



POO avanzada



Cooperativa de Enseñanza
JOSE RAMÓN OTERO

POO avanzada

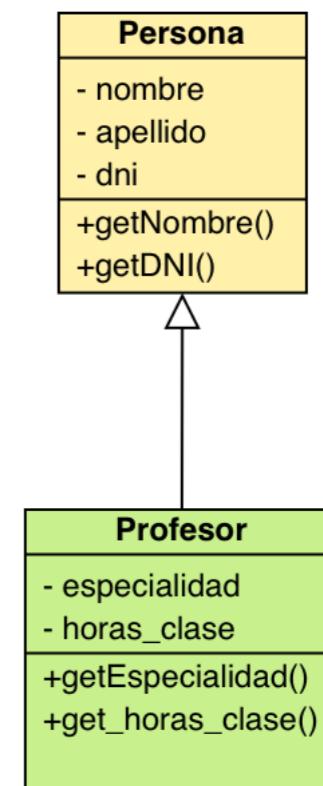
Herencia

La **herencia** es una propiedad que permite la declaración de nuevas clases a partir de otras ya existentes. Esto proporciona una de las ventajas principales de la Programación Orientada a Objetos: la reutilización de código previamente desarrollado ya que permite a una clase más específica incorporar la estructura y comportamiento de una clase más general. También muy útil para organizar, estructurar y encapsular el código.

POO avanzada

Herencia

Cuando una clase B se construye a partir de otra A mediante la herencia, la clase B hereda los atributos, métodos y clases internas de la clase A.



POO avanzada

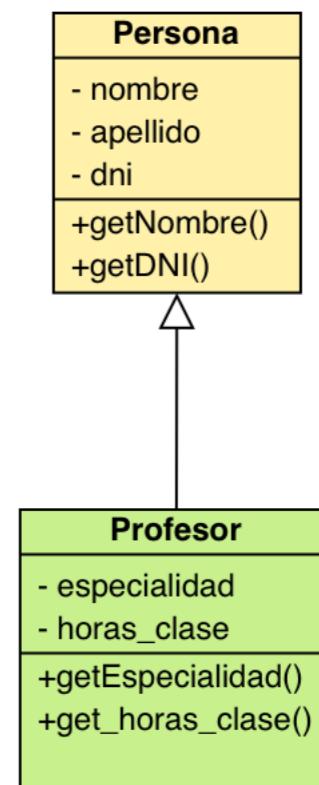
Herencia

¿Cómo saber si tenemos una relación de herencia?

Pregunta clave: **¿Es un tipo de?**

¿Profesor es un tipo de persona?

En este caso tenemos una relación de herencia entre Profesor y Persona, se representa con una flecha como la de la figura



POO avanzada

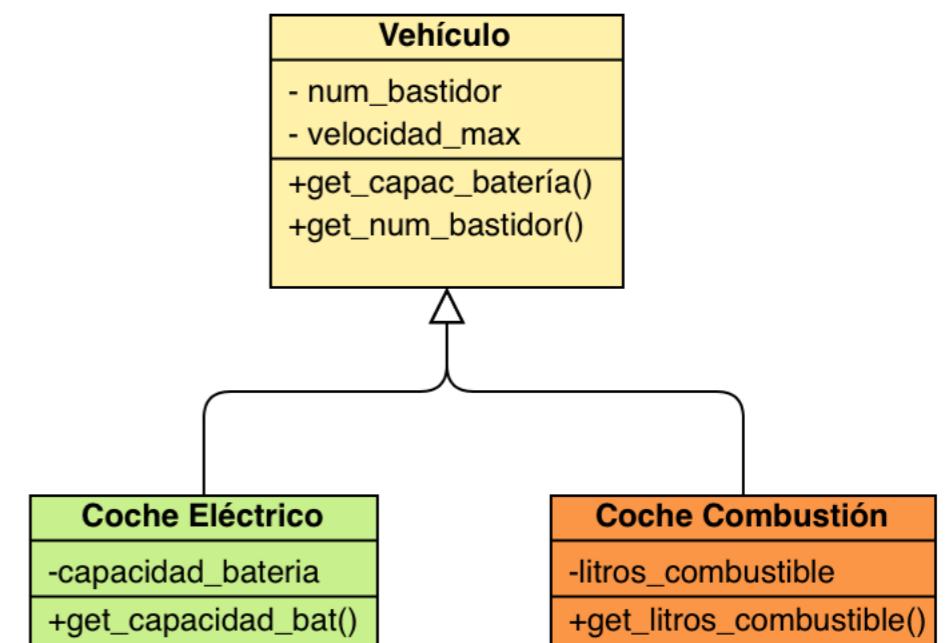
Herencia

**¿Coche eléctrico es un tipo de vehículo?
¿Y Coche de combustión?**

POO avanzada

Herencia

¿Coche eléctrico es un tipo de vehículo? ¿Y coche de combustión?



POO avanzada

Herencia

Se heredan todos los métodos y atributos public y protected, NO los private.

POO avanzada

Herencia

Para indicar que la clase B (clase descendiente, derivada, hija o subclase) hereda de la clase A (clase ascendiente, heredada, padre, base o superclase) se emplea la palabra reservada **extends** en la cabecera de la declaración de la clase descendiente.

POO avanzada

Herencia

Veamos cómo se implementa en Java

```
public class Vehiculo {  
  
    private String numBastidor;  
    private String color;  
    private int anioFab;  
    double velocidadMax;  
  
    public Vehiculo()  
    {  
        numBastidor = "___";  
        color = "___";  
        anioFab = 0;  
        velocidadMax = 0;  
    }  
  
    public Vehiculo(String numBastidor, String color,  
                    int anioFab, double velocidadMax)  
    {  
        this.numBastidor = numBastidor;  
        this.color = color;  
        this.anioFab = anioFab;  
        this.velocidadMax = velocidadMax;  
    }  
  
    public String getNumBastidor() {  
        return numBastidor;  
    }  
  
    public void mostrarInfo()  
    {  
        System.out.println("Número de bastidor: " + numBastidor);  
        System.out.println("Color: " + color);  
        System.out.println("Años de fabricación: " + anioFab);  
        System.out.println("Velocidad máxima: " + velocidadMax);  
    }  
}
```

```
public class CocheElectrico extends Vehiculo{  
    private int capacidadBateria;  
  
    public int getCapacidadBateria() {  
        return capacidadBateria;  
    }  
  
    public void setCapacidadBateria(int capacidadBateria) {  
        this.capacidadBateria = capacidadBateria;  
    }  
}
```

```
public class Concesionario {  
  
    public static void main(String[] args) {  
        CocheElectrico miCoche = new CocheElectrico();  
  
        miCoche.mostrarInfo();  
    }  
}
```

Consola

Número de bastidor: ___
Color: ___
Años de fabricación: 0
Velocidad máxima: 0.0

extends indica que
CocheElectrico
hereda de Vehiculo



Prog principal

Fíjate que se crea un
coche
eléctrico y se ejecuta
mostrarInfo() de la
clase padre

POO avanzada

Herencia

¿Qué constructor se ha ejecutado en el ejemplo anterior?

POO avanzada

Herencia

Si depuramos el programa anterior vemos que la variable miCoche de tipo CocheElectrico, fíjate en los atributos que tiene, ¿hay alguno de la clase padre?.

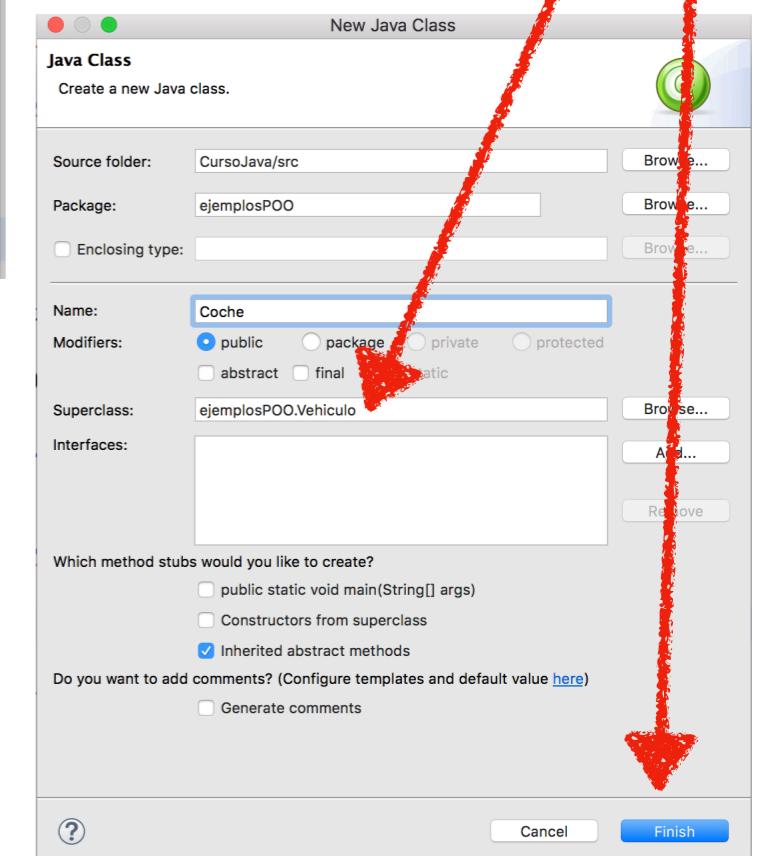
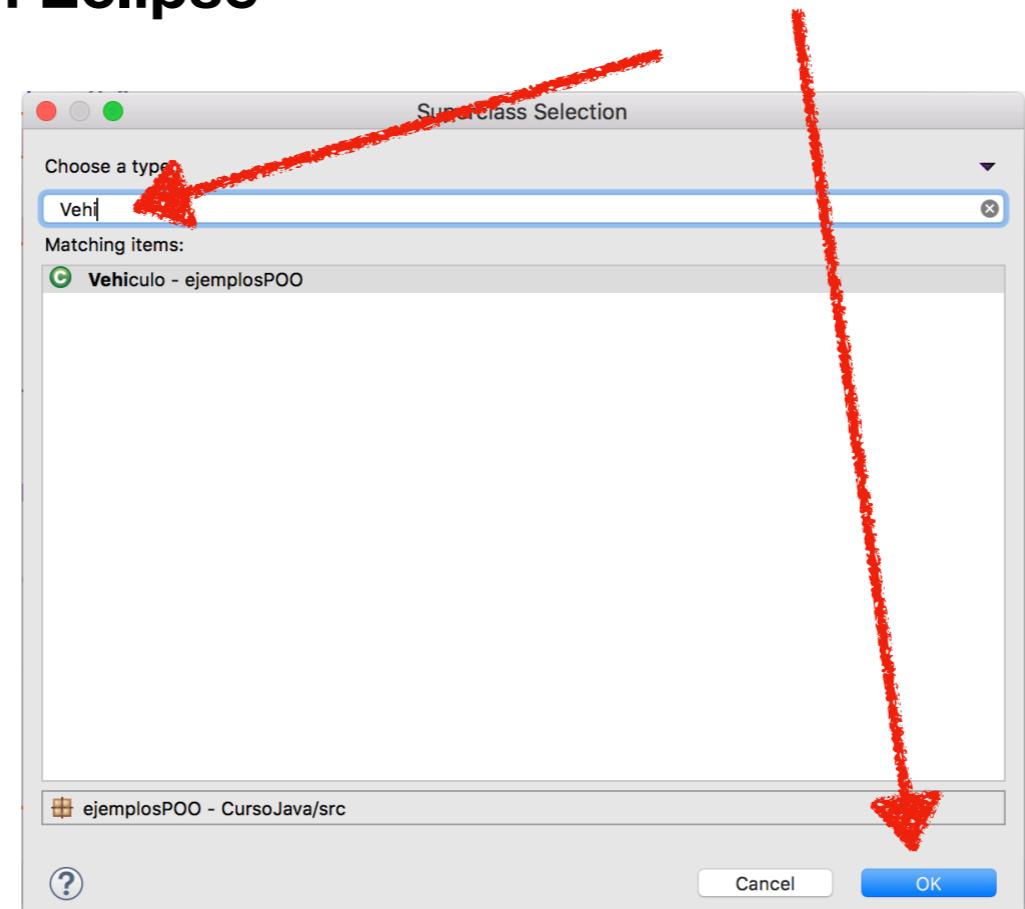
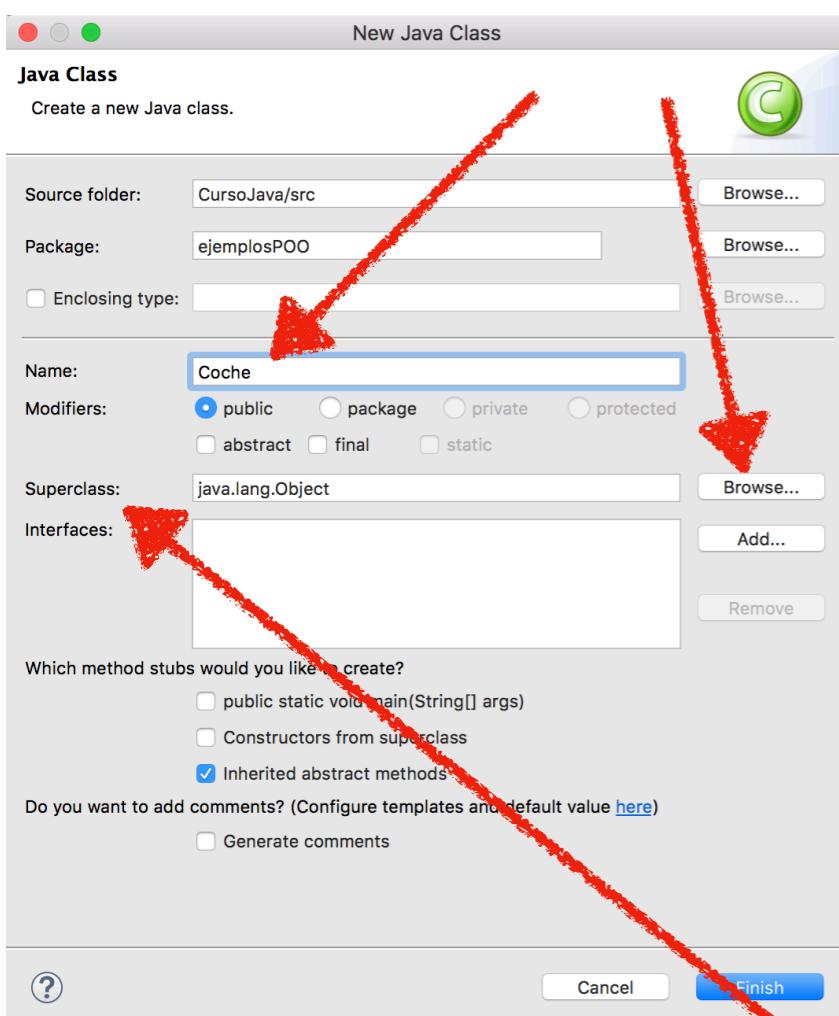
```
1 package herencia_Vehic;
2
3 public class Concesionario {
4
5     public static void main(String[] args) {
6         CocheElectrico miCoche = new CocheElectrico();
7
8         miCoche.mostrarInfo();
9
10    }
11
12 }
```

| Name | Value |
|--------------------------|------------------------|
| ↳ no method return value | |
| ↳ args | String[0] (id=27) |
| ↳ miCoche | CocheElectrico (id=18) |
| ↳ anioFab | 0 |
| ↳ capacidadBateria | 0 |
| > color | "---" (id=22) |
| > numBastidor | "---" (id=22) |
| △ velocidadMax | 0.0 |

POO avanzada

Herencia

Definición de clase hija en Eclipse



Clase padre

POO avanzada

Herencia

Ejercicio:

Realiza un concesionario similar al anterior pero que tenga un coche de combustión, muestra su información, pon un punto de ruptura y observa sus atributos.

POO avanzada

Herencia

```
public class CocheCombustion extends Vehiculo{
    private int litrosCombustible;

    public int getLitrosCombustible() {
        return litrosCombustible;
    }

    public void setLitrosCombustible(int litrosCombustible) {
        this.litrosCombustible = litrosCombustible;
    }
}

public class Concesionario_Comb {

    public static void main(String[] args) {
        CocheCombustion miCoche = new CocheCombustion();

        miCoche.mostrarInfo();
    }
}
```

Número de bastidor: ---
Color: ---
Años de fabricación: 0
Velocidad máxima: 0.0
|

POO avanzada

Herencia

De la misma manera que desde el hijo hemos usado el método mostrarInfo() de la clase padre podríamos ejecutar sus getters y setters

```
public class ConcesionarioV2 {  
  
    public class CocheElectrico extends Vehiculo{  
        private int capacidadBateria;  
  
        public int getCapacidadBateria() {  
            return capacidadBateria;  
        }  
  
        public void setCapacidadBateria(int capacidadBateria) {  
            this.capacidadBateria = capacidadBateria;  
        }  
    }  
  
    public static void main(String[] args) {  
        CocheElectrico miCoche = new CocheElectrico();  
  
        miCoche.setAnioFab(2020);  
        miCoche.setCapacidadBateria(300);  
        miCoche.setColor("Azul");  
        miCoche.setNumBastidor("345-FLSJGT");  
  
        miCoche.mostrarInfo();  
    }  
}
```

Número de bastidor: 345-FLSJGT
Color: Azul
Años de fabricación: 2020
Velocidad máxima: 0.0

POO avanzada

Herencia

Ejercicio: Haz lo mismo para el coche de combustión

POO avanzada

Herencia

Ejercicio:

Retomemos el ejercicio de la gestión de la biblioteca
(observar código en la siguiente página).

```

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;
    }

    //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;
    }

    public void mostrarInfo()
    {
        System.out.println("-----");
        System.out.println("El título es: " + titulo);
        System.out.println("El autor es: " + autor);
        System.out.println("El número de ejemplares es: " + numEjemplares);
        System.out.println("El número de ejemplares prestados es: " + numPrestados);
        System.out.println("-----");
    }
}

```

Setters y Getters

```

package herencia_biblioteca;
import java.util.Scanner;
public class GestionBibliotecaV2 {
    public static void main(String[] args) {

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

        libro1.mostrarInfo();
        libro1.prestarLibro();
        libro1.mostrarInfo();
    }
}

```

El título es: El quijote
El autor es: Cervantes
El número de ejemplares es: 3
El número de ejemplares prestados es: 0

El título es: El quijote
El autor es: Cervantes
El número de ejemplares es: 3
El número de ejemplares prestados es: 1

POO avanzada

Herencia

Supongamos que un año después en la biblioteca se quiere gestionar un tipo especial de libros, las enciclopedias, de forma que se almacene el año de publicación de cada una de ellas.

¿Cómo podríamos ampliar nuestro programa para gestionar de la misma forma las enciclopedias?

POO avanzada

```
public class Enciclopedia extends Libro
{
    private int anioEdicion;

    public int getAnioEdicion()
    {
        return anioEdicion;
    }

    public void setAnioEdicion(int anioEdicion)
    {
        this.anioEdicion = anioEdicion;
    }
}

public class GestionBibliotecaV3 {

    public static void main(String[] args) {

        Libro libro1 = new Libro("El quijote", "Cervantes", 3, 0);
        Enciclopedia enciclopedia1 = new Enciclopedia();

        enciclopedia1.setTitulo("Británica");
        enciclopedia1.setAutor("Robins");
        enciclopedia1.setNumEjemplares(4);
        enciclopedia1.setNumPrestados(0);
        enciclopedia1.setAnioEdicion(2001);

        libro1.mostrarInfo();
        libro1.prestarLibro();
        libro1.mostrarInfo();

        enciclopedia1.mostrarInfo();
        enciclopedia1.prestarLibro();
        enciclopedia1.mostrarInfo();
    }
}
```

El título es: El quijote
El autor es: Cervantes
El número de ejemplares es: 3
El número de ejemplares prestados es: 0

El título es: El quijote
El autor es: Cervantes
El número de ejemplares es: 3
El número de ejemplares prestados es: 1

El título es: Británica
El autor es: Robins
El número de ejemplares es: 4
El número de ejemplares prestados es: 0

El título es: Británica
El autor es: Robins
El número de ejemplares es: 4
El número de ejemplares prestados es: 1

POO avanzada

Herencia

En los ejemplos que hemos hecho hasta ahora puedes observar que si la clase hija no tiene constructor el compilador llama al constructor **sin parámetros** de la clase padre.

POO avanzada

Herencia

Prueba a comentar el constructor Libro(), verás que aparece un error en la clase Enciclopedia indicando que no existe constructor en el padre.

Pon `println("Soy constructor de libro")` en el constructor sin parámetros de la clase Libro para comprobar que se ejecuta cuando se llama al constructor de la clase hija Enciclopedia

POO avanzada

Herencia

Hasta ahora en nuestras clases concesionario hemos ejecutado el método `mostrarInfo()` de la clase padre en el objeto de la clase hija:

```
miCoche.mostrarInfo();
```

También podemos ejecutar métodos de la clase hija. Ver ejemplo:

POO avanzada

Herencia

```
public class Vehiculo {  
  
    private String numBastidor;  
    private String color;  
    private int anioFab;  
    double velocidadMax;  
  
    public Vehiculo()  
    {  
        numBastidor = "___";  
        color = "___";  
        anioFab = 0;  
        velocidadMax = 0;  
    }  
  
    public void mostrarInfo()  
    {  
        System.out.println("Número de bastidor: " + numBastidor);  
        System.out.println("Color: " + color);  
        System.out.println("Años de fabricación: " + anioFab);  
        System.out.println("Velocidad máxima: " + velocidadMax);  
    }  
}
```

```
public class CocheElectrico extends Vehiculo{  
    private int capacidadBateria;  
  
    public int getCapacidadBateria() {  
        return capacidadBateria;  
    }  
  
    public void setCapacidadBateria(int capacidadBateria) {  
        this.capacidadBateria = capacidadBateria;  
    }  
  
    public void mostrarInfoBateria() {  
        System.out.println("____");  
        System.out.println("Capacidad batería: " + capacidadBateria);  
    }  
}
```

```
public class ConcesionarioV2 {  
  
    public static void main(String[] args) {  
        CocheElectrico miCoche = new CocheElectrico();  
  
        miCoche.setAnioFab(2020);  
        miCoche.setCapacidadBateria(300);  
        miCoche.setColor("Azul");  
        miCoche.setNumBastidor("345-FLSJGT");  
    }  
}
```

Método clase padre → `miCoche.mostrarInfo();`

Método clase hija → `miCoche.mostrarInfoBateria();`

```
Número de bastidor: 345-FLSJGT  
Color: Azul  
Años de fabricación: 2020  
Velocidad máxima: 0.0  
----  
Capacidad batería: 300
```

POO avanzada

Herencia

Ejercicio: Realiza lo mismo para el coche de combustión y para la biblioteca.

