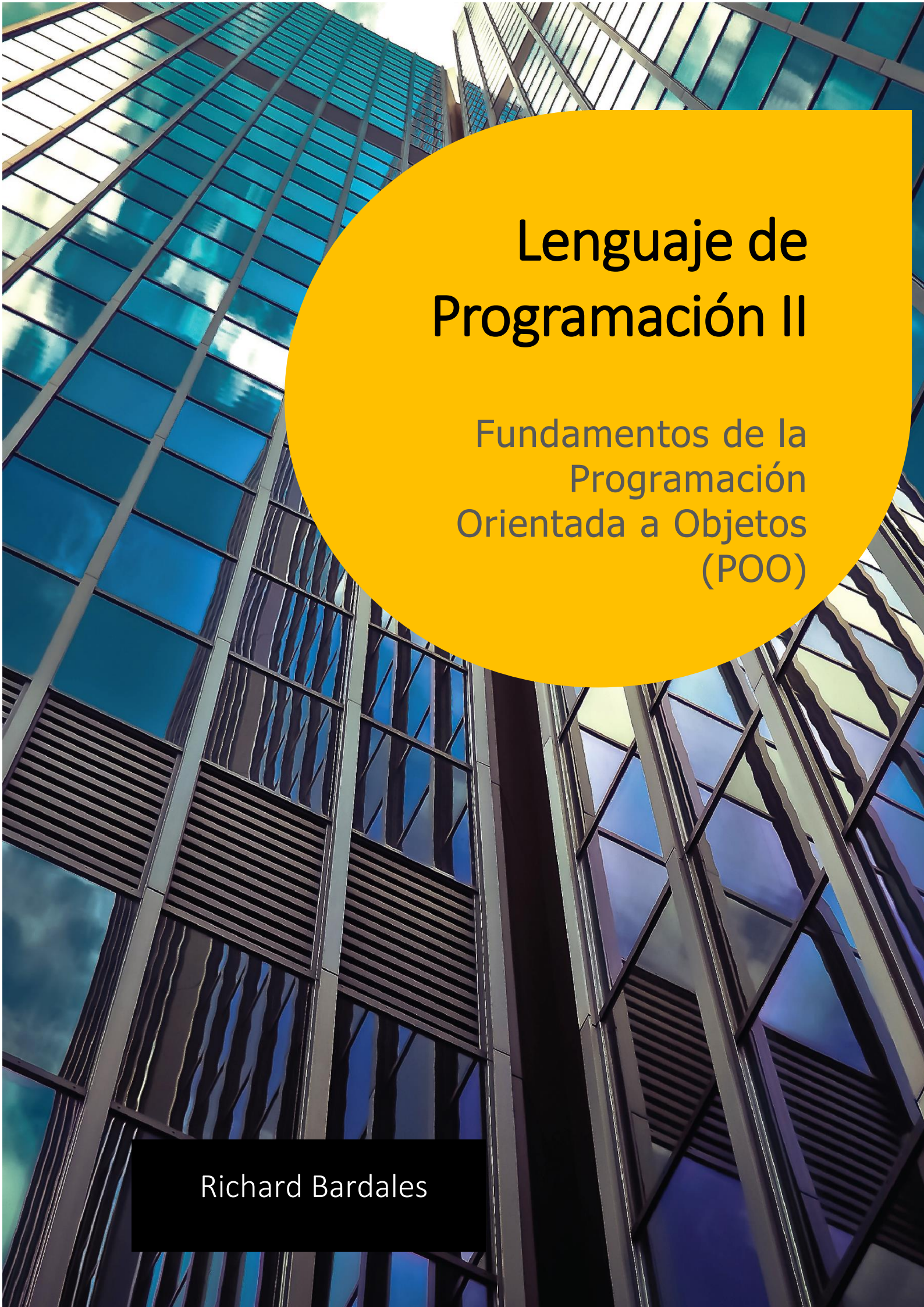


# Lenguaje de Programación II

## Fundamentos de la Programación Orientada a Objetos (POO)

Richard Bardales

# Capítulo 1

## Fundamentos de la Programación Orientada a Objetos (POO)

**01** **Conocer los conceptos generales de la POO.**

**02** **Desarrollo de aplicaciones basadas en POO.**

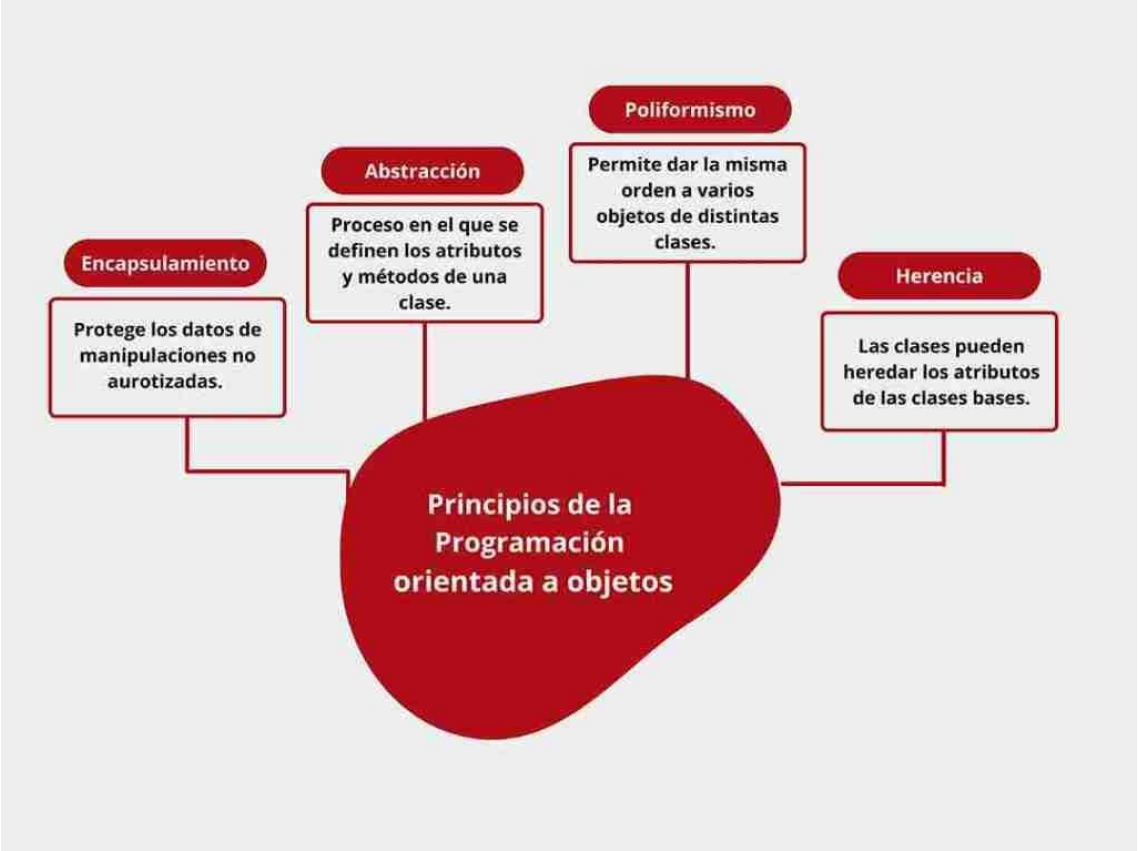
# Capítulo 1

## Conceptos generales de la POO

En el mundo de la programación, la POO es un paradigma (es decir, un modelo o un estilo de programación que nos da unas guías sobre cómo trabajar con él) que ha ganado una gran popularidad en los últimos años debido a su capacidad para crear aplicaciones más robustas, flexibles y fáciles de mantener. Esta metodología de desarrollo se basa en la idea de que los programas se pueden organizar como una colección de objetos interconectados, cada uno con su propio conjunto de datos y funcionalidades.

Esto nos ayuda muchísimo en sistemas grandes, ya que, en vez de pensar en funciones, pensamos en las relaciones o interacciones de los diferentes componentes del sistema.

## Los cuatro pilares de la Programación Orientada a Objetos



# Las Clases

En el mundo real, normalmente tenemos muchos objetos del mismo tipo. Por ejemplo, nuestro teléfono móvil es sólo uno de los miles que hay en el mundo. Si hablamos en términos de la programación orientada a objetos, podemos decir que nuestro objeto móvil es una instancia de una clase conocida como «móvil». Los móviles tienen características (marca, modelo, sistema operativo, pantalla, teclado, etc.) y comportamientos (hacer y recibir llamadas, enviar mensajes multimedia, transmisión de datos, etc.).

