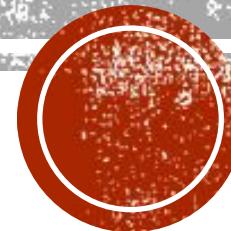


# **PROGRAMACIÓN**

## **UNIDAD 5: Programación orientada a objetos**



**Curso / Ciclo Formativo: 1º Desarrollo de aplicaciones web**

**Profesora: María Navarro Elbal**

**Mari Cruz Sanz Zamarrón**

# Introducción al concepto de objeto.

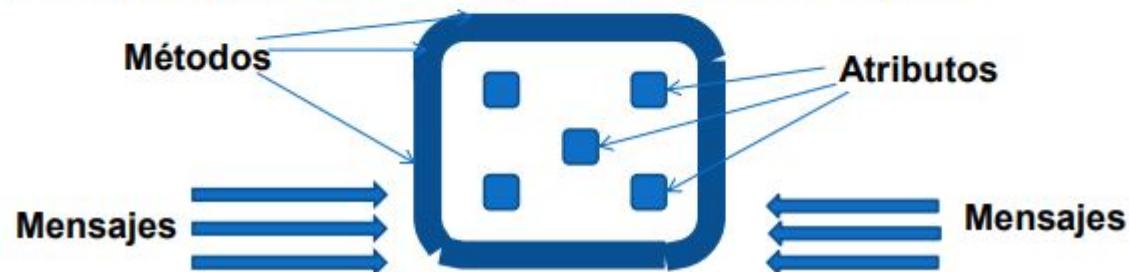
- **Objetos** pertenecen a clases.
- **Clases** son los “moldes” a partir de los cuales se generan objetos
- Los objetos se instancian y se generan.
- Objeto ≈ instancia.
- Escribir un programa → definir clases (estado y comportamiento).
- Ejecutar un programa → crear objetos.
- Clase → un archivo con el mismo nombre de la **clase.java**.
- Estructura de la clase:

```
[modificadores] class nombre_clase {  
    [Atributos]  
    [Métodos]  
}
```



# Introducción al concepto de objeto.

- Características del objeto:
  - **Identidad**
    - Único y diferente de otros objetos.
  - **Estado.**
    - Es el valor de los atributos del objeto.
  - **Comportamiento.**
    - Métodos o procedimientos que realiza dicho objeto.



# Introducción al concepto de objeto.

- **Mensajes.**
  - Comunicación de los objetos a través de mensajes.
  - Mensaje → ejecución de un método.
- **Métodos.**
  - Procedimientos que ejecuta el objeto cuando recibe un mensaje vinculado a ese método.
  - Puede enviar mensajes a otros objetos, solicitando información o acciones.



# Introducción al concepto de objeto.

- **Objeto.**
  - Dos características importantes:
    - Estado.
    - Comportamiento
  - Diferente complejidad de los objetos.
  - Beneficios de la POO.
    - **Modularidad.** El código de un objeto puede mantenerse y reescribirse sin que ello implique la reprogramación del código de otros objetos de la aplicación
    - **Reutilización de código.** Utilización de clases y objetos, sin conocer los detalles de la implementación (solo su interfaz).
    - **Facilidad de testeo y reprogramación.** Si un objeto da problemas en una aplicación => reemplazar el objeto por otro similar o reprogramarlo
    - **Ocultación de información.** Se ocultan los detalles de implementación



# Introducción al concepto de objeto.

- **Abstracción.**

- **RAE:** “abstraer es separar por medio de una operación intelectual las cualidades de un objeto para considerarlas aisladamente o para considerar el mismo objeto en su pura esencia o noción.”
- **POO:** abstraer las características de los objetos que van a tomar parte del programa y crear clases con sus atributos y sus métodos.

