

Programación Orientada a Objetos

Blanca Calderón

Declarar e implementar una clase

- **Objetivo:**
 - Ser capaz de **declarar una clase** con un conjunto de características (**atributos**) y comportamientos (**métodos**)
 - Ser capaz de **crear objetos** de una clase dada y modificar o restringir el acceso a su estado y su comportamiento
- **Plan de trabajo:**
 - Memorizar la **nomenclatura** básica de la programación orientada a objetos
 - Practicar el **modelado** de objetos con ejemplos sencillos para distinguir entre una clase, un objeto, su estado y su comportamiento
 - Repasar la **sintaxis** java para declarar **clases, atributos, constructores y métodos**
 - Recordar el mecanismo y la sintaxis para **paso de mensajes** entre objetos

Objetivos

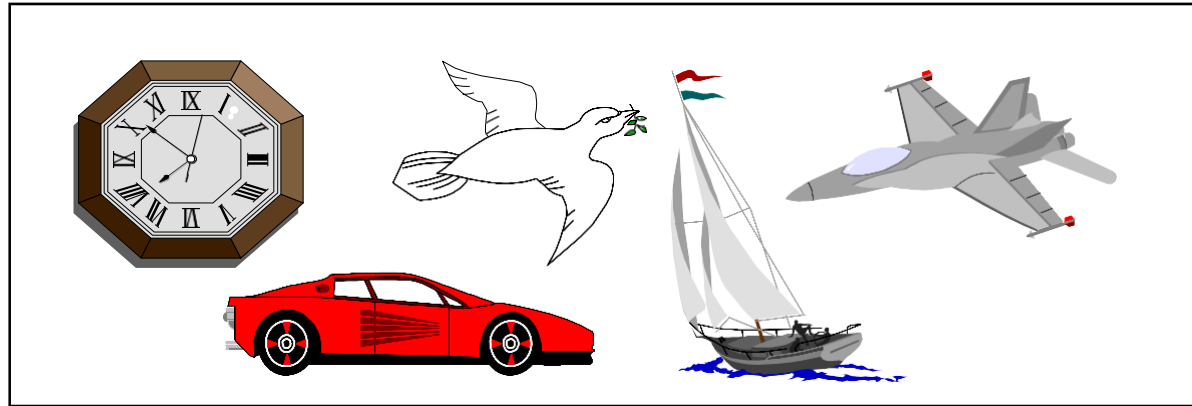


- Definir los **conceptos básicos** de la programación **basada** en objetos
 - Clases, objetos
 - Miembros (atributos, métodos)
 - Abstracción y ocultación de información
- Describir **relación** entre objeto y clase
- **Crear** un objeto sencillo y **modelar**
 - sus características (por medio de atributos)
 - su comportamiento (por medio de métodos)



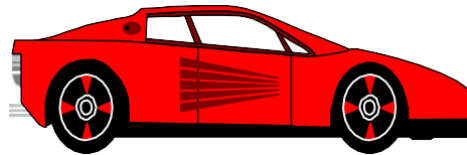
- ❖ Clases y objetos
- ❖ Encapsulación de objetos
 - ❖ Abstracción funcional
 - ❖ Abstracción de datos
- ❖ Miembros de una clase (atributos y métodos)
- ❖ Paso de mensajes
- ❖ Sobrecarga de métodos
- ❖ Constructores
- ❖ Modificadores y acceso

¿Qué es un objeto?



- Los **objetos** son representaciones (simples/complejas) (reales/imaginarias) de cosas: *reloj, avión, coche*
- No todo puede ser considerado como un objeto, algunas cosas son simplemente características o **atributos** de los objetos: *color, velocidad, nombre*

¿Qué es un objeto?



- **Abstracción funcional**

- Hay cosas que sabemos que los coches hacen pero no cómo lo hacen:

- avanzar
 - parar
 - girar a la derecha
 - girar a la izquierda

- **Abstracción de datos**

- Un coche tiene además ciertos atributos:

- color
 - velocidad
 - tamaño
 - etc.

¿Qué es un objeto?



