

CFGS Desarrollo de aplicaciones web

# Módulo profesional: Programación



**GENERALITAT  
VALENCIANA**

Conselleria d'Educació,  
Investigació, Cultura i Esport



**Unió Europea**

Fons Social Europeu

*L'FSE inverteix en el teu futur*



# Material elaborado por:

Anna Sanchis

# Revisado y editado por:

Edu Torregrosa Llácer



# Datos profesor

Edu Torregrosa Llácer

eduardotorregrosa@ieslluissimarro.org

Web del módulo: <https://moodle.aulaenlanube.com/>

# POO en JAVA



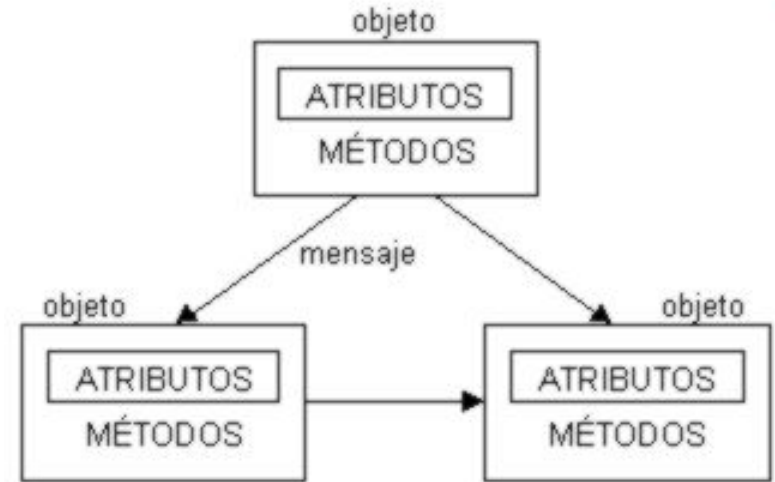
1. Introducción
  - 1.1. Elementos básicos de la POO
  - 1.2. Características de la POO
2. POO en JAVA
3. Métodos con clases
  - 3.1. Constructor
  - 3.2. Variable this
  - 3.3. Modificadores de acceso
4. Arrays de objetos

# Introducción

- Para empezar, todo parte del hecho de que el desarrollo de la programación de computadoras entró en crisis en los años 60 y 70 del s. XX .
- Englobó a una serie de sucesos que se venían observando en los proyectos de desarrollo de software:
  - Los proyectos no terminaban en plazo.
  - Los proyectos no se ajustaban al presupuesto inicial.
  - Baja calidad del software generado.
  - Software que no cumplía las especificaciones.
  - Código inmantenible que dificultaba la gestión y evolución del proyecto.

# Introducción

- La **programación orientada a objetos** gira entorno al concepto de **objeto**.
- Así un **objeto** es una entidad que tiene unos **atributos** particulares, los datos, y unas formas de operar sobre ellos, los **métodos** o procedimientos.
- Durante la ejecución, los objetos reciben y envían mensajes a otros objetos para realizar las acciones requeridas.



# Introducción

- Así que la **POO** es una manera de diseñar y **desarrollar** software que trata de **imitar** la **realidad** tomando algunos conceptos esenciales de ella;
- El principal concepto es el **objeto**, cuyas características son la identidad, el estado y el comportamiento.
  - La **identidad** es el nombre que distingue a un objeto de otro.
  - El **estado** son las características que lo describen.
  - El **comportamiento** es lo que puede hacer.

# Introducción

- Se debe tener presente que los **objetos**, se **abstraen** en **clases**.
- Por ejemplo:
  - De la clase perro pueden existir dos objetos Fido y Firuláis (esta es su identidad).
  - Fido es un san bernardo enorme, pinto, de 5 años de edad; mientras que Firuláis es un labrador, negro, de 3 años (este es su estado).
  - Ambos perros ladran, merodean, juegan, comen y duermen (este es su comportamiento).



# Introducción

- Si nos pidieran que hiciéramos un programa orientado a objetos que simulara lo anterior haríamos:
  - La **clase** Perro que tendría las **variables** raza, color y edad.
  - Los **métodos** ladrar(), merodear(), jugar(), comer() y dormir().
  - Firuláis y Fido son los **identificadores** que podríamos usar en una aplicación que pretenda mostrar dos objetos (instancias) de la clase Perro.

# Introducción

- Identificadores:

Son los nombres que pueden tener las clases, los métodos y las variables y no pueden contener espacios ni caracteres especiales. Estos nombres deben respetar ciertas convenciones según la siguiente tabla:

Tipo de identificador	Convención	Ejemplo
Clase	Comienza con mayúscula	HolaMundoOO
Método	Comienza con minúscula	mostrarSaludo ()
Variable	Comienza con minúscula	saludo

# Introducción

- Si nos pidieran que hiciéramos un programa orientado a objetos que simulara lo anterior haríamos:
  - La **clase** Perro que tendría las **variables** raza, color y edad.
  - Los **métodos** ladrar(), merodear(), jugar(), comer() y dormir().
  - Firuláis y Fido son los **identificadores** que podríamos usar en una aplicación que pretenda mostrar dos objetos (instancias) de la clase Perro.

