cálculo. La funcionalidad del prototipo 03 busca adicionar un nuevo elemento importante para la clasificación de los departamentos. En la Tabla 7.3 se presenta una relación de los objetivos por prototipo:

Tabla 7.3 Relación de Objetivos por prototipo

Prototipo	Objetivo general
02	Mejorar la interacción con el usuario para orientar al usuario de los datos a ingresar
03	Integrar cálculos adicionales para determinar la categoría del departamento

7.1.1.2 Definiciones

Para el prototipo 03 aparece una nueva definición en la lógica del negocio, llamado categoría de participación electoral. Este nuevo elemento se incorpora a la lista de definiciones. De acuerdo con el proceso de abstracción, este elemento pertenece al objeto departamento, pero depende

particularmente de otro atributo del mismo objeto. Al igual que el indicador de participación electoral, la categoría de participación podría tomarse o no como un atributo adicional, pero debe tenerse claro que este valor no será solicitado, sino determinado dependiendo del indicador de participación.

Tabla 7.4 Definiciones aplicables en desarrollo de los prototipos del capítulo

	Definición	Tipo de dato
Departamento	Cada una de las circunscripciones territoriales de primer nivel en que se divide el país para propósitos electorales	Objeto
Nombre	Identificación nominal del departamento	Cadena de caracteres
Potencial de sufragantes	Cantidad de personas habilitadas en un departamento para votar en un proceso electoral	Número entero
Total de sufragantes	Cantidad de personas que acudieron a votar en un departamento en un proceso electoral	Número entero
Indicador de participación electoral	Indicador consistente en el total de sufragantes expresado como porcentaje del potencial de sufragantes $Indicador\ de\ participación\ electoral = \frac{Total\ de\ sufragantes}{Potencial\ de\ sufragantes} * 100$	Número real
Categoría de participación electoral	Categoría a la que pertenece el departamento, dependiendo del porcentaje de participación electoral: Si el porcentaje es mayor o igual a 50%: alto Si el porcentaje es menor a 50% pero mayor o igual a 40%: medio Si el porcentaje es menor a 40%: bajo	Cadena de caracteres

7.1.1.3 Requisitos

De acuerdo con el alcance presentado, para el prototipo 02, se presenta una modificación en el requisito de interfaz de usuario. Los requisitos de interfaz de usuario son un **tipo de requisito** no funcional que permiten determinar el comportamiento del sistema frente al sistema, buscando así mejorar la **usabilidad** de la aplicación. En el alcance presentado en el prototipo 03, se presenta además una modificación en el requisito funcional.

7.1.1.3.1 Requisitos de interfaz de usuario

7.1.1.3.1.1 RI-01 Ingreso de información en casillas de edición

Para la interacción con el usuario, la aplicación debe presentar ventanas con casillas de edición en donde pueda digitar los datos solicitados. Al solicitar los datos, el sistema deberá indicar en la misma ventana de diálogo cuál es el dato requerido. A su vez, la aplicación presentará toda la información del departamento en una sola ventana que el usuario podrá controlar mediante un botón de Aceptar.

7.1.1.3.2 Requisitos funcionales

7.1.1.3.2.1 RF-01 Calcular el indicador de participación electoral de un departamento

La aplicación debe permitir al usuario introducir los siguientes datos básicos de un departamento: el nombre, el potencial de sufragantes y el total de sufragantes, y con estos datos debe calcular el indicador de participación electoral.

Una vez digitados los datos básicos, la aplicación debe mostrar por pantalla los mismos datos, y además presentar el indicador de participación electoral calculado y la categoría de participación a la que pertenece, de acuerdo con las reglas presentadas en la Tabla 10.2.

7.1.2 Diseño

7.1.2.1 Diseño de escenarios

No hay escenarios nuevos con respecto al primer prototipo. Se mantiene entonces el mismo diagrama de casos de uso.

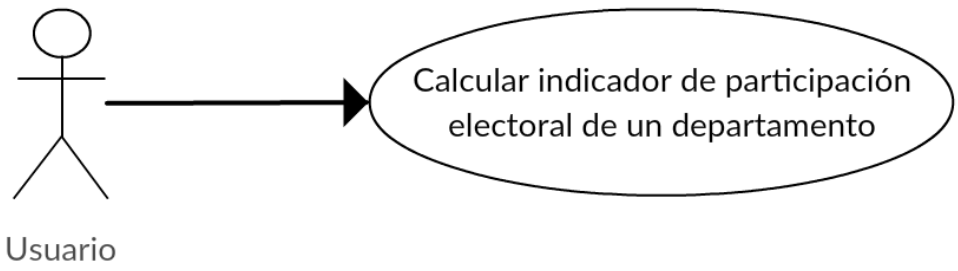


Figura 7.1 Diagrama de casos de uso para los prototipos 02 y 03

Sin embargo, se debe evidenciar en la especificación las mejoras planteadas. Para cumplir con el requisito de interfaz (RI-01), cada caja de texto debe indicarle al usuario cuál es el dato que se debe ingresar. Luego se deberá modificar la función de ingreso de información para ello. Adicionalmente se reduce la cantidad de **cuadros de diálogo** a presentar debido a que en el requisito se ha dejado indicado que todos los datos del departamento se presentarán en una sola ventana a la vez. Esto permite mejorar la **usabilidad** de la aplicación. Además se indica en la especificación la lista de los datos a presentarle al usuario, dentro de esta lista se incluye la categoría de participación a la que pertenece el departamento, con el fin de cumplir con el requisito funcional (RF-01).

Tabla 7.5 Especificación del caso de uso Calcular indicador de participación electoral de un departamento.

1	El usuario ingresa a la aplicación.
2	La aplicación presenta una pantalla con una caja de texto para que el usuario digite el nombre del departamento
3	El usuario digita el nombre del departamento y pulsa el botón *Aceptar*

4	La aplicación presenta una pantalla con una caja de texto para que el usuario digite el potencial de sufragantes del departamento
5	El usuario digita el potencial de sufragantes del departamento y pulsa el botón *Aceptar*
6	La aplicación presenta una pantalla con una caja de texto para que el usuario digite el total de sufragantes del departamento
7	El usuario digita el total de sufragantes del departamento y pulsa el botón *Aceptar*
8	La aplicación muestra en pantalla el nombre del departamento, el potencial de sufragantes, el total de sufragantes, el indicador de participación electoral del departamento y la categoría de participación a la que pertenece.
9	El usuario pulsa el botón *Aceptar* para terminar la ejecución de la aplicación

7.1.2.2 Diseño de estructura

Debido a que la **arquitectura en capas** refiere a establecer responsabilidades por cada clase, se debe precisar que es en la clase de la capa de presentación (es decir, la clase encargada de la comunicación con el usuario del sistema) donde se deben realizar los cambios indicados en los requisitos. Cada uno de los objetos de una capa puede comunicarse con otros objetos de otra capa a través de un **paso de mensajes**, que puede ser unidireccional (envío de datos desde el objeto emisor hacia el objeto receptor o viceversa), bidireccional (envío y retorno de datos) o puede simplemente no enviarse datos. Esto se realiza invocando el método del objeto receptor desde el objeto emisor.

Adicionalmente, y como buena práctica de programación, se recomienda que los objetos nuevos (y en particular los objetos de la capa de reglas, es decir los que representan objetos de negocio) se creen con un conjunto de valores iniciales predeterminados para sus atributos. Para lograr esto se cuenta con un método que permite inicializar los valores de los atributos del objeto creado, y se le llama **método constructor**.

Este método deberá tenerse en cuenta en la estructura para todas las clases, y muy particularmente para las clases de la capa de reglas. La única clase que no debe tener método constructor es la clase Main. La estructura no cambia respecto al prototipo inicial, por lo tanto, se mantienen las mismas clases en sus respectivas capas que indican sus responsabilidades particulares.

Tabla 7.6 Organización de la estructura de la aplicación en capas

Capa	Clase	Responsabilidad
Presentación	Presentación	Encargada del proceso de comunicación con el usuario. Cuenta con funcionalidades de solicitud y presentación de información.
Reglas de negocio	Departamento	Plantilla para crear un objeto de tipo *departamento*, que contenga los atributos y métodos requeridos.
Reglas de negocio	Main	Coordina las acciones de las otras clases
Persistencia de datos	-	-

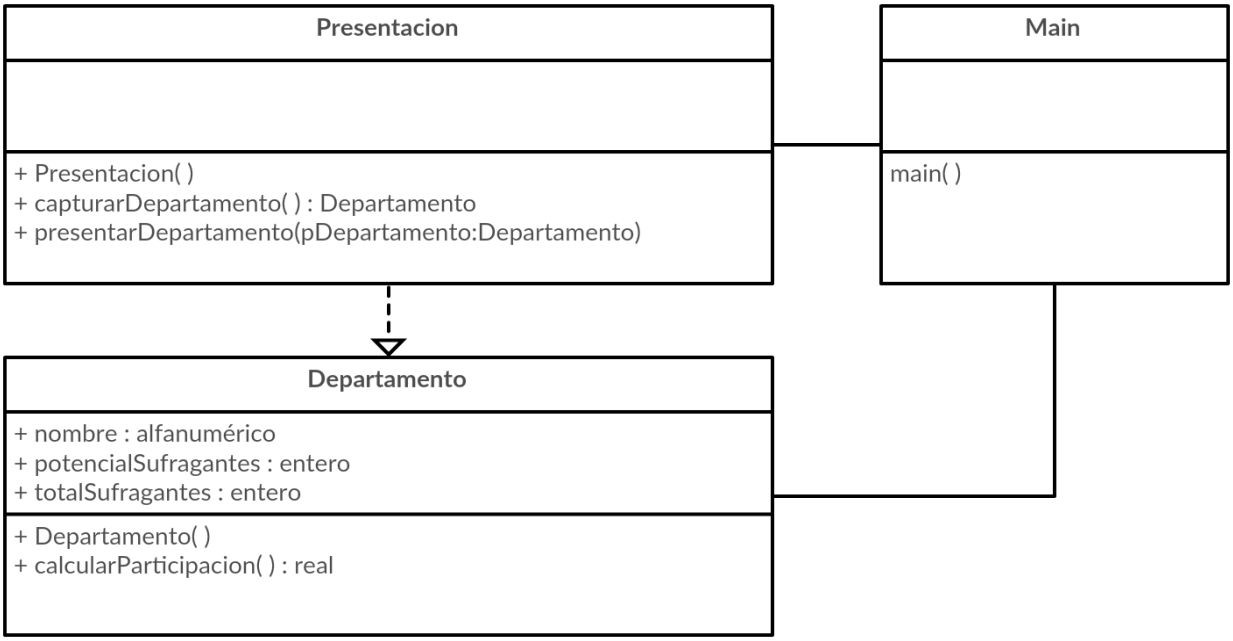


Figura 7.1 Diagrama de clases para el Prototipo 02

En el método de *calcularParticipacion* se presenta una situación particular en el cálculo: a pesar de que la fórmula está correcta, los datos que se generan son enteros. Esto es porque en Java, las **operaciones matemáticas** mantienen los **tipos de datos** usados en sus operandos, así sea una división. Esto significa que, para la operación matemática, al usar 3 operandos enteros, el resultado generado es un número entero, sin importar que hay una división de por medio. Para evitar este inconveniente, se debe hacer una **conversión especial llamada *casting*** al momento de la operación.

Adicionalmente, hay una nueva funcionalidad que indica que debe implementarse un método para determinar la categoría de participación electoral. Dado que los datos para calcular pertenecen al departamento, y que la categoría de participación electoral también pertenece al departamento, se debe agregar esta funcionalidad a la clase Departamento a través de un método llamado *calcularCategoriaParticipacion().* En este método se utilizará una **estructura de código condicional** anidada que permita implementar las reglas que se indicaron para su clasificación. Se recomienda referirse a los textos y libros de fundamentos de programación si no está familiarizado con las diferentes estructuras de control condicionales. La estructura de clases no cambia, se mantienen las mismas clases.

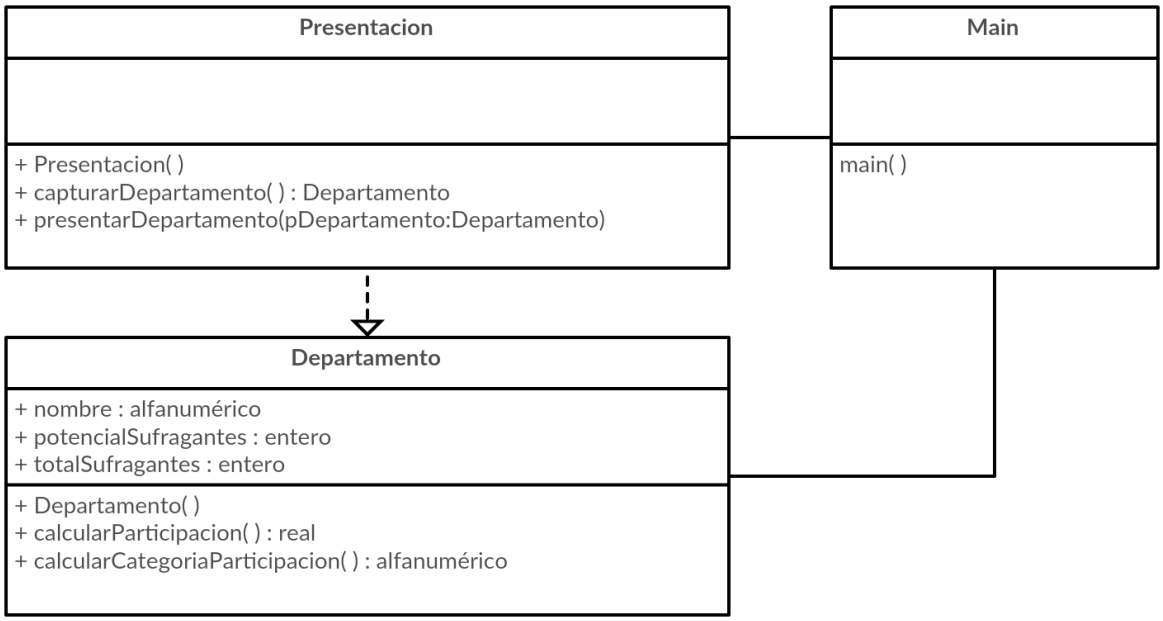


Figura 7.2 Diagrama de clases para el Prototipo 03

7.1.2.3 Diseño de algoritmos

7.1.2.3.1 Clase Departamento - Departamento( )

Descripción: Como método constructor, inicializa los atributos con valores neutros. Su algoritmo es secuencial.

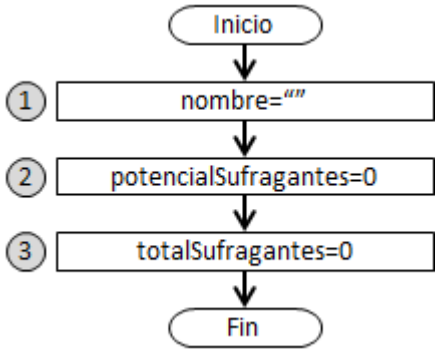


Figura 7.3 Diagrama de flujo para el método Departamento()

Tabla 7.7 Descripción del algoritmo para el método constructor Departamento()

1	Almacena el valor "" en el atributo nombre de cualquier objeto de tipo Departamento al momento de crearse.
2	Almacena el valor 0 en el atributo potencialSufragantes de cualquier objeto de tipo Departamento al momento de crearse
3	Almacena el valor 0 en el atributo totalSufragantes de cualquier objeto de tipo Departamento al momento de crearse

7.1.2.3.2 Clase Departamento – calcularParticipacion( ): real

Descripción: Como método de cálculo, aplica una fórmula para obtener como resultado el indicador de participación electoral con base en el potencial y el total de sufragantes del departamento. Debe devolver un número real. En esta parte se debe aplicar la **conversión de tipo casting** para el cálculo del índice de participación.

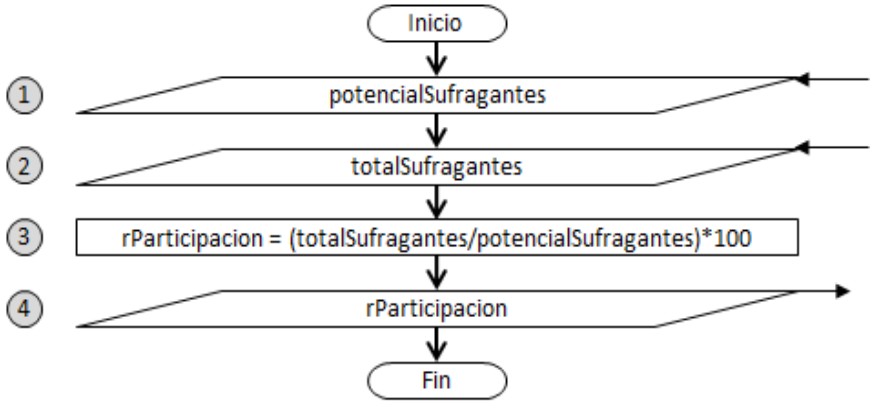


Figura 7.4 Diagrama de flujo para el método calcularParticipacion()

Tabla 7.8 Descripción del algoritmo para el método calcularParticipacion()

1	Se requiere como dato de entrada *potencialSufragantes,* se cumple por tener este dato como atributo propio.
2	Se requiere como dato de entrada *totalSufragantes,* se cumple por tener este dato como atributo propio.
3	Divide *totalSufragantes* entre *potencialSufragantes* y multiplica por cien (100). *totalSufragantes* debe ser tratado como (double) al momento del cálculo. Almacena el resultado de la operación en la variable *rParticipacion*.
4	Se genera como dato de salida o respuesta del método el valor de la variable *rParticipacion*.

7.1.2.3.3 Clase Departamento – calcularCategoriaParticipacion( ): cadena

Descripción: Método de cálculo para determinar la categoría de participación electoral del departamento, con base en el indicador de participación calculado por el método anterior.

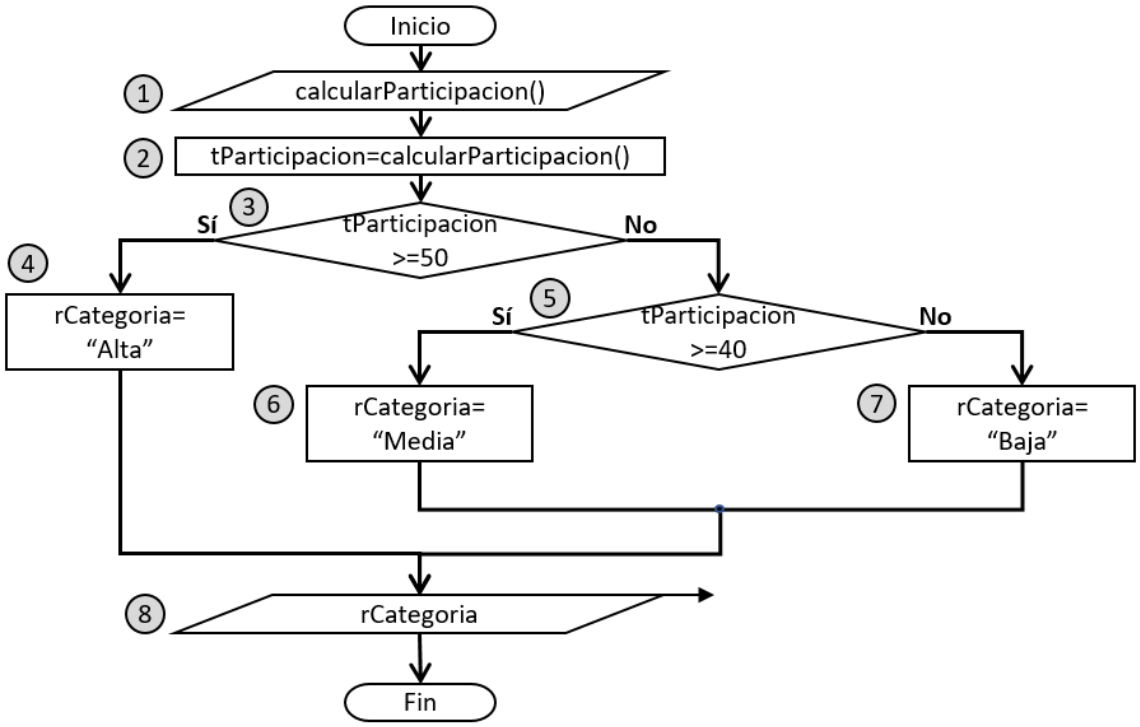


Figura 7.5 Diagrama de flujo para el método calcularCategoriaParticipacion()

Tabla 7.9 Descripción del algoritmo para el método calcularCategoriaParticipacion ()

1	Se requiere como dato de entrada el indicador de participación electoral del propio objeto. Para ello ordena al propio objeto que ejecute su método *calcularParticipacion()* y almacena en la variable tParticipacion la respuesta del método.
2	Si el valor tParticipacion es mayor o igual a 50:
3	Asigna en la variable rCategoria el valor “Alta”
4	De lo contrario: Si el valor tParticipacion es menor a 50 y el valor tParticipacion es mayor o igual a 40:
5	Asigna en la variable rCategoria el valor “Media”
6	De lo contrario: Asigna en la variable rCategoria el valor “Baja”
7	Se genera como dato de salida o respuesta del método el valor de la variable *rCategoria*.

7.1.2.3.4 Clase Presentacion - capturarDepartamento( ):Departamento

Descripción: Obtiene del usuario cada uno de los valores para los atributos, y entrega como respuesta el objeto de la clase Departamento con los valores de los atributos asignados. Debe devolver un objeto de tipo Departamento. Solo se presenta el diagrama de flujo y no la tabla descriptiva porque no hay cambios en este algoritmo respecto al prototipo previo.

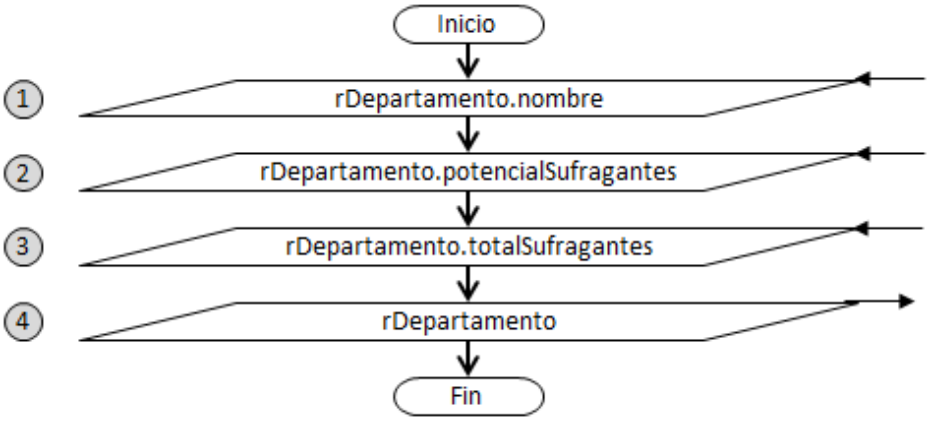


Figura 7.6 Diagrama de flujo para el método capturarDepartamento()

7.1.2.3.5 Clase Presentacion - presentarDepartamento(pDepartamento:Departamento)

Descripción: Recibe un objeto de la clase Departamento como parámetro. Construye un mensaje al que agrega cada uno de los valores de los atributos del objeto recibido, así como el resultado del método de cálculo del indicador de participación electoral y el resultado del método de cálculo de la categoría de participación electoral. Por último presenta el mensaje construido al usuario.

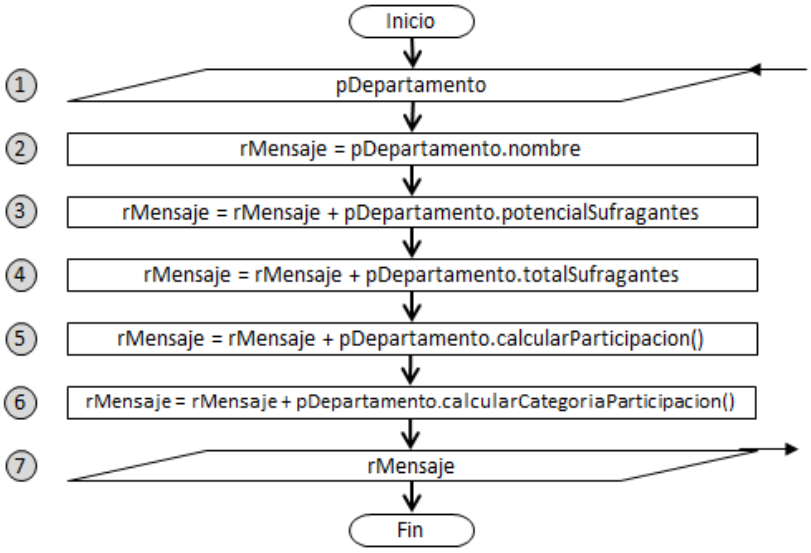


Figura 7.7 Diagrama de flujo para el método presentarDepartamento()

Tabla 7.10 Descripción del algoritmo para el método presentarDepartamento()

1	Se requiere como dato de entrada el objeto de tipo Departamento. En este caso se recibe el objeto como mensaje.
2	Asigna el nombre del departamento del objeto en la variable rMensaje

3	Concatena en la variable rMensaje el potencial de sufragantes del departamento almacenado en el objeto
4	Concatena en la variable rMensaje el total de sufragantes del departamento almacenado en el objeto
5	Concatena en la variable rMensaje la respuesta del método calcularParticipacion() del objeto.
6	Concatena en la variable rMensaje la respuesta del método calcularCategoriaParticipacion() del objeto.
7	Se genera como dato de salida el mensaje concatenado

7.1.2.3.6 Clase Main – main( )

Descripción: Coordina las acciones de las demás clases. Solicita a un objeto de la clase Presentacion que lleve a cabo la captura de los datos de un objeto de la clase Departamento, cuyos datos almacena luego en el objeto departamento. A continuación solicita al objeto de la clase Presentacion que lleve a cabo el método de presentación tomando como parámetro el objeto de la clase Departamento. No hay cambios respecto al prototipo anterior, por eso no se presenta la tabla descriptiva.

Diagrama de flujo para el método main()

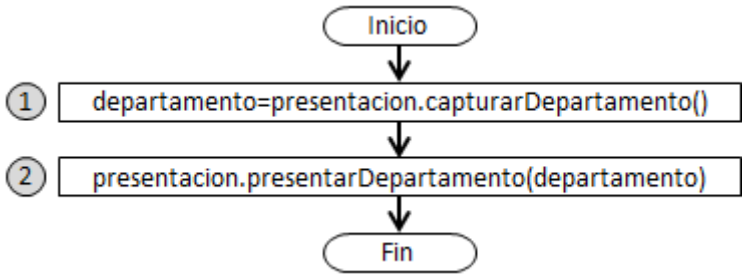


Figura 7.8 Diagrama de flujo para el método main()

7.1.3 Implementación

La estructura del proyecto no cambia. Solo se agrega el método nuevo en la clase Departamento para calcular la categoría de participación electoral.

Tabla 7.11 Codificación de la clase Departamento

```
1   package proyecto;
2
3   public class Departamento {
4      public String nombre;
5      public int potencialSufragantes;
6      public int totalSufragantes;
7
8      public Departamento() {
9         nombre = "";
10        potencialSufragantes = 0;
11        totalSufragantes = 0;
12     }
13
14     public double calcularParticipacion(){
15        double rParticipacion;
16        rParticipacion = ((double) totalSufragantes / potencialSufragantes) * 100;
17        return rParticipacion;
```

```
18      }
19
20      public String calcularCategoriaParticipacion() {
21          String rCategoria;
22          double tParticipacion;
23          tParticipacion = calcularParticipacion();
24          if (tParticipacion >= 50) {
25              rCategoria = “Alta”;
26          }
27          else {
28              if (tParticipacion >= 40) {
29                  rCategoria = “Media”;
30              }
31              else {
32                  rCategoria = “Baja”;
33              }
34          }
35          return rCategoria;
36      }
    }
```

Tabla 7.12 Codificación de la clase Presentacion

```
1   package proyecto;
2   import javax.swing.JOptionPane;
3
4   public class Presentacion {
5
6       public Presentacion() {
7
8       }
9
10      public Departamento capturarDepartamento() {
11          Departamento rDepartamento;
12          rDepartamento=new Departamento();
13          rDepartamento.nombre=JOptionPane.showInputDialog("Nombre: ");
14          rDepartamento.potencialSufragantes=Integer.parseInt(JOptionPane.showInputDialog("Potencial de
        sufragantes: "));
15          rDepartamento.totalSufragantes=Integer.parseInt(JOptionPane.showInputDialog("Total de
        sufragantes: "));
16          return rDepartamento;
17      }
18
19      public void presentarDepartamento(Departamento pDepartamento){
20          String rMensaje;
21          rMensaje="Departamento: "+pDepartamento.nombre;
22          rMensaje=rMensaje+"\nPotencial de sufragantes: "+pDepartamento.potencialSufragantes;
23          rMensaje=rMensaje+"\nTotal de sufragantes: "+pDepartamento.totalSufragantes;
24          rMensaje=rMensaje+"\nIndicador de participación: "+pDepartamento.calcularParticipacion()+" %";
25          rMensaje=rMensaje+"\nCategoría según participación:
        "+pDepartamento.calcularCategoriaParticipacion();
```

```
26          JOptionPane.showMessageDialog(null, rMensaje);
27      }
28  }
```

Tabla 7.13 Codificación de la clase Main

```
1   package proyecto;
2
3   public class Main {
4
5       public static void main(String[] args) {
6           Departamento departamento;
7           Presentacion presentacion;
8           presentacion=new Presentacion();
9           departamento=presentacion.capturarDepartamento();
10          presentacion.presentarDepartamento(departamento);
11      }
12  }
```

7.2 Síntesis de conceptos

7.2.1 Orientación a objetos

Refiriéndose a orientación a objetos, éstos presentan unas características como son: su **identidad**, la cual es la propiedad que permite diferenciar un objeto de otro; su **estado**, el cual es el conjunto de valores con los que cuentan sus atributos en un momento determinado; y su **comportamiento** que hace referencia a las capacidades, acciones y reacciones del objeto, es decir, el conjunto de métodos que fueron programados para dicho objeto. Por medio de la invocación de un método, un objeto solicita al otro objeto realizar o ejecutar algún tipo de acción relacionada. Al invocar un método, se puede realizar un **paso de mensajes**, es decir, se puede pasar información desde el objeto que invoca (emisor) hacia el invocado (receptor) y/o viceversa. Existen diferentes tipos de métodos como **constructores** (que permiten crean objetos), destructores (que se encargan de destruir los objetos, es decir eliminarlos de la memoria), selectores o *getters* (se usan para devolver todo o parte del estado de un objeto sin alterarlo), modificadores o *setters* (su función es cambiar todo o parte del objeto), iteradores (encargadas de visitar el contenido de una estructura de datos), entre otros.

Para el desarrollo de un programa se debe indicar o identificar cada uno de los elementos que serán utilizados en este. Uno de esos elementos son las variables, espacios de almacenamiento en la memoria usados con el fin de mantener datos para su procesamiento, de las cuales se debe indicar el nombre y que tipo de datos almacenará. Puesto que las variables serán utilizadas en algún tipo de operación en tiempo de ejecución en el programa, se recomienda inicializarlas, es decir, darles un valor inicial; esto es importante para evitar cualquier tipo de error en el momento de compilar el programa. Para esta inicialización se utiliza el **método constructor,** el cual toma el mismo nombre de la clase y su único propósito es realizar la inicialización de los datos de un nuevo

objeto, que cuando es creado se ejecuta de forma automática. Finalmente es importante resaltar tres reglas del **método constructor**: El constructor debe tener el mismo nombre de la clase, puede tener cero o más parámetros, y no devuelve ningún valor (Joyanes Aguilar & Zahonero Martínez, 2015).

7.2.2 Programación en Java – Buenas prácticas

- **Operadores**

Se deben tener en cuenta una serie de operadores en caso de que se requiera hacer operaciones aritméticas u operaciones lógicas. Las operaciones matemáticas únicamente se pueden realizar con tipos de datos numéricos, y nunca se puede usar texto para realizar operaciones. Estos operadores son:

Tabla 7.14 Operadores

Tipo de operador	Operador	Descripción
Operadores Aritméticos	-	Resta
	+	Suma
	*	Producto
	/	División
	%	Resto de una división entre enteros
Operadores de Relación	==	Igual que
	!=	Diferente que
	<	Menor que
	>	Mayor que
	<=	Menor o igual que
	>=	Mayor o igual que
Operadores Lógicos	!	Negación – NOT
	|	Suma lógica – OR
	^	Suma lógica exclusiva
	||	O
	&&	Y

- **Tipos de datos, *parsing* y *casting***

Un tipo de datos es la propiedad de una variable o atributo de una clase que determina qué valores puede tomar, cuáles operaciones se le pueden aplicar y cómo se almacena en la memoria. En Java, los tipos de datos básicos (o primitivos) usados son los siguientes:

Tabla 7.15 Tipos de Datos Java

Tipo de dato Java	Representación	Tamaño (Bytes)	Rango de Valores
byte	Numérico Entero con signo	1	-128 a 127
short	Numérico Entero con signo	2	-32768 a 32767
int	Numérico Entero con signo	4	-2147483648 a 2147483647

long	Numérico Entero con signo	8	-9223372036854775808 a 9223372036854775807
float	Numérico en Coma flotante de precisión simple Norma IEEE 754	4	±3.4x10^-38 a ±3.4x10^38
double	Numérico en Coma flotante de precisión doble Norma IEEE 754	8	±1.8x10^-308 a ±1.8x10^308
char	Caracter Unicode	2	\u0000 a \uFFFF
boolean	Dato lógico	-	true ó false
void	-	-	-

Durante el ingreso de los datos de entrada en Java, se debe considerar que hay algunos datos que se capturan como tipo String, un **tipo de datos** no primitivo que representa un conjunto o cadena de caracteres secuenciales. Al momento de realizar una captura de datos en una variable de tipo String, se considera que se está capturando texto. Sin embargo, cuando los datos capturados son numéricos, es necesario convertir el texto en un número de algún tipo de dato primitivo para poder realizar operaciones matemáticas, dado que los datos de tipo texto no permiten realizar estas operaciones. Para ello se usa el **parsing**, una técnica que permite realizar la conversión de un dato de texto (String) a una variable numérica para poder realizar los cálculos matemáticos requeridos. Los métodos usados para realizar el parsing son los siguientes:

Tabla 7.16 Métodos de Parsing

Método parsing	**Objetivo**
Byte.parseByte(variable_texto)	Convierte un texto capturado a un valor primitivo byte
Short.parseShort(variable_texto)	Convierte un texto capturado a un valor primitivo short
Integer.parseInt(variable_texto)	Convierte un texto capturado a un valor primitivo int
Long.parseLong(variable_texto)	Convierte un texto capturado a un valor primitivo long
Float.parseFloat(variable_texto)	Convierte un texto capturado a un valor primitivo float
Double.parseDouble(variable_texto)	Convierte un texto capturado a un valor primitivo double
Boolean.parseBoolean(variable_texto)	Convierte un texto capturado a un valor primitivo boolean

Por otra parte, cuando se requiere modificar el tipo de dato primitivo de una variable se realiza un procedimiento que convierte el valor de una variable en un tipo diferente al que inicialmente se definió. Este procedimiento se le conoce como **casting**. Para realizar esta conversión se debe indicar el tipo nuevo y la variable que se quiere convertir. La sintaxis es:

(*tipo*) valor_a_convertir

Esto quiere indicar que, en tiempo de ejecución, se requiere que la variable representada por *valor_a_convertir* presente el dato de forma temporal como el tipo de dato indicado dentro del paréntesis. Existen dos tipos de ***casting***, el *casting* implícito (*widening casting*) que se utiliza cuando un valor pequeño se ubica en un contenedor (variable) más grande; y el *casting* explícito (*narrowing casting*) en el que se ubica un valor grande en un contenedor pequeño, y es posible que ocurra perdida de datos.

- **Librería *JOptionPane***

En Java se encuentra el paquete javax.swing, el cual contiene la librería ***JOptionPane*** (Oracle, 2020). Esta librería facilita la creación y apertura de un **cuadro de dialogo**, que permite solicitar un valor, o informar a los usuarios. Esta librería contiene cuatro diferentes cuadros de dialogo que son:

Tabla 7.17 Cuadros de Diálogo

Método	Descripción	Sintaxis
showConfirmDialog	Este cuadro de diálogo muestra una pregunta de confirmación con las opciones si/no/cancelar	showConfirmDialog(Component parentComponent, Object message, String title, int optionType)
showInputDialog	Este cuadro de diálogo solicita un valor de entrada.	showInputDialog(Component parentComponent, Object message, String title, int messageType)
showMessageDialog	Este cuadro de diálogo muestra un mensaje con algo que ha sucedido.	showMessageDialog(Component parentComponent, Object message, String title, int messageType, Icon icon)
showOptionDialog	Este cuadro de diálogo permite unificar los tres anteriores cuadros.	showOptionDialog(Component parentComponent, Object message, String title, int optionType, int messageType, Icon icon, Object[] options, Object initialValue)

La librería está compuesta por unos parámetros que son:

Tabla 7.18 Parámetros de los Cuadros de Diálogo

Parámetro	Descripción
parentComponent	Este definirá el componente que será el padre del cuadro de diálogo.
message	Una cadena de caracteres que compone un mensaje descriptivo que se desee colocar en el cuadro de diálogo.
Icon	Permite la posibilidad de personalizar el cuadro de dialogo incluyendo un icono.
MessageType	Define el estilo de mensaje. Las posibles opciones son: ERROR_MESSAGE INFORMATION_MESSAGE WARNING_MESSAGE QUESTION_MESSAGE PLAIN_MESSAGE
OptionType	Define la opción de botones que puede tener el cuadro de diálogo. Las opciones son: DEFAULT_OPTION YES_NO_OPTION YES_NO_CANCEL_OPTION OK_CANCEL_OPTION

7.2.3 Ingeniería del software

Los **requisitos** del sistema hacen referencia a las características y servicios que se esperan obtener del sistema, así como las diferentes restricciones de este. Cuando se identifican los **requisitos** se debe tener en cuenta, las necesidades (requerimientos) de los clientes y como estos requisitos ayudan en la solución de una problemática. Así mismo se identifican diferentes tipos de **requisitos** entre los cuales están (Sommerville, 2012):

- **Funcionales**: Estos requisitos se tienen en cuenta para identificar el sistema que debe hacer, teniendo en cuenta el tipo de software y los usuarios.
- **No Funcionales**: Estos requisitos son aquellos que no están directamente relacionados con las funciones del sistema, estos tienen en cuenta las propiedades como, el tiempo de respuesta, capacidad de almacenamiento, aspectos de diseño, aspectos legales y éticos y seguridad. Estos requisitos van muy de la mano con los aspectos de calidad del sistema, por lo que se le conocen también como atributos de calidad. Entre estos se encuentran la confiabilidad, la seguridad, la portabilidad, la eficiencia, la mantenibilidad, la **usabilidad**, entre otros.
- **Sistema**: Describen el comportamiento externo del sistema y sus diferentes restricciones. A diferencia que los requisitos de usuario si pueden contener lenguaje técnico, ya que no van dirigidos a la compresión del usuario del sistema, sino a los ingenieros. Las especificaciones deben ser del sistema completo, sin tener en cuenta como se debe diseñar o implementar.
- **Usuario**: Describen los requisitos funcionales y no funcionales, de forma que sean comprendidos por los usuarios del sistema, teniendo en cuenta que no poseen conocimientos técnicos. Se expone el comportamiento externo del sistema, evitando vocabulario especializado o técnico.

Un aspecto importante de calidad que se debe tener en cuenta cuando se realiza un software es la **usabilidad**, la cual indica si puede soportar cualquier tipo de usuario y aplicar la sintaxis y semántica de la navegación que se requiere. La **usabilidad** es importante para cualquier tipo de software, ya que debe permitir al usuario utilizarlo sin esfuerzo, comprendiendo cada opción definida en la interfaz. Si un programa es complejo para el usuario, así contenga funciones valiosas, no tendrá éxito, ya que, puede generar frustración al usuario al no comprender como utilizarlo. En conclusión esta propiedad, refleja la sencillez del sistema y depende de los componentes técnicos del sistema, operaciones y del entorno operacional (Pressman, 2012).

Referencias

Joyanes Aguilar, L., & Zahonero Martínez, I. (2015). *Programación en C/C++, Java y UML* (2nd ed.). McGraw-Hill/Interamericana de España, S. A. U.

Oracle. (2020). *Java™ Platform, Standard Edition 7 API Specification*. https://docs.oracle.com/javase/7/docs/api/

Pressman, R. (2012). *Ingeniería del Software* (5th ed.). McGraw Hill.

Sommerville, I. (2012). *Ingeniería del Software* (9th ed.). Pearson Education.

8 Persistencia de datos

Cada uno de los prototipos anteriores ha permitido el ingreso de datos para presentar una salida, es decir un resultado, de la misma forma que la gran mayoría de aplicaciones de computador. No obstante, en dichas aplicaciones los datos ingresados previamente se retoman o se recuperan de alguna forma, mientras que en estos prototipos se desaparecen al terminar la ejecución del programa. La funcionalidad que permite mantener los datos ingresados en una aplicación para su almacenamiento, recuperación y procesamiento posterior se le conoce como persistencia de datos. Se planteará en esta serie de prototipos la forma de implementación de las funcionalidades de persistencia de los datos.

Dado que se ha trabajado con el caso de estudio a través de prototipado evolutivo desde los capítulos previos, se podrá evidenciar el avance en el desarrollo de la aplicación, incluyendo nuevas funciones como las de persistencia, con las restricciones en el ámbito del desarrollo de software: el uso de la Programación Orientada a Objetos y la aplicación de la Arquitectura de Capas. La Tabla 4.4 muestra las metas de aprendizaje para estos prototipos.

Tabla 8.1 Objetivos de aprendizaje del capítulo.

Desarrollo de software	Conceptos	Instrumentos
Análisis Diseño Implementación Mantenibilidad	Clase: responsabilidad, tarjeta CRC Algoritmo: persistencia (almacenamiento y recuperación) de datos, cadenas de caracteres, concatenación, archivos CSV, manejo de excepciones	Flujo normal de eventos Flujos alternos de eventos UML: Diagrama de casos de uso Especificación de casos de uso Diagrama de clases Diagrama de flujo

8.1 Almacenamiento de datos – Prototipo 4

El prototipo 4 debe hacer lo mismo que el prototipo 3, pero además debe almacenar en un archivo tipo texto los datos ingresados por el usuario, en registros de valores separados por comas. No se contempla todavía recuperación de los datos.

8.1.1 Análisis

8.1.1.1 Alcance

En los prototipos anteriores se han ingresado valores para presentar un resultado de forma inmediata, pero dichos datos desaparecen al terminar la ejecución del programa. Se busca ahora

que los datos ingresados no se borren, sino que se mantengan en un archivo plano, es decir, persistan fuera de la ejecución del programa.

8.1.1.2 Definiciones

Se mantienen las mismas definiciones en la lógica del negocio, no aparecen nuevas definiciones ni otros objetos de negocio. Ver Tabla 7.4 del capítulo anterior.

8.1.1.3 Requisitos

De acuerdo con el alcance presentado, se presenta una adición en los requisitos funcionales, la cual es el **almacenamiento de datos** de un departamento en un archivo plano. Por el momento, en este prototipo, no se tiene contemplado el proceso de recuperación de los datos en la misma aplicación. Los requisitos de interfaz de usuario como el *Ingreso de información en casillas de edición* permanecen como en el prototipo 3. Ver sección 7.1.1.3.1 Requisitos de interfaz de usuario del capítulo anterior.

8.1.1.3.1 Requisitos funcionales

8.1.1.3.1.1 Calcular el indicador de participación electoral de un departamento

Este requisito funcional, proveniente del prototipo 3 mantiene su definición para el prototipo 4, que se ocupará de las funcionalidades relacionadas con la persistencia de datos.

8.1.1.3.1.2 Almacenar la información de un departamento en un archivo plano

La aplicación debe poder almacenar en un archivo plano los datos básicos ingresados de un departamento: el nombre, el potencial de sufragantes y el total de sufragantes. Se requiere que el formato de almacenamiento sea en datos separados por punto y coma (formato csv).

8.1.2 Diseño

8.1.2.1 Diseño de escenarios

No hay escenarios nuevos con respecto al primer prototipo. Se mantiene entonces el mismo diagrama de casos de uso. Ver Figura 7.1 Diagrama de casos de uso para los prototipos 02 y 03 y su respectiva especificación en la tabla 7.5 del capítulo anterior.

El **almacenamiento de datos** en un archivo plano no se está diseñando para que el usuario interactúe con la funcionalidad, es decir, el usuario no da la orden, sino que será realizada de forma automática, por esta razón esta función no se especifica en el caso de uso.

8.1.2.2 Diseño de estructura

Se debe implementar la funcionalidad para tomar los datos del departamento, organizarlos en un formato particular (**CSV**) y almacenar esta información formateada en un archivo de texto. Como se había explicado previamente en la arquitectura de capas, se debe disgregar esta

funcionalidad en una clase que se encargue específicamente del proceso de **persistencia** de los datos. Esto significa que una clase particular debe tener asignada esta **responsabilidad**. Se generará entonces una nueva clase llamada Datos, y se deberá tener en cuenta dos funciones en dicha clase:

- La primera función es tomar los datos que se hayan tomado del departamento y organizarlos en un formato particular llamado **CSV**, que es un formato que toma cada uno de los datos de un objeto, los **concatena** separados por una coma o por punto y coma, y los organiza en una sola **cadena de caracteres**.
- La otra función es tomar esta información formateada y almacenarlo en un archivo de texto. El acceso a almacenamiento a un archivo tiene una particularidad: se debe identificar si hay forma de acceder al archivo al que se va a escribir, de lo contrario se podrá incurrir en un error de la aplicación. Para supervisar que no haya un fallo de escritura se debe usar un comando que permite el **manejo de errores o excepciones** llamado *try… catch*. Si el estudiante no está familiarizado con las estructuras de **manejo de excepciones**, se recomienda remitirse a libros y textos técnicos que refieran su uso.

La estructura de clases entonces debe incluir la clase de datos en la capa de **persistencia**. La siguiente tabla presenta la lista de las clases que integran el proyecto, organizado por capas e indicando sus **responsabilidades**. Se recomienda como buena práctica tener muy clara esta característica particular por cada clase. Existe una técnica particular para presentar y listar las clases que pertenecen a un proyecto en la que se generan tarjetas descriptivas (llamadas **tarjetas CRC**) por cada clase del proyecto.

Tabla 8.2 Organización de la estructura de la aplicación en capas

Capa	Clase	Responsabilidad
Presentación	Presentación	Encargada del proceso de comunicación con el usuario. Cuenta con funcionalidades de solicitud y presentación de información.
Reglas de negocio	Departamento	Plantilla para crear un objeto de tipo *departamento*, que contenga los atributos y métodos requeridos.
Reglas de negocio	Main	Coordina las acciones de las otras clases
Persistencia de datos	Datos	Encargada de almacenar y recuperar la información desde un archivo plano

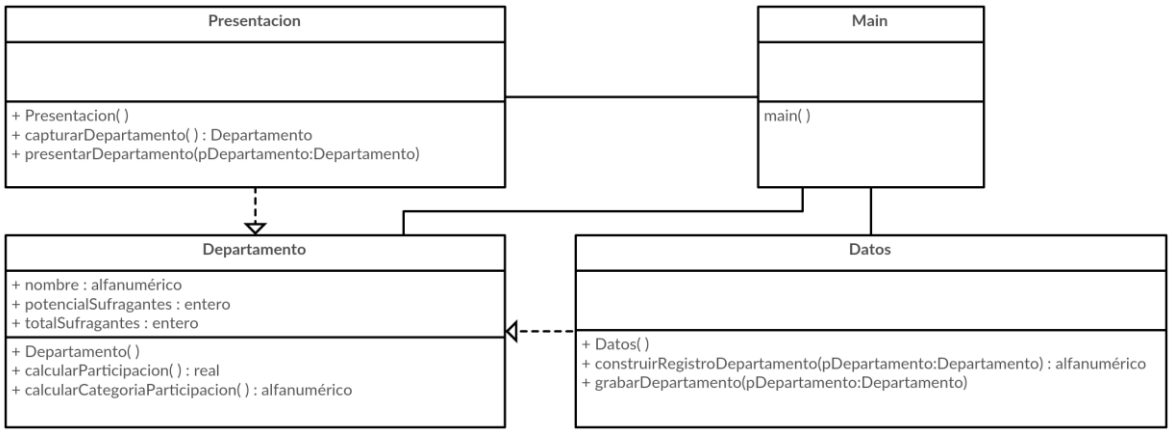


Figura 8.1 Diagrama de clases para el Prototipo 4

8.1.2.3 Diseño de algoritmos

8.1.2.3.1 Clase Departamento - Departamento( )

Como se vio en el capítulo anterior, la clase Departamento en su método constructor Departamento(), inicializa los atributos con valores neutros. Su algoritmo es secuencial. Ver Figura 7.4 Diagrama de flujo para el método Departamento() y Tabla 7.7 Descripción del algoritmo para el método constructor Departamento().

Lo mismo ocurre con los métodos calcularParticipación() y calcularCategoriaParticipacion( ) descritos anteriormente y que no sufren modificación alguna en el prototipo 4 del cual se ocupa el presente capítulo. Ver Figura 7.5 Diagrama de flujo para el método calcularParticipación y Figura 7.6 Diagrama de flujo para el método calcularCategoriaParticipacion() con sus respectivas descripciones en las Tablas 7.8 y 7.9.

8.1.2.3.2 Clase Presentacion - capturarDepartamento( ):Departamento

Para nuestro prototipo 4, la clase Presentación tampoco sufre modificaciones, de tal forma que sus algoritmos de implementación para los métodos: capturarDepartamento() y PresentarDepartamento() siguen quedando iguales a su definición y descripción anteriores. Ver Figuras 7.7 Diagrama de flujo para el método capturarDepartamento() y 7.8 Diagrama de flujo para el método presentarDepartamento()

8.1.2.3.3 Clase Datos - construirRegistroDepartamento(pDepartamento:Departamento): cadena

Descripción: Recibe un mensaje con un objeto de la clase Departamento como parámetro. Construye un registro, es decir una cadena de caracteres, a la que agrega cada uno de los valores de los atributos del objeto separados entre sí por un signo de coma. Por último, lo entrega el registro como respuesta al objeto que haya hecho la solicitud, es decir, que haya invocado ejecución del método.

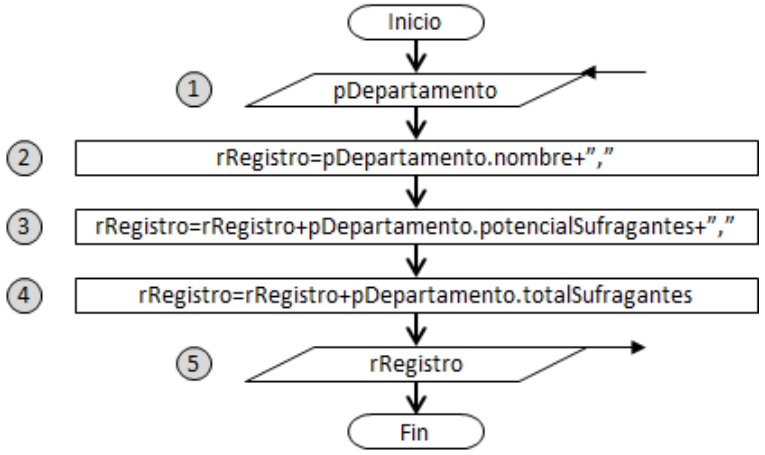


Figura 8.2 Diagrama de flujo para el método construirRegistroDepartamento ()

Tabla 8.3 Descripción del algoritmo para el método construirRegistroDepartamento()

1	Se requiere como dato de entrada el objeto de tipo Departamento. En este caso se recibe el objeto como mensaje.
2	Asigna el nombre del departamento del objeto en la variable rRegistro y concatena con una coma
3	Concatena en la variable rRegistro el potencial de sufragantes del departamento almacenado en el objeto y concatena con una coma
4	Concatena en la variable rRegistro el total de sufragantes del departamento almacenado en el objeto
5	Se genera como dato de salida el mensaje concatenado

8.1.2.3.4 Clase Datos - grabarDepartamento(pDepartamento:Departamento)

Descripción: Recibe un mensaje con un objeto de la clase Departamento como parámetro. Abre un archivo y a continuación invoca el método de construcción del registro, cuya respuesta es grabada en el archivo que finalmente es cerrado.

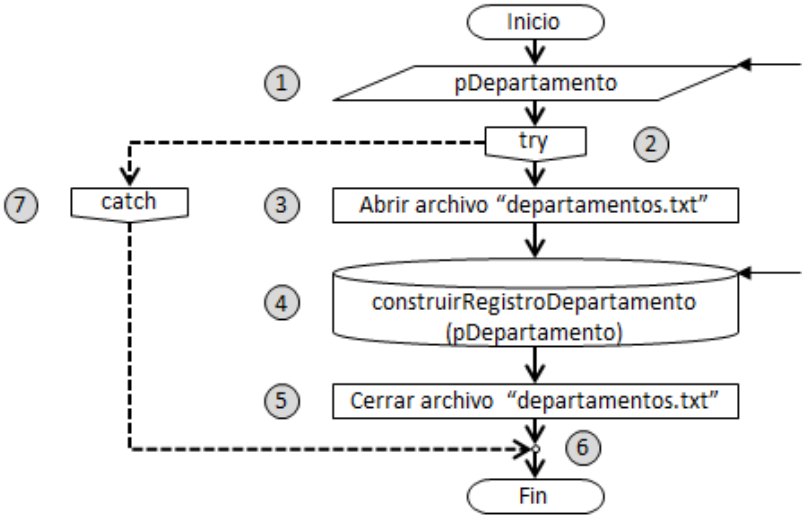


Figura 8.3 Diagrama de flujo para el método grabarDepartamento ()

Tabla 8.4 Descripción del algoritmo para el método grabarDepartamento ()

1	Se requiere como dato de entrada el objeto de tipo Departamento. En este caso se recibe el objeto como mensaje.
2	Inicia la estructura de manejo de errores en caso de poder abrir el archivo para su escritura
3	Abre el archivo de texto para escritura
4	Escribe en el archivo la respuesta del método construirRegistroDepartamento()
5	Cierra el archivo de texto para asegurar la escritura
6-7	Inicia la estructura para la ejecución de acciones en caso de fallo por no poder abrir el archivo para escritura

8.1.2.3.5 Clase Main – main( )

Descripción: Coordina las acciones de las demás clases. Solicita a un objeto de la clase Presentacion que lleve a cabo la captura de los datos de un objeto de la clase Departamento, cuyos datos almacena luego en el objeto departamento. A continuación, solicita al objeto de la clase Datos que grabe el objeto departamento. Por último, ordena al objeto de la clase Presentacion que lleve a cabo el método de presentación tomando como parámetro el objeto de la clase Departamento.

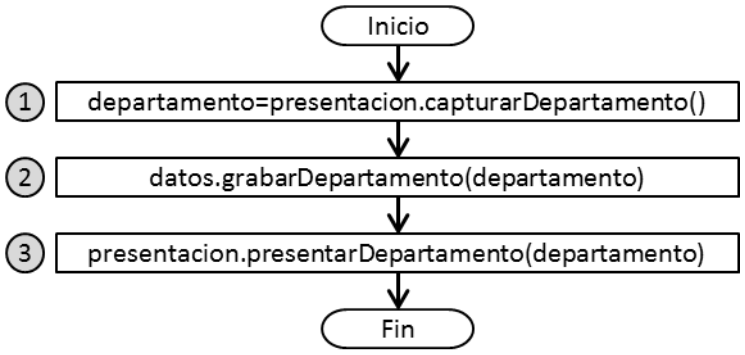


Figura 8.4 Diagrama de flujo para el método main()

Tabla 8.5 Descripción del algoritmo para el método main()

1	Ordena al objeto *presentacion* que ejecute su método *capturarDepartamento()* y entregue el resultado al objeto *departamento*.
2	Ordena al objeto datos que almacene la información del departamento capturado en un archivo plano.
3	Ordena al objeto *presentacion* que ejecute su método *presentarDepartamento()*, tomando como parámetro el objeto *departamento*.

8.1.3 Implementación

La estructura del proyecto cambia porque aparece una nueva clase llamada Datos que se encargará particularmente de organizar los datos del departamento en un registro con formato **csv** y de almacenar el registro **concatenado** en un archivo de texto. Por lo tanto, se agregará la nueva clase Datos en la estructura. Cabe recordar que todas las clases (excepto main) cuentan

con el método constructor para inicializar datos. En este caso, se incluirá el método constructor pero se dejará el método en blanco (al igual que el constructor de Presentación).

La codificación de la clase *Departamento* y la clase *Presentacion* no sufren modificación alguna, por lo cual no se presentan en este apartado. Ver Tablas 7.11 y 7.12 del capítulo anterior.

Tabla 8.6 Codificación de la clase Datos

```
1   package proyecto;
2   import java.io.*;
3
4   public class Datos {
5      public Datos(){
6
7      }
8
9      public String construirRegistroDepartamento(Departamento pDepartamento){
10        String rRegistro;
11        rRegistro=pDepartamento.nombre+",";
12        rRegistro=rRegistro+pDepartamento.potencialSufragantes+",";
13        rRegistro=rRegistro+pDepartamento.totalSufragantes;
14        return rRegistro;
15     }
16
17     public void grabarDepartamento(Departamento pDepartamento){
18        FileWriter fileWriter;
19        PrintWriter printWriter;
20        try{
21           fileWriter=new FileWriter("departamentos.txt",true);
22           printWriter=new PrintWriter(fileWriter);
23           printWriter.println(construirRegistroDepartamento(pDepartamento));
24           fileWriter.close();
25        }
26        catch(Exception e){
27
28        }
29     }
```

Tabla 8.7 Codificación de la clase Main

```
1   package proyecto;
2
3   public class Main {
4
5      public static void main(String[] args) {
6         Departamento departamento;
7         Presentacion presentacion;
8         Datos datos;
9         presentacion=new Presentacion();
10        datos=new Datos();
11        departamento=presentacion.capturarDepartamento();
```

```
12          datos.grabarDepartamento(departamento);
13          presentacion.presentarDepartamento(departamento);
14      }
15  }
```

8.2 Recuperación de datos – Prototipo 5

El prototipo 5 debe hacer lo mismo que el prototipo 4, además de permitir la **recuperación de los datos** de un departamento utilizando como criterio de búsqueda un nombre ingresado por el usuario.

8.2.1 Análisis

8.2.1.1 Alcance

Con el prototipo anterior, cada vez que un departamento se ingresa, su información ingresada se almacena en un archivo plano, ahora se busca recuperar esta información usando la aplicación. Para lograrlo, se debe primero indicar cuál es el registro que se desea buscar, y por esto se debe indicar el nombre del departamento a buscar.

8.2.1.2 Definiciones

Se mantienen las mismas definiciones en la lógica del negocio, no aparecen nuevas definiciones ni otros objetos de negocio. Ver Tabla 7.4 Definiciones aplicables en desarrollo de los prototipos del capítulo

8.2.1.3 Requisitos

Hay una adición en los requisitos funcionales, la cual es la **recuperación de datos** de un departamento a partir de lo almacenado en el archivo plano. Además, hay una modificación particular: la información del registro capturado no será mostrada inmediatamente se ingrese, sino que se solicitará el nombre del departamento a presentar.

8.2.1.3.1 Requisitos de interfaz de usuario

8.2.1.3.1.1 Ingreso de información en casillas de edición

El requisito de interfaz de usuario *Ingreso de información en casillas de edición* no sufre modificación en el prototipo 5.

Para la interacción con el usuario, la aplicación debe presentar ventanas con casillas de edición en donde pueda digitar los datos solicitados. Al solicitar los datos, el sistema deberá indicar en la misma ventana de diálogo cuál es el dato requerido.

8.2.1.3.2 Requisitos funcionales

A los Requisitos Funcionales: *Calcular el indicador de participación electoral de un departamento* y *Almacenar la información de un departamento en un archivo plano,* se agrega la funcionalidad de *Recuperar la información de un departamento en un archivo plano*

8.2.1.3.2.1 Recuperar la información de un departamento en un archivo plano

La aplicación debe poder recuperar desde un archivo plano los datos básicos ingresados de un departamento: el nombre, el potencial de sufragantes y el total de sufragantes, además de los datos calculados del indicador de participación electoral y la categoría de participación a la que pertenece, y se debe presentar al usuario. Para ello deberá preguntar por el nombre del departamento a buscar. Como restricción, se requiere el formato de almacenamiento sea en datos separados por punto y coma (formato CSV).

8.2.2 Diseño

8.2.2.1 Diseño de escenarios

No hay escenarios nuevos con respecto al primer prototipo. Se mantiene entonces el mismo diagrama de casos de uso, pero se espera un cambio en la interacción con el usuario: Luego de terminar de ingresar los datos de un nuevo departamento, se presentará un mensaje solicitando cuál es el registro por buscar. Ver Figura 7.1 Diagrama de casos de uso para los prototipos 02 y 03 y su descripción en la Tabla 7.5 del capítulo anterior.

8.2.2.2 Diseño de estructura

La clase que se encarga específicamente del proceso de **persistencia de los datos** cuenta actualmente con dos funciones, la de construir el registro del departamento en formato csv, y la de almacenar el registro en el archivo plano. Esto hace parte del proceso de **almacenamiento de datos**. Ahora se debe realizar el proceso inverso en dos funciones adicionales: leer el registro del archivo plano y a partir de la lectura de la línea, descomponer los datos y organizarlos en un objeto de tipo Departamento. Esto hace parte del proceso de **recuperación de datos**.

- La primera función es leer la información del archivo de texto. Al igual que el acceso al almacenamiento, el acceso a la lectura de un archivo tiene la misma particularidad: se debe identificar si hay forma de acceder al archivo del que se va a leer, de lo contrario se podrá incurrir en un error de la aplicación. Nuevamente se debe usar el comando try… catch para monitorear que no haya un fallo de lectura.
- La segunda función es tomar los datos que se hayan guardado del departamento y separarlos para armar cada dato individual en cada atributo de un objeto de tipo Departamento.

Hay un detalle adicional para tener en cuenta: si se ejecuta la aplicación varias veces, es posible que haya muchos registros de departamentos en el mismo archivo. De acuerdo con el requisito funcional, se debe obtener información del registro de departamento tomando como referencia de búsqueda su nombre. Por lo tanto, es necesario hacer una función adicional que, luego de leer

los registros de departamentos del archivo uno por uno y armarlo en un objeto, determine si ese registro es el que se estaba buscando para devolver el registro que corresponde. En resumen, se realizarán 3 nuevos métodos en la clase Datos.

Por otra parte, en el mismo requisito se indica que debe capturarse el dato del nombre de departamento a buscar en el archivo, para lo cual se debe implementar un nuevo método en la clase Presentacion que permita realizar esta pregunta.

