Universidad Nebrija

Escuela Politécnica Superior de Ingeniería
Departamento de Ingeniería Industrial

Fundamentos de la informática

4. Programación orientada a objetos con Java

Contenido

- Introducción
- La programación orientada a objetos
- Programación Java
- Introspección
- Utilidades de Java

Introducción

Análisis y diseño orientado a objetos

- La programación orientada a objetos es una técnica de análisis y diseño que se enfoca en los elementos de un sistema, sus atributos y responsabilidades
- El modelo abstracto está formado de clases. Una clase describe a un conjunto de objetos que comparte los mismos atributos, comportamiento y semántica
- Un objeto es una instancia de una clase

Programación orientada a objetos

Análisis y diseño orientado a objetos

- Las clases representan un esquema simplificado de la casuística de un problema determinado
- Para diseñar un sistema orientado a objetos es necesario responder las siguientes preguntas
  - ¿Cuáles son los elementos tangibles de un sistema?
  - ¿Cuáles son sus atributos?
  - ¿Cuáles son sus responsabilidades?
  - ¿Cómo se relacionan los elementos del sistema?
  - ¿Qué objeto debe “saber”...?
  - ¿Qué objeto debe “hacer”...?

Programación orientada a objetos

Conceptos básicos

- Encapsulación
- Herencia
- Polimorfismo

Programación orientada a objetos

Encapsulación

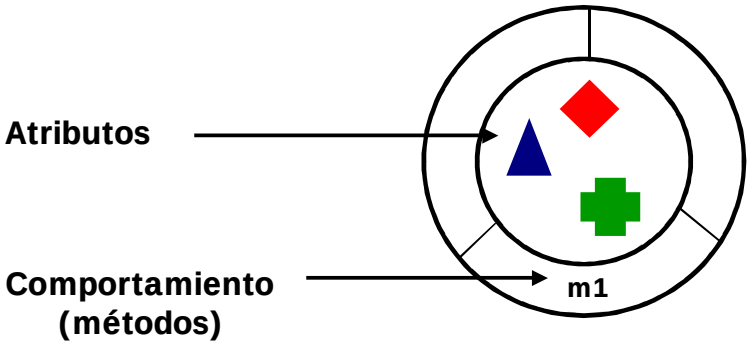
- La encapsulación consiste en formar un “paquete” con los atributos (variables) y el comportamiento (métodos) de un objeto

- Los métodos forman la membrana exterior de un objeto y “esconden” los detalles de implementación al usuario

Programación orientada a objetos

Encapsulación

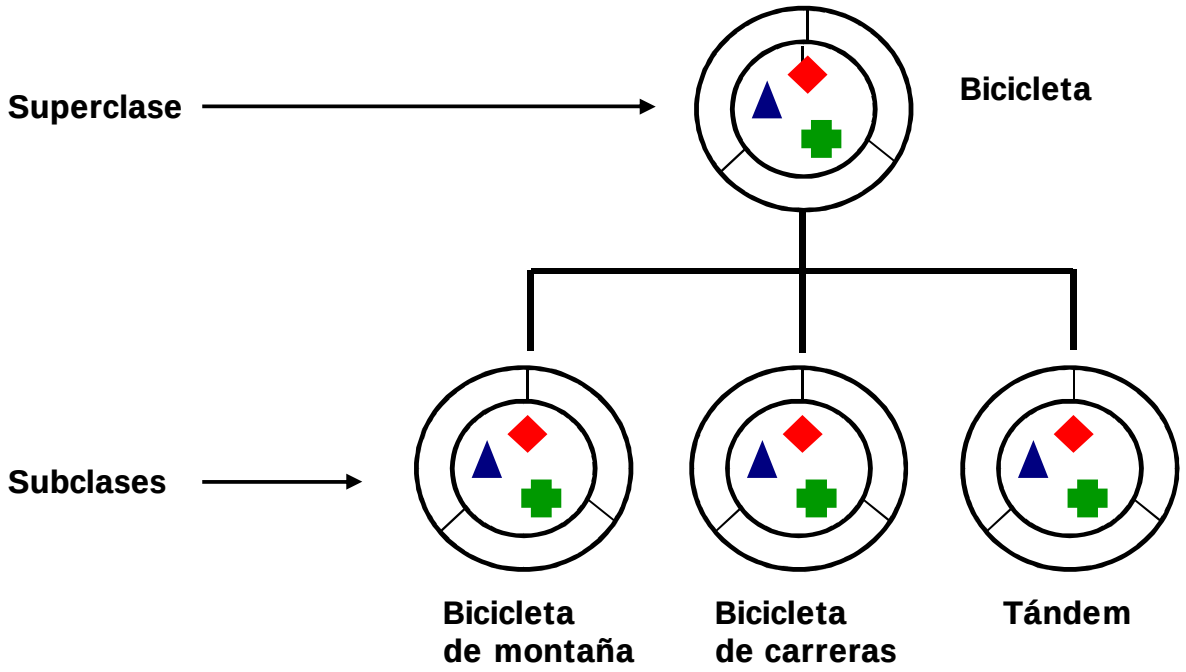
- La encapsulación hace que un sistema sea más fácil de comprender y facilita el mantenimiento de una aplicación



Programación orientada a objetos

Herencia

- La herencia es la capacidad de una clase para definirse en términos de otra clase y "heredar" atributos y responsabilidades de la clase de orden superior



Programación orientada a objetos

Polimorfismo

- El polimorfismo permite que distintos objetos pertenecientes a una misma clase “respondan” de diferentes formas a un mismo mensaje

- El polimorfismo permite modificar el comportamiento de un método en cada subclase. En este ejemplo, la superclase mascota tiene las subclases gato, pato y perro. El método “saludar” cada subclase es diferente

En este ejemplo cada tipo de mascota “saluda” de forma distinta

Programación orientada a objetos

Encapsulación, herencia y polimorfismo

- La reutilización de código es una de las grandes ventajas de la programación orientada a objetos
- Reduce el tiempo de desarrollo de aplicaciones e incrementa la productividad de los ingenieros de software
  - Reutilización de clases
  - Diseño de una nueva clase a partir de otra (herencia)

Programación orientada a objetos

Diagramas de clases

- En UML una clase se representa como un rectángulo dividido en tres partes: el nombre de la clase, sus atributos y métodos. Las clases abstractas se identifican por la letra cursiva

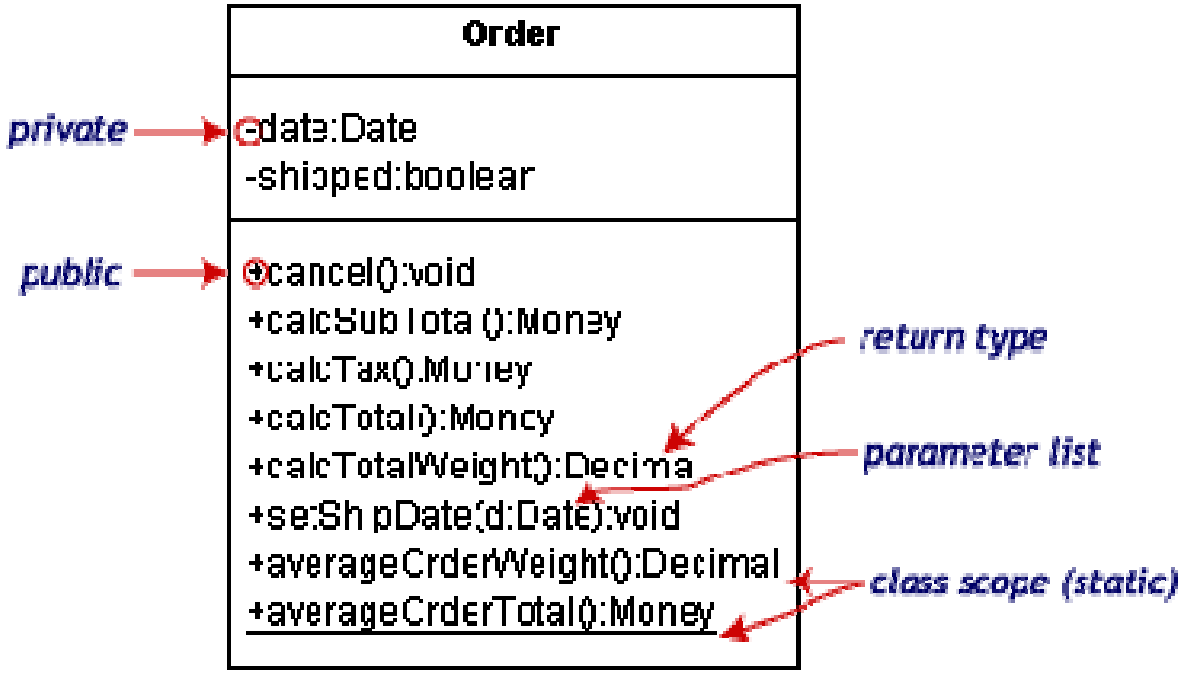
- Los métodos de una clase se especifican indicando el tipo de acceso (+, -, #), el nombre, la lista de parámetros y el tipo que devuelve

  - \+ Acceso público
  - \- Acceso privado
  - \# Acceso protegido

Programación orientada a objetos

Diagramas de clases

- El diagrama de una clase muestra el nombre de la clase, sus atributos y métodos