# Elementos básicos de la P00

1. **Clases**
2. **Atributos**
3. **Métodos**
4. **Mensajes**
5. **Instanciación**

# Elementos básicos de la P00

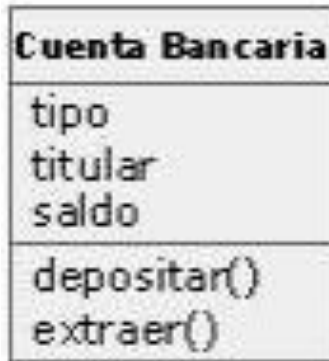
- **Clase:**

- Una clase es algo abstracto que define la "forma" del objeto, se podría hablar de la clase como el **molde de los objetos**.
- En el mundo real existen objetos del mismo tipo, por ejemplo tu bicicleta es solo una mas de todas las bicicletas del mundo. Entonces diríamos que tu bicicleta es una instancia de la clase Bicicleta.
- Todas las bicicletas tienen los **atributos**: color, cantidad de cambios, dueño y **métodos**: acelerar, frenar, pasar cambio, volver cambio.
- Las fábricas de bicicletas utilizan moldes para producir sus productos en serie, de la misma forma en POO utilizaremos la clase bicicleta (molde) para producir sus instancias (objetos).
- **Los objetos son instancias de clases.**

# Elementos básicos de la P00

## Clase (UML):

- Existe un lenguaje de modelado llamado UML mediante el cual podemos representar gráficamente todo un sistema orientado a objetos utilizando rectángulos, líneas y otro tipo de símbolos gráficos.
- Según UML, la clase "Cuenta Bancaria" se representará gráficamente como sigue:



# Elementos básicos de la P00

## Atributos

- Los atributos son las **características** individuales que diferencian un objeto de otro y determinan su apariencia, estado u otras cualidades.
- Los atributos se guardan en **variables** denominadas de **instancia**, y cada objeto particular puede tener valores distintos para estas variables.
- Las **variables de instancia**, son **declaradas** en la **clase** pero sus valores son fijados y cambiados en el objeto.

# Elementos básicos de la P00

## Métodos

- El **comportamiento** de los objetos de una clase se implementa mediante métodos.
- Un método es un conjunto de instrucciones que realizan una determinada **tarea** y son similares a las funciones de los lenguajes estructurados.



# Elementos básicos de la P00

## Mensajes

- Un objeto por si solo no tiene mucho significado. Ejemplo: el objeto "bicicleta" no tiene mucho sentido si no interactúa con un objeto "persona" que pedalee.
- La interacción entre objetos se produce mediante mensajes. **Los mensajes son llamadas a métodos de un objeto en particular.**
- Podemos decir que el objeto persona envía el mensaje "girar a la izquierda" al objeto bicicleta.
- Los mensajes pueden contener parámetros. Por ejemplo teniendo un método en la clase bicicleta llamado "Girar" que recibe como parámetro la dirección (derecha o izquierda).

# Elementos básicos de la P00

## Mensajes

- Un mensaje está compuesto por los siguientes tres elementos:
  - El **objeto destino**, hacia el cual el mensaje es enviado
  - El nombre del **método** a llamar
  - Los **parámetros** solicitados por el método

# Elementos básicos de la P00

Ejemplo de clase:

Class Hotel

Atributos

Nombre: Cadena;

Dirección: Cadena;

Dueño: Compañía;

Director: Persona;

Facilidades: Set (Tipos\_opcion):

Métodos

create ();

reservarHabitacion(Habitación:integer,

Huesped:Persona, Fecha\_llegada,

Fecha\_partida:Tipo\_fecha);

end Hotel.

CLASE HOTEL	
ATRIBUTOS	
Nombre	Cadena
Dirección	Cadena
Dueño	Compañía
Director	Persona
Facilidades	Set (Tipos_opcion)
MÉTODOS	
create()	
reservarHabitacion(entero, persona, fecha)	

# Elementos básicos de la P00

## Instanciación:

- Podemos interpretar que una clase es el plano que describe como es un objeto de la clase, por tanto podemos entender que a partir de la clase podemos fabricar objetos. A ese objeto construido se le denomina instancia, y al proceso de construir un objeto se le llama **instanciación** .
- Cuando se construye un objeto es necesario dar un valor inicial a sus atributos, es por ello que existe **un método especial en cada clase, llamado constructor**, que es ejecutado de forma automática cada vez que es instanciada una clase.
- Generalmente **el constructor se llama igual que la clase y no devuelve ningún valor**.

# Elementos básicos de la P00

Instancia de Hotel	
Nombre	La Pedregosa
Dirección	Av. Los proceres
Dueño	Instancia de Compañía
Gerente	Instancia de Persona
Facilidades	Piscina, Sauna, Golf

Instancia Compañía	
Nombre	Turisol
Oficina_general	Mérida
Teléfono	22454

Instancia Persona	
Nombre	Pedro perez
Dirección	Calle 22 entre Av. 2 y 3

# Características de la P00

- Son **4 las características básicas** que debe cumplir un objeto para denominarse como tal.
- Estas características son:
  - Abstracción.
  - Encapsulamiento.
  - Herencia.
  - Polimorfismo

# Características de la POO

Estas características de POO nos van a permitir:

- Aislar cada componente del resto de la aplicación.
- Aprovechar nuestro esfuerzo, en su buen funcionamiento.
- Controlar cada uno de los objetos,
- Desarrollar código más breve y conciso
- Reutilizar el código escrito.

# Características de la P00

## **Abstracción:**

- Es la capacidad de un objeto de cumplir sus funciones independientemente del contexto en el que se utilice.
- Ejemplo un objeto “cliente” siempre expondrá sus mismas propiedades y dará los mismos resultados a través de sus eventos, sin importar el ámbito en el cual se haya creado.