- **Encapsulamiento.**

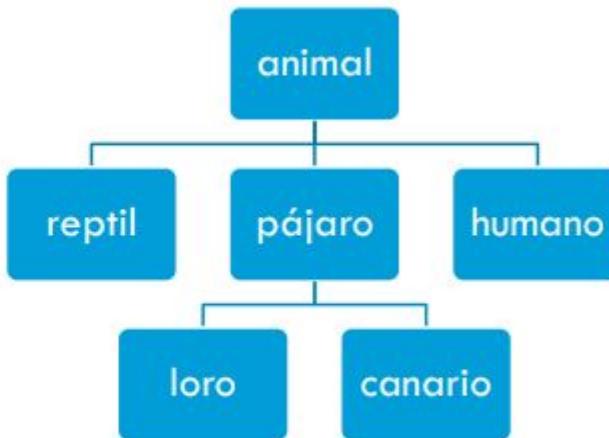
- Ver los objetos según su comportamiento externo, sin conocer los detalles de la implementación
- Clases = caja negra.



# Introducción al concepto de objeto.

- **Herencia.**

- Estructura **jerárquica** de clases.
- Las clases pueden tener subclases y superclases.
- Java **no** permite la herencia múltiple, solo la simple.
- Java puede implementar la herencia múltiple a través de **interfaces**.
- Si una clase hereda de su superclase, obtiene los métodos y las propiedades de la superclase



# Introducción al concepto de objeto.

- **Polimorfismo.**

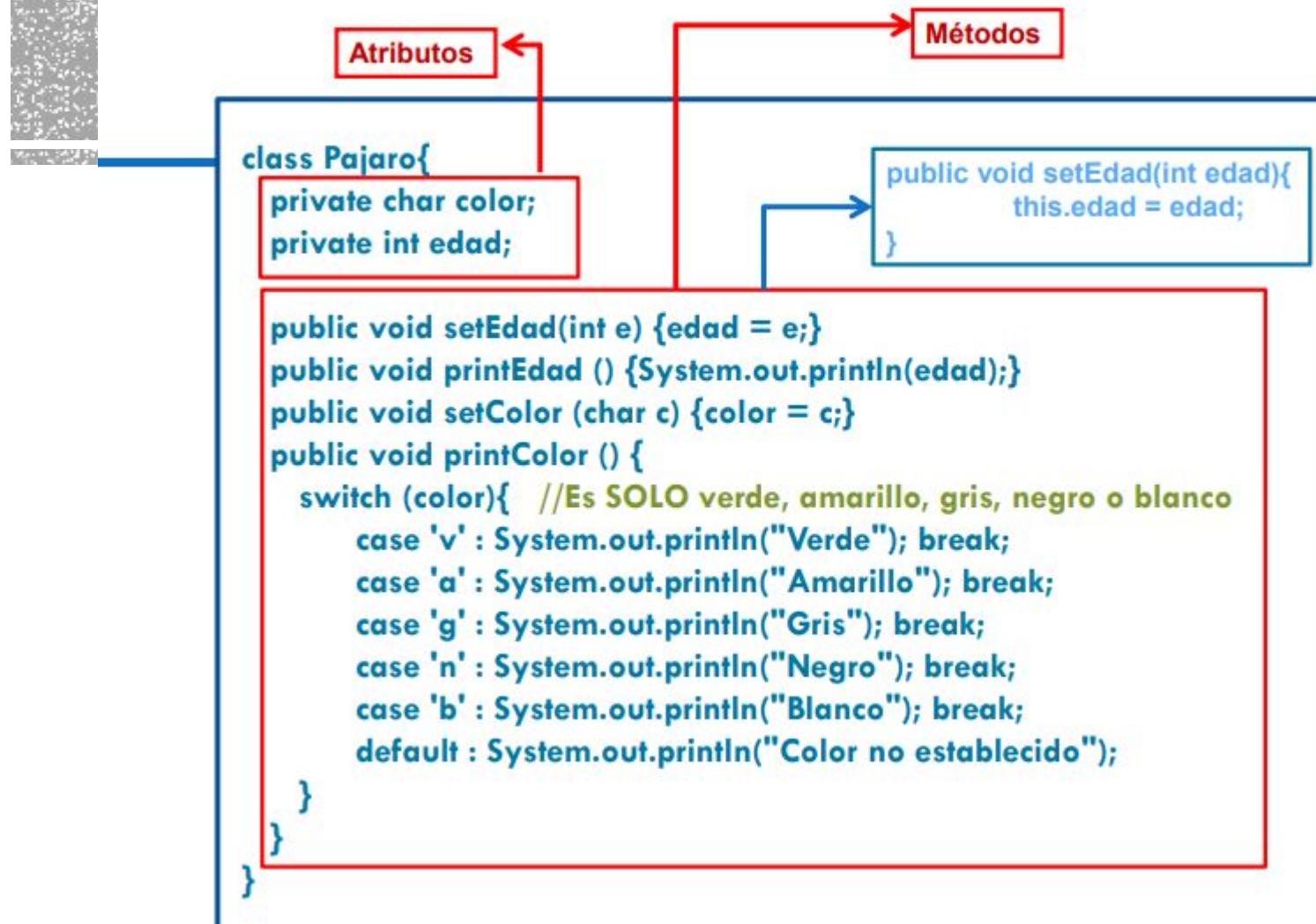
- Permite crear varias formas del mismo método → un mismo método ofrece comportamientos diferentes.