- Es una forma de agrupar un conjunto de datos (**estado**) y de funcionalidad (**comportamiento**) en un mismo bloque de código que luego puede ser referenciado desde otras partes de un programa
- La **clase** a la que pertenece el objeto puede considerarse como un nuevo **tipo de datos**

Ejemplo



Clase

Objetos

```
public class Coche {  
    private String color;  
    private int velocidad;  
    private float tamaño;  
  
    public Coche (String color, int velocidad, float tamaño){  
        this.color = color;  
        this.velocidad = velocidad;  
        this.tamaño = tamaño;  
    }  
  
    public void avanzar(){}  
    public void parar(){}  
    public void girarIzquierda(){}  
    public void girarDerecha(){}  
}
```

Estado

Constructor

Comportamiento

```
public static void main (String[] args){  
    Coche miCoche = new Coche ("verde", 80, 3.2f);  
    Coche tuCoche = new Coche ("rojo", 120, 4.1f);  
    Coche suCoche = new Coche ("amarillo", 100, 3.4f);  
}
```



- **this** referencia al objeto de la clase actual

Ejercicio 1



- Implementa la clase **Bicicleta**, que tiene tres atributos, **velocidadActual**, **platoActual** y **piñonActual**, de tipo entero y cuatro métodos **acelerar()**, **frenar()**, **cambiarPlato(int plato)**, y **cambiarPiñon(int piñon)**, donde el primero dobla la velocidad actual, el segundo reduce a la mitad la velocidad actual, y el tercero y cuarto ajustan el plato y el piñón actual respectivamente según los parámetros recibidos. La clase debe tener además un constructor que inicialice todos los atributos.
- Crea dos objetos de la clase bicicleta: **miBicicleta** y **tuBicicleta**

Encapsulación de objetos



- **Encapsulación:** describe la vinculación de un comportamiento y un estado a un objeto en particular.
- **Ocultación de información:** Permite definir qué partes del objeto son visibles (el interfaz público) que partes son ocultas (privadas)

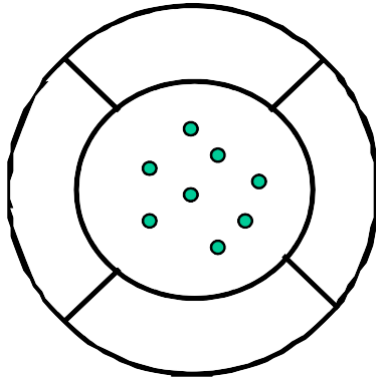


- La llave de contacto es un interfaz público del mecanismo de arranque de un coche
- La implementación de cómo arranca realmente es privada y sobre ella sólo puede actuar la llave de contacto

ventajas

El objeto puede cambiar y su interfaz pública ser compatible con el original: esto facilita reutilización de código

Encapsulación de objetos



MIEMBROS DE UNA CLASE

Los objetos encapsulan atributos permitiendo acceso a ellos únicamente a través de los métodos

- ▶ **Atributos (Variables):** Contenedores de valores
- ▶ **Métodos:** Contenedores de funciones

Un objeto tiene

- ▶ **Estado:** representado por el contenido de sus atributos
- ▶ **Comportamiento:** definido por sus métodos



Normalmente:

- ▶ Los métodos son públicos
- ▶ Los atributos son privados
- ▶ Puede haber métodos privados
- ▶ Es peligroso tener atributos públicos

Definición de objetos



Miembros públicos

- los miembros públicos describen **qué** pueden hacer los objetos de esa clase
 - Qué pueden hacer los objetos (métodos)
 - Qué son los objetos (su abstracción)

Miembros privados

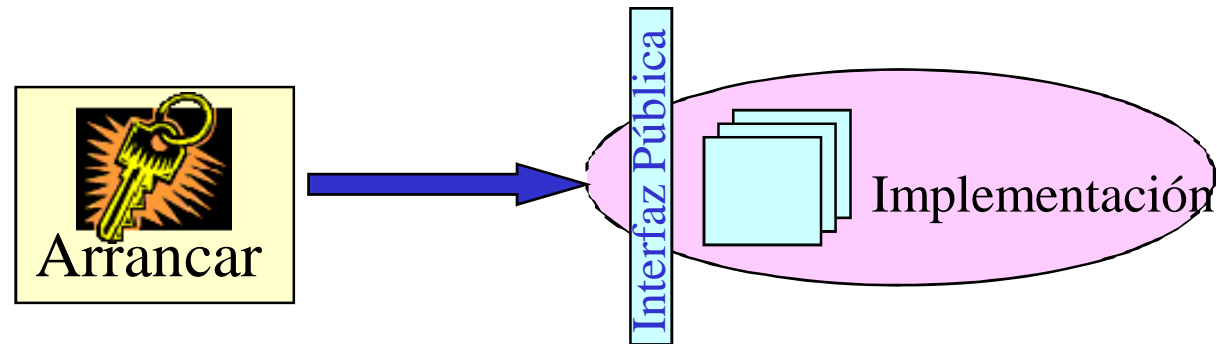
- describen la implementación de **cómo** lo hacen.
 - Ejemplo: el objeto contacto interacciona con el circuito eléctrico del vehículo, este con el motor, etc.
 - ***En sistemas orientados a objetos puros todo el estado es privado y sólo se puede cambiar a través del interfaz público.***
 - Ej: El método público frenar puede cambiar el valor del atributo privado velocidad.