# Características de la P00

## Encapsulamiento:

- Esta característica es la que denota la capacidad del objeto de responder a peticiones a través de sus **métodos** sin la necesidad de exponer los medios utilizados para llegar a brindar estos resultados.
- O sea, el método Reservar\_Habitacion() del objeto “Hotel” antes mencionado, siempre nos va a hacer la reserva de la habitación, sin necesidad de tener conocimiento de cuáles son los recursos que ejecuta para llegar a brindar este resultado.

# Características de la P00

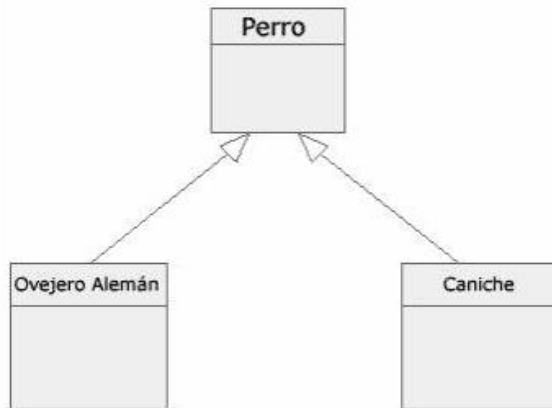
## Herencia:

- Es la característica por la cual los objetos para su creación se basan en una clase de base, heredando todas sus propiedades, métodos y eventos; los cuales a su vez pueden o no ser implementados y/o modificados.
- Por ejemplo puedo crear la **clase ClienteVip** que hereda todos los atributos y métodos de la clase Persona, además puede incluir otros específicos.

# Características de la P00

## Herencia:

- Por ejemplo ovejero alemán y caniche son diferentes razas de perros. En la terminología orientada a objetos "Ovejero Alemán" y "Caniche" son subclases de la clase perro. De forma similar Perro es la superclase de "Ovejero Alemán".



# Características de la P00

## **Herencia:**

- Cada subclase hereda los atributos de la superclase. Tanto la clase "Ovejero Alemán" como "Caniche" tendrán los atributos nombre, color de pelo, altura definidos en la clase Perro.
- Una subclase no está limitada únicamente a los atributos de su superclase, también puede tener atributos propios, o redefinir algunos definidos anteriormente en la superclase.
- No se está limitado tampoco a un solo nivel de herencia, se pueden tener todos los que se consideren necesarios.
- Gracias a la herencia, los programadores pueden reutilizar código una y otra vez.

# Características de la P00

## Polimorfismo:

- Polimorfismo significa que la misma **operación** puede **comportarse diferentemente** sobre distintas clases.
- Por ejemplo, la operación "mover" puede comportarse diferentemente sobre una clase llamada Ventana y una clase llamada PiezasAjedrez.

# P00 en JAVA

# P00 en JAVA

- Java es un **lenguaje orientado a objetos** y programar en Java consiste en escribir las definiciones de las clases y utilizar esas clases para crear objetos de forma que representen correctamente el problema que queremos resolver.
- Las clases son **predefinidas y definidas** por el programador.
- En función de la estructura de la clase y del uso tenemos dos tipos básicos:
  - **Clase – Tipo de datos:** definen el conjunto de posibles valores que tomarán los objetos y las operaciones que se realizarán en estos.
  - **Clase – Programa:** son los que inician la ejecución del código.

# Ejemplo de clase

```
public class Circulo {
    private double radio;
    private String color;
    private int centroX, centroY;
    //crea un círculo de radio 50, negro y centro en (100,100)
    public Circulo() {
        radio = 50;
        color = "negro";
        centroX = 100;
        centroY = 100;
    }
    //consulta el radio del círculo
    public double getRadio() {
        return radio;
    }
    //actualiza el radio del círculo a nuevoRadio
    public void setRadio(double nuevoRadio) {
        radio = nuevoRadio;
    }
    //decrementa el radio del círculo
    public void decrece() {
        radio = radio / 1.3;
    }
    //calcula el área del círculo
    public double area() {
        return 3.14 * radio * radio;
    }
    //obtiene un String con las componentes del círculo
    public String toString() {
        return "Círculo de radio " + radio + ", color " + color + " y centro (" + centroX + ", " + centroY + ")";
    }
}
```



# Ejemplo de clase

```
public class PrimerPrograma {  
  
    public static void main(String[] args) {  
  
        // Crear un círculo  
        Circulo c1 = new Circulo();  
        c1.setRadio(2.9);  
        System.out.println("Los datos del círculo: " + c1.toString());  
    }  
}
```

# Estructura de una clase

- A continuación se especifica el esquema de definición de una clase:

```
[ámbito] class NombreDeLaClase {
```

```
    // Definición de atributos
```

```
    [ámbito] tipo nombreVar1;
```

```
    [ámbito] tipo nombreVar2;
```

```
    .....
```

```
    // Definición de métodos
```

```
    // Constructores
```

```
    ...
```

```
    // Otros métodos
```

```
}
```

# Ámbito de declaración: private y public

- Toda la información declarada **private** es exclusiva del objeto e inaccesible desde fuera de la clase.
  - Cualquier intento de acceso a las variables de instancia radio o color que se realice desde fuera de la clase Circulo (p.e., en la clase PrimerPrograma) dará un error de compilación.

```
private double radio;  
private String color;
```

- Toda la información declarada **public** es accesible desde fuera de la clase.
  - En el caso de los métodos getRadio() o area() de la clase Circulo.

```
public double getRadio() {  
    return radio;    }
```

# Atributos de una clase

- Los atributos o variables de instancia representan información de cada objeto de la clase y se declaran de un tipo, suelen declararse de acceso privado.

```
// Definición de atributos  
[ámbito] tipo nombreVar1;  
[ámbito] tipo nombreVar2;
```

