Nivel 1: Problemas, Soluciones y Programas



Objetivos Pedagógicos

**Tarea 1:**

**Objetivo:** Identificar los aspectos que forman parte de un problema.

El problema: un banco quiere crear un programa para manejar sus cajeros automáticos.

Dicho programa sólo debe permitir retirar dinero y consultar el saldo de una cuenta. Identifique y discuta los aspectos que constituyen el problema. Si el enunciado no es explícito con respecto a algún punto, intente imaginar la manera de completarlo.

|  |  |
| --- | --- |
| **Cliente** | Banco |
| Usuario | Clientes con una cuenta en el banco. |
| Requerimiento funcional | El programa del cajero debe permitir retirar y consultar el saldo de una cuenta de un usuario del banco. |
| Mundo del problema | Cajeros a nivel nacional. |
| Requerimiento no funcional | Se debe tener una cuenta y el requisito de un usuario y contraseña |

### Tarea 2

**Objetivo:** Crear habilidad en la identificación y especificación de requerimientos funcionales. Para el caso de estudio 2, un simulador bancario, identifique y especifique tres requerimientos funcionales.

**Requerimiento Funcional 1**

|  |  |
| --- | --- |
| Nombre | R1:  Depositar Dinero |
| Resumen | Permite al usuario depositar un monto especifico de dinero en su cuenta bancaria, en este caso, en la cuenta corriente. |
| Entradas | -Nombre del titular de la cuenta  - Número de cuenta  - Monto a depositar |
| Resultado | 1: El dinero ha sido depositado en su cuenta. |

### Requerimiento Funcional 2

|  |  |
| --- | --- |
| Nombre | R2:  Retirar de dinero |
| Resumen | Permite al usuario retirar un monto especifico de dinero en su cuenta bancaria, en este caso, en la cuenta corriente. |
| Entradas | -Nombre del titular de la cuenta  - Número de cuenta  -Consultar monto de la cuenta  - Monto a retirar |
| Resultado | 1: El dinero ha sido retirado en su cuenta. |

### Requerimiento Funcional 3

### Tarea 3

|  |  |
| --- | --- |
| Nombre | RF3:  Avanzar mes a mes |
| Resumen | El cliente tendrá la posibilidad de avanzar en el tiempo para estar al tanto de cómo va evolucionando su dinero, en base al porcentaje de interés y otros factores. |
| Entradas | -Nombre del titular de la cuenta  -Número de cuenta  -Número de meses para avanzar en el tiempo |
| Resultado | El sistema va a calcular el monto y comportamiento del saldo de la cuenta transcurridos los mese que selecciono el cliente y/o usuario, luego de esto mostrará el resultado de estas operaciones. |

**Objetivo:** Crear habilidad en la identificación y especificación de requerimientos funcionales. Para el caso de estudio 3, un programa para manejar un triángulo identifique y especifique tres requerimientos funcionales.

**Requerimiento Funcional 1**

|  |  |
| --- | --- |
| Nombre | RF1:  Establecer los 3 puntos del triángulo |
| Resumen | El usuario podrá digitar y/o establecer los tres puntos que componen el triángulo |
| Entradas | * Punto nº 1 del triángulo * Punto nº 2 del triángulo * Punto nº 3 del triángulo |
| Resultado | El sistema mostrara como quedo el triángulo con los tres puntos establecidos |
| Nombre | RF2:  Calcular el perímetro del triángulo |
| Resumen | Dados los tres puntos, el programa debe calcular el perímetro de este triángulo |
| Entradas | La longitud de los tres lados del triángulo para sumarlos |
| Resultado | El resultado de la suma de los tres lados en pixeles |

|  |  |
| --- | --- |
| Nombre | RF3:  Calcular área del triángulo |
| Resumen | Se podrá calcular el área del triangulo si se sabe la medida de la base y la altura |
| Entradas | 1. Medida de la base del triángulo 2. Medida de la altura del triángulo |
| Resultado | Luego del cálculo (b \* a / 2), el sistema mostrara el área del triángulo en pixeles |

|  |
| --- |
|  |

### Ejemplo 4

**Objetivo:** Ilustrar la manera de identificar las entidades (llamadas también clases) del mundo del problema.

En este ejemplo se identifican las entidades que forman parte del mundo del problema para el caso 2 de este nivel: un simulador bancario.

|  |  |
| --- | --- |
| **Entidad** | **Descripción** |
| CuentaBancaria | Es la entidad más importante del mundo del problema, puesto que define su frontera (todo lo que está por fuera de la cuenta bancaria no nos interesa). Es buena práctica comenzar la etapa de análisis tratando de identificar la clase más importante del problema. Cuando el nombre de la entidad es compuesto, se usa por convención una letra mayúscula al comienzo de cada palabra. En otra época se utilizaban el carácter "\_" para separar las palabras (Cuenta\_Bancaria) pero eso está pasado de moda. |
| CuentaCorriente | Este es otro concepto que existe en el mundo del problema. Según el enunciado una cuenta corriente forma parte de una cuenta bancaria, luego esta entidad está "dentro" de la frontera que nos interesa. Por ahora no nos interesan los detalles de la cuenta corriente (por ejemplo si tiene un saldo o si paga intereses). En este momento sólo queremos identificar los elementos del mundo del problema que están involucrados en los requerimientos funcionales. |
| CuentaAhorros | Este es el tercer concepto que aparece en el mundo del problema.  De la misma manera que en el caso anterior, una cuenta bancaria "incluye" una cuenta de ahorros. Los nombres asignados a las clases deben ser significativos y dar una idea clara de la entidad del mundo que representan. No se debe exagerar con la longitud del nombre, porque de lo contrario los programas pueden resultar pesados de leer. |
| CDT | El nombre de esta clase se encuentra en mayúsculas, porque es una sigla. Otro nombre para esta clase habría podido ser el nombre completo del concepto: CertificadoDepositoTermino. En el lenguaje Java no es posible usar tildes en los nombres de los clases, así que nunca veremos una clase llamada CertificadoDepósitoTérmino. |
| Mes | El concepto de mes es el quinto y último concepto del caso de estudio. Esta será la clase que nos dirá en cuál mes de la simulación se encuentra la cuenta bancaria. No olvide que la simulación se hace mes a mes, y que existe un requerimiento funcional que permite avanzar un mes en la simulación. Si lee de nuevo el enunciado del caso, se dará cuenta de que todos los demás elementos del problema son características de las entidades anteriormente mencionadas o requerimientos funcionales. |

### Tarea 4

**Objetivo:** Identificar las entidades del mundo para el caso de estudio3: un programa que maneje un triángulo.

Lea el enunciado del caso y trate de guiarse por los sustantivos para identificar las entidades del mundo del problema.

|  |  |  |
| --- | --- | --- |
|  | **Nombre** | **Descripción** |
| Entidad | Puntos/medidas del triángulo | Los puntos con los cuales se van a diseñar el triangulo son una entidad importante, puesto que luego de definirlos se van a poder desarrollar las demás operaciones |
| Entidad | Perímetro | Esta clase nos definirá los valores u operaciones para saber el perímetro del triángulo diseñado |
| Entidad | Área | Esta clase nos definirá los valores u operaciones para saber el área del triángulo establecido |
| **Punto de reflexión: ¿Qué pasa si no identificamos bien las entidades del mundo?** | | |
| El sistema o programa no se podrá desarrollar bien, o simplemente se va a realizar algo que no se pidió. | | |
| **Punto de reflexión: ¿Cómo decidir si se trata efectivamente de una entidad y no sólo de una característica de una entidad ya identificada?** | | |
| Estableciendo o conociendo la importancia o relevancia de esta en el programa, igualmente el contexto del sistema a desarrollar. | | |

### Ejemplo 5

**Objetivo:** Mostrar la manera de identificar y modelar los atributos de una clase.

En este ejemplo se identifican las características de las clases Empleado y Fecha para el caso de estudio del empleado.

**Clase:** Empleado

|  |  |  |
| --- | --- | --- |
| **Atributo** | **Valores Posibles** | **Comentarios** |
| nombre | Cadena de caracteres | La primera característica que aparece en el enunciado es el nombre del empleado. El valor de este atributo es una cadena de caracteres (por ejemplo, "Juan") Seleccionamos "nombre" como nombre del atributo. Es importante que los nombres de los atributos sean significativos (deben dar una idea clara de lo que una característica representa), para facilitar así la lectura y la escritura de los programas. |
| apellido | Cadena de caracteres | El segundo atributo es el apellido del empleado. Al igual que en el caso anterior, el valor que puede tomar este atributo es una cadena de caracteres (por ejemplo, "Pérez"). Como nombre del atributo seleccionamos "apellido". El nombre de un atributo debe ser único dentro de la clase (no es posible dar el mismo nombre a dos atributos). |
| sexo | Masculino o Femenino | Esta característica puede tomar dos valores: masculino o femenino. En esta etapa de análisis basta con identificar los valores posibles. Es importante destacar que los valores posibles de este atributo (llamado "sexo") no son cadenas de caracteres. No nos interesan las palabras en español que pueden describir los valores posibles de esta característica, sino los valores en sí mismos. |
| salario | Valores enteros positivos | El salario está expresado en pesos y su valor es un número entero positivo. Aquí estamos suponiendo que el valor del salario no tiene centavos. |

### Tarea 5

Para cada una de las cinco entidades identificadas en el caso de estudio del simulador bancario, identifique los atributos, sus valores posibles, y escriba la clase en UML. No incluya las relaciones que puedan existir entre las clases, ya que eso lo haremos en la siguiente etapa del análisis. Por ahora trate de identificar las características de las entidades que son importantes para los requerimientos funcionales.