# Introducción al concepto de objeto.

- Datos = propiedades = atributos = variables.
- Métodos = funciones.
- **Atributos:**
  - Básicos.
  - Objeto de otra clase
- **this.**
  - Se usa para referirse a los atributos de una misma clase en los métodos de la propia clase.
  - Cuando hay ambigüedad entre nombres de parámetros de un método y atributos del objeto.
  - Otros usos, más adelante.





# Propiedades y métodos y objetos

```
class Test{
    public static void main(String [] args){
        Pajaro p;
        p = new Pajaro();
        p.setEdad(5);
        p.printEdad();
    }
}
```



# Programación de la consola: entrada y salida de información

- Clase **System**:
  - Entrada a través de teclado
  - Salida a través de pantalla
  - Tiene tres objetos asociados a tres flujos estándar: se abren cuando se ejecuta el programa y se cierran cuando finaliza. Son objetos **static**.
    - **System.out**: referencia a la salida estándar (pantalla)
    - **System.in**: referencia a la entrada estándar (teclado). El objeto **System.in** pertenece a la clase *InputStream*.
    - **System.err**: referencia a la salida de error estándar (pantalla). Usado para mostrar mensajes de error.



# Programación de la consola: entrada y salida de información

**Ejemplo de System.in**

```
char c;
try{
    c = (char)System.in.read(); //Lectura de un carácter por teclado
}catch (Exception e){
    e.printStackTrace(); //Indica el método donde se lanzó la excepción
}

//Para leer una línea completa desde teclado
InputStreamReader isr = new InputStreamReader(System.in);
BufferedReader buff = new BufferedReader (isr);
String ln = buff.readLine();

//Otra forma de leer datos desde teclado
String sdato = System.console().readLine(); //Lectura de un String
dato = Integer.parseInt(sdato); //Transformación del String en entero
```



