

Grado Superior

# **Desarrollo de Aplicaciones web**



# **Programación Tema 12**

## **Programación Orientada a Objetos (POO)**



# POO

## Objetos

La orientación a objetos es un paradigma de programación que facilita la creación de software de calidad. Mejora el mantenimiento, la ampliación y la reutilización del software.

La programación orientada a objetos se acerca al modo de pensar del hombre y no al de la máquina.



# POO

## Objetos

El **elemento básico** de este paradigma no es la función (elemento básico de la programación estructurada), sino un ente denominado **objeto**.

Un objeto es la representación de un elemento de la vida real en nuestro programa.

Por ejemplo: Un coche eléctrico



# POO

## Objetos

Un objeto posee una **identidad**, un **estado** y un **comportamiento**

La **identidad** queda definida por el valor de un conjunto de propiedades, en nuestro caso:

- Matrícula
- Color
- Número de bastidor
- Duración de la batería



# POO

## Objetos

**Identidad:** Diferentes objetos coche tendrán diferentes valores para cada una de sus propiedades



# POO

## Objetos

Un objeto posee una **identidad**, un **estado** y un **comportamiento**

El **estado** queda definido por un otra serie de propiedades, en nuestro caso:

- carga de la batería
- situación (valores: parado, en movimiento...)



Un objeto posee una **identidad**, un **estado** y un **comportamiento**

El **comportamiento** determina como responde el objeto ante peticiones de otros objetos.. Por ejemplo un objeto conductor puede lanzar el mensaje arrancar a un objeto coche. El comportamiento determina qué ocurre cuando se arranca el coche



# POO

## Clases

---

Una **clase** es como un molde, define de forma genérica a todos los objetos del mismo tipo.

Por ejemplo la clase Coche Eléctrico representa a todos los coches eléctricos.



# POO

## Clases

### Ejercicios:

- ¿Qué objetos visualizas en un taller de reparación de automóviles?  
¿Qué clases?

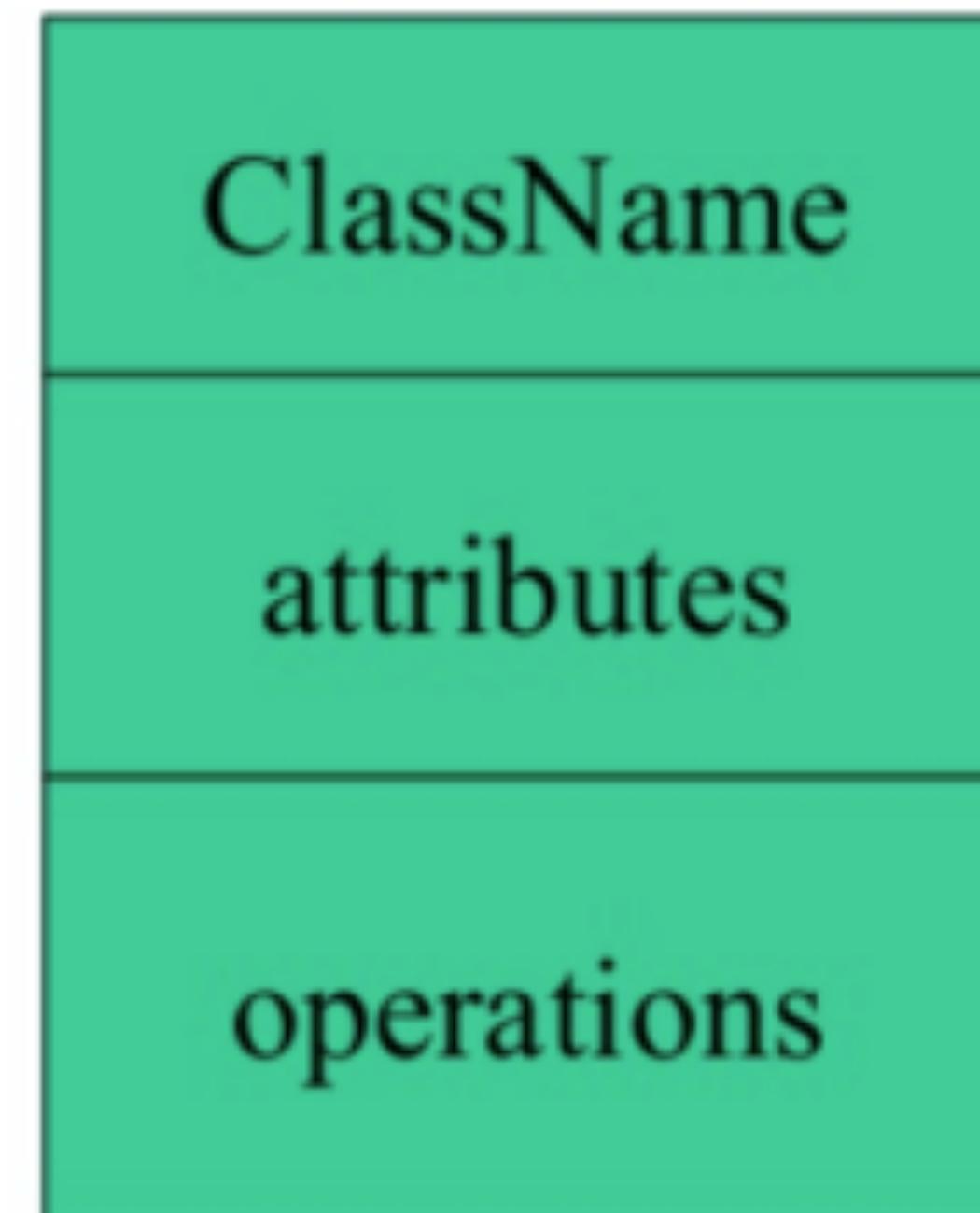


# POO

## Clases

---

**Notación UML:**



# POO

## Clases

---

### Ejercicios:

- ¿Qué objetos visualizas en tu centro educativo? ¿Qué clases?  
¿Qué propiedades y comportamientos tienen esos objetos?  
Utilizar notación UML
- ¿Y en un aeropuerto?
- ¿Y en un banco?



# POO

## Clases

---

En la vida real todos los objetos tienen una serie de características y un comportamiento. Por ejemplo, una puerta tiene color, forma, dimensiones, material... (características) y puede abrirse, cerrarse (comportamiento).

En Programación Orientada a Objetos, un objeto es una representación de un objeto de la vida real, tendrá unas características y realiza unas acciones que pueden operar con esos datos.



# POO

## Objetos

---

**En POO una clase se define mediante:**

**Campos o atributos o variables miembro:** componentes de un objeto que almacenan datos. Estos datos pueden ser de tipo primitivo (boolean, int, double, char...) o, instancias de otra clase. Un atributo representa una propiedad determinada de un objeto.

**Métodos (funciones):** Lleva a cabo una determinada acción o tarea con los atributos.



# POO

## Objetos

A partir de ahora cuando abordemos un desarrollo debemos identificar los objetos que se deben manejar en el sistema y por tanto las clases que los definen.

Una vez definidas las clases debemos ver qué atributos deben tener

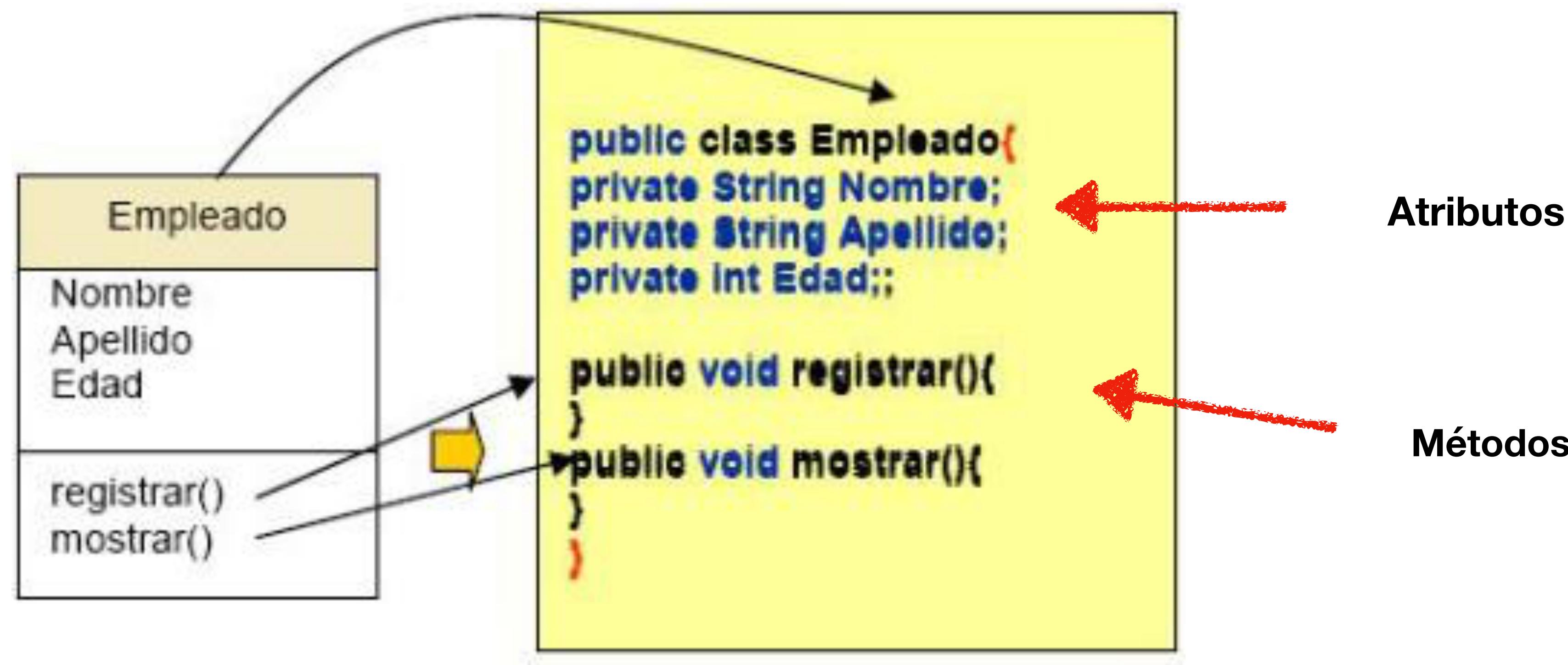
A continuación debemos analizar el comportamiento de los objetos y definir los métodos necesarios en las clases para implementar ese comportamiento.



# POO

## Objetos

### Paso de una clase definida en UML a una clase en Java



# POO

## Objetos

### Acceso:

- **public:** Un elemento public se puede acceder desde cualquier otra clase. Es el valor **por defecto**.
- **private:** Los elementos privados sólo se pueden utilizar dentro de la clase donde han sido definidos
- **protected:** Los elementos protegidos sólo se pueden utilizar dentro de la clase, sus herederos y cualquier clase definida en el mismo paquete.



# POO

## Objetos

**static:** Un elemento declarado como static es único para todos los objetos de la clase. Cuando se modifica todos los objetos de la clase ven la misma modificación. Debe declararse con un valor inicial.

Un método o campo declarado como estático puede ser accedido o invocado sin la necesidad de tener que instanciar un objeto de la clase. Por ejemplo Math.floor(..)



### Métodos:

- Un método es un trozo de código que puede ser llamado o invocado por el programa principal o por otro método para realizar alguna tarea específica. El método es llamado por su nombre o identificador seguido por una secuencia de parámetros o argumentos (datos utilizados por el propio método para sus cálculos) entre paréntesis.
- Cuando el método finaliza sus operaciones, devuelve habitualmente un valor simple al programa que lo llama, que utiliza dicho valor de la forma que le convenga. El tipo de dato devuelto por la sentencia return debe coincidir con el tipo de dato declarado en la cabecera del método.



# POO

## Objetos

---

### Métodos:

Sintaxis de declaración de un método:

```
[modificadores] tipoDeDato identificadorMetodo (parametros formales)
{
    declaraciones de variables locales;
    sentencia_1;
    sentencia_2;
    ...
    sentencia_n;
    // al final se puede incluir un return
}
```

La primera línea de código corresponde a la cabecera del método. Los modificadores especifican cómo puede llamarse al método, el tipo de dato indica el tipo de valor que devuelve la llamada al método y los parámetros (entre paréntesis) introducen información para la ejecución del método. Si no existen parámetros explícitos se dejan los paréntesis vacíos. A continuación, las sentencias entre llaves componen el cuerpo del método.