POO avanzada

Herencia

Sobreescritura de métodos

Cómo podríamos mejorar la forma de mostrar información de un objeto hijo? Hasta ahora tenemos que llamar a dos métodos.

```
public class ConcesionarioV2 {  
    public static void main(String[] args) {  
        CocheElectrico miCoche = new CocheElectrico();  
  
        miCoche.setAnioFab(2020);  
        miCoche.setCapacidadBateria(300);  
        miCoche.setColor("Azul");  
        miCoche.setNumBastidor("345-FLSJGT");  
  
        Método clase padre → miCoche.mostrarInfo();  
  
        Método clase hija → miCoche.mostrarInfoBateria();  
    }  
}
```

POO avanzada

Herencia

Sobreescritura de métodos

Podemos **sobreescribir** el método mostrarInfo, esto es declarar un método en la clase hija que se llame igual al declarado en la clase padre. Ver ejemplo:

```
public class ConcesionarioV2 {  
    public static void main(String[] args) {  
        CocheElectricoV1 miCoche = new CocheElectricoV1();  
  
        miCoche.setAnioFab(2020);  
        miCoche.setCapacidadBateria(300);  
        miCoche.setColor("Azul");  
        miCoche.setNumBastidor("345-FLSJGT");  
  
        miCoche.mostrarInfo();  
    }  
}
```

POO avanzada

Sobreescritura de métodos

```

private String numBastidor;
private String color;
private int anioFab;
double velocidadMax;

public Vehiculo()
{
    numBastidor = "___";
    color = "___";
    anioFab = 0;
    velocidadMax = 0;
}

public int getAnioFab() {
    return anioFab;
}

public void setAnioFab(int anioFab) {
    this.anioFab = anioFab;
}

public double getVelocidadMax() {
    return velocidadMax;
}

public void setVelocidadMax(double velocidadMax) {
    this.velocidadMax = velocidadMax;
}

public void mostrarInfo()
{
    System.out.println("Número de bastidor: " + numBastidor);
    System.out.println("Color: " + color);
    System.out.println("Años de fabricación: " + anioFab);
    System.out.println("Velocidad máxima: " + velocidadMax);
}

```

```

public class CocheElectricoV1 extends Vehiculo{
    private int capacidadBateria;

    public int getCapacidadBateria() {
        return capacidadBateria;
    }

    public void setCapacidadBateria(int capacidadBateria) {
        this.capacidadBateria = capacidadBateria;
    }

    public void mostrarInfo()
    {
        System.out.println("Número de bastidor: " + this.getAnioFab());
        System.out.println("Color: " + this.getColor());
        System.out.println("Años de fabricación: " + this.getAnioFab());
        System.out.println("Velocidad máxima: " + this.getVelocidadMax());
        System.out.println("-----");
        System.out.println("Capacidad batería: " + capacidadBateria);
    }
}

```

```

public class ConcesionarioV2 {

    public static void main(String[] args) {
        CocheElectricoV1 miCoche = new CocheElectricoV1();

        miCoche.setAnioFab(2020);
        miCoche.setCapacidadBateria(300);
        miCoche.setColor("Azul");
        miCoche.setNumBastidor("345-FLSJGT");

        miCoche.mostrarInfo();
    }
}

```

Observa que desde el objeto miCoche llamamos a métodos de la clase hija y de la clase padre

Número de bastidor: 2020
 Color: Azul
 Años de fabricación: 2020
 Velocidad máxima: 0.0

 Capacidad batería: 300

POO avanzada

Herencia

Si definimos un método con el modificador **final** impedimos que sea redefinido por una clase derivada

POO avanzada

Herencia

Ejercicio: Realiza lo mismo para el coche de combustión y para la biblioteca.

POO avanzada

Herencia

Observa que en el método mostrarInfo() accedemos a los atributos de la clase padre mediante llamada a los getters, que son públicos. ¿Podríamos acceder directamente a los atributos de la clase padre? ¿Por qué?

```
public class CocheElectricov1 extends Vehiculo{
    private int capacidadBateria;

    public int getCapacidadBateria() {
        return capacidadBateria;
    }

    public void setCapacidadBateria(int capacidadBateria) {
        this.capacidadBateria = capacidadBateria;
    }

    public void mostrarInfo()
    {
        System.out.println("Número de bastidor: " + this.getAnioFab());
        System.out.println("Color: " + this.getColor());
        System.out.println("Años de fabricación: " + this.getAnioFab());
        System.out.println("Velocidad máxima: " + this.getVelocidadMax());
        System.out.println("-----");
        System.out.println("Capacidad batería: " + capacidadBateria);
    }
}
```

POO avanzada

Herencia

Para poder acceder a los atributos de la clase padre deberíamos ponerlos como **protected** de esta forma no son visibles desde el exterior pero sí desde la clase hija.

Este modificador de acceso está específicamente pensado para la herencia. Cuando se utiliza sobre una propiedad o un método indica que dicha propiedad o método serán visibles por las subclases que además heredarán la propiedad o el método. Sin embargo permanecerán invisibles para el resto.

Pruébalo en el coche de combustión y en la biblioteca

POO avanzada

Herencia

super

A la hora de implementar el método `mostrarInfo()` en la clase hija, podríamos llamar directamente a un método de la clase padre. Para ello tenemos la palabra reservada **super**. De esta forma nos queda el código más compacto y estructurado. Ver ejemplo

POO avanzada

Herencia

super

```
private String numBastidor;
private String color;
private int anioFab;
double velocidadMax;

public Vehiculo()
{
    numBastidor = "___";
    color = "___";
    anioFab = 0;
    velocidadMax = 0;
}

public void mostrarInfo()
{
    System.out.println("Número de bastidor: " + numBastidor);
    System.out.println("Color: " + color);
    System.out.println("Años de fabricación: " + anioFab);
    System.out.println("Velocidad máxima: " + velocidadMax);
}
```

```
public class CocheElectricoV1 extends Vehiculo{
    private int capacidadBateria;

    public int getCapacidadBateria() {
        return capacidadBateria;
    }

    public void setCapacidadBateria(int capacidadBateria) {
        this.capacidadBateria = capacidadBateria;
    }

    public void mostrarInfo()
    {
        System.out.println("Número de bastidor: " + this.getNumBastidor());
        //System.out.println("Número de bastidor: " + this.numBastidor);
        System.out.println("Color: " + this.getColor());
        System.out.println("Años de fabricación: " + this.getAnioFab());
        System.out.println("Velocidad máxima: " + this.getVelocidadMax());
        System.out.println("-----");
        System.out.println("Capacidad batería: " + capacidadBateria);
    }
}
```

```
public class CocheElectricoV2 extends Vehiculo{
    private int capacidadBateria;

    public int getCapacidadBateria() {
        return capacidadBateria;
    }

    public void setCapacidadBateria(int capacidadBateria) {
        this.capacidadBateria = capacidadBateria;
    }

    public void mostrarInfo()
    {
        super.mostrarInfo();
        System.out.println("-----");
        System.out.println("Capacidad batería: " + capacidadBateria);
    }
}
```

POO avanzada

Herencia

Ejercicio: Realiza lo mismo para el coche de combustión y para la biblioteca.

POO avanzada

Herencia

Java permite múltiples niveles de herencia pero no la herencia múltiple, es decir una clase sólo puede heredar directamente de una clase ascendiente.

POO avanzada

Herencia

Sobreescritura de atributos:

Como hemos visto anteriormente la clase descendiente puede añadir sus propios métodos pero también puede sustituir u ocultar los heredados (sobreescribir).

Se puede declarar un nuevo atributo con el mismo identificador que uno heredado, quedando este atributo oculto. Esta técnica no es recomendable.

POO avanzada

Herencia

Sobreescritura de elementos:

Los métodos estáticos (declarados como static) no deben ser redefinidos. Técnicamente se puede pero el resultado no es adecuado

POO avanzada

Herencia

Constructores

Hasta ahora no hemos creado constructores en las clases hijas. Podemos hacerlo igual que hemos hecho hasta ahora.

Debemos tener en cuenta cómo se comportan los constructores de las clases hijas en relación con el constructor de la clase padre. Veamos:

POO avanzada

Herencia

Constructores

Los **constructores** no se heredan de la clase base a las clases derivadas. Pero sí se puede invocar al constructor de la clase base.

POO avanzada

Herencia

Constructores

1. Si tenemos un constructor sin parámetros en la clase hija y otro constructor sin parámetros en la clase padre, cuando creamos un objeto de la clase hija con new se invoca automáticamente al constructor de la clase padre. De esta forma se pueden inicializar los atributos de la clase padre.

Ver ejemplo

```
public class Vehiculo {  
  
    private String numBastidor;  
    private String color;  
    private int anioFab;  
    double velocidadMax;  
  
    public Vehiculo()  
    {  
        numBastidor = "___";  
        color = "___";  
        anioFab = 0;  
        velocidadMax = 0;  
  
        System.out.println("Se ejecuta el constructor de Vehículo\n");  
    }  
  
    public void mostrarInfo()  
    {  
        System.out.println("Número de bastidor: " + numBastidor);  
        System.out.println("Color: " + color);  
        System.out.println("Años de fabricación: " + anioFab);  
        System.out.println("Velocidad máxima: " + velocidadMax);  
    }  
}
```

Primero se ejecuta el constructor del padre

```
public class CocheElectricoV2 extends Vehiculo{  
    private int capacidadBateria;  
  
    CocheElectricoV2()  
    {  
        capacidadBateria = 0;  
  
        System.out.println("Se ejecuta el constructor de CocheElectricoV2\n");  
    }  
    public int getCapacidadBateria() {  
        return capacidadBateria;  
    }  
  
    public void setCapacidadBateria(int capacidadBateria) {  
        this.capacidadBateria = capacidadBateria;  
    }  
  
    public void mostrarInfo()  
    {  
        super.mostrarInfo();  
        System.out.println("-----");  
        System.out.println("Capacidad batería: " + capacidadBateria);  
    }  
}
```

```
public class ConcesionarioV2 {  
  
    public static void main(String[] args) {  
        CocheElectricoV2 miCoche = new CocheElectricoV2();  
  
        miCoche.setAnioFab(2020);  
        miCoche.setCapacidadBateria(300);  
        miCoche.setColor("Azul");  
        miCoche.setNumBastidor("345-FLSJGT");  
  
        miCoche.mostrarInfo();  
    }  
}
```

Se ejecuta el constructor de Vehículo
Se ejecuta el constructor de CocheElectricoV2

Número de bastidor: 345-FLSJGT
Color: Azul
Años de fabricación: 2020
Velocidad máxima: 0.0

Capacidad batería: 300

POO avanzada

Herencia

Constructores

Ejercicio: Compruébalo para el coche de combustión y para la biblioteca. Pon `println()` en los constructores.

POO avanzada

Herencia

Constructores

2. En la clase padre podemos tener varios constructores, el comportamiento sigue siendo el mismo, se llama al constructor sin parámetros de forma transparente. Ver ejemplo

```

public class Vehiculo {
    private String numBastidor;
    private String color;
    private int anioFab;
    double velocidadMax;

    public Vehiculo()
    {
        numBastidor = "___";
        color = "___";
        anioFab = 0;
        velocidadMax = 0;
    }

    System.out.println("Se ejecuta el constructor de Vehículo\n");
}

public Vehiculo(String numBastidor, String color,
                int anioFab, double velocidadMax)
{
    this.numBastidor = numBastidor;
    this.color = color;
    this.anioFab = anioFab;
    this.velocidadMax = velocidadMax;
}

public void mostrarInfo()
{
    System.out.println("Número de bastidor: " + numBastidor);
    System.out.println("Color: " + color);
    System.out.println("Años de fabricación: " + anioFab);
    System.out.println("Velocidad máxima: " + velocidadMax);
}

```

Primero se ejecuta el constructor del padre

```

public class CocheElectricoV2 extends Vehiculo{
    private int capacidadBateria;

    CocheElectricoV2()
    {
        capacidadBateria = 0;
    }

    System.out.println("Se ejecuta el constructor de CocheElectricoV2\n");

    public int getCapacidadBateria() {
        return capacidadBateria;
    }

    public void setCapacidadBateria(int capacidadBateria) {
        this.capacidadBateria = capacidadBateria;
    }

    public void mostrarInfo()
    {
        super.mostrarInfo();
        System.out.println("-----");
        System.out.println("Capacidad batería: " + capacidadBateria);
    }
}

public class ConcesionarioV2 {

    public static void main(String[] args) {
        CocheElectricoV2 miCoche = new CocheElectricoV2();

        miCoche.setAnioFab(2020);
        miCoche.setCapacidadBateria(300);
        miCoche.setColor("Azul");
        miCoche.setNumBastidor("345-FLSJGT");
        miCoche.mostrarInfo();
    }
}

```

Se ejecuta el constructor de Vehículo

Se ejecuta el constructor de CocheElectricoV2

Número de bastidor: 345-FLSJGT
 Color: Azul
 Años de fabricación: 2020
 Velocidad máxima: 0.0

 Capacidad batería: 300

POO avanzada

Herencia

Constructores

Ejercicio: Compruébalo para el coche de combustión y para la biblioteca. Pon `println()` en los constructores.

POO avanzada

Herencia

Constructores

3. Si en la clase hija tenemos un constructor con parámetros y en la padre varios constructores se sigue llamando de forma automática al constructor sin parámetros de la clase padre. Ver ejemplo

```

public class Vehiculo {
    private String numBastidor;
    private String color;
    private int anioFab;
    double velocidadMax;

    public Vehiculo()
    {
        numBastidor = "___";
        color = "___";
        anioFab = 0;
        velocidadMax = 0;
    }

    System.out.println("Se ejecuta el constructor de Vehículo\n");
}

public Vehiculo(String numBastidor, String color,
                int anioFab, double velocidadMax)
{
    this.numBastidor = numBastidor;
    this.color = color;
    this.anioFab = anioFab;
    this.velocidadMax = velocidadMax;
}

public void mostrarInfo()
{
    System.out.println("Número de bastidor: " + numBastidor);
    System.out.println("Color: " + color);
    System.out.println("Años de fabricación: " + anioFab);
    System.out.println("Velocidad máxima: " + velocidadMax);
}

```

Primero se ejecuta el constructor del padre

```

public class CocheElectricoV2 extends Vehiculo{
    private int capacidadBateria;

    CocheElectricoV3(int capacidadBateria)
    {
        this.capacidadBateria = capacidadBateria;
    }

    System.out.println("Se ejecuta el constructor de CocheElectricoV2\n");

    public int getCapacidadBateria() {
        return capacidadBateria;
    }

    public void setCapacidadBateria(int capacidadBateria) {
        this.capacidadBateria = capacidadBateria;
    }

    public void mostrarInfo()
    {
        super.mostrarInfo();
        System.out.println("-----");
        System.out.println("Capacidad batería: " + capacidadBateria);
    }
}

```

```

public class ConcesionarioV2 {

    public static void main(String[] args) {
        CocheElectricoV3 miCoche = new CocheElectricoV3(250);

        miCoche.setAnioFab(2020);
        miCoche.setColor("Azul");
        miCoche.setNumBastidor("345-FLSJGT");

        miCoche.mostrarInfo();
    }
}

```

Se ejecuta el constructor de Vehículo
Se ejecuta el constructor de CocheElectricoV2

Número de bastidor: 345-FLSJGT
Color: Azul
Años de fabricación: 2020
Velocidad máxima: 0.0

Capacidad batería: 250

POO avanzada

Herencia

Constructores

4. Si en la clase padre no hay ningún constructor el constructor de la clase hija llama automáticamente al constructor por defecto que crea Java automáticamente para la clase padre. Compruébalo quitando los constructores de la clase padre

POO avanzada

Herencia

Constructores

5. Si en la clase hija tenemos un constructor y en la clase padre varios constructores pero no existe el constructor sin parámetros el compilador nos da un error pidiendo que se diga explícitamente a qué constructor del padre se quiere llamar. Pruébalo en tu código.

Para solucionarlo, en el constructor de la clase hija podemos llamar explícitamente al constructor correspondiente de la clase padre mediante super, en la primera línea. Ver ejemplo

POO avanzada

```
public class Vehiculo {  
  
    private String numBastidor;  
    private String color;  
    private int anioFab;  
    double velocidadMax;  
  
    public Vehiculo(String numBastidor, String color,  
                    int anioFab, double velocidadMax)  
    {  
        this.numBastidor = numBastidor;  
        this.color = color;  
        this.anioFab = anioFab;  
        this.velocidadMax = velocidadMax;  
  
        System.out.println("Se ejecuta el constructor CON parámetros");  
    }  
}
```

```
public class CocheElectricoV3 extends Vehiculo{  
    private int capacidadBateria;  
  
    CocheElectricoV3(int capacidadBateria, String numBastidor, String color,  
                     int anioFab, double velocidadMax)  
    {  
        super(numBastidor,color,anioFab,velocidadMax);  
  
        this.capacidadBateria = capacidadBateria;  
  
        System.out.println("Se ejecuta el constructor de CocheElectricoV3\n");  
    }  
    public int getCapacidadBateria() {  
        return capacidadBateria;  
    }  
}
```

```
public class ConcesionarioV2 {  
  
    public static void main(String[] args) {  
  
        CocheElectricoV3 miCoche = new CocheElectricoV3(250,"DJF-34535","rojo",2004,159.4)  
  
        miCoche.mostrarInfo();  
  
    }  
}
```

Se ejecuta el constructor CON parámetros de Vehículo

Se ejecuta el constructor de CocheElectricoV3

Número de bastidor: DJF-34535
Color: rojo
Años de fabricación: 2004
Velocidad máxima: 159.4

Capacidad batería: 250

POO avanzada

Herencia

Ejercicio: Realizar algo similar para coche de combustión y biblioteca

POO avanzada

Herencia

Ejercicio: Tenemos que gestionar el personal de un centro educativo. Tenemos dos perfiles: profesores y administrativos.

Un administrativo tiene un salario base (dato de entrada) y puede hacer horas extras (dato de entrada) que completan su salario. El precio de la hora extra está definido internamente en el sistema. Se debe calcular el salario del administrativo teniendo en cuenta las horas extras que ha realizado

El sueldo de un profesor se calcula a partir del número de horas que hace semanalmente(dato entrada) multiplicado por el precio de la hora (definido internamente)

Debemos calcular el salario algunos profesores y algunos administrativos, para ello.

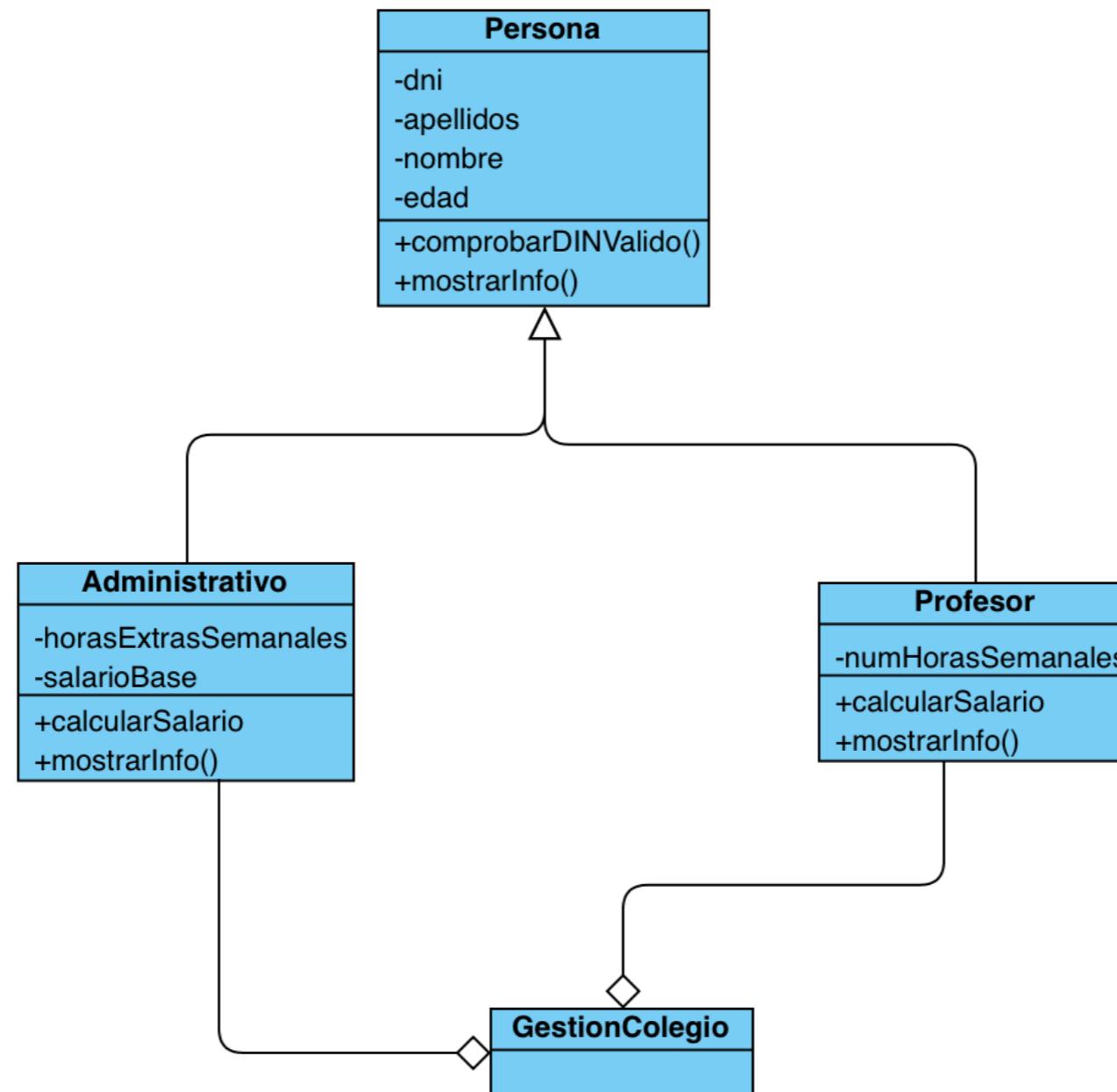
Antes de calcular el salario debemos comprobar que el dni de cada uno de ellos es correcto, para ello se comprueba que el dni tiene 9 caracteres.