**Clase:** Cuenta Bancaria

|  |  |
| --- | --- |
| **Atributo** | **Valores Posibles** |
| Numero de Cuenta | Valores enteros positivos |
| Nombre del Titular | Cadena de caracteres |
| Saldo total de la cuenta | Valores enteros |

Diagrama UML:

|  |
| --- |
| Cuenta Bancaria |
| Número de cuenta  Nombre del titular  Saldo total de la cuenta |
|  |

**Clase:** Cuenta Corriente

|  |  |
| --- | --- |
| **Atributo** | **Valores Posibles** |
| Número de la cedula del titular | Valores enteros |
| Nombre del titular | Cadena de caracteres |
| Saldo actual de la cuenta | Valores enteros |

|  |
| --- |
| Cuenta Corriente |
| Número de la cedula del titular  Nombre del titular  Saldo actual de la cuenta |
|  |

Diagrama UML:

**Clase:** Cuenta Ahorros

|  |  |
| --- | --- |
| **Atributo** | **Valores Posibles** |
| Nombre del titular | Cadena de caracteres |
| Interés Mensual | Números de punto flotante |
| Saldo de la cuenta | Valores enteros |

|  |
| --- |
| Cuenta de Ahorros |
| Nombre del titular  Interés mensual  Saldo de la cuenta |
|  |

Diagrama UML:

**Clase:** CDT

|  |  |
| --- | --- |
| **Atributo** | **Valores Posibles** |
| Nombre del titular | Cadena de caracteres |
| Cantidad de dinero a invertir | Valores enteros positivos |
| Interés mensual para recibir | Número con punto flotante |

|  |
| --- |
| CDT |
| Nombre del titular  Cantidad de dinero a invertir  Interés mensual para recibir |
|  |

Diagrama UML:

**Clase:** Mes

|  |  |
| --- | --- |
| **Atributo** | **Valores Posibles** |
| Nombre del mes | Cadena de caracteres |
| Dia del mes | Valores enteros entre 1 y 31 |
| Número de mes | Valores enteros entre 1 y 12 |

|  |
| --- |
| Mes |
| Nombre del mes  Dia del mes  Número de mes |
|  |

Diagrama UML:

# Elementos de un Programa

En esta parte del capítulo presentamos los distintos elementos que forman parte de un programa. No pretende ser una exposición exhaustiva, pero sí es nuestro objetivo dar una visión global de los distintos aspectos que intervienen en un programa.

En algunos casos la presentación de los conceptos es muy superficial. Ya nos tomaremos el tiempo en los niveles posteriores de profundizar poco a poco en cada uno de ellos. Por ahora lo único importante es poderlos usar en casos limitados. Esta manera de presentar los temas nos va a permitir generar las habilidades de uso de manera incremental, sin necesidad de estudiar toda la teoría ligada a un concepto antes de poder usarlo.

## Algoritmos e Instrucciones

Los algoritmos son uno de los elementos esenciales de un programa. Un **algoritmo** se puede ver como la solución de un problema muy preciso y pequeño, en el cual se define la secuencia de instrucciones que se debe seguir para resolverlo. Imagine, entonces, un programa como un conjunto de algoritmos, cada uno responsable de una parte de la solución del problema global.

Un algoritmo, en general, es una secuencia ordenada de pasos para realizar una actividad. Suponga, por ejemplo, que le vamos a explicar a alguien lo que debe hacer para viajar en el metro parisino. El siguiente es un algoritmo de lo que esta persona debe hacer para llegar a una dirección dada:

1. Compre un tiquete de viaje en los puntos de venta que se encuentran a la entrada de cada una de las estaciones del metro.
2. Identifique en el mapa del metro la estación donde está y el punto adonde necesita ir.
3. Localice el nombre de la estación de metro más cercana al lugar de destino.
4. Verifique si, a partir de donde está, hay alguna línea que pase por la estación destino.
5. Si encontró la línea, busque el nombre de la misma en la dirección de destino.
6. Suba al metro en el andén de la línea identificada en el paso anterior y bájese en la estación de destino.

### Tarea 6

**Objetivo:** Reflexionar sobre el nivel de precisión que debe ser usado en un algoritmo para evitar ambigüedades.

Suponga que usted es la persona que va a utilizar el algoritmo anterior, para moverse en el metro de París. Identifique qué problemas podría tener con las instrucciones anteriores.

Piense por ejemplo si están completas.

¿Se prestan para que se interpreten de maneras distintas? ¿Estamos suponiendo que quién lo lee usa su "sentido común", o cualquier persona que lo use va a resolver siempre el problema de la misma manera?

|  |  |
| --- | --- |
| Utilice este espacio para anotar sus conclusiones:   |  | | --- | | Pienso que las personas que lo usen deben tener sentido común, seguir las instrucciones para que no se pierdan. Aunque también puede ser que sigan algunos pasos sin tener un orden, por ejemplo, siguiendo el paso 4 antes del 3, entonces se pueden presentar algunos problemas. | |

### Tarea 7

**Objetivo:** Entender la complejidad que tiene la tarea de escribir un algoritmo.

Esta tarea es para ser desarrollada en parejas:

1. En el primer cuadrante haga un dibujo simple.
2. En el segundo cuadrante escriba las instrucciones para explicarle a la otra persona cómo hacer el dibujo.
3. Lea las instrucciones a la otra persona, quien debe intentar seguirlas sin ninguna ayuda adicional.
4. Compare el dibujo inicial y el dibujo resultante.

|  |  |
| --- | --- |
| **Dibujo:** | **Algoritmo** |
|  | -Primero haga un círculo, dentro del circulo haga dos puntos y una línea que simule una sonrisa.  -Luego haga una línea abajo del círculo.  -Después haga una línea en cada lado de la línea anterior (brazos).  -Por último, dibuje una línea en cada lado de la línea principal, pero un poco más debajo de las líneas del punto anterior (piernas). |
| **Haga una síntesis de los resultados obtenidos:** | |
| El dibujo hecho por mi compañero quedo parecido al mío, aunque con unos cambios, hubo una confusión en el último paso, pero luego se fue solucionando a medida que se ejecutaba. | |

Cuando es el computador el que sigue un algoritmo (en el caso del computador se habla de **ejecutar**), es evidente que las instrucciones que le demos no pueden ser como las definidas en el algoritmo del metro de París. Dado que el computador no tiene nada parecido al "sentido común", las instrucciones que le definamos deben estar escritas en un lenguaje que no dé espacio a ninguna ambigüedad (imaginemos al computador de una nave espacial diciendo "es que yo creí que eso era lo que ustedes querían que yo hiciera"). Por esta razón los algoritmos que constituyen la solución de un problema se deben traducir a un lenguaje increíblemente restringido y limitado (pero a su vez poderoso si vemos todo lo que con él podemos hacer), denominado un **lenguaje de programación**. Todo lenguaje de programación tiene su propio conjunto de reglas para decir las cosas, denominado la **sintaxis** del lenguaje.

Existen muchos lenguajes de programación en el mundo, cada uno con sus propias características y ventajas. Como dijimos anteriormente, en este libro utilizaremos el lenguaje de programación Java que es un lenguaje de propósito general (no fue escrito para resolver problemas en un dominio específico), muy utilizado hoy en día en el mundo entero, tanto a nivel científico como empresarial.

Un programa de computador está compuesto por un conjunto de algoritmos, escritos en un lenguaje de programación. Dichos algoritmos están estructurados de tal forma que, en conjunto, son capaces de resolver el problema.

## Clases y Objetos

Las clases son los elementos que definen la estructura de un programa. Tal como vimos en la etapa de análisis, las clases representan entidades del mundo del problema (más adelante veremos que también pueden pertenecer a lo que denominaremos el mundo de la solución). Por ahora, y para que se pueda dar una idea de lo que es un programa completo, imagine que los algoritmos están dentro de las clases, y que son estas últimas las que establecen la manera en que los algoritmos colaboran para resolver el problema global (ver figura 1.9). Esta

|  |
| --- |
|  |

### Tarea 8

**Objetivo:** Crear habilidad en la definición de los tipos de datos para representar las características de una clase.

Escriba en Java y en UML las declaraciones de los atributos (y las asociaciones) para las cinco clases del caso de estudio del simulador bancario.

|  |  |
| --- | --- |
| **Declaración en Java** | **Descripción de la clase en UML** |
| public class CuentaBancaria {  private int Numero cuenta; public str nombre titular; private int saldo total;  } | |  | | --- | | Cuenta Bancaria | | Número de cuenta  Nombre del titular  Saldo total de la cuenta | |  | |

|  |  |
| --- | --- |
| **Declaración en Java** | **Descripción de la clase en UML** |
| public class Cuentacorriente {  private int Numero cedula; public str nombre titular; private int saldo actual;  } | |  | | --- | | Cuenta Corriente | | Número de la cedula del titular  Nombre del titular  Saldo actual de la cuenta | |  | |
| **Declaración en Java** | **Descripción de la clase en UML** |
| public class CuentaAhorros {  public str nombre titular;  public float InteresMens  private int saldo total;  } | |  | | --- | | Cuenta de Ahorros | | Nombre del titular  Interés mensual  Saldo de la cuenta | |  | |

|  |  |
| --- | --- |
| **Declaración en Java** | **Descripción de la clase en UML** |
| public class CDT {  public str nombre titular.  private int CantDinero;  private float InteresMens;  } | |  | | --- | | CDT | | Nombre del titular  Cantidad de dinero a invertir  Interés mensual para recibir | |  | |
| **Declaración en Java** | **Descripción de la clase en UML** |
| public class Mes {  private int nombreMes; private int DiaMes;  private int NumeroMes;  } | |  | | --- | | Mes | | Nombre del mes  Dia del mes  Número de mes | |  | |