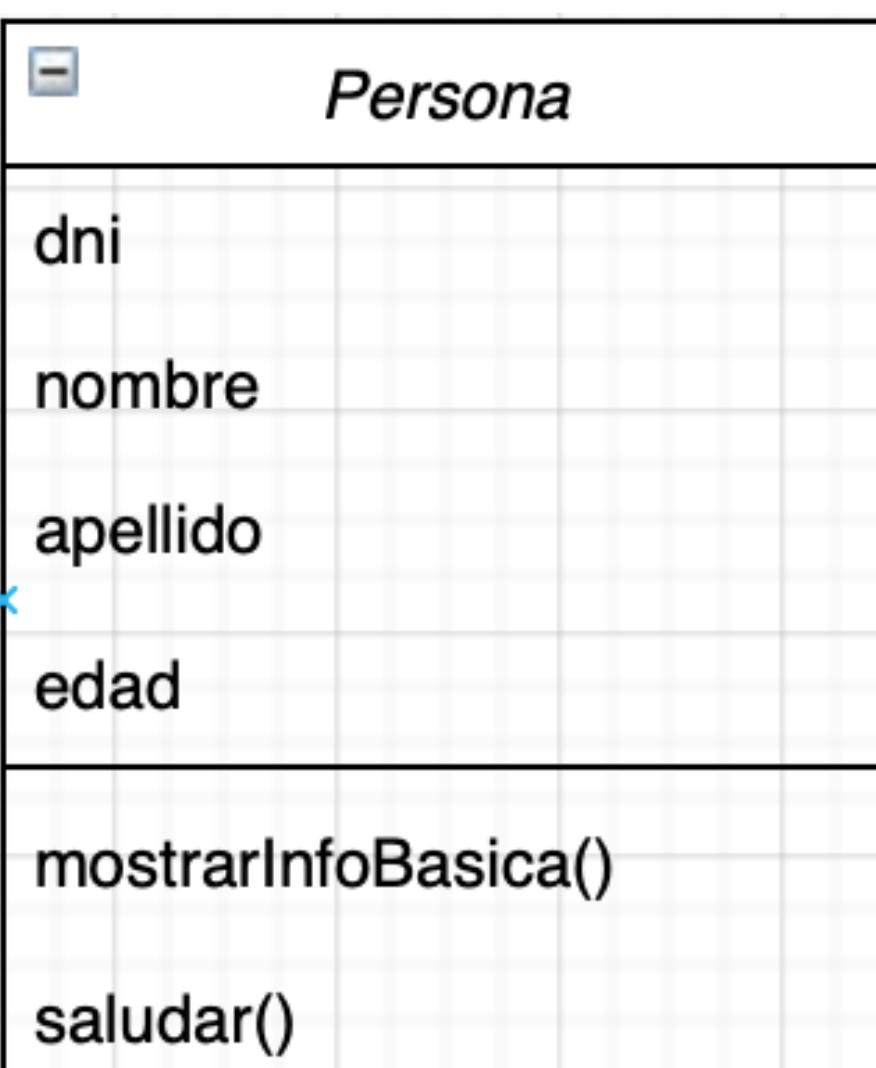
# POO

## Objetos

---

### Ejemplo:

Vamos a implementar en java la siguiente clase:



# POO

## Objetos

### Ejemplo:

Método especial:  
**constructor**

```
package poo;

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

    Persona(String docIdentidad, String nombrePersona, String apellidoPersona, int edadPersona)
    {
        dni = docIdentidad;
        nombre = nombrePersona;
        apellido = apellidoPersona;
        edad = edadPersona;
    }

    public void mostrarInfoBasicaPersona()
    {
        System.out.println("-----");
        System.out.println("DNI: " + dni);
        System.out.println("Nombre: " + nombre);
        System.out.println("Edad: " + edad);
        System.out.println("-----");
    }

    public void saludar()
    {
        System.out.println("Holaaaaaaa!!!!");
        System.out.println("Soy " + nombre);
    }
}
```

# POO

## Objetos

### Constructor:

Cuando se crea un objeto es posible definir un proceso de inicialización que prepare el objeto para ser usado. Esta inicialización se lleva a cabo invocando un método especial denominado **constructor**.

Esta invocación es implícita y se realiza automáticamente mediante el operador **new**



# POO

## Objetos

### Constructor:

El operador new devuelve una referencia al objeto que creó. Esta referencia normalmente se asigna a una variable del tipo apropiado, ejemplo:

```
Persona asistente1 = new Persona("2342424P", "Andrea", "Díaz", 31);
```



# POO

## Objetos

Observa que en la clase Persona no hay ningún método main() por tanto no se puede ejecutar ningún programa sólo con esta clase.

Para poder ejecutar un programa vamos a crear una clase que contenga main() y que cree objetos de tipo Persona

# POO

## Objetos

```
package poo;

public class Reunion {

    public static void main(String[] args) {
        Persona asistente1 = new Persona("2342424P", "Andrea", "Díaz", 31);
        Persona asistente2 = new Persona("325235K", "Lucas", "Pérez", 51);

        asistente1.saludar();
        asistente2.saludar();
    }
}

-----  
Reunion.java:  
Holaaaaaaa!!!!  
Soy Andrea  
*****  
Holaaaaaaa!!!!  
Soy Lucas  
*****
```

```
package poo;

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

    Persona(String docIdentidad, String nombrePersona, String apellidoPersona, int edadPersona)
    {
        dni = docIdentidad;
        nombre = nombrePersona;
        apellido = apellidoPersona;
        edad = edadPersona;
    }

    public void mostrarInfoBasicaPersona()
    {
        System.out.println("-----");
        System.out.println("DNI: " + dni);
        System.out.println("Nombre: " + nombre);
        System.out.println("Edad: " + edad);
        System.out.println("-----");
    }

    public void saludar()
    {
        System.out.println("Holaaaaaaa!!!!");
        System.out.println("Soy " + nombre);
    }
}
```

# POO

## Objetos

Podemos representar cualquier situación que se nos ocurra, por ejemplo vamos a controlar qué asistentes pueden asistir a la reunión en función de si son mayores de edad:



# POO

## Objetos

```
package poo;

public class Reunion2 {

    public static void main(String[] args) {
        Persona asistente1 = new Persona("2342424P", "Andrea", "Díaz", 17);
        Persona asistente2 = new Persona("325235K", "Lucas", "Pérez", 51);

        if (asistente1.esMayorDeEdad())
        {
            System.out.println("El asistente " + asistente1.getNombre() + " puede pasar");
        }
        else
        {
            System.out.println("El asistente " + asistente1.getNombre() + " no puede pasar");
        }
    }
}
```

```
final int MAYORIA_EDAD = 18;

Persona(String docIdentidad, String nombrePersona, String apellidoPersona, int edadPersona)
{
    dni = docIdentidad;
    nombre = nombrePersona;
    apellido = apellidoPersona;
    edad = edadPersona;
}

public void mostrarInfoBasicaPersona()
{
    System.out.println("-----");
    System.out.println("DNI: " + dni);
    System.out.println("Nombre: " + nombre);
    System.out.println("Edad: " + edad);
    System.out.println("-----");
}

public void saludar()
{
    System.out.println("Holaaaaaaaaaaaa!!!!");
    System.out.println("Soy " + nombre);
    System.out.println("*****");
}

public boolean esMayorDeEdad()
{
    boolean resultado = false;

    if (edad >= MAYORIA_EDAD )
    {
        resultado = true;
    }

    return resultado;
}

public String getNombre()
{
    return nombre;
}
```

# POO

## Objetos

Observa cómo podemos acceder a los atributos de una clase, consultándolos o modificándolos:



# POO

## Objetos

---

```
package poo;

public class Reunion2 {

    public static void main(String[] args) {
        Persona asistente1 = new Persona("2342424P", "Andrea", "Díaz", 17);
        Persona asistente2 = new Persona("325235K", "Lucas", "Pérez", 51);

        if (asistente1.esMayorDeEdad())
        {
            System.out.println("El asistente " + asistente1.getNombre() + " puede pasar");
        }
        else
        {
            System.out.println("El asistente " + asistente1.getNombre() + " no puede pasar");
        }

        asistente1.setEdad(22);

        System.out.println("El asistente " + asistente1.getNombre() + " ya puede pasar porque la edad es "
            + asistente1.getEdad() + "años");
    }

}
```

# POO

## Objetos

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

    final int MAYORIA_EDAD = 18;

    Persona(String docIdentidad, String nombrePersona, String apellidoPersona,
    {
        dni = docIdentidad;
        nombre = nombrePersona;
        apellido = apellidoPersona;
        edad = edadPersona;
    }

    public void mostrarInfoBasicaPersona()
    {
        System.out.println("-----");
        System.out.println("DNI: " + dni);
        System.out.println("Nombre: " + nombre);
        System.out.println("Edad: " + edad);
        System.out.println("-----");
    }

    public void saludar()
    {
        System.out.println("Holaaaaaaaa!!!!");
        System.out.println("Soy " + nombre);
        System.out.println("*****");
    }

    public boolean esMayorDeEdad()
    {
        boolean resultado = false;

        if (edad >= MAYORIA_EDAD )
        {
            resultado = true;
        }

        return resultado;
    }

    public String getNombre()
    {
        return nombre;
    }

    public void setEdad(int edadInput)
    {
        edad = edadInput;
    }

    public int getEdad()
    {
        return edad;
    }
}
```

# POO

## Objetos

**Ejercicio:** Una productora de videojuegos tiene dos videojuegos, la información asociada a cada videojuego es título, año de lanzamiento, nombre de la productora, edad mínima para jugar.

El sistema de la productora pregunta al usuario la edad que tiene, si la edad es mayor que la mínima para jugar a uno de los videojuegos mostrará la información del videojuego adecuado. Vuelve a repetir la operación hasta que el el usuario desee salir.



```

package poo;

import java.util.Scanner;

public class ProductoraJuegos {

    public static void main(String[] args) {
        String entradaUsuario = "";
        int edad = 0;
        JuegoOnLine juego1 = new JuegoOnLine("Fornite", 2012, "Epic Games", 12);
        JuegoOnLine juego2 = new JuegoOnLine("Minecraft", 2010, "Microsoft", 6);

        Scanner lectorEntrada = new Scanner(System.in);

        do {
            System.out.println("Introduce tu edad: ");
            edad = lectorEntrada.nextInt();

            if (juego1.esJuegoApto(edad)) {
                System.out.println("Tenemos un juego apto para ti: ");
                juego1.mostrarInfoJuego();
            }

            if (juego2.esJuegoApto(edad)) {
                System.out.println("Tenemos un juego apto para ti: ");
                juego2.mostrarInfoJuego();
            }

            System.out.println("¿Quieres terminar? s/n: ");
            entradaUsuario = lectorEntrada.next();
        } while ( !(entradaUsuario.equals("s")) );
        lectorEntrada.close();
    }
}

```

```

package poo;

public class JuegoOnLine {
    private String titulo;
    private int añoLanzamiento;
    private String nombreProductora;
    private int edadMinimaUsuario;