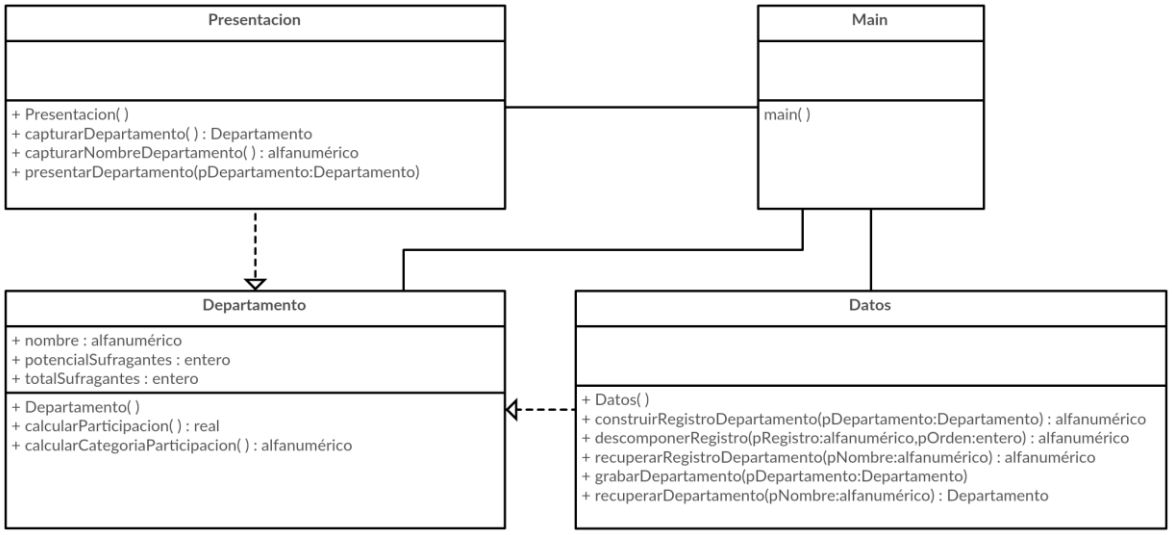


Figura 8.5 Diagrama de clases para el Prototipo 5

8.2.2.3 Diseño de algoritmos

8.2.2.3.1 Clase Departamento

La clase Departamento no tiene modificaciones en el prototipo 5, por lo que no se hará la descripción detallada de ésta. Ver secciones 7.1.2.3.1, 7.1.2.3.2 y 7.1.2.3.3 que definen los métodos correspondientes a esta clase.

8.2.2.3.2 Clase Presentacion - capturarDepartamento( ):Departamento

Descripción: Obtiene del usuario cada uno de los valores para los atributos, y entrega como respuesta el objeto de la clase Departamento con los valores de los atributos asignados.

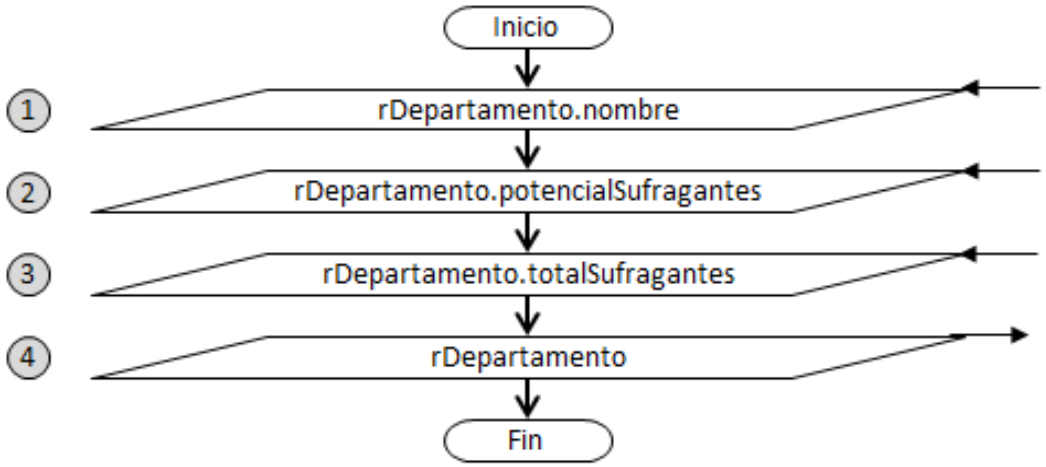


Figura 8.6 Diagrama de flujo para el método capturarDepartamento( )

8.2.2.3.3 Clase Presentacion - capturarNombreDepartamento( ):cadena

Descripción: Obtiene del usuario un nombre de departamento, y lo entrega como respuesta al objeto que haya hecho la solicitud, es decir, que haya enviado el mensaje para la ejecución del método.

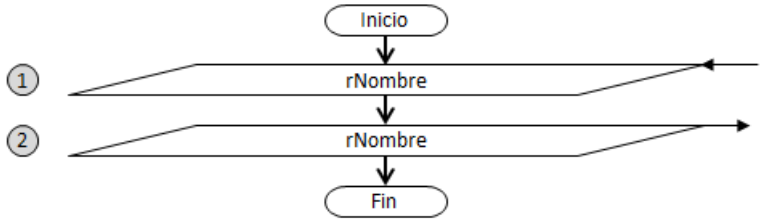


Figura 8.7 Diagrama de flujo para el método capturarNombreDepartamento ()

Tabla 8.8 Descripción del algoritmo para el método capturarNombreDepartamento ()

1	Se requiere como dato de entrada el nombre del departamento en la variable rNombre. Para esto, la aplicación debe presentar una pantalla con una caja de texto.
2	Se genera como dato de salida la variable capturada *rNombre*. Se realiza como respuesta de este método.

8.2.2.3.4 Clase Presentacion - presentarDepartamento(pDepartamento:Departamento)

Descripción: Recibe un objeto de la clase Departamento como parámetro. Construye un mensaje al que agrega cada uno de los valores de los atributos del objeto recibido, así como el resultado del método de cálculo del indicador de participación electoral y el resultado del método de cálculo de la categoría de participación electoral. Por último presenta el mensaje construido al usuario.

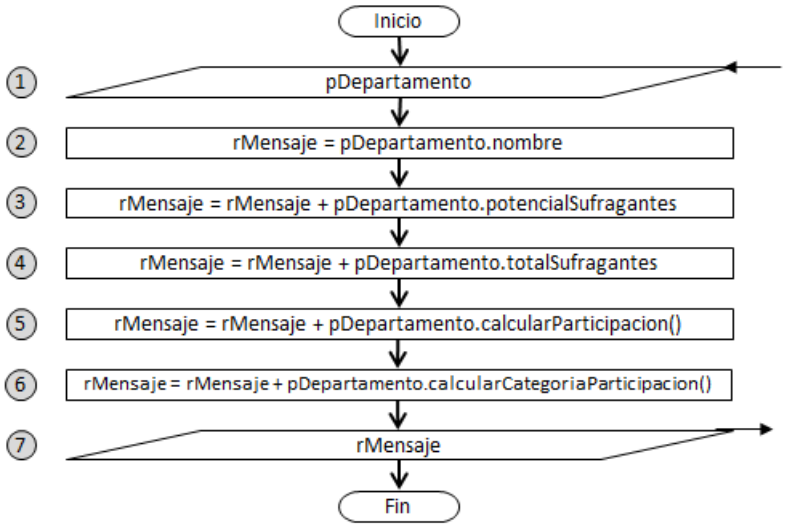


Figura 8.8 Diagrama de flujo para el método presentarDepartamento()

8.2.2.3.5 Clase Datos - construirRegistroDepartamento(pDepartamento:Departamento): cadena

Descripción: Recibe un mensaje con un objeto de la clase Departamento como parámetro. Construye un registro, es decir una cadena de caracteres, a la que agrega cada uno de los valores de los atributos del objeto separados entre sí por un signo de coma. Por último lo entrega el registro como respuesta al objeto que haya hecho la solicitud, es decir, que haya invocado ejecución del método.

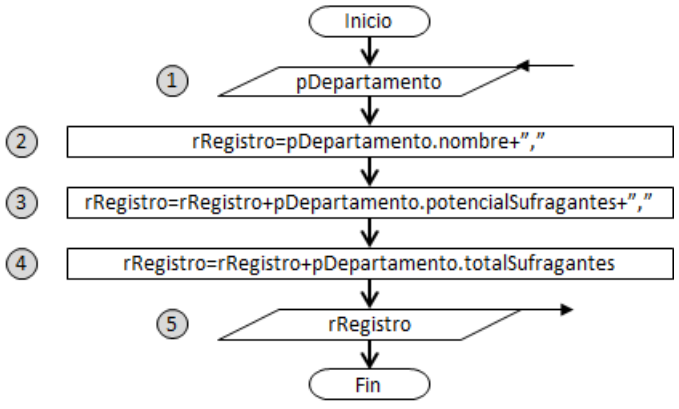


Figura 8.9 Diagrama de flujo para el método construirRegistroDepartamento ()

8.2.2.3.6 Clase Datos - descomponerRegistro (pRegistro:cadena, pOrden: entero):cadena

Descripción: Recibe un mensaje con dos parámetros; el primero de estos es un registro o cadena de caracteres que contiene valores separados entre sí por signos de coma; el segundo, un

número ordinal que indica el dato que se debe extraer del registro y entregar como respuesta final. El método hace un recorrido secuencial por los caracteres del registro, y cuando ubica el comienzo del dato solicitado, comienza a agregar cada caracter en una variable que al final entrega como respuesta al objeto que haya requerido la ejecución del método.

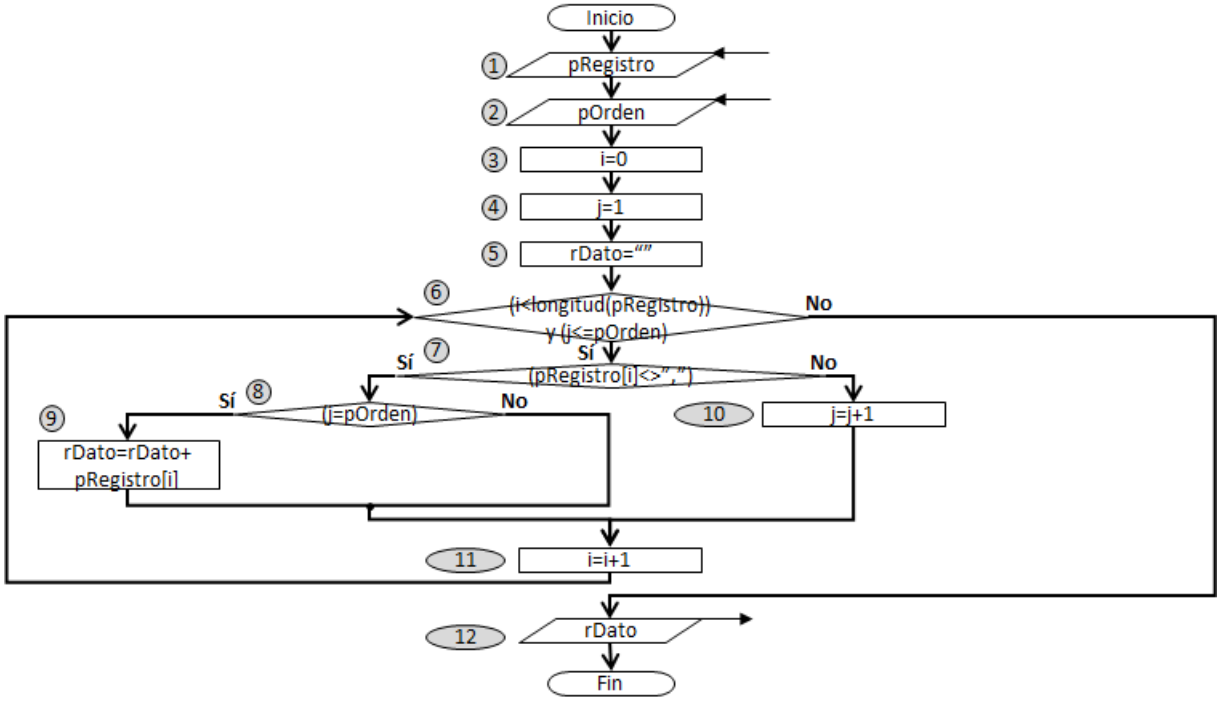


Figura 8.10 Diagrama de flujo para el método descomponerRegistro()

Tabla 8.9 Descripción del algoritmo para el método descomponerRegistro()

1	Se requiere como dato de entrada la variable pRegistro. En este caso se recibe la variable como mensaje.
2	Se requiere como dato de entrada la variable pOrden. En este caso se recibe la variable como mensaje.
3	Asigna a una variable i el valor 0, es decir la inicializa con un valor.
4	Inicializa una variable j con el valor 1
5	Inicializa una variable rDato con el valor ""
6	Mientras el valor de la variable i sea menor a la longitud de la cadena ingresada en la variable pRegistro y la variable j sea menor o igual al valor de la variable pOrden, se realizará lo siguiente:
7	Se revisa de la variable pRegistro el carácter que está en la posición i. Si el carácter es diferente a "," entonces:
8	Se revisa que la variable j sea igual al valor de pOrden. Si es igual, entonces:
9	Concatena el carácter leído de pRegistro en la variable rDato
10	Si el carácter leído es igual a "," entonces:
11	Incrementa en 1 el valor de la variable i
12	Se cierra el ciclo y se genera como dato de salida la variable concatenada *rDato*. Para ello se realiza como respuesta de este método.

8.2.2.3.7 Clase Datos - grabarDepartamento(pDepartamento:Departamento

Descripción: Recibe un mensaje con un objeto de la clase Departamento como parámetro. Abre un archivo y a continuación invoca el método de construcción del registro, cuya respuesta es grabada en el archivo que finalmente es cerrado.

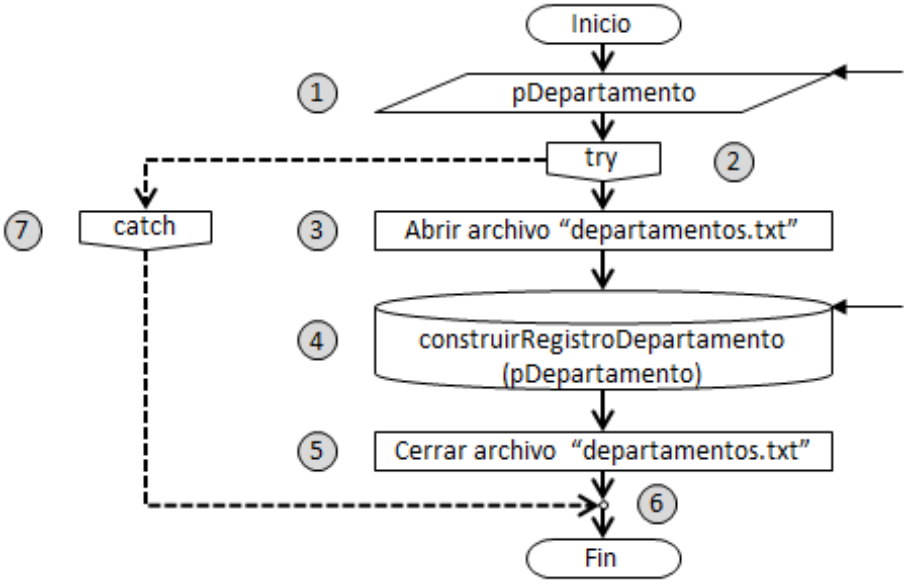


Figura 8.11 Diagrama de flujo para el método grabarDepartamento ()

8.2.2.3.8 Clase Datos - recuperarRegistroDepartamento(pNombre:cadena):cadena

Descripción: Recibe un mensaje con una cadena de caracteres como parámetro, que contiene un nombre de departamento a localizar en el archivo. Abre un archivo para lectura y a continuación lee uno a uno los registros hasta encontrar el que contiene los datos del departamento requerido, para lo cual solicita la descomposición del registro y la obtención del primer dato que corresponde al nombre. Por último, entrega como respuesta el registro que contenga el nombre buscado.

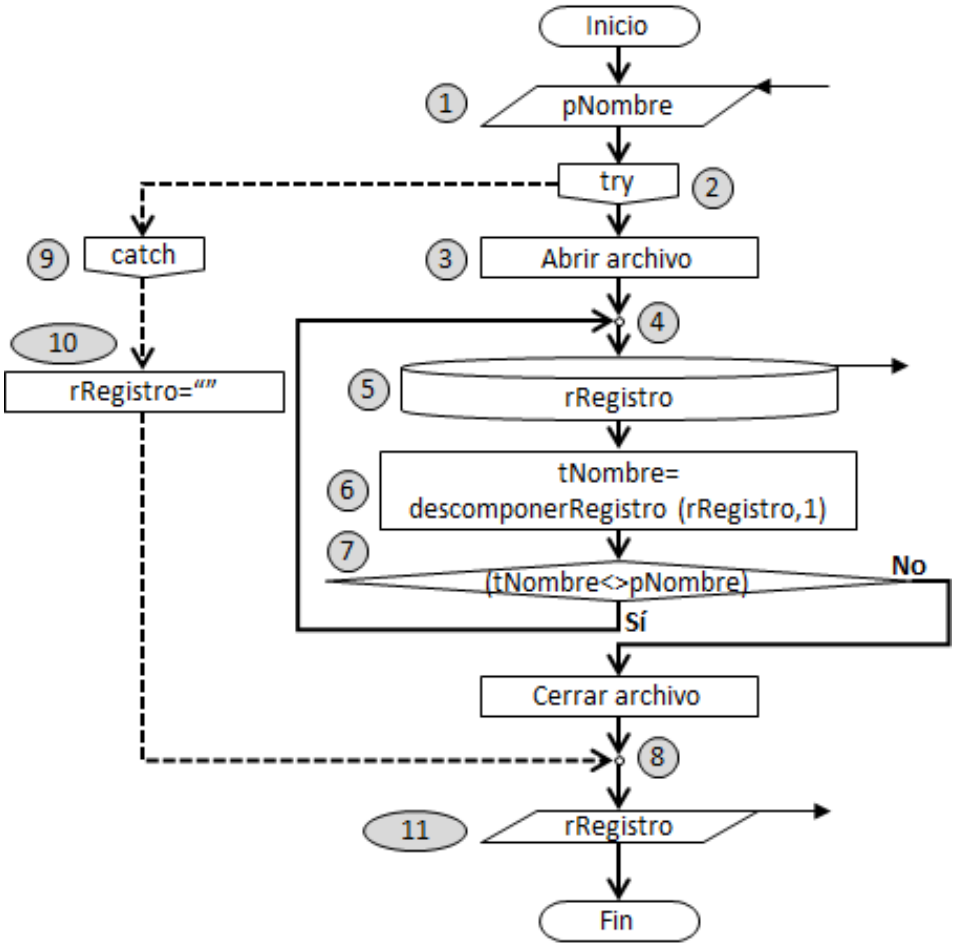


Figura 8.12 Diagrama de flujo para el método recuperarRegistroDepartamento()

Tabla 8.10 Descripción del algoritmo para el método recuperarRegistroDepartamento()

1	Se requiere como dato de entrada la variable pNombre. En este caso se recibe la variable como mensaje.
2	Inicia la estructura de manejo de errores en caso de poder abrir el archivo para su lectura
3	Abre el archivo de texto para lectura
4	Inicia el ciclo de lectura para cada una de las líneas de texto del archivo, condicionada al final del ciclo
5	Lee desde el archivo una línea de texto del archivo y la almacena en la variable rRegistro
6	Almacena en la variable tNombre la respuesta del método descomponerRegistro() del objeto, envíandole como parámetros la línea de texto leída y el valor 1 para extraer el dato del nombre del departamento desde la línea de texto.
7	Si el valor de la variable tNombre es diferente al parámetro recibido pNombre entonces debe repetir el ciclo, de lo contrario se detiene el ciclo.
8	Cierra el archivo de texto para lectura
9	Inicia la estructura para la ejecución de acciones en caso de fallo por no poder abrir el archivo para lectura
10	Asigna a la variable rRegistro el valor “”
11	Cerrada la estructura de acciones de fallo, se genera como dato de salida la variable rRegistro. Para ello se realiza como respuesta de este método.

8.2.2.3.9 Clase Datos - recuperarDepartamento(pNombre:cadena):Departamento

Descripción: Recibe un mensaje con una cadena de caracteres como parámetro, que contiene un nombre de departamento a localizar en el archivo. Solicita la ejecución del método de recuperación de registro mediante el nombre dado, y una vez obtenida la respuesta de aquel método, toma el registro hallado y solicita su descomposición en tres partes. Por último entrega como respuesta el objeto de la clase Departamento correspondiente al nombre dado como criterio de búsqueda.

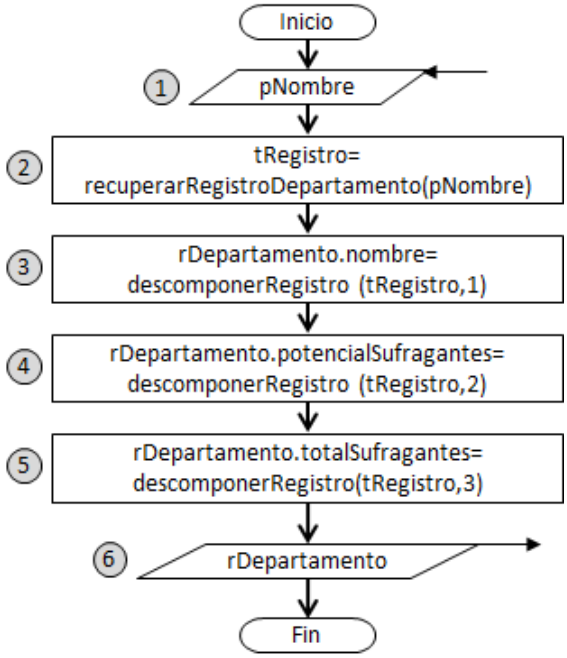


Figura 8.13 Diagrama de flujo para el método recuperarDepartamento()

Tabla 8.11 Descripción del algoritmo para el método recuperarDepartamento()

1	Se requiere como dato de entrada la variable pNombre. En este caso se recibe la variable como mensaje.
2	Almacena en la variable tRegistro la respuesta del método recuperarDepartamento () del objeto, envíandole como parámetro la variable pNombre
3	Se crea un objeto nuevo de tipo Departamento y se almacena en su atributo nombre la respuesta del método descomponerRegistro() del objeto, envíandole como parámetros el valor tRegistro y el valor 1 para extraer el dato del nombre del departamento.
4	Se almacena en el atributo potencialSufragantes del objeto de tipo Departamento la respuesta del método descomponerRegistro() del objeto, envíandole como parámetros el valor tRegistro y el valor 2 para extraer el dato del potencial de sufragantes del departamento.
5	Se almacena en el atributo totalSufragantes del objeto de tipo Departamento la respuesta del método descomponerRegistro() del objeto, envíandole como parámetros el valor tRegistro y el valor 3 para extraer el dato del total de sufragantes del departamento.
6	Se genera como dato de salida el objeto creado de tipo Departamento con todos sus atributos llenos.

8.2.2.3.10 Clase Main – main( )

Descripción: Coordina las acciones de las demás clases. Solicita a un objeto de la clase Presentacion que lleve a cabo la captura de un objeto de la clase Departamento, que almacena luego en el objeto departamento. A continuación, solicita al objeto de la clase Datos que grabe el objeto departamento. Enseguida solicita al objeto datos la recuperación de un departamento mediante un nombre capturado a su turno por el objeto presentación; el resultado de la operación de recuperación se almacena en el objeto departamento. Por último, ordena al objeto de la clase Presentacion que lleve a cabo el método de presentación tomando como parámetro el objeto de la clase Departamento.

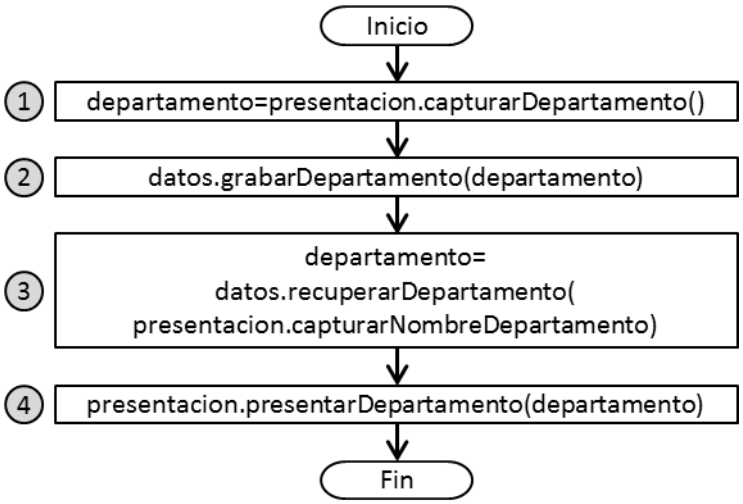


Figura 8.14 Diagrama de flujo para el método main()

Tabla 8.12 Descripción del algoritmo para el método main()

1	Ordena al objeto *presentacion* que ejecute su método *capturarDepartamento()* y entregue el resultado al objeto *departamento*.
2	Ordena al objeto datos que almacene la información del departamento capturado en un archivo plano.
3	Ordena al objeto *presentacion* que ejecute su método *capturarNombreDepartamento()*, y luego el resultado se toma como parámetro para ordenar al objeto *datos* que ejecute su método *recuperarDepartamento()*. El resultado de este método se almacena en el objeto departamento
4	Ordena al objeto *presentacion* que ejecute su método *presentarDepartamento()*, tomando como parámetro el objeto *departamento*.

8.2.3 Implementación

La estructura del proyecto no cambia. Se incluirán los 3 métodos adicionales de la clase Datos y el método de la clase Presentacion.

Tabla 8.13 Codificación de la clase Departamento

```
1  package proyecto;
2
3  public class Departamento {
4     public String nombre;
5     public int potencialSufragantes;
6     public int totalSufragantes;
7
8     public Departamento(){
9        nombre="";
10       potencialSufragantes=0;
11       totalSufragantes=0;
12    }
13
14    public double calcularParticipacion(){
15       double rParticipacion;
16       rParticipacion=((double)totalSufragantes/potencialSufragantes)*100;
17       return rParticipacion;
18    }
19
20    public String calcularCategoriaParticipacion(){
21       double tParticipacion;
22       String rCategoria;
23       tParticipacion=calcularParticipacion();
24       if (tParticipacion>=50){
25          rCategoria="Alta";
26       }
27       else{
28          if (tParticipacion>=40){
29             rCategoria="Media";
30          }
31          else{
32             rCategoria="Baja";
33          }
34       }
35       return rCategoria;
36    }
37 }
```

Tabla 8.14 Codificación de la clase Presentacion

```
1  package proyecto;
2  import javax.swing.JOptionPane;
3
4  public class Presentacion {
5     public Presentacion(){
6
7     }
```

```
8
9     public Departamento capturarDepartamento(){
10        Departamento rDepartamento;
11        rDepartamento=new Departamento();
12        rDepartamento.nombre=JOptionPane.showInputDialog("Nombre: ");
13        rDepartamento.potencialSufragantes=Integer.parseInt(JOptionPane.showInputDialog("Potencial de
      sufragantes: "));
14        rDepartamento.totalSufragantes=Integer.parseInt(JOptionPane.showInputDialog("Total de
      sufragantes: "));
15        return rDepartamento;
16    }
17
18    public String capturarNombreDepartamento(){
19        String rNombre;
20        rNombre=JOptionPane.showInputDialog("Departamento: ");
21        return rNombre;
22    }
23
24    public void presentarDepartamento(Departamento pDepartamento){
25        String rMensaje;
26        rMensaje="Departamento: "+pDepartamento.nombre;
27        rMensaje=rMensaje+"\nPotencial de sufragantes: "+pDepartamento.potencialSufragantes;
28        rMensaje=rMensaje+"\nTotal de sufragantes: "+pDepartamento.totalSufragantes;
29        rMensaje=rMensaje+"\nIndicador de participación: "+pDepartamento.calcularParticipacion()+" %";
30        rMensaje=rMensaje+"\nCategoría según participación:
      "+pDepartamento.calcularCategoriaParticipacion();
31        JOptionPane.showMessageDialog(null, rMensaje);
32    }
33 }
```

Tabla 8.15 Codificación de la clase Datos

```
1  package proyecto;
2
3  import java.io.*;
4
5  public class Datos {
6     public Datos(){
7
8     }
9
10    public String construirRegistroDepartamento(Departamento pDepartamento){
11       String rRegistro;
12       rRegistro=pDepartamento.nombre+",";
13       rRegistro=rRegistro+pDepartamento.potencialSufragantes+",";
14       rRegistro=rRegistro+pDepartamento.totalSufragantes;
15       return rRegistro;
16    }
17
18    public String descomponerRegistro(String pRegistro, int pOrden){
19       int i,j;
```

```
20          String rDato;
21          i=0;
22          j=1;
23          rDato="";
24          while ((i<pRegistro.length()) && (j<=pOrden)) {
25             if (pRegistro.substring(i,i+1).compareTo(",")!=0){
26                if (j==pOrden){
27                   rDato=rDato+pRegistro.substring(i,i+1);
28                }
29             }
30             else{
31                j=j+1;
32             }
33             i=i+1;
34          }
35          return rDato;
36       }
37
38       public String recuperarRegistroDepartamento(String pNombre){
39          FileReader fileReader;
40          BufferedReader bufferedReader;
41          String rRegistro,tNombre;
42          try{
43             fileReader=new FileReader("departamentos.txt");
44             bufferedReader=new BufferedReader(fileReader);
45             do{
46                rRegistro=bufferedReader.readLine();
47                tNombre=descomponerRegistro(rRegistro, 1);
48             } while (tNombre.compareToIgnoreCase(pNombre)!=0);
49             fileReader.close();
50          }
51          catch (Exception e){
52             rRegistro="";
53          }
54          return rRegistro;
55       }
56
57       public Departamento recuperarDepartamento(String pNombre){
58          String tRegistro;
59          Departamento rDepartamento;
60          rDepartamento=new Departamento();
61          tRegistro=recuperarRegistroDepartamento(pNombre);
62          rDepartamento.nombre=descomponerRegistro(tRegistro,1);
63          rDepartamento.potencialSufragantes=Integer.parseInt(descomponerRegistro(tRegistro,2));
64          rDepartamento.totalSufragantes=Integer.parseInt(descomponerRegistro(tRegistro,3));
65          return rDepartamento;
66       }
67
68       public void grabarDepartamento(Departamento pDepartamento){
69          FileWriter fileWriter;
70          PrintWriter printWriter;
71          try{
72             fileWriter=new FileWriter("departamentos.txt",true);
```

```
73              printWriter=new PrintWriter(fileWriter);
74              printWriter.println(construirRegistroDepartamento(pDepartamento));
75              fileWriter.close();
76          }
77          catch(Exception e){
78
79          }
80      }
81  }
```

Tabla 8.16 Codificación de la clase Main

```
1   package proyecto;
2
3   public class Main {
4
5       public static void main(String[] args) {
6           Departamento departamento;
7           Presentacion presentacion;
8           Datos datos;
9           presentacion=new Presentacion();
10          datos=new Datos();
11          departamento=presentacion.capturarDepartamento();
12          datos.grabarDepartamento(departamento);
13          departamento=datos.recuperarDepartamento(presentacion.capturarNombreDepartamento());
14          presentacion.presentarDepartamento(departamento);
15      }
16  }
```

8.3 Navegación – Prototipo 6

8.3.1 Análisis

8.3.1.1 Alcance

Hasta el prototipo 5, la aplicación ha brindado soporte computacional para que mediante el potencial de sufragantes y el total de sufragantes se calcule el indicador de participación electoral y se determine la categoría de participación de un departamento.

Cada ventana de captura de datos muestra la orientación suficiente sobre el dato que solicita. En cuanto a la presentación de datos se ha mostrado en una sola ventana, con los rótulos necesarios para que el usuario entienda a qué corresponde cada uno. Adicionalmente se ha implementado persistencia en archivo secuencial, en formato de valores separados por comas, y se ha implementado una función de búsqueda de un registro de departamento particular.

Se busca ahora en el prototipo 6 mejorar la usabilidad de la aplicación, incluyendo una función de navegación entre funciones, es decir, debe permitir al usuario seleccionar de forma independiente si quiere registrar nuevos datos o consultar información registrada previamente.

8.3.1.2 Definiciones

Se mantienen las mismas definiciones en la lógica del negocio, no aparecen nuevas definiciones ni otros objetos de negocio.

Tabla 8.17 Definiciones aplicables en desarrollo del prototipo 06.

	Definición	Tipo de dato
Departamento	Cada una de las circunscripciones territoriales de primer nivel en que se divide el país para propósitos electorales	Objeto
Nombre	Identificación nominal del departamento	Cadena de caracteres
Potencial de sufragantes	Cantidad de personas habilitadas en un departamento para votar en un proceso electoral	Número entero
Total de sufragantes	Cantidad de personas que acudieron a votar en un departamento en un proceso electoral	Número entero
Indicador de participación electoral	Indicador consistente en el total de sufragantes expresado como porcentaje del potencial de sufragantes $Indicador\ de\ participación\ electoral = \frac{Total\ de\ sufragantes}{Potencial\ de\ sufragantes} * 100$	Número real
Categoría de participación electoral	Categoría a la que pertenece el departamento, dependiendo del porcentaje de participación electoral: Si el porcentaje es mayor o igual a 50%: alto Si el porcentaje es menor a 50% pero mayor o igual a 40%: medio Si el porcentaje es menor a 40%: bajo	Cadena de caracteres

8.3.1.3 Requisitos

Hay una adición en los requisitos de interfaz, la cual es la selección de las funcionalidades, el ingreso de datos de departamentos y la recuperación de datos de datos de un departamento a partir de lo almacenado en el archivo plano.

8.3.1.3.1 Requisitos de interfaz de usuario

8.3.1.3.1.1 Ingreso de información en casillas de edición

Para la interacción con el usuario, la aplicación debe presentar ventanas con casillas de edición en donde pueda digitar los datos solicitados. Al solicitar los datos, el sistema deberá indicar en la misma ventana de diálogo cuál es el dato requerido.

8.3.1.3.1.2 Presentación de menú de selección de funciones

La aplicación debe presentar un menú para seleccionar una de 3 funciones: registrar la información de un departamento para su almacenamiento, consultar información de un departamento a partir de la recuperación de sus datos almacenados, o finalizar la aplicación.

8.3.1.3.2 Requisitos funcionales

8.3.1.3.2.1 Calcular el indicador de participación electoral de un departamento

La aplicación debe permitir al usuario introducir los siguientes datos básicos de un departamento: el nombre, el potencial de sufragantes y el total de sufragantes, y con estos datos debe calcular el indicador de participación electoral.

8.3.1.3.2.2 Almacenar la información de un departamento en un archivo plano

La aplicación debe poder almacenar en un archivo plano los datos básicos ingresados de un departamento: el nombre, el potencial de sufragantes y el total de sufragantes. Se requiere que el formato de almacenamiento sea en datos separados por punto y coma (formato CSV).

8.3.1.3.2.3 Recuperar la información de un departamento en un archivo plano

La aplicación debe poder recuperar desde un archivo plano los datos básicos ingresados de un departamento: el nombre, el potencial de sufragantes y el total de sufragantes, además de los datos calculados del indicador de participación electoral y la categoría de participación a la que pertenece, y se debe presentar al usuario. Para ello deberá preguntar por el nombre del departamento a buscar. Como restricción, se requiere el formato de almacenamiento sea en datos separados por punto y coma (formato CSV).

8.3.2 Diseño

8.3.2.1 Diseño de escenarios

Se presenta una división en el escenario con respecto al prototipo 5. El usuario debe poder seleccionar una de las 3 opciones en el menú. Dado que la opción de finalizar (salir) de la aplicación solo implica terminar la ejecución del programa, se presentará en el diagrama de casos de uso las dos opciones principales con las que interactuará el usuario.

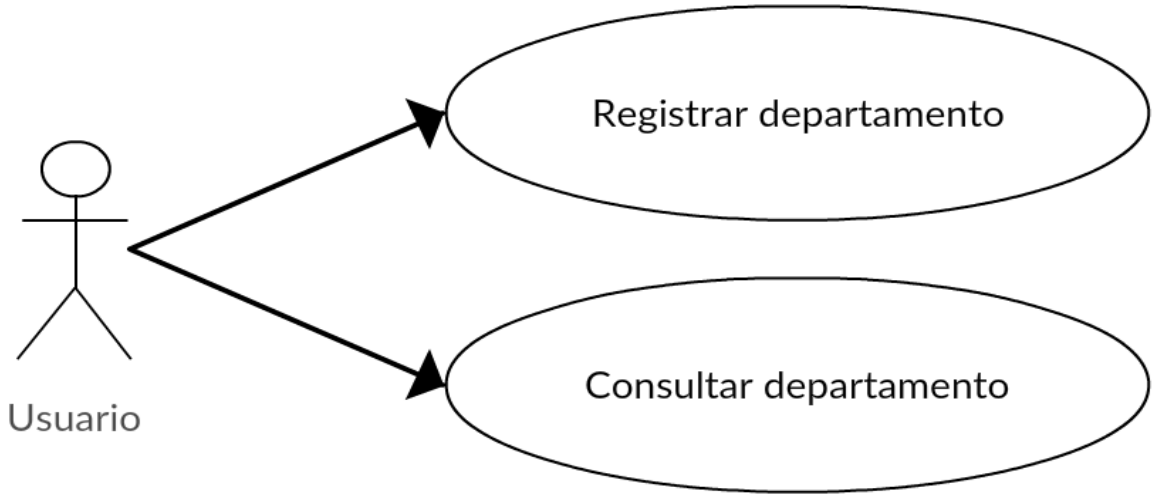


Figura 8.15 Diagrama de casos de uso para el Prototipo 5

Cada caso de uso debe contar con su propia especificación, por ello se generará la división de las acciones que se esperarán en cada una de ellas:

Tabla 8.18 Especificación del caso de uso Registrar departamento.

1	El usuario ingresa a la aplicación.
2	La aplicación presenta el menú de selección, por lo que el usuario selecciona la opción 1: Registrar Departamento
2.1	Si el usuario selecciona una opción mayor que 2, la aplicación finaliza.
3	La aplicación presenta una pantalla con una caja de texto para que el usuario digite el nombre del departamento
4	El usuario digita el nombre del departamento y pulsa el botón *Aceptar*
5	La aplicación presenta una pantalla con una caja de texto para que el usuario digite el potencial de sufragantes del departamento
6	El usuario digita el potencial de sufragantes del departamento y pulsa el botón *Aceptar*
7	La aplicación presenta una pantalla con una caja de texto para que el usuario digite el total de sufragantes del departamento
8	El usuario digita el total de sufragantes del departamento y pulsa el botón *Aceptar*
9	La aplicación muestra en pantalla el nombre del departamento, el potencial de sufragantes, el total de sufragantes, el indicador de participación electoral del departamento y la categoría de participación a la que pertenece el departamento que acabó de ingresar.
10	La aplicación vuelve a presentar el menú de selección (paso 2)

Tabla 8.19 Especificación del caso de uso Consultar departamento.

1	El usuario ingresa a la aplicación.
2	La aplicación presenta el menú de selección, por lo que el usuario selecciona la opción 2: Consultar Departamento
2.1	Si el usuario selecciona una opción mayor que 2, la aplicación finaliza.
3	El usuario digita el nombre del departamento a buscar y pulsa el botón *Aceptar*
4	La aplicación muestra en pantalla el nombre del departamento buscado, el potencial de sufragantes, el total de sufragantes, el indicador de participación electoral del departamento y la categoría de participación a la que pertenece.
5	La aplicación vuelve a presentar el menú de selección (paso 2)

Cabe aclarar en este punto que estas dos especificaciones de casos de uso presentan el **flujo normal de eventos** esperado, con el que se diseñará la aplicación para que el usuario se comporte tal cual se describe. No obstante, es posible que el usuario no desee, por ejemplo, tomar ninguna de las opciones, sino decida salir de la aplicación. Por esta razón aparece en cada especificación el paso 2.1. Estas alteraciones posibles en el comportamiento se deben explicar también en la especificación, presentándose como **flujos alternos**. Existen varias formas de presentar estos flujos alternos en una especificación de caso de uso, lo importante es dejar claro al programador la explicación del comportamiento esperado de la aplicación desde el punto de vista del usuario final.

Adicionalmente se podría pensar también en el comportamiento esperado si los datos ingresados no fueran correctos, con lo que aparecerían otros **flujos alternos**. Sin embargo, esta

explicación de eventos en el caso de errores se explicará más adelante, dado que el alcance para este no incluye el diseño de estos **flujos alternos**.

8.3.2.2 Diseño de estructura

En el diseño anterior se utilizó la clase Presentacion para realizar la captura de los datos de un departamento, así como generar la presentación de los datos de un departamento cualquiera. Ahora aparece un nuevo aspecto a programar que busca presentar un menú de opciones para seleccionar la funcionalidad requerida. Se sugiere crear una nueva clase perteneciente a la capa de presentación, que se llamará Menu y encargará de mostrar al usuario las opciones de menú para que pueda seleccionar. A partir de esta respuesta, la clase Main deberá redireccionar a la función que se desee ejecutar.

La estructura de clases entonces debe incluir la nueva clase de menú en la capa de presentación.

Tabla 8.20 Organización de la estructura de la aplicación en capas

Capa	Clase	Responsabilidad
Presentación	Presentación	Encargada del proceso de comunicación con el usuario. Cuenta con funcionalidades de solicitud y presentación de información.
Presentación	Menú	Encargada del proceso de comunicación con el usuario para permitir seleccionar una de las opciones de funcionalidad
Reglas de negocio	Departamento	Plantilla para crear un objeto de tipo *departamento*, que contenga los atributos y métodos requeridos.
Reglas de negocio	Main	Coordina las acciones de las otras clases
Persistencia de datos	Datos	Encargada de almacenar y recuperar la información desde un archivo plano

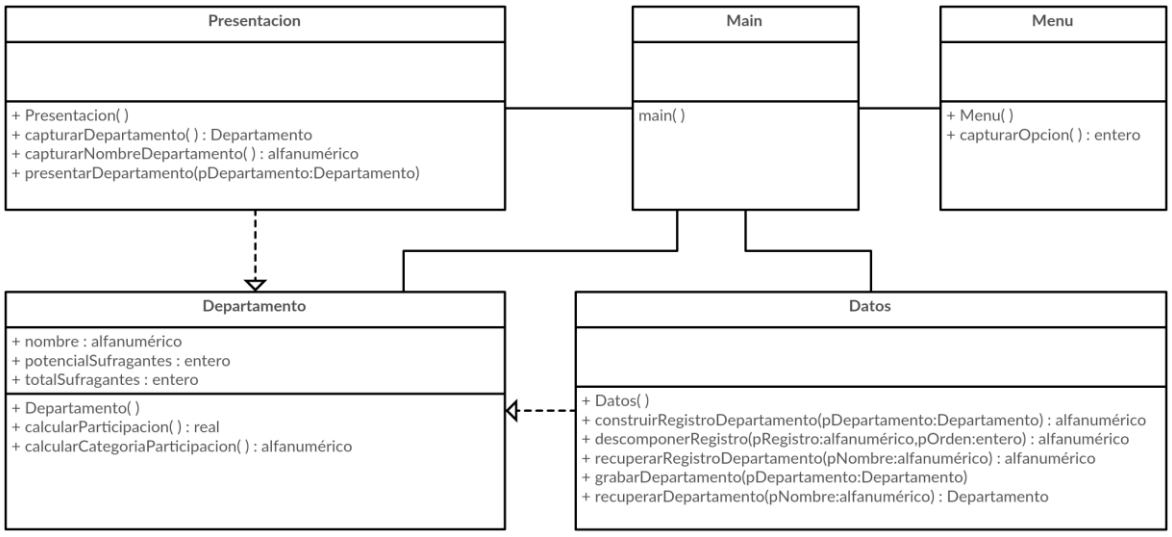


Figura 8.16 Diagrama de clases para el Prototipo 6

8.3.2.3 Diseño de algoritmos

8.3.2.3.1 Clase Departamento

La clase Departamento no sufre modificaciones en el nuevo prototipo, por lo cual mantiene su misma descripción. Ver Secciones 7.1.2.3.1, 7.1.2.3.2 y 7.1.2.3.3 del capítulo anterior.

8.3.2.3.2 Clase Presentacion - capturarDepartamento( ):Departamento

La clase Presentación mantiene la misma estructura que en el prototipo anterior (Prototipo 5), por lo cual mantiene su misma descripción. Ver Secciones 8.2.2.3.2, 8.2.2.3.3 y 8.2.2.3.4 del presente capítulo. Se procede entonces a describir la clase Menu.

8.3.2.3.3 Clase Menu - capturarOpcion(): entero

Descripción: Presenta al usuario un mensaje con las opciones disponibles en la aplicación, y obtiene de dicho usuario el número correspondiente a la opción seleccionada.

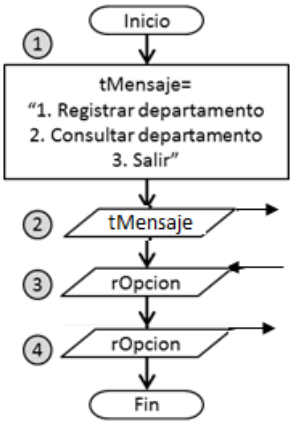


Figura 8.17 Diagrama de flujo para el método capturarOpcion()

Tabla 8.21 Descripción del algoritmo para el método capturarOpcion()

1	Almacena en una variable tMensaje el texto que contiene las opciones disponibles en forma de menú.
2	Presenta en pantalla el contenido de tMensaje
3	Solicita la opción para que la ingrese el usuario. La selección se almacena en la variable rOpcion
4	Se genera como dato de salida el valor de la variable rOpcion.

8.3.2.3.4 Clase Datos - construirRegistroDepartamento(pDepartamento:Departamento): cadena

Descripción: Recibe un mensaje con un objeto de la clase Departamento como parámetro. Construye un registro, es decir una cadena de caracteres, a la que agrega cada uno de los valores de los atributos del objeto separados entre sí por un signo de coma. Por último, lo entrega el

registro como respuesta al objeto que haya hecho la solicitud, es decir, que haya invocado ejecución del método.

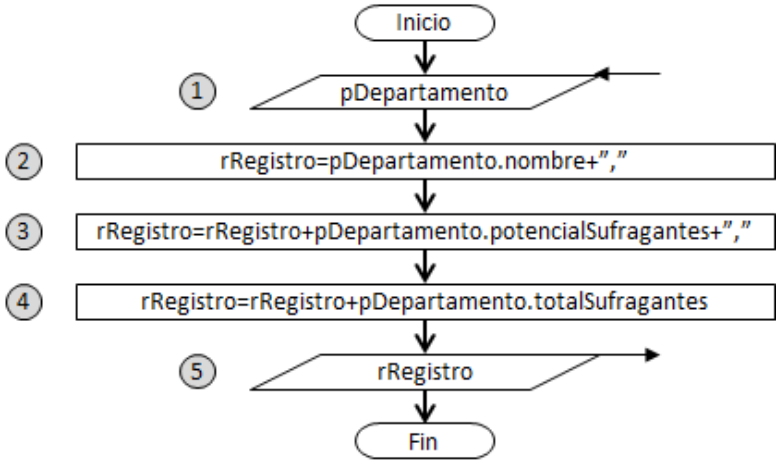


Figura 8.18 Diagrama de flujo para el método construirRegistroDepartamento ()

8.3.2.3.5 Clase Datos - descomponerRegistro (pRegistro:cadena, pOrden: entero):cadena

Descripción: Recibe un mensaje con dos parámetros; el primero de estos es un registro o cadena de caracteres que contiene valores separados entre sí por signos de coma; el segundo, un número ordinal que indica el dato que se debe extraer del registro y entregar como respuesta final. El método hace un recorrido secuencial por los caracteres del registro, y cuando ubica el comienzo del dato solicitado, comienza a agregar cada caracter en una variable que al final entrega como respuesta al objeto que haya requerido la ejecución del método.

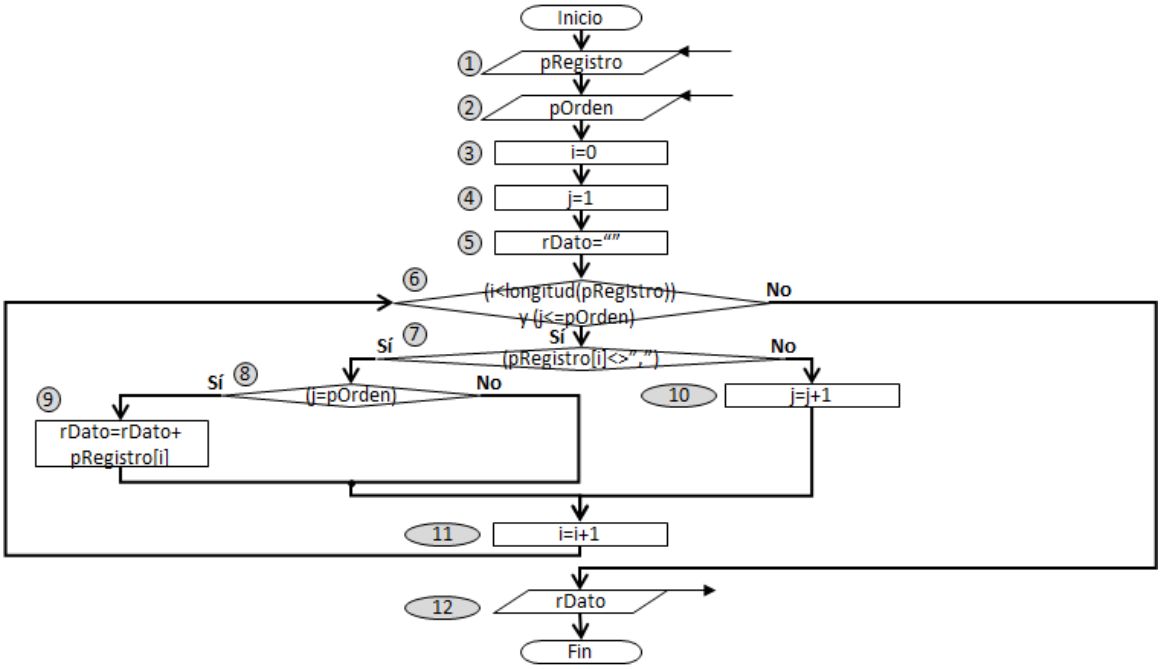


Figura 8.19 Diagrama de flujo para el método descomponerRegistro()

8.3.2.3.6 Clase Datos - grabarDepartamento(pDepartamento:Departamento

Descripción: Recibe un mensaje con un objeto de la clase Departamento como parámetro. Abre un archivo y a continuación invoca el método de construcción del registro, cuya respuesta es grabada en el archivo que finalmente es cerrado.

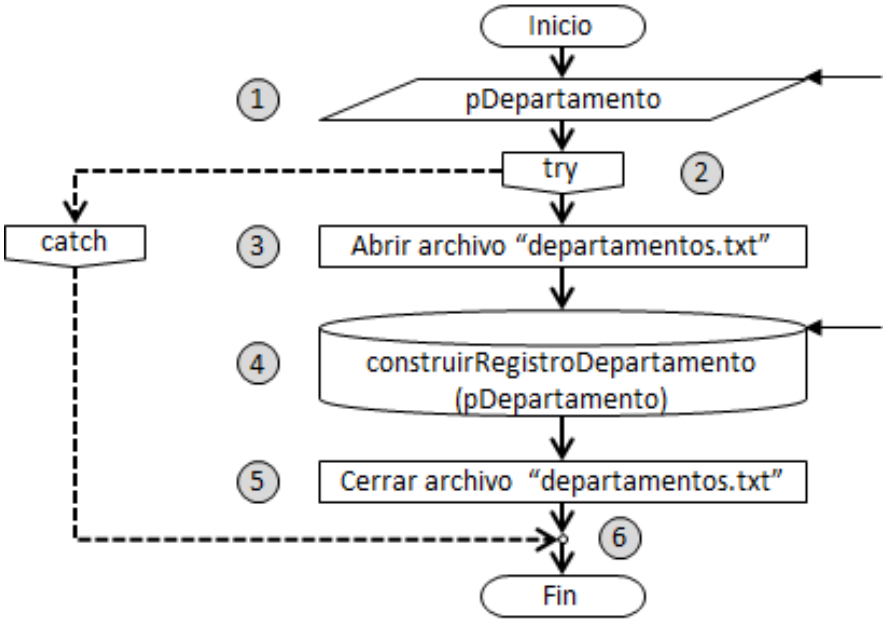


Figura 8.20 Diagrama de flujo para el método grabarDepartamento ()

8.3.2.3.7 Clase Datos - recuperarRegistroDepartamento(pNombre:cadena):cadena

Descripción: Recibe un mensaje con una cadena de caracteres como parámetro, que contiene un nombre de departamento a localizar en el archivo. Abre un archivo para lectura y a continuación lee uno a uno los registros hasta encontrar el que contiene los datos del departamento requerido, para lo cual solicita la descomposición del registro y la obtención del primer dato que corresponde al nombre. Por último, entrega como respuesta el registro que contenga el nombre buscado.

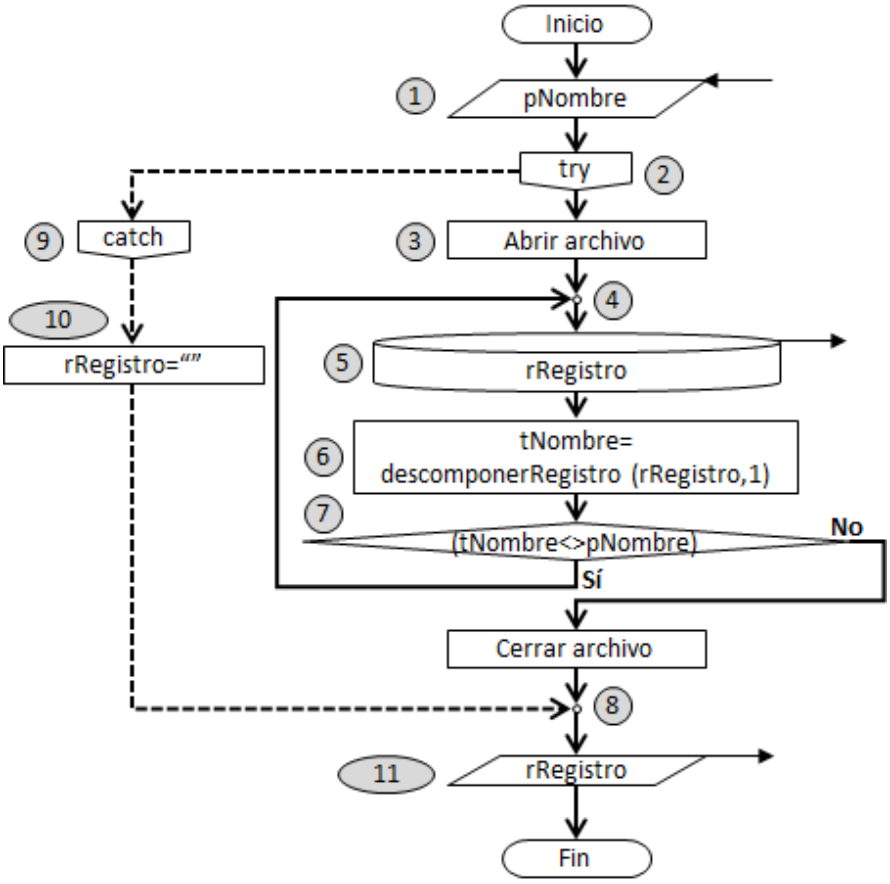


Figura 8.21 Diagrama de flujo para el método recuperarRegistroDepartamento()

8.3.2.3.8 Clase Datos - recuperarDepartamento(pNombre:cadena):Departamento

Descripción: Recibe un mensaje con una cadena de caracteres como parámetro, que contiene un nombre de departamento a localizar en el archivo. Solicita la ejecución del método de recuperación de registro mediante el nombre dado, y una vez obtenida la respuesta de aquel método, toma el registro hallado y solicita su descomposición en tres partes. Por último, entrega como respuesta el objeto de la clase Departamento correspondiente al nombre dado como criterio de búsqueda.

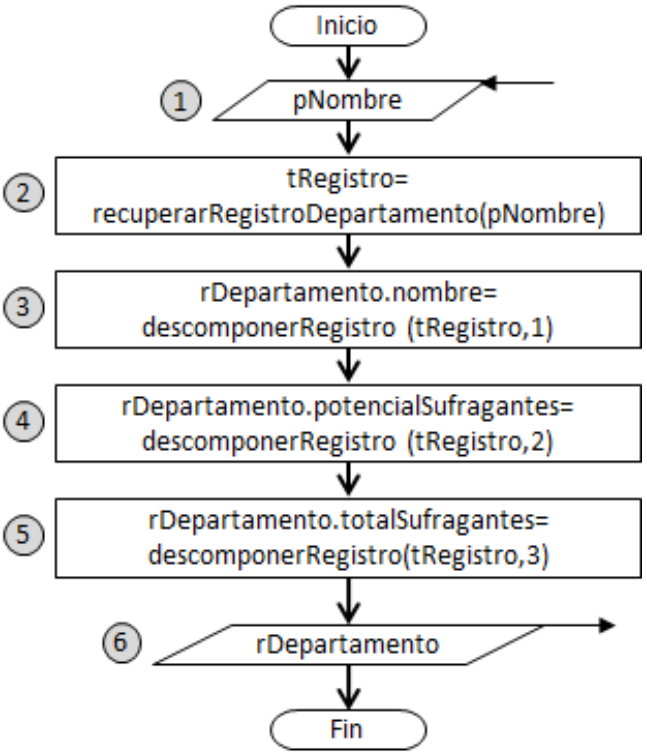


Figura 8.22 Diagrama de flujo para el método recuperarDepartamento()

8.3.2.3.9 Clase Main – main( )

Descripción: Coordina las acciones de las demás clases. Solicita a un objeto de la clase Menu que reciba de parte del usuario la opción requerida por él, que almacena en una variable.

Si la opción seleccionada es 1, coordina las clases para que el usuario pueda registrar un departamento, es decir, solicita a un objeto de la clase Presentacion que lleve a cabo la captura de un objeto de la clase Departamento, que almacena luego en el objeto departamento, solicita al objeto de la clase Datos que grabe el objeto departamento y finalmente muestra la información del departamento ingresado.

Si la opción seleccionada es 2, coordina las clases para que el usuario pueda consultar dando como parámetro un nombre de departamento, solicita al objeto datos la recuperación de un departamento mediante el nombre capturado por el objeto presentación; el resultado de la operación de recuperación se almacena en el objeto departamento. Por último, ordena al objeto de la clase Presentacion que lleve a cabo el método de presentación tomando como parámetro el objeto de la clase Departamento.

Todo esto se repite mientras la opción seleccionada sea menor o igual que 2.

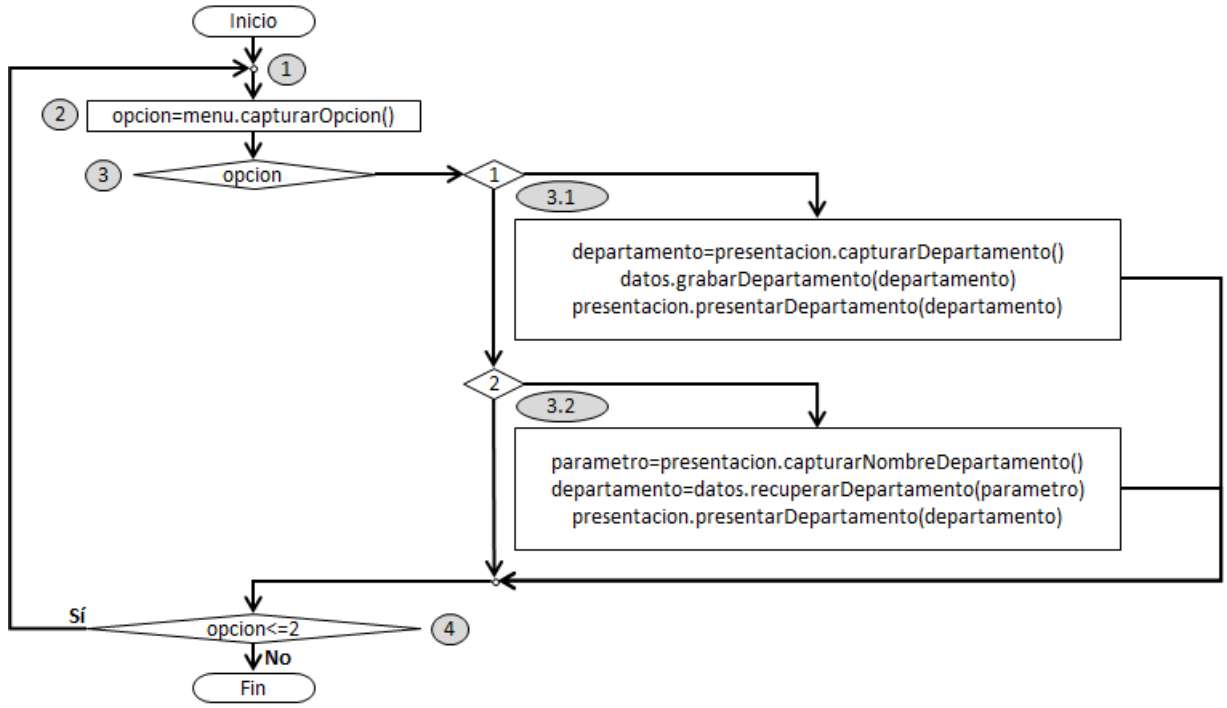


Figura 8.23 Diagrama de flujo para el método main()

Tabla 8.22 Descripción del algoritmo para el método main()

1	Inicia el ciclo de condición posterior
2	Ordena al objeto menú que ejecute su método *capturarOpcion()* y almacene el resultado en la variable opción
3	Se revisa la condición por cada caso dependiendo del valor de la variable opción:
3.1	En el caso que la variable sea 1: Ordena al objeto *presentacion* que ejecute su método *capturarDepartamento()* y entregue el resultado al objeto *departamento*. Ordena al objeto datos que almacene la información del departamento capturado en un archivo plano. Ordena al objeto *presentacion* que ejecute su método *presentarDepartamento()*, tomando como parámetro el objeto *departamento* que ha capturado.
3.2	En el caso que la variable sea 2: Ordena al objeto *presentacion* que ejecute su método *capturarNombreDepartamento()*, y luego el resultado se almacena en la variable parámetro. Se usa la variable parámetro y se ordena al objeto *datos* que ejecute su método *recuperarDepartamento()*. El resultado de este método se almacena en el objeto departamento. Ordena al objeto *presentacion* que ejecute su método *presentarDepartamento()*, tomando como parámetro el objeto *departamento*.
4	Se verifica la condición del ciclo, si la opción es menor o igual a 2 se devuelve al inicio del ciclo, de lo contrario se sale del ciclo, finalizando la ejecución del programa.

8.3.3 Implementación

La estructura del proyecto cambia porque hay una clase nueva llamada Menu. Adicionalmente se modificará el método main de la clase Main.