**Por ejemplo:**

```
public class Circulo {  
    // Definición de atributos  
    private double radio;  
    private String color;  
    private int centroX, centroY; ...  
}
```

# Métodos de una clase

- Los **métodos** definen las operaciones que se pueden hacer sobre los objetos de la clase y se describen indicando:
  - **La cabecera:** nombre, tipo de resultado y lista de parámetros necesarios para hacer el cálculo.
  - **El cuerpo:** contiene la secuencia de instrucciones necesarias

**Por ejemplo:**

```
public class Circulo {  
    ...  
    public double area() {  
        return 3.14 * radio * radio;  
    }  
}
```

# Métodos de una clase

- Los clasificamos según su función:
  - **Constructores:** permiten crear el objeto
  - **Modificadores:** permiten modificar el estado (valores de los atributos)
  - **Consultores:** permiten conocer, sin cambiar, el estado del objeto

```
public class Circulo {  
    //Constructor vacío  
    public Circulo() { radio = 50; color = "negro"; }  
    //Constructor con 3 parámetros  
    public Circulo(double r, String c, int px, int py)  
    { radio = r; color = c; centroX = px; centroY = py; }  
    //Consultor: getter  
    public double getRadio() { return radio; }  
    //Consultor: setter  
    public void setRadio(double nuevoRadio) { radio = nuevoRadio; }  
}
```

# P00 en JAVA

- Tipos de clases
  - Debemos tener en cuenta que según la estructura de la clase y el uso que hagamos de ésta tenemos 2 tipos de clases:
    - Clase - Tipo de datos: definen el conjunto de posibles valores que pueden tomar los objetos y las operaciones que se pueden realizar sobre éstos.
    - Clase - Aplicación: son los que inician la ejecución del código.
  - Así, en nuestro ejemplo tenemos la clase Television que será del primer tipo y la clase aplicación que será del segundo tipo.

# P00 en JAVA

- A esta clase le vamos a añadir un atributo llamado canal que va a ser de tipo int

```
public class Televisor {  
    int canal; }  
}
```

- A la Clase Televisor le vamos a añadir los métodos el constructor por defecto, subirCanal(), bajarCanal() y getCanal()

```
class Televisor {  
    int canal;  
    // El constructor por defecto, aunque no es necesario  
    public Televisor(){}  
    public void subirCanal() {  
        canal ++; }  
    public void bajarCanal() {  
        canal --; }  
    public int getCanal() {  
        return canal; }  
}
```



# P00 en JAVA

- Añadimos a la clase Aplicación el siguiente código:

```
Televisor tv;
```

- en este punto estamos declarando una variable de referencia tv. De momento su valor es null ya que todavía no apunta a ninguna Instancia u Objeto

```
tv = new Televisor();
```

- el operador new nos indica que se acaba de crear un nuevo Objeto, que es una Instancia de la Clase Televisor
- ahora la variable de referencia tv contiene la dirección de memoria de dicha Instancia

# P00 en JAVA

- Para invocar los métodos de un Objeto, tenemos que tener primeramente una referencia a ese objeto y después escribir un punto "." y finalmente el nombre del método que queremos llamar.
- Este código se lo añadimos a la Clase **Aplicación**

```
tv.subirCanal();  
System.out.println("El canal seleccionado es el: " + tv.getCanal());  
tv.bajarCanal();  
System.out.println("El canal seleccionado es el: " + tv.getCanal());
```

# P00 en JAVA

- Los Constructores se declaran de la siguiente forma:

```
nombreDelConstructor(listaDeParámetros){  
    cuerpoDelConstructor  
}
```

- **tipoValorDevuelto**: un Constructor no devuelve ningún valor, ni siquiera void.
  - **nombreDelConstructor**: el nombre del Constructor es el mismo que el nombre de la Clase.
- Y siguiendo la convención de nombres en Java, este nombre tendría que tener
    - la primera letra de la primera palabra compuesta en mayúsculas.
    - la primera letra de la segunda y restantes palabras compuesta en mayúsculas

# P00 en JAVA

**Sobrecarga de métodos:** tenemos dos constructores uno sin argumento y el otro con un argumento de tipo int, es una de las tres formas de implementar el Polimorfismo.

```
public class Televisor {  
    int canal;  
    public Televisor() {};  
    public Televisor(int valorCanal) {  
        canal = valorCanal;  
    }  
}
```

# P00 en JAVA

Ahora vamos a crear dos objetos de tipo Televisor. Uno de ellos estará referenciado por una variable de referencia llamada tv1 y el otro por otra variable de referencia llamada tv2, en clase **Aplicacion**.

```
public static void main(String[ ] args) {  
    ...  
    Televisor tv1 = new Televisor();  
    System.out.println("El canal por defecto es : "+ tv1.canal);  
    tv1.canal = 8;  
    System.out.println("El canal del primer televisor es el: " + tv1.getCanal());  
    Televisor tv2=new Televisor(6);  
    System.out.println("El canal del segundo televisor es el: " +tv2.getCanal());  
}
```

# P00 en JAVA

- Encapsulación
  - A través de la Encapsulación se puede **controlar** qué partes de un programa pueden acceder a las variables y métodos de un Objeto.
  - La encapsulación se basa en el **control de acceso o ámbito**.
- Los ámbitos pueden ser:
  - **public** : puede ser accedido por cualquier parte del programa
  - **private** : sólo puede ser accedido por otros miembros de su Clase