Programación orientada a objetos

Diagramas de clases

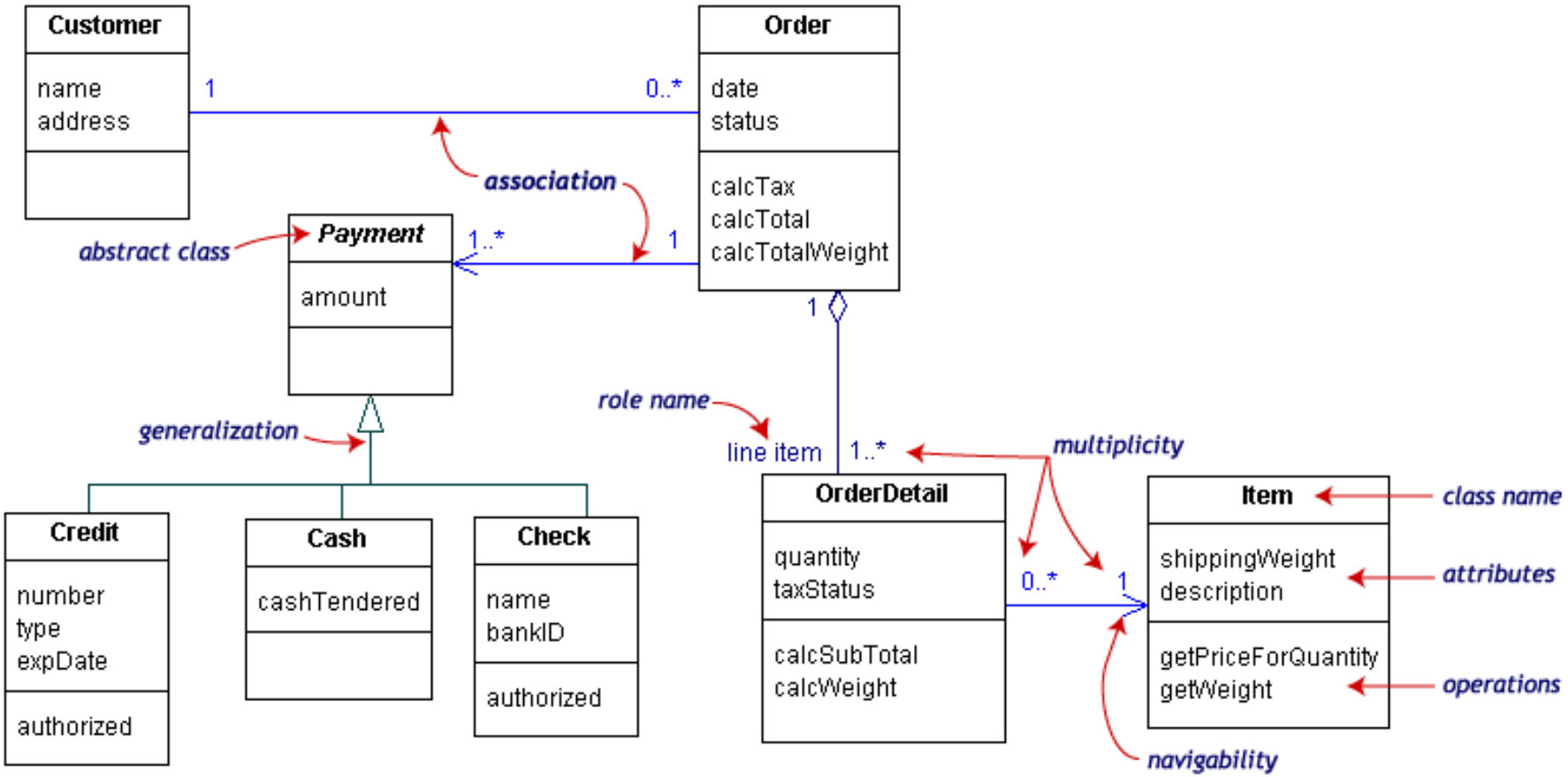
- El diagrama de clases ofrece una vista general de un sistema, muestra las clases y las relaciones existentes entre ellas

- El diagrama de clases es estático, muestra las interacciones entre clases pero no lo que sucede como resultado de esa interacción

- Cada diagrama se compone de clases y asociaciones. Cada asociación afecta a un número determinado de instancias de una clase, denominado “multiplicidad”

Programación orientada a objetos

Diagramas de clases



Programación orientada a objetos

Relaciones entre clases

- Asociación. Relaciona instancias de dos clases. Existe una asociación entre dos clases cuando una instancia de una clase debe saber sobre otra instancia para llevar a cabo sus funciones

- Agregación. Es un tipo de asociación donde una clase pertenece a una colección. Se utiliza un diamante en el extremo de la asociación de la clase que representa el todo

- Generalización. Indica una relación de herencia entre dos clases. Se utiliza un triángulo en el extremo de la relación para señalar a la superclase

Programación orientada a objetos

Asociaciones entre clases

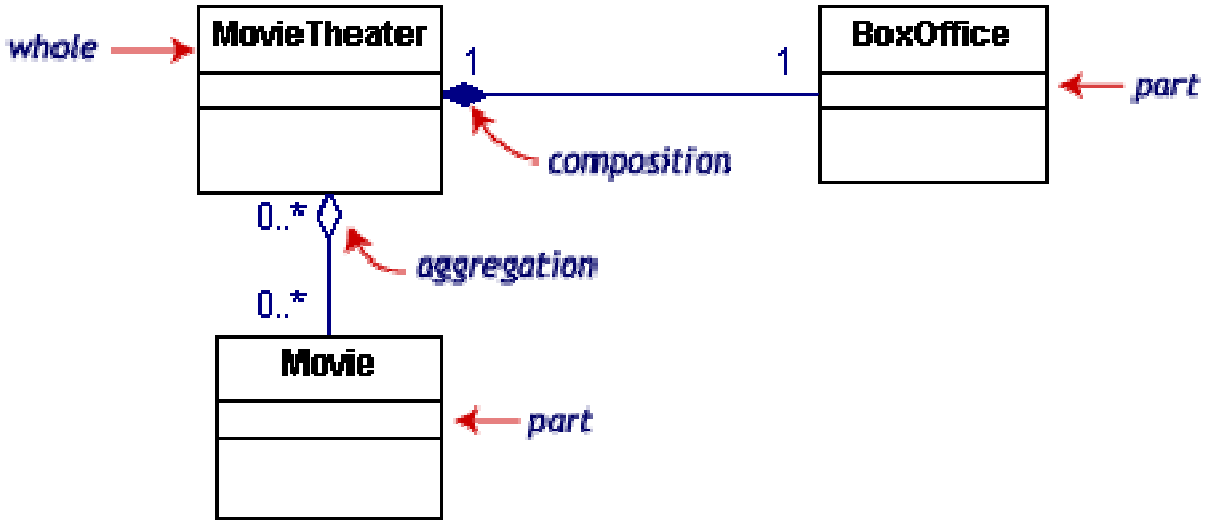
- Rol. Se utiliza para aclarar la naturaleza de la asociación
- Navegabilidad. La flecha de la asociación indica la dirección de navegación
- Multiplicidad. Determina el número posible de instancias

Multipicidad	Significado
0..1	Cero o una instancia. La notación *n..m* indica de *n* a *m* instancias
0..*	No existe límite de instancias, desde cero hasta un número ilimitado
1	Exactamente una instancia
1..*	Al menos una instancia

Programación orientada a objetos

Asociaciones entre clases

- Agregación: Asociación que relaciona a un objeto que es parte de otro
- Composición: Agregación fuerte, la parte no puede existir sin el todo



Programación orientada a objetos

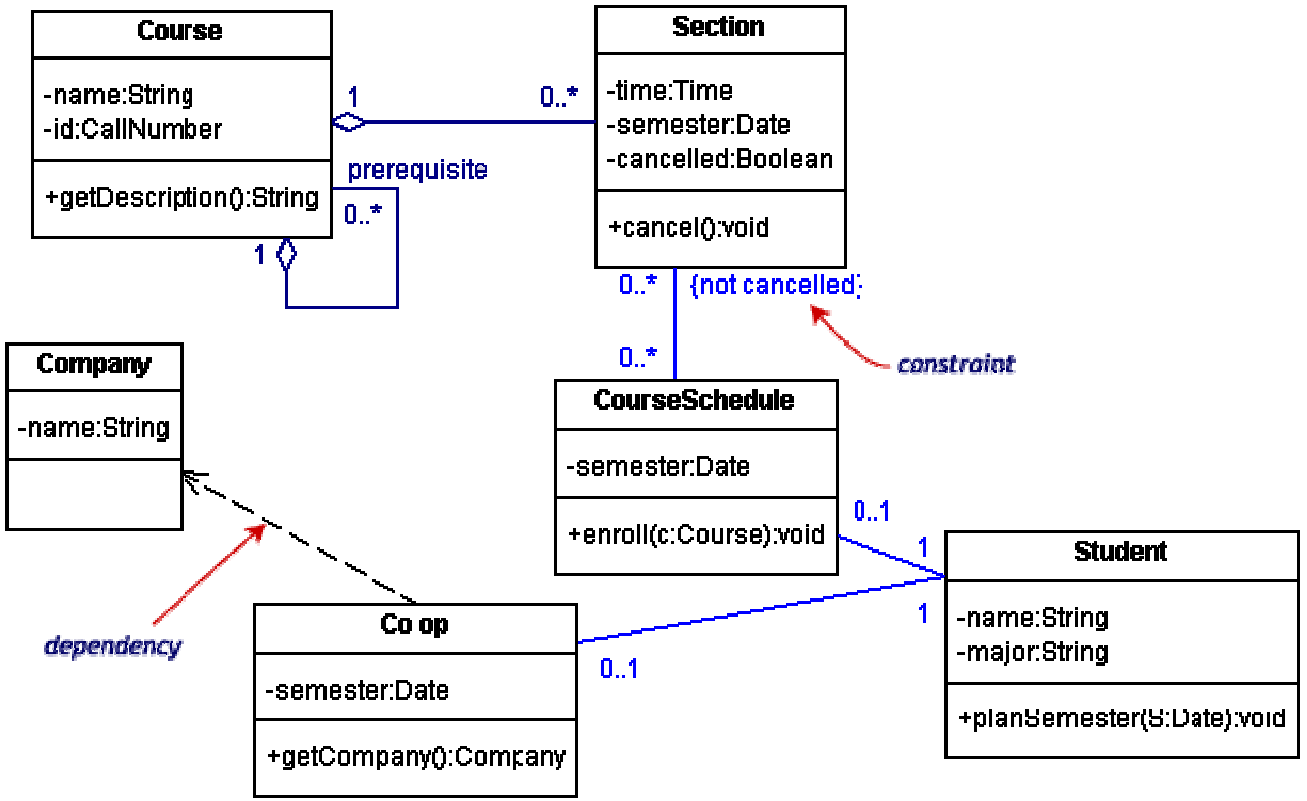
Dependencias y limitaciones

- Una dependencia es una relación entre dos clases. Si se produce un cambio en una de ellas puede producir cambios en la otra. Las dependencias se representan con líneas de puntos

- Una limitación es una condición que se aplica al diseño. Las limitaciones se expresan utilizando los símbolos { }