Se deber crear un profesor utilizando el constructor sin parámetros y otro usando el constructor con parámetros. La clase administrativo no tiene definido constructor, se debe crear un administrativo teniéndolo en cuenta.

Realiza el diagrama de clases con UML

POO avanzada

Herencia



POO avanzada

Herencia

```
package herencia_colegio;

public class GestionColegio {

    public static void main(String[] args) {
        double salario = 0;
        Profesor profesor1 = new Profesor("34434533P", "Pepa", "López", 34, 20);
        Profesor profesor2 = new Profesor();

        Administrativo administrativo1 = new Administrativo();

        profesor2.mostrarInfo();
        profesor2.setDni("79579459U");
        profesor2.setNombre("Manolo");
        profesor2.setEdad(32);
        profesor2.setNumHorasSemanales(20);
        profesor2.mostrarInfo();

        if (profesor2.esDniValido())
        {
            salario = profesor2.calcularSalario(23);
            System.out.println("El salario del profesor es :" + salario);
        }
        profesor2.mostrarInfo();

        administrativo1.mostrarInfo();

        administrativo1.setDni("54545455U");
        administrativo1.setNombre("Andrés");
        administrativo1.setEdad(42);
        administrativo1.setHorasExtrasSemanales(10);
        administrativo1.setSalarioBase(900);

        if (administrativo1.esDniValido())
        {
            salario = administrativo1.calcularSalario();
            System.out.println("El salario del administrativo es :" + salario);
        }

        administrativo1.mostrarInfo();
    }
}
```

POO avanzada

Herencia

```
package herencia_colegio;

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

    Persona(){
        dni = "---";
        nombre = "---";
        apellidos = "---";
        edad = 0;
        System.out.println("Ejecutado constructor sin parámetros de Persona");
    }
    Persona(String dni, String nombre, String apellidos, int edad){
        this.dni = dni;
        this.nombre = nombre;
        this.apellidos = apellidos;
        this.edad = edad;
    }

    public String getDni() {
        return dni;
    }

    public void setDni(String dni) {
        this.dni = dni;
    }

    public boolean esDniValido() {
        boolean resultado = false;

        if (dni.length() == 9)
            resultado = true;

        return resultado;
    }

    public void mostrarInfo()
    {
        System.out.println("dni: " + dni);
        System.out.println("nombre: " + nombre);
        System.out.println("apellidos: " + apellidos);
        System.out.println("edad: " + edad);
    }
}
```

```

package herencia_colegio;

public class Profesor extends Persona{
    int numHorasSemanales;

    Profesor()
    {
        numHorasSemanales = 0;
    }

    Profesor (String dni, String nombre, String apellidos, int edad, int numHorasSemanales){
        super(dni, nombre, apellidos, edad);
        this.numHorasSemanales = numHorasSemanales;
    }

    public int getNumHorasSemanales() {
        return numHorasSemanales;
    }

    public void setNumHorasSemanales(int numHorasSemanales) {
        this.numHorasSemanales = numHorasSemanales;
    }

    public double calcularSalario(double numHorasSemanales)
    {
        double montanteSalario = 0;
        final double PRECIO_HORA_ENSEÑANZA = 90;

        montanteSalario = numHorasSemanales * PRECIO_HORA_ENSEÑANZA;

        return montanteSalario;
    }

    public void mostrarInfo()
    {
        System.out.println("-----Profesor-----");
        super.mostrarInfo();
        System.out.println("número de horas semanales: " + numHorasSemanales);
        System.out.println("-----");
    }
}

```

```

public class Administrativo extends Persona{
    int horasExtrasSemanales;
    double salarioBase;

    public int getHorasExtrasSemanales() {
        return horasExtrasSemanales;
    }

    public void setHorasExtrasSemanales(int horasExtrasSemanales) {
        this.horasExtrasSemanales = horasExtrasSemanales;
    }

    public double getSalarioBase() {
        return salarioBase;
    }

    public void setSalarioBase(double salarioBase) {
        this.salarioBase = salarioBase;
    }

    public double calcularSalario()
    {
        double montanteSalario = 0;
        final double PRECIO_HORA_EXTRA = 50;

        montanteSalario = (horasExtrasSemanales * PRECIO_HORA_EXTRA) + salarioBase;

        return montanteSalario;
    }

    public void mostrarInfo()
    {
        System.out.println("-----Administrativo-----");
        super.mostrarInfo();
        System.out.println("salario base: " + salarioBase);
        System.out.println("horas extra semanales: " + horasExtrasSemanales);
        System.out.println("-----");
    }
}

```

POO avanzada

Fechas

Java tiene varias clases para gestionar fechas, tradicionalmente se han usado las clases Date o Calendar.

En Java 1.8 se introdujo la clase LocalDate (paquete java.time), nos permite trabajar cómodamente con fechas. Revisa la documentación de esta clase.

POO avanzada

Fechas

El siguiente ejemplo nos permite crear una fecha y mostrarla con el formato que deseemos. Revisa la documentación de `DateTimeFormatter`

```
import java.time.LocalDate;
import java.time.Period;
import java.time.format.DateTimeFormatter;

//Creamos un objeto con la fecha actual
LocalDate fecha1 = LocalDate.now();
System.out.println("Sin formatear: " + fecha1);

//Podemos formatear la fecha con DateTimeFormatter
DateTimeFormatter formatoFecha = DateTimeFormatter.ofPattern("dd/MMM/yyyy");

String fechaFormateada = fecha1.format(formatoFecha);
System.out.println("Fecha formateada: " + fechaFormateada);
```

Sin formatear: 2022-01-22
Fecha formateada: 22/Jan/2022

POO avanzada

Fechas

El siguiente ejemplo nos permite crear una fecha y mostrarla con el formato que deseemos. Revisa la documentación de `DateTimeFormatter`

```
//Creamos un objeto con la fecha actual
LocalDate fecha1 = LocalDate.now();
System.out.println("Sin formatear: " + fecha1);

//Podemos formatear la fecha con DateTimeFormatter
DateTimeFormatter formatoFecha = DateTimeFormatter.ofPattern("dd/MMM/yyyy");

String fechaFormateada = fecha1.format(formatoFecha);
System.out.println("Fecha formateada: " + fechaFormateada);
```

```
Sin formatear: 2022-01-22
Fecha formateada: 22/Jan/2022
```

POO avanzada

Fechas

Podemos definir una fecha determinada, también podemos obtener el mes o el año de una fecha:

```
//Asignar una fecha determinada
fecha1 = LocalDate.of(2025,10,23);
fechaFormatoada = fecha1.format(formatoFecha);
System.out.println("Fecha formateada: " + fechaFormatoada);

//Obtenemos el día y el mes:
int dia = fecha1.getDayOfMonth();
System.out.println("El día del mes es: " + dia);

int mes = fecha1.getMonthValue();
System.out.println("El mes es: " + mes);
```

Fecha formateada: 23/oct/2025
El día del mes es: 23
El mes es: 10

POO avanzada

Fechas

También podemos calcular el tiempo entre dos fechas, revisa en la documentación de Java el método until de la clase LocalDate y la clase Period:

```
//Calculamos tiempo desde una fecha hasta otra
Period periodo = fecha2.until(fecha1);
System.out.println("El número de días entre: " + fecha2Formateada +
    " y " + fechaFormateada + " es " + periodo.getDays() +
    ",meses: " + periodo.getMonths() + " y años: " + periodo.getYears());
```

El número de días entre: 22/Jan/2022 y 23/Oct/2025 es 1 ,meses: 9 y años: 3

POO avanzada

Herencia-Fechas

Ejercicio: Un año después de implantar el sistema de gestión del banco, se lanza la cuenta verde, esta cuenta cada vez que se hace un ingreso el banco dona un 0,1% en un bono verde (dinero) asociado a la cuenta. Se irá acumulando cada vez que se haga un ingreso y cuando llegue la fecha de conversión del bono (dato de entrada) se donará el dinero acumulado a una ONG que trabaje por la mejora del medio ambiente.

La cuenta verde tendrá un constructor por defecto que inicialice el nombre, número de cuenta, interés y saldo a "—", "---", 0, 0. Además inicializa la fecha de conversión del bono a 1/1/2030, la muestra en varios formatos, y la cuantía del bono a 10€.

Tendrá otro constructor que recibirá como parámetros de entrada el día, mes y año de la fecha de conversión del bono. El resto de parámetros se inicializan como en el punto anterior.

Tendrá otro constructor que permitirá crear todos los valores anteriores a partir de unos datos de entrada.

Además habrá un método que permita mostrar los años, meses y días que faltan para que se haga la donación de lo almacenado en el bono a la ONG.

Crea variables Cuenta y CuentaVerde, haz ingresos en ellas, saca dinero, haz transferencias de unas a otras y muestra información de ellas.

POO avanzada

```
public class Banco {  
  
    private static Cuenta cuentaCorriente1 = null;  
  
    private static Cuenta_Verde cuentaVerde1 = null;  
    private static Cuenta_Verde cuentaVerde2 = null;  
    private static Cuenta_Verde cuentaVerde3 = null;  
  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
        String tiempoParaFinBono = "";  
        cuentaCorriente1 = new Cuenta("Ana","IWT-98493859",2.3, 2000);  
  
        cuentaVerde1 = new Cuenta_Verde(2023,5,28);  
        cuentaVerde2 = new Cuenta_Verde();  
        cuentaVerde3 = new Cuenta_Verde("Inés","UPT-93859",2.3,25000,2023,5,28);  
  
        cuentaVerde1.setNombre("Andrés");  
        cuentaVerde1.setNumeroCuenta("U0-808080");  
        cuentaVerde1.setSaldo(2000);  
        cuentaVerde1.setTipoInteres(2);  
  
        hacerIngreso(cuentaVerde1,2000);  
  
        cuentaVerde1.setFechaConversBono(2025,7,21);  
  
        sacarDinero(cuentaCorriente1,5000);  
  
        hacerTransferencia(cuentaCorriente1, cuentaVerde1, 2000);  
  
        cuentaCorriente1.mostrarInfo();  
        cuentaVerde1.mostrarInfo();  
  
        tiempoParaFinBono = cuentaVerde1.calcularTiempoParaConversion();  
        System.out.println("El tiempo pendiente para conversión es: " + tiempoParaFinBono);  
    }  
}
```

POO avanzada

```
import java.text.SimpleDateFormat;
import java.time.LocalDate;
import java.time.Period;
import java.time.format.DateTimeFormatter;

public class Cuenta_Verde extends Cuenta{

    double cuantia_bono_verde;
    LocalDate fechaConversBono;

    final double SALDO_INICIAL = 10;
    final double PORCENTAJE_DONACION = 0.001;

    Cuenta_Verde()
    {
        super("___", "___", 0, 0);

        fechaConversBono = LocalDate.of(2030,1,1);
        cuantia_bono_verde = SALDO_INICIAL;

        //Para ver la hora por consola:
        DateTimeFormatter formatoFecha = DateTimeFormatter.ofPattern("dd/MMM/yyyy");

        String fechaFormateada = fechaConversBono.format(formatoFecha);
        System.out.println("Fecha por defecto de la conversion es: " + fechaFormateada);
    }

    Cuenta_Verde(int anioConversion, int mesConversion, int diaConversion)
    {
        super("___", "___", 0, 0);
        fechaConversBono = LocalDate.of(anioConversion,mesConversion,diaConversion);
        cuantia_bono_verde = SALDO_INICIAL;
    }

    Cuenta_Verde(String inNombre, String inNumeroCuenta, double inTipoInteres,
                double inSaldo, int anioConversion, int mesConversion, int diaConversion)
    {
        super(inNombre,inNumeroCuenta,inTipoInteres,inSaldo);
        fechaConversBono = LocalDate.of(anioConversion,mesConversion,diaConversion);
        cuantia_bono_verde = SALDO_INICIAL;
    }
}
```

POO avanzada

```
public void setFechaConversBono(int anioConversion, int mesConversion, int diaConversion)
{
    this.fechaConversBono = LocalDate.of(anioConversion,mesConversion,diaConversion);;

}

@Override
public boolean ingresar(double cantidad) {
    boolean resultado = super.ingresar(cantidad);

    if (resultado)
    {
        cuantia_bono_verde += (cantidad * PORCENTAJE_DONACION);
    }

    return resultado;
}

public String calcularTiempoParaConversion()
{
    LocalDate fechaActual = LocalDate.now();

    //Calculamos tiempo desde una fecha hasta otra
    Period periodo = fechaActual.until(fechaConversBono);
    String salida = "Días: " + periodo.getDays() +
                    ",meses: " + periodo.getMonths() +
                    ", años: " + periodo.getYears();

    return salida;
}

@Override
public void mostrarInfo() {
    super.mostrarInfo();

    DateTimeFormatter formatoFecha = DateTimeFormatter.ofPattern("dd/MMM/yyyy");

    String fechaFormateada = fechaConversBono.format(formatoFecha);
    System.out.println("Fecha de la conversion es: " + fechaFormateada);
    System.out.println("La cuantía del bono es: " + cuantia_bono_verde);
    System.out.println("*****");
}
```

POO avanzada

Herencia

Observa que cuando hemos sobreescrito el método ingresar hemos añadido la **directiva @Override**, es opcional. Esto nos ayuda porque fuerza al compilador de java a comprobar que se codifica adecuadamente la sobreescritura del método y así evita errores en tiempo de ejecución

POO avanzada

Herencia

Es posible asignar referencias (variables) de una superclase a objetos de una de sus subclases (pero no al revés).

```
public static void main(String[] args) {  
    Vehiculo vehiculo1 = new Vehiculo();  
  
    Coche2 coche2 = new Coche2("222-CDS", "rojo", 2016, 300);  
  
    vehiculo1 = coche2; //Un objeto de la clase  
                      //padre puede apuntar a un objeto de la clase hija  
  
    Xcoche2 = vehiculo1; //pero no alrevés. ERROR, tipos incompatibles
```

POO avanzada

Herencia

Ejercicio: Pruébalo con las clases libro y enciclopedia

POO avanzada

Herencia

Se debe tener en cuenta que los objetos nunca cambian de tipo, una variable de la clase padre se puede asignar a un objeto de la clase hija **pero** no pueden acceder a propiedades o métodos que no sean propios de la clase padre.

```
3 public class ConcesionarioV3 {  
4  
5     public static void main(String[] args) {  
6         Vehiculo vehiculo1 = new Vehiculo();  
7  
8         CocheElectricoV2 miCoche = new CocheElectricoV2();  
9  
10        miCoche.setAnioFab(2020);  
11        miCoche.setCapacidadBateria(300);  
12        miCoche.setColor("Azul");  
13        miCoche.setNumBastidor("345-FLSJGT");  
14  
15        miCoche.mostrarInfo();  
16  
17  
18        vehiculo1 = miCoche;  
19        vehiculo1.setNumBastidor("3535UIU");  
20        vehiculo1.setCapacidadBateria(234);  
21  
22    }
```

POO avanzada

Herencia

Ejercicio: Pruébalo con las clases libro y enciclopedia

POO avanzada

Herencia

Esta propiedad de la **asignación de variables de objetos de la clase padre a objetos de la clase hija** es muy útil para **realizar código más compacto y reutilizar código**. Observa en el siguiente ejemplo cómo tenemos creado un método:

mostrarInformacionVehiculoTaller(Vehiculo *vehiculo*)

y llamamos a ese método pasando parámetros de tipo Vehiculo, de tipo Coche_Electrico y de tipo Coche_Combustion

Java detecta dinámicamente qué tipo de vehiculo es y llama al método mostrarInfo() de la clase apropiada.

POO avanzada

Herencia

```
public static void main(String[] args) {
    Vehiculo vehiculo1 = new Vehiculo();

    CocheElectricoV2 miCoche = new CocheElectricoV2();

    miCoche.setAnioFab(2020);
    miCoche.setCapacidadBateria(300);
    miCoche.setColor("Azul");
    miCoche.setNumBastidor("345-FLSJGT");

    CocheCombustion miOtroCoche = new CocheCombustion();
    miOtroCoche.setColor("Rojo");
    miOtroCoche.setNumBastidor("222-WEDEW");
    miOtroCoche.setLitrosCombustible(200);

    mostrarInformacionVehiculoTaller(vehiculo1);
    mostrarInformacionVehiculoTaller(miCoche);
    mostrarInformacionVehiculoTaller(miOtroCoche);

}

public static void mostrarInformacionVehiculoTaller(Vehiculo vehiculo)
{
    System.out.println("*****%%%%%%%%%%%%%*****");
    LocalDate fecha1 = LocalDate.now();
    DateTimeFormatter formatoFecha = DateTimeFormatter.ofPattern("dd/MMM/yyyy");
    String fechaFormateada = fecha1.format(formatoFecha);
    System.out.println("Hoy, día: " + fechaFormateada + " Tenemos el siguiente coche");

    vehiculo.mostrarInfo();

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

POO avanzada

Herencia

Realiza un **ejercicio** similar con la clase Libro, Enciclopedia y una nueva clase Comic que tiene como atributo String tipollustración, además de los atributos de Libro.

POO avanzada

Herencia

Supongamos ahora que tenemos que crear un array de coches para manejar los coches del taller. Tenemos coches eléctricos y coches de combustión, ¿cómo declararías el array?

POO avanzada

Herencia

```
public static void main(String[] args) {
    Vehiculo[] arrayVehiculos = new Vehiculo[3];

    arrayVehiculos[0] = new Vehiculo();

    CocheElectricov2 miCoche = new CocheElectricov2();

    miCoche.setAnioFab(2020);
    miCoche.setCapacidadBateria(300);
    miCoche.setColor("Azul");
    miCoche.setNumBastidor("345-FLSJGT");

    arrayVehiculos[1] = miCoche;

    CocheCombustion miOtroCoche = new CocheCombustion();
    miOtroCoche.setColor("Rojo");
    miOtroCoche.setNumBastidor("222-WEDEW");
    miOtroCoche.setLitrosCombustible(200);

    arrayVehiculos[2] = miOtroCoche;
    mostrarInformacionVehiculosTaller(arrayVehiculos);
}

public static void mostrarInformacionVehiculosTaller(Vehiculo[] listaVehiculos)
{
    LocalDate fecha1 = LocalDate.now();
    DateTimeFormatter formatoFecha = DateTimeFormatter.ofPattern("dd/MMM/yyyy");
    String fechaFormateada = fecha1.format(formatoFecha);

    System.out.println("*****%#####*****");
    System.out.println("Hoy, día: " + fechaFormateada + " Tenemos el siguiente coche");
    for (Vehiculo vehiculo: listaVehiculos)
    {
        if (vehiculo != null)
        {
            vehiculo.mostrarInfo();
        }
    }
    System.out.println("*****%#####*****");
}
```

POO avanzada

Herencia

Observa como, aunque el array sea de Vehiculos, se llama al mostrarInfo() de la clase apropiada. Java lo resuelve dinámicamente.

POO avanzada

Herencia

Realiza un **ejercicio** similar con la clase Libro, Enciclopedia y Comic.

POO avanzada

Herencia

Supongamos ahora que en la función

```
public static void  
mostrarInformacionVehiculosTaller(Vehiculo[]  
listaVehiculos)
```

queremos que se muestre para el caso de los vehículos de combustión un texto “OFERTA por tecnología obsoleta”. ¿Cómo lo harías?

POO avanzada

Herencia

instanceof

Devuelve True o False. Permite comprobar si un determinado objeto pertenece a una clase concreta.

```
if (coche2 instanceof Vehiculo)
{
    System.out.println("El coche 2 es un vehiculo");
}

if (coche2 instanceof Coche2)
{
    System.out.println("El coche 2| es un Coche2");
}
```

POO avanzada

Herencia

```
public static void main(String[] args) {
    Vehiculo[] arrayVehiculos = new Vehiculo[5];
    arrayVehiculos[0] = new Vehiculo();
    CocheElectricov2 miCoche = new CocheElectricov2();

    miCoche.setAnioFab(2020);
    miCoche.setCapacidadBateria(300);
    miCoche.setColor("Azul");
    miCoche.setNumBastidor("345-FLSJGT");

    arrayVehiculos[1] = miCoche;

    CocheCombustion miOtroCoche = new CocheCombustion();
    miOtroCoche.setColor("Rojo");
    miOtroCoche.setNumBastidor("222-WEDEW");
    miOtroCoche.setLitrosCombustible(200);

    arrayVehiculos[2] = miOtroCoche;
    mostrarInformacionVehiculosTaller(arrayVehiculos);
}

public static void mostrarInformacionVehiculosTaller(Vehiculo[] listaVehiculos)
{
    LocalDate fecha1 = LocalDate.now();
    DateTimeFormatter formatoFecha = DateTimeFormatter.ofPattern("dd/MMM/yyyy");
    String fechaFormateada = fecha1.format(formatoFecha);

    System.out.println("*****%#####*****");
    System.out.println("Hoy, día: " + fechaFormateada + " Tenemos el siguiente coche");

    for (Vehiculo vehiculo: listaVehiculos)
    {
        if (vehiculo != null)
        {
            if (vehiculo instanceof CocheCombustion)
            {
                System.out.println("***** OFERTA POR TECNOLOGÍA OBSOLETA *****");
            }
            vehiculo.mostrarInfo();
        }
    }
    System.out.println("*****%#####*****");
}
```

POO avanzada

Herencia

instanceof

Realiza un **ejercicio** parecido para el caso de la Biblioteca.
Para el caso del Comic se comprueba si estamos en el mes de abril para poner un cartel de oferta ya que ese mes se celebra la feria del Comic

POO avanzada

Herencia

Supongamos ahora que queremos que para el caso de los coches eléctricos queremos obtener la capacidad de la batería para resaltarlo en la oferta. Observa que nos da un error de compilación porque no podemos ejecutar desde una referencia a un objeto de la clase padre un método de una clase hija

```
-- 29  public static void mostrarInformacionVehiculosTaller(Vehiculo[] listaVehiculos)
30  {
31      LocalDate fecha1 = LocalDate.now();
32      DateTimeFormatter formatoFecha = DateTimeFormatter.ofPattern("dd/MMM/yyyy");
33      String fechaFormateada = fecha1.format(formatoFecha);
34
35      System.out.println("*****%*****");
36      System.out.println("Hoy, día: " + fechaFormateada + " Tenemos el siguiente coche");
37
38      for (Vehiculo vehiculo: listaVehiculos)
39      {
40          if (vehiculo != null)
41          {
42              if (vehiculo instanceof CocheElectrico)
43              {
44                  System.out.println("**BATERIA ALTA CAPACIDAD: " + vehiculo.getCapacidadBateria());
45              }
46              vehiculo.mostrarInfo();
47          }
48      }
49      System.out.println("*****%*****");
50  }
```

POO avanzada

Herencia

Solución: **Casting**, con cuidado, debemos asegurarnos con instanceof que es una referencia a la clase adecuada, de lo contrario tendríamos un error en tiempo de ejecución.