# P00 en JAVA

A la Clase Televisor le tenemos que hacer las siguientes modificaciones:

```
public class Televisor {
    private int canal;
    public Televisor() {}
    public Televisor(int valorCanal) {
        canal=valorCanal;}
    public void subirCanal() {
        setCanal(canal + 1); }
    public void bajarCanal() {
        setCanal(canal - 1); }
    public int getCanal() {
        return canal; }
    public void setCanal(int valorCanal) {
        if (valorCanal < 0){
            canal = 0;    }
        else {
            canal = valorCanal; }
    }}
}
```

# P00 en JAVA

- **private int canal;**

- Al indicar que el ámbito es private estamos diciendo que sólo desde dentro de la Clase Televisor se puede acceder al atributo canal
- setCanal(canal + 1); setCanal(canal - 1);
- Cuando el usuario baja el canal, se podría dar el caso que llegara al canal 0 y si sigue bajando el canal llegaría al canal -1. Para evitar esto invocamos al método setCanal(...) pasándole como argumento el resultado de la operación de bajar el canal.
  - si más adelante nos dijeran que por ejemplo el canal más alto no puede superar el número 99, simplemente tendríamos que ampliar el filtro en el método setCanal(...)

- **public void setCanal(int valorCanal) { ... }**

- public, este método podrá ser llamado tanto desde el Constructor de su Clase como desde cualquier otra parte del programa



# P00 en JAVA

**Subir y bajar la intensidad del color del televisor** : Ahora queremos que el Televisor estándar además de ofrecer la operativa de cambiar los canales, también nos permita aumentar y disminuir la intensidad del color. Para ello vamos a crear cuatro nuevos métodos

```
public void subirColor(){
    System.out.println("Televisor - subirColor(): estoy subiendo el color");
    subirColorAyuda();
}
private void subirColorAyuda() {
    System.out.println("Televisor - subirColorAyuda(): sigo subiendo el color");
}
public void bajarColor(){
    System.out.println("Televisor - bajarColor(): estoy bajando el color");
    bajarColorAyuda();
}
private void bajarColorAyuda() {
    System.out.println("Televisor - bajarColorAyuda(): sigo bajando el color");
}
```

# P00 en JAVA

Compilamos la Clase Televisor y seguidamente modificamos la clase Aplicacion

```
package paqtvestandar;
    public class Aplicacion {
        public static void main(String[] args) {
            ...
            tv.subirColor();
            //tv.subirColorAyuda();
        }
    }
```

Si descomentamos la invocación al método subirColorAyuda() apuntado por la variable de referencia tv, vamos a poder ver que el compilador se queja (y con razón!) porque estamos intentando acceder a un método de ámbito private

# P00 en JAVA

- Implementa las Clases Televisor de tal forma que cuando creamos una instancia de Televisor, ésta ya tenga el volumen por defecto en posición 5.
- Por lo que respecta a la implementación de los métodos subirVolumen() y bajarVolumen(), no tendremos en cuenta los valores negativos ni tampoco el valor máximo del volumen, así que tendremos que implementar estos métodos con el siguiente código
  - `volumen = volumen + 1`
  - `volumen = volumen - 1`
- Desde la Clase Aplicacion tenemos que:
  - crear una instancia de Televisor
  - seguir cambiando los canales y subiendo el color como en el ejemplo anterior
  - subir el volumen una posición
  - mostrar un mensaje diciendo que "La posición del volumen es: "....

# Ejercicios

**Ejercicio 1:** Modificar el ejemplo anterior para añadirle el atributo color.

- Cuando se cree el objeto por defecto se inicializara a 7.
- El método subirColor será el que incremente la variable color.
- El método bajarColor será el que decremente la variable color.
- Los métodos subirColorAyuda y bajarColorAyuda deben mostrar el valor del color.

**Ejercicio 2:** Modifica el ejercicio anterior para que los canales sean del 0 al 10, y en el caso de que estés en el canal 10 y subas el canal nos de el 0 y en el caso de que estés en el canal 0 y bajes el canal nos de el 10.

**Ejercicio 3:** Modifica el ejercicio anterior para que la intensidad de color sea de 1 a 7, y en el caso de que la intensidad sea 7 y subas la intensidad se quede en 7 y en el caso de que la intensidad sea 1 y la bajes se quede en 1.

**Ejercicio 4:** Modifica el ejercicio anterior para que el volumen sea de 0 a 15, y en el caso de que el volumen sea 15 y lo subas se quede en 15 y en el caso de que el volumen sea 0 y lo bajes se quede en 0.

# Métodos con clases

# Métodos con clases

- Los métodos definen las operaciones que se pueden realizar sobre la clase
- Declaración de métodos:

**[acceso][static][final] tipo\_retorno nombreMetodo ([args])**

**{/\*Cuerpo método\*/}**

- Donde:
  - **acceso**: public, private, protected, package
  - **static**: se puede acceder a los métodos sin necesidad de instanciar un objeto de la clase
  - **final**: constante, evita que un método sea sobrescrito
  - **tipo\_retorno**: tipo primitivo, referencia o void
  - **nombreMetodo**: identificador del método
  - **args**: lista de parámetros separados por comas

# Métodos con clases

- Devolución de valores:
  - En la declaración se debe especificar el **tipo de dato** devuelto por el método.
  - Si el método no devuelve nada, el tipo de retorno será **void**.
  - Si el método devuelve algo:
    - Se pueden devolver tipos primitivos u objetos, ya sean objeto predefinidos u objetos de clases creadas por el propio usuario.
    - El cuerpo del método deberá contener una instrucción similar a `→ return valorDevuelto;`

