

CAPÍTULO II: PROGRAMACIÓN ORIENTADA AL OBJETO CON JAVA

2.1. Aproximación a la orientación al objeto

Se trata de resolver problemas, identificando los **objetos** asociados con el problema y convirtiéndolos en objetos del programa.

Por ejemplo, son objetos:

José García

Compañía ABC S.A.

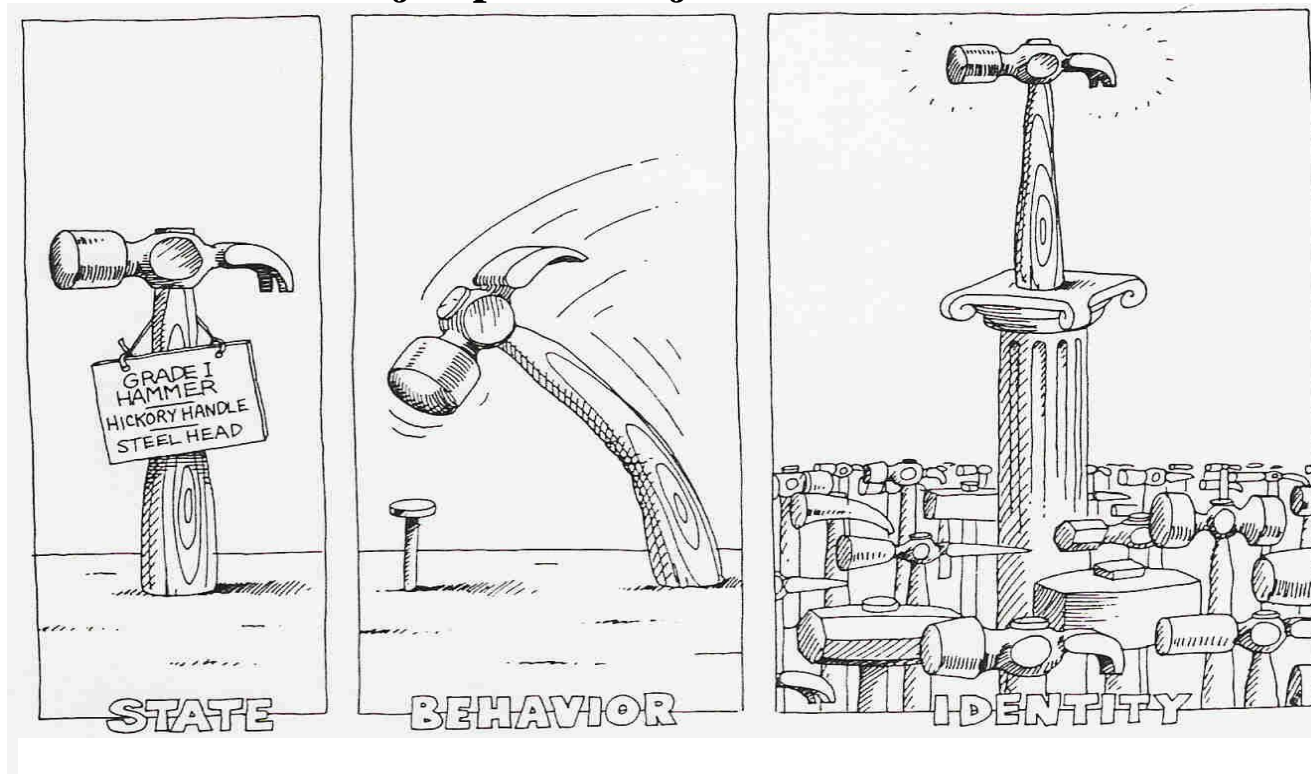
Lassie

Proceso Número 7468

Un objeto es, sencillamente, algo que tiene sentido en el contexto de la aplicación.

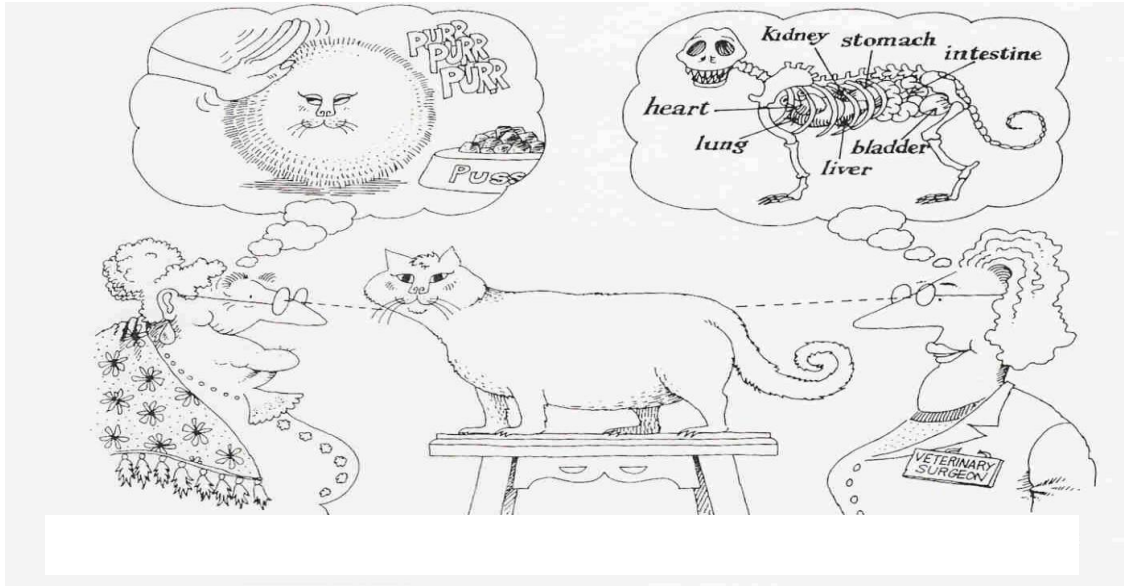
Un objeto tiene estado, exhibe una conducta bien definida y tiene una identidad única.

Ejemplo del objeto martillo

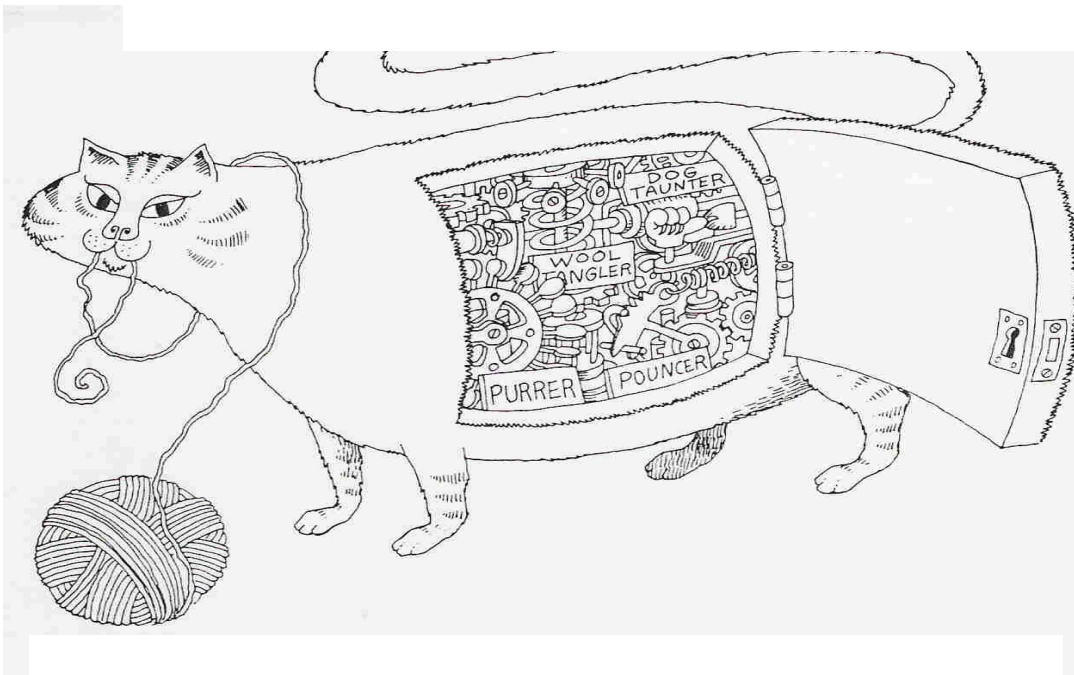


Conceptos asociados en la orientación al objeto

- **Abstracción:** Se enfoca en las características esenciales de algún objeto, relativa a la perspectiva de quien mira al objeto.



- **Encapsulación:** Oculta los detalles de la implementación de un objeto.



Para la encapsulación se deben respetar los siguientes puntos:

- La abstracción de un objeto debería preceder a las decisiones sobre su implementación.
- Una vez que se ha seleccionado la implementación, debe tratarse como un secreto de la abstracción, oculto para la mayoría de los clientes.
- Ninguna parte de un sistema complejo debe depender de los detalles internos de otras partes.

El elemento de la abstracción es la **clase**.

- **Modularidad:**

Permite subdividir una aplicación en partes más pequeñas (llamados módulos), cada una de las cuales debe ser tan independientes como sea posible de la aplicación en sí y de las restantes partes.

La modularización consiste en dividir un programa en módulos que se puedan compilar por separado, pero que tienen conexiones con otros módulos.

La modularidad es la propiedad de un sistema que permite su descomposición en un conjunto de módulos cohesivos y débilmente acoplados.

La modularidad empaqueta las abstracciones en unidades discretas.

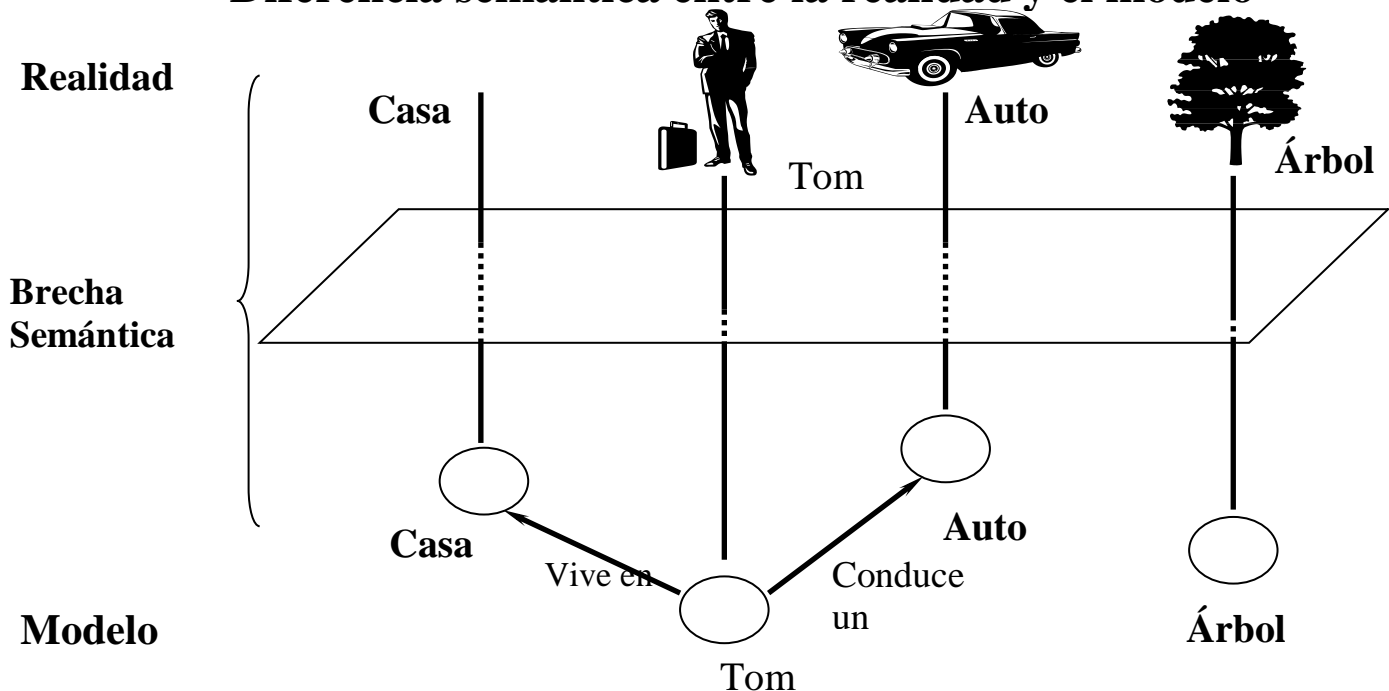
En el caso de Java se hace a través de los paquetes.



- **Jerarquía**
 - **Polimorfismo**
- Conceptos que se verán más adelante

Una ventaja de utilizar la orientación al objeto es que la brecha semántica se minimiza, ya que los objetos de la realidad son directamente mapeados en objetos en el modelo.

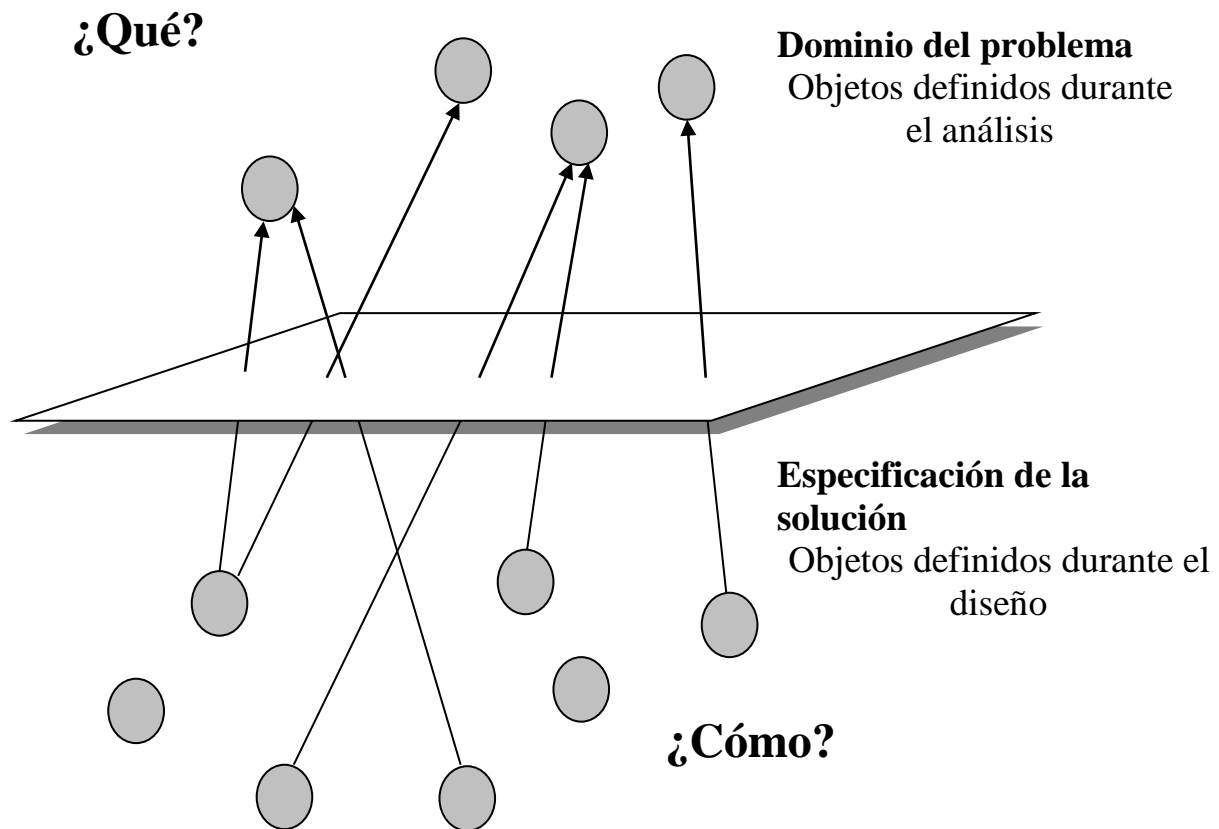
Diferencia semántica entre la realidad y el modelo



En la etapa de análisis aparecen los objetos del dominio del problema, es decir, los objetos del mundo real.

Cuando se especifica la solución, aparecen nuevos objetos, desde el punto de vista de la solución del problema.

Capas de especificaciones de objetos resultantes del proceso de refinamiento

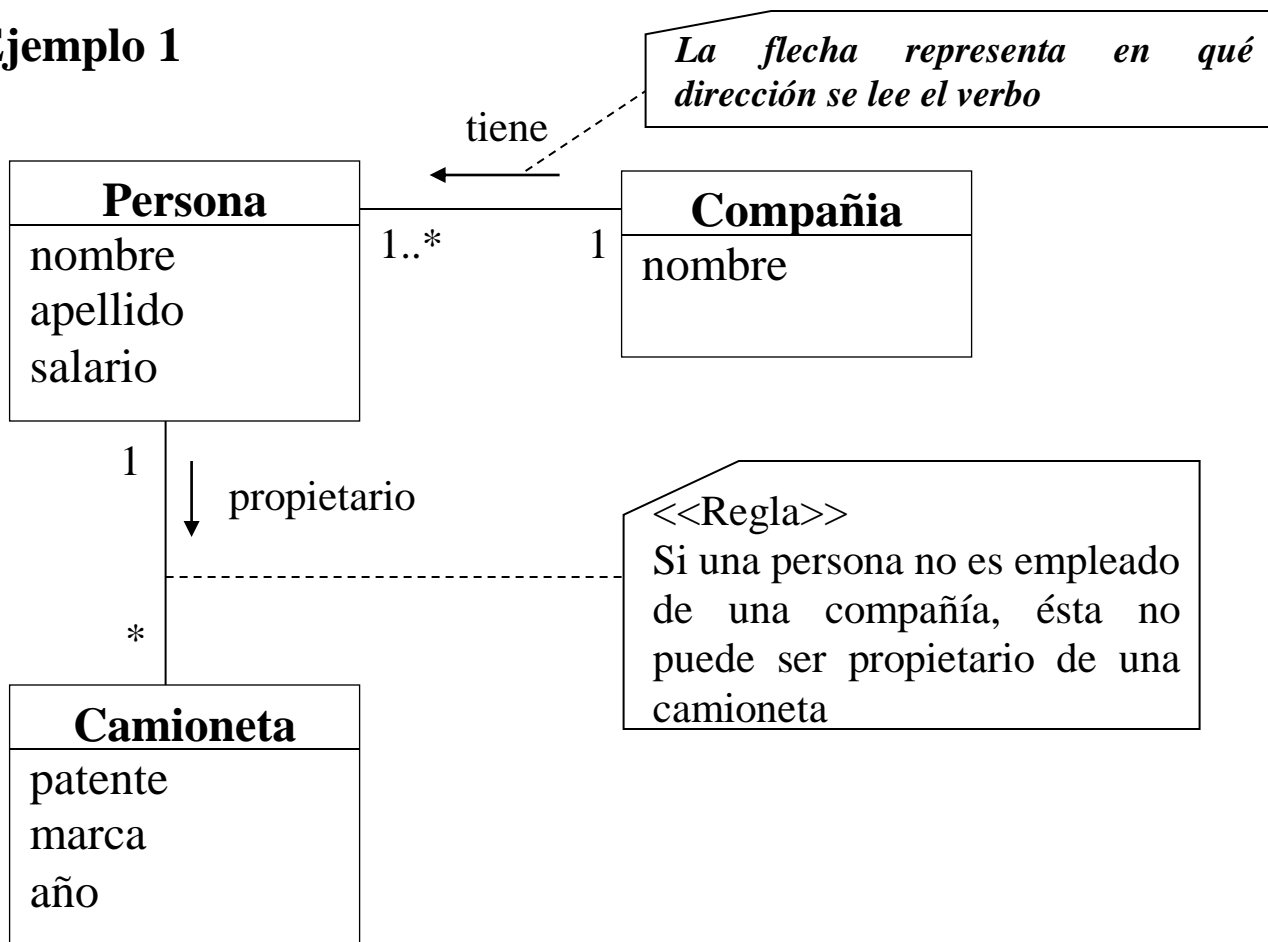


2.2. Modelo del dominio

- “Un modelo de dominio es una representación de conceptos en un dominio del problema” [MO95, Fowler96]
- “El modelo de dominio representa las “cosas” que existen o eventos que transcurren en el entorno de un negocio.” [I. Jacobsen]

Ejemplos de modelo del dominio

Ejemplo 1



Relacionamientos entre conceptos

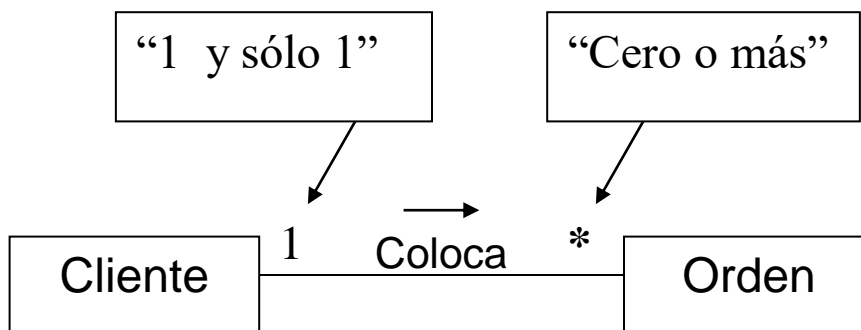
Concepto de relacionamiento

El relacionamiento entre conceptos es una asociación natural en el contexto del problema

Ejemplo

Para un problema dado, se pueden efectuar las siguientes aseveraciones:

- Un cliente **coloca** cero o más ordenes de compra.
- Una orden de compra es **colocada por** uno y sólo un cliente.



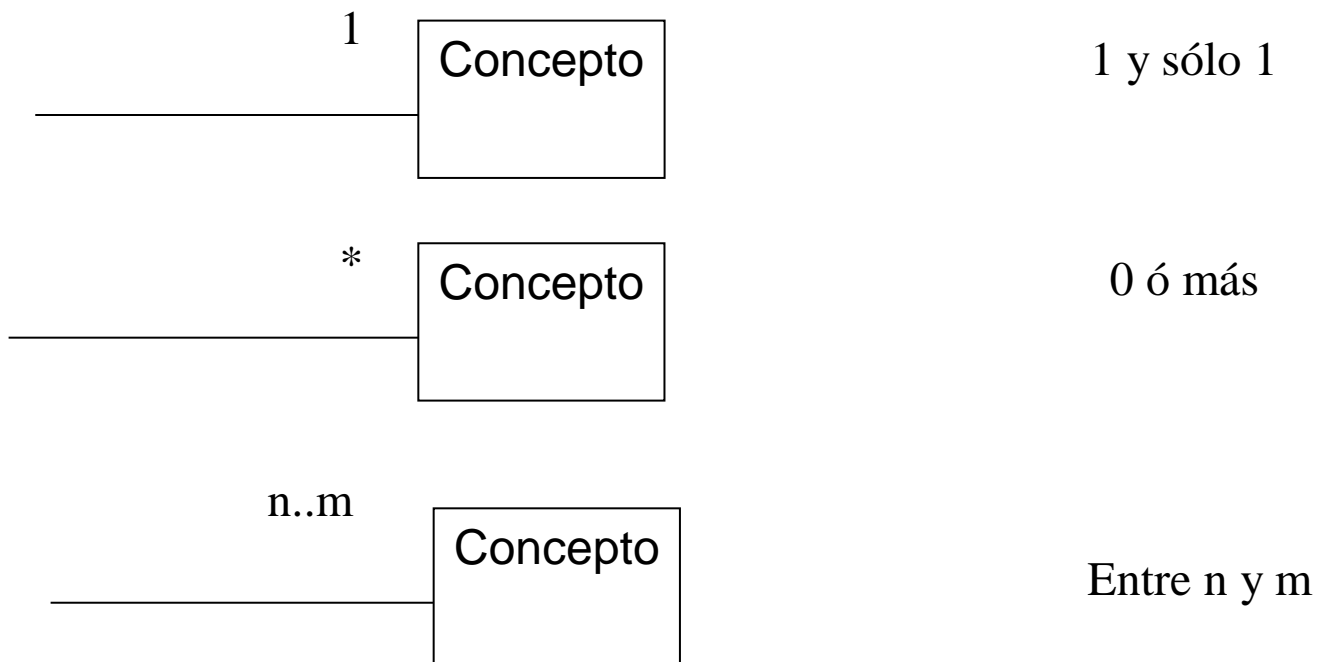
- La línea entre las clases muestra el relacionamiento.
- El verbo describe el relacionamiento.
- Notar que todos los relacionamientos son bidireccionales, en forma implícita, indicando con ello que pueden ser interpretados en ambas direcciones.

Multiplicidad del relacionamiento

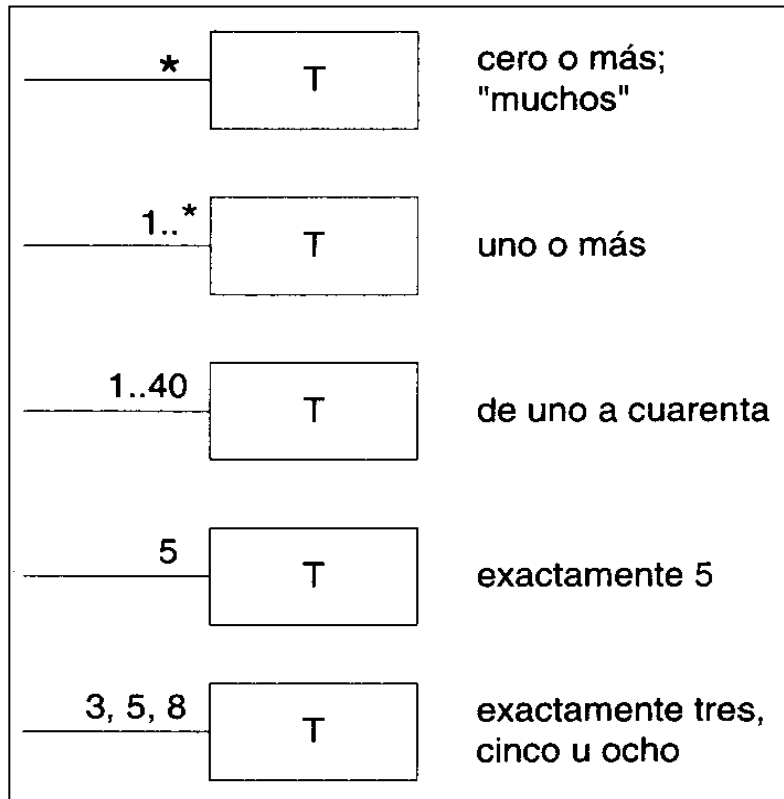
La multiplicidad define el mínimo y el máximo número de ocurrencias de un concepto asociados a una única ocurrencia del concepto relacionado.

Debido a que todos los relacionamientos son bidireccionales, la multiplicidad debe ser definida en ambas direcciones, para cada relacionamiento.

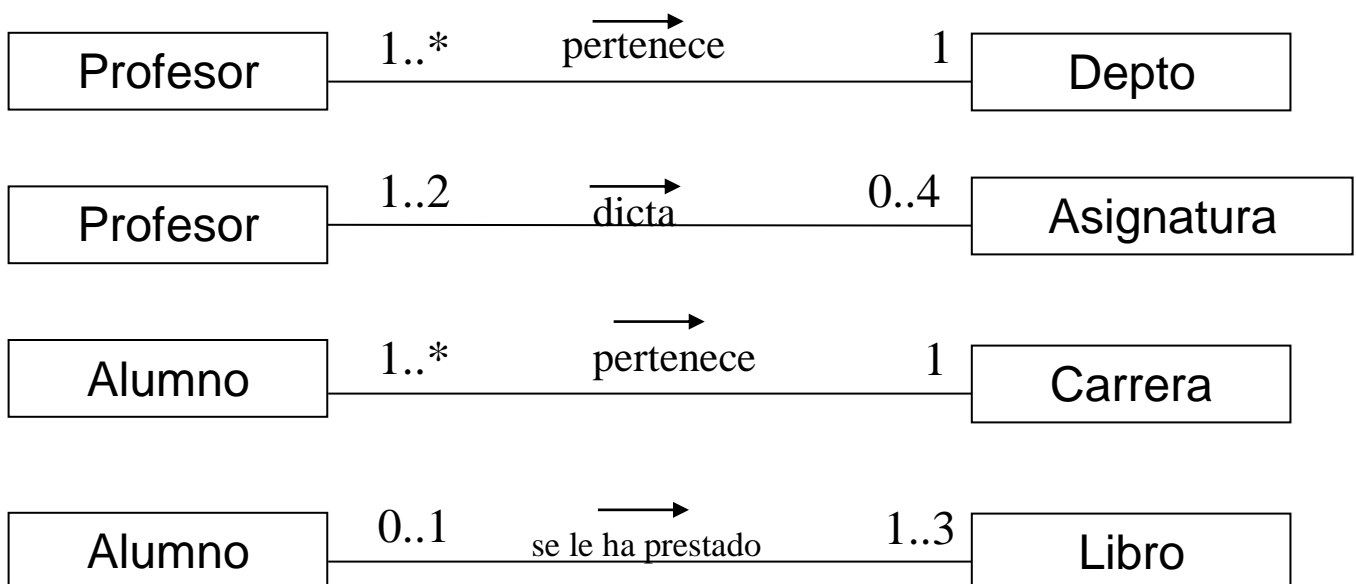
Tipos de multiplicidad



La multiplicidad define cuántas instancias de un concepto A pueden asociarse a un concepto B.



Ejemplos



Características del modelo del dominio

- ☐ **Concepto:** Denota un tipo de objeto.
- ☐ **Atributos:** Describen un conjunto de valores asociados a un concepto.
- ☐ **Asociaciones:** Representa cómo dos o más conceptos se encuentran vinculados.
- ☐ **Reglas adicionales:** Se describen mediante notas las reglas que no pueden ser representadas por la simbología.

Concepto. Los tipos de conceptos pueden ser:

- ☐ Objetos de negocio: Representa cosas que son manipuladas en un negocio. (Ej: Boleta)
- ☐ Objetos del mundo real: Cosas que la empresa realiza un seguimiento. (Personas, Sitios)
- ☐ Eventos que transcurren: Venta y pagos.

Recomendaciones para el Modelo del Dominio

- ☐ Listar los conceptos idóneos
- ☐ Dibujar los conceptos en un modelo de dominio
- ☐ Incorporar las asociaciones necesarias
- ☐ Agregar atributos necesarios
- ☐ ¿Cómo identificar Conceptos?
 - ☐ Sustantivos y frases en descripción textual del dominio
- ☐ ¿Cómo identificar Atributos?
 - ☐ Concepto que puede ser representado por un número o un texto.

Caso de Estudio: Modelo del dominio

- ☐ Desarrollar un modelo de dominio que permita modelar el requerimiento de la Farmacia “Dr Pepito”.
- ☐ La Farmacia “Dr Pepito” requiere de un software que le permita administrar venta de remedios.
- ☐ El software debe considerar los siguientes aspectos:
 - ☐ Algunos medicamentos sólo pueden ser vendidos bajo receta medica, o en su defecto, con la aprobación del farmacéutico.
 - ☐ Los vendedores deberán registrar los remedios que se compran en un TPV, en donde se ingresará el código del remedio, cantidad y % de descuento.
 - ☐ Al finalizar la venta, los clientes podrán cancelar los remedios en efectivo, tarjeta o cheque al día.
 - ☐ Al terminar el pago, el vendedor le entrega los productos al cliente junto con el comprobante de pago.

Conceptos

Farmaceutico

Farmacia

Vendedor

TPV

Remedio

Venta

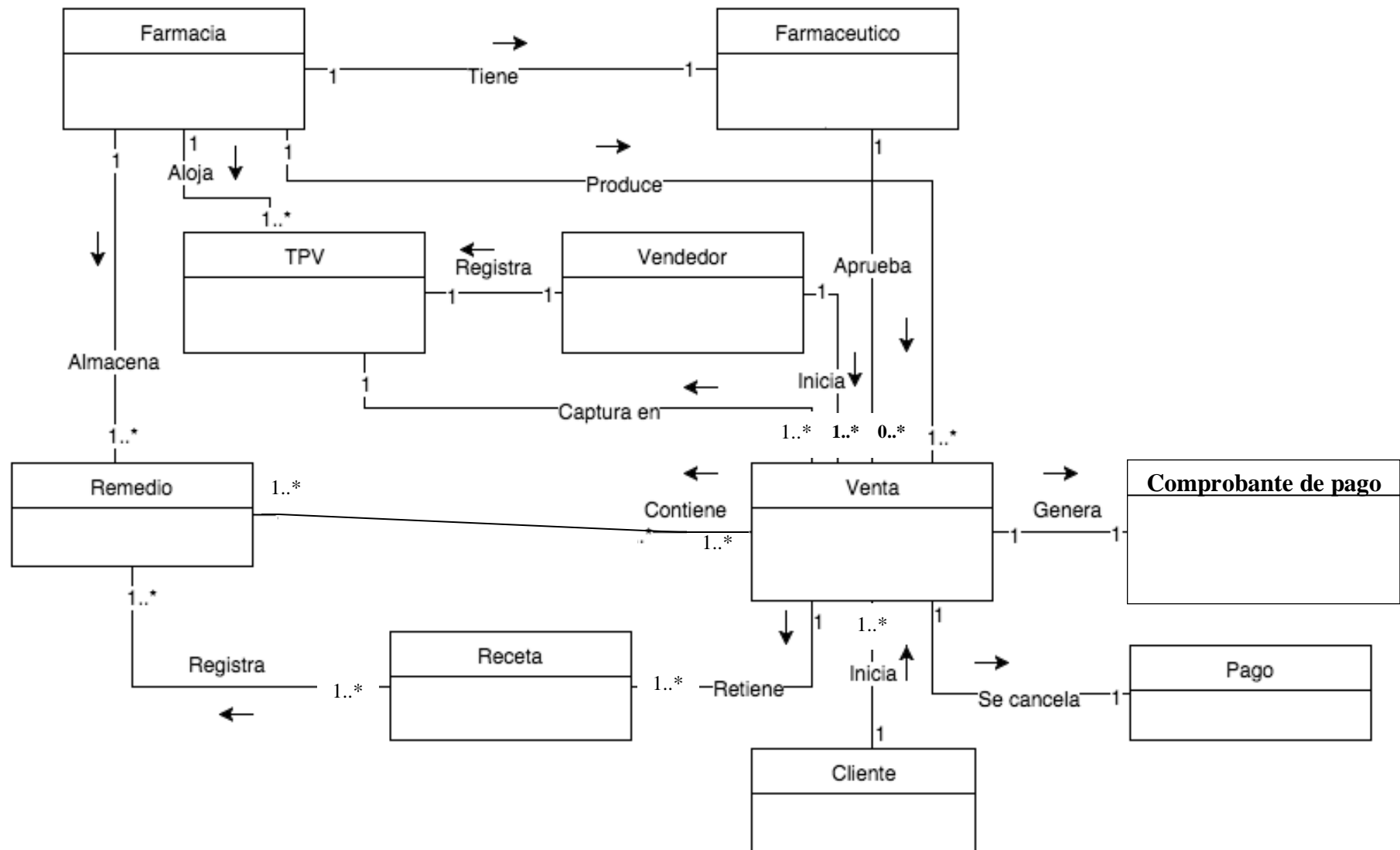
Cliente

Receta

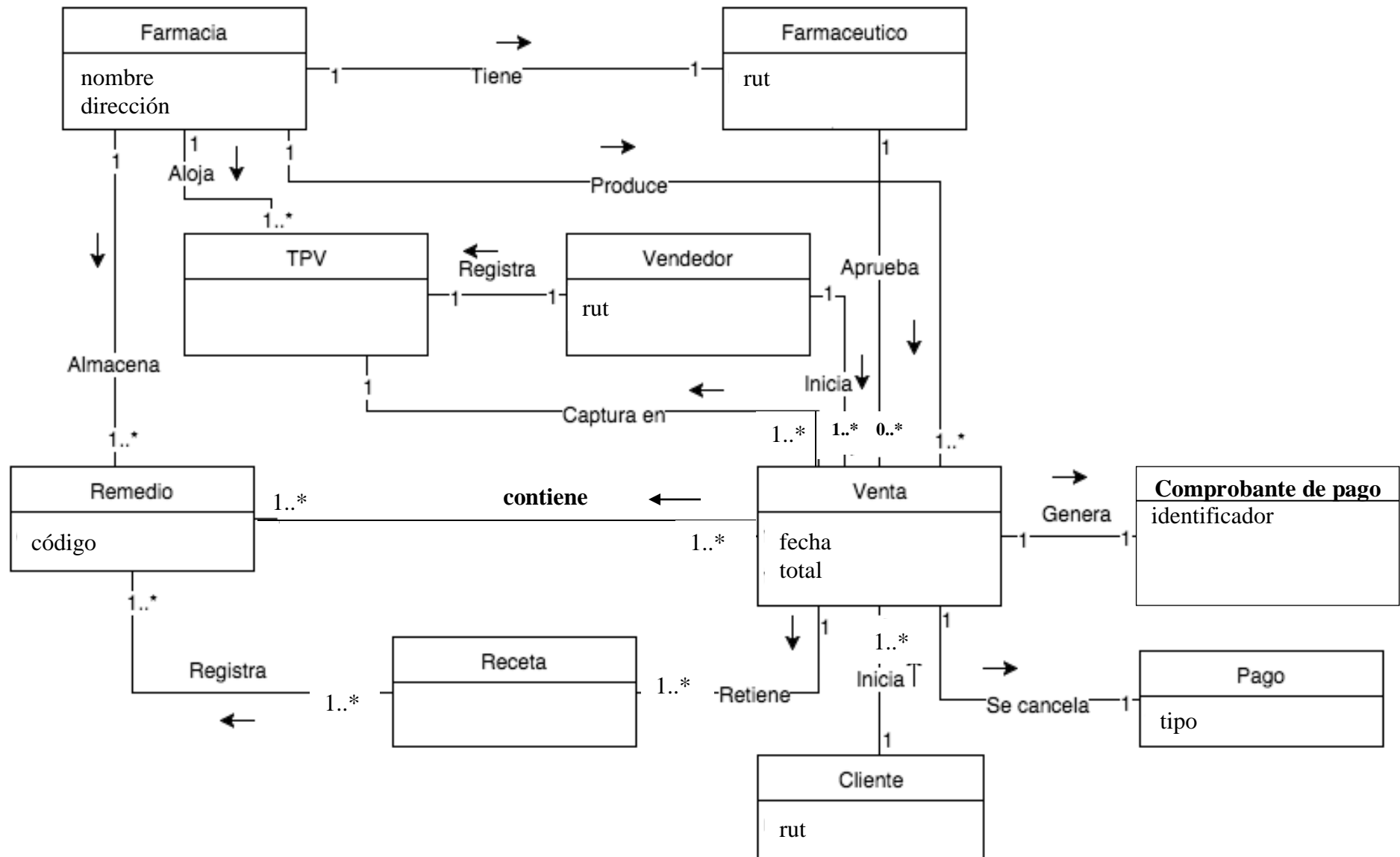
Pago

Comprobante de pago

Asociaciones



Atributos



- ☐ El Modelo del Dominio se obtiene identificando:

- 1. Entidades**

- a. Nombre
 - b. Propiedades o características

- 2. Relaciones**

- a. Nombre
 - b. Dirección
 - c. Cardinalidad

- 3. Restricciones**

- ☐ El objetivo es que lo entienda el cliente
- ☐ A diferencia, el Diagrama de Clases es para el diseñador del software
- ☐ No se necesita colocar el tipo de los atributos. El cliente no sabe de eso

Problema Tutores

En los últimos años la Universidad Católica del Norte se ha preocupado por el rendimiento de sus alumnos y ha puesto en marcha el plan piloto “Tutores” en algunas carreras específicas. Como el número de carreras ha sido pequeño, toda la información del plan ha sido realizada a mano hasta el momento. El plan ha dado como resultado una mejora superior a lo esperado, es por ello, que la UCN ha decidido expandir el plan al resto de los alumnos.

Modelo del Dominio

UCN

Carrera

Asignatura

Tutor

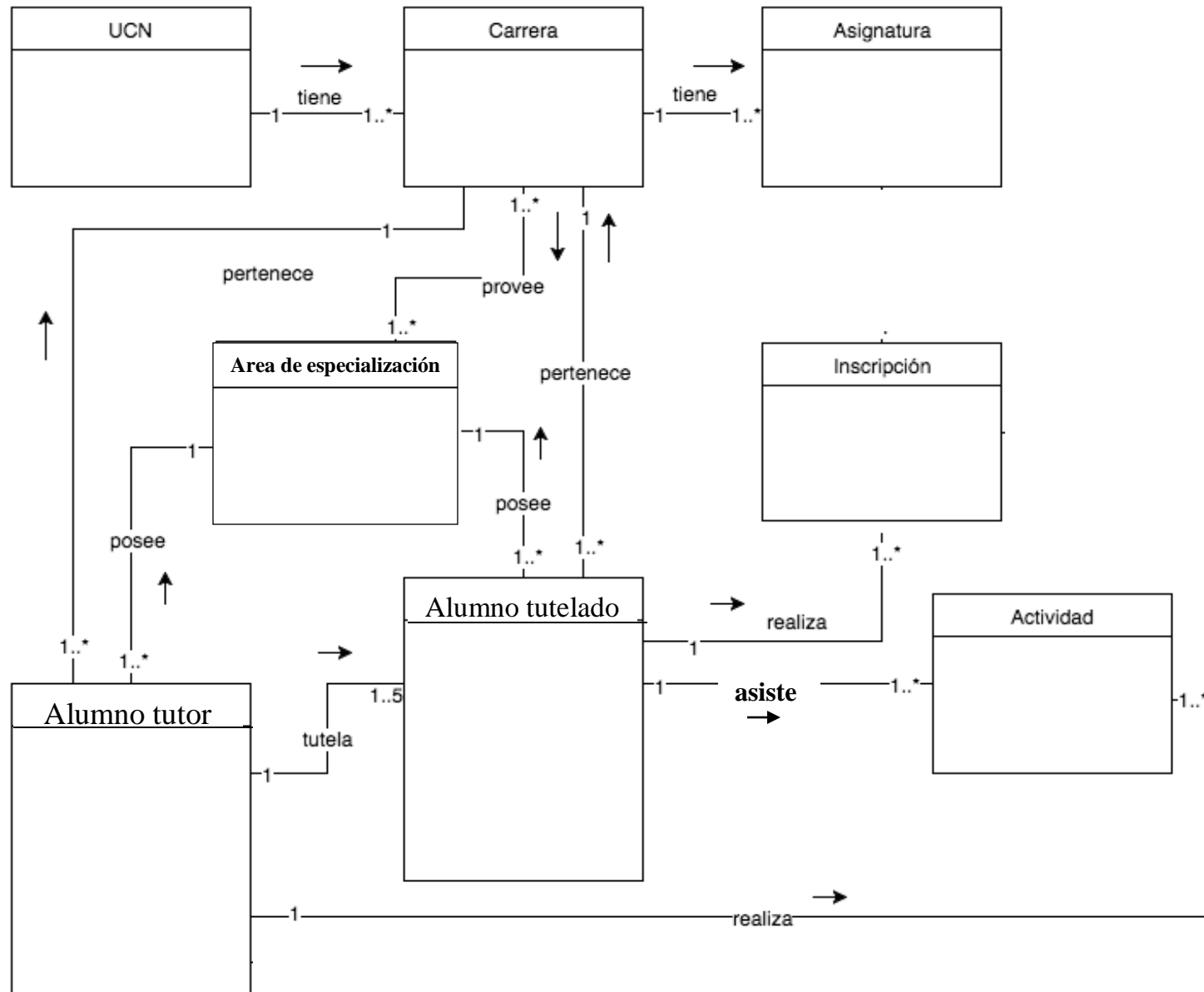
Tutelado

Inscripción

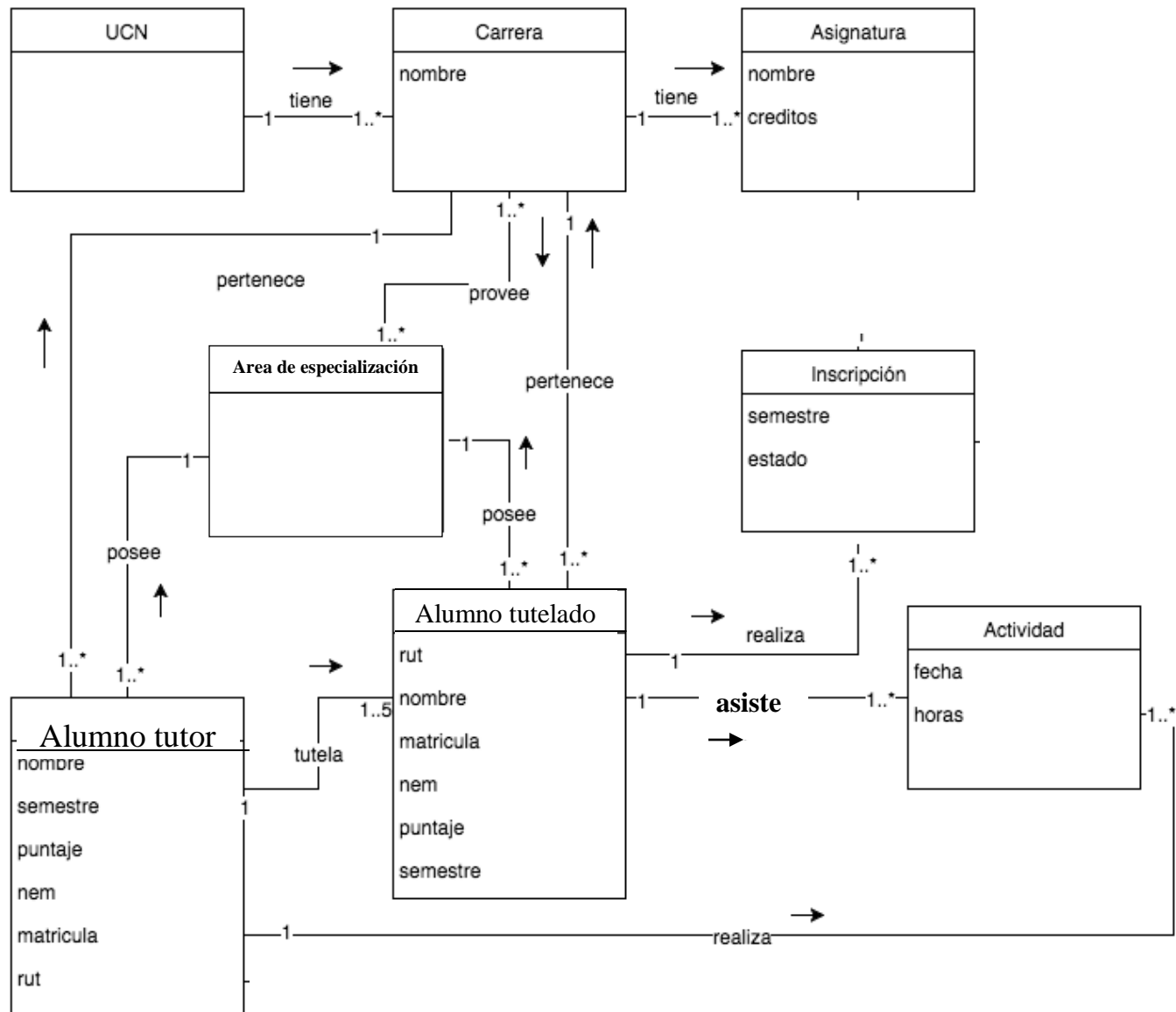
Area de especialización

Actividad

Modelo del Dominio



Modelo del Dominio



Modelo del Dominio

¿Tiene restricciones?, ¿Cuáles?

Restricciones Tutores

- ☐ Nombre del alumno no puede ser inferior a 2 letras.
- ☐ Rut del alumno debe ser válido.
- ☐ Nota de enseñanza media debe estar en el intervalo 1.0 – 7.0. Se aceptan las notas con un dígito decimal.
- ☐ Puntaje de ingreso debe ser un valor entero entre 500 y 900 puntos.
- ☐ Semestre que cursa debe ser un valor entre 1 y 14.
- ☐ Un alumno sólo puede ser tutor si ha cursado a lo menos dos semestres de su carrera y posee menos de cinco asignaturas inscritas en el semestre.
- ☐ ...

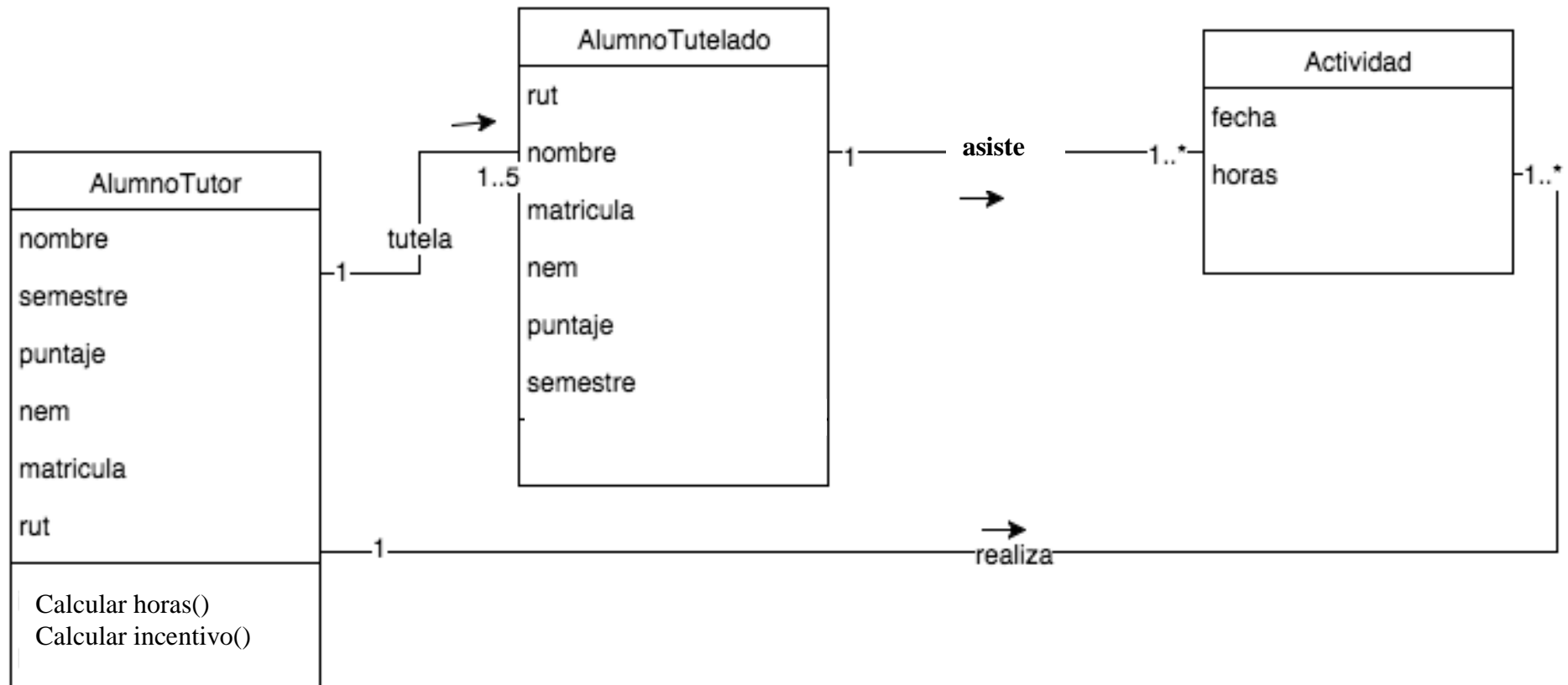
Operaciones en el modelo del dominio

En el modelo del dominio se pueden colocar **operaciones** cuando éstas son parte del modelo del negocio y son calculables. A este nivel, en el análisis se está preocupado de los conceptos, atributos y relacionamientos y normalmente no estamos preocupados de identificar las operaciones

Operaciones Tutores

- ☐ Obtener la cantidad de horas en que el tutor realizó la tutela.
- ☐ Calcular el incentivo que le corresponde.

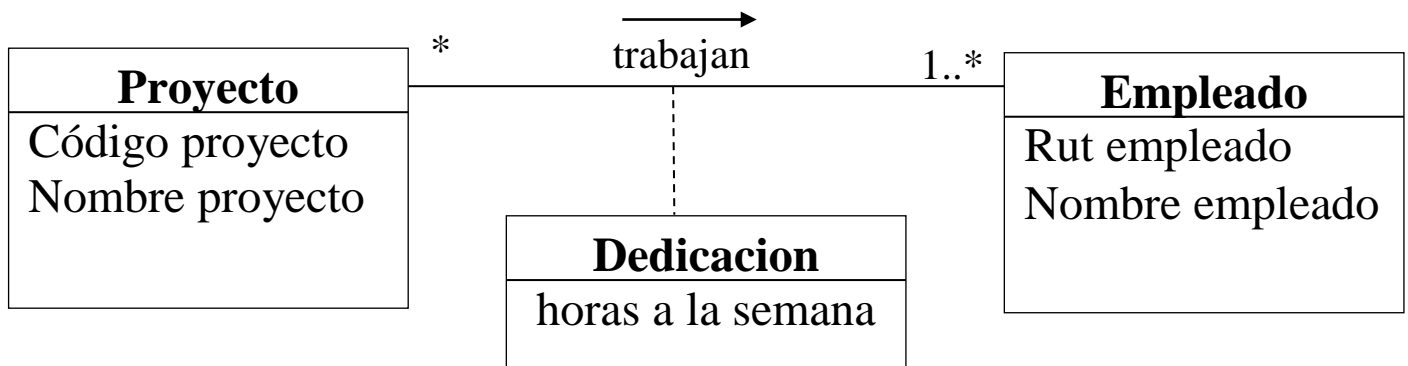
Modelo del Dominio + Operaciones



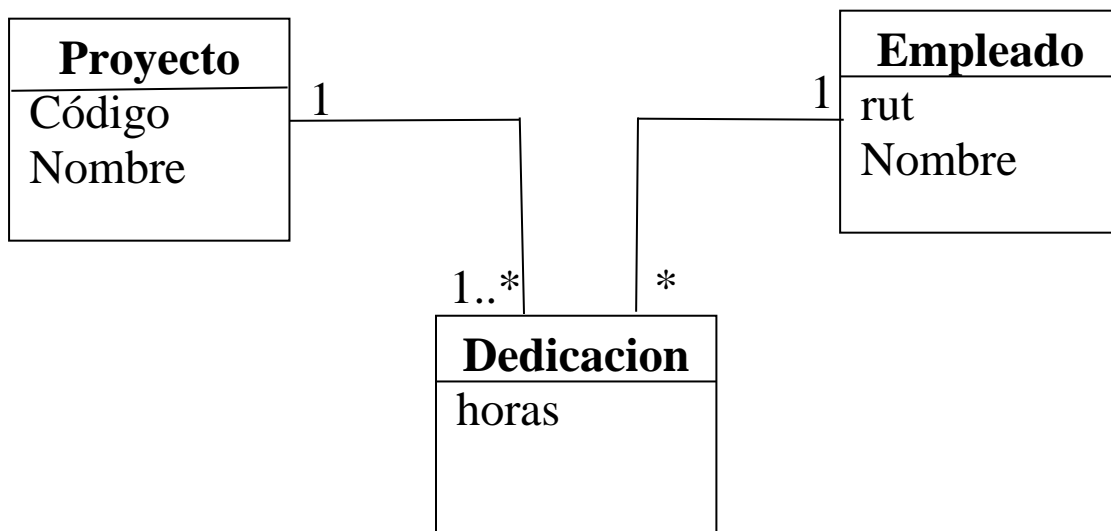
Ejercicio

Se debe construir una aplicación maneje los proyectos en los que trabaja un empleado. En un proyecto trabajan muchos empleados y un empleado puede trabajar en varios proyectos. Es importante saber cuántas horas a la semana le dedica cada empleado a cada uno de los proyectos en que trabaja .