Cuando se fabrican los móviles, los fabricantes aprovechan el hecho de que los móviles comparten esas características comunes y construyen modelos o plantillas comunes, para que a partir de esas se puedan crear muchos móviles del mismo modelo. A ese modelo o plantilla le llamamos clase, y a los equipos que sacamos a partir de ella la llamamos objetos.

Esto mismo se aplica a los objetos de software, se puede tener muchos objetos del mismo tipo y mismas características.

Definición teórica: La clase es un modelo o prototipo que define las variables y métodos comunes a todos los objetos de cierta clase. También se puede decir que una clase es una plantilla genérica para un conjunto de objetos de similares características.

Por otro lado, una instancia de una clase es otra forma de llamar a un objeto. En realidad, no existe diferencia entre un objeto y una instancia. Sólo que el objeto es un término más general, pero los objetos y las instancias son ambas, representación de una clase.

# Atributos

En la programación orientada a objetos, los atributos son una propiedad o característica que se puede asignar a un objeto (elemento). Mediante el uso de atributos se pueden asignar valores específicos a ciertos elementos.

# Métodos

Un método es una función que se define dentro de una clase y se utiliza para representar el comportamiento de un objeto.

## Métodos Genéricos

Comencemos con el formato básico de un método:

```
[public | private] {void|tipo} identificadorMetodo (tipo param1, tipo param2…) {…}
```

## Métodos para accesos a atributos (Getter y Setter)

Cada atributo tiene dos posibles acciones: leer su valor o establecerlo. No tenemos por qué realizar ambas acciones, depende del diseño que se busque, de hecho, muchas clases no tiene estos métodos.

## Constructores

Los constructores son métodos especiales que reúnen las tareas de inicialización de los objetos de una clase; por lo tanto, el constructor establece el estado inicial de todos los objetos que se instancian.

No es obligatorio definir constructores, si no se realiza, existe un constructor por defecto sin parámetros.

La ejecución del constructor es implícita a la creación de una instancia.

La restricción sintáctica del constructor es que debe llamarse igual que la clase y no devuelve ningún tipo ni lleva la palabra void.

# Objetos

Entender que es un objeto es la clave para entender cualquier lenguaje orientado a objetos.

Tomemos como ejemplo un ordenador. No necesitamos ser expertos para saber que un ordenador está compuesta internamente por varios componentes: placa base, procesador, disco duro, tarjeta de video, etc…

El trabajo en conjunto de todos estos componentes hace funcionar un ordenador, sin embargo, no necesitamos saber cómo trabajan cada uno de estos componentes. Cada componente es una unidad autónoma, y todo lo que necesitamos saber es cómo interactúan entre sí los componentes, saber por ejemplo si el procesador y las memorias son compatibles con la placa base, o conocer donde se coloca la tarjeta de video. Cuando conocemos como interaccionan los componentes entre sí, podremos armar fácilmente un ordenador.

¿Qué tiene que ver esto con la programación? La programacion orientada a objetos trabaja de esta manera. Todo el programa está construido en base a diferentes componentes (objetos), cada uno tiene un rol específico en el programa y todos los componentes pueden comunicarse entre ellos de formas predefinidas.

Todo objeto del mundo real tiene 2 componentes: variables de clase y métodos.

Por ejemplo, los automóviles pueden tener como variables de clase (marca, modelo, color, velocidad máxima, etc.) y como métodos (frenar, acelerar, retroceder, llenar combustible, cambiar llantas, etc.).

Los objetos de Software, al igual que los objetos del mundo real, también tienen variables de clase y métodos. Un objeto de software mantiene sus características en una o más «variables», e implementa su comportamiento con «métodos». Un método es una función o subrutina asociada a un objeto.

Imaginemos que tenemos aparcado en el garaje un Ford Focus color azul que corre hasta 260 km/h. Si pasamos ese objeto del mundo real al mundo del software, tendremos un objeto Automóvil con sus características predeterminadas:

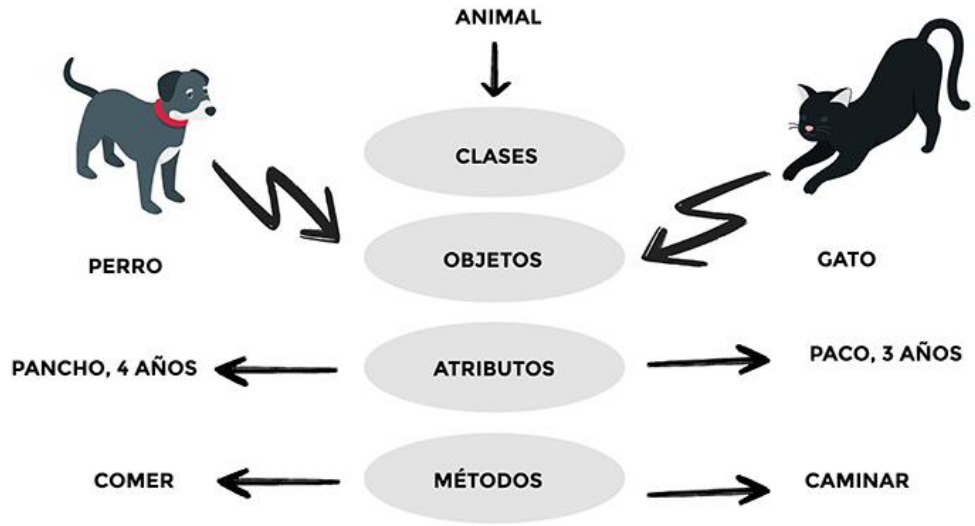
Marca = Ford

Modelo = Focus

Color = Azul

Velocidad Máxima = 260 km/h

Por lo tanto, dentro de una clase tendremos como variables de clase las características descritas anteriormente y como métodos tendremos en este caso concreto (frenar, acelerar, retroceder, llenar combustible, etc…).




**Vamos a ver un ejemplo práctico de como se hace esto en Java:**

```
class Automovil {

   // VARIABLES DE CLASE
   private String marca;
   private String modelo;
```

```
   private String color;
   private String velocidadMaxima;

   // CONSTRUCTOR QUE INICIALIZA LAS VARIABLES DE CLASE
   public Automovil(String marca, String modelo, String color, String
velocidadMaxima) {
      this.marca = marca;
      this.modelo = modelo;
      this.color = color;
      this.velocidadMaxima = velocidadMaxima;
   }
   // METODOS GETTER Y SETTER PARA PODER RECUPERAR O CAMBIAR
   // LOS DATOS DE LAS VARIABLES DE CLASE

   public String getMarca() {
      return marca;
   }
   public void setMarca(String marca) {
      this.marca = marca;
   }
   public String getModelo() {
      return modelo;
   }

   public void setModelo(String modelo) {
      this.modelo = modelo;
   }
   public String getColor() {
      return color;
   }
   public void setColor(String color) {
      this.color = color;
   }
   public String getVelocidadMaxima() {
      return velocidadMaxima;
   }
   public void setVelocidadMaxima(String velocidadMaxima) {
      this.velocidadMaxima = velocidadMaxima;
   }
}
```

# Visibilidad

Hasta ahora hemos trabajado con dos tipos de visibilidad: public y private. Recordemos que un elemento público es visible para todos y privado sólo para uso interno.

En Java se distinguen dos tipos más de visibilidad: protected y sin calificador (friendly).

Cuando calificamos a un elemento como protected, significa que es visible dentro del mismo paquete o cualquier subclase de forma directa (mediante super.elemento). Si pretendemos que una subclase acceda de forma directa a los atributos de la superclase, el calificador apropiado es protected.

Cuando dejamos a un elemento sin calificar (friendly), significa que sólo es visible dentro del mismo paquete.

# Herencia

La herencia es uno de los conceptos más cruciales en la POO. La herencia consiste en que una clase puede heredar las variables y los métodos de otra clase (la clase que hereda es llamada superclase o clase padre). Esto significa que una subclase, aparte de los atributos y métodos propios, tiene incorporados los atributos y métodos heredados de la superclase. De esta manera se crea una jerarquía de herencia.

Si partimos de la clase Automovil que creamos anteriormente podríamos crear clases hijas de Automovil, la cuales heredarán todo lo que contenga la clase Automovil.

**¿Como se hereda en Java?**

Para crear herencia se usa la palabra reservada extends:

```
class Coche extends Automovil {


}
```

# Polimorfismo

## Polimorfismo estático (de compilación)

Este tipo de polimorfismo, también conocido como de compilación, se utiliza para crear múltiples métodos con el mismo nombre en la misma clase, que contengan diferentes números de parámetros o bien parámetros de distintos tipos.