# Métodos con clases

- Los modificadores de acceso se utilizan para controlar el acceso a clases, atributos y métodos.
- Tipos de modificadores de acceso:
  - public
  - private
  - protected
  - package
- Clases
  - public
  - package
- Variables y métodos
  - public, private, protected y package



# Métodos con clases

```
Class Punto {
```

```
    public int x;
```

```
    public int y;
```



**ATRIBUTOS**

```
    public double CalcularDistanciaCentro(){
```

```
        double z;
```

```
        z = Math.sqrt((x*x)+(y*y));
```

```
        return z;
```

```
    }
```

```
}
```



**MÉTODO**

# Métodos con clases

- Los métodos se aplican siempre a un objeto de la clase usando el operador punto ( . ) salvo los métodos declarados como static. Dicho objeto es un argumento implícito:


```
nombreObjeto.metodo ( [args] ) ;
```

- Argumentos explícitos: van entre paréntesis, a continuación del nombre del método. Los tipos primitivos se pasan por valor, para cambiarlos hay que encapsularlos dentro de un objeto y pasar el objeto.
- Ejemplo:

```
double d = p1.CalcularDistanciaCentro();  
System.out.println("distancia:  " + d);
```

# Métodos con clases

- Los métodos pueden definir variables locales:
  - Visibilidad limitada al propio método
  - Variables locales no se inicializan por defecto

```
public double calcularDistanciCentro() {  
    double z;   
    z = Math.sqrt((x*x)+(y*y));  
    return z;  
}
```

# Métodos con clases

- **Método constructor**
- Método que se llama automáticamente cada vez que se llama un objeto de una clase.
- Características:
  - Reservar memoria e inicializar las variables de la clase
  - No tienen valor de retorno (ni siquiera void)
  - Tiene el mismo nombre que la clase
- Sobrecarga:
  - Una clase puede tener varios constructores, que se diferencia por el tipo y su número de sus argumentos

# Métodos con clases

- Constructores por defecto:
  - Constructor sin argumentos que inicializa:
    - Tipos primitivos a su valor por defecto
    - Strings y referencias a objetos a null
- Creación de objetos:

```
Punto p = new Punto(1, 1);
```

- Declaración: Punto p;
- Instanciación: new Punto(1, 1); crea el objeto
- Inicialización: new Punto(1, 1); llama al constructor que es el que llama al objeto.

# Métodos con clases

```
public class Punto {  
    public int x;  
    public int y;  
    public Punto(int a) { x=a; y=a; }  
    public Punto(int a, int b) { x=a; y=b; }  
    public double CalcularDistanciaCentro(){  
        double z;  
        z = Math.sqrt((x*x)+(y*y));  
        return z; }  
}
```

# Métodos con clases

- La variable this
- Definida implícitamente en el cuerpo de los métodos
- Dentro de un método o de un constructor, this hace **referencia al objeto actual**


```
public class Punto {  
    public int x = 0;  
    public int y = 0;  
    public Punto(int a, int b) {  
        x = a;  
        y = b;  
    }  
}
```

```
public class Punto {  
    public int x = 0;  
    public int y = 0;  
    public Punto(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
}
```

# Métodos con clases

- Un constructor puede llamar a otro constructor de su propia clase si:
  - El constructor al que se llama está definido
  - Se llama utilizando la palabra `this` en la primera sentencia

```
public Cliente(String n, long dni){  
    nombre = n;  
    DNI = dni;}  
public Cliente(String n, long dni, long tel){  
    nombre = n;  
    DNI = dni;  
    telefono = tel;}  
public Cliente(String n, long dni, long tel){  
    this(n,dni);  
    telefono = tel;}
```

A red line originates from the `this(n,dni);` statement in the third constructor and points back to the first constructor's signature, illustrating the call to another constructor in the same class.



# Métodos con clases

- Puede haber dos o más métodos que se llamen igual en la misma clase
- Se tienen que diferenciar en los parámetros (tipo o número)
- El tipo de retorno es insuficiente para diferenciar dos métodos

# Métodos con clases

- Modificadores de acceso
  - Los modificadores de acceso permiten al diseñador de una clase determinar quién accede a los datos y métodos miembros de una clase.
  - Los modificadores de acceso preceden a la declaración de un elemento de la clase (ya sea dato o método), de la siguiente forma:

```
[modificadores] tipo_variable nombre;
```

```
[modificadores] tipo_devuelto nombre_Metodo (lista_Argumentos);
```

# Métodos con clases

- Existen los siguientes modificadores de acceso:
- **public** - Todo el mundo puede acceder al elemento. Si es un dato miembro, todo el mundo puede ver el elemento, es decir, usarlo y asignarlo. Si es un método todo el mundo puede invocarlo.
- **private** - Sólo se puede acceder al elemento desde métodos de la clase, o sólo puede invocarse el método desde otro método de la clase.
- **protected** - Se explicará en el capítulo dedicado a la herencia.
- **sin modificador** - Se puede acceder al elemento desde cualquier clase del package donde se define la clase

Pueden utilizarse estos modificadores para cualquier tipo de miembros de la clase, incluidos los constructores (con lo que se puede limitar quien puede crear instancias de la clase).