Tabla 8.23 Codificación de la clase Departamento

```
1  package proyecto;
2
3  public class Departamento {
4     public String nombre;
5     public int potencialSufragantes;
6     public int totalSufragantes;
7
8     public Departamento(){
9        nombre="";
10       potencialSufragantes=0;
11       totalSufragantes=0;
12    }
13
14    public double calcularParticipacion(){
15       double rParticipacion;
16       rParticipacion=((double)totalSufragantes/potencialSufragantes)*100;
17       return rParticipacion;
18    }
19
20    public String calcularCategoriaParticipacion(){
21       double tParticipacion;
22       String rCategoria;
23       tParticipacion=calcularParticipacion();
24       if (tParticipacion>=50){
25          rCategoria="Alta";
26       }
27       else{
28          if (tParticipacion>=40){
29             rCategoria="Media";
30          }
31          else{
32             rCategoria="Baja";
33          }
34       }
35       return rCategoria;
36    }
37 }
```

Tabla 8.24 Codificación de la clase Presentacion

```
1  package proyecto;
2  import javax.swing.JOptionPane;
3
4  public class Presentacion {
5     public Presentacion(){
6
7     }
8
9     public Departamento capturarDepartamento(){
```

```
10        Departamento rDepartamento;
11        rDepartamento=new Departamento();
12        rDepartamento.nombre=JOptionPane.showInputDialog("Nombre: ");
13        rDepartamento.potencialSufragantes=Integer.parseInt(JOptionPane.showInputDialog("Potencial de
      sufragantes: "));
14        rDepartamento.totalSufragantes=Integer.parseInt(JOptionPane.showInputDialog("Total de
      sufragantes: "));
15        return rDepartamento;
16     }
17
18     public String capturarNombreDepartamento(){
19        String rNombre;
20        rNombre=JOptionPane.showInputDialog("Departamento: ");
21        return rNombre;
22     }
23
24     public void presentarDepartamento(Departamento pDepartamento){
25        String rMensaje;
26        rMensaje="Departamento: "+pDepartamento.nombre;
27        rMensaje=rMensaje+"\nPotencial de sufragantes: "+pDepartamento.potencialSufragantes;
28        rMensaje=rMensaje+"\nTotal de sufragantes: "+pDepartamento.totalSufragantes;
29        rMensaje=rMensaje+"\nIndicador de participación: "+pDepartamento.calcularParticipacion()+" %";
30        rMensaje=rMensaje+"\nCategoría según participación:
      "+pDepartamento.calcularCategoriaParticipacion();
31        JOptionPane.showMessageDialog(null, rMensaje);
32     }
33  }
```

Tabla 8.25 Codificación de la clase Menu

```
1   package proyecto;
2   import javax.swing.JOptionPane;
3
4   public class Menu {
5      public Menu(){
6
7      }
8
9      public int capturarOpcion(){
10        String tMensaje;
11        int rOpcion;
12        tMensaje="1. Registrar departamento\n"+
13              "2. Consultar departamento\n"+
14              "3. Salir\n";
15        rOpcion=Integer.parseInt(JOptionPane.showInputDialog(tMensaje));
16        return rOpcion;
17     }
18  }
```

Tabla 8.26 Codificación de la clase Datos

```
1  package proyecto;
2  import java.io.*;
3
4  public class Datos {
5     public Datos(){
6
7     }
8
9     public String construirRegistroDepartamento(Departamento pDepartamento){
10       String rRegistro;
11       rRegistro=pDepartamento.nombre+",";
12       rRegistro=rRegistro+pDepartamento.potencialSufragantes+",";
13       rRegistro=rRegistro+pDepartamento.totalSufragantes;
14       return rRegistro;
15    }
16
17    public String descomponerRegistro(String pRegistro, int pOrden){
18       int i,j;
19       String rDato;
20       i=0;
21       j=1;
22       rDato="";
23       while ((i<pRegistro.length()) && (j<=pOrden)) {
24          if (pRegistro.substring(i,i+1).compareTo(",")!=0){
25             if (j==pOrden){
26                rDato=rDato+pRegistro.substring(i,i+1);
27             }
28          }
29          else{
30             j=j+1;
31          }
32          i=i+1;
33       }
34       return rDato;
35    }
36
37    public String recuperarRegistroDepartamento(String pNombre){
38       FileReader fileReader;
39       BufferedReader bufferedReader;
40       String rRegistro,tNombre;
41       try{
42          fileReader=new FileReader("departamentos.txt");
43          bufferedReader=new BufferedReader(fileReader);
44          do{
45             rRegistro=bufferedReader.readLine();
46             tNombre=descomponerRegistro(rRegistro, 1);
47          } while (tNombre.compareToIgnoreCase(pNombre)!=0);
48          fileReader.close();
49       }
50       catch (Exception e){
51          rRegistro="";
```

```
52        }
53        return rRegistro;
54     }
55
56     public Departamento recuperarDepartamento(String pNombre){
57        String tRegistro;
58        Departamento rDepartamento;
59        rDepartamento=new Departamento();
60        tRegistro= recuperarRegistroDepartamento(pNombre);
61        rDepartamento.nombre= descomponerRegistro(tRegistro,1);
62        rDepartamento.potencialSufragantes=Integer.parseInt(descomponerRegistro(tRegistro,2));
63        rDepartamento.totalSufragantes=Integer.parseInt(descomponerRegistro(tRegistro,3));
64        return rDepartamento;
65     }
66
67     public void grabarDepartamento(Departamento pDepartamento){
68        FileWriter fileWriter;
69        PrintWriter printWriter;
70        try{
71           fileWriter=new FileWriter("departamentos.txt",true);
72           printWriter=new PrintWriter(fileWriter);
73           printWriter.println(construirRegistroDepartamento(pDepartamento));
74           fileWriter.close();
75        }
76        catch(Exception e){
77
78        }
79     }
80  }
```

Tabla 8.27 Codificación de la clase Main

```
1   package proyecto;
2
3   public class Main {
4
5      public static void main(String[] args) {
6         Departamento departamento;
7         Presentacion presentacion;
8         Menu menu;
9         Datos datos;
10        int opcion;
11        String parametro;
12        presentacion=new Presentacion();
13        menu=new Menu();
14        datos=new Datos();
15        opcion=0;
16        do {
17           opcion=menu.capturarOpcion();
18           switch (opcion){
19              case 1:{
```

```
20                  departamento=presentacion.capturarDepartamento();
21                  datos.grabarDepartamento(departamento);
22                  presentacion.presentarDepartamento(departamento);
23                  break;
24              }
25              case 2:{
26                  parametro=presentacion.capturarNombreDepartamento();
27                  departamento=datos.recuperarDepartamento(parametro);
28                  presentacion.presentarDepartamento(departamento);
29                  break;
30              }
31          }
32      } while (opcion<=2);
33  }
34  }
```

8.4 Datos y cálculos adicionales – Prototipo 7

8.4.1 Análisis

8.4.1.1 Alcance

El prototipo 07 debe tener una funcionalidad similar a la del prototipo 06, pero permitiendo el registro de la cantidad de votos válidos en el departamento, así como el cálculo de la cantidad de votos no válidos y el indicador de error.

8.4.1.2 Definiciones

Aparecen dos nuevos conceptos en la lógica de negocio: el primero tiene que ver con los votos válidos y no válidos, valores que se aprecian en cualquier votación. Por definición un voto no válido es aquel que no indica de forma clara la voluntad del sufragante. Sin embargo, dado que se registra el total de sufragantes y la cantidad de votos válidos, en el software no se registra la cantidad de votos no válidos, sino que se calculan mediante la siguiente fórmula:

Votos no válidos = Total de sufragantes – Votos válidos.

El segundo tiene que ver con el indicador de error de votación, el cual es la razón de votos no válidos respecto al total de sufragantes. La fórmula para determinar esta razón es:

Indicador de error en la votación = Votos no válidos / Total de sufragantes

Tabla 8.28 Definiciones aplicables en desarrollo del prototipo 7.

	Definición	Tipo de dato
Departamento	Cada una de las circunscripciones territoriales de primer nivel en que se divide el país para propósitos electorales	Objeto
Nombre	Identificación nominal del departamento	Cadena de caracteres

Potencial de sufragantes	Cantidad de personas habilitadas en un departamento para votar en un proceso electoral	Número entero
Total de sufragantes	Cantidad de personas que acudieron a votar en un departamento en un proceso electoral	Número entero
Indicador de participación electoral	Indicador consistente en el total de sufragantes expresado como porcentaje del potencial de sufragantes $Indicador\ de\ participación\ electoral = \frac{Total\ de\ sufragantes}{Potencial\ de\ sufragantes} * 100$	Número real
Categoría de participación electoral	Categoría a la que pertenece el departamento, dependiendo del porcentaje de participación electoral: Si el porcentaje es mayor o igual a 50%: alto Si el porcentaje es menor a 50% pero mayor o igual a 40%: medio Si el porcentaje es menor a 40%: bajo	Cadena de caracteres
Votos válidos	Cantidad de votos que indican de forma clara la voluntad del sufragante	Número entero
Votos no válidos	Cantidad de votos que NO indican de forma clara la voluntad del sufragante. Se calcula respecto al total de sufragantes $Votos\ no\ válidos = Total\ de\ sufragantes - Votos\ válidos$	Número entero
Indicador de error en la votación	Indicador consistente en dividir el total de votos no válidos entre el total de sufragantes. $Indicador\ de\ error\ en\ la\ votación = \frac{Votos\ no\ válidos}{Total\ de\ sufragantes}$	Número real

8.4.1.3 Requisitos

Hay una modificación en los requisitos funcionales, la cual es el ingreso, cálculo y la presentación de más datos relacionados con el departamento. Como se ingresan datos nuevos, se deben almacenar estos datos también en el archivo y se deben poder recuperar también.

8.4.1.3.1 Requisitos de interfaz de usuario

8.4.1.3.1.1 Ingreso de información en casillas de edición

Para la interacción con el usuario, la aplicación debe presentar ventanas con casillas de edición en donde pueda digitar los datos solicitados. Al solicitar los datos, el sistema deberá indicar en la misma ventana de diálogo cuál es el dato requerido.

8.4.1.3.1.2 Presentación de menú de selección de funciones

La aplicación debe presentar un menú para seleccionar una de 3 funciones: registrar la información de un departamento para su almacenamiento, consultar información de in departamento a partir de la recuperación de sus datos almacenados, o finalizar la aplicación.

8.4.1.3.2 Requisitos funcionales

8.4.1.3.2.1 Calcular el indicador de participación electoral de un departamento

La aplicación debe permitir al usuario introducir los siguientes datos básicos de un departamento: el nombre, el potencial de sufragantes, el total de sufragantes y la cantidad de

votos válidos, y con estos datos debe calcular el indicador de participación electoral, los votos no válidos y el indicador de error de votación.

8.4.1.3.2.2 Almacenar la información de un departamento en un archivo plano

La aplicación debe poder almacenar en un archivo plano los datos básicos ingresados de un departamento: el nombre, el potencial de sufragantes, el total de sufragantes y el total de votos válidos. Se requiere que el formato de almacenamiento sea en datos separados por punto y coma (formato CSV).

8.4.1.3.2.3 Recuperar la información de un departamento en un archivo plano

La aplicación debe poder recuperar desde un archivo plano los datos básicos ingresados de un departamento: el nombre, el potencial de sufragantes, el total de sufragantes, el total de votos válidos además de los datos calculados del indicador de participación electoral, la categoría de participación a la que pertenece, los votos no válidos y el indicador de error de votación y se debe presentar al usuario. Para ello deberá preguntar por el nombre del departamento a buscar. Como restricción, se requiere el formato de almacenamiento sea en datos separados por punto y coma (formato CSV).

8.4.2 Diseño

8.4.2.1 Diseño de escenarios

Se mantiene el mismo diseño de escenarios respecto a las dos funciones básicas del programa accedidas a través del menú. No obstante, en el registro del departamento se debe agregar el dato adicional que se debe ingresar. Además, se debe mostrar el nuevo dato ingresado y los nuevos datos calculados.

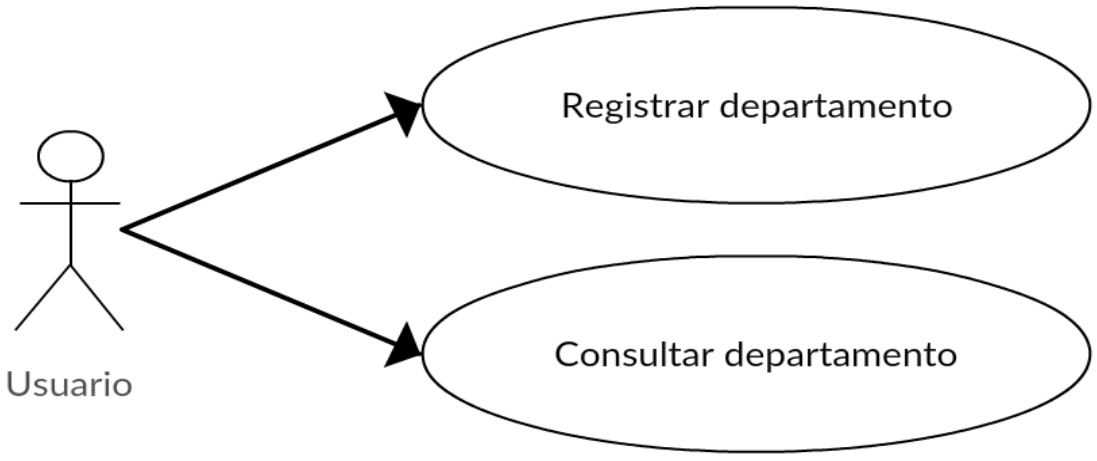


Figura 8.24 Diagrama de casos de uso para el Prototipo 7

Tabla 8.29 Especificación del caso de uso Registrar departamento.

1	El usuario ingresa a la aplicación.
2	La aplicación presenta el menú de selección, por lo que el usuario selecciona la opción 1: Registrar Departamento
2.1	Si el usuario selecciona una opción mayor que 2, la aplicación finaliza.
3	La aplicación presenta una pantalla con una caja de texto para que el usuario digite el nombre del departamento
4	El usuario digita el nombre del departamento y pulsa el botón *Aceptar*
5	La aplicación presenta una pantalla con una caja de texto para que el usuario digite el potencial de sufragantes del departamento
6	El usuario digita el potencial de sufragantes del departamento y pulsa el botón *Aceptar*
7	La aplicación presenta una pantalla con una caja de texto para que el usuario digite el total de sufragantes del departamento
8	El usuario digita el total de sufragantes del departamento y pulsa el botón *Aceptar*
9	La aplicación presenta una pantalla con una caja de texto para que el usuario digite la cantidad de votos válidos obtenidos en el departamento
10	El usuario digita la cantidad de votos válidos del departamento y pulsa el botón *Aceptar*
11	La aplicación muestra en pantalla el nombre del departamento, el potencial de sufragantes, el total de sufragantes, la cantidad de votos válidos, el indicador de participación electoral del departamento, la categoría de participación, la cantidad de votos no válidos y el indicador de error de votación en el departamento que acabó de ingresar.
12	La aplicación vuelve a presentar el menú de selección (paso 2)

Tabla 8.30 Especificación del caso de uso Consultar departamento.

1	El usuario ingresa a la aplicación.
2	La aplicación presenta el menú de selección, por lo que el usuario selecciona la opción 2: Consultar Departamento
2.1	Si el usuario selecciona una opción mayor que 2, la aplicación finaliza.
3	El usuario digita el nombre del departamento a buscar y pulsa el botón *Aceptar*
4	La aplicación muestra en pantalla el nombre del departamento buscado, el potencial de sufragantes, el total de sufragantes, la cantidad de votos válidos, el indicador de participación electoral del departamento, la categoría de participación, la cantidad de votos no válidos y el indicador de error de votación en el departamento.
5	La aplicación vuelve a presentar el menú de selección (paso 2)

8.4.2.2 Diseño de estructura

La estructura no se modifica, la ventaja que se ha ganado al usar la arquitectura en capas es el aspecto de **mantenibilidad**, con la cual se puede fácilmente determinar en qué secciones de la estructura se deben incluir las nuevas funciones sin dañar o desarmar el código previamente desarrollado. En este caso se sabe que se debe solicitar un dato nuevo, el cual implica la creación de un nuevo atributo en la clase Departamento (votos válidos), la modificación del constructor para inicializar el nuevo dato, y los métodos de cálculo de votos no válidos y de indicador de error. Para poder ingresar el nuevo dato se deberá modificar el método de captura de la clase Presentacion, y para visualizarlo se deberá modificar el método de presentación del departamento en la misma clase. Finalmente, no hay variación del menú ni los métodos de grabación y recuperación de los datos desde el archivo.

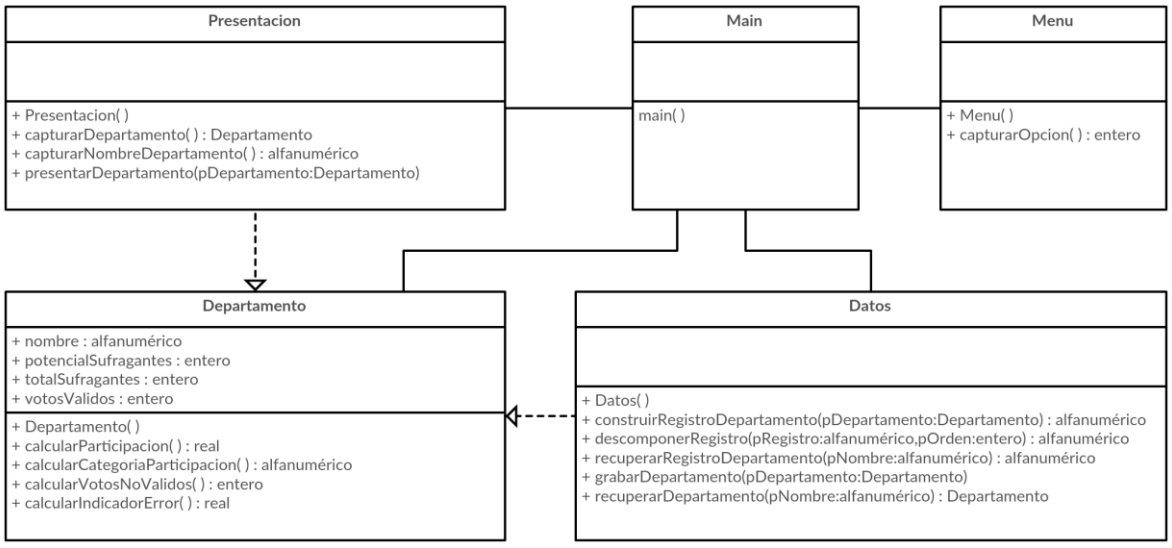


Figura 8.25 Diagrama de clases para el Prototipo 7

8.4.2.3 Diseño de algoritmos

8.4.2.3.1 Clase Departamento - Departamento( )

Descripción: Constructor que inicializa los atributos con valores neutros.

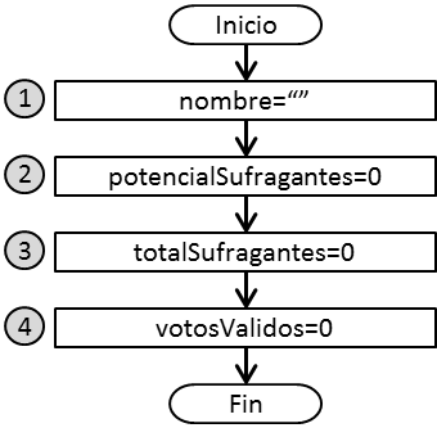


Figura 8.26 Diagrama de flujo para el método Departamento()

Tabla 8.31 Descripción del algoritmo para el método constructor Departamento()

1	Almacena el valor "" en el atributo nombre de cualquier objeto de tipo Departamento al momento de crearse.
2	Almacena el valor 0 en el atributo potencialSufragantes de cualquier objeto de tipo Departamento al momento de crearse
3	Almacena el valor 0 en el atributo totalSufragantes de cualquier objeto de tipo Departamento al momento de crearse
4	Almacena el valor 0 en el atributo votosValidos de cualquier objeto de tipo Departamento al momento de crearse

8.4.2.3.2 Clase Departamento – calcularParticipacion( ): real

Descripción: Método de cálculo para obtener el indicador de participación electoral con base en el potencial y el total de sufragantes del departamento.

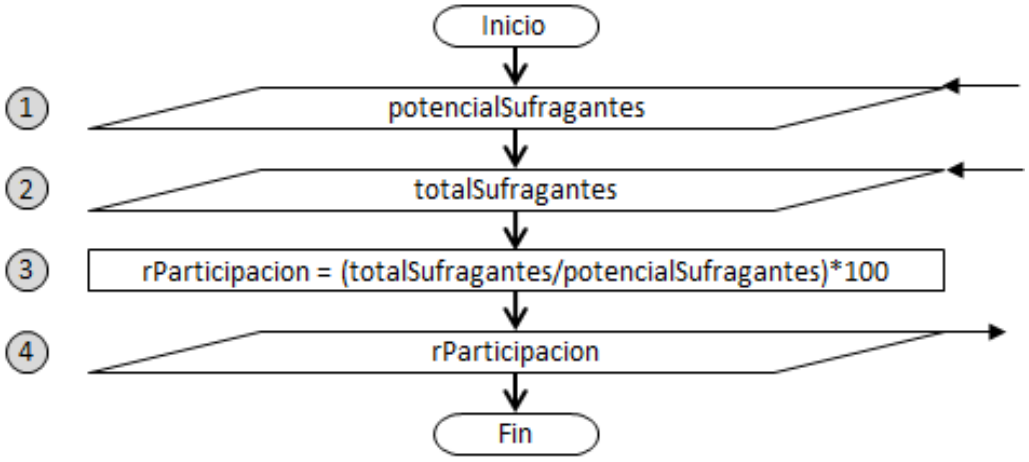


Figura 8.27 Diagrama de flujo para el método calcularParticipacion()

8.4.2.3.3 Clase Departamento – calcularCategoriaParticipacion( ): cadena

Descripción: Método de cálculo para determinar la categoría de participación electoral del departamento, con base en el indicador de participación calculado por el método anterior.

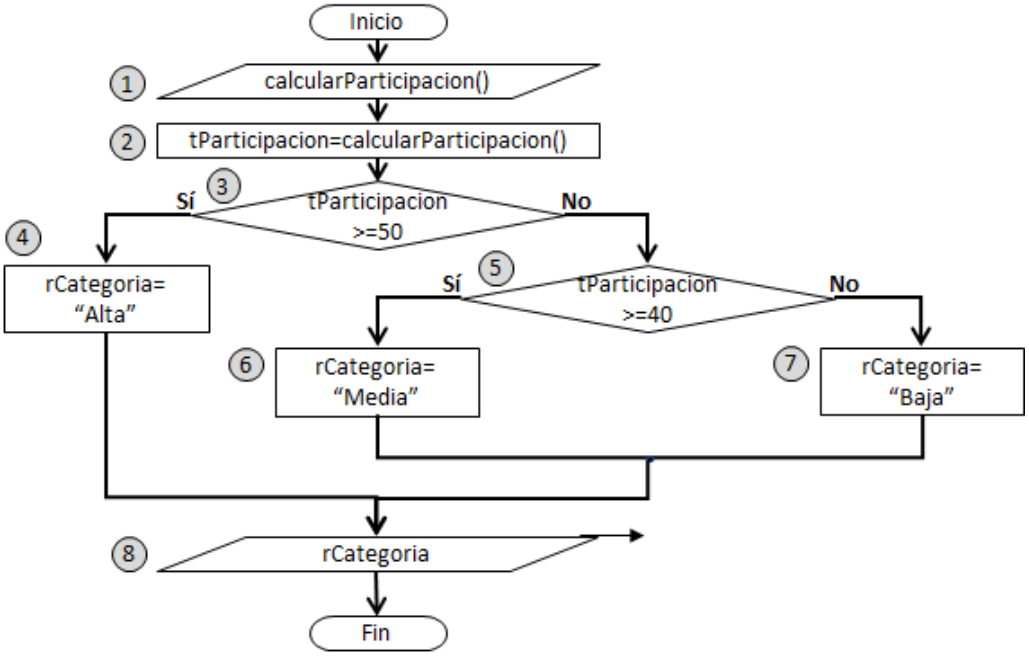


Figura 8.28 Diagrama de flujo para el método calcularCategoriaParticipacion()

8.4.2.3.4 Clase Departamento - calcularVotosNoValidos( ):entero

Descripción: Calcular la cantidad de votos no válidos con base en el total de sufragantes y los votos válidos en el departamento

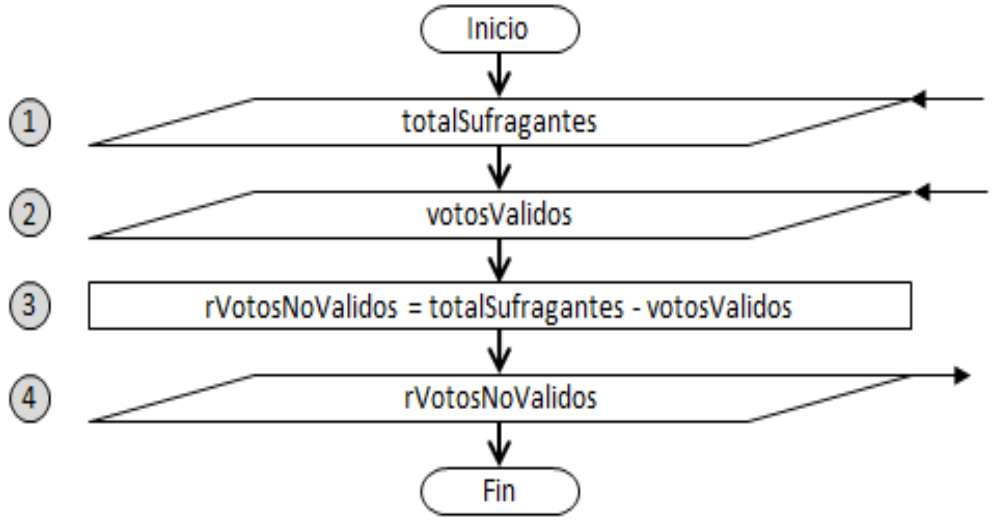


Figura 8.29 Diagrama de flujo para el método calcularVotosNoValidos()

Tabla 8.32 Descripción del algoritmo para el método calcularVotosNoValidos ()

1	Se requiere como dato de entrada *totalSufragantes,* se cumple por tener este dato como atributo propio.
2	Se requiere como dato de entrada *votosValidos,* se cumple por tener este dato como atributo propio.
3	Resta *totalSufragantes* menos *votosValidos*. Almacena el resultado de la operación en la variable *rVotosNoValidos*.
4	Se genera como dato de salida o respuesta del método el valor de la variable *rVotosNoValidos*.

8.4.2.3.5 Clase Departamento - calcularIndicadorError( ):real

Descripción: Calcular el indicador de participación electoral con base en el potencial y el total de sugfagantes del departamento

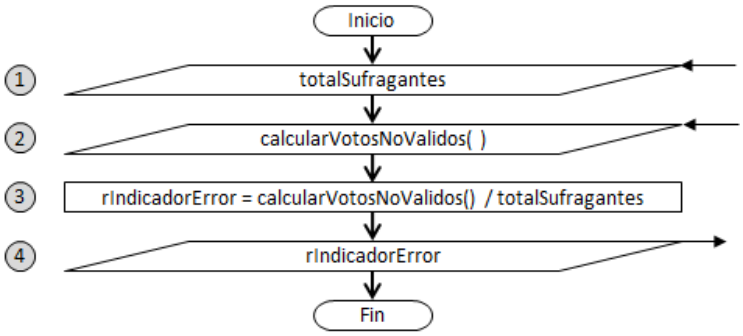


Figura 8.30 Diagrama de flujo para el método calcularIndicadorError()

Tabla 8.33 Descripción del algoritmo para el método calcularIndicadorError()

1	Se requiere como dato de entrada *totalSufragantes,* se cumple por tener este dato como atributo propio.
2	Se requiere como dato de entrada el resultado del método *calcularVotosNoValidos.*
3	Divide el resultado obtenido de *calcularVotosNoValidos* entre *totalSufragantes*. Almacena el resultado de la operación en la variable *rIndicadorError*.
4	Se genera como dato de salida o respuesta del método el valor de la variable *rIndicadorError*.

8.4.2.3.6 Clase Presentacion - capturarDepartamento( ):Departamento

Descripción: Obtiene del usuario cada uno de los valores para los atributos, y entrega como respuesta el objeto de la clase Departamento con los valores de los atributos asignados.

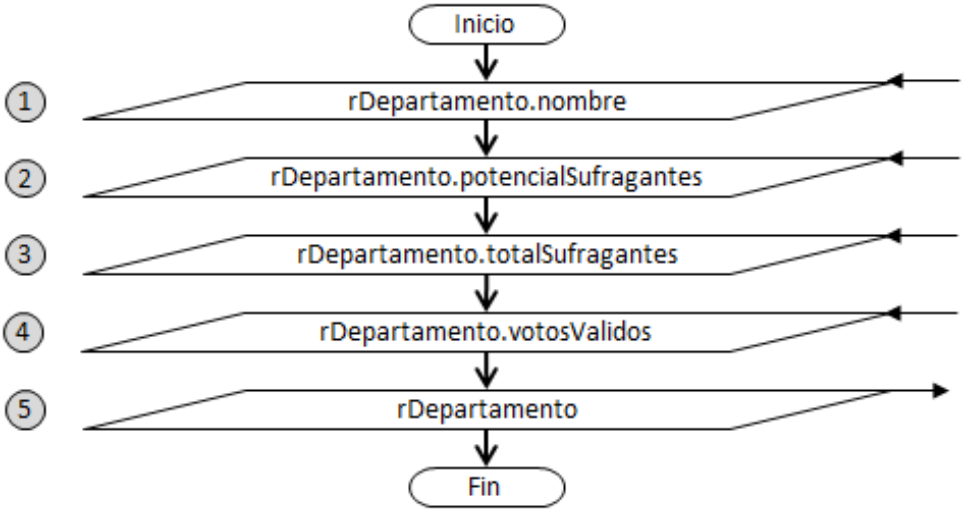


Figura 8.31 Diagrama de flujo para el método capturarDepartamento()

Tabla 8.34 Descripción del algoritmo para el método capturarDepartamento()

1	Se requiere como dato de entrada el nombre del departamento para almacenar en el objeto de tipo Departamento. Para cada dato de entrada, la aplicación debe presentar una pantalla con una caja de texto.

2	Se requiere como dato de entrada el potencial de sufragantes del departamento para almacenar en el objeto
3	Se requiere como dato de entrada el total de sufragantes del departamento para almacenar en el objeto
4	Se requiere como dato de entrada el total de votos válidos en el departamento para almacenar en el objeto
5	Se genera como dato de salida el objeto *rDepartamento*, de la clase *Departamento*. Para ello se realiza como respuesta de este método.

8.4.2.3.7 Clase Presentacion - capturarNombreDepartamento( ):cadena

Descripción: Obtiene del usuario un nombre de departamento, y lo entrega como respuesta al objeto que haya hecho la solicitud, es decir, que haya enviado el mensaje para la ejecución del método.

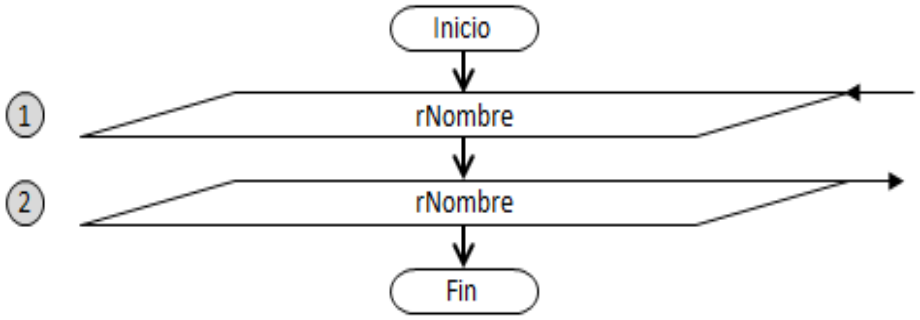


Figura 8.32 Diagrama de flujo para el método capturarNombreDepartamento ()

8.4.2.3.8 Clase Presentacion - presentarDepartamento(pDepartamento:Departamento)

Descripción: Recibe un objeto de la clase Departamento como parámetro. Construye un mensaje al que agrega cada uno de los valores de los atributos del objeto recibido, así como el resultado de los métodos de cálculo del indicador de participación electoral, de la categoría de participación electoral, de la cantidad de votos no válidos y del indicador de error. Por último, presenta el mensaje construido al usuario.

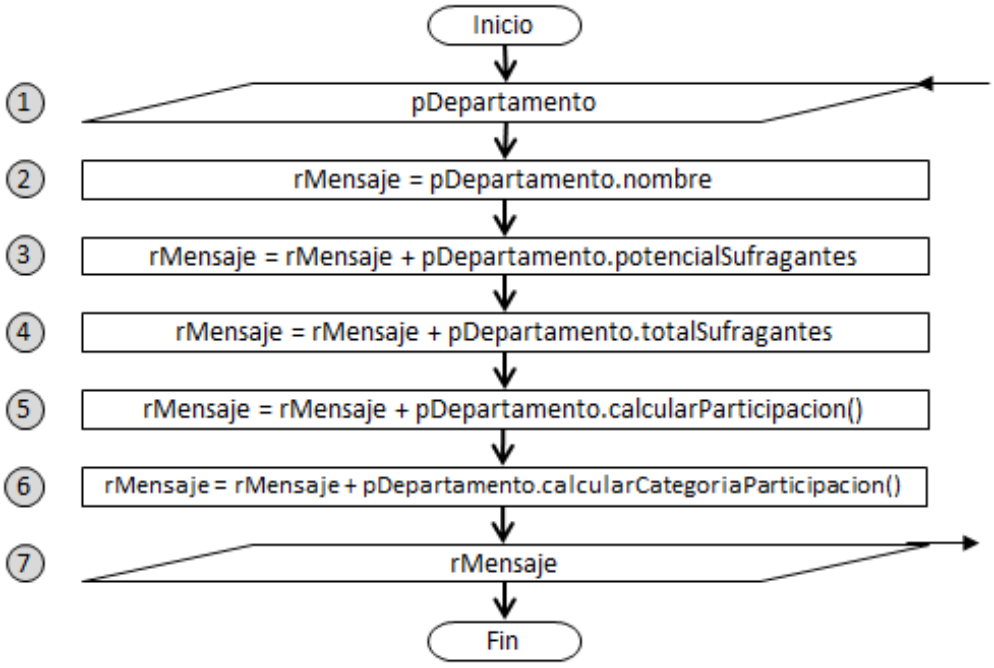


Figura 8.33 Diagrama de flujo para el método presentarDepartamento()

Tabla 8.35 Descripción del algoritmo para el método presentarDepartamento()

1	Se requiere como dato de entrada el objeto de tipo Departamento. En este caso se recibe el objeto como mensaje.
2	Asigna el nombre del departamento del objeto en la variable rMensaje
3	Concatena en la variable rMensaje el potencial de sufragantes del departamento almacenado en el objeto
4	Concatena en la variable rMensaje el total de sufragantes del departamento almacenado en el objeto
5	Concatena en la variable rMensaje el total de votos válidos del departamento almacenado en el objeto
6	Concatena en la variable rMensaje la respuesta del método calcularParticipacion() del objeto.
7	Concatena en la variable rMensaje la respuesta del método calcularCategoriaParticipacion() del objeto.
8	Concatena en la variable rMensaje la respuesta del método calcularVotosNoValidos() del objeto.
9	Concatena en la variable rMensaje la respuesta del método calcularIndicadorError() del objeto.
10	Se genera como dato de salida el mensaje concatenado

8.4.2.3.9 Clase Menu - capturarOpcion(): entero

Descripción: Presenta al usuario un mensaje con las opciones disponibles en la aplicación, y obtiene de dicho usuario el número correspondiente a la opción seleccionada.

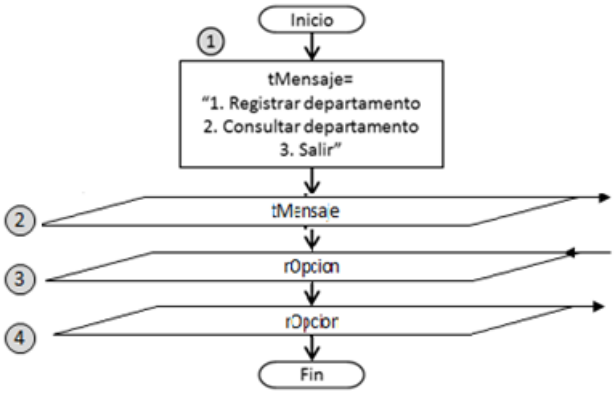


Figura 8.34 Diagrama de flujo para el método capturarOpcion()

Tabla 8.36 Descripción del algoritmo para el método capturarOpcion()

1	Almacena en una variable tMensaje el texto que contiene las opciones disponibles en forma de menú.
2	Presenta en pantalla el contenido de tMensaje
3	Solicita la opción para que la ingrese el usuario. La selección se almacena en la variable rOpcion
4	Se genera como dato de salida el valor de la variable rOpcion.

8.4.2.3.10 Clase Datos - construirRegistroDepartamento(pDepartamento:Departamento): cadena

Descripción: Recibe un mensaje con un objeto de la clase Departamento como parámetro. Construye un registro, es decir una cadena de caracteres, a la que agrega cada uno de los valores de los atributos del objeto separados entre sí por un signo de coma. Por último, lo entrega el registro como respuesta al objeto que haya hecho la solicitud, es decir, que haya invocado ejecución del método.

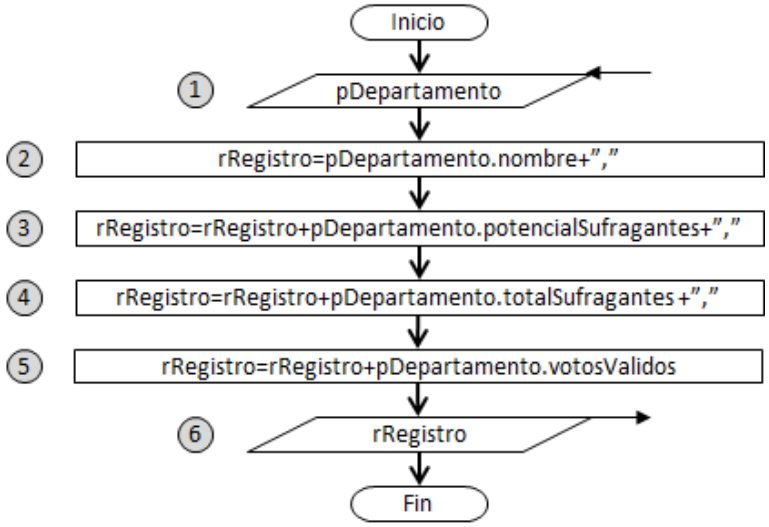


Figura 8.35 Diagrama de flujo para el método construirRegistroDepartamento ()

Tabla 8.37 Descripción del algoritmo para el método construirRegistroDepartamento()

1	Se requiere como dato de entrada el objeto de tipo Departamento. En este caso se recibe el objeto como mensaje.
2	Asigna el nombre del departamento del objeto en la variable rRegistro y concatena con una coma
3	Concatena en la variable rRegistro el potencial de sufragantes del departamento almacenado en el objeto y concatena con una coma
4	Concatena en la variable rRegistro el total de sufragantes del departamento almacenado en el objeto
5	Concatena en la variable rRegistro el total de votos válidos del departamento almacenado en el objeto
6	Se genera como dato de salida el mensaje concatenado

8.4.2.3.11 Clase Datos - descomponerRegistro (pRegistro:cadena, pOrden: entero):cadena

Descripción: Recibe un mensaje con dos parámetros; el primero de estos es un registro o cadena de caracteres que contiene valores separados entre sí por signos de coma; el segundo, un número ordinal que indica el dato que se debe extraer del registro y entregar como respuesta final. El método hace un recorrido secuencial por los caracteres del registro, y cuando ubica el comienzo del dato solicitado, comienza a agregar cada caracter en una variable que al final entrega como respuesta al objeto que haya requerido la ejecución del método.

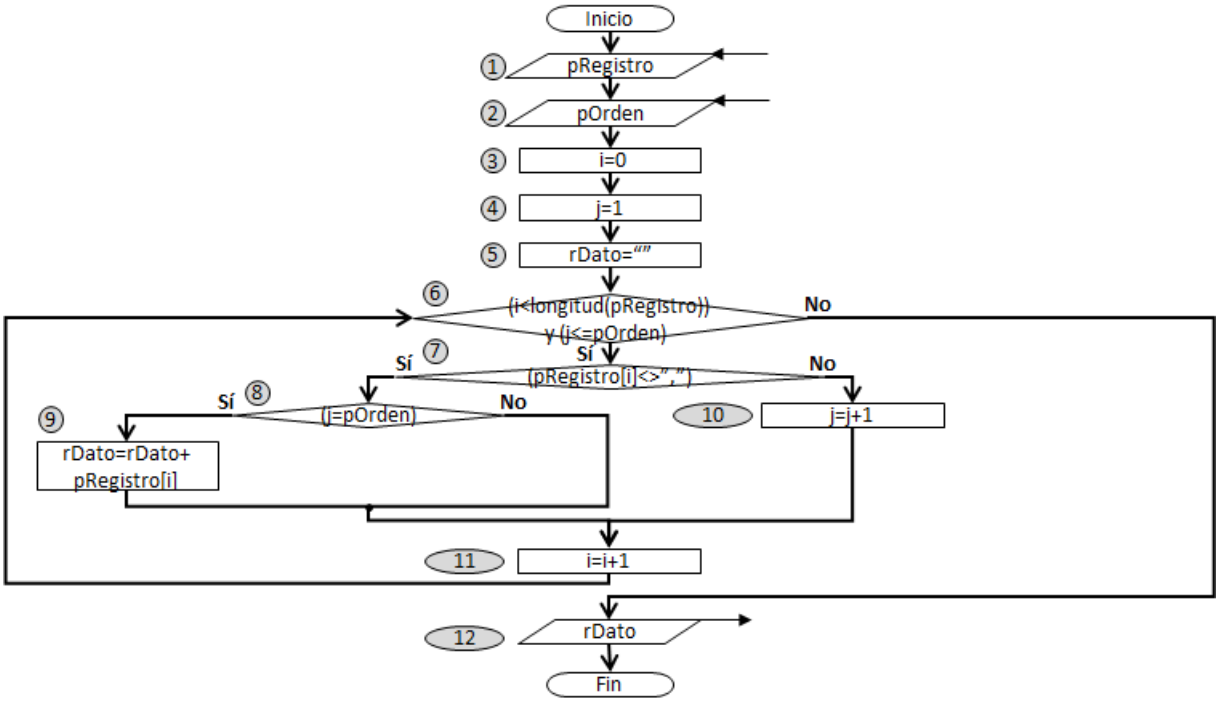


Figura 8.36 Diagrama de flujo para el método descomponerRegistro()

8.4.2.3.12 Clase Datos - grabarDepartamento(pDepartamento:Departamento

Descripción: Recibe un mensaje con un objeto de la clase Departamento como parámetro. Abre un archivo y a continuación invoca el método de construcción del registro, cuya respuesta es grabada en el archivo que finalmente es cerrado.

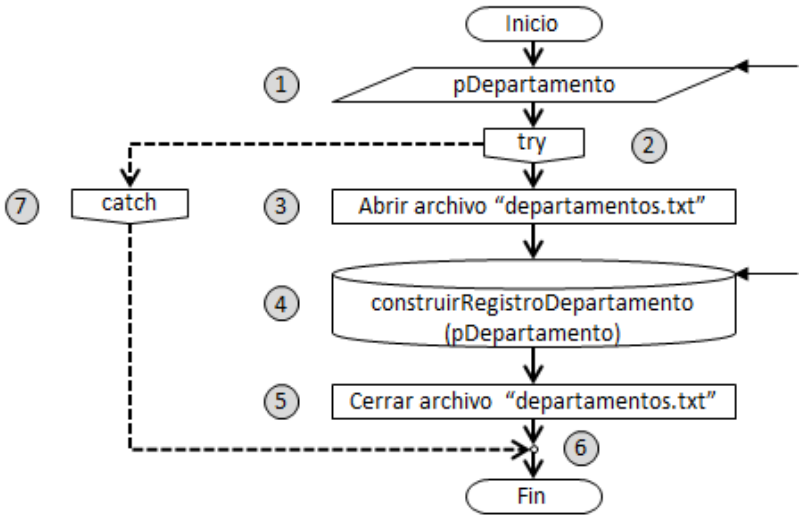


Figura 8.37 Diagrama de flujo para el método grabarDepartamento ()

8.4.2.3.13 Clase Datos - recuperarRegistroDepartamento(pNombre:cadena):cadena

Descripción: Recibe un mensaje con una cadena de caracteres como parámetro, que contiene un nombre de departamento a localizar en el archivo. Abre un archivo para lectura y a continuación lee uno a uno los registros hasta encontrar el que contiene los datos del departamento requerido, para lo cual solicita la descomposición del registro y la obtención del primer dato que corresponde al nombre. Por último, entrega como respuesta el registro que contenga el nombre buscado.

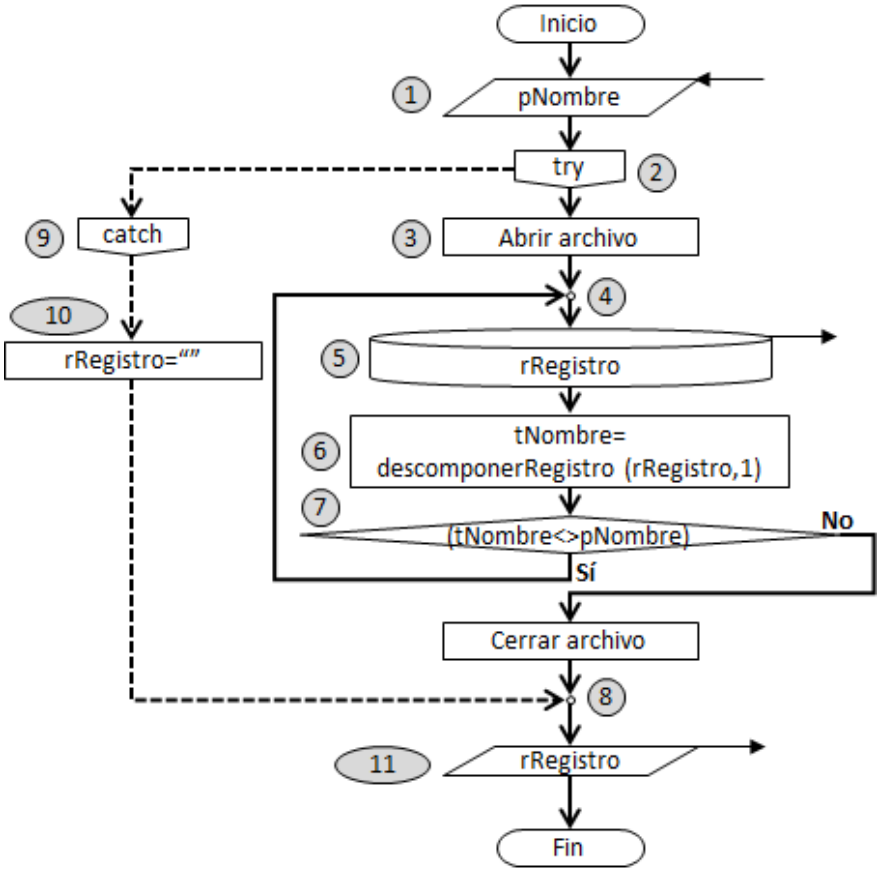


Figura 8.38 Diagrama de flujo para el método recuperarRegistroDepartamento()

8.4.2.3.14 Clase Datos - recuperarDepartamento(pNombre:cadena):Departamento

Descripción: Recibe un mensaje con una cadena de caracteres como parámetro, que contiene un nombre de departamento a localizar en el archivo. Solicita la ejecución del método de recuperación de registro mediante el nombre dado, y una vez obtenida la respuesta de aquel método, toma el registro hallado y solicita su descomposición en cuatro partes. Por último, entrega como respuesta el objeto de la clase Departamento correspondiente al nombre dado como criterio de búsqueda.

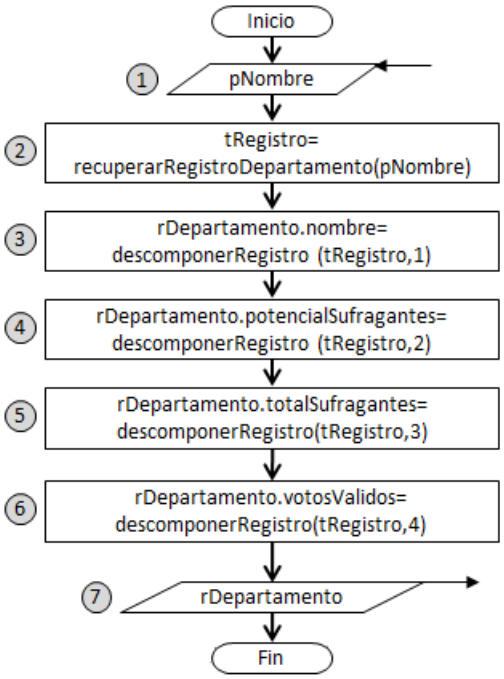


Figura 8.39 Diagrama de flujo para el método recuperarDepartamento()

Tabla 8.38 Descripción del algoritmo para el método recuperarDepartamento()

1	Se requiere como dato de entrada la variable pNombre. En este caso se recibe la variable como mensaje.
2	Almacena en la variable tRegistro la respuesta del método recuperarDepartamento () del objeto, envíandole como parámetro la variable pNombre
3	Se crea un objeto nuevo de tipo Departamento y se almacena en su atributo nombre la respuesta del método descomponerRegistro() del objeto, envíandole como parámetros el valor tRegistro y el valor 1 para extraer el dato del nombre del departamento.
4	Se almacena en el atributo potencialSufragantes del objeto de tipo Departamento la respuesta del método descomponerRegistro() del objeto, envíandole como parámetros el valor tRegistro y el valor 2 para extraer el dato del potencial de sufragantes del departamento.
5	Se almacena en el atributo totalSufragantes del objeto de tipo Departamento la respuesta del método descomponerRegistro() del objeto, envíandole como parámetros el valor tRegistro y el valor 3 para extraer el dato del total de sufragantes del departamento.
6	Se almacena en el atributo votosValidos del objeto de tipo Departamento la respuesta del método descomponerRegistro() del objeto, envíandole como parámetros el valor tRegistro y el valor 4 para extraer el dato del total de sufragantes del departamento.
7	Se genera como dato de salida el objeto creado de tipo Departamento con todos sus atributos llenos.

8.4.2.3.15 Clase Main – main( )

Descripción: Coordina las acciones de las demás clases. Solicita a un objeto de la clase Menu que reciba de parte del usuario la opción requerida por él, que almacena en una variable.

Si la opción seleccionada es 1, coordina las clases para que el usuario pueda registrar un departamento, es decir, solicita a un objeto de la clase Presentacion que lleve a cabo la captura

de un objeto de la clase Departamento, que almacena luego en el objeto departamento, solicita al objeto de la clase Datos que grabe el objeto departamento y finalmente muestra la información del departamento ingresado.

Si la opción seleccionada es 2, coordina las clases para que el usuario pueda consultar dando como parámetro un nombre de departamento, solicita al objeto datos la recuperación de un departamento mediante el nombre capturado por el objeto presentación; el resultado de la operación de recuperación se almacena en el objeto departamento. Por último, ordena al objeto de la clase Presentacion que lleve a cabo el método de presentación tomando como parámetro el objeto de la clase Departamento.

Todo esto se repite mientras la opción seleccionada sea menor o igual que 2.

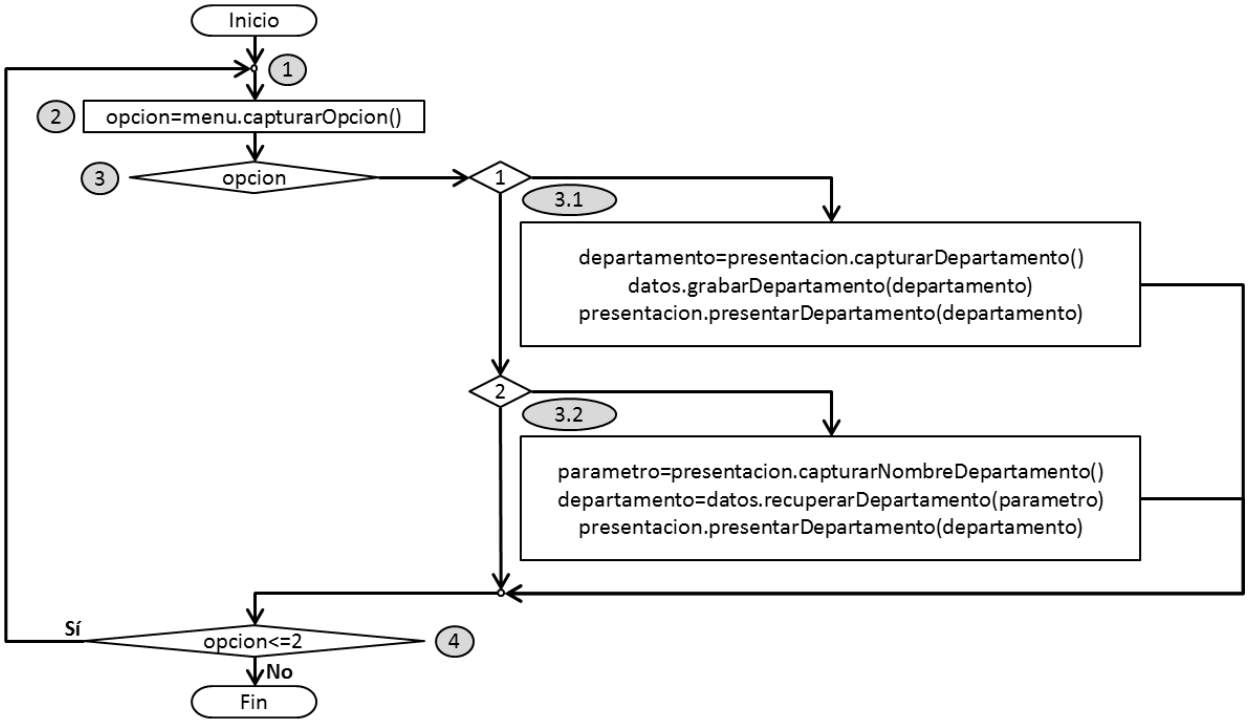


Figura 8.40 Diagrama de flujo para el método main()

8.4.3 Implementación

La estructura del proyecto no cambia, pero se modificarán varios métodos de las diferentes capas: en Presentación para solicitar y mostrar la nueva información; en Departamento para ingresar el nuevo atributo y los nuevos métodos de cálculo; y en Datos para almacenar y recuperar la nueva información desde el archivo.

Tabla 8.39 Codificación de la clase Departamento

```
1  package proyecto;
2
3  public class Departamento {
4     public String nombre;
5     public int potencialSufragantes;
6     public int totalSufragantes;
7     public int votosValidos;
8
9     public Departamento(){
10       nombre="";
11       potencialSufragantes=0;
12       totalSufragantes=0;
13       votosValidos=0;
14    }
15
16    public double calcularParticipacion(){
17       double rParticipacion;
18       rParticipacion=((double)totalSufragantes/potencialSufragantes)*100;
19       return rParticipacion;
20    }
21
22    public String calcularCategoriaParticipacion(){
23       double tParticipacion;
24       String rCategoria;
25       tParticipacion=calcularParticipacion();
26       if (tParticipacion>=50){
27          rCategoria="Alta";
28       }
29       else{
30          if (tParticipacion>=40){
31             rCategoria="Media";
32          }
33          else{
34             rCategoria="Baja";
35          }
36       }
37       return rCategoria;
38    }
39
40    public int calcularVotosNoValidos(){
41       int rVotosNoValidos;
42       rVotosNoValidos=totalSufragantes-votosValidos;
43       return rVotosNoValidos;
44    }
45
46    public double calcularIndicadorError(){
47       double rIndicadorError;
48       rIndicadorError=(double)calcularVotosNoValidos()/totalSufragantes;
49       return rIndicadorError;
50    }
51 }
```

Tabla 8.40 Codificación de la clase Presentacion

```
1  package proyecto;
2  import javax.swing.JOptionPane;
3
4  public class Presentacion {
5     public Presentacion(){
6
7     }
8
9     public Departamento capturarDepartamento(){
10       Departamento rDepartamento;
11       rDepartamento=new Departamento();
12       rDepartamento.nombre=JOptionPane.showInputDialog("Nombre: ");
13       rDepartamento.potencialSufragantes=Integer.parseInt(JOptionPane.showInputDialog("Potencial de
   sufragantes: "));
14       rDepartamento.totalSufragantes=Integer.parseInt(JOptionPane.showInputDialog("Total de
   sufragantes: "));
15       rDepartamento.votosValidos=Integer.parseInt(JOptionPane.showInputDialog("Votos válidos: "));
16       return rDepartamento;
17    }
18
19    public String capturarNombreDepartamento(){
20       String rNombre;
21       rNombre=JOptionPane.showInputDialog("Departamento: ");
22       return rNombre;
23    }
24
25    public void presentarDepartamento(Departamento pDepartamento){
26       String rMensaje;
27       rMensaje="Departamento: "+pDepartamento.nombre;
28       rMensaje=rMensaje+"\nPotencial de sufragantes: "+pDepartamento.potencialSufragantes;
29       rMensaje=rMensaje+"\nTotal de sufragantes: "+pDepartamento.totalSufragantes;
30       rMensaje=rMensaje+"\nVotos válidos: "+pDepartamento.votosValidos;
         rMensaje=rMensaje+"\nIndicador de participación: "+pDepartamento.calcularParticipacion()+" %";
31       rMensaje=rMensaje+"\nCategoría según participación:
32 "+pDepartamento.calcularCategoriaParticipacion();
33       rMensaje=rMensaje+"\nVotos no válidos: "+pDepartamento.calcularVotosNoValidos();
34       rMensaje=rMensaje+"\nIndicador de error: "+pDepartamento.calcularIndicadorError();
35       JOptionPane.showMessageDialog(null, rMensaje);
36    }
37 }
```

Tabla 8.41 Codificación de la clase Menu

```
1  package proyecto;
2  import javax.swing.JOptionPane;
3
4  public class Menu {
5     public Menu(){
6
```

```
7     }
8
9     public int capturarOpcion(){
10       String tMensaje;
11       int rOpcion;
12       tMensaje="1. Registrar departamento\n"+
13             "2. Consultar departamento\n"+
14             "3. Salir\n";
15       rOpcion=Integer.parseInt(JOptionPane.showInputDialog(tMensaje));
16       return rOpcion;
17    }
18 }
```

Tabla 8.42 Codificación de la clase Datos

```
1  package proyecto;
2  import java.io.*;
3
4  public class Datos {
5     public Datos(){
6
7     }
8
9     public String construirRegistroDepartamento(Departamento pDepartamento){
10       String rRegistro;
11       rRegistro=pDepartamento.nombre+",";
12       rRegistro=rRegistro+pDepartamento.potencialSufragantes+",";
13       rRegistro=rRegistro+pDepartamento.totalSufragantes+",";
14       rRegistro=rRegistro+pDepartamento.votosValidos;
15       return rRegistro;
16    }
17
18    public String descomponerRegistro(String pRegistro, int pOrden){
19       int i,j;
20       String rDato;
21       i=0;
22       j=1;
23       rDato="";
24       while ((i<pRegistro.length()) && (j<=pOrden)) {
25          if (pRegistro.substring(i,i+1).compareTo(",")!=0){
26             if (j==pOrden){
27                rDato=rDato+pRegistro.substring(i,i+1);
28             }
29          }
30          else{
31             j=j+1;
32          }
33          i=i+1;
34       }
35       return rDato;
36    }
```

```
37
38      public String recuperarRegistroDepartamento(String pNombre){
39          FileReader fileReader;
40          BufferedReader bufferedReader;
41          String rRegistro,tNombre;
42          try{
43              fileReader=new FileReader("departamentos.txt");
44              bufferedReader=new BufferedReader(fileReader);
45              do{
46                  rRegistro=bufferedReader.readLine();
47                  tNombre=descomponerRegistro(rRegistro, 1);
48              } while (tNombre.compareToIgnoreCase(pNombre)!=0);
49              fileReader.close();
50          }
51          catch (Exception e){
52              rRegistro="";
53          }
54          return rRegistro;
55      }
56
57      public Departamento recuperarDepartamento(String pNombre){
58          String tRegistro;
59          Departamento rDepartamento;
60          rDepartamento=new Departamento();
61          tRegistro=recuperarRegistroDepartamento(pNombre);
62          rDepartamento.nombre=descomponerRegistro(tRegistro,1);
63          rDepartamento.potencialSufragantes=Integer.parseInt(descomponerRegistro(tRegistro,2));
64          rDepartamento.totalSufragantes=Integer.parseInt(descomponerRegistro(tRegistro,3));
65          rDepartamento.votosValidos=Integer.parseInt(descomponerRegistro(tRegistro,4));
66          return rDepartamento;
67      }
68
69      public void grabarDepartamento(Departamento pDepartamento){
70          FileWriter fileWriter;
71          PrintWriter printWriter;
72          try{
73              fileWriter=new FileWriter("departamentos.txt",true);
74              printWriter=new PrintWriter(fileWriter);
75              printWriter.println(this.construirRegistroDepartamento(pDepartamento));
76              fileWriter.close();
77          }
78          catch(Exception e){
79
80          }
81      }
82  }
```

Tabla 8.43 Codificación de la clase Main

```
1  package proyecto;
2
3  public class Main {
4
5     public static void main(String[] args) {
6        Departamento departamento;
7        Presentacion presentacion;
8        Menu menu;
9        Datos datos;
10       int opcion;
11       String parametro;
12       presentacion=new Presentacion();
13       menu=new Menu();
14       datos=new Datos();
15       opcion=0;
16       do {
17          opcion=menu.capturarOpcion();
18          switch (opcion){
19             case 1:{
20                departamento=presentacion.capturarDepartamento();
21                datos.grabarDepartamento(departamento);
22                presentacion.presentarDepartamento(departamento);
23                break;
24             }
25             case 2:{
26                parametro=presentacion.capturarNombreDepartamento();
27                departamento=datos.recuperarDepartamento(parametro);
28                presentacion.presentarDepartamento(departamento);
29                break;
30             }
31          }
32       } while (opcion<=2);
33    }
34 }
```

8.5 Síntesis de conceptos

8.5.1 Orientación a objetos

- **Persistencia de datos**

Cuando se desarrolla una aplicación que gestione datos, es muy importante que los datos se conserven mientras se ejecutan los diferentes procesos del programa y así obtener los resultados esperados. El programa debe tener la capacidad de que la información se guarde y se conserve de forma permanente (**almacenamiento de datos**) y a su vez se pueda recuperar o leer para ser nuevamente utilizada (**recuperación de datos**) a esto se le conoce como ***persistencia***.

En la arquitectura de capas, se recomienda que las clases que se encarguen de realizar los procesos de **persistencia de datos** se encuentren en una capa particular, que tiene comunicación directa con las clases de la capa de reglas o de lógica de negocio. Sin embargo, se recomienda, por aspectos de seguridad que al programar no se realice ningún tipo de vínculo de comunicación entre las clases de persistencia y las clases de la capa de presentación, puesto que deben pasar siempre por los filtros y reglas programados en las clases de lógica de negocio.