    JuegoOnLine(String tituloJuego, int añoLanzamientoJuego,
               String nombreProductoraJuego,
               int edadMinimaUsuarioJuego)
    {
        titulo = tituloJuego;
        añoLanzamiento = añoLanzamientoJuego;
        nombreProductora = nombreProductoraJuego;
        edadMinimaUsuario = edadMinimaUsuarioJuego;
    }

    public void mostrarInfoJuego()
    {
        System.out.println("-----");
        System.out.println("Título: " + titulo);
        System.out.println("Año de lanzamiento: " + añoLanzamiento);
        System.out.println("Nombre de Productora: " + nombreProductora);
        System.out.println("Nombre de Productora: " + edadMinimaUsuario);
        System.out.println("-----");
    }

    public boolean esJuegoApto(int edad)
    {
        boolean apto = false;

        if (edad > edadMinimaUsuario)
        {
            apto = true;
        }
        else
        {
            apto = false;
        }

        return apto;
    }
}

```



### Constructor:

- El nombre del constructor tiene que ser igual al de la clase.
- Puede recibir cualquier número de argumentos de cualquier tipo.
- No devuelve ningún valor (en su declaración no se declara ni siquiera void).
- El constructor sólo es invocado cuando se crea el objeto (con el operador new). No puede invocarse explícitamente en ningún otro momento.

# POO

## Objetos

---

### propiedades de la POO

---

- **Encapsulamiento.** Una clase se compone tanto de variables (propiedades) como de funciones y procedimientos (métodos). De hecho no se pueden definir variables (ni funciones) fuera de una clase (es decir no hay variables *globales*).
- **Ocultación.** Hay una zona oculta al definir la clases (zona privada) que sólo es utilizada por esa clases y por alguna clase relacionada. Hay una zona pública (llamada también **interfaz** de la clase) que puede ser utilizada por cualquier parte del código.
- **Polimorfismo.** Cada método de una clase puede tener varias definiciones distintas. En el caso del parchís: partida.empezar(4) empieza una partida para cuatro jugadores, partida.empezar(rojo, azul) empieza una partida de dos jugadores para los colores rojo y azul; estas son dos formas distintas de emplear el método empezar, que es polimórfico.
- **Herencia.** Una clase puede heredar propiedades de otra.

### Variables:

El **ámbito o alcance de una variable** es la zona del programa donde la variable es accesible.

El ámbito lo determina el lugar donde se declara la variable.

En Java las variables se pueden clasificar según su ámbito en:

- **Variables miembro** de una clase o atributos de una clase
- **Variables locales**. Son las declaradas dentro de un método
- **Variables de bloque**



# POO

## Variables:

```
public class UnaClase {  
    private int numero1;  Variable miembro o atributo de la clase  
    public void calcular() {  
        int a = 1;  Variable local del método  
        {  
            System.out.println(a + ", " + numero1);  
            int b = 2;  Variable de bloque  
            System.out.println(a + ", " + b);  
            {  
                int c = 3;  Variable de bloque  
                System.out.println(a + ", " + b + ", " + c);  
            } // Fin del ámbito de c. Final del bloque donde se ha declarado  
            System.out.println(a + ", " + b + ", " + c); // esta línea provoca un  
                                                // error de compilación.  
                                                // c está fuera de su ámbito  
                                                // y por tanto, no declarada  
        } //Fin del ámbito de b. Final del bloque donde se ha declarado  
    } // Fin del ámbito de a. Final del método calcular  
} //Fin del ámbito de número1. Final de la clase
```

# POO

## Variables:

### **Variables miembro** de una clase o atributos de una clase

Son las declaradas dentro de una clase y fuera de cualquier método.

Son accesibles en cualquier método de la clase.

Pueden ser inicializadas.

Si no se les asigna un valor inicial, el compilador les asigna uno por defecto:

- 0 para las numéricas
- '\0' para las de tipo char
- null para String y resto de referencias a objetos.



# POO

## Variables:

---

### Variables locales:

Son las declaradas dentro de un método.

Su ámbito es el método donde se declaran.

No son visibles desde otros métodos.

Distintos métodos de la clase pueden contener variables con el mismo nombre. Se trata de variables distintas.

El nombre de una variable local debe ser único dentro de su ámbito.

Si se declara una variable local con el mismo nombre que una variable miembro de la clase, la variable local oculta a la miembro. La variable miembro queda inaccesible en el ámbito de la variable local con el mismo nombre.

Se crean en memoria cuando se declaran y se destruyen cuando acaba la ejecución del método.

No tienen un valor inicial por defecto. El programador es el encargado de asignarles valores iniciales válidos.



# POO

## Variables:

---

### Variables de bloque

Son las declaradas dentro de un bloque de instrucciones delimitado por llaves {}.

Su ámbito está entre esas dos llaves.

No son visibles desde otros bloques.

Distintos bloques pueden contener variables con el mismo nombre. Se trata de variables distintas.

Si un bloque de instrucciones contiene dentro otro bloque de instrucciones, en el bloque interior no se puede declarar una variable con el mismo nombre que otra del bloque exterior.

Se crean en memoria cuando se declaran y se destruyen cuando acaba la ejecución del bloque.

No tienen un valor inicial por defecto. El programador es el encargado de asignarles valores iniciales válidos.



# POO

---

## Ejercicio

Debemos realizar un programa para gestionar un concesionario de coches. El concesionario tendrá 2 coches, cada uno con su matrícula, tamaño de batería, color, precio, indicador de si está a la venta o bien es para alquilar.

El sistema tendrá el siguiente menú para el usuario:

1. Mostrar la información de los coches
2. Cambiar la matrícula del primer coche. Se le preguntará al usuario la nueva matrícula
3. Consultar la disponibilidad(venta o alquiler) del segundo coche
4. Salir



# POO

## Variables:

```
public class Coche {
    private String matricula;
    private String color;
    private int tamanioBateria;
    private int precio;
    private String disponibilidad;

    Coche(String matriculaCoche, String colorCoche,
          int tamanioBateriaCoche, int precioCoche,
          String disponibilidadCoche)
    {
        matricula = matriculaCoche;
        color = colorCoche;
        tamanioBateria = tamanioBateriaCoche;
        precio = precioCoche;
        disponibilidad = disponibilidadCoche;
    }

    public void setMatricula(String nuevaMatricula)
    {
        matricula = nuevaMatricula;
    }
    public String getDisponibilidad()
    {
        return disponibilidad;
    }
    public void mostrarInfoCoche()
    {
        System.out.println("-----");
        System.out.println("matricula: " + matricula);
        System.out.println("color: " + color);
        System.out.println("disponibilidad: " + disponibilidad);
        System.out.println("-----");
    }
}
```

# POO

## Variables:

```
package poo;

import java.util.Scanner;

public class Concesionario {

    public static void main(String[] args) {
        String entradaUsuario = "";
        Scanner lectorEntrada = new Scanner(System.in);

        Coche coche1 = new Coche("4355-DEL", "rojo", 355, 34356, "VENTA");
        Coche coche2 = new Coche("3543-SDE", "verde", 300, 150, "ALQUILER");

        do {
            System.out.println("1. Mostrar la información de los coches\n"
                + "2. Cambiar la matrícula del coche1.\n"
                + "3. Consultar el estado (venta o alquiler) del coche 2\n"
                + "4. Salir\n");

            entradaUsuario = lectorEntrada.next();

            switch(entradaUsuario) {
                case "1":
                    coche1.mostrarInfoCoche();
                    coche2.mostrarInfoCoche();
                    break;
                case "2":
                    System.out.println("Introduzca la matrícula");
                    entradaUsuario = lectorEntrada.next();
                    coche1.setMatricula(entradaUsuario);
                    break;
                case "3":
                    System.out.println(coche2.getDisponibilidad());
                    break;
                case "4":
                    System.out.println("Adios");
                    break;
                default:
                    System.out.println("Opción errónea");
            }
        }while (!(entradaUsuario.equals("4")));

        lectorEntrada.close();
    }
}
```

# POO

---

Observa los atributos son privados. Encapsular y ocultar los atributos a otras clases es muy importante si queremos hacer nuestro código robusto, de calidad...

Pero a veces necesitamos modificarlos o consultar esos atributos. Para ello creamos getters y setters, No debemos poner los atributos públicos

# POO

---

Observa que cada una de las opciones del menú sólo tiene un par de líneas de código, si tuviese más de 3 o 4 líneas conviene hacer un método aparte que realice las acciones correspondientes. Así nuestro código queda mucho mejor organizado. Si es necesario podemos pasar parámetros a esos métodos

Estos métodos deberían ser privados ya que son para uso interno de nuestra clase concesionario y static porque se usan desde el main que es estático.

# POO

---

**Ejercicio:** Realiza una modificación del ejercicio anterior, incluye getters and setters  
(Observa que Eclipse tiene una opción para generar los getters y los setters automáticamente).

Las opciones de menú también cambian:

1. Mostrar la información de los coches
2. Cambiar la matrícula de un coche. Se le preguntará al usuario si se debe cambiar al coche 1 o al coche 2 y a continuación se le pregunta la matrícula
3. Consultar la disponibilidad(venta o alquiler) de un coche, se le pregunta al usuario si es del coche 1 o el 2
4. Mostrar el tamaño de la batería de los coches y preguntar al usuario si quiere cambiar la de alguno de ellos en cuyo caso se la pide el tamaño y se actualiza el coche correspondiente.
5. Salir



```

import java.util.Scanner;

public class ConcesionarioV2
{
    static Scanner lectorEntrada;

    public static void main(String[] args)
    {
        String entradaUsuario = "";
        lectorEntrada = new Scanner(System.in);

        CocheV2 coche1 = new CocheV2("4355-DEL", "rojo", 355, 34356, "VENTA");
        CocheV2 coche2 = new CocheV2("3543-SDE", "verde", 300, 150, "ALQUILER");

        do {
            System.out.println("1. Mostrar la información de los coches\n"
                + "2. Cambiar la matrícula del coche1.\n"
                + "3. Consultar el estado (venta o alquiler) del coche 2\n"
                + "4. Salir\n");

            entradaUsuario = lectorEntrada.next();

            switch(entradaUsuario)
            {
                case "1":
                    coche1.mostrarInfoCoche();
                    coche2.mostrarInfoCoche();
                    break;
                case "2":
                    cambiarMatricula(coche1, coche2);
                    break;
                case "3":
                    System.out.println(coche2.getDisponibilidad());
                    break;
                case "4":
                    System.out.println("Adios");
                    break;
                default:
                    System.out.println("Opción errónea");
            }
        }while (!(entradaUsuario.equals("4")));

        lectorEntrada.close();
    }
}

```

**Observa dónde se ha declarado la variable Scanner y el paso de parámetros al método**

```

private static void cambiarMatricula(CocheV2 inCoche1, CocheV2 inCoche2)
{
    int numeroCoche = 0;
    String nuevaMatricula = "";

    System.out.println("¿Qué coche quiere cambiar? 1/2");
    numeroCoche = lectorEntrada.nextInt();

    System.out.println("¿Qué matrícula desea poner?");
    nuevaMatricula = lectorEntrada.next();

    if (numeroCoche == 1) {
        inCoche1.setMatricula(nuevaMatricula);
    }
    else
    {
        inCoche2.setMatricula(nuevaMatricula);
    }
}

```

# POO

---

**Cuidado con el paso de parámetros!**

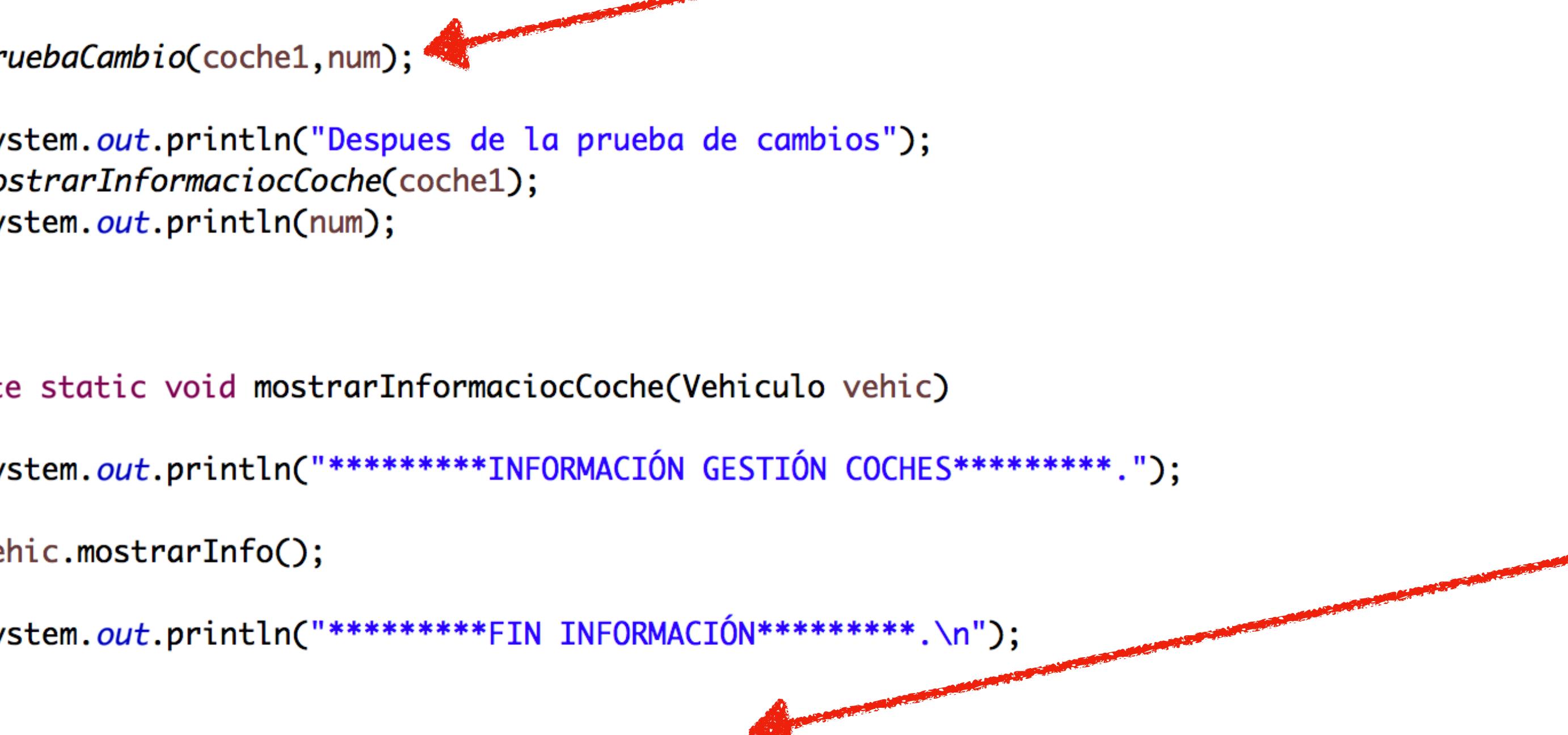
**Cuando el argumento es de tipo primitivo** (int, double, char, boolean, float, short, byte), cuando se invoca al método se reserva un nuevo espacio en memoria para el parámetro en el método. El método no puede modificar el parámetro inicial.

**Cuando el argumento es un objeto** el método recibe una copia de la dirección de memoria donde se encuentra el objeto. La referencia no puede modificarse pero sí se pueden modificar los contenidos de los objetos durante la ejecución del método.

# POO

## Cuidado con el paso de parámetros!