Construya el modelo del dominio



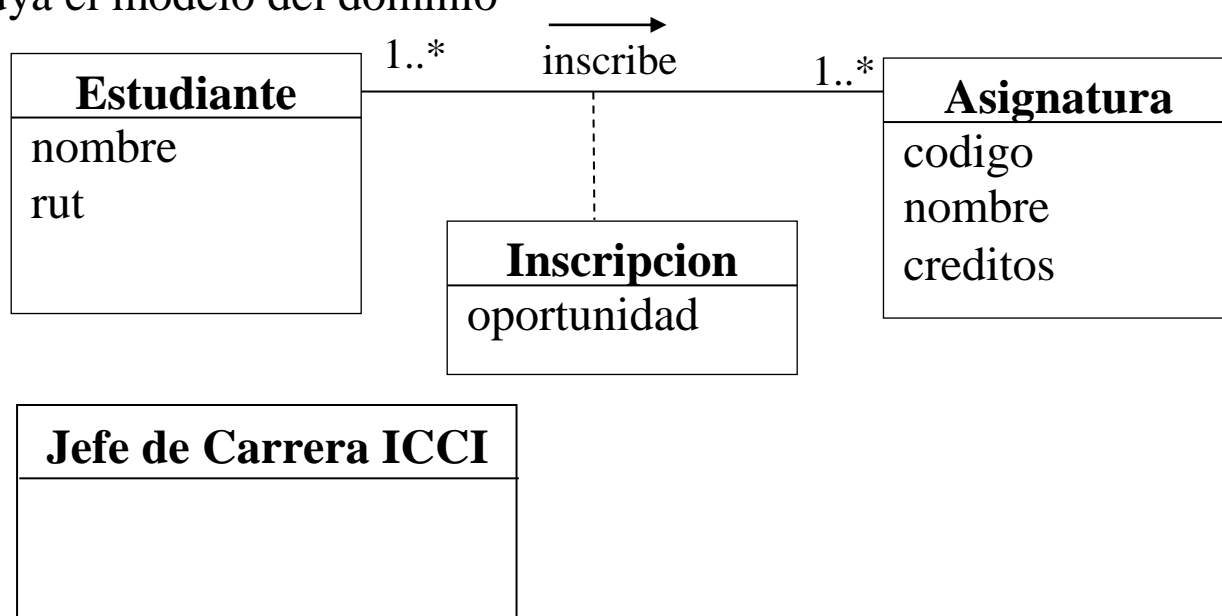
Otra forma



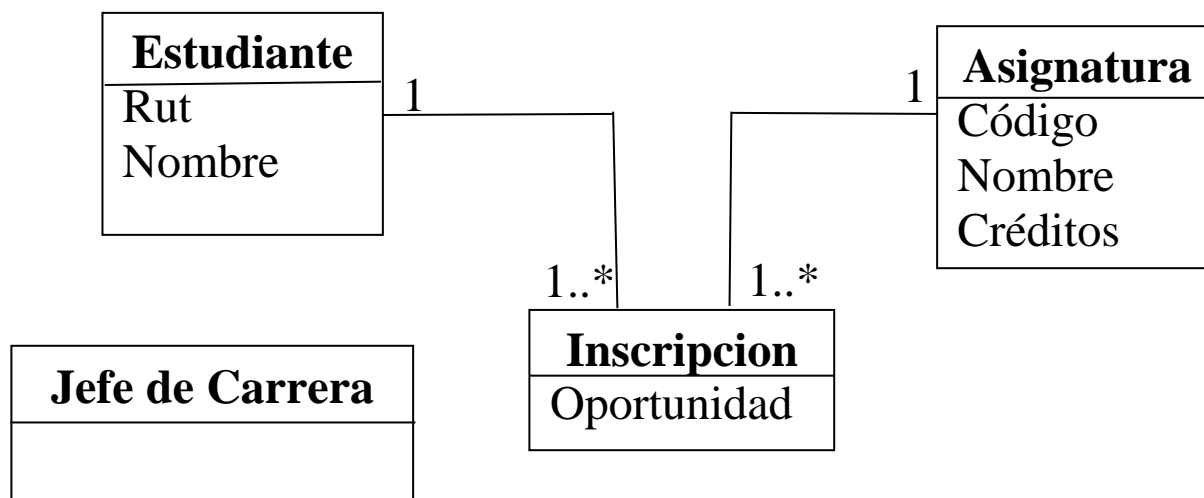
Ejercicio

Se debe construir una aplicación para que el jefe de carrera de ICCI pueda inscribirle a los estudiantes de la carrera las asignaturas de un determinado semestre. Para esto el jefe de carrera atiende a cada uno de los alumnos en su oficina, utilizando la aplicación para la inscripción. Cabe señalar que cuando el jefe de carrera le inscribe una asignatura a un estudiante, debe indicar en qué oportunidad la está inscribiendo. De una asignatura interesa el código, nombre y créditos. De un estudiante, su rut y nombre. Esta aplicación tendrá como usuario solo al jefe de carrera.

Construya el modelo del dominio



Otra forma



2.3. Conceptos de orientación al objeto

2.3.1. Cinco palabras claves del vocabulario O.O

Objeto: Colección de datos privados y operaciones públicas.

Clase: Descripción de un conjunto de objetos con propiedades (atributos) similares, con relaciones comunes con otros y con una semántica común.

Instancia: Una instancia de una clase, es un objeto de esa clase.

Método: Un cuerpo de procedimiento que implementa una operación.

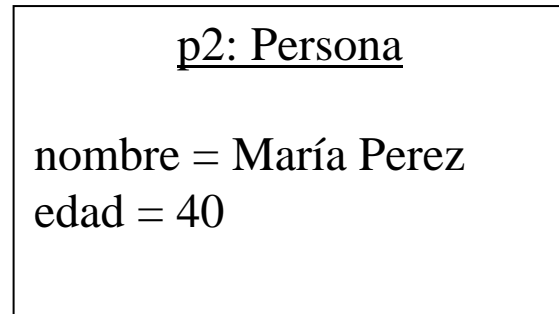
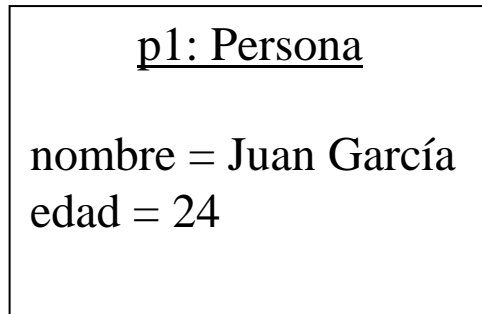
Mensaje: Un llamado a un procedimiento. Un requerimiento para ejecutar un método.

Ejemplo:

- Clase con sus atributos:

Persona
nombre: String edad : int

- Objetos con sus valores



Objeto:

- Colección de campos, junto a una colección de procedimientos (llamados métodos) que operan en los campos.
- Un objeto encapsula los componentes pasivos (campos) y los componentes activos (métodos) en una única entidad.
- Este encapsulamiento incrementa la modularidad del programa: al aislar un objeto del resto del programa, se obtiene un programa que es más fácil de entender y de modificar.
- **El estado de un objeto**, es representado por un conjunto de atributos.
- Los atributos mantienen información acerca de un objeto.
- **El comportamiento de un objeto**, representa las acciones que pueden ser realizadas sobre el objeto.
- Estas acciones ocurren cuando un mensaje es enviado hacia el objeto.

Clase:

- Todos los tipos que no son primitivos son CLASES
- Una clase es un conjunto de objetos que comparten una estructura común y un comportamiento común.
- Un objeto particular es simplemente una instancia de una clase.
- Un objeto no es una clase.
- ¡ Una clase puede ser un objeto ! (Por lo tanto, a la clase se le puede enviar un mensaje, por ejemplo, new)

Ejemplos de Clases:

- Puerta Para una empresa que fabrica puertas y ventanas
- Empleado Para un sistema de remuneraciones
- Habitación Para un sistema de reservas de un hotel
- Vehículo Para un sistema de arriendo de vehículos

Ejemplos de Atributos:

- Puerta largo
 ancho
 espesor
 tipo de madera

- Empleado
rut
nombre
dirección particular
fono particular
fono empresa
- Habitación
número
cantidad de camas
metros cuadrados
citófono
valor
- Vehículo
patente
marca
modelo
año

Ejemplo de Comportamiento:

- Puerta
abrir
cerrar
poner llave
sacar llave

Ejercicio

Se tiene que hacer un programa que maneje información de los alumnos y profesores de la UCN. Identifique las clases e indique para ellas atributos y comportamiento. Muestre para cada una de las clases identificadas, 2 ejemplos de objetos.

Alumno
numeroMatricula: String nombre: String edad : int
getNumeroMatricula(): String

Profesor
nombre: String tipo: int
getNombre(): String

<u>a1: Alumno</u>
numeroMatricula = 111 nombre = Juan García edad = 24

<u>a2: Alumno</u>
numeroMatricula = 222 nombre = Juan Perez edad = 19

<u>p1: Profesor</u>
nombre = Raúl Vega tipo = 1

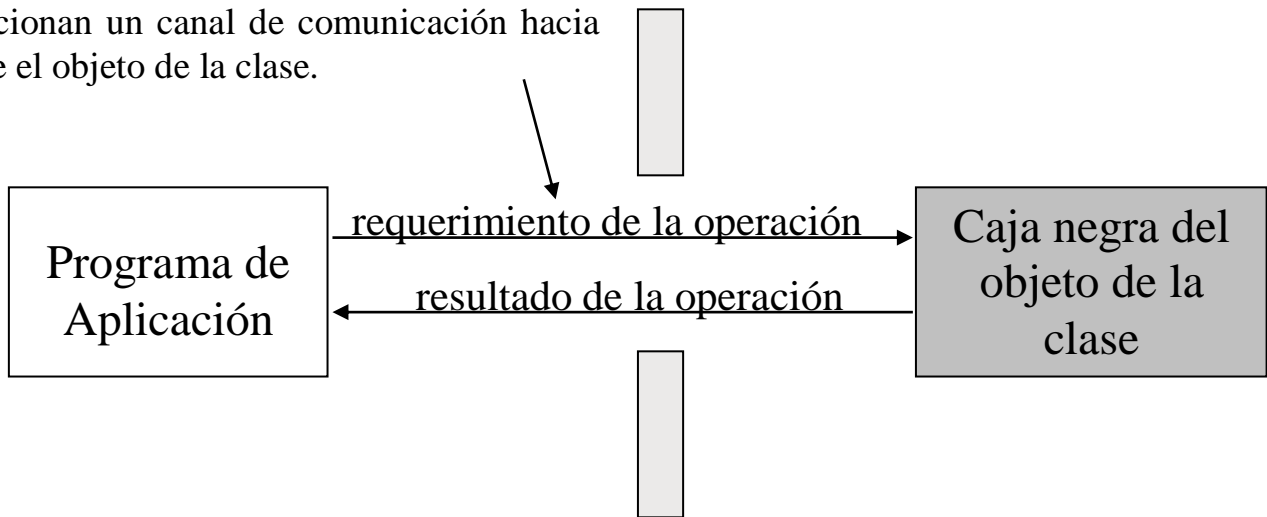
<u>p2: Profesor</u>
nombre = Pedro Soto tipo = 2

2.3.2. Interface, implementación y diagrama de una clase

Interface de una clase

- La interface de una clase proporciona su vista externa y por lo tanto enfatiza la abstracción, mientras esconde su estructura y los secretos de su comportamiento.
- En un nivel abstracto, una clase es una interface que define la conducta de sus objetos

Las operaciones, o funciones e interfaces, proporcionan un canal de comunicación hacia y desde el objeto de la clase.



Implementación de una clase

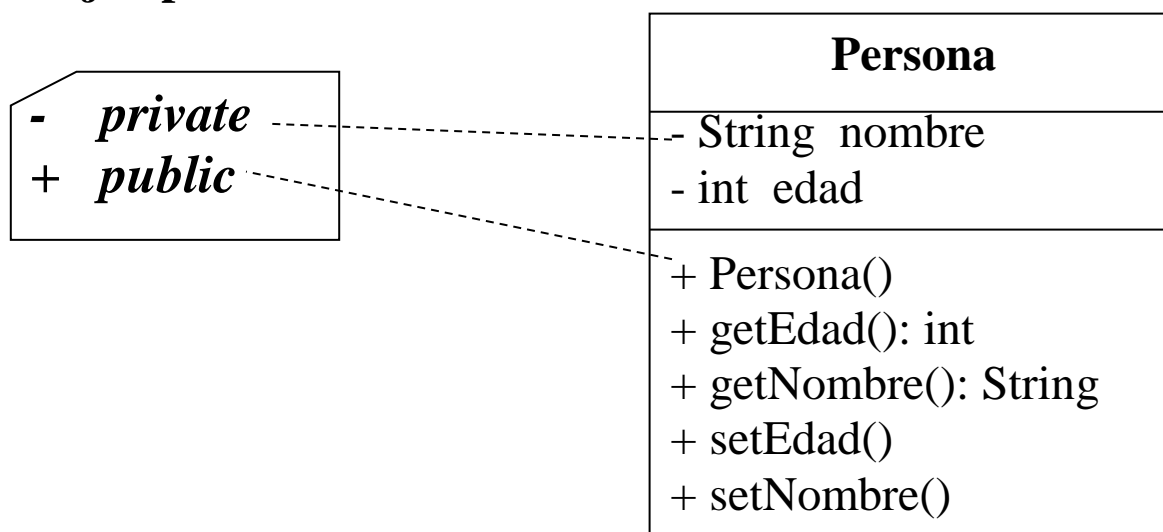
- La implementación de una clase corresponde a su vista interna, la que abarca los secretos de su comportamiento.
- La implementación de una clase consiste principalmente de la implementación de todas las operaciones definidas en la interface de la clase.
- Se puede dividir la interface de una clase en tres tipos de visibilidades:
 - PUBLIC
 - PROTECTED
 - PRIVATE

- **PUBLIC:** Es una declaración que está visible a todos los clientes.
- **PROTECTED:** Es una declaración que está visible solamente a la misma clase y a sus subclases.
- **PRIVATE:** Es una declaración que está visible solamente a la misma clase.

Diagrama de clase

Nombre de Clase		
Visibilidad	tipo de dato 1	nombre atributo 1
Visibilidad	tipo de dato 2	nombre atributo 2
...		
Visibilidad	nombre operación 1 (lista argum.1): tipo de resultado 1	
Visibilidad	nombre operación 2 (lista argum.2): tipo de resultado 2	
...		

Ejemplo:



2.3.3. Variables y Métodos de Instancia, Variables y Métodos de Clase

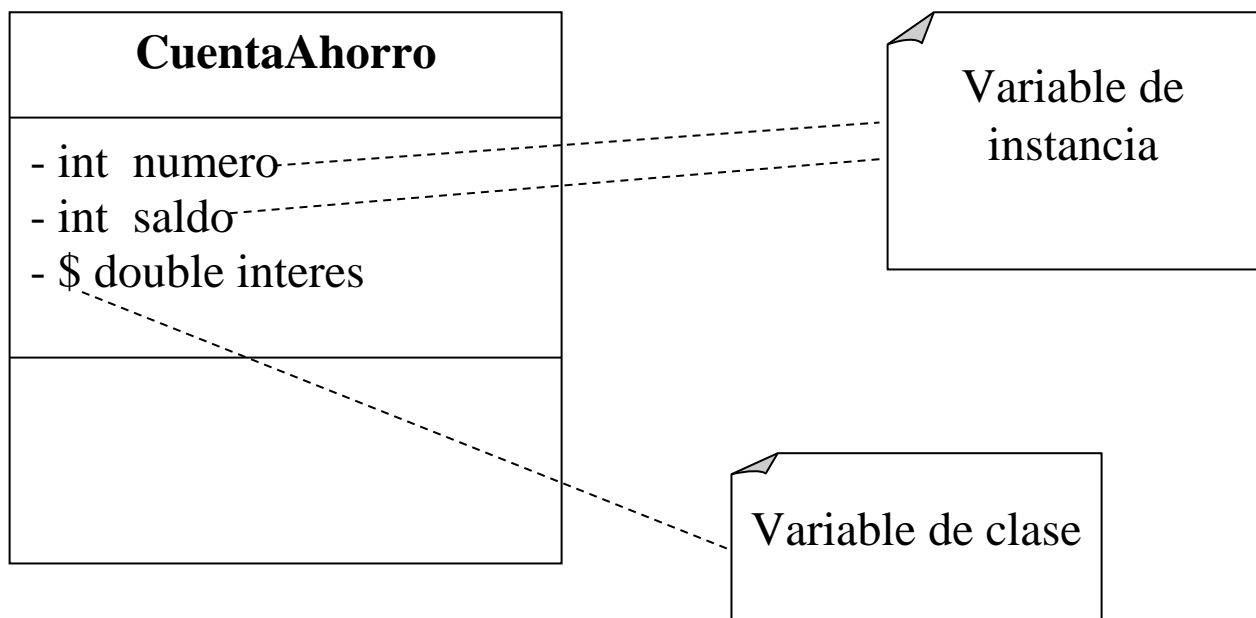
Variable de Instancia:

- Una **variable de instancia** pertenece a una instancia. Su valor sólo puede ser cambiado por operaciones que pertenecen a la instancia.
- Existe tanto como lo hace la instancia, es decir, la existencia de la variable de instancia depende de la existencia del objeto que la contiene.

Variable de Clase:

- Variable compartida por todas las instancias de una clase (una ÚNICA copia).
- Todos los objetos de la clase la pueden ver.

Ejemplo:



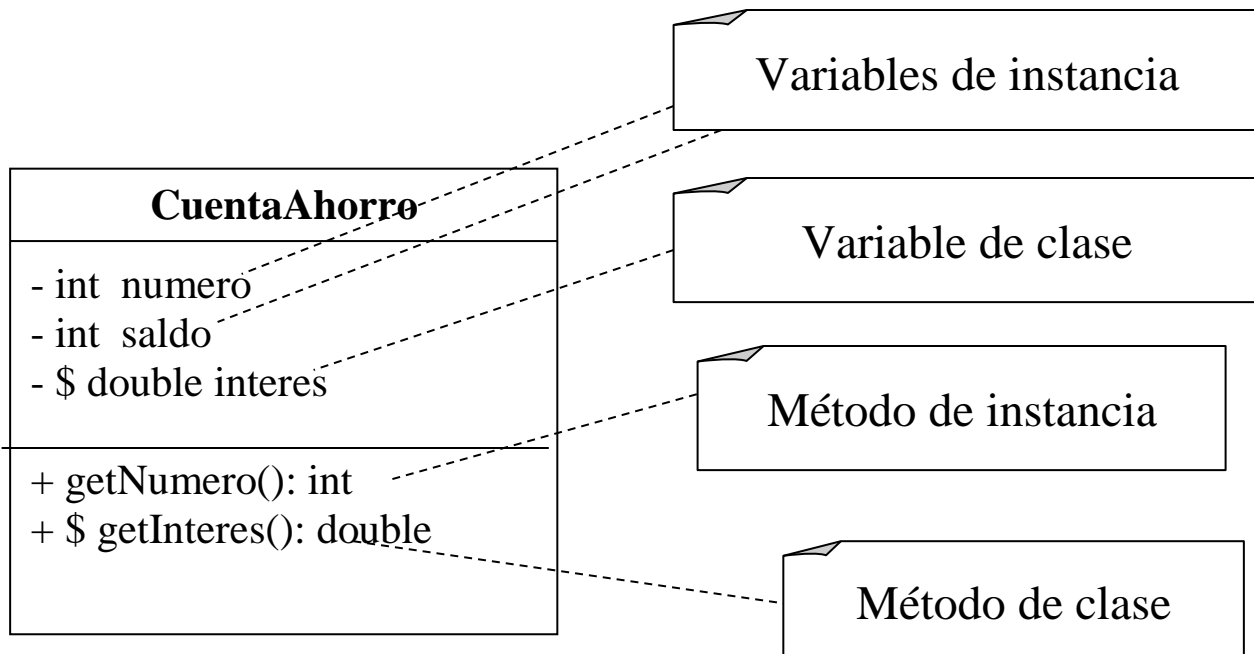
Métodos de instancia:

- Pertenece a un objeto.

Métodos de clase:

- Está asociado a una clase.

Ejemplo:



Forma de invocación:

- `c.getNumero()` `c` es un objeto de la clase `CuentaAhorro`
- `CuentaAhorro.getInteres()`

2.3.4. Operaciones sobre un objeto

En la práctica se ha encontrado que un cliente realiza típicamente cinco tipos de operaciones sobre un objeto:

- Modificador
- Selector
- Iterador
- Constructor
- Destructor

Modificador: Es una operación que altera el estado de un objeto (Métodos set).

Selector: Es una operación que accesa al estado de un objeto, pero no lo altera (Métodos get).

Iterador: Es una operación que permite que todas las partes de un objeto sean accesadas, en algún orden bien definido.

Constructor: Es una operación que crea un objeto y/o inicializa su estado.

Destructor: Es una operación que libera el estado de un objeto y/o destruye el objeto mismo.

2.3.5. Declaración, creación, asignación, comparación y destrucción de objetos

A. Declaración del objeto

`NombreClase nombreVariable;`

La declaración no establece almacenamiento para la variable.

Ejemplo: `Persona p;`

B. Creación de objetos

Un objeto en JAVA, se construye únicamente por la expresión de creación dinámica, que tiene la forma:

`new NombreClase (parámetros_actuales);`

Ejemplo: `new Persona (25);`

La evaluación de esta expresión significa:

- a) Asignación de memoria para un nuevo objeto de la clase `NombreClase`, asociándole valores iniciales a las variables del objeto.
- b) Se invoca el constructor apropiado de la clase, con los parámetros correspondientes.
- c) Se regresa como resultado la referencia al objeto creado.

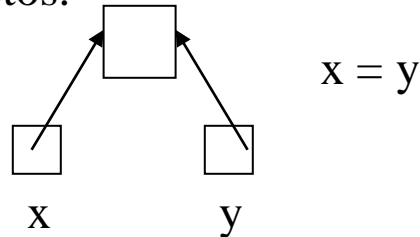
Si no hay rutina constructora, se crea igual el espacio y todos los atributos quedan en nulo.

C. Asignación de Objetos

Ejemplo: `miObjeto = tuObjeto;`

“*Shallow Copy*” o Copia Superficial

Nuevo almacenamiento no es creado para el objeto destino, en la asignación de objetos.



D. Comparación de Objetos

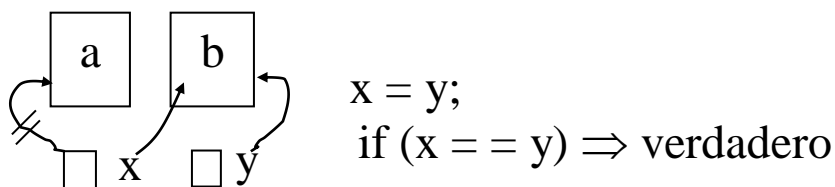
Los operadores `==` (igual) y `!=` (no igual) se utilizan para comparar expresiones cuyos valores son **referencias** a objetos.

Esta comparación prueba si al evaluar las dos expresiones, el resultado es o no, la **referencia al mismo objeto** (identidad versus igualdad).

Semántica por referencia

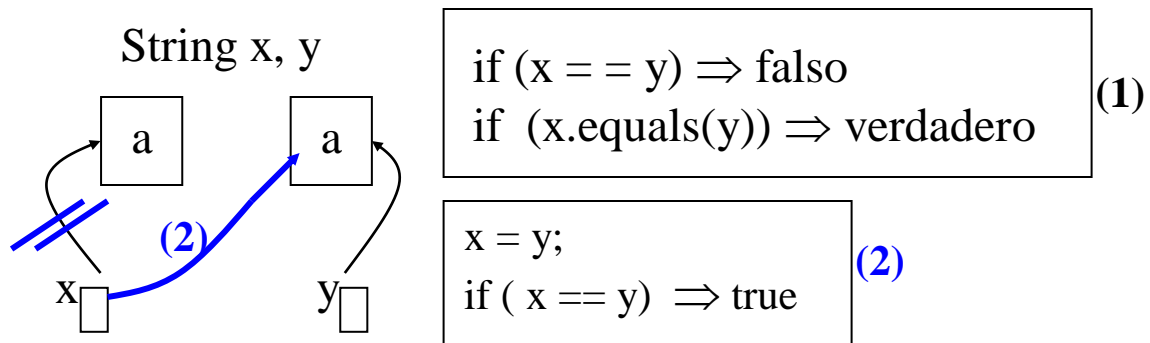
Las variables de tipo objeto almacenan **referencias** al objeto. Los objetos en JAVA, tienen una “**Semántica por Referencia**” (“Reference Semantics”). Es decir, ellos son realmente punteros.

Ejemplo: `x, y` son objetos de una clase

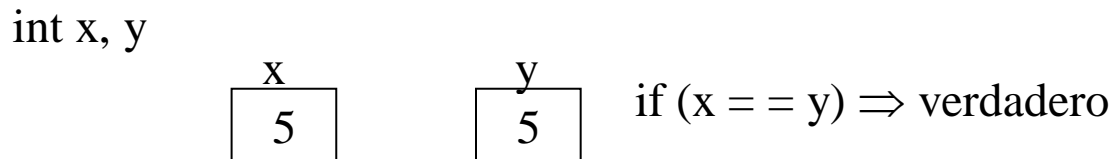


***x e y referencian
al mismo objeto***

Ejemplo de identidad:



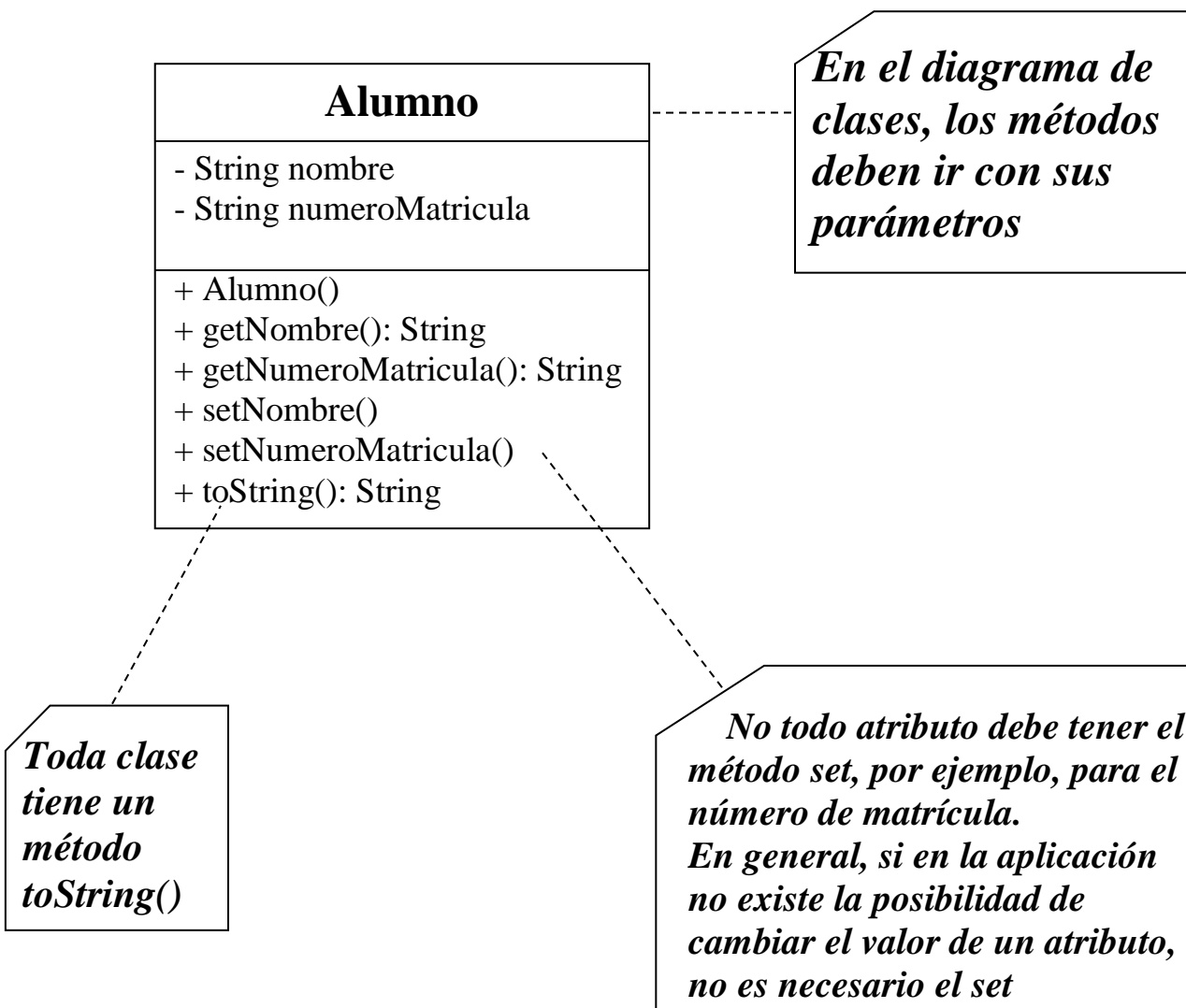
Ejemplo de igualdad:



E. Destrucción de objetos

En Java no existe el concepto de destructor explícito de objetos.

La recuperación de memoria de los objetos, que ya no son útiles, se realiza automáticamente por el proceso recolector de basura, el cual se ejecuta de manera concurrente con el programa en Java.

Ejemplo:

```

public class Alumno {
    private String nombre;
    private String numeroMatricula;

    /**
     * @param nom
     * @param matricula
     */
    public Alumno(String nom, String matricula) {
        nombre = nom;
        numeroMatricula=matricula;
    }

    /**
     * @return the nombre
     */
    public String getNombre() {
        return nombre;
    }

    /**
     * @param nom the nombre to set
     */
    public void setNombre(String nom) {
        nombre = nom;
    }

    /**
     * @return the numeroMatricula
     */
    public String getNumeroMatricula() {
        return numeroMatricula;
    }

    /**
     * @param matricula the
     * numeroMatricula to set
     */
    public void setNumeroMatricula(String matricula) {
        numeroMatricula=matricula;
    }

    /* (non-Javadoc)
     * @see java.lang.Object#toString()
     */
    @Override
    public String toString() {
        return "Alumno [nombre=" + nombre + ", numeroMatricula=" +
            numeroMatricula + "]\n";
    }
}

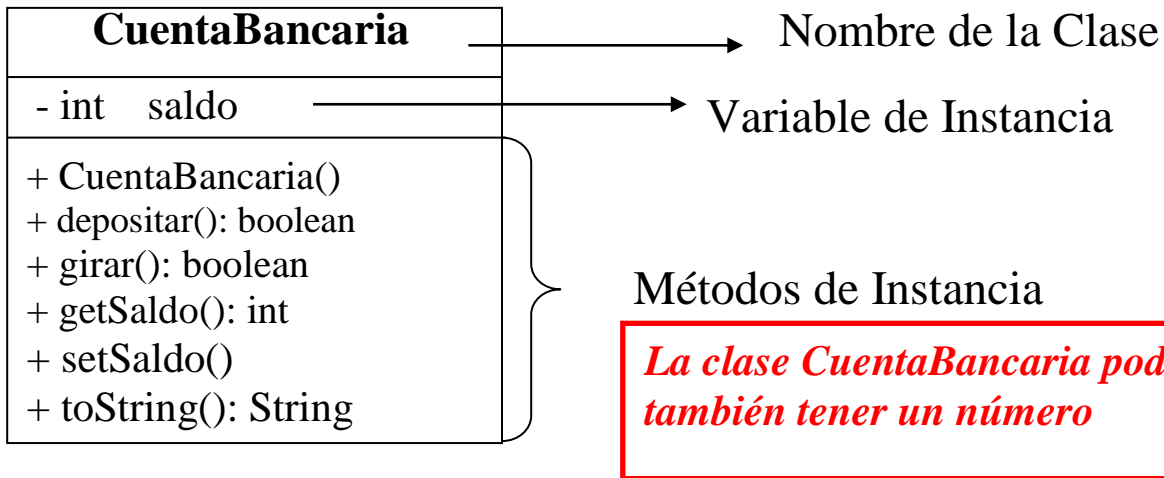
```

*El método toString() sobre
escribe desde
java.lang.Object.toString*

Ejemplo de programa en Java

Se desea manejar la información de las cuentas bancarias de un banco. Una cuenta tiene un saldo. Se puede depositar y girar sobre la cuenta bancaria. La aplicación debe hacer un depósito sobre una cuenta y luego un giro sobre la misma cuenta. Una vez hechas las transacciones sobre la cuenta, se debe desplegar su saldo.

Diagrama de clases



```

public class CuentaBancaria {

    private int saldo;

    /**
     * @param saldo
     */
    public CuentaBancaria(int saldoInicial) {
        saldo = saldoInicial;
    }

    /**
     * @return the saldo
     */
    public int getSaldo() {
        return saldo;
    }
}
  
```

```

/**
 * @param saldo the saldo to set
 */
public void setSaldo(int saldoNuevo) {
    saldo = saldoNuevo;
}

/* (non-Javadoc)
 * @see java.lang.Object#toString()
 */
@Override
public String toString() {
    return "CuentaBancaria [saldo=" + saldo + "]";
}

public boolean depositar(int cantidad) {
    if (cantidad >= 0) {
        saldo = saldo + cantidad;
        return true;
    }
    else {
        return false;
    }
}

public boolean girar(int cantidad) {
    if (cantidad >= 0 && saldo >= cantidad) {
        saldo = saldo - cantidad;
        return true;
    }
    else {
        return false;
    }
}

} //Fin cuentaBancaria

```

Toda clase tiene un método toString()

```

import ucn.Stdout;

public class BankApp {

    public static void main(String[] args) {
        //Crea una cuenta
        CuentaBancaria cuentaBancaria = new CuentaBancaria(100);

        StdOut.println("Saldo antes de las transacciones: "
                        + cuentaBancaria.getSaldo());
        StdOut.println("Cuenta bancaria antes de las transacciones: "
                        + cuentaBancaria.toString());

        cuentaBancaria.depositar(74); // efectuar un depósito
        cuentaBancaria.girar(20); // efectuar un retiro

        StdOut.println("Despues de las transacciones: " +
                        cuentaBancaria.getSaldo());
        StdOut.println("Cuenta bancaria antes de las transacciones: "
                        + cuentaBancaria.toString());

    } //fin main
} //Fin BankApp

```

Se imprime

```

Saldo antes de las transacciones: 100

Cuenta bancaria antes de las transacciones:CuentaBancaria [saldo=100]

Despues de las transacciones: 154

Cuenta bancaria despues de las transacciones: CuentaBancaria [saldo=154]

```

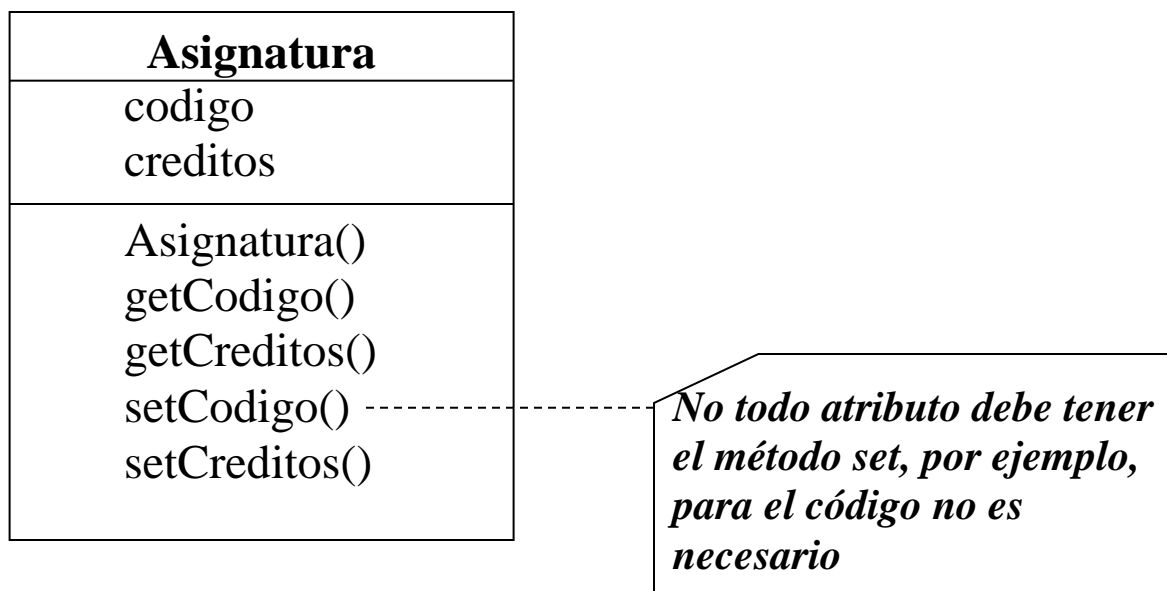
Diagrama de objetos

cuentaBancaria: CuentaBancaria

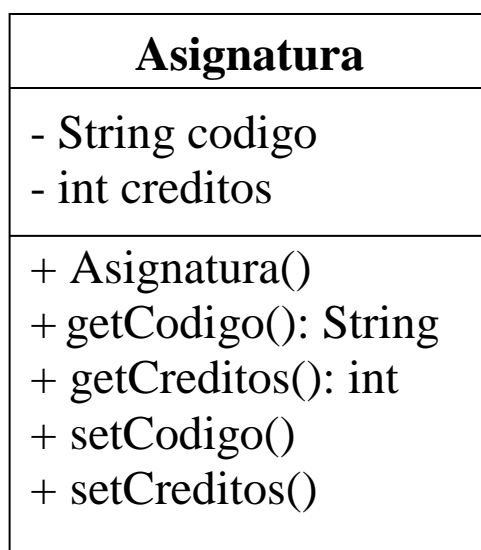
~~saldo = 100~~
~~174~~
 154

Ejercicio

Dado el siguiente diagrama de clases, complételo y escriba en Java la clase Asignatura y un programa Java que cree un objeto Asignatura cuyo código es “cc 571” y créditos = 14. Una vez creado el objeto se debe desplegar por pantalla sus datos.



Solución



```

public class Asignatura {
    private String codigo;
    private int credits;

    public Asignatura(String cod, int cred) {
        codigo = cod;
        credits = cred;
    }

    public String getCodigo() {
        return codigo;
    }

    public int getCredits() {
        return credits;
    }

    public void setCodigo(String cod) {
        codigo = cod;
    }

    public void setCredits(int cred) {
        credits = cred;
    }

    -----
}

```

Podríamos tener el método toString()

```

public class App {

    public static void main (String [ ] args) {
        Asignatura asignatura = new Asignatura ("cc 571", 14);
        StdOut.println("Datos de la asignatura: Codigo = " +
            asignatura.getCodigo() + " credits= " +
            asignatura.getCredits());
    }
}

```

Notas:

- La rutina constructora no tiene un tipo asociado al resultado, debido a que retorna el objeto.
- Los parámetros en JAVA son sólo de modo IN.

Ejemplo

El programa siguiente, crea dos objetos Punto diferentes y pone valores únicos en cada uno.

```

public class Punto {
    private int x;
    private int y;

    public void setX(int xx){
        x = xx;
    }
    public void setY(int yy){
        y = yy;
    }
    public int getX(){
        return x;
    }
    public int getY(){
        return y;
    }
    -----
}

```

Punto
- int x - int y
+ setX() + setY() + getX(): int + getY(): int

Podríamos tener el método toString()

```

public class DosPuntos {

    public static void main(String args[]) {
        Punto p1 = new Punto();
        Punto p2 = new Punto();
    }
}

```

```

    p1.setX(10);
    p1.setY(20);

    p2.setX(42);
    p2.setY(99);

    StdOut.println ("x = " + p1.getX() + " y = " + p1.getY());

    StdOut.println ("x = " + p2.getX() + " y = " + p2.getY());
}
}

```

Imprime

<u>p1: Punto</u>
x = 10
y = 20

<u>p2: Punto</u>
x = 42
y = 99

x = 10	y = 20
x = 42	y = 99

```

public class Punto {
    private int x;
    private int y;

    public void init (int xx, int yy) {
        x = xx;
        y = yy;
    }

    public int getX(){
        return x;
    }

    public int getY(){
        return y;
    }
}

```

El programa siguiente, aprovecha el método `init` para efectuar la inicialización del objeto.

```
public class TwoPointsInit {  
  
    public static void main(String args[]) {  
  
        Punto p1 = new Punto();  
        Punto p2 = new Punto();  
  
        p1.init(10, 20);  
        p2.init(42, 99);  
  
        StdOut.println ("x = " + p1.getX() + " y = " + p1.getY());  
        StdOut.println ("x = " + p2.getX() + " y = " + p2.getY());  
    }  
  
}
```

2.3.6. Mensajes a THIS

- Dentro de la implementación de un método, el identificador **this** representa un parámetro implícito, que referencia al objeto para el cual el método fue invocado.
- “Un mensaje a this, es un mensaje al mismo objeto”
- En: C++: this
 Smalltalk: self
 Delphi Pascal: self
- Si el objeto al que se quiere enviar el mensaje es el mismo, se debe usar el **this**.

Ejemplo 1 :

Inicializar las variables de una clase, utilizando el constructor.

Punto
- int x - int y
+ Punto() + getX(): int + getY(): int

```

public class Punto {
    private int x;
    private int y;

    public Punto(int x, int y) {
        this.x = x;
        this.y = y;
    }
    public int getX(){
        return this.x;
    }
    public int getY(){
        return this.y;
    }
    //...Métodos set
}

public class CrearPunto {

    public static void main(String args[]){
        Punto punto = new Punto(10, 20);
        StdOut.println("x = " + punto.getX() + " y = " + punto.getY());
    }
}

```

Ejemplo 2:

Empleado
...
+ calcSueldoTotal(): int + calcDescuentoTotal(): int + calcSueldoFinal(): int

SueldoFinal =
 this.calcSueldoTotal() –
 this.calcDescuentoTotal();

2.3.7. Sobrecarga de métodos

Ejemplo 1: Un constructor llama a otro constructor, para construir la instancia.

```
public class Punto {
    private int x;
    private int y;
    public Punto(int x, int y) {
        this.x = x;
        this.y = y;
    }
    public Punto() {
        this(-1, -1); //this.Punto(-1,-1);
    }
}
```

Ejemplo 2: Versión de la clase Punto que utiliza la sobrecarga de método para crear un constructor alternativo que establece algunos valores por omisión para las coordenadas x e y.

```
public class Punto {
    private int x;
    private int y;
    public Punto(int x, int y) {
        this.x = x;
        this.y = y;
    }
    public Punto() {
        x = -1;
        y = -1;
    }
    public int getX(){
        return x;
    }
}
```



```

    public int getY(){
        return y;
    }
}

```

```

public class PointCreateAlt {

```

```

    public static void main(String args[]) {
        Punto p = new Punto();
        StdOut.println("x = " + p.getX() + " y = " + p.getY());
    }
}

```

Ejemplo 3: Existen dos versiones de un método llamado distancia, asociado a la clase Punto. Un método toma un par x, y, y el otro toma otro objeto Punto.

```

public class Punto {
    private int x;
    private int y;

    public Punto(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public int getX(){
        return x;
    }

    public int getY(){
        return y;
    }

    //....Métodos set

```

Punto
- int x - int y
+ Punto() + setX() + setY() + getX(): int + getY(): int + distancia(int x, int y): double + distancia(Punto p): double

En el diagrama de clases, se deben colocar todas las versiones de los métodos sobrecargados. Se diferencian por los parámetros

```

public double distancia(int x, int y) {
    int dx = this.x - x;
    int dy = this.y - y;
    return Math.sqrt(dx*dx + dy*dy);
}

```

```

public double distancia(Punto p) {
    return this.distancia(p.x, p.y);
}

```

```

}

```

```

public class PointDist {

```

```

    public static void main (String args[]) {
        Punto p1 = new Punto(0, 0);
        Punto p2 = new Punto(30, 40);
        StdOut.println(    "p1 = " + p1.getX() + ", " + p1.getY());
        StdOut.println(    "p2 = " + p2.getX() + ", " + p2.getY());

        StdOut.println(    "p1.distancia(p2) = " +
                                p1.distancia(p2));
        StdOut.println( "p1.distancia(60, 80) = " +
                                p1.distancia(60, 80));
    }
}

```

```

}

```

p1: Punto

x = 0

y = 0

p2: Punto

x = 30

y = 40

En el ejemplo se utiliza el método estático sqrt de la clase Math para calcular la raíz cuadrada de su parámetro.

2.4. Relacionamientos entre Clases/Objetos

2.4.1. Concepto de relacionamiento

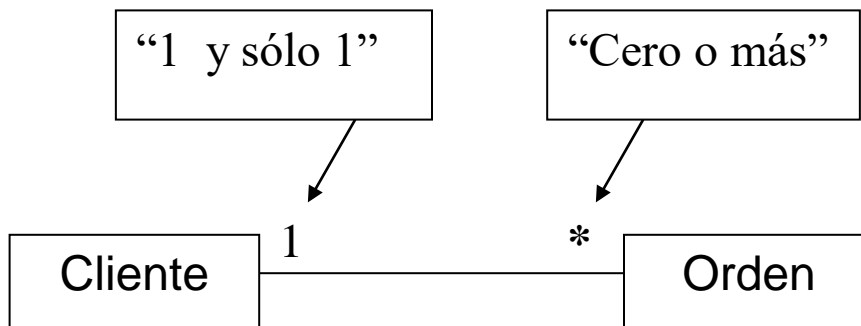
Conceptualmente, los objetos y las clases no existen aisladas.

El relacionamiento entre clases/objetos es una asociación natural en el problema que existe entre uno o más objetos/clases.

Ejemplo

Para un problema dado, se pueden efectuar las siguientes aseveraciones:

- Un cliente **coloca** cero o más ordenes de compra.
- Una orden de compra es **colocada por** uno y sólo un cliente.

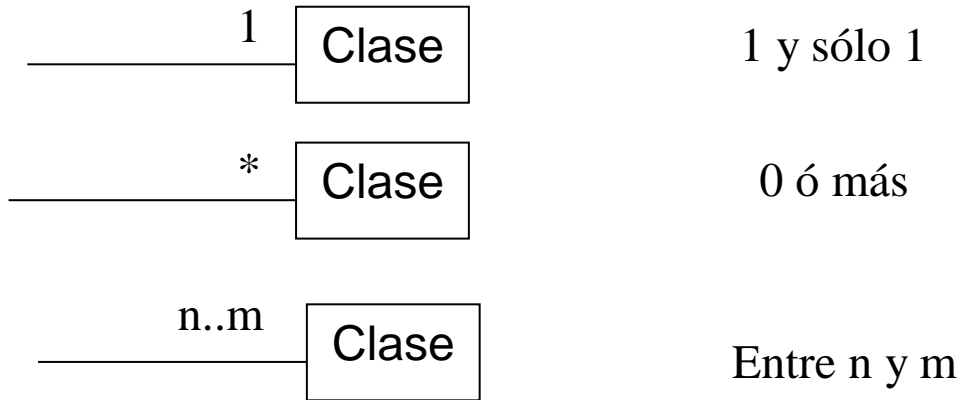


- La línea entre las clases muestra el relacionamiento.

2.4.2. Multiplicidad del relacionamiento

Igual que en el modelo del dominio

Tipos de multiplicidad



2.4.3. Agregación

Una clase especial de relacionamiento puede **existir entre objetos / clases**:

- Algunas veces los objetos / clases **están formados** por otros objetos.
- Un objeto **está compuesto** por otros objetos.
- Este tipo especial de relacionamiento se llama **agregación**.
- Un objeto es fabricado con otros objetos.


Ejemplos:

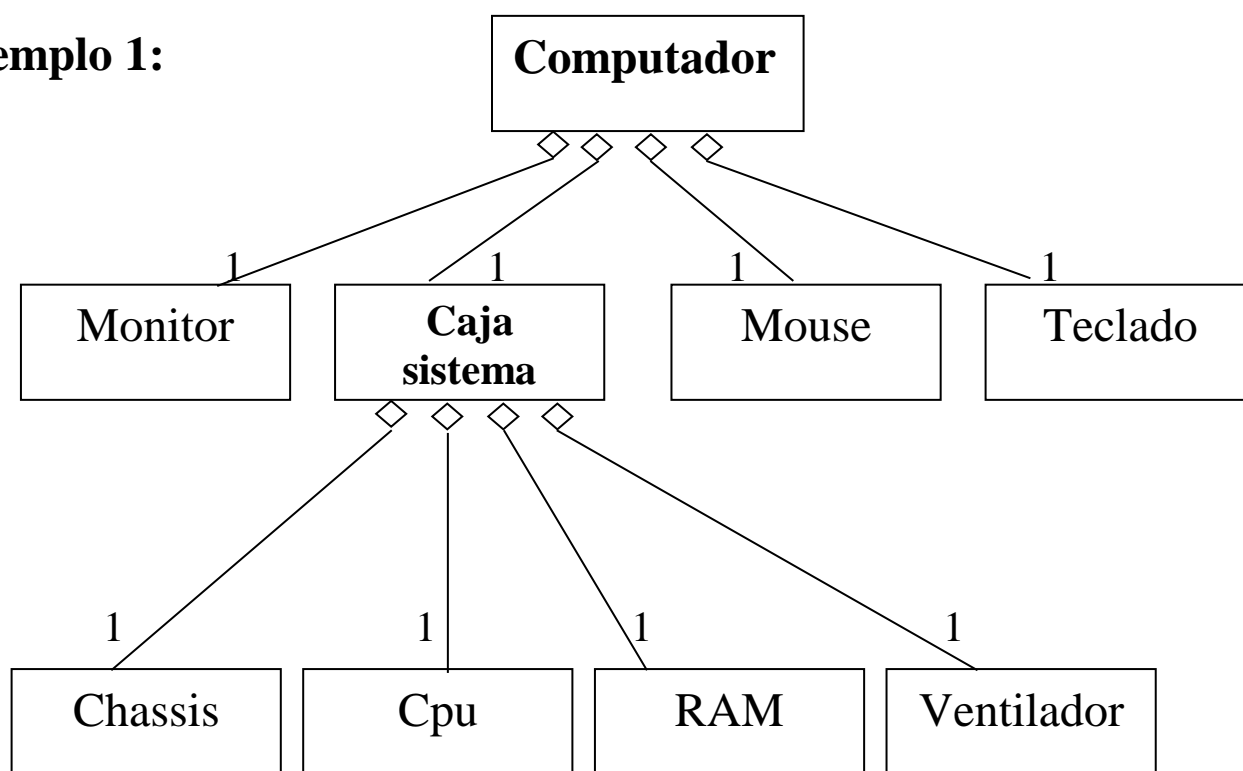
Auto:

- Transmisión
- Tubo de escape
- Motor
- Suspensión
- Ruedas

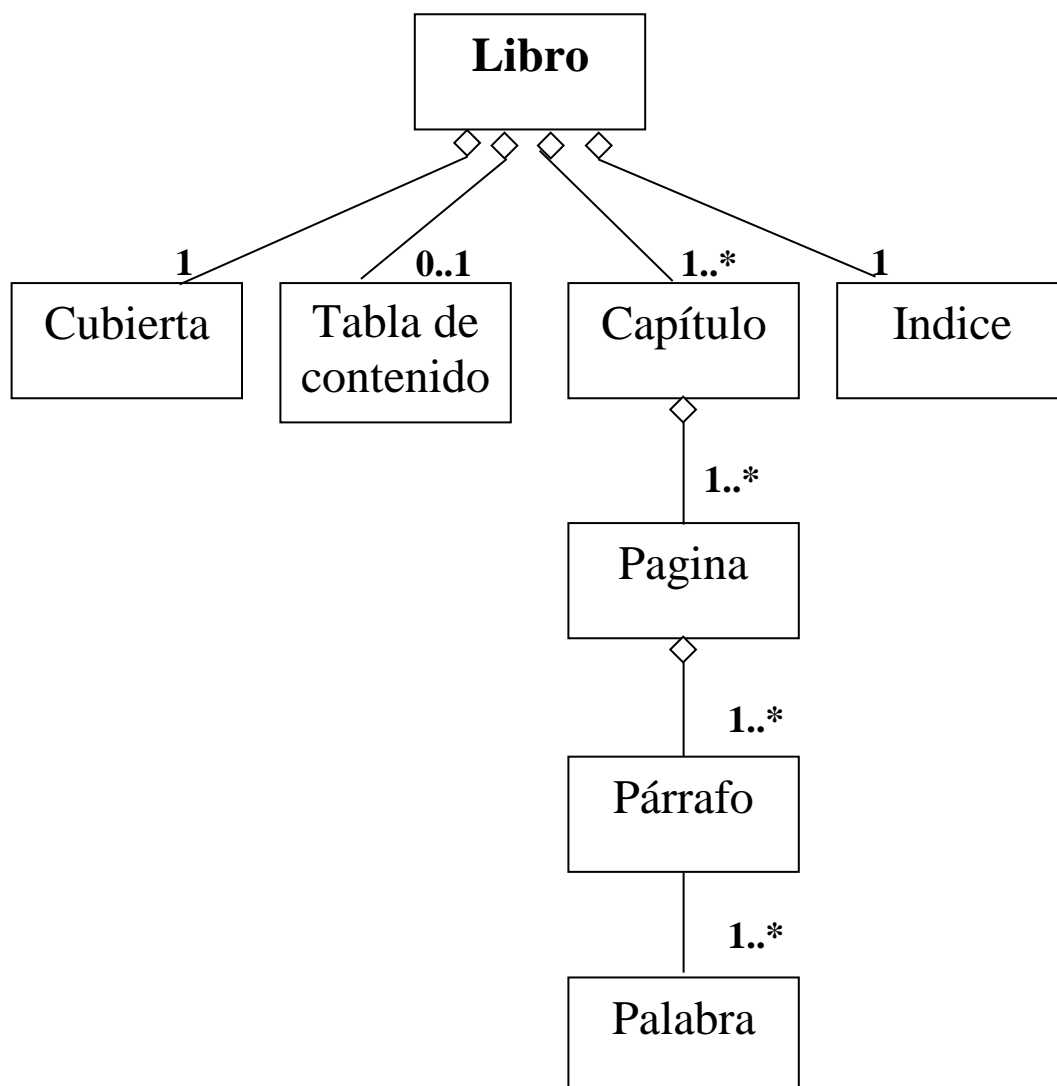
Ventana:

- Botones
- Labels
- Menús

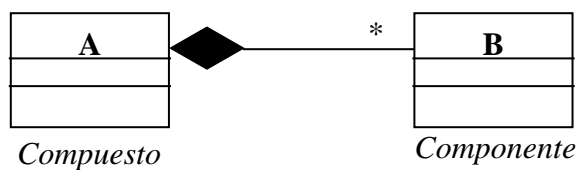
Se utiliza el símbolo  en un extremo del relacionamiento para indicar la agregación.