## Polimorfismo dinámico (de ejecución)

El polimorfismo dinámico o de ejecución es aquel en el que la clase hija tiene una definición propia, pero que depende de la clase en la que está anidada, es decir sobrescribe el método en la clase hija.

# Interfaces

Una interface en Java es un tipo abstracto que representa un conjunto de métodos que las clases pueden implementar y un conjunto de constantes estáticas, es similar a una clase abstracta pura. Igual que las clases abstractas, los interfaces no se puede instanciar. Su utilidad es crear variables de tipo interface e independizarse de las clases que contiene.

IBox.java

```java
public interface IBox {
   public void put(Object objeto);
   public Object get();
}
```

CBox.java

```java
public class CBox implements IBox {
   private Object objeto;

   public CBox(){
      this.objeto=null;
   }

   @Override
   public Object get() {
      return this.objeto;
   }

   @Override
   public void put(Object objeto) {
      this.objeto = objeto;
   }
}
```

# Clases Abstractas

Una clase abstracta es una clase que no se puede instanciar, pero si heredar. También, se pueden definir constructores. Se utilizan para englobar clases de diferentes tipos, pero con aspectos comunes. Se pueden definir métodos sin implementar y obligar a las subclases a implementarlos.

Por ejemplo, podemos tener una clase Figura, que representa una figura general, y subclases con figuras concretas (Triangulo, Circulo...). Podemos establecer métodos comunes como dibujar, que, sin conocer el tipo de figura, sea capaz de dibujarla. Pero para que esto funcione correctamente, cada figura concreta debe implementar su versión particular de dibujar.

Figura.java

```java
public abstract class Figura {
   private String tipo;
   public Figura(String tipo) {
      this.tipo = tipo;
   }
   // getters & setters
   public abstract double area();
}
```

Triangulo.java

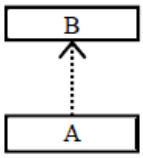
```java
public class Triangulo extends Figura {
   private int base, altura;
   public Triangulo(String tipo, int base, int altura) {
      super(tipo);
      this.setBase(base);
      this.setAltura(altura);
   }
   // getters & setters
   @Override
   public double area() {
      return (double) this.base * this.altura / 2;
   }
}
```

# Relación de Clases

## Relación de uso

Se dice que existe una relación de uso entre la clase A y la clase B, si un objeto de la clase A lanza mensajes al objeto de la clase B, para utilizar sus servicios.

Es una relación que se establece entre un cliente y un servidor, en un instante dado, a través de algún método. No es una relación duradera, sino esporádica.




```java
public class A {
   ...
   public void metodo(B b){
      b.servicio();
   }
   ...
}
```

## Relación de agregación

Se dice que la clase A tiene una relación de agregación con la clase B, si un objeto de la clase A delega funcionalidad en los objetos de la clase B.

El ciclo de vida de ambos no coincide. La clase A delega parte de su funcionalidad, pero no crea el objeto B. Varios objetos pueden estar asociados con el objeto B. Al objeto A, le deben pasar la referencia del objeto B. Es una relación duradera entre cliente (A) y servidor (B), en la vida del cliente.

```java
public class Cliente {
   private Servidor servidor;
   public Cliente(Servidor servidor) {
      this.servidor = servidor;
```

```
    }
    ...
}
```

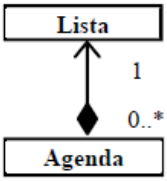
```
public class Servidor {
   ...
   public String servicio() {
      return "Servido";
   }
   ...
}
```

## Relación de composición

Se dice que existe una relación de composición entre la clase A y la clase B, si la clase A contiene un objeto de la clase B, y delega parte de sus funciones en la clase B.

Los ciclos de vida de los objetos de A y B coinciden. El objeto A es el encargado de crear la instancia de B y de almacenarla, siendo su visibilidad privada. Cuando se destruye A, también se destruye B.

Por ejemplo, se pretende realizar una clase llamada Agenda y para ello se apoya en la clase Lista, es decir, dentro de la clase Agenda se crea una instancia de la clase Lista, y algunas de las funcionalidades que aporta la clase Agenda se realizan con los métodos de la clase Lista.




```
public class Agenda {
   private Lista lista;

   public Agenda() {
      this.setLista(new Lista());
   }
   ...
}
```

## Relación de composición

Se dice que existe una relación de composición entre la clase A y la clase B, si la clase A contiene un objeto de la clase B, y delega parte de sus funciones en la clase B.

Los ciclos de vida de los objetos de A y B coinciden. El objeto A es el encargado de crear la instancia de B y de almacenarla, siendo su visibilidad privada. Cuando se destruye A, también se destruye B.

Por ejemplo, se pretende realizar una clase llamada Agenda y para ello se apoya en la clase Lista, es decir, dentro de la clase Agenda se crea una instancia de la clase Lista, y algunas de las funcionalidades que aporta la clase Agenda se realizan con los métodos de la clase Lista.

# Paquetes

Los paquetes agrupan un conjunto de clases que trabajan conjuntamente sobre el mismo ámbito. Es una facilidad ofrecida por Java para agrupar sintácticamente clases que van juntas conceptualmente y definir un alto nivel de protección para los atributos y los métodos.

La ventaja del uso de paquetes es que las clases quedan ordenadas y no hay colisión de nombres. Si dos programadores llaman igual a sus clases y luego hay que juntar el código de ambos, basta explicitar a qué paquete nos referimos en cada caso.

La forma de nombrar los paquetes es con palabras (normalmente en minúsculas) separadas por puntos, estableciendo una jerarquía de paquetes; la jerarquía de paquetes es independiente de la jerarquía de clases. Las clases deben almacenarse en una estructura de carpetas que coincidan con la estructura de paquetes.