## Métodos

### Ejemplo 9

**Objetivo:** Mostrar la sintaxis que se usa en Java para declarar un método.

Usamos para esto el caso de estudio del empleado, con tres métodos sin cuerpo, suponiendo que cada uno debe resolver el problema que ahí mismo se describe. La declaración que aquí se muestra hace parte de la declaración de la clase (los métodos van después de la declaración de los atributos). Se deja un cuarto método al final como tarea para el lector; en este caso, a partir de la descripción, debe determinar los parámetros, el retorno y la signatura del método.

public void cambiarSalario( int nuevoSalario)

{

// Aquí va el cuerpo del método

}

**Nombre:** cambiarSalario

**Parámetros:** nuevoSalario de tipo entero. Si no se entrega este valor como parámetro es imposible cambiar el salario del empleado. Note que al definir un parámetro se debe dar un nombre al valor que se espera y un tipo.

**Retorno:** ninguno ( void ) puesto que el objetivo del método no es calcular ningún valor, sino modificar el valor de un atributo del empleado.

**Descripción:** cambia el salario del empleado, asignándole el valor que se entrega como parámetro.

public int darSalario( )

{

// Aquí va el cuerpo del método

}

**Nombre:** darSalario

**Parámetros:** ninguno, puesto que con la información que ya tienen los objetos de la clase Empleado es posible resolver el problema.

**Descripción:** retorna el salario actual del empleado.

public int calcularPrestaciones( )

{

// Aquí va el cuerpo del método

}

**Nombre:** calcularPrestaciones

**Parámetros:** ninguno. Al igual que en el método anterior, no se necesita información externa al empleado para poder calcular sus prestaciones.

**Retorno:** las prestaciones anuales a las que tiene derecho el empleado. Dado que el salario es entero, vamos a suponer que las prestaciones también lo son.

**Descripción:** retorna el valor de las prestaciones anuales a las que tiene derecho el empleado.

**Nombre:** aumentarSalario

**Parámetros:**



**Retorno:**



**Descripción:** aumenta el salario del empleado en un porcentaje que corresponde a la inflación anual del país.

¿Cuáles son los métodos que se deben tener en una clase? Esa es una pregunta que se contestará en niveles posteriores. Por ahora, supongamos que la clase tiene ya definidos los métodos que necesita para poder resolver la parte del problema que le corresponde y trabajemos en el cuerpo de ellos. En el diagrama de clases de UML, se utiliza la tercera zona de la caja de una clase para poner las signaturas de los métodos, tal como se ilustra en la figura 1.14.

|  |
| --- |
| **Fig. 1.14 Sintaxis en UML para mostrar las signaturas de los métodos de una clase** |
|  |

### Tarea 9

**Objetivo:** Escribir y entender en Java la signatura de algunos métodos del caso de estudio del simulador bancario.

Complete la siguiente información, ya sea escribiendo la signatura del método que se describe, o interpretando la signatura que se da. Todos los métodos de esta tarea son de la clase CuentaAhorros.

public void consignarValor( int valor )

{

}

|  |  |  |
| --- | --- | --- |
| **Nombre:** | | Consignar valor |
| Parámetros: | | ValoraConsignar valor entero: Un valor en positivo para agregar a la cuenta |
| Retorno: | | Ninguno, solo se le agrega el nuevo monto a la cuenta |
| Descripción: | | El usuario ingresara un valor para consignar en la cuenta de Ahorros |
| **Nombre:** | **darSaldo** | |
| Parámetros: | ninguno. | |
| Retorno: | valor de tipo entero (ignora los centavos en el momento de dar el saldo). | |
| Descripción: | retorna el saldo de la cuenta de ahorros. | |

**Signatura del Método:**

|  |
| --- |
| Cuenta de Ahorros |
| Nombre del titular  Interés mensual  Saldo de la cuenta |
| Int DarSaldo() |

|  |  |
| --- | --- |
| **Nombre:** | **retirarValor** |
| Parámetros: | valor de tipo entero, que indica el monto que se quiere retirar de la cuenta de ahorros. |
| Retorno: | ninguno. |
| Descripción: | retira de la cuenta de ahorros el valor que se entrega como parámetro. |

**Signatura del Método:**

|  |
| --- |
| Cuenta de Ahorros |
| Nombre del titular  Interés mensual  Saldo de la cuenta |
| Void RetirarValor(Valor retirado) |

|  |  |
| --- | --- |
| **Nombre:** | **darInteresMensual** |
| Parámetros: | ninguno. |
| Retorno: | valor de tipo real. |
| Descripción: | retorna el interés mensual que paga una cuenta de ahorros. |

**Signatura del Método:**

|  |
| --- |
| Cuenta de Ahorros |
| Nombre del titular  Interés mensual  Saldo de la cuenta |
| Int DarInteresMensual() |

|  |  |
| --- | --- |
| **Nombre:** | **actualizarSaldoPorPasoMes** |
| Parámetros: | ninguno. |
| Retorno: | ninguno. |
| Descripción: | actualiza el saldo de la cuenta de ahorros simulando que acaba de transcurrir un mes y que se deben agregar los correspondientes intereses ganados. |

**Signatura del Método:**

|  |
| --- |
| Cuenta de Ahorros |
| Nombre del titular  Interés mensual  Saldo de la cuenta |
| void actualizarSaldoPorPasoMes(Saldo-del-mes-seleccionado) |

## La Instrucción de Retorno

En el cuerpo de un método van las instrucciones que resuelven un problema puntual o prestan un servicio a otras clases. El computador obedece las instrucciones, una después de otra, hasta llegar al final del cuerpo del método. Hay instrucciones de diversos tipos, la más sencilla de las cuales es la instrucción de retorno ( return ).

## La Instrucción de Asignación

Los métodos que no están hechos para calcular un valor, sino para modificar el estado del objeto, utilizan la instrucción de asignación ( = ) para definir el nuevo valor que debe tener el atributo. Si existiera, por ejemplo, un método para duplicar el salario de un empleado, el siguiente sería el cuerpo de dicho método:

|  |
| --- |
| public class Empleado  { ...  public void duplicarSalario( )  {  salario = salario \* 2;  }  } |

En la parte izquierda de la asignación va el atributo que va a ser modificado (más adelante se extenderá a otros elementos del lenguaje, pero por ahora puede suponer que sólo se hacen asignaciones sobre los atributos). En la parte derecha va una **expresión** que indica el nuevo valor que debe guardarse en el atributo. Pueden formar parte de una expresión los atributos (incluso el que va a ser modificado), los parámetros y los valores constantes (como el 2 en el ejemplo anterior). Los elementos que forman parte de una expresión se denominan **operandos**. Adicionalmente en la expresión están los **operadores**, que indican cómo calcular el valor de la expresión. Los operadores aritméticos son la suma ( + ), la resta ( - ), la multiplicación ( \* ) y la división ( / ).

## La Instrucción de Llamada de un Método

En algunos casos, como parte de la solución del problema, es necesario llamar un método de un objeto con el cual existe una asociación. Suponga que un empleado necesita saber el año en el que él ingresó a la empresa. Esa información la tiene el objeto de la clase Fecha que está siendo referenciado por su atributo  fechaIngreso . Puesto que la clase Empleado no tiene acceso directo a los atributos de la clase Fecha, debe llamar el método de dicha clase que presta ese servicio (o que sabe resolver ese problema puntual). La sintaxis para hacerlo y el proceso de llamada (o invocación) se ilustran a continuación:

|  |  |  |
| --- | --- | --- |
|  | **Diagrama de objetos para ilustrar la**  **llamada del método: la empleada María**  **Gómez ingresó a la empresa a trabajar en el año 2005.** | |
|  |  | |
| public class Empleado  { ...  public void miProblema( )  {  int valor = fechaIngreso.darAnio( ); ...  }  } | | |

Dentro de un método de la clase Empleado se necesita saber el año de ingreso a la empresa.

Invocamos el método darAnio( ) sobre el objeto de la clase Fecha que representa la fecha de ingreso. Ese método debe retornar 2005 si el diagrama de objetos es el mostrado en la figura anterior.

Para pedir un servicio a través de un método,

|  |
| --- |
| public class Empleado  { ...  public int calcularSalarioAnual( )  {  return salario \* 12;  }  public double calcularImpuesto( )  {  int total = calcularSalarioAnual( ); return total \* 19.5 / 100;  }  } |

Suponga que queremos calcular el monto de los impuestos que debe pagar el empleado en un año. Los impuestos se calculan como el 19,5% del total de salarios recibidos en un año.

Si ya tenemos un método que calcula el valor total del salario anual, ¿por qué no lo utilizamos como parte de la solución? Eso nos va a permitir disminuir la complejidad del problema puntual del método, porque nos podemos concentrar en la parte que "nos falta" para resolverlo.