Programación orientada a objetos

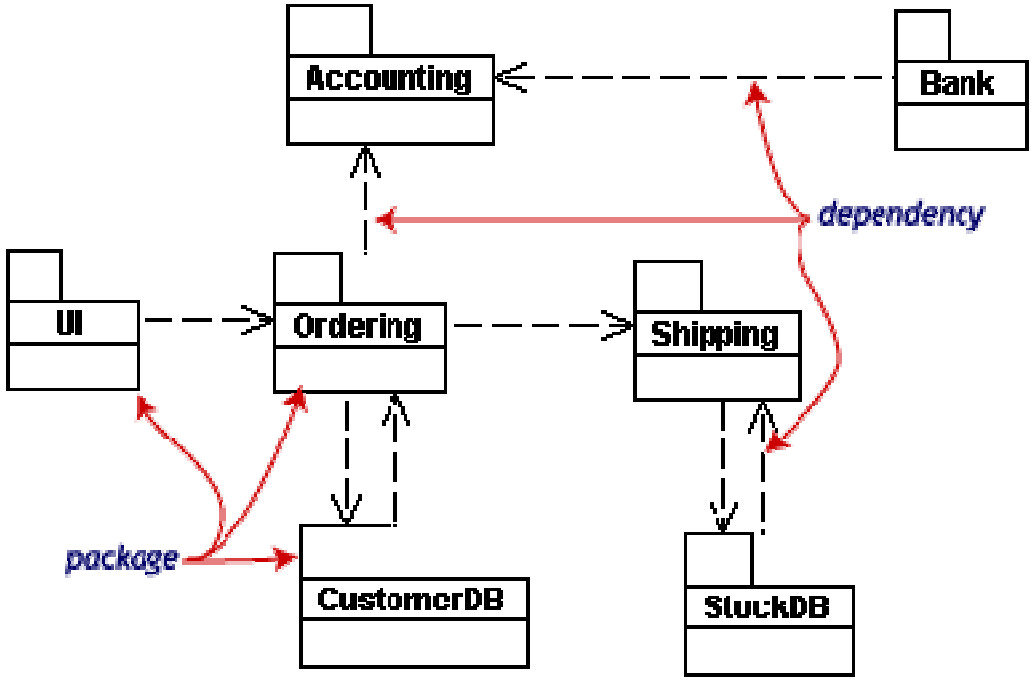
Dependencias y limitaciones



Programación orientada a objetos

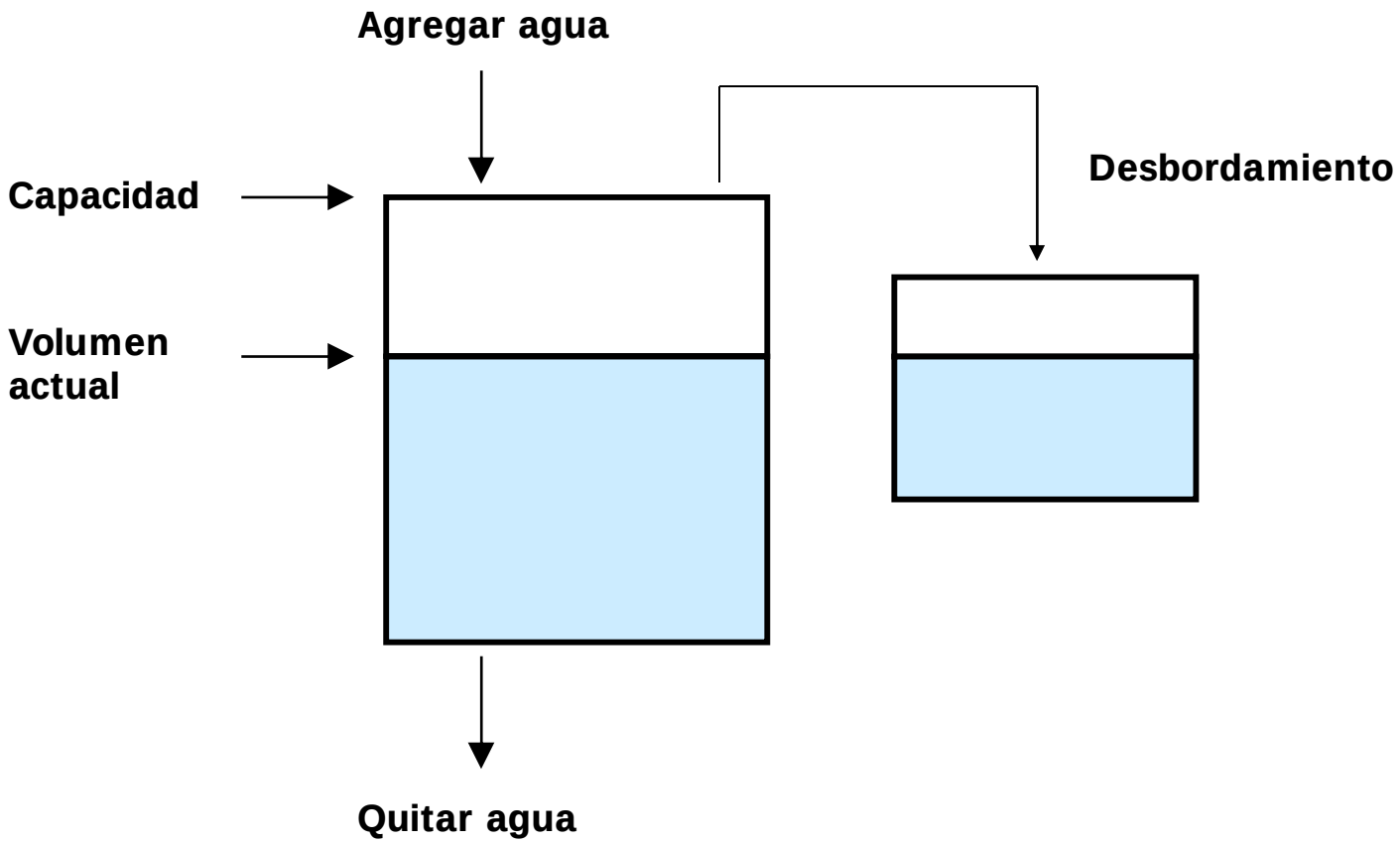
Diagramas de clases

- Las clases se agrupan en paquetes. El uso de paquetes permite organizar diagramas de clases complejos



Programación orientada a objetos

¿Cuál es el modelo de clases para un depósito de agua?



Programación orientada a objetos

¿Cuál es el modelo de clases para un depósito de agua?

El modelo de clases está compuesto por:

- **Elementos tangibles (objetos)**
- **Atributos (campos)**
- **Responsabilidades (operaciones o métodos)**
- **Limitaciones de diseño (contrato del objeto)**

Programación orientada a objetos

¿Cuál es el modelo de clases para un depósito de agua?

- **Elementos tangibles (objetos)**
  - **Depósito de agua**
- **Atributos (campos)**
  - **Capacidad (m^3)**
  - **Volumen actual (m^3)**

Programación orientada a objetos

¿Cuál es el modelo de clases para un depósito de agua?

- Responsabilidades (operaciones o métodos)
  - Agregar agua (m^3)
  - Quitar agua (m^3)

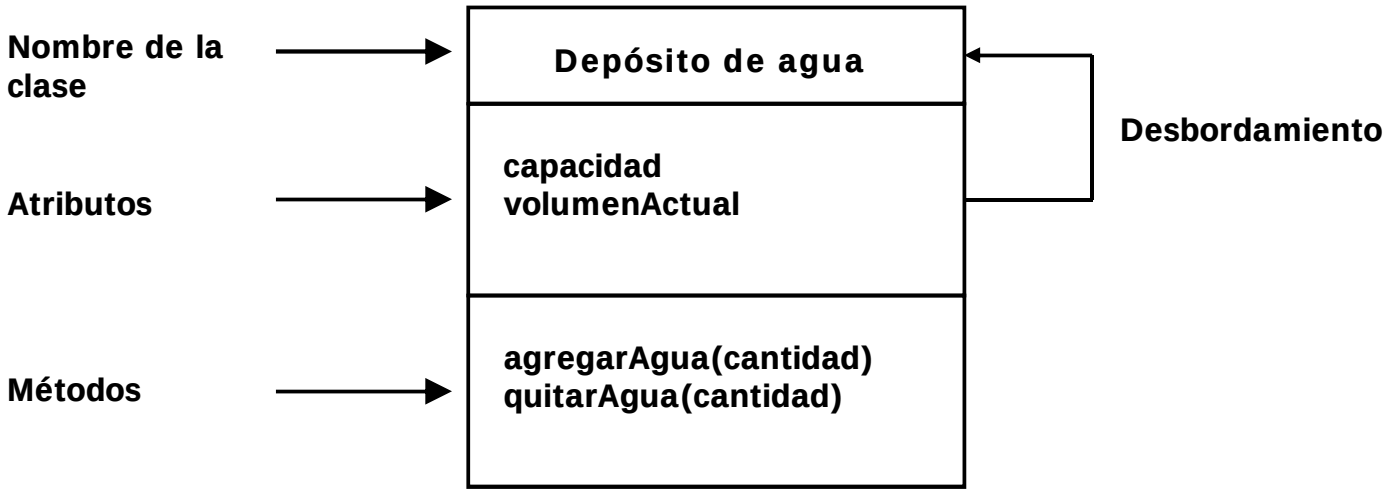
Programación orientada a objetos

¿Cuál es el modelo de clases para un depósito de agua?

- **Limitaciones de diseño (contrato del objeto)**
  - **La capacidad del depósito debe ser mayor que cero**
  - **La capacidad es constante y se expresa en m3**
  - **No se puede quitar agua de un depósito vacío**
  - **No se puede añadir agua si no existe un depósito para recoger el agua en caso de desbordamiento**
  - **Si al añadir agua hay desbordamiento, el agua que se desborda se debe almacenar en un depósito auxiliar**
  - **No se puede agregar y quitar agua al mismo tiempo**

Programación orientada a objetos

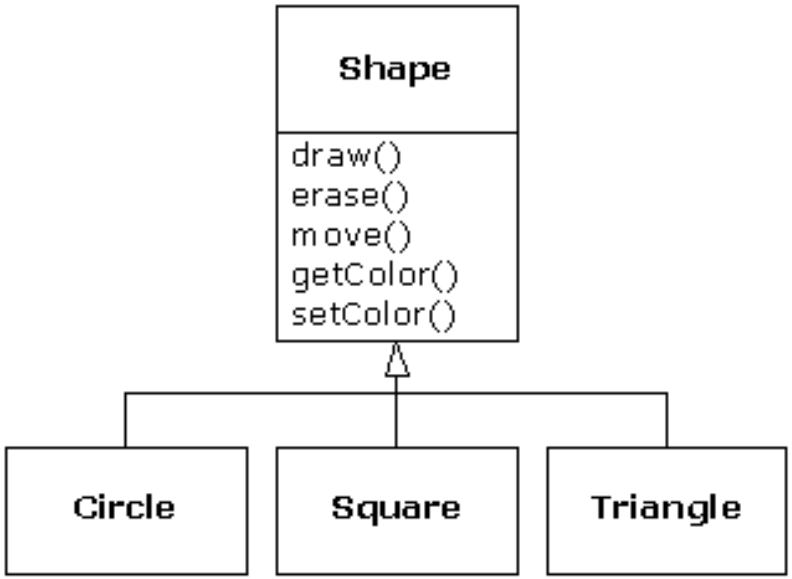
¿Cuál es el modelo de clases para un depósito de agua?