8.5.2 Programación en Java – Buenas prácticas

- **Manejo de cadenas de caracteres, concatenación y formato CSV**

En la mayoría de los programas se utilizan las ***cadenas de caracteres***, que son secuencias de caracteres de un tamaño o longitud específica, y pueden contener palabras o frases (Dean & Dean, 2009). Para crear una cadena de caracteres a partir de código debe indicarse el mensaje que lo contiene entre comillas, como por ejemplo: “hola mundo”. Java crea un objeto tipo String que puede contener una cadena de caracteres. Estas cadenas de caracteres también se pueden ***concatenar***, es decir se pueden unir o enlazar varias cadenas de caracteres en una sola. La concatenación de cadenas de caracteres se realiza utilizando el operador +. Un ejemplo de esto puede ser **concatenar** un mensaje creado con comillas y una variable de tipo String que contenga un valor, así:

```
String mensaje = “hola mundo”;

System.out.println(“Nuevo mensaje: ” + mensaje);
```

Lo que se observaría en pantalla sería este mensaje:

```
Nuevo mensaje: hola mundo
```

La **concatenación** de datos es indispensable en todo programa que deba manipular archivos planos para lectura o escritura. Estos archivos planos son comúnmente usados para transmitir información entre diferentes sistemas de información. Para ello es posible usar el formato **CSV**, un formato particular que permite representar datos en forma de tabla, en las que los datos de un registro se separan por comas como su nombre en inglés lo indica “*comma separated values*". Los ***archivos CSV*** son un tipo de documento que contienen registros cuyos datos son valores separados por comas, y cada registro es separado del otro por medio del salto de línea.

- **Manejo de errores o excepciones**

Es posible que un programa presente un error cuando se esté ejecutando, que pueden ser debido a un error de cálculo por un dato faltante, o por algún ingreso invalido del mismo usuario, es decir algún comportamiento inesperado del código que lleve a su terminación fallida. Es por eso por lo que existen técnicas para contrarrestar este tipo de situaciones como el **manejo de excepciones**.

Para este **manejo de excepciones** se utilizan los bloques try y catch, los cuales tienen como función “atrapar” mensajes de excepción y fallas del programa, donde try realiza una llamada a

instrucciones que pueden considerarse como de posible “riesgo”. Una vez try realiza el llamado a estas instrucciones ”riesgosas”, en caso de que algo salga mal, inmediatamente se dirige al bloque catch que procede a ejecutar las instrucciones definidas en este, las cuales actúan como un método de control para detener algún error al momento de la ejecución. En caso de que no ocurra nada cuando se realizan las instrucciones “riesgosas” por medio del try no se ejecutan las instrucciones del catch.

La sintaxis para los bloques try y catch se presenta en este ejemplo:

Tabla 8.44 Sintaxis para los bloques try y catch

```
1  try
2  {
3    <instrucciones try>
4  }
5  catch (<excepción-Clase> <parametros>)
6  {
7     <instrucciones catch>
   }
```

8.5.3 Ingeniería del software

- **Tarjetas CRC**

Existe una técnica común en la práctica de la ingeniería del software en la cual se crean **tarjetas CRC** (clase – **responsabilidad** – colaboración) cuya finalidad es tener una representación sencilla y organizada para identificar cada una de las clases, sus **responsabilidades** representadas en sus atributos y operaciones y sus colaboradores, que son aquellas clases complemento que proporcionan la información necesaria para que la clase complete todas las **responsabilidades** identificadas. Este modelo es representado por medio del uso de tarjetas que pueden ser reales o virtuales. Estas **tarjetas CRC** estas sujetas a modificaciones, ya sean correcciones o agregados según como se requiera, y finalmente su propósito a parte de una representación organizada de las diferentes clases, es que se puedan realizar cambios en una tarjeta, ya sea editándola, desechándola o reorganizando y no cuando sea necesario reorganizar el código fuente, lo cual puede generar gastos más grandes. Para definir las **responsabilidades** es importante tener en cuenta estos lineamientos:

- Inteligencia del sistema: Cuando se desarrolla un programa se debe tener presente que pueden tener un grado de inteligencia, haciendo referencia a lo que el sistema sabe y puede hacer, esto debe estar distribuido en diferentes clases que contengan diferentes **responsabilidades**. Existen dos tipos de clases unas que contienen pocas **responsabilidades** que sirven como apoyo a otras clases, como aquellas clases que se le asignan muchas más **responsabilidades**, siendo estas el otro tipo de clase. Si se concentran todas las **responsabilidades** en pocas clases, esto hará que sean más complejos los cambios y al fin será necesario incluir más clases. Por el contrario, si se logra una distribución de **responsabilidades** más pareja, facilitará el mantenimiento del software.

- Enunciado de forma general: cuando se enuncia una **responsabilidad** debe ser de la forma más general y deben ser definidas en un nivel de jerarquía de clases.

- Localización de información en una sola clase: Cuando se habla de la **responsabilidad** de almacenar y manipular información, esta **responsabilidad** debe estar definida en una única clase, ya que si se encuentra distribuida será complejo el mantenimiento del software.

- Excepción en distribución de **responsabilidades** en diferentes clases: Existen casos que se considera apropiado compartir **responsabilidades** entre clases, esto ocurre cuando varios objetos que se encuentran relacionados requieren tener el mismo comportamiento en un mismo tiempo.

Cuando se definen las colaboraciones, se deben tener en cuenta las dos formas que una clase cumple sus **responsabilidades**: 1) Tiene sus propias operaciones y atributos, cumpliendo así sus responsabilidades particulares; 2) Es utilizada para colaborar con otras clases.

- **Atributos de calidad**

El software debe contar con atributos de calidad, uno de ellos es la **mantenibilidad**, que hace referencia a que el programa tenga la capacidad de adaptarse, que se pueda ampliar, configurar, probar y que sea compatible, así mismo que la instalación del sistema sea de forma sencilla (Pressman, 2012). En otras palabras, esta característica representa la capacidad del software para ser modificado de forma efectiva y eficiente, debido a necesidades de mejora, arreglo o perfeccionamiento. De acuerdo con la normatividad ISO 25000, esta característica se subdivide a su vez en las siguientes subcaracterísticas:

- Modularidad: Capacidad de un sistema o programa de ordenador (compuesto de componentes discretos) que permite que un cambio en un componente tenga un impacto mínimo en los demás.
- Reusabilidad: Capacidad de un activo que permite que sea utilizado en más de un sistema software o en la construcción de otros activos.
- Analizabilidad: Facilidad con la que se puede evaluar el impacto de un determinado cambio sobre el resto del software, diagnosticar las deficiencias o causas de fallos en el software, o identificar las partes a modificar.
- Capacidad para ser modificado: Capacidad del producto que permite que sea modificado de forma efectiva y eficiente sin introducir defectos o degradar el desempeño.
    - Capacidad para ser probado: Facilidad con la que se pueden establecer criterios de prueba para un sistema o componente y con la que se pueden llevar a cabo las pruebas para determinar si se cumplen dichos criterios

Referencias

Dean, J., & Dean, R. (2009). *Introducción a la programación con Java* (1st ed.). McGraw-Hill Interamericana.

Pressman, R. (2012). *Ingeniería del Software* (5th ed.). McGraw Hill.

9 Consistencia de los datos

En el capítulo anterior se evidenció la implementación de las funciones de persistencia de datos a través de las diferentes capas. Los prototipos ahora permiten el almacenamiento y recuperación de los datos que se ingresen, pero se debe tener en cuenta también en los diseños de las aplicaciones la posibilidad de que los usuarios no ingresen adecuadamente los datos. Información incompleta, incorrecta o repetida podría ser ingresada y podría ocasionar incluso serios problemas en una organización.

Se buscará ahora con estos prototipos incluir las funciones que permitan, de alguna forma, garantizar la consistencia de los datos ingresados. La Tabla 9.1 muestra las metas de aprendizaje para estos prototipos.

Tabla 9.1 Objetivos de aprendizaje del capítulo.

Desarrollo de software	Conceptos	Instrumentos
Análisis Diseño Implementación	Clase: Métodos *setters* y *getters*, encapsulamiento, objeto *this* Algoritmo: condiciones de validación de datos	UML: Diagrama de casos de uso Especificación de casos de uso Diagrama de clases Diagrama de flujo

9.1 Validación general - Prototipo 8

9.1.1 Análisis

9.1.1.1 Alcance

El prototipo 7, aunque funcional, tiene como deficiencia el permitir la asignación de datos inconsistentes que luego serán presentados en pantalla y grabados en el archivo. El prototipo 8 busca incorporar un método de validación general; en caso de inconsistencia de los datos a su ingreso, no permite la grabación de estos, ni presentar los datos posteriormente, es decir, el registro quedaría invalidado inmediatamente.

9.1.1.2 Definiciones

No hay modificaciones en los datos requeridos en el negocio. Sin embargo, aparecen unas reglas particulares que deben cumplir cada uno de los datos ingresados. Estas reglas se conocen como **condiciones de validación de datos**. Para este prototipo, se establecen las siguientes:

- El nombre del departamento no puede quedar vacío.

- El potencial de sufragantes debe ser mayor que cero.
- El total de sufragantes debe ser mayor que cero, pero menor o igual al potencial de sufragantes.
- La cantidad de votos válidos debe ser mayor que cero, pero menos o igual al total de sufragantes.

Tabla 9.2 Definiciones aplicables en desarrollo del prototipo 08.

	Definición	Tipo de dato
Departamento	Cada una de las circunscripciones territoriales de primer nivel en que se divide el país para propósitos electorales	Objeto
Nombre	Identificación nominal del departamento	Cadena de caracteres
Potencial de sufragantes	Cantidad de personas habilitadas en un departamento para votar en un proceso electoral	Número entero
Total de sufragantes	Cantidad de personas que acudieron a votar en un departamento en un proceso electoral	Número entero
Indicador de participación electoral	Indicador consistente en el total de sufragantes expresado como porcentaje del potencial de sufragantes $Indicador\ de\ participación\ electoral = \frac{Total\ de\ sufragantes}{Potencial\ de\ sufragantes} * 100$	Número real
Categoría de participación electoral	Categoría a la que pertenece el departamento, dependiendo del porcentaje de participación electoral: Si el porcentaje es mayor o igual a 50%: alto Si el porcentaje es menor a 50% pero mayor o igual a 40%: medio Si el porcentaje es menor a 40%: bajo	Cadena de caracteres
Votos válidos	Cantidad de votos que indican de forma clara la voluntad del sufragante	Número entero
Votos no válidos	Cantidad de votos que NO indican de forma clara la voluntad del sufragante. Se calcula respecto al total de sufragantes $Votos\ no\ válidos = Total\ de\ sufragantes - Votos\ válidos$	Número entero
Indicador de error en la votación	Indicador consistente en dividir el total de votos no válidos entre el total de sufragantes. $Indicador\ de\ error\ en\ la\ votación = \frac{Votos\ no\ válidos}{Total\ de\ sufragantes}$	Número real

9.1.1.3 Requisitos

Hay una modificación en el requisito funcional de ingreso, cálculo y presentación de los datos relacionados con el departamento. Como es posible que los datos ingresados estén erróneos (es decir, que no cumplan con las **condiciones de validación de datos**), no se deben almacenar estos datos en el archivo y no se deben poder mostrar, indicando que hay una inconsistencia en los datos.

9.1.1.3.1 Requisitos de interfaz de usuario

9.1.1.3.1.1 Ingreso de información en casillas de edición

Para la interacción con el usuario, la aplicación debe presentar ventanas con casillas de edición en donde pueda digitar los datos solicitados. Al solicitar los datos, el sistema deberá indicar en la misma ventana de diálogo cuál es el dato requerido.

9.1.1.3.1.2 Presentación de menú de selección de funciones

La aplicación debe presentar un menú para seleccionar una de 3 funciones: registrar la información de un departamento para su almacenamiento, consultar información de in departamento a partir de la recuperación de sus datos almacenados, o finalizar la aplicación.

9.1.1.3.2 Requisitos funcionales

9.1.1.3.2.1 Calcular el indicador de participación electoral de un departamento

La aplicación debe permitir al usuario introducir los siguientes datos básicos de un departamento: el nombre, el potencial de sufragantes, el total de sufragantes y la cantidad de votos válidos, y con estos datos debe calcular el indicador de participación electoral, los votos no válidos y el indicador de error de votación. Los datos ingresados deben cumplir las condiciones de validación requeridas.

9.1.1.3.2.2 Almacenar la información de un departamento en un archivo plano

La aplicación debe poder almacenar en un archivo plano los datos básicos ingresados de un departamento: el nombre, el potencial de sufragantes, el total de sufragantes y el total de votos válidos. Se requiere que el formato de almacenamiento sea en datos separados por punto y coma (formato CSV).

9.1.1.3.2.3 Recuperar la información de un departamento en un archivo plano

La aplicación debe poder recuperar desde un archivo plano los datos básicos ingresados de un departamento: el nombre, el potencial de sufragantes, el total de sufragantes, el total de votos válidos además de los datos calculados del indicador de participación electoral, la categoría de participación a la que pertenece, los votos no válidos y el indicador de error de votación y se debe presentar al usuario. Para ello deberá preguntar por el nombre del departamento a buscar. Como restricción, se requiere el formato de almacenamiento sea en datos separados por punto y coma (formato CSV).

9.1.2 Diseño

9.1.2.1 Diseño de escenarios

Se mantiene el mismo diseño de escenarios respecto a las dos funciones básicas del programa accedidas a través del menú. Se debe incluir la presentación del error de la **validación de datos** en caso de inconsistencia.

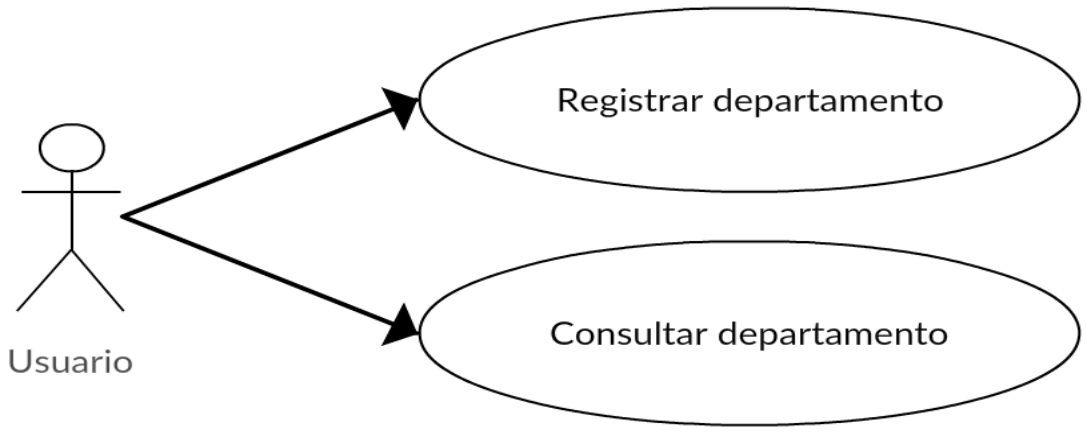


Figura 9.1 Diagrama de casos de uso para el Prototipo 8

Tabla 9.3 Especificación del caso de uso *Registrar departamento*.

1	El usuario ingresa a la aplicación.
2	La aplicación presenta el menú de selección, por lo que el usuario selecciona la opción 1: Registrar Departamento
2.1	Si el usuario selecciona una opción mayor que 2, la aplicación finaliza.
3	La aplicación presenta una pantalla con una caja de texto para que el usuario digite el nombre del departamento
4	El usuario digita el nombre del departamento y pulsa el botón *Aceptar*
5	La aplicación presenta una pantalla con una caja de texto para que el usuario digite el potencial de sufragantes del departamento
6	El usuario digita el potencial de sufragantes del departamento y pulsa el botón *Aceptar*
7	La aplicación presenta una pantalla con una caja de texto para que el usuario digite el total de sufragantes del departamento
8	El usuario digita el total de sufragantes del departamento y pulsa el botón *Aceptar*
9	La aplicación presenta una pantalla con una caja de texto para que el usuario digite la cantidad de votos válidos obtenidos en el departamento
10	El usuario digita la cantidad de votos válidos del departamento y pulsa el botón *Aceptar*
11	Si los datos son válidos: la aplicación muestra en pantalla el nombre del departamento, el potencial de sufragantes, el total de sufragantes, la cantidad de votos válidos, el indicador de participación electoral del departamento, la categoría de participación, la cantidad de votos no válidos y el indicador de error de votación en el departamento que acabó de ingresar.
12	Si los datos no son válidos: la aplicación muestra en pantalla el mensaje “Datos inconsistentes. No se pueden presentar”
13	La aplicación vuelve a presentar el menú de selección (paso 2)

Tabla 9.4 Especificación del caso de uso *Consultar departamento*.

1	El usuario ingresa a la aplicación.
2	La aplicación presenta el menú de selección, por lo que el usuario selecciona la opción 2: Consultar Departamento
2.1	Si el usuario selecciona una opción mayor que 2, la aplicación finaliza.
3	El usuario digita el nombre del departamento a buscar y pulsa el botón *Aceptar*
4	La aplicación muestra en pantalla el nombre del departamento buscado, el potencial de sufragantes, el total de sufragantes, la cantidad de votos válidos, el indicador de participación electoral del departamento, la categoría de participación, la cantidad de votos no válidos y el indicador de error de votación en el departamento.

5	La aplicación vuelve a presentar el menú de selección (paso 2)

9.1.2.2 Diseño de estructura

La estructura no se modifica. Solo se incluye una función adicional en la clase Departamento que permitirá determinar la validez de los datos que se han capturado. Este método será utilizado al momento de grabar los datos en el archivo y al momento de presentar la información en pantalla.

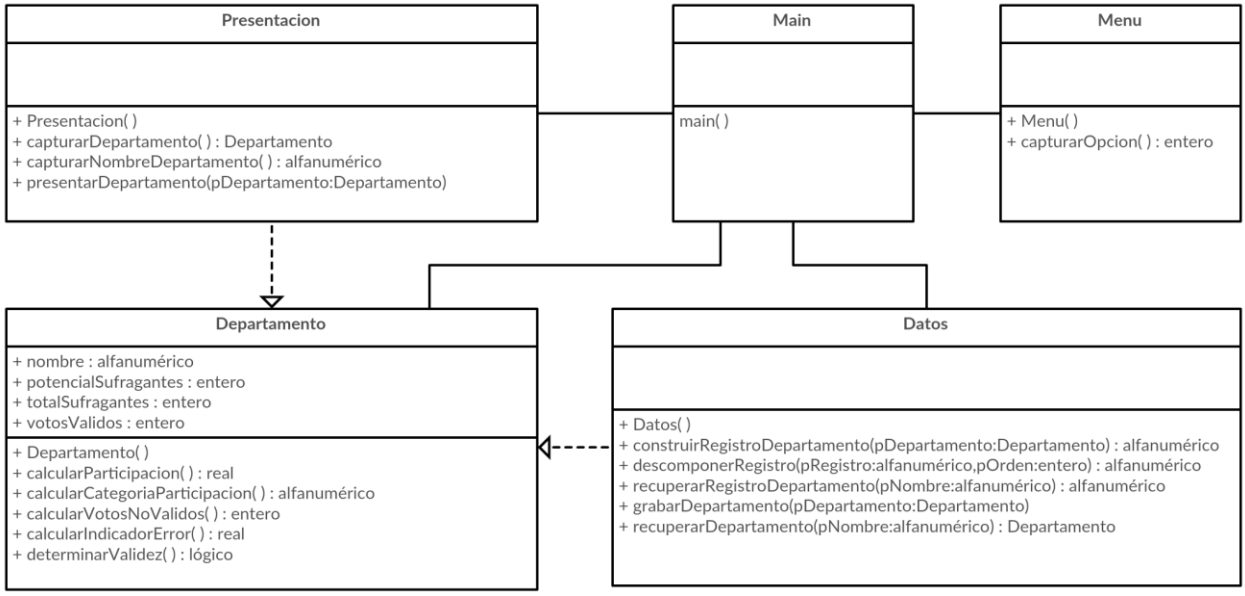


Figura 9.2 Diagrama de clases para el Prototipo 8

9.1.2.3 Diseño de algoritmos

9.1.2.3.1 Clase Departamento - Departamento( )

Descripción: Constructor que inicializa los atributos con valores neutros.

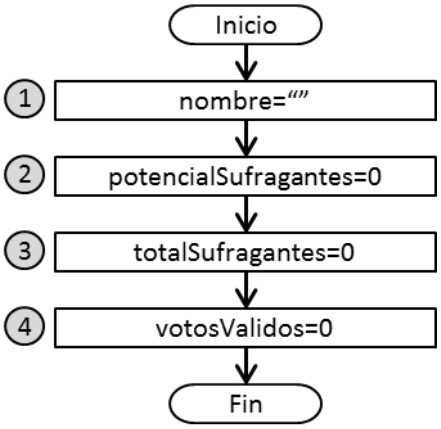


Figura 9.3 Diagrama de flujo para el método Departamento()

9.1.2.3.2 Clase Departamento – calcularParticipacion( ): real

Descripción: Método de cálculo para obtener el indicador de participación electoral con base en el potencial y el total de sufragantes del departamento.

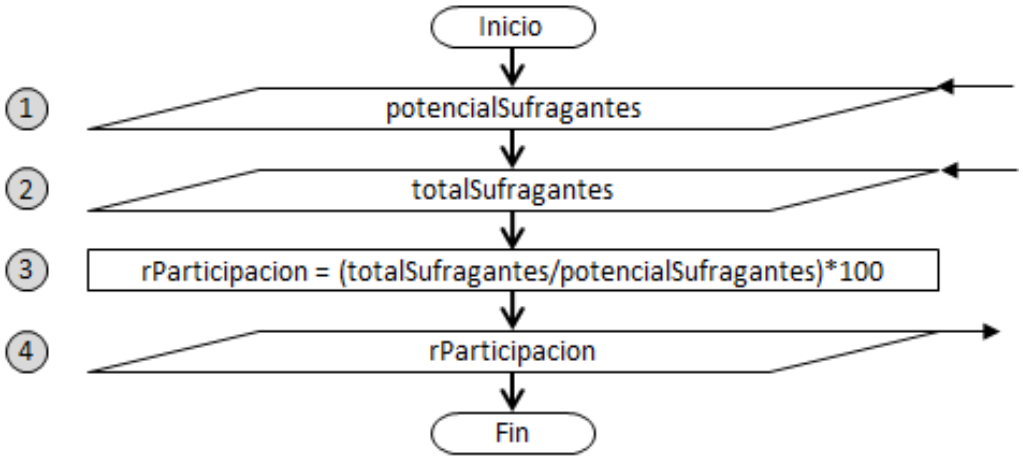


Figura 9.4 Diagrama de flujo para el método calcularParticipacion()

9.1.2.3.3 Clase Departamento – calcularCategoriaParticipacion( ): cadena

Descripción: Método de cálculo para determinar la categoría de participación electoral del departamento, con base en el indicador de participación calculado por el método anterior.

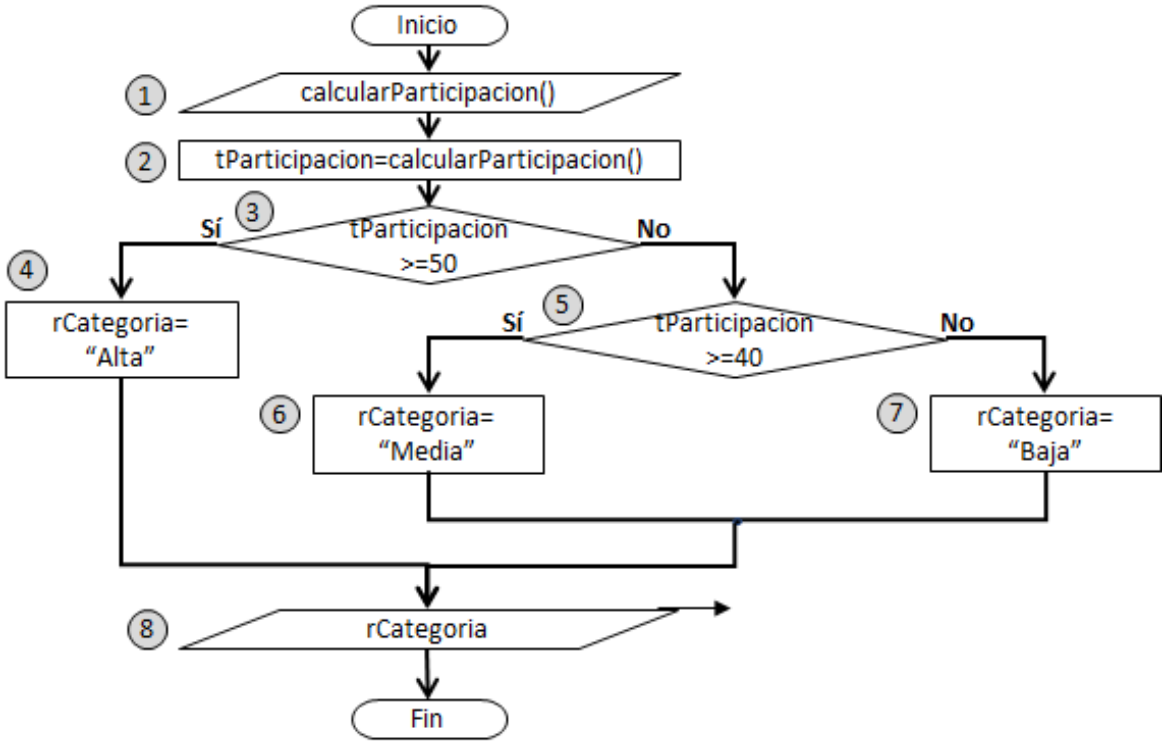


Figura 9.5 Diagrama de flujo para el método calcularCategoriaParticipacion()

9.1.2.3.4 Clase Departamento - calcularVotosNoValidos( ):entero

Descripción: Calcular la cantidad de votos no válidos con base en el total de sufragantes y los votos válidos en el departamento

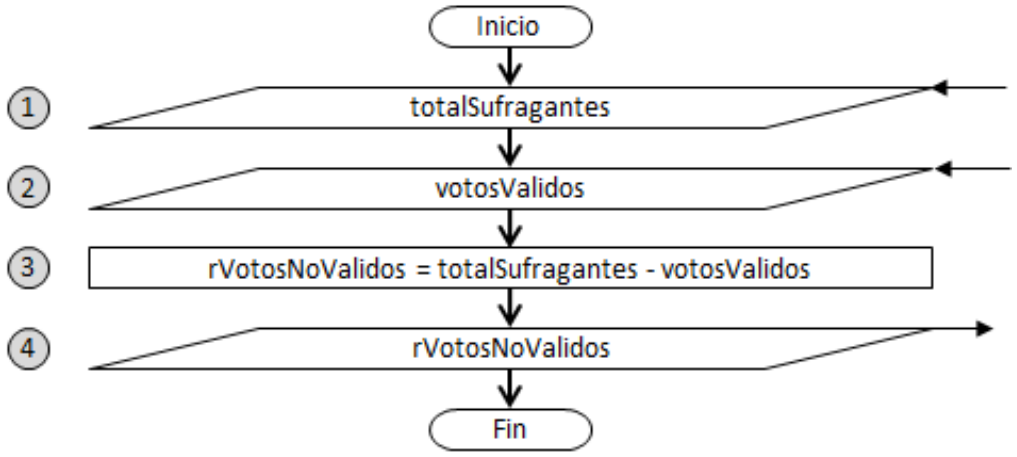


Figura 9.6 Diagrama de flujo para el método calcularVotosNoValidos()

9.1.2.3.5 Clase Departamento - calcularIndicadorError( ):real

Descripción: Calcular el indicador de participación electoral con base en el potencial y el total de sugfagantes del departamento

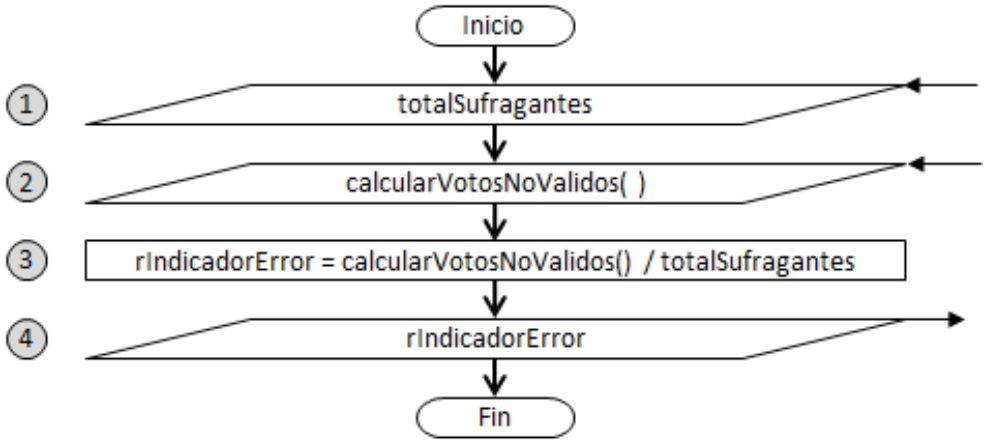


Figura 9.7 Diagrama de flujo para el método calcularIndicadorError()

9.1.2.3.6 Clase Departamento - determinarValidez( ):lógico

Descripción: Determinar la consistencia interna de los valores de los atributos, revisando que se cumplan todas las condiciones de validación.

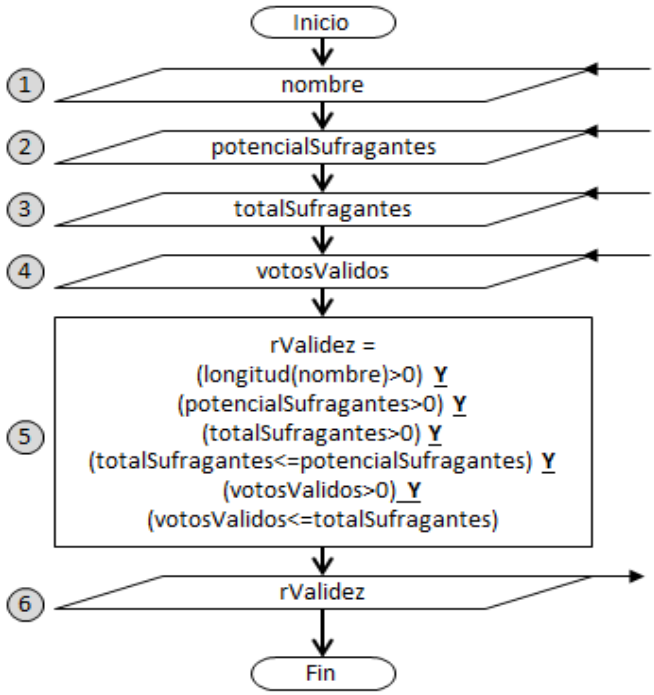


Figura 9.8 Diagrama de flujo para el método determinarValidez()

Tabla 9.5 Descripción del algoritmo para el método determinarValidez()

1	Se requiere como dato de entrada *nombre,* se cumple por tener este dato como atributo propio.
2	Se requiere como dato de entrada *potencialSufragantes,* se cumple por tener este dato como atributo propio.
3	Se requiere como dato de entrada *totalSufragantes,* se cumple por tener este dato como atributo propio.
4	Se requiere como dato de entrada *votosValidos,* se cumple por tener este dato como atributo propio.
5	Se calcula el resultado lógico de la comparación de cada una de las condiciones de validación para los datos de *nombre, potencialSufragantes, totalSufragantes* y *votosValidos*. Almacena el resultado de la operación en la variable *rValidez*.
6	Se genera como dato de salida o respuesta del método el valor de la variable *rValidez*.

9.1.2.3.7 Clase Presentacion - capturarDepartamento( ):Departamento

Descripción: Obtiene del usuario cada uno de los valores para los atributos, y entrega como respuesta el objeto de la clase Departamento con los valores de los atributos asignados.

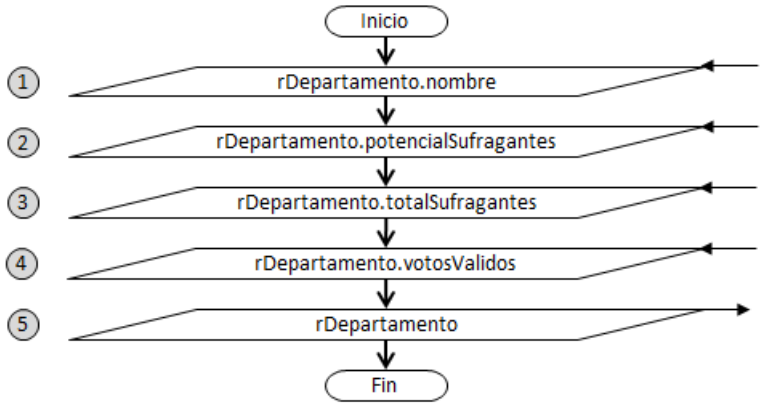


Figura 9.9 Diagrama de flujo para el método capturarDepartamento()

9.1.2.3.8 Clase Presentacion - capturarNombreDepartamento( ):cadena

Descripción: Obtiene del usuario un nombre de departamento, y lo entrega como respuesta al objeto que haya hecho la solicitud, es decir, que haya enviado el mensaje para la ejecución del método.

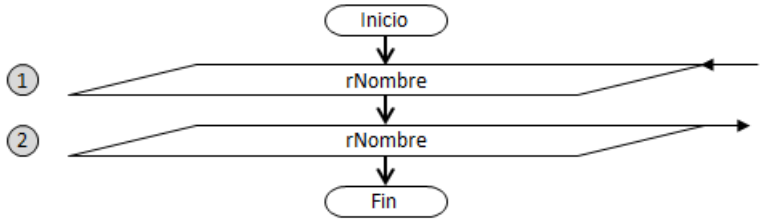


Figura 9.10 Diagrama de flujo para el método capturarNombreDepartamento()

9.1.2.3.9 Clase Presentacion - presentarDepartamento(pDepartamento:Departamento)

Descripción: Recibe un mensaje con un objeto de la clase Departamento como parámetro. Si el objeto cumple con las condiciones de validación, construye un mensaje al que agrega cada uno de los valores de los atributos del objeto recibido, así como el resultado de los métodos de cálculo del indicador de participación electoral, de la categoría de participación electoral, de la cantidad de votos no válidos y del indicador de error. En caso de que el objeto no cumpla con las condiciones de validación, construye un mensaje de error. Por último muestra al usuario el mensaje construido.

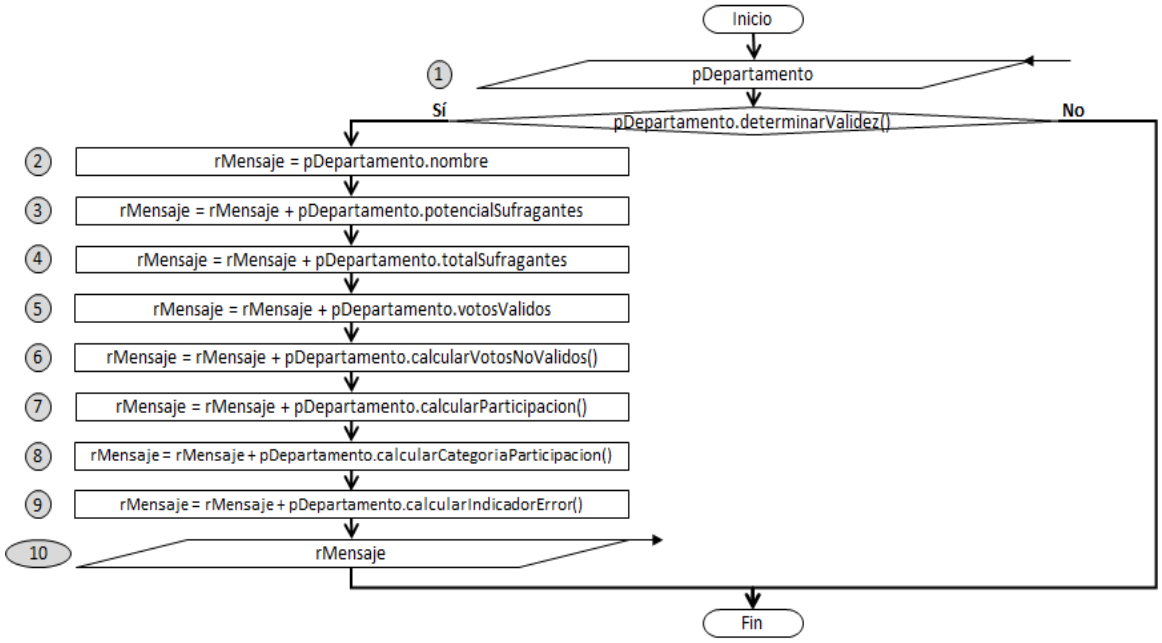


Figura 9.11 Diagrama de flujo para el método presentarDepartamento()

Tabla 9.6 Descripción del algoritmo para el método presentarDepartamento()

1	Se requiere como dato de entrada el objeto de tipo Departamento. En este caso se recibe el objeto como mensaje.
2	Se revisa la validez del objeto de tipo Departamento.
3	Si el objeto SI es válido: Asigna el nombre del departamento del objeto en la variable rMensaje
4	Concatena en la variable rMensaje el potencial de sufragantes del departamento almacenado en el objeto
5	Concatena en la variable rMensaje el total de sufragantes del departamento almacenado en el objeto
6	Concatena en la variable rMensaje el total de votos válidos del departamento almacenado en el objeto
7	Concatena en la variable rMensaje la respuesta del método calcularParticipacion() del objeto.
8	Concatena en la variable rMensaje la respuesta del método calcularCategoriaParticipacion() del objeto.
9	Concatena en la variable rMensaje la respuesta del método calcularVotosNoValidos() del objeto.
10	Concatena en la variable rMensaje la respuesta del método calcularIndicadorError() del objeto.
11	Si el objeto NO es válido: Asigna un mensaje de error “Datos inconsistentes. No se pueden presentar” en la variable rMensaje
12	Se genera como dato de salida el mensaje armado

9.1.2.3.10 Clase Menu - capturarOpcion(): entero

La clase Menu mantiene su especificación en el nuevo prototipo: presenta al usuario un mensaje con las opciones disponibles en la aplicación, y obtiene de dicho usuario el número correspondiente a la opción seleccionada. Las especificaciones de diseño por lo tanto son las mismas del prototipo anterior. Ver sección 8.4.2.3.9 del capítulo anterior.

9.1.2.3.11 Clase Datos - construirRegistroDepartamento(pDepartamento:Departamento): cadena

Descripción: Recibe un mensaje con un objeto de la clase Departamento como parámetro. Construye un registro, es decir una cadena de caracteres, a la que agrega cada uno de los valores de los atributos del objeto separados entre sí por un signo de coma. Por último lo entrega el registro como respuesta al objeto que haya hecho la solicitud, es decir, que haya invocado ejecución del método.

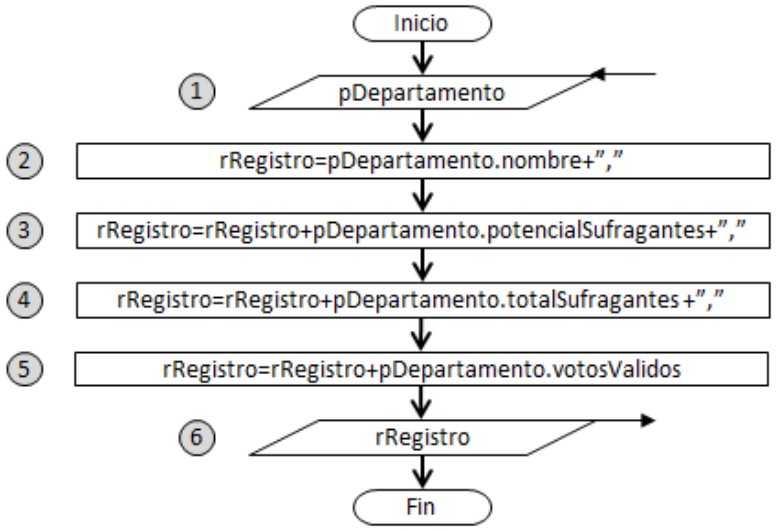


Figura 9.12 Diagrama de flujo para el método construirRegistroDepartamento ()

9.1.2.3.12 Clase Datos - descomponerRegistro (pRegistro:cadena, pOrden: entero):cadena

Descripción: Recibe un mensaje con dos parámetros; el primero de estos es un registro o cadena de caracteres que contiene valores separados entre sí por signos de coma; el segundo, un número ordinal que indica el dato que se debe extraer del registro y entregar como respuesta final. El método hace un recorrido secuencial por los caracteres del registro, y cuando ubica el comienzo del dato solicitado, comienza a agregar cada caracter en una variable que al final entrega como respuesta al objeto que haya requerido la ejecución del método.

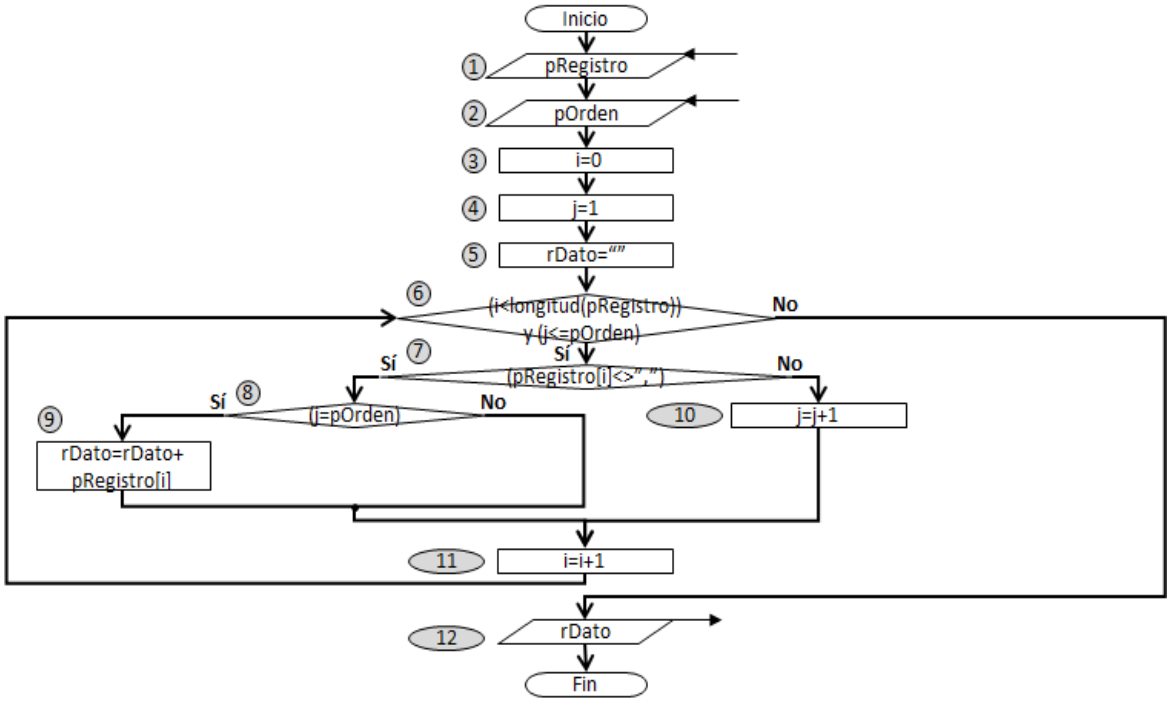


Figura 9.13 Diagrama de flujo para el método descomponerRegistro()

9.1.2.3.13 Clase Datos - grabarDepartamento(pDepartamento:Departamento

Descripción: Recibe un mensaje con un objeto de la clase Departamento como parámetro. Si el objeto cumple las condiciones de validación, abre un archivo y a continuación invoca el método de construcción del registro, cuya respuesta es grabada en el archivo que finalmente es cerrado; en caso contrario, omite la grabación del departamento.

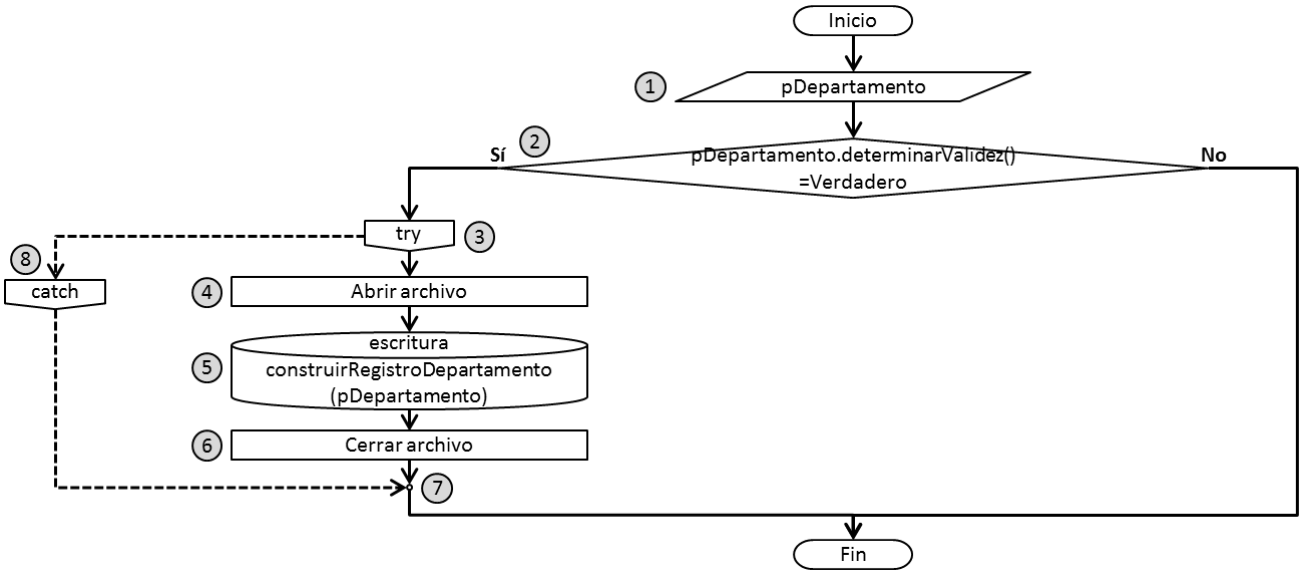


Figura 9.14 Diagrama de flujo para el método grabarDepartamento ()

Tabla 9.7 Descripción del algoritmo para el método grabarDepartamento ()

1	Se requiere como dato de entrada el objeto de tipo Departamento. En este caso se recibe el objeto como mensaje.
2	Se revisa la validez del objeto de tipo Departamento.
3	Si el objeto SI es válido: Inicia la estructura de manejo de errores en caso de poder abrir el archivo para su escritura
4	Abre el archivo de texto para escritura
5	Escribe en el archivo la respuesta del método construirRegistroDepartamento()
6	Cierra el archivo de texto para asegurar la escritura
7	Inicia la estructura para la ejecución de acciones en caso de fallo por no poder abrir el archivo para escritura

9.1.2.3.14 Clase Datos - recuperarRegistroDepartamento(pNombre:cadena):cadena

Descripción: Recibe un mensaje con una cadena de caracteres como parámetro, que contiene un nombre de departamento a localizar en el archivo. Abre un archivo para lectura y a continuación lee uno a uno los registros hasta encontrar el que contiene los datos del departamento requerido, para lo cual solicita la descomposición del registro y la obtención del primer dato que corresponde al nombre. Por último entrega como respuesta el registro que contenga el nombre buscado.

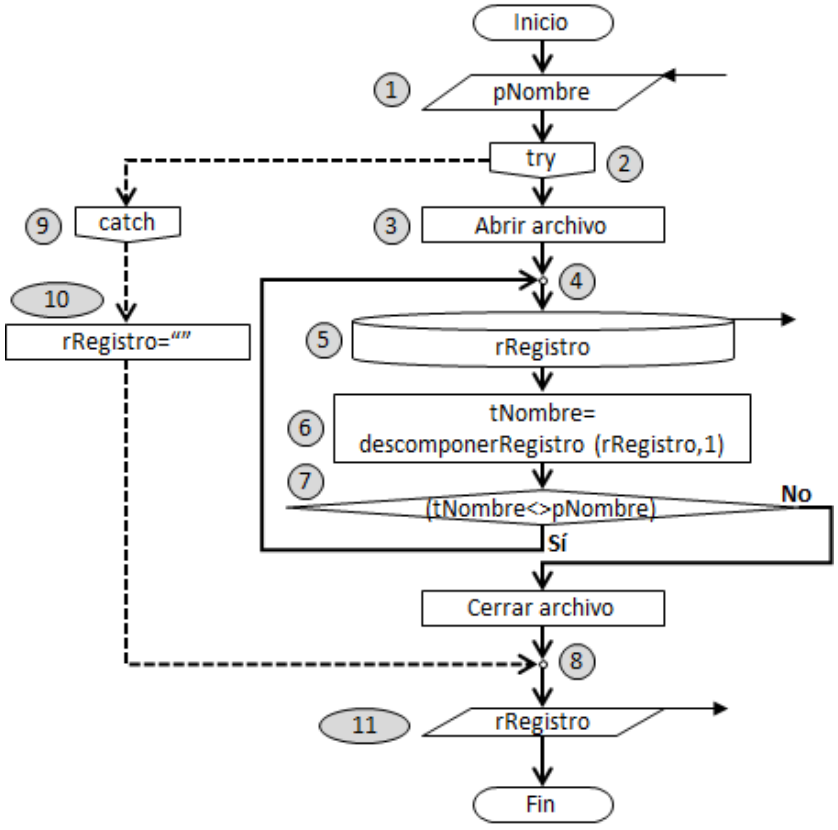


Figura 9.15 Diagrama de flujo para el método recuperarRegistroDepartamento()

9.1.2.3.15 Clase Datos - recuperarDepartamento(pNombre:cadena):Departamento

Descripción: Recibe un mensaje con una cadena de caracteres como parámetro, que contiene un nombre de departamento a localizar en el archivo. Solicita la ejecución del método de recuperación de registro mediante el nombre dado, y una vez obtenida la respuesta de aquel método, toma el registro hallado y solicita su descomposición en cuatro partes. Por último, entrega como respuesta el objeto de la clase Departamento correspondiente al nombre dado como criterio de búsqueda.

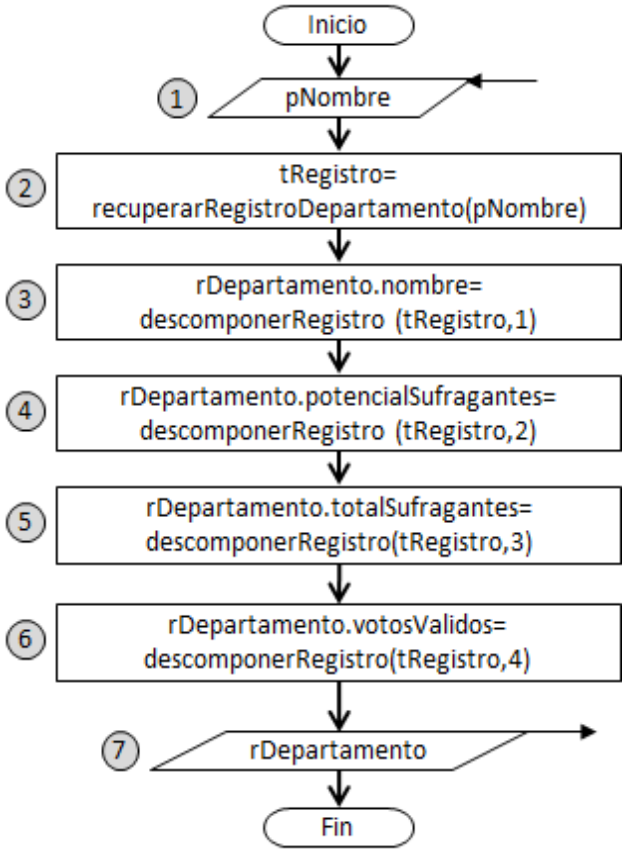


Figura 9.16 Diagrama de flujo para el método recuperarDepartamento()

9.1.2.3.16 Clase Main – main( )

Descripción: Coordina las acciones de las demás clases. Solicita a un objeto de la clase Menu que reciba de parte del usuario la opción requerida por él, que almacena en una variable.

La clase Main mantiene su especificación en el nuevo prototipo, la cual está ampliamente descrita en la sección 8.4.2.3.15 del capítulo anterior.

9.1.3 Implementación

La estructura del proyecto no cambia, solo se incluye un método en la clase Departamento para la **validación de los datos**, que es reutilizado para su presentación y su grabación.

Tabla 9.8 Codificación de la clase *Departamento*

```
1  package proyecto;
2
3  public class Departamento {
4     public String nombre;
5     public int potencialSufragantes;
6     public int totalSufragantes;
7     public int votosValidos;
8
9     public Departamento(){
10       nombre="";
11       potencialSufragantes=0;
12       totalSufragantes=0;
13       votosValidos=0;
14    }
15
16    public double calcularParticipacion(){
17       double rParticipacion;
18       rParticipacion=((double)totalSufragantes/potencialSufragantes)*100;
19       return rParticipacion;
20    }
21
22    public String calcularCategoriaParticipacion(){
23       double tParticipacion;
24       String rCategoria;
25       tParticipacion=calcularParticipacion();
26       if (tParticipacion>=50){
27          rCategoria="Alta";
28       }
29       else{
30          if (tParticipacion>=40){
31             rCategoria="Media";
32          }
33          else{
34             rCategoria="Baja";
35          }
36       }
37       return rCategoria;
38    }
39
40    public int calcularVotosNoValidos(){
41       int rVotosNoValidos;
42       rVotosNoValidos=totalSufragantes-votosValidos;
43       return rVotosNoValidos;
44    }
45
```

```
46    public double calcularIndicadorError(){
47       double rIndicadorError;
48       rIndicadorError=(double)calcularVotosNoValidos()/totalSufragantes;
49       return rIndicadorError;
50    }
51
52    public boolean determinarValidez(){
53       boolean rValidez;
54       rValidez=((nombre.length()>0) &&
55             (potencialSufragantes>0) &&
56             (totalSufragantes>0) &&
57             (totalSufragantes<=potencialSufragantes) &&
58             (votosValidos>0) &&
59             (votosValidos<=totalSufragantes));
60       return rValidez;
61    }
62 }
```

Tabla 9.9 Codificación de la clase *Presentacion*

```
1  package proyecto;
2  import javax.swing.JOptionPane;
3
4  public class Presentacion {
5     public Presentacion(){
6
7     }
8
9     public Departamento capturarDepartamento(){
10       Departamento rDepartamento;
11       rDepartamento=new Departamento();
12       rDepartamento.nombre=JOptionPane.showInputDialog("Nombre: ");
13       rDepartamento.potencialSufragantes=Integer.parseInt(JOptionPane.showInputDialog("Potencial de
      sufragantes: "));
14       rDepartamento.totalSufragantes=Integer.parseInt(JOptionPane.showInputDialog("Total de
      sufragantes: "));
15       rDepartamento.votosValidos=Integer.parseInt(JOptionPane.showInputDialog("Votos válidos: "));
16       return rDepartamento;
17    }
18
19    public String capturarNombreDepartamento(){
20       String rNombre;
21       rNombre=JOptionPane.showInputDialog("Departamento: ");
22       return rNombre;
23    }
24
25    public void presentarDepartamento(Departamento pDepartamento){
26       String rMensaje;
27       if (pDepartamento.determinarValidez()){
28          rMensaje="Departamento: "+pDepartamento.nombre;
29          rMensaje=rMensaje+"\nPotencial de sufragantes: "+pDepartamento.potencialSufragantes;
```

```
30          rMensaje=rMensaje+"\nTotal de sufragantes: "+pDepartamento.totalSufragantes;
31          rMensaje=rMensaje+"\nVotos válidos: "+pDepartamento.votosValidos;
32          rMensaje=rMensaje+"\nIndicador de participación: "+pDepartamento.calcularParticipacion()+" %";
33          rMensaje=rMensaje+"\nCategoría según participación:
    "+pDepartamento.calcularCategoriaParticipacion();
34          rMensaje=rMensaje+"\nVotos no válidos: "+pDepartamento.calcularVotosNoValidos();
35          rMensaje=rMensaje+"\nIndicador de error: "+pDepartamento.calcularIndicadorError();
36       }
37       else{
38          rMensaje="Datos inconsistentes. No se pueden presentar";
39       }
40       JOptionPane.showMessageDialog(null, rMensaje);
41    }
42 }
```

Tabla 9.10 Codificación de la clase *Menu*

```
1  package proyecto;
2  import javax.swing.JOptionPane;
3
4  public class Menu {
5     public Menu(){
6
7     }
8
9     public int capturarOpcion(){
10       String tMensaje;
11       int rOpcion;
12       tMensaje="1. Registrar departamento\n"+
13             "2. Consultar departamento\n"+
14             "3. Salir\n";
15       rOpcion=Integer.parseInt(JOptionPane.showInputDialog(tMensaje));
16       return rOpcion;
17    }
18 }
```

Tabla 9.11 Codificación de la clase *Datos*

```
1  package proyecto;
2  import java.io.*;
3
4  public class Datos {
5     public Datos(){
6
7     }
8
9     public String construirRegistroDepartamento(Departamento pDepartamento){
10       String rRegistro;
11       rRegistro=pDepartamento.nombre+",";
```

```
12      rRegistro=rRegistro+pDepartamento.potencialSufragantes+",";
13      rRegistro=rRegistro+pDepartamento.totalSufragantes+",";
14      rRegistro=rRegistro+pDepartamento.votosValidos;
15      return rRegistro;
16    }
17
18    public String descomponerRegistro(String pRegistro, int pOrden){
19      int i,j;
20      String rDato;
21      i=0;
22      j=1;
23      rDato="";
24      while ((i<pRegistro.length()) && (j<=pOrden)) {
25        if (pRegistro.substring(i,i+1).compareTo(",")!=0){
26          if (j==pOrden){
27            rDato=rDato+pRegistro.substring(i,i+1);
28          }
29        }
30        else{
31          j=j+1;
32        }
33        i=i+1;
34      }
35      return rDato;
36    }
37
38    public String recuperarRegistroDepartamento(String pNombre){
39      FileReader fileReader;
40      BufferedReader bufferedReader;
41      String rRegistro,tNombre;
42      try{
43        fileReader=new FileReader("departamentos.txt");
44        bufferedReader=new BufferedReader(fileReader);
45        do{
46          rRegistro=bufferedReader.readLine();
47          tNombre=descomponerRegistro(rRegistro, 1);
48        } while (tNombre.compareToIgnoreCase(pNombre)!=0);
49        fileReader.close();
50      }
51      catch (Exception e){
52        rRegistro="";
53      }
54      return rRegistro;
55    }
56
57    public Departamento recuperarDepartamento(String pNombre){
58      String tRegistro;
59      Departamento rDepartamento;
60      rDepartamento=new Departamento();
61      tRegistro=recuperarRegistroDepartamento(pNombre);
62      rDepartamento.nombre=descomponerRegistro(tRegistro,1);
63      rDepartamento.potencialSufragantes=Integer.parseInt(descomponerRegistro(tRegistro,2));
64      rDepartamento.totalSufragantes=Integer.parseInt(descomponerRegistro(tRegistro,3));
```

```
65        rDepartamento.votosValidos=Integer.parseInt(descomponerRegistro(tRegistro,4));
66        return rDepartamento;
67     }
68
69     public void grabarDepartamento(Departamento pDepartamento){
70        FileWriter fileWriter;
71        PrintWriter printWriter;
72        if (pDepartamento.determinarValidez()){
73           try{
74              fileWriter=new FileWriter("departamentos.txt",true);
75              printWriter=new PrintWriter(fileWriter);
76              printWriter.println(construirRegistroDepartamento(pDepartamento));
77              fileWriter.close();
78           }
79           catch(Exception e){
80
81           }
82        }
83     }
84  }
```

Tabla 9.12 Codificación de la clase *Main*

```
1   package proyecto;
2
3   public class Main {
4
5      public static void main(String[] args) {
6         Departamento departamento;
7         Presentacion presentacion;
8         Menu menu;
9         Datos datos;
10        int opcion;
11        String parametro;
12        presentacion=new Presentacion();
13        menu=new Menu();
14        datos=new Datos();
15        opcion=0;
16        do {
17           opcion=menu.capturarOpcion();
18           switch (opcion){
19              case 1:{
20                 departamento=presentacion.capturarDepartamento();
21                 datos.grabarDepartamento(departamento);
22                 presentacion.presentarDepartamento(departamento);
23                 break;
24              }
25              case 2:{
26                 parametro=presentacion.capturarNombreDepartamento();
27                 departamento=datos.recuperarDepartamento(parametro);
28                 presentacion.presentarDepartamento(departamento);
```

```
29                  break;
30              }
31          }
32      } while (opcion<=2);
33  }
34  }
```

9.2 Ocultamiento de los atributos – Prototipo 9

9.2.1 Análisis

9.2.1.1 Alcance

El prototipo anterior solo hace una validación conjunta de todos los atributos, pero permite temporalmente que algunos de estos tengan valores inconsistentes. Además de la validación general de la consistencia de los datos, el prototipo 9 debe incorporar validaciones previas a la asignación de valores a los atributos, y protección de estos mediante el **ocultamiento** de los atributos. La modificación del alcance en este prototipo no es visible para el usuario, pero sí es muy importante para proteger los datos y para disminuir los errores en el ingreso de la información.

9.2.1.2 Definiciones

Se mantienen las mismas definiciones. Para integrar las reglas de validación, se añade a la tabla de definiciones las condiciones requeridas al ingreso o al momento de realizar el cálculo.

Tabla 9.13 Definiciones aplicables en desarrollo del prototipo 09.

	Definición	Tipo de dato	Condiciones de validación
Departamento	Cada una de las circunscripciones territoriales de primer nivel en que se divide el país para propósitos electorales	Objeto	
Nombre	Identificación nominal del departamento	Cadena de caracteres	Valor ingresado. El nombre del departamento no puede quedar vacío.
Potencial de sufragantes	Cantidad de personas habilitadas en un departamento para votar en un proceso electoral	Número entero	Valor ingresado. El potencial de sufragantes debe ser mayor que cero.
Total de sufragantes	Cantidad de personas que acudieron a votar en un departamento en un proceso electoral	Número entero	Valor ingresado. El total de sufragantes debe ser mayor que cero, pero menor o igual al potencial de sufragantes.
Indicador de participación electoral	Indicador consistente en el total de sufragantes expresado como porcentaje del potencial de sufragantes	Número real	Valor calculado. $Indicador\ de\ participación\ electoral = \frac{Total\ de\ sufragantes}{Potencial\ de\ sufragantes} * 100$

Categoría de participación electoral	Categoría a la que pertenece el departamento, dependiendo del porcentaje de participación electoral.	Cadena de caracteres	Valor calculado. Si el porcentaje es mayor o igual a 50%: alto Si el porcentaje es menor a 50% pero mayor o igual a 40%: medio Si el porcentaje es menor a 40%: bajo
Votos válidos	Cantidad de votos que indican de forma clara la voluntad del sufragante	Número entero	Valor ingresado. La cantidad de votos válidos debe ser mayor que cero, pero menos o igual al total de sufragantes.
Votos no válidos	Cantidad de votos que NO indican de forma clara la voluntad del sufragante.	Número entero	Valor calculado. $Votos\ no\ válidos = Total\ de\ sufragantes - Votos\ válidos$
Indicador de error en la votación	Indicador consistente en dividir el total de votos no válidos entre el total de sufragantes.	Número real	Valor calculado. $Indicador\ de\ error\ en\ la\ votación = \frac{Votos\ no\ válidos}{Total\ de\ sufragantes}$

9.2.1.3 Requisitos

No hay modificación en los requisitos de interfaz de usuario ni en los funcionales.

9.2.1.3.1 Requisitos de interfaz de usuario

9.2.1.3.1.1 Ingreso de información en casillas de edición

Para la interacción con el usuario, la aplicación debe presentar ventanas con casillas de edición en donde pueda digitar los datos solicitados. Al solicitar los datos, el sistema deberá indicar en la misma ventana de diálogo cuál es el dato requerido.