```
2 public class GestionCoches4 {  
3     ...  
5     public static void main(String[] args)  
6     {  
7         Vehiculo coche1 = new Vehiculo();  
8         int num = 1;  
9         coche1.setMatricula("AAA-1111");  
10        coche1.setDisponibilidad(true);  
11  
12        mostrarInformaciocCoche(coche1);  
13  
14        pruebaCambio(coche1, num);  
15  
16        System.out.println("Despues de la prueba de cambios");  
17        mostrarInformaciocCoche(coche1);  
18        System.out.println(num);  
19    }  
20  
21  
22    private static void mostrarInformaciocCoche(Vehiculo vehic)  
23    {  
24        System.out.println("*****INFORMACIÓN GESTIÓN COCHES*****.");  
25  
26        vehic.mostrarInfo();  
27  
28        System.out.println("*****FIN INFORMACIÓN*****.\n");  
29    }  
30  
31    private static void pruebaCambio(Vehiculo vehic, int numero)  
32    {  
33        vehic.setMatricula("ZZZ-9999");  
34        numero+=25;  
35  
36        System.out.println("*****Dentro de prueba cambio *****.\n");  
37        mostrarInformaciocCoche(vehic);  
38        System.out.println(numero);  
39        System.out.println("*****Fin prueba cambio *****.\n");  
40    } José Ramón Otero  
41
```



# POO

---

**Cuidado con el paso de parámetros!**

**Probadlo en vuestro ejercicio**



# POO

## Javadoc

Podemos utilizar clases que no hemos hecho nosotros, otras personas pueden usar nuestras clases. Para ello es importante que las clases estén bien documentadas.

Java dispone de una herramienta para generar la documentación de un proyecto, javadoc. Está integrada en Eclipse

```

1 package ejemplosPOO;
2
3 /**
4 * <h1>Clase Vehículo</h1>
5 * @author juanluisballesterosfernande
6 * @version 1.0
7 * Clase Vehículo: Representa vehículos genéricos
8 *
9 *
10 */
11 public class Vehiculo {
12
13     private String matricula="";
14     private String color="";
15     private int anio=0;
16     private boolean disponible = false;
17
18 /**
19 * Constructor por defecto
20 */
21 public Vehiculo()
22 {
23     matricula = "-----";
24     color = "---";
25     anio = 0;
26 }
27
28 /**
29 * Constructor a partir de los siguientes parámetros
30 * @param inMatricula. Se introduce como parámetro de entrada la
31 * matrícula del vehículo
32 * @param inColor
33 * @param inAnio
34 */
35
36 public Vehiculo(String inMatricula, String inColor, int inAnio)
37 {
38     matricula = inMatricula;
39     color = inColor;
40     anio = inAnio;
41 }
42
43 /**
44 * @param inMatricula
45 */
46
47 public void setMatricula(String inMatricula)
48 {
49     matricula = inMatricula;
50 }

```

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

**ejemplosPOO**

## Class Vehiculo

java.lang.Object  
ejemplosPOO.Vehiculo

---

public class Vehiculo  
extends java.lang.Object

### Clase Vehículo

Version:  
1.0 Clase Vehículo: Representa vehículos genéricos

Author:  
juanluisballesterosfernande

#### Constructor Summary

**Constructors**

Constructor and Description
<b>Vehiculo()</b> Constructor por defecto
<b>Vehiculo(java.lang.String inMatricula, java.lang.String inColor, int inAnio)</b> Constructor a partir de los siguientes parámetros

#### Method Summary

All Methods Instance Methods Concrete Methods

Modifier and Type	Method and Description
boolean	<b>getDisponibilidad()</b>
java.lang.String	<b>getMatricula()</b>
void	<b>mostrarInfo()</b>
void	<b>setDisponibilidad(boolean inDisponible)</b>
void	<b>setMatricula(java.lang.String inMatricula)</b>

Methods inherited from class java.lang.Object

**Ejercicio:** Documenta cada uno de los métodos del ejercicio del concesionario, genera la documentación con Javadoc y comprueba si otro programador podría usar las clases que has creado.

# POO

## Librerías

Cuando programamos, usamos clases que han programado otras personas. Esas clases suelen estar en librerías. La reutilización de código es básica en la programación.

# POO

## Enum

Hay situaciones en las que tenemos un conjunto de constantes relacionadas entre sí, que podríamos utilizar en nuestro código. Por ejemplo NORTE, SUR, ESTE, OESTE si nuestro programa trabaja con direcciones geográficas. O LUNES, MARTES, MIÉRCOLES... si nuestro programa trabaja con días de la semana.

# POO

## Enum

En este caso existe una estructura llamada enum que nos va a resultar muy útil: El enumerado

<https://www.w3schools.com/java/javaEnums.asp>

# POO

## Enum

```
public class ejemploENUM {  
    enum PuntoCardinal  
    {  
        NORTE,  
        SUR,  
        ESTE,  
        OESTE  
    }  
    public static void main(String[] args)  
    {  
  
        PuntoCardinal puntoCard = PuntoCardinal.SUR;  
  
        switch (puntoCard)  
        {  
            case NORTE:  
                System.out.println("Has elegido Norte");  
                break;  
            case SUR:  
                System.out.println("Has elegido Sur");  
                break;  
            case ESTE:  
                System.out.println("Has elegido Este");  
                break;  
            case OESTE:  
                System.out.println("Has elegido Oeste");  
                break;  
            default:  
                System.out.println("Error");  
        }  
  
        if (puntoCard == PuntoCardinal.SUR)  
        {  
            //Acciones  
        }  
    }  
}
```

# POO

## Enum

Ejercicio: Una planta química tiene 4 niveles de alerta: EXTREMO, ALTO, MODERADO, BAJO.

El nivel de alerta se decide en función de los grados de temperatura de un sensor. Extremo: mayor que 80, alto: entre 50 y 80, moderado: entre 30 y 50, bajo: menor que 30

El programa debe pedir al usuario los grados de temperatura, en caso de que el nivel de alerta sea bajo o moderado debe mostrar un mensaje de calma, si es alto debe llandar dos avisos y si es extremo 3 avisos.



# POO

## Enum

```
public class EjercicioEnum {  
  
    enum NivelAlerta  
    {  
        EXTREMO,  
        ALTO,  
        MODERADO,  
        BAJO  
    }  
    public static void main(String[] args)  
    {  
        int temperaturaPlanta = 0;  
        NivelAlerta nivelAlertaPlanta = NivelAlerta.BAJO;  
        Scanner lectorEntrada = new Scanner(System.in);  
  
        System.out.println("¿Qué temperatura hace? ");  
        temperaturaPlanta = lectorEntrada.nextInt();  
  
        if (temperaturaPlanta > 80) {  
            nivelAlertaPlanta = NivelAlerta.EXTREMO;  
            //otras acciones  
        }  
        else if ((temperaturaPlanta <= 80) && (temperaturaPlanta > 50))  
        {  
            nivelAlertaPlanta = NivelAlerta.ALTO;  
            //otras acciones  
        }  
        else  
        {  
            // etc  
        }  
  