Programación orientada a objetos

Clases y objetos

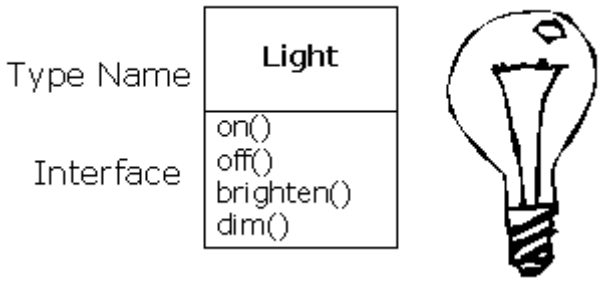
- Todos los objetos de una clase comparten las mismas características



Programación orientada a objetos

Clases y objetos

- A partir de una clase se puede obtener un objeto (instancia) que se comunica con otros objetos mediante mensajes
- Cada objeto tiene un estado propio
- Cada objeto responde a un conjunto de peticiones definidas en la interfaz de la clase a la que pertenece



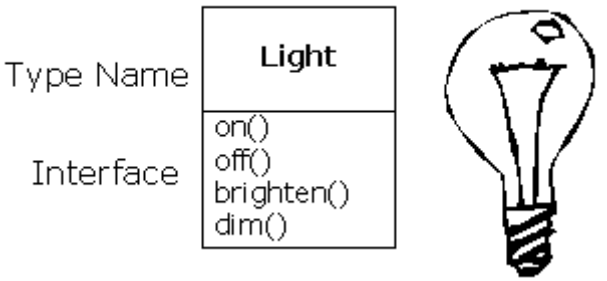
Programación orientada a objetos

Clases y objetos

- La interfaz ofrece al exterior el comportamiento del objeto. El código y los datos internos que se utilizan para responder a una petición se denomina implementación

- Un objeto se compone de:
  - Interfaz: qué se ofrece
  - Implementación: cómo se ofrece

```
Light bombilla = new Light();
bombilla.on();
```



Programación orientada a objetos

Encapsulación

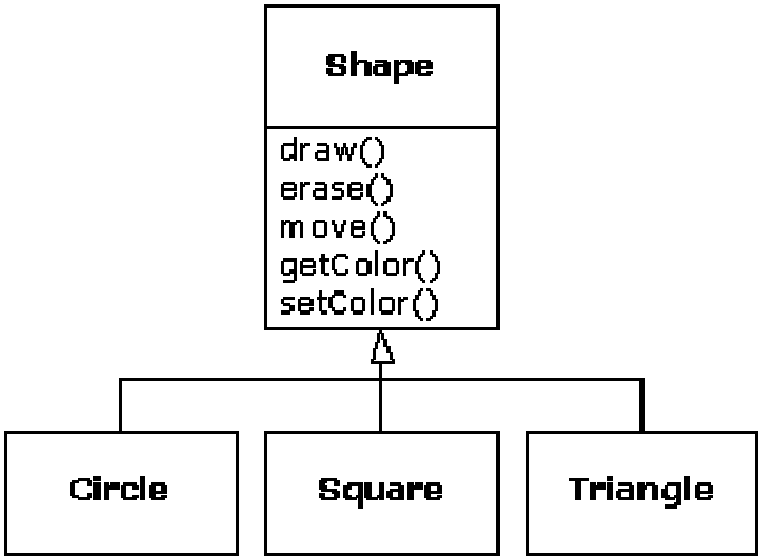
- La encapsulación permite ocultar los datos y la funcionalidad de un objeto. Además, facilita la reutilización de objetos

- La interfaz de la clase de un objeto permite que otros objetos accedan a los atributos y a los métodos públicos de un objeto

Programación orientada a objetos

Clases y subclases

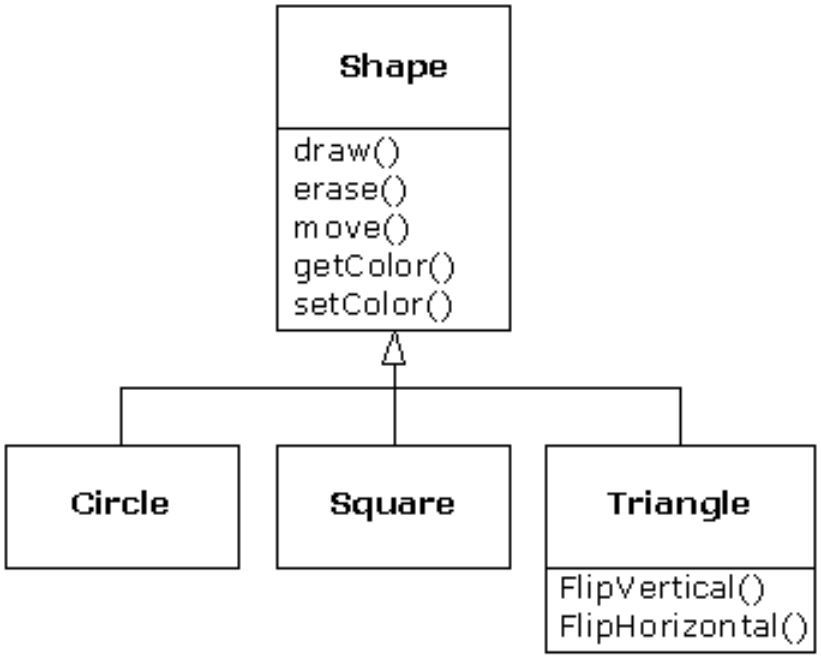
- La herencia permite diseñar clases a partir de otras clases. Una subclase puede añadir o modificar la funcionalidad de la clase de orden superior



Programación orientada a objetos

Clases y subclases

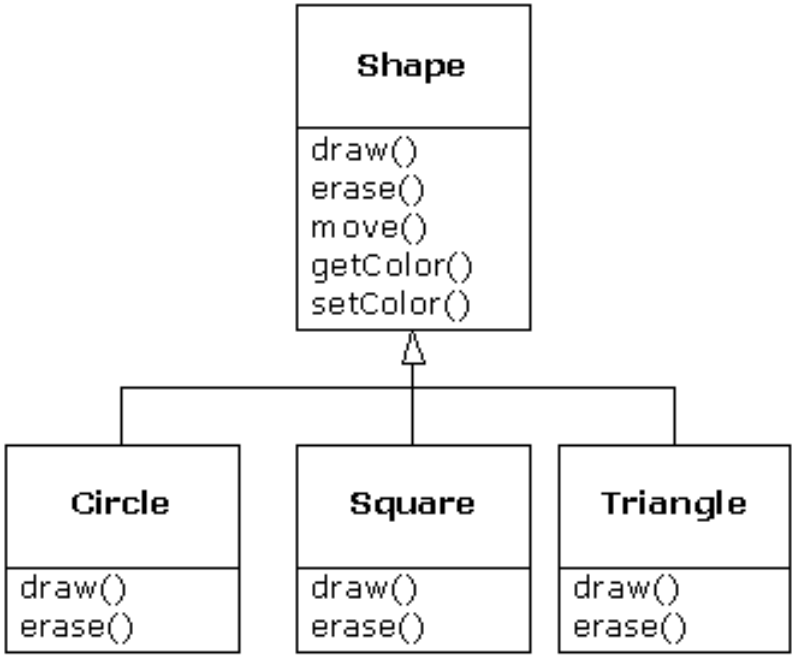
- Una subclase puede ampliar o extender el comportamiento de la clase de orden superior



Programación orientada a objetos

Clases y subclases

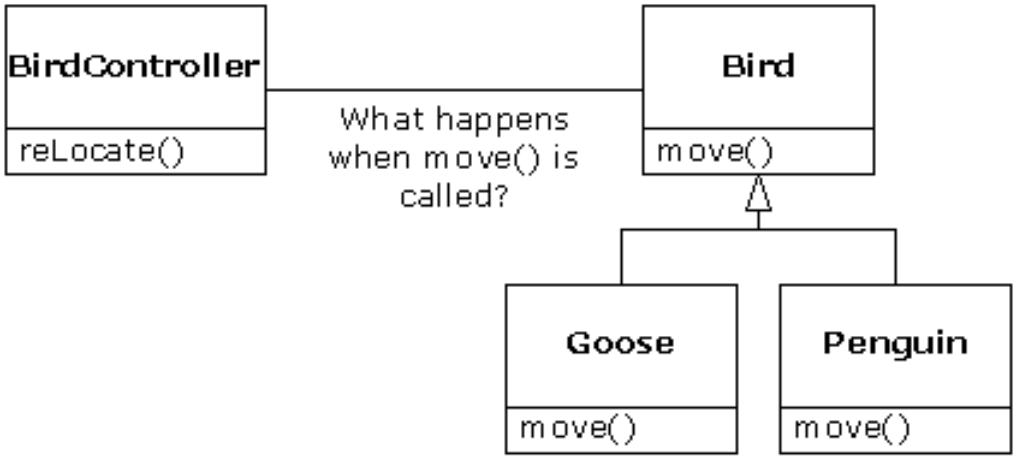
- Una subclase puede modificar el comportamiento de la clase de orden superior



Programación orientada a objetos

Polimorfismo y sobreescritura de métodos

- El polimorfismo es la capacidad de modificar el comportamiento de una clase mediante la sobreescritura de métodos



Programación orientada a objetos

¿Cómo identificar las clases y los métodos de un sistema?

- **Identificar las clases del sistema. Normalmente, las clases corresponden con los sustantivos que se utilizan para describir el sistema**
- **Definir los atributos de las clases. En general, los atributos coinciden con los adjetivos que se emplean para describir las clases**
- **Definir los métodos de cada clase. Normalmente los verbos o las acciones que describen el sistema corresponderán con los métodos**

Programación Java

Java es un lenguaje orientado a objetos

- **En Java, todo es un objeto. Los objetos se definen en términos de la clase a la que pertenecen**

  String nombre;

  Persona alumno;

- **El estado de un objeto se modifica utilizando los métodos de la clase**

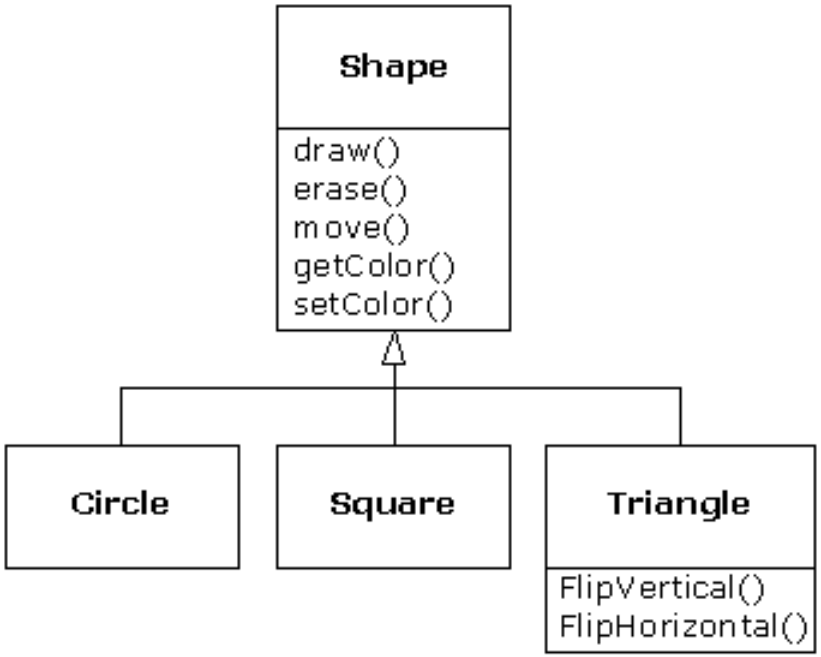
Programación Java

Clases y objetos

- Las clases representan objetos de la realidad (dominio) al que se refiere un programa Java
- Las clases se definen a partir de un conjunto de atributos y métodos. Cada atributo puede estar limitado a un dominio de valores
- Las clases son conceptos abstractos. Los objetos son instancias de las clases, se crean durante la ejecución de los programas