9.2.1.3.1.2 Presentación de menú de selección de funciones

La aplicación debe presentar un menú para seleccionar una de 3 funciones: registrar la información de un departamento para su almacenamiento, consultar información de un departamento a partir de la recuperación de sus datos almacenados, o finalizar la aplicación.

9.2.1.3.2 Requisitos funcionales

9.2.1.3.2.1 Calcular el indicador de participación electoral de un departamento

La aplicación debe permitir al usuario introducir los siguientes datos básicos de un departamento: el nombre, el potencial de sufragantes, el total de sufragantes y la cantidad de votos válidos, y con estos datos debe calcular el indicador de participación electoral, los votos no válidos y el indicador de error de votación. Los datos ingresados deben cumplir las condiciones de validación requeridas.

9.2.1.3.2.2 Almacenar la información de un departamento en un archivo plano

La aplicación debe poder almacenar en un archivo plano los datos básicos ingresados de un departamento: el nombre, el potencial de sufragantes, el total de sufragantes y el total de votos válidos. Se requiere que el formato de almacenamiento sea en datos separados por punto y coma (formato csv).

9.2.1.3.2.3 Recuperar la información de un departamento en un archivo plano

La aplicación debe poder recuperar desde un archivo plano los datos básicos ingresados de un departamento: el nombre, el potencial de sufragantes, el total de sufragantes, el total de votos válidos además de los datos calculados del indicador de participación electoral, la categoría de participación a la que pertenece, los votos no válidos y el indicador de error de votación y se debe presentar al usuario. Para ello deberá preguntar por el nombre del departamento a buscar. Como restricción, se requiere el formato de almacenamiento sea en datos separados por punto y coma (formato csv).

9.2.2 Diseño

9.2.2.1 Diseño de escenarios

Se mantiene el mismo diseño de escenarios descrito en la sección 9.1.2.1 del presente capítulo.

9.2.2.2 Diseño de estructura

La estructura no se modifica. Lo que se debe incluir es una forma de asegurar y proteger los datos de los objetos. En el momento que el usuario ingresa, por ejemplo, un valor negativo en una variable numérica, el valor ingresado no debe ser tenido en cuenta sino que debe mantenerse el dato que ha sido inicializado desde su constructor. No obstante, deben protegerse los datos no solo desde el punto de vista del usuario, sino también desde el punto de vista del programador. Un programador, por ejemplo, podría alterar directamente el valor de un atributo de la clase principal, desde cualquier parte.

Existe una propiedad muy conocida en el ámbito de la programación orientada a objetos y se le conoce como **encapsulamiento**. Esta refiere dos características muy interesantes: la primera se le puede llamar con el nombre de **apropiación** o **encapsulamiento** propiamente dicho, y se refiere a que, debido a que se puede crear una clase con sus atributos y métodos, y se puede crear un objeto a partir de la instancia de una clase, el propio objeto tiene la capacidad de utilizar todas sus características internas para él mismo. Es decir, atributos y métodos se encapsulan en una sola entidad, y ella accede a sus propios datos y funciones. Suena obvio, sin embargo esto lleva a la segunda característica: el objeto, además, tiene la capacidad de permitir o no que otros objetos externamente cuenten con sus características internas. Esta segunda característica se le conoce con el nombre de **ocultamiento**, y permite mantener un nivel de seguridad y protección de sus datos.

Esta característica de seguridad implica que, si se requiere proteger la información de los atributos de una clase, es necesario pensar en crear en tres aspectos: la primera es la

diferenciación en el uso de sus propios atributos o métodos a través de la palabra ***this***, la visibilidad (ya sea pública, protegida o privada) de sus atributos y métodos para que puedan ser accedidos desde métodos de otras clases, y la creación de métodos públicos especiales para registrar u obtener los valores de los atributos de las clases de negocio.

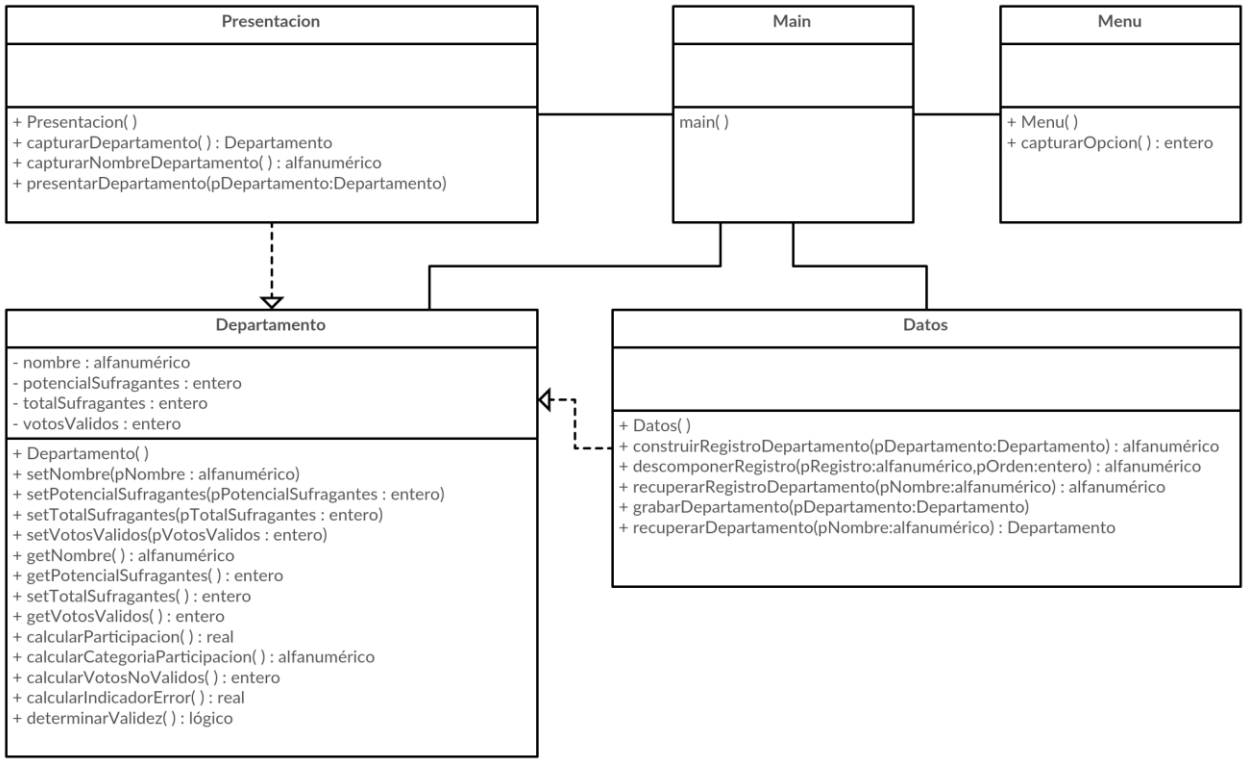


Figura 9.17 Diagrama de clases para el Prototipo 9

9.2.2.3 Diseño de algoritmos

9.2.2.3.1 Clase Departamento - Departamento( )

Descripción: Constructor que inicializa los atributos con valores neutros.

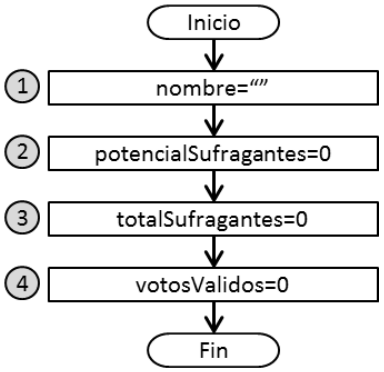


Figura 9.18 Diagrama de flujo para el método Departamento()

9.2.2.3.2 Clase Departamento - setNombre(pNombre : alfanumérico)

Descripción: Método que asigna al atributo nombre el valor dado como parámetro, previa comprobación de su validez.

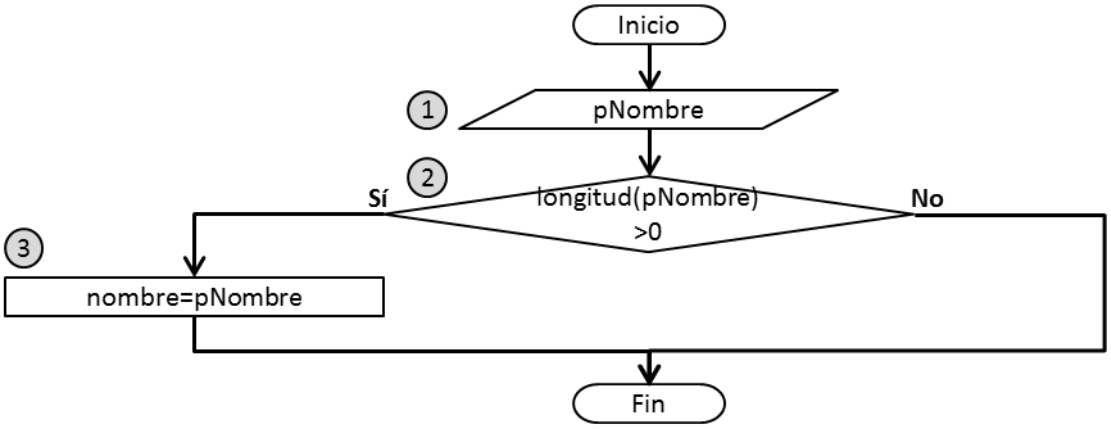


Figura 9.19 Diagrama de flujo para el método setNombre()

Tabla 9.14 Descripción del algoritmo para el método setNombre()

1	Se requiere como dato de entrada *pNombre.* En este caso se recibe la variable como mensaje.
2	Se revisa si la longitud del texto de *pNombre* es mayor a 0
3	Si se cumple la condición: Asigna al atributo *nombre* el valor de la variable recibida *pNombre*

9.2.2.3.3 Clase Departamento - setPotencialSufragantes(pPotencialSufragantes : entero)

Descripción: Método que asigna al atributo potencialSufragantes el valor dado como parámetro, previa comprobación de su validez.

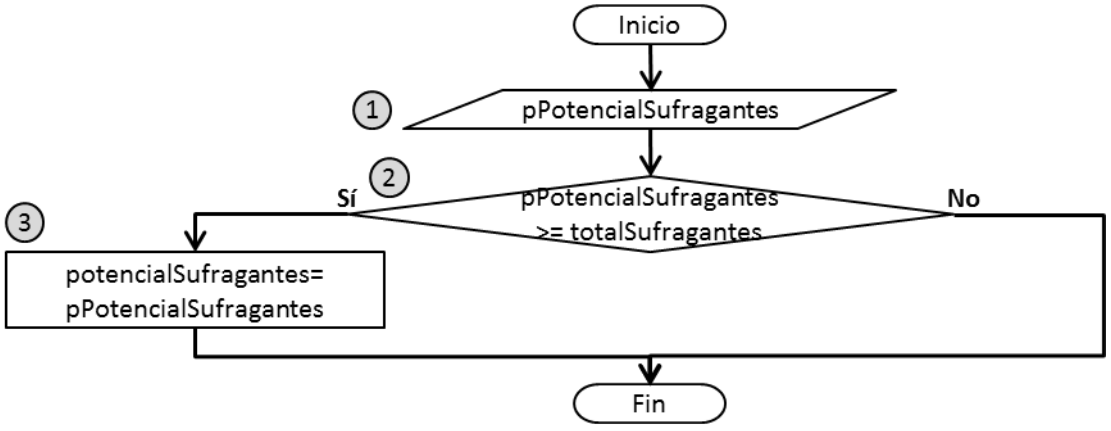


Figura 9.20 Diagrama de flujo para el método setPotencialSufragantes()

Tabla 9.15 Descripción del algoritmo para el método setPotencialSufragantes()

1	Se requiere como dato de entrada *pPotencialSufragantes.* En este caso se recibe la variable como mensaje.
2	Se revisa si el valor de la variable *pPotencialSufragantes* es mayor o igual al valor del atributo *totalSufragantes*
3	Si se cumple la condición: Asigna al atributo *potencialSufragantes* el valor de la variable recibida *pPotencialSufragantes*

9.2.2.3.4 Clase Departamento - setTotalSufragantes(pTotalSufragantes : entero)

Descripción: Método que asigna al atributo totalSufragantes el valor dado como parámetro, previa comprobación de su validez.

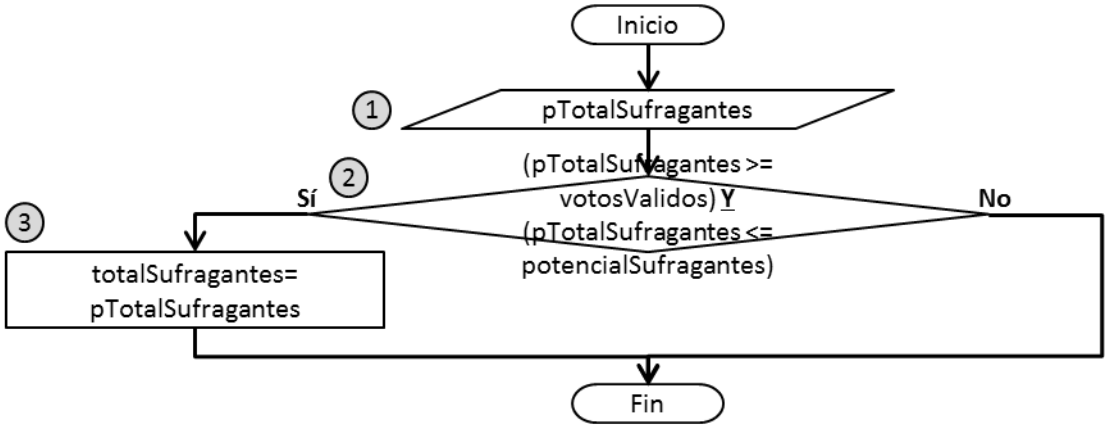


Figura 9.21 Diagrama de flujo para el método setTotalSufragantes()

Tabla 9.16 Descripción del algoritmo para el método setTotalSufragantes()

1	Se requiere como dato de entrada *pTotalSufragantes.* En este caso se recibe la variable como mensaje.
2	Se revisa si el valor de la variable *pTotalSufragantes* es mayor o igual al valor del atributo *votosValidos* y es menor o igual al valor del atributo *potencialSufragantes*
3	Si se cumple la condición: Asigna al atributo *totalSufragantes* el valor de la variable recibida *pTotalSufragantes*

9.2.2.3.5 Clase Departamento - setVotosValidos(pVotosValidos : entero)

Descripción: Método que asigna al atributo votosValidos el valor dado como parámetro, previa comprobación de su validez.

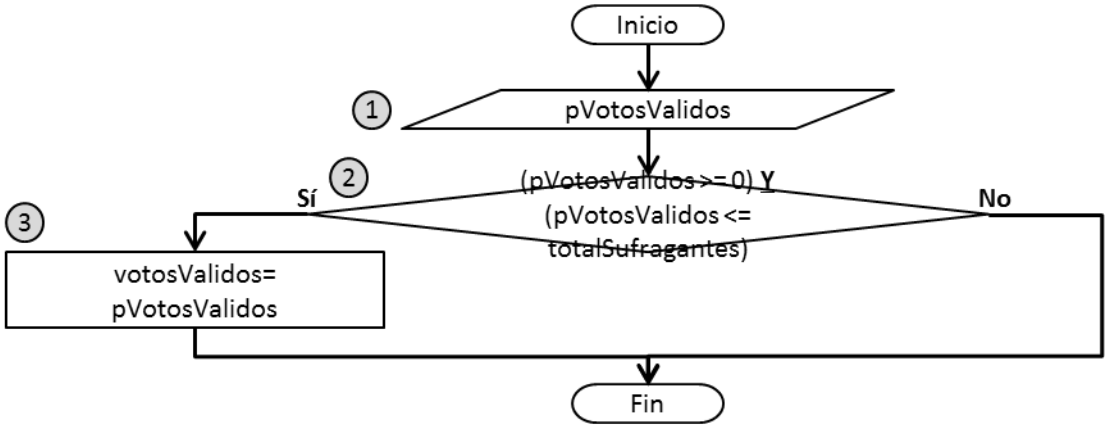


Figura 9.22 Diagrama de flujo para el método setVotosValidos()

Tabla 9.17 Descripción del algoritmo para el método setVotosValidos()

1	Se requiere como dato de entrada *pVotosValidos.* En este caso se recibe la variable como mensaje.
2	Se revisa si el valor de la variable *pVotosValidos* es mayor o igual al 0 y es menor o igual al valor del atributo *totalSufragantes*
3	Si se cumple la condición: Asigna al atributo *votosValidos* el valor de la variable recibida *pVotosValidos*

9.2.2.3.6 Clase Departamento - getNombre( ):alfanumérico

Descripción: Entrega como respuesta el valor del atributo nombre.

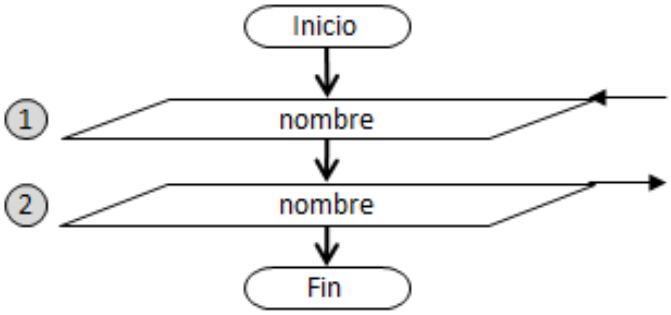


Figura 9.23 Diagrama de flujo para el método getNombre()

Tabla 9.18 Descripción del algoritmo para el método getNombre()

1	Se requiere como dato de entrada *nombre,* se cumple por tener este dato como atributo propio.
2	Se genera como dato de salida el valor del atributo *nombre*. Para ello se realiza como respuesta de este método.

9.2.2.3.7 Clase Departamento - getPotencialSufragantes( ):entero

Descripción: Entrega como respuesta el valor del atributo potencialSufragantes.

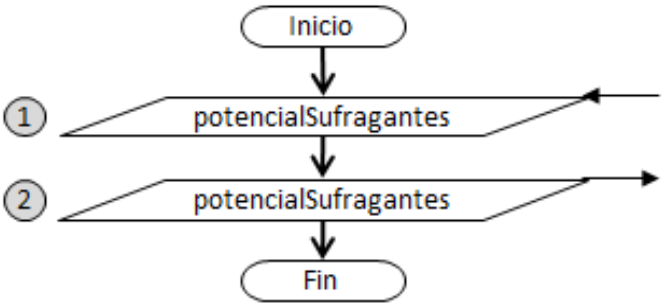


Figura 9.24 Diagrama de flujo para el método getPotencialSufragantes()

Tabla 9.19 Descripción del algoritmo para el método getPotencialSufragantes()

1	Se requiere como dato de entrada *potencialSufragantes,* se cumple por tener este dato como atributo propio.
2	Se genera como dato de salida el valor del atributo *potencialSufragantes*. Para ello se realiza como respuesta de este método.

9.2.2.3.8 Clase Departamento - getTotalSufragantes( ):entero

Descripción: Entrega como respuesta el valor del atributo totalSufragantes.

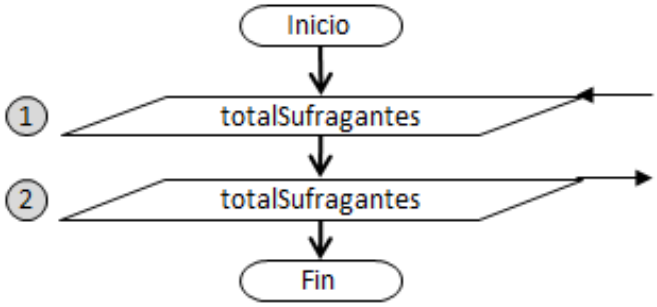


Figura 9.25 Diagrama de flujo para el método getTotalSufragantes()

Tabla 9.20 Descripción del algoritmo para el método getTotalSufragantes()

1	Se requiere como dato de entrada *totalSufragantes,* se cumple por tener este dato como atributo propio.
2	Se genera como dato de salida el valor del atributo *totalSufragantes*. Para ello se realiza como respuesta de este método.

9.2.2.3.9 Clase Departamento - getVotosValidos( ):entero

Descripción: Entrega como respuesta el valor del atributo votosValidos.

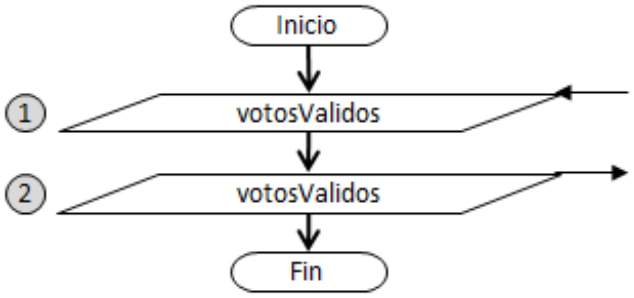


Figura 9.26 Diagrama de flujo para el método getVotosValidos()

Tabla 9.21 Descripción del algoritmo para el método getVotosValidos()

1	Se requiere como dato de entrada *votosValidos,* se cumple por tener este dato como atributo propio.
2	Se genera como dato de salida el valor del atributo *votosValidos*. Para ello se realiza como respuesta de este método.

9.2.2.3.10 Clase Departamento – calcularParticipacion( ): real

Descripción: Método de cálculo para obtener el indicador de participación electoral con base en el potencial y el total de sufragantes del departamento.

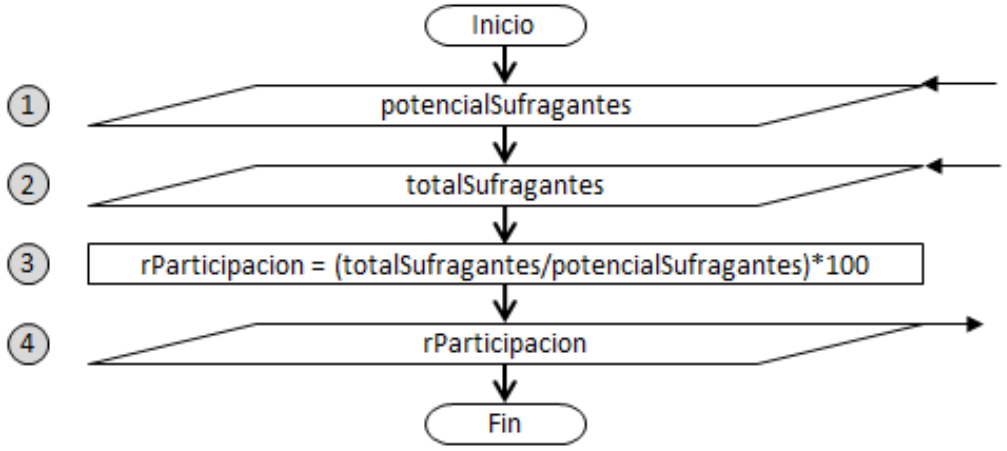


Figura 9.27 Diagrama de flujo para el método calcularParticipacion()

9.2.2.3.11 Clase Departamento – calcularCategoriaParticipacion( ): cadena

Descripción: Método de cálculo para determinar la categoría de participación electoral del departamento, con base en el indicador de participación calculado por el método anterior.

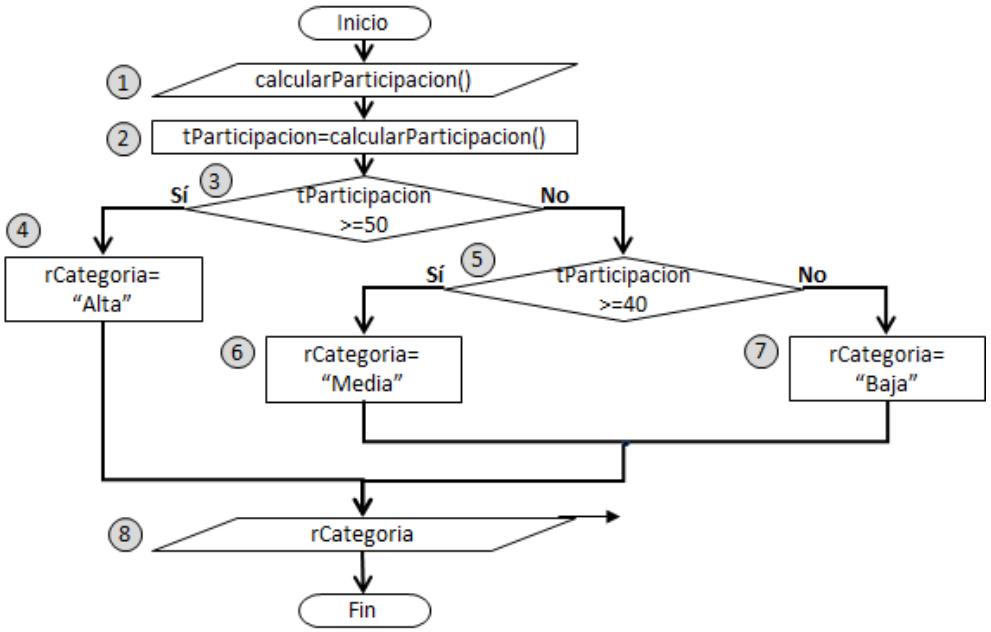


Figura 9.28 Diagrama de flujo para el método calcularCategoriaParticipacion()

9.2.2.3.12 Clase Departamento - calcularVotosNoValidos( ):entero

Descripción: Calcular la cantidad de votos no válidos con base en el total de sufragantes y los votos válidos en el departamento

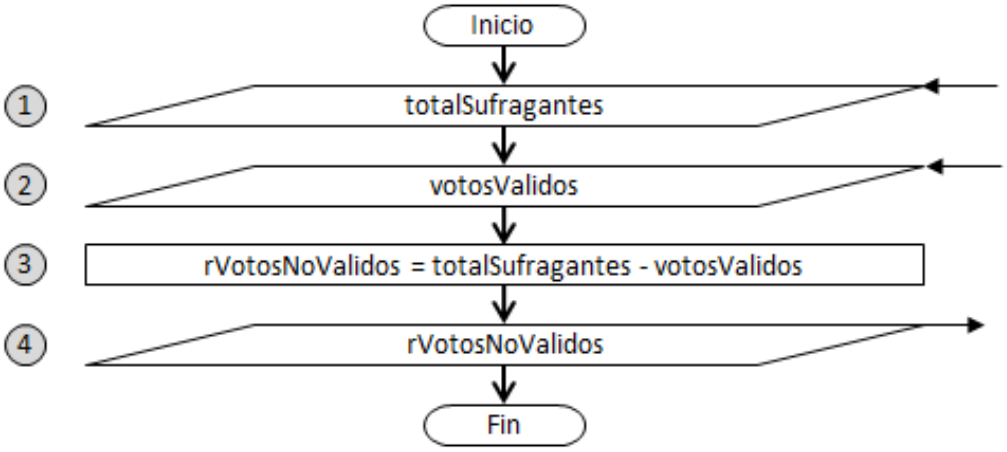


Figura 9.29 Diagrama de flujo para el método calcularVotosNoValidos()

9.2.2.3.13 Clase Departamento - calcularIndicadorError( ):real

Descripción: Calcular el indicador de participación electoral con base en el potencial y el total de sugfagantes del departamento

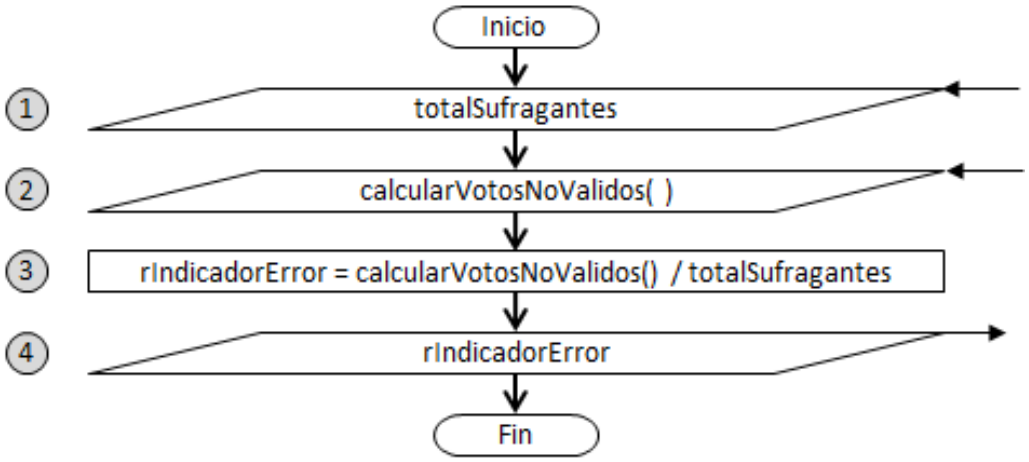


Figura 9.30 Diagrama de flujo para el método calcularIndicadorError()

9.2.2.3.14 Clase Departamento - determinarValidez( ):lógico

Descripción: Determinar la consistencia interna de los valores de los atributos, revisando que se cumplan todas las condiciones de validación.

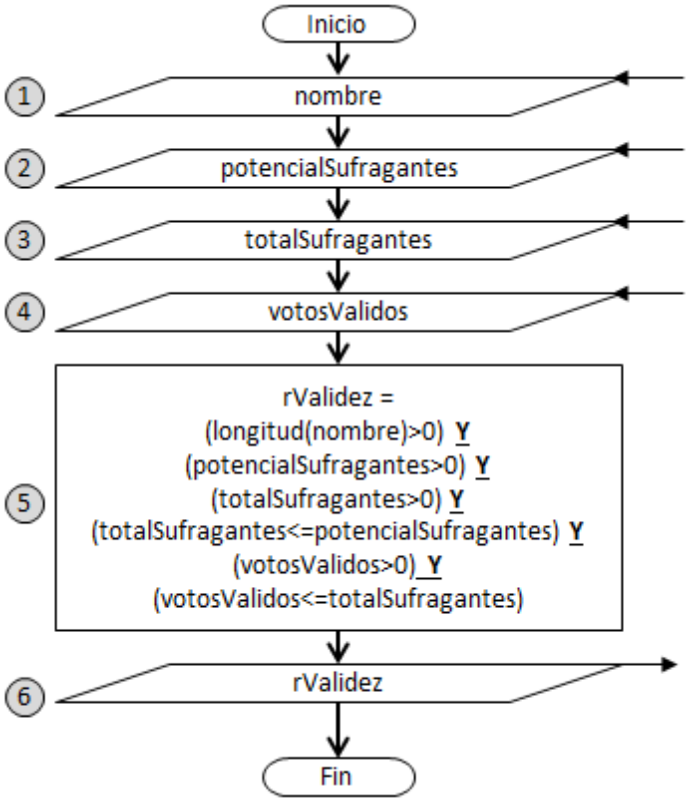


Figura 9.31 Diagrama de flujo para el método determinarValidez()

9.2.2.3.15 Clase Presentacion

La clase Presentacion mantiene su estructura y por lo tanto la misma definición tal como se describe en las secciones 9.1.2.3.7 a 9.1.2.3.9 del presente capítulo.

9.2.2.3.16 Clase Menu

La clase Menu mantiene su estructura y por lo tanto la misma definición tal como se describe en la sección 8.4.2.3.9 del capítulo anterior.

9.2.2.3.17 Clase Datos

La clase Datos mantiene su estructura y por lo tanto la misma definición tal como se describe en las secciones 9.1.2.3.11 a 9.1.2.3.15 del presente capítulo.

9.2.2.3.18 Clase Main

Descripción: Coordina las acciones de las demás clases. Solicita a un objeto de la clase Menu que reciba de parte del usuario la opción requerida por él, que almacena en una variable.

La clase Main mantiene su especificación en el nuevo prototipo, la cual está ampliamente descrita en la sección 8.4.2.3.15 del capítulo anterior.

9.2.3 Implementación

La estructura del proyecto no cambia, solo se incluye un método en la clase Departamento para la validación de los datos, que es reutilizado para su presentación y su grabación.

Tabla 9.22 Codificación de la clase *Departamento*

```
1  package proyecto;
2
3  public class Departamento {
4     private String nombre;
5     private int potencialSufragantes;
6     private int totalSufragantes;
7     private int votosValidos;
8
9     public Departamento(){
10       this.nombre="";
11       this.potencialSufragantes=0;
12       this.totalSufragantes=0;
13       this.votosValidos=0;
14    }
15
16    public void setNombre(String pNombre){
17       if (pNombre.length()>0){
18          this.nombre=pNombre;
19       }
20    }
21
22    public void setPotencialSufragantes(int pPotencialSufragantes){
23       if (pPotencialSufragantes>this.totalSufragantes){
24          this.potencialSufragantes=pPotencialSufragantes;
25       }
26    }
27
28    public void setTotalSufragantes(int pTotalSufragantes){
29       if ((pTotalSufragantes>=this.votosValidos) && (pTotalSufragantes<=this.potencialSufragantes)){
30          this.totalSufragantes=pTotalSufragantes;
31       }
32    }
33
34    public void setVotosValidos(int pVotosValidos){
35       if ((pVotosValidos>=0) && (pVotosValidos<=this.totalSufragantes)){
36          this.votosValidos=pVotosValidos;
37       }
38    }
39
40    public String getNombre(){
41       return this.nombre;
42    }
43
44    public int getPotencialSufragantes(){
45       return this.potencialSufragantes;
```

```
46      }
47
48      public int getTotalSufragantes(){
49         return this.totalSufragantes;
50      }
51
52      public int getVotosValidos(){
53         return this.votosValidos;
54      }
55
56      public double calcularParticipacion(){
57         double rParticipacion;
58         rParticipacion=((double)this.totalSufragantes/this.potencialSufragantes)*100;
59         return rParticipacion;
60      }
61
62      public String calcularCategoriaParticipacion(){
63         double tParticipacion;
64         String rCategoria;
65         tParticipacion=this.calcularParticipacion();
66         if (tParticipacion>=50){
67            rCategoria="Alta";
68         }
69         else{
70            if (tParticipacion>=40){
71               rCategoria="Media";
72            }
73            else{
74               rCategoria="Baja";
75            }
76         }
77         return rCategoria;
78      }
79
80      public int calcularVotosNoValidos(){
81         int rVotosNoValidos;
82         rVotosNoValidos=this.totalSufragantes-this.votosValidos;
83         return rVotosNoValidos;
84      }
85
86      public double calcularIndicadorError(){
87         double rIndicadorError;
88         rIndicadorError=(double)this.calcularVotosNoValidos()/this.totalSufragantes;
89         return rIndicadorError;
90      }
91
92      public boolean determinarValidez(){
93         boolean rValidez;
94         rValidez=((this.nombre.length()>0) &&
95                  (this.potencialSufragantes>0) &&
96                  (this.totalSufragantes>0) &&
97                  (this.totalSufragantes<=this.potencialSufragantes) &&
98                  (this.votosValidos>0) &&
```

```
99                  (this.votosValidos<=this.totalSufragantes));
100         return rValidez;
101      }
102   }
```

Tabla 9.23 Codificación de la clase *Presentacion*

```
1   package proyecto;
2   import javax.swing.JOptionPane;
3
4   public class Presentacion {
5      public Presentacion(){
6
7      }
8
9      public Departamento capturarDepartamento(){
10        Departamento rDepartamento;
11        rDepartamento=new Departamento();
12        rDepartamento.setNombre(JOptionPane.showInputDialog("Nombre: "));
13        rDepartamento.setPotencialSufragantes(Integer.parseInt(JOptionPane.showInputDialog("Potencial de
    sufragantes: ")));
14        rDepartamento.setTotalSufragantes(Integer.parseInt(JOptionPane.showInputDialog("Total de
    sufragantes: ")));
15        rDepartamento.setVotosValidos(Integer.parseInt(JOptionPane.showInputDialog("Votos válidos: ")));
16        return rDepartamento;
17     }
18
19     public String capturarNombreDepartamento(){
20        String rNombre;
21        rNombre=JOptionPane.showInputDialog("Departamento: ");
22        return rNombre;
23     }
24
25     public void presentarDepartamento(Departamento pDepartamento){
26        String rMensaje;
27        if (pDepartamento.determinarValidez()){
28           rMensaje="Departamento: "+pDepartamento.getNombre();
29           rMensaje=rMensaje+"\nPotencial de sufragantes: "+pDepartamento.getPotencialSufragantes();
30           rMensaje=rMensaje+"\nTotal de sufragantes: "+pDepartamento.getTotalSufragantes();
31           rMensaje=rMensaje+"\nVotos válidos: "+pDepartamento.getVotosValidos();
32           rMensaje=rMensaje+"\nIndicador de participación: "+pDepartamento.calcularParticipacion()+" %";
33           rMensaje=rMensaje+"\nCategoría según participación:
    "+pDepartamento.calcularCategoriaParticipacion();
34           rMensaje=rMensaje+"\nVotos no válidos: "+pDepartamento.calcularVotosNoValidos();
35           rMensaje=rMensaje+"\nIndicador de error: "+pDepartamento.calcularIndicadorError();
36        }
37        else{
38           rMensaje="Datos inconsistentes. No se pueden presentar";
39        }
40        JOptionPane.showMessageDialog(null, rMensaje);
41     }
```

```
42  }
```

Tabla 9.24 Codificación de la clase *Menu*

```
1   package proyecto;
2   import javax.swing.JOptionPane;
3
4   public class Menu {
5      public Menu(){
6
7      }
8
9      public int capturarOpcion(){
10        String tMensaje;
11        int rOpcion;
12        tMensaje="1. Registrar departamento\n"+
13              "2. Consultar departamento\n"+
14              "3. Salir\n";
15        rOpcion=Integer.parseInt(JOptionPane.showInputDialog(tMensaje));
16        return rOpcion;
17     }
18  }
```

Tabla 9.25 Codificación de la clase *Datos*

```
1   package proyecto;
2   import java.io.*;
3
4   public class Datos {
5      public Datos(){
6
7      }
8
9      public String construirRegistroDepartamento(Departamento pDepartamento){
10        String rRegistro;
11        rRegistro=pDepartamento.getNombre()+",";
12        rRegistro=rRegistro+pDepartamento.getPotencialSufragantes()+",";
13        rRegistro=rRegistro+pDepartamento.getTotalSufragantes()+",";
14        rRegistro=rRegistro+pDepartamento.getVotosValidos();
15        return rRegistro;
16     }
17
18     public String descomponerRegistro(String pRegistro, int pOrden){
19        int i,j;
20        String rDato;
21        i=0;
22        j=1;
23        rDato="";
24        while ((i<pRegistro.length()) && (j<=pOrden)) {
```

```
25          if (pRegistro.substring(i,i+1).compareTo(",")!=0){
26             if (j==pOrden){
27                rDato=rDato+pRegistro.substring(i,i+1);
28             }
29          }
30          else{
31             j=j+1;
32          }
33          i=i+1;
34       }
35       return rDato;
36    }
37
38    public String recuperarRegistroDepartamento(String pNombre){
39       FileReader fileReader;
40       BufferedReader bufferedReader;
41       String rRegistro,tNombre;
42       try{
43          fileReader=new FileReader("departamentos.txt");
44          bufferedReader=new BufferedReader(fileReader);
45          do{
46             rRegistro=bufferedReader.readLine();
47             tNombre=this.descomponerRegistro(rRegistro, 1);
48          } while (tNombre.compareToIgnoreCase(pNombre)!=0);
49          fileReader.close();
50       }
51       catch (Exception e){
52          rRegistro="";
53       }
54       return rRegistro;
55    }
56
57    public Departamento recuperarDepartamento(String pNombre){
58       String tRegistro;
59       Departamento rDepartamento;
60       rDepartamento=new Departamento();
61       tRegistro=this.recuperarRegistroDepartamento(pNombre);
62       rDepartamento.setNombre(this.descomponerRegistro(tRegistro,1));
63       rDepartamento.setPotencialSufragantes(Integer.parseInt(this.descomponerRegistro(tRegistro,2)));
64       rDepartamento.setTotalSufragantes(Integer.parseInt(this.descomponerRegistro(tRegistro,3)));
65       rDepartamento.setVotosValidos(Integer.parseInt(this.descomponerRegistro(tRegistro,4)));
66       return rDepartamento;
67    }
68
69    public void grabarDepartamento(Departamento pDepartamento){
70       FileWriter fileWriter;
71       PrintWriter printWriter;
72       if (pDepartamento.determinarValidez()){
73          try{
74             fileWriter=new FileWriter("departamentos.txt",true);
75             printWriter=new PrintWriter(fileWriter);
76             printWriter.println(this.construirRegistroDepartamento(pDepartamento));
77             fileWriter.close();
```

```
78          }
79          catch(Exception e){
80
81          }
82       }
83    }
84 }
```

Tabla 9.26 Codificación de la clase *Main*

```
1  package proyecto;
2
3  public class Main {
4
5     public static void main(String[] args) {
6        Departamento departamento;
7        Presentacion presentacion;
8        Menu menu;
9        Datos datos;
10       int opcion;
11       String parametro;
12       presentacion=new Presentacion();
13       menu=new Menu();
14       datos=new Datos();
15       opcion=0;
16       do {
17          opcion=menu.capturarOpcion();
18          switch (opcion){
19             case 1:{
20                departamento=presentacion.capturarDepartamento();
21                datos.grabarDepartamento(departamento);
22                presentacion.presentarDepartamento(departamento);
23                break;
24             }
25             case 2:{
26                parametro=presentacion.capturarNombreDepartamento();
27                departamento=datos.recuperarDepartamento(parametro);
28                presentacion.presentarDepartamento(departamento);
29                break;
30             }
31          }
32       } while (opcion<=2);
33    }
34 }
```

9.3 Protección frente a fallas por datos erróneos – Prototipo 10

9.3.1 Análisis

9.3.1.1 Alcance

A pesar de que previene el manejo de objetos con datos inconsistentes, el prototipo anterior está expuesto a errores de digitación por parte del usuario o a recuperación infructuosa de datos. Además de las medidas de aseguramiento de la consistencia de los datos, se requiere evitar que la aplicación termine abruptamente. El prototipo 10 debe tener la misma funcionalidad del prototipo 09, agregando la implementación de medidas de protección para evitar que la aplicación falle cuando el usuario digita un dato incorrecto o cuando no se encuentra un registro en el archivo. La utilización del prototipo requiere eliminar el archivo anterior.

9.3.1.2 Definiciones

Se mantienen las mismas definiciones y reglas de validación.

Tabla 9.27 Definiciones aplicables en desarrollo del prototipo 10.

	Definición	Tipo de dato	Condiciones de validación
Departamento	Cada una de las circunscripciones territoriales de primer nivel en que se divide el país para propósitos electorales	Objeto	
Nombre	Identificación nominal del departamento	Cadena de caracteres	Valor ingresado. El nombre del departamento no puede quedar vacío.
Potencial de sufragantes	Cantidad de personas habilitadas en un departamento para votar en un proceso electoral	Número entero	Valor ingresado. El potencial de sufragantes debe ser mayor que cero.
Total de sufragantes	Cantidad de personas que acudieron a votar en un departamento en un proceso electoral	Número entero	Valor ingresado. El total de sufragantes debe ser mayor que cero, pero menor o igual al potencial de sufragantes.
Indicador de participación electoral	Indicador consistente en el total de sufragantes expresado como porcentaje del potencial de sufragantes	Número real	Valor calculado. $Indicador\ de\ participación\ electoral = \frac{Total\ de\ sufragantes}{Potencial\ de\ sufragantes} * 100$
Categoría de participación electoral	Categoría a la que pertenece el departamento, dependiendo del porcentaje de participación electoral.	Cadena de caracteres	Valor calculado. Si el porcentaje es mayor o igual a 50%: alto Si el porcentaje es menor a 50% pero mayor o igual a 40%: medio Si el porcentaje es menor a 40%: bajo
Votos válidos	Cantidad de votos que indican de forma clara la voluntad del sufragante	Número entero	Valor ingresado. La cantidad de votos válidos debe ser mayor que

			cero, pero menos o igual al total de sufragantes.
Votos no válidos	Cantidad de votos que NO indican de forma clara la voluntad del sufragante.	Número entero	Valor calculado. $Votos\ no\ válidos = Total\ de\ sufragantes - Votos\ válidos$
Indicador de error en la votación	Indicador consistente en dividir el total de votos no válidos entre el total de sufragantes.	Número real	Valor calculado. $Indicador\ de\ error\ en\ la\ votación = \frac{Votos\ no\ válidos}{Total\ de\ sufragantes}$

9.3.1.3 Requisitos

No hay modificación en los requisitos de interfaz de usuario ni en los funcionales.

9.3.1.3.1 Requisitos de interfaz de usuario

9.3.1.3.1.1 Ingreso de información en casillas de edición

Para la interacción con el usuario, la aplicación debe presentar ventanas con casillas de edición en donde pueda digitar los datos solicitados. Al solicitar los datos, el sistema deberá indicar en la misma ventana de diálogo cuál es el dato requerido.

9.3.1.3.1.2 Presentación de menú de selección de funciones

La aplicación debe presentar un menú para seleccionar una de 3 funciones: registrar la información de un departamento para su almacenamiento, consultar información de in departamento a partir de la recuperación de sus datos almacenados, o finalizar la aplicación.

9.3.1.3.2 Requisitos funcionales

9.3.1.3.2.1 Calcular el indicador de participación electoral de un departamento

La aplicación debe permitir al usuario introducir los siguientes datos básicos de un departamento: el nombre, el potencial de sufragantes, el total de sufragantes y la cantidad de votos válidos, y con estos datos debe calcular el indicador de participación electoral, los votos no válidos y el indicador de error de votación. Los datos ingresados deben cumplir las condiciones de validación requeridas.

9.3.1.3.2.2 Almacenar la información de un departamento en un archivo plano

La aplicación debe poder almacenar en un archivo plano los datos básicos ingresados de un departamento: el nombre, el potencial de sufragantes, el total de sufragantes y el total de votos válidos. Se requiere que el formato de almacenamiento sea en datos separados por punto y coma (formato csv).

9.3.1.3.2.3 Recuperar la información de un departamento en un archivo plano

La aplicación debe poder recuperar desde un archivo plano los datos básicos ingresados de un departamento: el nombre, el potencial de sufragantes, el total de sufragantes, el total de votos válidos además de los datos calculados del indicador de participación electoral, la categoría de participación a la que pertenece, los votos no válidos y el indicador de error de votación y se debe presentar al usuario. Para ello deberá preguntar por el nombre del departamento a buscar. Como restricción, se requiere el formato de almacenamiento sea en datos separados por punto y coma (formato csv).

9.3.2 Diseño

9.3.2.1 Diseño de escenarios

Se mantiene el mismo diseño de escenarios.

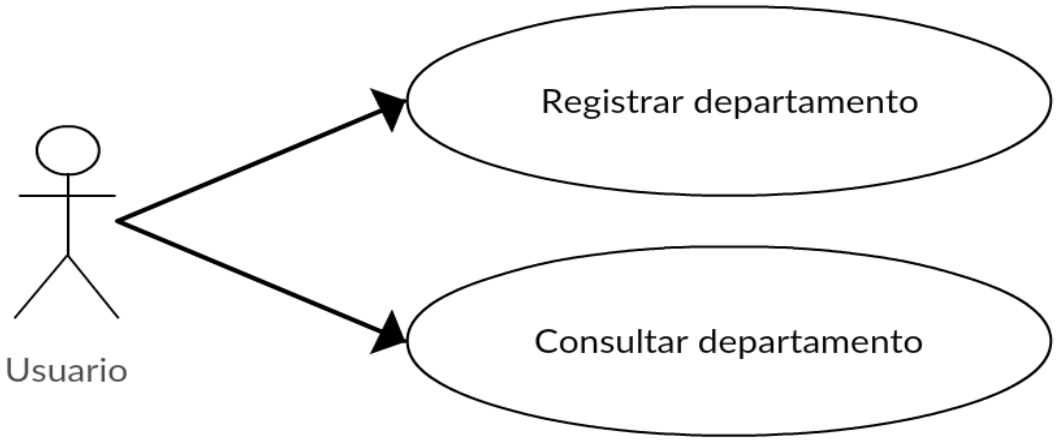


Figura 9.32 Diagrama de casos de uso para el Prototipo 10

Tabla 9.28 Especificación del caso de uso *Registrar departamento*.

1	El usuario ingresa a la aplicación.
2	La aplicación presenta el menú de selección, por lo que el usuario selecciona la opción 1: Registrar Departamento
2.1	Si el usuario selecciona una opción mayor que 2, la aplicación finaliza.
3	La aplicación presenta una pantalla con una caja de texto para que el usuario digite el nombre del departamento
4	El usuario digita el nombre del departamento y pulsa el botón *Aceptar*
5	La aplicación presenta una pantalla con una caja de texto para que el usuario digite el potencial de sufragantes del departamento
6	El usuario digita el potencial de sufragantes del departamento y pulsa el botón *Aceptar*
7	La aplicación presenta una pantalla con una caja de texto para que el usuario digite el total de sufragantes del departamento
8	El usuario digita el total de sufragantes del departamento y pulsa el botón *Aceptar*

9	La aplicación presenta una pantalla con una caja de texto para que el usuario digite la cantidad de votos válidos obtenidos en el departamento
10	El usuario digita la cantidad de votos válidos del departamento y pulsa el botón *Aceptar*
11	Si los datos son válidos: la aplicación muestra en pantalla el nombre del departamento, el potencial de sufragantes, el total de sufragantes, la cantidad de votos válidos, el indicador de participación electoral del departamento, la categoría de participación, la cantidad de votos no válidos y el indicador de error de votación en el departamento que acabó de ingresar.
12	Si los datos no son válidos: la aplicación muestra en pantalla el mensaje "Datos inconsistentes. No se pueden presentar"
13	La aplicación vuelve a presentar el menú de selección (paso 2)

Tabla 9.29 Especificación del caso de uso *Consultar departamento*.

1	El usuario ingresa a la aplicación.
2	La aplicación presenta el menú de selección, por lo que el usuario selecciona la opción 2: Consultar Departamento
2.1	Si el usuario selecciona una opción mayor que 2, la aplicación finaliza.
3	El usuario digita el nombre del departamento a buscar y pulsa el botón *Aceptar*
4	La aplicación muestra en pantalla el nombre del departamento buscado, el potencial de sufragantes, el total de sufragantes, la cantidad de votos válidos, el indicador de participación electoral del departamento, la categoría de participación, la cantidad de votos no válidos y el indicador de error de votación en el departamento.
5	La aplicación vuelve a presentar el menú de selección (paso 2)

9.3.2.2 Diseño de estructura

La estructura no se modifica. Los algoritmos de captura de datos y de recuperación de datos se modifican mediante una estructura de manejo de errores. Si por alguna razón, los datos capturados desde teclado o los leídos desde el archivo cuentan con errores, se resguarda la creación de objeto sin que la aplicación termine abruptamente.

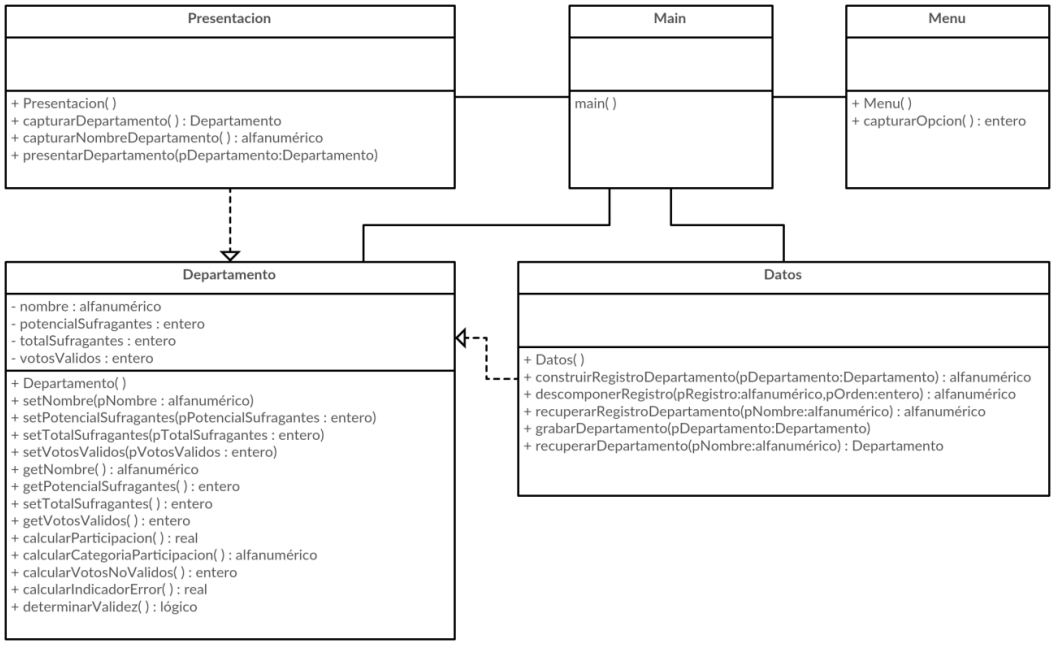


Figura 9.33 Diagrama de clases para el Prototipo 10

9.3.2.3 Diseño de algoritmos

9.3.2.3.1 Clase Departamento - Departamento( )

Descripción: Constructor que inicializa los atributos con valores neutros.

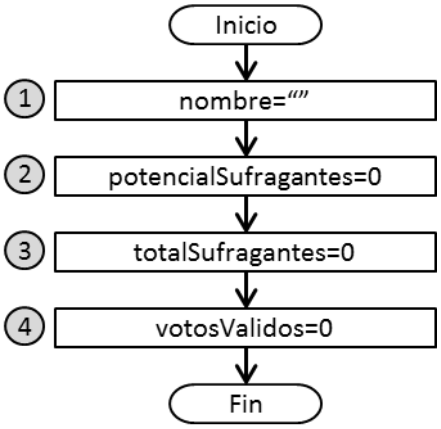


Figura 9.34 Diagrama de flujo para el método Departamento()

9.3.2.3.2 Clase Departamento - setNombre(pNombre : alfanumérico)

Descripción: Método que asigna al atributo nombre el valor dado como parámetro, previa comprobación de su validez.

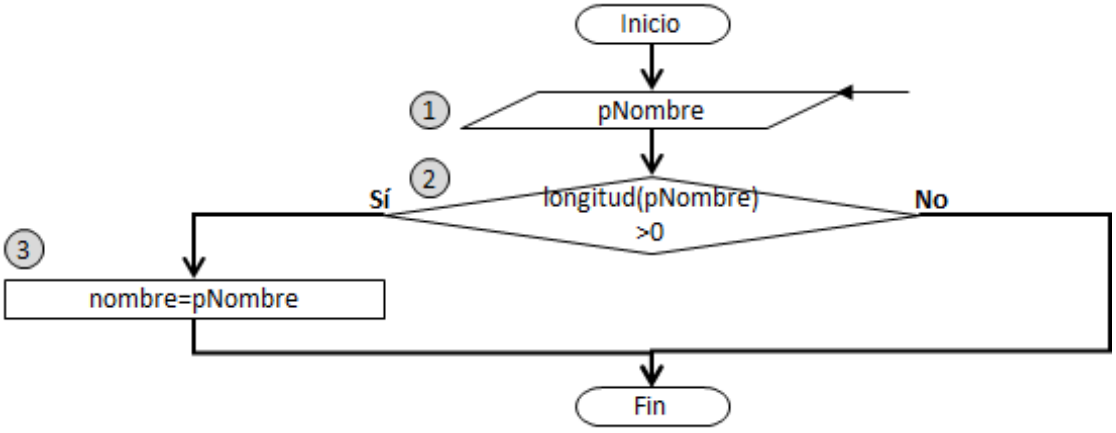


Figura 9.35 Diagrama de flujo para el método setNombre()

9.3.2.3.3 Clase Departamento - setPotencialSufragantes(pPotencialSufragantes : entero)

Descripción: Método que asigna al atributo potencialSufragantes el valor dado como parámetro, previa comprobación de su validez.

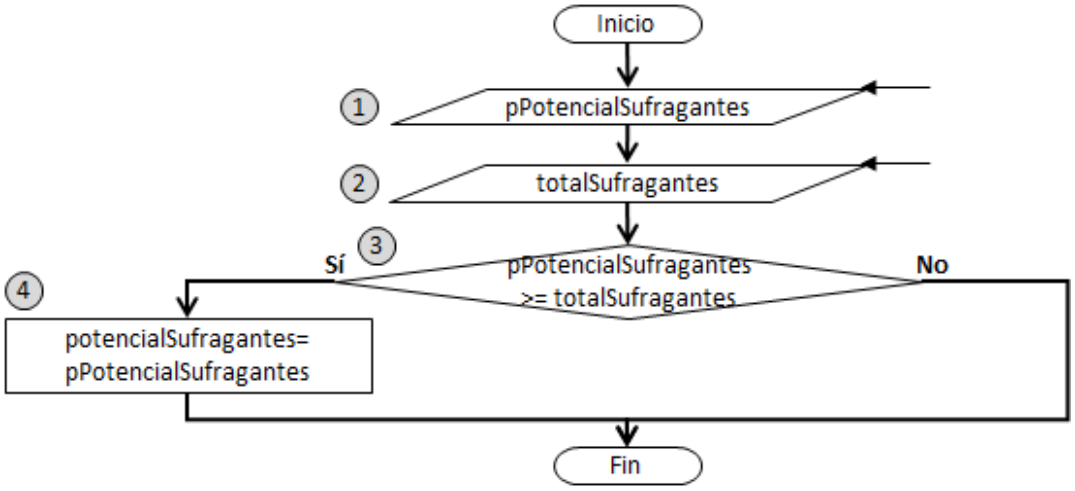


Figura 9.36 Diagrama de flujo para el método setPotencialSufragantes()

9.3.2.3.4 Clase Departamento - setTotalSufragantes(pTotalSufragantes : entero)

Descripción: Método que asigna al atributo totalSufragantes el valor dado como parámetro, previa comprobación de su validez.

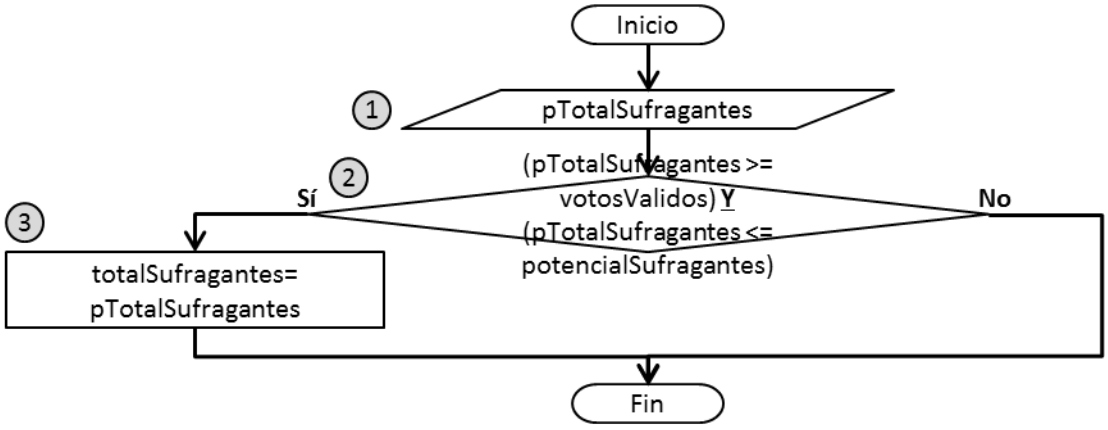


Figura 9.37 Diagrama de flujo para el método setTotalSufragantes()

9.3.2.3.5 Clase Departamento - setVotosValidos(pVotosValidos : entero)

Descripción: Método que asigna al atributo votosValidos el valor dado como parámetro, previa comprobación de su validez.

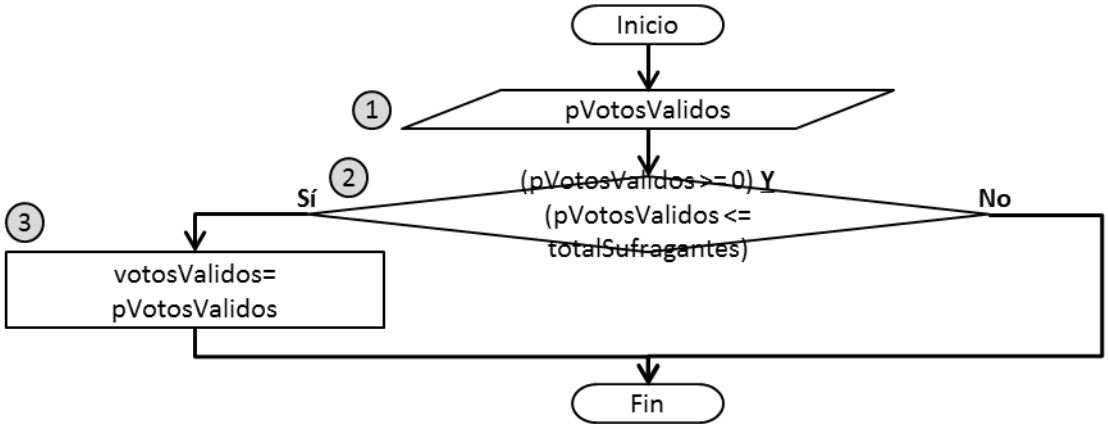


Figura 9.38 Diagrama de flujo para el método setVotosValidos()

Tabla 9.30 Descripción del algoritmo para el método setVotosValidos()

1	Se requiere como dato de entrada *pVotosValidos.* En este caso se recibe la variable como mensaje.
2	Se revisa si el valor de la variable *pVotosValidos* es mayor o igual al 0 y es menor o igual al valor del atributo *totalSufragantes*
3	Si se cumple la condición: Asigna al atributo *votosValidos* el valor de la variable recibida *pVotosValidos*

9.3.2.3.6 Clase Departamento - getNombre( ):alfanumérico

Descripción: Entrega como respuesta el valor del atributo nombre.

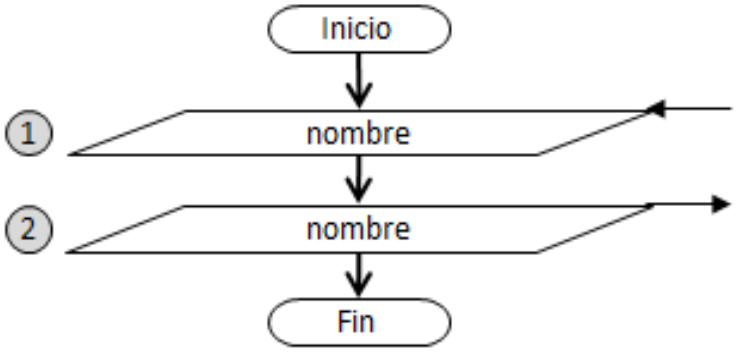


Figura 9.39 Diagrama de flujo para el método getNombre()

9.3.2.3.7 Clase Departamento - getPotencialSufragantes( ):entero

Descripción: Entrega como respuesta el valor del atributo potencialSufragantes.

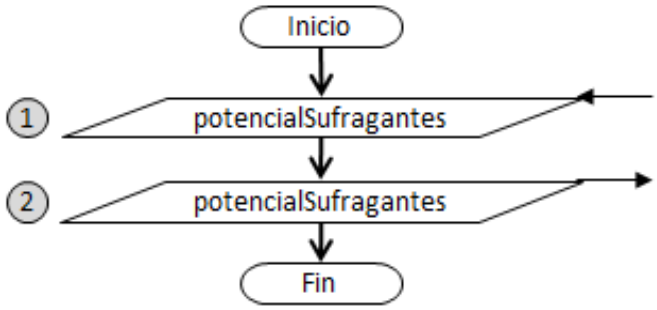


Figura 9.40 Diagrama de flujo para el método getPotencialSufragantes()

9.3.2.3.8 Clase Departamento - getTotalSufragantes( ):entero

Descripción: Entrega como respuesta el valor del atributo totalSufragantes.