Para invocar un método sobre el mismo objeto, basta con utilizar su nombre sin necesidad de explicar sobre cuál objeto queremos hacer la llamada. Por defecto se hace sobre él mismo.

Note que utilizamos una variable ( total ) como parte del cuerpo del método. Una variable se utiliza para almacenar valores intermedios dentro del cuerpo de un método. Una variable debe tener un nombre y un tipo, y sólo puede utilizarse dentro del método dentro del cual fue declarada. En el siguiente capítulo volveremos a tratar el tema de las variables.

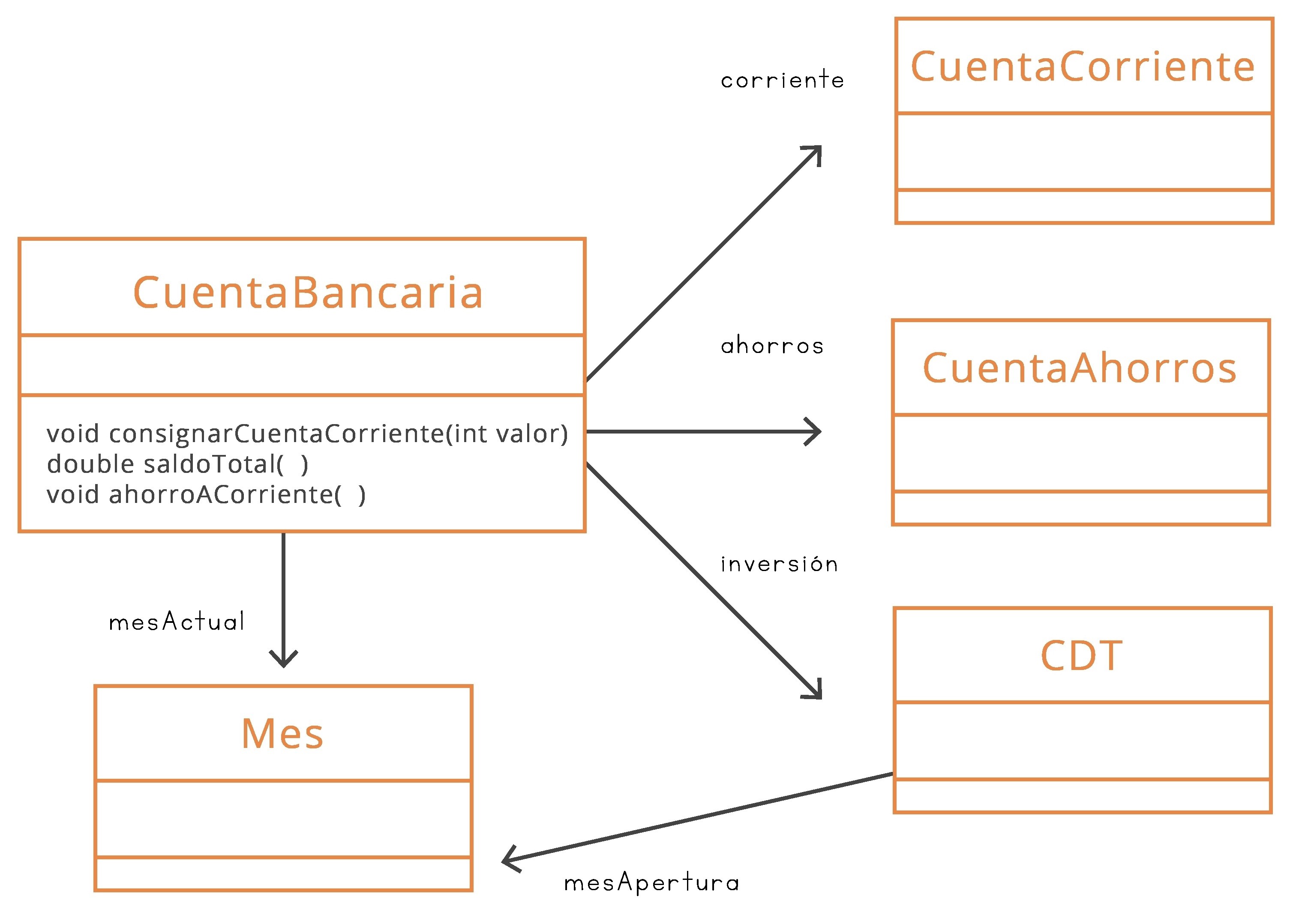
### Ejemplo 10

**Objetivo:** Ilustrar la construcción de los métodos de una clase.

Para el caso de estudio del simulador bancario, en este ejemplo se muestra el código de algunos métodos, en donde se pueden apreciar los distintos tipos de instrucción que hemos visto hasta ahora.

|  |
| --- |
| package uniandes.cupi2.simuladorBancario.mundo;  public class CuentaBancaria  {  //------------------------------------ // Atributos  //------------------------------------ private String cedula; private String nombre;  private CuentaCorriente corriente; private CuentaAhorros ahorros; private CDT inversion; private Mes mesActual; ...  } |

Declaración de los atributos de la clase que representa la cuenta bancaria. Note de nuevo la manera en que se declaran las relaciones con otras clases (como atributos, cuyo nombre corresponde al nombre de la asociación).



public void consignarCuentaCorriente( int valor )

{

corriente.consignarValor( valor );

}

Para depositar en la cuenta corriente un valor que llega como parámetro, la cuenta bancaria pide dicho servicio al objeto que representa la cuenta corriente, usando la asociación que hay entre los dos y el método consignarValor() de la clase CuentaCorriente.

public double saldoTotal( )

{

return corriente.darSaldo( ) + ahorros.darSaldo( ) +

inversión.valorPresente( mesActual );

}

Para calcular y retornar el saldo total de la cuenta bancaria, el método pide a cada uno de los productos que la componen que calcule su valor actual. Luego, suma dichos valores y los retorna como el resultado. Fíjese que una expresión puede estar separada en varias líneas, mientras no aparezca el símbolo ";" de final de una instrucción.

Para calcular el valor presente del CDT se le debe pasar como parámetro el mes en el que va la simulación.

public void ahorroACorriente( )

{

int temp = ahorros.calcularSaldo( ); ahorros.retirar( temp ); corriente.consignarValor( temp );

}

Este método pasa todo el dinero depositado en la cuenta de ahorros a la cuenta corriente. Fíjese que es indispensable utilizar una variable ( temp ) para almacenar el valor temporal que se debe mover. ¿Se podría hacer sin esa variable? Las variables se declaran dentro del método que la va a utilizar y se pueden usar dentro de las expresiones que van en el cuerpo del método.

Si hay necesidad de convertir un valor real en un valor entero, se puede usar el operador de conversión  (int) . Dicho operador se utiliza de la siguiente manera:  int respuesta = ( int )( 1000 / 33 ); En ese caso, el computador primero evalúa la expresión y luego elimina las cifras decimales.

### Tarea 10

Escriba el cuerpo de los métodos de la clase CuentaBancaria (caso de estudio 2) cuya signatura aparece a continuación. Utilice los nombres de los atributos que aparecen en la declaración de la clase. Suponga que existen los métodos que necesite en las clases CuentaCorriente, CuentaAhorros, CDT y Mes.

Public void ahorrar (int valor)

{

CuentaAhorros.consignar(valor);

}

Pasa de la cuenta corriente a la cuenta de ahorros el valor que se entrega como parámetro (suponiendo que hay suficientes fondos).

Public void RetirarAhorro (int valor)

{

CuentaAhorros.retirar(valor);

}

Retira un valor dado de la cuenta de ahorros (suponiendo que hay suficientes fondos).

Public int darSaldoCorriente() {

return cuentaCorriente.getSaldo();

}

Retorna el saldo que hay en la cuenta corriente. No olvide que éste es un método de la clase CuentaBancaria.

public void retirarTodo(int saldoCuentaCorriente, int saldoCuentaAhorros )

{

cuentaCorriente.retirar(saldoCuentaCorriente);

cuentaAhorros.retirar(saldoCuentaAhorros);

}

Retira todo el dinero que hay en la cuenta corriente y en la cuenta de ahorros.

public void duplicarAhorro( )

{

cuentaAhorros.depositar(saldoActual);

}

Duplica la cantidad de dinero que hay en la cuenta de ahorros.

public void avanzarSimulacion( )

{

mesActual.avanzar(saldoactual);

}

Avanza un mes la simulación de la cuenta bancaria.

Dentro de un método:

Para hacer referencia a un atributo basta con utilizar su nombre ( salario ). Para invocar un método sobre el mismo objeto, se debe dar únicamente el nombre del método y la lista de valores para los parámetros (cambiarSalario( 2000000 )). Para invocar un método sobre un objeto con el cual se tiene una asociación, se debe dar el nombre de la asociación, seguido de un punto y luego la lista de valores para los parámetros (fechaIngreso.darDia( )).

## Llamada de Métodos con Parámetros

Este tema se profundizará en los capítulos posteriores. Por ahora sólo queremos dar una idea global del proceso de llamada de un método con parámetros. Para eso vamos a contestar siete preguntas:

¿Cuándo necesita parámetros un método? Un método necesita parámetros cuando la información que tiene el objeto en sus atributos no es suficiente para resolver el problema que le plantean.

¿Cómo se declara un parámetro? En la signatura del método se define el tipo de dato del parámetro y se le asocia un nombre. Es conveniente que este nombre dé una idea clara del valor que se va a recibir por ese medio.