Interacciones entre objetos



- El ***modelado de objetos*** modela:
 - Los objetos y
 - Sus interrelaciones
- Para realizar su tarea el objeto puede ***delegar*** trabajos en otro que puede ser parte de él mismo o de cualquier otro objeto del sistema.
- Los objetos interaccionan entre sí enviándose ***mensajes***

Paso de Mensajes



- Un objeto envía un *mensaje* a otro
 - Esto lo hace mediante una **llamada** a sus atributos o métodos
- Los mensajes son tratados por la **interfaz pública** del objeto que los recibe
 - Eso quiere decir que sólo podemos hacer llamadas a aquellos atributos o métodos de otro objeto que sean **públicos o accesibles** desde el objeto que hace la llamada
- El objeto receptor reaccionará
 - **Cambiando su estado:** es decir modificando sus atributos
 - **Enviando otros mensajes:** es decir llamando a otros atributos o métodos del mismo objeto (públicos o privados) o de otros objetos (públicos o accesibles desde ese objeto)

Ejemplo



Clase Coche

```
public class Coche {  
  
    private String color;  
    private int velocidad;  
    private float tamaño;  
    private Rueda[] ruedas;  
    private Motor motor;  
  
    public Coche (String color, int velocidad,  
                  float tamaño, Rueda[] ruedas,  
                  Motor motor){  
        this.color = color;  
        this.velocidad = velocidad;  
        this.tamaño = tamaño;  
        this.ruedas = ruedas;  
        this.motor = motor;  
    }  
  
    public void avanzar(){  
        motor.inyectarCarburante();  
        for (int i=0; i < ruedas.length; i++){  
            ruedas[i].girar();  
        }  
    }  
  
    public static void main (String[] args){  
  
        Rueda[] ruedas = {new Rueda(20,"Dunlop"),  
                           new Rueda(20,"Dunlop"),  
                           new Rueda(22, "Dunlop"),  
                           new Rueda(22, "Dunlop")};  
        Coche miCoche = new Coche ("verde", 80,3.2f,  
                                     ruedas, new Motor("Diesel",100));  
    }  
}
```

} Paso de
mensajes

Clase Motor

```
public class Motor {  
  
    private String tipo;  
    private int caballos;  
  
    public Motor(String tipo, int caballos){  
        this.tipo = tipo;  
        this.caballos = caballos;  
    }  
  
    public void inyectarCarburante(){...}  
}
```

Clase Rueda

```
public class Rueda {  
  
    private double diametro;  
    private String fabricante;  
  
    public Rueda (double diametro, String fabricante){  
        this.diametro = diametro;  
        this.fabricante = fabricante;  
    }  
  
    public void girar(){...}  
}
```

Clasificación de objetos



- **Clase:** Conjunto de objetos con estados y comportamientos similares
 - Podemos referirnos a la clase “Coche” (cualquier instancia de la clasificación coche)
- “Mi coche” es un **objeto**, es decir una **instancia** particular de la clase coche
- La clasificación depende del problema a resolver

Objetos vs. Clases

Una **clase** es una entidad abstracta

- Es un tipo de clasificación de datos
- Define el comportamiento y atributos de un grupo de estructura y comportamiento similar

Clase Coche

Métodos: arrancar, avanzar, parar, ...

Atributos: color, velocidad, etc.