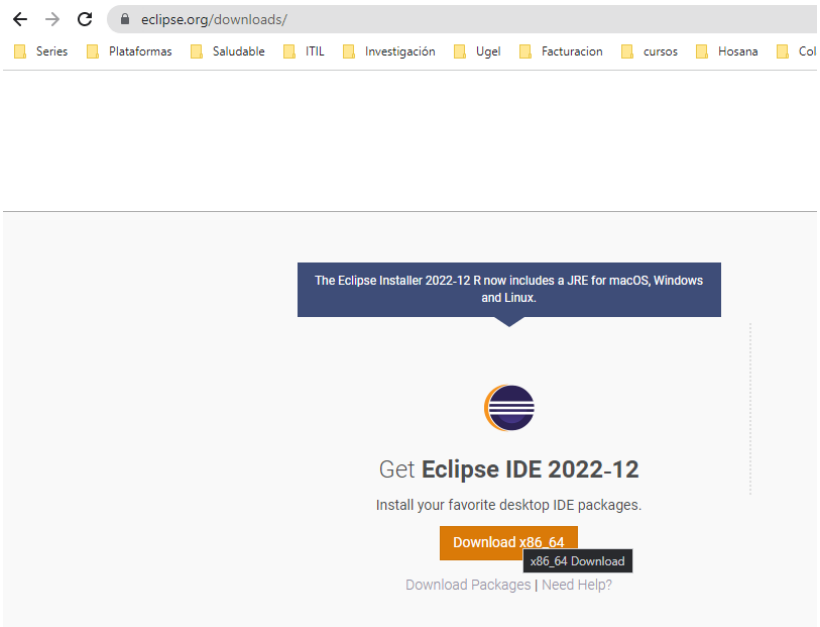
# Instalación de Java y Eclipse

**Java:** Es el lenguaje de programación, con el cual programaremos nuestras aplicaciones web.

**Eclipse:** Es el entorno de desarrollo, en el cual programaremos nuestras aplicaciones web.

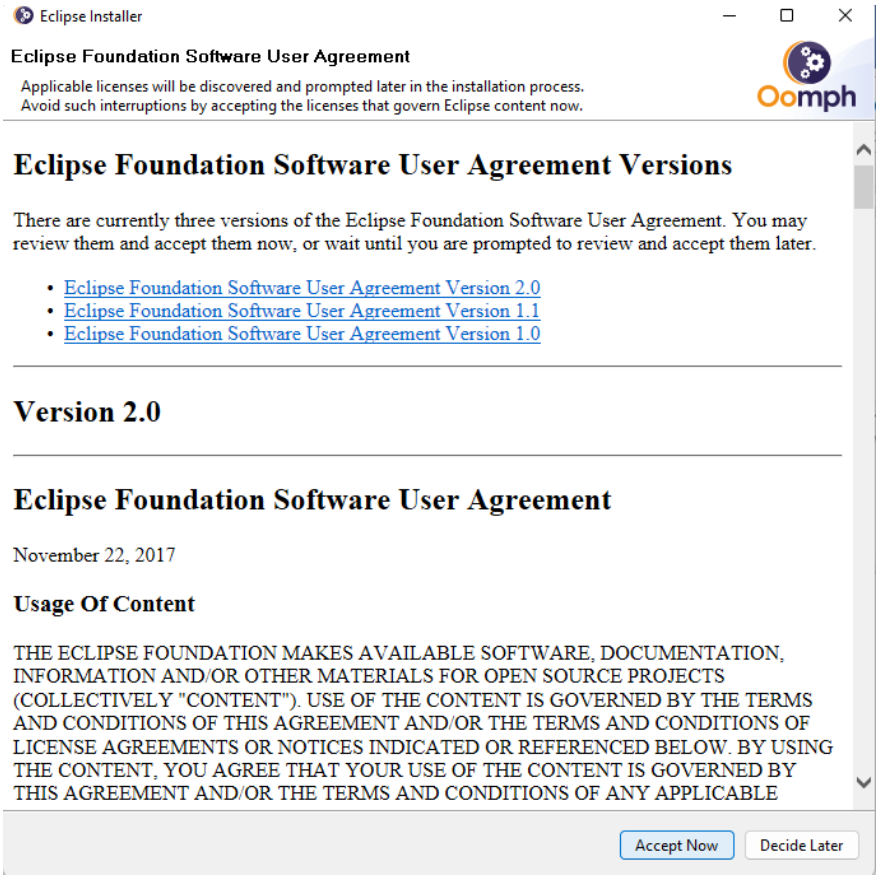
# Descargamos e instalamos Eclipse

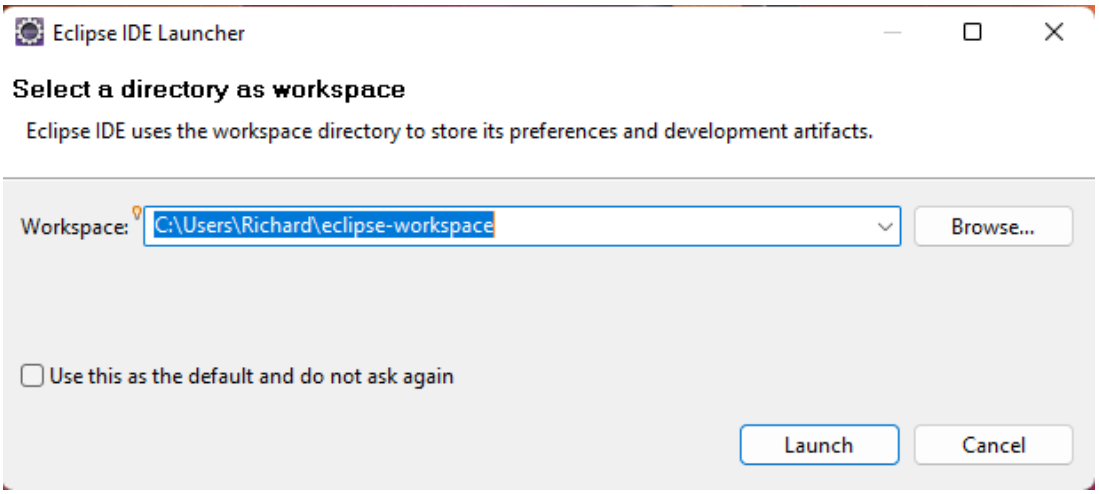
Nos dirigimos a google y buscamos eclipse, luego nos dirigimos al menú de descargas.



Una vez descargado, el archivo .exe, lo ejecutamos como administrador y elegimos la opción **Eclipse IDE for Enterprise Java and Web Developer**

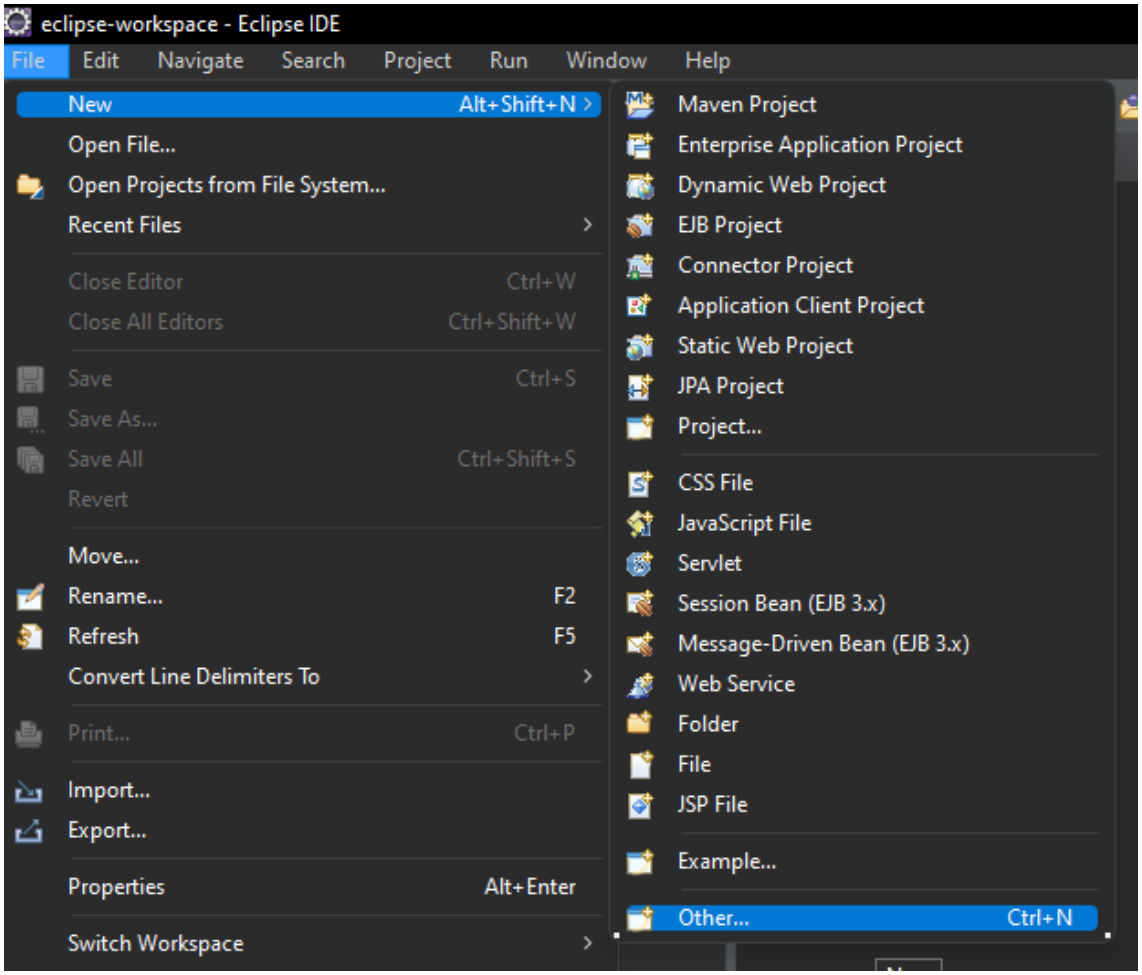

eclipseinstaller by Oomph
Eclipse IDE for Enterprise Java and Web Developers
details
Tools for developers working with Java and Web applications, including a Java IDE, tools for JavaScript, TypeScript, JavaServer Pages and Faces, Yaml, Markdown, Web Services, JPA and Data
Java 17+ VM
pse.org/justj/jres/17/updates/release/latest
Installation Folder
C:\Users\Richard\eclipse\jee-2022-12
create start menu entry
create desktop shortcut
INSTALL
Start installing the product into 'C:\Users\Ri
BACK