```
if (vehiculo instanceof CocheElectrico)
{
    System.out.println("**BATERIA ALTA CAPACIDAD: " + ((CocheElectricoV2)vehiculo).getCapacidadBateria());
}
vehiculo.mostrarInfo();
```

POO avanzada

```
public static void main(String[] args) {
    Vehiculo[] arrayVehiculos = new Vehiculo[5];
    arrayVehiculos[0] = new Vehiculo();
    CocheElectricoV2 miCoche = new CocheElectricoV2();

    miCoche.setAnioFab(2020);
    miCoche.setCapacidadBateria(300);
    miCoche.setColor("Azul");
    miCoche.setNumBastidor("345-FLSJGT");

    arrayVehiculos[1] = miCoche;

    CocheCombustion miOtroCoche = new CocheCombustion();
    miOtroCoche.setColor("Rojo");
    miOtroCoche.setNumBastidor("222-WEDEW");
    miOtroCoche.setLitrosCombustible(200);

    arrayVehiculos[2] = miOtroCoche;
    mostrarInformacionVehiculosTaller(arrayVehiculos);
}

public static void mostrarInformacionVehiculosTaller(Vehiculo[] listaVehiculos)
{
    LocalDate fecha1 = LocalDate.now();
    DateTimeFormatter formatoFecha = DateTimeFormatter.ofPattern("dd/MMM/yyyy");
    String fechaFormateada = fecha1.format(formatoFecha);

    System.out.println("*****%%%%%%%%*****");
    System.out.println("Hoy, día: " + fechaFormateada + " Tenemos el siguiente coche");

    for (Vehiculo vehiculo: listaVehiculos)
    {
        if (vehiculo != null)
        {
            if (vehiculo instanceof CocheElectrico)
            {
                System.out.println("**BATERIA ALTA CAPACIDAD: " + ((CocheElectricoV2)vehiculo).getCapacidadBateria());
            }
            vehiculo.mostrarInfo();
        }
    }
    System.out.println("*****%%%%%%%%*****");
}
```

POO avanzada

Herencia

Realiza un **ejercicio** similar con la clase Libro, Enciclopedia y Comic.

POO avanzada

Herencia

Ejercicio: Debemos realizar un programa que gestione los influencers con los que trabaja una agencia de publicidad. Se gestionarán 3 tipos de influencers: Celebritys, Gamers y Foodies. Los datos que se gestionan para todos ellos son nombre, apellidos, nif, ingresos mensuales, número de seguidores, red social de referencia.

De los celebritys se almacenará su profesión (músico, cantante, modelo...). De los Gamers su nick y el juego en el que destacan y la fecha del último campeonato jugado, de los Foodies el tipo de cocina en el que más destacan (japonesa, mediterránea, vietnamita, innovadora...)

El sistema debe implementar un directorio de influencers que permita buscar por ingresos, agrupar por tipos de influencers según el siguiente menu:

1. Insertar Gamer
2. Insertar Celebrity
3. Insertar Foodie
4. Borrar influencer
5. mostrar Agrupados Por Categoria
6. mostrar los que tienen ingresos superiores a un valor dado
7. Salir

```

public class AgenciaInfluencers {

    static DirectorioInfluencers directorio;
    static Scanner lectorEntrada = null;
    public static void main(String[] args) {

        int tamanioDirectorio = 0;
        int opcion = 0;

        lectorEntrada = new Scanner(System.in);

        System.out.print("Introduce tamaño directorio: ");
        tamanioDirectorio = lectorEntrada.nextInt();

        directorio = new DirectorioInfluencers(tamanioDirectorio);

        do
        {
            System.out.println("1. Insertar Gamer\n"
                + "2. Insertar Celebrity \n"
                + "3. Insertar Foodie\n"
                + "4. Borrar influencer \n"
                + "5. mostrar Agrupados Por Categoria \n"
                + "6. mostrar los que tienen ingresos superiores a un valor dado \n"
                + "7. Salir\n");

            System.out.print("Introduce opción: ");
            opcion = lectorEntrada.nextInt();

            switch(opcion)
            {
                case 1:
                    insertarGamer();
                    break;
                case 2:
                    insertarCelebrity();
                    break;
                case 3:
                    insertarFoodie();
                    break;
                case 4:
                    borrarInfluencer();
                    break;
                case 5:
                    directorio.mostrarAgrupadosPorCategoria();
                    break;
                case 6:
                    mostrarIngresosSuperioresA();
                    break;
                case 7:
                    System.out.println("Saliendo...");
                    break;
                default:
                    System.out.println("Opción errónea");
            }
        }while(opcion != 7);

        lectorEntrada.close();
    }

    private static void insertarGamer()
    {
        Gamer gamer = null;
        String docIdentidad;
        String nombrePersona;
        String apellidoPersona;
        int edadPersona;
        double ingresosMensuales;
        int numeroSeguidores;
        String redSocialReferencia;
        String nick;
        String juegoReferencia;
        LocalDate fechaUltimoCampeonato;
        int anioUltimoCampeonato;

        System.out.print("Introduce docIdentidad: ");
        docIdentidad = lectorEntrada.next();
        System.out.print("Introduce nombrePersona: ");
        nombrePersona = lectorEntrada.next();
        System.out.print("Introduce apellidoPersona: ");
        apellidoPersona = lectorEntrada.next();
        System.out.print("Introduce edadPersona: ");
        edadPersona = lectorEntrada.nextInt();
        System.out.print("Introduce ingresosMensuales: ");
        ingresosMensuales = lectorEntrada.nextDouble();
        System.out.print("Introduce numeroSeguidores: ");
        numeroSeguidores = lectorEntrada.nextInt();
        System.out.print("Introduce redSocialReferencia: ");
        redSocialReferencia = lectorEntrada.next();
        System.out.print("Introduce nick: ");
        nick = lectorEntrada.next();
        System.out.print("Introduce juegoReferencia: ");
        juegoReferencia = lectorEntrada.next();
        System.out.print("Introduce año ultimo campeonato: ");
        anioUltimoCampeonato = lectorEntrada.nextInt();
        fechaUltimoCampeonato = LocalDate.of(anioUltimoCampeonato, 1, 1);

        gamer = new Gamer(docIdentidad,nombrePersona,apellidoPersona,
                           edadPersona,ingresosMensuales,numeroSeguidores,
                           redSocialReferencia,nick,juegoReferencia,fechaUltimoCampeonato);
    }

    if (directorio.insertarInfluencer(gamer) == false)
    {
        System.out.println("Error al insertar Gamer");
    }

    private static void borrarInfluencer() {
        String dni;
        System.out.println("Intro dni: \n");
        dni = lectorEntrada.next();

        if (directorio.borrarInfluencer(dni) == false)
        {
            System.out.println("Error al borrar influencer");
        }
    }

    private static void mostrarIngresosSuperioresA()
    {
        double ingresos;

        System.out.print("Introduce valor ingreso: ");
        ingresos = lectorEntrada.nextDouble();

        directorio.mostrarIngresosSuperioresA(ingresos);
    }
}

```

POO avanzada

```

public class DirectorioInfluencers {
    Influencer[] listaInfluencers;

    DirectorioInfluencers(int numeroInfluencers)
    {
        listaInfluencers = new Influencer[numeroInfluencers];
    }

    public boolean insertarInfluencer(Influencer influencer)
    {
        boolean insertado = false;
        int contador = 0;

        //Recorremos el array buscando un hueco, cuando lo insertamos
        //salimos del bucle
        while ((contador < listaInfluencers.length) && !(insertado))
        {
            if (listaInfluencers[contador] == null)
            {
                listaInfluencers[contador] = influencer;
                insertado = true;
            }
            contador++;
        }

        return insertado;
    }

    //Lo borramos a partir del dni
    public boolean borrarInfluencer(String dni)
    {
        boolean borrado = false;

        for (int i = 0; i < listaInfluencers.length; i++)
        {
            if (listaInfluencers[i] != null)
            {
                if (dni.equals(listaInfluencers[i].getDni()))
                {
                    listaInfluencers[i] = null;
                    borrado = true;
                }
            }
        }

        return borrado;
    }

    public void mostrarAgrupadosPorCategoria()
    {
        System.out.println("Gamers: -----");
        for (Influencer influencer: listaInfluencers)
        {
            if (influencer instanceof Gamer)
            {
                influencer.mostrarInfo();
                System.out.println("-----");
            }
        }

        System.out.println("Foodies: -----");
        for (Influencer influencer: listaInfluencers)
        {
            if (influencer instanceof Foodie)
            {
                influencer.mostrarInfo();
                System.out.println("-----");
            }
        }

        System.out.println("Celebritys: -----");
        for (Influencer influencer: listaInfluencers)
        {
            if (influencer instanceof Celebrity)
            {
                influencer.mostrarInfo();
                System.out.println("-----");
            }
        }

        public void mostrarIngresosSuperioresA(double ingresos)
        {
            System.out.println("Influencers que ganan más de " + ingresos + ":");

            for (Influencer influencer: listaInfluencers)
            {
                if (influencer != null)
                {
                    if (influencer.getIngresosAnuales() > ingresos)
                    {
                        influencer.mostrarInfo();
                    }
                }
            }
        }
    }
}

```

Herencia

```
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, 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("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;  
    }  
  
    public String getDni() {  
        return dni;  
    }  
  
    public void setDni(String dni) {  
        this.dni = dni;  
    }  
}
```

POO avanzada

Herencia

```
public class Influencer extends Persona{
    double ingresosAnuales;
    int numeroSeguidores;
    String redSocialReferencia;

    Influencer()
    {
        super("___","___","___",0);
        ingresosAnuales = 0;
        numeroSeguidores = 0;
        redSocialReferencia = "___";
    }

    Influencer(String docIdentidad, String nombrePersona, String apellidoPersona,
               int edadPersona,double ingresosMensuales, int numeroSeguidores, String redSocialReferencia)
    {
        super(docIdentidad,nombrePersona,apellidoPersona,edadPersona);
        this.ingresosAnuales = ingresosMensuales;
        this.numeroSeguidores = numeroSeguidores;
        this.redSocialReferencia = redSocialReferencia;
    }

    public double getIngresosAnuales() {
        return ingresosAnuales;
    }

    public void setIngresosAnuales(double ingresosAnuales) {
        this.ingresosAnuales = ingresosAnuales;
    }

    public int getNumeroSeguidores() {
        return numeroSeguidores;
    }

    public void setNumeroSeguidores(int numeroSeguidores) {
        this.numeroSeguidores = numeroSeguidores;
    }

    public String getRedSocialReferencia() {
        return redSocialReferencia;
    }

    public void setRedSocialReferencia(String redSocialReferencia) {
        this.redSocialReferencia = redSocialReferencia;
    }

    public void mostrarInfo()
    {
        super.mostrarInfoBasicaPersona();
        System.out.println("ingresosMensuales: " + ingresosAnuales);
        System.out.println("numeroSeguidores: " + numeroSeguidores);
        System.out.println("redSocialReferencia: " + redSocialReferencia);
    }
}
```

POO avanzada

Herencia

```
+ import java.time.LocalDate;[]

public class Gamer extends Influencer{
    String nick;
    String juegoReferencia;
    LocalDate fechaUltimoCampeonato;

    - Gamer()
    {
        nick = "--";
        juegoReferencia = "";
        fechaUltimoCampeonato = LocalDate.of(2020,1,1);
    }

    - Gamer(String docIdentidad, String nombrePersona, String apellidoPersona,
            int edadPersona,double ingresosMensuales, int numeroSeguidores,
            String redSocialReferencia,String nick,String juegoReferencia,LocalDate fechaUltimoCampeonato)
    {
        super(docIdentidad, nombrePersona, apellidoPersona,
              edadPersona, ingresosMensuales, numeroSeguidores,
              redSocialReferencia);
        this.nick = nick;
        this.juegoReferencia = juegoReferencia;
        this.fechaUltimoCampeonato = fechaUltimoCampeonato;
    }

    - public void mostrarInfo()
    {
        DateTimeFormatter formatoFecha = DateTimeFormatter.ofPattern("dd/MMM/yyyy");
        String fechaFormateada = fechaUltimoCampeonato.format(formatoFecha);
        super.mostrarInfo();
        System.out.println("Nick: " + nick);
        System.out.println("juegoReferencia: " + juegoReferencia);

        System.out.println("Fecha Último campeonato: " + fechaFormateada);
    }
}
```

POO avanzada

Herencia

```
public class Foodie extends Influencer{
    String tipoCocina;

    Foodie()
    {
        tipoCocina = "--";
    }

    Foodie(String docIdentidad, String nombrePersona, String apellidoPersona,
            int edadPersona,double ingresosMensuales, int numeroSeguidores,
            String redSocialReferencia,String tipoCocina)
    {
        super(docIdentidad, nombrePersona, apellidoPersona,
              edadPersona, ingresosMensuales, numeroSeguidores,
              redSocialReferencia);
        this.tipoCocina = tipoCocina;
    }

    public void mostrarInfo()
    {
        super.mostrarInfo();
        System.out.println("tipoCocina: " + tipoCocina);
    }
}

public class Celebrity extends Influencer{
    String profesion;

    Celebrity()
    {
        profesion = "--";
    }

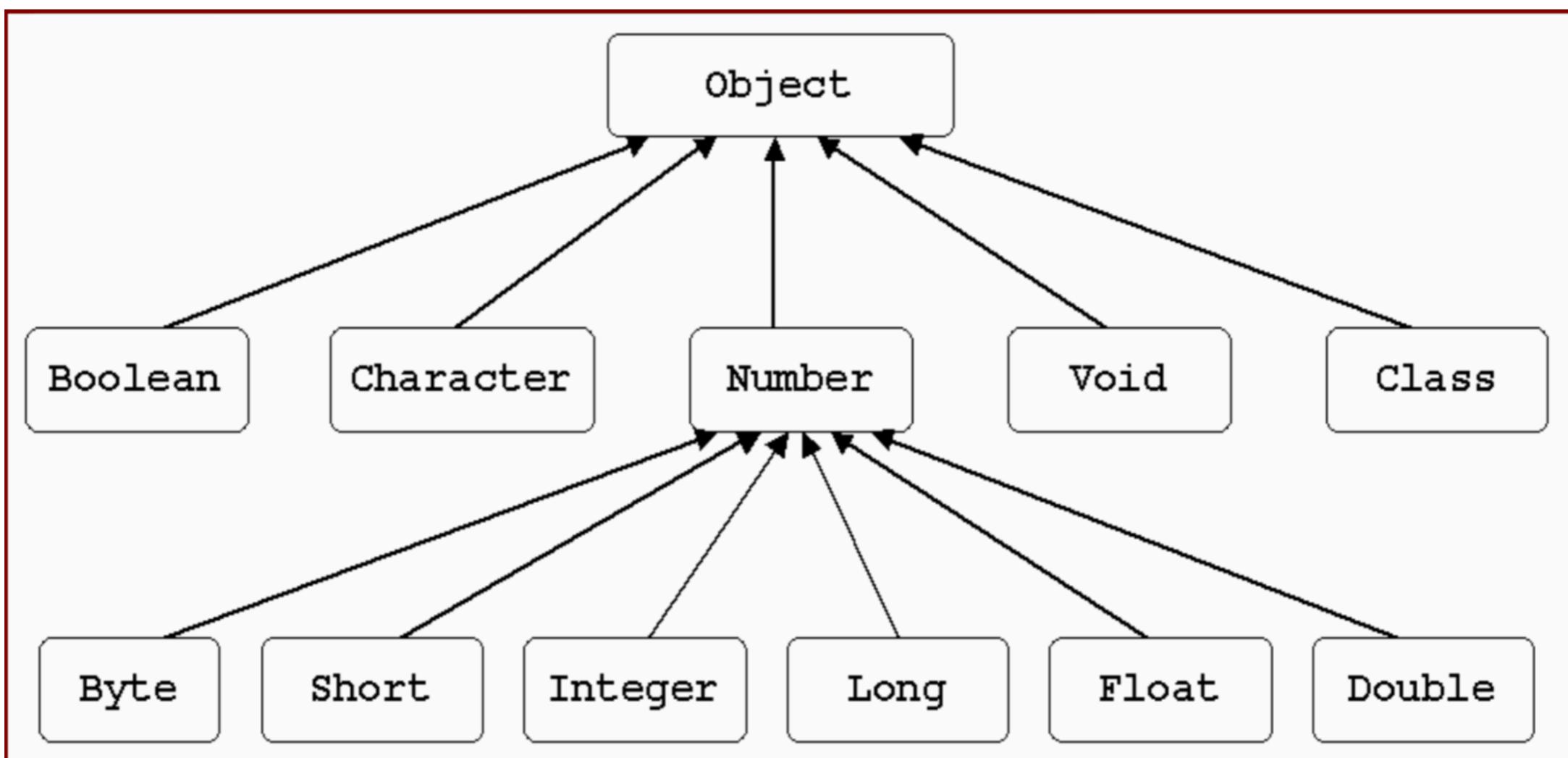
    Celebrity(String docIdentidad, String nombrePersona, String apellidoPersona,
              int edadPersona,double ingresosMensuales, int numeroSeguidores,
              String redSocialReferencia,String profesion)
    {
        super(docIdentidad, nombrePersona, apellidoPersona,
              edadPersona, ingresosMensuales, numeroSeguidores,
              redSocialReferencia);
        this.profesion = profesion;
    }

    public void mostrarInfo()
    {
        super.mostrarInfo();
        System.out.println("profesion: " + profesion);
    }
}
```

POO avanzada

Herencia

Independientemente de utilizar la palabra reservada `extends` en su declaración, todas las clases derivan de una superclase llamada `Object`. Ésta es la clase raíz de toda la jerarquía de clases de Java.



POO avanzada

Herencia

¿Podríamos usar métodos heredados de Object en nuestras clases? Por ejemplo:

```
cocheElectricos.metodoDeObject()
```

POO avanzada

Herencia

Como consecuencia de ello, todas las clases tienen algunos métodos heredados de la clase Object, algunos de sus métodos:

| Método | Función |
|------------|--|
| clone() | Genera una instancia a partir de otra de la misma clase. |
| equals() | Devuelve un valor lógico que indica si dos instancias de la misma clase son iguales. |
| toString() | Devuelve un String que contiene una representación como cadena de caracteres de una instancia. |
| finalize() | Finaliza una instancia durante el proceso de recogida de basura. |
| hashCode() | Devuelve una clave hash para la instancia |
| getClass() | Devuelve la clase a la que pertenece una instancia. |

POO avanzada

Herencia

¿Cuándo consideramos que un objeto es igual a otro? ¿Qué devuelve el siguiente código?