# Programación de la consola: entrada y salida de información

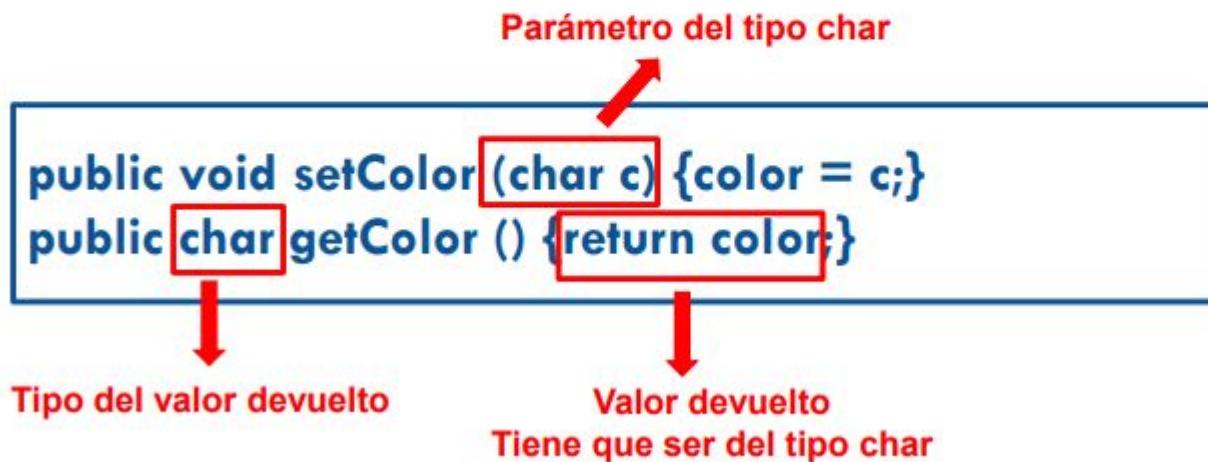
- Ejemplo de `System.in.err()`

```
int a = 10, b = 0, c;
try{
    c = a / b;
} catch (ArithmetricException e) {
    System.err.println("Error: " + e.getMessage());
    return;
}
System.out.println("Resultado: " + c);
```



# Parámetros y valores devueltos

- **Parámetros.** Valores que se especifican cuando se llama a un método.
- Pueden tener un tipo básico o ser un objeto.
- Pueden devolver un valor o **void** (excepto los constructores)



# Constructores y destructores de objetos

- **Constructor.**

- Procedimiento llamado automáticamente cuando se crea un objeto de esa clase.
- Si no se especifica, Java genera uno por defecto.
- Inicializa el objeto.
- Se llama igual que la clase a la que pertenece.

- **Destructor.**

- Se ejecuta automáticamente siempre que se destruye un objeto de dicha clase.
- Liberan recursos y cierran flujos abiertos.
- No reciben parámetros.
- No está permitida la sobrecarga.
- No hay destructores en Java como en C++.



# Constructores y destructores de objetos

```
class Pajaro{  
    private char color;  
    private int edad;  
  
    Pajaro() {color = 'v'; edad = 0;} //Constructor de la clase pájaro.  
    Pajaro(char c, int e) {color = c; edad = e}; //Constructor de la clase pájaro  
    /*Resto de los métodos de la clase*/  
  
    public static void main(String [] args){  
        Pajaro p1, p2;  
        p1 = new Pajaro();  
        p2 = new Pajaro('a', 3);  
    }  
}
```

**Constructor sobrecargado.**  
Permite crear objetos de la clase pájaro de distintas formas.



# Uso de métodos estáticos y dinámicos

- **static.**

- Método o atributo que se va a crear para la clase a la que pertenece solo una instancia de ese método o atributo.
- Atributo **numPajaros** que cuenta los pájaros que se van creando.