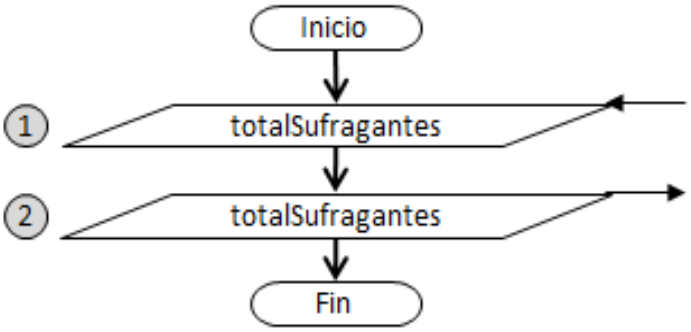


Figura 9.41 Diagrama de flujo para el método getTotalSufragantes()

9.3.2.3.9 Clase Departamento - getVotosValidos( ):entero

Descripción: Entrega como respuesta el valor del atributo votosValidos.

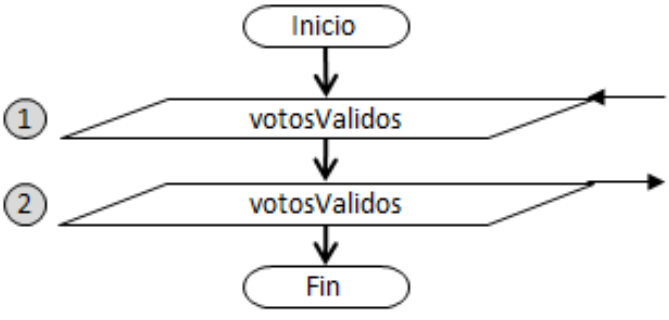


Figura 9.42 Diagrama de flujo para el método getVotosValidos()

9.3.2.3.10 Clase Departamento – calcularParticipacion( ): real

Descripción: Método de cálculo para obtener el indicador de participación electoral con base en el potencial y el total de sufragantes del departamento.

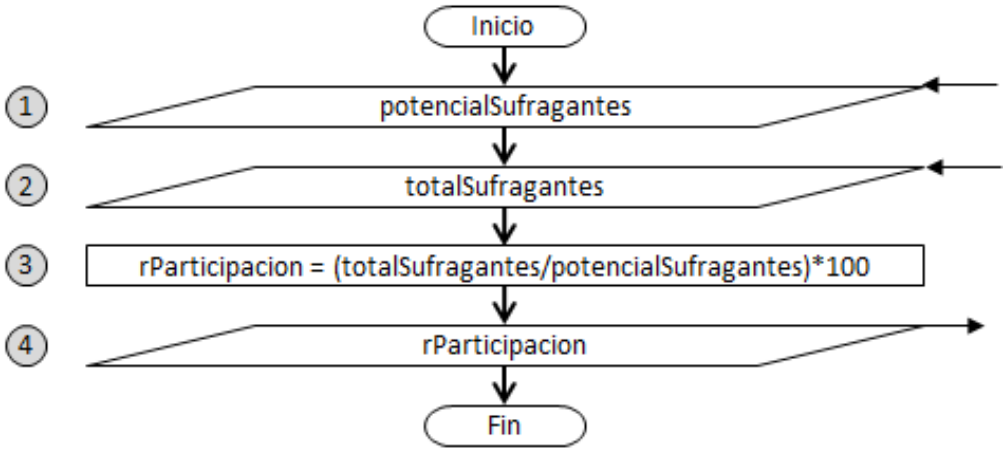


Figura 9.43 Diagrama de flujo para el método calcularParticipacion()

9.3.2.3.11 Clase Departamento – calcularCategoriaParticipacion( ): cadena

Descripción: Método de cálculo para determinar la categoría de participación electoral del departamento, con base en el indicador de participación calculado por el método anterior.

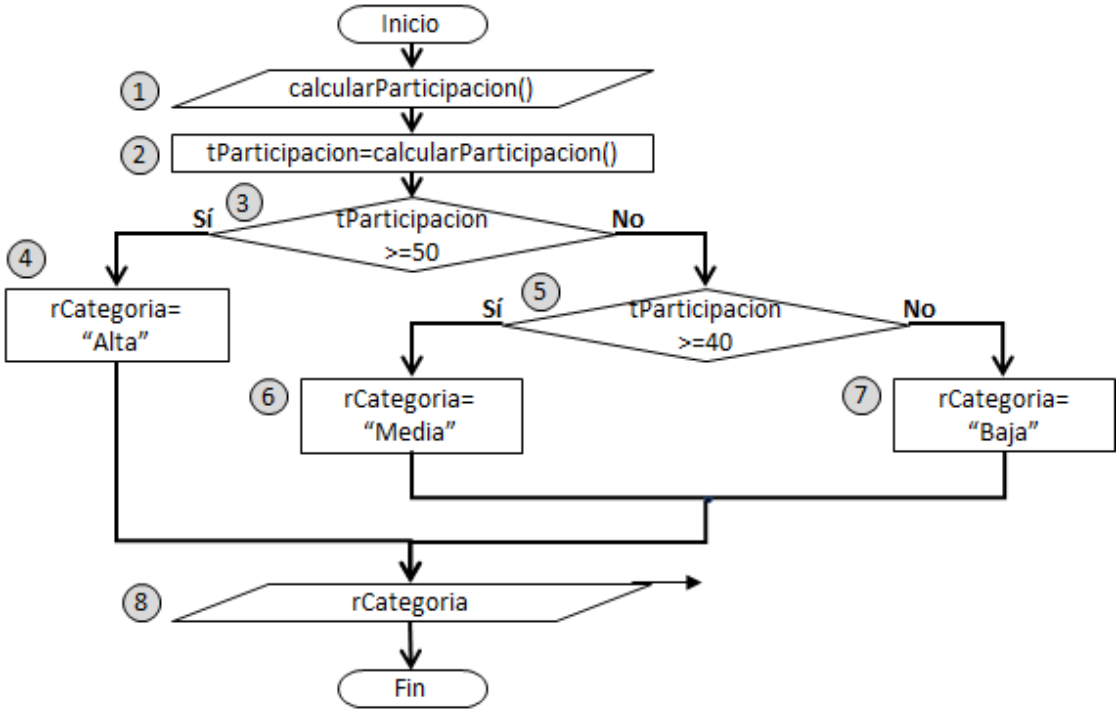


Figura 9.44 Diagrama de flujo para el método calcularCategoriaParticipacion()

9.3.2.3.12 Clase Departamento - calcularVotosNoValidos( ):entero

Descripción: Calcular la cantidad de votos no válidos con base en el total de sufragantes y los votos válidos en el departamento

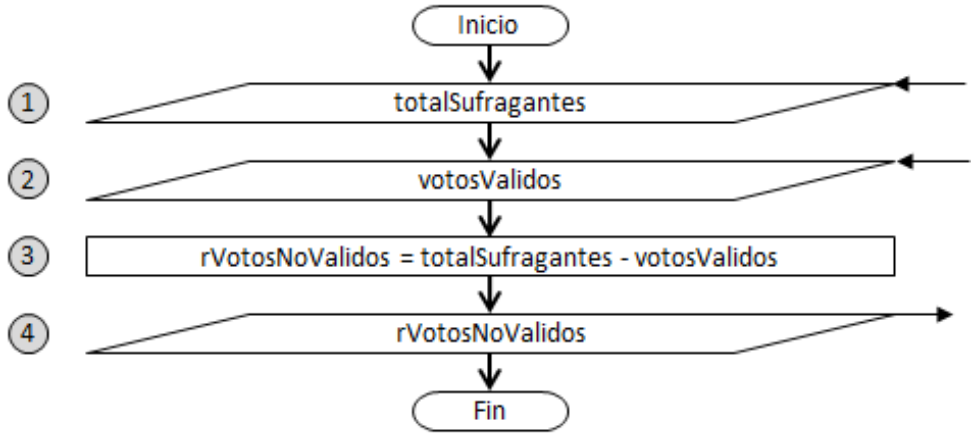


Figura 9.45 Diagrama de flujo para el método calcularVotosNoValidos()

9.3.2.3.13 Clase Departamento - calcularIndicadorError( ):real

Descripción: Calcular el indicador de participación electoral con base en el potencial y el total de sugfagantes del departamento

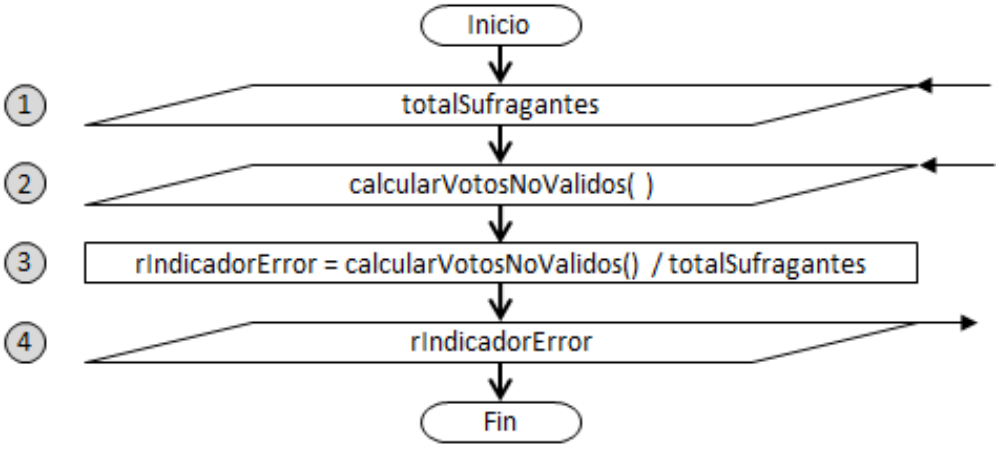


Figura 9.46 Diagrama de flujo para el método calcularIndicadorError()

9.3.2.3.14 Clase Departamento - determinarValidez( ):lógico

Descripción: Determinar la consistencia interna de los valores de los atributos, revisando que se cumplan todas las condiciones de validación.

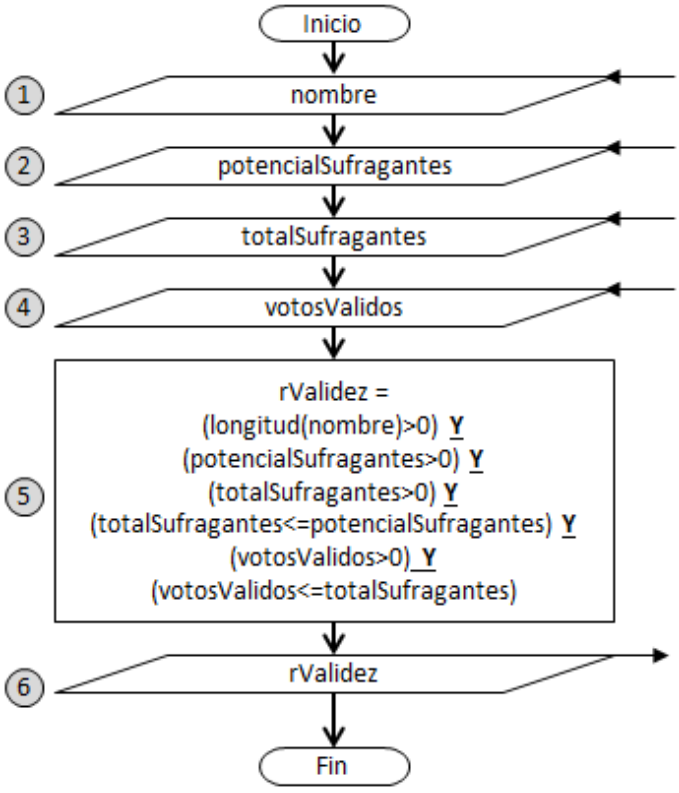


Figura 9.47 Diagrama de flujo para el método determinarValidez()

9.3.2.3.15 Clase Presentacion - capturarDepartamento( ):Departamento

Descripción: Obtiene del usuario cada uno de los valores para los atributos, los cuales asigna mediante los métodos set, y entrega como respuesta el objeto de la clase Departamento con los valores de los atributos asignados.

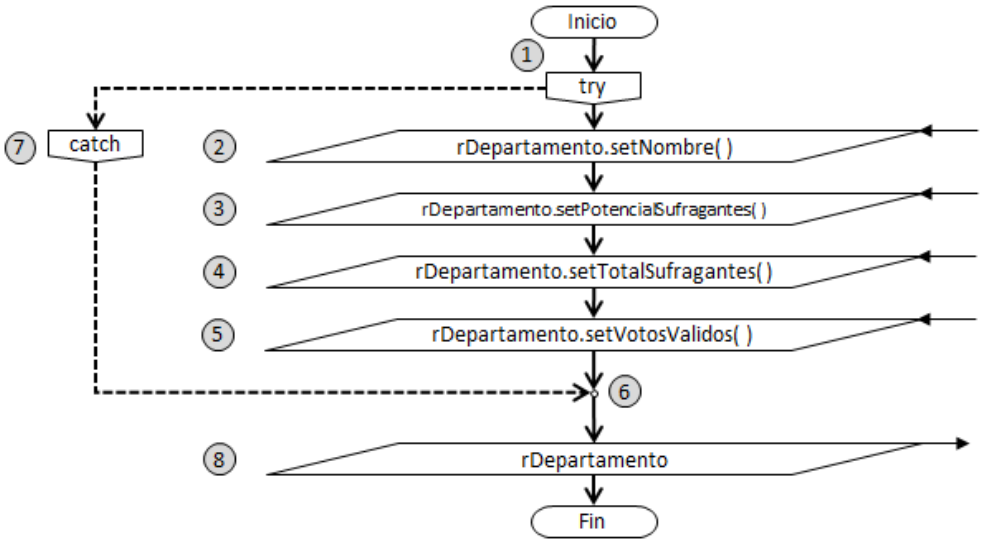


Figura 9.48 Diagrama de flujo para el método capturarDepartamento()

Tabla 9.31 Descripción del algoritmo para el método capturarDepartamento()

1	Inicia la estructura de manejo de errores en caso de poder ingresar valores correctos a los atributos del objeto de tipo Departamento
2	Se requiere como dato de entrada el nombre del departamento para almacenar en el objeto de tipo Departamento. Para cada dato de entrada, la aplicación debe presentar una pantalla con una caja de texto.
3	Se requiere como dato de entrada el potencial de sufragantes del departamento para almacenar en el objeto
4	Se requiere como dato de entrada el total de sufragantes del departamento para almacenar en el objeto
5	Se requiere como dato de entrada el total de votos válidos en el departamento para almacenar en el objeto
6-7	Inicia la estructura para la ejecución de acciones en caso de fallo por no ingresar valores correctos a los atributos
8	Se genera como dato de salida el objeto *rDepartamento*, de la clase *Departamento*. Para ello se realiza como respuesta de este método.

9.3.2.3.16 Clase Presentacion - capturarNombreDepartamento( ):cadena

Descripción: Obtiene del usuario un nombre de departamento, y lo entrega como respuesta al objeto que haya hecho la solicitud, es decir, que haya enviado el mensaje para la ejecución del método.

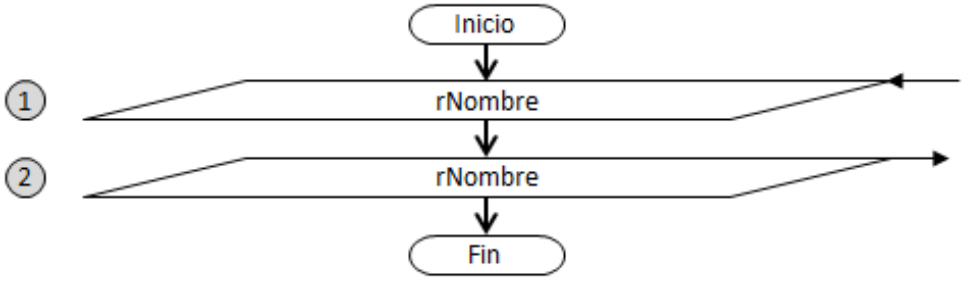


Figura 9.49 Diagrama de flujo para el método capturarNombreDepartamento ()

9.3.2.3.17 Clase Presentacion - presentarDepartamento(pDepartamento:Departamento)

Descripción: Recibe un mensaje con un objeto de la clase Departamento como parámetro. Si el objeto cumple con las condiciones de validación, construye un mensaje al que agrega cada uno de los valores de los atributos del objeto recibido, así como el resultado de los métodos de cálculo del indicador de participación electoral, de la categoría de participación electoral, de la cantidad de votos no válidos y del indicador de error. En caso de que el objeto no cumpla con las condiciones de validación, construye un mensaje de error. Por último muestra al usuario el mensaje construido.

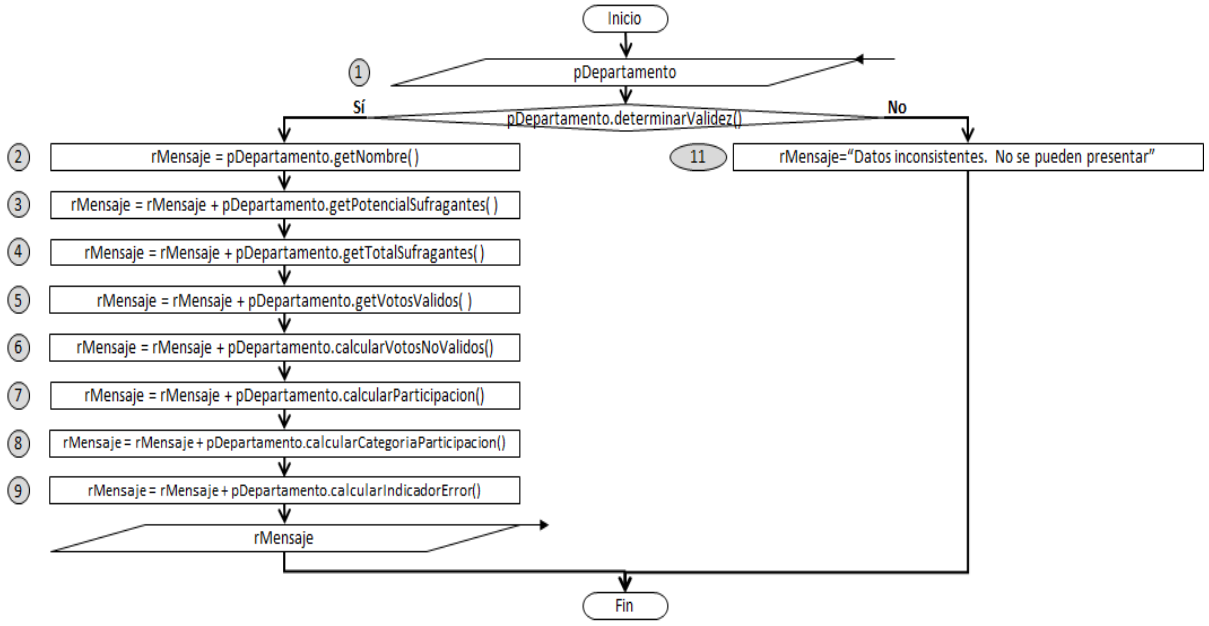


Figura 9.50 Diagrama de flujo para el método presentarDepartamento()

9.3.2.3.18 Clase Menu - capturarOpcion(): entero

Descripción: Presenta al usuario un mensaje con las opciones disponibles en la aplicación, y obtiene de dicho usuario el número correspondiente a la opción seleccionada.

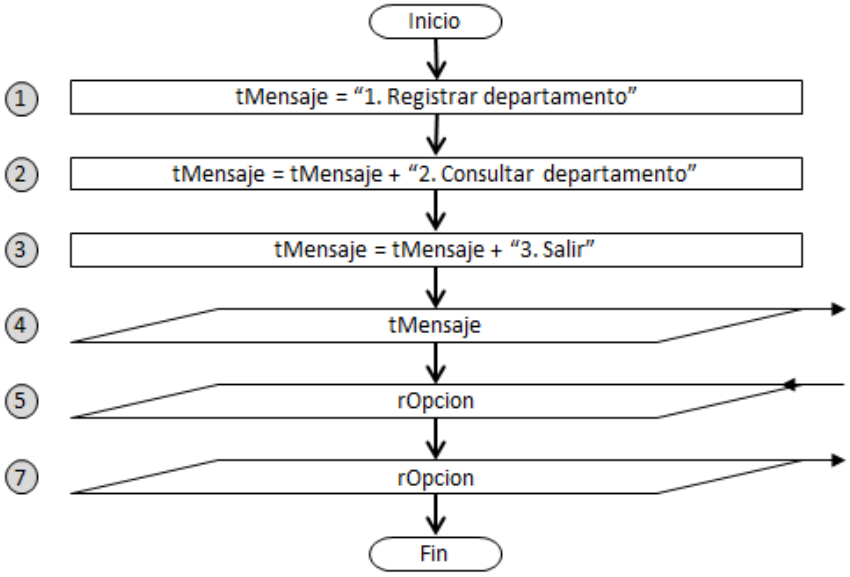


Figura 9.51 Diagrama de flujo para el método capturarOpcion()

9.3.2.3.19 Clase Datos - construirRegistroDepartamento(pDepartamento:Departamento): cadena

Descripción: Recibe un mensaje con un objeto de la clase Departamento como parámetro. Construye un registro, es decir una cadena de caracteres, a la que agrega cada uno de los valores de los atributos del objeto separados entre sí por un signo de coma. Por último lo entrega el registro como respuesta al objeto que haya hecho la solicitud, es decir, que haya invocado ejecución del método.

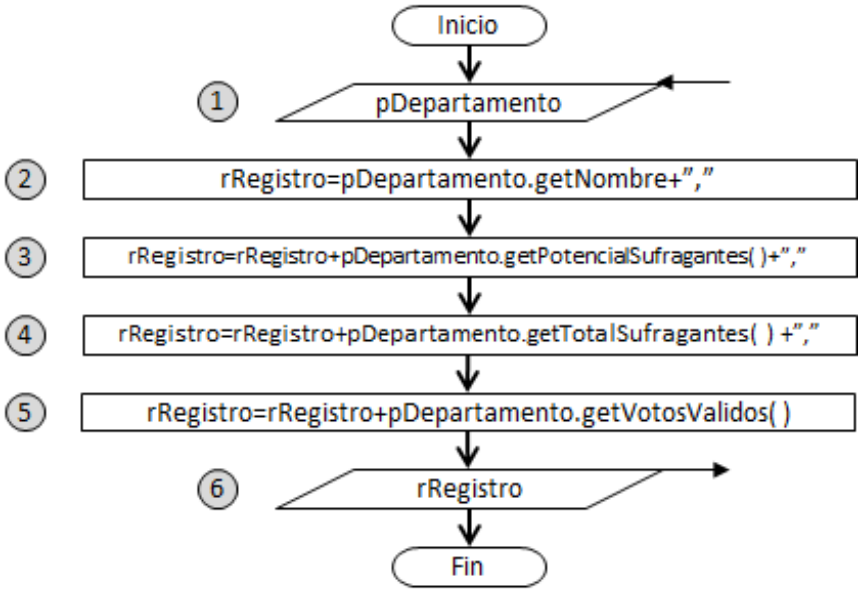


Figura 9.52 Diagrama de flujo para el método construirRegistroDepartamento ()

9.3.2.3.20 Clase Datos - descomponerRegistro (pRegistro:cadena, pOrden: entero):cadena

Descripción: Recibe un mensaje con dos parámetros; el primero de estos es un registro o cadena de caracteres que contiene valores separados entre sí por signos de coma; el segundo, un número ordinal que indica el dato que se debe extraer del registro y entregar como respuesta final. El método hace un recorrido secuencial por los caracteres del registro, y cuando ubica el comienzo del dato solicitado, comienza a agregar cada caracter en una variable que al final entrega como respuesta al objeto que haya requerido la ejecución del método.

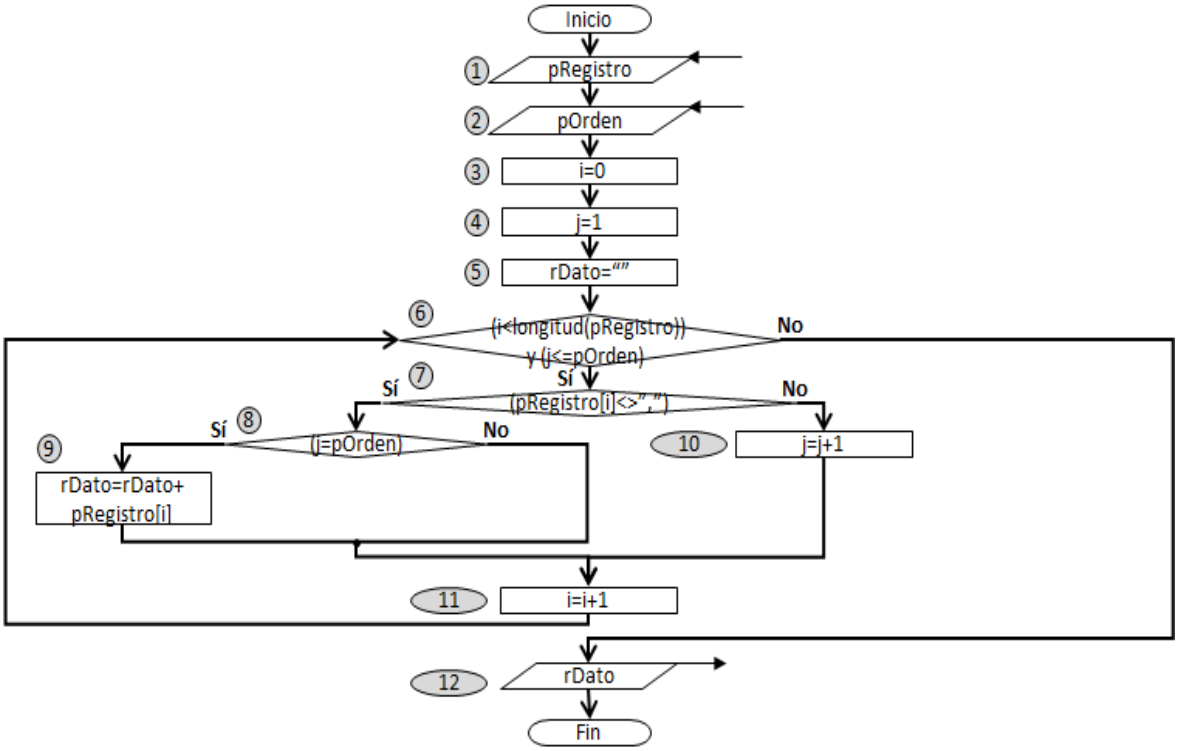


Figura 9.53 Diagrama de flujo para el método descomponerRegistro()

9.3.2.3.21 Clase Datos - grabarDepartamento(pDepartamento:Departamento

Descripción: Recibe un mensaje con un objeto de la clase Departamento como parámetro. Si el objeto cumple las condiciones de validación, abre un archivo y a continuación invoca el método de construcción del registro, cuya respuesta es grabada en el archivo que finalmente es cerrado; en caso contrario, omite la grabación del departamento.

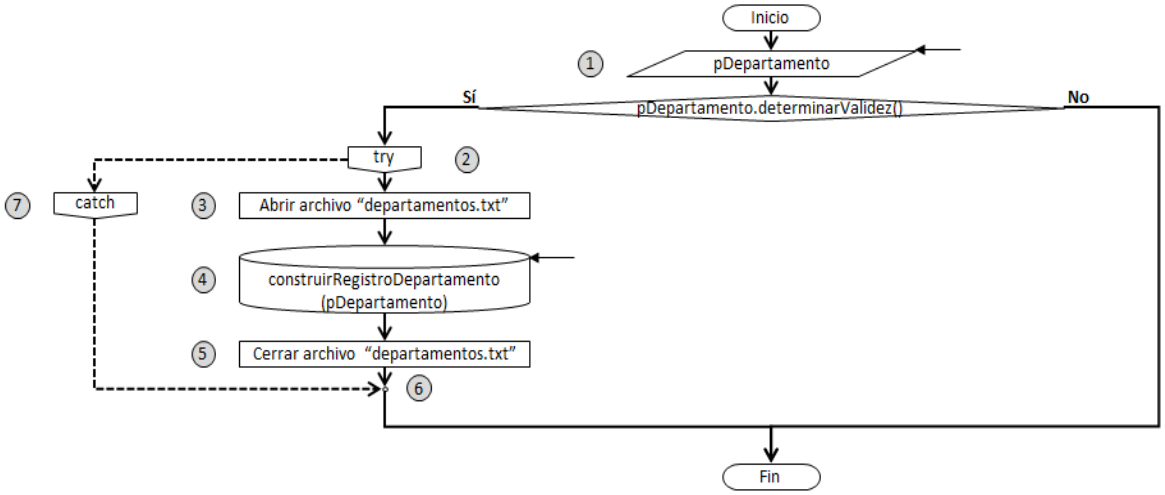


Figura 9.54 Diagrama de flujo para el método grabarDepartamento ()

9.3.2.3.22 Clase Datos - recuperarRegistroDepartamento(pNombre:cadena):cadena

Descripción: Recibe un mensaje con una cadena de caracteres como parámetro, que contiene un nombre de departamento a localizar en el archivo. Abre un archivo para lectura y a continuación lee uno a uno los registros hasta encontrar el que contiene los datos del departamento requerido, para lo cual solicita la descomposición del registro y la obtención del primer dato que corresponde al nombre. Por último entrega como respuesta el registro que contenga el nombre buscado.

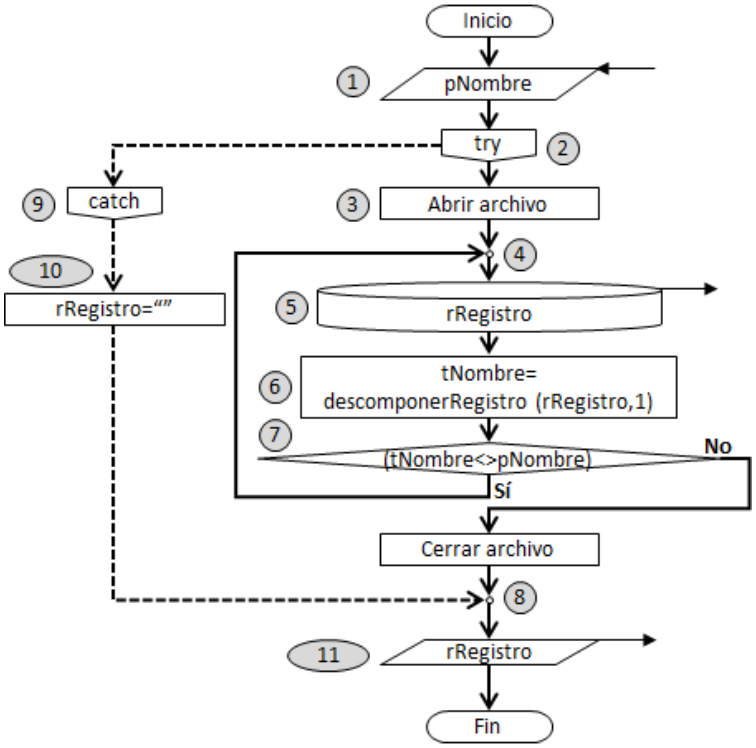


Figura 9.55 Diagrama de flujo para el método recuperarRegistroDepartamento()

9.3.2.3.23 Clase Datos - recuperarDepartamento(pNombre:cadena):Departamento

Descripción: Recibe un mensaje con una cadena de caracteres como parámetro, que contiene un nombre de departamento a localizar en el archivo. Solicita la ejecución del método de recuperación de registro mediante el nombre dado, y una vez obtenida la respuesta de aquel método, toma el registro hallado y solicita su descomposición en cuatro partes. Por último entrega como respuesta el objeto de la clase Departamento correspondiente al nombre dado como criterio de búsqueda.

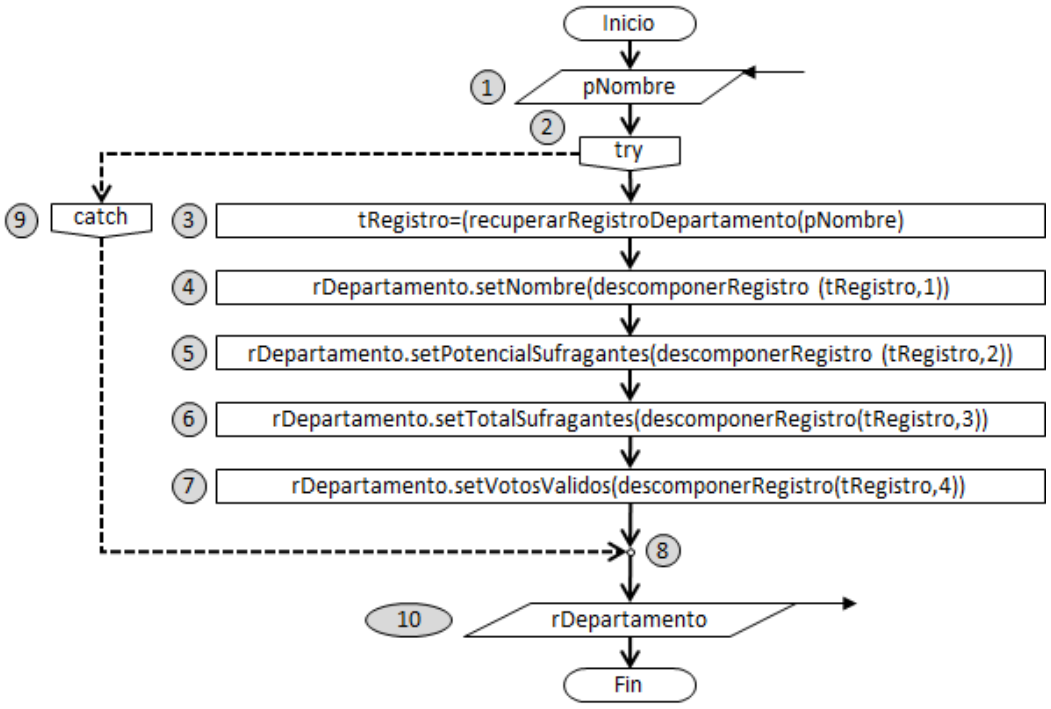


Figura 9.56 Diagrama de flujo para el método recuperarDepartamento()

Tabla 9.32 Descripción del algoritmo para el método recuperarDepartamento()

1	Se requiere como dato de entrada la variable pNombre. En este caso se recibe la variable como mensaje.
2	Inicia la estructura de manejo de errores en caso de poder recuperar valores correctos a los atributos del objeto de tipo Departamento
3	Almacena en la variable tRegistro la respuesta del método recuperarDepartamento () del objeto, envíandole como parámetro la variable pNombre
4	Se crea un objeto nuevo de tipo Departamento y asigna en su atributo nombre la respuesta del método descomponerRegistro() del objeto, envíandole como parámetros el valor tRegistro y el valor 1 para extraer el dato del nombre del departamento.
5	Se asigna en el atributo potencialSufragantes del objeto de tipo Departamento la respuesta del método descomponerRegistro() del objeto, envíandole como parámetros el valor tRegistro y el valor 2 para extraer el dato del potencial de sufragantes del departamento.
6	Se asigna en el atributo totalSufragantes del objeto de tipo Departamento la respuesta del método descomponerRegistro() del objeto, envíandole como parámetros el valor tRegistro y el valor 3 para extraer el dato del total de sufragantes del departamento.

7	Se asigna en el atributo votosValidos del objeto de tipo Departamento la respuesta del método descomponerRegistro() del objeto, envíandole como parámetros el valor tRegistro y el valor 4 para extraer el dato del total de sufragantes del departamento.
8-9	Inicia la estructura para la ejecución de acciones en caso de fallo por no recuperar valores correctos a los atributos
10	Se genera como dato de salida el objeto creado de tipo Departamento.

9.3.2.3.24 Clase Main – main( )

Descripción: Coordina las acciones de las demás clases. Solicita a un objeto de la clase Menu que reciba de parte del usuario la opción requerida por él, que almacena en una variable.

Si la opción seleccionada es 1, coordina las clases para que el usuario pueda registrar un departamento, es decir, solicita a un objeto de la clase Presentacion que lleve a cabo la captura de un objeto de la clase Departamento, que almacena luego en el objeto departamento, solicita al objeto de la clase Datos que grabe el objeto departamento y finalmente muestra la información del departamento ingresado.

Si la opción seleccionada es 2, coordina las clases para que el usuario pueda consultar dando como parámetro un nombre de departamento, solicita al objeto datos la recuperación de un departamento mediante el nombre capturado por el objeto presentación; el resultado de la operación de recuperación se almacena en el objeto departamento. Por último ordena al objeto de la clase Presentacion que lleve a cabo el método de presentación tomando como parámetro el objeto de la clase Departamento.

Todo esto se repite mientras la opción seleccionada sea menor o igual que 2.

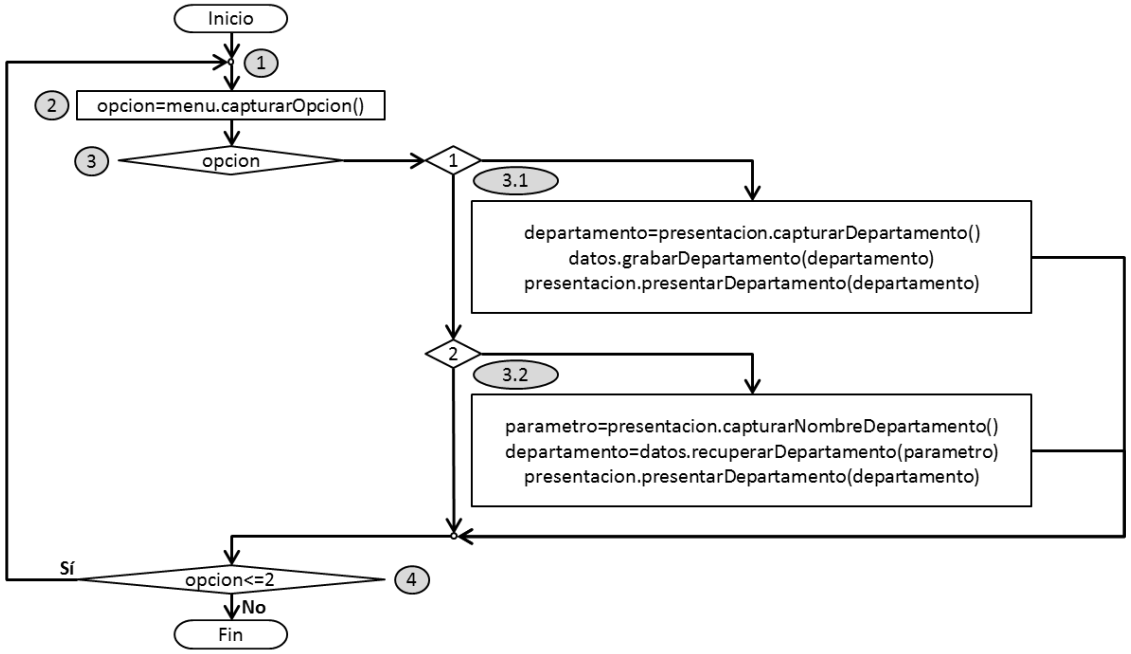


Figura 9.57 Diagrama de flujo para el método main()

9.3.3 Implementación

La estructura del proyecto no cambia, solo se incluye un método en la clase Departamento para la validación de los datos, que es reutilizado para su presentación y su grabación.

Tabla 9.33 Codificación de la clase *Departamento*

```
1  package proyecto;
2
3  public class Departamento {
4     private String nombre;
5     private int potencialSufragantes;
6     private int totalSufragantes;
7     private int votosValidos;
8
9     public Departamento(){
10       this.nombre="";
11       this.potencialSufragantes=0;
12       this.totalSufragantes=0;
13       this.votosValidos=0;
14    }
15
16    public void setNombre(String pNombre){
17       if (pNombre.length()>0){
18          this.nombre=pNombre;
19       }
20    }
21
22    public void setPotencialSufragantes(int pPotencialSufragantes){
23       if (pPotencialSufragantes>this.totalSufragantes){
24          this.potencialSufragantes=pPotencialSufragantes;
25       }
26    }
27
28    public void setTotalSufragantes(int pTotalSufragantes){
29       if ((pTotalSufragantes>=this.votosValidos) && (pTotalSufragantes<=this.potencialSufragantes)){
30          this.totalSufragantes=pTotalSufragantes;
31       }
32    }
33
34    public void setVotosValidos(int pVotosValidos){
35       if ((pVotosValidos>=0) && (pVotosValidos<=this.totalSufragantes)){
36          this.votosValidos=pVotosValidos;
37       }
38    }
39
40    public String getNombre(){
41       return this.nombre;
42    }
43
44    public int getPotencialSufragantes(){
45       return this.potencialSufragantes;
```

```
46      }
47
48      public int getTotalSufragantes(){
49         return this.totalSufragantes;
50      }
51
52      public int getVotosValidos(){
53         return this.votosValidos;
54      }
55
56      public double calcularParticipacion(){
57         double rParticipacion;
58         rParticipacion=((double)this.totalSufragantes/this.potencialSufragantes)*100;
59         return rParticipacion;
60      }
61
62      public String calcularCategoriaParticipacion(){
63         double tParticipacion;
64         String rCategoria;
65         tParticipacion=this.calcularParticipacion();
66         if (tParticipacion>=50){
67            rCategoria="Alta";
68         }
69         else{
70            if (tParticipacion>=40){
71               rCategoria="Media";
72            }
73            else{
74               rCategoria="Baja";
75            }
76         }
77         return rCategoria;
78      }
79
80      public int calcularVotosNoValidos(){
81         int rVotosNoValidos;
82         rVotosNoValidos=this.totalSufragantes-this.votosValidos;
83         return rVotosNoValidos;
84      }
85
86      public double calcularIndicadorError(){
87         double rIndicadorError;
88         rIndicadorError=(double)this.calcularVotosNoValidos()/this.totalSufragantes;
89         return rIndicadorError;
90      }
91
92      public boolean determinarValidez(){
93         boolean rValidez;
94         rValidez=((this.nombre.length()>0) &&
95                  (this.potencialSufragantes>0) &&
96                  (this.totalSufragantes>0) &&
97                  (this.totalSufragantes<=this.potencialSufragantes) &&
98                  (this.votosValidos>0) &&
```

```
99                  (this.votosValidos<=this.totalSufragantes));
100         return rValidez;
101      }
102   }
```

Tabla 9.34 Codificación de la clase *Presentacion*

```
1   package proyecto;
2   import javax.swing.JOptionPane;
3
4   public class Presentacion {
5      public Presentacion(){
6
7      }
8
9      public Departamento capturarDepartamento(){
10        Departamento rDepartamento;
11        rDepartamento=new Departamento();
12        rDepartamento.setNombre(JOptionPane.showInputDialog("Nombre: "));
13        try{
14           rDepartamento.setPotencialSufragantes(Integer.parseInt(JOptionPane.showInputDialog("Potencial
      de sufragantes: ")));
15           rDepartamento.setTotalSufragantes(Integer.parseInt(JOptionPane.showInputDialog("Total de
      sufragantes: ")));
16           rDepartamento.setVotosValidos(Integer.parseInt(JOptionPane.showInputDialog("Votos válidos: ")));
17        }
18        catch (Exception e){
19
20        }
21        return rDepartamento;
22     }
23
24     public String capturarNombreDepartamento(){
25        String rNombre;
26        rNombre=JOptionPane.showInputDialog("Departamento: ");
27        return rNombre;
28     }
29
30     public void presentarDepartamento(Departamento pDepartamento){
31        String rMensaje;
32        if (pDepartamento.determinarValidez()){
33           rMensaje="Departamento: "+pDepartamento.getNombre();
34           rMensaje=rMensaje+"\nPotencial de sufragantes: "+pDepartamento.getPotencialSufragantes();
35           rMensaje=rMensaje+"\nTotal de sufragantes: "+pDepartamento.getTotalSufragantes();
36           rMensaje=rMensaje+"\nVotos válidos: "+pDepartamento.getVotosValidos();
37           rMensaje=rMensaje+"\nIndicador de participación: "+pDepartamento.calcularParticipacion()+" %";
38           rMensaje=rMensaje+"\nCategoría según participación:
      "+pDepartamento.calcularCategoriaParticipacion();
39           rMensaje=rMensaje+"\nVotos no válidos: "+pDepartamento.calcularVotosNoValidos();
40           rMensaje=rMensaje+"\nIndicador de error: "+pDepartamento.calcularIndicadorError();
41        }
```

```
42          else{
43             rMensaje="Datos inconsistentes. No se pueden presentar";
44          }
45          JOptionPane.showMessageDialog(null, rMensaje);
46       }
47    }
```

Tabla 9.35 Codificación de la clase *Menu*

```
1     package proyecto;
2     import javax.swing.JOptionPane;
3
4     public class Menu {
5        public Menu(){
6
7        }
8
9        public int capturarOpcion(){
10          String tMensaje;
11          int rOpcion;
12          tMensaje="1. Registrar departamento\n"+
13                "2. Consultar departamento\n"+
14                "3. Salir\n";
15          rOpcion=Integer.parseInt(JOptionPane.showInputDialog(tMensaje));
16          return rOpcion;
17       }
18    }
```

Tabla 9.36 Codificación de la clase *Datos*

```
1     package proyecto;
2     import java.io.*;
3
4     public class Datos {
5        public Datos(){
6
7        }
8
9        public String construirRegistroDepartamento(Departamento pDepartamento){
10          String rRegistro;
11          rRegistro=pDepartamento.getNombre()+",";
12          rRegistro=rRegistro+pDepartamento.getPotencialSufragantes()+",";
13          rRegistro=rRegistro+pDepartamento.getTotalSufragantes()+",";
14          rRegistro=rRegistro+pDepartamento.getVotosValidos();
15          return rRegistro;
16       }
17
18       public String descomponerRegistro(String pRegistro, int pOrden){
19          int i,j;
```

```
20          String rDato;
21          i=0;
22          j=1;
23          rDato="";
24          while ((i<pRegistro.length()) && (j<=pOrden)) {
25             if (pRegistro.substring(i,i+1).compareTo(",")!=0){
26                if (j==pOrden){
27                   rDato=rDato+pRegistro.substring(i,i+1);
28                }
29             }
30             else{
31                j=j+1;
32             }
33             i=i+1;
34          }
35          return rDato;
36       }
37
38       public String recuperarRegistroDepartamento(String pNombre){
39          FileReader fileReader;
40          BufferedReader bufferedReader;
41          String rRegistro,tNombre;
42          try{
43             fileReader=new FileReader("departamentos.txt");
44             bufferedReader=new BufferedReader(fileReader);
45             do{
46                rRegistro=bufferedReader.readLine();
47                tNombre=this.descomponerRegistro(rRegistro, 1);
48             } while (tNombre.compareToIgnoreCase(pNombre)!=0);
49             fileReader.close();
50          }
51          catch (Exception e){
52             rRegistro="";
53          }
54          return rRegistro;
55       }
56
57       public Departamento recuperarDepartamento(String pNombre){
58          String tRegistro;
59          Departamento rDepartamento;
60          rDepartamento=new Departamento();
61          tRegistro=this.recuperarRegistroDepartamento(pNombre);
62          try{
63             rDepartamento.setNombre(this.descomponerRegistro(tRegistro,1));
64             rDepartamento.setPotencialSufragantes(Integer.parseInt(this.descomponerRegistro(tRegistro,2)));
65             rDepartamento.setTotalSufragantes(Integer.parseInt(this.descomponerRegistro(tRegistro,3)));
66             rDepartamento.setVotosValidos(Integer.parseInt(this.descomponerRegistro(tRegistro,4)));
67          }
68          catch (Exception e){
69
70          }
71          return rDepartamento;
72       }
```

```
73
74    public void grabarDepartamento(Departamento pDepartamento){
75       FileWriter fileWriter;
76       PrintWriter printWriter;
77       if (pDepartamento.determinarValidez()){
78          try{
79             fileWriter=new FileWriter("departamentos.txt",true);
80             printWriter=new PrintWriter(fileWriter);
81             printWriter.println(this.construirRegistroDepartamento(pDepartamento));
82             fileWriter.close();
83          }
84          catch(Exception e){
85
86          }
87       }
88    }
89 }
```

Tabla 9.37 Codificación de la clase *Main*

```
1  package proyecto;
2
3  public class Main {
4
5     public static void main(String[] args) {
6        Departamento departamento;
7        Presentacion presentacion;
8        Menu menu;
9        Datos datos;
10       int opcion;
11       String parametro;
12       presentacion=new Presentacion();
13       menu=new Menu();
14       datos=new Datos();
15       opcion=0;
16       do {
17          opcion=menu.capturarOpcion();
18          switch (opcion){
19             case 1:{
20                departamento=presentacion.capturarDepartamento();
21                datos.grabarDepartamento(departamento);
22                presentacion.presentarDepartamento(departamento);
23                break;
24             }
25             case 2:{
26                parametro=presentacion.capturarNombreDepartamento();
27                departamento=datos.recuperarDepartamento(parametro);
28                presentacion.presentarDepartamento(departamento);
29                break;
30             }
31          }
```

```
32          } while (opcion<=2);
33      }
34  }
```

9.4 Omisión de registros duplicados – Prototipo 11

9.4.1 Análisis

9.4.1.1 Alcance

El prototipo anterior permite que se graben en el archivo múltiples registros correspondientes al mismo departamento. Además de las medidas de aseguramiento de la consistencia de los datos, se requiere evitar la grabación de registros duplicados de un mismo departamento, para lo cual antes de la grabación se lleva a cabo una búsqueda de un registro anterior con el mismo nombre del que se pretende grabar. El prototipo 11 debe tener la misma funcionalidad del prototipo 10, agregando la implementación de medidas de protección del archivo, para evitar la grabación de registros duplicados para un mismo departamento.

9.4.1.2 Definiciones

Se mantienen las mismas definiciones. Aparece una nueva regla de validación: no puede haber un departamento con el mismo nombre que otro que ya esté registrado en archivo.

Tabla 9.38 Definiciones aplicables en desarrollo del prototipo 11.

	Definición	Tipo de dato	Condiciones de validación
Departamento	Cada una de las circunscripciones territoriales de primer nivel en que se divide el país para propósitos electorales	Objeto	
Nombre	Identificación nominal del departamento	Cadena de caracteres	Valor ingresado. El nombre del departamento no puede quedar vacío y no puede haber nombres de departamentos repetidos.
Potencial de sufragantes	Cantidad de personas habilitadas en un departamento para votar en un proceso electoral	Número entero	Valor ingresado. El potencial de sufragantes debe ser mayor que cero.
Total de sufragantes	Cantidad de personas que acudieron a votar en un departamento en un proceso electoral	Número entero	Valor ingresado. El total de sufragantes debe ser mayor que cero, pero menor o igual al potencial de sufragantes.
Indicador de participación electoral	Indicador consistente en el total de sufragantes expresado como porcentaje del potencial de sufragantes	Número real	Valor calculado. $Indicador\ de\ participación\ electoral = \frac{Total\ de\ sufragantes}{Potencial\ de\ sufragantes} * 100$
Categoría de participación electoral	Categoría a la que pertenece el departamento, dependiendo del	Cadena de caracteres	Valor calculado. Si el porcentaje es mayor o igual a 50%: alto

	porcentaje de participación electoral.		Si el porcentaje es menor a 50% pero mayor o igual a 40%: medio Si el porcentaje es menor a 40%: bajo
Votos válidos	Cantidad de votos que indican de forma clara la voluntad del sufragante	Número entero	Valor ingresado. La cantidad de votos válidos debe ser mayor que cero, pero menos o igual al total de sufragantes.
Votos no válidos	Cantidad de votos que NO indican de forma clara la voluntad del sufragante.	Número entero	Valor calculado. $Votos\ no\ válidos = Total\ de\ sufragantes - Votos\ válidos$
Indicador de error en la votación	Indicador consistente en dividir el total de votos no válidos entre el total de sufragantes.	Número real	Valor calculado. $Indicador\ de\ error\ en\ la\ votación = \frac{Votos\ no\ válidos}{Total\ de\ sufragantes}$

9.4.1.3 Requisitos

No hay modificación en los requisitos de interfaz de usuario, sin embargo el nuevo prototipo exige la implementación de medidas, para evitar la grabación de registros duplicados.

9.4.1.3.1 Requisitos de interfaz de usuario

9.4.1.3.1.1 Ingreso de información en casillas de edición

Para la interacción con el usuario, la aplicación debe presentar ventanas con casillas de edición en donde pueda digitar los datos solicitados. Al solicitar los datos, el sistema deberá indicar en la misma ventana de diálogo cuál es el dato requerido.

9.4.1.3.1.2 Presentación de menú de selección de funciones

La aplicación debe presentar un menú para seleccionar una de 3 funciones: registrar la información de un departamento para su almacenamiento, consultar información de in departamento a partir de la recuperación de sus datos almacenados, o finalizar la aplicación.

9.4.1.3.2 Requisitos funcionales

9.4.1.3.2.1 Calcular el indicador de participación electoral de un departamento

La aplicación debe permitir al usuario introducir los siguientes datos básicos de un departamento: el nombre, el potencial de sufragantes, el total de sufragantes y la cantidad de votos válidos, y con estos datos debe calcular el indicador de participación electoral, los votos no válidos y el indicador de error de votación. Los datos ingresados deben cumplir las condiciones de validación requeridas.

9.4.1.3.2.2 Almacenar la información de un departamento en un archivo plano

La aplicación debe poder almacenar en un archivo plano los datos básicos ingresados de un departamento: el nombre, el potencial de sufragantes, el total de sufragantes y el total de votos

válidos. Se requiere que el formato de almacenamiento sea en datos separados por punto y coma (formato csv). Se requiere aquí, la implementación de medidas de protección del archivo, para evitar la grabación de registros duplicados para un mismo departamento.

9.4.1.3.2.3 Recuperar la información de un departamento en un archivo plano

La aplicación debe poder recuperar desde un archivo plano los datos básicos ingresados de un departamento: el nombre, el potencial de sufragantes, el total de sufragantes, el total de votos válidos además de los datos calculados del indicador de participación electoral, la categoría de participación a la que pertenece, los votos no válidos y el indicador de error de votación y se debe presentar al usuario. Para ello deberá preguntar por el nombre del departamento a buscar. Como restricción, se requiere el formato de almacenamiento sea en datos separados por punto y coma (formato csv).

9.4.2 Diseño

9.4.2.1 Diseño de escenarios

Se mantiene el mismo diseño de escenarios.

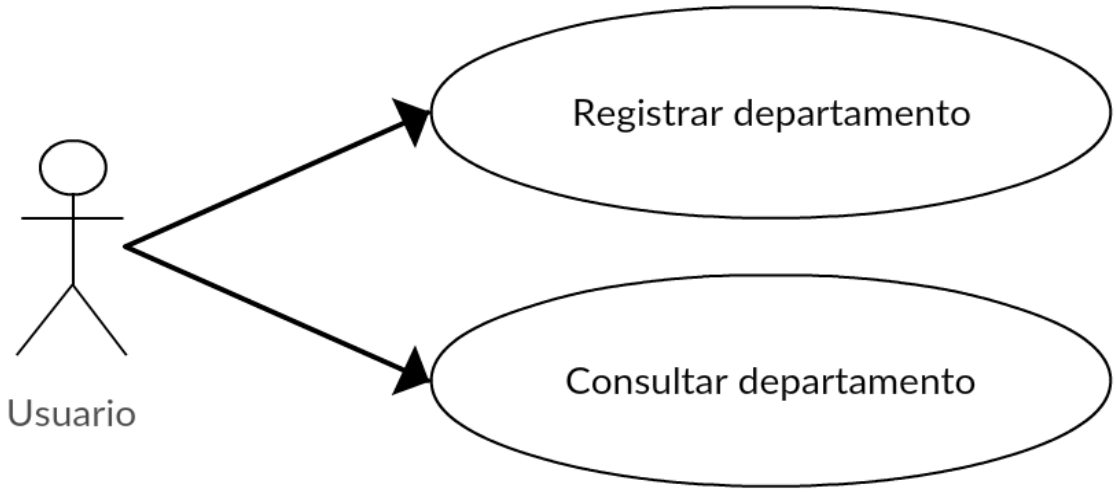


Figura 9.58 Diagrama de casos de uso para el Prototipo 11

Tabla 9.39 Especificación del caso de uso *Registrar departamento*.

1	El usuario ingresa a la aplicación.
2	La aplicación presenta el menú de selección, por lo que el usuario selecciona la opción 1: Registrar Departamento
2.1	Si el usuario selecciona una opción mayor que 2, la aplicación finaliza.
3	La aplicación presenta una pantalla con una caja de texto para que el usuario digite el nombre del departamento
4	El usuario digita el nombre del departamento y pulsa el botón *Aceptar*

5	La aplicación presenta una pantalla con una caja de texto para que el usuario digite el potencial de sufragantes del departamento
6	El usuario digita el potencial de sufragantes del departamento y pulsa el botón *Aceptar*
7	La aplicación presenta una pantalla con una caja de texto para que el usuario digite el total de sufragantes del departamento
8	El usuario digita el total de sufragantes del departamento y pulsa el botón *Aceptar*
9	La aplicación presenta una pantalla con una caja de texto para que el usuario digite la cantidad de votos válidos obtenidos en el departamento
10	El usuario digita la cantidad de votos válidos del departamento y pulsa el botón *Aceptar*
11	Si los datos son válidos: la aplicación muestra en pantalla el nombre del departamento, el potencial de sufragantes, el total de sufragantes, la cantidad de votos válidos, el indicador de participación electoral del departamento, la categoría de participación, la cantidad de votos no válidos y el indicador de error de votación en el departamento que acabó de ingresar.
12	Si los datos no son válidos: la aplicación muestra en pantalla el mensaje “Datos inconsistentes. No se pueden presentar”
13	La aplicación vuelve a presentar el menú de selección (paso 2)

Tabla 9.40 Especificación del caso de uso *Consultar departamento*.

1	El usuario ingresa a la aplicación.
2	La aplicación presenta el menú de selección, por lo que el usuario selecciona la opción 2: Consultar Departamento
2.1	Si el usuario selecciona una opción mayor que 2, la aplicación finaliza.
3	El usuario digita el nombre del departamento a buscar y pulsa el botón *Aceptar*
4	La aplicación muestra en pantalla el nombre del departamento buscado, el potencial de sufragantes, el total de sufragantes, la cantidad de votos válidos, el indicador de participación electoral del departamento, la categoría de participación, la cantidad de votos no válidos y el indicador de error de votación en el departamento.
5	La aplicación vuelve a presentar el menú de selección (paso 2)

9.4.2.2 Diseño de estructura

La estructura no se modifica. Los algoritmos de captura de datos y de recuperación de datos se modifican mediante una estructura de manejo de errores. Si por alguna razón, los datos capturados desde teclado o los leídos desde el archivo cuentan con errores, se resguarda la creación de objeto sin que la aplicación termine abruptamente.

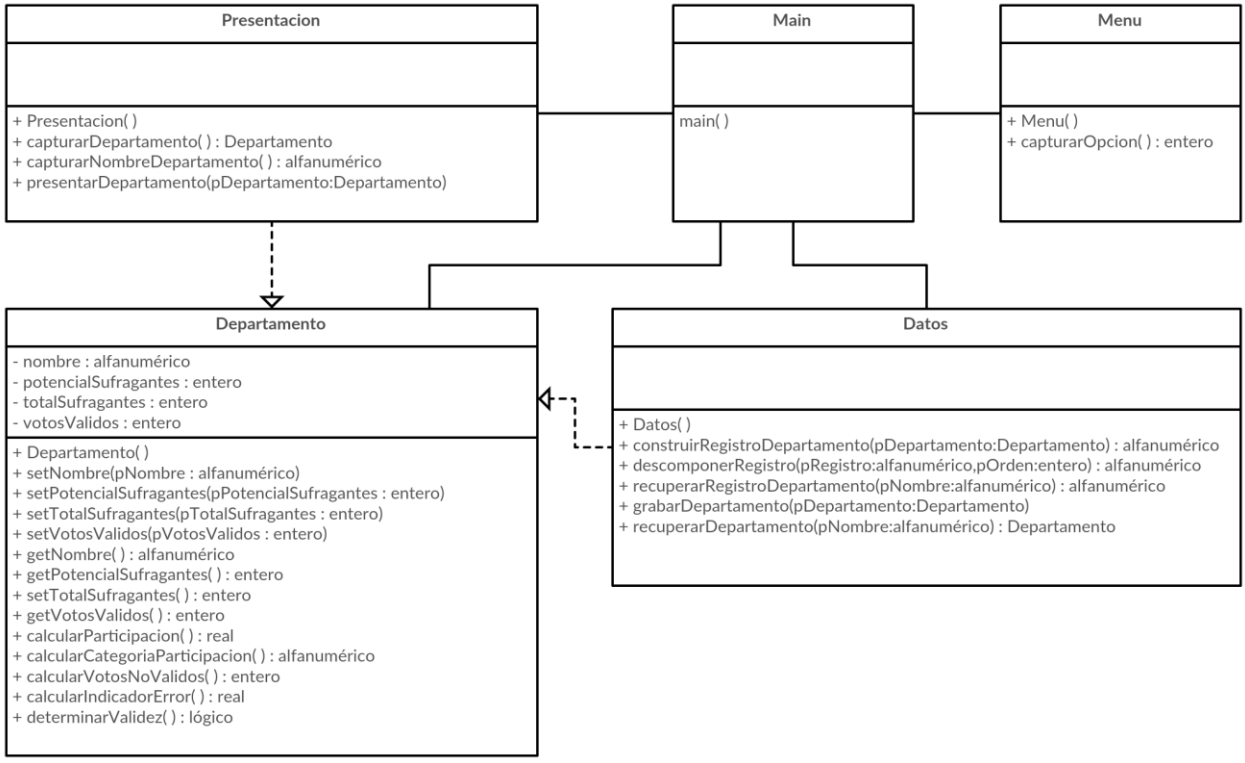


Figura 9.59 Diagrama de clases para el Prototipo 11

9.4.2.3 Diseño de algoritmos

9.4.2.3.1 Clase Departamento

La clase Departamento mantiene su especificación en el nuevo prototipo, Las especificaciones de diseño por lo tanto son las mismas del prototipo anterior. Ver secciones 9.2.2.3.1 a 9.2.2.3.14 del presente capítulo.

9.4.2.3.2 Clase Presentacion

La clase Presentacion mantiene su especificación en el nuevo prototipo, Las especificaciones de diseño por lo tanto son las mismas del prototipo anterior. Ver secciones 9.1.2.3.7 a 9.1.2.3.9 del presente capítulo.

9.4.2.3.3 Clase Menu - capturarOpcion(): entero

La clase Menu mantiene su estructura y por lo tanto la misma definición tal como se describe en la sección 8.4.2.3.9 del capítulo anterior.

9.4.2.3.4 Clase Datos - construirRegistroDepartamento(pDepartamento:Departamento): cadena

Descripción: Recibe un mensaje con un objeto de la clase Departamento como parámetro. Construye un registro, es decir una cadena de caracteres, a la que agrega cada uno de los valores de los atributos del objeto separados entre sí por un signo de coma. Por último lo entrega el registro como respuesta al objeto que haya hecho la solicitud, es decir, que haya invocado ejecución del método.