```
public class Persona {  
  
    private String nombre;  
    private int edad;  
  
    public Persona()  
    {  
        nombre = "Pepe";  
        edad = 30;  
    }  
  
    public Persona(String nombre, int edad)  
    {  
        this.nombre = nombre;  
        this.edad = edad;  
    }  
}
```

```
1 package ejemplosPOOAvanzada;  
2  
3  
4 public class EjEquals1  
5 {  
6  
7     public static void main(String[] args)  
8     {  
9         Persona persona1;  
10        Persona persona3 = new Persona("Ana",35);  
11        Persona persona4 = new Persona("Ana",35);  
12  
13        persona1 = persona3;  
14  
15        if (persona1 == persona3)  
16        {  
17            System.out.println("persona1 es igual que persona3");  
18        }  
19  
20        if (persona4 == persona3)  
21        {  
22            System.out.println("persona4 es igual que persona3");  
23        }  
24    }  
25  
26 }
```

POO avanzada

Herencia

Haz un ejemplo similar con la clase Libro

POO avanzada

Herencia

Para poder usar de forma adecuada equals() y que nos permita comprobar si dos objetos son iguales si tienen exactamente los mismos atributos debemos redefinir el método equals() en nuestras clases. Ver siguiente ejemplo:

POO avanzada

Herencia

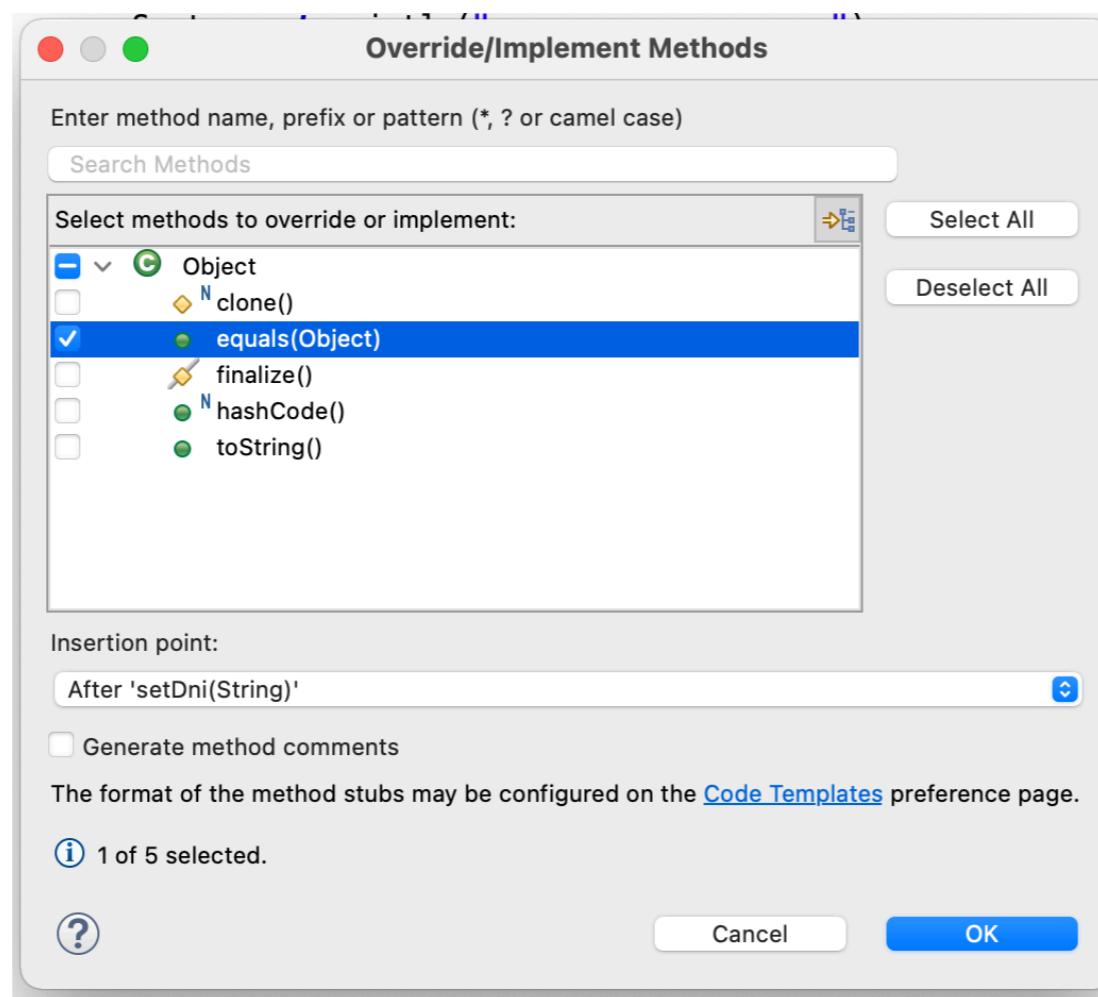
Si redefinimos el método equals en Persona:

```
public class Persona {  
  
    private String nombre;  
    private int edad;  
  
    public Persona()  
    {  
        nombre = "Pepe";  
        edad = 30;  
    }  
  
    public Persona(String nombre, int edad)  
    {  
        this.nombre = nombre;  
        this.edad = edad;  
    }  
  
    public boolean equals (Object objeto)  
    {  
        boolean igual = false;  
  
        if ((objeto!=null) && (objeto instanceof Persona))  
        {  
            if (((Persona)objeto).nombre.equals(nombre) && ((Persona)objeto).edad==edad)  
            {  
                igual = true;  
            }  
        }  
  
        return igual;  
    }  
  
}  
  
public class EjEquals1  
{  
  
    public static void main(String[] args)  
    {  
        Persona persona1;  
        Persona persona3 = new Persona("Ana",35);  
        Persona persona4 = new Persona("Ana",35);  
  
        persona1 = persona3;  
  
        if (persona1 == persona3)  
        {  
            System.out.println("persona1 es igual que persona3");  
        }  
  
        //if (persona4 == persona3)  
        if (persona4.equals(persona3))  
        {  
            System.out.println("persona4 es igual que persona3");  
        }  
    }  
}
```

POO avanzada

Herencia

Si en Eclipse, en nuestra clase Persona, botón derecho → source → Override/Implement vemos cómo Java detecta que Persona hereda de Object y nos permite elegir el método equals para sobreescribirlo



POO avanzada

Herencia

Haz un ejemplo similar con la clase Libro

POO avanzada

Herencia

toString()

toString() también está declarado en la clase Object. Convierte el objeto a un String, si hacemos println(objeto), se llama interanamente a toString de forma transparente.

Podemos por tanto llamarlo en cualquier objeto, observa el resultado:

```
package ejemplo_toString;

public class GestionPersonas {
    public static void main(String[] args) {
        Persona personal1 =new Persona("573948573Z","Cristina","López",25);
        System.out.println(personal1);
    }
}
```

```
package ejemplo_toString;

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



The screenshot shows a Java application console window. The title bar says "Console". The text area contains the output of the println statement from the GestionPersonas class. It shows the string representation of the Persona object, which includes its attributes: dni ("573948573Z"), nombre ("Cristina"), apellido ("López"), and edad (25). The output is preceded by the class name "ejemplo_toString.Persona@65ae6ba4".

```
<terminated> GestionPersonas [Java Application] /Library/J
ejemplo_toString.Persona@65ae6ba4
```

POO avanzada

Herencia

toString()

Como has visto, al llamar al método `toString()` definido por defecto se muestra una representación interna que no nos vale de mucho.

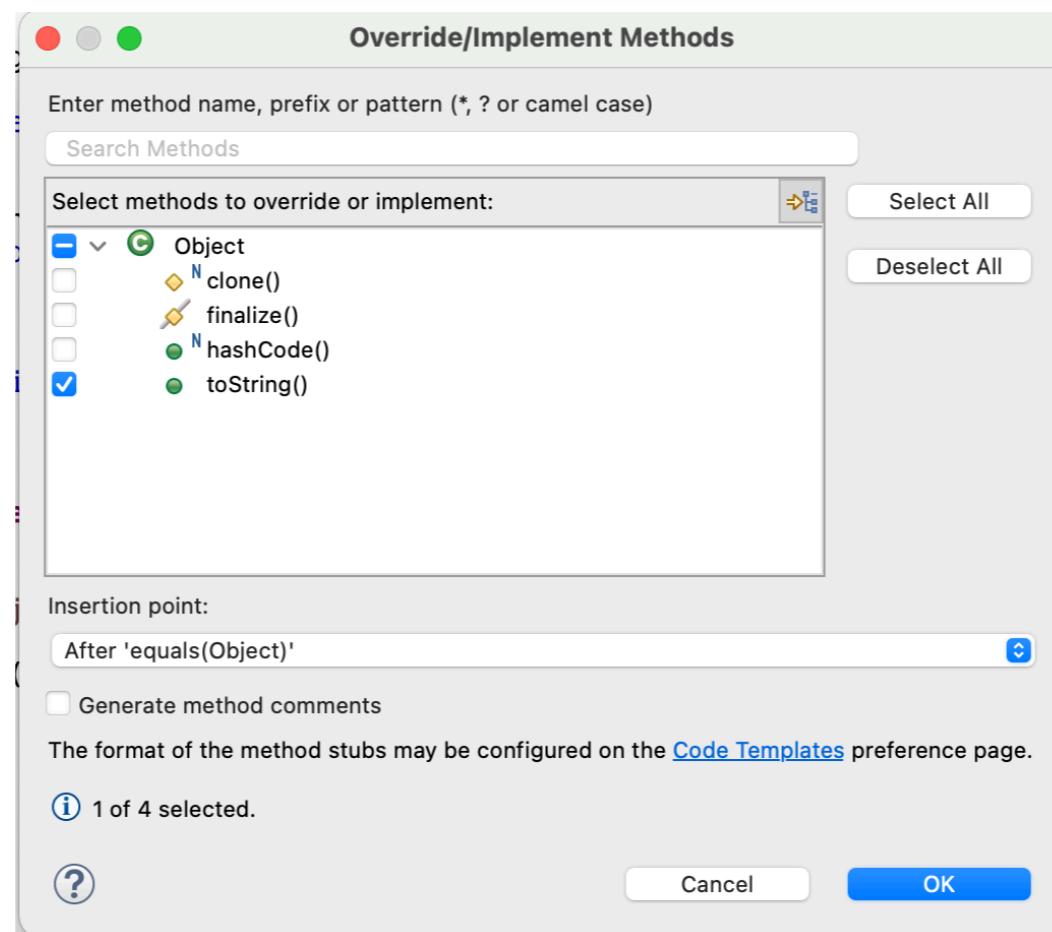
¿Cómo haríamos para que nos muestre información bien formateada de sus atributos?

POO avanzada

Herencia

toString()

Podemos sobreescribir el método `toString()` de la misma forma que hicimos con `equals()`



POO avanzada

Herencia

toString()

```
package ejemplo_toString;

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

@Override
public String toString() {
    String cadenaSalida = "---- Persona --- \n";
    cadenaSalida = cadenaSalida + "dni: " + this.dni + "\n";
    cadenaSalida = cadenaSalida + "nombre: " + this.nombre + "\n";
    cadenaSalida = cadenaSalida + "apellido: " + this.apellido + "\n";
    cadenaSalida = cadenaSalida + "edad: " + this.edad + "\n";
    cadenaSalida = cadenaSalida + "-----\n";
    return cadenaSalida;
}
```

```
package ejemplo_toString;

public class GestionPersonas {
    public static void main(String[] args) {
        Persona persona1 =new Persona("573948573Z","Cristina","López",25);
        System.out.println(persona1);
    }
}
```



Console X
<terminated> GestionPersonas [Java Application] /
---- Persona ---
dni: 573948573Z
nombre: Cristina
apellido: López
edad: 25

POO avanzada

Herencia

Haz un ejemplo similar con la clase Libro

POO avanzada

Herencia

Una **clase abstracta** es una clase de la que no se pueden crear objetos. Su utilidad consiste en permitir que otras clases hereden de ella. De esta forma, proporciona un modelo de referencia a seguir a la vez que una serie de métodos de utilidad general.

POO avanzada

Herencia

Una clase abstracta puede tener constructor aunque no se creen objetos.
Se usa para que las hijas puedan llamar al constructor de la madre para inicializar sus atributos.

POO avanzada

Herencia

Las clases abstractas se declaran empleando la palabra reservada `abstract`:

```
public abstract class ClaseAbst1 {  
}
```

POO avanzada

Herencia

Una clase abstracta puede componerse de varios atributos y métodos pero **debe tener, al menos, un método abstracto.**

Los **métodos abstractos** no se implementan en el código de la clase abstracta pero las clases descendientes de ésta han de implementarlos o volver a declararlos como abstractos (en cuyo caso la subclase también debe declararse como abstracta).

POO avanzada

Herencia

Para declarar un método abstracto debe indicarse el tipo de dato que devuelve y el número y tipo de parámetros:

```
abstract modificador tipo_retorno  
nombreMétodo( lista_parametros );
```

POO avanzada

Herencia

En el siguiente ejemplo se crea una clase abstracta llamada AbsFigura. Observa como se define en esa clase un método abstracto calcularArea() que se implementa en la clase hija.

Observa cómo en la clase abstracta se implementa un método mayorQue que llama a calcularArea() aunque éste no esté implementado en la clase padre sino en la hija.

Observa también cómo tiene un constructor aunque no se creen objetos de la clase padre.

POO avanzada

Herencia

Ejemplo

```
public abstract class AbsFigura
{
    private String nombre;

    public AbsFigura(String nombreFigura )
    {
        nombre = nombreFigura;
    }

    abstract public double calcularArea();

    final public boolean mayorQue(AbsFigura otra)
    {
        boolean resultado = false;

        if (otra != null)
        {
            resultado = calcularArea()>otra.calcularArea();
        }
        return resultado;
    }
}
```

POO avanzada

Herencia

Ejemplo

```
public class Rectangulo extends AbsFigura
{
    private double base;
    private double altura;

    public Rectangulo(double largo, double ancho)
    {
        super("Rectangulo");
        base=largo;
        altura=ancho;
    }

    public double calcularArea() //implementa el método abstracto
    {
        return base * altura;
    }
}
```

POO avanzada

Herencia

Ejemplo

```
public class PruebaAbst1 {  
  
    public static void main (String [] args )  
    {  
        Rectangulo rect1 = new Rectangulo(12.5, 23.7);  
        Rectangulo rect2 = new Rectangulo(8.6, 33.1);  
  
        System.out.println("Area de r1 = " + rect1.calcularArea());  
        System.out.println("Area de r2 = " + rect2.calcularArea());  
  
        if (rect1.mayorQue(rect2))  
            System.out.println("El rectangulo de mayor area es rect1");  
        else  
            System.out.println("El rectangulo de mayor area es rect2");  
    }  
}
```

POO avanzada

Herencia

Ejercicio: Amplía el ejercicio anterior creando una clase triángulo que hereda de AbsFigura. Crea en el programa principal rectángulos y triángulos y compara si unos son mayores que otros

POO avanzada

Herencia

```
public class Triangulo extends AbsFigura
{
    private double base;
    private double altura;

    public Triangulo(double largo, double ancho)
    {
        super("Triángulo");
        base=largo;
        altura=ancho;
    }

    public double calcularArea() //implementa el método abstracto
    {
        return base * altura/2;
    }
}
```

POO avanzada

Herencia

```
public class PruebaAbst1 {  
  
    public static void main (String [] args )  
    {  
        Rectangulo rect1 = new Rectangulo(12.5, 23.7);  
        Triangulo triang1 = new Triangulo(8.6, 33.1);  
  
        System.out.println("Area de rect1 = " + rect1.calcularArea());  
        System.out.println("Area de triang = " + triang1.calcularArea());  
  
        if (rect1.mayorQue(triang1))  
            System.out.println("El rectangulo tiene mayor área");  
        else  
            System.out.println("El triángulo tiene mayor área");  
    }  
  
}
```

POO avanzada

Herencia

Ejercicio:

Tenemos que realizar una aplicación para un proveedor de ordenadores. Se gestionan ordenadores portátiles y workstations.

El programa principal debe crear un array de 10 elementos, se insertarán desde código 5 elementos (3 portátiles y 2 workstations) dos de ellos con los mismos atributos.

El programa debe mostrar los elementos repetidos.

Se utizarán los métodos equals() y toString() para comparar y para mostrar la información de un objeto.

Los portátiles manejarán los atributos modelo, modelo de procesador, capacidad de Ram, precio, tamaño de pantalla y potencia de procesador.

Los workstations manejarán los atributos modelo, modelo de procesador, capacidad de Ram, precio, capacidad de disco y número de procesadores.

Debe haber un método que nos permita fijar el precio de un ordenador. Para ello este método multiplica un factor de cálculo interno (0,23) por la potencia relativa del ordenador.

La potencia relativa, para el caso de los portátiles se calcula multiplicando la potencia por el tamaño de la pantalla en pulgadas por el tamaño de la RAM. Para el caso de las worstations se calcula multiplicando la capacidad del disco por el número de procesadores por el tamaño de la RAM por 100

Debe haber una clase abstracta.

```
public class ProveedorOrdenadores {  
    public static void main(String[] args) {  
        Ordenador[] arrayOrdenadores = new Ordenador[10];  
  
        arrayOrdenadores[0] = new OrdPortatil("Ultrium", "i9", 16, 15, 245);  
        arrayOrdenadores[1] = new WorkStation("Sulkium", "M1", 16, 5, 10);  
        arrayOrdenadores[2] = new OrdPortatil("Natiuum", "i7", 8, 14, 180);  
        arrayOrdenadores[3] = new WorkStation("Strium", "M2", 32, 4, 8);  
        arrayOrdenadores[4] = new OrdPortatil("Natiuum", "i7", 8, 14, 180); //repetido  
        arrayOrdenadores[5] = new WorkStation("Strium", "M2", 32, 4, 8); //repetido  
  
        arrayOrdenadores[0].fijarPrecio();  
        arrayOrdenadores[1].fijarPrecio();  
        arrayOrdenadores[2].fijarPrecio();  
        arrayOrdenadores[3].fijarPrecio();  
        arrayOrdenadores[4].fijarPrecio();  
        arrayOrdenadores[5].fijarPrecio();  
  
        mostrarRepetidos(arrayOrdenadores);  
    }  
  
    private static void mostrarRepetidos(Ordenador[] listaOrdenadores)  
    {  
        for (int i=0; i<listaOrdenadores.length; i++)  
        {  
            for (int j = i+1; j < listaOrdenadores.length; j++)  
            {  
                if ((listaOrdenadores[i] != null) && (listaOrdenadores[j] != null))  
                {  
                    if (listaOrdenadores[i].equals(listaOrdenadores[j]))  
                    {  
                        System.out.println("Ordenador repetido: \n");  
                        System.out.println(listaOrdenadores[i]);  
                    }  
                }  
            }  
        }  
    }  
}
```

```
public abstract class Ordenador {  
    private String modelo;  
    private String procesador;  
    private int capacidadRam;  
    private double precio;  
  
    Ordenador(String modelo, String procesador, int capacidadRam)  
    {  
        this.modelo = modelo;  
        this.procesador = procesador;  
        this.capacidadRam = capacidadRam;  
        this.precio = 0;  
    }  
    ──────────► abstract public double calcularPotenciaRelativa();  
  
    public void fijarPrecio()  
    {  
        final double FACTOR_CALCULO_PRECIO = 0.23;  
        this.precio = calcularPotenciaRelativa() * FACTOR_CALCULO_PRECIO;  
    }  
  
    @Override  
    public boolean equals(Object objeto) {  
        boolean igual = false;  
  
        if ((objeto!=null) && (objeto instanceof Ordenador))  
        {  
            if ( ((Ordenador)objeto).modelo.equals(this.modelo) &&  
                ((Ordenador)objeto).procesador.equals(this.procesador) &&  
                ((Ordenador)objeto).capacidadRam == this.capacidadRam &&  
                ((Ordenador)objeto).precio == this.precio )  
            {  
                igual = true;  
            }  
        }  
  
        return igual;  
    }  
  
    @Override  
    public String toString() {  
        String cadenaSalida = "---- Ordenador --- \n";  
  
        cadenaSalida = cadenaSalida + "modelo: " + this.modelo + "\n";  
        cadenaSalida = cadenaSalida + "procesador: " + this.procesador + "\n";  
        cadenaSalida = cadenaSalida + "capacidadRam: " + this.capacidadRam + "\n";  
        cadenaSalida = cadenaSalida + "precio: " + this.precio + "\n";  
  
        cadenaSalida = cadenaSalida + "-----\n";  
  
        return cadenaSalida;  
    }  
}
```

```
public class OrdPortatil extends Ordenador{
    private double tamanoPantalla;
    private double potenciaProcesador;

    OrdPortatil(String modelo, String procesador, int capacidadRam,
                double tamanoPantalla, double potenciaProcesador)
    {
        super(modelo,procesador,capacidadRam);
        this.tamanoPantalla = tamanoPantalla;
        this.potenciaProcesador = potenciaProcesador;
    }

    public double calcularPotenciaRelativa()
    {
        return tamanoPantalla * this.getCapacidadRam() * potenciaProcesador;
    }

    @Override
    public boolean equals(Object objeto) {
        boolean igual = false;

        if ((objeto!=null) && (objeto instanceof OrdPortatil) && super.equals(objeto))
        {
            if(
                ((OrdPortatil)objeto).tamanoPantalla == this.tamanoPantalla &&
                ((OrdPortatil)objeto).potenciaProcesador == this.potenciaProcesador)
            {
                igual = true;
            }
        }

        return igual;
    }

    @Override
    public String toString() {
        String cadenaSalida = "---- OrdPortatil --- \n";
        cadenaSalida = cadenaSalida + super.toString();

        cadenaSalida = cadenaSalida + "tamanoPantalla: " + this.tamanoPantalla + "\n";
        cadenaSalida = cadenaSalida + "potenciaProcesador: " + this.potenciaProcesador + "\n";
        cadenaSalida = cadenaSalida + "-----\n";

        return cadenaSalida;
    }
}
```

```
public class WorkStation extends Ordenador{

    private double capacidadDisco;
    private double numProcesadores;

    WorkStation(String modelo, String procesador, int capacidadRam,
                double capacidadDisco, double numProcesadores)
    {
        super(modelo, procesador, capacidadRam);
        this.capacidadDisco = capacidadDisco;
        this.numProcesadores = numProcesadores;
    }

    public double calcularPotenciaRelativa()
    {
        return capacidadDisco * this.getCapacidadRam() * numProcesadores * 100;
    }

    @Override
    public boolean equals(Object objeto) {
        boolean igual = false;

        if ((objeto!=null) && (objeto instanceof WorkStation) && super.equals(objeto))
        {
            if(
                ((WorkStation)objeto).capacidadDisco == this.capacidadDisco &&
                ((WorkStation)objeto).numProcesadores == this.numProcesadores)
            {
                igual = true;
            }
        }

        return igual;
    }

    @Override
    public String toString() {
        String cadenaSalida = "---- WorkStation --- \n";
        cadenaSalida = cadenaSalida + super.toString();

        cadenaSalida = cadenaSalida + "capacidadDisco: " + this.capacidadDisco + "\n";
        cadenaSalida = cadenaSalida + "numProcesadores: " + this.numProcesadores + "\n";
        cadenaSalida = cadenaSalida + "-----\n";

        return cadenaSalida;
    }
}
```

POO avanzada

Clases Finales

Una clase declarada con la palabra reservada **final** no puede tener clases descendientes. Por ejemplo, la clase predefinida de Java Math o String está declarada como final.

```
public final class ClaseFinal{  
}
```

POO avanzada

Clases Finales

Un mecanismo que los hackers utilizan para atacar sistemas es crear subclases de una clase.

La clase `String` del paquete `java.lang` es una clase final sólo por esta razón. La clase `String` es tan vital para la operación del compilador y del intérprete que el sistema Java debe garantizar que siempre que un método o un objeto utilicen un `String`, obtenga un objeto `java.lang.String` y no algún otro string. Esto asegura que ningún string tendrán propiedades extrañas, inconsistentes o indeseables.