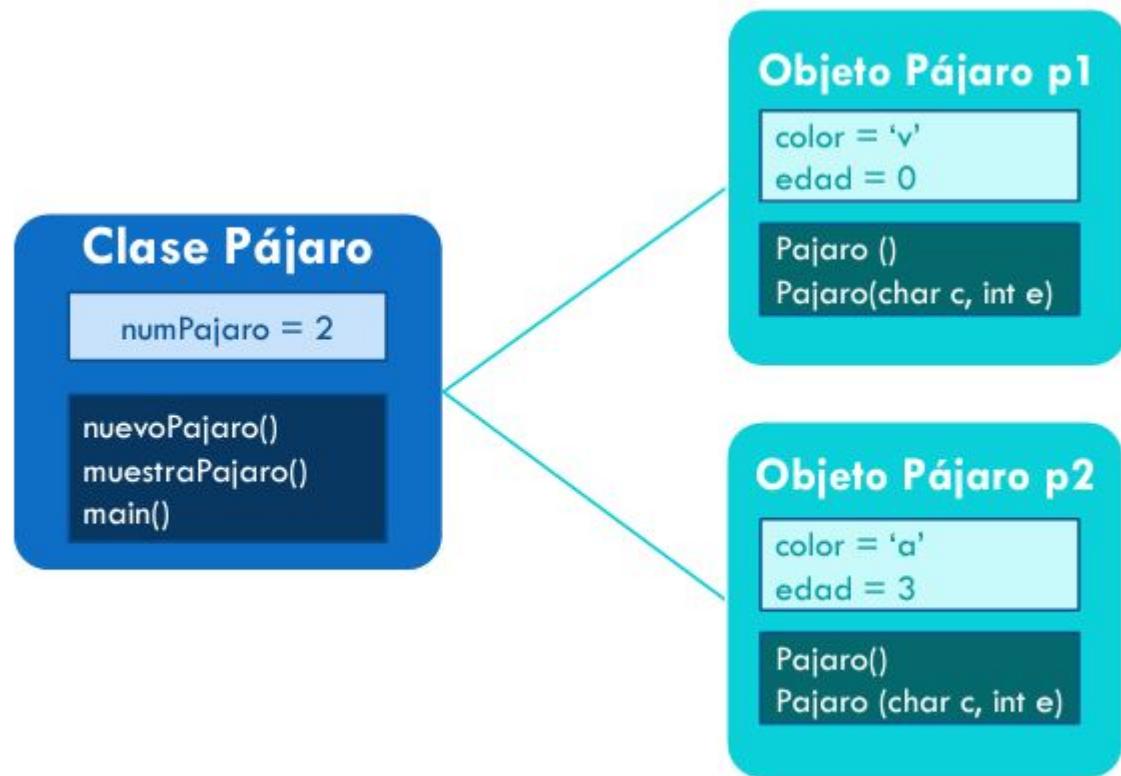
# Uso de métodos estáticos y dinámicos

- **static.**

- Método o atributo que se va a crear para la clase a la que pertenece solo una instancia de ese método o atributo.
- Atributo **numPajaros** que cuenta los pájaros que se van creando.



# Uso de métodos estáticos y dinámicos



# Uso de métodos estáticos y dinámicos

```
class Pajaro{  
    private static int numPajaros = 0;  
    private char color;  
    private int edad;  
  
    static void nuevoPajaro() {numPajaros++;}  
    Pajaro() {color = 'v'; edad = 0; nuevoPajaro();}  
    Pajaro(char c, int e) {color = c; edad = e; nuevoPajaro();}  
    static void muestraPajaros() {System.out.println(numPajaros);}  
  
    public static void main(String [] args){  
        Pajaro p1, p2;  
        p1 = new Pajaro();  
        p2 = new Pajaro('a', 3);  
        p1.muestraPajaros();  
        p2.muestraPajaros();  
    }  
}
```



# Ejercicios

1. Realiza una clase Temperatura que convierta grados Celsius a Fahrenheit y viceversa. Para ello crea dos métodos :
  - **double celsiusToFarenheit(double )**
  - **double farenheitToCelsius(double)**
  - En la construcción ten en cuenta las siguientes fórmulas:
    - $C = (F - 32)/1,8$
    - $F = (1,8) C + 32$



# Ejercicios

2. Tenemos la siguiente clase Coche:

```
class Coche{  
    private int velocidad;  
    Coche() {velocidad = 0;}  
}
```

Añade a la clase Coche los siguientes métodos:

**int getVelocidad()**. Devuelve la velocidad actual.

**void acelera (int mas)**. Actualiza la velocidad a *mas* km más.

**void frena (int menos)**. Actualiza la velocidad a *menos* km menos.

# Cabecera de la clase

**modificadores + nombre\_clase + modificadores**

- Delante de **class**:
  - **public**:
    - Acceso a la clase de tipo público.
    - Cualquier clase puede hacer uso de ella
    - Si es pública => nombre\_clase = nombre\_archivo
    - Si no pública => acceso únicamente las clases de su mismo paquete.
    - 1 archivo => 1 clase pública
  - **final**:
    - Finaliza con ella la cadena de herencia (sin subclases).
  - **abstract**:
    - Clases que declaran un método sin implementar (se implementan en sus subclases).
- Una clase abstracta, ¿puede ser final?