        switch (nivelAlertaPlanta)  
        {  
            case EXTREMO:  
                System.out.println("ALARMA!!");  
                System.out.println("ALARMA!!");  
                System.out.println("ALARMA!!");  
                break;  
            case ALTO:  
                System.out.println("CUIDADÍN");  
                System.out.println("CUIDADÍN");  
                break;  
            case MODERADO:  
                System.out.println("OK");  
                break;  
            case BAJO:  
                System.out.println("OK");  
                break;  
            default:  
                System.out.println("Error");  
        }  
        lectorEntrada.close();  
    }  
}
```

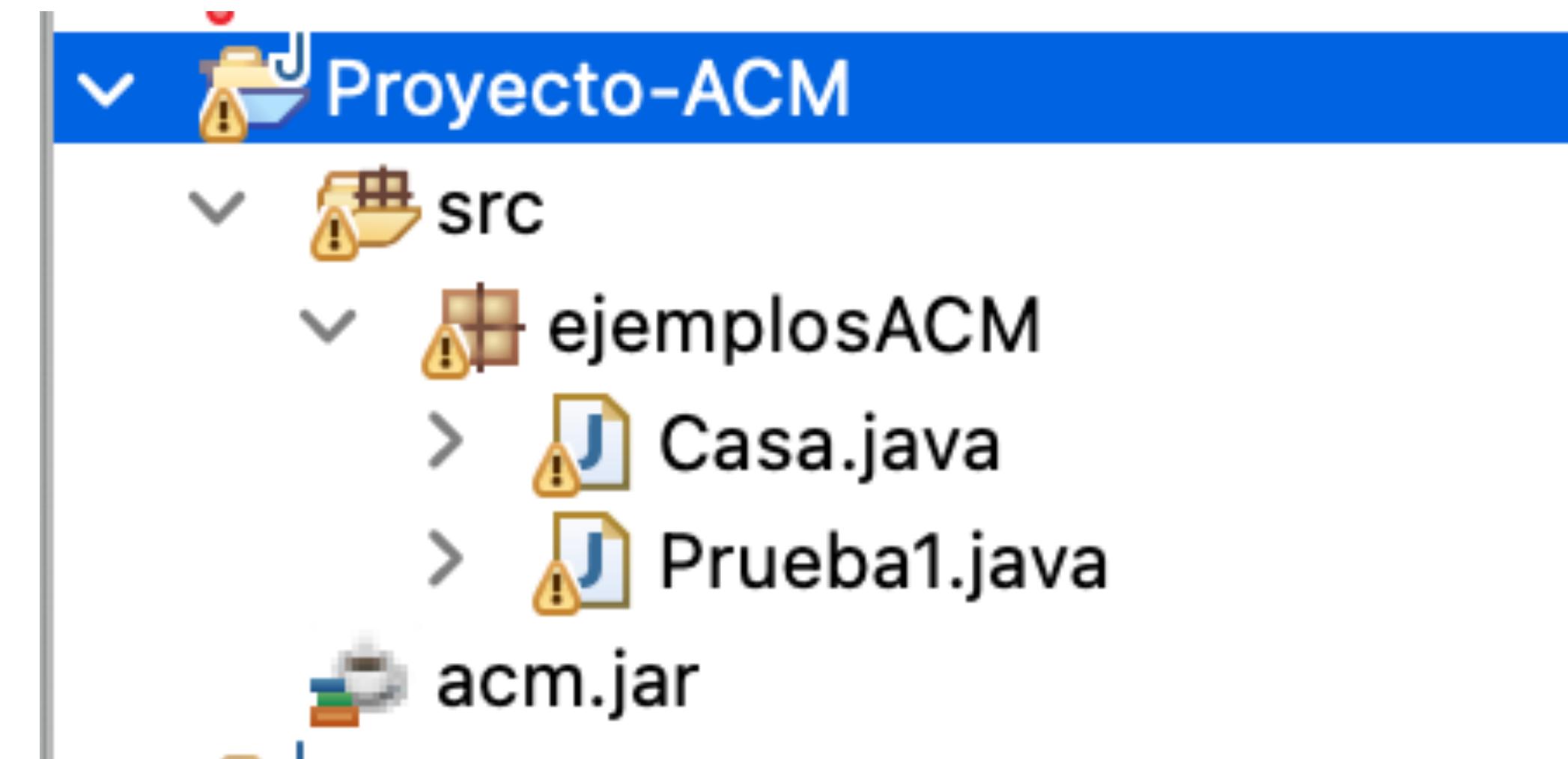
Vamos a usar una librería que nos ayudará a aprender mejor la POO: ACM Library

A partir de la documentación y del .jar de esa librería podremos construir aplicaciones gráficas sencillas

Vamos a crear un proyecto que use la librería ACM.

Documentación [ACM library](#)

Llevamos el .jar a nuestro proyecto:



En las propiedades del proyecto la incluimos en el Classpath



# POO

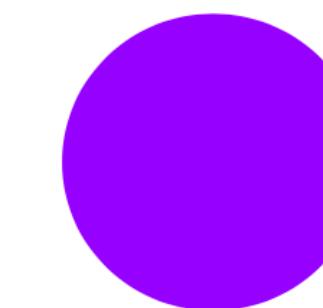
## Librerías

Esta librería tiene Clases que representan objetos gráficos que podemos usar en nuestro código creando objetos.  
Puedes ver las clases y sus métodos en la documentación:  
[ACM library](#)

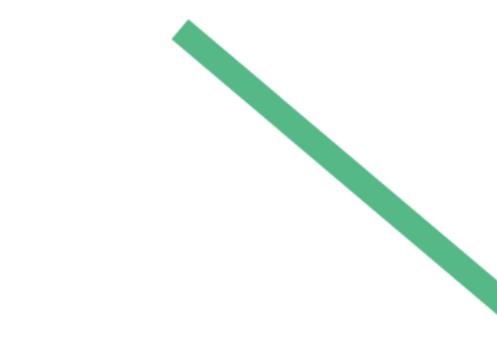
GRect



GOval



GLine



GLabel

Hello

GImage



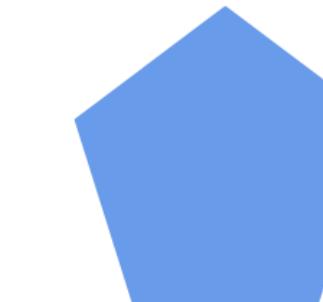
GRoundRect



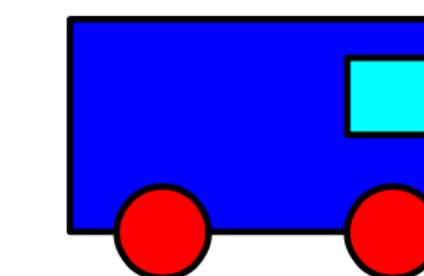
GArc



GPolygon



GCompound



and  
more  
...

Practicaremos la POO realizando programas gráficos con esta librería. Crearemos objetos y los manipularemos mediante los métodos públicos de cada clase.

# POO

## Librerías

```
package ejemplosACM;

import acm.program.*;
import acm.graphics.*; // Stanford graphical objects import java.awt.*;
import java.awt.*;

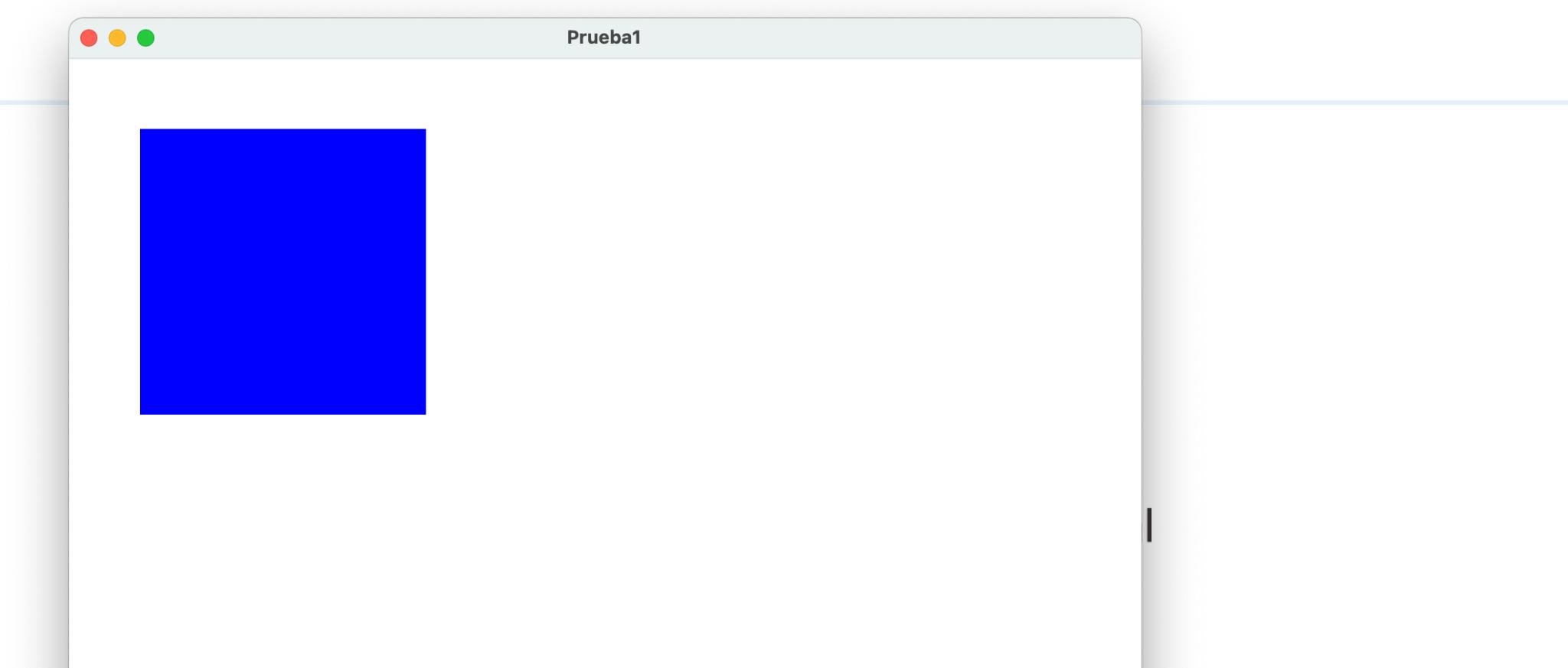
public class Prueba1 extends GraphicsProgram{

    public void run()
    {
        GRect rect = new GRect(200, 200);
        rect.setFilled(true);
        rect.setColor(Color.BLUE);
        add(rect, 50, 50);
    }

}
```

BLACK	BLUE	CYAN	DARK_GRAY	GRAY
GREEN	LIGHT_GRAY	MAGENTA	ORANGE	PINK
RED	WHITE	YELLOW		

Enumerado



Observa que hemos usado métodos públicos de la clase `GRect`, busca los métodos disponibles de esa clases en la documentación.

## Algunos métodos útiles de la clase GraphicsProgram

Method	Description
<code>add(<i>gobj</i>);</code> <code>add(<i>gobj</i>, <i>x</i>, <i>y</i>);</code>	adds a graphical object to the window
<code>getElementAt(<i>x</i>, <i>y</i>)</code>	return the object at the given ( <i>x,y</i> ) position(s)
<code>getElementCount()</code>	return number of graphical objects onscreen
<code>getWidth(), getHeight()</code>	return dimensions of window
<code>remove(<i>gobj</i>);</code>	removes a graphical object from the window
<code>removeAll();</code>	remove all graphical objects from window
<code>: setSize(<i>w</i>, <i>h</i>);</code>	set size of drawing area
<code>setBackground(<i>color</i>);</code>	set window's background color

**Ejercicio.** Realiza un programa que pinte un óvalo en una ventana. El tamaño de la ventana es 900x400. El ancho, el alto y el color del óvalo se piden al usuario por consola

# POO

## Librerías

```
public class Prueba2 extends GraphicsProgram{

    public void run()
    {
        final int ANCHO_VENTANA = 900;
        final int ALTO_VENTANA = 400;
        GOval ovalo1;
        int anchoOvalo = 0;
        int altoOvalo = 0;
        String colorEntrada = "";
        Color colorOvalo = Color.BLACK;
        Scanner lectorEntrada = new Scanner(System.in);

        setSize(ANCHO_VENTANA,ALTO_VENTANA);
        setBackground(Color.RED);

        System.out.println("Qué ancho quiere para el óvalo?");
        anchoOvalo = lectorEntrada.nextInt();
        System.out.println("Qué alto quiere para el óvalo?");
        altoOvalo = lectorEntrada.nextInt();

        ovalo1 = new GOval(anchoOvalo,altoOvalo);

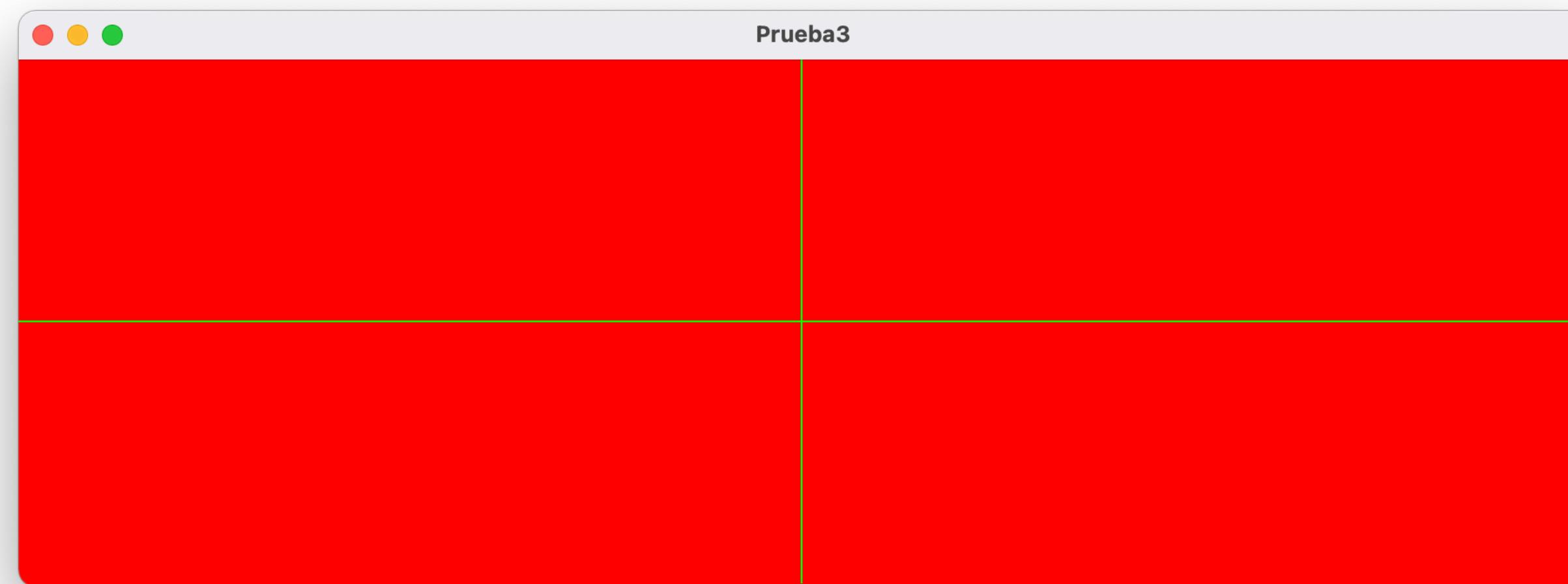
        System.out.println("Qué color quiere para el óvalo?");
        colorEntrada = lectorEntrada.next();

        switch (colorEntrada)
        {
            case "rojo":
                colorOvalo = Color.RED;
                break;
            case "azul":
                colorOvalo = Color.BLUE;
                break;
            case "verde":
                colorOvalo = Color.GREEN;
                break;
            case "naranja":
                colorOvalo = Color.ORANGE;
                break;
            default:
                System.out.println("Color erróneo");
        }

        ovalo1.setFilled(true);
        ovalo1.setFillColor(colorOvalo);
        add(ovalo1, 50, 50);

        lectorEntrada.close();
    }
}
```

**Ejercicio.** Realiza un programa que pida el usuario el tamaño de la ventana y dibuje dos líneas en el centro como en la figura de abajo



# POO

## Librerías