¿Cómo se utiliza el valor del parámetro? Basta con utilizar el nombre del parámetro en el cuerpo del método, de la misma manera en que se utilizan los atributos.

¿Se puede utilizar el parámetro por fuera del cuerpo del método? No. En ningún caso. Aquél que hace la llamada del método, ¿cómo hace para definir los valores de los parámetros? En el momento de hacer la llamada, se deben pasar tantos valores como parámetros está esperando el método. Esos valores pueden ser constantes (por ejemplo, 500), atributos del objeto que hace la llamada (por ejemplo,  salario ), parámetros del método desde el cual se hace la llamada (por ejemplo,  nuevoSalario ), o expresiones que mezclen los tres anteriores (por ejemplo, salario + nuevoSalario \*

500 ).

¿Cómo se hace la relación entre esos valores y los parámetros? Los valores se deben pasar teniendo en cuenta el orden en el que se declararon los parámetros. Eso se ilustra en la figura 1.15.

¿Qué sucede si se pasan más (o menos) valores que parámetros? El compilador informa que hay un error en la llamada. Lo mismo sucede si los tipos de datos de los valores no coinciden con los tipos de datos de los parámetros.

|  |
| --- |
| **Fig. 1.15 Llamada de un método con parámetros** |
|  |

Tenemos una clase C1, con un método m1() que tiene tres parámetros.

Tenemos una clase C2, con un atributo de la clase C1. Desde allí vamos a llamar el método m1() de la primera clase.

Debemos pasarle 3 valores en el momento de invocar el método. El primer valor es el parámetro x del método m2( ). El segundo valor es una expresión que incluye una constante y un atributo. El tercer valor es una constante de tipo cadena de caracteres. Al hacer la llamada se hace la correspondencia uno a uno entre los valores y los parámetros.

Después de hacer la correspondencia se calcula cada valor y se le asigna al respectivo parámetro. Esta copia del valor se hace para todos los tipos simples de datos. Una vez que se han inicializado los parámetros se inicia la ejecución del método.

# Diseño de la Solución

En esta sección se da una visión global de la etapa de diseño, la segunda etapa del proceso de desarrollo de un programa.

Si hacemos el paralelo con el trabajo de un arquitecto que construye un edificio, podemos imaginar que éste, una vez que ha terminado de entender lo que el cliente quiere, empieza la etapa de diseño del edificio. La figura 1.16 pretende mostrar que la actividad de diseño se suele desarrollar a través de refinamientos sucesivos: el arquitecto primero hace un bosquejo de lo que quiere construir, luego hace los cálculos necesarios para verificar si esta solución es viable (debe por ejemplo estimar los materiales y el costo de mano de obra). Si llega a la conclusión de que no cumple por alguna razón las restricciones impuestas por el cliente (o se le ocurre una manera mejor de hacerlo), realiza los ajustes del caso y repite de nuevo la etapa de cálculos. La actividad termina cuando el arquitecto decide que encontró una buena solución al problema. En ese momento comienza a elaborar un conjunto de planos que van a ser utilizados como guía para la construcción del edificio.

|  |
| --- |
| **Fig. 1.16 El diseño es una actividad iterativa hasta encontrar una solución** |
|  |

En el caso de la construcción de un programa, la actividad de diseño sigue el mismo esquema: nuestro bosquejo inicial es el modelo conceptual del mundo del problema, nuestros cálculos consisten en verificar los requerimientos no funcionales y calcular el costo de implementación, y nuestros planos son, entre otros, diagramas detallados escritos en UML. En cada refinamiento introducimos o ajustamos algunos de los elementos del programa y así nos vamos aproximando a una solución adecuada.

Como se muestra en la figura 1.17, los documentos de diseño (nuestros "planos") deben hacer referencia al menos a tres aspectos:

1. El diseño de la interfaz de usuario.
2. La arquitectura de la solución.
3. El diseño de las clases.

|  |
| --- |
| **Fig. 1.17 Entradas y salidas de la etapa de diseño** |
|  |

Como entrada tenemos el análisis del problema, dividido en tres partes: requerimientos funcionales, mundo del problema y requerimientos no funcionales.

La salida es el diseño del programa, que incluye la interfaz de usuario, la arquitectura y el diseño de las clases.

## La Interfaz de Usuario

La interfaz de usuario es la parte de la solución que permite que los usuarios interactúen con el programa. A través de la interfaz, el usuario puede utilizar las operaciones del programa que implementan los requerimientos funcionales. La manera de construir esta interfaz será el tema del nivel 5 de este libro. Hasta entonces, todas las interfaces que se necesitan para completar los programas de los casos de estudio serán dadas.

## La Arquitectura de la Solución

En general, cuando se quiere resolver un problema, es bueno contar con mecanismos que ayuden a dividirlo en problemas más pequeños. Estos problemas son menos complejos que el problema original y, por lo tanto, más fáciles de resolver.

Por ejemplo, si se quiere construir un aeropuerto, al plantear la solución, los diseñadores identifican sus grandes partes: las pistas de aterrizaje, las salas de llegada y salida de pasajeros, la torre de control, etc. Luego tratan de diseñar esas partes por separado, sabiendo que cada diseño es más sencillo que el diseño completo del aeropuerto.

En los problemas en los que vamos a trabajar a lo largo del libro, se pueden identificar 3 grandes grupos de clases:

1. Las clases que implementan la interfaz de usuario. 2. Las clases que implementan el modelo del mundo.

3. Las clases que implementan las pruebas.

4.

5. Cada uno de estos grupos va a ir en un paquete distinto. Esta manera de separar la aplicación en estos tres paquetes la vamos a llamar la **arquitectura básica** y la estaremos utilizando en la gran mayoría de los casos de estudio de este libro. La figura 1.18 ilustra la arquitectura de la solución para el caso de estudio del empleado, en la cual se puede apreciar que hay tres paquetes, que cada uno tiene en su interior un grupo de clases, y que estos paquetes están relacionados (la relación está indicada por las flechas punteadas).

|  |
| --- |
| **Fig. 1.18 Arquitectura de paquetes del caso de estudio del empleado** |
|  |

Sin entrar por ahora en mayores detalles, podemos decir que en el paquete de la interfaz estarán las clases que implementan los elementos gráficos y de interacción, lo mismo que las clases que implementan los requerimientos funcionales y las clases que crean las instancias del modelo del mundo. Es allí donde están agrupadas todas esas responsabilidades. Este es el tema del nivel 5 de este libro. Por ahora, paciencia...

## El Diseño de las Clases

El objetivo de esta parte de la etapa de diseño es mostrar los detalles de cada una de las clases que van a hacer parte del programa. Para esto vamos a utilizar el diagrama de clases de UML, con toda la información que presentamos en las secciones anteriores (clases, atributos y signaturas de los métodos). En el nivel 4, veremos la manera de precisar las responsabilidades y compromisos de cada uno de los métodos (exactamente qué debe hacer cada método), de manera que la persona que vaya a implementar los métodos no deba guiarse únicamente por los nombres de los mismos.

# Construcción de la Solución

## Visión Global

|  |
| --- |
| **Fig. 1.19 Entradas y salidas de la etapa de construcción de la solución** |
|  |

## Tipos de Archivos

Dentro de cada uno de los proyectos de desarrollo en Java incluidos en este libro, aparecen nueve tipos distintos de archivos, los cuales contienen partes de la solución. A continuación se describe cada uno de ellos:

|  |  |  |  |
| --- | --- | --- | --- |
| **Tipo de archivo** | **¿Qué contiene?** | **¿Cómo se usa?** | **¿Cómo se construye?** |
| .bat | Es un archivo de texto, que contiene una lista de comandos para el computador (para ser precisos, son comandos para el sistema operativo Microsoft Windows ®). Puede usarse para indicar la manera de compilar o ejecutar un programa. | Se puede usar de dos maneras: desde la consola de comandos del sistema operativo, o haciendo doble clic desde el explorador de archivos. | Desde cualquier editor de texto, como el bloc de notas (notepad). |
| .class | Es un archivo que contiene el código compilado de una clase Java. El compilador genera este archivo, que después podrá ser ejecutado. En el proyecto habrá un archivo .class por cada archivo .java. | Lo usa el computador para ejecutar un programa. | Se construye llamando el compilador del lenguaje, e indicándole el archivo .java que debe compilar. |
| .doc | Es un archivo que tiene parte de la especificación del problema (el enunciado general y los requerimientos funcionales). Tiene el formato usado por Microsoft Word®. | Se requiere tener instalado en el computador la aplicación Microsoft Word®. Para abrirlo basta con hacer doble clic en el archivo desde el explorador de archivos. | Se crea y modifica desde la aplicación  Microsoft Word®. |
| .html | Es un archivo con la documentación de una clase, generada automáticamente por la utilidad Javadoc. | Se requiere tener instalado en el computador un navegador de  Internet. Para abrirlo basta con hacer doble clic en el archivo desde el explorador de archivos. | Lo crea automáticamente la aplicación Javadoc, que extrae y organiza la documentación de una clase escrita en Java. |
|  | Es un archivo en el que están |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| .jar | empaquetados todos los archivos .class de un programa. Su objetivo es facilitar la instalación de un programa en el computador de un usuario. En lugar de tener que copiar cientos de archivos .class se empaquetan todos ellos en un solo .jar. | Lo usa el computador para ejecutar un programa. | Se construye utilizando la utilidad jar que viene con el compilador de Java. |
| .java | Es un archivo con la implementación de una clase en Java. | Se le pasa al compilador para que cree a partir de él un .class, que será posteriormente ejecutado por el computador. | Desde cualquier editor de texto. En nuestro caso, el ambiente de desarrollo Eclipse va a permitir editar este tipo de archivo, dándonos ayudas para detectar errores de sintaxis. |
| .mdl | Es un archivo con los diagramas de clases y de arquitectura del programa. Están escritos en el formato de Rational ROSE®. | Se requiere tener instalado en el computador la  aplicación Rational ROSE®. Para  abrirlo basta con hacer doble clic en el explorador de archivos. | Se crea, modifica e imprime desde la aplicación  Rational ROSE®. |
| .jpeg | Es un archivo con una imagen. Lo usamos para mostrar los distintos diagramas del programa. Esto permite visualizar el diseño a aquellos que no cuenten con el programa Rational ROSE®. | Cualquier programa  de imágenes  (incluso los navegadores de Internet) pueden leer estos archivos. | Se crean con cualquier editor de imágenes. |
| .zip | Es un archivo que empaqueta un conjunto de archivos. Tiene la ventaja de que los almacena de manera comprimida y hace que ocupen menos espacio. | Muchas  herramientas en el mercado permiten manejar este tipo de archivos. Si tiene alguna de ellas instalada en su computador, un doble clic desde el explorador de archivos iniciará la | Se construyen  utilizando las  mismas herramientas que permiten extraer  de allí los archivos que contienen. |
|  |  | aplicación. |  |