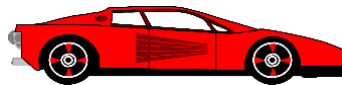
—————→ Nombre de la clase
—————→ Métodos (funciones)
—————→ Atributos (datos)

Un **objeto** es una instancia de una clase

- Un objeto se distingue de otros miembros de la clase por sus atributos

Objeto Ferrari

Perteneciente a la
clase coche



Nombre: Ferrari
Métodos: arrancar, avanzar, parar, ...
Atributos: color = "rojo"; velocidad 300Km/h

- Una **clase** se declara, un **objeto** además se crea

Sobrecarga (Overloading)

¿Qué es?



- Podemos definir una clase con dos métodos con el mismo nombre si los argumentos son distintos.
- Se utiliza mucho para los constructores.
- Sabemos cuál de los dos métodos tenemos que ejecutar por los parámetros que le pasamos cuando le llamamos.

Sobrecarga (Overloading)

¿Para qué sirve?



Clase

Objetos

```
public class Coche {  
    private String color;  
    private int velocidad;  
    private float tamaño;  
  
    public Coche (String color, int velocidad, float tamaño){  
        this.color = color;  
        this.velocidad = velocidad;  
        this.tamaño = tamaño;  
    }  
  
    public void avanzar(){}  
    public void avanzar(int metros){}  
    public void avanzar(int metros, int velocidad){}  
  
    public void parar(){}  
    public void girarIzquierda(){}  
    public void girarDerecha(){}  
}
```

} Sobrecarga

```
public static void main (String[] args){  
    Coche miCoche = new Coche ("verde", 80, 3.2f);  
    Coche tuCoche = new Coche ("rojo", 120, 4.1f);  
    Coche suCoche = new Coche ("amarillo", 100, 3.4f);  
  
    miCoche.avanzar();  
    tuCoche.avanzar(1000);  
    suCoche.avanzar(1000,120);  
}
```

Son métodos distintos porque, aunque tengan el mismo nombre tienen distintos argumentos. Tienen distinta funcionalidad

Ejercicio 2



- Sobre la clase **Bicicleta**, implementa los método sobrecargados **cambiarPlato()**, y **cambiarPiñon()**, que no reciben argumentos y que cambian el plato actual y el piñón actual a un valor por defecto, en concreto, 1.

Constructores



- Cuando se crea un objeto sus miembros se **inicializan** con un método constructor
- Los constructores:
 - llevan el **mismo nombre** que la clase
 - **No** tienen **tipo** de resultado (ni siquiera void)
- Conviene que haya al menos 1 constructor
- Pueden existir varios que se distinguirán por los parámetros que aceptan (**sobrecarga**)
- Si no existen se crea un **constructor por defecto** sin parámetros que inicializa las variables a su valor por defecto.
- Si la clase tiene algún constructor, el constructor por defecto deja de existir. En ese caso, si queremos que haya un constructor sin parámetros tendremos que declararlo explícitamente.

Constructores



```
public class Coche {  
  
    private String color;  
    private int velocidad;  
    private float tamaño;  
  
    public Coche (){}  
  
    public Coche (String color){  
        this.color = color;  
    }  
  
    public Coche (String color, int velocidad){  
        this.color = color;  
        this.velocidad = velocidad;  
    }  
  
    public Coche (String color, int velocidad, float tamaño){  
        this.color = color;  
        this.velocidad = velocidad;  
        this.tamaño = tamaño;  
    }  
  
    public void avanzar(){}  
    public void parar(){}  
    public void girarIzquierda(){}  
    public void girarDerecha(){}  
}
```

Array de objetos de la clase Coche

```
public static void main (String[] args){  
  
    Coche[] concesionario = {new Coche("verde"), new Coche("rojo",120),  
                             new Coche("amarillo",100,3.2f)};  
  
}
```

Sobrecarga de constructores

Ejercicio 3



- Sobre la clase **Bicicleta**, implementa un constructor adicional que no recibe parámetros y que inicializa la velocidad actual a 0, y el plato actual y el piñón actual a 1.