Ejemplo 1:

Ejemplo 2:



2.4.4. Composición



La diferencia entre composición y agregación es que en la composición si se elimina el compuesto se elimina también los componentes. En el caso de la agregación no.

2.5. Contenedores implementados con arreglos

2.5.1. Definir una clase Contenedor

- Se mejora la interface con los clientes de la clase.
- El usuario de la clase, en este caso Contenedor, debe estar libre para concentrarse en **¿qué?**, en vez de **¿cómo?**

¿Qué va a ser insertado?

¿Qué va a ser eliminado?

¿Qué va a ser accesado?, en vez de conocer ¿Cómo estas actividades son llevadas a cabo?

- El contenedor debe tener métodos tales como: buscar, eliminar, agregar, obtener el elemento i-ésimo del contenedor, etc.

ListaDoubles
- double [] lista - int cantidadElementos - int max
+ ListaDoubles() + encontrar(): int + insertar(): boolean + eliminar(): boolean + getCantidadElementos(): int + getElemI(): double

// ListaDoubles.java

public class ListaDoubles {

private double[] lista; **// referencia el arreglo**

private int cantidadElementos; **// número de items**

private int max; **//Máxima cantidad de elementos para el contenedor**

//-----

public ListaDoubles(int max) {**// constructor**

lista = new double[max];**//crea el arreglo**

cantidadElementos = 0; **// no tiene items todavía**

this.max = max;

}

//-----

public int encontrar(double claveBusqueda){

// Si encuentra el valor especificado, retorna la posición donde

// lo encuentra. Sino, retorna un -1

int j;

for(j=0; j< cantidadElementos; j++) { **// para cada elemento**

if(lista[j] == claveBusqueda){ **// ¿item encontrado?**

break; **// sale del loop**

}

}

if(j == cantidadElementos) { **//no lo encontré**

return -1;

}

else{

return j; **// si lo encontré, en la posición j**

}

} // end encontrar()


```

public boolean insertar(double valor){ //inserta elemento en el contenedor
    if (cantidadElementos < max) { //Hay espacio
        lista[cantidadElementos] = valor; // lo inserta
        cantidadElementos ++; // incrementa el tamaño
        return true;
    }
    else {
        return false;
    }
}
//-----
public boolean eliminar(double valor){
    int j;
    for(j=0; j< cantidadElementos; j++){
        if( valor == lista[j] ) {
            break;
        }
    }
    if(j== cantidadElementos){//no lo encontré
        return false;
    }
    else { // si lo encontré
        //Corrimiento
        for(int k=j; k<cantidadElementos - 1;k++){
            lista[k] = lista[k+1];
        }
        cantidadElementos--; // decrementa tamaño
        return true;
    } //fin else
} // end eliminar()
//-----
public int getCantidadElementos() {
    return cantidadElementos;
}
//-----
public double getElemI(int i) {
    return lista[i];
}
// end class ListaDoubles

```

```

int j = 0;
while( j< cantidadElementos && valor!=lista[j]){
    j++;
}

```

**Busca
el
valor**

```

int j;
for(j=0; j< cantidadElementos; j++){
    if( valor == lista[j] ) {
        break;
    }
}

```

```

int pos=this.encontrar(valor);
if (pos != -1){
    //Corrimiento
    //.....
}

```

***Se debe chequear que el i esté en
rango del 0 y cantidadElementos***

```

public class ListaDoublesApp {
    public static void main(String[] args){
        int tamMaximo = 100; // tamaño máximo
        ListaDoubles ld; // referencia al arreglo
        ld = new ListaDoubles(tamMaximo); // crea el contenedor

        ld.insertar(77); // inserta 10 items
        ld.insertar(99);
        ld.insertar(44);
        ld.insertar(55);
        ld.insertar(22);
        ld.insertar(88);
        ld.insertar(11);
        ld.insertar(00);
        ld.insertar(66);
        ld.insertar (33);

        // Despliega los items del contenedor
        for (int i =0; i <ld.getCantidadElementos () ; i++){
            StdOut.println(ld.getElemI(i));
        }

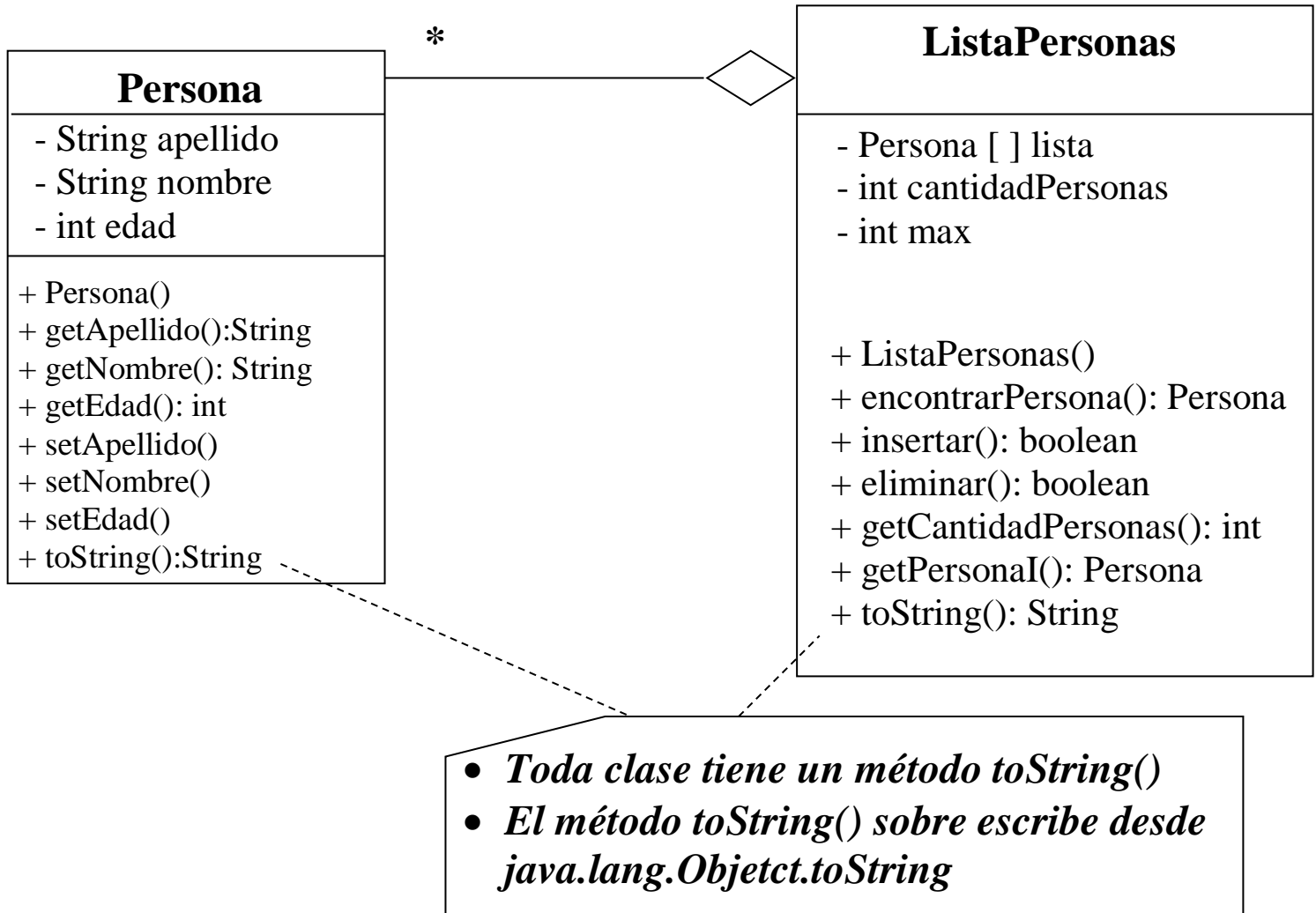
        int claveBusqueda = 35; // busca el item
        if( ld.encontrar(claveBusqueda)!= -1){
            StdOut.println("Encontrado " + claveBusqueda);
        }
        else {
            StdOut.println("No Encontrado " + claveBusqueda);
        }

        ld.eliminar(00); // elimina 3 items
        ld.eliminar(55);
        ld.eliminar(99);
        // Despliega los items del contenedor
        for (int i =0; i <ld.getCantidadElementos () ; i++){
            StdOut.println(ld.getElemI(i));
        }
    } // end main()
} // end class ListaDoublesApp

```

Ejercicio

En el ejemplo anterior, el contenedor almacena datos de un tipo primitivo. El siguiente ejemplo, muestra cómo guardar objetos.



```

public class Persona {
    private String apellido;
    private String nombre;
    private int edad;
    /**
     * @param apellido
     * @param nombre
     * @param edad
     */
    public Persona(String apellido, String nombre, int edad) {
        this.apellido = apellido;
        this.nombre = nombre;
        this.edad = edad;
    }
}
  
```

```

/**
 * @return the apellido
 */
public String getApellido() {
    return apellido;
}

/**
 * @param apellido the apellido to set
 */
public void setApellido(String apellido) {
    this.apellido = apellido;
}

/**
 * @return the nombre
 */
public String getNombre() {
    return nombre;
}

/**
 * @param nombre the nombre to set
 */
public void setNombre(String nombre) {
    this.nombre = nombre;
}

/**
 * @return the edad
 */
public int getEdad() {
    return edad;
}

/**
 * @param edad the edad to set
 */
public void setEdad(int edad) {
    this.edad = edad;
}

/* (non-Javadoc)
 * @see java.lang.Object#toString()
 */
@Override
public String toString() {
    return "Persona [apellido=" + apellido + ", nombre=" +
        nombre + ", edad=" + edad + "];";
}
} //Fin Persona

```

```

public class ListaPersonas {

    private Persona []lista;
    private int cantidadPersonas;
    private int max;

    public ListaPersonas(int max) {
        lista = new Persona [max];
        cantidadPersonas = 0;
        this.max = max;
    }
}

```

En el método ingresarPersona, es mejor traspasar el objeto que recibir sus atributos y crearlo. Lo anterior, debido a que si hay un cambio en la persona habría que intervenir este código.

```

public boolean ingresarPersona(Persona persona) {
    if (cantidadPersonas < max) {
        lista[cantidadPersonas] = persona;
        cantidadPersonas ++;
        return true;
    }
    else {
        return false;
    }
}

```

```

public int getCantidadPersonas() {
    return cantidadPersonas;
}

```

```

public Persona getPersonaI(int i) {
    if (i >= 0 && i < cantidadPersonas) {
        return lista[i];
    }
    else {
        return null;
    }
}

```

```

public Persona buscarPersona(String apellido){
    int i;
    for(i = 0; i < cantidadPersonas; i++){
        if (lista[i].getApellido().equals(apellido)){
            break;
        }
    }
    if (i == cantidadPersonas){
        return null;
    }
    else{
        return lista[i];
    }
}

public boolean eliminar(String apellido) {
    // elimina la persona del contenedor
    int j;
    for(j=0; j< cantidadPersonas; j++){
        if(lista[j].getApellido().equals(apellido)) {
            break; //sale del for
        }
    }
    if(j== cantidadPersonas) { // no lo encontré
        return false;
    }
    else { // lo encontré
        for(int k=j;k<cantidadPersonas-1;k++){//corrimiento
            lista[k] = lista[k+1];
        }
        cantidadPersonas --; // decrementa el tamaño
        return true;
    } //end else
} //Fin eliminar

//Se obtiene un string con todos los
//elementos de la lista
public String toString(){
    String r = "";
    for (int i=0;i<cantidadPersonas;i++){
        r=r+lista[i].toString()+"\n";
    }
    return r;
}

} //Fin ListaPersonas

```

Es mejor sobrescribir el método toString que se obtiene desde el Eclipse o NetBeans para un contenedor.

Lo anterior debido a que invoca toString para cada elemento y puede que el contador no esté lleno y existan espacios nulos

```

import ucn.Stdout;
public class ListaPersonasApp {

    public static void desplegarPersonas(ListaPersonas
                                         listaPersonas) {

        // despliega los items
        StdOut.println("Despliegue de las personas");
        for(int i=0; i<listaPersonas.getCantidadPersonas();i++){
            Persona p = listaPersonas.getPersonaI(i);
            StdOut.println(p.getApellido() + " " +
                           p.getNombre() + " " +p.getEdad());
        }
        StdOut.println();
    }

    public static void main(String[] args) {
        int maxSize = 100; // tamaño del contenedor
        ListaPersonas listaPersonas =
                                new ListaPersonas(maxSize);

        //referencia y crea la lista

        // inserta 10 items
        Persona p = new Persona("Martinez", "Jose", 24);
        listaPersonas.ingresarPersona(p);
        Persona p9 = new Persona("Tapia", "Luis", 37);
        listaPersonas.ingresarPersona(p9);
        Persona p1 = new Persona("Viorklumds", "Jorge", 43);
        listaPersonas.ingresarPersona(p1);
        Persona p2 = new Persona("Ferrada", "Cecilia", 63);
        listaPersonas.ingresarPersona(p2);
        Persona p3 = new Persona("Vega", "Carlos", 21);
        listaPersonas.ingresarPersona(p3);
        Persona p4 = new Persona("Salazar", "Rozana", 29);
        listaPersonas.ingresarPersona(p4);
        Persona p5 = new Persona("Galleguillos", "Ingrid", 72);
        listaPersonas.ingresarPersona(p5);
        Persona p6 = new Persona("Alfaro", "Eduardo", 54);
        listaPersonas.ingresarPersona(p6);
        Persona p7 = new Persona("Sanchez", "Yazmin", 22);
        listaPersonas.ingresarPersona(p7);
        Persona p8 = new Persona("Vergara", "Vladimir", 18);
        listaPersonas.ingresarPersona(p8);
    }
}

```

```

desplegarPersonas(listaPersonas);

StdOut.println("Despliegue de las personas");
StdOut.println(listaPersonas.toString());
StdOut.println();

StdOut.println("Buscando a Martinez");
String apellido = "Martinez"; // busca el item
Persona found=listaPersonas.buscarPersona(apellido);
if(found != null) {
    StdOut.print("Encontrado ");
    StdOut.println(found.getApellido() + " " +
        found.getNombre() + " " + found.getEdad());
}
else {
    StdOut.println("No encontrado " + apellido);
}
StdOut.println();

StdOut.println("Eliminando Viorklumds, Vega y Alfaro");
// elimina 3 items
boolean pudoEliminar=listaPersonas.eliminar("Viorklumds");
if (pudoEliminar) {
    StdOut.println("si elimino a Viorklumds");
}
pudoEliminar =listaPersonas.eliminar("Vega");
if (pudoEliminar) {
    StdOut.println("si elimino a Vega");
}
pudoEliminar =listaPersonas.eliminar("Alfaro");
if (pudoEliminar) {
    StdOut.println("si elimino a Alfaro");
}
StdOut.println();

desplegarPersonas(listaPersonas);

StdOut.println("Despliegue de las personas");
StdOut.println(listaPersonas.toString());
StdOut.println();

} //Fin main
} // Fin ListaPersonasApp

```


Se imprime:

```

Despliegue de las personas
Martinez Jose 24
Tapia Luis 37
Viorklumds Jorge 43
Ferrada Cecilia 63
Vega Carlos 21
Salazar Rozana 29
Galleguillos Ingrid 72
Alfaro Eduardo 54
Sanchez Yazmin 22
Vergara Vladimir 18

```

***Despliegue con método estático de la
App desplegarPersonas***

```

Despliegue de las personas
Persona [apellido=Martinez, nombre=Jose, edad=24]
Persona [apellido=Tapia, nombre=Luis, edad=37]
Persona [apellido=Viorklumds, nombre=Jorge, edad=43]
Persona [apellido=Ferrada, nombre=Cecilia, edad=63]
Persona [apellido=Vega, nombre=Carlos, edad=21]
Persona [apellido=Salazar, nombre=Rozana, edad=29]
Persona [apellido=Galleguillos, nombre=Ingrid, edad=72]
Persona [apellido=Alfaro, nombre=Eduardo, edad=54]
Persona [apellido=Sanchez, nombre=Yazmin, edad=22]
Persona [apellido=Vergara, nombre=Vladimir, edad=18]

```

***Despliegue
utilizando en
StdOut.println
el método
toString de la
clase
ListaPersonas***

```

Buscando a Martinez
Encontrado Martinez Jose 24

```

```

Eliminando Viorklumds, Vega y Alfaro
si elimino a Viorklumds
si elimino a Vega
si elimino a Alfaro

```

```

Despliegue de las personas
Martinez Jose 24
Tapia Luis 37
Ferrada Cecilia 63
Salazar Rozana 29
Galleguillos Ingrid 72
Sanchez Yazmin 22
Vergara Vladimir 18

```

```

Despliegue de las personas
Persona [apellido=Martinez, nombre=Jose, edad=24]
Persona [apellido=Tapia, nombre=Luis, edad=37]
Persona [apellido=Ferrada, nombre=Cecilia, edad=63]
Persona [apellido=Salazar, nombre=Rozana, edad=29]
Persona [apellido=Galleguillos, nombre=Ingrid, edad=72]
Persona [apellido=Sanchez, nombre=Yazmin, edad=22]
Persona [apellido=Vergara, nombre=Vladimir, edad=18]

```

Supongamos que en la creación de los objetos Persona se usara la misma referencia p.

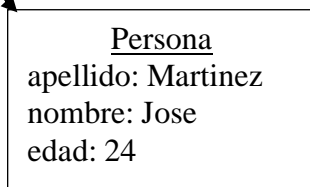
```
// inserta 10 items
Persona p = new Persona("Martinez", "Jose", 24);
listaPersonas.ingresarPersona(p);
p = new Persona("Tapia", "Luis", 37);
listaPersonas.ingresarPersona(p);
p = new Persona("Viorklumds", "Jorge", 43);
listaPersonas.ingresarPersona(p);
p = new Persona("Ferrada", "Cecilia", 63);
listaPersonas.ingresarPersona(p);
p = new Persona("Vega", "Carlos", 21);
listaPersonas.ingresarPersona(p);
p = new Persona("Salazar", "Rozana", 29);
listaPersonas.ingresarPersona(p);
p = new Persona("Galleguillos", "Ingrid", 72);
listaPersonas.ingresarPersona(p);
p = new Persona("Alfaro", "Eduardo", 54);
listaPersonas.ingresarPersona(p);
p = new Persona("Sanchez", "Yazmin", 22);
listaPersonas.ingresarPersona(p);
p = new Persona("Vergara", "Vladimir", 18);
listaPersonas.ingresarPersona(p);
```

listaPersonas



Con:

```
Persona p = new Persona("Martinez", "Jose", 24);
listaPersonas.ingresarPersona(p);
```



p

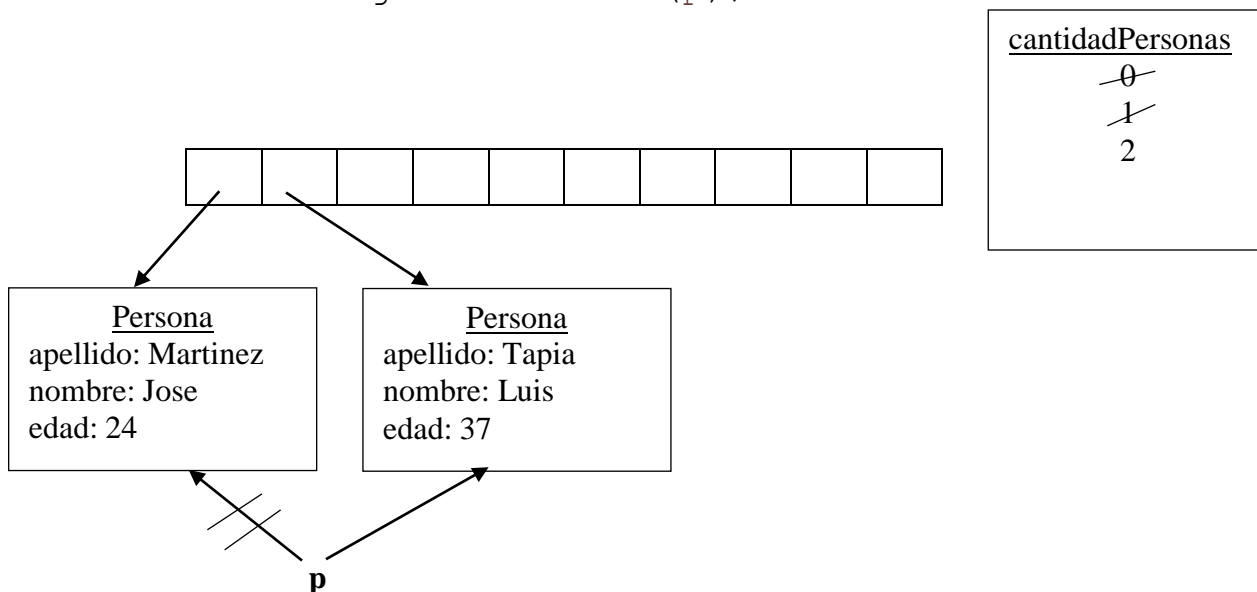
cantidadPersonas

~~0~~
1

`p = listaPersonas[0]`

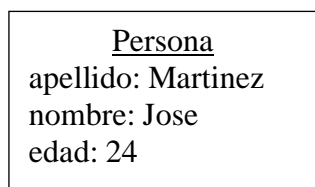
Con:

```
p = new Persona("Tapia", "Luis", 37);
listaPersonas.ingresarPersona(p);
```



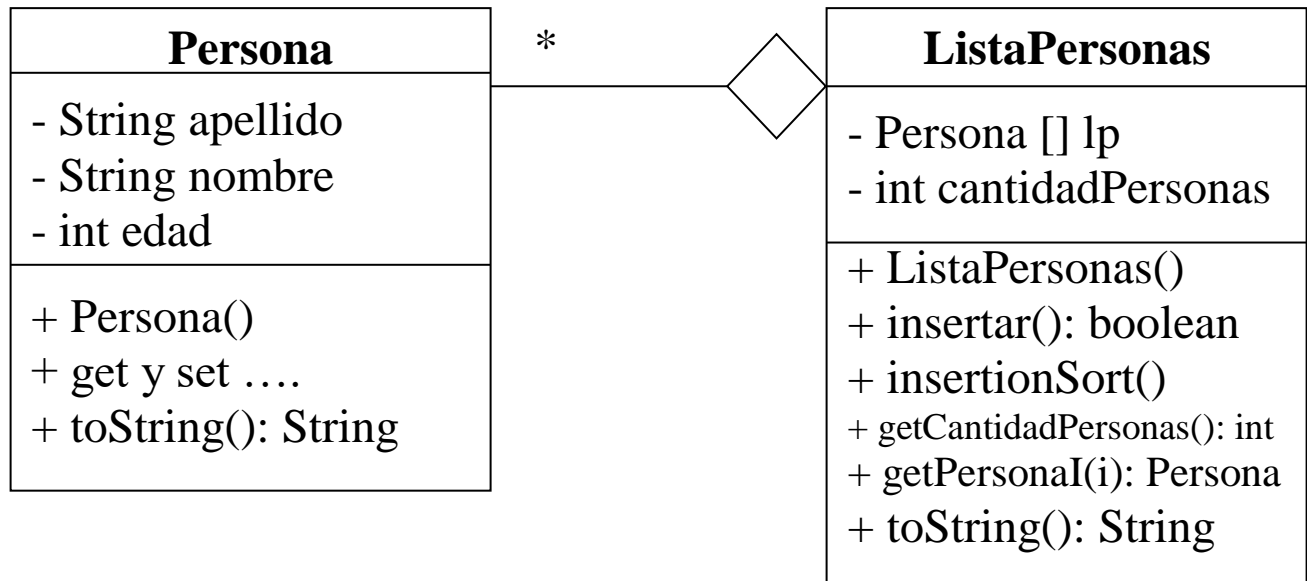
`p ≠ listaPersonas[0]` `p = listaPersonas[1]`

El objeto



no es basura, ya que se puede llegar a él a través de `listaPersonas[0]`

2.5.2. Ordenamiento de un contenedor de objetos



```

public class Persona {
    private String apellido;
    private String nombre;
    private int edad;

    public Persona(String apellido, String nombre, int edad) {//constructor
        this.apellido = apellido;
        this.nombre = nombre;
        this.edad = edad
    }

    public String getApellido(){ // obtiene el apellido
        return apellido;
    }

    // get y set ....

    // toString()....

}//fin clase Persona
  
```

```

public class ListaPersonas {
    private Persona[] lista; //referencia al arreglo
    private int cantidadPersonas; //número de ítems
    private int max;

    public ListaPersonas(int max) { //constructor
        lista = new Persona[max]; //crea el arreglo
        cantidadPersonas = 0; //no hay ítems todavía
        this.max = max;
    }

    public boolean insertar(Persona persona){
        if (cantidadPersonas < max){
            lista[cantidadPersonas] = persona;
            cantidadPersonas++; //incrementa el tamaño
            return true;
        }
        else{
            return false;
        }
    }

    public int getCantidadPersonas(){
        return cantidadPersonas;
    }

    public Persona getPersonaI(int i){
        if (i >= 0 && i < cantidadPersonas){
            return lista[i];
        }
        return null;
    }
}

```

```

public void insertionSort(){
    int in, out;
    for(out=1; out<cantidadPersonas; out++) {
        Persona temp = lista[out];
        in = out;
        while(in>0 &&
            lista[in-1].getApellido().compareTo(temp.getApellido ())>0){
            lista[in] = lista[in-1];
            --in;
        }
        lista[in] = temp;
    } //fin del for;
} //fin insertionSort()

//toString()....

} //fin clase ListaPersonas

```

```

public class ListaPersonasApp {

    public static void main(String[] args){

        int max = 100; //tamaño del contenedor
        ListaPersonas listaPersonas; //referencia al contenedor

        //Crea el contenedor
        listaPersonas = new ListaPersonas(max) ;

        listaPersonas.insertar(new Persona("Martínez", "José", 24));
        listaPersonas.insertar(new Persona("Tapia", "Luis", 37));
        listaPersonas.insertar(new Persona("Viorklumds", "Jorge", 43));
        listaPersonas.insertar(new Persona("Ferrada", "Cecilia", 63));
        listaPersonas.insertar(new Persona("Salazar", "Rozana", 29));
    }
}

```

```

Persona persona = new Persona("Galleguillos","Ingrid",72);
listaPersonas.insertar(persona);
Persona persona1 = new Persona("Alfaro","Eduardo",54);
listaPersonas.insertar(persona1);
Persona persona2 = new Persona("Sanchez", "Yazmin", 22);
listaPersonas.insertar(persona2);
Persona persona3 = new Persona("Vergara","Vladimir",18);
listaPersonas.insertar(persona3);

```

```

StdOut.println("Antes de ordenar:") ;
desplegarPersonas(listaPersonas);

listaPersonas.insertionSort();

StdOut.println("Despues de ordenar:") ;
desplegarPersonas(listaPersonas);

```

Se podría desplegar la lista de personas usando el método toString(), como en el caso del ejemplo anterior

```

} //fin main()

```

```

public static void desplegarPersonas (ListaPersonas listaPersonas){
    //Despliega las personas
    for(int i = 0; i < listaPersonas.getCantidadPersonas(); i++){
        Persona persona = listaPersonas.getPersonaI(i);
        StdOut.println(persona.getApellido());
    }
}

```

```

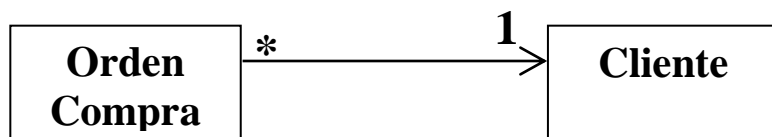
} //fin clase ListaPersonasApp

```

2.6. Navegabilidad

2.6.1 Concepto de navegabilidad

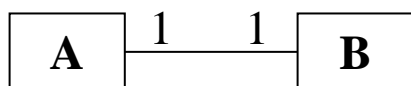
Se pueden agregar flechas en los relacionamientos para indicar navegabilidad.



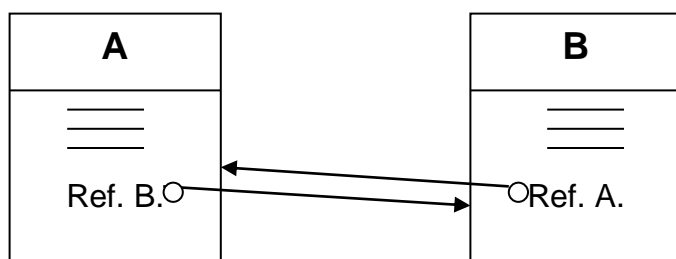
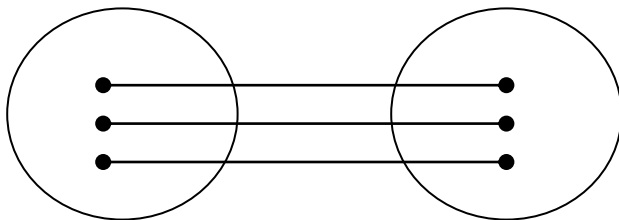
- A partir de la orden de compra, se puede conocer el cliente asociado.
- A partir de un cliente **no** se puede conocer las ordenes de compra que tiene asociadas.

La responsabilidad no es simétrica, existe sólo responsabilidad a un lado de la línea.

2.6.2. Implementación de asociaciones 1:1



Ejemplo: Asociación entre Universidad y Rector.



Ejercicio

Un médico tiene rut, nombre, registro médico y especialidad. Un paciente tiene rut, nombre, diagnóstico y médico de cabecera (sólo 1).

La aplicación debe realizar lo siguiente:

- a) Ingresar un médico
- b) Ingresar un paciente (sólo rut y nombre)
- c) Ingresar el diagnóstico de un paciente
- d) Asignar un médico a un paciente

Se pide: Modelo del dominio, diagrama de clases del dominio de la aplicación y código

Modelo del dominio

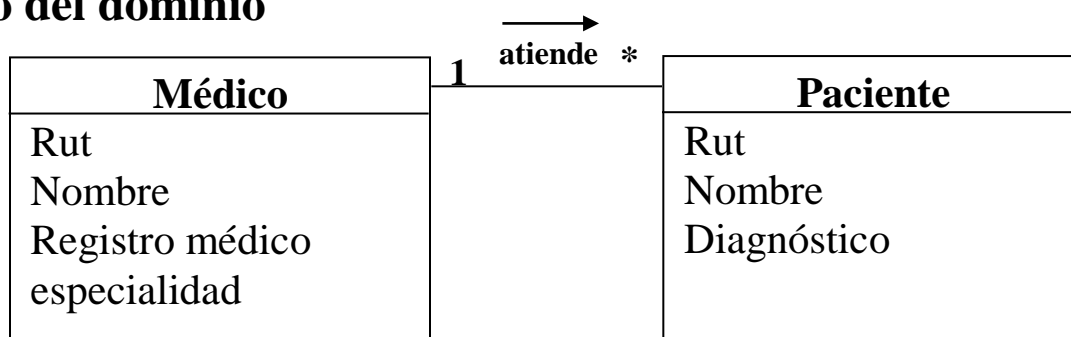
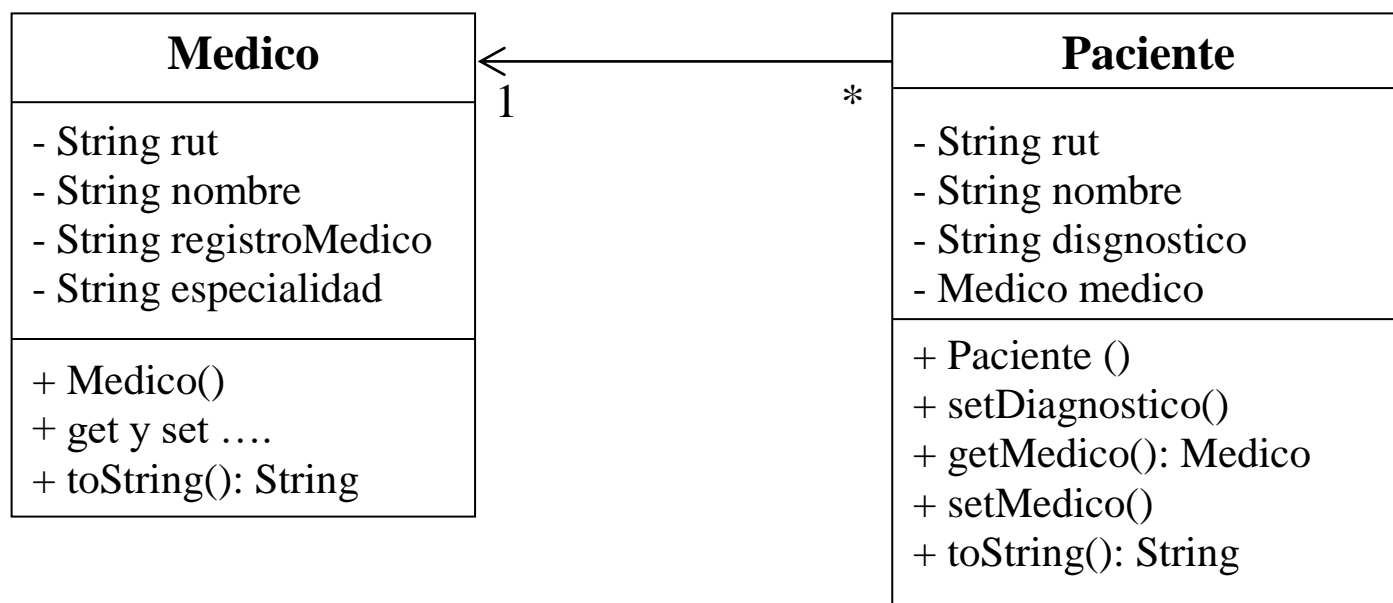


Diagrama de Clases del dominio de la aplicación



Refino de la solución

```

main( ) {
    Crear el objeto médico
    Crear el objeto paciente
    Ingresar diagnóstico del objeto paciente
    Asociar el objeto médico al objeto paciente
}

```

```

public class Medico {
    private String rut;
    private String nombre;
    private String registroMedico;
    private String especialidad;

    public Medico(String rut, String nombre, String registro, String especialidad){
        this.rut = rut;
        this.nombre = nombre;
        this.registroMedico= registro;
        this.especialidad = especialidad;
    }
    //get y set ....
    //toString()...
} //Fin clase Medico

```

```

public class Paciente {
    private String rut;
    private String nombre;
    private String diagnostico;
    private Medico medico;

    public Paciente (String rut, String nombre){
        this.rut =rut;
        this.nombre = nombre;
        medico = null;
        diagnostico = null;
    }

```

```

public void setDiagnostico(String diagnostico){
    this.diagnostico = diagnostico;
}

public void setMedico(Medico m){ //asociaMedico()
    medico = m;
}

public Medico getMedico (){
    return medico;
}

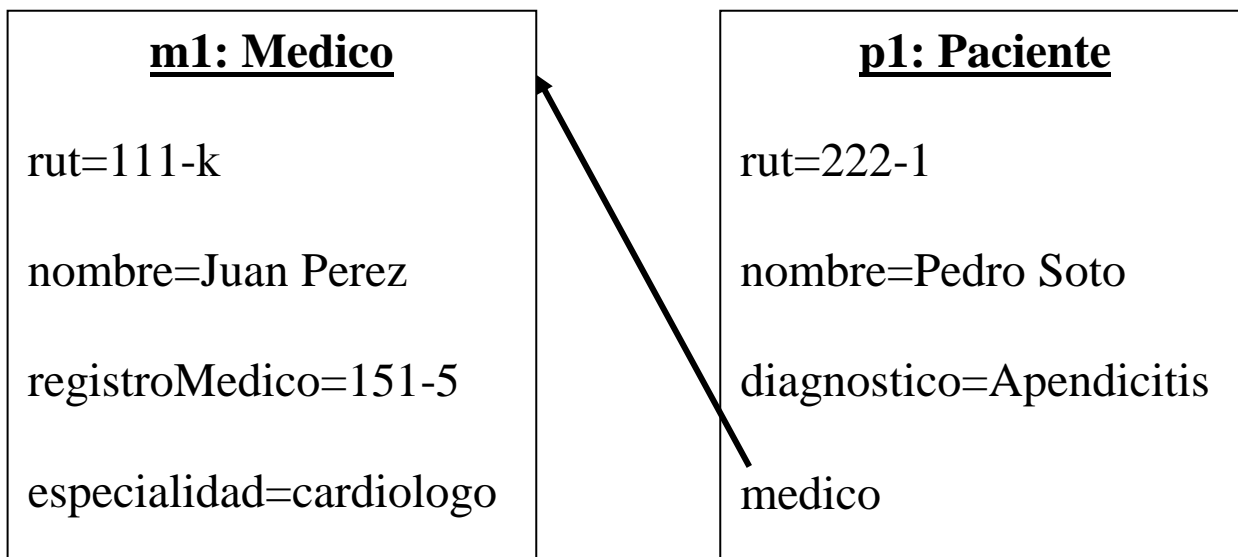
//get y set .....
//toString() ....
} //Fin clase Paciente

public class App {
    public static void main (String [ ] args) {
        Medico m1 = new Medico("111-k", "Juan Perez",
                                "151-5", "cardiologo");

        Paciente p1 = new Paciente("222-1", "Pedro Soto");
        p1.setDiagnostico("Apendicitis");
        p1.setMedico(m1);
    }
}

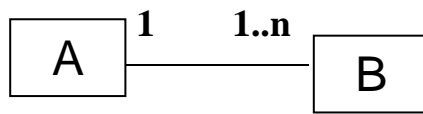
```

Diagrama de Objetos

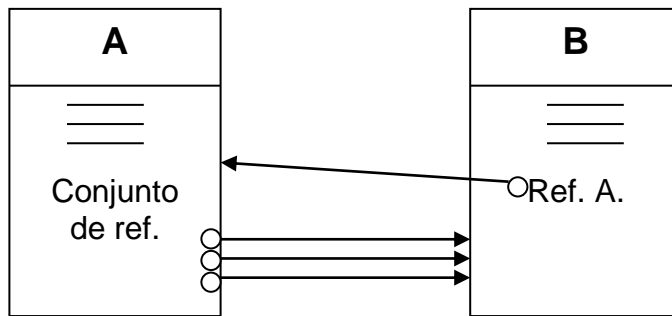


2.6.3. Implementación de una asociación 1:N y/o N:N

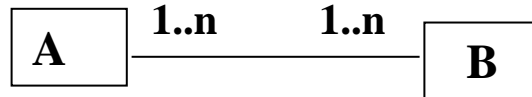
Asociación 1: N



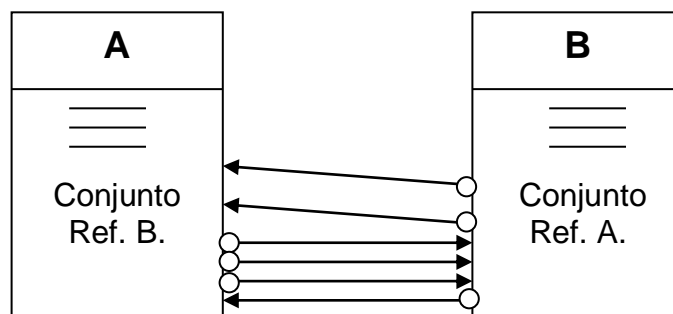
Ejemplo: Asociación entre Departamento y Profesor.



Asociación N: N



Ejemplo: Asociación entre Producto y Cliente.



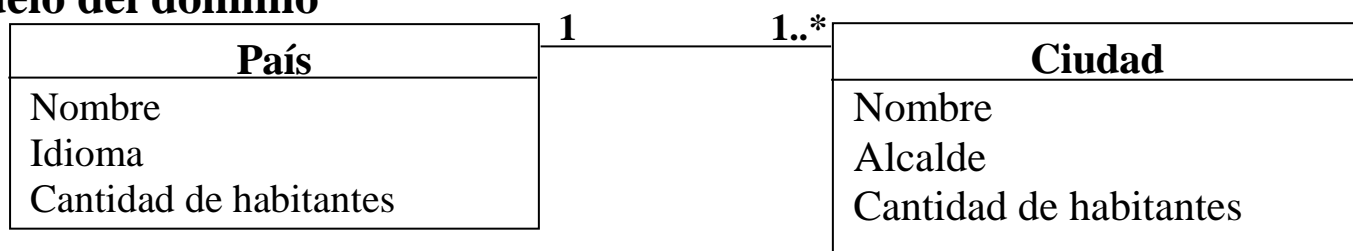
Para implementar estas asociaciones se usan los contenedores.

Ejercicio

Se necesita manejar información de un país y sus ciudades. Para un país interesa saber su nombre, idioma y cantidad de habitantes. Para una ciudad, su nombre, alcalde (suponga que sólo 1 alcalde por ciudad) y cantidad de habitantes. Se pide que haga un programa en Java que:

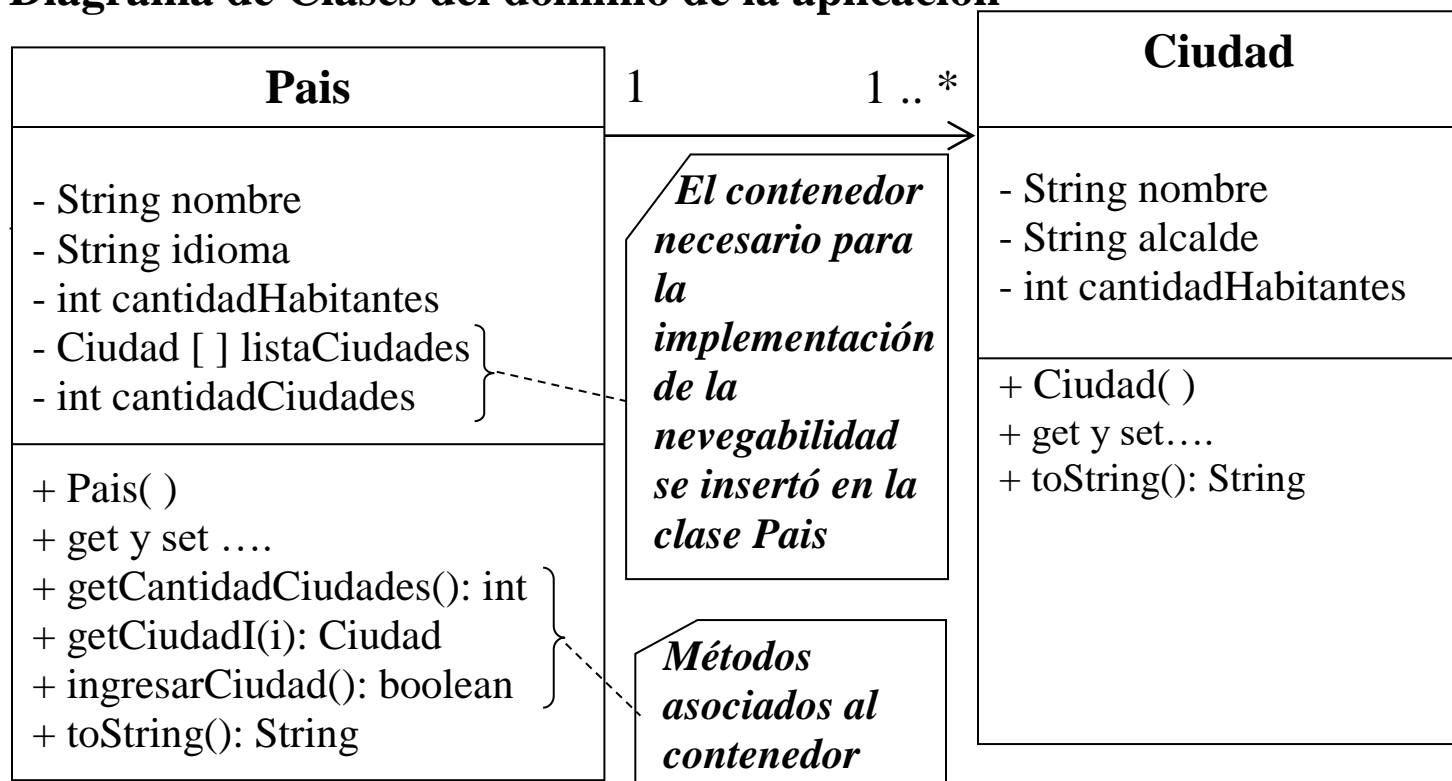
- Ingrese el país Chile. Suponga que tiene 18.000.000 de habitantes.
- Ingrese N ciudades de Chile, donde N se lee desde pantalla (lea desde pantalla los datos de cada ciudad)
- Una vez ingresada la información del país y sus ciudades, despliegue el nombre del país y el nombre de cada una de sus ciudades.

Modelo del dominio



Se debe navegar de país a sus ciudades

Diagrama de Clases del dominio de la aplicación



Refino de la solución

```

main( ) {
    Crear el objeto país (p)
    Leer N
    for i = 1 to N
        Crear el objeto ciudad (c)
        Asociar el objeto ciudad al objeto país (de país a ciudad)
    end for
    “Desplegar el nombre del país y el nombre de sus ciudades”
}

```

“Desplegar el nombre del país y sus ciudades”

```

desplegar nombre del país
Obtener la cantidad de ciudades del país (N)
Para cada ciudad (for I = 0 to N – 1):
    Obtener la ciudad
    obtener su nombre
    desplegar el nombre

```

```

public class Ciudad {
    private String nombre;
    private String alcalde;
    private int cantidadHabitantes;

    public Ciudad (String nombre, String alcalde, int cantidadHabitantes) {
        this.nombre = nombre;
        this.alcalde = alcalde;
        this.cantidadHabitantes = cantidadHabitantes;
    }

    public String getNombre ( ) {
        return nombre;
    }

    //get y set .....
    //toString()...
}

```

```

public class Pais {
    private String nombre;
    private String idioma;
    private int cantidadHabitantes;
    private Ciudad [ ] listaCiudades;
    private int cantidadCiudades;
    private int max

    public Pais (String nombre, String, idioma, int cantidadHabitantes,
                                                         int max) {

        this.nombre = nombre;
        this.idioma= idioma;
        this.cantidadHabitantes = cantidadHabitantes;
        listaCiudades = new Ciudad[max];
        cantidadCiudades = 0;
        this.max = max;
    }

    public boolean ingresarCiudad (Ciudad ciudad) {
        if (cantidadCiudades < max) {
            listaCiudades [cantCiudades] = ciudad;
            cantidadCiudades ++;
            return true;
        }
        return false;
    }

    public String getNombre ( ) {
        return nombre;
    }

    public int getCantidadCiudades( ) {
        return cantidadCiudades;
    }
}

```

```

public Ciudad getCiudadI (int i ) {
    if (i >= 0 && i < cantidadCiudades){
        return refCiudades[i];
    }
    else{
        return null;
    }
}

```

```

//toString()

```

```

} //Fin clase Pais

```

*Debieran haber subprogramas
(métodos estáticos de la App) para
le lectura y despliegue de los
datos*

```

public class App {

```

```

public static void main(String [ ] args) {
    StdOut.print("Ingrese cantidad de ciudades de Chile");
    int cantidadCiudades = StdIn.readInt();

    Pais pais = new Pais ("Chile", "Español",
                          18000000, cantidadCiudades);

    for (int I = 1; I <= cantidadCiudades; I++) {
        StdOut.print("Ingrese nombre ciudad");
        String nombre = StdIn.readString();
        StdOut.print("Ingrese Alcalde");
        String alcalde = StdIn.readString();
        StdOut.print("Ingrese cantidad de habitantes");
        int cantidadHabitantes = StdIn.readInt();
        Ciudad ciudad = new Ciudad(nombre, alcalde, cantidadHabitantes);
        pais.ingresarCiudad(ciudad);
    }
}

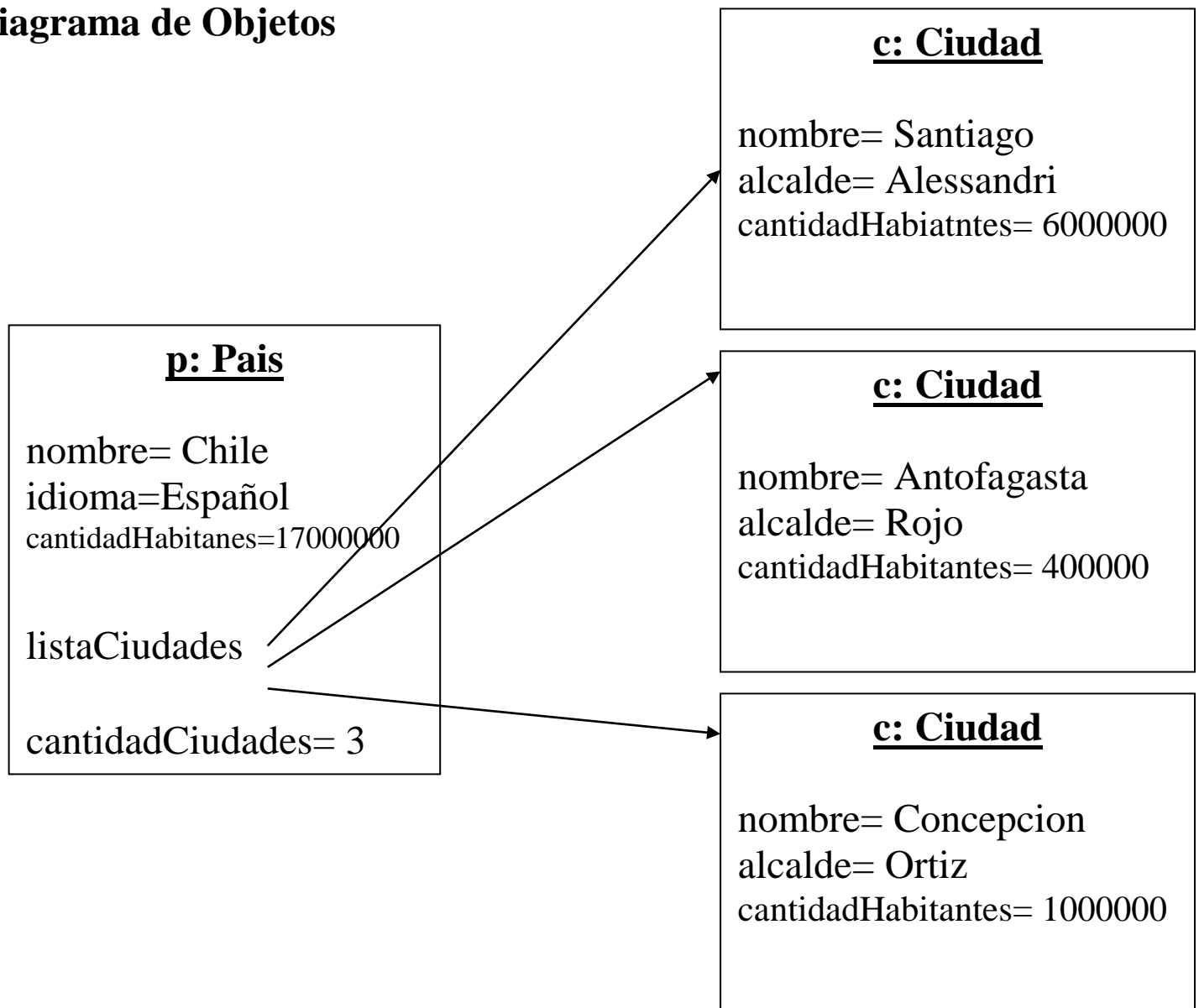
```


// Desplegar datos en pantalla del país y sus ciudades

```
System.out.println(pais.getNombre ( ));
cantidadCiudades = pais.getCantidadCiudades();
for (i = 0; i < cantidadCiudades ; i++) {
    Ciudad ciudad = pais.getCiudadI(i);
    StdOut.println(ciudad.getNombreC( ));
}
}
```

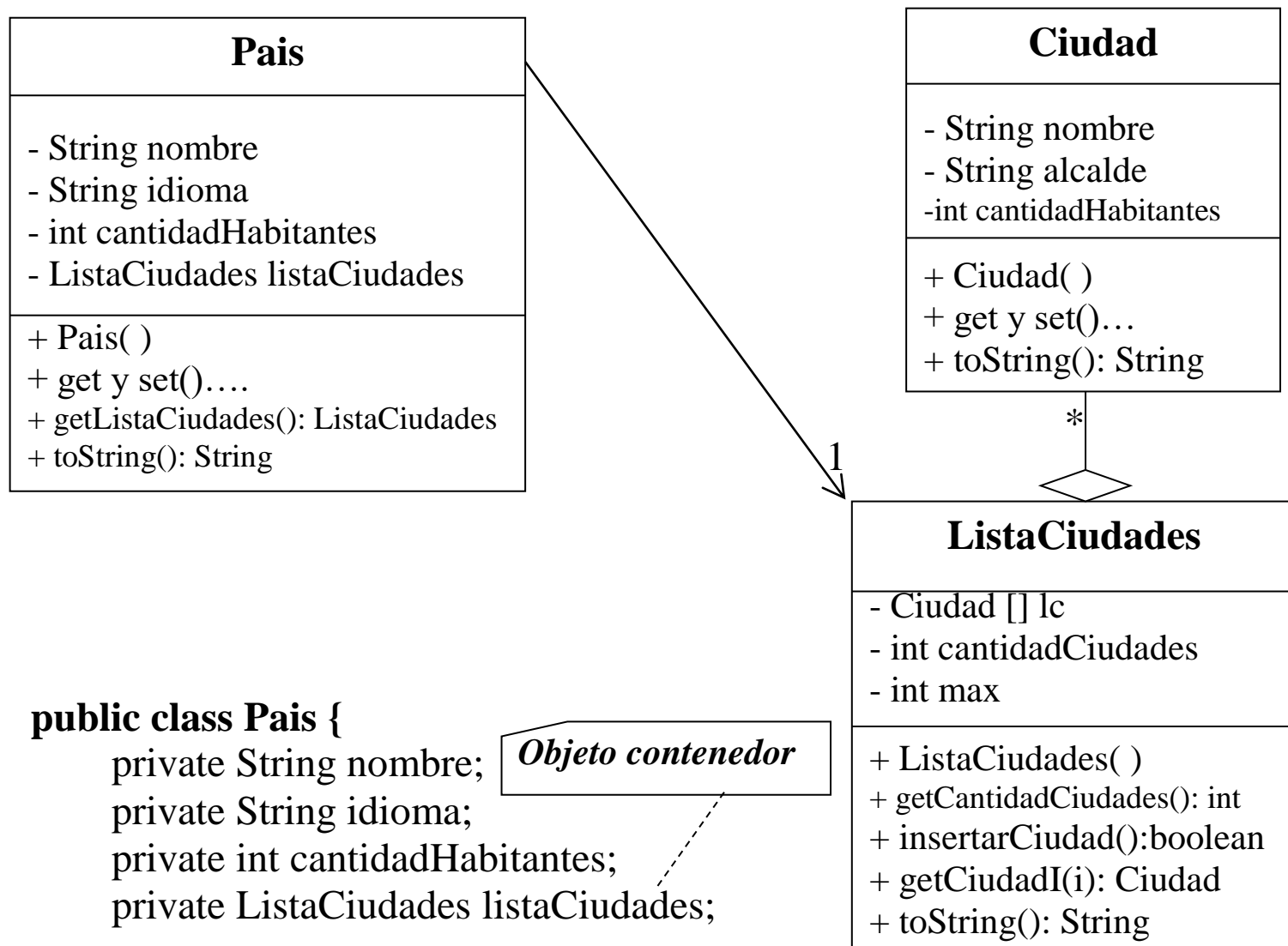
Se podría desplegar la información del país y sus ciudades, mediante el método to String()