## Organización de los Elementos de Trabajo

Sigamos con el paralelo que estábamos haciendo con el edificio. Una vez terminados los planos debemos pasar a la etapa de construcción. Antes de empezar a abrir el hueco para los cimientos y de comprar los materiales que se necesitan, es necesario fijar todas las normas de organización. Lo primero es decidir dónde se va a poner cada elemento para la construcción: dónde van los ladrillos, dónde va el cemento, etc. Luego, cómo vamos a llamar las cosas. Si hay varios tipos de puertas, por ejemplo, nos debemos poner de acuerdo en la manera de etiquetarlas.

### Proyectos y Directorios

Un proyecto de desarrollo va siempre en un directorio, cuyo nombre indica su contenido. En nuestro caso el nombre del directorio comienza por el nivel, seguido del nombre del caso de estudio (por ejemplo, n1\_empleado).

Dentro del directorio principal, se encuentran siete directorios, con el contenido que se muestra en la figura 1.20.

|  |
| --- |
| **Fig. 1.20 Estructura de directorios dentro de un proyecto de desarrollo** |
|  |

Comencemos entonces a recorrer cada uno de estos directorios, utilizando para esto el proyecto de desarrollo del caso de estudio del empleado. En la tarea 11 se dan los pasos para poder comenzar este recorrido.

### Tarea 11

**Objetivo:** Preparar la organización para iniciar el recorrido por los elementos de un proyecto de desarrollo, utilizando como ejemplo el caso de estudio del empleado. Siga los pasos que se enuncian a continuación:

1. Descargue de aquí al disco de su computador el proyecto de nivel 1 llamado n1\_empleado. Descomprímalo (está en formato zip) y recorra los directorios internos utilizando el explorador de archivos.
2. Verifique que en su computador se encuentre instalado el compilador de Java. Si no está instalado, vaya al **anexo A** del libro y siga las instrucciones para instalarlo. Algunos programas del libro están escritos para versiones de Java posteriores a la versión 1.4.
3. Verifique que en su computador se encuentre instalado el ambiente de desarrollo Eclipse. Si no está instalado, vaya al **anexo B** del libro y siga las instrucciones para instalarlo.

### El Directorio bin

El directorio bin (por binary) contiene todos los archivos ejecutables del proyecto (archivos .bat), que permiten, entre otras cosas, ejecutar el programa, ejecutar las pruebas y generar la documentación del código. En este directorio hay siempre siete archivos, que incluyen las instrucciones para hacer las tareas que se describen a continuación: build.bat: permite compilar el programa (todos los .java) y generar el archivo empaquetado (.jar) que será instalado en el computador del usuario.

buildTest.bat: permite compilar las pruebas del programa y generar el respectivo archivo empaquetado (.jar).

clean.bat: borra todos los archivos del proyecto que pueden ser calculados a partir de otros. Esto es, elimina de los directorios todos los archivos .class, los archivos .jar generados y los archivos .html que corresponden a la documentación creada por Javadoc.

cleanTest.bat: borra todos los archivos de las pruebas que pueden ser calculados a partir de otros.

doc.bat: genera la documentación del programa (archivos .html), utilizando la herramienta Javadoc.

run.bat: ejecuta el programa. Este es el único archivo ejecutable que va al computador del usuario. Debe ser invocado después de haber ejecutado el archivo build, puesto que utiliza los archivos empaquetados para la ejecución.

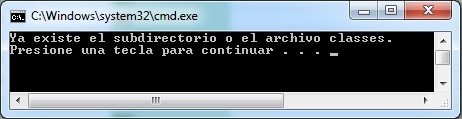
runTest.bat: ejecuta las pruebas del programa. Debe ser utilizado después de haber ejecutado el archivo buildTest, puesto que utiliza los archivos empaquetados para la ejecución.

### Tarea 12

**Objetivo:** Recorrer y utilizar los archivos que se encuentran en el directorio bin de un proyecto.

Siga los pasos que se dan a continuación:

1. Abra el explorador de archivos y sitúese en el directorio bin del proyecto que instaló en la tarea 11.
2. Seleccione cualquiera de los archivos y utilice la opción de editar. Esto debe iniciar el bloc de notas y debe permitirle leer su contenido. Por ahora no es importante que entienda lo que allí verá.
3. Ejecute el archivo build.bat (con un doble clic por ejemplo). Esto debe abrir una ventana como la que aparece a continuación, en la que el computador informa los resultados de la compilación del programa.



1. Ejecute el archivo run.bat. Esto debe ejecutar el programa que se construyó para el caso del empleado. Utilice la interfaz de usuario para darle los datos del empleado y usar las opciones de cálculo que propone. Pruebe que todo funcione correctamente.
2. Ejecute el archivo buildTest.bat. Esto debe compilar las pruebas del programa.
3. Ejecute el archivo runTest.bat y vea cómo se ejecutan las pruebas del programa, usando la plataforma JUnit. Por ahora no es importante entender lo que hacen las pruebas: imagine que son un programa que verifica que otro programa está bien escrito.
4. Ejecute el archivo doc.bat, para generar la documentación del programa. Si quiere ver las instrucciones que lo hacen, edite el archivo y mire cómo se ejecuta la herramienta Javadoc.

### El Directorio source

En este directorio encontrará los archivos fuente, en los que está la implementación en Java de cada una de las clases. Cada clase está en un archivo distinto, dentro de un directorio que refleja la jerarquía de paquetes. Esta relación entre paquetes y directorios es la que permite al compilador encontrar las clases en el espacio de trabajo. En la figura 1.21 se ilustra esta relación.

|  |
| --- |
| **Fig. 1.21 Relación entre los paquetes y la jerarquía de directorios** |
|  |

### Tarea 13

**Objetivo:** Recorrer los archivos fuente de un programa y ver la relación entre la jerarquía de directorios y la estructura de paquetes.

Siga los pasos que se dan a continuación.

1. Abra el explorador de archivos y sitúese en el directorio source del proyecto que instaló en la tarea 11.
2. Entre al directorio "uniandes". Dentro de éste entre al directorio "cupi2" y luego al directorio "empleado". Allí deben aparecer los directorios "interfaz" y "mundo". Entre en cualquiera de ellos y utilice el bloc de notas para ver el contenido de un archivo .java. Es importante decir que si se mueve un archivo a otro directorio, o se cambia el paquete al que pertenece sin desplazar físicamente el archivo al nuevo directorio, el programa no se va a compilar correctamente.

### El Directorio classes

En este directorio están todos los archivos .class. Tiene la misma jerarquía de directorios que se usa para los archivos fuente. No es muy interesante su contenido, porque para poder ver estos archivos por dentro se necesitan editores especiales. Si intenta abrir uno de estos archivos con editores de texto normales, va a obtener unos caracteres que aparentemente no tienen ningún sentido.

Estos archivos tienen por dentro el bytecode (código binario) producto de compilar la correspondiente clase Java.

### El Directorio test

En este directorio están todos los archivos que hacen las pruebas automáticas del programa. Por ahora lo único importante es saber que en su interior hay varios directorios, con archivos .class, .jar y .java. En un nivel posterior entraremos a mirar este directorio. Por ahora, con saber ejecutar las pruebas con el respectivo archivo .bat es suficiente.