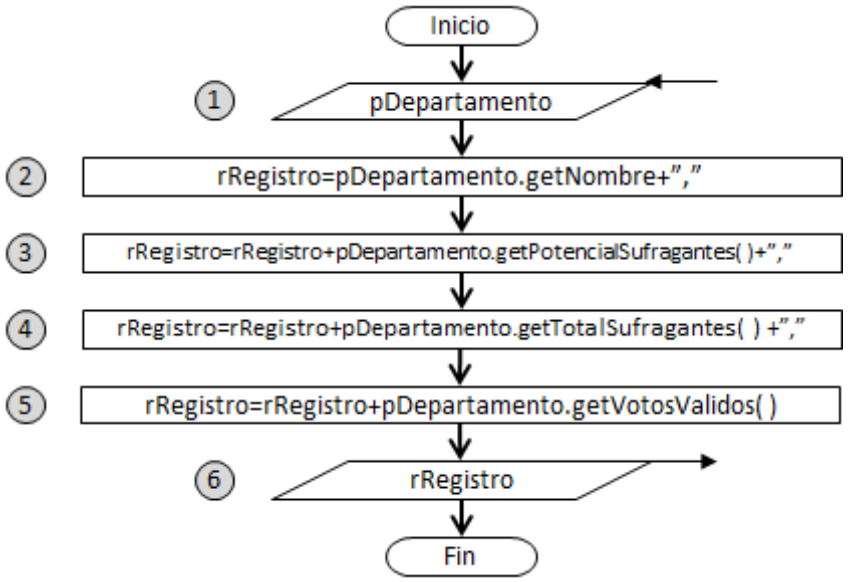


Figura 9.60 Diagrama de flujo para el método construirRegistroDepartamento ()

9.4.2.3.5 Clase Datos - descomponerRegistro (pRegistro:cadena, pOrden: entero):cadena

Descripción: Recibe un mensaje con dos parámetros; el primero de estos es un registro o cadena de caracteres que contiene valores separados entre sí por signos de coma; el segundo, un número ordinal que indica el dato que se debe extraer del registro y entregar como respuesta final. El método hace un recorrido secuencial por los caracteres del registro, y cuando ubica el comienzo del dato solicitado, comienza a agregar cada caracter en una variable que al final entrega como respuesta al objeto que haya requerido la ejecución del método.

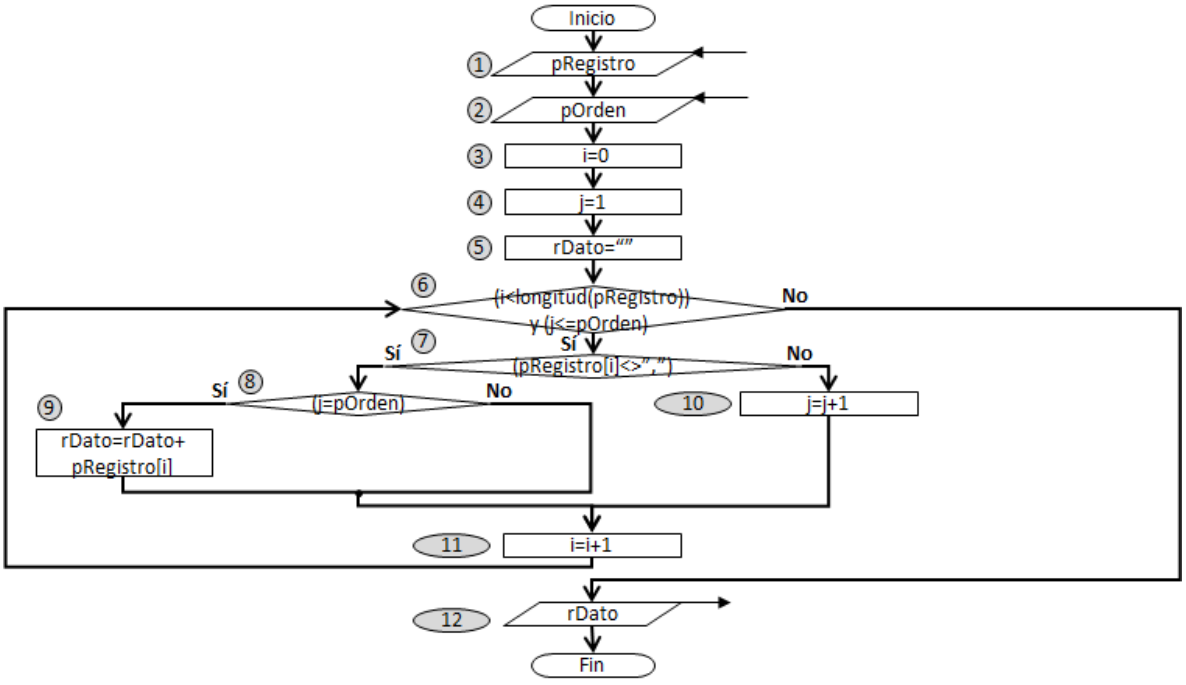


Figura 9.61 Diagrama de flujo para el método descomponerRegistro()

9.4.2.3.6 Clase Datos - grabarDepartamento(pDepartamento:Departamento

Descripción: Recibe un mensaje con un objeto de la clase Departamento como parámetro. Si el objeto cumple las condiciones de validación, y además se determina que no hay un registro válido de un departamento con el mismo nombre en el archivo, entonces abre el archivo y a continuación invoca el método de construcción del registro, cuya respuesta es grabada en el archivo que finalmente es cerrado; en caso contrario, omite la grabación del departamento.

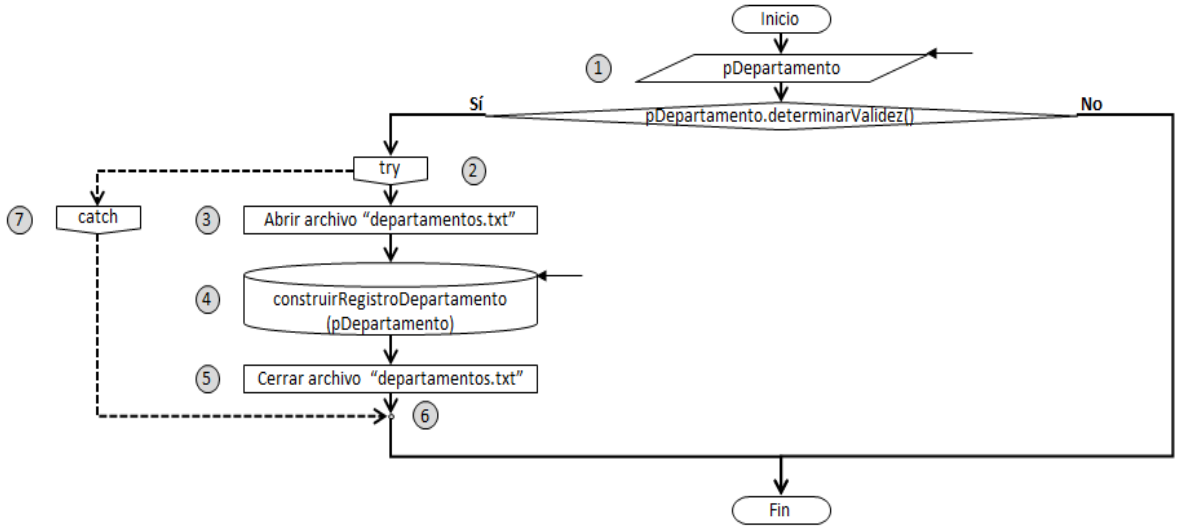


Figura 9.62 Diagrama de flujo para el método grabarDepartamento ()

Tabla 9.41 Descripción del algoritmo para el método grabarDepartamento ()

1	Se requiere como dato de entrada el objeto de tipo Departamento. En este caso se recibe el objeto como mensaje.
2	Se revisa la validez del objeto de tipo Departamento y se revisa la invalidez de un registro en el archivo con el nombre del departamento
3	Si el objeto SI es válido: Inicia la estructura de manejo de errores en caso de poder abrir el archivo para su escritura
4	Abre el archivo de texto para escritura
5	Escribe en el archivo la respuesta del método construirRegistroDepartamento()
6	Cierra el archivo de texto para asegurar la escritura
7-8	Inicia la estructura para la ejecución de acciones en caso de fallo por no poder abrir el archivo para escritura

9.4.2.3.7 Clase Datos - recuperarRegistroDepartamento(pNombre:cadena):cadena

Descripción: Recibe un mensaje con una cadena de caracteres como parámetro, que contiene un nombre de departamento a localizar en el archivo. Abre un archivo para lectura y a continuación lee uno a uno los registros hasta encontrar el que contiene los datos del departamento requerido, para lo cual solicita la descomposición del registro y la obtención del primer dato que corresponde al nombre. Por último entrega como respuesta el registro que contenga el nombre buscado.

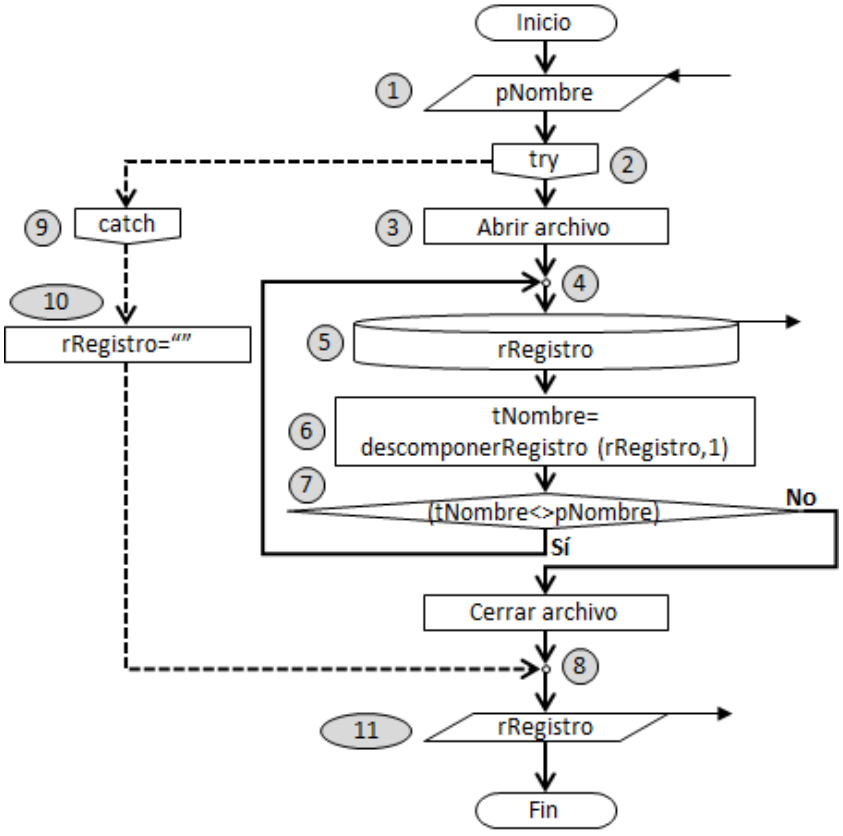


Figura 9.63 Diagrama de flujo para el método recuperarRegistroDepartamento()

9.4.2.3.8 Clase Datos - recuperarDepartamento(pNombre:cadena):Departamento

Descripción: Recibe un mensaje con una cadena de caracteres como parámetro, que contiene un nombre de departamento a localizar en el archivo. Solicita la ejecución del método de recuperación de registro mediante el nombre dado, y una vez obtenida la respuesta de aquel método, toma el registro hallado y solicita su descomposición en cuatro partes. Por último entrega como respuesta el objeto de la clase Departamento correspondiente al nombre dado como criterio de búsqueda.

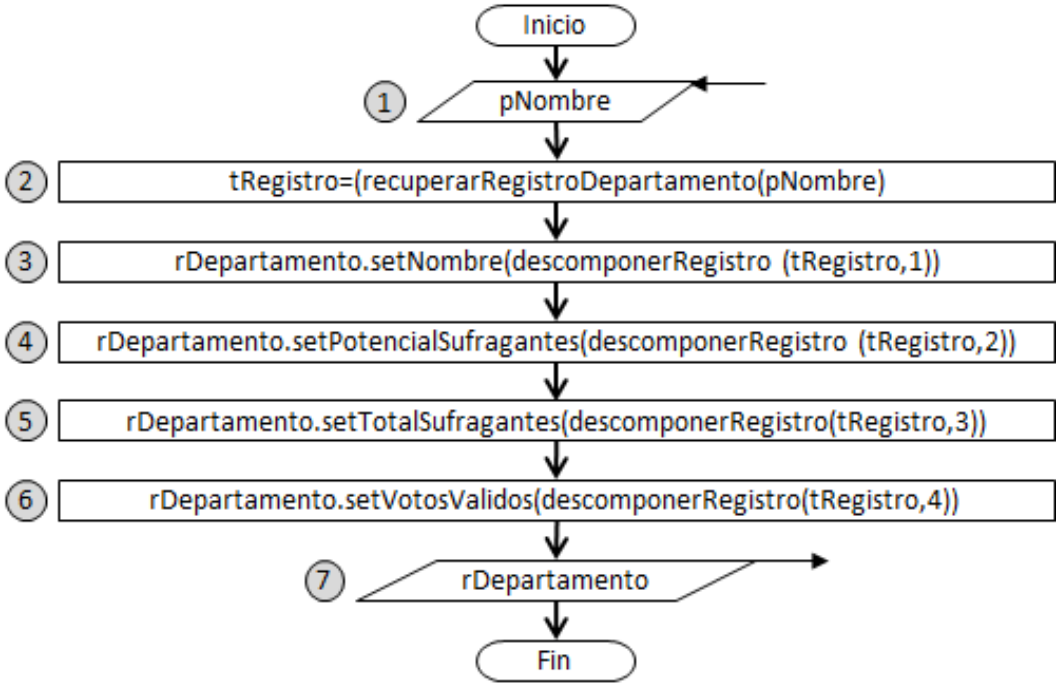


Figura 9.64 Diagrama de flujo para el método recuperarDepartamento()

9.4.2.3.9 Clase Main – main( )

Descripción: Coordina las acciones de las demás clases. Solicita a un objeto de la clase Menu que reciba de parte del usuario la opción requerida por él, que almacena en una variable.

Si la opción seleccionada es 1, coordina las clases para que el usuario pueda registrar un departamento, es decir, solicita a un objeto de la clase Presentacion que lleve a cabo la captura de un objeto de la clase Departamento, que almacena luego en el objeto departamento, solicita al objeto de la clase Datos que grabe el objeto departamento y finalmente muestra la información del departamento ingresado.

Si la opción seleccionada es 2, coordina las clases para que el usuario pueda consultar dando como parámetro un nombre de departamento, solicita al objeto datos la recuperación de un

departamento mediante el nombre capturado por el objeto presentación; el resultado de la operación de recuperación se almacena en el objeto departamento. Por último ordena al objeto de la clase Presentacion que lleve a cabo el método de presentación tomando como parámetro el objeto de la clase Departamento.

Todo esto se repite mientras la opción seleccionada sea menor o igual que 2.

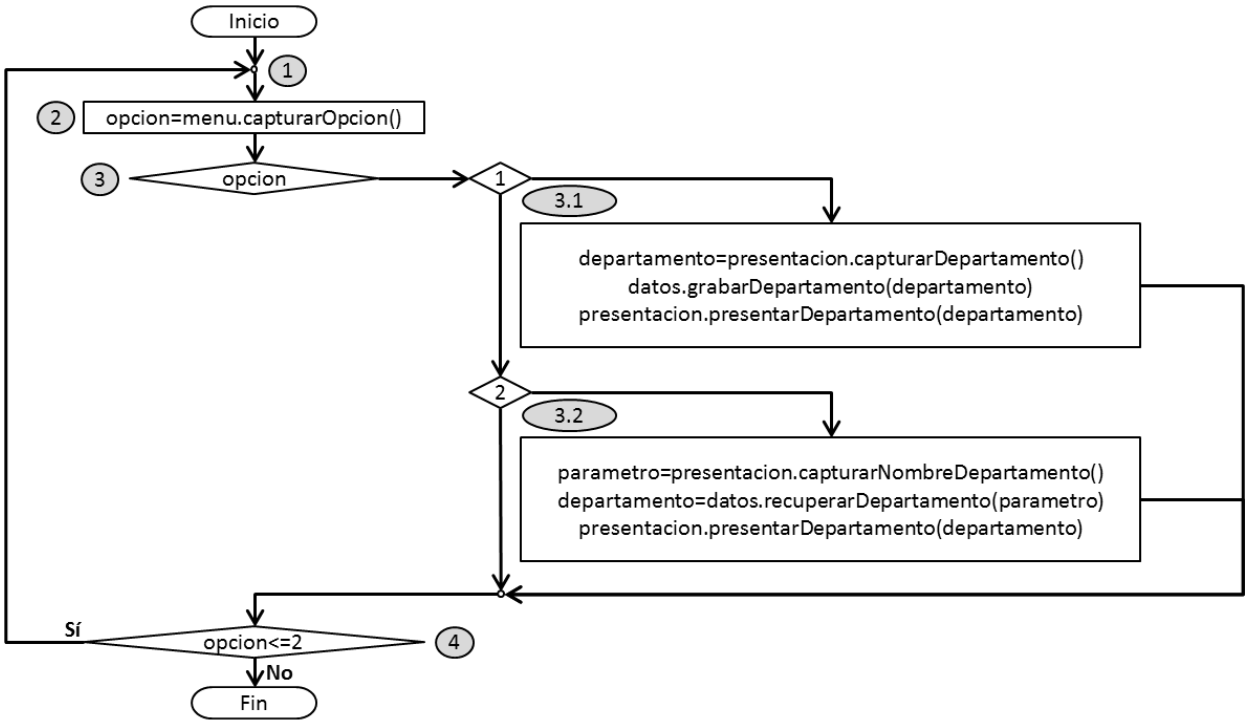


Figura 9.65 Diagrama de flujo para el método main()

9.4.3 Implementación

La estructura del proyecto no cambia, solo se incluye un método en la clase Departamento para la validación de los datos, que es reutilizado para su presentación y su grabación.

Tabla 9.42 Codificación de la clase *Departamento*

```
1  package proyecto;
2
3  public class Departamento {
4     private String nombre;
5     private int potencialSufragantes;
6     private int totalSufragantes;
7     private int votosValidos;
8
9     public Departamento(){
```

```
10      this.nombre="";
11      this.potencialSufragantes=0;
12      this.totalSufragantes=0;
13      this.votosValidos=0;
14   }
15
16   public void setNombre(String pNombre){
17      if (pNombre.length()>0){
18         this.nombre=pNombre;
19      }
20   }
21
22   public void setPotencialSufragantes(int pPotencialSufragantes){
23      if (pPotencialSufragantes>this.totalSufragantes){
24         this.potencialSufragantes=pPotencialSufragantes;
25      }
26   }
27
28   public void setTotalSufragantes(int pTotalSufragantes){
29      if ((pTotalSufragantes>=this.votosValidos) && (pTotalSufragantes<=this.potencialSufragantes)){
30         this.totalSufragantes=pTotalSufragantes;
31      }
32   }
33
34   public void setVotosValidos(int pVotosValidos){
35      if ((pVotosValidos>=0) && (pVotosValidos<=this.totalSufragantes)){
36         this.votosValidos=pVotosValidos;
37      }
38   }
39
40   public String getNombre(){
41      return this.nombre;
42   }
43
44   public int getPotencialSufragantes(){
45      return this.potencialSufragantes;
46   }
47
48   public int getTotalSufragantes(){
49      return this.totalSufragantes;
50   }
51
52   public int getVotosValidos(){
53      return this.votosValidos;
54   }
55
56   public double calcularParticipacion(){
57      double rParticipacion;
58      rParticipacion=((double)this.totalSufragantes/this.potencialSufragantes)*100;
59      return rParticipacion;
60   }
61
62   public String calcularCategoriaParticipacion(){
```

```
63          double tParticipacion;
64          String rCategoria;
65          tParticipacion=this.calcularParticipacion();
66          if (tParticipacion>=50){
67             rCategoria="Alta";
68          }
69          else{
70             if (tParticipacion>=40){
71                rCategoria="Media";
72             }
73             else{
74                rCategoria="Baja";
75             }
76          }
77          return rCategoria;
78       }
79
80       public int calcularVotosNoValidos(){
81          int rVotosNoValidos;
82          rVotosNoValidos=this.totalSufragantes-this.votosValidos;
83          return rVotosNoValidos;
84       }
85
86       public double calcularIndicadorError(){
87          double rIndicadorError;
88          rIndicadorError=(double)this.calcularVotosNoValidos()/this.totalSufragantes;
89          return rIndicadorError;
90       }
91
92       public boolean determinarValidez(){
93          boolean rValidez;
94          rValidez=((this.nombre.length()>0) &&
95                (this.potencialSufragantes>0) &&
96                (this.totalSufragantes>0) &&
97                (this.totalSufragantes<=this.potencialSufragantes) &&
98                (this.votosValidos>0) &&
99                (this.votosValidos<=this.totalSufragantes));
100         return rValidez;
101      }
102   }
```

Tabla 9.43 Codificación de la clase *Presentacion*

```
1   package proyecto;
2   import javax.swing.JOptionPane;
3
4   public class Presentacion {
5      public Presentacion(){
6
7      }
8
```

```
9     public Departamento capturarDepartamento(){
10       Departamento rDepartamento;
11       rDepartamento=new Departamento();
12       rDepartamento.setNombre(JOptionPane.showInputDialog("Nombre: "));
13       try{
14          rDepartamento.setPotencialSufragantes(Integer.parseInt(JOptionPane.showInputDialog("Potencial
    de sufragantes: ")));
15          rDepartamento.setTotalSufragantes(Integer.parseInt(JOptionPane.showInputDialog("Total de
    sufragantes: ")));
16          rDepartamento.setVotosValidos(Integer.parseInt(JOptionPane.showInputDialog("Votos válidos: ")));
17       }
18       catch (Exception e){
19
20       }
21       return rDepartamento;
22    }
23
24    public String capturarNombreDepartamento(){
25       String rNombre;
26       rNombre=JOptionPane.showInputDialog("Departamento: ");
27       return rNombre;
28    }
29
30    public void presentarDepartamento(Departamento pDepartamento){
31       String rMensaje;
32       if (pDepartamento.determinarValidez()){
33          rMensaje="Departamento: "+pDepartamento.getNombre();
34          rMensaje=rMensaje+"\nPotencial de sufragantes: "+pDepartamento.getPotencialSufragantes();
35          rMensaje=rMensaje+"\nTotal de sufragantes: "+pDepartamento.getTotalSufragantes();
36          rMensaje=rMensaje+"\nVotos válidos: "+pDepartamento.getVotosValidos();
37          rMensaje=rMensaje+"\nIndicador de participación: "+pDepartamento.calcularParticipacion()+" %";
38          rMensaje=rMensaje+"\nCategoría según participación:
    "+pDepartamento.calcularCategoriaParticipacion();
39          rMensaje=rMensaje+"\nVotos no válidos: "+pDepartamento.calcularVotosNoValidos();
40          rMensaje=rMensaje+"\nIndicador de error: "+pDepartamento.calcularIndicadorError();
41       }
42       else{
43          rMensaje="Datos inconsistentes. No se pueden presentar";
44       }
45       JOptionPane.showMessageDialog(null, rMensaje);
46    }
47 }
```

Tabla 9.44 Codificación de la clase *Menu*

```
1  package proyecto;
2  import javax.swing.JOptionPane;
3
4  public class Menu {
5     public Menu(){
6
```

```
7     }
8
9     public int capturarOpcion(){
10       String tMensaje;
11       int rOpcion;
12       tMensaje="1. Registrar departamento\n"+
13             "2. Consultar departamento\n"+
14             "3. Salir\n";
15       rOpcion=Integer.parseInt(JOptionPane.showInputDialog(tMensaje));
16       return rOpcion;
17    }
18 }
```

Tabla 9.45 Codificación de la clase *Datos*

```
1  package proyecto;
2  import java.io.*;
3
4  public class Datos {
5     public Datos(){
6
7     }
8
9     public String construirRegistroDepartamento(Departamento pDepartamento){
10       String rRegistro;
11       rRegistro=pDepartamento.getNombre()+",";
12       rRegistro=rRegistro+pDepartamento.getPotencialSufragantes()+",";
13       rRegistro=rRegistro+pDepartamento.getTotalSufragantes()+",";
14       rRegistro=rRegistro+pDepartamento.getVotosValidos();
15       return rRegistro;
16    }
17
18    public String descomponerRegistro(String pRegistro, int pOrden){
19       int i,j;
20       String rDato;
21       i=0;
22       j=1;
23       rDato="";
24       while ((i<pRegistro.length()) && (j<=pOrden)) {
25          if (pRegistro.substring(i,i+1).compareTo(",")!=0){
26             if (j==pOrden){
27                rDato=rDato+pRegistro.substring(i,i+1);
28             }
29          }
30          else{
31             j=j+1;
32          }
33          i=i+1;
34       }
35       return rDato;
36    }
```

```
37
38    public String recuperarRegistroDepartamento(String pNombre){
39       FileReader fileReader;
40       BufferedReader bufferedReader;
41       String rRegistro,tNombre;
42       try{
43          fileReader=new FileReader("departamentos.txt");
44          bufferedReader=new BufferedReader(fileReader);
45          do{
46             rRegistro=bufferedReader.readLine();
47             tNombre=this.descomponerRegistro(rRegistro, 1);
48          } while (tNombre.compareToIgnoreCase(pNombre)!=0);
49          fileReader.close();
50       }
51       catch (Exception e){
52          rRegistro="";
53       }
54       return rRegistro;
55    }
56
57    public Departamento recuperarDepartamento(String pNombre){
58       String tRegistro;
59       Departamento rDepartamento;
60       rDepartamento=new Departamento();
61       tRegistro=this.recuperarRegistroDepartamento(pNombre);
62       try{
63          rDepartamento.setNombre(this.descomponerRegistro(tRegistro,1));
64          rDepartamento.setPotencialSufragantes(Integer.parseInt(this.descomponerRegistro(tRegistro,2)));
65          rDepartamento.setTotalSufragantes(Integer.parseInt(this.descomponerRegistro(tRegistro,3)));
66          rDepartamento.setVotosValidos(Integer.parseInt(this.descomponerRegistro(tRegistro,4)));
67       }
68       catch (Exception e){
69
70       }
71       return rDepartamento;
72    }
73
74    public void grabarDepartamento(Departamento pDepartamento){
75       FileWriter fileWriter;
76       PrintWriter printWriter;
77       if ((pDepartamento.determinarValidez()) &&
78          (!this.recuperarDepartamento(pDepartamento.getNombre()).determinarValidez())){
79          try{
80             fileWriter=new FileWriter("departamentos.txt",true);
81             printWriter=new PrintWriter(fileWriter);
82             printWriter.println(this.construirRegistroDepartamento(pDepartamento));
83             fileWriter.close();
84          }
85          catch(Exception e){
86
87          }
88       }
89    }
```

```
90  }
```

Tabla 9.46 Codificación de la clase *Main*

```
1   package proyecto;
2
3   public class Main {
4
5      public static void main(String[] args) {
6         Departamento departamento;
7         Presentacion presentacion;
8         Menu menu;
9         Datos datos;
10        int opcion;
11        String parametro;
12        presentacion=new Presentacion();
13        menu=new Menu();
14        datos=new Datos();
15        opcion=0;
16        do {
17           opcion=menu.capturarOpcion();
18           switch (opcion){
19              case 1:{
20                 departamento=presentacion.capturarDepartamento();
21                 datos.grabarDepartamento(departamento);
22                 presentacion.presentarDepartamento(departamento);
23                 break;
24              }
25              case 2:{
26                 parametro=presentacion.capturarNombreDepartamento();
27                 departamento=datos.recuperarDepartamento(parametro);
28                 presentacion.presentarDepartamento(departamento);
29                 break;
30              }
31           }
32        } while (opcion<=2);
33     }
34  }
```

9.5 Síntesis de conceptos

9.5.1 Orientación a objetos

- **Encapsulamiento, ocultación, *setters* y *getters***

Una de las características de POO es que permite el mantenimiento de los programas y ya que tienen una organización que permite que se identifiquen la solución a los problemas en términos más reales, identificando objetos facilita a las personas generar un programa organizado por medio de objetos, ya que se tiende a visualizar los problemas teniendo en cuenta objetos del

mundo real. Los objetos proporcionan **encapsulamiento**. Este término hace referencia a los datos contenidos en el objeto están protegidos u ocultos dentro de este. Los datos están dentro de una capa protectora, la cual no permite acceder a ellos. Pero como se puede generar una interacción con el resto del programa, si es necesario acceder a estos datos desde otro objeto, es allí donde se requiere de una serie de métodos del objeto.

Cuando se necesita acceder a los diferentes datos, se requiere la ayuda de los métodos, existen métodos que permiten mostrar o modificar el valor de un atributo. Específicamente cuando se requiere mostrar un valor o devolver el valor de un atributo, se hace referencia a métodos ***getters***. Cuando se requiere modificar o ingresar el valor de un atributo se hace referencia a métodos ***setters***. Estos métodos deben ser de acceso público, ya que en ocasiones se requiere acceder a los datos desde fuera de la clase donde están contenidos.

Un ejemplo de uso de los métodos ***get*** y los métodos ***set*** se presenta a continuación:

```
1  public tipo_dato_atributo getAtributo (){
2      return atributo;
3  }
4
5  public void setAtributo (tipo_dato_atributo variable){
6     this.atributo = variable;
7  }
```

En el ejemplo anterior se puede observar la utilización del prefijo ***this***, este se utiliza para referirse a dos objetos de forma segura y comprensible. ***this*** se requiere a un objeto que es llamado y se tiene un prefijo de referencia para hacer referencia al otro objeto. El uso de este prefijo se hace necesario para evitar ambigüedad, podría omitirse en caso de no presentarse ambigüedad.

9.5.2 Programación en Java – Buenas prácticas

- **Validación de datos**

Cuando se desarrolla un software es necesario contemplar todas las posibilidades de errores que se pudieran presentar, esto se debe realizar por medio de las **validaciones de datos**. Una de estas validaciones, es la validación de entrada, donde se realiza una verificación a los datos ingresados del usuario. En caso de que los datos sean correctos el sistema continúa, de no ser así se debe entrar en un ciclo que le indique al usuario que los datos que ingreso son erróneos y solicitar ingresar nuevamente los datos. Uno de los ciclos que se pueden incluir es el ciclo indefinido *while*, donde la condición que debe contener es que se contemplen todos los posibles accesos erróneos y así mismo solicite nuevamente los datos.

10 Arreglos de datos

En los capítulos anteriores se ha evidenciado la evolución en el desarrollo de una aplicación para el análisis de la participación electoral en Colombia, a desarrollarse mediante el paradigma de Programación Orientada a Objetos y aplicando la Arquitectura de Capas. En estos momentos el prototipo 11 presenta funcionalidades útiles para la captura, almacenamiento, recuperación y presentación de los datos ingresados de votos por departamento. Ahora lo que se busca es mejorar la eficiencia algorítmica para el manejo de dichos datos a través de arreglos. La Tabla 4.4 muestra las metas de aprendizaje para estos prototipos.

Tabla 10.1 Objetivos de aprendizaje del capítulo.

Desarrollo de software	Conceptos	Instrumentos
Análisis Diseño Implementación	Algoritmo: arreglos y manejo de arreglos	UML: Diagrama de casos de uso Especificación de casos de uso Diagrama de clases Diagrama de flujo

10.1 Nuevas categorías de clasificación – Prototipo 12

10.1.1 Análisis

10.1.1.1 Alcance

El prototipo 12 debe tener la misma funcionalidad del prototipo 11, pero debe tener en cuenta una mayor cantidad de categorías de clasificación de los departamentos según su índice de participación. Esto se logrará mediante más niveles de anidamiento de condicionales.

10.1.1.2 Definiciones

Se mantienen las mismas definiciones. Se modifica una condición de validación, que es la de la determinación de la categoría de participación electoral, la cual se clasificará de acuerdo con la siguiente tabla:

Tabla 10.2 Categorías de participación electoral.

Indicador de participación electoral	Categoría de participación electoral
Mayor que 60%	Muy alta
Menor de 60% pero mayor o igual que 50%	Alta
Menor de 50% pero mayor o igual de 40%	Media
Menor que 40% pero mayor o igual a 30%	Baja
Menor que 30%	Muy baja

Tabla 10.3 Definiciones aplicables en desarrollo del prototipo 11.

	Definición	Tipo de dato	Condiciones de validación
Departamento	Cada una de las circunscripciones territoriales de primer nivel en que se divide el país para propósitos electorales	Objeto	
Nombre	Identificación nominal del departamento	Cadena de caracteres	Valor ingresado. El nombre del departamento no puede quedar vacío y no puede haber nombres de departamentos repetidos.
Potencial de sufragantes	Cantidad de personas habilitadas en un departamento para votar en un proceso electoral	Número entero	Valor ingresado. El potencial de sufragantes debe ser mayor que cero.
Total de sufragantes	Cantidad de personas que acudieron a votar en un departamento en un proceso electoral	Número entero	Valor ingresado. El total de sufragantes debe ser mayor que cero, pero menor o igual al potencial de sufragantes.
Indicador de participación electoral	Indicador consistente en el total de sufragantes expresado como porcentaje del potencial de sufragantes	Número real	Valor calculado. $Indicador\ de\ participación\ electoral = \frac{Total\ de\ sufragantes}{Potencial\ de\ sufragantes} * 100$
Categoría de participación electoral	Categoría a la que pertenece el departamento, dependiendo del porcentaje de participación electoral.	Cadena de caracteres	Valor calculado. Si el porcentaje es mayor o igual a 60%: muy alto Si el porcentaje es menor a 60% pero mayor o igual a 50%: alto Si el porcentaje es menor a 50% pero mayor o igual a 40%: medio Si el porcentaje es menor a 40% pero mayor o igual a 30%: bajo Si el porcentaje es menor a 30%: muy bajo
Votos válidos	Cantidad de votos que indican de forma clara la voluntad del sufragante	Número entero	Valor ingresado. La cantidad de votos válidos debe ser mayor que cero, pero menos o igual al total de sufragantes.
Votos no válidos	Cantidad de votos que NO indican de forma clara la voluntad del sufragante.	Número entero	Valor calculado. $Votos\ no\ válidos = Total\ de\ sufragantes - Votos\ válidos$
Indicador de error en la votación	Indicador consistente en dividir el total de votos no válidos entre el total de sufragantes.	Número real	Valor calculado. $Indicador\ de\ error\ en\ la\ votación = \frac{Votos\ no\ válidos}{Total\ de\ sufragantes}$

10.1.1.3 Requisitos

No hay modificación en los requisitos de interfaz de usuario ni en los funcionales.

10.1.1.3.1 Requisitos de interfaz de usuario

10.1.1.3.1.1 Ingreso de información en casillas de edición

Para la interacción con el usuario, la aplicación debe presentar ventanas con casillas de edición en donde pueda digitar los datos solicitados. Al solicitar los datos, el sistema deberá indicar en la misma ventana de diálogo cuál es el dato requerido.

10.1.1.3.1.2 Presentación de menú de selección de funciones

La aplicación debe presentar un menú para seleccionar una de 3 funciones: registrar la información de un departamento para su almacenamiento, consultar información de in departamento a partir de la recuperación de sus datos almacenados, o finalizar la aplicación.

10.1.1.3.2 Requisitos funcionales

10.1.1.3.2.1 Calcular el indicador de participación electoral de un departamento

La aplicación debe permitir al usuario introducir los siguientes datos básicos de un departamento: el nombre, el potencial de sufragantes, el total de sufragantes y la cantidad de votos válidos, y con estos datos debe calcular el indicador de participación electoral, los votos no válidos y el indicador de error de votación. Los datos ingresados deben cumplir las condiciones de validación requeridas.

10.1.1.3.2.2 Almacenar la información de un departamento en un archivo plano

La aplicación debe poder almacenar en un archivo plano los datos básicos ingresados de un departamento: el nombre, el potencial de sufragantes, el total de sufragantes y el total de votos válidos. Se requiere que el formato de almacenamiento sea en datos separados por punto y coma (formato csv).

10.1.1.3.2.3 Recuperar la información de un departamento en un archivo plano

La aplicación debe poder recuperar desde un archivo plano los datos básicos ingresados de un departamento: el nombre, el potencial de sufragantes, el total de sufragantes, el total de votos válidos además de los datos calculados del indicador de participación electoral, la categoría de participación a la que pertenece, los votos no válidos y el indicador de error de votación y se debe presentar al usuario. Para ello deberá preguntar por el nombre del departamento a buscar. Como restricción, se requiere el formato de almacenamiento sea en datos separados por punto y coma (formato csv).

10.1.2 Diseño

10.1.2.1 Diseño de escenarios

Se mantiene el mismo diseño de escenarios. A pesar de que cambian las reglas de cálculo, no se notarán diferencias desde el punto de vista del usuario.

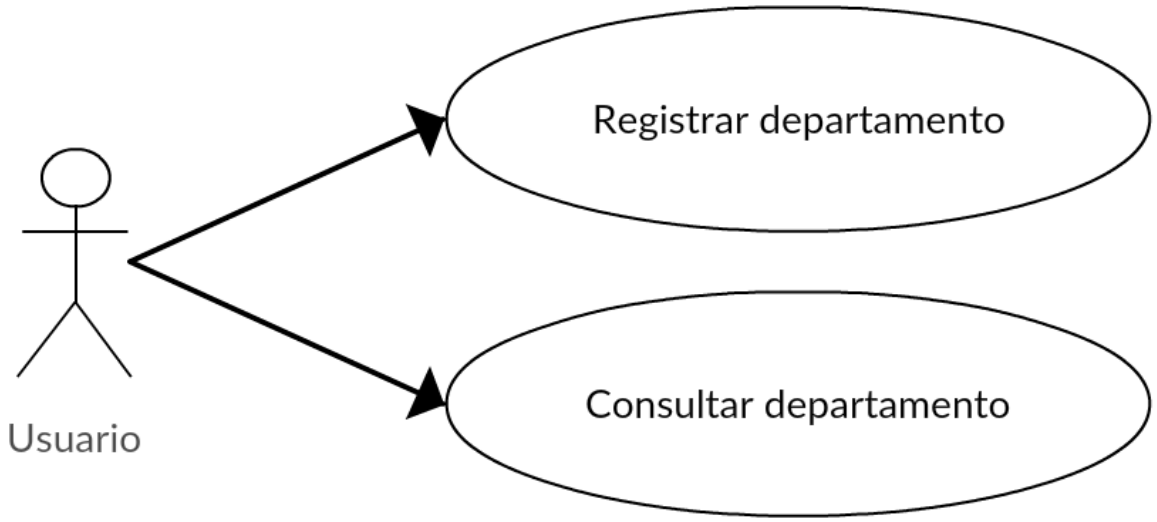


Figura 10.1 Diagrama de casos de uso para el Prototipo 12

Tabla 10.4 Especificación del caso de uso *Registrar departamento*.

1	El usuario ingresa a la aplicación.
2	La aplicación presenta el menú de selección, por lo que el usuario selecciona la opción 1: Registrar Departamento
2.1	Si el usuario selecciona una opción mayor que 2, la aplicación finaliza.
3	La aplicación presenta una pantalla con una caja de texto para que el usuario digite el nombre del departamento
4	El usuario digita el nombre del departamento y pulsa el botón *Aceptar*
5	La aplicación presenta una pantalla con una caja de texto para que el usuario digite el potencial de sufragantes del departamento
6	El usuario digita el potencial de sufragantes del departamento y pulsa el botón *Aceptar*
7	La aplicación presenta una pantalla con una caja de texto para que el usuario digite el total de sufragantes del departamento
8	El usuario digita el total de sufragantes del departamento y pulsa el botón *Aceptar*
9	La aplicación presenta una pantalla con una caja de texto para que el usuario digite la cantidad de votos válidos obtenidos en el departamento
10	El usuario digita la cantidad de votos válidos del departamento y pulsa el botón *Aceptar*
11	Si los datos son válidos: la aplicación muestra en pantalla el nombre del departamento, el potencial de sufragantes, el total de sufragantes, la cantidad de votos válidos, el indicador de participación electoral del departamento, la categoría de participación, la cantidad de votos no válidos y el indicador de error de votación en el departamento que acabó de ingresar.
12	Si los datos no son válidos: la aplicación muestra en pantalla el mensaje “Datos inconsistentes. No se pueden presentar”
13	La aplicación vuelve a presentar el menú de selección (paso 2)

Tabla 10.5 Especificación del caso de uso *Consultar departamento*.

1	El usuario ingresa a la aplicación.
2	La aplicación presenta el menú de selección, por lo que el usuario selecciona la opción 2: Consultar Departamento
2.1	Si el usuario selecciona una opción mayor que 2, la aplicación finaliza.

3	El usuario digita el nombre del departamento a buscar y pulsa el botón *Aceptar*
4	La aplicación muestra en pantalla el nombre del departamento buscado, el potencial de sufragantes, el total de sufragantes, la cantidad de votos válidos, el indicador de participación electoral del departamento, la categoría de participación, la cantidad de votos no válidos y el indicador de error de votación en el departamento.
5	La aplicación vuelve a presentar el menú de selección (paso 2)

10.1.2.2 Diseño de estructura

La estructura no se modifica. Solo se modificará el método que permite clasificar y determinar la categoría de participación, volviéndolo más complejo.

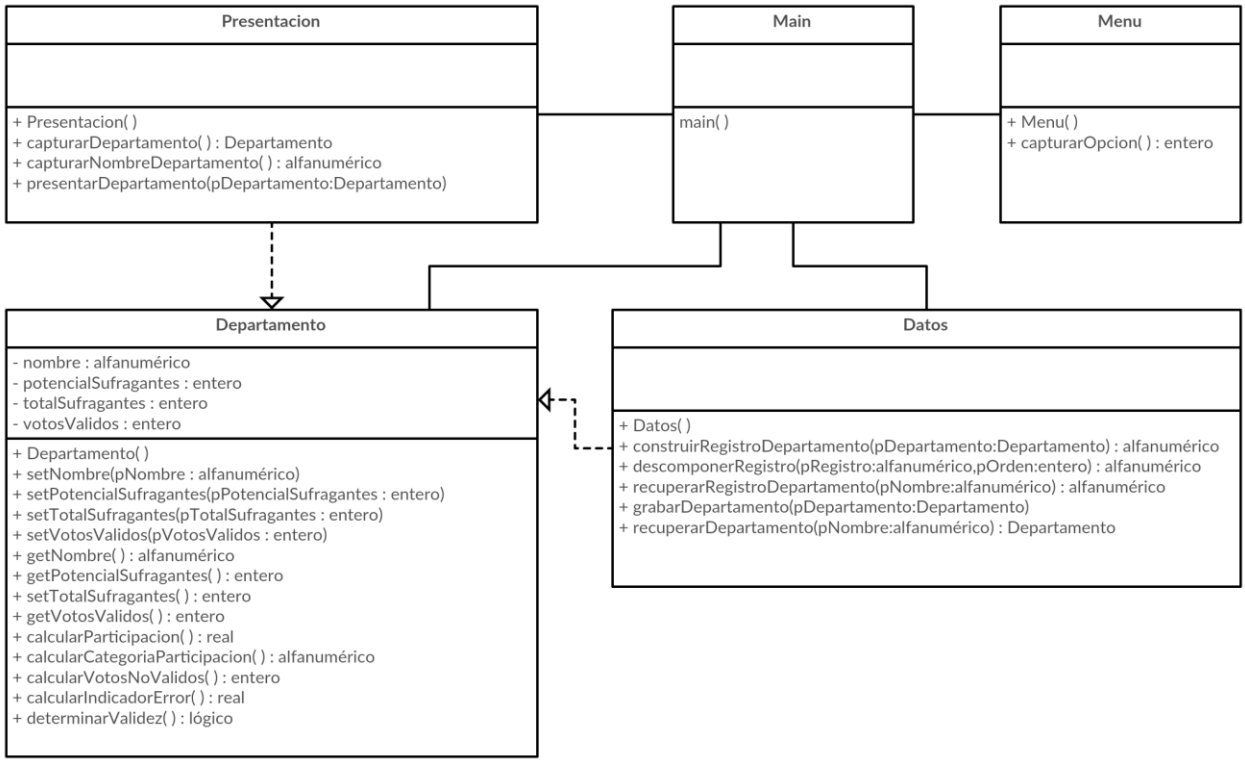


Figura 10.2 Diagrama de clases para el Prototipo 12

10.1.2.3 Diseño de algoritmos

10.1.2.3.1 Clase Departamento - Departamento( )

Descripción: Constructor que inicializa los atributos con valores neutros.

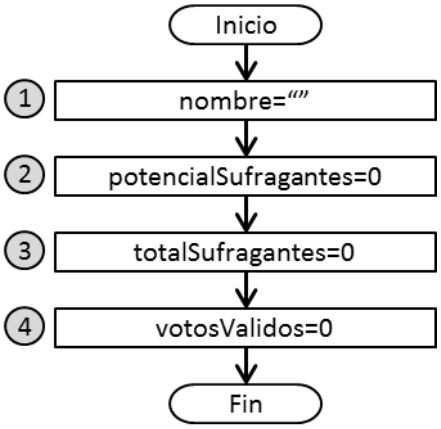


Figura 10.3 Diagrama de flujo para el método Departamento()

10.1.2.3.2 Clase Departamento - setNombre(pNombre : alfanumérico)

Descripción: Método que asigna al atributo nombre el valor dado como parámetro, previa comprobación de su validez.

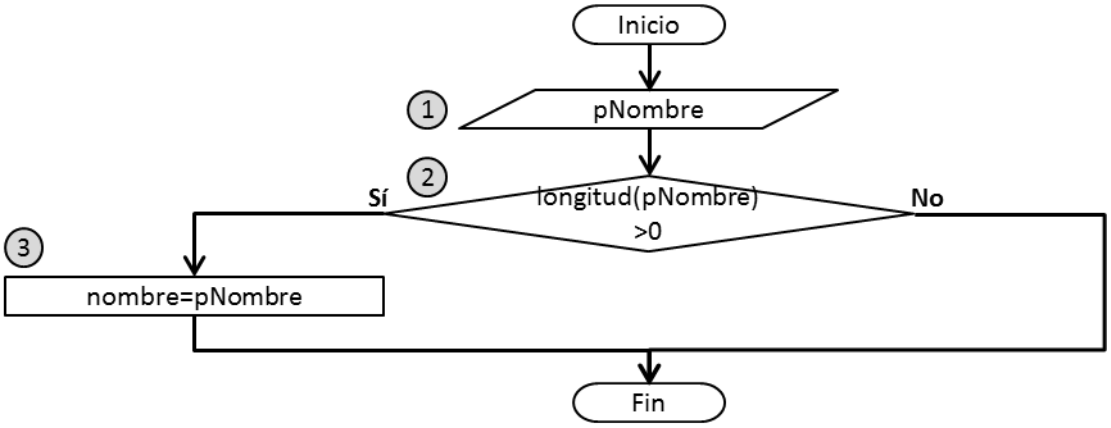


Figura 10.4 Diagrama de flujo para el método setNombre()

10.1.2.3.3 Clase Departamento - setPotencialSufragantes(pPotencialSufragantes : entero)

Descripción: Método que asigna al atributo potencialSufragantes el valor dado como parámetro, previa comprobación de su validez.

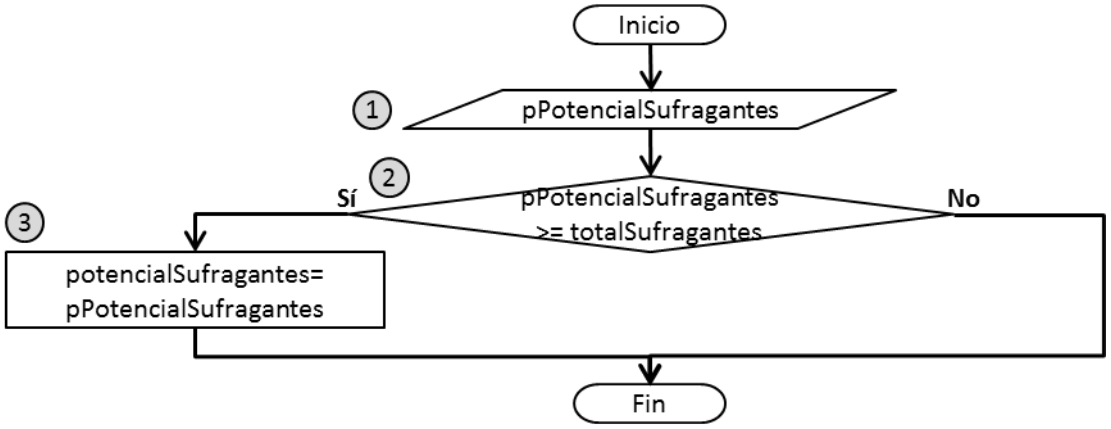


Figura 10.5 Diagrama de flujo para el método setPotencialSufragantes()

10.1.2.3.4 Clase Departamento - setTotalSufragantes(pTotalSufragantes : entero)

Descripción: Método que asigna al atributo totalSufragantes el valor dado como parámetro, previa comprobación de su validez.

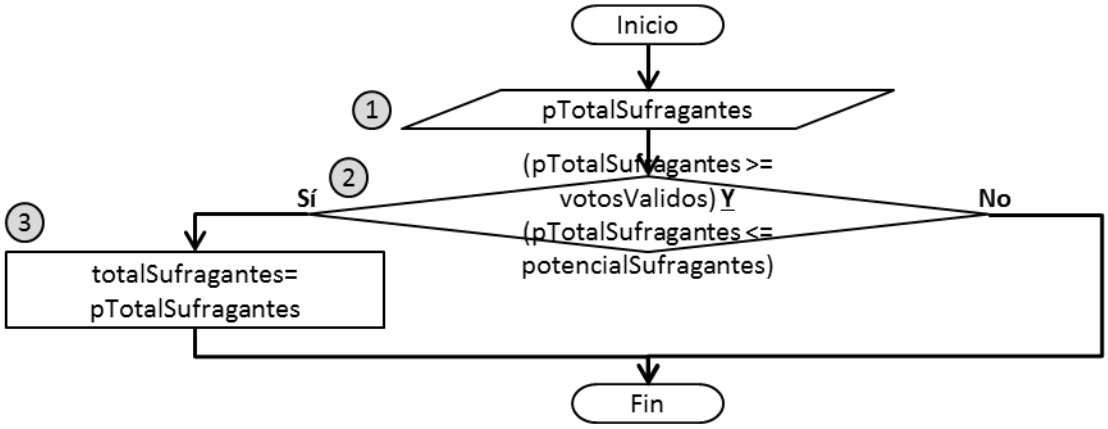


Figura 10.6 Diagrama de flujo para el método setTotalSufragantes()

10.1.2.3.5 Clase Departamento - setVotosValidos(pVotosValidos : entero)

Descripción: Método que asigna al atributo votosValidos el valor dado como parámetro, previa comprobación de su validez.

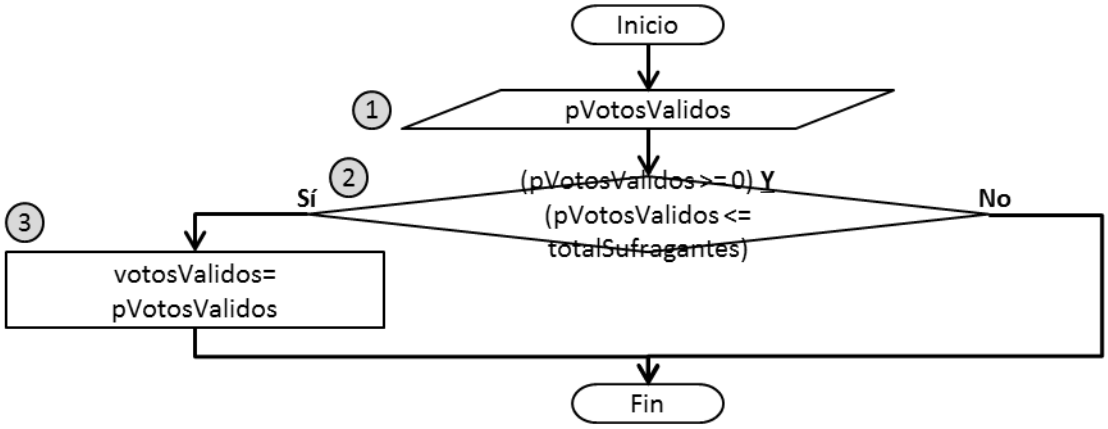


Figura 10.7 Diagrama de flujo para el método setVotosValidos()

10.1.2.3.6 Clase Departamento - getNombre( ):alfanumérico

Descripción: Entrega como respuesta el valor del atributo nombre.

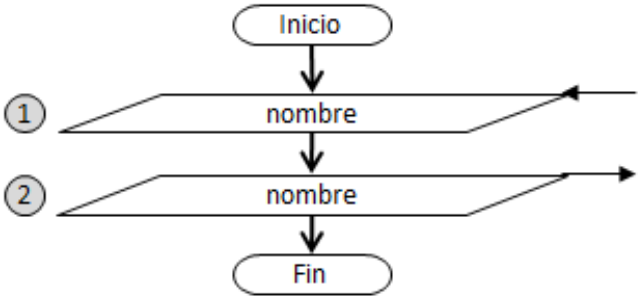


Figura 10.8 Diagrama de flujo para el método getNombre()

10.1.2.3.7 Clase Departamento - getPotencialSufragantes( ):entero

Descripción: Entrega como respuesta el valor del atributo potencialSufragantes.

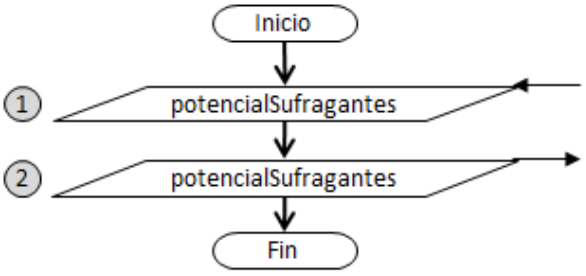


Figura 10.9 Diagrama de flujo para el método getPotencialSufragantes()

10.1.2.3.8 Clase Departamento - getTotalSufragantes( ):entero

Descripción: Entrega como respuesta el valor del atributo totalSufragantes.

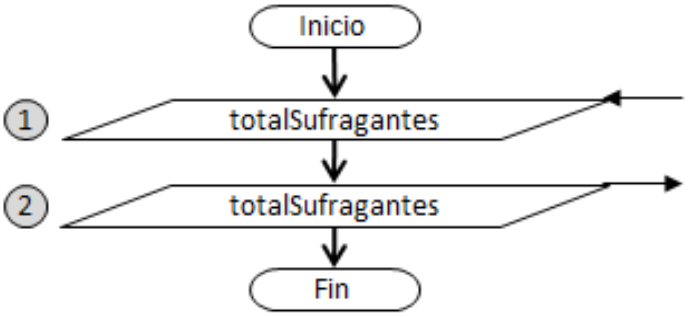


Figura 10.10 Diagrama de flujo para el método getTotalSufragantes()

10.1.2.3.9 Clase Departamento - getVotosValidos( ):entero

Descripción: Entrega como respuesta el valor del atributo votosValidos.

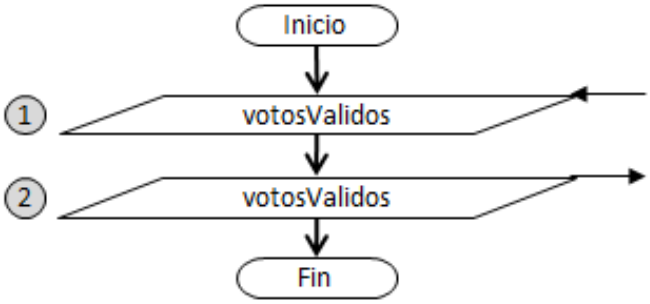


Figura 10.11 Diagrama de flujo para el método getVotosValidos()

10.1.2.3.10 Clase Departamento – calcularParticipacion( ): real

Descripción: Método de cálculo para obtener el indicador de participación electoral con base en el potencial y el total de sufragantes del departamento.

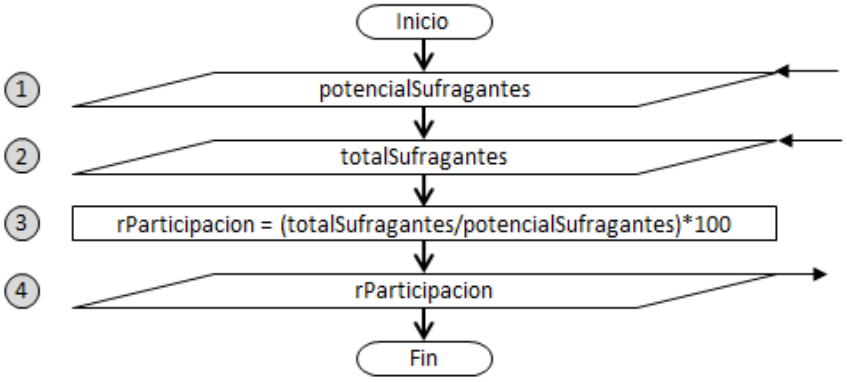


Figura 10.12 Diagrama de flujo para el método calcularParticipacion()

10.1.2.3.11 Clase Departamento – calcularCategoriaParticipacion( ): cadena

Descripción: Método de cálculo para determinar la categoría de participación electoral del departamento, con base en el indicador de participación calculado por el método anterior. Se presenta un conjunto de 5 niveles de categoría para clasificación.

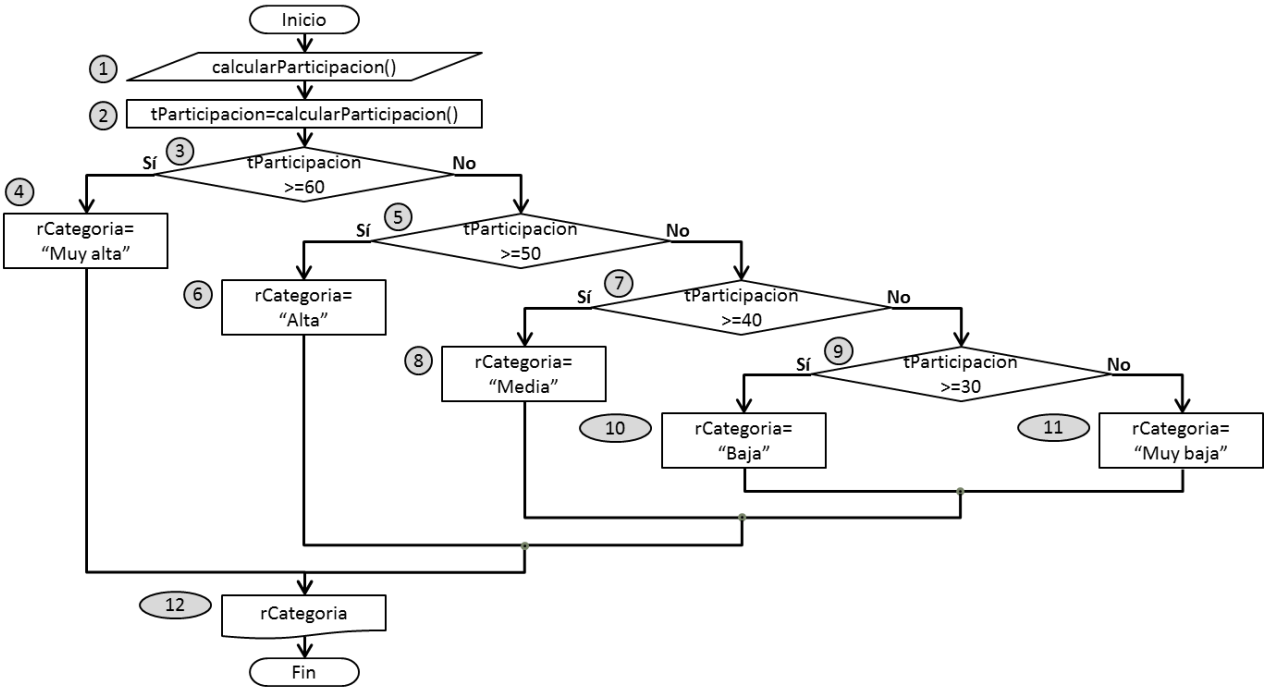


Figura 10.13 Diagrama de flujo para el método calcularCategoriaParticipacion()

Tabla 10.6 Descripción del algoritmo para el método calcularCategoriaParticipacion ()

1	Se requiere como dato de entrada el indicador de participación electoral del propio objeto. Para ello ordena al propio objeto que ejecute su método *calcularParticipacion()* y almacena en la variable tParticipacion la respuesta del método.
2	Si el valor tParticipacion es mayor o igual a 60:
3	Asigna en la variable rCategoria el valor “Muy alta”
4	De lo contrario: Si el valor tParticipacion es menor a 60 y el valor tParticipacion es mayor o igual a 50:
5	Asigna en la variable rCategoria el valor “Alta”
6	De lo contrario: Si el valor tParticipacion es menor a 50 y el valor tParticipacion es mayor o igual a 40:
7	Asigna en la variable rCategoria el valor “Media”
8	De lo contrario: Si el valor tParticipacion es menor a 40 y el valor tParticipacion es mayor o igual a 30:
9	Asigna en la variable rCategoria el valor “Baja”
10	De lo contrario: Asigna en la variable rCategoria el valor “Muy baja”
11	Se genera como dato de salida o respuesta del método el valor de la variable *rCategoria*.

10.1.2.3.12 Clase Departamento - calcularVotosNoValidos( ):entero

Descripción: Calcular la cantidad de votos no válidos con base en el total de sufragantes y los votos válidos en el departamento

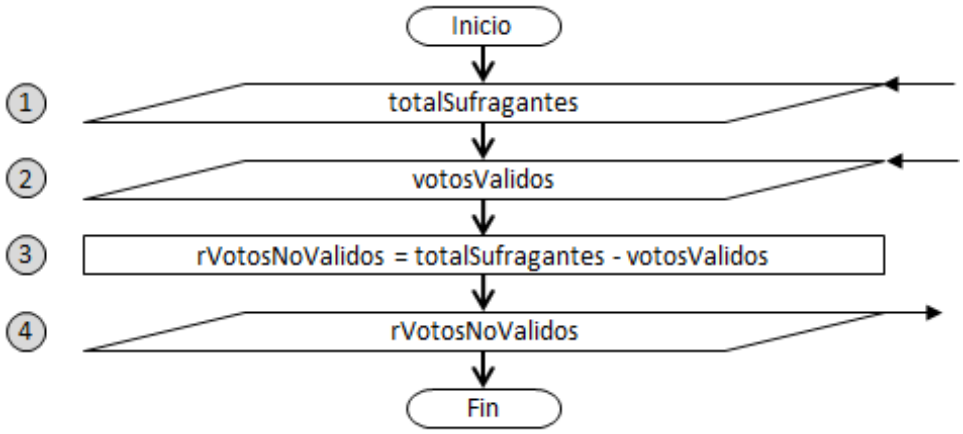


Figura 10.14 Diagrama de flujo para el método calcularVotosNoValidos()

10.1.2.3.13 Clase Departamento - calcularIndicadorError( ):real

Descripción: Calcular el indicador de participación electoral con base en el potencial y el total de sufragantes del departamento

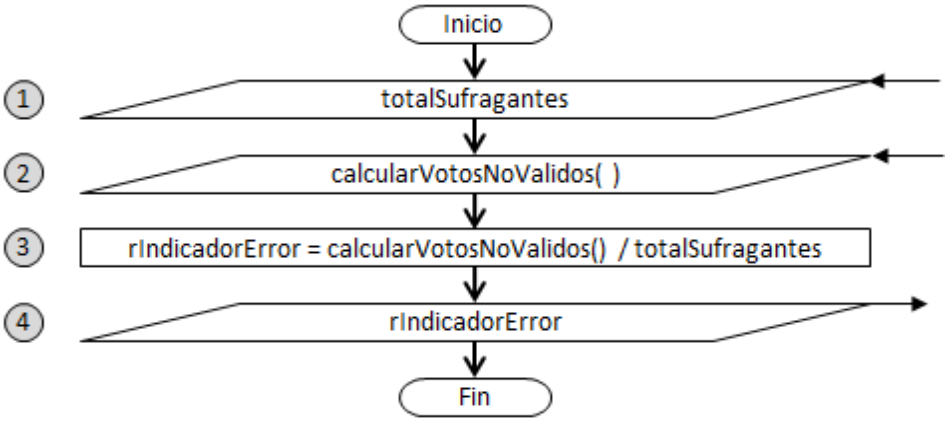


Figura 10.15 Diagrama de flujo para el método calcularIndicadorError()

10.1.2.3.14 Clase Departamento - determinarValidez( ):lógico

Descripción: Determinar la consistencia interna de los valores de los atributos, revisando que se cumplan todas las condiciones de validación.