Diagrama de Objetos



Otra versión del mismo problema

Diagrama de Clases del dominio de la aplicación



```

public class Pais {
    private String nombre;
    private String idioma;
    private int cantidadHabitantes;
    private ListaCiudades listaCiudades;

```

Objeto contenedor

```

    public Pais (String nom, String id, int cant, int max) {
        nombre = nom;
        idioma= id;
        cantidadHabitantes = cant;
        listaCiudades = new ListaCiudades(max);
    }

```

```

    public int getCantidadHabitantes() {
        return cantidadHabitantes;
    }

```

```

public String getIdioma() {
    return idioma;
}
public String getNombre () {
    return nombre;
}
public ListaCiudades getListasCiudades() {
    return listaCiudades;
}
//toString().....
}

public class Ciudad {
    private String nombre;
    private String alcalde;
    private int cantidadHabitantes;

    public Ciudad (String nom, String alc, int cant) {
        nombre = nom;
        alcalde= alc;
        cantidadHabitantes = cant;
    }
    public String getAlcalde() {
        return alcalde;
    }
    public int getCantidadHabitantes() {
        return cantidadHabitantes;
    }
    public String getNombre () {
        return nombre;
    }
    //toString()....
}

```

```

public class ListaCiudades {
    private Ciudad [] lc;
    private int cantidadCiudades;
    private int max;

    public ListaCiudades (int max) {
        lc = new Ciudad [max];
        cantidadCiudades = 0;
        this.max = max;
    }

    public int getCantidadCiudades() {
        return cantCiudades;
    }

    public Ciudad getCiudadI(int i) {
        if (i >= 0 && i < cantidadCiudades){
            return lc[i];
        }
        else{
            return null;
        }
    }

    public boolean insertarCiudad(Ciudad ciudad) {
        if (cantidadCiudades < max){
            lc[cantidadCiudades] = ciudad;
            cantidadCiudades ++;
            return true ;
        }
        else{
            return false ;
        }
    }
    //toString()...
}

```

Debieran haber subprogramas (métodos estáticos de la App) para le lectura y despliegue de los datos

public class App {

public static void main(String[] args) {

StdOut.print ("Ingrese cantidad de ciudades de Chile");

int cantidadCiudades = StdIn.readInt();

Pais pais = new Pais ("Chile", "Español",
18000000, cantidadCiudades);

ListaCiudades listaCiudadesPais = pais.getListCiudades();

for (int I = 1; I <= N; I++) {

StdOut.print("Ingrese nombre ciudad");

String nom = StdIn.readString();

StdOut.print("Ingrese Alcalde");

String alcalde = StdIn.readString();

StdOut.print("Ingrese cantidad de habitantes");

int cant = StdIn.readInt();

Ciudad ciudad = new Ciudad(nom, alcalde, cant);

listaCiudadesPais.insertarCiudad(ciudad);

//pais.getListCiudades().insertarCiudad(ciudad);

}

// Desplegar datos en pantalla del país y sus ciudades

StdOut.println(pais.getNombre ());

listaCiudadesPais = pais.getListCiudades();

cantidadCiudades = listaCiudadesPais.getCantidadCiudades();

for (int i = 0; i < N ; i++) {

Ciudad ciudad = listaCiudadesPais.getCiudadI(i);

StdOut.println(ciudad.getNombre ());

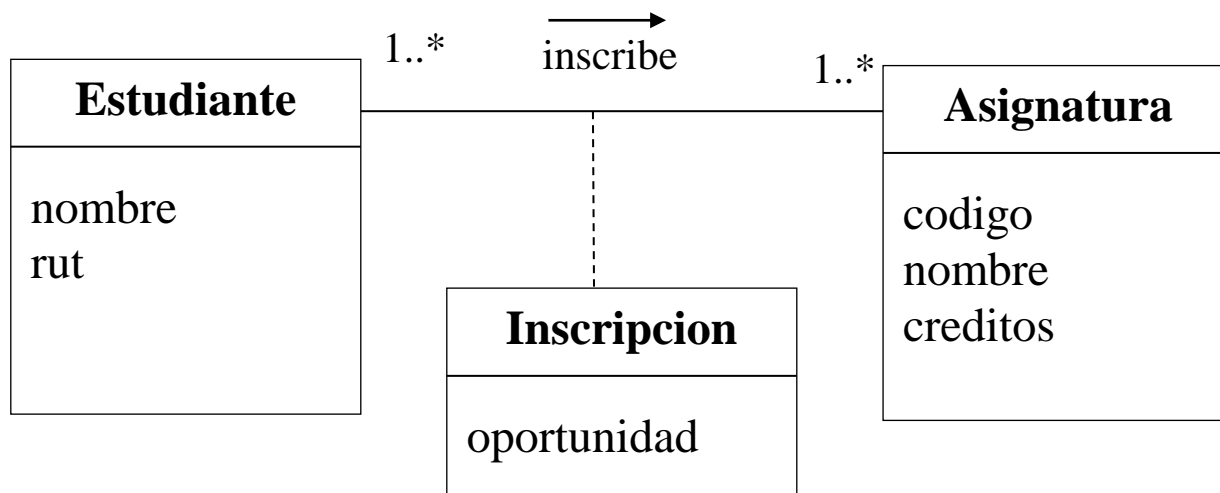
}

//Fin main

} //Fin App

Modelo del dominio y diagrama de clases del dominio de la aplicación asociado

Dado el modelo del dominio visto anteriormente, se muestra el diagrama de clases asociado



Otra forma

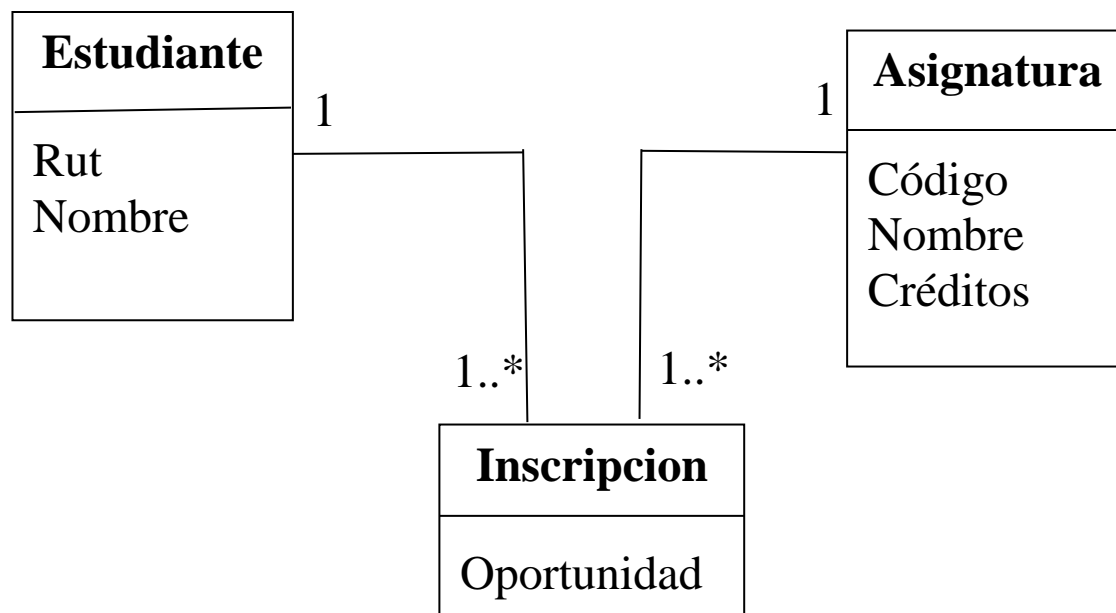
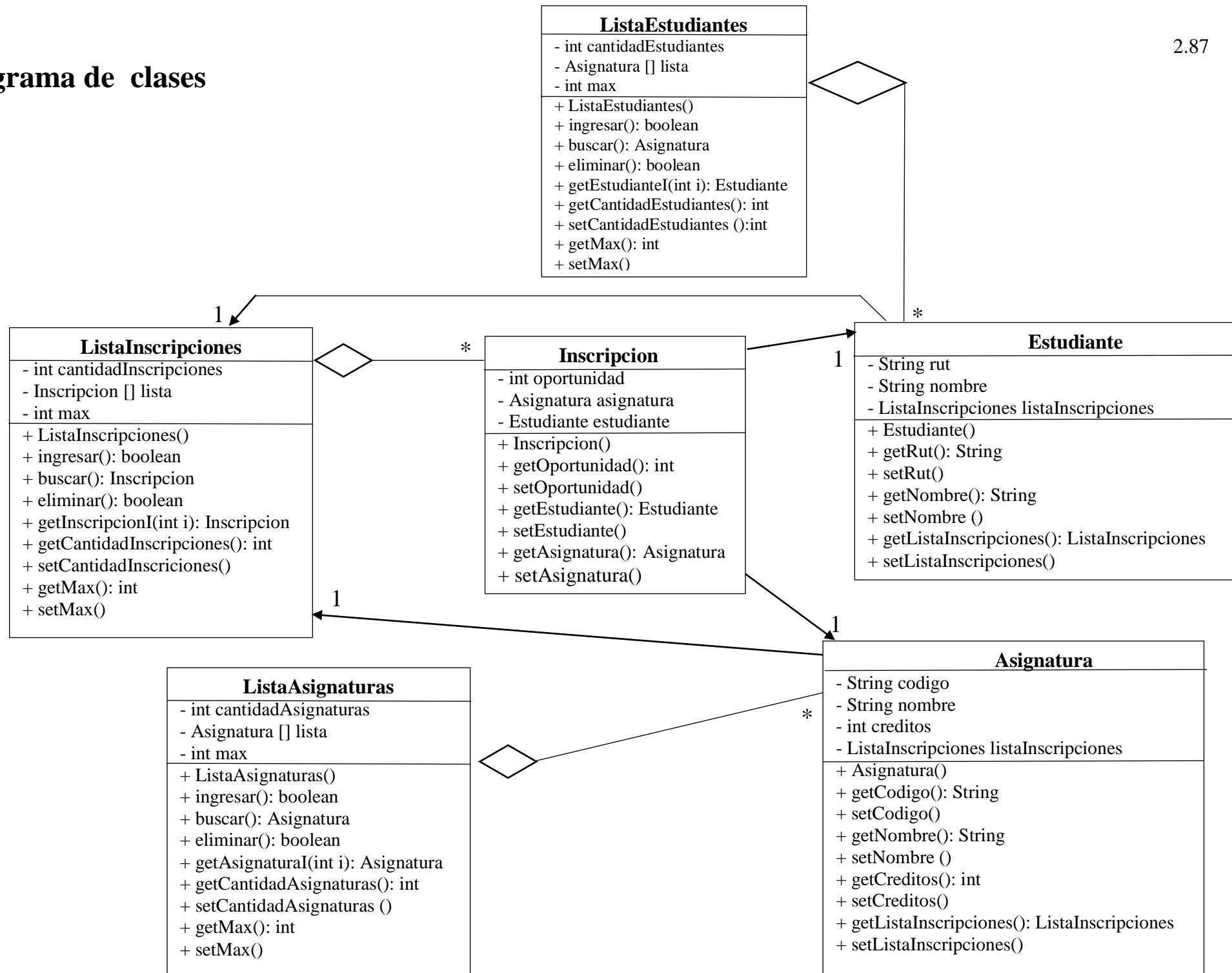


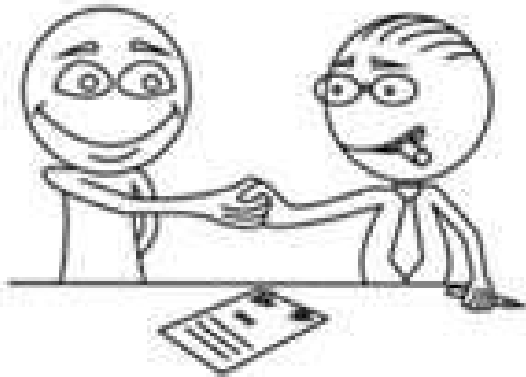
Diagrama de clases



2.7. Arquitectura de una aplicación

2.7.1. Contratos

- ☐ Luego de realizar el modelo de dominio, el siguiente paso es crear los contratos.
- ☐ Para crear los contratos se deben identificar todas las operaciones del sistema y asignar estas operaciones a las entidades correspondientes.
- ☐ Un contrato es una operación atómica del sistema, se hace o no se hace
- ☐ El diseño por contrato establece los derechos y responsabilidades para cada elemento que conforma un sistema.



Los contratos son:

- *Para el cliente, de manera que él tenga claro lo que hará su aplicación*
- *Para el diseñador y/o programador de manera de tener claro lo que hace y no hace la aplicación*

- ☐ Un contrato define lo que se desea lograr con una operación
- ☐ Describe lo **qué** sucederá y no **cómo**.
- ☐ Definen los pre, post condiciones y aspectos invariantes del sistema.
- ☐ En un contrato no va la visibilidad
- ☐ Para cada operación del sistema se redacta un contrato.

Beneficio de los Contratos

- Mejora la documentación, lo que implica mejor calidad

Estructura de un contrato

Operación	Nombre de la operación y parámetros de entrada y salida
Descripción	Breve descripción de la operación
Precondiciones	<p>Suposiciones relevantes sobre el estado del sistema o de los objeto o conceptos del modelo de dominio antes de la ejecución de la operación.</p> <p>Son las condiciones que tiene que cumplir el sistema, no la lógica de la aplicación</p> <p><i>Si no existe o no se cumple la precondición, el contrato no es válido (lanzar un error)</i></p>
Postcondiciones	<p>Resultado esperado de la operación y estado de los objetos o conceptos del dominio después de que se complete la operación.</p> <ul style="list-style-type: none"> • <i>Si solo se obtiene información, no hay post condición</i> • <i>Si se ingresa, actualiza o elimina información, si hay post condición</i>

Ejemplos de Contratos

Operación	float sqrt(float número)
Descripción	Calcula la raíz cuadrada de número
Precondiciones	El valor del número debe ser mayor o igual a 0.
Postcondiciones	<p>El resultado de la operación contemplará un margen de error de a lo mucho de un 0,1%.</p> <p>sqrt(número) implica que el resultado obtenido será un número decimal mayor o igual a 0.</p>

Operación	append (Collection c, Object o)
Descripción	Agrega el objeto o a la colección c
Precondiciones	Las variables c y o deben estar inicializadas, es decir, no deben ser nulas. (c!=null && o != null)
Postcondiciones	Se agrega el objeto o a la colección c. El tamaño de c incrementa en 1.

Contratos Tutores

Operación	Calcular horas()
Descripción	Calcula el número de horas que tiene asignadas un tutor
Precondiciones	El tutor debe haber realizado al menos una actividad de tutela.
Postcondiciones	El valor resultante debe ser mayor o igual a 2.

Operación	Calcular incentivo(horas)
Descripción	Calcula el incentivo que se le da al tutor, en función de las horas asignadas
Precondiciones	El mínimo valor de hora es 2.
Postcondiciones	El valor resultante debe ser un entero positivo.

2.7.2. Interfaces para el diseño

- Java admite un concepto similar a la clase, llamado interfaz.
- Una interfaz es una colección de métodos abstractos. Un método abstracto no tiene código.
- Una interfaz puede ser pública o privada.
- Todos los métodos dentro de la interfaz son públicos y abstractos (no se encuentran implementados).
- Las variables de clase en una interfaz, si es que existen, son implícitamente finales, públicas y estáticas (en otras palabras, son constantes).
- Una interfaz permite al programador describir un conjunto de capacidades que debe implementar una clase.

Ejemplo: Interfaz Collection

interface Collection {

int MAXIMUN = 500;

void add(Object obj);

void delete(Object obj);

Object find(Object obj);

int currentCount (Object obj);

}

class FIFOQueue implements Collection {

.....

void add (Object obj) {

}

void delete (Object obj) {

}

Object find (Object obj) {

}

int currentCount (Object obj){

}

.....

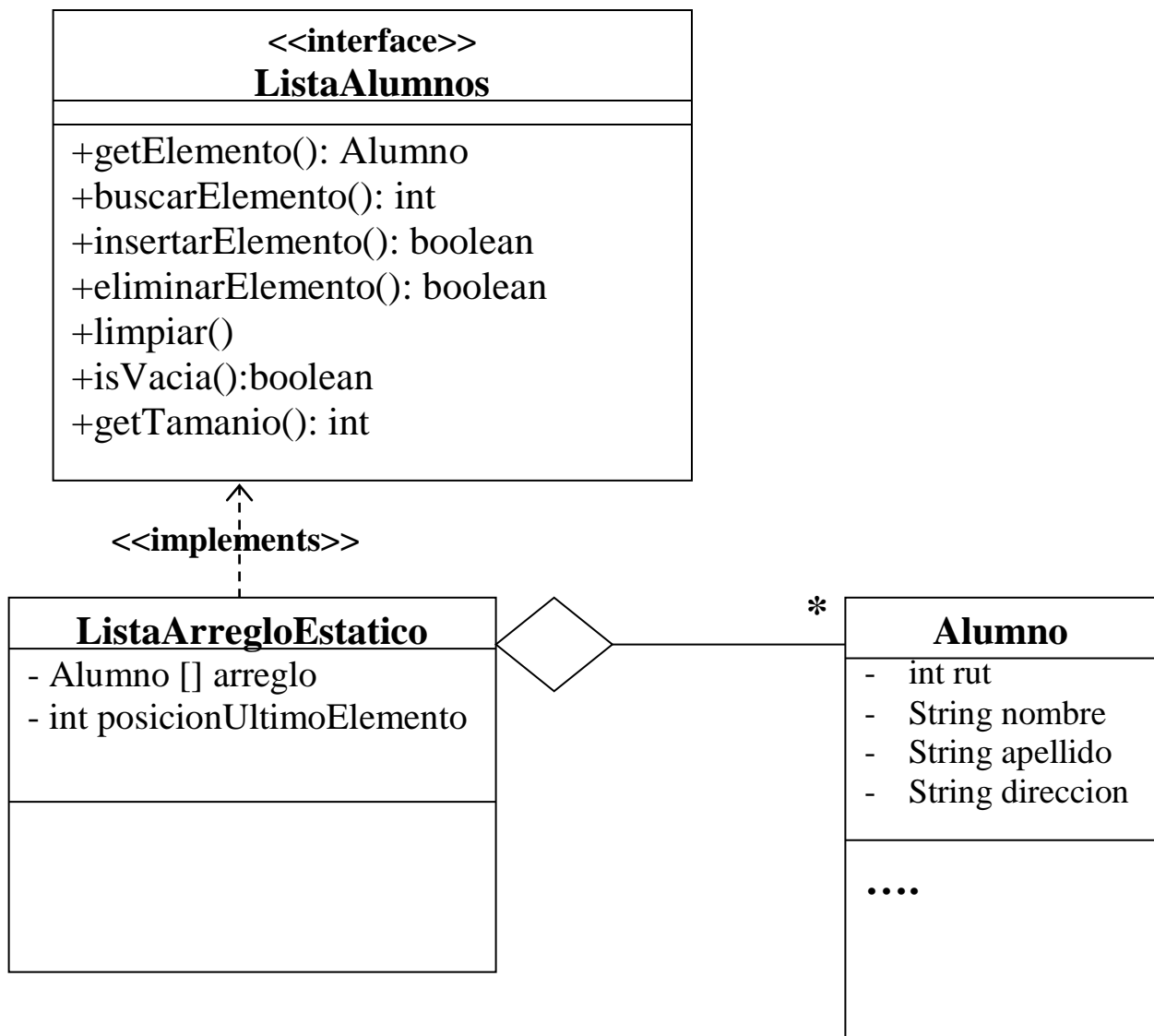
}

*Falta un arreglo o algo
para tener los elementos
de la colección*

- Una clase que implementa una interfaz, **no está heredando nada**. Simplemente, se compromete a implementar los métodos definidos en la interfaz.
- Este es el origen de la palabra clave implements.
- **Polimorfismo en JAVA:** Una variable declarada de una cierta interface, puede mantener valores en cualquier clase, que implementa la interface.

Ejemplo

Diagrama de clases



```

public class Alumno {
    private int rut;
    private String nombre;
    private String apellido;
    private String direccion;

    public Alumno() {
    }

    public String getApellido() {
        return this.apellido;
    }

    public void setApellido(String apellido) {
        this.apellido = apellido;
    }

    public String getDireccion() {
        return this.direccion;
    }

    public void setDireccion(String direccion) {
        this.direccion = direccion;
    }

    public String getNombre() {
        return this.nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public int getRut() {
        return this.rut;
    }

    public void setRut(int rut) {
        this.rut = rut;
    }

    //toString()....
}//Fin Alumno

```

```
/**
 * Interface que representa el comportamiento de una lista de
 * Alumnos. Esta puede ser implementada de la forma que se desee,
 * es decir utilizando:
 * - Arreglos estaticos:
 * - Listas con nexos.
 */
public interface ListaAlumnos {

    /**
     * Metodo que dada la posicion de un elemento dentro de la lista
     * retorna el elemento en esa posicion.
     * Si la posicion en la lista no existe retorna null.
     * @param posicion
     * @return Alumno/null
     */
    public Alumno getElemento(int posicion);
```

```
/**  
 * Metodo que dado un elemento retorna la posicion en la cual se  
 * encuentra dentro de la lista. Retorna -1 si el dato no se encuentra.  
 * @param elemento  
 * @return int posicion del elemento dentro de la lista.  
 */  
public int buscarElemento(Alumno elemento);
```

```
/**  
 * Metodo que inserta un elemento al final de la lista, retorna true si  
 * fue exitosa la insercion (hay espacio para almacenarlo) o false si  
 * la lista se encuentra llena.  
 * @param elemento  
 * @return boolean true/false  
 */  
public boolean insertarElemento(Alumno elemento);
```

```
/**  
 * Metodo que elimina el elemento en la "posicion" de la lista.  
 * Si la posición no existe, retorna falso. En caso contrario  
 * elimina el dato y retorna true.  
 * @param posicion  
 * @return boolean true/false  
 */  
public boolean eliminarElemento(int posicion);
```

```
/**  
 * Metodo que elimina todos los datos de la lista, es decir la  
 * deja vacia  
 */  
public void limpiar();
```



```

/**
 * Metodo que retorna true si y solo si no hay ningún elemento
 * presente en la lista
 * @return boolean (true/false)
 */
public boolean isVacia();

/**
 * Metodo que retorna la cantidad de elementos que posee la lista,
 * por ejemplo: Si la lista se encuentra vacia retorna 0;
 * @return int cantidad de elementos en la lista.
 */
public int getTamanio();

} //Fin interface ListaAlumnos

/**
 * Clase que implementa la interface Lista que permite almacenar
 * Alumnos
 */
public class ListaArregloEstatico implements ListaAlumnos {
    private Alumno[] arreglo;
    private int posicionUltimoElemento;

    /**
     * Constructor de la clase ListaArregloEstatico
     * @param _capacidadMaxima : Cantidad maxima de elementos
     * que puede almacenar esta lista.
     */
    public ListaArregloEstatico(int capacidadMaxima) {
        this.arreglo = new Alumno[capacidadMaxima];
        this.posicionUltimoElemento = -1;
    }

```

```

public Alumno getElemento(int posicion) {
    if (posicion < 0 || posicion > this.arreglo.length) {
        return null;
    }
    // Si no hay un dato en esa posicion
    if (posicion > this.posicionUltimoElemento) {
        return null;
    }
    return this.arreglo[posicion];
}

```

```

public int buscarElemento(Alumno elemento) {
    // Recorro el arreglo tratando de encontrar el elemento
    for (int i=0; i <= this.posicionUltimoElemento; i++) {
        if (this.arreglo[i].getRut()==elemento.getRut()) {
            return i;
        }
    }
    // Si no lo encuentre, retorno -1
    return -1;
}

```

```

public boolean insertarElemento(Alumno elemento) {
    // Si llegue justo a la capacidad maxima no puedo
    // almacenar mas.
    if (this.posicionUltimoElemento == arreglo.length-1) {
        return false;
    }
    // Incremento el contador de elementos e ingreso el elemento.
    this.posicionUltimoElemento++;
    this.arreglo[this.posicionUltimoElemento] = elemento;
    return true;
}

```

```

public void limpiar() {
    int tamano = this.arreglo.length;
    this.arreglo = new Alumno[tamano];
    this.posicionUltimoElemento = -1;
}

```

```

public boolean isVacia() {
    return (this.posicionUltimoElemento == -1);
}

```

```

public int getTamanio() {
    return this.posicionUltimoElemento+1;
}

```

```

public boolean eliminarElemento(int posicion) {
    // Si intento eliminar una posicion que no existe retorno falso
    if (posicion < 0 || posicion > this.arreglo.length) {
        return false;
    }
    // Si no hay un dato en esa posicion
    if (posicion > this.posicionUltimoElemento) {
        return false;
    }
    // Se procede a realizar el corrimiento para realizar la
    // eliminacion
    for (int i = posicion; i < this.posicionUltimoElemento; i++) {
        this.arreglo[i] = this.arreglo[i+1];
    }
    this.posicionUltimoElemento--;
    return true;
}

```

```

} //Fin class ListaArregloEstatico

```

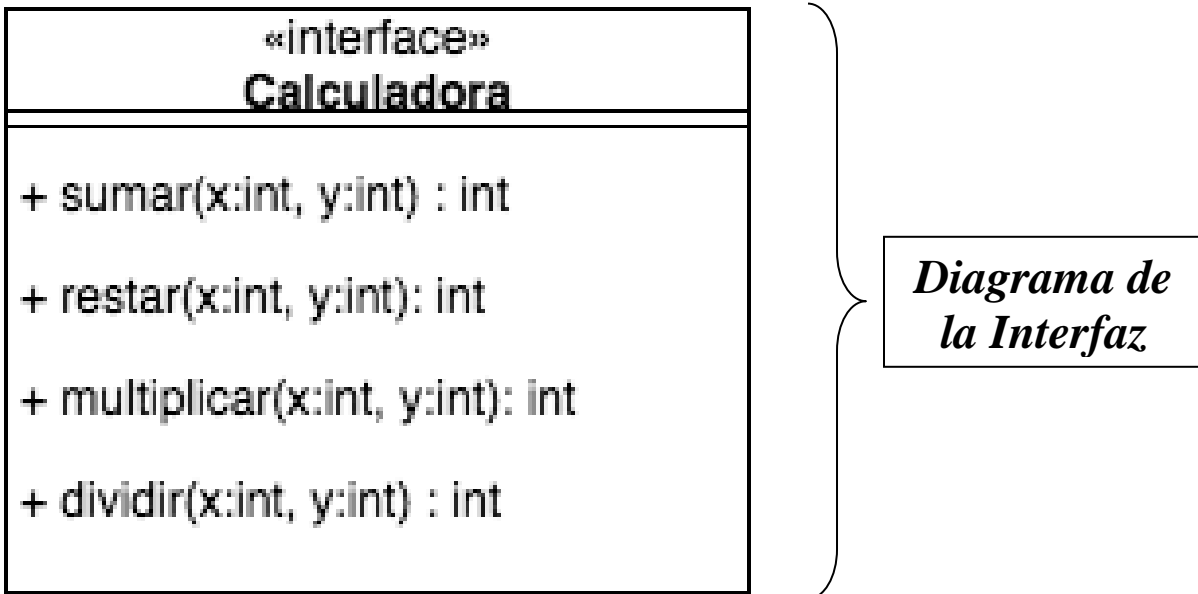
```
public class TestLista {  
  
    public static void main(String[] args) {  
        // Lista que puede almacenar a lo mas 2 objetos.  
        ListaAlumnos lista = new ListaArregloEstatico(2);  
  
        Alumno alumno1 = new Alumno();  
        alumno1.setNombre("Javier");  
        alumno1.setApellido("Mandiola");  
  
        // inserto el alumno1 en la lista  
        if (lista.insertarElemento(alumno1)) {  
            StdOut.println("Alumno 1 insertado con exito");  
        }  
        else {  
            StdOut.println("Alumno 1 NO insertado");  
        }  
  
        Alumno alumno2 = new Alumno();  
        alumno2.setNombre("Claudio");  
        alumno2.setApellido("Vasquez");  
  
        // inserto el alumno2 en la lista  
        if (lista.insertarElemento(alumno2)) {  
            StdOut.println("Alumno 2 insertado con exito");  
        }  
        else {  
            StdOut.println("Alumno 2 NO insertado");  
        }  
  
        Alumno alumno3 = new Alumno();  
        alumno3.setNombre("Paul");  
        alumno3.setApellido("Beltrand");  
    }  
}
```

```
// inserto el alumno3 en la lista, pero como la lista  
//no tiene espacio esta operacion no es exitosa.  
if (lista.insertarElemento(alumno3)) {  
    StdOut.println("Alumno 3 insertado con exito");  
}  
else {  
    StdOut.println("Alumno 3 NO insertado");  
}  
  
//Buscar al alumno 2  
StdOut.println("Alumno 2 encontrado en posicion:" +  
               lista.buscarElemento(alumno2));  
  
// Eliminacion de elementos en la lista  
int k = 0;  
if (lista.eliminarElemento(k)) {  
    StdOut.println("Alumno en posicion " + k + " eliminado!");  
}  
else {  
    StdOut.println("Alumno en posicion " + k + "No existe!!");  
}  
  
} //Fin main  
  
}// Fin class TestLista
```

2.7.3. Implementación de contratos mediante interfaces

Ejemplo Calculadora

Se requiere implementar una calculadora con las 4 operaciones básicas.



Contratos en Java y su documentación

```

/**
 * Conjunto de operaciones básicas que debe realizar una calculadora de números enteros.
 * @author eaguilar
 * @version 2016.03.16
 */
public interface Calculadora {

```

```

/**
 * Suma dos números recibidos por parametros (x,y).
 * Restricciones:
 * pre:
 * Los números x, y debe ser valores enteros en el rango pow(-2,31) y pow(2, 31)-1.
 *
 * pos:
 * El resultado de la suma debe estar en el rango pow(-2,31) y pow(2, 31)-1
 * @param x primer sumando
 * @param y segundo sumando
 * @return suma de x e y.
 */
int sumar(int x, int y);

```

```

/**
 * Resta dos números recibidos por parametro (x, y)
 * Restricciones:
 * pre:
 *     Los números x, y debe ser valores enteros en el rango pow(-2,31) y pow(2, 31)-1.
 * pos:
 *     El resultado de la resta debe estar en el rango pow(-2,31) y pow(2, 31)-1.
 * @param x es el número que se le resta una cierta cantidad.
 * @param y es la cantidad que se resta.
 * @return resta de x e y
 */
int restar(int x, int y);

```

```

/**
 * Multiplica dos números recibidos por parametro (x,y)
 * Restricciones:
 * pre:
 *     Los números x, y deben ser valores enteros en el rango pow(-2,31) y pow(2, 31)-1.
 * pos:
 *     El resultado de la multiplicación debe estar en el rango pow(-2,31) y pow(2, 31)-1.
 * @param x es el multiplicando
 * @param y es el multiplicador
 * @return multiplicacion de x e y
 */
int multiplicar(int x, int y);

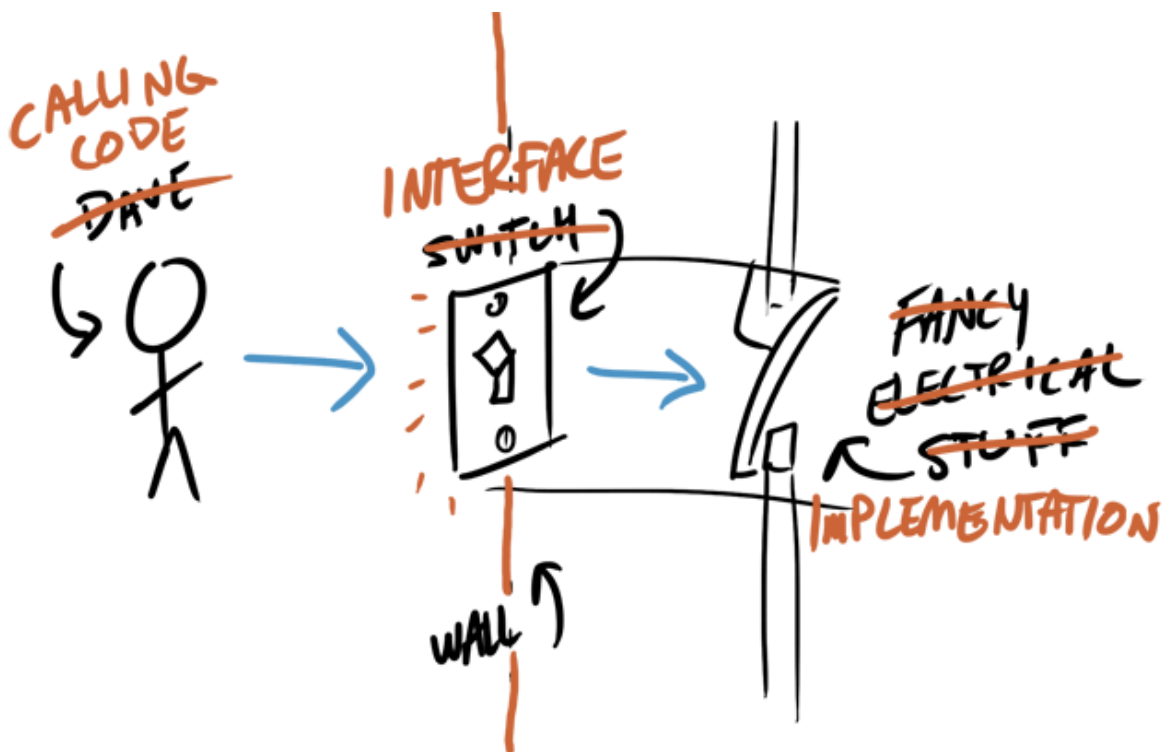
```



```
/**
 * Divide dos números recibidos por parametro (x,y)
 * pre:
 *     Los números x, y deben ser valores enteros en el rango pow(-2,31) y pow(2, 31)-1.
 *     El valor de y debe ser distinto de 0.
 * pos:
 *     El resultado de la división debe estar en el rango pow(-2,31) y pow(2, 31)-1.
 *     Si el resultado de la división es decimal, se tomará el valor entero más cercano. Es decir si la cifra decimal
 *     es mayor o igual a 5 entonces se aproxima al valor superior, en caso contrario se aproxima al valor inferior.
 * @param x es el dividendo
 * @param y es el divisor
 * @return división de x, y
 */
int dividir(int x, int y);
```

Implementación de los contratos

- La implementación de los contratos, implica escribir una **Clase**, la cual debe incluir como mínimo **todos** los sub-programas (métodos) que la interface declara.
- Cuando se trabaja con interfaces, se tiene concentrada en ella todas las operaciones que hará el sistema
- **Un contrato se debe implementar en un método de la interface.**



```
/**
 * Clase que implementa las operaciones básicas de números enteros declarados
 * en el contrato de la interfaz Calculadora.
 * @author eaguilar
 * @version 2016.03.16
 */

public class CalculadoraImpl implements Calculadora {

    @Override
    public int sumar(int x, int y) {
        return x+y;
    }

    @Override
    public int restar(int x, int y) {
        return x-y;
    }

    @Override
    public int multiplicar(int x, int y) {
        return x*y;
    }

    @Override
    public int dividir(int x, int y) {
        if((x%y)*2>=y){
            return (x/y)+1;
        }else{
            return x/y;
        }
    }
}
```

Ejercicio

- Dada la siguiente interfaz, realizar los contratos para cada una de las operaciones.
- Construir una clase que implemente las operaciones de la interfaz.
- Diseñar el diagrama de clases.

<<interface>> BancoUCN	
- \$ int MAX_MONTO = 500	
+ agregarAlumno(nombre: String, rut: String, montoInicial: int): boolean	
+ cargarMonedero(rut: String, clave: String, monto: int): boolean	
+ realizarCompra(rut: String, clave: String, monto: int): boolean	

```
/**
 * Conjunto de funcionalidad que debe realizar
 * el sistema.
 * @author Diego P. Urrutia <durrutia@ucn.cl>
 * @version 2014.03.20
 */

public interface BancoUCN {
    /**
     * Agrega un alumno al sistema con un saldo inicial.
     * Restricciones:
     *     Que no exista el alumno.
     *
     * @param nombre del alumno.
     * @param rut del alumno.
     * @param montoInicial del monedero del alumno.
     * @return true si fue posible agregar el alumno,
     *         falso en otro caso.
     */
    public boolean agregarAlumno(String nombre,
        String rut, String clave, int montoInicial);
}
```

```
/**
 * Agrega un monto al monedero de un alumno.
 * Restricciones:
 *     Que exista el alumno
 *
 * @param rut del alumno.
 * @param monto de ucn a agregar al monedero.
 * @return true si fue posible cargar el monedero.
 */
```

```
public boolean cargarMonedero(String rut,
                               String clave,int monto);
```

```
/**
 * Realiza el pago de una compra por parte del alumno
 * Restricciones:
 *     Que el alumno exista.
 *     La clave debe ser la del alumno.
 *
 * @param rut del alumno.
 * @param clave del alumno.
 * @param monto a descontar del monedero del alumno.
 * @return true si fue posible realizar la compra.
 */
```

```
boolean realizarCompra(String rut, String clave,
                          int monto);
```

```
}
```

Implementación de la interfaz

```

/**
 * Clase que implementa el “como” funciona el sistema,
 * a partir del “que”
 * declarado por los contratos.
 * @author Diego P. Urrutia <durrutia@ucn.cl>
 * @version 2014.03.20
 */

public class BancoUCNImpl implements BancoUCN {

    private ListaAlumnos listaAlumnos;

    /**
     * @see BancoUCN#agregarAlumno
     */
    public boolean agregarAlumno(String nombre,
                                   String rut, String clave, int montoInicial) {
        .....
    }
}

```

Diagrama de Clases

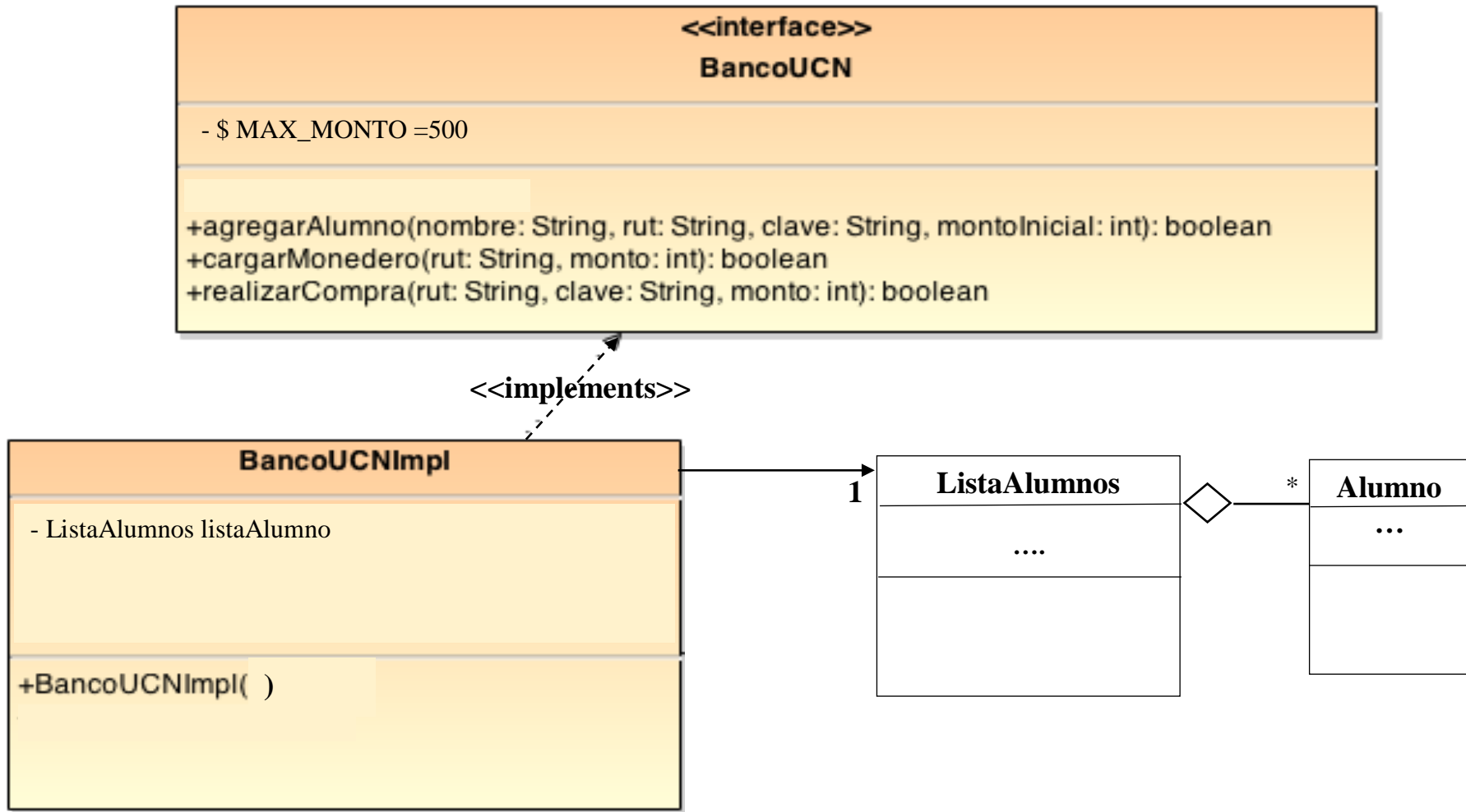


Diagrama de clase

- En un diagrama de clases, el “estereotipo” es una marca que se coloca sobre el título o sobre una relación para indicar su comportamiento, por ejemplo <<interface>>.
- En general las clases e interfaces siempre están relacionadas, en este caso, se indica por una línea segmentada, que indica una relación débil de dependencia

2.7.4. Modularización de la aplicación mediante paquetes

¡Las clases no andan sueltas, se agrupan en **paquetes**!

Los paquetes contienen en su interior definiciones de una o más clases lógicamente relacionadas.

Es una forma de modularizar los programas en JAVA.

El paquete es una colección de clases que se encuentran en el mismo directorio.

Forma general de un **archivo fuente de Java**:

- Una única sentencia de paquete (opcional).
- Las sentencias de importación deseadas (opcional).
- Una única declaración de clase pública.
- Las clases privadas dentro del paquete que se requieran (opcional).

Formato general de la sentencia package es:

```
package paq1 [.paq 2[.paq 3]];
```

Un paquete declarado como:

```
package java.awt.imagen;
```

se debe almacenar en:

```
java\ awt\imagen (Windows)
```

ó

```
java/ awt /imagen (UNIX)
```

Si no se especifica ningún paquete, la clase se incorpora al paquete por omisión, también conocido como el paquete innominado, pues carece de nombre.

Todas las clases que se encuentran en archivos de un mismo directorio forman un mismo paquete.

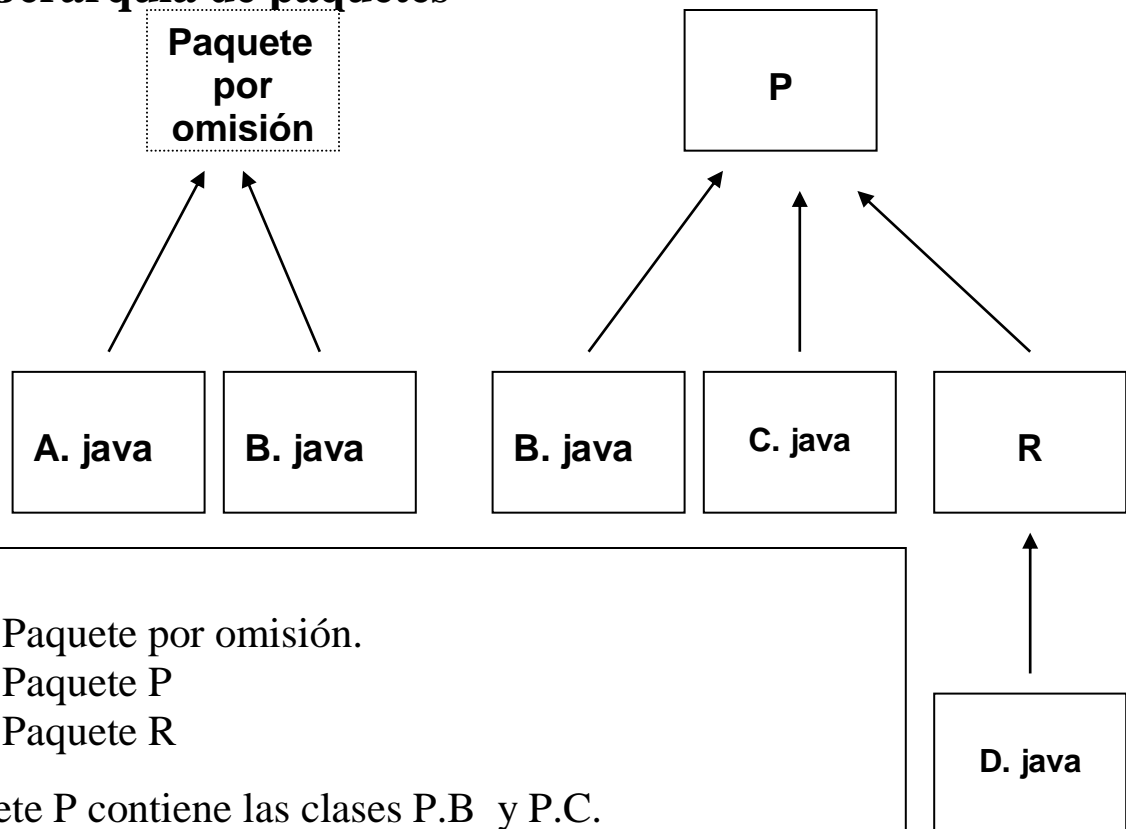
Los paquetes se organizan a su vez en una jerarquía de paquetes.

En JAVA, los paquetes son recipientes con clases, que se utilizan para mantener el espacio de nombres de clase, dividido en compartimentos.

Por ejemplo, un paquete permite que se cree una clase llamada LISTA, que se puede almacenar en un paquete propio, sin preocuparse por conflictos con otra clase llamada LISTA, escrita por otra persona.

Los paquetes se almacenan de una manera jerárquica y se importan explícitamente, en las definiciones de clases nuevas

Ejemplo de Jerarquía de paquetes

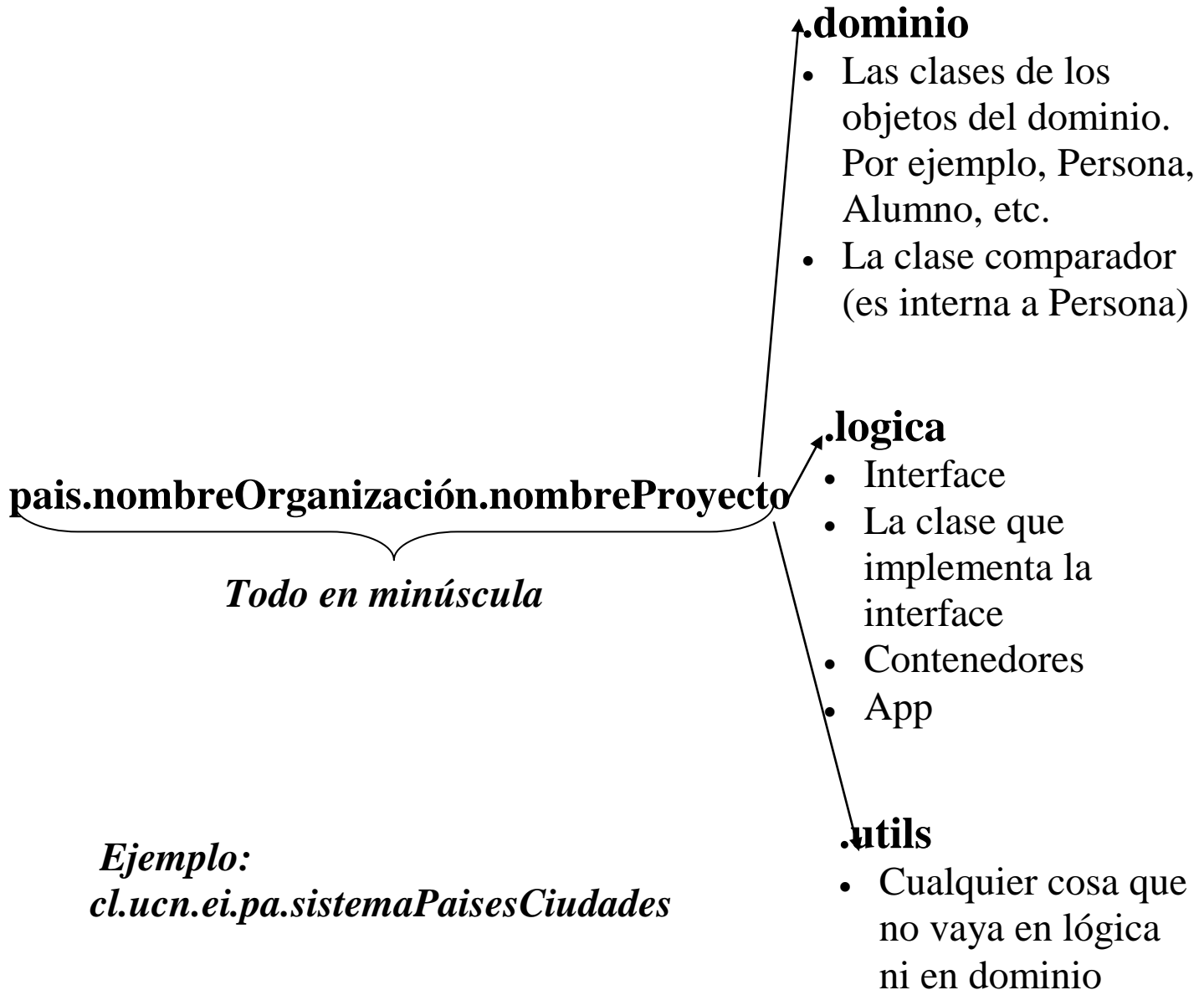


Paquetes:

Paquete por omisión.
Paquete P
Paquete R

- El paquete P contiene las clases P.B y P.C.
- El paquete P.R contiene la clase P.R.D.
- Las clases A y B, se acceden directamente.

¿Cómo se organiza un proyecto en los paquetes?



Recuerde que si el código de un paquete referencia al código que está en otro paquete, se debe hacer el import correspondiente

2.7.5. Ejemplos

Ejemplo 1

Se desea manejar la información de una cuenta bancaria de un banco. Una cuenta tiene un saldo. Se puede depositar y girar sobre la cuenta bancaria. La aplicación debe hacer un depósito sobre la cuenta y luego un giro sobre la misma cuenta. Una vez hechas las transacciones sobre la cuenta, se debe desplegar su saldo.

Modelo del dominio

Cuenta Bancaria
<ul style="list-style-type: none"> • saldo

Contratos

Operación	Girar (cuenta bancaria, monto giro)
Descripción	Se gira dinero desde la cuenta
Precondiciones	Que exista la cuenta bancaria
Postcondiciones	Saldo actualizado de la cuenta

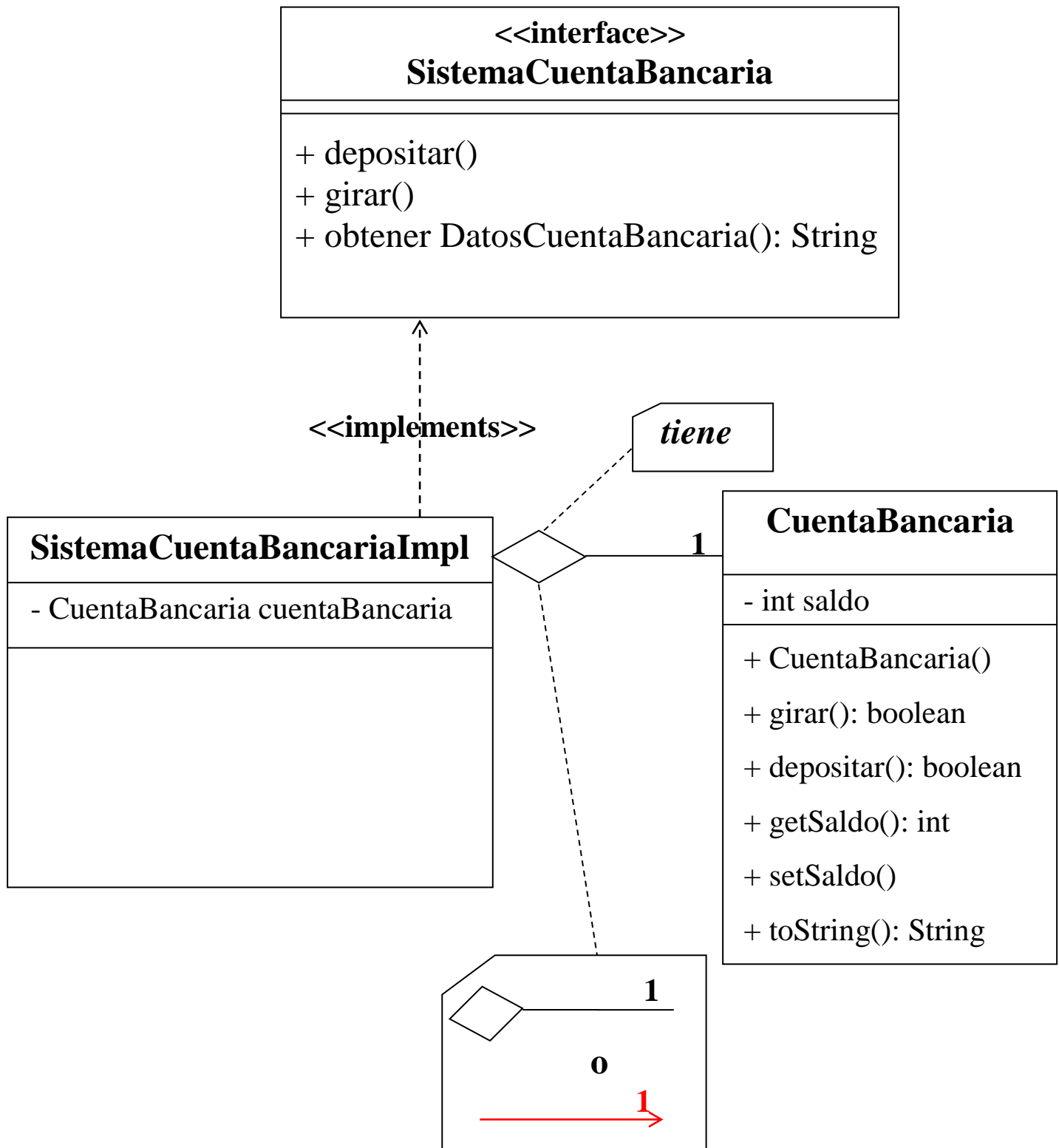
Operación	Depositar (cuenta bancaria, monto depósito)
Descripción	Se deposita dinero en la cuenta
Precondiciones	Que exista la cuenta bancaria
Postcondiciones	Saldo actualizado de la cuenta

Operación	Obtener datos cuenta bancaria
Descripción	Se obtienen los datos de la cuenta bancaria
Precondiciones	Que exista la cuenta bancaria
Postcondiciones	

Diagrama de clases del dominio de la aplicación

CuentaBancaria
- int saldo
+ CuentaBancaria() + girar(): boolean + depositar(): boolean + getSaldo(): int + setSaldo() + toString(): String

Diagrama de clases



```

/**
 * © 2016 Escuela Ingenieria, UCN
 */
package cl.ucn.ei.pa.proyectocuentabancaria.dominio;

/**
 * @author Loreto Telgie
 * @Version: 1 septiembre 2016
 */
public class CuentaBancaria {
    private int saldo;

    /**
     * @return the saldo
     */
    public final int getSaldo() {
        return saldo;
    }

    /**
     * @param saldo the saldo to set
     */
    public final void setSaldo(int saldo) {
        this.saldo = saldo;
    }

    /**
     * @param saldo
     */
    public CuentaBancaria(int saldo) {
        this.saldo = saldo;
    }

    public boolean girar(int montoGiro) {
        if (montoGiro > 0 && montoGiro < saldo) {
            saldo = saldo - montoGiro;
            return true;
        }
        else {
            return false;
        }
    }
}

```

Una alternativa a tener la lógica de la aplicación en una clase del dominio, es tenerla en la clase que implementa la interfaz.