Modificadores y acceso

Static (miembros estáticos)



- Modificador `static`
- Sólo existen `una vez por clase`, independientemente del número de instancias (objetos) de la clase que hayamos creado y aunque no exista ninguna.
- El método o el atributo **se comportan siempre de la misma manera**
- Se puede acceder a los miembros estáticos utilizando el `nombre de la clase`.
- Un método estático `no` puede acceder a miembros no estáticos directamente, tiene que crear primero un objeto

Modificadores y acceso

Static (miembros estáticos)



Atributo estático

```
public class Coche {  
  
    private String color;  
    private int velocidad;  
    private float tamaño;  
    private static int numeroRuedas = 4; }  
  
...  
    public static void main (String[] args){  
        System.out.println(Coche.numeroRuedas);  
    }  
}
```

Atributo estático

Otros ejemplos

```
int radium = 3;  
double areaCircle = Math.PI * radium * radium;  
  
int minValue = Integer.MIN_VALUE; => -231  
int maxValue = Integer.MAX_VALUE; => 231-1
```

Método estático: Tiene acceso a atributos estáticos
No necesitamos crear instancias

```
public static void main(String args[]) {  
  
    int x1 = Integer.parseInt(args[0]);  
    double y1 = Double.parseDouble(args[1]);  
  
}
```

```
Math.sqrt(100);  
Math.cos(76);
```

<http://docs.oracle.com/javase/7/docs/api/java/lang/Math.html>

Modificadores y acceso

Static. Algunas reglas



- Los miembros **estáticos** se invocan con:

```
NombreClase.metodoEstatico();  
NombreClase.atributoEstatico;
```

- Para acceder a los miembros no estáticos necesitamos disponer de una instancia (objeto) de la clase

```
NombreClase nombreObjeto = new NombreClase();
```

- Los miembros **no** estáticos se invocan con:

```
nombreObjeto.metodoNormal();  
nombreObjeto.atributoNormal;
```

- Cuando invocación (llamada) a un miembro estático de la clase se realiza dentro de la propia clase se puede omitir el nombre de la misma. Es decir podemos escribir:

```
metodoEstatico();  
atributoEstatico;
```

en lugar
de:

```
NombreClase.metodoEstatico();  
NombreClase.atributoEstatico;
```

Acceso

Métodos get() y set()




- Los atributos de una clase son generalmente privados para evitar que puedan ser accesibles / modificables desde **cualquier otra clase**.
- A veces nos interesa que algunas clases determinadas sí puedan acceder a los atributos.
- Uso de métodos **get()** y **set()**

```
public class Coche {  
  
    private String color;  
    private int velocidad;  
  
    public void setColor(String color){  
        this.color = color;  
    }  
  
    public String getColor(){  
        return this.color;  
    }  
  
    public void setVelocidad(int velocidad){  
        this.velocidad = velocidad;  
    }  
  
    public int getVelocidad(){  
        return this.velocidad;  
    }  
  
}
```

Ejercicio 4



- Sobre la clase **Bicicleta**, implementa los métodos **get()** y **set()** necesarios para poder acceder y modificar todos los atributos.

MODIFICADORES		clase	metodo	atributo
acceso	public	Accesible desde cualquier otra clase		
	(friendly)	Accesible sólo desde clases de su propio paquete		
	private		Accesibles sólo dentro de la clase	
	static	Clase de nivel máximo. Se aplica a clases internas	Es el mismo para todos los objetos de la clase. Se utiliza: NombreClase.metodo();	Es la misma para todos los objetos de la clase. Se utiliza: NombreClase.atributo;

Paquetes



- Un *paquete* agrupa *clases* (e *interfaces*)
- Las jerarquías de un paquete se corresponden con las jerarquías de directorios
- Para referirse a miembros y clases de un paquete se utiliza la notación de separarlos por puntos.
 - Ej: Cuando importamos paquetes de clases matemáticas

```
import java.math.BigDecimal;
```

- La clase `BigDecimal` está en el directorio `java/math` dentro del JDK
- No es necesario importar todas las clases: `package java/lang`
 - `String`
 - `Integer`
 - `NullPointerException`
 - `ArrayIndexOutOfBoundsException`

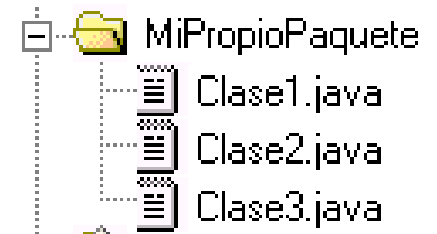
Paquetes



- ***¿Cómo crear mis propios paquetes?***

- Almaceno mis clases en un directorio con el nombre del paquete
- Pongo al principio de todas las clases que pertenezcan al paquete la instrucción

```
package MiPropioPaquete;
```



- Si quiero importar las clases de ese paquete desde otras clases y/o proyectos, pongo al principio de cada clase

```
import MiPropioPaquete.Clase1
```