```
1 package ejemplosACM;
2
3+import acm.program.*;
4
5
6
7
8
9 public class Prueba3 extends GraphicsProgram{
10
11+    public void run()
12    {
13        int anchoVentana = 0;
14        int altoVentana = 0;
15        GLine linea1, linea2;
16        Scanner lectorEntrada = new Scanner(System.in);
17
18        System.out.println("Qué ancho quiere para la pantalla?");
19        anchoVentana = lectorEntrada.nextInt();
20        System.out.println("Qué alto quiere para la pantalla?");
21        altoVentana = lectorEntrada.nextInt();
22
23        setSize(anchoVentana,altoVentana);
24        setBackground(Color.RED);
25
26        linea1 = new GLine(anchoVentana/2,0,anchoVentana/2,altoVentana);
27        linea1.setColor(Color.GREEN);
28
29        add(linea1);
30
31        linea2 = new GLine(0,altoVentana/2,anchoVentana,altoVentana/2);
32        linea2.setColor(Color.GREEN);
33        add(linea2);
34
35        lectorEntrada.close();
36
37    }
38
39
40 }
```

Para mover un objeto basta con modificar sus coordenadas X e Y. Para ello las clases tienen un método move(dx,dy) que desplaza en el eje X dx pixels, lo mismo para el eje Y

[https://cs.stanford.edu/people/eroberts/jtf/javadoc/  
student/](https://cs.stanford.edu/people/eroberts/jtf/javadoc/student/)

**Ejemplo.** El siguiente programa dibuja una bola en la parte superior de la pantalla y se va moviendo hasta llegar hasta la parte de abajo, en ese momento debe pararse.

Para que se mueva lentamente podemos parar la ejecución durante 20 milisegundos en cada movimiento con la sentencia `pause(numMiliseg)` (método de `acm`)

# POO

## Librerías

```
package ejemplosACM;

import acm.program.*;
import acm.graphics.*; // Stanford graphical objects import java.awt.*;
import java.awt.*;
import java.lang.Thread;

public class BolaEnMovimiento extends GraphicsProgram {
    private static final int TIEMPO_PARADA = 20;
    private static final int ANCHO_VENTANA = 600;
    private static final int ALTO_VENTANA = 600;
    private static final double ALTO_BOLA = 20;
    private static final double ANCHO_BOLA = 20;
    private static final int POS_X_INICIO = 200;
    private static final int POS_X_FIN = 0;

    private static final int DIFERENCIAL_Y = 2;

    private Goval bola;

    public void run()
    {
        bola = new Goval(ALTO_BOLA, ANCHO_BOLA);

        setSize(ANCHO_VENTANA,ALTO_VENTANA);
        setBackground(Color.RED);

        bola.setFilled(true);
        bola.setColor(Color.BLUE);
        add(bola,POS_X_INICIO,POS_X_FIN);

        while(bola.getY() < (ALTO_VENTANA - (2*ALTO_BOLA)))
        {
            avanzarUnFotograma();
            pause(TIEMPO_PARADA);
        }
    }

    private void avanzarUnFotograma()
    {
        bola.move(0, DIFERENCIAL_Y);
    }
}
```

**Ejercicio.** Realiza un programa que dibuja una bola en la parte superior de la pantalla y se va moviendo hasta llegar hasta la parte de abajo, en ese momento rebota y vuelve a subir, cuando llegue a la parte superior debe volver a bajar. Así sucesivamente durante 2 min, después se para

Para que se mueva lentamente podemos parar la ejecución durante 20 milisegundos en cada movimiento con la sentencia `pause(numMiliseg)` (método de `acm`)

# POO

## Librerías

```
package ejemplosACM;

import acm.program.*;
import acm.graphics.*; // Stanford graphical objects import java.awt.*;
import java.awt.*;
import java.lang.Thread;

public class BolaEnMovimiento2 extends GraphicsProgram {
    private static final int TIEMPO_PARADA = 20;
    private static final int TIEMPO_MAXIMO = 120000; //2 minutos
    private static final int ANCHO_VENTANA = 600;
    private static final int ALTO_VENTANA = 400;
    private static final double ALTO_BOLA = 20;
    private static final double ANCHO_BOLA = 20;
    private static final int POS_X_INICIO = 200;
    private static final int POS_X_FIN = 0;

    private static final int DIFERENCIAL_Y = 2;

    private G0val bola;
    private int diferencialY;

    public void run()
    {
        int tiempo = 0;
        bola = new G0val(ALTO_BOLA, ANCHO_BOLA);

        setSize(ANCHO_VENTANA,ALTO_VENTANA);
        setBackground(Color.RED);

        bola.setFilled(true);
        bola.setColor(Color.BLUE);
        add(bola,POS_X_INICIO,POS_X_FIN);

        while(tiempo < TIEMPO_MAXIMO)
        {
            avanzarUnFotograma();
            pause(TIEMPO_PARADA);
            tiempo += TIEMPO_PARADA;
        }
    }

    private void avanzarUnFotograma()
    {
        if (bola.getY() > (ALTO_VENTANA - (2*ALTO_BOLA)))
        {
            diferencialY = - DIFERENCIAL_Y;
        }
        if (bola.getY() == 0)
        {
            diferencialY = DIFERENCIAL_Y;
        }
        bola.move(0, diferencialY);
    }
}
```



**setLocation(X,Y)** permite posicionar un elemento en las coordenadas indicadas, revisalo en la documentación.

Ejemplo: El siguiente programa modifica la posición de un círculo cuando pasan 2 segundos

# POO

## Librerías

```
package ejemplosACM;

import acm.program.*;
import acm.graphics.*; // Stanford graphical objects import java.awt.*;
import java.awt.*;

public class EjemploSetLocation extends GraphicsProgram {

    private static final int ANCHO_VENTANA = 600;
    private static final int ALTO_VENTANA = 600;
    private static int TIEMPO_PARADA = 2000;

    public void run()
    {
        G0val circulo1 = new G0val(100, 100);

        setSize(ANCHO_VENTANA,ALTO_VENTANA);
        setBackground(Color.RED);

        circulo1.setFilled(true);
        circulo1.setColor(Color.BLUE);
        add(circulo1,0,0);

        pause(TIEMPO_PARADA);

        circulo1.setLocation(ANCHO_VENTANA/2,ALTO_VENTANA/2);
    }
}
```

**Ejercicio.** Realiza un programa que dibuje 3 círculos de distintos colores que cambien de posición cada medio segundo durante 2 minutos. La nueva posición es aleatoria dentro de la ventana.

# POO

## Librerías

```
public void run()
{
    final double TIEMPO_JUEGO = 10000;

    double coordPosX = 0;
    double coordPosY = 0;
    double contadorTiempo = 0;

    G0val circulo1 = new G0val(100, 100);
    G0val circulo2 = new G0val(100, 100);
    G0val circulo3 = new G0val(100, 100);

    setSize(ANCHO_VENTANA,ALTO_VENTANA);
    setBackground(Color.RED);

    circulo1.setFilled(true);
    circulo1.setColor(Color.BLUE);
    add(circulo1,100,100);

    circulo2.setFilled(true);
    circulo2.setColor(Color.GREEN);
    add(circulo2,400,100);

    circulo3.setFilled(true);
    circulo3.setColor(Color.YELLOW);
    add(circulo3,200,200);

    while (contadorTiempo < TIEMPO_JUEGO)
    {
        moverCirculo(circulo1);
        moverCirculo(circulo2);
        moverCirculo(circulo3);

        pause(TIEMPO_PARADA);

        contadorTiempo += 500;
    }
}
```

```
private static void moverCirculo(G0val circulo)
{
    double coordPosX = Math.random() * ANCHO_VENTANA;
    double coordPosY = Math.random() * ALTO_VENTANA;

    circulo.setLocation(coordPosX, coordPosY);
}
```

# POO

## Garbage collector

Algunos lenguajes orientados a objetos necesitan que vigilemos todos los objetos que creamos y que los destruyamos explícitamente cuando ya no sean necesarios. Java permite crear los objetos que queramos. La máquina virtual Java elimina los objetos cuando ya no están siendo utilizados. Este proceso se denomina **recolección de basura** (garbage collector).

# POO

## Constructor por defecto:

Si para una clase no definimos ningún método constructor Java crea uno automáticamente por defecto.

El **constructor por defecto** es un constructor sin parámetros que no hace nada. Los atributos del objeto son iniciados con los valores predeterminados por el sistema.

# POO

## **Constructor de copia:**

Se puede crear un objeto a partir de otro de su misma clase escribiendo un constructor llamado constructor copia.

Este constructor copia los atributos de un objeto existente al objeto que se está creando.

Los constructores copia tiene un solo argumento, el cual es una referencia a un objeto de la misma clase que será desde el que queremos copiar.



# POO

## Constructor de copia:

```
package poo;

public class ConcesionarioV3 {
    public static void main(String[] args) {
        CocheV3 coche1 = new CocheV3("4355-DEL", "rojo", 355, 34356, "VENTA");
        CocheV3 cocheCopia = new CocheV3(coche1);

        coche1.mostrarInfoCoche();
        cocheCopia.mostrarInfoCoche();
    }
}
```

```
matricula: 4355-DEL
color: rojo
disponibilidad: VENTA
```

```
matricula: 4355-DEL
color: rojo
disponibilidad: VENTA
```



```
public class CocheV3 {
    private String matricula;
    private String color;
    private int tamanioBateria;
    private int precio;
    private String disponibilidad;

    CocheV3(String matriculaCoche, String colorCoche,
            int tamanioBateriaCoche, int precioCoche,
            String disponibilidadCoche)
    {
        matricula = matriculaCoche;
        color = colorCoche;
        tamanioBateria = tamanioBateriaCoche;
        precio = precioCoche;
        disponibilidad = disponibilidadCoche;
    }

    CocheV3(final CocheV3 cocheEntrada)
    {
        matricula = cocheEntrada.matricula;
        color = cocheEntrada.color;
        tamanioBateria = cocheEntrada.tamanioBateria;
        precio = cocheEntrada.precio;
        disponibilidad = cocheEntrada.disponibilidad;
    }

    public void setMatricula(String nuevaMatricula)
    {
        matricula = nuevaMatricula;
    }
    public String getDisponibilidad()
    {
        return disponibilidad;
    }
    public void mostrarInfoCoche()
    {
        System.out.println("-----");
        System.out.println("matricula: " + matricula);
        System.out.println("color: " + color);
        System.out.println("disponibilidad: " + disponibilidad);
        System.out.println("-----");
    }
}
```

**final para evitar que se modifique accidentalmente**

# POO

## **Constructor de copia:**

**Ejercicio:** Recupera el ejercicio que gestionaba juegos online, implementa el constructor de copia en la clase JuegoOnline y utilízalo desde la clase principal copiando un videojuego y comprobando que son iguales



# POO

---

## Ejercicio. Biblioteca:

Crea una clase llamada Libro que guarde la información de cada uno de los libros de una biblioteca. La clase debe guardar el título del libro, autor, número de ejemplares del libro y número de ejemplares prestados. La clase contendrá los siguientes métodos:

Constructor sin parámetros, constructor con parámetros, constructor de copia

Métodos Setters/getters

Método préstamo que incremente el atributo correspondiente cada vez que se realice un préstamo del libro. No se podrán prestar libros de los que no queden ejemplares disponibles para prestar. Devuelve true si se ha podido realizar la operación y false en caso contrario.

Método devolución que decremente el atributo correspondiente cuando se produzca la devolución de un libro. No se podrán devolver libros que no se hayan prestado. Devuelve true si se ha podido realizar la operación y false en caso contrario.

Habrá una clase principal llamada Biblioteca que contendrá el método main(). Desde este método se crearán varios libros, pidiendo datos al usuario, utilizando el constructor sin parámetros, el constructor con parámetros y el constructor de copia. Para los objetos creados con el constructor sin parámetros se modificarán los atributos utilizando métodos set.

Se crearán métodos auxiliares a main() en la clase biblioteca que permitan prestar y devolver libros (pasados como parámetros) y que informen si se ha realizado o no la operación. Se creará otro método auxiliar que muestre toda la información de un libro (pasado como parámetro) utilizando los métodos get de la clase Libro.