Clases y métodos

- Los métodos de las clases realizan acciones que modifican las propiedades de los objetos



Programación Java

El uso de métodos

- Los métodos se utilizan para estructurar un programa Java de forma lógica
- El uso de métodos permite dividir un problema en problemas más pequeños que son más fáciles de resolver (“Divide y vencerás”)
- Los métodos encapsulan un conjunto de instrucciones que se puede ejecutar tantas veces como sea necesario. El código del método queda oculto y sólo es necesario conocer su interfaz (parámetros y valor de retorno)

Programación Java

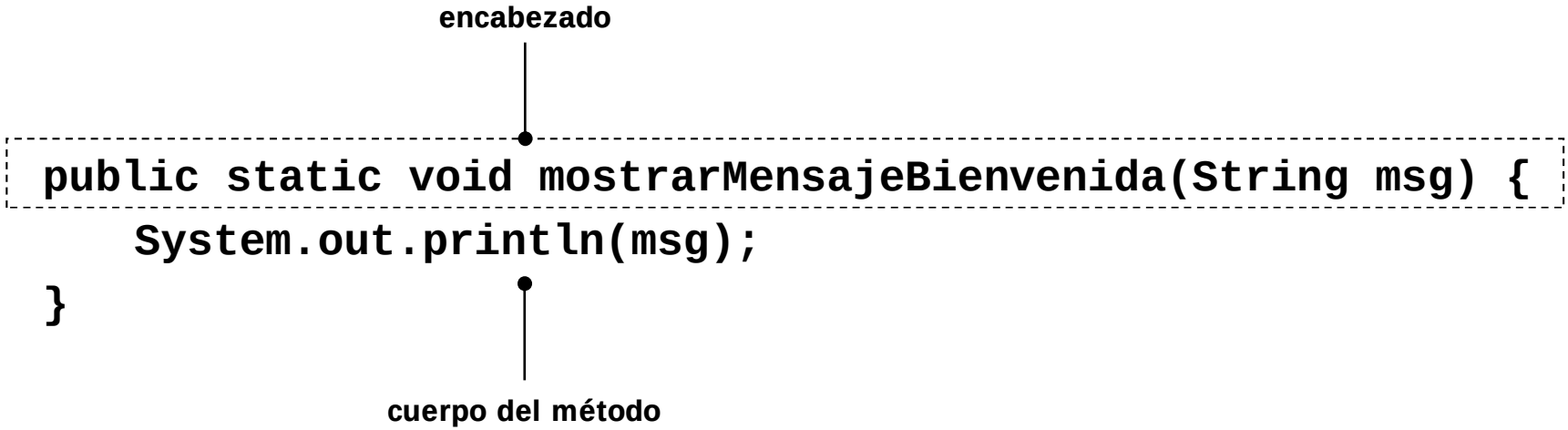
El uso de métodos

- Un método lleva a cabo una tarea específica de diversa naturaleza: input, output o cálculo

- Para crear un método se tiene que escribir la declaración del método, el encabezado y el cuerpo. Dentro del cuerpo pueden haber declaraciones de variables y constantes. Si las hay se conocen como variables locales y solo pueden ser utilizadas dentro del método

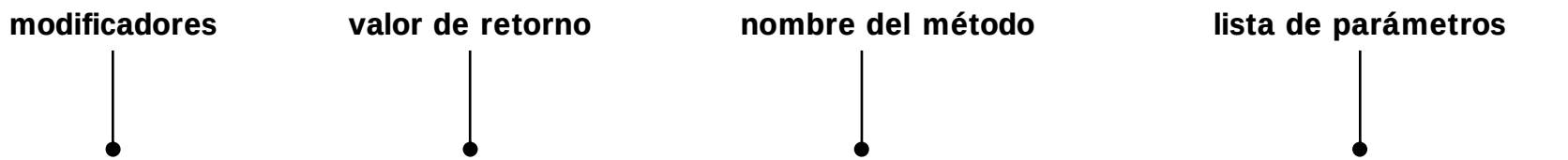
- En otros lenguajes, los métodos se denominan subrutinas o funciones

Declaración de un método: encabezado y cuerpo



Programación Java

El encabezado de un método




```
public static void mostrarMensajeBienvenida(String msg) {
    System.out.println(msg);
}
```

Programación Java

El encabezado de un método

- **Modificadores**

public	**el método puede invocarse fuera de la clase**
protected	**el método puede invocarse fuera de la clase**
private	**no se puede invocar fuera de la clase**
static	**el método pertenece a la clase, no a un objeto**

- **Tipo del valor de retorno (“Return type”)**

  int | long | double | String o cualquier otro tipo de dato para métodos que devuelven valores, void si el método no devuelve un valor

Programación Java

El encabezado de un método

- **Nombre del método**

  Debe ser un nombre descriptivo de la tarea que realiza el método

- **Lista de parámetros**

  La lista de parámetros es opcional. Contiene la declaración de parámetros (variables) que permiten al método recibir datos del exterior

Programación Java

Ejecución de métodos

- Un método se ejecuta cuando es invocado o llamado
- El método main se ejecuta automáticamente al comenzar la ejecución de un programa
- Para ejecutar un método basta indicarel nombre del método y la lista de parámetros

```
mostrarMensajeBienvenida(“Hola mundo”);
```

Ejecución de métodos

- Los valores que se pasan a un método cuando es invocado se denominan argumentos
- Para invocar un método que tiene en su lista de parámetros un valor entero se debe pasar una constante entera, una variable entera o una expresión de tipo entero

Programación Java

Ejecución de métodos

- Los métodos pueden ser de dos tipos: métodos sin valor de retorno o métodos con valor de retorno
- Los métodos sin valor de retorno se declaran con el valor de retorno void. El método main devuelve void
- Los métodos con valor de retorno se declaran con el tipo de dato correspondiente: int, char, long, double, String o cualquier otro tipo. Estos métodos utilizan la instrucción return para finalizar la ejecución del método, devolver el valor y el control del programa al punto donde se realizó la llamada

Programación Java

Ejecución de métodos

```
public static int suma (int num1, int num2){
    return (num1 + num2);
}
```

El valor de retorno se almacena en la variable entera c:

```
int a = 5, b = 10, c;

c = suma(5, 10);
c= suma(a, b);
```

Programación Java

Parámetros y argumentos

- El tipo del argumento que se pasa a un método al invocarlo tiene que corresponder con el tipo del parámetro en la declaración.
- La correspondencia entre argumentos y parámetros es uno a uno según el orden
- En el ejemplo anterior el 5 corresponde con num1 y el 10 con num2. Las constantes 5 y 10 son argumentos, mientras que num1 y num2 son parámetros. Los argumentos se usan en tiempo de ejecución y los parámetros en la definición del método

Programación Java

Parámetros, argumentos y valor de retorno

```
public static void main() {
   int x, y;

   Scanner teclado = new Scanner(System.in);

   System.out.print("Introduzca el valor de x: ");
   x = teclado.nextInt();

   System.out.print("Introduzca el valor de y: ");
   y = teclado.nextInt();

   System.out.print("La suma es: " + suma(x, y));
}
```

Programación Java

Parámetros, argumentos y valor de retorno

```
public static int leerNumero() {
   Scanner teclado = new Scanner(System.in);

   System.out.printl("Introduzca un numero: ");

   int x = teclado.nextInt();

   return x;
}
```

Programación Java

Parámetros, argumentos y valor de retorno

- Los argumentos que se pasan a un método deben ser compatibles con la el tipo de dato del parámetro
- El tipo de dato de la variable donde se recibe el valor de retorno de un método debe corresponder con el tipo de dato de retorno del método
- Java hace conversiones automáticas si el argumento es menor en tamaño que el parámetro (widening) pero no al revés (narrowing)

Programación Java

Parámetros, argumentos y valor de retorno

```
public static void main() {
   int x, y;

   x = leerNumero();
   y = leerNumero();

   System.out.println("La suma es: " + suma(x, y));
}
```

Programación Java

El control de errores de ejecución en los métodos

- Java requiere que todo método que se relacione con una entidad externa como un archivo de texto, controle los errores que se produzcan en tiempo de ejecución
- Java denomina excepciones a los errores en tiempo de ejecución
- El control de las excepciones de un programa Java se realiza con la instrucción throws IOException

Programación Java

El control de errores en tiempo de ejecución

- Una excepción es error que se produce en tiempo de ejecución de un programa Java
- Cuando se produce una excepción se envía un objeto (excepción) al método que ha provocado el error de ejecución
- Si el método que provoca el error no captura la excepción, entonces es la máquina virtual quien la captura y normalmente interrumpe la ejecución del programa

Programación Java

El control de errores en tiempo de ejecución

```
public static void main(String[] args) {
    int suma, num[] = { 1, 2, 3 };

    for (int i=0; i <= 3; i++) {
      suma=+num[i];
      System.out.println(num[i]);
    }

    System.out.println(suma/0);
}
```

Programación Java

El control de errores en tiempo de ejecución

- En el ejemplo anterior se producen dos excepciones:

  java.lang.ArrayIndexOutOfBoundsException
  java.lang.ArithmeticException

- IndexOutOfBounds se produce al acceder a num[3] en la iteración del for para i = 3. El array num almacena 3 valores, en los índices 0, 1 y 2, el índice 3 no existe

- ArithmeticException se produce al dividir por cero en la última instrucción del programa

Programación Java

Las instrucciones try, catch, finally

- try ejecuta un bloque de código y lanza la excepción a la primera sentencia catch que gestione ese tipo de excepción
- catch captura la excepción y realiza las acciones correspondientes
- finally se ejecuta al final de try y después de catch. Captura las excepciones no capturadas por sentencias catch

Las instrucciones try, catch, finally

```
try {
        // bloque de código que gestiona la
excepción
}
catch ( tipoExcepción1 objeto ) {
        // Gestión del tipo de excepción 1
}
catch ( tipoExcepción2 objeto ) {
        // Gestión de tipo de excepción 2
}
finally {
        // Gestión de otras excepciones
}
```