### El Directorio docs

En este directorio hay dos subdirectorios:

specs: contiene todos los documentos de diseño. Allí encontrará: (1) el archivo Descripción.doc, con el enunciado del caso de estudio, (2) el archivo RequerimientosFuncionales.doc con la especificación de los requerimientos funcionales, (3) el archivo Modelo.mdl con los diagramas de clases del diseño y (4) un conjunto de archivos .jpg con las imágenes de los distintos diagramas de clases.

api: contiene los archivos de la documentación de las clases del programa. Estos archivos son generados automáticamente por la aplicación Javadoc. En la raíz de este directorio encontrará un archivo llamado "index.html". Al abrirlo podrá comenzar a navegar por toda la documentación del programa. Si no puede encontrar este archivo, ejecute el archivo doc.bat del directorio bin, que es el encargado de generar esta documentación. La figura 1.22 muestra un ejemplo de cómo se visualiza un archivo de documentación de una clase.

|  |
| --- |
| **Fig. 1.22 Ejemplo de la visualización de un archivo de documentación de una clase** |
|  |

### El Directorio lib

En este directorio encontrará el archivo empaquetado para instalar en el computador del usuario. En el caso de estudio del empleado dicho archivo se llama empleado.jar. Este archivo tiene la misma estructura interna de un archivo .zip, así que si desea ver su contenido puede utilizar cualquiera de los programas que permiten manejar esos archivos.

En su interior deberá encontrar todos los archivos .class del proyecto.

### El Directorio data

Este directorio contiene archivos con información que utiliza el programa, ya sea para almacenar datos (si tuviéramos una base de datos estaría en ese directorio) o para leerlos (por ejemplo, en el caso de estudio del empleado, allí se guarda la foto en un archivo con formato jpeg).

## Eclipse: Un Ambiente de Desarrollo

Un ambiente (o entorno) de desarrollo es una aplicación que facilita la construcción de programas. Principalmente, debe ayudarnos a escribir el código, a compilarlo y a ejecutarlo. Eclipse es un ambiente de múltiples usos, uno de los cuales es ayudar al desarrollo de programas escritos en Java. Es una herramienta de uso gratuito, muy flexible y adaptable a las necesidades y gustos de los programadores.

**Tarea 14**

**Objetivo:** Estudiar tres funcionalidades básicas del ambiente de desarrollo:

1. Cómo abrir un proyecto que ya existe (como el del caso de estudio).
2. Cómo leer y modificar los archivos de las clases Java.
3. Cómo ejecutar el programa.

Siga los pasos que se enuncian a continuación.

**1. ¿Cómo abrir en Eclipse el programa n1\_empleado? Puede hacerlo de dos formas:**

**Opción 1:** Creando el proyecto directamente en la estructura de directorios

Descomprima el archivo .zip que contiene el proyecto (por ejemplo en C:/temp/). Cree un proyecto Java en Eclipse (menú *File/New/Project*), con la ruta del directorio (C:/temp/n1\_empleado) y el nombre del proyecto (n1\_empleado).

Puede aceptar la creación ahora (botón "*Finish*"), o navegar a la siguiente ventana ("*Next*") para ver las propiedades del proyecto.

**Opción 2:** Importando el proyecto de la estructura de directorios:

Descomprima el archivo .zip que contiene el proyecto (por ejemplo en C:/temp/) Elija la opción de importación (menú *File/Import*…). En el diálogo en el que le preguntan la fuente de la importación seleccione "Existing Project into Workspace".

Seleccione la carpeta del proyecto (C:/temp/n1\_empleado) y finalice.

1. **¿Cómo explorar en Eclipse el contenido de un proyecto abierto?**

Utilice la vista llamada navegador. Si la vista no está disponible, búsquela en el menú *Window/Show* *View/ Navigator*.

Revise la estructura de directorios del proyecto n1\_empleado y recuerde el contenido de cada uno de ellos (puede ocurrir que algunos directorios no contengan archivos en el proyecto que está explorando).

1. **¿Cómo explorar en Eclipse un proyecto Java que esté abierto?**

Utilice la vista llamada "Package Explorer". Si la vista anterior no está disponible, búsquela en el menú *Window/ Show View/Package Explorer*.

Revise las propiedades del proyecto. Puede editar las propiedades haciendo clic derecho sobre el proyecto o mediante el menú *Project/Properties*.

Seleccione de la ventana de propiedades (de las opciones que aparecen a la izquierda) las opciones de construcción de Java ("*Java Build Path*") y revise la configuración del proyecto.

Observe la estructura de paquetes del proyecto.

1. **¿Cómo editar una clase Java?**

Utilizando la vista llamada "*Package Explorer*" localice el directorio con los archivos fuente del proyecto.

Dando doble clic sobre cualquiera de los archivos que allí se encuentran (Empleado.java, por ejemplo), el editor lo abre y permite al programador que lo modifique.

Agregue un comentario en algún punto de la clase Empleado, teniendo cuidado de no afectar el contenido del archivo, y sálvelo de nuevo con la opción del menú *File/Save*.

Cierre el archivo después de haberlo salvado.

1. **¿Cómo ejecutar el programa en un proyecto abierto en Eclipse?**

Utilizando la vista llamada "Package Explorer" localice el directorio con los archivos fuente del proyecto.

Localice la clase InterfazEmpleado en el paquete que contiene las clases de la interfaz. Cada programa en Java tiene una clase por la cual comienza la ejecución. Siempre se debe localizar esta clase para poder iniciar el programa.

Elija el comando "*Run/Java Application*". Puede hacerlo desde la barra de herramientas, el menú principal o el menú emergente que aparece al hacer clic derecho sobre la clase.

Con este comando el programa comienza su ejecución. El programa y Eclipse siguen funcionando simultáneamente. Para terminar el programa, basta con cerrar su ventana.

Localice la vista llamada consola. Si la vista no está disponible, búsquela en el menú *Window/Show View/Console*. Allí pueden aparecer algunos mensajes de error de ejecución. En esa vista hay un botón rojo pequeño, que permite terminar la ejecución del programa.

Debe estar claro que el ambiente de desarrollo es una herramienta para el programador, y que lo normal es que dicho ambiente no esté instalado en el computador del usuario.

# Hojas de Trabajo

## Hoja de Trabajo Nº 1: Una Encuesta

Descargue esta hoja de trabajo a través de los siguientes enlaces: [Descargar PDF](https://bit.ly/apo1-nivel1-ht1-pdf-format) | [Descargar Word](https://bit.ly/apo1-nivel1-ht1-word-format).

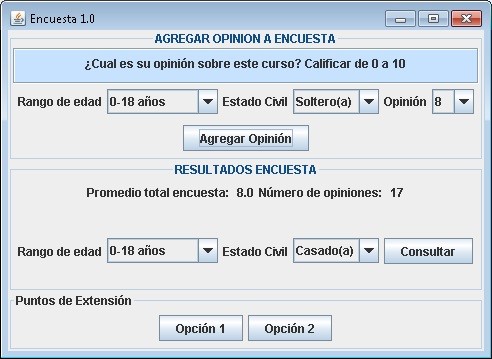
**Enunciado.** Analice la siguiente lectura e identique el mundo del problema, lo que se espera de la aplicación y las restricciones para desarrollarla.

Se quiere construir un programa para manejar los resultados de una encuesta de opinión.

En la encuesta se dividieron las personas en 3 rangos de edad: (1) menos de 18, (2) entre 18 y 55 y (3) más de 55 años. La encuesta consiste en una única pregunta, en la cual se le pide a la persona que calique la calidad de un curso dando un valor entre 0 y 10. En el momento de hacer la pregunta, la persona debe informar si es soltera o casada. El programa debe permitir agregar una nueva opinión a la encuesta. Esto es, debe permitir que se añada una nueva persona en un rango de edad (por ejemplo en el rango 2), que da una calicación al curso (por ejemplo 4) y que dice si es casada o soltera.

El programa debe informar el valor total de la encuesta. Esto es, debe promediar todas las notas dadas y presentar el resultado en pantalla. También debe debe ser capaz de informar valores parciales de la encuesta. En ese caso se debe especificar un rango de edad y un estado civil. El programa presenta por pantalla el promedio de las calicaciones del curso dadas por todas las personas que cumplen el perl pedido. Puede suponer que en el momento de calcular los resultados hay por lo menos una persona de cada perl.

La interfaz de usuario de este programa es la que se muestra a continuación:



**Requerimientos funcionales.** Describa tres requerimientos funcionales de la aplicación que haya identicado en el enunciado.

### Requerimiento Funcional 1

|  |  |
| --- | --- |
| **Nombre** | RF1:  Recibir rango de edad |
| Resumen | Los usuarios podrán seleccionar el rango en el cual esta la edad de estos mismos |
| Entradas | Rango de edad de la persona (0-18 años/ 19-29 años…) |
| Resultado | Rango de Edad: Rango seleccionado |

### Requerimiento Funcional 2

|  |  |
| --- | --- |
| **Nombre** | RF2:  Recibir estado civil |
| Resumen | La persona podrá escoger entre dos opciones cual es su estado civil: Soltero(a) o casado(a) |
| Entradas | Seleccione su estado civil: -Soltero -Casado |
| Resultado | Estado Civil: Estado Seleccionado |

### Requerimiento Funcional 3

|  |  |
| --- | --- |
| **Nombre** | RF3:  Escoger una calificación para el curso |
| Resumen | Se podrá escoger un valor entre 0 y 10 para calificar el curso |
| Entradas | Opinión: (0 – 10) |
| Resultado | Opinión seleccionada: Valor Opinión |

**Entidades del mundo.** Identifique las entidades del mundo y descríbalas brevemente.

|  |  |
| --- | --- |
| **Entidad** | **Descripción** |
| Rango de edad | El rango en el que esta la edad de la persona y/o usuario que está llenando la encuesta |
| Estado civil | Es este campo se recibirá el estado civil del usuario, ya sea soltero o casado. |

**Características de las entidades.** Identifique las características de cada una de las entidades y escriba la clase en UML con el tipo de datos adecuado.