```

package poo;

import java.util.Scanner;

public class GestionBiblioteca {

    public static void main(String[] args) {
        Scanner lectorEntrada = new Scanner(System.in);

        String titulo, autor;
        int ejemplares;

        Libro libro3;
        Libro libro1 = new Libro("El quijote", "Cervantes", 3, 0);
        Libro libro2 = new Libro();

        System.out.print("Introduce titulo: ");
        titulo = lectorEntrada.next();
        System.out.print("Introduce autor: ");
        autor = lectorEntrada.next();
        System.out.print("Numero de ejemplares: ");
        ejemplares = lectorEntrada.nextInt();

        libro2.setTitulo(titulo);
        libro2.setAutor(autor);
        libro2.setEjemplares(ejemplares);

        libro3 = new Libro(libro1); //Llamamos al constructor de co
        mostrarDatosLibro(libro1);
        mostrarDatosLibro(libro3);

        prestarLibro(libro1);
        prestarLibro(libro2);

        mostrarDatosLibro(libro1);
        mostrarDatosLibro(libro2);

        devolverLibro(libro1);
        mostrarDatosLibro(libro1);

        lectorEntrada.close();
    }

    /**
     * Se realiza un préstamo de libro. Se controla que queden ejemplares.
     *
     * @param libro
     */
    private static void prestarLibro(Libro libro) {
        // se realiza otro préstamo de libro
        if (libro.prestarLibro()) {
            System.out.println("Se ha prestado el libro " + libro.getTitulo());
        } else {
            System.out.println("No quedan ejemplares del libro " + libro.getTitulo() + " p");
        }
    }

    /**
     * Devuelve un libro. Se controla el posible error
     * @param libro
     */
    private static void devolverLibro(Libro libro) {
        if (libro.devolverLibro()) {
            System.out.println("Se ha devuelto el libro " + libro.getTitulo());
        } else {
            System.out.println("No hay ejemplares del libro " + libro.getTitulo() + " pres");
        }
    }

    private static void mostrarDatosLibro(Libro libro) {
        System.out.println("Libro:");
        System.out.println("Titulo: " + libro.getTitulo());
        System.out.println("Autor: " + libro.getAutor());
        System.out.println("Ejemplares: " + libro.getEjemplares());
        System.out.println("Prestados: " + libro.getPrestados());
        System.out.println();
    }
}

```

Sn Profesional  
nón Otero

```
public class Libro {
    private String titulo;
    private String autor;
    private int numEjemplares;
    private int numPrestados;

    //Constructor sin parámetros
    public Libro() {
        titulo = "----";
        autor = "----";
        numEjemplares = 0;
        numPrestados = 0;
    }

    public Libro(String inTitulo, String inAutor, int inEjemplares,
                int inPrestados) {
        titulo = inTitulo;
        autor = inAutor;
        numEjemplares = inEjemplares;
        numPrestados = inPrestados;
    }

    //Constructor de copia
    public Libro(final Libro libroEntrada) {
        titulo = libroEntrada.titulo;
        autor = libroEntrada.autor;
        numEjemplares = libroEntrada.numEjemplares;
        numPrestados = libroEntrada.numPrestados;
    }

    //getters y setters
    public String getAutor() {
        return autor;
    }

    public void setAutor(String inAutor) {
        autor = inAutor;
    }

    public int getEjemplares() {
        return numEjemplares;
    }

    public void setEjemplares(int inEjemplares) {
        numEjemplares = inEjemplares;
    }

    public int getPrestados() {
        return numPrestados;
    }

    public void setPrestados(int inPrestados) {
        numPrestados = inPrestados;
    }

    public String getTitulo() {
        return titulo;
    }

    public void setTitulo(String titulo) {
        this.titulo = titulo;
    }

    //método para realizar el préstamo de un libro
    public boolean prestarLibro() {
        boolean prestado = true;
        if (numPrestados < numEjemplares) {
            numPrestados++;
        } else {
            prestado = false;
        }
        return prestado;
    }

    //método para realizar la devolución de un libro
    public boolean devolverLibro() {
        boolean devuelto = true;
        if (numPrestados == 0) {
            devuelto = false;
        } else {
            numPrestados--;
        }
        return devuelto;
    }
}
```

# POO

---

## this

Es una palabra reservada en java que sirve para referenciarse el objeto a sí mismo. Casos en los que se usa:



# POO

---

this

**Caso1:** Cuando hay ambigüedad a la hora de trabajar con variables:

```
public class Persona {  
  
    private String nombre;  
    private String apellido;  
  
    /**  
     * Carga el nombre y apellido de la persona.  
     * @param nombre  
     * @param apellido  
     */  
    public void cargarNombreApellido(String nombre, String apellido){  
        this.nombre = nombre;  
        this.apellido = apellido;  
    }  
}
```

# POO

---

## this

**Caso2:** Cuando llamamos internamente al constructor(debe ser la primera instrucción):

```
package ejemplosPOO;

public class Rectangulo {
    private int x, y;
    private int ancho, alto;

    public Rectangulo() {
        this(0, 0, 1, 1);
    }
    public Rectangulo(int ancho, int alto) {
        this(0, 0, ancho, alto);
    }
    public Rectangulo(int x, int y, int ancho, int alto) {
        this.x = x;
        this.y = y;
        this.ancho = ancho;
        this.alto = alto;
    }
}
```

# POO

---

**Ejercicio:** Modifica el ejercicio de la biblioteca para usar `this` de las dos formas que hemos visto.

A partir de ahora lo usaremos en los constructores

# POO

---

Vamos a practicar la creación de clases usando la librería ACM.



# POO

---

En el siguiente **ejemplo** creamos la clase marioneta con un rectángulo y un óvalo



# POO

---

```
public class Marioneta extends GCompound
{
    private Goval cabeza;
    private GRect cuerpo;

    private static double FACTOR_AJUSTE_CABEZA = 0.4;
    private static double FACTOR_AJUSTE CUERPO = 0.6;

    Marioneta(double ancho, double alto)
    {
        cabeza = new Goval(ancho*FACTOR_AJUSTE_CABEZA, alto*FACTOR_AJUSTE_CABEZA);
        cuerpo = new GRect(ancho*FACTOR_AJUSTE_CUERPO, alto*FACTOR_AJUSTE_CUERPO);

        add(cabeza, ((FACTOR_AJUSTE_CABEZA*ancho/4)), 0);
        add(cuerpo, 0, (0+(alto*FACTOR_AJUSTE_CABEZA)));
    }
}
```



# POO

---

## Ejercicio:

Crea dos marionetas u otro objeto que te interese.

Queremos las marionetas se puedan vestir, para ello crearemos un método vestir que reciba como parámetro un color y coloree el cuerpo de la marioneta

También queremos que se pueda pintar la cara de las marionetas, para ello crearemos un método pintarCabeza que reciba como parámetro un color y coloree la cabeza de la marioneta.

Realiza una clase Escenario que contenga el método principal run()

Debe crear dos marionetas, vestirlas, pintarles la cara y hacer que den saltos.



# POO

```
import acm.graphics.*; // Stanford graphical objects import java.awt.*;  
import java.awt.*;
```

```
public class Marioneta extends GCompound  
{  
    private Goval cabeza;  
    private GRect cuerpo;  
  
    private static double FACTOR_AJUSTE_CABEZA = 0.4;  
    private static double FACTOR_AJUSTE CUERPO = 0.6;  
  
    Marioneta(double ancho, double alto)  
    {  
        cabeza = new Goval(ancho*FACTOR_AJUSTE_CABEZA, alto*FACTOR_AJUSTE_CABEZA);  
        cuerpo = new GRect(ancho*FACTOR_AJUSTE_CUERPO, alto*FACTOR_AJUSTE_CUERPO);  
  
        add(cabeza, ((FACTOR_AJUSTE_CABEZA*ancho/4)), 0);  
        add(cuerpo, 0, (0+(alto*FACTOR_AJUSTE_CABEZA)));  
    }  
    public void vestir(Color color)  
    {  
        cuerpo.setFilled(true);  
        cuerpo.setColor(color);  
    }  
    public void pintarCabeza(Color color)  
    {  
        cabeza.setFilled(true);  
        cabeza.setColor(color);  
    }  
}
```

# POO

```
package ejemplosACM;

import java.awt.Color;
import acm.graphics.*; // Stanford graphical objects :
import acm.program.GraphicsProgram;

public class Escenario extends GraphicsProgram{

    private static final int ANCHO_VENTANA = 600;
    private static final int ALTO_VENTANA = 400;
    private static final int TIEMPO_PARADA = 20;
    private static final int ALTURA_SALTO = 100;

    Marioneta munieco1 = null;
    Marioneta munieco2 = null;

    public void run() {

        munieco1 = new Marioneta(100,200);
        munieco2 = new Marioneta(100,200);

        setSize(ANCHO_VENTANA,ALTO_VENTANA);
        setBackground(Color.RED);

        munieco1.vestir(Color.BLUE);
        munieco2.vestir(Color.GREEN);

        add(munieco1,100,100);
        munieco1.pintarCabeza(Color.YELLOW);
        add(munieco2,200,120);
        munieco2.pintarCabeza(Color.MAGENTA);

        moverMuniecos();
    }

    private void moverMuniecos()
    {
        for (int i = 0; i < ALTURA_SALTO; i++)
        {
            avanzarUnFotograma(0,-1);
            pause(TIEMPO_PARADA);
        }

        for (int i=0; i <ALTURA_SALTO; i++)
        {
            avanzarUnFotograma(0,1);
            pause(TIEMPO_PARADA);
        }
    }

    private void avanzarUnFotograma( int dX,int dY)
    {
        munieco1.move(dX, dY);
        munieco2.move(dX, dY);
    }
}
```



# POO

---

## Práctica. Gestión de un banco

Vamos a realizar la gestión de un banco. Para ello escribe una clase Cuenta para representar una cuenta bancaria.

Los datos de la cuenta son: nombre del cliente (String), número de cuenta (String), tipo de interés (double) y saldo (double).

La clase contendrá los siguientes métodos:

Constructor sin parámetros

Constructor con todos los parámetros

Constructor con los parámetros nombre y número de cuenta que llame internamente al constructor de 4 parámetros

Constructor copia.

Métodos setters/getters para asignar y obtener los datos de la cuenta.

Métodos ingreso y reintegro. Un ingreso consiste en aumentar el saldo en la cantidad que se indique. Esa cantidad no puede ser negativa. Un reintegro consiste en disminuir el saldo en una cantidad pero antes se debe comprobar que hay saldo suficiente. La cantidad no puede ser negativa. Los métodos ingreso y reintegro devuelven true si la operación se ha podido realizar o false en caso contrario.

Método transferencia que permita pasar dinero de una cuenta a otra siempre que en la cuenta de origen haya dinero suficiente para poder hacerla. Ejemplo de uso del método transferencia:

```
cuentaOrigen.transferencia(cuentaDestino, importe);
```

que indica que queremos hacer una transferencia desde cuentaOrigen a cuentaDestino del importe indicado.

Realiza una clase Banco que tendrá el método main y que creará una cuenta bancaria pidiendo los datos al usuario. Creará otras cuentas mediante un constructor sin parámetros, mediante un constructor con parámetros y mediante un constructor de copia. Habrá un menú que permitirá mostrar los datos de las cuentas y el saldo de las mismas, realizar transferencias, ingresos, reintegros...