# Métodos con clases

Modificadores de acceso para clases

Las clases en sí mismas pueden declararse:

- **public** - Todo el mundo puede usar la clase. Se pueden crear instancias de esa clase, siempre y cuando alguno de sus constructores sea accesible
- sin modificador - La clase puede ser usada e instanciada por clases dentro del package donde se define

Las clases no pueden declararse ni **protected**, ni **private**

# Métodos con clases

- Los elementos (atributos y métodos) definidos como **static** son independientes de los objetos de la clase
- Atributos estáticos - variables de clase:
  - Un atributo static es una variable global de la clase
  - Un objeto de la clase no copia los atributos static → todas las instancias comparten la misma variable
  - Uso de atributos estáticos:
    - nombreClase.nombreAtributoEstático

# Métodos con clases

```
public class MiClase {  
    String nombre;  
    static int contador;  
    public MiClase(String n) { nombre = n; contador++; }  
    public void imprimeContador()  
    { System.out.println("Contador: " + contador); }  
}  
  
...  
  
MiClase o1 = new MiClase("Primero");  
MiClase o2 = new MiClase("Segundo");  
o2.imprimeContador(); //2  
MiClase.contador = 1000;  
o2.imprimeContador(); //1000
```

# Métodos con clases

- Métodos estáticos:

- Un método static es un método global de la clase
- Un objeto no hace copia de los métodos static
- Suelen utilizarse para acceder a atributos estáticos
- No pueden hacer uso de la referencia this

- Llamada a métodos estáticos:

```
nombreClase.nombreMetodoEstático()
```

- Ejemplo:

```
public static void actualizaContador( ) {  contador = 0;  }  
//Principal:  
...  
MiClase.actualizaContador( );  
o.imprimeContador( );
```

# Métodos con clases

```
public class Persona{
    //Atributo estático de la clase
    private static int nPersonas;
    //Cada instancia incrementa este atributo
    public Persona ( ) {
        nPersonas++;
    }
    //Método estático que retorna un atributo estático
    public static int getNPersonas( ) {
        return nPersonas;
    }
}

public static void main (String[] args) {
    //Se crean instancias
    Persona p1 = new Persona( );
    Persona p2 = new Persona( );
    Persona p3 = new Persona( );
    //Accedemos al método estático para ver el número
    //de instancias de tipo Personas creadas
    System.out.println(Persona.getNPersonas( ));
}
```



# Métodos con clases

- **Un Atributo static:**

- No es específico de cada objeto. Solo hay una copia del mismo y su valor es compartido por todos los objetos de la clase.
- Podemos considerarlo como una variable global a la que tienen acceso todos los objetos de la clase.
- Existe y puede utilizarse aunque no existan objetos de la clase.
- Para acceder a un atributo de la clase se escribe:
- `NombreClase.atributo`

- **Un método static:**

- Tiene acceso solo a los atributos estáticos de la clase.
- No es necesario instanciar un objeto para poder utilizarlo.
- Para acceder a un método de la clase se escribe:
- `NombreClase.metodo()`

# Métodos con clases

- La palabra reservada **final** indica que su valor no puede cambiar
- Si se define como final:
  - **Una clase** → No puede tener clases hijas (seguridad y eficiencia del compilador)
  - **Un método** → No puede ser redefinido por una subclase
  - **Una variable** → Tiene que ser inicializada al declararse y su valor no puede cambiarse
- Suele combinarse con el modificador **static**
- El identificador de una variable final debe escribirse en mayúsculas

# Métodos con clases

```
public class Constantes{  
    //Constantes públicas  
    public static final float PI = 3.141592f;  
    public static final float E = 2.728281f;  
    //main  
    public static void main (String[] args) {  
        System.out.println("PI = " + Constantes.PI);  
        System.out.println("E = " + Constantes.E);  
    }  
}
```

Arrays de objetos

# Arrays de objetos

## La clase precio

```
public class Precio {  
    public double euros;  
  
    public double getPrecio() {  
        return euros;  
    }  
    public void setPrecio(double x) {  
        euros=x;  
    }  
}
```

# Arrays de objetos

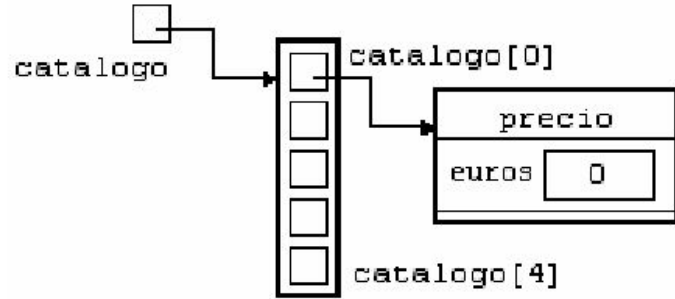
## La clase PruebaPrecio

```
public class PruebaPrecio {  
    public static void main (String [] args ) {  
        Precio p = new Precio();  
        p.setPrecio(56.8);  
        System.out.println("Valor = " + p.getPrecio());  
        Precio q = new Precio(); // Crea una referencia y el objeto  
        q.euros=75.6;             // Asigna 75.6 al atributo euros  
        System.out.println("Valor = " + q.euros);  
    }  
}
```

# Arrays de objetos

El uso de vectores no tiene por qué restringirse a elementos de tipo primitivo. Por ejemplo, pueden crearse referencias a arrays de la clase Precio o de la clase String:

```
Precio [ ] catalogo;  
catalogo = new Precio [5];  
catalogo[0] = new Precio();
```



Primero se crea la referencia al array de punteros, luego se crea el array de punteros y, finalmente, se crea la instancia de la clase precio y se almacena su dirección de memoria en el primer elemento del array de punteros.