### Entidad 1

|  |  |
| --- | --- |
| **Atributo** | **Valores Posibles** |
| Edad | Valores enteros positivos |
| Estado civil | Cadena de caracteres |
| **Diagrama UML** | |
| |  | | --- | | Encuesta | | + Edad  +Estado civil | |  | | |

### Entidad 2

|  |  |
| --- | --- |
| **Atributo** | **Valores Posibles** |
| Valor de la calificación | Valores enteros |
| **Diagrama UML** | |
| |  | | --- | | Porcentaje\_Calificacion | | + Valor de la calificación | |  | | |

**Relaciones entre entidades.** Dibuje las entidades en UML (sin atributos ni métodos) y las relaciones que existan entre ellas.

|  |
| --- |
| Porcentaje Calificación |
| + Valor de la calificación |
|  |

|  |
| --- |
| Encuesta |
| + Edad  +Estado civil |
|  |

**Métodos de las entidades.** Lea las siguientes descripciones de métodos y escriba su implementación en el lenguaje Java.

### Método 1

|  |  |
| --- | --- |
| **Clase** | **RangoEncuesta** |
| Nombre | darNumeroCasados |
| Parámetros | Ninguno. |
| Retorno | El número de personas casadas que respondieron la encuesta, en el rango de edad de la clase. |
| Descripción | Retorna el número de personas casadas que respondieron la encuesta, en el rango de edad de la clase. |
| **Implementación en Java** | |
|  | |

### Método 2

|  |  |
| --- | --- |
| **Clase** | **RangoEncuesta** |
| Nombre | darTotalOpinionCasados |
| Parámetros | Ninguno. |
| Retorno | La suma de todas las opiniones de los encuestados casados en el rango de edad de la clase. |
| Descripción | Retorna la suma de todas las opiniones de los encuestados casados en el rango de edad de la clase. |

|  |  |
| --- | --- |
| **Clase** | **RangoEncuesta** |
| Nombre | darPromedio |
| Parámetros | Ninguno. |
| Retorno | El promedio de la encuesta en el rango de edad de la clase. |
| Descripción | Retorna el promedio de la encuesta en el rango de edad de la clase. Para esto suma todas las opiniones y divide por el número total de encuestados. |
| **Implementación en Java** | |
|  | |

### Método 4

|  |  |
| --- | --- |
| **Clase** | **RangoEncuesta** |
| Nombre | agregarOpinionCasado |
| Parámetros | Opinión del encuestado. |
| Retorno | Ninguno. |
| Descripción | Añade la opinión de una persona casada en el rango de edad que representa la clase. |
| **Implementación en Java** | |
|  | |

### Método 5

|  |  |
| --- | --- |
| **Clase** | **RangoEncuesta** |
| Nombre | darPromedioCasados |
| Parámetros | Ninguno. |
| Retorno | El promedio de la encuesta en el rango de edad de la clase considerando sólo los casados. |
| Descripción | Retorna el promedio de la encuesta en el rango de edad de la clase. Para esto suma todas las opiniones de los casados y divide por el número total de ellos. |

**Método 6**

|  |  |
| --- | --- |
| **Clase** | **Encuesta** |
| Nombre | agregarOpinionRango1Casado |
| Parámetros | Opinión del encuestado. |
| Retorno | Ninguno. |
| Descripción | Añade la opinión de una persona casada en el rango de edad 1 de la encuesta. |
| **Implementación en Java** | |
|  | |

### Método 7

|  |  |
| --- | --- |
| **Clase** | **Encuesta** |
| Nombre | agregarOpinionRango2Soltero |
| Parámetros | (1) estado civil, (2) opinión. |
| Retorno | Ninguno. |
| Descripción | Añade la opinión de una persona soltera en el rango de edad 2 de la encuesta. |
| **Implementación en Java** | |
|  | |

### Método 8

|  |  |
| --- | --- |
| **Clase** | **Encuesta** |
| Nombre | darPromedio |
| Parámetros | Ninguno. |
| Retorno | El promedio de la encuesta en todos los rangos de edad. |
| Descripción | Retorna el promedio de la encuesta en todos los rangos de edad. Para esto suma todas las opiniones y divide por el número total de encuestados. |



### Método 9

|  |  |
| --- | --- |
| **Clase** | **Encuesta** |
| Nombre | darPromedioCasados |
| Parámetros | Ninguno. |
| Retorno | El promedio de la encuesta en todos los rangos de edad de la clase, considerando sólo los casados. |
| Descripción | Retorna el promedio de la encuesta en todos los rangos de edad. Para esto suma todas las opiniones de los casados y divide por el número total de ellos. |
| **Implementación en Java** | |
|  | |

## 9.2 Hoja de Trabajo Nº 2: Una Alcancía

Descargue esta hoja de trabajo a través de los siguientes enlaces: Descargar PDF | Descargar Word.

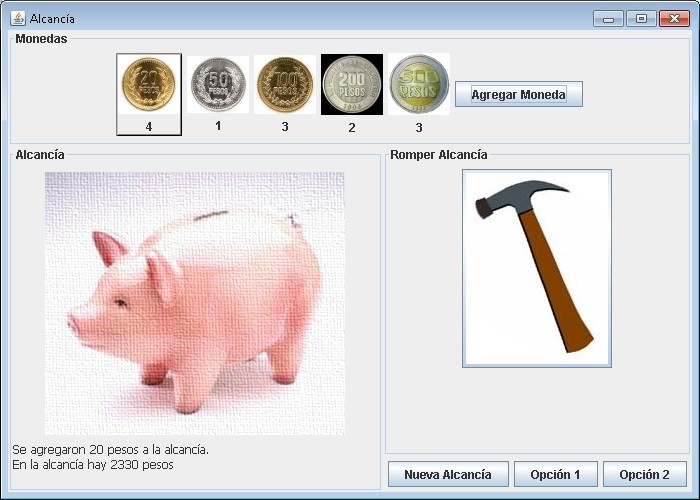
**Enunciado:** Analice la siguiente lectura e identique el mundo del problema, lo que se espera de la aplicación y las restricciones para desarrollarla.

Se quiere construir un programa para manejar una alcancía. En la alcancía es posible guardar monedas de distintas denominaciones: $20, $50, $100, $200 y $500. No se guardan billetes o monedas de otros valores.

El programa debe dar las siguientes opciones: (1) agregar una moneda de una de las denominaciones que maneja, (2) informar cuántas monedas tiene de cada denominación,

(3) calcular el total de dinero ahorrado y (4) romper la alcancía, vaciando su contenido.

La interfaz de usuario de este programa es la que se muestra a continuación:



**Requerimientos funcionales.** Describa tres requerimientos funcionales de la aplicación que haya identicado en el enunciado.

### Requerimiento Funcional 1

|  |  |
| --- | --- |
| **Nombre** | **R1 – Guardar una moneda de $20 en la alcancía.** |
| [Resumen](https://bit.ly/apo1-nivel1-ht2-word-format) |  |
| Entradas |  |
| Resultado |  |

### Requerimiento Funcional 2

|  |  |
| --- | --- |
| **Nombre** | **R2 – Contar el número de monedas de $50 que hay en la alcancía.** |
| Resumen |  |
| Entradas |  |
| Resultado |  |

### Requerimiento Funcional 3

|  |  |
| --- | --- |
| **Nombre** | **R3 – Calcular el total de dinero ahorrado en la alcancía.** |
| Resumen |  |
| Entradas |  |
| Resultado |  |

### Requerimiento Funcional 4

|  |  |
| --- | --- |
| **Nombre** | **R4 – Romper la alcancía.** |
| Resumen |  |
| Entradas |  |
| Resultado |  |

**Entidades del mundo.** Identique las entidades del mundo y descríbalas brevemente.

|  |  |
| --- | --- |
| **Entidad** | **Descripción** |
|  |  |

**Características de las entidades.** Identique las características de cada una de las entidades y escriba la clase en UML con el tipo de datos adecuado.

### Entidad 1

|  |  |
| --- | --- |
| **Atributo** | **Valores Posibles** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| **Diagrama UML** | |
|  | |

**Métodos de las entidades.** Complete las siguientes descripciones de métodos y escriba su implementación en el lenguaje Java.

### Método 1

|  |  |
| --- | --- |
| **Clase** | **Alcancia** |
| Nombre | AgregarMoneda20 |
| Parámetros |  |
| Retorno |  |
| Descripción |  |
| **Implementación en Java** | |
|  | |

### Método 2

|  |  |
| --- | --- |
| **Clase** | **Alcancia** |
| Nombre | AgregarMoneda500 |
| Parámetros |  |
| Retorno |  |
| Descripción |  |
| **Implementación en Java** | |
|  | |

### Método 3

|  |  |
| --- | --- |
| **Clase** | **Alcancia** |
| Nombre | darTotalDinero |
| Parámetros |  |
| Retorno |  |
| Descripción |  |
| **Implementación en Java** | |
|  | |

### Método 4

|  |  |
| --- | --- |
| **Clase** | **Alcancia** |
| Nombre | darNumeroMonedas100 |
| Parámetros |  |
| Retorno |  |
| Descripción |  |
| **Implementación en Java** | |
|  | |

### Método 5

|  |  |
| --- | --- |
| **Clase** | **Alcancia** |
| Nombre | romperAlcancia |
| Parámetros |  |
| Retorno |  |
| Descripción |  |
| **Implementación en Java** | |
|  | |