# Cabecera de la clase

- Detrás de **class**
  - **extends**
    - Indica que la clase hereda de otra superclase.
  - **implements**
    - Implementa los métodos de una o varios interfaces.

```
[public][final | abstract] class <nombre_clase> [extends superclase]  
[implements <interface_1>[, <interface_2>] [, <interface_3>]...]
```



# Librerías de objetos (Paquetes)

- Paquete o **package**.
  - Conjunto de clases relacionadas entre sí ordenadas de forma arbitraria.
  - Las clases que forman parte de un paquete, no derivan todas de una misma superclase.
  - Un paquete puede contener otros paquetes.
  - Se evita que dos clases tengan el mismo nombre (estarán en dos paquetes distintos).
  - Privilegios a los miembros de otras clases del mismo paquete.
  - Permiten organizar las clases en grupos.



Paquete o librería	Descripción
java.io	Librería de E/S. permite la comunicación del programa con ficheros y periféricos.
java.lang	Paquete con clases esenciales de Java. No hace falta ejecutar la sentencia import para utilizar clases. Librería por defecto.
java.util	Librería con clases de utilidad general para el programador.
java.applet	Librería para desarrollar applets.
java.awt	Librerías con componentes para el desarrollo de interfaces de usuario.
java.swing	Librerías con componentes para el desarrollo de interfaces de usuario. Similar al paquete awt.
java.net	En combinación con la librería java.io, va a permitir crear aplicaciones que realicen comunicaciones en la red local.
java.math	Librería con todo tipo de utilidades matemáticas.
java.sql	Librería especializada en el manejo y comunicación con bases de datos.
java.security	Librería que implementa mecanismos de seguridad.
java.rmi	Paquete que permite el acceso a objetos situados en otros equipos (objetos remotos)
java.beans	Librería que permite la creación y manejo de componentes javabeans.

# Ejercicios

4. ¿Está correctamente definida la siguiente clase? ¿Compilará o habrá que modificarla para poder generar el fichero .class?

```
class Pajaro{  
    public void setEdad (int e) {edad = e;}  
    public void printEdad() {System.out.println(edad);}  
    public void setColor(char c) {color = c;}  
    private char color;  
    private int edad;  
}
```



# Ejercicios

5. El siguiente código tiene algunos problemas de compilación

```
public class Satelite {  
    private double meridiano;  
    private double paralelo;  
    private double distancia_tierra;  
    satelite (double m, double p, double d){  
        meridiano = m;  
        paralelo = p;  
        distancia_tierra = d;  
    }  
    satelite (){  
        meridiano = paralelo = distancia_tierra = 0;  
    }  
    public void setPosicion (double m, double p, double d){  
        meridiano = m;  
        paralelo = p;  
        distacnia_tierra = d;  
    }  
    public void printPosicion (){  
        System.out.println("El satélite se encuentra en el paralelo " + "paralelo" + "Meridiano " +  
        "meridiano" + " a una distancia de la tierra de " + "distancia_tierra" + " kilómetros");  
    }  
}
```

# Ejercicios

6. Crea una clase **Rebajas** con un método **descubrePorcentaje()** que descubra el descuento aplicado en un producto. El método recibe el precio original del producto y el rebajado y halla el porcentaje.



# Ejercicios

7. Realiza una clase número que almacene un número entero y tenga las siguientes características:

**Constructor** por defecto que inicializa a 0 el número entero.

**Constructor** que inicializa el número interno.

Método **añade** que permite sumarle un número al valor interno.

Método **resta** que resta un número al valor interno.

Método **getValor**.

Método **getDoble** (el doble del valor interno)

Método **getTriple**.

Método **setNumero**.