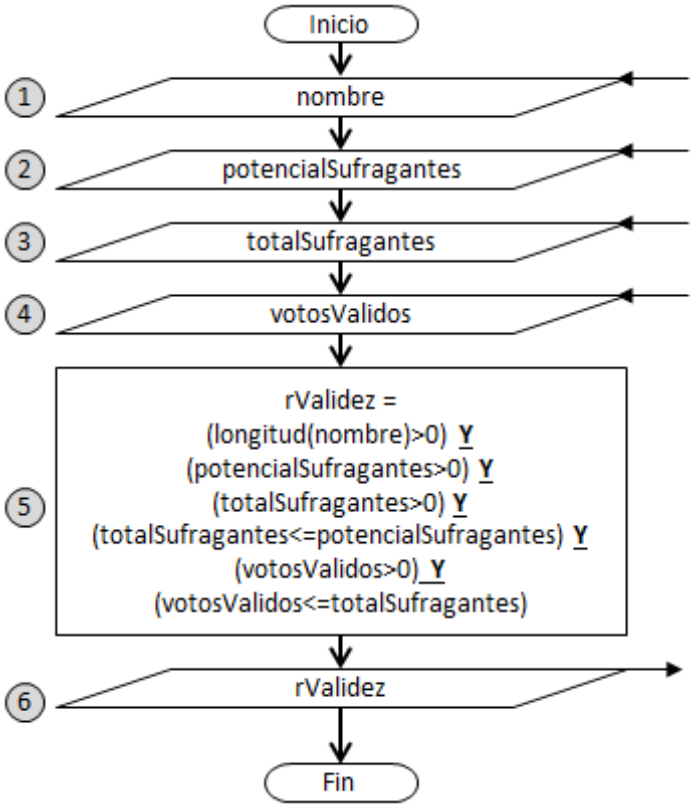


Figura 10.16 Diagrama de flujo para el método determinarValidez()

10.1.2.3.15 Clase Presentacion

La clase Presentacion mantiene su estructura y por lo tanto la misma definición tal como se describe en las secciones 9.1.2.3.7 a 9.1.2.3.9 del capítulo anterior.

10.1.2.3.16 Clase Menu

La clase Menu mantiene su estructura y por lo tanto la misma definición tal como se describe en la sección 8.4.2.3.9 del capítulo 8.

10.1.2.3.17 Clase Datos

La clase Datos mantiene su estructura y por lo tanto la misma definición tal como se describe en las secciones 9.4.2.3.4 a 9.4.2.3.8 del capítulo anterior

10.1.2.3.18 Clase Main

Descripción: Coordina las acciones de las demás clases. Solicita a un objeto de la clase Menu que reciba de parte del usuario la opción requerida por él, que almacena en una variable.

La clase Main mantiene su especificación en el nuevo prototipo, la cual está ampliamente descrita en la sección 9.4.2.3.9 del capítulo anterior.

10.1.3 Implementación

La estructura del proyecto no cambia, solo se incluye un método en la clase Departamento para la validación de los datos, que es reutilizado para su presentación y su grabación.

Tabla 10.7 Codificación de la clase *Departamento*

```
1   package proyecto;
2
3   public class Departamento {
4      private String nombre;
5      private int potencialSufragantes;
6      private int totalSufragantes;
7      private int votosValidos;
8
9      public Departamento(){
10        nombre="";
11        potencialSufragantes=0;
12        totalSufragantes=0;
13        votosValidos=0;
14     }
15
16     public void setNombre(String pNombre){
17        if (pNombre.length()>0){
18           this.nombre=pNombre;
19        }
20     }
21
22     public void setPotencialSufragantes(int pPotencialSufragantes){
23        if (pPotencialSufragantes>this.totalSufragantes){
24           this.potencialSufragantes=pPotencialSufragantes;
25        }
26     }
27
28     public void setTotalSufragantes(int pTotalSufragantes){
29        if ((pTotalSufragantes>=this.votosValidos) && (pTotalSufragantes<=this.potencialSufragantes)){
30           this.totalSufragantes=pTotalSufragantes;
31        }
32     }
33
34     public void setVotosValidos(int pVotosValidos){
35        if ((pVotosValidos>=0) && (pVotosValidos<=this.totalSufragantes)){
```

```
36          this.votosValidos=pVotosValidos;
37       }
38    }
39
40    public String getNombre(){
41       return this.nombre;
42    }
43
44    public int getPotencialSufragantes(){
45       return this.potencialSufragantes;
46    }
47
48    public int getTotalSufragantes(){
49       return this.totalSufragantes;
50    }
51
52    public int getVotosValidos(){
53       return this.votosValidos;
54    }
55
56    public double calcularParticipacion(){
57       double rParticipacion;
58       rParticipacion=((double)totalSufragantes/potencialSufragantes)*100;
59       return rParticipacion;
60    }
61
62    public String calcularCategoriaParticipacion(){
63       double tParticipacion;
64       String rCategoria;
65       tParticipacion=this.calcularParticipacion();
66       if (tParticipacion>=60){
67          rCategoria="Muy alta";
68       }
69       else{
70          if (tParticipacion>=50){
71             rCategoria="Alta";
72          }
73          else{
74             if (tParticipacion>=40){
75                rCategoria="Media";
76             }
77             else{
78                if (tParticipacion>=30){
79                   rCategoria="Baja";
80                }
81                else{
82                   rCategoria="Muy baja";
83                }
84             }
85          }
86       }
87       return rCategoria;
88    }
```

```
89
90      public int calcularVotosNoValidos(){
91          int rVotosNoValidos;
92          rVotosNoValidos=this.totalSufragantes-this.votosValidos;
93          return rVotosNoValidos;
94      }
95
96      public double calcularIndicadorError(){
97          double rIndicadorError;
98          rIndicadorError=(double)this.calcularVotosNoValidos()/this.totalSufragantes;
99          return rIndicadorError;
100     }
101
102     public boolean determinarValidez(){
103         boolean rValidez;
104         rValidez=((this.nombre.length()>0) &&
105                 (this.potencialSufragantes>0) &&
106                 (this.totalSufragantes>0) &&
107                 (this.totalSufragantes<=this.potencialSufragantes) &&
108                 (this.votosValidos>0) &&
109                 (this.votosValidos<=this.totalSufragantes));
110         return rValidez;
111     }
112 }
```

Tabla 10.8 Codificación de la clase *Presentacion*

```
1   package proyecto;
2   import javax.swing.JOptionPane;
3
4   public class Presentacion {
5       public Presentacion(){
6
7       }
8
9       public Departamento capturarDepartamento(){
10          Departamento rDepartamento;
11          rDepartamento=new Departamento();
12          rDepartamento.setNombre(JOptionPane.showInputDialog("Nombre: "));
13          try{
14              rDepartamento.setPotencialSufragantes(Integer.parseInt(JOptionPane.showInputDialog("Potencial
    de sufragantes: ")));
15              rDepartamento.setTotalSufragantes(Integer.parseInt(JOptionPane.showInputDialog("Total de
    sufragantes: ")));
16              rDepartamento.setVotosValidos(Integer.parseInt(JOptionPane.showInputDialog("Votos válidos: ")));
17          }
18          catch (Exception e){
19
20          }
21          return rDepartamento;
22      }
```

```
23
24    public String capturarNombreDepartamento(){
25       String rNombre;
26       rNombre=JOptionPane.showInputDialog("Departamento: ");
27       return rNombre;
28    }
29
30    public void presentarDepartamento(Departamento pDepartamento){
31       String rMensaje;
32       if (pDepartamento.determinarValidez()){
33          rMensaje="Departamento: "+pDepartamento.getNombre();
34          rMensaje=rMensaje+"\nPotencial de sufragantes: "+pDepartamento.getPotencialSufragantes();
35          rMensaje=rMensaje+"\nTotal de sufragantes: "+pDepartamento.getTotalSufragantes();
36          rMensaje=rMensaje+"\nVotos válidos: "+pDepartamento.getVotosValidos();
37          rMensaje=rMensaje+"\nIndicador de participación: "+pDepartamento.calcularParticipacion()+" %";
38          rMensaje=rMensaje+"\nCategoría según participación:
    "+pDepartamento.calcularCategoriaParticipacion();
39          rMensaje=rMensaje+"\nVotos no válidos: "+pDepartamento.calcularVotosNoValidos();
40          rMensaje=rMensaje+"\nIndicador de error: "+pDepartamento.calcularIndicadorError();
41       }
42       else{
43          rMensaje="Datos inconsistentes. No se pueden presentar";
44       }
45       JOptionPane.showMessageDialog(null, rMensaje);
46    }
47 }
```

Tabla 10.9 Codificación de la clase *Menu*

```
1  package proyecto;
2  import javax.swing.JOptionPane;
3
4  public class Menu {
5     public Menu(){
6
7     }
8
9     public int capturarOpcion(){
10       String tMensaje;
11       int rOpcion;
12       tMensaje="1. Registrar departamento\n"+
13             "2. Consultar departamento\n"+
14             "3. Salir\n";
15       rOpcion=Integer.parseInt(JOptionPane.showInputDialog(tMensaje));
16       return rOpcion;
17    }
18 }
```

Tabla 10.10 Codificación de la clase *Datos*

```
1  package proyecto;
2  import java.io.*;
3
4  public class Datos {
5     public Datos(){
6
7     }
8
9     public String construirRegistroDepartamento(Departamento pDepartamento){
10       String rRegistro;
11       rRegistro=pDepartamento.getNombre()+",";
12       rRegistro=rRegistro+pDepartamento.getPotencialSufragantes()+",";
13       rRegistro=rRegistro+pDepartamento.getTotalSufragantes()+",";
14       rRegistro=rRegistro+pDepartamento.getVotosValidos();
15       return rRegistro;
16    }
17
18    public String descomponerRegistro(String pRegistro, int pOrden){
19       int i,j;
20       String rDato;
21       i=0;
22       j=1;
23       rDato="";
24       while ((i<pRegistro.length()) && (j<=pOrden)) {
25          if (pRegistro.substring(i,i+1).compareTo(",")!=0){
26             if (j==pOrden){
27                rDato=rDato+pRegistro.substring(i,i+1);
28             }
29          }
30          else{
31             j=j+1;
32          }
33          i=i+1;
34       }
35       return rDato;
36    }
37
38    public String recuperarRegistroDepartamento(String pNombre){
39       FileReader fileReader;
40       BufferedReader bufferedReader;
41       String rRegistro,tNombre;
42       try{
43          fileReader=new FileReader("departamentos.txt");
44          bufferedReader=new BufferedReader(fileReader);
45          do{
46             rRegistro=bufferedReader.readLine();
47             tNombre=this.descomponerRegistro(rRegistro, 1);
48          } while (tNombre.compareToIgnoreCase(pNombre)!=0);
49          fileReader.close();
50       }
51       catch (Exception e){
```

```
52          rRegistro="";
53       }
54       return rRegistro;
55    }
56
57    public Departamento recuperarDepartamento(String pNombre){
58       String tRegistro;
59       Departamento rDepartamento;
60       rDepartamento=new Departamento();
61       tRegistro=this.recuperarRegistroDepartamento(pNombre);
62       try{
63          rDepartamento.setNombre(this.descomponerRegistro(tRegistro,1));
64          rDepartamento.setPotencialSufragantes(Integer.parseInt(this.descomponerRegistro(tRegistro,2)));
65          rDepartamento.setTotalSufragantes(Integer.parseInt(this.descomponerRegistro(tRegistro,3)));
66          rDepartamento.setVotosValidos(Integer.parseInt(this.descomponerRegistro(tRegistro,4)));
67       }
68       catch (Exception e){
69
70       }
71       return rDepartamento;
72    }
73
74    public void grabarDepartamento(Departamento pDepartamento){
75       FileWriter fileWriter;
76       PrintWriter printWriter;
77       if ((pDepartamento.determinarValidez()) &&
78          (!this.recuperarDepartamento(pDepartamento.getNombre()).determinarValidez())){
79          try{
80             fileWriter=new FileWriter("departamentos.txt",true);
81             printWriter=new PrintWriter(fileWriter);
82             printWriter.println(this.construirRegistroDepartamento(pDepartamento));
83             fileWriter.close();
84          }
85          catch(Exception e){
86
87          }
88       }
89    }
90 }
```

Tabla 10.11 Codificación de la clase *Main*

```
1  package proyecto;
2
3  public class Main {
4
5     public static void main(String[] args) {
6        Departamento departamento;
7        Presentacion presentacion;
8        Menu menu;
9        Datos datos;
```

```
10        int opcion;
11        String parametro;
12        presentacion=new Presentacion();
13        menu=new Menu();
14        datos=new Datos();
15        opcion=0;
16        do {
17           opcion=menu.capturarOpcion();
18           switch (opcion){
19              case 1:{
20                 departamento=presentacion.capturarDepartamento();
21                 datos.grabarDepartamento(departamento);
22                 presentacion.presentarDepartamento(departamento);
23                 break;
24              }
25              case 2:{
26                 parametro=presentacion.capturarNombreDepartamento();
27                 departamento=datos.recuperarDepartamento(parametro);
28                 presentacion.presentarDepartamento(departamento);
29                 break;
30              }
31           }
32        } while (opcion<=2);
33     }
34  }
```

10.2 Optimización de manejo de categorías de clasificación– Prototipo 13

10.2.1 Análisis

10.2.1.1 Alcance

En el diseño del prototipo anterior, al aparecer más categorías de clasificación, el algoritmo incrementa su complejidad de forma significativa. Para el prototipo 13 se busca un algoritmo más eficiente de manejo de las categorías, de modo que una mayor cantidad de estas no impliquen modificaciones significativas de la codificación.

10.2.1.2 Definiciones

Se mantienen las mismas definiciones, y las mismas condiciones de validación del prototipo anterior.

Tabla 10.12 Definiciones aplicables en desarrollo del prototipo 13.

	Definición	Tipo de dato	Condiciones de validación
Departamento	Cada una de las circunscripciones territoriales de primer nivel en que se divide el país para propósitos electorales	Objeto	
Nombre	Identificación nominal del departamento	Cadena de caracteres	Valor ingresado. El nombre del departamento no puede quedar

			vacío y no puede haber nombres de departamentos repetidos.
Potencial de sufragantes	Cantidad de personas habilitadas en un departamento para votar en un proceso electoral	Número entero	Valor ingresado. El potencial de sufragantes debe ser mayor que cero.
Total de sufragantes	Cantidad de personas que acudieron a votar en un departamento en un proceso electoral	Número entero	Valor ingresado. El total de sufragantes debe ser mayor que cero, pero menor o igual al potencial de sufragantes.
Indicador de participación electoral	Indicador consistente en el total de sufragantes expresado como porcentaje del potencial de sufragantes	Número real	Valor calculado. $Indicador\ de\ participación\ electoral = \frac{Total\ de\ sufragantes}{Potencial\ de\ sufragantes} * 100$
Categoría de participación electoral	Categoría a la que pertenece el departamento, dependiendo del porcentaje de participación electoral.	Cadena de caracteres	Valor calculado. Si el porcentaje es mayor o igual a 60%: muy alto Si el porcentaje es menor a 60% pero mayor o igual a 50%: alto Si el porcentaje es menor a 50% pero mayor o igual a 40%: medio Si el porcentaje es menor a 40% pero mayor o igual a 30%: bajo Si el porcentaje es menor a 30%: muy bajo
Votos válidos	Cantidad de votos que indican de forma clara la voluntad del sufragante	Número entero	Valor ingresado. La cantidad de votos válidos debe ser mayor que cero, pero menos o igual al total de sufragantes.
Votos no válidos	Cantidad de votos que NO indican de forma clara la voluntad del sufragante.	Número entero	Valor calculado. $Votos\ no\ válidos = Total\ de\ sufragantes - Votos\ válidos$
Indicador de error en la votación	Indicador consistente en dividir el total de votos no válidos entre el total de sufragantes.	Número real	Valor calculado. $Indicador\ de\ error\ en\ la\ votación = \frac{Votos\ no\ válidos}{Total\ de\ sufragantes}$

10.2.1.3 Requisitos

No hay modificación en los requisitos de interfaz de usuario ni en los funcionales.

10.2.1.3.1 Requisitos de interfaz de usuario

10.2.1.3.1.1 Ingreso de información en casillas de edición

Para la interacción con el usuario, la aplicación debe presentar ventanas con casillas de edición en donde pueda digitar los datos solicitados. Al solicitar los datos, el sistema deberá indicar en la misma ventana de diálogo cuál es el dato requerido.

10.2.1.3.1.2 Presentación de menú de selección de funciones

La aplicación debe presentar un menú para seleccionar una de 3 funciones: registrar la información de un departamento para su almacenamiento, consultar información de in departamento a partir de la recuperación de sus datos almacenados, o finalizar la aplicación.

10.2.1.3.2 Requisitos funcionales

10.2.1.3.2.1 Calcular el indicador de participación electoral de un departamento

La aplicación debe permitir al usuario introducir los siguientes datos básicos de un departamento: el nombre, el potencial de sufragantes, el total de sufragantes y la cantidad de votos válidos, y con estos datos debe calcular el indicador de participación electoral, los votos no válidos y el indicador de error de votación. Los datos ingresados deben cumplir las condiciones de validación requeridas.

10.2.1.3.2.2 Almacenar la información de un departamento en un archivo plano

La aplicación debe poder almacenar en un archivo plano los datos básicos ingresados de un departamento: el nombre, el potencial de sufragantes, el total de sufragantes y el total de votos válidos. Se requiere que el formato de almacenamiento sea en datos separados por punto y coma (formato csv).

10.2.1.3.2.3 Recuperar la información de un departamento en un archivo plano

La aplicación debe poder recuperar desde un archivo plano los datos básicos ingresados de un departamento: el nombre, el potencial de sufragantes, el total de sufragantes, el total de votos válidos además de los datos calculados del indicador de participación electoral, la categoría de participación a la que pertenece, los votos no válidos y el indicador de error de votación y se debe presentar al usuario. Para ello deberá preguntar por el nombre del departamento a buscar. Como restricción, se requiere el formato de almacenamiento sea en datos separados por punto y coma (formato csv).

10.2.2 Diseño

10.2.2.1 Diseño de escenarios

Se mantiene el mismo diseño de escenarios, debido a que no se afecta el comportamiento al optimizar el algoritmo de clasificación.

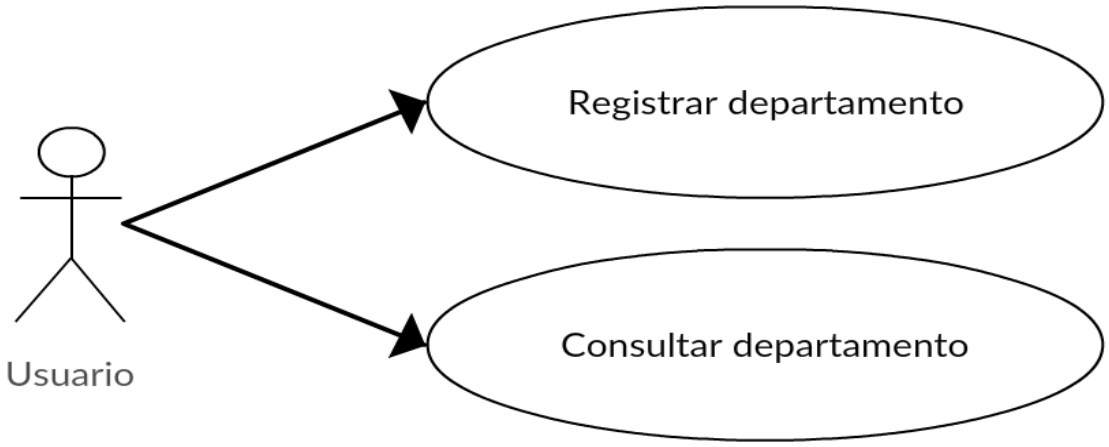


Figura 10.17 Diagrama de casos de uso para el Prototipo 13

Tabla 10.13 Especificación del caso de uso *Registrar departamento*.

1	El usuario ingresa a la aplicación.
2	La aplicación presenta el menú de selección, por lo que el usuario selecciona la opción 1: Registrar Departamento
2.1	Si el usuario selecciona una opción mayor que 2, la aplicación finaliza.
3	La aplicación presenta una pantalla con una caja de texto para que el usuario digite el nombre del departamento
4	El usuario digita el nombre del departamento y pulsa el botón *Aceptar*
5	La aplicación presenta una pantalla con una caja de texto para que el usuario digite el potencial de sufragantes del departamento
6	El usuario digita el potencial de sufragantes del departamento y pulsa el botón *Aceptar*
7	La aplicación presenta una pantalla con una caja de texto para que el usuario digite el total de sufragantes del departamento
8	El usuario digita el total de sufragantes del departamento y pulsa el botón *Aceptar*
9	La aplicación presenta una pantalla con una caja de texto para que el usuario digite la cantidad de votos válidos obtenidos en el departamento
10	El usuario digita la cantidad de votos válidos del departamento y pulsa el botón *Aceptar*
11	Si los datos son válidos: la aplicación muestra en pantalla el nombre del departamento, el potencial de sufragantes, el total de sufragantes, la cantidad de votos válidos, el indicador de participación electoral del departamento, la categoría de participación, la cantidad de votos no válidos y el indicador de error de votación en el departamento que acabó de ingresar.
12	Si los datos no son válidos: la aplicación muestra en pantalla el mensaje “Datos inconsistentes. No se pueden presentar”
13	La aplicación vuelve a presentar el menú de selección (paso 2)

Tabla 10.14 Especificación del caso de uso *Consultar departamento*.

1	El usuario ingresa a la aplicación.
2	La aplicación presenta el menú de selección, por lo que el usuario selecciona la opción 2: Consultar Departamento
2.1	Si el usuario selecciona una opción mayor que 2, la aplicación finaliza.
3	El usuario digita el nombre del departamento a buscar y pulsa el botón *Aceptar*

4	La aplicación muestra en pantalla el nombre del departamento buscado, el potencial de sufragantes, el total de sufragantes, la cantidad de votos válidos, el indicador de participación electoral del departamento, la categoría de participación, la cantidad de votos no válidos y el indicador de error de votación en el departamento.
5	La aplicación vuelve a presentar el menú de selección (paso 2)

10.2.2.2 Diseño de estructura

La estructura no se modifica. Solo se modificará el método que permite clasificar y determinar la categoría de participación, volviéndolo más complejo.

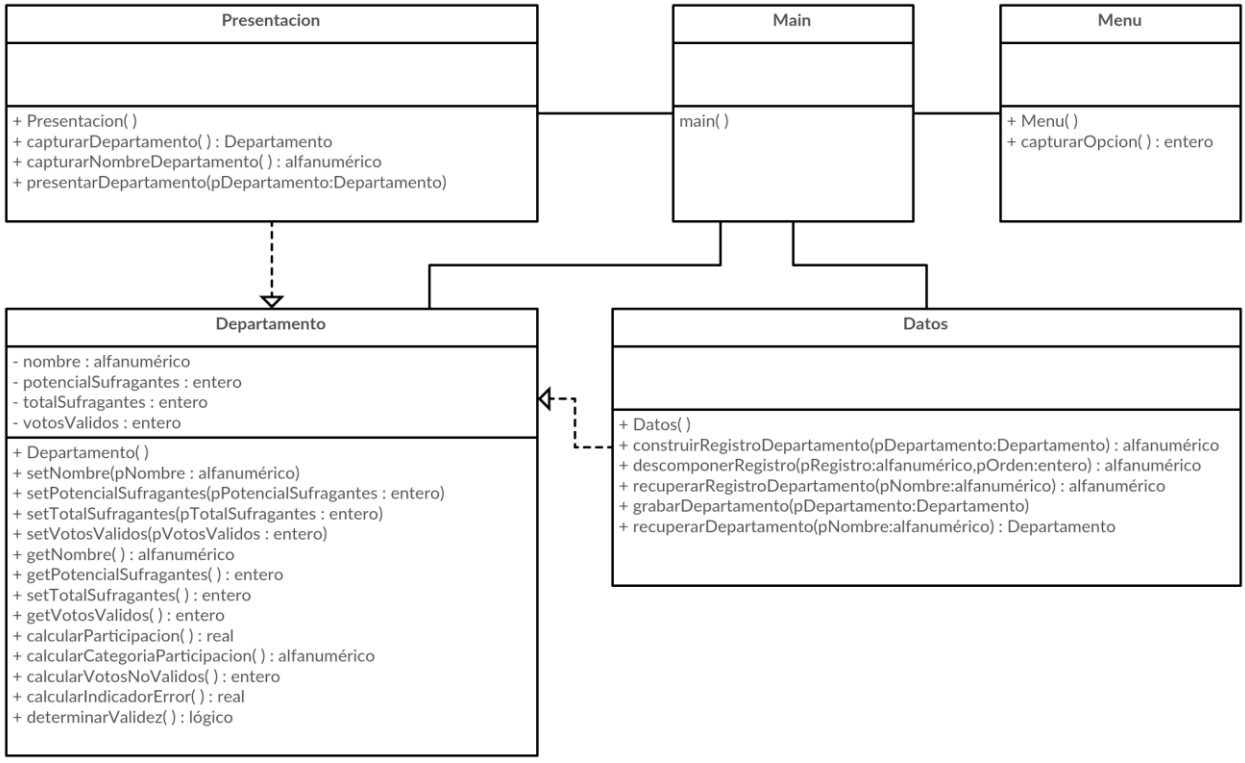


Figura 10.18 Diagrama de clases para el Prototipo 13

10.2.2.3 Diseño de algoritmos

10.2.2.3.1 Clase Departamento - Departamento( )

Descripción: Constructor que inicializa los atributos con valores neutros.

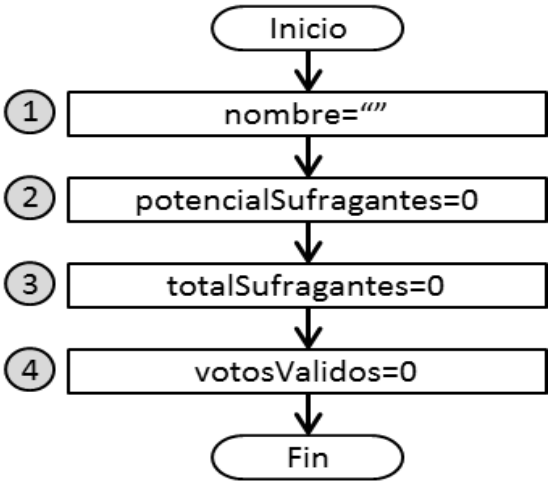


Figura 10.19 Diagrama de flujo para el método Departamento()

10.2.2.3.2 Clase Departamento - setNombre(pNombre : alfanumérico)

Descripción: Método que asigna al atributo nombre el valor dado como parámetro, previa comprobación de su validez.

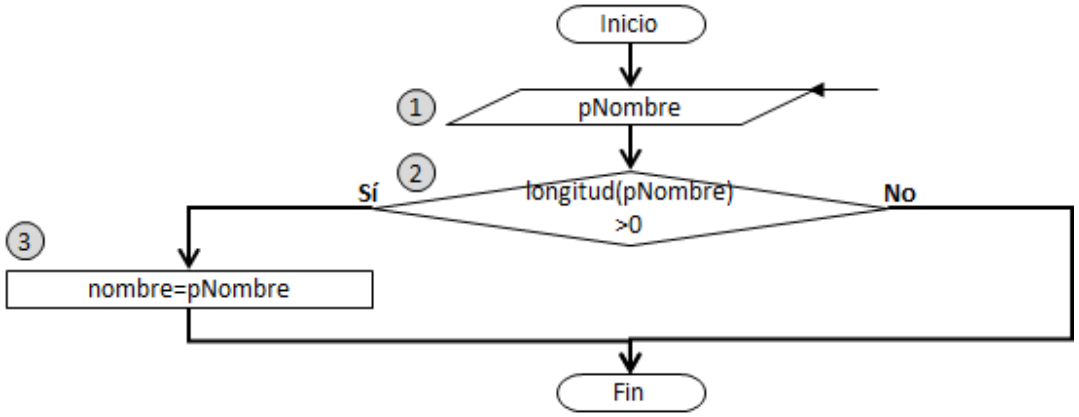


Figura 10.20 Diagrama de flujo para el método setNombre()

10.2.2.3.3 Clase Departamento - setPotencialSufragantes(pPotencialSufragantes : entero)

Descripción: Método que asigna al atributo potencialSufragantes el valor dado como parámetro, previa comprobación de su validez.

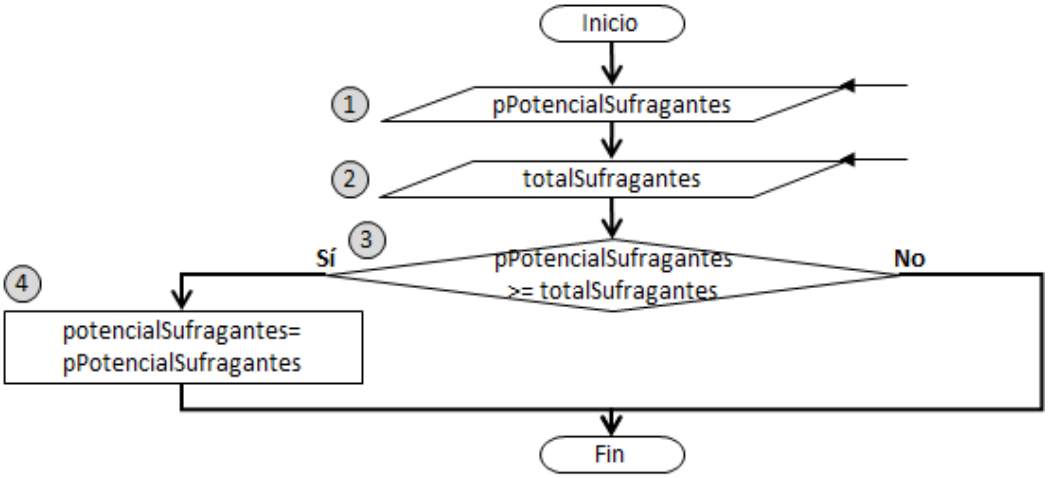


Figura 10.21 Diagrama de flujo para el método setPotencialSufragantes()

10.2.2.3.4 Clase Departamento - setTotalSufragantes(pTotalSufragantes : entero)

Descripción: Método que asigna al atributo totalSufragantes el valor dado como parámetro, previa comprobación de su validez.

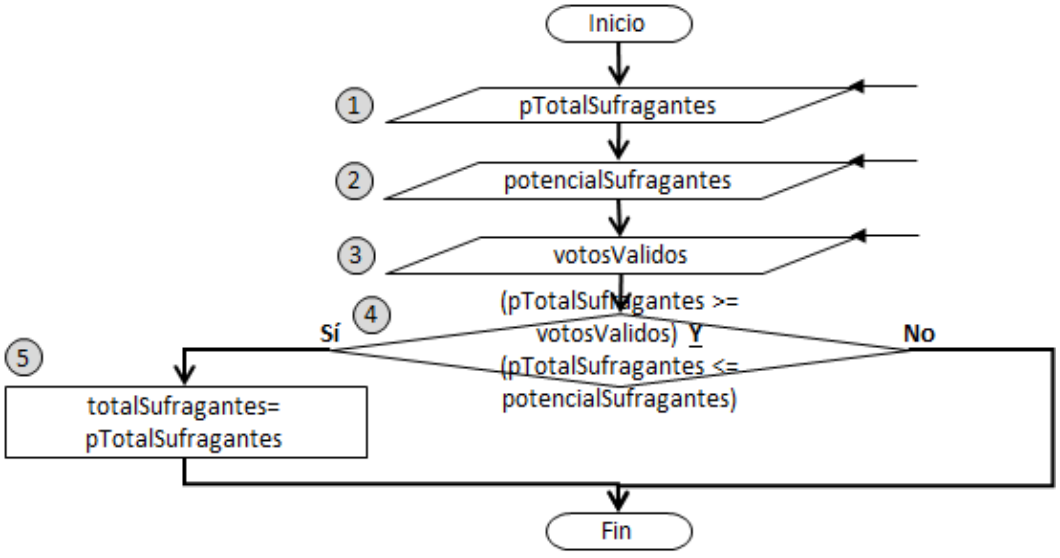


Figura 10.22 Diagrama de flujo para el método setTotalSufragantes()

10.2.2.3.5 Clase Departamento - setVotosValidos(pVotosValidos : entero)

Descripción: Método que asigna al atributo votosValidos el valor dado como parámetro, previa comprobación de su validez.

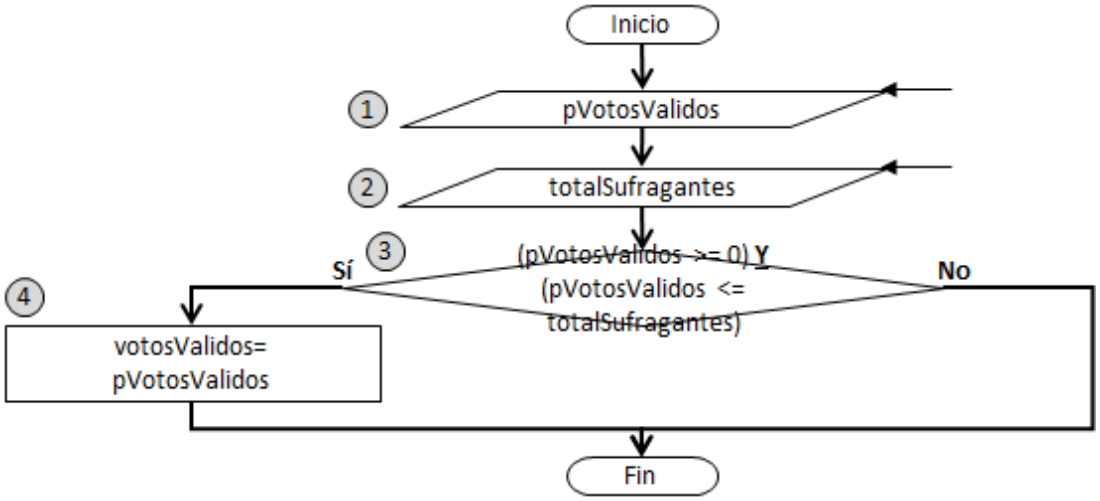


Figura 10.23 Diagrama de flujo para el método setVotosValidos()

10.2.2.3.6 Clase Departamento - getNombre( ):alfanumérico

Descripción: Entrega como respuesta el valor del atributo nombre.

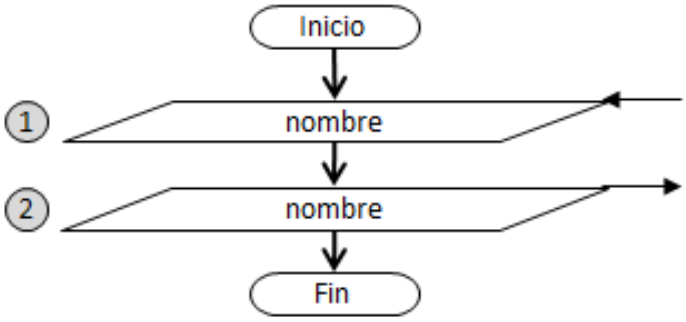


Figura 10.24 Diagrama de flujo para el método getNombre()

10.2.2.3.7 Clase Departamento - getPotencialSufragantes( ):entero

Descripción: Entrega como respuesta el valor del atributo potencialSufragantes.

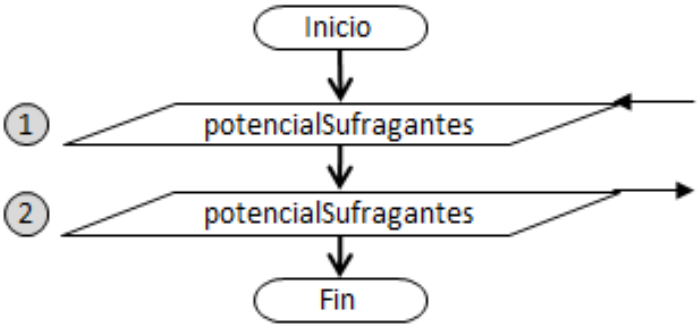


Figura 10.25 Diagrama de flujo para el método getPotencialSufragantes()

10.2.2.3.8 Clase Departamento - getTotalSufragantes( ):entero

Descripción: Entrega como respuesta el valor del atributo totalSufragantes.

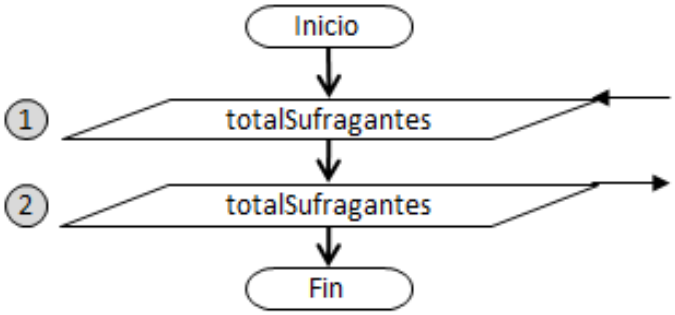


Figura 10.26 Diagrama de flujo para el método getTotalSufragantes()

10.2.2.3.9 Clase Departamento - getVotosValidos( ):entero

Descripción: Entrega como respuesta el valor del atributo votosValidos.

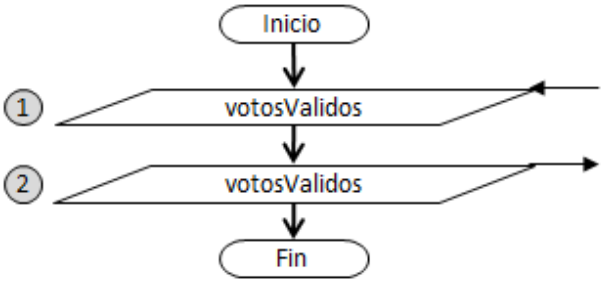


Figura 10.27 Diagrama de flujo para el método getVotosValidos()

10.2.2.3.10 Clase Departamento – calcularParticipacion( ): real

Descripción: Método de cálculo para obtener el indicador de participación electoral con base en el potencial y el total de sufragantes del departamento.

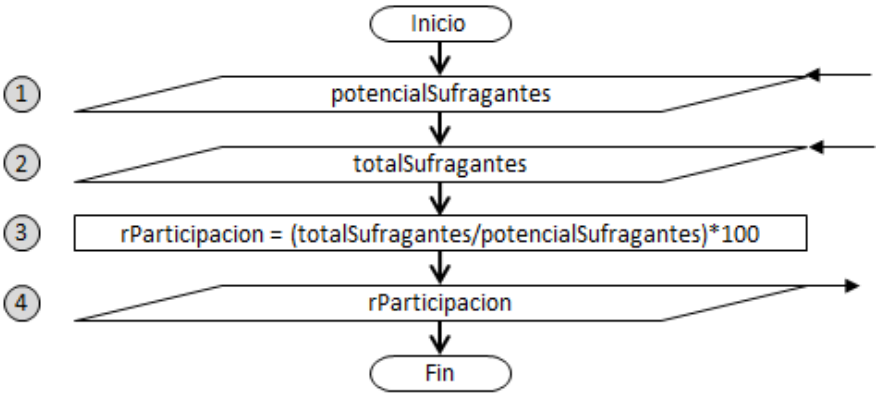


Figura 10.28 Diagrama de flujo para el método calcularParticipacion()

10.2.2.3.11 Clase Departamento – calcularCategoriaParticipacion( ): cadena

Descripción: Método de cálculo para determinar la categoría de participación electoral del departamento, con base en el indicador de participación calculado por el método anterior. Se presenta un conjunto de 5 niveles de categoría para clasificación, pero cambia completamente el algoritmo de clasificación usando un arreglo.

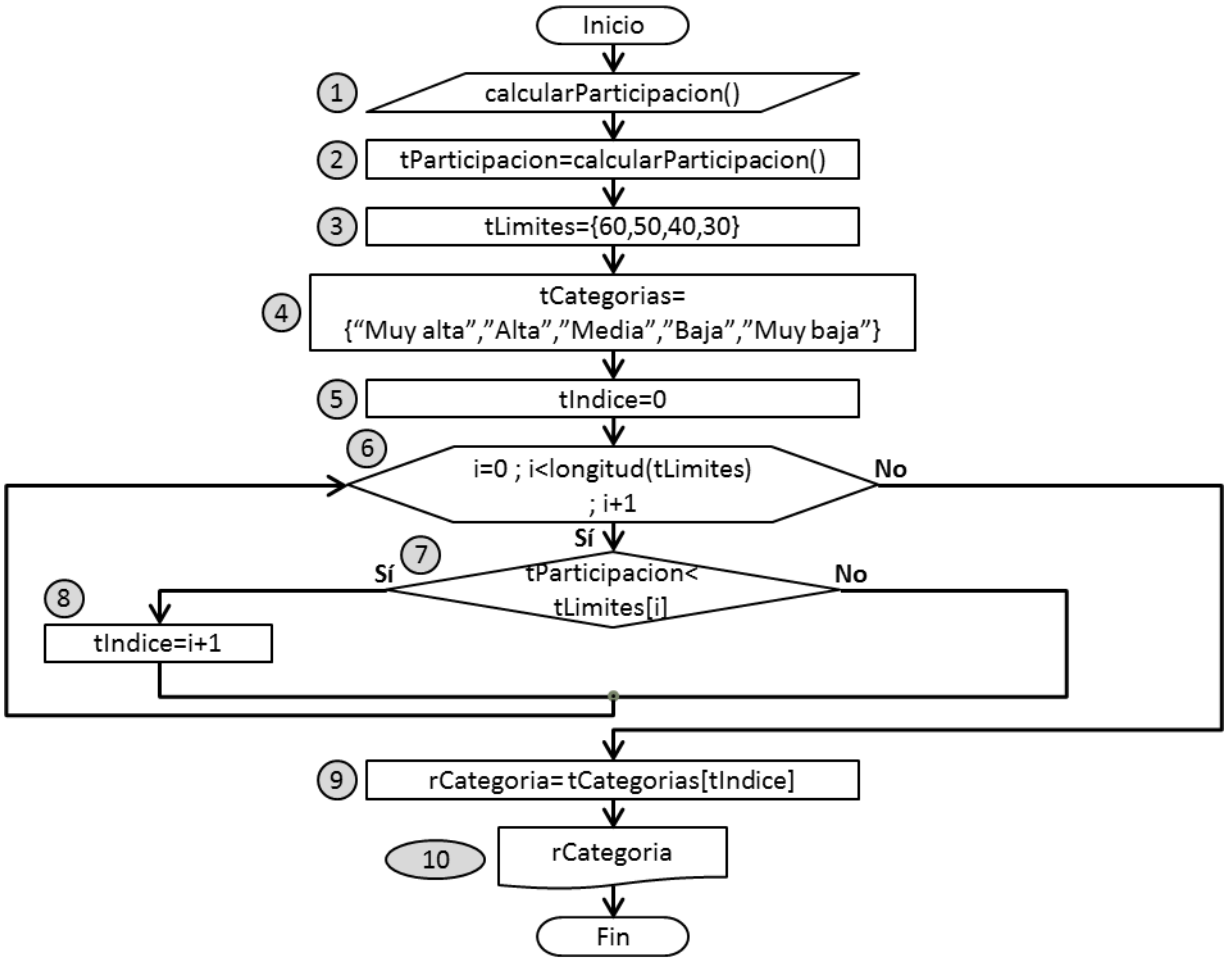


Figura 10.29 Diagrama de flujo para el método calcularCategoriaParticipacion()

Tabla 10.15 Descripción del algoritmo para el método calcularCategoriaParticipacion ()

1-2	Se requiere como dato de entrada el indicador de participación electoral del propio objeto. Para ello ordena al propio objeto que ejecute su método *calcularParticipacion()* y almacena en la variable tParticipacion la respuesta del método.
3	Se crea un arreglo conteniendo los valores de los límites inferiores de las clasificaciones, y se almacena en la variable tLimites
4	Se crea un arreglo conteniendo los nombres de las clasificaciones, y se almacena en la variable tCategorias
5	Asigna en la variable tIndice el valor 0
6	Inicia un ciclo definido con control i desde 0 hasta la cantidad de datos del arreglo tLimites
7	Revisa que el valor de la variable tParticipaciones sea menor que el dato del arreglo tLimites en su posición i
8	Si es así:

	Asigna en la variable tIndice el valor de i + 1
9	Finaliza el ciclo definido, y asigna el valor del dato del arreglo tCategorias en su posición i en la variable *rCategoria*
10	Se genera como dato de salida o respuesta del método el valor de la variable *rCategoria*.

10.2.2.3.12 Clase Departamento - calcularVotosNoValidos( ):entero

Descripción: Calcular la cantidad de votos no válidos con base en el total de sufragantes y los votos válidos en el departamento

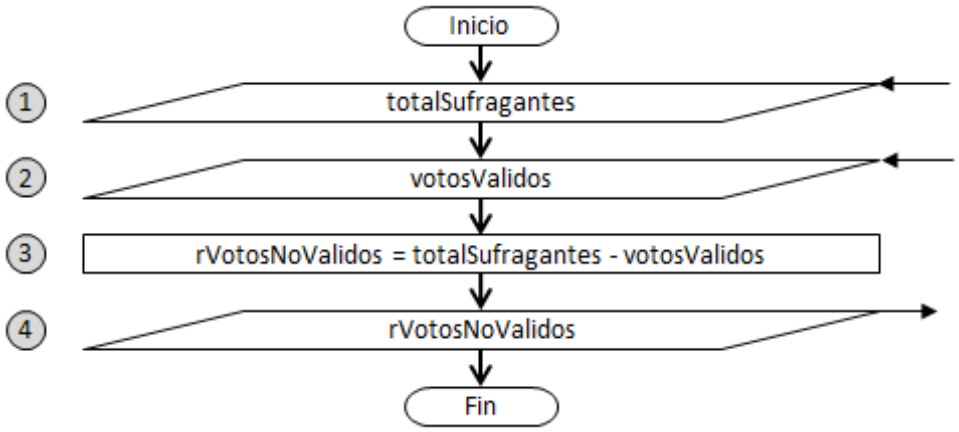


Figura 10.30 Diagrama de flujo para el método calcularVotosNoValidos()

10.2.2.3.13 Clase Departamento - calcularIndicadorError( ):real

Descripción: Calcular el indicador de participación electoral con base en el potencial y el total de sugfagantes del departamento

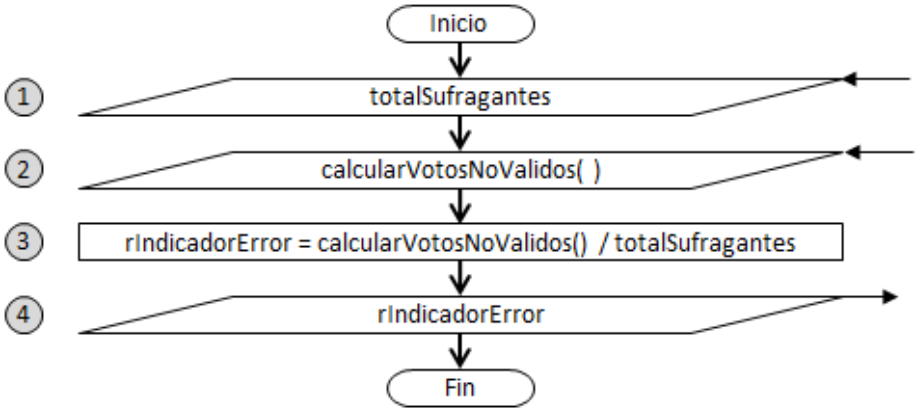


Figura 10.31 Diagrama de flujo para el método calcularIndicadorError()

10.2.2.3.14 Clase Departamento - determinarValidez( ):lógico

Descripción: Determinar la consistencia interna de los valores de los atributos, revisando que se cumplan todas las condiciones de validación.

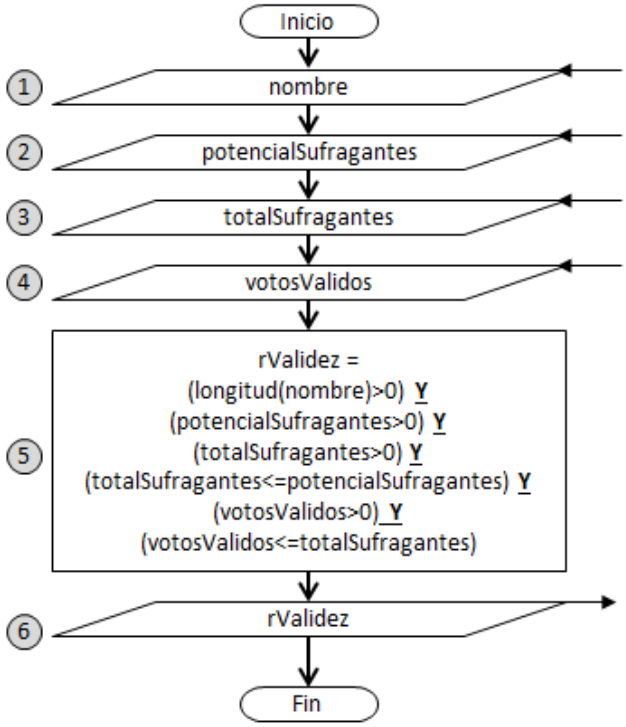


Figura 10.32 Diagrama de flujo para el método determinarValidez()

10.2.2.3.15 Clase Presentacion

La clase Presentacion mantiene su estructura y por lo tanto la misma definición tal como se describe en las secciones 9.1.2.3.7 a 9.1.2.3.9 del capítulo anterior.

10.2.2.3.16 Clase Menu

La clase Menu mantiene su estructura y por lo tanto la misma definición tal como se describe en la sección 8.4.2.3.9 del capítulo 8.

10.2.2.3.17 Clase Datos

La clase Datos mantiene su estructura y por lo tanto la misma definición tal como se describe en las secciones 9.4.2.3.4 a 9.4.2.3.8 del capítulo anterior

10.2.2.3.18 Clase Main

La clase Main mantiene su especificación en el nuevo prototipo, la cual está ampliamente descrita en la sección 9.4.2.3.9 del capítulo anterior.

10.2.3 Implementación

La estructura del proyecto no cambia, solo se incluye un método en la clase Departamento para la validación de los datos, que es reutilizado para su presentación y su grabación.

Tabla 10.16 Codificación de la clase *Departamento*

```
1   package proyecto;
2
3   public class Departamento {
4      private String nombre;
5      private int potencialSufragantes;
6      private int totalSufragantes;
7      private int votosValidos;
8
9      public Departamento(){
10        nombre="";
11        potencialSufragantes=0;
12        totalSufragantes=0;
13        votosValidos=0;
14     }
15
16     public void setNombre(String pNombre){
17        if (pNombre.length()>0){
18           this.nombre=pNombre;
19        }
20     }
21
22     public void setPotencialSufragantes(int pPotencialSufragantes){
23        if (pPotencialSufragantes>this.totalSufragantes){
24           this.potencialSufragantes=pPotencialSufragantes;
25        }
26     }
27
28     public void setTotalSufragantes(int pTotalSufragantes){
29        if ((pTotalSufragantes>=this.votosValidos) && (pTotalSufragantes<=this.potencialSufragantes)){
30           this.totalSufragantes=pTotalSufragantes;
31        }
32     }
33
34     public void setVotosValidos(int pVotosValidos){
35        if ((pVotosValidos>=0) && (pVotosValidos<=this.totalSufragantes)){
36           this.votosValidos=pVotosValidos;
37        }
38     }
39
40     public String getNombre(){
```

```
41      return this.nombre;
42    }
43
44    public int getPotencialSufragantes(){
45      return this.potencialSufragantes;
46    }
47
48    public int getTotalSufragantes(){
49      return this.totalSufragantes;
50    }
51
52    public int getVotosValidos(){
53      return this.votosValidos;
54    }
55
56    public double calcularParticipacion(){
57      double rParticipacion;
58      rParticipacion=((double)totalSufragantes/potencialSufragantes)*100;
59      return rParticipacion;
60    }
61
62    public String calcularCategoriaParticipacion(){
63      double tParticipacion;
64      String rCategoria;
65      int i;
66      int tIndice;
67      tParticipacion=this.calcularParticipacion();
68      int[] tLimites={60,50,40,30};
69      String[] tCategorias={"Muy alto","Alto","Medio","Bajo","Muy bajo"};
70      tIndice=0;
71      for (i=0;i<tLimites.length;i=i+1){
72        if (tParticipacion<tLimites[i]){
73          tIndice=i+1;
74        }
75      }
76      rCategoria=tCategorias[tIndice];
77      return rCategoria;
78    }
79
80    public int calcularVotosNoValidos(){
81      int rVotosNoValidos;
82      rVotosNoValidos=this.totalSufragantes-this.votosValidos;
83      return rVotosNoValidos;
84    }
85
86    public double calcularIndicadorError(){
87      double rIndicadorError;
88      rIndicadorError=(double)this.calcularVotosNoValidos()/this.totalSufragantes;
89      return rIndicadorError;
90    }
91
92    public boolean determinarValidez(){
93      boolean rValidez;
```

```
94          rValidez=((this.nombre.length()>0) &&
95                  (this.potencialSufragantes>0) &&
96                  (this.totalSufragantes>0) &&
97                  (this.totalSufragantes<=this.potencialSufragantes) &&
98                  (this.votosValidos>0) &&
99                  (this.votosValidos<=this.totalSufragantes));
100         return rValidez;
101     }
102 }
```

Tabla 10.17 Codificación de la clase *Presentacion*

```
1   package proyecto;
2   import javax.swing.JOptionPane;
3
4   public class Presentacion {
5      public Presentacion(){
6
7      }
8
9      public Departamento capturarDepartamento(){
10        Departamento rDepartamento;
11        rDepartamento=new Departamento();
12        rDepartamento.setNombre(JOptionPane.showInputDialog("Nombre: "));
13        try{
14           rDepartamento.setPotencialSufragantes(Integer.parseInt(JOptionPane.showInputDialog("Potencial
    de sufragantes: ")));
15           rDepartamento.setTotalSufragantes(Integer.parseInt(JOptionPane.showInputDialog("Total de
    sufragantes: ")));
16           rDepartamento.setVotosValidos(Integer.parseInt(JOptionPane.showInputDialog("Votos válidos: ")));
17        }
18        catch (Exception e){
19
20        }
21        return rDepartamento;
22     }
23
24     public String capturarNombreDepartamento(){
25        String rNombre;
26        rNombre=JOptionPane.showInputDialog("Departamento: ");
27        return rNombre;
28     }
29
30     public void presentarDepartamento(Departamento pDepartamento){
31        String rMensaje;
32        if (pDepartamento.determinarValidez()){
33           rMensaje="Departamento: "+pDepartamento.getNombre();
34           rMensaje=rMensaje+"\nPotencial de sufragantes: "+pDepartamento.getPotencialSufragantes();
35           rMensaje=rMensaje+"\nTotal de sufragantes: "+pDepartamento.getTotalSufragantes();
36           rMensaje=rMensaje+"\nVotos válidos: "+pDepartamento.getVotosValidos();
37           rMensaje=rMensaje+"\nIndicador de participación: "+pDepartamento.calcularParticipacion()+" %";
```

```
38          rMensaje=rMensaje+"\nCategoría según participación:
"+pDepartamento.calcularCategoriaParticipacion();
39          rMensaje=rMensaje+"\nVotos no válidos: "+pDepartamento.calcularVotosNoValidos();
40          rMensaje=rMensaje+"\nIndicador de error: "+pDepartamento.calcularIndicadorError();
41       }
42       else{
43          rMensaje="Datos inconsistentes. No se pueden presentar";
44       }
45       JOptionPane.showMessageDialog(null, rMensaje);
46    }
47 }
```

Tabla 10.18 Codificación de la clase *Menu*

```
1  package proyecto;
2  import javax.swing.JOptionPane;
3
4  public class Menu {
5     public Menu(){
6
7     }
8
9     public int capturarOpcion(){
10       String tMensaje;
11       int rOpcion;
12       tMensaje="1. Registrar departamento\n"+
13             "2. Consultar departamento\n"+
14             "3. Salir\n";
15       rOpcion=Integer.parseInt(JOptionPane.showInputDialog(tMensaje));
16       return rOpcion;
17    }
18 }
```

Tabla 10.19 Codificación de la clase *Datos*

```
1  package proyecto;
2  import java.io.*;
3
4  public class Datos {
5     public Datos(){
6
7     }
8
9     public String construirRegistroDepartamento(Departamento pDepartamento){
10       String rRegistro;
11       rRegistro=pDepartamento.getNombre()+",";
12       rRegistro=rRegistro+pDepartamento.getPotencialSufragantes()+",";
13       rRegistro=rRegistro+pDepartamento.getTotalSufragantes()+",";
14       rRegistro=rRegistro+pDepartamento.getVotosValidos();
```

```
15      return rRegistro;
16    }
17
18    public String descomponerRegistro(String pRegistro, int pOrden){
19      int i,j;
20      String rDato;
21      i=0;
22      j=1;
23      rDato="";
24      while ((i<pRegistro.length()) && (j<=pOrden)) {
25        if (pRegistro.substring(i,i+1).compareTo(",")!=0){
26          if (j==pOrden){
27            rDato=rDato+pRegistro.substring(i,i+1);
28          }
29        }
30        else{
31          j=j+1;
32        }
33        i=i+1;
34      }
35      return rDato;
36    }
37
38    public String recuperarRegistroDepartamento(String pNombre){
39      FileReader fileReader;
40      BufferedReader bufferedReader;
41      String rRegistro,tNombre;
42      try{
43        fileReader=new FileReader("departamentos.txt");
44        bufferedReader=new BufferedReader(fileReader);
45        do{
46          rRegistro=bufferedReader.readLine();
47          tNombre=this.descomponerRegistro(rRegistro, 1);
48        } while (tNombre.compareToIgnoreCase(pNombre)!=0);
49        fileReader.close();
50      }
51      catch (Exception e){
52        rRegistro="";
53      }
54      return rRegistro;
55    }
56
57    public Departamento recuperarDepartamento(String pNombre){
58      String tRegistro;
59      Departamento rDepartamento;
60      rDepartamento=new Departamento();
61      tRegistro=this.recuperarRegistroDepartamento(pNombre);
62      try{
63        rDepartamento.setNombre(this.descomponerRegistro(tRegistro,1));
64        rDepartamento.setPotencialSufragantes(Integer.parseInt(this.descomponerRegistro(tRegistro,2)));
65        rDepartamento.setTotalSufragantes(Integer.parseInt(this.descomponerRegistro(tRegistro,3)));
66        rDepartamento.setVotosValidos(Integer.parseInt(this.descomponerRegistro(tRegistro,4)));
67      }
```

```
68        catch (Exception e){
69
70        }
71        return rDepartamento;
72     }
73
74     public void grabarDepartamento(Departamento pDepartamento){
75        FileWriter fileWriter;
76        PrintWriter printWriter;
77        if ((pDepartamento.determinarValidez()) &&
78           (!this.recuperarDepartamento(pDepartamento.getNombre()).determinarValidez())){
79           try{
80              fileWriter=new FileWriter("departamentos.txt",true);
81              printWriter=new PrintWriter(fileWriter);
82              printWriter.println(this.construirRegistroDepartamento(pDepartamento));
83              fileWriter.close();
84           }
85           catch(Exception e){
86
87           }
88        }
89     }
90  }
```

Tabla 10.20 Codificación de la clase *Main*

```
1   package proyecto;
2
3   public class Main {
4
5      public static void main(String[] args) {
6         Departamento departamento;
7         Presentacion presentacion;
8         Menu menu;
9         Datos datos;
10        int opcion;
11        String parametro;
12        presentacion=new Presentacion();
13        menu=new Menu();
14        datos=new Datos();
15        opcion=0;
16        do {
17           opcion=menu.capturarOpcion();
18           switch (opcion){
19              case 1:{
20                 departamento=presentacion.capturarDepartamento();
21                 datos.grabarDepartamento(departamento);
22                 presentacion.presentarDepartamento(departamento);
23                 break;
24              }
25              case 2:{
```

```
26                  parametro=presentacion.capturarNombreDepartamento();
27                  departamento=datos.recuperarDepartamento(parametro);
28                  presentacion.presentarDepartamento(departamento);
29                  break;
30              }
31          }
32      } while (opcion<=2);
33  }
34  }
```

10.3 Síntesis de conceptos

10.3.1 Programación en Java – Buenas prácticas

- **Uso de arreglos**

Si se tiene una colección de elementos del mismo tipo y se quiere utilizar un mismo nombre para estos, es posible realizarlo por medio de un **arreglo.** Más específicamente, un **arreglo**, es una estructura donde se almacenan datos del mismo tipo. Es necesario declarar inicialmente el tamaño y tipo de datos que este tendrá.

La sintaxis para declarar un **arreglo** es:

tipo_de_variable[ ] nombre_del_array = new tipo_de_variable[dimensión];

tipo_de_variable[ ] nombre_del_array;

nombre_del_array = new tipo_de_variable[dimensión];

Por defecto los **arreglos** se cargan con valores iniciales. Los **arreglos** de datos numéricos se inicializan con el valor 0 en cada uno de sus espacios. Los **arreglos** de datos de cadenas y letras inicializan con valores vacíos, y los **arreglos** de datos booleanos se inicializan con valor falso.

- **Manejo de arreglos:**

Cuando se trabaja con **arreglos** se puede realizar una serie de operaciones básicas con ellos. Dos de las más comunes son la búsqueda y el ordenamiento.

- Búsqueda: El **arreglo** se debe recorrer y cada valor del arreglo se va comparando con el valor que se está buscando. El siguiente código es un ejemplo de un algoritmo de búsqueda:

Tabla 10.21 Algoritmo de búsqueda

```
1   i = 0;
2   while (i < lista.length) {
3      if (lista[i] == valor_a_buscar) {
4         <hacer algo y detener ciclo>
5      }
6      i++;
7   }
```

- Ordenamiento: En ocasiones se requiere organizar grandes cantidades de datos en un **arreglo**, para acceder rápidamente a un valor. Este es un ejemplo de ordenamiento ascendente:

Tabla 10.22 Algoritmo de ordenamiento

```
1  for (i = 0; i < lista.length – 1; i++) {
2     for (j = i + 1; j < lista.length; j++) {
3        if (lista[i] > lista[j]) {
4           temp = lista[i];
5           lista[i] = lista[j];
6           lista[j] = temp;
7        }
8     }
9  }
```