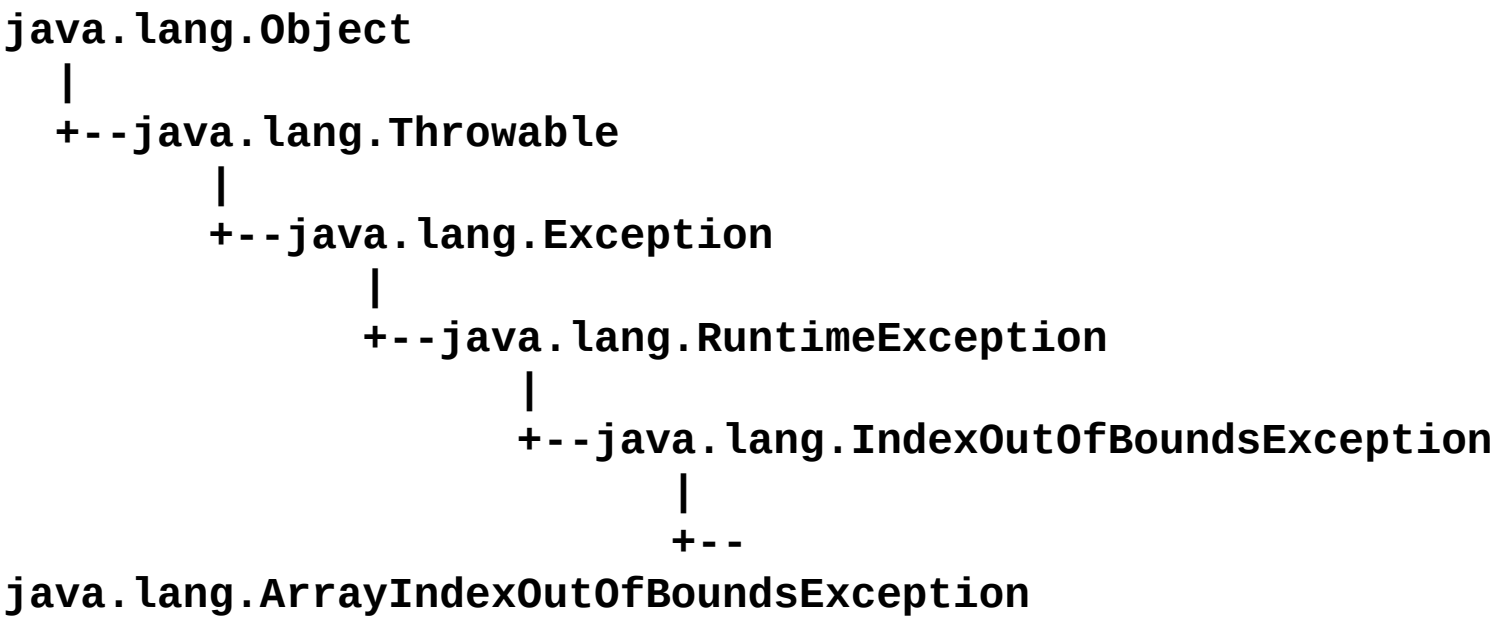
Programación Java

Las instrucciones try, catch, finally

```
public static void main(String[] args) {
   try {
     int suma, num[] = { 1, 2, 3 };

     for (int i=0; i <= 3; i++) {
       suma=+num[i];
       System.out.println(num[i]);
     }
   }
   catch (ArrayIndexOutOfBoundsException e) {
    System.out.print("i fuera de rango " + e.getMessage());
   }
   finally {
      System.out.println(e.getMessage());
    }
}
```

Jerarquía de excepciones



Programación Java

Jerarquía de excepciones

```
public static void main(String[] args) {
   try {
     int suma, num[] = { 1, 2, 3 };

     for (int i=0; i <= 3; i++) {
       suma=+num[i];
       System.out.println(num[i]);
     }
   }
   catch (ArrayIndexOutOfBoundsException e) {
    System.out.print("i fuera de rango " + e.getMessage());
   }
   catch (RuntimeException e){
    System.out.println("error " + e.getMessage());
   }
}
```

Programación Java

La instrucción throws

- throws indica al compilador las excepciones que puede lanzar un método
- Es necesario para todas las excepciones, excepto para las de tipo Error o RuntimeException y sus subclases
- Si se utiliza input-output es necesario declarar la excepción throws IOException. Si no se hace, el compilador muestra un mensaje de error indicando que la excepción no está declarada

Programación Java

Las excepciones de input-output

```
public static void main(String[] args) throws IOException {
   try {
     Scanner teclado = new Scanner(System.in);

     System.out.printl("Introduzca un numero: ");

     int x = teclado.nextInt();

   }
   catch (NumberFormatException e) {
     System.out.print("Numero no válido !!" );
   }
   catch (IOException e) {
     System.out.print( "Error de entrada" );
   }
}
```

Programación Java

Clases y objetos

- Una clase es una representación abstracta de un conjunto de objetos. Los objetos de una clase tienen las mismas características y el mismo comportamiento
- Un objeto es una instancia de una clase. En Java los objetos se crean con el operador new
- Cada objeto tiene sus propios atributos, lo que le diferencia de otros objetos pertenecientes a la misma clase

Programación Java

Clases y objetos

- La clase String permite definir objetos para almacenar cadenas de caracteres

```
String hola = new String(“Hola”)
String mundo = new String(“mundo”);
```

- Los objetos hola y mundo son cadenas de caracteres, cada objeto ocupa un espacio en memoria y tiene sus propios atributos, “hola” y “mundo”, respectivamente

Programación Java

Clases y objetos

- Las clases se utilizan para definir tipos de datos o módulos
- Un tipo de dato describe un conjunto de objetos y operaciones (métodos). Un módulo es una unidad de descomposición de software

Programación Java

Clases y objetos

- Un objeto es una instancia de una clase. Encapsula estado y comportamiento. Los objetos se crean por instanciación de las clases
- Un objeto puede describir una entidad física o una entidad abstracta
- Cada objeto tiene sus propios atributos, lo que le diferencia de otros objetos pertenecientes a la misma clase

Programación Java

Objetos

- Un objeto tiene las siguientes características
  - Identidad
  - Estado
  - Comportamiento
- La identidad de un objeto le permite distinguirse de otros objetos. El estado de un objeto está determinado por sus datos y el comportamiento del objeto está determinado por los métodos de la clase a la que pertenece

Programación Java

Objetos

- **Identidad**

  La identidad de un objeto le identifica unívocamente y no cambia durante la vida del objeto. Es independiente del estado del objeto

- **Estado**

  El estado de un objeto evoluciona en el tiempo, está determinado por los valores de sus atributos. Cada atributo toma un valor en un dominio

Objetos

- **Comportamiento**

  Los métodos que definen el comportamiento de un objeto describen sus acciones y agrupan sus responsabilidades

  Las acciones de un objeto dependen de su estado y de un estímulo externo, un mensaje enviado por otro objeto

  El estado y el comportamiento están relacionados. Por ejemplo, un avión no puede aterrizar si está en tierra

Programación Java

Objetos: interfaz vs. implementación

- Los objetos se relacionan a través de interfaces bien definidas. No es necesario que conozcan los detalles de la implementación de otros objetos
- La interfaz del objeto está definida por los métodos públicos de la clase

Programación Java

Las clases en Java

- Las clases Java se definen en ficheros independientes, normalmente con extensión .java
- La clase se carga en memoria cuando es necesario
- Para definir una clase en Java se utiliza la palabra reservada class

```
public class MiClaseJava {

}
```

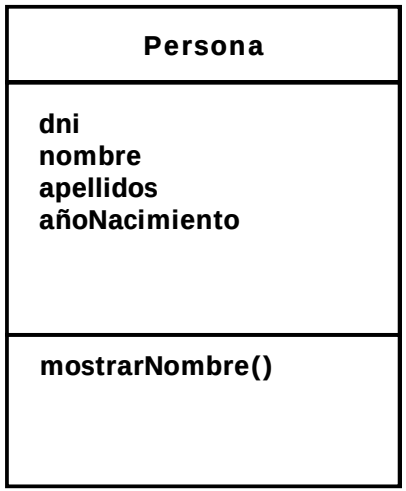
Programación Java

Las clases en Java

- El nombre de una clase debe ser un identificador válido en Java
- Por norma, los nombres de las clases comienzan con mayúsculas
- Las clases públicas deben definirse en ficheros .java

Las clases en Java

- La clase Persona tiene cuatro atributos: dni, nombre, apellido y año de nacimiento
- El método mostrarNombre() muestra el nombre por la consola



Las clases en Java

```
public class Persona {
    public String dni, nombre, apellidos;
    public int añoNacimiento;

    public void mostrarNombre() {
       System.out.println(nombre + " " + apellidos);
    }
}
```

Las clases en Java

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

        Persona p1 = new Persona();
        Persona p2 = new Persona();

        p1.nombre = "Juan";
        p1.apellidos = "Gonzalez";

        p2.nombre = "Luis";
        p2.apellidos = "Gómez";

        p1.mostraNombre();
        p2.mostrarNombre();
    }
}
```

Programación Java

Métodos de clases

- Por norma, debe evitarse el uso de propiedades públicas. Es preferible definir métodos para modificar el valor de las propiedades de un objeto

- Los métodos set se utilizan para modificar (escribir) el valor de los atributos de un objeto

- Los métodos get se utilizan para consultar (leer) el valor de los atributos de un objeto

  getNombre()
  setNombre()

Métodos de clases

```
public class Persona {
    private String dni, nombre, apellidos;
    private int añoNacimiento;

    public void mostrarNombre() {
       System.out.println(nombre + " " + apellidos);
    }

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

    public set añoNacimiento(int año) {
       añoNacimiento = año;
    }
}
```

Programación Java

Métodos de clases

- En este ejemplo, el método setNombre(String nombre) recibe el parámetro nombre para modificar el atributo nombre de la clase. La palabra reservada this hace referencia a la instancia de la clase y permite diferenciar estas dos variables. El método setNombre asigna el valor del parámetro nombre al atributo nombre del objeto

```
this.nombre = nombre;
```

- El método setAñoNacimiento no utiliza this porque el identificador del parámetro (año) es distinto del identificador del atributo de la clase (añoNacimiento)

Métodos de clases

```
public class Cuenta {
    private double saldo, double limiteCredito;

    public void ingresar(double cantidad) {
       saldo += cantidad;
    }
    public void retirar(double cantidad) {
       saldo -= cantidad;
    }
    public void setLimiteCredito(double cantidad) {
       limiteCredito = cantidad;
    }
    public double getLimiteCredito() {
       return limiteCredito;
    }
}
```

Programación Java

Constructores

- El operador new crea una instancia de la clase invocando un método especial de la clase, denominado constructor

- El método constructor inicializa el estado del objeto