```
private static Cuenta cuentaCorriente1 = null;
private static Cuenta cuentaCorriente2 = null;
private static Cuenta cuentaCorriente3 = null;
```

```
public static void main(String[] args) {
    // TODO Auto-generated method stub
```

```
cuentaCorriente1 = new Cuenta("Ana", "IWT-98493859", 2.3, 20000);
cuentaCorriente2 = new Cuenta("Pepe", "RET-04806840");
cuentaCorriente3 = new Cuenta(cuentaCorriente1);
```

```
cuentaCorriente3.setNombre("Andrea");
cuentaCorriente3.setNumeroCuenta("YTY-04694096");
```

```
hacerIngreso(cuentaCorriente2, 3000);
sacarDinero(cuentaCorriente1, 5000);
```

```
hacerTransferencia(cuentaCorriente1, cuentaCorriente2, 2000);
```

```
mostrarInfoCuenta(cuentaCorriente1);
mostrarInfoCuenta(cuentaCorriente2);
mostrarInfoCuenta(cuentaCorriente3);
```

}

```
private static void mostrarInfoCuenta(Cuenta cuenta)
{
```

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

```
    System.out.println("Nombre: " + cuenta.getNombre());
```

```
    System.out.println("Número de cuenta: " + cuenta.getNumeroCuenta());
```

```
    System.out.println("Nombre: " + cuenta.getTipoInteres());
```

```
    System.out.println("Nombre: " + cuenta.getSaldo());
```

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

}

```
private static void hacerTransferencia(Cuenta cuentaOrigen, Cuenta cuentaDestino, double importe)
{
```

```
    if (cuentaOrigen.hacerTransferencia(cuentaDestino, importe))
```

```
    {
        System.out.println("Transferencia realizada");
    }
```

```
    else
```

```
    {
        System.out.println("Error en la transferencia");
    }
```

}



```
private static void hacerIngreso(Cuenta cuentaDestino, double importe)
{
    if (cuentaDestino.ingresar(importe))
    {
        System.out.println("Ingreso realizado");
    }
    else
    {
        System.out.println("No se ha podido realizar el ingreso");
    }
}

private static void sacarDinero(Cuenta cuentaDestino, double importe)
{
    if (cuentaDestino.hacerReintegro(importe))
    {
        System.out.println("Ingreso realizado");
    }
    else
    {
        System.out.println("No se ha podido realizar el ingreso");
    }
}
```

```

package poo;

public class Cuenta {

    private String nombre;
    private String numeroCuenta;
    private double tipoInteres;
    private double saldo;

    //Constructor sin parámetros
    public Cuenta() {
        this.nombre = "--";
        this.numeroCuenta = "00000";
    }

    //Constructor con parámetros
    public Cuenta(String inNombre, String inNumeroCuenta) {
        this(inNombre,inNumeroCuenta,0,0);
    }

    //Constructor con parámetros
    public Cuenta(String inNombre, String inNumeroCuenta,
                  double inTipoInteres, double inSaldo) {
        this.nombre = inNombre;
        this.numeroCuenta = inNumeroCuenta;
        this.tipoInteres = inTipoInteres;
        this.saldo = inSaldo;
    }

    //Constructor copia
    public Cuenta(final Cuenta inCuenta) {
        this.nombre = inCuenta.nombre;
        this.numeroCuenta = inCuenta.numeroCuenta;
        this.tipoInteres = inCuenta.tipoInteres;
        this.saldo = inCuenta.saldo;
    }

    //getters y setters
    public void setNombre(String inNombre) {
        this.nombre = inNombre;
    }

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

```

```

public void setTipoInteres(double inTipoInteres) {
    this.tipoInteres = inTipoInteres;
}

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

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

public String getNumeroCuenta() {
    return this.numeroCuenta;
}

public double getTipoInteres() {
    return this.tipoInteres;
}

public double getSaldo() {
    return this.saldo;
}

//método ingreso
public boolean ingresar(double cantidad) {
    boolean ingresoCorrecto = true;

    if (cantidad < 0) {
        ingresoCorrecto = false;
    } else {
        this.saldo = this.saldo + cantidad;
    }

    return ingresoCorrecto;
}

```

```
//método reintegro
public boolean hacerReintegro(double cantidad) {
    boolean reintegroCorrecto = true;

    if (cantidad < 0)
    {
        reintegroCorrecto = false;
    } else if (this.saldo >= cantidad)
    {
        this.saldo -= cantidad;
    }
    else
    {
        reintegroCorrecto = false;
    }

    return reintegroCorrecto;
}

//método transferencia
public boolean hacerTransferencia(Cuenta cuentaDestino, double inDinero)
{
    boolean correcto = true;

    if (inDinero < 0) {
        correcto = false;
    } else if (this.saldo >= inDinero)
    {
        hacerReintegro(inDinero);
        cuentaDestino.ingresar(inDinero);
    } else {
        correcto = false;
    }

    return correcto;
}
```

**Ejercicio.** Realiza un programa que dibuje un círculo que se mueva durante 3 minutos siguiendo la siguiente función senoidal:

$$y = 100 + (30 * \text{seno}(x/40))$$



# POO

## Librerías

```
package ejemplosACM;

import acm.program.*;
import acm.graphics.*; // Stanford graphical objects import java.awt.*;
import java.awt.*;
import java.lang.Thread;

public class MoverSiguiendoFuncion extends GraphicsProgram {

    private static final int ANCHO_VENTANA = 900;
    private static final int ALTO_VENTANA = 600;
    private static final int TIEMPO_PARADA = 30;
    private static final int COORD_X_CIRCULO = 50;
    private static final int COORD_Y_CIRCULO = 50;
    private static final int DIAMETRO_CIRCULO = 40;

    double coordenadaXCirculo = 0;
    double coordenadaYCirculo = 0;
    Goval circulo1 = new Goval(DIAMETRO_CIRCULO, DIAMETRO_CIRCULO);

    public void run()
    {
        final double TIEMPO_JUEGO = 30000;
        double contadorTiempo = 0;

        setSize(ANCHO_VENTANA,ALTO_VENTANA);
        setBackground(Color.RED);

        circulo1.setFilled(true);
        circulo1.setColor(Color.BLUE);
        add(circulo1,COORD_X_CIRCULO,COORD_Y_CIRCULO);
        coordenadaXCirculo = COORD_X_CIRCULO;

        while (contadorTiempo < TIEMPO_JUEGO)
        {
            avanzarUnFotograma();
            pause(TIEMPO_PARADA);
            contadorTiempo += TIEMPO_PARADA;
        }
    }

    private void avanzarUnFotograma()
    {
        coordenadaXCirculo+=2;
        coordenadaYCirculo = evaluarFuncion(coordenadaXCirculo);
        circulo1.setLocation(coordenadaXCirculo, coordenadaYCirculo);
    }

    //y = 100 + (30*seno(x/40))
    private static double evaluarFuncion(double inputX)
    {
        double outputY = 0;

        outputY = 100 + 30*Math.sin(inputX / 40);

        return outputY;
    }
}
```



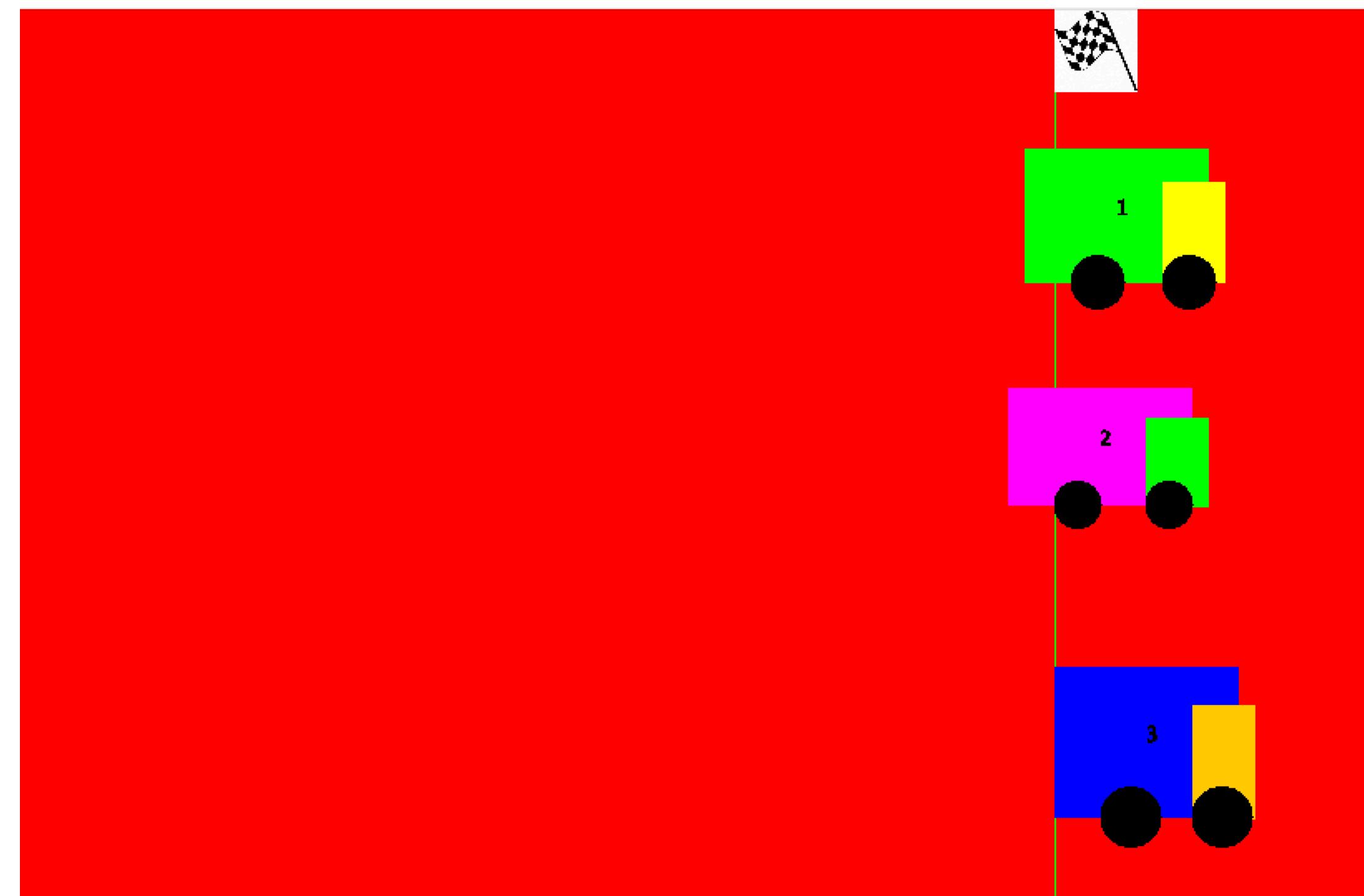
# POO

## Librerías

---

### Práctica. Carrera de coches

Debemos realizar una app gráfica para una empresa de eSports que realice una carrera de 3 coches. Los coches deben seguir una trayectoria no lineal. Cada coche tendrá un dorsal. La carrera finaliza cuando el primer coche pasa la meta. En ese momento aparece una bandera de cuadros y finaliza la carrera.



### Práctica. Carrera de coches V2

La empresa que gestiona la carreta también tiene una agencia de pilotos llamada “Ibai Llanos Co” y con nif “5454636W”. Dicha agencia gestiona 3 pilotos. Cada piloto tiene un nombre, apellidos, ranking de esta temporada (1,2 o 3) y una puntuación esta temporada.

Al comienzo de la carrera la empresa consulta a la agencia la información de sus pilotos y asigna el coche 1 al piloto con ranking 1, el coche2 al piloto con ranking 2, ...

Al finalizar la carrera se comprueba el coche ganador y se actualiza la puntuación de su piloto con 10 puntos. Finalmente se muestran los pilotos con sus respectivos rankings, puntuaciones...

### Práctica. Carrera de coches V3

La empresa que gestiona la carrera, le asigna a la Agencia de Pilotos un presupuesto, de forma que la agencia paga al piloto que gane 10000 euros.

El pago se realiza a través del banco que colabora con la agencia. Nuestra aplicación debe gestionar también ese banco, de forma que se creará al inicio del programa con 3 cuentas bancarias, una para cada piloto.

Cada piloto tendrá un número de cuenta bancaria, de tal forma que cuando un piloto gane una carrera se le ingresará en su cuenta el dinero de la carrera. Al finalizar la carrera, además de mostrar la información de puntos de los pilotos, el banco mostrará un extracto de sus cuentas bancarias.



# ¡Gracias!