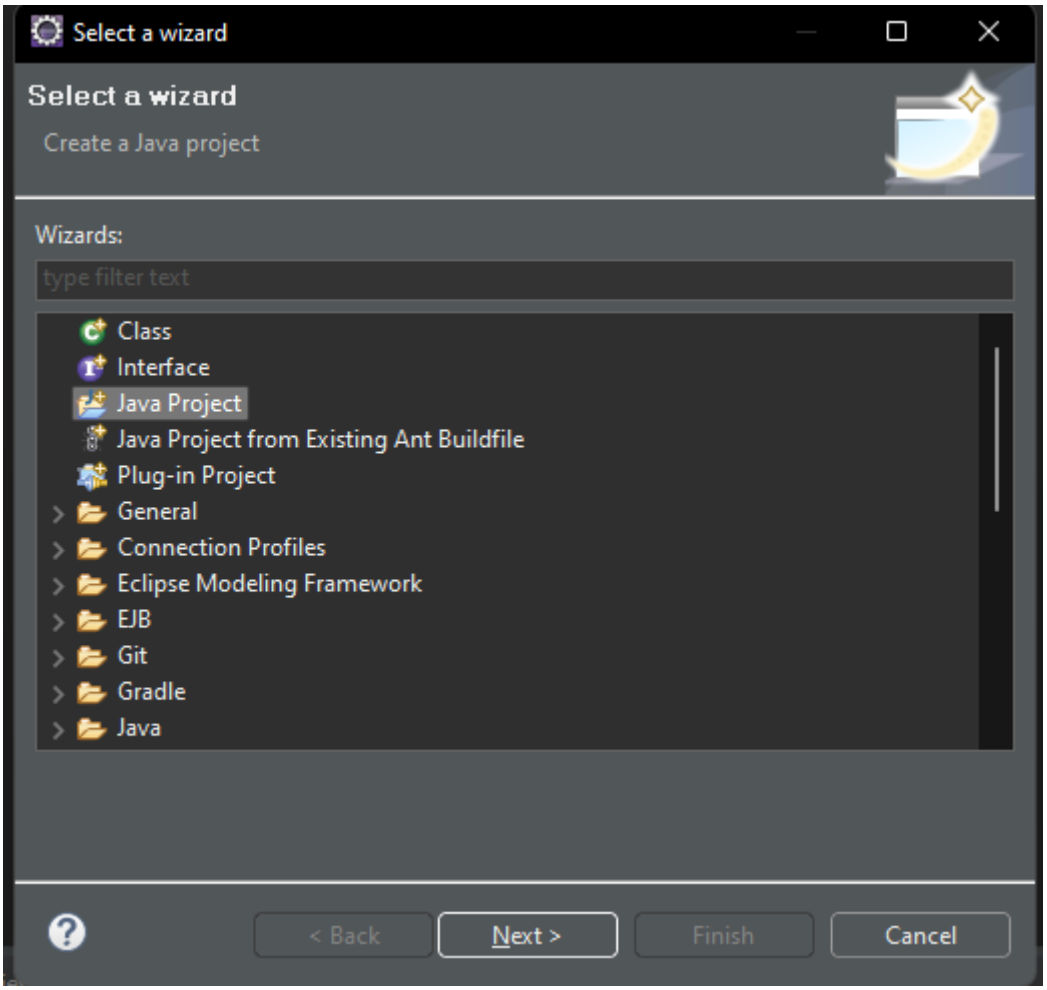

Eclipse Installer
Eclipse Foundation Software User Agreement
Applicable licenses will be discovered and prompted later in the installation process.
Avoid such interruptions by accepting the licenses that govern Eclipse content now.
Oomph
Eclipse Foundation Software User Agreement Versions
There are currently three versions of the Eclipse Foundation Software User Agreement. You may review them and accept them now, or wait until you are prompted to review and accept them later.
Eclipse Foundation Software User Agreement Version 2.0
Eclipse Foundation Software User Agreement Version 1.1
Eclipse Foundation Software User Agreement Version 1.0
Version 2.0
Eclipse Foundation Software User Agreement
November 22, 2017
Usage Of Content
THE ECLIPSE FOUNDATION MAKES AVAILABLE SOFTWARE, DOCUMENTATION, INFORMATION AND/OR OTHER MATERIALS FOR OPEN SOURCE PROJECTS (COLLECTIVELY "CONTENT"). USE OF THE CONTENT IS GOVERNED BY THE TERMS AND CONDITIONS OF THIS AGREEMENT AND/OR THE TERMS AND CONDITIONS OF LICENSE AGREEMENTS OR NOTICES INDICATED OR REFERENCED BELOW. BY USING THE CONTENT, YOU AGREE THAT YOUR USE OF THE CONTENT IS GOVERNED BY THIS AGREEMENT AND/OR THE TERMS AND CONDITIONS OF ANY APPLICABLE
Accept Now
Decide Later



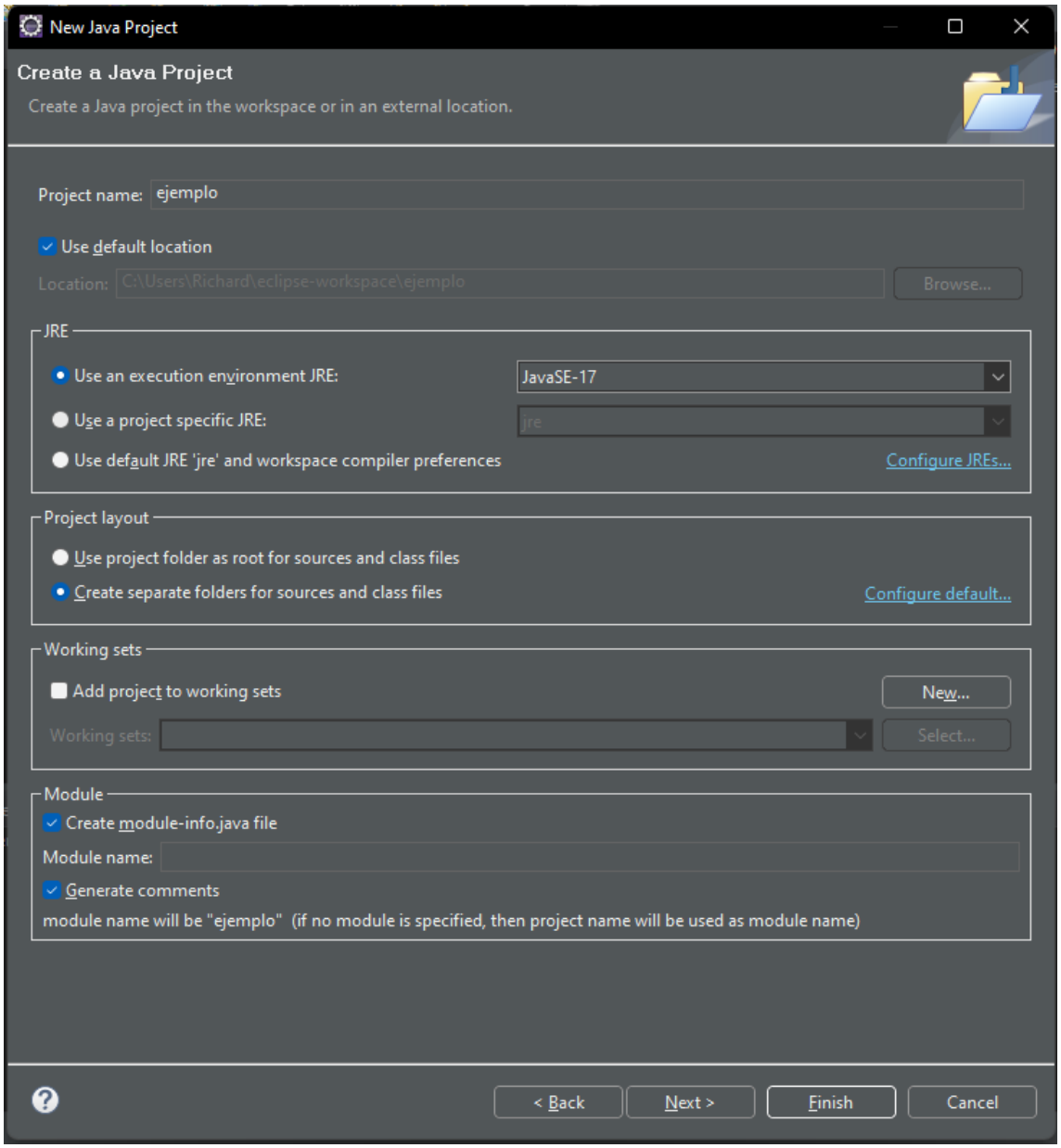

# Configuramos un proyecto Java

Nos dirigimos a eclipse a la opción **File**->**New -> Other-> Java Project**.