Lo anterior con la finalidad de que las clases del dominio de la aplicación tengan solo la rutina constructora y los get y set.


```
public boolean depositar (int montoDeposito) {  
    if (montoDeposito > 0) {  
        saldo = saldo + montoDeposito;  
        return true;  
    }  
    else {  
        return false;  
    }  
}
```

Es un chequeo de la lógica de la aplicación. Se podría tener en la clase que implementa la interfaz

```
/* (non-Javadoc)  
 * @see java.lang.Object#toString()  
 */
```

```
@Override
```

```
public String toString() {  
    return "CuentaBancaria [saldo=" + saldo + "];  
}
```

Toda clase tiene un método toString()

```
}//Fin CuentaBancaria
```

```

/**
 * © 2016 Escuela Ingenieria, UCN
 */
package cl.ucn.ei.pa.proyectocuentabancaria.logica;

/**
 * @author Loreto Telgie
 * @Version: 1 Septiembre 2016
 */

public interface SistemaCuentaBancaria {

    /**
     * @param montoDeposito: monto a depositar
     *
     * Descripción Se deposita dinero en la cuenta
     * Precondiciones: Que exista la cuenta bancaria
     * Postcondiciones: Saldo actualizado de la cuenta
     */
    public boolean depositar (int montoDeposito);

    /**
     * @param montoGiro: monto a girar
     * Descripción Se gira dinero desde la cuenta
     * Precondiciones: Que exista la cuenta bancaria
     * Postcondiciones: Saldo actualizado de la cuenta
     */
    public boolean girar(int montoGiro);

    /**
     * @return un string con los datos de la cuenta bancaria
     * Descripcion: Retorna los datos de la cuenta bnacraia
     * Precondiciones: Que exista la cuenta bancaria
     * Postcondiciones
     */
    public String obtenerDatosCuentaBancaria();
}

```

```

/**
 * © 2016 Escuela Ingenieria, UCN
 */
package cl.ucn.ei.pa.proyectocuentabancaria.logica;

import cl.ucn.ei.pa.proyectocuentabancaria.dominio.CuentaBancaria;

/**
 * @author Loreto Telgie
 * @version: 1 septiembre 2016
 */
public class SistemaCuentaBancariaImpl implements

```

```

    SistemaCuentaBancaria{

```

```

    private CuentaBancaria cuentaBancaria; ;

```

```

    public SistemaCuentaBancariaImpl(int saldoInicial) {
        this.cuentaBancaria = new CuentaBancaria(saldoInicial);
    }

```

Supuesto: El sistema cuando se levanta crea el objeto cuenta bancaria, es decir existe la cuenta

```

/**
 * @see cl.ucn.ei.pa.proyectocuentabancaria.logica.
 *      SistemaCuentaBancaria#depositar(int)
 */
public boolean depositar(int montoDeposito) {
    if (cuentaBancaria==null) {
        throw new NullPointerException("Cuenta bancaria
                                       no existe para hacer un deposito");
    }
    else{
        return cuentaBancaria.depositar(montoDeposito);
    }
}

```

- Se trabaja con excepciones para implementar las precondiciones indicadas en los contratos
- Aquí se lanza la excepción

```

/**
 * @see cl.ucn.ei.pa.proyectocuentabancaria.logica.
 *                                     SistemaCuentaBancaria#girar(int)
 */
public boolean girar(int montoGiro) {
    if (cuentaBancaria==null) {
        throw new NullPointerException("Cuenta bancaria
                                     no existe para hacer un giro");
    }
    else{
        return cuentaBancaria.girar(montoGiro);
    }
}

```

- *Se trabaja con excepciones para implementar las precondiciones indicadas en los contratos*
- *Aquí se lanza la excepción*

```

/**
 * @see cl.ucn.ei.pa.proyectocuentabancaria.logica.
 *                                     SistemaCuentaBancaria#obtenerDatosCuentaBancaria()
 */
public String obtenerDatosCuentaBancaria() {
    if (cuentaBancaria==null) {
        throw new NullPointerException ("Cuenta bancaria
                                     no existe");
    }
    return cuentaBancaria.toString();
}

```

```

} //Fin SistemaCuentaBancariaImpl

```

```
/**
 * @author Loreto Telgie
 * @version 1 septiembre 2016
 */
```

- *En la App se leen los datos y se despliegan resultados*
- *La App representa la interfaz para el ingreso y despliegue de los datos (podría ser una interfaz gráfica)*

```
public class App {
```

```
    public static void main(String[] args) {
```

```
        StdOut.print("Ingrese saldo inicial de la cuenta: ");
        int saldo = StdIn.readInt();
        //Debiera estar la validación de los datos de entrada
```

```
        SistemaCuentaBancaria sistema =
            new SistemaCuentaBancariaImpl(saldo);
```

```
        try{
            StdOut.println("Datos cuenta bancaria " +
                           sistema.obtenerDatosCuentaBancaria());
        } catch (NullPointerException ex) {
            StdOut.println(ex.getMessage());
        }
```

- *Se trabaja con excepciones para implementar las precondiciones indicadas en los contratos*
- *Aquí se está capturando la excepción*

```
        try {
            if (!sistema.depositar(74)) {
                StdOut.println("Deposito no efectuado,
                               monto deposito menor que 0");
            }
        } catch (NullPointerException ex) {
            StdOut.println(ex.getMessage());
        }
```

```
        if (!sistema.depositar(-5)) {
            StdOut.println("Deposito no efectuado,
                           monto deposito menor que 0");
        }
```

*Hubiera
salido el
mensaje de
despliegue*

```

try{
    if (!sistema.girar(20)){
        StdOut.println("Giro no efectuado, monto giro
                        menor que 0 o mayor que el saldo");
    }
} catch (NullPointerException ex) {
    StdOut.println(ex.getMessage());
}

```

- *Se trabaja con excepciones para implementar las precondiciones indicadas en los contratos*
- *Aquí se está capturando la excepción*

```

try{
    StdOut.println("Datos cuenta bancaria " +
                    sistema.obtenerDatosCuentaBancaria());
} catch (NullPointerException ex) {
    StdOut.println(ex.getMessage());
}

```

```

} //Fin main

```

```

} //Fin App

```

Si se ingresa como saldo inicial el valor 100

Se imprime

Saldo antes de las transacciones: 100
 Antes de las transacciones: 154

Diagrama de objetos

cuentaBancaria: CuentaBancaria

saldo = 100

~~174~~

154

Nota: En los talleres se debe trabajar con manejo de excepciones

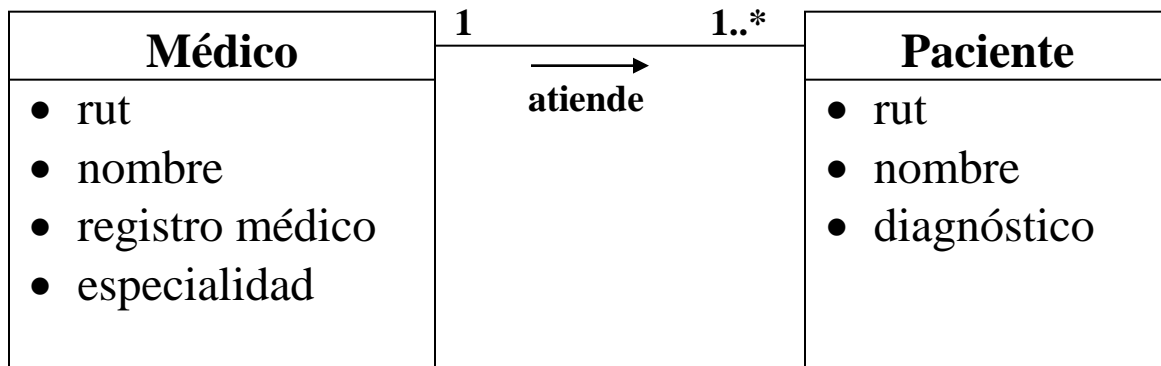
Ejemplo 2

Un médico tiene rut, nombre, registro médico y especialidad. Un paciente tiene rut, nombre, diagnóstico y médico de cabecera (sólo 1).

La aplicación debe realizar lo siguiente:

- Ingresar un médico
- Ingresar un paciente (sólo rut y nombre)
- Ingresar el diagnóstico de un paciente
- Asignar un médico a un paciente
- Desplegar al médico
- Desplegar al paciente, incluyendo el nombre de su médico

Modelo del dominio



Contratos

Operación	Ingresar médico (rut, nombre, registro médico, especialidad)
Descripción	Se ingresa un médico
Precondiciones	
Postcondiciones	Médico ingresado

Nota

La validación de los datos de entrada (cuando se lee) se hace en la App.

Operación	Ingresar paciente (rut, nombre)
Descripción	Se ingresa un paciente
Precondiciones	
Postcondiciones	Paciente ingresado

Operación	Asignar diagnóstico al paciente
Descripción	Se le asigna un diagnóstico al paciente
Precondiciones	Que exista el paciente
Postcondiciones	Diagnóstico asignado al paciente

Operación	Asignar médico al paciente
Descripción	Se le asigna un médico al paciente
Precondiciones	<ul style="list-style-type: none"> • Que exista el médico
Postcondiciones	Médico asignado al paciente

Operación	Obtener médico
Descripción	Se obtiene el médico
Precondiciones	Que exista el médico
Postcondiciones	

Operación	Obtener paciente
Descripción	Se obtiene al paciente
Precondiciones	Que exista el paciente
Postcondiciones	

Operación	Obtener datos del paciente y el de su médico
Descripción	Se obtiene los datos del paciente y los de su médico
Precondiciones	Que exista el paciente
Postcondiciones	

Operación	Obtener datos del medico
Descripción	Se obtiene los datos del médico
Precondiciones	Que exista el médico
Postcondiciones	

Operación	Obtener datos del médico
Descripción	Se obtiene los datos del médico
Precondiciones	Que exista el médico
Postcondiciones	

Diagrama de clases del dominio de la aplicación

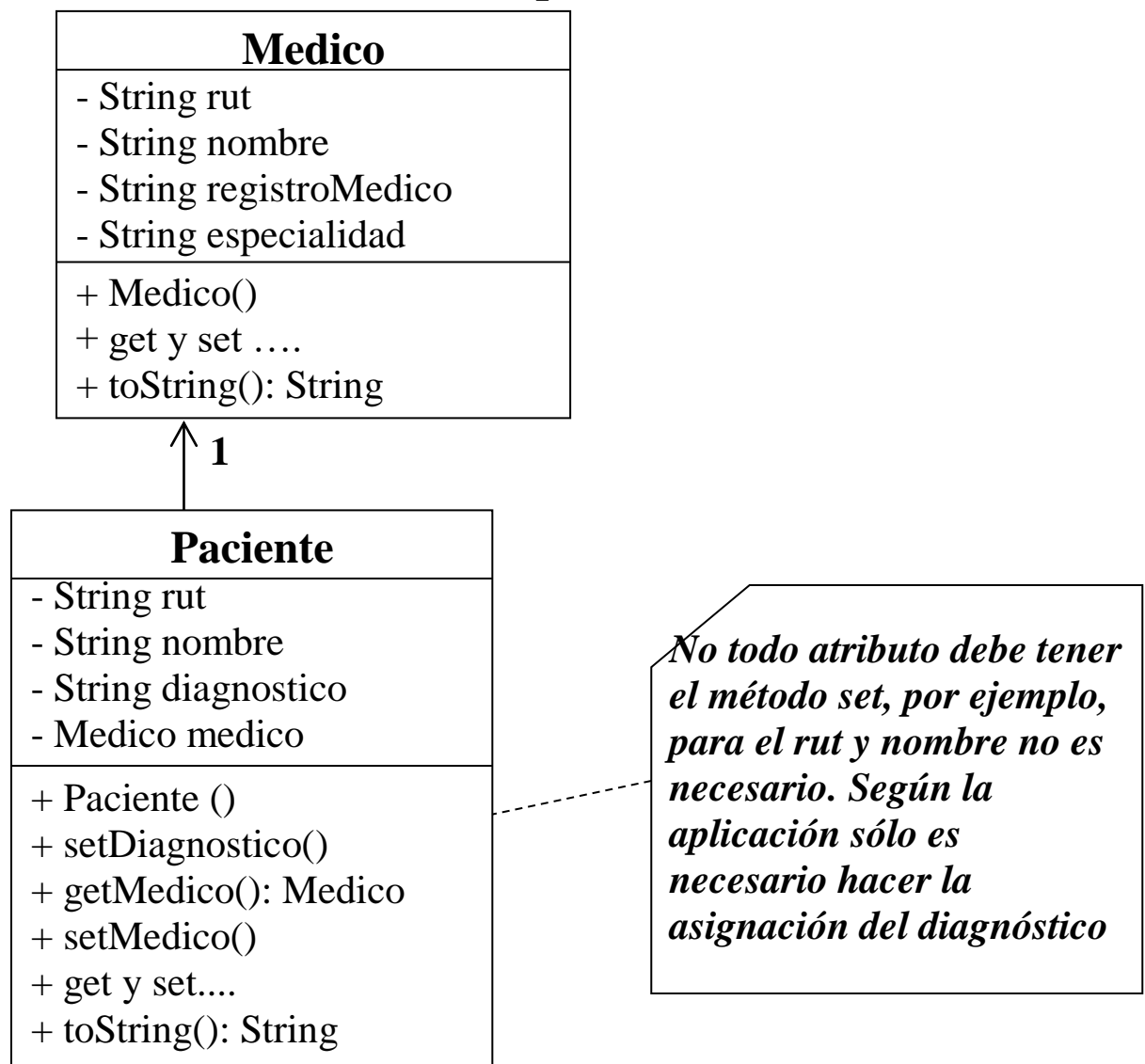


Diagrama de clases

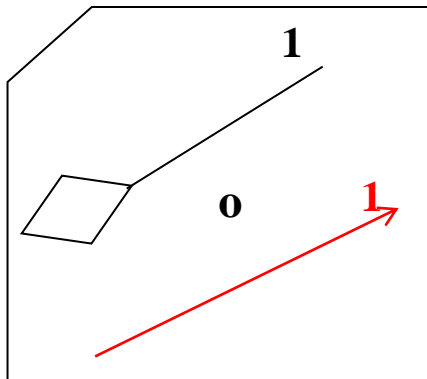
2.130

Los métodos de la interfaz corresponden a los contratos. Un método por contrato

<<interface>> SistemaMedico

```
+ ingresarMedico()
+ ingresar Paciente()
+ asignarDiagnostico()
+ asignarMedico()
+ obtenerMedico(): Medico
+ obtenerPaciente(): Paciente
+ obtenerDatosMedico():String
+ obtenerDatosPacienteYSuMedico():String
```

<<implements>>



SistemaMedicoImpl

```
- Medico medico
- Paciente paciente
```

Medico

```
- String rut
- String nombre
- String registroMedico
- String especialidad
```

```
+ Medico()
+ get y set ....
+ toString();
```

Paciente

```
- String rut
- String nombre
- String diagnostico
- Medico medico
```

```
+ Paciente ()
+ setDiagnostico()
+ getMedico(): Medico
+ setMedico()
+ get y set...
+ toString(): String
```

```

package cl.ucn.ei.pa.sistemamedico.dominio;
public class Medico {
    private String rut;
    private String nombre;
    private String registroMedico;
    private String especialidad;
    public Medico(String rut, String nombre,
                  String registroMedico, String especialidad) {
        this.rut = rut;
        this.nombre = nombre;
        this.registroMedico = registroMedico;
        this.especialidad = especialidad;
    }
    public String getRut() {
        return rut;
    }
    public void setRut(String rut) {
        this.rut = rut;
    }
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public String getRegistroMedico() {
        return registroMedico;
    }
    public void setRegistroMedico(String registroMedico) {
        this.registroMedico = registroMedico;
    }
    public String getEspecialidad() {
        return especialidad;
    }
    public void setEspecialidad(String especialidad) {
        this.especialidad = especialidad;
    }
    @Override
    public String toString() {
        return "Medico [rut=" + rut + ", nombre=" + nombre +
            ", registroMedico=" + registroMedico + ",
            especialidad=" + especialidad + "]";
    }
}
} // Fin clase Medico

```

Toda clase tiene un método toString()

```

package cl.ucn.ei.pa.sistemamedico.dominio;
public class Paciente {
    private String rut;
    private String nombre;
    private String diagnostico;
    private Medico medico;
    public Paciente(String rut, String nombre) {
        this.rut = rut;
        this.nombre = nombre;
        this.diagnostico = null;
        this.medico = null;
    }
    public String getRut() {
        return rut;
    }
    public void setRut(String rut) {
        this.rut = rut;
    }
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public String getDiagnostico() {
        return diagnostico;
    }
    public void setDiagnostico(String diagnostico) {
        this.diagnostico = diagnostico;
    }
    public Medico getMedico() {
        return medico;
    }
    public void setMedico(Medico medico) {
        this.medico = medico;
    }
    @Override
    public String toString() {
        return "Paciente [" + (rut != null ? "rut=" + rut +
            ", " : "") + (nombre != null ? "nombre=" + nombre +
            ", " : "") + (diagnostico != null ? "diagnostico=" +
            diagnostico + ", " : "") +
            (medico != null ? "medico=" + medico : "") + "]";
    }
}
} //Fin Paciente

```

Esta versión de toString() tiene seleccionada la opción del eclipse “saltar valores nulos”

Cuando un objeto está asociado con otro, es preferible sobrescribir el toString que se obtiene con el Eclipse

```

package cl.ucn.ei.pa.sistemamedico.logica;
import cl.ucn.ei.pa.sistemamedico.dominio.*;
/**
 * @author Loreto Telgie
 *
 */
public interface SistemaMedico {

    public void ingresarMedico(String rut, String nombre,
                               String registroMedico,String especialidad);

    public void ingresarPaciente(String rut, String nombre);

    public void asociarDiagnostico(Paciente paciente,
                                   String diagnostico);

    public void asociarMedico(Paciente paciente,
                              Medico medico);

    public Medico obtenerMedico();

    public Paciente obtenerPaciente();

    public String obtenerDatosMedico();

    public String obtenerDatosPacienteySuMedico();

}

```

```

package cl.ucn.ei.pa.sistemamedico.logica;
import cl.ucn.ei.pa.sistemamedico.dominio.*;

/**
 * @author Loreto Telgie
 *
 */
public class SistemaMedicoImpl implements SistemaMedico{

    private Medico medico;
    private Paciente paciente;

    {
        public SistemaMedicoImpl () {
            medico= null;
            paciente= null;
        }
    }
}

```

Cuando se levanta el sistema no hay ningún paciente ni tampoco un médico

```

public void ingresarMedico(String rut, String nombre,
                           String registroMedico,String especialidad){
    Medico medico = new Medico(rut, nombre, registroMedico,
                               especialidad);
    this.medico = medico;
}

public void ingresarPaciente(String rut, String nombre){
    Paciente paciente = new Paciente (rut, nombre);
    this.paciente = paciente;
}

public void asociarDiagnostico(Paciente p, String diagnostico){
    if (p==null){
        throw new NullPointerException ("Paciente no existe");
    }
    p.setDiagnostico(diagnostico);
}

public void asociarMedico(Paciente p , Medico m){
    if (p==null || m == null){
        throw new NullPointerException ("Paciente y/o medico no existe");
    }
    p.setMedico(m);
}

public Medico obtenerMedico(){
    if (medico == null){
        throw new NullPointerException ("Medico no existe");
    }
    return medico;
}

public Paciente obtenerPaciente(){
    if (paciente==null){
        throw new NullPointerException ("Paciente no existe ");
    }
    return paciente;
}

public String obtenerDatosMedico(){
    if (medico==null){
        throw new NullPointerException ("Medico no existe");
    }
    return medico.toString();
}

public String obtenerDatosPacienteySuMedico(){
    if (paciente==null){
        throw new NullPointerException ("Paciente no existe");
    }
    return paciente.toString();
}

```

- Se trabaja con excepciones para implementar las precondiciones indicadas en los contratos
- Aquí se lanza la excepción

```

package cl.ucn.ei.pa.sistemamedico.logica;

import cl.ucn.ei.pa.sistemamedico.dominio.*;
import cl.ucn.ei.pa.sistemamedico.logica.SistemaMedico;
import cl.ucn.ei.pa.sistemamedico.logica.SistemaMedicoImpl;
import ucn.Stdout;

/**
 * @author Loreto Telgie
 */
public class App {

    public static void main(String[] args) {
        SistemaMedico sistema = new SistemaMedicoImpl();

        String rut= "111-K";
        String nombre = "Juan Perez";
        String registroMedico = "151-5";
        String especialidad ="cardiologo";
        //Debiera estar la validación de los datos de entrada

        sistema.ingresarMedico(rut,nombre,
                               registroMedico,especialidad);

        rut= "222-1";
        nombre = "Pedro Soto";
        //Debiera estar la validación de los datos de entrada

        sistema.ingresarPaciente(rut, nombre);

        try{
            Paciente paciente = sistema.obtenerPaciente();
            sistema.asociarDiagnostico(paciente, "Apendicitis");
        } catch (NullPointerException ex){
            StdOut.println(ex.getMessage());
        }
    }
}

```

- *En la App se leen los datos (deben validarse) y se despliegan resultados*
- *La App representa la interfaz para el ingreso y despliegue de los datos (podría ser una interfaz gráfica)*

- *Los objetos se crean en la clase que implementa la interfaz*
- *Se debe intentar trabajar sin objetos en la App, sino que con los datos primitivos*

```

    try {
        Medico medico = sistema.obtenerMedico();
        Paciente paciente = sistema.obtenerPaciente();
        sistema.asociarMedico(paciente, medico);
    } catch (NullPointerException ex) {
        StdOut.println(ex.getMessage());
    }

    try {
        String datosMedico = sistema.obtenerDatosMedico();
        StdOut.println("Datos del medico: " + datosMedico);
    } catch (NullPointerException ex) {
        StdOut.println(ex.getMessage());
    }

    try {
        String datosPaciente =
            sistema.obtenerDatosPacienteYSuMedico();
        StdOut.println("Datos del paciente: " +
            datosPaciente);
    } catch (NullPointerException ex) {
        StdOut.println(ex.getMessage());
    }

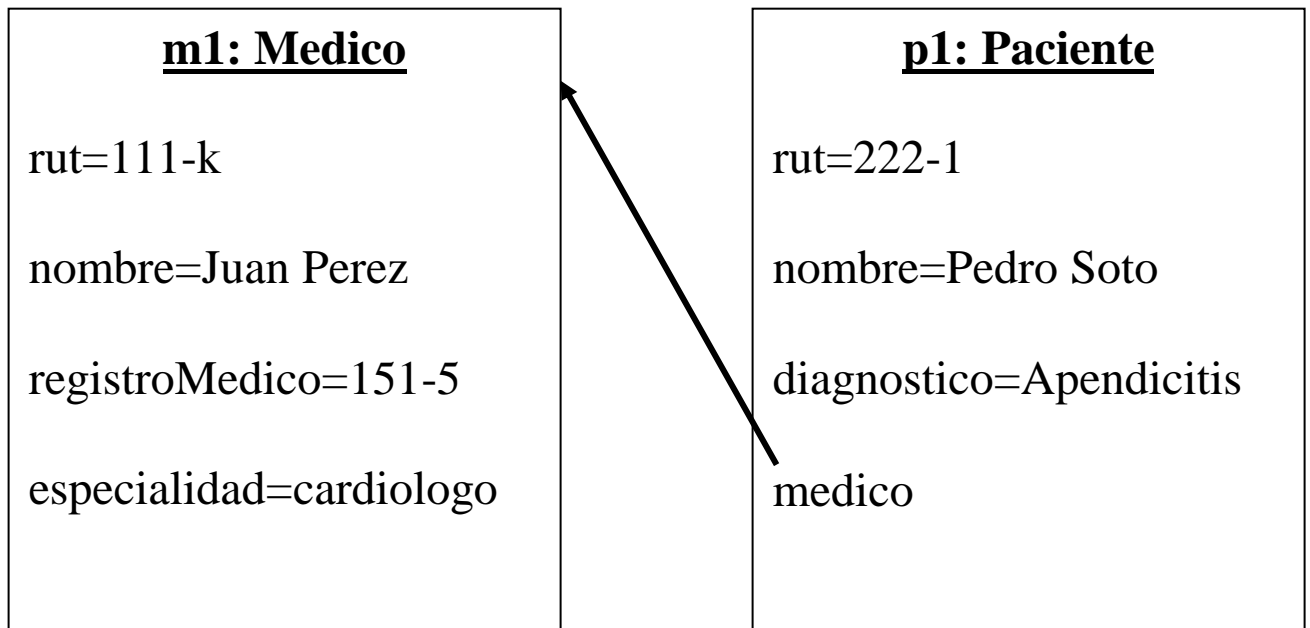
    } // Fin main

} // Fin App

```

- *Se trabaja con excepciones para implementar las precondiciones indicadas en los contratos*
- *Aquí se está capturando la excepción*

Diagrama de Objetos

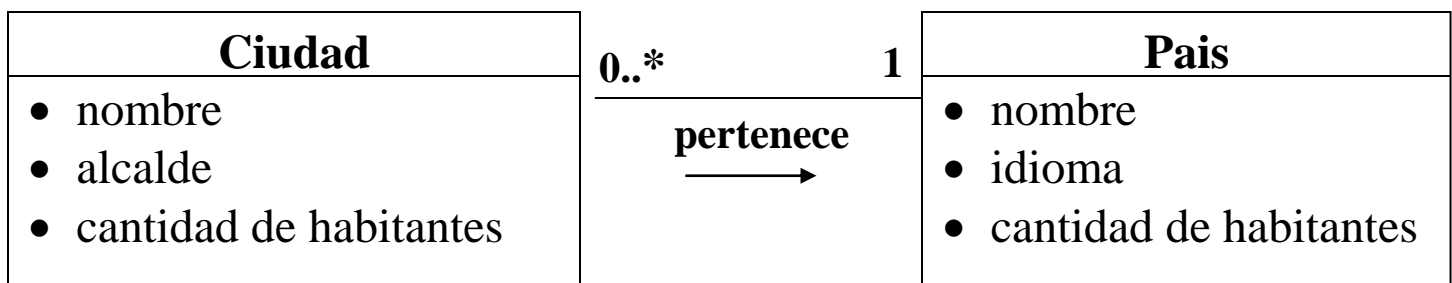


Ejemplo 3

Se necesita manejar información de un país y sus ciudades. Para un país interesa saber su nombre, idioma y cantidad de habitantes. Para una ciudad, su nombre, alcalde (suponga sólo 1 alcalde por ciudad) y cantidad de habitantes. Se pide que haga un programa en Java que:

- Ingrese el país Chile. Suponga que tiene 18.000.000 de habitantes.
- Ingrese N ciudades de Chile, donde N se lee desde pantalla (lea desde pantalla los datos de cada ciudad)
- Una vez ingresada la información del país y sus ciudades, despliegue todos los datos del país y los datos de cada una de sus ciudades.

Modelo del dominio



Contratos

Operación	Ingresar un país (nombre, idioma, cantidad habitantes pais)
Descripción	Se ingresa un país
Precondiciones	
Postcondiciones	País creado

Operación	Ingresar ciudad de un pais (nombre, alcalde, cantidadHabitantes)
Descripción	Se ingresa una ciudad, quedando asociada al país
Precondiciones	Que exista el país
Postcondiciones	Se crea la ciudad y se asocia con el pais

Operación	Obtener datos del país y sus ciudades
Descripción	Se obtienen los datos del país con sus ciudades
Precondiciones	Que exista el país
Postcondiciones	

Operación	Obtener el país y sus ciudades
Descripción	Se obtienen el país con sus ciudades
Precondiciones	Que exista el país
Postcondiciones	

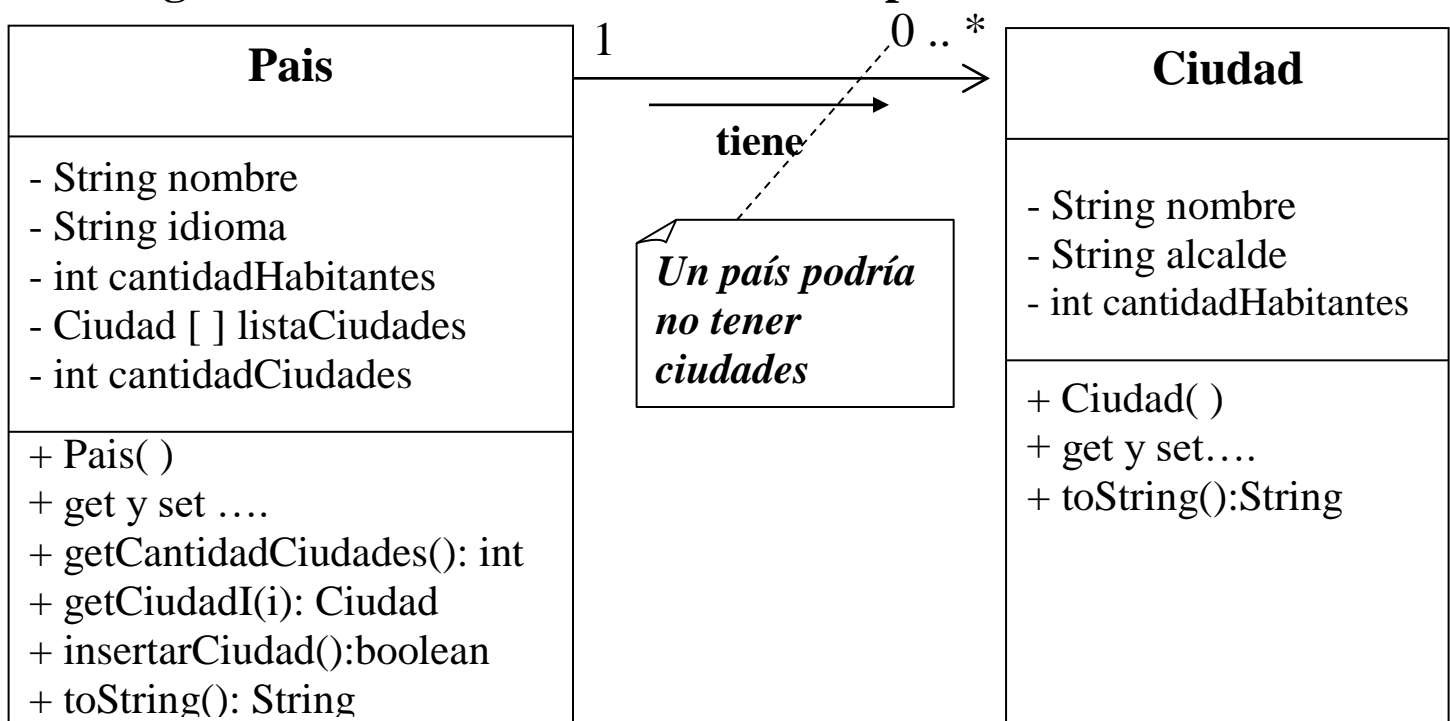
La diferencia entre estos 2 últimos contratos es que:

- **Obtener datos del país y sus ciudades:** *Obtiene un string con todos los datos. Para imprimir los datos basta con una impresión*
- **Obtener el país y sus ciudades:** *Obtiene el país. Para imprimir se deben ir sacando cada uno de los elementos del objeto país y luego desplegarlo*

En la aplicación se usa una de las 2 alternativas

Versión 1

Diagrama de Clases del domino de la aplicación




```
package cl.ucn.ei.pa.paisciudades.dominio;

public class Ciudad {

    private String nombre;
    private String alcalde;
    private int cantidadHabitantes;

    public Ciudad(String nombre, String alcalde,
                  int cantidadHabitantes) {
        this.nombre = nombre;
        this.alcalde = alcalde;
        this.cantidadHabitantes = cantidadHabitantes;
    }
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public String getAlcalde() {
        return alcalde;
    }
    public void setAlcalde(String alcalde) {
        this.alcalde = alcalde;
    }
    public int getCantidadHabitantes() {
        return cantidadHabitantes;
    }
    public void setCantidadHabitantes(
        int cantidadHabitantes) {
        this.cantidadHabitantes = cantidadHabitantes;
    }
}
```

Recuerde que en los talleres el código debe estar documentado

```

    public String toString() {
        return "Ciudad [nombre=" + nombre + ", alcalde=" +
            alcalde + ", cantidadHabitantes=" +
            cantidadHabitantes + "];"
    }
}

```

```

package cl.ucn.ei.pa.paisciudades.dominio;

public class Pais {
    private String nombre;
    private String idioma;
    private int cantidadHabitantes;
    private Ciudad [] listaCiudades;
    private int cantidadCiudades;
    private int max;

    public Pais(String nombre, String idioma,
                int cantidadHabitantes, int max) {
        this.nombre = nombre;
        this.idioma = idioma;
        this.cantidadHabitantes = cantidadHabitantes;
        listaCiudades = new Ciudad[max];
        cantidadCiudades = 0;
        this.max = max;
    }

    public final String getNombre() {
        return nombre;
    }

    public final void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public final String getIdioma() {
        return idioma;
    }
}

```

```

public final void setIdioma(String idioma) {
    this.idioma = idioma;
}
public final int getCantidadHabitantes() {
    return cantidadHabitantes;
}
public final void setCantidadHabitantes(
    int cantidadHabitantes) {
    this.cantidadHabitantes = cantidadHabitantes;
}
public void setCantidadCiudades(int
    cantidadCiudades){
    this.cantidadCiudades = cantidadCiudades;
}
public int getCantidadCiudades() {
    return cantidadCiudades;
}
public boolean insertarCiudad(Ciudad ciudad) {
    if (cantidadCiudades < max) {
        listaCiudades[cantidadCiudades] = ciudad;
        cantidadCiudades++;
        return true;
    } else {
        return false;
    }
}
public Ciudad getCiudadI(int i) {
    if (i >= 0 && i < cantidadCiudades) {
        return listaCiudades[i];
    } else {
        return null;
    }
}

```

```

    public String toString() {
        return "Pais [nombre=" + nombre + ", idioma=" +
            idioma + ", cantidadHabitantes=" +
            cantidadHabitantes + ", listaCiudades=" +
            Arrays.toString(listaCiudades) +
            ", cantidadCiudades="+cantidadCiudades +
            ", max=" + max + "]" ;
    }
}

```

```

package cl.ucn.ei.pa.paisciudades.logica;

import cl.ucn.ei.pa.paisciudades.dominio.Pais;

public interface SistemaPaisCiudades {

    public void ingresarPais(String nombre,
        String idioma, int cantidadHabitantes,
        int cantidadCiudades);

    public void ingresarCiudadDeUnPais(String nombre,
        String alcalde, int cantidadHabitantes)

    public Pais obtenerPaisCiudades();

    public String obtenerDatosPaisCiudades();

}

```

Uno de los 2


```

package cl.ucn.ei.pa.paisciudades.logica;
import cl.ucn.ei.pa.paisciudades.dominio.*;
public class SistemaPaisCiudadesImpl implements
                                SistemaPaisCiudades {
    private Pais pais;

    public SistemaPaisCiudadesImpl() {
        pais = null;
    }

    public void ingresarPais(String nombre, String idioma,
        int cantidadHabitantes, int cantidadCiudades) {

        pais = new Pais(nombre, idioma,
            cantidadHabitantes, cantidadCiudades);
    }

    public void ingresarCiudadDeUnPais(String nombre,
        String alcalde, int cantidadHabitantes){
        Ciudad ciudad = new Ciudad(nombre, alcalde,
            cantidadHabitantes);

        if (pais!= null){
            pais.insertarCiudad(ciudad);
        }
        else{
            throw new NullPointerException("No existe pais");
        }
    }

    public Pais obtenerPaisCiudades() {
        if (pais!= null){
            return pais;
        }
        else{
            throw new NullPointerException ("No existe pais");
        }
    }
}

```

Uno de los 2:

- *obtenerPaisCiudades*

o

- *obtenerDatosPaisCiudades*

```

public String obtenerDatosPaisCiudades() {
    if (pais != null) {
        return pais.toString();
    }
    else {
        throw new NullPointerException ("No existe pais");
    }
}
}

```

```

package cl.ucn.ei.pa.paisciudades.logica;

import ucn.Stdout;
import ucn.StdIn;
import cl.ucn.ei.pa.paisciudades.dominio.*;

public class App {

    public static void
        LeerCiudadesPais (SistemaPaisCiudades sistema) {

        StdOut.print("Cantidad de ciudades de Chile: ");
        int N = StdIn.readInt();

        sistema.ingresarPais("Chile", "Espanol", 18000000, N);

        for (int i = 1; i <= N; i++) {
            StdOut.print("Ingrese nombre ciudad: ");
            String nombre = StdIn.readString();
            StdOut.print("Ingrese Alcalde: ");
            String alcalde = StdIn.readString();
            StdOut.print("Ingrese cantidad habitantes: ");
            int cantidadHabitantes = StdIn.readInt();

            try {
                boolean ingreso =
                    sistema.ingresarCiudadDeUnPais(nombre,
                                                    alcalde, cantidadHabitantes);

                if (!ingreso) {
                    StdOut.println("No hay espacio para ingresar ciudades");
                }
            } catch (NullPointerException ex) {
                StdOut.println(ex.getMessage());
            }
        }
    }
}

```

Que no exista espacio, no es una precondition en un contrato

```

public static void desplegarPaisCiudades (Pais pais) {
    StdOut.println("nombre pais: "+pais.getNombre() +
        ", idioma: " + pais.getIdioma() +
        ", cantidad habitanes: "
        + pais.getCantidadHabitantes());
    for (int i = 0; i < pais.getCantidadCiudades(); i++) {
        Ciudad ciudad = pais.getCiudadI(i);
        if (ciudad != null) {
            StdOut.println("ciudad " +
                ciudad.getNombre() + ", alcalde: " +
                ciudad.getAlcalde() + ", habitantes:" +
                ciudad.getCantidadHabitantes());
        }
    }
}

```

```

public static void main(String[] args) {
    SistemaPaisCiudades sistema = new
        SistemaPaisCiudadesImpl();
    LeerCiudadesPais(sistema);

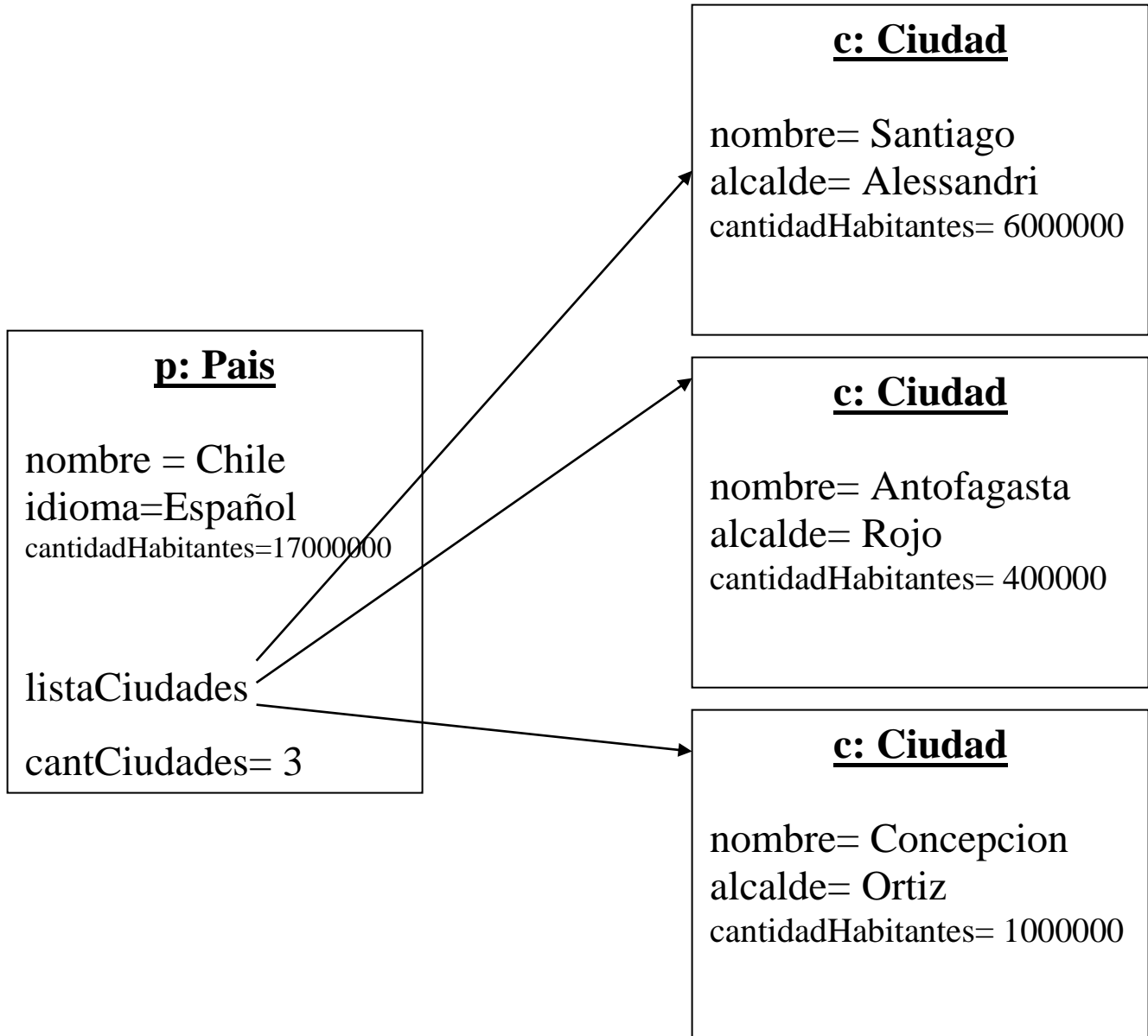
    try {
        StdOut.println(sistema.
            obtenerDatosPaisCiudades());

        Pais pais = sistema.obtenerPaisCiudades();
        App.desplegarPaisCiudades(pais);
    } catch (NullPointerException ex) {
        StdOut.println(ex.getMessage());
    }
} //Fin main
} //Fin App

```

*Para desplegar los datos,
una de las 2 alternativas*

Diagrama de objetos



Versión 2

Diagrama de clases del dominio de la aplicación

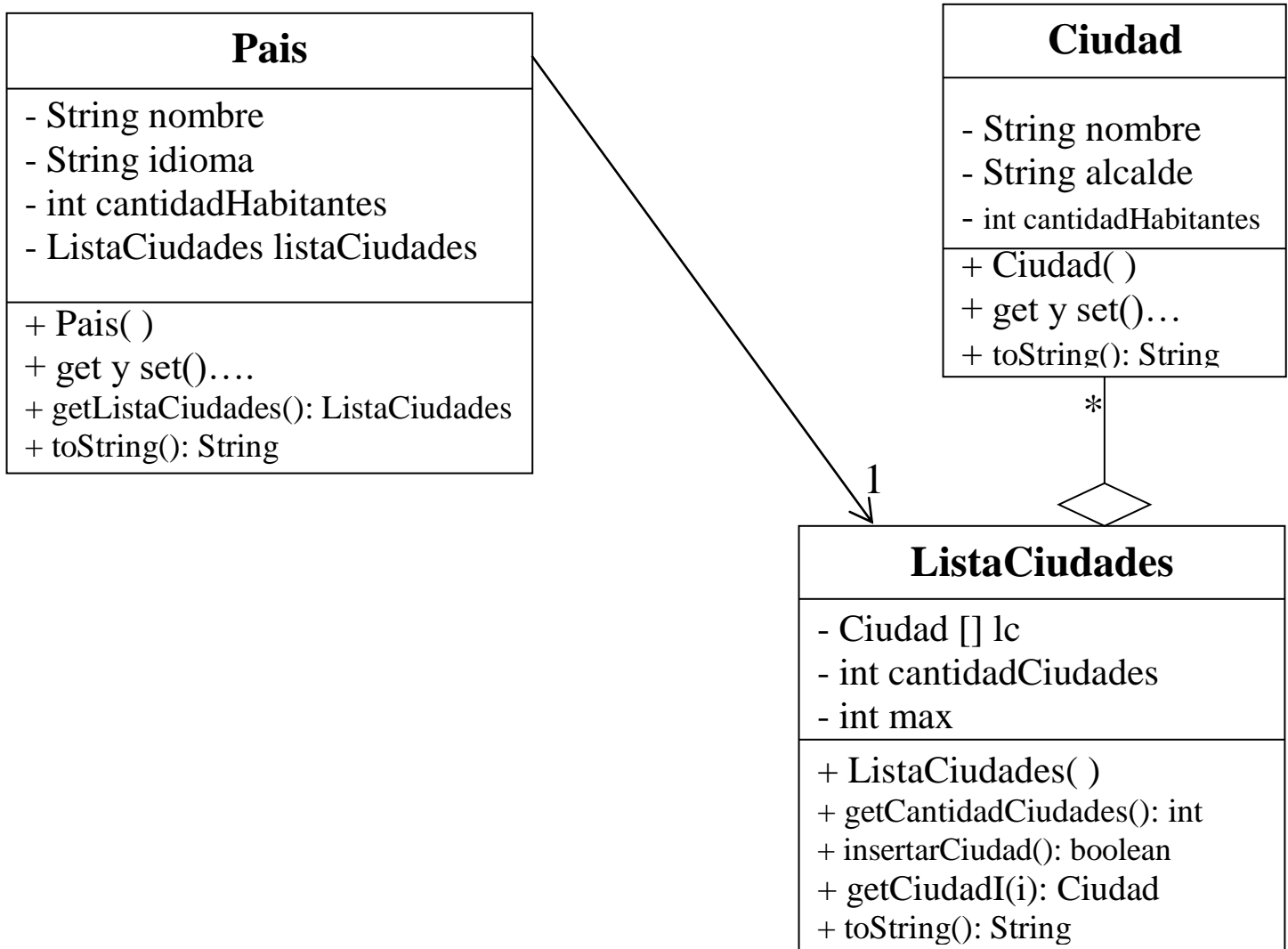
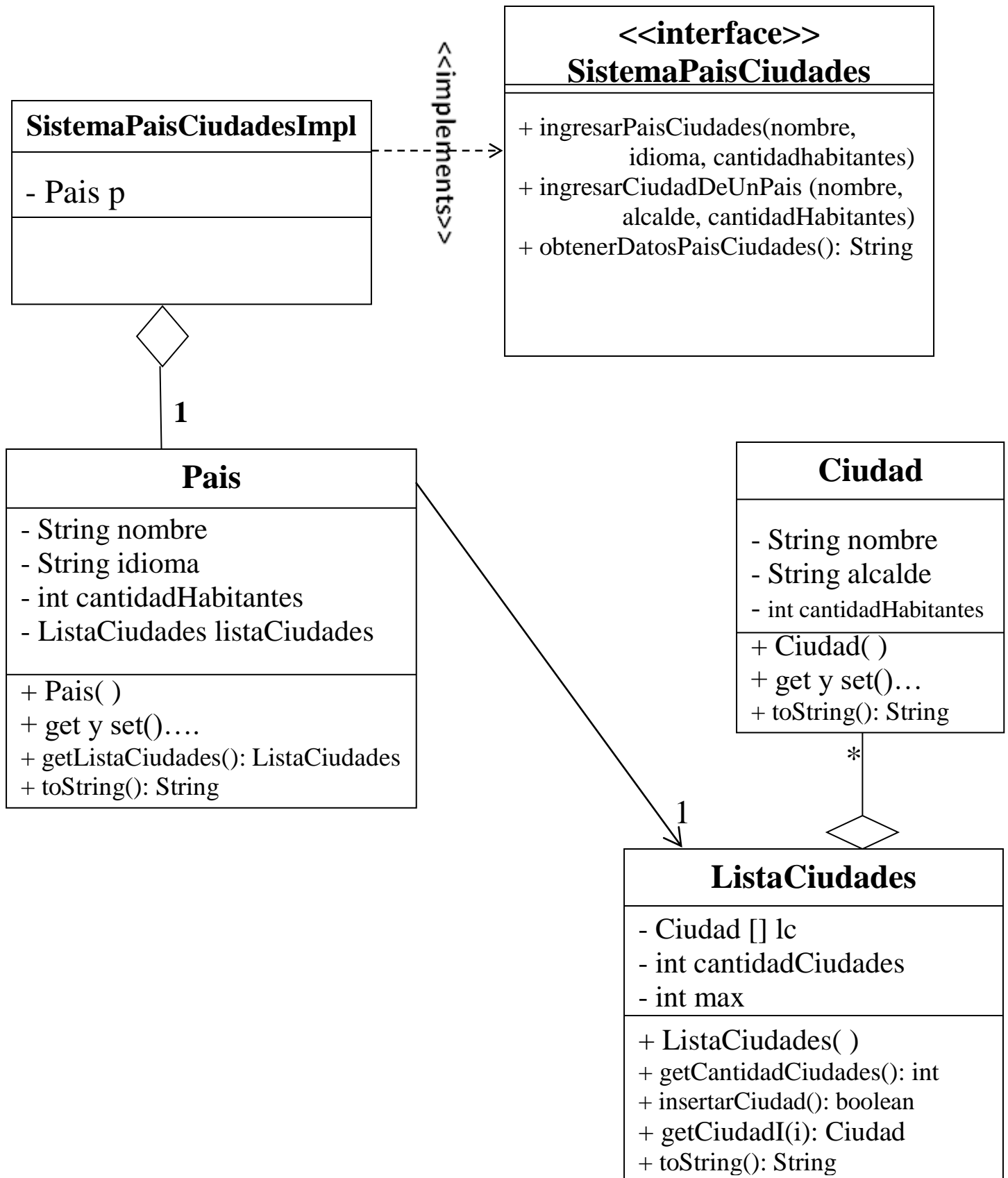


Diagrama de clases



```
package cl.ucn.ei.pa.paisciudades.dominio;

public class Ciudad {

    private String nombre;
    private String alcalde;
    private int cantidadHabitantes;

    public Ciudad(String nombre, String alcalde,
                  int cantidadHabitantes) {

        this.nombre = nombre;
        this.alcalde = alcalde;
        this.cantidadHabitantes = cantidadHabitantes;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getAlcalde() {
        return alcalde;
    }

    public void setAlcalde(String alcalde) {
        this.alcalde = alcalde;
    }

    public int getCantidadHabitantes() {
        return cantidadHabitantes;
    }

    public void setCantidadHabitantes(int cantidadHabitantes) {
        this.cantidadHabitantes = cantidadHabitantes;
    }

    @Override
    public String toString() {
        return "Ciudad [nombre=" + nombre + ", alcalde=" +
            alcalde + ", cantidadHabitantes=" +
            cantidadHabitantes + "]";
    }

}
```

```

package cl.ucn.ei.pa.paisciudades.dominio;
import cl.ucn.ei.pa.paisciudades.logica.ListaCiudades;

public class Pais {
    private String nombre;
    private String idioma;
    private int cantidadHabitantes;
    private ListaCiudades listaCiudades;

    public Pais(String nombre, String idioma, int cantidadHabitantes,
                                                         cantidadCiudades) {

        this.nombre = nombre;
        this.idioma = idioma;
        this.cantidadHabitantes = cantidadHabitantes;
        listaCiudades = new ListaCiudades(cantidadCiudades);
    }

    public final String getNombre() {
        return nombre;
    }

    public final void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public final String getIdioma() {
        return idioma;
    }

    public final void setIdioma(String idioma) {
        this.idioma = idioma;
    }

    public final int getCantidadHabitantes() {
        return cantidadHabitantes;
    }

    public final void setCantidadHabitantes(
                                                         int cantidadHabitantes) {

        this.cantidadHabitantes = cantidadHabitantes;
    }

    public ListaCiudades getListaciudades() {
        return listaCiudades;
    }

    public String toString() {
        return "Pais [" + (nombre != null ? "nombre=" + nombre +
            ", " : "") + (idioma != null ? "idioma=" + idioma +
            ", " : "") + "cantidadHabitantes=" +
            cantidadHabitantes + ", " + (listaCiudades != null ?
            "listaCiudades=" + listaCiudades : "") + "]";
    }
}

```



```

package cl.ucn.ei.pa.paisciudades.logica;
import cl.ucn.ei.pa.paisciudades.dominio.Ciudad;
public class ListaCiudades {
    private Ciudad[] lc;
    private int cantidadCiudades;
    private int max;

    public ListaCiudades(int max) {
        lc = new Ciudad[max];
        cantidadCiudades = 0;
        this.max = max;
    }

    public int getCantidadCiudades() {
        return cantidadCiudades;
    }

    public void setCantidadCiudades(int cantidadCiudades) {
        this.cantidadCiudades = cantidadCiudades;
    }

    public boolean insertarCiudad(Ciudad ciudad) {
        if (cantidadCiudades < max) {
            lc[cantidadCiudades] = ciudad;
            cantidadCiudades++;
            return true;
        } else {
            return false;
        }
    }

    public Ciudad getCiudadI(int i) {
        if (i >= 0 && i < cantidadCiudades) {
            return lc[i];
        } else {
            return null;
        }
    }

    @Override
    public String toString() {
        return "ListaCiudades [" + (lc != null ? "lc=" +
            Arrays.toString(lc) + ", " : "") +
            "cantidadCiudades=" + cantidadCiudades + ", max=" +
            max + "]";
    }
}
} //Fin ListaCiudades

```

Para imprimir el contenedor, es mejor construir el método toString en vez del que se obtiene con Eclipse o NetBeans. La idea es ir obteniendo elemento por elemento del contendor

```
package cl.ucn.ei.pa.paisciudades.logica;

public interface SistemaPaisCiudades {

    public void ingresarPais(String nombre, String idioma,
        int cantidadHabitantes, int cantidadCiudades);

    public boolean ingresarCiudadDeUnPais(String nombre,
        String alcalde, int cantidadHabitantes);

    public String obtenerDatosPaisCiudades();
}
```

```
package cl.ucn.ei.pa.paisciudades.logica;

import cl.ucn.ei.pa.paisciudades.dominio.*;

public class SistemaPaisCiudadesImpl implements
    SistemaPaisCiudades {

    private Pais pais;

    public SistemaPaisCiudadesImpl() {
        pais = null;
    }

    public void ingresarPais(String nombre, String idioma,
        int cantidadHabitantes, int cantidadCiudades){
        pais=new Pais(nombre,idioma,cantidadHabitantes,
            cantidadCiudades);
    }
}
```

```
public boolean ingresarCiudadDeUnPais(String nombre,
                                       String alcalde, int cantidadHabitantes){
    Ciudad ciudad = new Ciudad(nombre, alcalde,
                                cantidadHabitantes);

    if (pais != null){
        boolean ingreso =
            pais.getListaCiudades().insertarCiudad(ciudad);
        return ingreso;
    }
    else{
        throw new NullPointerException ("No existe pais");
    }
}

public String obtenerDatosPaisCiudades() {
    if (pais != null){
        return pais.toString();
    }
    else{
        throw new NullPointerException ("No existe pais");
    }
}
}
```

```

package cl.ucn.ei.pa.paisciudades.logica;

import ucn.Stdout;
import ucn.StdIn;
import cl.ucn.ei.pa.paisciudades.dominio.*;
import cl.ucn.ei.pa.paisciudades.logica.SistemaPaisCiudades;
import cl.ucn.ei.pa.paisciudades.logica.SistemaPaisCiudadesImpl;

public class App {

    public static void LeerCiudadesPais(
        SistemaPaisCiudades sistema) {

        StdOut.print("Ingrese cantidad ciudades Chile: ");
        int N = StdIn.readInt();

        sistema.ingresarPais("Chile", "Espanol", 17000000, N);

        for (int i = 1; i <= N ; i++) {
            StdOut.print("Ingrese nombre ciudad: ");
            String nombre = StdIn.readString();
            StdOut.print("Ingrese Alcalde: ");
            String alcalde = StdIn.readString();
            StdOut.print("Ingrese cantidad habitantes: ");
            int cantidad = StdIn.readInt();

            boolean ingreso =
                sistema.ingresarCiudadDeUnPais(nombre,
                    alcalde, cantidad);
            if(!ingreso){
                StdOut.println("No se ingreso ciudad.
                                No hay espacio");
            }
        }
    }
}

```

```

public static void main(String[] args) {
    SistemaPaisCiudades sistema=new SistemaPaisCiudadesImpl();
    LeerCiudadesPais(sistema);

    try {
        StdOut.println(sistema.obtenerDatosPaisCiudades());
        //En vez de tener obtenerDatosPaisCiudadades
        //se podría tener obtenerPaisCiudadades,
        //que retona un país y ahí obtener cada parte
        //del país y desplegarlas
    } catch (NullPointerException ex) {
        StdOut.println(ex.getMessage());
    }

} //Fin main

} //Fin App

```

Ejemplo 4

Se desea construir un programa que maneje información de países y de ciudades. Un país está relacionado con varias ciudades; una ciudad está relacionada con un único país.