```
Cuenta miCuentaCorriente = new Cuenta();
```

- Por defecto, Java crea un constructor sin parámetros. El valor inicial de los atributos del objeto depende de la inicialización de las variables de instancia

Constructores

- Java permite definir métodos constructores para especificar los valores iniciales de los objetos
- El nombre de los métodos constructores coincide con el nombre de la clase

Constructores

```
public class Cuenta {
    private double saldo, double limiteCredito;
    private static final double LIMITE = 500;

    public Cuenta() {
       this.balance = 0;
       this.limiteCredito = LIMITE;
    }

    public Cuenta(double limiteCredito) {
       this.balance = 0;
       this.limiteCredito = limiteCredito;
    }

}
```

Constructores

```
public class MisCuentas {

    public static void main(String[] args) {

        Cuenta miCuenta1 = new Cuenta();
        Cuenta miCuenta2 = new Cuenta(6000);

        System.out.print("Limite cuenta 1 (normal): ");
        System.out.println(miCuenta1.getLimiteCredito());

        System.out.print("Limite cuenta 2 (vip): ");
        System.out.println(miCuenta2.getLimiteCredito());

    }
}
```

Programación Java

Constructores

- Java permite definir varios constructores diferentes para crear los objetos de la clase. La declaración de los constructores debe ser diferente y deben utilizar distintos parámetros

- En este ejemplo se han definido dos constructores, uno sin parámetros y otro con un parámetro para especificar el límite de crédito de la cuenta

Programación Java

Referencias a objetos

- Cualquier tipo definido en Java con una clase es un tipo no primitivo
- Cuando se declara una variable de tipo primitivo, se reserva un espacio de memoria para almacenar su valor
- Cuando se declara una variable de tipo no primitivo, se reserva un espacio de memoria para almacenar la referencia al objeto, no el objeto en sí. Es por esto que es necesario utilizar el operador new

Referencias a objetos

- Cuando se utilizan tipos de datos primitivos las sentencias de asignación copian el valor de las variables

```
int x = 100;
int y = 200;
int z;

z = y;      // z almacena el valor inicial de y (200)
y = x;      // y almacena el valor inicial de x (100)
x = z;      // el valor de x es 200
```

Programación Java

Referencias a objetos

- Cuando se utilizan tipos de datos no primitivos las sentencias de asignación copian la referencia al objeto

```
Cuenta miCuenta1 = new Cuenta();
Cuenta miCuenta2 = new Cuenta(6000);
Cuenta miCuenta3;

miCuenta3 = miCuenta2;  // se copia la referencia a miCuenta2
miCuenta3.retirar(2000); // hace un retiro de miCuenta2
```

Programación Java

Inicialización por defecto de las variables de instancia

- Las variables de tipo numérico byte, short, int o long se inicializan a cero. Las variables float se inicializan a +0.0f y las variables double a +0.0
- Las variables de tipo char se inicializan a '\u0000'
- Las variables de tipo boolean se inicializan a false
- Las referencias a objetos se inicializan a null

Programación Java

Ámbito de las variables

Los principales tipos de ámbitos en Java son:

- Ámbito de objeto. Los atributos de un objeto que no se han declarado static
- Ámbito de método. Las variables y los objetos declarados en un método
- Ámbito de clase. Las variables declaradas static en una clase

Ámbito de las variables

Los principales tipos de ámbitos en Java son:

- Ámbito de objeto. Los atributos de un objeto están ligados al espacio de vida del objeto y son accesibles por cualquier método del objeto que no sea static.
- Ámbito de método. Las variables y los objetos declarados en un método están delimitadas por el ámbito dfel método

Ámbito de las variables

- Ámbito de clase. Las variables declaradas static en una clase viven independientemente de las instancias de la clase. Si las variables son públicas, es posible acceder a ellas desde que se declara la clase

Ámbito de las variables

```
public class Cuenta {
    private double saldo, double limiteCredito;
    public static final double LIMITE = 500;

    public Cuenta() {
       this.balance = 0;
       this.limiteCredito = LIMITE;
    }

    public Cuenta(double limiteCredito) {
       this.balance = 0;
       this.limiteCredito = limiteCredito;
    }

}
```

Ámbito de las variables

- Las variables saldo y limiteCredito son variables de objeto
- La variable LIMITE es una variable de clase. Es una variable pública compartida por todas las instancias de la clase

Programación Java

Ámbito de los métodos de clase

- Un método static se puede invocar desde la clase, sin necesidad de una instancia

```
public static void main (String[] args) {
   String pi = new String("3.141592");
   double PI;

   PI = Double.parseDouble(pi);
}
```

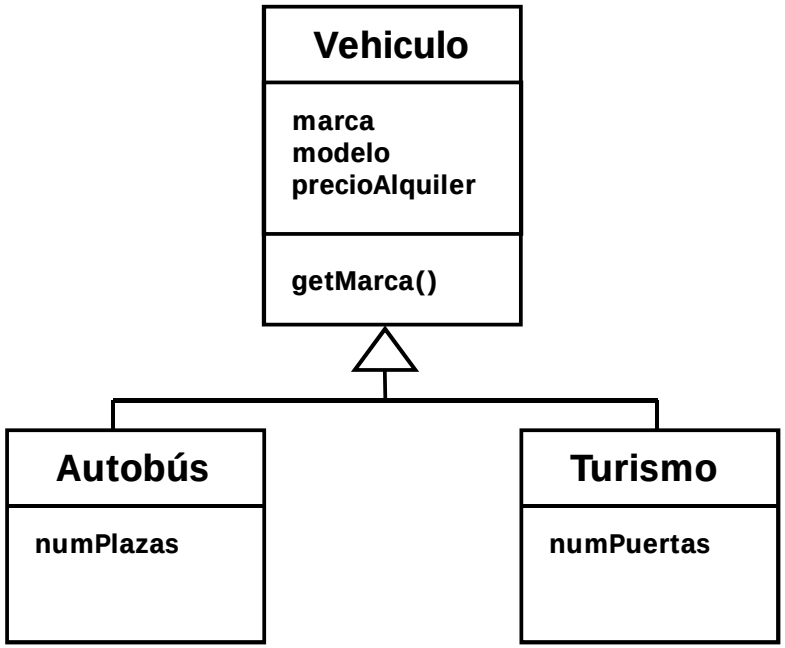
- parseDouble es un método static de la clase Double

Relaciones entre clases

- En el diseño orientado a objetos existen dos tipos de relaciones básicas: la herencia y la agregación
- La herencia se utiliza para especializar y extender las características y el comportamiento de una clase
- La agregación se utiliza cuando existe una relación jerárquica entre las clases. Esta relación puede ser del tipo “es parte de” o “forma parte de”

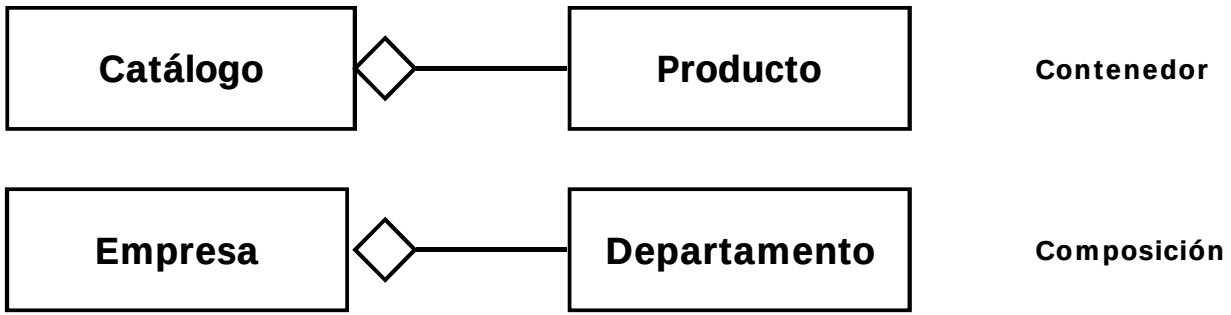
Relaciones entre clases: Herencia

- En las relaciones de herencia las subclases heredan atributos de la superclase (marca, modelo y precio) y añaden atributos propios



Relaciones entre clases: Agregación

- En las relaciones de agregación se existe el todo y la parte. Puede ser de tipo composición o contenedor. En la composición la parte no puede existir sin el todo



Herencia y polimorfismo

```
public class Vehiculo {
   private String marca, modelo;
   private double precioAlquiler;

   public Vehiculo(String marca, String modelo) {
      this.marca = marca;
      this.modelo = modelo;
   }
   public void setPrecioAlquiler(double precioAlquiler) {
      this.precioAlquiler = precioAlquiler;
   }
   public void imprimirAtributos() {
      System.out.print(" Modelo: " + marca + " " +
         modelo + "\t" + "Precio: " + precioAlquiler);
   }
}
```

Herencia y polimorfismo

```
public class Autobus extends Vehiculo {
   private int numPlazas;

   public Autobus(String marca, String modelo,
                  double precioAlquiler, int numPlazas) {
      super(marca, modelo, precioAlquiler);
      setPlazas(numPlazas);
   }
   public void setPlazas(int numPlazas) {
      this.numPlazas = numPlazas;
   }
   public void imprimirAtributos() {
      super.imprimirAtributos();
      System.out.print("\t" + "Plazas : " + numPlazas);
   }
}
```

Herencia y polimorfismo

```
public class Turismo extends Vehiculo {
   private int numPuertas;

   public Turismo(String marca, String modelo,
                  double precioAlquiler, int numPuertas) {
      super(marca, modelo, precioAlquiler);
      setPuertas(numPuertas);
   }
   public void setPuertas(int numPuertas) {
      this.numPuertas = numPuertas;
   }
   public void imprimirAtributos() {
      super.imprimirAtributos();
      System.out.print("\t" + "Puertas: " + numPuertas);
   }
}
```

Herencia y polimorfismo

- Las clases Autobus y Turismo son subclases de Vehiculo. Estas clases son especializaciones de Vehiculo y “extienden” sus atributos y comportamiento

- Los constructores de las subclases utilizan el constructor de la clase Vehiculo. El constructor de ls superclase se invoca con super

- El método imprimirAtributos() se sobreescribe en las subclases Autobus y Turismo para mostrar los atributos propios de la subclase

Herencia y polimorfismo