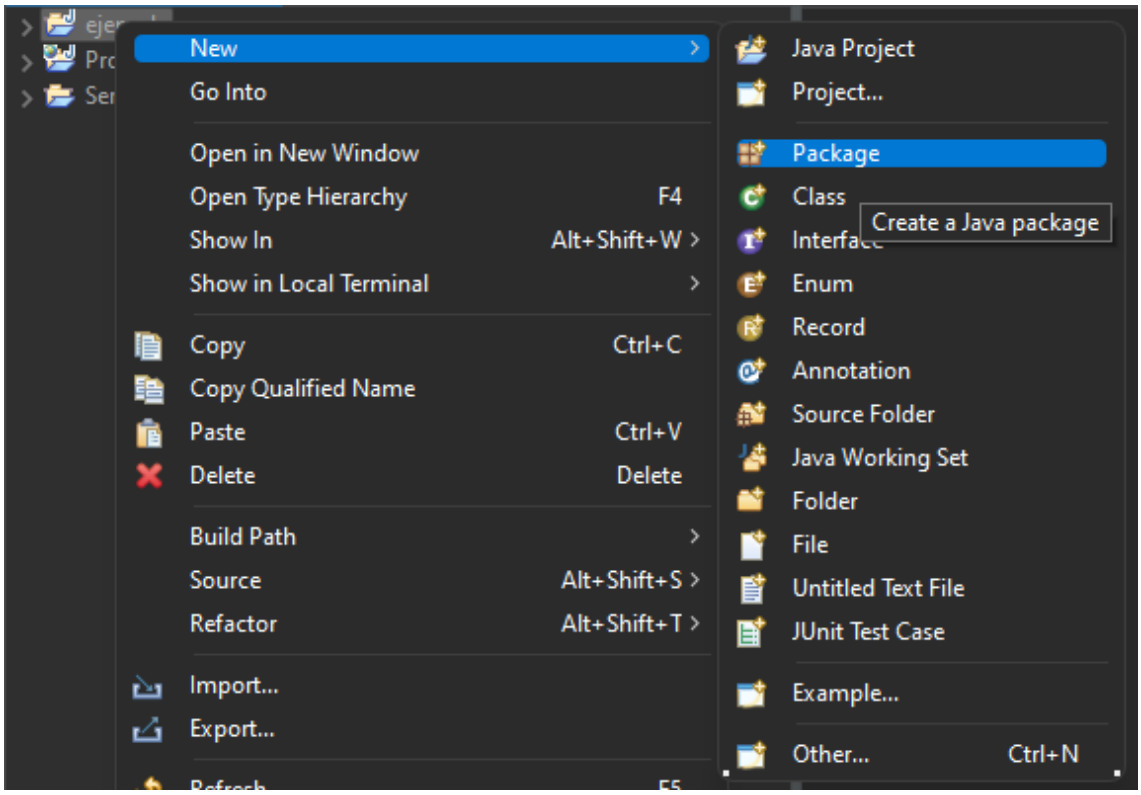




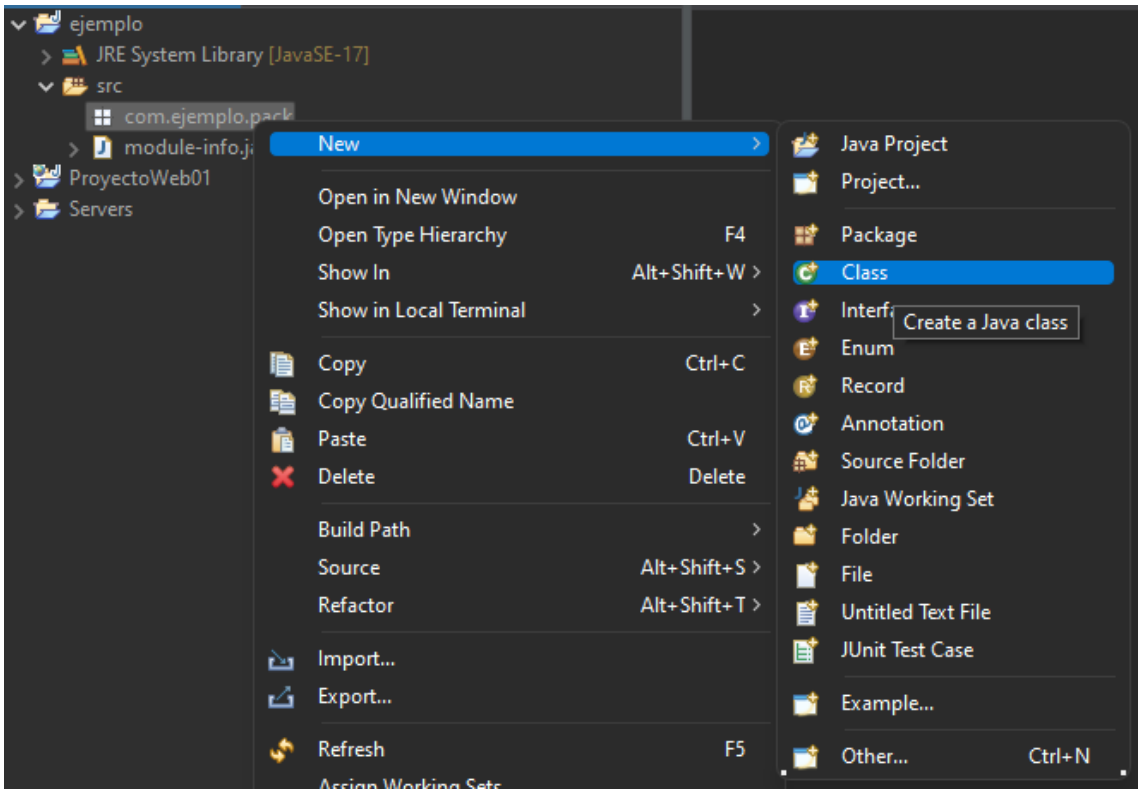

Le asignamos un nombre y le damos clic en **Finalizar**



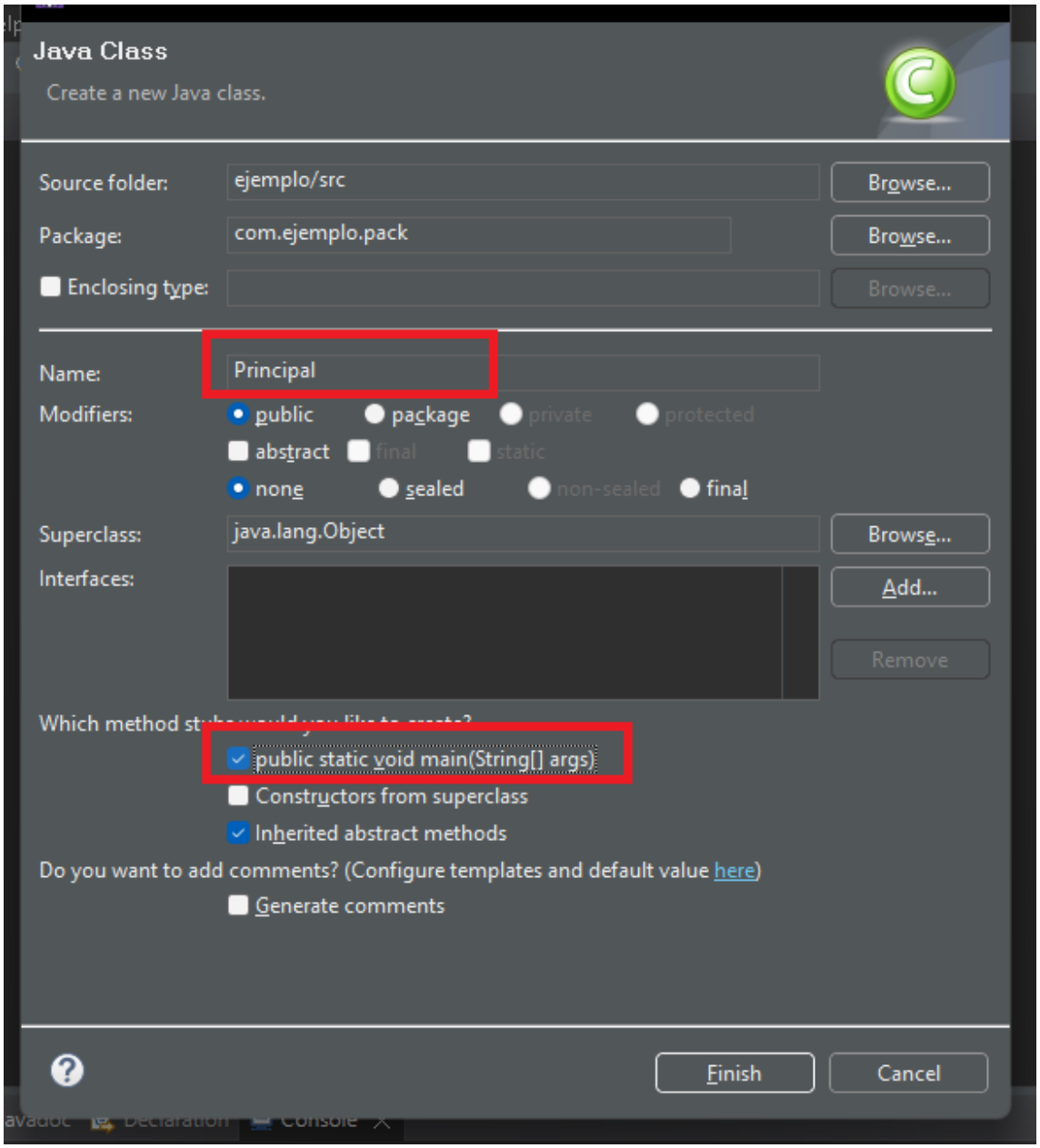

Agregamos un nuevo paquete (com.ejemplo.pack)




Agregamos una nueva clase




Tiene que respetar la convención, iniciamos con mayúscula y activamos para que pueda agregar el método main


Java Class
Create a new Java class.
Source folder:
ejemplo/src
Browse...
Package:
com.ejemplo.pack
Browse...
Enclosing type:
Browse...
Name:
Principal
Modifiers:
public
package
private
protected
abstract
final
static
none
sealed
non-sealed
final
Superclass:
java.lang.Object
Browse...
Interfaces:
Add...
Remove
Which method st
public static void main(String[] args)
Constructors from superclass
Inherited abstract methods
Do you want to add comments? (Configure templates and default value here)
Generate comments
Finish
Cancel

# Ejercicios con POO

1) Crea una clase llamada Cuenta que tendrá los siguientes atributos: titular y cantidad (puede tener decimales).

El titular será obligatorio y la cantidad es opcional. Crea dos constructores que cumpla lo anterior.

Crea sus métodos get, set y toString.

Tendrá dos métodos especiales:

ingresar(double cantidad): se ingresa una cantidad a la cuenta, si la cantidad introducida es negativa, no se hará nada.

retirar(double cantidad): se retira una cantidad a la cuenta, si restando la cantidad actual a la que nos pasan es negativa, la cantidad de la cuenta pasa a ser 0.

**Desarrollo**

```
public class Cuenta {

    //Atributos
    private String titular;
    private double cantidad;

    //Constructores
    public Cuenta(String titular) {
        this(titular, 0); //Sobrecarga
    }

    public Cuenta(String titular, double cantidad) {
        this.titular = titular;
        //Si la cantidad es menor que cero, lo ponemos a cero
        if (cantidad < 0) {
            this.cantidad = 0;
        } else {
            this.cantidad = cantidad;
        }
    }

    //Metodos
    public String getTitular() {
        return titular;
    }

    public void setTitular(String titular) {
        this.titular = titular;
    }

    public double getCantidad() {
        return cantidad;
    }

    public void setCantidad(double cantidad) {
```

```
        this.cantidad = cantidad;
    }

    /**
     * Ingresa dinero en la cuenta,
     * solo si es positivo la cantidad
     *
     * @param cantidad
     */
    public void ingresar(double cantidad) {
        if(cantidad > 0){
            this.cantidad += cantidad;
        }
    }

    /**
     * Retira una cantidad en la cuenta, si se quedara en negativo se
quedaria
     * en cero
     *
     * @param cantidad
     */
    public void retirar(double cantidad) {
        if (this.cantidad - cantidad < 0) {
            this.cantidad = 0;
        } else {
            this.cantidad -= cantidad;
        }
    }

    /**
     * Devuelve el estado del objeto
     *
     * @return
     */
    @Override
    public String toString() {
        return "El titular " + titular + " tiene " + cantidad + "