Datos asociados a un país son:

- Nombre del país.
- Cantidad de habitantes.
- Idioma que se habla (sólo uno).
- Continente en que se encuentra.

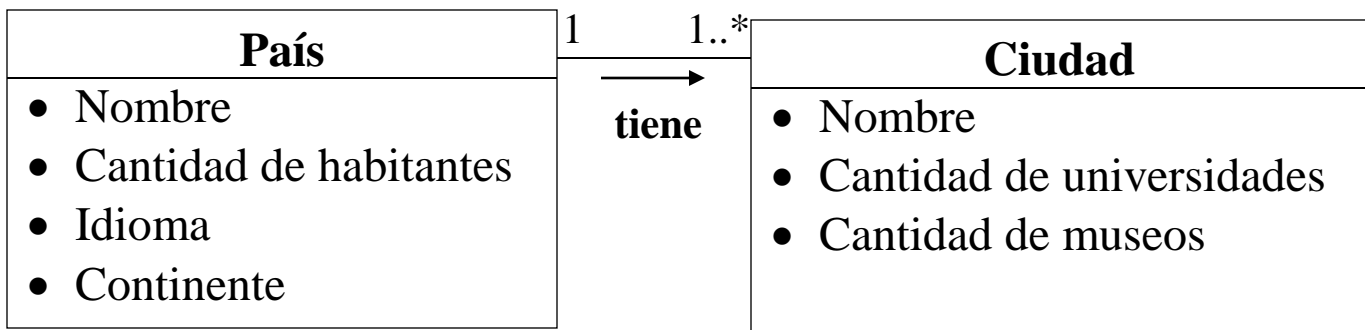
Datos asociados a una ciudad son:

- Nombre de la ciudad.
- Cantidad de universidades que tiene la ciudad.
- Cantidad de museos que tiene la ciudad.

Se pide, un programa que haga:

- Ingreso de datos de países y ciudades.
- Dado el nombre del país, encontrar el nombre de las ciudades relacionadas.
- Cantidad de países europeos (Continente = “Europa”).
- Indicar el nombre del país que tiene mayor cantidad de habitantes.
- Dado el nombre de una ciudad encontrar los datos del país relacionado.

Modelo del dominio



Contratos

Suponiendo que los datos se leen de la siguiente forma:

- 1°. Cantidad de países
- 2°. Por cada país los datos del país, incluyendo la cantidad de ciudades del país
- 3°. Por cada ciudad del país, los datos de la ciudad

Los contratos asociados al ingreso de los datos serían:

Operación	Ingresar país (nombre, cantidad habitantes, idioma, continente, cantidad de ciudades)
Descripción	Se ingresa un país a la lista de países
Precondiciones	
Postcondiciones	País ingresado

*Que exista espacio en el contenedor,
no es una precondición*

Operación	Ingresar ciudad (nombre, cantidad universidades, cantidad museos)
Descripción	Se ingresa una ciudad a la lista de ciudades
Precondiciones	
Postcondiciones	Ciudad ingresada

Operación	AsociarPaísCiudad (nombrePaís, nombreCiudad)
Descripción	Se asocia ciudad con país y el país con la ciudad
Precondiciones	<ul style="list-style-type: none"> • Que exista el país • Que exista la ciudad
Postcondiciones	Asociaciones hechas entre país y ciudad

El resto de los contratos son:

Operación	Encontrar ciudades de un país (nombre país)
Descripción	Se obtienen los datos de las ciudades
Precondiciones	Que exista el país
Postcondiciones	

Operación	Obtener cantidad países europeos
Descripción	Se obtiene la cantidad de países europeos
Precondiciones	
Postcondiciones	

Operación	Obtener datos del país con más habitantes
Descripción	Se obtienen los datos del país con la mayor
Precondiciones	
Postcondiciones	

Operación	Encontrar el país de una ciudad (nombre ciudad)
Descripción	Se obtiene los datos del país de una
Precondiciones	Que exista la ciudad
Postcondiciones	

Diagrama de clases del dominio de la aplicación

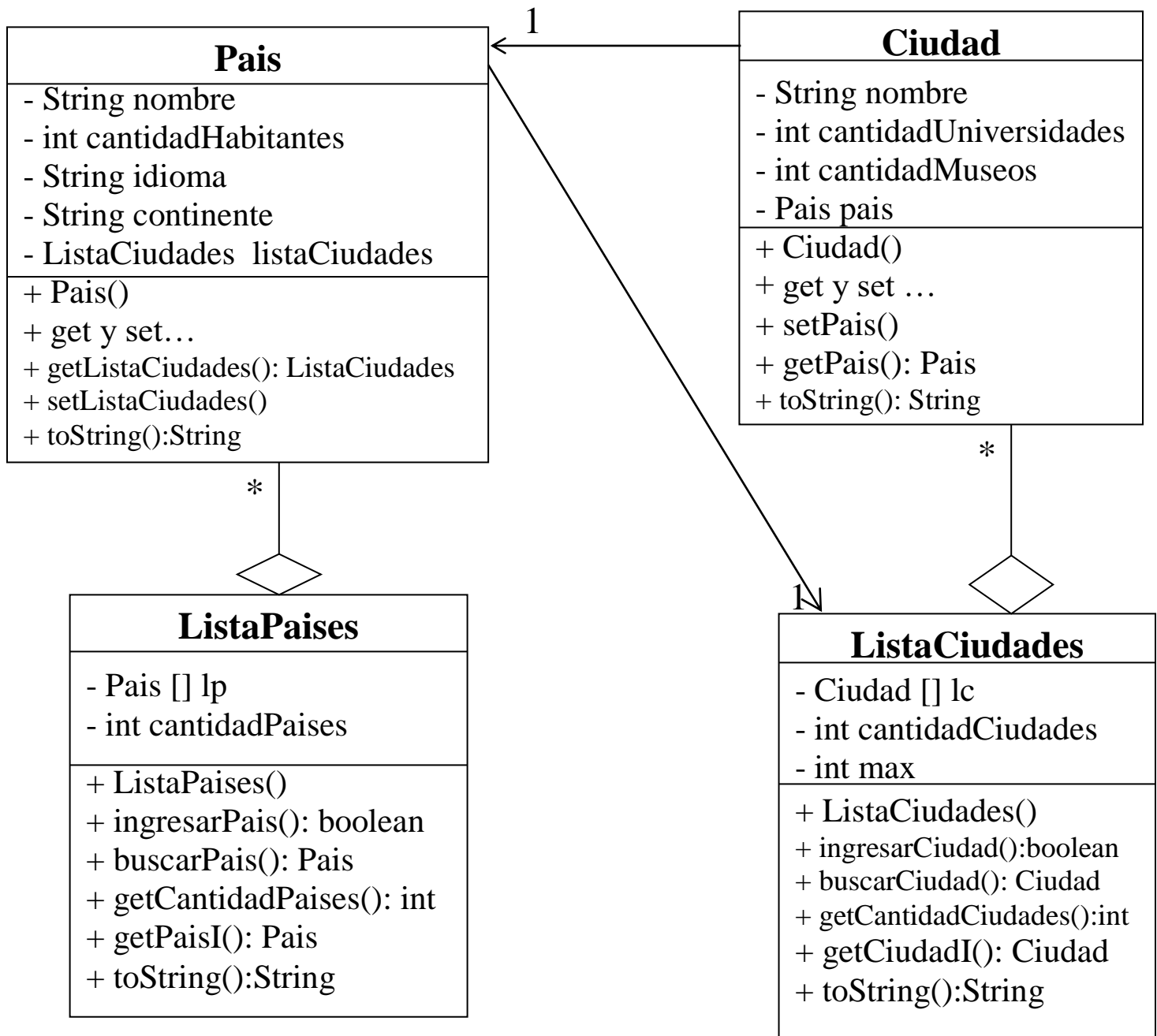
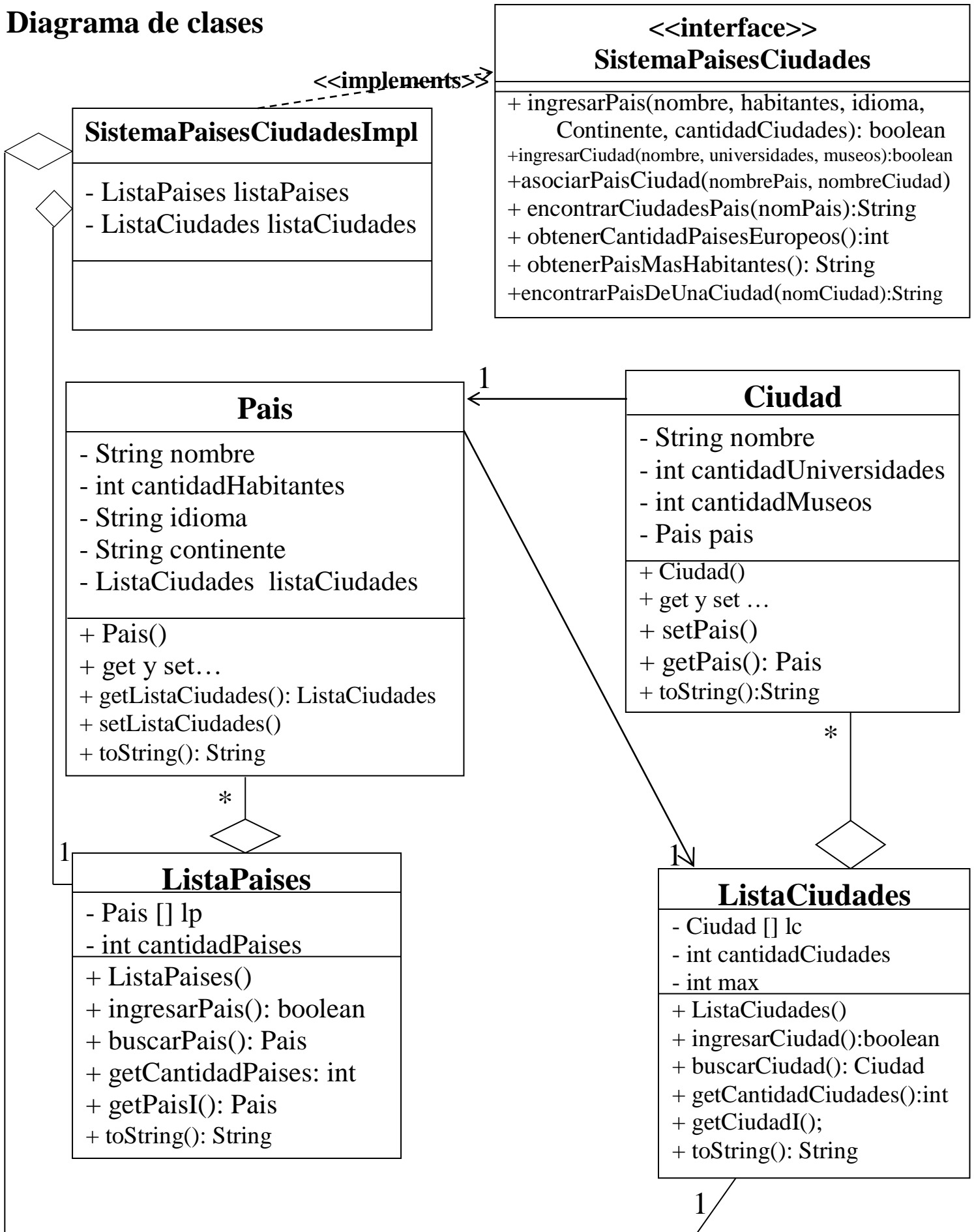


Diagrama de clases



```
package cl.ucn.ei.pa.sistemapaisesciudades.dominio;

public class Ciudad {
    private String nombre;
    private int cantidadUniversidades;
    private int cantidadMuseos;
    private Pais pais;

    public Ciudad(String nombre, int cantidadUniversidades,
                  int cantidadMuseos) {
        this.nombre = nombre;
        this.cantidadUniversidades = cantidadUniversidades;
        this.cantidadMuseos = cantidadMuseos;
        pais = null;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public int getCantidadUniversidades() {
        return cantidadUniversidades;
    }

    public void setCantidadUniversidades(int cantidadUniversidades) {
        this.cantidadUniversidades = cantidadUniversidades;
    }

    public int getCantidadMuseos() {
        return cantidadMuseos;
    }

    public void setCantidadMuseos(int cantidadMuseos) {
        this.cantidadMuseos = cantidadMuseos;
    }
}
```

```

    public Pais getPais() {
        return pais;
    }

    public void setPais(Pais pais) {
        this.pais = pais;
    }

    // @Override
    public String toString() {
        return "Ciudad [" + (nombre != null ? "nombre=" + nombre +
            ", " : "") +
            "cantidadUniversidades=" + cantidadUniversidades +
            ", cantidadMuseos=" + cantidadMuseos + "];"
    }
}

```

```

package cl.ucn.ei.pa.sistemapaísesciudades.dominio;
import cl.ucn.ei.pa.sistemapaísesciudades.logica.ListaCiudades;

public class Pais {
    private String nombre;
    private int cantidadHabitantes;
    private String idioma;
    private String continente;
    private ListaCiudades listaCiudades;
    private int cantidadCiudades;

    public Pais(String nombre, int cantidadHabitantes, String idioma,
        String continente, int cantidadCiudades) {
        this.nombre = nombre;
        this.cantidadHabitantes = cantidadHabitantes;
        this.idioma = idioma;
        this.continente = continente;
        this.cantidadCiudades = cantidadCiudades;
        listaCiudades = new ListaCiudades(cantidadCiudades);
    }
}

```

```
public String getNombre() {  
    return nombre;  
}  
  
public void setNombre(String nombre) {  
    this.nombre = nombre;  
}  
  
public int getCantidadHabitantes() {  
    return cantidadHabitantes;  
}  
  
public void setCantidadHabitantes(int cantidadHabitantes) {  
    this.cantidadHabitantes = cantidadHabitantes;  
}  
  
public String getIdioma() {  
    return idioma;  
}  
  
public void setIdioma(String idioma) {  
    this.idioma = idioma;  
}  
  
public String getContinente() {  
    return continente;  
}  
  
public void setContinente(String continente) {  
    this.continente = continente;  
}  
  
public ListaCiudades getListaCiudades() {  
    return listaCiudades;  
}  
  
public void setListaCiudades(ListaCiudades listaCiudades) {  
    this.listaCiudades = listaCiudades;  
}
```

```

public String toString(){
    String salida = nombre;
    for (int i=0; i<listaCiudades.getCantidadCiudades(); i++){
        salida = salida + listaCiudades.getCiudadI(i).toString();
    }
    return salida;
}
}

```

```

package cl.ucn.ei.pa.sistemapaisesciudades.logica;

import cl.ucn.ei.pa.sistemapaisesciudades.dominio.Ciudad;

public class ListaCiudades {
    private Ciudad []lc;
    private int cantidadCiudades;
    private int max;

    public ListaCiudades(int max){
        lc = new Ciudad [max];
        cantidadCiudades = 0;
        this.max = max;
    }

    public boolean ingresarCiudad(Ciudad ciudad){
        if (cantidadCiudades < max){
            lc[cantidadCiudades]= ciudad;
            cantidadCiudades ++;
            return true;
        }
        else{
            return false;
        }
    }

    public int getCantidadCiudades(){
        return cantidadCiudades;
    }
}

```

```
public Ciudad getCiudadI(int i){
    if (i >=0 && i < cantidadCiudades){
        return lc[i];
    }
    else{
        return null;
    }
}

public Ciudad buscarCiudad(String nombre){
    int i;
    for(i = 0; i < cantidadCiudades; i++){
        if (lc[i].getNombre().equals(nombre)){
            break;
        }
    }
    if (i == cantidadCiudades){
        return null;
    }
    else{
        return lc[i];
    }
}

public String toString() {
    String salida = "";
    for (int i = 0; i < cantidadCiudades; i++){
        salida = salida + lc[i].toString();
    }
    return salida;
}

}
```

```
package cl.ucn.ei.pa.sistemapaísesciudades.logica;

import cl.ucn.ei.pa.sistemapaísesciudades.dominio.Pais;

public class ListaPaises {
    private Pais []lp;
    private int cantidadPaises;
    private int max;

    public ListaPaises(int max){
        lp = new Pais [max];
        cantidadPaises = 0;
        this.max = max;
    }

    public boolean ingresarPais(Pais pais){
        if (cantidadPaises < max){
            lp[cantidadPaises]= pais;
            cantidadPaises ++;
            return true;
        }
        else{
            return false;
        }
    }

    public int getCantidadPaises(){
        return cantidadPaises;
    }

    public Pais getPaisI(int i){
        if (i >=0 && i < cantidadPaises){
            return lp[i];
        }
        else{
            return null;
        }
    }
}
```



```

public Pais buscarPais(String nombre){
    int i;
    for(i = 0; i < cantidadPaises; i++){
        if (lp[i].getNombre().equals(nombre)){
            break;
        }
    }
    if (i == cantidadPaises){
        return null;
    }
    else{
        return lp[i];
    }
}

public String toString() {
    String salida = "";
    for (int i = 0; i < cantidadPaises; i++){
        salida = salida + lp[i].toString();
    }
    return salida;
}
}

```

```

package cl.ucn.ei.pa.sistemapaisesciudades.logica;

public interface SistemaPaisesCiudades {

    public boolean ingresarPais(String nombre, int cantidadHabitantes,
        String idioma, String continente, int cantidadCiudades);

    public boolean ingresarCiudad(String nombre,
        int cantidadUniversidades, int cantidadMuseos);

    public void asociarPaisCiudad (String nombrePais,
        String nombreCiudad);

    public int obtenerCantidadPaisesEuropeos();

    public String paisMasHabitantes();

    public String paisDeterminadaCiudad(String nombreCiudad);

    public String ciudadesDeterminadoPais(String nombrePais);
}

```

```

package cl.ucn.ei.pa.sistemapaísesciudades.logica;

import cl.ucn.ei.pa.sistemapaísesciudades.dominio.*;

public class SistemaPaísesCiudadesImpl implements SistemaPaísesCiudades{

    private ListaPaíses listaPaíses;
    private ListaCiudades listaCiudades;

    public SistemaPaísesCiudadesImpl(int cantidadPaíses) {
        listaPaíses = new ListaPaíses(cantidadPaíses);
        listaCiudades = new ListaCiudades(1);
    }

    public boolean ingresarPaís(String nombre, int cantidadHabitantes,
                                String idioma, String continente, int cantidadCiudades){
        País país = new País(nombre, cantidadHabitantes, idioma,
                                continente, cantidadCiudades);
        boolean ingreso = listaPaíses.ingresarPaís(país);
        return ingreso;
    }

    public boolean ingresarCiudad(String nombre,
                                   int cantidadUniversidades, int cantidadMuseos){
        Ciudad ciudad = new Ciudad (nombre, cantidadUniversidades,
                                   cantidadMuseos);
        boolean ingreso = listaCiudades.ingresarCiudad(ciudad);
        return ingreso;
    }

    public void asociarPaísCiudad (String nombrePaís,
                                   String nombreCiudad){
        País país = listaPaíses.buscarPaís(nombrePaís);
        Ciudad ciudad = listaCiudades.buscarCiudad(nombreCiudad);
        if (país != null && ciudad != null){
            ciudad.setPaís(país);
            boolean ingreso =
                país.getListaCiudades().ingresarCiudad(ciudad);
        }
        else{
            throw new NullPointerException("No existe ciudad y/o país");
        }
    }
}

```

```
public int obtenerCantidadPaisesEuropeos(){
    int contadorPaisesEuropeos = 0;
    for (int i = 0; i < listaPaises.getCantidadPaises(); i++){
        if (listaPaises.getPaisI(i).getContinente().
                                                    equals("Europa")) {
            contadorPaisesEuropeos++;
        }
    }
    return contadorPaisesEuropeos;
}

public String paisMasHabitantes(){
    int mayor = -1;
    Pais paisMayor= null;
    for (int i = 0; i < listaPaises.getCantidadPaises(); i++){
        Pais pais = listaPaises.getPaisI(i);
        if (pais.getCantidadHabitantes() > mayor) {
            mayor = pais.getCantidadHabitantes();
            paisMayor = pais;
        }
    }
    if (paisMayor != null){
        return paisMayor.toString();
    }
    else {
        return "no existe pais con mas habitantes";
    }
}

public String paisDeterminadaCiudad(String nombre){
    Ciudad ciudad = listaCiudades.buscarCiudad(nombre);
    if (ciudad != null){
        return ciudad.getPais().getNombre();
    }
    else{
        throw new NullPointerException("No existe ciudad");
    }
}
```

```

public String ciudadesDeterminadoPais(String nombrePais){
    Pais pais = listaPaises.buscarPais(nombrePais);
    if (pais != null){
        return pais.getListaCiudades().toString();
    }
    else{
        throw new NullPointerException("No existe el pais");
    }
}
}

```

```

package cl.ucn.ei.pa.sistemapaísesciudades.logica;

import ucn.Stdout;
import ucn.StdIn;
import cl.ucn.ei.pa.sistemapaísesciudades.logica.SistemaPaisesCiudades;
import cl.ucn.ei.pa.sistemapaísesciudades.logica.SistemaPaisesCiudadesImpl;

public class App {

    public static void main(String[] args) {
        StdOut.print("Ingrese cantidad de países: ");
        int cantidadPaises = StdIn.readInt();
        SistemaPaisesCiudades sistema =
            new SistemaPaisesCiudadesImpl(cantidadPaises);

        LeerPaisesCiudades(sistema, cantidadPaises);

        int cantidad = sistema.obtenerCantidadPaisesEuropeos();
        StdOut.println("Cantidad de países europeos: "+ cantidad);

        StdOut.println("País mas habitantes "+
            sistema.paisMasHabitantes());

        StdOut.print("nombre ciudad para buscar el país asociado ");
        String nombreCiudad = StdIn.readString();
    }
}

```

[illegible]

Si los datos se leen de la siguiente forma:

- 1°. Cantidad de países, cantidad de ciudades
- 2°. Por cada país los datos del país
- 3°. Por cada ciudad, los datos de la ciudad, incluyendo el nombre del país al que pertenece

Los contratos ingresarPaís, ingresarCiudad y asociarPaísCiudad, asociados al ingreso de los datos serían reemplazados por los siguientes:

Operación	Ingresar país (nombre, cantidad habitantes, idioma, continente)
Descripción	Se ingresa un país a la lista de países
Precondiciones	
Postcondiciones	País ingresado

Operación	Ingresar ciudad de un país (nombre, cantidad universidades, cantidad museos, nombre país)
Descripción	<ul style="list-style-type: none"> • Se ingresa una ciudad a la lista de ciudades • Se asocia la ciudad con el país: <ul style="list-style-type: none"> ➤ La ciudad se ingresa a la lista de ciudades del país ➤ La ciudad queda asociada con el país
Precondiciones	Que exista el país
Postcondiciones	Ciudad ingresada en la lista de ciudades Ciudad asociada con país

Código que cambia

```
package cl.ucn.ei.pa.sistemapaisesciudades.logica;

public interface SistemaPaisesCiudades {

    public boolean ingresarPais(String nombre, int cantidadHabitantes,
                                String idioma, String continente);

    public boolean ingresarCiudadDeUnPais(String nombre,
                                           int cantidadUniversidades, int cantidadMuseos,
                                           String nombrePais);

    public int obtenerCantidadPaisesEuropeos();

    public String paisMasHabitantes();

    public String paisDeterminadaCiudad(String nombreCiudad);

    public String ciudadesDeterminadoPais(String nombrePais);

}
```

Parte de Clase SistemaPaisesCiudadesImpl

```
public class SistemaPaisesCiudadesImpl implements
    SistemaPaisesCiudades {

    private ListaPaises listaPaises;
    private ListaCiudades listaCiudades;

    public SistemaPaisesCiudadesImpl(int cantidadPaises,
                                       int cantidadCiudades) {
        listaPaises = new ListaPaises(cantidadPaises);
        listaCiudades = new ListaCiudades(cantidadCiudades);
    }

}
```



```

public boolean ingresarPais(String nombre, int cantidadHabitantes,
                             String idioma, String continente){

    Pais pais = new Pais(nombre, cantidadHabitantes, idioma,
                           continente);

    boolean ingreso = listaPaises.ingresarPais(pais);
    return ingreso;
}

public boolean ingresarCiudadDeUnPais(String nombre,
                                       int cantidadUniversidades, int cantidadMuseos,
                                       String nombrePais){

    Pais pais = listaPaises.buscarPais(nombrePais);
    if(pais== null){
        throw new NullPointerException("No existe el pais");
    }
    Ciudad ciudad = new Ciudad (nombre, cantidadUniversidades,
                                cantidadMuseos);

    boolean ingreso = listaCiudades.ingresarCiudad(ciudad);
    ciudad.setPais(pais);
    ingreso = pais.getListaCiudades().ingresarCiudad(ciudad);
    return ingreso;
}

.....

```

Parte de la App

```

public static void main(String[] args) {
    StdOut.print("Ingrese cantidad de paises: ");
    int cantidadPaises = StdIn.readInt();
    StdOut.print("Ingrese cantidad de ciudades: ");
    int cantidadCiudades = StdIn.readInt();
    SistemaPaisesCiudades sistema =
        new SistemaPaisesCiudadesImpl(cantidadPaises,
                                       cantidadCiudades);

    LeerPaises(sistema, cantidadPaises);
    LeerCiudades(sistema, cantidadCiudades);

    .....
}

```

```

public static void LeerCiudades(SistemaPaisesCiudades sistema,
                                int cantidadCiudades){

    for (int j = 0; j < cantidadCiudades ; j++){
        //Para cada ciudad
        StdOut.println("Ingrese datos de una ciudad ");
        StdOut.print("Nombre ciudad: ");
        String nombreCiudad = StdIn.readString();
        StdOut.print("Cantidad de universidades: ");
        int cantidadUniversidades = StdIn.readInt();
        StdOut.print("Cantidad de museos: ");
        int cantidadMuseos = StdIn.readInt();
        StdOut.print("nombre del pais al que pertenece: ");
        String nombrePaisPertenece = StdIn.readString();
        try{
            boolean ingreso =
                sistema.ingresarCiudadDeUnPais(nombreCiudad,
                cantidadUniversidades, cantidadMuseos,
                nombrePaisPertenece);
            //Se ingresa la ciudad a la lista de todas
            //las ciudades
            if(!ingreso){
                StdOut.println("No se ingreso la ciudad.
                                No hay espacio");
            }
        }
        catch(NullPointerException ex){
            StdOut.println(ex.getMessage());
        }
    }
}

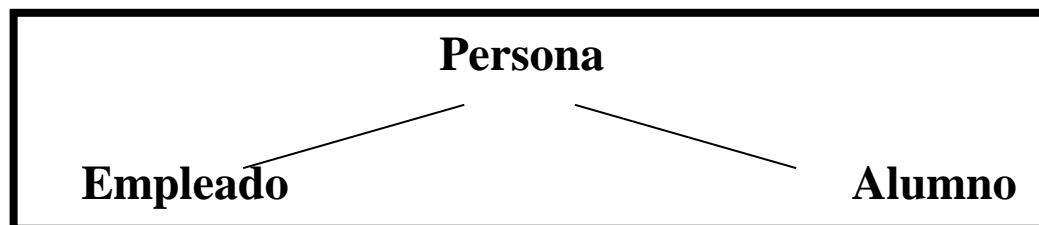
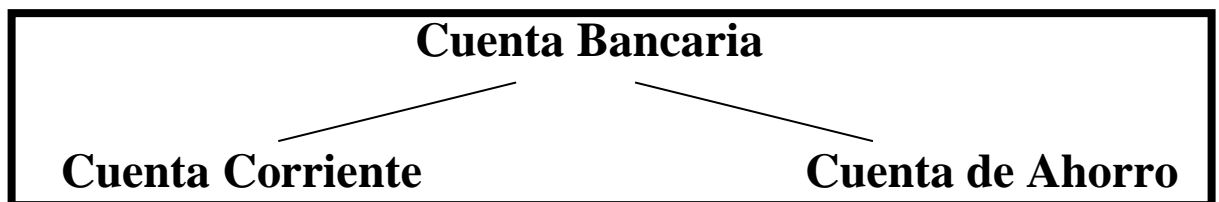
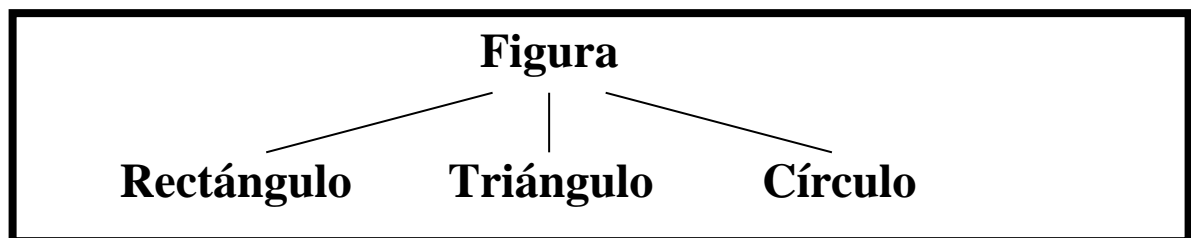
```


2.8. Herencia en Java

2.8.1. Concepto

- Se permite definir una nueva clase B, como una extensión de una clase previa A.
- B se denomina **subclase** de A y A se denomina **superclase** de B
- B es la **Clase derivada** y A es la **Clase base**
- Una subclase típicamente aumenta o redefine la estructura existente y el comportamiento de su superclase.

Ejemplos:

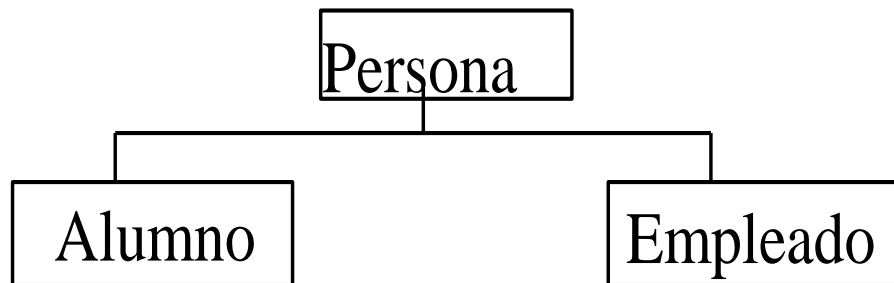


Ejemplo

Se tiene una jerarquía de clases, formada por una clase base (Persona) y dos clases derivadas (Empleado y Alumno), junto a la correspondiente definición de las propiedades, variables de instancia, y métodos, de cada una de ellas.

Como las clases Empleado y Alumno heredan propiedades de la superclase Persona, los objetos asociados a estas clases tienen las propiedades definidas para la clase propiamente tal, y las heredadas de la clase Persona.

Superclase Persona, con subclase Alumno y Empleado



Persona

Variables de instancia:

- nombre
- direccion
- fecha de nacimiento

Métodos:

- Inic-datos-persona
- Informa-datos-persona

Empleado

Variables de instancia:

- num-empleado
- sueldo

Métodos:

- Inic-datos-empleado
- Informa-sueldo

Alumno

Variables de instancia:

- num-alumno
- carrera

Métodos:

- Inic-datos-alumno
- Informa-carrera

Se recomienda no más de 3 niveles

Herencia de propiedades en una jerarquía de clases



La habilidad de un lenguaje para soportar la herencia, distingue a los lenguajes orientados al objeto, de los lenguajes basados en objetos.

Las **reglas de herencia**, permiten que un objeto de una subclase B puede aparecer, donde quiera que un objeto de una superclase A es esperado.

Herencia simple: Hay una única superclase.

Herencia Múltiple: Hay varias superclases.

- ¡ JAVA tiene herencia simple !
- Smalltalk : Herencia Simple
- C++ : Herencia Múltiple
- Eiffel : Herencia Múltiple

Las **reglas de herencia** son:

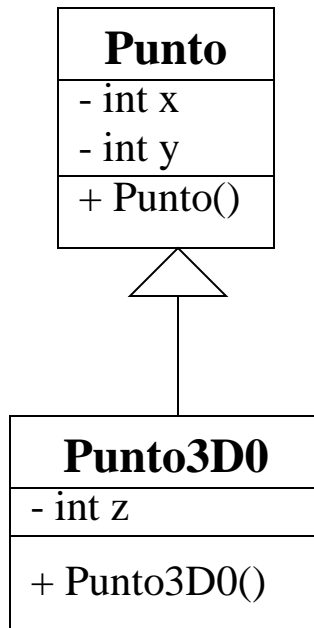
- La herencia simple genera una jerarquía. En la raíz de la jerarquía se encuentra la clase Object.
- Todas las clases, excepto la clase Object, tienen exactamente una superclase.
- Una subclase hereda variables y métodos de su superclase.
- Una subclase puede declarar variables y métodos, diferentes de aquellos heredados.
- Los métodos en la subclase, pueden redefinir a los métodos heredados (“OVERRIDE”).
- Una variable heredada también se puede redefinir.

Ejemplo:

```
public class Punto{
```

```
    private int x;
    private int y;
```

```
    public Punto(int x, int y){
        this.x = x;
        this.y = y;
    }
}
```



De la super clase los atributos se colocan privados y los métodos pudieran ser protegidos

Los atributos se colocan privados, porque para eso se tienen los get y set.

```
class Punto3D0 extends Punto {
```

```
    private int z;
```

La rutina constructora no se hereda

```
    public Punto3D0(int x, int y, int z) {
        super(x,y); //Se ejecuta la rutina constructora del padre
        this.z = z;
    }
```

```
    public Punto3D0() {
        this(-1, -1, -1);
    }
}
```

protected

La declaración es visible a la misma clase y a sus subclases

Si en vez de super (x,y), estuviera:

- *this.x = x;*
- *this.y = y;*

No compila:

- *Como x e y son privados no se pueden ver, aunque se hereden*
- *Se requiere el super*

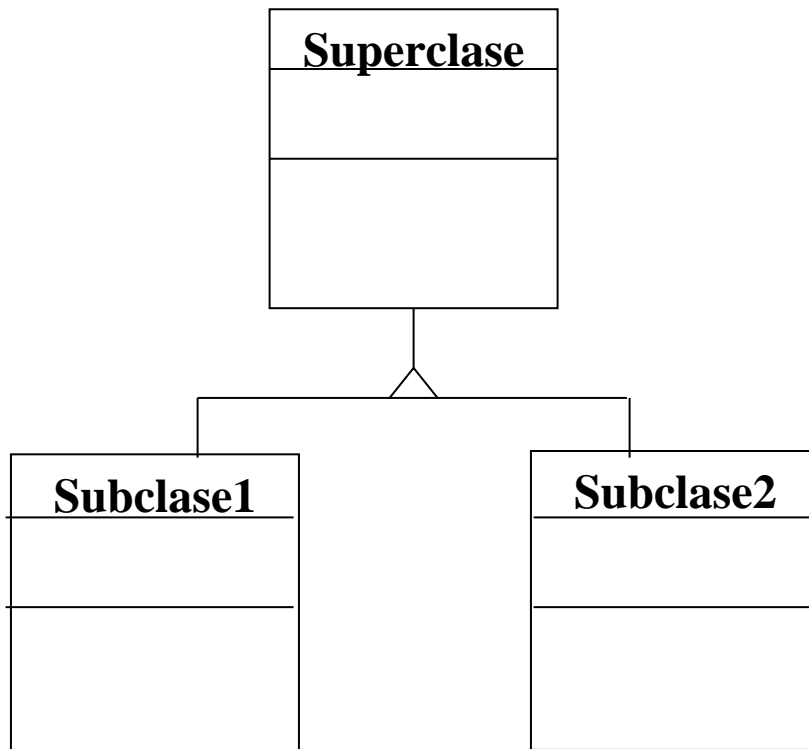
Aunque la visibilidad fuese protected para los atributos x e y, siempre se requiere invocar la rutina constructora del padre. Esto se hace a través del super en la rutina constructora del hijo

En JAVA, si la cláusula **extends** es omitida en la declaración de una clase, ésta tendrá en forma implícita a la clase Object como su superclase inmediata, que es la raíz de la jerarquía de clases de JAVA.

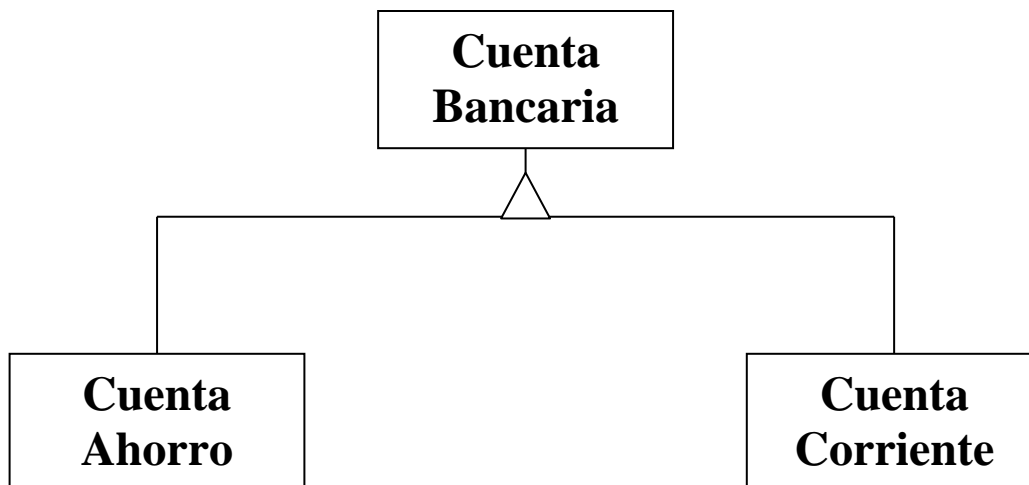
De esta manera, todas las clases son subclases directas o indirectas de la clase Object.

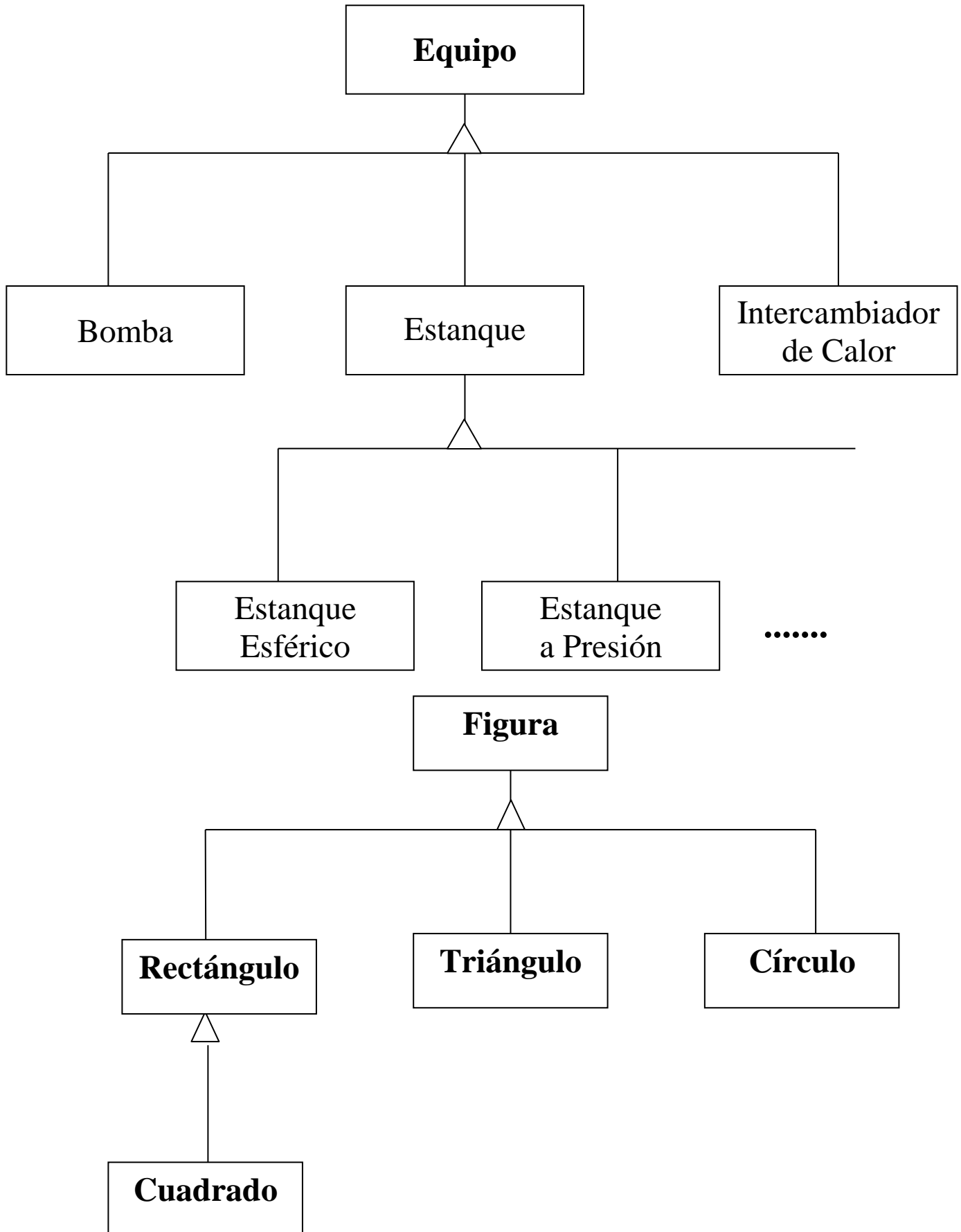
Notación

La representación de la herencia, de acuerdo a la notación utilizada por UML.



Ejemplos:





Mensajes a super

Cuando existe sobreescritura de métodos y/o atributos, la palabra **super** permite tener acceso a los miembros públicos y protegidos (métodos y atributos) de la clase padre.

Si quiero obtener un atributo sobreescrito privado de la clase padre, se usa ***super.getAtributo()***

Cuando una subclase sobrescribe un método heredado, el nombre especial **super**, permite que el método sobrescrito sea usado, es decir usar el método del padre.

Dentro de una subclase, un mensaje a super, invoca el método que la superclase debería usar para ese mensaje.

Ejemplo: Se define la subclase Punto3D, que utiliza el constructor super para inicializar las variables x e y, después imprime el punto en 3D resultante.

```
public class Punto {
    private int x;
    private int y;

    public Punto(int x, int y){
        this.x = x;
        this.y = y;
    }

    public int getX(){
        return x;
    }

    public int getY(){
        return y;
    }
}
```

Recuerde que los atributos de la super clase se colocan privados, ya que para eso se tienen los métodos get y set asociados . Los métodos pudieran ser protegidos

```
public class Punto3D extends Punto {  
    private int z;  
  
    public Punto3D(int x, int y, int z) {  
        //llamado al constructor Punto(x,y)  
        super(x, y);  
        //Se invoca el método que la superclase debería usar  
        //para este mensaje  
        this.z = z;  
    }  
  
    public int getZ(){  
        return z;  
    }  
}
```



```
public class App {  
  
    public static void main (String args[]) {  
        Punto3D p = new Punto3D(10,20,30);  
        StdOut.println("x = " + p.getX() + " y = " + p.getY() + " z = " +  
                                p.getZ());  
    }  
}
```

Resultado:

x =10 y = 20 z =30

Ejemplo:

```

public class Punto {
    private int x, y;
    public Punto(int x, int y){
        this.x = x;
        this.y = y;
    }
    public int getX(){
        return x;
    }
    public int getY(){
        return y;
    }
}

public class Punto3D extends Punto {
    private int z;
    public Punto3D(int x, int y, int z) {
        super(x, y); //llamado al constructor Punto(x,y)
        this.z = z;
    }
    public int getZ(){
        return z;
    }
}

public class Punto3DConstruct {
    public static void main(String args[]) {
        Punto3D p = new Punto3D(10, 20, 30);
        StdOut.println("x = " + p.getX() + " y = " +
            p.getY() + " z = " + p.getZ());
        StdOut.println("p = " + p);
    }
}

```

Resultado:

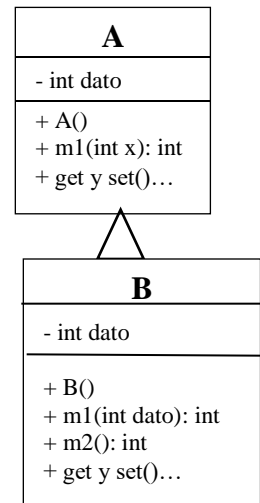
x = 10 y = 20 z = 30	
p = <u>Punto3D</u>	@ <u>1cc807</u>
Clase del Objeto	Dirección

```

public class A {
    private int dato;

    public A(int dato) {
        this.dato = dato;
    }
    public int getDato() {
        StdOut.println("Pasa por getDato() del padre");
        return dato;
    }
    public void setDato(int dato) {
        this.dato = dato;
    }
    public int m1(int x) {
        StdOut.println("Pasa por m1 del padre");
        return x + 5;
    }
}

```



```

public class B extends A{
    private int dato;

    public B(int dato, int dato1) {
        super(dato);
        this.dato = dato1;
    }
    public int getDato() {
        StdOut.println("Pasa por getDato del hijo");
        return dato;
    }
    public void setDato(int dato) {
        this.dato = dato;
    }
    public int m1(int dato) {
        StdOut.println("Pasa por m1 del hijo");
        return super.m1(dato); //Invoca al m1 sobreescrito del padre
    }
    public int m2() {
        StdOut.println("Pasa por m2");
        return super.m1(3) + super.getDato();
        //Invoca al m1 sobreescrito del padre y
        //al dato sobreescrito del padre
    }
}

```

```

public class App {

    public static void main(String[] args) {
        A a = new A(5);
        B b = new B(3,2);

        StdOut.println("a.m1(5)= " + a.m1(5));

        StdOut.println("b.m1(2)= " + b.m1(2));

        StdOut.println("b.m2()= " + b.m2());
    }
}

```

Se imprime

```

Pasa por m1 del padre
a.m1(5)= 10
Pasa por m1 del hijo
Pasa por m1 del padre
b.m1(2)= 7
Pasa por m2
Pasa por m1 del padre
Pasa ppor getDato() del padre
b.m2()= 11

```

Ejemplo

La clase Punto3D hereda de su superclase Punto la implementación del método de la distancia para puntos en 2D. Necesita por lo tanto sobrescribir esa definición con una nueva para puntos en 3D.

En la clase Punto3D hay sobrecarga de la distancia en 3D y hay redefinición (sobrescritura) de la distancia en 2D.

```
public class Punto {
    private int x;
    private int y;

    public Punto(int x, int y) {
        this.x=x;
        this.y=y;
    }

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }

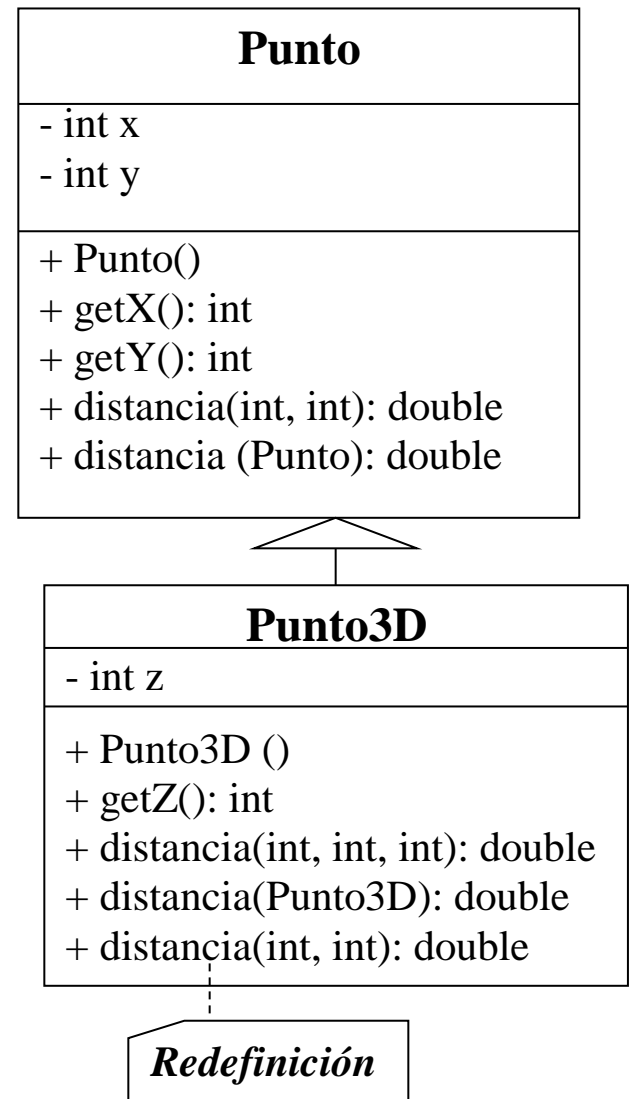
    public double distancia
        (int x, int y) {
        StdOut.println("Primer método
        distancia del padre Punto.
        Parametros: x,y");
        int dx = this.x - x;
        int dy = this.y - y;
        return Math.sqrt(dx*dx + dy*dy);
    }

    public double distancia(Punto p) {
        System.out.println("Segundo
        metodo distancia del padre
        Punto. Parametros: Punto");
        return this.distancia(p.x, p.y);
    }
}

//Fin clase Punto
```

*Si el objeto que invoca es de la clase Punto, se invoca el primer método de distancia de la clase Punto.
Si el objeto que invoca es de la clase Punto3D, se invoca el tercer método de distancia de la clase Punto3D*

*Si hay distintas versiones de un método, se deben colocar todas en el diagrama de clases.
Los métodos en el diagrama de clases deben llevar los parámetros*



```
public class Punto3D extends Punto{
    private int z;
```

```
    public Punto3D(int x, int y, int z) {
        super(x, y);
        this.z = z;
    }
```

El super es necesario para invocar la rutina constructora del padre

```
    public int getZ() {
        return z;
    }
```

```
    public double distancia(int x, int y, int z) {
        System.out.println("Primer metodo distancia del hijo
                             Punto3D. Parametros: x, y, z");

        int dx = this.getX() - x;
        //Se tiene que usar el getX(), porque es private
        int dy = this.getY() - y;
        int dz = this.z - z;
        return Math.sqrt(dx*dx + dy*dy + dz*dz);
    }
```

```
    public double distancia(Punto3D otro) {
        System.out.println("Segundo metodo distancia del hijo
                             Punto3D. Parametros: Punto3D");

        return this.distancia(otro.getX(), otro.getY(), otro.z);
    }
```

Si el objeto que invoca es de la clase Punto3D, se invoca el primer método de distancia de la clase Punto3D.

```
    public double distancia(int x, int y) {
        System.out.println("Tercer metodo distancia del hijo
                             Punto3D. Parametros: x, y");

        double dx = (double) this.getX() / z - x;
        double dy = (double) this.getY() / z - y;
        return Math.sqrt(dx*dx + dy*dy);
    }
```

```
}
```



```

public class Punto3DDist {
    public static void main(String[] args) {
        Punto3D p1 = new Punto3D(30, 40, 10);
        Punto3D p2 = new Punto3D(0, 0, 0);
        Punto p = new Punto(4, 6);
    }
}

```

<p><u>p2: Punto3D</u></p> <p>x = 0</p> <p>y = 0</p> <p>z = 0</p>	<p><u>p: Punto</u></p> <p>x = 4</p> <p>y = 6</p>	<p><u>p1: Punto3D</u></p> <p>x = 30</p> <p>y = 40</p> <p>z = 10</p>
--	--	---

```

        StdOut.println("p1 = " + p1.getX() + ", " + p1.getY() +
                        ", " + p1.getZ());

        StdOut.println("p2 = " + p2.getX() + ", " + p2.getY() +
                        ", " + p2.getZ());

        StdOut.println("p = " + p.getX() + ", " + p.getY());

        StdOut.println("-----");
        StdOut.println("p1.distancia(p2) = " + p1.distancia(p2));
        StdOut.println("-----");
        StdOut.println("p1.distancia(4, 6) = " + p1.distancia(4, 6));
        StdOut.println("-----");
        StdOut.println("p1.distancia(p) = " + p1.distancia(p));
    }
}

```

Se imprime

```

p1 = 30, 40, 10
p2 = 0, 0, 0
p = 4, 6

```

```

-----
Segundo metodo distancia del hijo Punto3D. Parametros:
Punto3D

```

```

Primer metodo distancia del hijo Punto3D. Parametros: x, y, z
p1.distancia(p2) = 50.99019513592785

```

```

-----
Tercer metodo distancia del hijo Punto3D. Parametros: x, y
p1.distancia(4, 6) = 2.23606797749979

```

```

-----
Segundo metodo distancia del padre Punto. Parametros: Punto
Tercer metodo distancia del hijo Punto3D. Parametros: x, y
p1.distancia(p) = 2.23606797749979

```

Notas:

Hay una tensión importante, entre la herencia y el encapsulamiento.