POO avanzada

Java en línea de comandos

Hasta ahora hemos utilizado Eclipse.
Veamos cómo podríamos programar en Java usando el terminal del ordenador:

POO avanzada

Java en línea de comandos

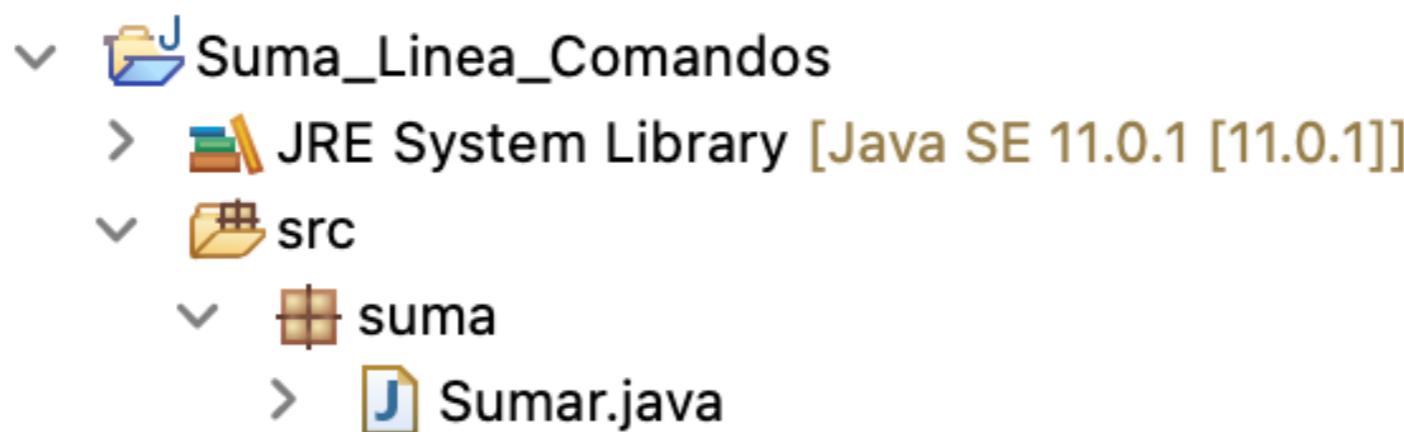
Debemos tener claras las diferencias entre JDK, JRE y JVM.

<https://www.guru99.com/difference-between-jdk-jre-jvm.html>

POO avanzada

Java en línea de comandos

En nuestros proyectos en Eclipse podemos ver cómo incluyen el JRE



POO avanzada

Java en línea de comandos

Veamos un programa sencillo como el siguiente:

```
public class Sumar {  
    public static void main(String[] args) {  
        int resultadoSuma = 0;  
        int numero1 = Integer.parseInt(args[0]);  
        int numero2 = Integer.parseInt(args[1]);  
        resultadoSuma = numero1 + numero2;  
        System.out.println(resultadoSuma);  
    }  
}
```

Parámetros de entrada (array de String)



Observa que tenemos que convertir de String a int.

POO avanzada

Java en línea de comandos

Echa un vistazo a las clases Integer, Double. Y más concretamente a:

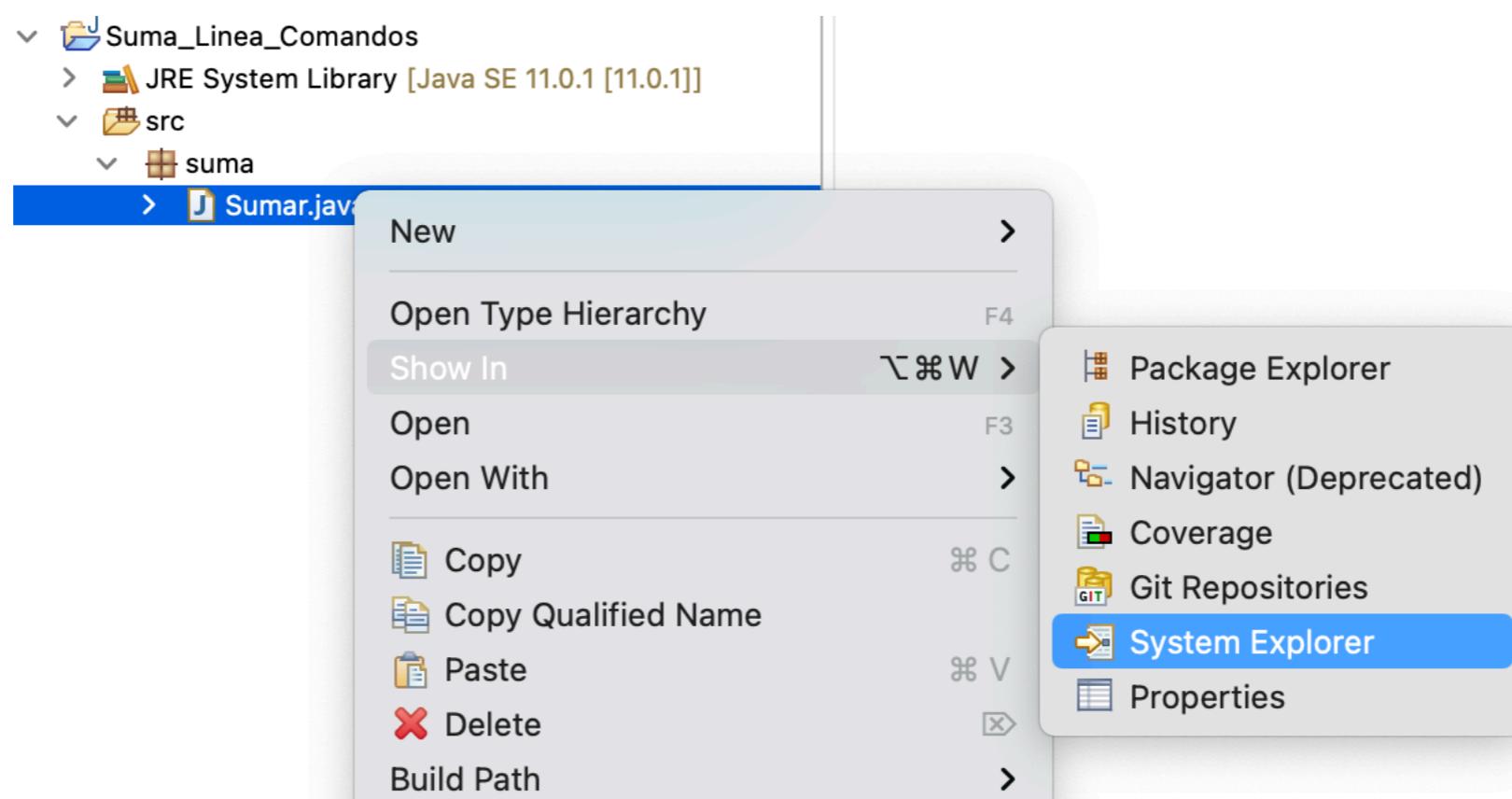
Integer.parseInt: <https://docs.oracle.com/javase/8/docs/api/java/lang/Integer.html#parseInt-java.lang.String->

Double.parseDouble: <https://docs.oracle.com/javase/8/docs/api/java/lang/Double.html#parseDouble-java.lang.String->

POO avanzada

Java en línea de comandos

Veamos ahora cómo podríamos compilar y ejecutar nuestro programa por la línea de comandos:



POO avanzada

Java en línea de comandos

Veamos ahora cómo podríamos compilar y ejecutar nuestro programa por la línea de comandos:

```
-rw-r--r-- 1 juanluisballesterosfernande staff 280 Feb  8 12:47 Sumar.java
[MBP-de-Juan:suma juanluisballesterosfernande$ javac -d . Sumar.java
[MBP-de-Juan:suma juanluisballesterosfernande$ ls -l
total 8
-rw-r--r-- 1 juanluisballesterosfernande staff 280 Feb  8 12:47 Sumar.java
drwxr-xr-x@ 3 juanluisballesterosfernande staff   96 Feb  8 12:48 suma
[MBP-de-Juan:suma juanluisballesterosfernande$ java suma.Sumar 3 4
7
[MBP-de-Juan:suma juanluisballesterosfernande$ java suma.Sumar 1000 2000
3000
```

POO avanzada

Java en línea de comandos

Modifica el programa del proveedor de ordenadores para que se puedan coger los datos de un ordenador desde la línea de comandos, a continuación se crea y se incluye en el array. Luego se muestra la lista de ordenadores del proveedor.

Compila el programa y ejecútalo desde la línea de comandos

POO avanzada

Java en línea de comandos

Recogemos
parámetros de
entrada

```
public class ProveedorOrdenadores {  
  
    public static void main(String[] args) {  
        Ordenador[] arrayOrdenadores = new Ordenador[10];  
  
        arrayOrdenadores[0] = new OrdPortatil(args[0], args[1],  
                                         Integer.parseInt(args[2]),  
                                         Integer.parseInt(args[3]),  
                                         Integer.parseInt(args[4]));  
  
        arrayOrdenadores[1] = new OrdPortatil("Ultrium", "i9", 16, 15, 245);  
        arrayOrdenadores[2] = new WorkStation("Sulkium", "M1", 16, 5, 10);  
        arrayOrdenadores[3] = new OrdPortatil("Natum", "i7", 8, 14, 180);  
  
        arrayOrdenadores[0].fijarPrecio();  
        arrayOrdenadores[1].fijarPrecio();  
        arrayOrdenadores[2].fijarPrecio();  
        arrayOrdenadores[3].fijarPrecio();  
  
        mostrarOrdenadores(arrayOrdenadores);  
    }  
  
    private static void mostrarOrdenadores(Ordenador[] listaOrdenadores)  
    {  
        for (int i=0;i<listaOrdenadores.length;i++)  
        {  
            if ((listaOrdenadores[i] != null) )  
            {  
                System.out.println(listaOrdenadores[i]);  
            }  
        }  
    }  
}
```

POO avanzada

Java en línea de comandos

```
juanluisballesterosfernande$ ls -l
42 Feb  8 11:42 OrdPortatil.java
09 Feb  8 12:06 Ordenador.java
06 Feb  8 11:59 ProveedorOrdenadores.java
08 Feb  8 11:42 WorkStation.java
```

Compilamos



```
juanluisballesterosfernande$ javac -d . *
```

```
juanluisballesterosfernande$ ls -l
42 Feb  8 11:42 OrdPortatil.java
09 Feb  8 12:06 Ordenador.java
06 Feb  8 11:59 ProveedorOrdenadores.java
08 Feb  8 11:42 WorkStation.java
92 Feb  8 12:23 c_abstractas_ordenadores_linea_comandos
```

```
juanluisballesterosfernande$ ls -l c_abstractas_ordenadores_linea_comandos
598 Feb  8 12:23 OrdPortatil.class
!62 Feb  8 12:23 Ordenador.class
!86 Feb  8 12:23 ProveedorOrdenadores.class
503 Feb  8 12:23 WorkStation.class
```

POO avanzada

Java en línea de comandos

```
[MBP-de-Juan:c_abstractas_ordenadores_linea_comandos juanluisballesterosfernande$ java c_abstractas_ordenadores_linea_comandos.ProveedorOrdenadores pentium p5 66 67 68
---- OrdPortatil ---
---- Ordenador ---
modelo: pentium
procesador: p5
capacidadRam: 66
precio: 69160.08
-----
tamanoPantalla: 67.0
potenciaProcesador: 68.0
-----

---- OrdPortatil ---
---- Ordenador ---
modelo: Ultrium
procesador: i9
capacidadRam: 16
precio: 13524.0
-----
tamanoPantalla: 15.0
potenciaProcesador: 245.0
-----

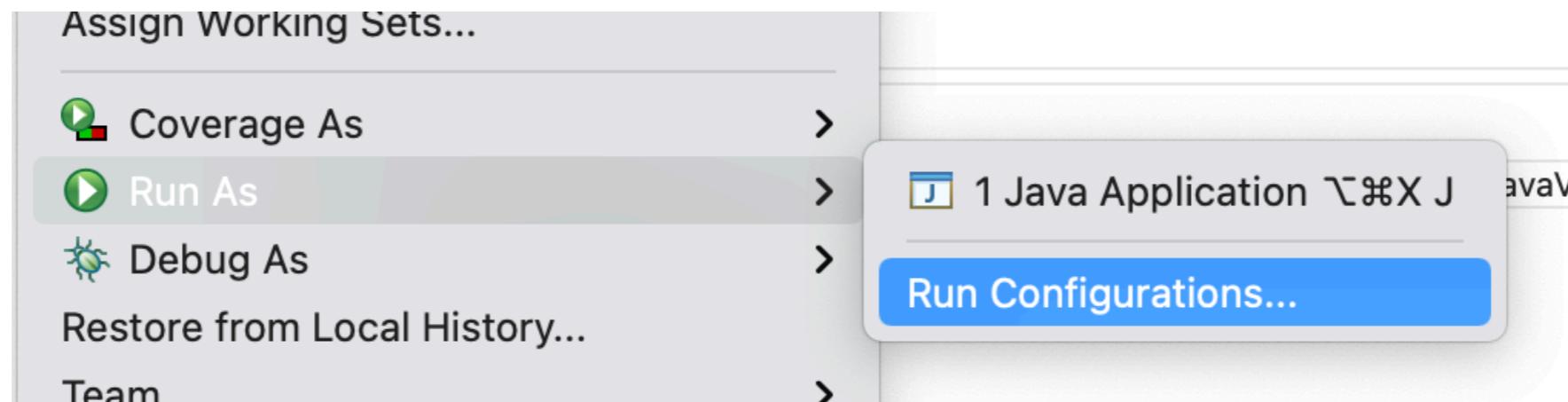
---- WorkStation ---
---- Ordenador ---
modelo: Sulkiun
procesador: M1
capacidadRam: 16
precio: 18400.0
-----
capacidadDisco: 5.0
numProcesadores: 10.0
-----

---- OrdPortatil ---
---- Ordenador ---
modelo: Natiun
procesador: i7
capacidadRam: 8
precio: 4636.8
-----
tamanoPantalla: 14.0
potenciaProcesador: 180.0
-----
```

POO avanzada

Java en línea de comandos

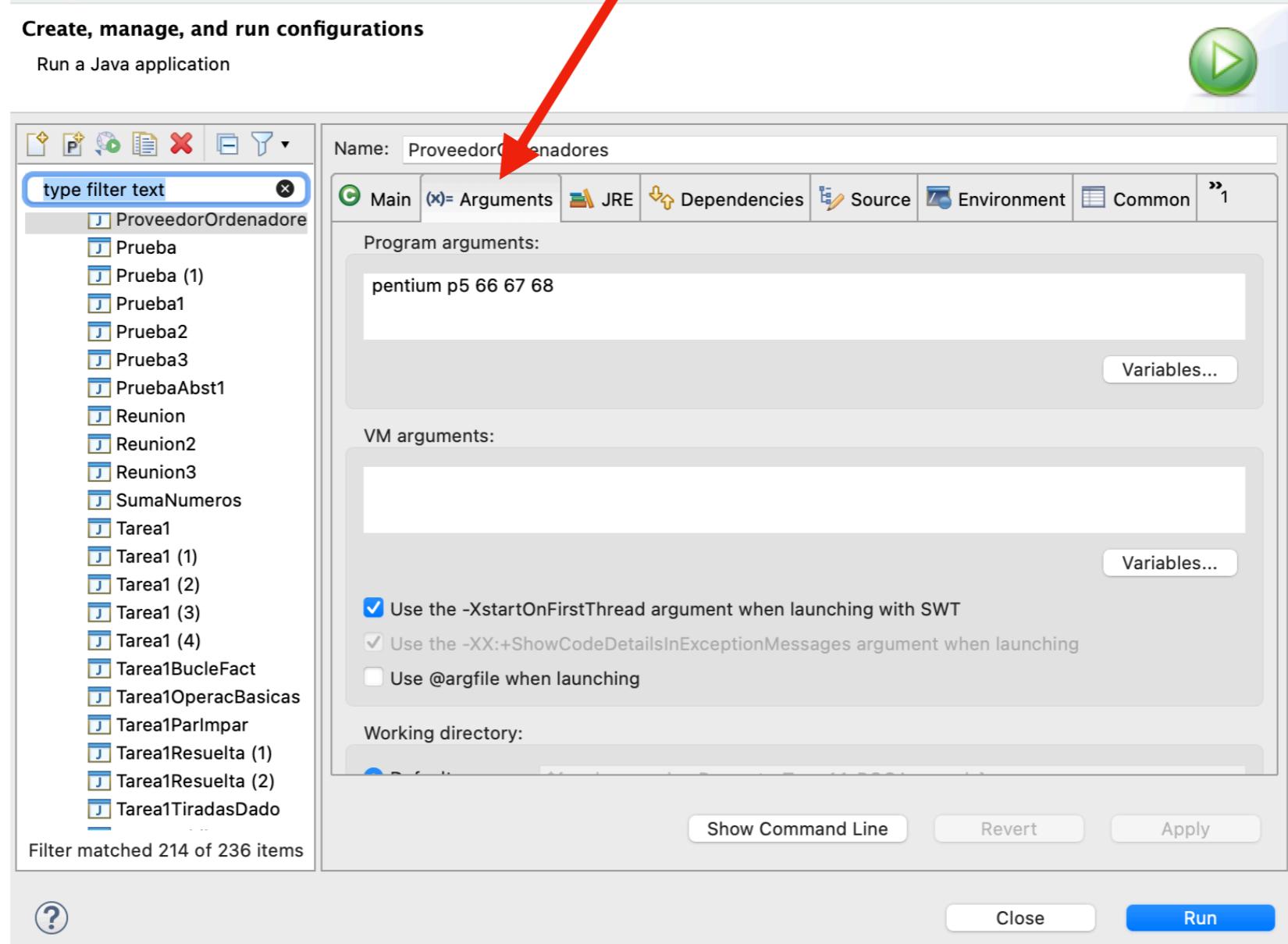
Podemos simular el paso de parámetros desde línea de comandos desde Eclipse:



POO avanzada

Java en línea de comandos

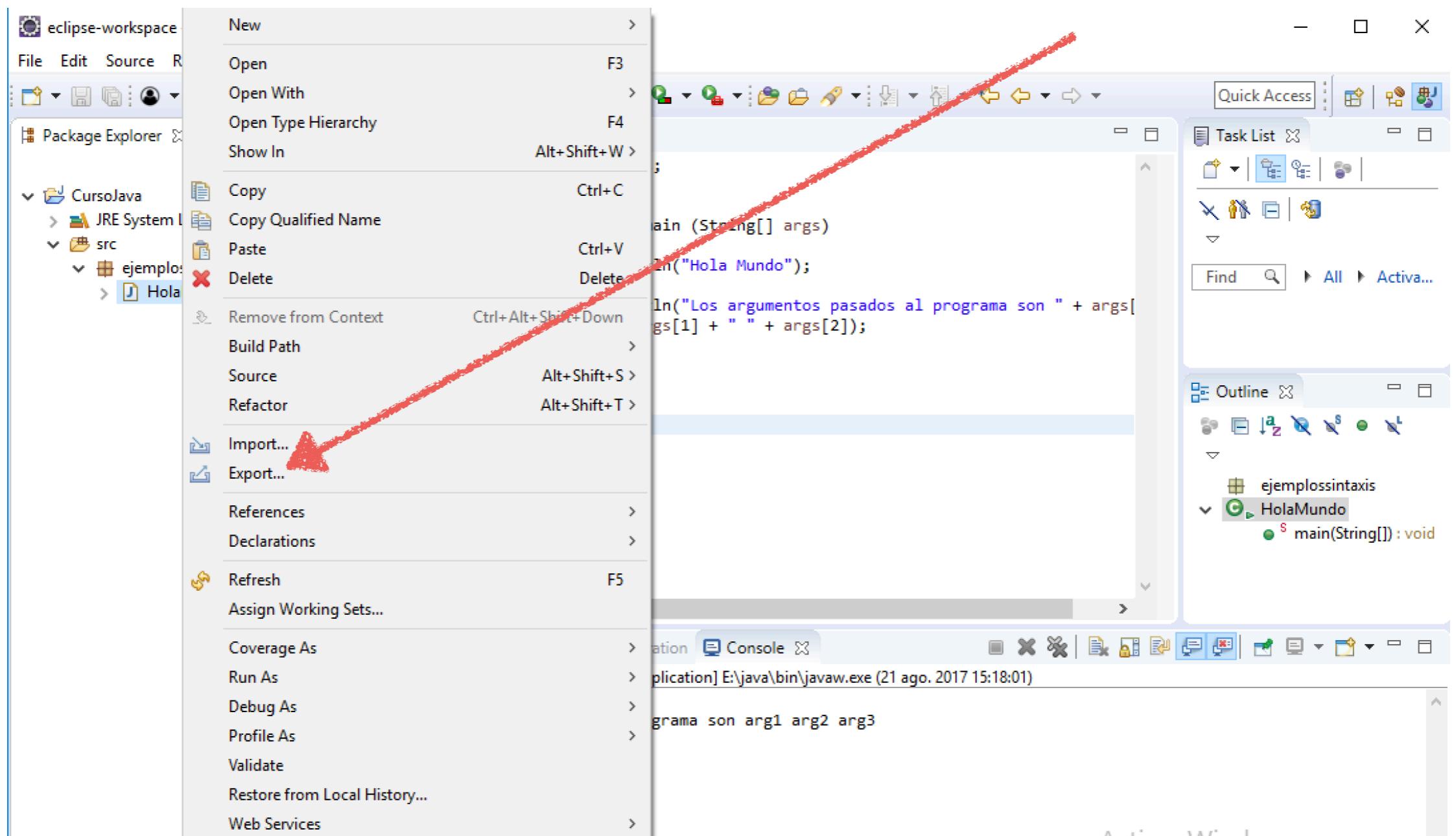
Podemos simular el paso de parámetros desde línea de comandos desde Eclipse:



POO avanzada

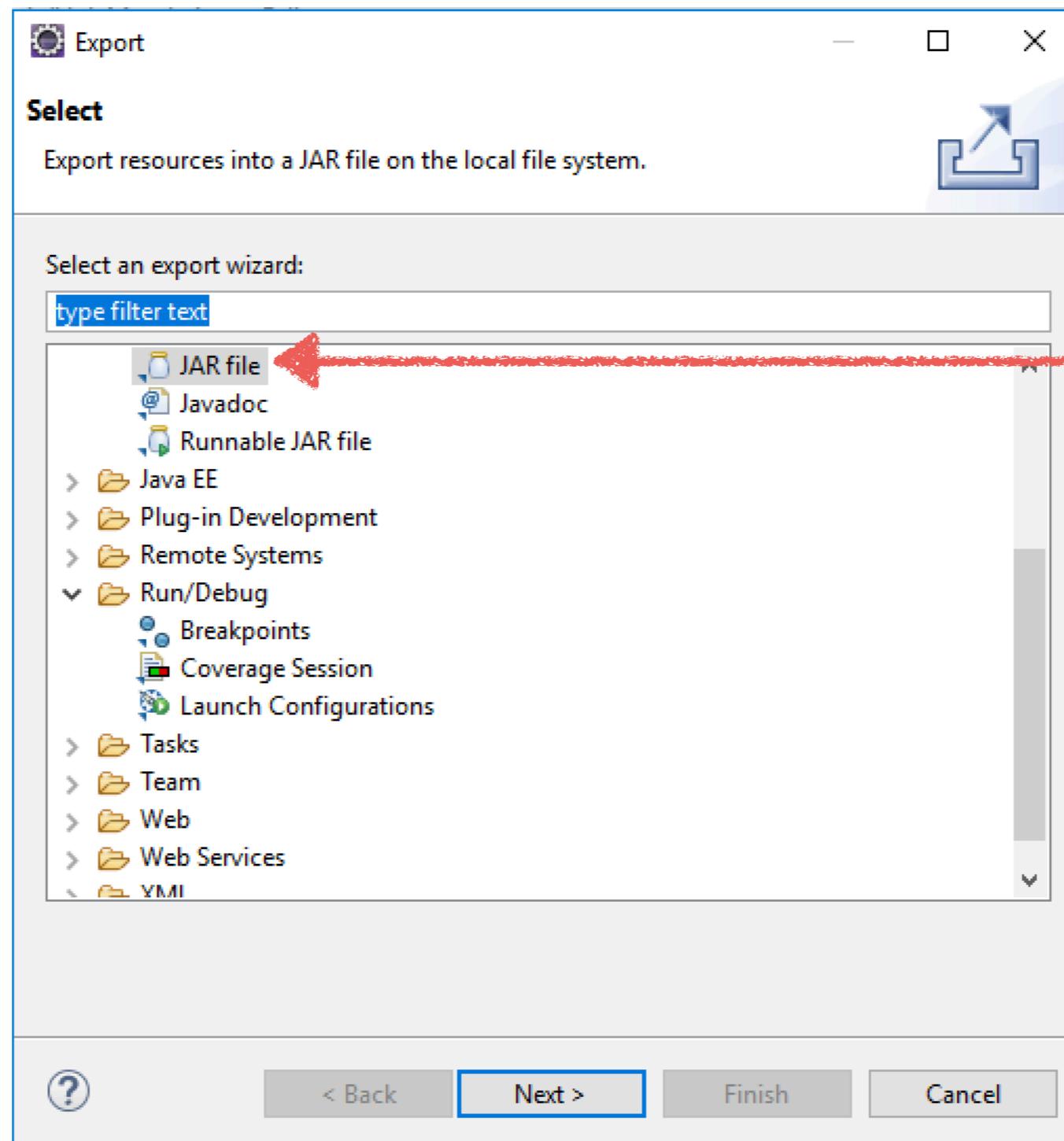
ficheros .jar

Podemos empaquetar nuestro programa en un fichero .jar:



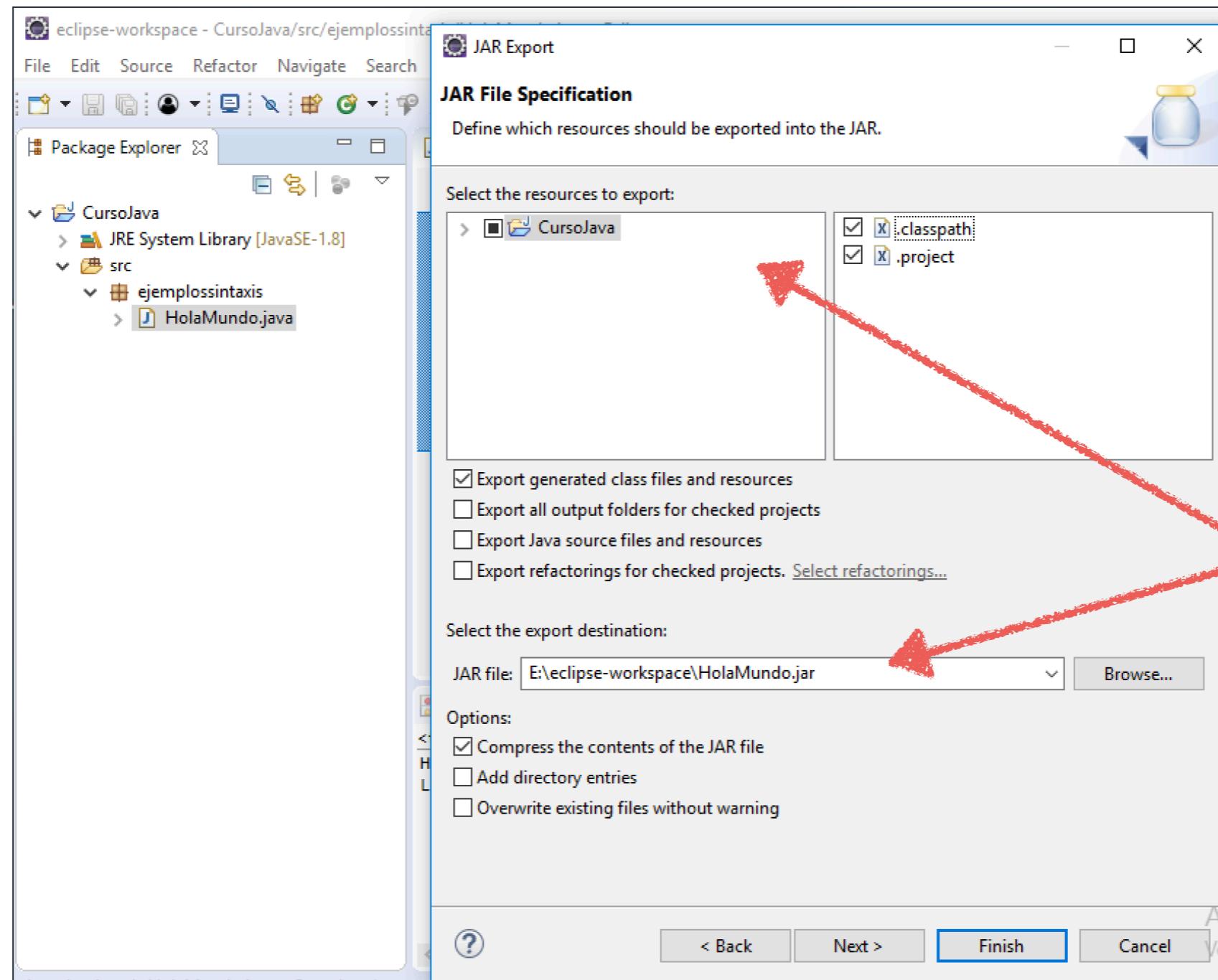
POO avanzada

ficheros .jar



POO avanzada

ficheros .jar



POO avanzada

Módulos

- En java 9 se introdujo el concepto de módulos para organizar mejor el código en programas muy complejos. Un módulo agrupa varios paquetes:

<https://www.arquitecturajava.com/java-9-modules/>

POO avanzada

Ejercicio clases abstractas

Necesitamos implementar un programa que gestione un puerto en el que atracan barcos pesqueros y barcos mercantes.

Los datos que se gestionan para cualquier tipo de barco son la matrícula, los metros de eslora, el número de tripulantes y el número de pasajeros. Para el caso de los mercantes se debe gestionar también la carga (en kg), mientras que para los veleros se debe tener en cuenta el número de camarotes y el precio del alquiler por día.

El puerto dispone de 60 atraques. Para los barcos atracados se realizan una tarea de gestión rutinaria que consiste en mostrar la información de cada barco, el ratio de pasajeros/tripulantes de cada barco y el valor económico de cada uno de ellos.

El ratio de pasajeros/tripulantes se calcula igual para cualquier tipo de barco.

Para el caso de los barcos mercantes el valor económico se corresponde con los metros de eslora multiplicado por 3500 y por la carga en kg.

Para el caso de los barcos veleros el valor económico se corresponde con los metros de eslora multiplicado por 5500 y el número de tripulantes.

Utilizad arrays normales. Cread los diferentes objetos en el código, no es necesario pedir datos al usuario.

Se debe crear al menos una clase abstracta.

Ejercicio clases abstractas

```
package examen2eva;

public class Puerto {

    // static Barco [] ArrayBarco = new Barco[60];
    public static void main(String[] args) {
        final int ATRAQUES = 60;
        Barco [] ArrayBarco = new Barco[ATRAQUES];

        anadirBarcos(ArrayBarco);
        mostrarInfo(ArrayBarco);

    }

    public static void anadirBarcos(Barco ArrayBarco[]) {
        ArrayBarco[0] = new Velero("sdadfgf", 23, 40, 999, 20, 40);
        ArrayBarco[1] = new Velero("sdg", 234, 34, 64, 56, 75);
        ArrayBarco[2] = new Mercante("dfdsg", 35, 346, 346, 1);
        ArrayBarco[3] = new Mercante("gfhkj", 56, 43, 46, 7);
    }

    public static void mostrarInfo(Barco ArrayBarco[]) {

        for(int i = 0; i<ArrayBarco.length;i++)
        {
            if(ArrayBarco[i] != null)
            {
                System.out.println(ArrayBarco[i]);
                System.out.println("RATIO PASAJEROS/TRIPULANTES "+ArrayBarco[i].ratioPasajerosTripulantes());
                System.out.println("PRECIO ESTIMADO "+ ArrayBarco[i].estimarPrecio());
            }
        }

    }
}
```

Ejercicio clases abstractas

```
package examen2eva;

public abstract class Barco {

    protected String matricula;
    protected int metrosEslora;
    protected int numTribulantes;
    protected int numPasajeros;

    public Barco(String matricula, int metrosEslora, int numTribulantes, int numPasajeros) {

        this.matricula = matricula;
        this.metrosEslora = metrosEslora;
        this.numTribulantes = numTribulantes;
        this.numPasajeros = numPasajeros;
    }

    @Override
    public String toString() {
        return "Barco [matricula=" + matricula + ", metrosEslora=" + metrosEslora + ", numTribulantes=" + numTribulantes + ", numPasajeros=" + numPasajeros + "]";
    }

    public double ratioPasajerosTripulantes() {

        return numPasajeros/numTribulantes;
    }

    abstract int estimarPrecio();
}
```

Ejercicio clases abstractas

```
package examen2eva;

public class Mercante extends Barco {

    public int carga;

    public Mercante(String matricula, int metrosEslora, int numTribulantes, int numPasajeros, int numCoches) {
        super(matricula, metrosEslora, numTribulantes, numPasajeros);
        this.carga = carga;
    }

    @Override
    public String toString() {
        return "Mercante [carga=" + carga + ", matricula=" + matricula + ", metrosEslora=" + metrosEslora +
               ", numTribulantes=" + numTribulantes + ", numPasajeros=" + numPasajeros + ", numCoches=" +
               numCoches + "]";
    }

    public int estimarPrecio() {
        return carga * metrosEslora;
    }
}
```

Ejercicio clases abstractas

```
package examen2eva;

public class Velero extends Barco {

    public int camarotes;
    public int alquilerDia;

    public Velero(String matricula, int metrosEslora, int numTribulantes, int numPasajeros, int
        super(matricula, metrosEslora, numTribulantes, numPasajeros);
        this.camarotes = camarotes;
        this.alquilerDia = alquilerDia;
    }

    @Override
    public String toString() {
        return "Velero [camarotes=" + camarotes + ", alquilerDia=" + alquilerDia + ", matricula=" +
            + ", metrosEslora=" + metrosEslora + ", numTribulantes=" + numTribulantes + ",
            + numPasajeros + "]";
    }

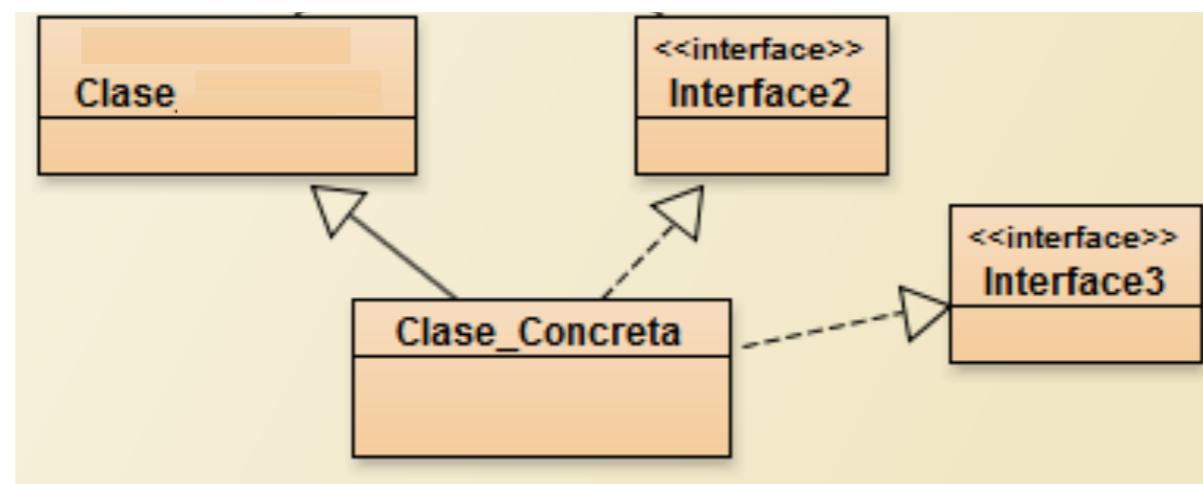
    public int estimarPrecio() {
        return alquilerDia * metrosEslora;
    }
}
```

POO avanzada

Interfaces

Recordemos que una clase sólo puede heredar de una única clase madre. Esto tiene ciertas limitaciones que se resuelven gracias a los interfaces.

Una clase puede **heredar** sólo de una clase (concepto) pero puede **implementar** varios interfaces (comportamiento)



POO avanzada

Interfaces

Una **interfaz** es una especie de plantilla para que sea utilizada por otras clases. Normalmente una interfaz se compone de un conjunto de declaraciones de cabeceras de métodos (sin implementar, de forma similar a un método abstracto) que especifican un protocolo de **comportamiento**.

POO avanzada

Interfaces

```
1
2
3 public interface Relacionable {
4     boolean esMayorQue(Relacionable param);
5     boolean esMenorQue(Relacionable param);
6     boolean esIgualQue(Relacionable param);
7 }
```

Observa que param es de tipo Relacionable

POO avanzada

Diferencias entre una clase abstracta y un Interfaz

- Todos los métodos de una **interfaz** se declaran implícitamente como abstractos y públicos.
- Una **clase abstracta** no puede implementar los métodos declarados como abstractos, una **interfaz** no puede implementar ningún método (ya que todos son abstractos).

POO avanzada

Diferencias entre una clase abstracta y un Interfaz

La interface puede definirse public o sin modificador de acceso, y tiene el mismo significado que para las clases. Normalmente utilizaremos public. Si tiene el modificador public el archivo .java que la contiene debe tener el mismo nombre que la interfaz.

Igual que las clases, al compilar el archivo .java de la interface se genera un archivo .class

POO avanzada

Interfaces

Las interfaces no pueden ser instanciadas, solo pueden ser implementadas por clases o extendidas por otras interfaces.

A diferencia de las clases que pueden heredar de una sola clase, las interfaces en Java pueden heredar de varias interfaces.

POO avanzada

Interfaces

Los nombres de las interface suelen acabar en **able** aunque no es necesario: configurable, arrancable, dibujable, comparable, clonable, etc. De esta forma se indica que estamos definiendo un comportamiento, NO un concepto

Una interfaz puede **heredar** de otra interfaz padre

POO avanzada

Interfaces

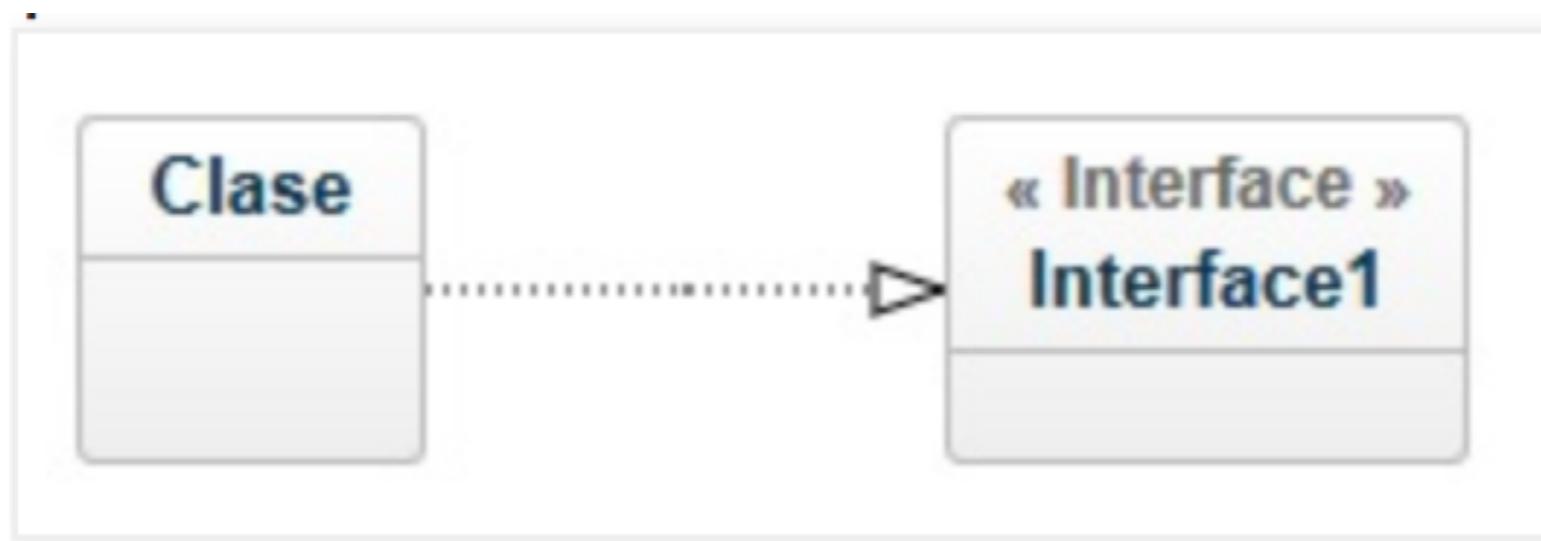
Las interfaces juegan un papel fundamental en la creación de aplicaciones Java:

- Las interfaces definen un protocolo de comportamiento y proporcionan un formato común para implementarlo en las clases.
- Utilizando interfaces es posible que clases no relacionadas, situadas en distintas jerarquías de clases sin relaciones de herencia, tengan comportamientos comunes.

POO avanzada

Interfaces

En UML una clase que implementa una interface se representa mediante una flecha con línea discontinua apuntando a la interface:



Interfaces

Ejemplo

```
1
2
3 public interface Relacionable {
4     boolean esMayorQue(Relacionable param);
5     boolean esMenorQue(Relacionable param);
6     boolean esIgualQue(Relacionable param);
7 }
```

Observa que param es de tipo Relacionable

Interfaces

```
1 package ejInterfaces;
2
3 public class Fraccion implements Relacionable {
4
5     private int num;
6     private int den;
7
8     public Fraccion() {
9         this.num = 0;
10        this.den = 1;
11    }
12
13     public Fraccion(int num, int den) {
14         this.num = num;
15         this.den = den;
16         simplificar();
17     }
18
19     public Fraccion(int num) {
20         this.num = num;
21         this.den = 1;
22     }
23
24     public void setDen(int den) {
25         this.den = den;
26         this.simplificar();
27     }
28
29     public void setNum(int num) {
30         this.num = num;
31         this.simplificar();
32     }
33
34     public int getDen() {
35         return den;
36     }
37
38     public int getNum() {
39         return num;
40     }
41
42     //sumar fracciones
43     public Fraccion sumar(Fraccion f) {
44         Fraccion aux = new Fraccion();
45         aux.num = num * f.den + den * f.num;
46         aux.den = den * f.den;
47         aux.simplificar();
48         return aux;
49     }
50
51     //restar fracciones
52     public Fraccion restar(Fraccion f) {
53         Fraccion aux = new Fraccion();
54         aux.num = num * f.den - den * f.num;
55         aux.den = den * f.den;
56         aux.simplificar();
57         return aux;
58     }
59
60     //multiplicar fracciones
61     public Fraccion multiplicar(Fraccion f) {
62         Fraccion aux = new Fraccion();
63         aux.num = num * f.num;
64         aux.den = den * f.den;
65         aux.simplificar();
66         return aux;
67     }
68
69     //dividir fracciones
70     public Fraccion dividir(Fraccion f) {
71         Fraccion aux = new Fraccion();
72         aux.num = num * f.den;
73         aux.den = den * f.num;
74         aux.simplificar();
75         return aux;
76     }
```

Interfaces

```
78     //Cálculo del máximo común divisor por el algoritmo de Euclides
79     private int mcd() {
80         int u = Math.abs(num); //valor absoluto del numerador
81         int v = Math.abs(den); //valor absoluto del denominador
82         if (v == 0) {
83             return u;
84         }
85         int r;
86         while (v != 0) {
87             r = u % v;
88             u = v;
89             v = r;
90         }
91         return u;
92     }
93
94     private void simplificar() {
95         int n = mcd(); //se calcula el mcd de la fracción
96         num = num / n;
97         den = den / n;
98     }
99
100    public String toString() { //Sobreescritura del método toString heredado de Object
101        simplificar();
102        return num + "/" + den;
103    }
```

Interfaces

```
105 //Implementación del método abstracto de la interface
106 public boolean esMayorQue(Relacionable param)
107 {
108     boolean resultado = true;
109     Fraccion frac;
110
111     if (param == null)
112     {
113         resultado = false;
114     }
115     else
116     {
117         if (!(param instanceof Fraccion))
118         {
119             resultado = false;
120         }
121         else
122         {
123             frac = (Fraccion) param;
124             this.simplificar();
125             frac.simplificar();
126             if ((num / (double) den) <= (frac.num / (double) frac.den))
127             {
128                 resultado = false;
129             }
130         }
131     }
132
133     return resultado;
134 }
```

Interfaces

```
135
136 //Implementación del método abstracto de la interface
137 public boolean esMenorQue(Relacionable param)
138 {
139     boolean resultado = true;
140     Fraccion frac;
141
142     if (param == null)
143     {
144         resultado = false;
145     }
146     else
147     {
148         if (!(param instanceof Fraccion))
149         {
150             resultado = false;
151         }
152         else
153         {
154             frac = (Fraccion) param;
155             this.simplificar();
156             frac.simplificar();
157             if ((num / (double) den) >= (frac.num / (double) frac.den))
158             {
159                 resultado = false;
160             }
161         }
162     }
163     return resultado;
164 }
165
```