euros en la cuenta";
    }

}


public class CuentaApp {

    public static void main(String[] args) {

        Cuenta cuenta_1 = new Cuenta("DiscoDurodeRoer");
        Cuenta cuenta_2 = new Cuenta("Fernando", 300);

        //Ingresa dinero en las cuentas
        cuenta_1.ingresar(300);
        cuenta_2.ingresar(400);

        //Retiramos dinero en las cuentas
        cuenta_1.retirar(500);
        cuenta_2.retirar(100);

        //Muestro la informacion de las cuentas
```

```
        System.out.println(cuenta_1); // 0 euros
        System.out.println(cuenta_2); // 600 euros

    }

}
```

2) Haz una clase llamada Persona que siga las siguientes condiciones:

- Sus atributos son: nombre, edad, DNI, sexo (H hombre, M mujer), peso y altura. No queremos que se accedan directamente a ellos. Piensa que modificador de acceso es el más adecuado, también su tipo. Si quieres añadir algún atributo puedes hacerlo.
- Por defecto, todos los atributos menos el DNI serán valores por defecto según su tipo (0 números, cadena vacía para String, etc.). Sexo sera hombre por defecto, usa una constante para ello.
- Se implantarán varios constructores:
    - Un constructor por defecto.
    - Un constructor con el nombre, edad y sexo, el resto por defecto.
    - Un constructor con todos los atributos como parámetro.
- Los métodos que se implementaran son:
    - calcularIMC(): calculara si la persona está en su peso ideal (peso en kg/(altura^2 en m)), si esta fórmula devuelve un valor menor que 20, la función devuelve un -1, si devuelve un número entre 20 y 25 (incluidos), significa que esta por debajo de su peso ideal la función devuelve un 0 y si devuelve un valor mayor que 25 significa que tiene sobrepeso, la función devuelve un 1. Te recomiendo que uses constantes para devolver estos valores.
        - esMayorDeEdad(): indica si es mayor de edad, devuelve un booleano.
        - comprobarSexo(char sexo): comprueba que el sexo introducido es correcto. Si no es correcto, sera H. No sera visible al exterior.
        - toString(): devuelve toda la información del objeto.
        - generaDNI(): genera un número aleatorio de 8 cifras, genera a partir de este su número su letra correspondiente. Este método sera invocado cuando se construya el objeto. Puedes dividir el método para que te sea más fácil. No será visible al exterior.
        - Métodos set de cada parámetro, excepto de DNI.

Ahora, crea una clase ejecutable que haga lo siguiente:

- Pide por teclado el nombre, la edad, sexo, peso y altura.

- Crea 3 objetos de la clase anterior, el primer objeto obtendrá las anteriores variables pedidas por teclado, el segundo objeto obtendrá todos los anteriores menos el peso y la altura y el último por defecto, para este último utiliza los métodos set para darle a los atributos un valor.
- Para cada objeto, deberá comprobar si está en su peso ideal, tiene sobrepeso o por debajo de su peso ideal con un mensaje.
- Indicar para cada objeto si es mayor de edad.
- Por último, mostrar la información de cada objeto.