El uso de la herencia expone alguno de los secretos de la clase heredada.

En la práctica, esto significa que para comprender el significado de una clase particular, a menudo se deben estudiar todas sus superclases, incluyendo a veces su vista interna.

2.8.2. Selección de método dinámica (Polimorfismo)

Cuando se aplica un método a un objeto, el tipo del objeto se comprueba durante la compilación, para asegurarse de que el método llamado existe en la clase declarada.

Durante la ejecución, el objeto puede corresponder a alguna subclase del objeto declarado. En este caso y cuando la subclase sobrescriba (redefina) el método al que se llama, se utiliza la instancia real para decidir a qué método llamar.

Polimorfismo = “LATE BINDING”
(enlace tardío)

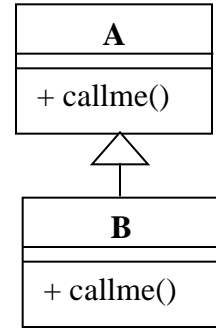
Polimorfismo en Java:

En Java todas las variables son polimórficas.

Una variable declarada de una cierta clase, puede mantener valores de cualquiera de sus subclases.

Ejemplo: Dos clases tienen una relación subclase/superclase, con un único método que se sobrescribe en la subclase.

```
public class A {
    public void callme() {
        StdOut.println("Inside A's callme method");
    }
}
```



```
public class B extends A{
    public void callme(){
        StdOut.println("Inside B's callme method");
    }
}
```

```
public class Dispatch {
    public static void main(String args[]) {
        A a = new B(); //a pertenece a la clase A.
                        // Se ha almacenado una referencia a una
                        //instancia de la clase B en ella.
        a.callme();    //invoca a callme de la clase B.
    }
}
```

El objeto corresponde al de la subclase. Recuerde que la regla de herencia dice que un objeto de una subclase B puede aparecer donde quiera que un objeto de una superclase A es

Resultado
Inside B's callme method

Callme tiene que estar declarado en la clase A. Esto se chequea en tiempo de compilación

El polimorfismo, es uno de los mecanismos más poderosos que ofrece la orientación al objeto, para soportar la reutilización del código y la robustez.

El ambiente de programación, proporciona un gran número de clases, las que pueden ser adaptadas, mediante la herencia, a las necesidades de una aplicación en particular.

Cambios en el main para ejercicio de la clase padre Punto y clase hija Punto3D

```
.....
Punto3D p2 = new Punto3D(0, 0, 0);
Punto p4 = new Punto3D(40, 50, 60); //Cumple la regla de herencia
Punto3D p5 = (Punto3D) p4;
StdOut.println("p2="+p2.getX()+" ", " +p2.getY()+"", " + p2.getZ());
StdOut.println("p4 = " + p4.getX() + " , " + p4.getY());
//No se puede desplegar z en p4
StdOut.println("p5="+p5.getX()+"", " +p5.getY()+"", " +p5.getZ());
StdOut.println("-----")
```

En tiempo de compilación se chequea que **distancia(Punto3D)** esté definido en la clase **Punto**, ya que la declaración de **p4** es **Punto**. Esto se cumple, porque en la clase **Punto** está el método **distancia(Punto)**. Se debe considerar que un objeto de la clase **Punto3D** también es un objeto de la clase **Punto**.

En tiempo de ejecución se ejecuta el segundo método **distancia** de la clase **Punto**, **distancia(Punto)**, el cual invoca a **this.distancia(int, int)**. Este último método está tanto en la clase **Punto** como en la clase **Punto3D**, está redefinido en **Punto3D**. Se aplica el de la clase **Punto3D**, porque en tiempo de ejecución **p4** es una instancia de **Punto3D**

```
StdOut.println("p4.distancia(p2) = " + p4.distancia(p2));
StdOut.println("-----")
```

En tiempo de compilación se chequea que **distancia(Punto3D)** esté definido en la clase **Punto3D**, ya que **p5** es de la clase **Punto3D**.

En tiempo de ejecución se ejecuta el segundo método **distancia** de la clase **Punto3D**, **distancia(Punto3D)**, el cual invoca a **this.distancia(int, int, int)**. Este último método corresponde al primer método **distancia** de la clase **Punto3D**.

```
StdOut.println("p5.distancia(p2) = " + p5.distancia(p2));
StdOut.println("-----")
```

```
StdOut.println("p5.distancia(1,1,2) = " + p5.distancia(1,1,2));
```

Se imprime

```
p2 = 0, 0, 0
```

```
p4 = 40, 50
```

```
p5 = 40, 50, 60
```

```
-----
Segundo metodo distancia del padre (Punto). Parametros: Punto
```

```
Tercer metodo distancia del hijo Punto3D. Parametros: x, y
```

```
p4.distancia(p2) = 1.0671873729054748
```

```
-----
Segundo metodo distancia del hijo Punto3D. Parametros: Punto3D
```

```
Primer metodo distancia del hijo Punto3D. Parametros: x, y, z
```

```
p5.distancia(p2) = 87.74964387392122
```

```
-----
Primer metodo distancia del hijo Punto3D. Parametros: x, y, z
```

```
p5.distancia(1,1,2) = 85.3580693314932
```

```
-----
```

2.8.3. Clase abstractas

A) Conceptos

- Algunas de las clases tendrán instancias, otras no.
- La clase que no tienen instancias, se denomina **clases abstractas**.
- Una clase abstracta es escrita con el propósito de que sus subclasses le agreguen estructura y comportamiento, a través de completar la implementación de sus (usualmente) métodos incompletos.
- No se pueden crear instancias de dichas clases directamente, con el operador **new**.

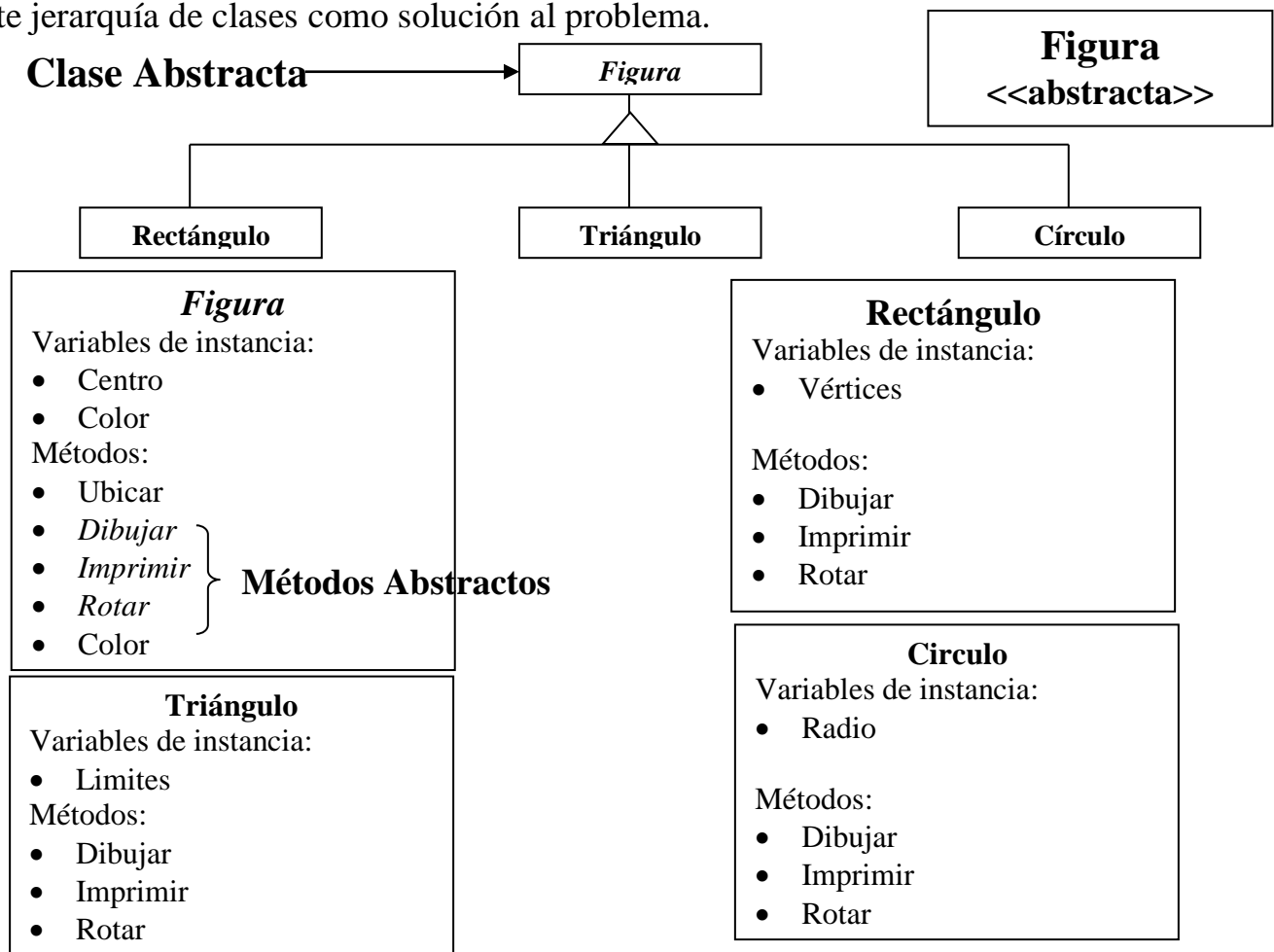
Ejemplo:

Se desea construir una aplicación que manipula figuras (círculos, triángulos y rectángulos) en un sistema gráfico, en base a las siguientes operaciones:

- Indicar ubicación de una figura.
- Dibujar una figura en el terminal.
- Imprimir una figura.
- Rotar una figura.
- Indicar el color de una figura.

Para que una clase se defina como abstracta debe tener al menos un método abstracto. Sino lo tiene, no necesita ser abstracta

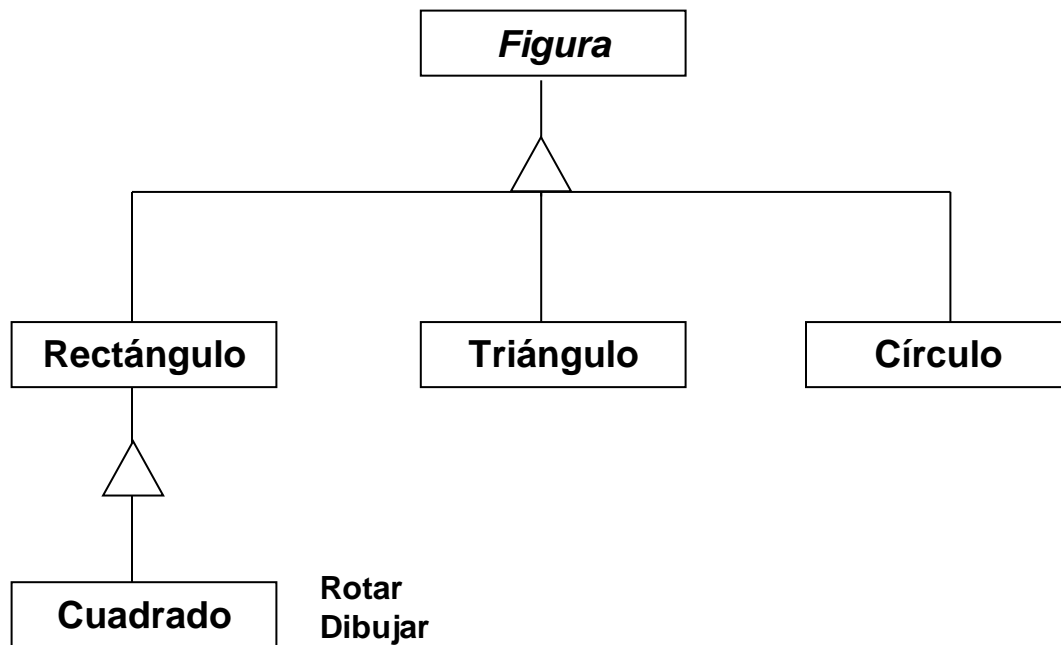
Las características del modelo orientado al objeto para definir clases y hacer explícitas las propiedades comunes de las clases mediante el mecanismo de la herencia, permite obtener la siguiente jerarquía de clases como solución al problema.



Debe hacerse notar, que las clases Rectángulo, Triángulo y Círculo, sobrescriben los procedimientos Dibujar, Imprimir y Rotar, los que son específicos para cada una de las clases. En esta solución, según sea el tipo de la información que se está procesando, se ejecuta el procedimiento adecuado para la situación que se presenta; por ejemplo, si el método Rotar se aplica a un objeto de la clase Rectángulo, el método asociado a esa clase es ejecutado.

Para extender o reusar el sistema, basta con crear una subclase dentro de la jerarquía de clases mostrada anteriormente, que agregue o modifique propiedades de la superclase, sin necesidad de modificar las clases existentes.

Por ejemplo, si se desea incorporar la clase Cuadrado al sistema, sólo se modifica la definición de clases anterior.



La nueva clase sobrescribe los métodos Dibujar y Rotar, por procedimientos específicos que se aplican solamente a las instancias de la clase Cuadrado.

Clase abstracta

Es una clase que nunca es instanciada. Contiene los métodos comunes a todas las subclases.

Es usada como una base, a partir de la cual, otras clases pueden heredar.

Método Abstracto

Es un método cuya implementación no es definida en la declaración de la clase en que aparece. Su definición se hace en una clase descendente.

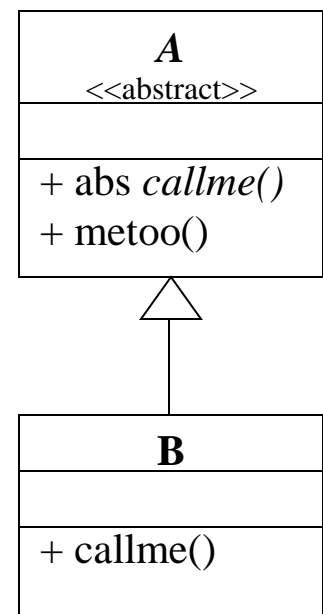
Ejemplo: Se tiene una clase con un método abstracto, seguida de una clase que implementa ese método.

```
public abstract class A { //clase abstracta
    public abstract void callme(); //método abstracto

    public void metoo(){
        StdOut.println("Inside A's metoo method");
    }
}
```

```
public class B extends A {
    public void callme(){
        StdOut.println("Inside B's callme method");
    }
}
```

```
public class Abstract {
    public static void main(String args[]){
        A a = new B();
        a.callme(); //Se usa el del hijo, ya que el objeto es una
                    //instancia de la subclase
        a.metoo();
    }
}
```



Resultado

```
Inside B's callme method
Inside A's metoo method
```

Modificador final, para calificar una variable, método o clase

Por defecto todos los métodos y las variables de instancia, se pueden sobrescribir.

En Java, si se desea declarar que ya no se quiere que las subclases sobrescriban las variables o métodos, esto se puede declarar como **final**.

- **Ejemplo para variables:**

```
final int NUEVOARCHIVO= 1;
```

Por convención los nombres de los valores constantes están en mayúscula.

- **final** asociado a un método, indica que este no puede ser sobrescrito por las subclases.
- **final**, usado como modificador de una clase, indica que la clase no puede tener subclases.

```
public final class Integer
```

```
public final class Math
```


B)Ejemplos

Ejemplo 1: Cuenta bancaria

En un banco se manejan cuentas de ahorro y cuentas corrientes. Estos dos tipos de cuenta son idénticos, excepto en las siguientes diferencias:

- Las cuentas de ahorro se numeran como 1xxxxx, empezando por 100001.
- Las cuentas corrientes se numeran como 5xxxxx, empezando por 500001.
- Las cuentas de ahorro no tienen un cargo mensual (mantención), si el saldo es superior a \$200.000. Si no, se cobran \$5.000 por mes. Las cuentas corrientes pagan una cuota mensual de \$5.000
- Las cuentas corrientes no pagan ningún interés, hasta que el saldo supera los \$500.000. A las cuentas de ahorro siempre se les paga interés.
- La aplicación muestra el resultado de ingresar \$100.000 al mes, tanto en una cuenta corriente como en una cuenta de ahorro a una cierta tasa de interés anual. Esto se hace por 10 meses. Se debe chequear que la tasa de interés esté entre 0 y 20.

Modelo del dominio

Cuenta de ahorro
número de cuenta
saldo
tasa de interés

Cuenta corriente
número de cuenta
saldo
tasa de interés

Contratos

Operación	depositar (monto depósito)
Descripción	Se deposita dinero en la cuenta corriente y en la cuenta de ahorro
Precondiciones	Que exista la cuenta corriente Que exista la cuenta de ahorro
Postcondiciones	Saldo actualizado de la cuenta corriente Saldo actualizado de la cuenta de ahorro

Recuerde que el chequeo del monto del depósito > 0 es una condición de la lógica del problema, no del sistema. Por lo tanto no va en el contrato

Operación	Descontar mantención ()
Descripción	Se descuenta al saldo de las 2 cuentas el valor de la mantención, según corresponda
Precondiciones	Que existan las 2 cuentas bancaria (la corriente y la de ahorro)
Postcondiciones	Saldo actualizado de c/u de las cuentas

Operación	Agregar interés (tasa de interés)
Descripción	Se agrega al saldo de las 2 cuenta el monto asociado a los intereses, según corresponda
Precondiciones	Que existan las 2 cuentas bancaria (la corriente y la de ahorro)
Postcondiciones	Saldo actualizado de c/u de las cuentas

Operación	Chequear tasa de interés (tasa interés)
Descripción	Se chequea que la tasa de interés esté entre 0 y 20
Precondiciones	
Postcondiciones	Tasa de interés válida

Operación	Obtener datos cuenta ahorro
Descripción	Se obtienen los datos de la cuenta de ahorro
Precondiciones	Que exista la cuenta de ahorro
Postcondiciones	

Operación	Obtener datos cuenta corriente
Descripción	Se obtienen los datos de la cuenta corriente
Precondiciones	Que exista la cuenta corriente
Postcondiciones	

Diagrama de clases del dominio de la aplicación

El depositar y girar podrían estar solo en la clase que implementa la interfaz, de manera que la clase CuentaBancaria, quede solo con los get y set

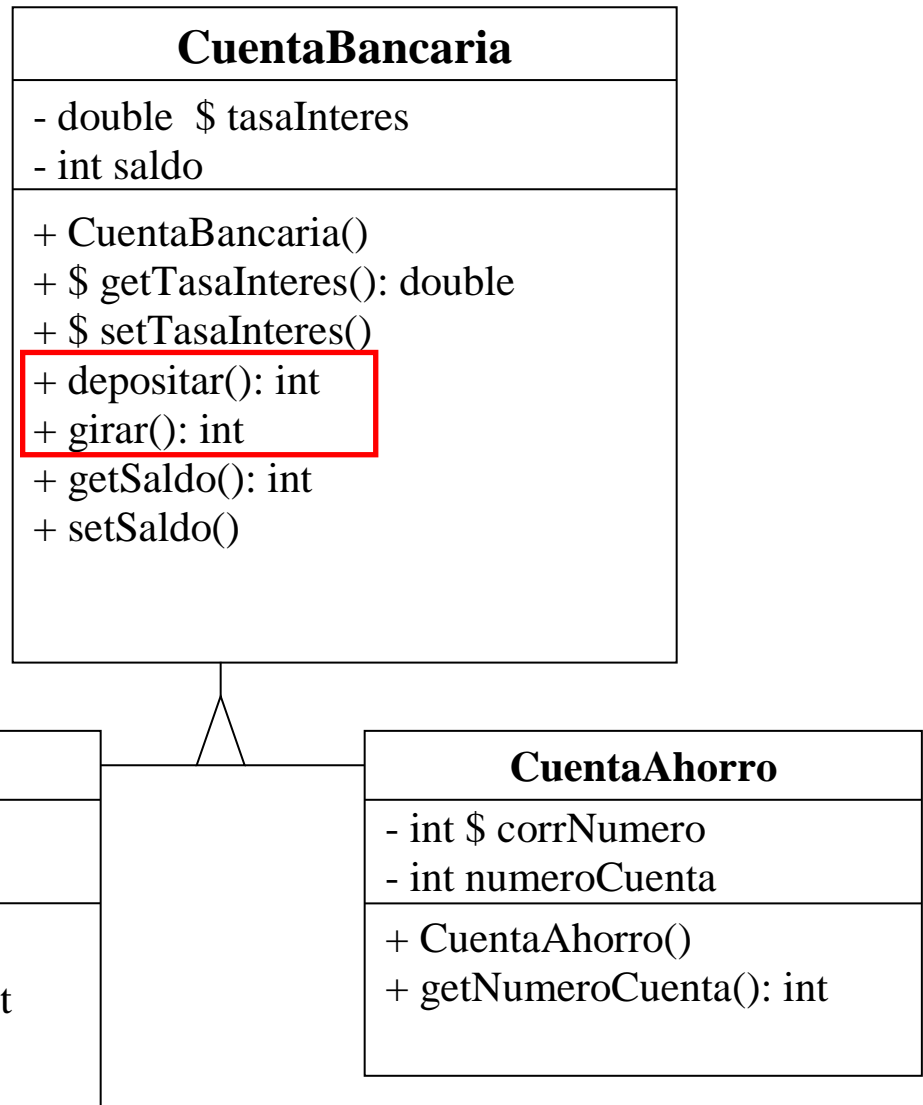
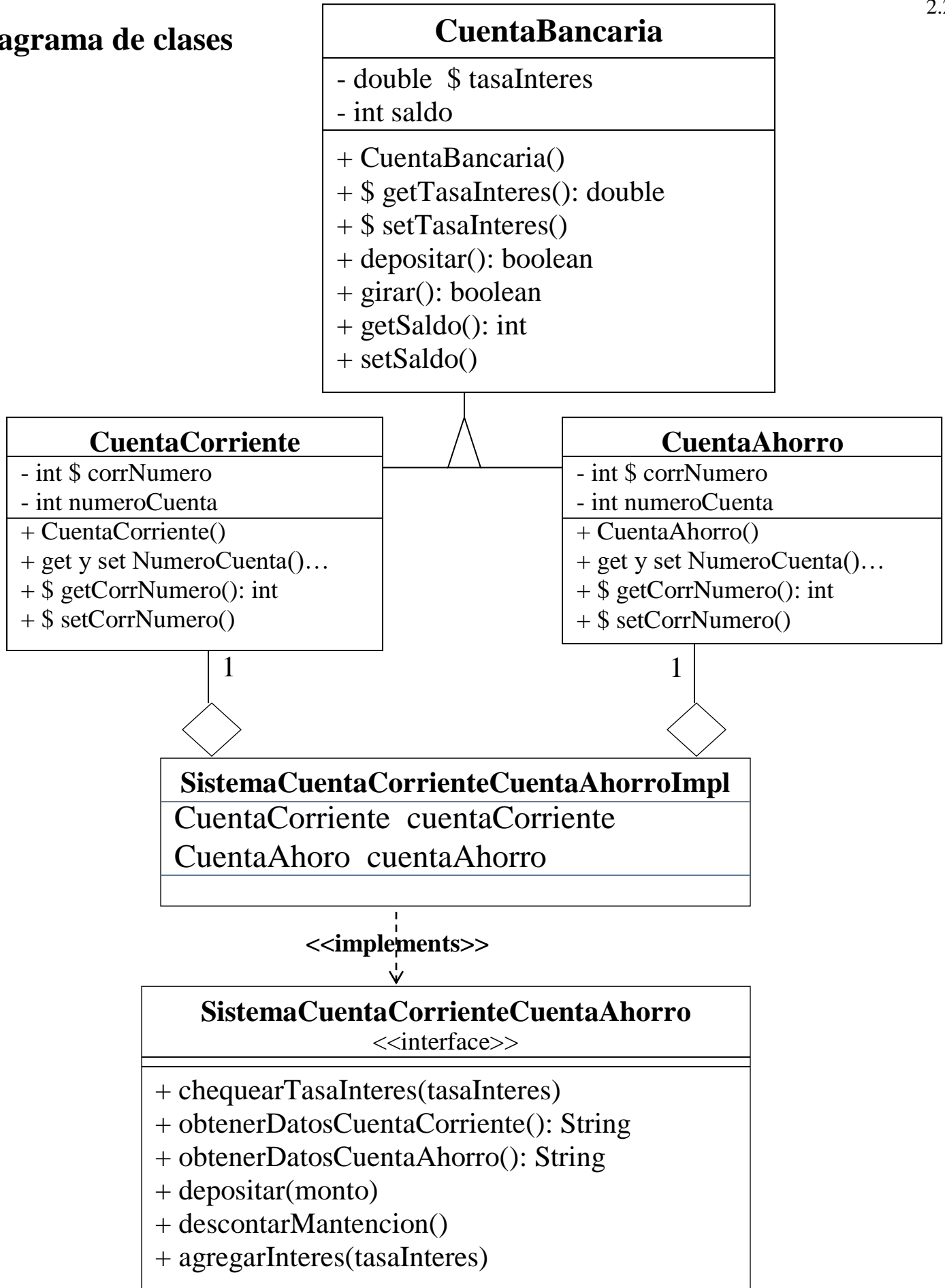


Diagrama de clases



```
package cl.ucn.ei.pa.sistematicuentaahorro.cuentacorriente.dominio;

public class CuentaBancaria {
    private double saldo;
    private static double tasaInteres;

    public CuentaBancaria(double saldo) {
        this.saldo = saldo;
    }

    public double getSaldo() {
        return saldo;
    }

    public void setSaldo(double saldo) {
        this.saldo = saldo;
    }

    public static double getTasaInteres() {
        return tasaInteres;
    }

    public static void setTasaInteres(double tasaInteres) {
        CuentaBancaria.tasaInteres = tasaInteres;
    }

    public boolean depositar(double montoDeposito){
        if (montoDeposito > 0){
            saldo = saldo + montoDeposito;
            return true;
        }
        else{
            return false;
        }
    }
}
```

```

public boolean girar(double montoGiro){
    if (montoGiro >0 && montoGiro < saldo){
        saldo = saldo - montoGiro;
        return true;
    }
    else{
        return false;
    }
}

@Override
public String toString() {
    return "CuentaBancaria [saldo=" + saldo + "]";
}
}

```

```

package cl.ucn.ei.pa.sistemacuentaahorroocuentacorriente.dominio;

public class CuentaAhorro extends CuentaBancaria{
    private int numeroCuenta ;
    private static int correlativoNumeroCuenta = 100001;

    public CuentaAhorro(int saldo) {
        super(saldo);
        numeroCuenta = correlativoNumeroCuenta;
        //numeroCuenta = correlativoNumeroCuenta++;
        correlativoNumeroCuenta++;
    }

    public int getNumeroCuenta() {
        return numeroCuenta;
    }

    public void setNumeroCuenta(int numeroCuenta) {
        this.numeroCuenta = numeroCuenta;
    }
}

```

```

    public static int getCorrelativoNumeroCuenta() {
        return correlativoNumeroCuenta;
    }

    public static void setCorrelativoNumeroCuenta(
        int correlativoNumeroCuenta) {
        CuentaAhorro.correlativoNumeroCuenta=correlativoNumeroCuenta;
    }

    @Override
    public String toString() {
        return "CuentaAhorro [numeroCuenta=" + numeroCuenta +
            ", saldo=" + getSaldo() + "];"
    }
}

```

```

package cl.ucn.ei.pa.sistematicuentaahorro cuentacorriente.dominio;

public class CuentaCorriente extends CuentaBancaria{
    private int numeroCuenta ;
    private static int correlativoNumeroCuenta = 500001;

    public CuentaCorriente(int saldo) {
        super(saldo);
        numeroCuenta = correlativoNumeroCuenta;
        //numeroCuenta = correlativoNumeroCuenta++;
        correlativoNumeroCuenta++;
    }

    public int getNumeroCuenta() {
        return numeroCuenta;
    }

    public void setNumeroCuenta(int numeroCuenta) {
        this.numeroCuenta = numeroCuenta;
    }
}

```

```

public static int getCorrelativoNumeroCuenta() {
    return correlativoNumeroCuenta;
}

public static void setCorrelativoNumeroCuenta(
    int correlativoNumeroCuenta) {
    CuentaCorriente.correlativoNumeroCuenta =
        correlativoNumeroCuenta;
}

@Override
public String toString() {
    return "CuentaCorriente [numeroCuenta=" + numeroCuenta +
        ", saldo= " + getSaldo()+"]" ;
}
}

```

```

package cl.ucn.ei.pa.sistematicuentaahorro cuentacorriente.logica;

public interface SistemaCuentaCorrienteCuentaAhorro {

    public void chequearTasaInteres(double tasaInteres);
    public String obtenerDatosCuentaCorriente();
    public String obtenerDatosCuentaAhorro() ;
    public void depositar(int monto);
    public void descontarMantencion();
    public void agregarInteres();
}

```



```

package cl.ucn.ei.pa.sistemacuentaahorro cuentacorriente.logica;

import cl.ucn.ei.pa.sistemacuentaahorro cuentacorriente.dominio.*;

public class SistemaCuentaCorrienteCuentaAhorroImpl implements
    SistemaCuentaCorrienteCuentaAhorro{

    private CuentaCorriente cuentaCorriente;
    private CuentaAhorro cuentaAhorro;

    public SistemaCuentaCorrienteCuentaAhorroImpl(){
        //Cuando se levanta el sistema, se crean las
        //2 cuentas con un saldo 0
        cuentaAhorro = new CuentaAhorro(0);
        cuentaCorriente = new CuentaCorriente(0);
    }

    public void chequearTasaInteres(double tasaInteres){
        if (tasaInteres >=0.0 && tasaInteres <= 20.0){
            CuentaBancaria.setTasaInteres(tasaInteres);
        }
        else{
            throw new IllegalArgumentException(
                "Tasa interes fuera de rango");
        }
    }

    public String obtenerDatosCuentaCorriente(){
        return cuentaCorriente.toString();
    }

    public String obtenerDatosCuentaAhorro() {
        return cuentaAhorro.toString();
    }
}

```

```
public void depositar(int monto){
    if (cuentaCorriente != null && cuentaAhorro != null ){
        cuentaAhorro.depositar(monto);
        cuentaCorriente.depositar(monto);
    }
    else{
        throw new NullPointerException ("Cuenta no existe");
    }
}

public void descontarMantencion(){
    if (cuentaCorriente != null && cuentaAhorro != null ){
        cuentaCorriente.girar(5);
        if (cuentaAhorro.getSaldo() <= 200){
            cuentaAhorro.girar(5);
        }
    }
    else{
        throw new NullPointerException ("Cuenta no existe");
    }
}

public void agregarInteres(){
    if (cuentaCorriente != null && cuentaAhorro != null ){
        cuentaCorriente.depositar((CuentaBancaria.getTasaInteres()/12) *
                                   cuentaCorriente.getSaldo());

        cuentaAhorro.depositar((CuentaBancaria.getTasaInteres()/12) *
                                cuentaAhorro.getSaldo());
    }
    else{
        throw new NullPointerException ("Cuenta no existe");
    }
}
}
```

```
package cl.ucn.ei.pa.sistemacuentaahorro cuentacorriente.logica;

import ucn.StdIn;
import ucn.Stdout;

public class App {

    public static void main(String[] args) {

        SistemaCuentaCorrienteCuentaAhorro sistema =
            new SistemaCuentaCorrienteCuentaAhorroImpl();

        StdOut.print("Ingrese tasa de interes: ");
        double tasaInteres = StdIn.readDouble();
        try{
            sistema.chquearTasaInteres(tasaInteres);
        }catch(IllegalArgumentException ex){
            StdOut.println(ex.getMessage());
        }

        try{
            StdOut.println("datos cuenta corriente: + " +
                sistema.obtenerDatosCuentaCorriente());
        }catch(NullPointerException ex){
            StdOut.println(ex.getMessage());
        }

        StdOut.println();

        try{
            StdOut.println("datos cuenta ahorro: + " +
                sistema.obtenerDatosCuentaAhorro());
        }catch(NullPointerException ex){
            StdOut.println(ex.getMessage());
        }

        StdOut.println();
    }
}
```

```
//10 depositos de 100 c/u en las cuentas
for(int i = 1; i <= 10; i++){
    try{
        sistema.depositar(100);
        sistema.descontarMantencion();
        sistema.agregarInteres();

        StdOut.println("datos cuenta corriente: + " +
            sistema.obtenerDatosCuentaCorriente());
        StdOut.println();
        StdOut.println("datos cuenta ahorro: + " +
            sistema.obtenerDatosCuentaAhorro());
        StdOut.println();
    }catch(NullPointerException ex1){
        StdOut.println(ex1.getMessage());
    }
}
}
```

Ejemplo 2: Vehículos

Construir un programa que maneje información de vehículos. Un vehículo tiene un número de patente, marca y año de fabricación.

Entre los vehículos es posible distinguir los autos y las camionetas. Para los autos se tiene además, la lectura del cuenta kilómetros y la capacidad del estanque de combustible. Para las camionetas se tiene además, su capacidad de carga.

La siguiente funcionalidad debe estar disponible para el usuario del programa:

- Ingresar información de una cantidad indeterminada de autos y camionetas (no se conoce la secuencia en que viene los datos; sí primero un auto, luego un auto, luego una camioneta, etc.; la secuencia es cualquiera)
- Listado de cada uno de los vehículos con sus datos asociados, en la misma secuencia en que fueron ingresados.
- Dada la patente de un vehículo, se piden los datos de ese vehículo. Además se pide el costo asociado por reparación. Si es camioneta el costo es \$100.000 por unidad de capacidad. Si es auto, es \$20.000 por capacidad del estanque.
- Cantidad total de vehículos, cantidad total de autos, cantidad total de camionetas.
- Datos de la camioneta con la mayor capacidad de carga.

Se pide:

- Modelo del dominio
- Contratos
- Diagrama de clases del dominio de la aplicación
- Diagrama de clases
- Código Java

Modelo del dominio

Auto
<ul style="list-style-type: none"> • Patente • Marca • Año • Lectura cuenta kilómetros • Capacidad estanque combustible

Camioneta
<ul style="list-style-type: none"> • Patente • Marca • Año • Capacidad de carga

Contratos

Operación	Ingresar camioneta (patente, marca, año, capacidad carga)
Descripción	Se ingresa una camioneta
Precondiciones	
Postcondiciones	Camioneta ingresada

Operación	Ingresar auto (patente, marca, año, kilómetros, capacidad estanque)
Descripción	Se ingresa un auto
Precondiciones	
Postcondiciones	Auto ingresado

Operación	Obtener vehículos
Descripción	Se obtienen todos los vehículos
Precondiciones	
Postcondiciones	

Uno de los 2

Operación	Obtener datos de todos los vehículos
Descripción	Se obtienen los datos de todos los vehículos
Precondiciones	
Postcondiciones	

Operación	Obtener datos de un vehículo, incluyendo su costo de reparación (patente)
Descripción	Se obtiene los datos del vehículo
Precondiciones	Que exista el vehículo
Postcondiciones	

Operación	Obtener total de autos
Descripción	Se obtiene la cantidad total de autos
Precondiciones	
Postcondiciones	

Operación	Obtener total de camionetas
Descripción	Se obtiene la cantidad total de camionetas
Precondiciones	
Postcondiciones	

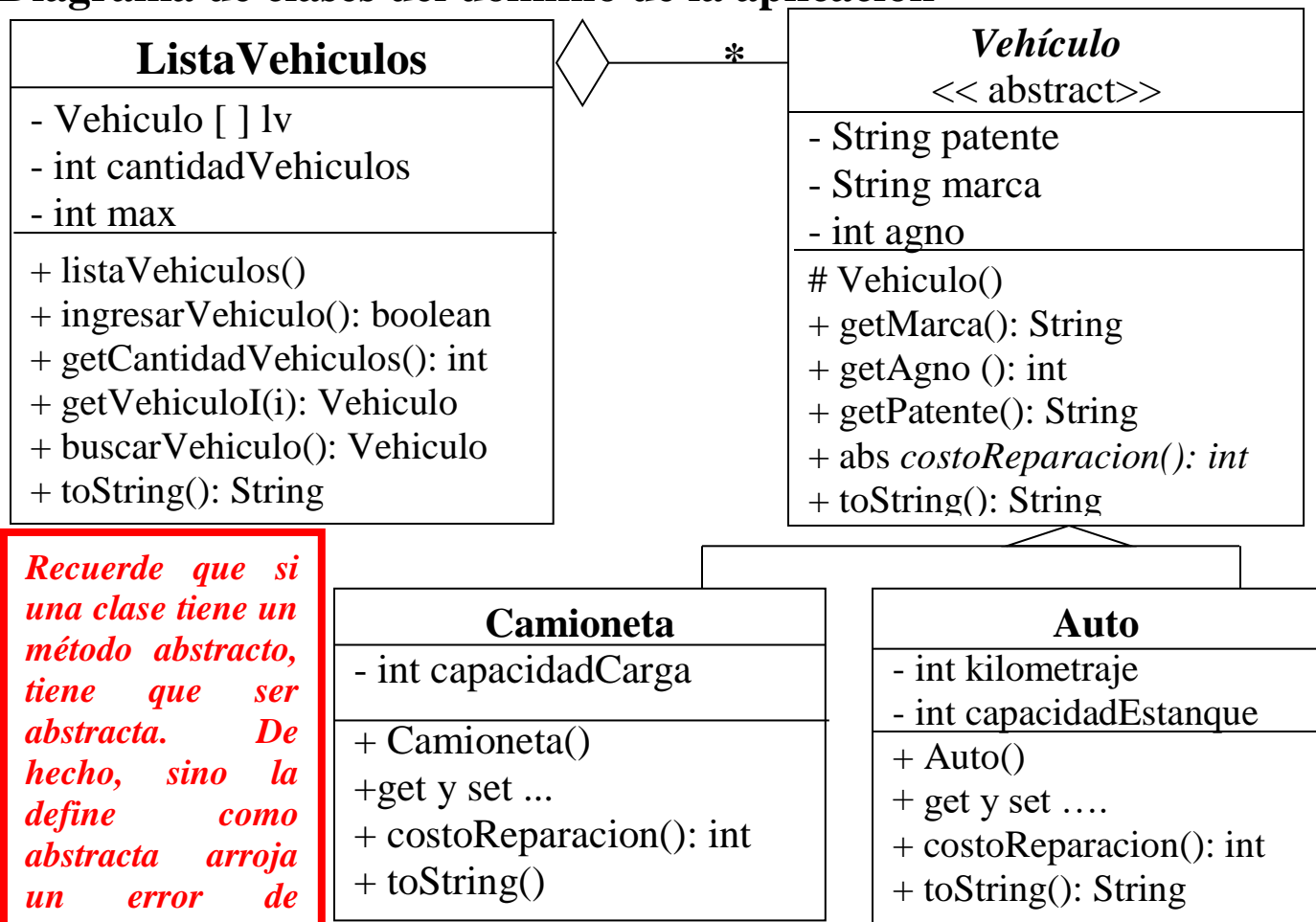
Operación	Obtener total de vehículos
Descripción	Se obtiene la cantidad total de vehículos
Precondiciones	
Postcondiciones	

Operación	Obtener camioneta con mayor capacidad de carga
Descripción	Se obtiene la camioneta con la mayor capacidad de carga
Precondiciones	
Postcondiciones	

Uno de los 2

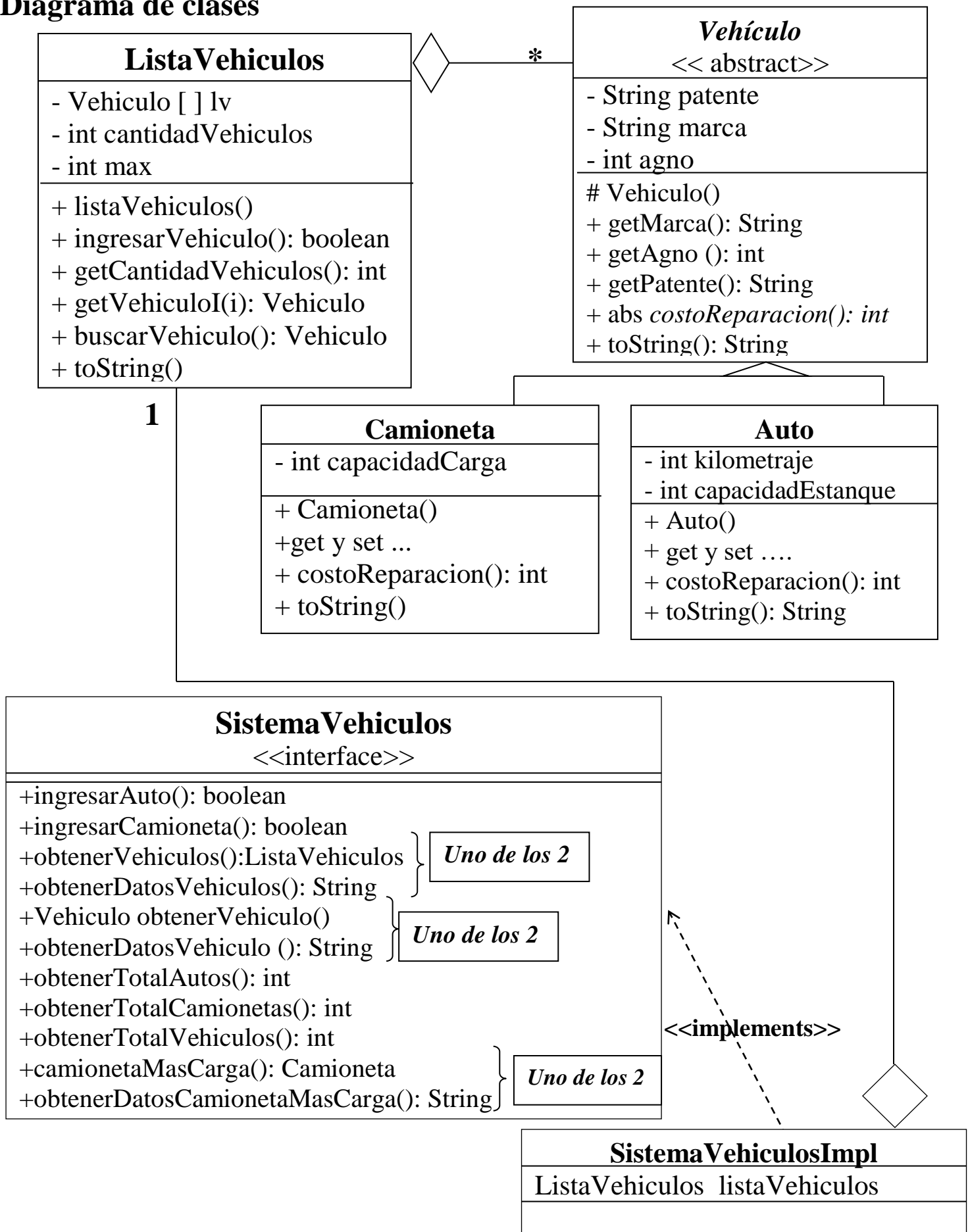
Operación	Obtener datos de la camioneta con mayor capacidad de carga
Descripción	Se obtienen los datos de la camioneta con la mayor capacidad de carga
Precondiciones	
Postcondiciones	

Diagrama de clases del dominio de la aplicación



Recuerde que si una clase tiene un método abstracto, tiene que ser abstracta. De hecho, sino la define como abstracta arroja un error de compilación

Diagrama de clases



```

package cl.ucn.ei.pa.sistemavehiculos.dominio;

public abstract class Vehiculo {
    private String patente;
    private String marca;
    private int agno;

    protected Vehiculo(String patente , String marca,
                        int agno) {

        this.patente = patente;
        this.marca = marca;
        this.agno = agno;
    }

    public String getPatente() {
        return (patente);
    }

    public String getMarca() {
        return marca;
    }

    public int getAgno () {
        return agno;
    }

    abstract public int costoReparacion();
    //Método abstracto

    @Override
    public String toString() {
        return "Vehiculo [" + (patente != null ?
            "patente=" + patente + ", " : "") +
            (marca != null ? "marca=" + marca + ", " : "") +
            "agno=" + agno + "]";
    }
}

```

```

package cl.ucn.ei.pa.sistemavehiculos.dominio;

public class Auto extends Vehiculo{

    private int kilometraje;
    private int capacidadEstanque;

    public Auto(String patente, String marca, int agno,
                int kilometraje, int capacidadEstanque) {
        super(patente , marca , agno);
        this.capacidadEstanque = capacidadEstanque;
        this.kilometraje=kilometraje;
    }

    public int getKilometraje() {
        return kilometraje;
    }

    public int getCapacidadEstanque() {
        return capacidadEstanque;
    }

    public int costoReparacion () {
        return (capacidadEstanque * 20000);
    }

    @Override
    public String toString() {
        return "patente: " + getPatente() + " marca: " +
            getMarca() + " agno: " + getAgno() + " " +
            "Auto [kilometraje=" + kilometraje +
            ", capacidadEstanque=" + capacidadEstanque + " ]";
    }
}

```

```
package cl.ucn.ei.pa.sistemavehiculos.dominio;

public class Camioneta extends Vehiculo{
    private int capacidadCarga;

    public Camioneta (String patente, String marca,
                      int agno, int capacidadCarga) {
        super(patente, marca, agno);
        this.capacidadCarga = capacidadCarga;
    }

    public int getCapacidadCarga() {
        return capacidadCarga;
    }

    public int costoReparacion () {
        return (capacidadCarga * 100000);
    }

    @Override
    public String toString() {
        return "patente: " + getPatente() + " marca: " +
            getMarca() + " agno: " + getAgno() + " Camioneta
            [capacidadCarga=" + capacidadCarga + "];
    }
}
```

```

package cl.ucn.ei.pa.sistemavehiculos.logica;

import cl.ucn.ei.pa.sistemavehiculos.dominio.Auto;
import cl.ucn.ei.pa.sistemavehiculos.dominio.Camioneta;
import cl.ucn.ei.pa.sistemavehiculos.dominio.Vehiculo;

public class ListaVehiculos {
    private Vehiculo[ ] lv;
    private int cantidadVehiculos;
    int max;

    public ListaVehiculos(int max) {
        lv = new Vehiculo [max];
        cantidadVehiculos =0;
        this.max = max;
    }

    public int getCantidadVehiculos() {
        return cantidadVehiculos;
    }

    public Vehiculo getVehiculoI (int i) {
        if (i >= 0 && i < cantidadVehiculos){
            return lv[i];
        }
        return null;
    }

    //Ingresa un vehículo a la lista
    public boolean ingresarVehiculo(Vehiculo v) {
        if (cantidadVehiculos < max){
            lv[cantidadVehiculos]= v;
            cantidadVehiculos++;
            return true;
        }
        else{
            return false;
        }
    }
}

```

```

//Busca una patente de un vehículo en la lista
public Vehiculo buscarVehiculo(String patente) {
    int j=0;
    while (j < cantidadVehiculos &&
           !lv[j].getPatente().equals(patente)) {
        j++;
    }
    if (j == cantidadVehiculos) {
        return null;
    }
    else{
        return lv[j];
    }
}

@Override
public String toString() {
    String salida = "";
    for(int i = 0; i < cantidadVehiculos; i++){
        Vehiculo vehiculo = getVehiculoI(i);

        if (vehiculo instanceof Camioneta) {
            Camioneta camioneta = (Camioneta) vehiculo;
            salida = salida + camioneta.toString();
        }
        else {
            Auto auto = (Auto) vehiculo;
            salida = salida + auto.toString();
        }
        salida = salida + " costo reparacion" +
            vehiculo.costoReparacion() + "\n";
    }
    return salida;
}
}

```

```
package cl.ucn.ei.pa.sistemavehiculos.logica;
import cl.ucn.ei.pa.sistemavehiculos.dominio.*;

public interface SistemaVehiculos {

    public boolean ingresarAuto (String patente,
                                  String marca, int año, int kilometros,
                                  int capacidadEstanque);

    public boolean ingresarCamioneta (String patente,
                                       String marca, int año, int capacidadCarga);

    public String obtenerDatosVehiculos ();

    public String obtenerDatosVehiculo (String patente);

    public int obtenerTotalAutos ();

    public int obtenerTotalCamionetas ();

    public int obtenerTotalVehiculos ();

    public String obtenerDatosCamionetaMasCarga ();

}
```

```
package cl.ucn.ei.pa.sistemavehiculos.logica;

import cl.ucn.ei.pa.sistemavehiculos.dominio.*;

public class SistemaVehiculosImpl implements SistemaVehiculos{

    private ListaVehiculos listaVehiculos;

    public SistemaVehiculosImpl() {
        listaVehiculos = new ListaVehiculos(10);
    }

    public boolean ingresarAuto(String patente, String marca,
        int agno, int kilometros, int capacidadEstanque) {

        Vehiculo auto = new Auto (patente, marca, agno,
            kilometros, capacidadEstanque);
        boolean ingreso =
            listaVehiculos.ingresarVehiculo(auto);
        return ingreso;
    }

    public boolean ingresarCamioneta (String patente,
        String marca, int año, int capacidadCarga) {

        Vehiculo camioneta = new Camioneta(patente, marca,
            año, capacidadCarga);
        boolean ingreso =
            listaVehiculos.ingresarVehiculo(camioneta);

        return ingreso;
    }
}
```



```

public String obtenerDatosVehiculos () {
    if (listaVehiculos!= null) {
        return listaVehiculos.toString();
    }
    else{
        return null;
    }
}

public String obtenerDatosVehiculo (String patente){
    Vehiculo vehiculo =
        listaVehiculos.buscarVehiculo(patente);
    if (vehiculo !=null) {
        if (vehiculo instanceof Auto) {
            Auto auto = (Auto) vehiculo;
            return auto.toString()+"costo reparacion: "
                + auto.costoReparacion();
        }
        else{
            Camioneta camioneta=(Camioneta) vehiculo;
            return camioneta.toString()+
                "costo reparacion: " +
                camioneta.costoReparacion();
        }
    }
    throw new NullPointerException(
        "No existe vehiculo");
}

```

```
public int obtenerTotalAutos () {
    int cantidadAutos = 0;

    for (int i=0;
        i<listaVehiculos.getCantidadVehiculos();i++) {

        Vehiculo vehiculo =
            listaVehiculos.getVehiculoI(i);

        if (vehiculo instanceof Auto) {
            cantidadAutos++;
        }
    }
    return cantidadAutos;
}

public int obtenerTotalCamionetas () {
    int cantidadCamionetas = 0;
    for (int i=0;
        i< listaVehiculos.getCantidadVehiculos();i++) {
        Vehiculo vehiculo =
            listaVehiculos.getVehiculoI(i);
        if (vehiculo instanceof Camioneta) {
            cantidadCamionetas++;
        }
    }
    return cantidadCamionetas;
}

public int obtenerTotalVehiculos () {
    return listaVehiculos.getCantidadVehiculos();
    //return (listaVehiculos.obtenerTotalCamionetas +
    //listaVehiculos.obtenerTotalAutos);
}
```

```

public String obtenerDatosCamionetaMasCarga () {
    int mayor= -1;
    Camioneta camionetaMasCarga= null;
    for(int i = 0;
        i<listaVehiculos.getCantidadVehiculos();i++){

        Vehiculo vehiculo =
            listaVehiculos.getVehiculoI(i);
        if (vehiculo instanceof Camioneta){
            Camioneta camioneta = (Camioneta) vehiculo;
            if (camioneta.getCapacidadCarga()> mayor){
                mayor = camioneta.getCapacidadCarga();
                camionetaMasCarga = camioneta;
            }
        }
    }
    if(camionetaMasCarga != null){
        return camionetaMasCarga.toString();
    }
    else{
        return null;
    }
}
}

```

```

package cl.ucn.ei.pa.sistemavehiculos.logica;

import cl.ucn.ei.pa.sistemavehiculos.dominio.*;
import ucn.StdIn;
import ucn.Stdout;

public class App {

    public static void leerUnVehiculo (SistemaVehiculos
                                         sistema) {

        //Ingresa datos de un vehiculo
        String patente,marca;
        int kilometraje,capacidadEstanque,agno,tipo;

        StdOut.print("Tipo vehiculo Auto[1] Camioneta[2]");
        tipo = StdIn.readInt();
        if (tipo == 1) {
            StdOut.print("Ingrese patente = ");
            patente = StdIn.readString();
            StdOut.print("Ingrese la marca = ");
            marca = StdIn.readString();
            StdOut.print("Ingrese agno fabricacion = ");
            agno = StdIn.readInt();
            StdOut.print("Ingrese el kilometraje=");
            kilometraje = StdIn.readInt();
            StdOut.print("Ingrese capacidad estanque = ");
            capacidadEstanque = StdIn.readInt();

            boolean ingreso =
                sistema.ingresarAuto(patente, marca, agno,
                                     kilometraje, capacidadEstanque);
            if(!ingreso){
                StdOut.println("No se hizo el ingreso del
                               auto. No hay espacio");
            }
        }
    }
}

```

```

    else {
        if (tipo == 2) {
            StdOut.print("Ingrese patente = ");
            patente = StdIn.readString();
            StdOut.print("Ingrese la marca =");
            marca = StdIn.readString();
            StdOut.print("Ingrese agno fabricacion= ");
            agno = StdIn.readInt();
            StdOut.print("Ingrese capacidad carga = ");
            int capacidadCarga = StdIn.readInt();
            boolean ingreso =
                sistema.ingresarCamioneta(patente,
                                           marca, agno, capacidadCarga);
            if(!ingreso){
                StdOut.println("No se hizo el ingreso
                               de la camioneta. No hay espacio");
            }
        }
    }
}

```

```

public static void desplegarMenu () {
    StdOut.println(" ");
    StdOut.println(" M E N U");
    StdOut.println("[1]  Ingresar vehiculo ");
    StdOut.println("[2]  Listado de vehiculos ");
    StdOut.println("[3]  Buscar patente y desplegar
                        costo de reparacion");
    StdOut.println("[4]  Cantidad de vehiculos (autos
                        y camionetas) ");
    StdOut.println("[5]  Datos de la camioneta con la
                        mayor capacidad de carga ");
    StdOut.println("[6]  Salir ");
}

```

```

public static void menu(SistemaVehiculos sistema){
    desplegarMenu();
    StdOut.print("Ingrese opcion:");
    int opcion = StdIn.readInt();
    while(opcion != 6){
        switch(opcion) {

            case 1:
                leerUnVehiculo (sistema);
                break;

            case 2:
                StdOut.println(" ");
                StdOut.println(sistema.
                                obtenerDatosVehiculos());
                break;

            case 3:
                StdOut.print("Patente a buscar: ") ;
                String patente = StdIn.readString();
                try{
                    StdOut.println(" ");
                    StdOut.println(sistema.
                                    obtenerDatosVehiculo(patente));
                } catch (NullPointerException ex) {
                    StdOut.println(ex.getMessage());
                }
                break;
        }
    }
}

```

```

        case 4: int totalVehiculos =
                    sistema.obtenerTotalVehiculos();
                    StdOut.println("Total de vehiculos: " +
                                    totalVehiculos);
                    int totalAutos= sistema.obtenerTotalAutos();
                    StdOut.println("Total autos: " + totalAutos);
                    int totalCamionetas =
                        sistema.obtenerTotalCamionetas();
                    StdOut.println("Total camionetas: " +
                                    totalCamionetas);
        break;

        case 5:
            //Muestra los datos de la camioneta con la
            //mayor capacidad de carga

            String datosCamioneta = sistema.
                obtenerDatosCamionetaMasCarga()
            if (datosCamioneta!= null) {
                StdOut.println(datosCamioneta);
            }
        }
        break;

        } //fin switch
        desplegarMenu();
        StdOut.print("Ingrese opcion:");
        opcion = StdIn.readInt();
    }

}

public static void main(String[] args) {
    SistemaVehiculos sistema =
        new SistemaVehiculosImpl();

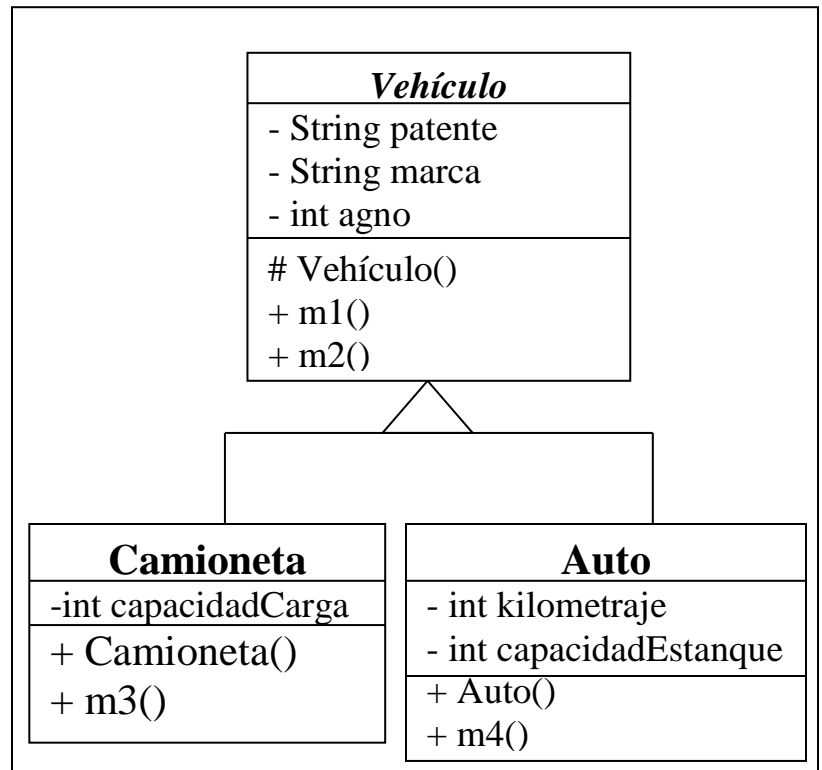
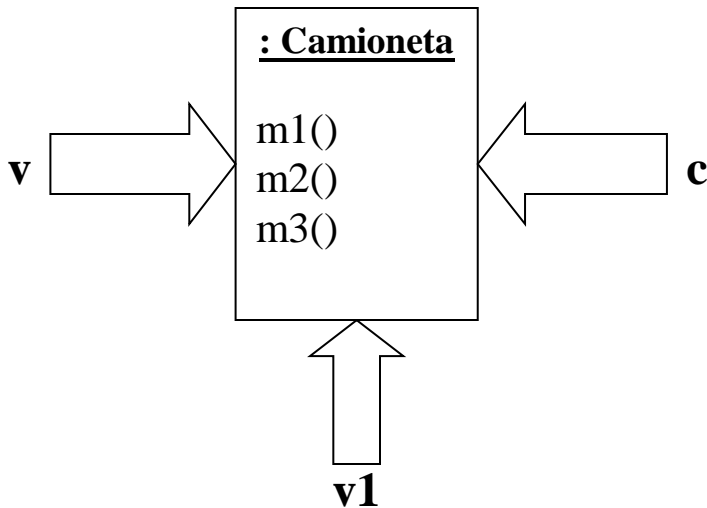
    menu(sistema);
}
}

```

Concepto de Casting

Suponga la siguiente instrucción Java:

```
Vehiculo v = new Camioneta()
```



¿Desde qué punto de vista quiero ver al objeto: como camioneta o como vehículo?