# Arrays de objetos

```
public class ArrayPrecios {  
    public static void main (String [] args) {  
        Precio [ ] catalogo;  
        catalogo = new Precio [5];  
        for (int i=0; i<catalogo.length; i++) {  
            catalogo[i] = new Precio();  
            catalogo[i].setPrecio(10*Math.random());  
            System.out.println("Producto "+ i + " : " + catalogo[i].getPrecio());  
        }  
        //Busqueda del máximo precio  
        double maximo=catalogo[0].getPrecio();  
        for (int i=1; i<catalogo.length; i++) {  
            if (maximo<catalogo[i].getPrecio())  
                maximo=catalogo[i].getPrecio();  
        }  
        System.out.println("El mas caro vale "+ maximo + " euros");  
    } }  
}
```



**ArrayList de objetos**

# ArrayList de objetos

La clase **ArrayList** permite almacenar elementos en memoria de manera dinámica.

La principal diferencia con los arrays es que el número de elementos que almacena no está limitado por un número prefijado.

# ArrayList de objetos

La declaración de un ArrayList se hace según el siguiente formato:

**ArrayList<nombreClase> nombreDeLista;**

- Entre <> indicamos la clase o tipos básicos de los objetos que se almacenarán.
- Ejemplo:
  - ArrayList<String> listaPaises;
  - ArrayList<Persona> listaPersonas;

# Creación del ArrayList

La creación de un ArrayList se hace según el siguiente formato:

```
nombreDeLista = new ArrayList();
```

También se puede declarar a la vez que se crea:

**ArrayList<nombreClase> nombreDeLista = new ArrayList();**

La clase ArrayList forma parte del paquete java.util por lo que hay que incluir en la parte inicial del código el paquete

```
import java.util.ArrayList
```

# Insertar elementos al final

El método **add** de la clase ArrayList posibilita añadir elementos.

Se colocan después del último elemento que hubiera en el ArrayList

```
boolean add(Object elementoAInsertar);
```

## Ejemplo:

```
ArrayList<String> listaPaises = new ArrayList();  
listaPaises.add("España");    //Ocupa la posición 0  
listaPaises.add("Francia");   //Ocupa la posición 1  
listaPaises.add("Portugal");  //Ocupa la posición 2
```

# Insertar elementos en una determinada posición

Es posible insertar un elemento en una determinada posición desplazando el elemento que se encontraba en esa posición, y todos los siguientes, una posición más.

```
void add(int posicion, Object elemento);
```

Se utiliza el método add indicando como primer parámetro el número de la posición.

## Ejemplo:

```
ArrayList<String> listaPaises = new ArrayList();  
listaPaises.add("España");  
listaPaises.add("Francia");  
listaPaises.add("Portugal");  
//El orden hasta ahora es: España, Francia, Portugal  
listaPaises.add(1, "Italia");  
//El orden ahora es: España, Italia, Francia, Portugal
```

# Eliminar elementos

Para eliminar un determinado elemento se emplea el método **remove** al que se le puede indicar por parámetro un valor int con la posición a suprimir o bien , se puede especificar el elemento a eliminar si es encontrado en la lista.

## Ejemplo:

```
Object remove(int posicion)
```

```
boolean remove(Object elementoASuprimir)
```

```
ArrayList<String> listaPaises = new ArrayList();  
listaPaises.add("España");  
listaPaises.add("Francia");  
listaPaises.add("Portugal");  
//El orden hasta ahora es: España, Francia, Portugal  
listaPaises.add(1, "Italia");  
//El orden ahora es: España, Italia, Francia, Portugal  
listaPaises.remove(2);  
//Eliminada Francia, queda: España, Italia, Portugal  
listaPaises.remove("Portugal");  
//Eliminada Portugal, queda: España, Italia
```

# Consultar un elemento

El método **get** permite obtener el elemento almacenado en una determinada posición

```
Object get(int posicion)
```

## Ejemplo:

```
System.out.println(listaPaises.get(3));  
//Siguiendo el ejemplo anterior, mostraría: Portugal
```



# Modificar un elemento

El método **set** permite modificar el elemento almacenado en una determinada posición

```
Object set(int posicion, Object nuevoElemento)
```

## Ejemplo:

```
listaPaises.set(1, "Alemania");  
//Se modifica el país que había en la posición 1 por Alemania
```

# Buscar un elemento

El método **indexOf** retorna un valor int con la posición que ocupa el elemento que se indica por parámetro

```
int indexOf(Object elemento)
```

## Ejemplo:

```
String paisBuscado = "Francia";  
int pos = listaPaises.indexOf(paisBuscado);  
if(pos != -1)  
    System.out.println(paisBuscado + " se ha encontrado en la  
    posición: "+pos);  
else  
    System.out.println(paisBuscado + " no se ha encontrado");
```

# Buscar un elemento

Para ello utilizaremos un bucle como lo haríamos con los arrays convencionales. Para obtener el número de elementos se puede utilizar el método **size()**.

## Ejemplo:

```
for(int i=0; i<listaPaíses.size(); i++)  
    System.out.println(listaPaíses.get(i));
```

# Buscar un elemento con foreach

El foreach nos permite recorrer los elementos de un arrayList sin utilizar un índice. Aunque la flexibilidad es menor que el bucle for normal, es muy útil para recorrer un arrayList.

## **Ejemplo:**

```
for (String s : listaPaíses)  
    System.out.println(s);
```

# Otros métodos de interés

- **void clear()**: Borra todo el contenido de la lista
- **Object clone()**: Retorna una copia de la lista
- **boolean contains(Object elemento)**: Retorna true si se encuentra el elemento en la lista, y false en caso contrario.
- **boolean isEmpty()**: Retorna true si la lista está vacía
- **Object[] toArray()**: Convierte la lista a un array

# Preguntas