POO avanzada

Interfaces

```
166  
167 //Implementación del método abstracto de la interface  
168 public boolean esIgualQue(Relacionable param)  
169 {  
170     boolean resultado = true;  
171     Fraccion frac;  
172  
173     if (param == null)  
174     {  
175         resultado = false;  
176     }  
177     else  
178     {  
179         if (!(param instanceof Fraccion))  
180         {  
181             resultado = false;  
182         }  
183         else  
184         {  
185             frac = (Fraccion) param;  
186             this.simplificar();  
187             frac.simplificar();  
188             if (num != frac.num)  
189             {  
190                 resultado = false;  
191             }  
192             if (den != frac.den)  
193             {  
194                 resultado = false;  
195             }  
196         }  
197     }  
198  
199     return resultado;  
200 }  
201 }
```

Interfaces

```
1 package ejInterfaces;
2
3 public class PruebaFracciones
4 {
5     public static void main(String[] args)
6     {
7         //Creamos dos fracciones y mostramos cuál es la mayor y cuál menor.
8         Fraccion frac1 = new Fraccion(3, 5);
9         Fraccion frac2 = new Fraccion(2, 8);
10
11        if (frac1.esMayorQue(frac2)) {
12            System.out.println(frac1 + " > " + frac2);
13        } else {
14            System.out.println(frac1 + " <= " + frac2);
15        }
16    }
17 }
18 }
```

POO avanzada

Interfaces

Ejercicio: Realiza la clase Línea que implemente el interfaz relacionable

Interfaces

```
1 package ejInterfaces;
2
3 public class Linea implements Relacionable
4 {
5     private double x1;
6     private double y1;
7     private double x2;
8     private double y2;
9
10    public Linea(double x1, double y1, double x2, double y2) {
11        this.x1 = x1;
12        this.y1 = y1;
13        this.x2 = x2;
14        this.y2 = y2;
15    }
16
17    public double longitud() {
18        double l = Math.sqrt((x2 - x1) * (x2 - x1) + (y2 - y1) * (y2 - y1));
19        return l;
20    }
}
```

Interfaces

```
20
21      //Implementación del método abstracto de la interface
22
23  public boolean esMayorQue(Relacionable param)
24  {
25      boolean resultado = true;
26      Linea linea;
27
28      if (param == null)
29      {
30          resultado = false;
31      }
32      else
33      {
34          if (!(param instanceof Linea))
35          {
36              resultado = false;
37          }
38          else
39          {
40              linea = (Linea) param;
41              resultado = this.longitud() > linea.longitud();
42          }
43      }
44
45      return resultado;
46 }
```



Podemos declarar variables y parámetros de tipo Relacionable

Interfaces

```
47 //Implementación del método abstracto de la interface
48
49 public boolean esMenorQue(Relacionable param) {
50
51     boolean resultado = true;
52     Linea linea;
53
54     if (param == null)
55     {
56         resultado = false;
57     }
58     else
59     {
60         if (!(param instanceof Linea))
61         {
62             resultado = false;
63         }
64         else
65         {
66             linea = (Linea) param;
67             resultado = (this.longitud() < linea.longitud());
68         }
69     }
70
71     return resultado;
72 }
```

POO avanzada

Interfaces

```
73 //Implementación del método abstracto de la interface
74
75 public boolean esIgualQue(Relacionable param)
76 {
77     boolean resultado = true;
78     Linea linea;
79
80     if (param == null)
81     {
82         resultado = false;
83     }
84     else
85     {
86         if (!(param instanceof Linea)) {
87             resultado = false;
88         }
89         else
90         {
91             linea = (Linea) param;
92             resultado = (this.longitud() == linea.longitud());
93         }
94     }
95
96     return resultado;
97
98 }
99
100 }
101 }
```

Interfaces

```
1 package ejInterfaces;
2
3 public class PruebaLineas
4 {
5     public static void main(String[] args)
6     {
7         Linea lin1 = new Linea(2, 2, 4, 1);
8         Linea lin2 = new Linea(5, 2, 10, 8);
9         if (lin1.esMayorQue(lin2)) {
10             System.out.println("La línea primera es mayor");
11         }
12         else
13         {
14             System.out.println("La línea primera es menor o igual que la 2");
15         }
16     }
17 }
18 }
```

POO avanzada

Interfaces

A veces el definir una interfaz o no depende de si otras clases de nuestro desarrollo necesitan tener ese comportamiento

POO avanzada

Interfaces

Ejercicio: Necesitamos gestionar un videoclub online en el que prestamos series y videojuegos.

Realizar el diagrama de clases con UML

Las series y videojuegos deben implementar un Interfaz llamado Entregable con los siguientes métodos:

- entregar(): cambia el atributo prestado a true.
- devolver(): cambia el atributo prestado a false.
- isEntregado(): devuelve el estado del atributo prestado.

Además debemos implementar una clase abstracta que defina el método abstracto compararCon (Tipo miObjeto) Decidir el tipo más adecuado (debe devolver mayor, menor o igual)

Las series y videojuegos deben sobreescribir el método `toString` y el método `equals`

El programa principal debe crear una lista de videojuegos y series. Debe entregar y devolver algunos de ellos, mostrar el número de entregados y mostrar el videoJuego mayor y la serie mayor.

POO avanzada

Interfaces

```
1 package interfaces2;
2
3 public interface Entregable {
4
5     public void entregar();
6
7     public void devolver();
8
9     public boolean isEntregado();
10
11 }
12
13
```

POO avanzada

```
package interfaces2;

public abstract class ProductoMultimedia implements Entregable{

    /**
     * Constante que indica que un objeto es mayor que otro
     */
    public final static int MAYOR=1;

    /**
     * Constante que indica que un objeto es menor que otro
     */
    public final static int MENOR=-1;

    /**
     * Constante que indica que un objeto es igual que otro
     */
    public final static int IGUAL=0;

    protected String titulo;
    protected String genero;
    protected String compañia;
    protected boolean entregado;

    public String getTitulo() {
        return titulo;
    }

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

    public String getGenero() {
        return genero;
    }

    public void setGenero(String genero) {
        this.genero = genero;
    }

    public String getCompañia() {
        return compañia;
    }

    public void setCompañia(String compañia) {
        this.compañia = compañia;
    }

    public abstract int compararCon(ProductoMultimedia producto);

    /**
     * Cambia el estado de entregado a true
     */
    public void entregar() {
        entregado=true;
    }

    /**
     * Cambia el estado de entregado a false
     */
    public void devolver() {
        entregado=false;
    }

    /**
     * Indica el estado de entregado
     * @return
     */
    public boolean isEntregado()
    {
        return entregado;
    }
}
```

POO avanzada

```
package interfaces2;

public class Videojuego extends ProductoMultimedia implements Entregable{

    //Constantes
    /**
     * Horas estimadas por defecto
     */
    private final static int HORAS_ESTIMADAS_DEF=0;

    //Atributos
    /**
     * Horas estimadas del videojuego
     */
    private int horasEstimadas;

    //Constructor
    /**
     * Constructo por defecto
     */
    public Videojuego(){
        this("",HORAS_ESTIMADAS_DEF, "", "");
    }

    /**
     * Constructor con 2 parametros
     * @param titulo del videojuego
     * @param compañia del videojuego
     */
    public Videojuego(String titulo, String compañia){
        this(titulo,HORAS_ESTIMADAS_DEF, "", compañia);
    }

    //Métodos públicos
    /**
     * Devuelve el numero de paginas del videojuego
     * @return numero de paginas del videojuego
     */
    public int getHorasEstimadas() {
        return horasEstimadas;
    }

    /**
     * Modifica el numero de paginas del videojuego
     * @param horasEstimadas
     */
    public void setHorasEstimadas(int horasEstimadas) {
        this.horasEstimadas = horasEstimadas;
    }
}
```

```


/*
 * Compara dos videojuegos segun el numero de paginas
 * @return codigo numerico
 * <ul>
 * <li>1: El videojuego 1 es mayor que el videojuego 2</li>
 * <li>0: Los videojuegos son iguales</li>
 * <li>-1: El videojuego 1 es menor que el videojuego 2</li></ul>
 */

public int compararCon(ProductoMultimedia producto) {
    int estado=MENOR;

    if (producto != null)
    {
        //Hacemos un casting de objetos para usar el metodo get
        Videojuego ref=(Videojuego)producto;
        if (horasEstimadas>ref.getHorasEstimadas()){
            estado=MAYOR;
        }else if(horasEstimadas==ref.getHorasEstimadas()){
            estado=IGUAL;
        }
    }

    return estado;
}

/*
 * Muestra informacion del videojuego
 * @return cadena con toda la informacion del videojuego
 */

public String toString(){
    return "Informacion del videojuego: \n" +
        "tTitulo: "+titulo+"\n" +
        "tHoras estimadas: "+horasEstimadas+"\n" +
        "tGenero: "+genero+"\n" +
        "tcompania: "+compania;
}


```

```


/*
 * Indica si dos videojuegos son iguales, siendo el titulo y compania iguales
 * @param videoJuego videojuego a comparar
 * @return true si son iguales y false si son distintos
 */
public boolean equals(Object objeto)
{
    boolean igual = false;

    if (objeto != null)
    {
        if (objeto instanceof Videojuego)
        {
            if (titulo.equalsIgnoreCase(((Videojuego)objeto).getTitulo()) &&
                compania.equalsIgnoreCase(((Videojuego)objeto).getCompania()))
            {
                igual = true;
            }
        }
    }
    return igual;
}


```

```
package interfaces2;

public class Serie extends ProductoMultimedia implements Entregable
{
    //Constantes
    /**
     * Numero de temporadas por defecto
     */
    private final static int NUM_TEMPORADAS_DEF=0;

    //Atributos
    /**
     * Numero de temporadas de la serie
     */
    private int numeroTemporadas;

    /**
     * Creador de la serie
     */
    private String creador;

    //Constructor
    /**
     * Constructor por defecto
     */
    public Serie(){
        this("",NUM_TEMPORADAS_DEF, "", "");
    }

    /**
     * Contructor con 2 parametros
     * @param titulo de la Serie
     * @param creador de la Serie
     */
    public Serie(String titulo, String creador){
        this(titulo,NUM_TEMPORADAS_DEF, "", creador);
    }

    //Métodos públicos
    /**
     * Constructor con 4 parametros
     * @param titulo de la Serie
     * @param numeroTemporadas de la Serie
     * @param genero de la Serie
     * @param creador de la Serie
     */
    public Serie(String titulo, int numeroTemporadas, String genero, String creador){
        this.titulo=titulo;
        this.numeroTemporadas=numeroTemporadas;
        this.genero=genero;
        this.creador=creador;
        this.entregado=false;
    }

    /**
     * Devuelve la numeroTemporadas de la serie
     * @return numeroTemporadas de la serie
     */
    public int getnumeroTemporadas() {
        return numeroTemporadas;
    }

    /**
     * Modifica la numeroTemporadas de la serie
     * @param numeroTemporadas a cambiar
     */
    public void setnumeroTemporadas(int numeroTemporadas) {
        this.numeroTemporadas = numeroTemporadas;
    }

    /**
     * Devuelve el genero de la serie
     * @return genero de la serie
     */
    public String getGenero() {
        return genero;
    }
}
```

```

/**
 * Modifica el genero de la serie
 * @param genero a cambiar
 */
public void setGenero(String genero) {
    this.genero = genero;
}

/**
 * Devuelve el creador de la serie
 * @return creador de la serie
 */
public String getcreador() {
    return creador;
}

/**
 * Modifica el creador de la serie
 * @param creador a cambiar
 */
public void setcreador(String creador) {
    this.creador = creador;
}

/**
 * Compara dos series segun su numero de temporadas.
 * @param objeto a comparar
 * @return codigo numerico
 * <ul>
 * <li>1: La Serie 1 es mayor que la Serie 2</li>
 * <li>0: Las Series son iguales</li>
 * <li>-1: La Serie 1 es menor que la Serie 2</li></ul>
 */
public int compararCon(ProductoMultimedia producto) {
    int estado=MENOR;

    if (producto != null)
    {
        //Hacemos un casting de objetos para usar el metodo get
        Serie ref=(Serie)producto;
        if (numeroTemporadas>ref.getnumeroTemporadas())
            estado=MAYOR;
        else if(numeroTemporadas==ref.getnumeroTemporadas())
            estado=IGUAL;
    }
}

return estado;
}

/**
 * Muestra informacion de la Serie
 * @return cadena con toda la informacion de la Serie
 */
public String toString(){
    return "Informacion de la Serie: \n" +
        "tTitulo: "+titulo+"\n" +
        "tNumero de temporadas: "+numeroTemporadas+"\n" +
        "tGenero: "+ genero+"\n" +
        "tCreador: "+ creador;
}

/**
 * Indica si dos Series son iguales, siendo el titulo y creador iguales
 * @param a Serie a comparar
 * @return true si son iguales y false si son distintos
 */
public boolean equals(Object objeto)
{
    boolean igual = false;

    if (objeto != null)
    {
        if ( titulo.equalsIgnoreCase( ((Serie)objeto).getTitulo() ) &&
            creador.equalsIgnoreCase( ((Serie)objeto).getcreador() ) )
        {
            igual = true;
        }
    }

    return igual;
}

```

```

package interfaces2;

public class EntregablesApp
{
    static ProductoMultimedia[] listaProductos;

    public static void main(String[] args)
    {
        listaProductos = new ProductoMultimedia[10];
        CargarProductos();
        MostrarMayor();
    }

    private static void CargarProductos()
    {
        int entregados=0;

        listaProductos[0]=new Serie();
        listaProductos[1]=new Serie("Juego de tronos", "George R. R. Martin ");
        listaProductos[2]=new Serie("Los Simpson", 25, "Humor", "Matt Groening");
        listaProductos[3]=new Serie("Padre de familia", 12 , "Humor", "Seth MacFarlane");
        listaProductos[4]=new Serie("Breaking Bad", 5, "Thriller", "Vince Gilligan");

        listaProductos[5]=new Videojuego();
        listaProductos[6]=new Videojuego("Assasin creed 2", 30, "Aventura", "EA");
        listaProductos[7]=new Videojuego("God of war 3", "Santa Monica");
        listaProductos[8]=new Videojuego("Super Mario 3DS", 30, "Plataforma", "Nintendo");
        listaProductos[9]=new Videojuego("Final fantasy X", 200, "Rol", "Square Enix");

        listaProductos[1].entregar();
        listaProductos[4].entregar();
        listaProductos[5].entregar();
        listaProductos[8].entregar();

        //Recorremos los arrays para contar cuantos entregados hay, tambien los devolvemos
        for(int i=0;i<listaProductos.length;i++)
        {
            if (listaProductos[i] != null)
            {
                if (listaProductos[i].isEntregado())
                {
                    entregados+=1;
                    listaProductos[i].devolver();
                }
            }
        }
        System.out.println("Hay "+entregados+" articulos entregados\n");
    }
}

```

```
private static void MostrarMayor()
{
    Serie serieMayor=new Serie();
    Videojuego videojuegoMayor=new Videojuego();

    for(int i=1;i<listaProductos.length;i++)
    {
        if (listaProductos[i] != null)
        {
            if (listaProductos[i] instanceof Serie)
            {
                if(listaProductos[i].compararCon(serieMayor) == ProductoMultimedia.MAYOR)
                {
                    serieMayor=(Serie) listaProductos[i];
                }
            }
            if (listaProductos[i] instanceof Videojuego)
            {
                if(listaProductos[i].compararCon(videojuegoMayor) == ProductoMultimedia.MAYOR)
                {
                    videojuegoMayor=(Videojuego) listaProductos[i];
                }
            }
        }
    }

    //Mostramos toda la informacion del videojuego y serie mayor
    System.out.println("----- VIDEOJUEGO MAYOR -----");
    System.out.println(videojuegoMayor);

    System.out.println("\n----- SERIE MAYOR -----");
    System.out.println(serieMayor);
}
```

POO avanzada

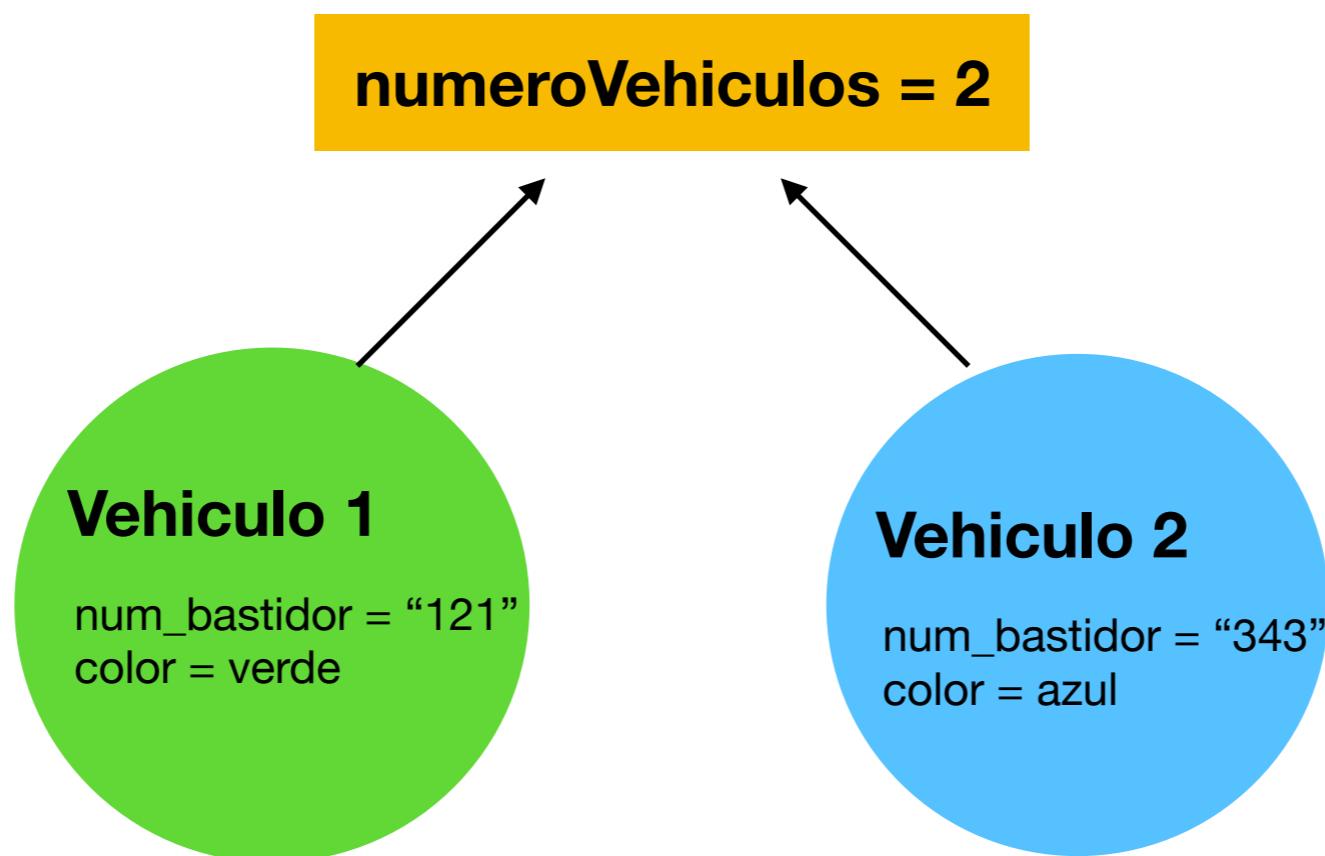
Static

Cuando definimos un elemento como **static** quiere decir que ese elemento no puede replicarse en distintas instancias (objetos) sino que todas las instancias lo comparten.

POO avanzada

Static

Para el caso de una variable de una clase, si está definida como **static**, todas las instancias(objetos) de esa clase comparten la misma variable. Observa el ejemplo



POO avanzada

Static

Para el caso de una variable de una clase, si está definida como **static**, todas las instancias(objetos) de esa clase comparten la misma variable. Observa el ejemplo. La forma adecuada de acceder a un elemento estático es `nombreClase.elementoStatic`

```
public class Vehiculo {  
  
    private String numBastidor;  
    private String color;  
    private int anioFab;  
    private double velocidadMax;  
  
    public static int numeroVehiculos;  
  
    public Vehiculo(String numBastidor, String color,  
                    int anioFab, double velocidadMax)  
    {  
        this.numBastidor = numBastidor;  
        this.color = color;  
        this.anioFab = anioFab;  
        this.velocidadMax = velocidadMax;  
  
        numeroVehiculos++;  
    }  
}  
  
3 public class PruebaStatic {  
4  
5     public static void main(String[] args) {  
6         Vehiculo vehiculo1 = new Vehiculo("EWTWGWT-111", "VERDE", 2001, 190);  
7         Vehiculo vehiculo2 = new Vehiculo("DFZGSGF-G777", "AZUL", 2021, 200);  
8         Vehiculo vehiculo3 = new Vehiculo("JGKGK-905", "BLANCO", 2012, 210);  
9         Vehiculo vehiculo4 = new Vehiculo("DFGDF-4545", "ROJO", 2000, 120);  
10  
11         System.out.println(Vehiculo.numeroVehiculos);  
12         System.out.println(vehiculo1.numeroVehiculos);  
13         System.out.println(vehiculo2.numeroVehiculos);  
14         System.out.println(vehiculo3.numeroVehiculos);  
15         System.out.println(vehiculo4.numeroVehiculos);  
16     }  
17 }  
18 }
```

Da warnings indicando que estamos accediendo a una variable estática de forma no estática



```
Console X  
<terminated> PruebaStatic [Java Application] /Library/Java/JavaVirtualMachines/jdk-11.0.1.jdk/Contents/Home/bin/java (Feb 13, 2022, 4:22:03 PM - 4:  
4  
4  
4  
4  
4
```

POO avanzada

Static

Ejercicio: Probadlo con la clase libro y biblioteca

POO avanzada

Static

Para el caso de un método de una clase, si está definido como **static**, pertenece a una clase en lugar de a un objeto. Lo llamamos mediante `nombreClase.nombreMétodo`. Observa cómo la clase Math tiene sus métodos declarados como estáticos, por eso hemos estado usando sentencias como `Math.random()` sin crear objetos Math

<https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html>

Más sobre static: <https://www.baeldung.com/java-static>