- **Si se ve como vehículo**, sólo puedo ver los métodos m1 y m2.

Se puede achicar la interface del objeto (Vehiculo tiene menos cosas que camioneta)

✓✓ Vehiculo v1 = c

✓✓ Vehiculo v1 = (Vehiculo) c;

- **Si se ve como camioneta**, se deben ver los métodos m1, m2 y m3.

No se puede agrandar la interface del objeto (Camioneta tiene más cosas que Vehiculo)

~~Camioneta c = v~~

Por lo tanto: ✓✓ Camioneta c = (Camioneta) v

Ejemplo 3

Se tiene información de los funcionarios de la empresa de computación Mandiosoft Ltda. en la cual se encuentra la información de cada uno de los 3 tipos de funcionarios del área de desarrollo de sistemas. Estos pueden ser Programadores, Analistas o Ingenieros.

Todos los funcionarios tienen información en común, como lo es rut, nombre, dirección, sueldo base y el tipo de funcionario. Además si es programador interesa el lenguaje de programación que domina (sólo uno por programador), las horas extras y el nivel de programador (4: experto, 3: avanzado, 2: intermedio, 1: rookie). De los analistas además interesa saber los años de experiencia y finalmente de los ingenieros interesa el título y la cantidad de cargas familiares.

La empresa Mandiosoft desarrolla proyectos para otras empresas, en cada uno de estos proyectos participa siempre 1 analista, 1 programador y 1 ingeniero. De cada proyecto interesa el nombre, el código, la duración en meses y su costo total.

Los datos se leen de la siguiente manera:

- Se tiene un archivo con los datos de todos los proyectos.
- Se tiene un archivo con los datos de todos los funcionarios, donde en cada registro viene la información de un analista o de un ingeniero o de un programador. Cada registro debe indicar cuál de ellos es.
- Se tiene un archivo con la asignación de los trabajadores a los proyectos. Por cada registro viene el código del proyecto y el rut del funcionario

Luego de esta lectura de los archivos es necesario realizar la asignación de personal, explicada anteriormente

Requerimientos

Una vez ingresada la información, se necesita:

- Listado de Proyectos con los costos involucrados (por meses y total)
- Listado de funcionarios con sus respectivos sueldos.
- Dado un proyecto entregar el listado de gente involucrada.
- Dado un funcionario entregar el listado de proyectos en que participa.
- Para cada funcionario ingeniero, su nombre y su título.

Los sueldos se calculan de la siguiente forma, para el programador además de su sueldo base, cada hora extra realizada cuesta \$5000 además de un bono extra de \$30000 por nivel y otro bono de 20% del valor total mensual del proyecto por cada proyecto en el que participa. En el caso del analista su sueldo se calcula con el sueldo base más un bono de \$5000 por cada año de experiencia más el 25% del valor total mensual del proyecto por cada proyecto en el que participa. Finalmente el Ingeniero recibe un sueldo base, más \$8000 por cada carga familiar más un 30% del valor total mensual del proyecto por cada proyecto en el que participa.

Suponga que cada funcionario trabaja a lo más en 5 proyectos.

Modelo del dominio

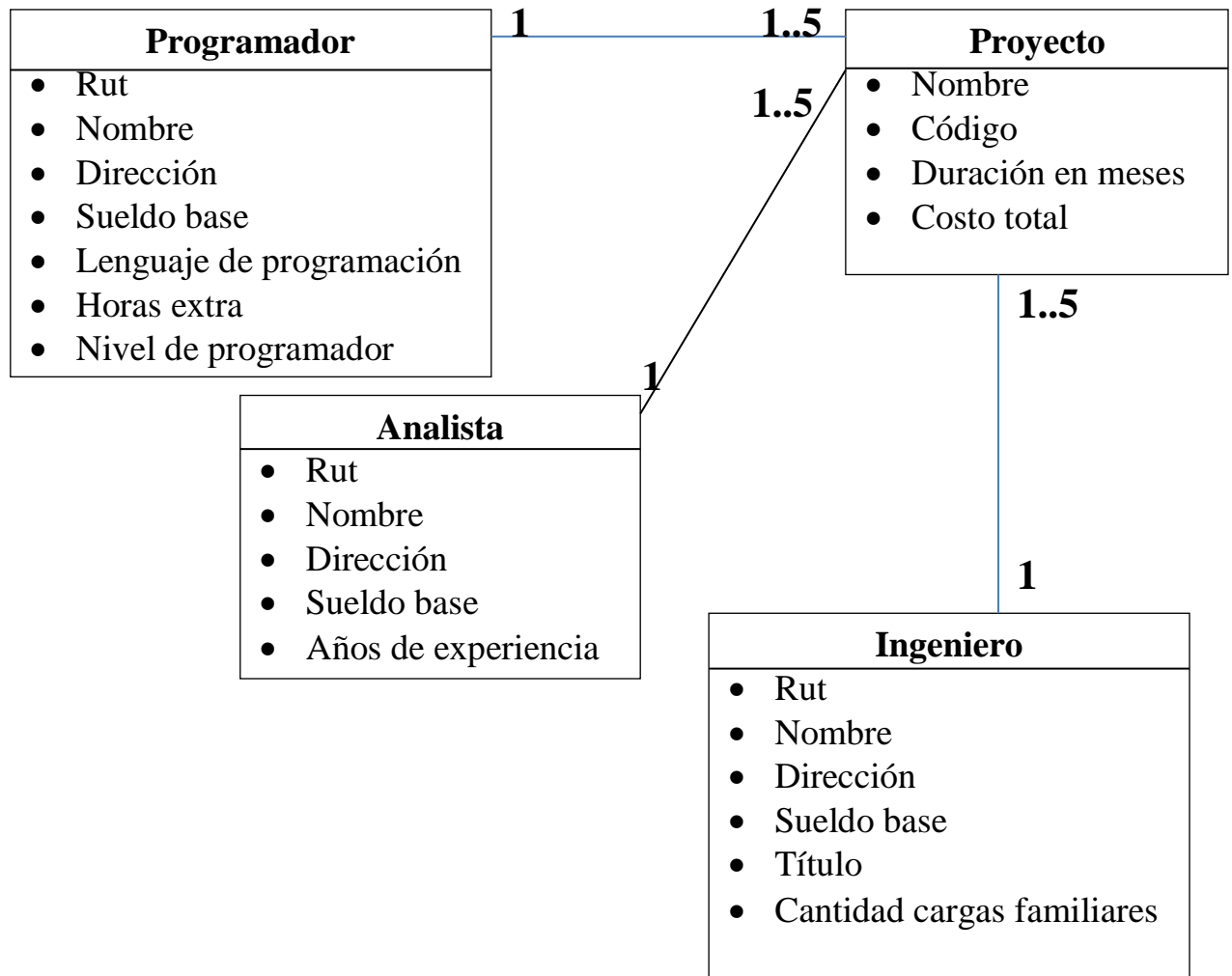
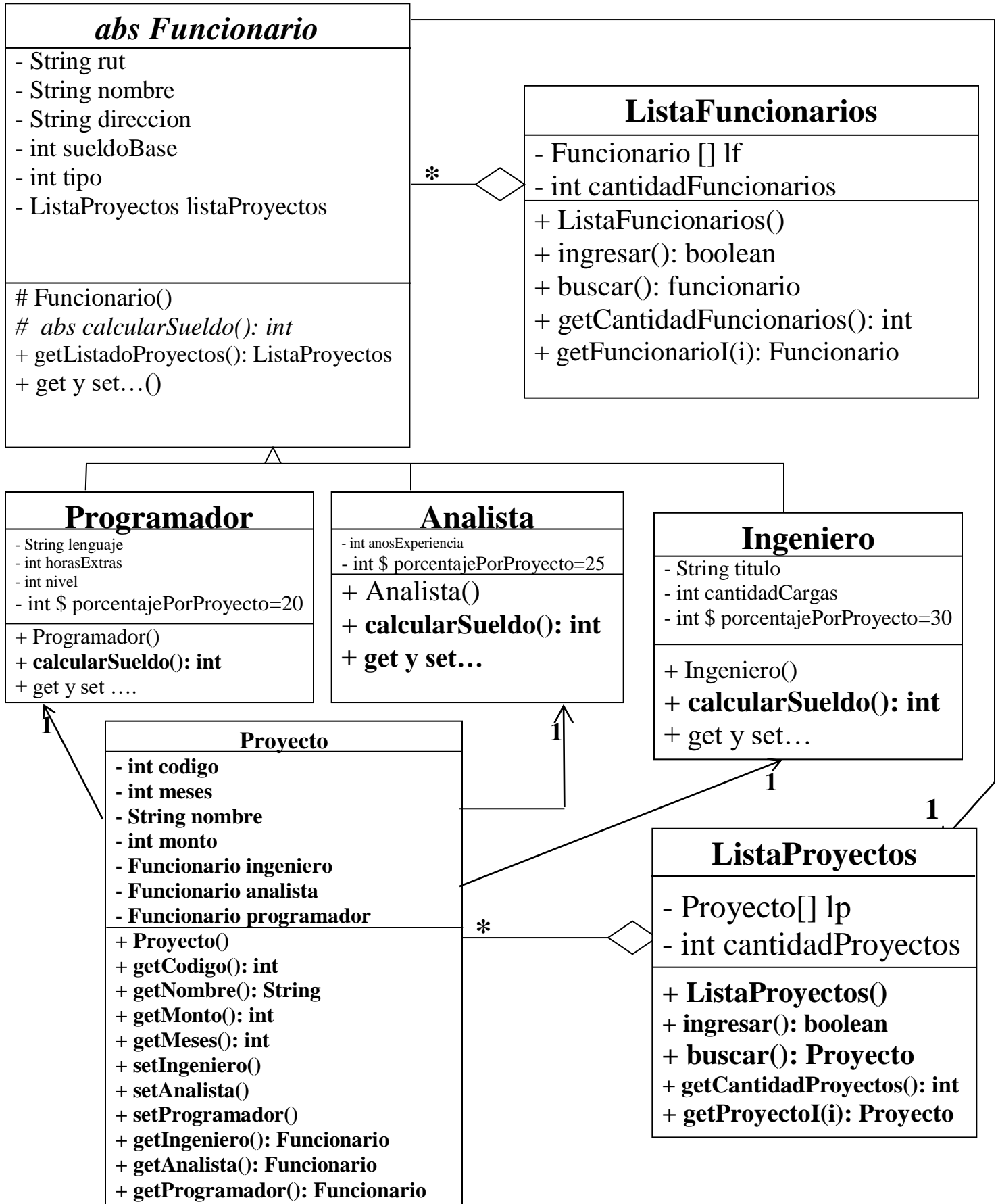


Diagrama de clases del dominio de la aplicación



Contratos

Operación	Ingresar analista (rut, nombre, dirección, sueldo base, años experiencia)
Descripción	Se ingresa un analista
Precondiciones	
Postcondiciones	Analista ingresado

Operación	Ingresar programador (rut, nombre, dirección, sueldo base, lenguaje programacion, horas extra, nivel programador)
Descripción	Se ingresa un programador
Precondiciones	
Postcondiciones	Programador ingresado

Operación	Ingresar ingeniero (rut, nombre, dirección, titulo, cargas)
Descripción	Se ingresa un ingeniero
Precondiciones	
Postcondiciones	Analista ingresado

Operación	Ingresar proyecto (codigo, nombre, monto, meses)
Descripción	Se ingresa un proyecto
Precondiciones	
Postcondiciones	Proyecto ingresado

Operación	AsociarFuncionarioProyecto (codigo proyecto, rut)
Descripción	Se asocia un funcionario al proyecto: <ul style="list-style-type: none"> • El proyecto queda como parte de la lista de proyectos del funcionario • El proyecto queda asociado con el funcionario
Precondiciones	<ul style="list-style-type: none"> • Que exista el proyecto • Que exista el funcionario
Postcondiciones	Proyecto con funcionario asociados

Operación	Obtener datos de los proyectos
Descripción	Se obtienen todos datos de los proyectos, incluido por cada uno de ellos el costo total y mensual
Precondiciones	
Postcondiciones	

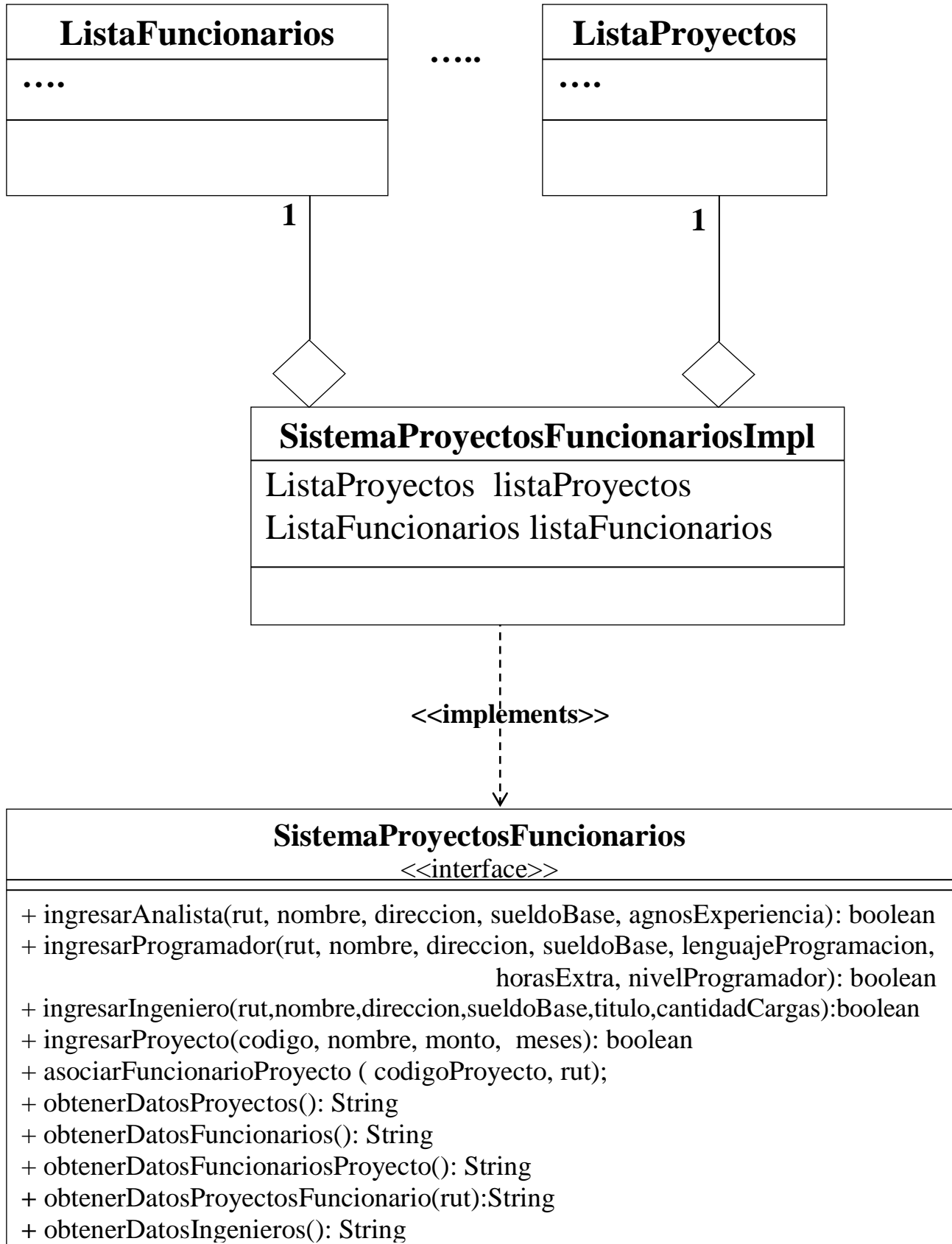
Operación	Obtener datos de los funcionarios
Descripción	Se obtienen todos los datos de los funcionarios, incluido el sueldo
Precondiciones	
Postcondiciones	

Operación	Obtener datos de los funcionarios de un proyecto (código proyecto)
Descripción	Se obtienen los datos de los funcionarios asociados a un proyecto
Precondiciones	Que exista el proyecto
Postcondiciones	

Operación	Obtener datos proyectos de un funcionario (rut)
Descripción	Se obtienen los datos de los proyectos asociados a un funcionario
Precondiciones	Que exista el funcionario
Postcondiciones	

Operación	Obtener datos de los ingenieros
Descripción	Se obtienen los datos de los ingenieros
Precondiciones	
Postcondiciones	

Diagrama de clases




```

package cl.ucn.ei.pa.sistemaFuncionariosProyectos.dominio;

import cl.ucn.ei.pa.sistemaFuncionariosProyectos.logica.ListaProyectos;

public abstract class Funcionario {
    private String rut;
    private String nombre;
    private String direccion;
    private int sueldoBase;
    private int tipo;
    private ListaProyectos listaProyectos;

    protected Funcionario(String rut, String nombre, String direccion,
                           int sueldo, int tipo){

        this.rut = rut;
        this.nombre = nombre;
        this.direccion = direccion;
        this.sueldoBase = sueldo;
        this.tipo = tipo;
        listaProyectos = new ListaProyectos(5);
    }

    public String getRut() {
        return rut;
    }
    public void setRut(String rut) {
        this.rut = rut;
    }
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public String getDireccion() {
        return direccion;
    }
    public void setDireccion(String direccion) {
        this.direccion = direccion;
    }
}

```

No vamos a usar el método toString() para ninguna de las clases

```

public int getSueldoBase() {
    return sueldoBase;
}
public void setSueldoBase(int sueldoBase) {
    this.sueldoBase = sueldoBase;
}
public int getTipo() {
    return tipo;
}
public void setTipo(int tipo) {
    this.tipo = tipo;
}

public ListaProyectos getListaProyectos() {
    return listaProyectos;
}
public void setListaProyectos(ListaProyectos listaProyectos) {
    this.listaProyectos = listaProyectos;
}

abstract public double calcularSueldo() ;
}

```

```

package cl.ucn.ei.pa.sistemaFuncionariosProyectos.dominio;
import cl.ucn.ei.pa.sistemaFuncionariosProyectos.logica.ListaProyectos;
public class Programador extends Funcionario {

    private String lenguaje;
    private int horasExtra;
    private int nivel;

    private static int porcentajePorProyecto=20;

    public Programador (String rut, String nombre, String direccion,
        int sueldo, String lenguaje, int horasExtra, int nivel){
        super (rut,nombre,direccion,sueldo,1);
        this. lenguaje = lenguaje;
        this. horasExtra = horasExtra;
        this. nivel  = nivel;
    }
}

```

```

public String getLenguaje() {
    return lenguaje;
}
public void setLenguaje(String lenguaje) {
    this.lenguaje = lenguaje;
}
public int getHorasExtra() {
    return horasExtra;
}
public void setHorasExtra(int horasExtra) {
    this.horasExtra = horasExtra;
}
public int getNivel() {
    return nivel;
}
public void setNivel(int nivel) {
    this.nivel = nivel;
}
public double calcularSueldo(){
    double suma = 0;
    int monto;
    double bono;
    int meses;
    ListaProyectos listaProyectos = this.getListProyectos();
    for (int i=0; i<listaProyectos.getCantidadProyectos();i++){
        Proyecto proyecto = listaProyectos.getProyectoI(i);
        monto = proyecto.getMonto();
        meses = proyecto.getMeses();
        bono=(monto/meses)*
                (Programador.getPorcentajePorProyecto()/100);
        suma = suma + bono;
    }
    double sueldo = this.getSueldoBase() +
                    5000*horasExtra+30000*nivel+suma;
    return sueldo;
}
public static int getPorcentajePorProyecto() {
    return porcentajePorProyecto;
}
public static void setPorcentajePorProyecto(int porcentajePorProyecto) {
    Programador.porcentajePorProyecto = porcentajePorProyecto;
}
}

```

```

package cl.ucn.ei.pa.sistemaFuncionariosProyectos.dominio;

import cl.ucn.ei.pa.sistemaFuncionariosProyectos.logica.ListaProyectos;

public class Analista extends Funcionario{
    private int anosExperiencia;
    private static int porcentajePorProyecto=25;

    public Analista (String rut, String nombre, String direccion,
                     int sueldo, int anosExperiencia){
        super(rut,nombre,direccion,sueldo,2);
        this. anosExperiencia = anosExperiencia;
    }
    public int getAnosExperiencia() {
        return anosExperiencia;
    }
    public void setAnosExperiencia(int anosExperiencia) {
        this.anosExperiencia = anosExperiencia;
    }
    public double calcularSueldo(){
        double suma = 0;
        int monto;
        double bono;
        int meses;
        ListaProyectos listaProyectos = this.getListaProyectos();
        for (int i=0; i<listaProyectos.getCantidadProyectos();i++){
            Proyecto proyecto = listaProyectos.getProyectoI(i);
            monto = proyecto.getMonto();
            meses = proyecto.getMeses();
            bono=(monto/meses)*
                    (Analista.getPorcentajePorProyecto()/100);
            suma = suma + bono;
        }
        double sueldo = this.getSueldoBase() + 5000 * anosExperiencia
                        + suma;

        return sueldo;
    }
    public static int getPorcentajePorProyecto() {
        return porcentajePorProyecto;
    }
    public static void setPorcentajePorProyecto(int porcentajePorProyecto){
        Analista.porcentajePorProyecto = porcentajePorProyecto;
    }
}

```

```

package cl.ucn.ei.pa.sistemaFuncionariosProyectos.dominio;
import cl.ucn.ei.pa.sistemaFuncionariosProyectos.logica.ListaProyectos;
public class Ingeniero extends Funcionario{
    private String titulo;
    private int cantidadCargas;
    private static int porcentajePorProyecto=30;

    public Ingeniero(String rut,String nombre, String direccion,
                      int sueldo, String titulo, int cantidadCargas){
        super (rut,nombre,direccion,sueldo,3);
        this.titulo = titulo;
        this.cantidadCargas = cantidadCargas;
    }

    public double calcularSueldo(){
        double suma = 0;
        int monto;
        double bono;
        int meses;
        ListaProyectos listaProyectos = this.getListaProyectos();
        for (int i=0; i<listaProyectos.getCantidadProyectos();i++){
            Proyecto proyecto = listaProyectos.getProyectoI(i);
            monto = proyecto.getMonto();
            meses = proyecto.getMeses();
            bono=(monto/meses)*
                (Ingeniero.getPorcentajePorProyecto()/100);
            suma = suma + bono;
        }
        double sueldo = this.getSueldoBase() +
                        8000*this.cantidadCargas + suma;
        return sueldo;
    }

    public String getTitulo() {
        return titulo;
    }
    public int getCantidadCargas() {
        return cantidadCargas;
    }
}

```

```

public void setCantidadCargas(int cantidadCargas) {
    this.cantidadCargas = cantidadCargas;
}
public void setTitulo(String titulo) {
    this.titulo = titulo;
}
public static int getPorcentajePorProyecto() {
    return porcentajePorProyecto;
}
public static void setPorcentajePorProyecto(int porcentajePorProyecto){
    Ingeniero.porcentajePorProyecto = porcentajePorProyecto;
}
}

```

```

package cl.ucn.ei.pa.sistemaFuncionariosProyectos.dominio;

public class Proyecto {
    private String codigo;
    private String nombre;
    private int monto;
    private int meses;
    private Funcionario ingeniero;
    private Funcionario analista;
    private Funcionario programador;

    public Proyecto (String codigo,String nombre,int monto,int meses){
        this.codigo = codigo;
        this.nombre = nombre;
        this.meses = meses;
        this.monto = monto;
        ingeniero =null;
        analista = null;
        programador = null;
    }

    public void setIngeniero (Funcionario ingeniero){
        this.ingeniero = ingeniero;
    }
    public void setAnalista (Funcionario analista){
        this.analista = analista;
    }
}

```

```
public void setProgramador (Funcionario programador){
    this. programador = programador;
}

public String getCodigo() {
    return codigo;
}
public int getMonto() {
    return monto;
}
public int getMeses() {
    return meses;
}
public String getNombre() {
    return nombre;
}
public Funcionario getIngeniero (){
    return ingeniero;
}
public Funcionario getAnalista() {
    return analista;
}
public Funcionario getProgramador() {
    return programador;
}
public void setCodigo(String codigo) {
    this.codigo = codigo;
}
public void setNombre(String nombre) {
    this.nombre = nombre;
}
public void setMonto(int monto) {
    this.monto = monto;
}
public void setMeses(int meses) {
    this.meses = meses;
}
}
```

```
package cl.ucn.ei.pa.sistemaFuncionariosProyectos.logica;
import cl.ucn.ei.pa.sistemaFuncionariosProyectos.dominio.Funcionario;
public class ListaFuncionarios {
    private Funcionario []lf;
    private int cantidadFuncionarios;
    private int max;

    public ListaFuncionarios(int max){
        lf = new Funcionario [max];
        cantidadFuncionarios = 0;
        this.max = max;
    }

    public boolean ingresarFuncionario(Funcionario funcionario){
        if (cantidadFuncionarios < max){
            lf[cantidadFuncionarios]= funcionario;
            cantidadFuncionarios ++;
            return true;
        }
        else{
            return false;
        }
    }

    public int getCantidadFuncionarios(){
        return cantidadFuncionarios;
    }

    public Funcionario getFuncionarioI(int i){
        if (i >=0 && i < cantidadFuncionarios){
            return lf[i];
        }
        else{
            return null;
        }
    }
}
```



```

public Funcionario buscarFuncionario(String rut){
    int i;
    for(i = 0; i < cantidadFuncionarios; i++){
        if (lf[i].getRut().equals(rut)){
            break;
        }
    }
    if (i == cantidadFuncionarios){
        return null;
    }
    else{
        return lf[i];
    }
}
}

```

```

package cl.ucn.ei.pa.sistemaFuncionariosProyectos.logica;

import cl.ucn.ei.pa.sistemaFuncionariosProyectos.dominio.Proyecto;

public class ListaProyectos {
    private Proyecto []lp;
    private int cantidadProyectos;
    private int max;

    public ListaProyectos(int max){
        lp = new Proyecto [max];
        cantidadProyectos = 0;
        this.max = max;
    }

    public boolean ingresarProyecto(Proyecto Proyecto){
        if (cantidadProyectos < max){
            lp[cantidadProyectos]= Proyecto;
            cantidadProyectos ++;
            return true;
        }
        else{
            return false;
        }
    }
}

```

```
public int getCantidadProyectos(){
    return cantidadProyectos;
}

public Proyecto getProyectoI(int i){
    if (i >=0 && i < cantidadProyectos){
        return lp[i];
    }
    else{
        return null;
    }
}

public Proyecto buscarProyecto(String codigo){
    int i;
    for(i = 0; i < cantidadProyectos; i++){
        if (lp[i].getCodigo().equals(codigo)){
            break;
        }
    }
    if (i == cantidadProyectos){
        return null;
    }
    else{
        return lp[i];
    }
}
}
```

```
package cl.ucn.ei.pa.sistemaFuncionariosProyectos.logica;

public interface SistemaFuncionariosProyectos {

    public boolean ingresarAnalista(String rut,String nombre,
        String direccion, int sueldoBase,int agnosExperiencia);

    public boolean ingresarProgramador(String rut,String nombre,
        String direccion, int sueldoBase, String lenguajeProgramacion,
        int horasExtra, int nivelProgramador);

    public boolean ingresarIngeniero(String rut, String nombre,
        String direccion, int sueldoBase, String titulo,
        int cantidadCargas);

    public boolean ingresarProyecto(String codigo,String nombre,
        int monto, int meses);

    public void asociarFuncionarioProyecto (String codigoProyecto,
        String rut);

    public String obtenerDatosProyectos();

    public String obtenerDatosFuncionarios();

    public String obtenerDatosFuncionariosProyecto(String codigo);

    public String obtenerDatosProyectosFuncionario(String rut);

    public String obtenerDatosIngenieros();

}
```

```

package cl.ucn.ei.pa.sistemaFuncionariosProyectos.logica;

import cl.ucn.ei.pa.sistemaFuncionariosProyectos.dominio.*;

public class SistemaFuncionariosProyectosImpl implements
    SistemaFuncionariosProyectos{

    private ListaProyectos listaProyectos;
    private ListaFuncionarios listaFuncionarios;

    public SistemaFuncionariosProyectosImpl() {
        this.listaProyectos = new ListaProyectos(6);
        this.listaFuncionarios=new ListaFuncionarios(30);
    }

    public boolean ingresarAnalista(String rut,String nombre,
        String direccion, int sueldoBase, int agnosExperiencia){

        Funcionario elAnalista = new Analista (rut,nombre,direccion,
            sueldoBase, agnosExperiencia);

        boolean ingreso =
            listaFuncionarios.ingresarFuncionario(elAnalista);
        return ingreso;
    }

    public boolean ingresarProgramador(String rut, String nombre,
        String direccion, int sueldoBase,
        String lenguajeProgramacion, int horasExtra,
        int nivelProgramador){

        Funcionario elProgramador = new Programador (rut, nombre,
            direccion, sueldoBase, lenguajeProgramacion,
            horasExtra, nivelProgramador);

        boolean ingreso =
            listaFuncionarios.ingresarFuncionario(elProgramador);
        return ingreso;
    }
}

```

```

public boolean ingresarIngeniero(String rut,String nombre,
    String direccion, int sueldoBase, String titulo,
    int cantidadCargas){

    Funcionario elIngeniero = new Ingeniero(rut,nombre,direccion,
        sueldoBase, titulo, cantidadCargas);
    boolean ingreso =
        listaFuncionarios.ingresarFuncionario(elIngeniero);
    return ingreso;
}

public boolean ingresarProyecto(String codigo,String nombre,
    int monto, int meses){
    Proyecto proyecto= new Proyecto(codigo, nombre,monto, meses);
    boolean ingreso = listaProyectos.ingresarProyecto(proyecto);
    return ingreso;
}

public void asociarFuncionarioProyecto (String codigoProyecto,
    String rut){
    Funcionario funcionario =
        listaFuncionarios.buscarFuncionario(rut);
    Proyecto proyecto =
        listaProyectos.buscarProyecto(codigoProyecto);
    if (funcionario != null && proyecto != null){
        if (funcionario instanceof Analista){
            proyecto.setAnalista(funcionario);
        }
        else{
            if (funcionario instanceof Programador){
                proyecto.setProgramador(funcionario);
            }
            else{
                proyecto.setIngeniero(funcionario);
            }
        }
    }
    else{
        throw new NullPointerException("No existe el proyecto
            y/o el funcionario");
    }
}
}

```

```

public String obtenerDatosProyectos(){
    String salida = "\nDatos de todos los proyectos\n";
    for(int i = 0; i<listaProyectos.getCantidadProyectos();i++){

        Proyecto proyecto = listaProyectos.getProyectoI(i);
        salida = salida + "codigo: "+proyecto.getCodigo()+
            ", nombre: "+ proyecto.getNombre()+
            ", meses: " + proyecto.getMeses()+
            ", monto mensual: " +
            proyecto.getMonto()/proyecto.getMeses()+
            ", monto total: "+ proyecto.getMonto()+ "\n";
    }
    return salida;
}

public String obtenerDatosFuncionarios(){
    String salida = "\nListado de funcionarios\n";
    for(int i = 0;
        i<listaFuncionarios.getCantidadFuncionarios();i++){

        Funcionario funcionario =
            listaFuncionarios.getFuncionarioI(i);

        salida = salida + "rut: "+funcionario.getRut()+
            ", nombre: "+ funcionario.getNombre()+
            ", direccion: "+funcionario.getDireccion()+
            ", sueldo base: "+funcionario.getSueldoBase()+
            ", sueldo: "+funcionario.calcularSueldo()+ "\n";
    }
    return salida;
}

```

```

public String obtenerDatosProyectosFuncionario (String rut){
    String salida = "\nProyectos del funcionario "+rut+ "\n";
    Funcionario funcionario = listaFuncionarios.buscarFuncionario(rut);
    if (funcionario != null){
        ListaProyectos listaProyectos =
            funcionario.getListaProyectos();
        for(int i=0; i<listaProyectos.getCantidadProyectos();i++){
            Proyecto proyecto = listaProyectos.getProyectoI(i);
            salida = salida + proyecto.getNombre() + "\n";
        }
        return salida;
    }
    else {
        throw new NullPointerException(
            "No existe el funcionario");
    }
}

public String obtenerDatosIngenieros(){
    String salida = "\nDatos de los ingenieros\n";
    for(int i=0; i<listaFuncionarios.getCantidadFuncionarios();i++){
        Funcionario funcionario =
            listaFuncionarios.getFuncionarioI(i);
        if (funcionario instanceof Ingeniero){
            salida = salida + "nombre "+ funcionario.getNombre();
            Ingeniero ingeniero = (Ingeniero) funcionario;
            salida=salida+", titulo "+ingeniero.getTitulo()+"\n";
        }
    }
    return salida;
}

```

```
public String obtenerDatosFuncionariosProyecto(String codigo){
    String salida = "\nFuncionarios del proyecto "+codigo+ "\n";
    Proyecto proyecto = listaProyectos.buscarProyecto(codigo);
    if (proyecto == null) {
        throw new NullPointerException("No existe el proyecto");
    }
    else {
        if (proyecto.getIngeniero() != null) {
            salida = salida + "ingeniero " +
                proyecto.getIngeniero().getNombre()+ "\n";
        }
        if(proyecto.getAnalista() != null) {
            salida = salida + "analista " +
                proyecto.getAnalista().getNombre()+ "\n";
        }
        if (proyecto.getProgramador() != null) {
            salida = salida + "programador " +
                proyecto.getProgramador().getNombre()+ "\n";
        }
    }
    return salida;
}
```



```

package cl.ucn.ei.pa.sistemaFuncionariosProyectos.logica;
import java.io.IOException;

import ucn.ArchivoEntrada;
import ucn.Registro;
import ucn.StdIn;
import ucn.StdOut;

public class App {

    public static void main(String[] args) throws IOException{
        SistemaFuncionariosProyectos sistema =
            new SistemaFuncionariosProyectosImpl();

        LeerProyectos(sistema);

        LeerFuncionarios(sistema);

        LeerPersonalProyectos(sistema);

        StdOut.println(sistema.obtenerDatosProyectos());

        StdOut.println(sistema.obtenerDatosFuncionarios());

        StdOut.print("Rut funcionario para ver sus proyectos: ");
        String rut = StdIn.readString();
        try{
            StdOut.println(sistema.obtenerDatosProyectosFuncionario(rut));
        }catch(NullPointerException ex){
            StdOut.println(ex.getMessage());
        }

        StdOut.print("Codigo proyecto para ver sus funcionarios: ");
        String codigo = StdIn.readString();
        try{
            StdOut.println(sistema.obtenerDatosFuncionariosProyecto(codigo));
        }catch(NullPointerException ex){
            StdOut.println(ex.getMessage());
        }

        StdOut.println(sistema.obtenerDatosIngenieros());
    }
}

```

```

public static void leerFuncionarios(
    SistemaFuncionariosProyectos sistema) throws IOException{
    ArchivoEntrada archivo =
        new ArchivoEntrada("Funcionarios.txt");
    boolean ingreso = true;
    while(!archivo.isEndFile() && ingreso) {
        Registro registro = archivo.getRegistro();
        int tipo = registro.getInt();
        if (tipo == 1) {//Programador
            String rut = registro.getString();
            String nombre = registro.getString();
            String direccion = registro.getString();
            int sueldoBase=registro.getInt();
            String lenguaje = registro.getString();
            int horasExtra=registro.getInt();
            int nivel=registro.getInt();
            ingreso = sistema.ingresarProgramador(rut, nombre,
                direccion,sueldoBase,lenguaje,horasExtra,nivel);
            if(!ingreso) {
                StdOut.println("No hay espacio para mas");
            }
        }
        else {
            if(tipo == 2) {//Analista
                String rut = registro.getString();
                String nombre = registro.getString();
                String direccion = registro.getString();
                int sueldoBase=registro.getInt();
                int agnosExperiencia=registro.getInt();
                ingreso = sistema.ingresarAnalista(rut, nombre,
                    direccion,sueldoBase,agnosExperiencia);
                if(!ingreso) {
                    StdOut.println("No hay espacio para mas ");
                }
            }
        }
    }
}

```

```

        else {//Ingeniero
            String rut = registro.getString();
            String nombre = registro.getString();
            String direccion = registro.getString();
            int sueldoBase=registro.getInt();
            String titulo = registro.getString();
            int cantidadCargas=registro.getInt();
            ingreso = sistema.ingresarIngeniero(rut, nombre,
                direccion,sueldoBase,titulo,cantidadCargas);

            if(!ingreso) {
                StdOut.println("No hay espacio para mas");
            }
        }
    }
    archivo.close();
    StdOut.println("Terminado de leer el archivo Funcionarios");
}

public static void leerProyectos(
    SistemaFuncionariosProyectos sistema) throws IOException{
    ArchivoEntrada archivo = new ArchivoEntrada("Proyectos.txt");
    boolean ingreso = true;
    while(!archivo.isEndFile() && ingreso) {
        Registro registro = archivo.getRegistro();
        String nombre= registro.getString();
        String codigo = registro.getString();
        int duracion = registro.getInt();
        int costo = registro.getInt();
        ingreso=sistema.ingresarProyecto(codigo,nombre, costo,duracion);
        if(!ingreso) {
            StdOut.println("No hay espacio para mas");
        }
    }
    archivo.close();
    StdOut.println("Terminado de leer el archivo Proyectos");
}

```

```
public static void leerPersonalProyectos(  
    SistemaFuncionariosProyectos sistema) throws IOException{  
    ArchivoEntrada archivo =  
        new ArchivoEntrada("FuncionariosProyectos.txt");  
    while(!archivo.isEndFile()) {  
        Registro registro = archivo.getRegistro();  
        String codProyecto= registro.getString();  
        String rut= registro.getString();  
        sistema.asociarFuncionarioProyecto(codProyecto, rut);  
    }  
    archivo.close();  
    StdOut.println("Terminado de leer FuncionariosProyectos");  
}  
}
```

2.9. Visibilidad de clases y control de acceso para los miembros de una clase

A) Visibilidad de clases

Al declarar una clase se puede especificar que es pública usando el atributo `public`. De este modo la clase podrá ser usada por cualquier otra clase.

Si la clase no es pública, entonces la clase sólo puede ser usada dentro del paquete que la contiene.

`{public} {final / abstract} class...`

Sólo se puede especificar uno de los atributos puestos en la misma llave.

B) Control de acceso para los miembros de una clase

Java soporta cuatro niveles de acceso para las variables y los métodos de una clase:

- `private`
- `protected`
- `public`
- `package` (sin modificador)

Control de acceso para los miembros de una clase

	Clase	Subclase	Package	Mundo
Modificador:				
private	X			
protected	X	X*	X	
public	X	X	X	X
package	X		X	

PRIVADO:

Es el nivel de acceso más restrictivo. Un miembro privado es accesible **sólo** en la clase en la que se encuentra definido.

Ejemplo: private

```
public class Alpha {
    private int soyPrivado;

    private void metodoPrivado() {
        StdOut.println("Método Privado");
    }
}
```

```
public class Beta{
    void metodoAcceso() {
        Alpha a = new Alpha();
        a.soyPrivado = 10; //ilegal
        a.metodoPrivado(); //ilegal
    }
}
```

La clase Alpha contiene un método que compara el objeto actual (this) con otro objeto de la misma clase:

```
public class Alpha {
    private int soyPrivado;
    boolean esIgualA(Alpha otraAlpha) {
        if(this.soyPrivado == otraAlpha.soyPrivado){
            return true;
        }
        else{
            return false;
        }
    }
}
```

PROTECTED:

Permite que la clase, que las subclases, y que todas las clases en el mismo paquete tengan acceso al miembro declarado como protegido.

Ejemplo: protected

```
package Greek;
public class Alpha {
    protected int soyProtegido;

    protected void metodoProtegido() {
        StdOut.println("Metodo protegido");
    }
}
```

```
package Greek;
public class Gamma {
    void metodoAcceso() {
        Alpha a = new Alpha();
        a.soyProtegido = 10; //legal
        a.metodoProtegido(); //legal
    }
}
```

En el caso de una subclase A que reside en un paquete diferente, esta subclase A puede referenciar los miembros protegidos de la clase B sólo en el caso de aquellos objetos de la subclase A o de subclases de A.

```
import Greek.*;
package Latin;

public class Delta extends Alpha {
    void metodoAcceso(Alpha a, Delta d) {
        a.soyProtegido = 10; //illegal
        d.soyProtegido = 10; //legal
        a.metodoProtegido(); //illegal
        d.metodoProtegido(); //legal
    }
}
```

PUBLIC:

Cualquier clase en cualquier paquete, tiene acceso a los miembros públicos de la clase.

Ejemplo: public

```
package Greek;

public class Alpha {
    public int soyPublico;

    public void metodoPublico() {
        StdOut.println("Método Público");
    }
}
```



```

import Greek.*;
package Roman;
public class Beta {
    void metodoAcceso() {
        Alpha a = new Alpha();
        a.soyPublico = 10; //legal
        a.metodoPublico(); //legal
    }
}

```

PACKAGE:

Si un miembro de una clase no tiene modificador, entonces será visible a todas las clases del mismo paquete. Esto es lo establecido por defecto.

Ejemplo: package

```

package Greek;
public class Alpha {
    int soyUnPaquete;

    void metodoPaquete() {
        StdOut.println("Método Paquete");
    }
}

```

```

package Greek
public class Beta {
    void metodoAcceso() {
        Alpha a = new Alpha();
        a.soyUnPaquete = 10; //legal
        a. metodoPaquete(); //legal
    }
}

```

C) Resumen de los atributos que pueden tener variables y métodos

VARIABLE:

{private / public / protected}

{final}

{static}

MÉTODO:

{private / public / protected}

{final / abstract}

{static}

Sólo se puede especificar uno de los atributos puestos en la misma llave.

Ejercicio

```

package P1;
import P2.C4;
public class C1{
    private int a1;
    protected int a2;
    private void imprimir(){
        StdOut.print("Valor: " + this.a1);
    }
    public int getA1(){
        return this.a1;
    }
    protected void setA1(int v){
        this.a1 = v;
    }
    public void aumentarC4(C4 a){
        C5 c = a.getC5();
        this.a1 = c.getA1();
        a.setA1(50*this.a1);
    }
}

```

La clase C4, aunque es pública, no es accesible desde C1, ya que está en otro paquete. Para usarla debería tener:

- import P2.C4;
- ó
- P2.C4 a

La clase C5, aunque es pública, no es accesible desde C1, ya que está en otro paquete. Para usarla debería tener:

- import P2.C5;
- ó
- P2.C5 c = a.getC5();

```

package P1;
public class C2{
    protected int a1;
    public final int a2 = 1000;
    protected void imprimir(){
        StdOut.print("Valor: " + this.a1);
    }
    public int metodo1(C2 a, C4 b){
        return a.getA1() * b.a2;
    }
}

```

La clase C4, aunque es pública, no es accesible desde C2, ya que está en otro paquete.

Para usarla debería tener:

- import P2.C4;
- ó
- P2.C4 b

```

    public int getA1(){
        return this.a1;
    }
}

```

package P1;

public final class C3{

```

    private int a1;
    private int a2;
    public int a3;

```

```

    private int calcular(C1 a, C2 b, C3 c){
        a.a2 = a.a2*2;
        b.a1 = b.a1+15;
        c.a1 = a.a1*b.a1;
        return c.a1;
    }

```

El atributo a1 de la clase C1 es privado, por lo tanto no lo puedo acceder.

```

    public int metodo1(C2 a){
        int base = this.a1 * 100;
        int multiplicador = a.getA1();
        int total = base * multiplicador;
        if (total > 5000) {
            a.a2 = total;
        }
        else {
            this.a2 = total;
        }
        return total;
    }
}

```

El atributo a2 de la clase C2 es final, por lo tanto no puede modificarse.

```
package P2;
import P1.*;
```

```
public class C4 extends C1{
    private int a1;
    public int a2;
    private C5 a3;

    private void metodo1(C1 a, C4 b){
        int temp = b.a2*5;
        b.a1 = a.a2 * temp;
    }

    public C5 getC5(){
        return a3;
    }

    public void setA1(int v){
        this.a1 = v;
    }
}
```

El atributo a2 de la clase C1 no es visible, ya que es protected y para que se pueda usar debiera estar en el mismo paquete de la clase C4, lo que no es así. De hecho la clase C4 está en el paquete P2 y la clase C1 en el P1.

Aunque se importó el paquete P1, se debe considerar el hecho de que cuando residen en paquetes distintos, la subclase puede referenciar los miembros protected solo para los objetos de la subclase o de sus subclases. De hecho, **a** no es de la subclase, es de la super clase.

```
package P2;
public class C5 extends C3{
    private int a1;

    private void imprimir(){
        StdOut.print("Valor: " + this.a1);
    }

    public int get_a1(){
        return this.a1;
    }
}
```

La clase C3 es final, por lo tanto no se pueden definir subclases desde C3.

Ejercicio

El siguiente código, ¿Tiene algún error?, ¿Cuál es?

```
public class Punto{
    protected int x;
    protected int y;

    Punto(int x, int y){
        this.x=x;
        this.y=y;
    }
    public int getX() {
        return x;
    }
}

public class Punto3d extends Punto{
    private int z;
    Punto3d(int x,int y){
        super(x,y);
    }
    public void setZ(int z){
        this.z=z;
    }
}

public class Principal{
    public static void main (String[] args){
        Punto miPunto=new Punto3d(30,50);
        miPunto.setZ(40);
    }
}
```

Ejercicios Propuestos

A) Ejercicio: Programa para una empresa de capacitación.

“Soy el Director de una empresa de capacitación que imparte cursos en el área de tecnología de información. Dictamos muchos cursos, cada uno de los cuales tiene un código, un nombre y una tarifa.

Programación en UNIX y Programación SQL, son dos de nuestros cursos más populares. Los cursos pueden durar desde un día a cuatro días. Un instructor puede enseñar varios cursos. Pablo Rogers y María González, son dos de nuestros principales instructores. Para cada instructor se necesita conocer, su nombre y su número telefónico. Cada curso es impartido por un único instructor. Primero se crea un curso, y luego se le asigna el profesor. Los estudiantes pueden tomar varios cursos a la vez; para cada estudiante se necesita conocer su nombre y su número telefónico. Algunos de nuestros estudiantes y profesores en algunas ocasiones no proporcionan sus números telefónicos.”

Para el enunciado anterior, se pide el diagrama de clases.

Las Operaciones que los objetos realizan, resultan de:

- Consultas.
- Actualizaciones.
- Procesos sobre los datos.

Se deben identificar las entidades externas que interactúan con el sistema, y las diferentes maneras en que usan el sistema.

- Agregar, eliminar y modificar alumno.
- Agregar, eliminar y modificar curso.
- Agregar, eliminar y modificar profesor.
- Agregar, eliminar estudiante de un curso.
- Asignar, modificar un profesor a un curso.
- Encontrar todos los alumnos de un curso.
- Encontrar todos los cursos que dicta un profesor.

B) Ejercicio: Declaración del problema

El programa debe manejar información relacionada con los empleados de la empresa, y sus datos asociados. Para cada empleado se manejan datos de:

- Rut
- Nombre
- Dirección
- Fecha de contratación
- Teléfono particular

Los empleados están asignados a un único departamento. La empresa cuenta con varios departamentos, para los que se tienen datos de:

- Código
- Nombre
- Localización

Además, la empresa cuenta con un conjunto de vehículos, los que pueden ser asignados a los empleados (máximo un vehículo por empleado). Para cada vehículo se registra:

- Código
- Fecha de su última mantención
- Fecha de expiración de la revisión técnica
- Lectura del cuenta kilómetros
- Capacidad del estanque de combustible

Algunos empleados trabajan con contrato por horas. A ellos se les contrata por una cierta cantidad de horas mensuales y se acuerda con cada uno el precio a cancelar por hora. En el momento de pagarles se verifica la cantidad total de horas trabajadas en el mes y si se registran horas de sobre tiempo, se les paga 1,5 veces su precio normal por cada hora extra.

Algunos empleados trabajan con contrato a plazo indefinido, fijándoseles un sueldo base a pagar por mes. Estos empleados tienen fijada además, una

bonificación que asciende al 25% del sueldo base y un bono de antigüedad, que varía para cada trabajador.

Algunos empleados pertenecen a uno de los varios sindicatos que funcionan al interior de la empresa. Cada sindicato tiene un código, un nombre, un teléfono, una dirección y fija un monto de cuota mensual a pagar por cada uno de sus afiliados, para ser descontada de su remuneración. Sólo los empleados con contrato a plazo indefinido, pueden pertenecer a un sindicato.

Se necesita:

1. Ingresar datos al programa.

2. Peticiones que el sistema debe responder:

- Dado un nombre de departamento, encontrar los nombres de sus empleados.
- Dado un nombre de empleado, encontrar el nombre de su departamento.
- Dado un nombre de empleado, encontrar (si existe), el vehículo asignado.
- Dado un código de vehículo, encontrar el nombre del empleado que lo usa.
- Calcular el sueldo neto mensual de un empleado horas dado.
- Calcular el sueldo neto mensual de un empleado plazo indefinido dado.
- Calcular el sueldo neto mensual de un empleado dado.
- Dado un código un código de sindicato, encontrar los nombres de sus afiliados.
- Para cada sindicato, indicar la cantidad de sus afiliados.
- Total de empleados
- Total de empleados horas.
- Total de empleados plazo indefinido.
- Total de vehículos.
- Total de vehículos asignados a empleados horas.

- Total de vehículos asignados a empleados plazo indefinido.
- Total de vehículos asignados a los empleados hora.
- Sueldo neto mensual mayor y menor de los empleados.
- Sueldo neto mensual mayor y menor de los empleados hora.
- Sueldo neto mensual mayor y menor de los empleados plazo indefinido.
- Total cancelado en remuneraciones brutas mensuales.