```
public class Vehiculos {
 public static void main(String[] args) {
   List listaVehiculos = new ArrayList();

   listaVehiculos.add(new Vehiculo("Mercedes","E",55));
   listaVehiculos.add(new Autobus("Mercedes","M20",100,20));
   listaVehiculos.add(new Turismo("BMW","525",50,4));
   listaVehiculos.add(new Turismo("Audi","A4",55,4));
   listaVehiculos.add(new Autobus("Mercedes","M10",120,30));
   listaVehiculos.add(new Turismo("VW","Passat",35,4));
   listaVehiculos.add(new Turismo("Audi","A3",30,2));
   listaVehiculos.add(new Turismo("Mercedes","E",60,4));
   listaVehiculos.add(new Vehiculo("Audi","A3",30));
   imprimirElementosLista((ArrayList)listaVehiculos);
 }
}
```

Herencia y polimorfismo

- La clase main utiliza el objeto listaVehiculos, una instancia de la clase List, para almacenar los vehículos

- Cuando se añade un objeto vehículo a listaVehiculos, se invoca al constructor de la clase correspondiente: Vehiculo, Autobus o Turismo

```
Turismo("Audi","A4",55,4)
Vehiculo("Mercedes","E",55)
Autobus("Mercedes","M20",100,20)
```

Modificadores de acceso a clases de un mismo paquete

Modificador	Heredado	Accesible
default (sin modificador)	SI	SI
public	SI	SI
protected	SI	SI
private	NO	NO

Modificadores de acceso a clases de otro paquete

Modificador	Heredado	Accesible
default (sin modificador)	NO	NO
public	SI	SI
protected	SI	NO
private	NO	NO

Modificadores de acceso

- Los atributos y métodos private de una superclase no se heredan en las subclases
- Los atributos y métodos protected de una superclase se heredan y son accesibles desde las subclases
- Los atributos y métodos public de una clase son accesibles dentro y fuera del paquete al que pertenece la clase
- Si no es utiliza un delimitador, entonces el atributo o el método son públicos

Programación Java

Modificadores de acceso

- Utilizar public si el atributo o método se debe heredar y ser usado desde otro paquete
- Utilizar protected si el atributo o método se debe heredar y sólo debe ser usado desde el paquete al que pertenece la clase
- Utilizar private si el atributo o método es privado de la clase

Clases abstractas

- Las clases abstractas se utilizan para especificar una interfaz y no implementan los métodos
- Las clases abstractas heredan métodos abstractos, métodos declarados que no han sido implementados. Las subclases se encargan de implementar los métodos
- Las clases abstractas no se pueden instanciar

Programación Java

La palabra reservada final

- Si final precede a una variable, es una constante que no se puede modificar
- Si final precede a un método, no se puede sobreescribir
- Si final precede una clase, no permite definir subclases a partir de ella

Composición y agregación

- La composición se utiliza para definir clases compuestas por otras clases. En este tipo de relaciones todo-parte, normalmente la parte no puede existir sin el todo (p.e. los departamentos de una empresa)
- Si se trata de una clase contenedor, entonces la parte puede existir sin necesidad del todo (p.e. los productos de un catálogo)
- Para crear contenedores es recomendable utilizar las clases Vector o List

Vectores

- La clase Vector de Java es un array de longitud variable. Esta clase tiene los siguientes constructores

Vector()	Tamaño 100 y se duplica el tamaño
Vector(int size)	Tamaño dado y se duplica el tamaño
Vector(int size, int inc)	Tamaño dado e incremento

- Para añadir un elemento al vector se invoca el método add, indicando el objeto a insertar add(Object element)

```
Vector listaVehiculos = new Vector(50);
listaVehiculos.add(new Turismo("Audi","A4",55,4))
```

Vectores

- La clase Vector define métodos que tienen en cuenta el índice o posición dentro del Vector
- El primer elemento se almacena en la posición cero. Para insertar un objeto en una posición determinada se utiliza el método insertElementAt(Object element, int position)
- Cuando se utiliza este método el resto de elementos del vector son desplazados
- El método size() devuelve el número de elementos del vector, el método capacity() indica su capacidad máxima

Vectores

- En el siguiente ejemplo listaVehiculos.size() es 1 y listaVehiculos.capacity() es 50

```
Vector listaVehiculos = new Vector(50);
listaVehiculos.add(new Turismo("Audi","A4",55,4))
```

- El método removeElementAt(int position) elimina el objeto almacenado en una posición del vector y desplaza los elementos para llenar la posición libre

- Los métodos elementAt(int position) y get(int position) devuelven el objeto almacenado en la posición indicada

Listas

- La clase ArrayList es una lista ordenada de objetos. Esta clase tiene los siguientes constructores

ArrayList()	Tamaño por defecto
ArrayList(int initialCapacity)	Tamaño dado
ArrayList(Collection coll)	Tamaño dado por la colección

- El métodos add(Object element) inserta un objeto en la lista y add(Object element, int position) inserta un elemento en la posición indicada. Para acceder a un objeto de la lista se utiliza el método get(int position)

Listas

```
public class Vehiculos {
 public static void main(String[] args) {
   List listaVehiculos = new ArrayList();

   listaVehiculos.add(new Vehiculo("Mercedes","E",55));
   listaVehiculos.add(new Autobus("Mercedes","M20",100,20));
   listaVehiculos.add(new Turismo("BMW","525",50,4));
   listaVehiculos.add(new Turismo("Audi","A4",55,4));
   listaVehiculos.add(new Autobus("Mercedes","M10",120,30));
   listaVehiculos.add(new Turismo("VW","Passat",35,4));
   listaVehiculos.add(new Turismo("Audi","A3",30,2));
   listaVehiculos.add(new Turismo("Mercedes","E",60,4));
   listaVehiculos.add(new Vehiculo("Audi","A3",30));
   imprimirElementosLista((ArrayList)listaVehiculos);
 }
}
```

Listas

```
private static void imprimirLista(ArrayList lista) {

   for (int i = 0; i < lista.size(); i++) {
      Vehiculo v = (Vehiculo)lista.get(i);

      v.imprimirAtributos();

      System.out.println("");
   }
}
```

Introspección

Información sobre las clases

- Java ofrece utilidades de introspección para obtener información interna de las clases. Los objetos Java pueden informar sobre su propia estructura
- La introspección permite que un objeto o una clase informe de la clase a la que pertenece
- La clase más importante es java.lang.Class. Esta clase es un descriptor o referencia de una clase y permite describir cualquier clase Java. El paquete java.lang.reflect es relevante para la introspección

Introspección

Conocer la clase a la que pertenece un objeto

- **Es posible saber la clase a la que pertenece un objeto en tiempo de ejecución**

```
Persona p = new Persona();
Class claseObjeto = p.getClass();
System.out.print("La clase es: " + claseObjeto.getName());
```

- **El objeto claseObjeto pertenede a la clase Class, que es un descriptor de clase. Este descriptor devuelve el nombre de la clase con el método getName()**

Introspección

Conocer la clase a la que pertenece una clase

- **Si el descriptor de clase se aplica a una clase entonces:**

  Exception.class.getName()
  Scanner.class.getName()

- **El método getName() devuelve el nombre de la clase y la jerarquía de paquetes**

  java.lang.Exception
  java.util.Scanner

Utilidades

Ordenación con Comparator

- Para ordenar los elementos de una lista de objetos de diverso tipo es necesario definir una clase que implemente la interfaz Comparator

- Comparator es una interfaz que define un conjunto de métodos para comparar objetos. El método compare() de Comparator se utiliza para comparar y ordenar. La clase que implemente Comparator debe sobreescribir el método compare()

Utilidades

Ordenación con Comparator (Clase Persona)

```
public class Persona {
   private int id;
   private String nombre, apellidos;

   public Persona(int id) {
      this.id = id;
   }
   public Persona(int id, String nombre, String apellidos) {
      this.id = id;
      this.nombre = nombre;
      this.apellidos = apellidos;
   }
   public getId() {
      return id;
   }
}
```

Utilidades

Ordenación con Comparator (Clase ComparadorPersona)

```
import java.util.Comparator;

public class ComparadorPersona implements Comparator {

   public int compare(Object a, Object b) {
      if (((Persona)a).getId() < ((Persona)b).getId())
         return –1;

      if (((Persona)a).getId() > ((Persona)b).getId())
         return 1;

      return 0;
   }
}
```

Utilidades

Ordenación con Comparator (Programa principal)

```
import java.util.*;

public class Personas {

   private static void imprimirLista(ArrayList lista) {
      System.out.println("");

      for (int i = 0; i < lista.size(); i++) {
         Persona p = (Persona)lista.get(i);

         System.out.println("[" + p.getId() + "] " +
               p.getNombre() + " " + p.getApellidos());
      }
   }
```

Utilidades

Ordenación con Comparator (Programa principal y 2)

```
public static void main(String[] args) {
   List listaPersonas = new ArrayList();

   listaPersonas.add(new Persona(10, "Juan", "González"));
   listaPersonas.add(new Persona(30, "Luis", "López"));
   listaPersonas.add(new Persona(40, "María", "Soto"));
   listaPersonas.add(new Persona(50, "Pedro", "Sánchez"));
   listaPersonas.add(new Persona(60, "Ana", "Rodríguez"));

   imprimirLista((ArrayList)listaPersonas);

   listaPersonas.remove(2);

   listaPersonas.add(new Persona(20, "Mariana", "Rocha"));
   listaPersonas.add(new Persona(90, "Beatriz", "Sada"));
   listaPersonas.add(new Persona(70, "Mar", "García"));
```

Utilidades

Ordenación con Comparator (Programa principal y 3)

```
        imprimirLista((ArrayList)listaPersonas);

        /* el método sort de la clase Collections permite
         * ordenar la lista con un objeto ComparadorPersona
        /*

        Collections.sort(listaPersonas,new ComparadorPersona());

        imprimirLista((ArrayList)listaPersonas);

      } // fin de static void main

} // fin de la clase Personas
```

Utilidades

Números aleatorios

- La clase Random de java.util.Random genera números aleatorios

```
Random r = new Random();

for (int i=1; i<=10; i++ )
    System.out.println( r.nextDouble() );
```

- La clase Random ofrece double nextGaussian(), que da números aleatorios con distribución gaussiana centrada en cero y con una desviación estándar en torno a 1. nextInt(n) da números entre 0 y n-1

Utilidades

Funciones matemáticas

- La clase Math ofrece funciones matemáticas como:

Valor absoluto	abs(x)
Potencia	pow(x, y) para elevar x^y
Raíz cuadrada	sqrt(x) para la raíz cuadrada
Comparación	min(X, y), max(x, y)
Trigonométricas	sin(x), cos(x), tan(x)
	asin(x), acos(x), atan(x)
Exponencial	exp(x)
Logarítmica	log(x)
e	E
pi	PI

Utilidades

Tareas temporizadas

- Una tarea temporizada se repite de forma periódica. La clase abstracta TimerTask tiene un método abstracto run() que permite ejecutar una tarea utilizando un temporizador

- El temporizador es un objeto de la clase Timer que controla el tiempo y ejecuta la tarea temporizada invocando su método run(). La clase TareaTemporizada es una subclase de TimerTask

```
TareaTemporizada tarea = new TareaTemporizada();
Timer temporizador = new Timer();
temporizador.schedule(tarea, inicio, periodo);
```