Puedes usar métodos en la clase ejecutable, para que os sea más fácil.

```
public class Persona {

    //Constantes
    /**
     * Sexo por defecto
     */
    private final static char SEXO_DEF = 'H';

    /**
     * El peso de la persona esta por debajo del peso ideal
     */
    public static final int INFRAPESO = -1;

    /**
     * El peso de la persona esta en su peso ideal
     */
    public static final int PESO_IDEAL = 0;

    /**
     * El peso de la persona esta por encima del peso ideal
     */
    public static final int SOBREPESO = 1;

    //Atributos
    /**
     * Nombre de la persona
     */
    private String nombre;

    /**
     * Edad de la persona
     */
    private int edad;

    /**
     * DNI de la persona, se genera al construir el objeto
     */
    private String DNI;

    /**
     * Sexo de la persona, H hombre M mujer
     */
    private char sexo;
```

```
    /**
     * Peso de la persona
     */
    private double peso;

    /**
     * Altura de la persona
     */
    private double altura;

    //Contructores
    /**
     * Constructor por defecto
     */
    public Persona() {
        this("", 0, SEXO_DEF, 0, 0);
    }

    /**
     * Constructor con 3 parametroe
     *
     * @param nombre de la persona
     * @param edad de la persona
     * @param sexo de la persona
     */
    public Persona(String nombre, int edad, char sexo) {
        this(nombre, edad, sexo, 0, 0);
    }

    /**
     * Constructor con 5 parametros
     *
     * @param nombre de la persona
     * @param edad de la persona
     * @param sexo de la persona
     * @param peso de la persona
     * @param altura de la persona
     */
    public Persona(String nombre, int edad, char sexo, double peso,
double altura) {
        this.nombre = nombre;
        this.edad = edad;
        this.peso = peso;
        this.altura = altura;
        generarDni();
        this.sexo = sexo;
        comprobarSexo();
    }

    //Métodos privados
    private void comprobarSexo() {

        //Si el sexo no es una H o una M, por defecto es H
        if (sexo != 'H' && sexo != 'M') {
            this.sexo = SEXO_DEF;
        }
    }

    private void generarDni() {
```

```
        final int divisor = 23;

        //Generamos un número de 8 digitos
        int numDNI = ((int) Math.floor(Math.random() * (100000000 -
10000000) + 10000000));
        int res = numDNI - (numDNI / divisor * divisor);

        //Calculamos la letra del DNI
        char letraDNI = generaLetraDNI(res);

        //Pasamos el DNI a String
        DNI = Integer.toString(numDNI) + letraDNI;
    }

    private char generaLetraDNI(int res) {
        char letras[] = {'T', 'R', 'W', 'A', 'G', 'M', 'Y',
            'F', 'P', 'D', 'X', 'B', 'N', 'J', 'Z',
            'S', 'Q', 'V', 'H', 'L', 'C', 'K', 'E'};

        return letras[res];
    }

    //Métodos publicos
    /**
     * Modifica el nombre de la persona
     *
     * @param nombre a cambiar
     */
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    /**
     * Modifica la edad de la persona
     *
     * @param edad a cambiar
     */
    public void setEdad(int edad) {
        this.edad = edad;
    }

    /**
     * Modifica el sexo de la persona, comprueba que es correcto
     *
     * @param sexo a cambiar
     */
    public void setSexo(char sexo) {
        this.sexo = sexo;
    }

    /**
     * Modifica el peso de la persona
     *
     * @param peso a cambiar
     */
    public void setPeso(double peso) {
        this.peso = peso;
    }
```

```
/**
 * Modifica la altura de la persona
 *
 * @param altura a cambiar
 */
public void setAltura(double altura) {
    this.altura = altura;
}

/**
 * Calcula el indice de masa corporal
 *
 * @return codigo numerico
 * <ul><li>-1: la persona esta por debajo de su peso ideal</li>
 * <li>0: la persona esta en su peso ideal</li>
 * <li>1: la persona esta por encima de su peso ideal</li></ul>
 */
public int calcularIMC() {
    //Calculamos el peso de la persona
    double pesoActual = peso / (Math.pow(altura, 2));
    //Segun el peso, devuelve un codigo
    if (pesoActual >= 20 && pesoActual <= 25) {
        return PESO_IDEAL;
    } else if (pesoActual < 20) {
        return INFRAPESO;
    } else {
        return SOBREPESO;
    }
}

/**
 * Indica si la persona es mayor de edad
 *
 * @return true si es mayor de edad y false es menor de edad
 */
public boolean esMayorDeEdad() {
    boolean mayor = false;
    if (edad >= 18) {
        mayor = true;
    }
    return mayor;
}

/**
 * Devuelve informacion del objeto
 *
 * @return cadena con toda la informacion
 */
@Override
public String toString() {
    String sexo;
    if (this.sexo == 'H') {
        sexo = "hombre";
    } else {
        sexo = "mujer";
    }
    return "Informacion de la persona:\n"
            + "Nombre: " + nombre + "\n"
            + "Sexo: " + sexo + "\n"
            + "Edad: " + edad + " años\n"
            + "DNI: " + DNI + "\n"
```

```
                + "Peso: " + peso + " kg\n"
                + "Altura: " + altura + " metros\n";
    }

}


import java.util.Locale;
import java.util.Scanner;
import javax.swing.JOptionPane;

public class PersonaApp_Scanner {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        sc.useDelimiter("\n");
        sc.useLocale(Locale.US);

        //Introducimos los datos
        System.out.println("Introduce el nombre");
        String nombre = sc.next();

        System.out.println("Introduce la edad");
        int edad = sc.nextInt();

        System.out.println("Introduce el sexo");
        char sexo = sc.next().charAt(0);

        System.out.println("Introduce el peso");
        double peso = sc.nextDouble();

        System.out.println("Introduce la altura");
        double altura = sc.nextDouble();

        //Creamos objetos con cada constructor
        Persona persona1 = new Persona();
        Persona persona2 = new Persona(nombre, edad, sexo);
        Persona persona3 = new Persona(nombre, edad, sexo, peso,
altura);

        //Los datos que no esten completos los insertamos con los
metodos set
        persona1.setNombre("Laura");
        persona1.setEdad(30);
        persona1.setSexo('M');
        persona1.setPeso(60);
        persona1.setAltura(1.60);

        persona2.setPeso(90.5);
        persona2.setAltura(1.80);

        //Usamos metodos para realizar la misma accion para cada
objeto
        System.out.println("Persona1");
        MuestraMensajePeso(persona1);
        MuestraMayorDeEdad(persona1);
        System.out.println(persona1.toString());
```

```
        System.out.println("Persona2");
        MuestraMensajePeso(persona2);
        MuestraMayorDeEdad(persona2);
        System.out.println(persona2.toString());

        System.out.println("Persona3");
        MuestraMensajePeso(persona3);
        MuestraMayorDeEdad(persona3);
        System.out.println(persona3.toString());
    }

    public static void MuestraMensajePeso(Persona p) {
        int IMC = p.calcularIMC();
        switch (IMC) {
            case Persona.PESO_IDEAL:
                System.out.println("La persona esta en su peso
ideal");
                break;
            case Persona.INFRAPESO:
                System.out.println("La persona esta por debajo de su
peso ideal");
                break;
            case Persona.SOBREPESO:
                System.out.println("La persona esta por encima de su
peso ideal");
                break;
        }
    }

    public static void MuestraMayorDeEdad(Persona p) {

        if (p.esMayorDeEdad()) {
            System.out.println("La persona es mayor de edad");
        } else {
            System.out.println("La persona no es mayor de edad");
        }
    }

}
```

3) Haz una clase llamada Password que siga las siguientes condiciones:

- Que tenga los atributos longitud y contraseña . Por defecto, la longitud sera de 8.
- Los constructores serán los siguiente:
    - Un constructor por defecto.
- Un constructor con la longitud que nosotros le pasemos. Generara una contraseña aleatoria con esa longitud.
- Los métodos que implementa serán:

- esFuerte(): devuelve un booleano si es fuerte o no, para que sea fuerte debe tener mas de 2 mayúsculas, mas de 1 minúscula y mas de 5 números.
- generarPassword(): genera la contraseña del objeto con la longitud que tenga.
- Método get para contraseña y longitud.
- Método set para longitud.

Ahora, crea una clase clase ejecutable:

- Crea un array de Passwords con el tamaño que tu le indiques por teclado.
- Crea un bucle que cree un objeto para cada posición del array.
- Indica también por teclado la longitud de los Passwords (antes de bucle).
- Crea otro array de booleanos donde se almacene si el password del array de Password es o no fuerte (usa el bucle anterior).
- Al final, muestra la contraseña y si es o no fuerte (usa el bucle anterior). Usa este simple formato:

contraseña1 valor_booleano1

contraseña2 valor_bololeano2

```
public class Password {

    //Constantes

    /**
     * Longitud por defecto
     */
    private final static int LONG_DEF=8;

    //Atributos

    /**
     * Longitud de la contraseña
     */
    private int longitud;
    /**
     * caracteres de la contraseña
     */
    private String contraseña;

    //Metodos publicos

    /**
     * Devuelve la longitud
     * @return longitud de la contraseña
     */
    public int getLongitud() {
        return longitud;
    }
```

```
    /**
     * Modifica la longitud de la contraseña
     * @param longitud a cambiar
     */
    public void setLongitud(int longitud) {
        this.longitud = longitud;
    }

    /**
     * Devuelve la contraseña
     * @return contraseña
     */
    public String getContraseña() {
        return contraseña;
    }

    /**
     * Genera una contraseña al azar con la longitud que este definida
     * @return contraseña
     */
    public String generaPassword (){
        String password="";
        for (int i=0;i<longitud;i++){
            //Generamos un numero aleatorio, segun este elige si
añadir una minuscula, mayuscula o numero
            int eleccion=((int)Math.floor(Math.random()*3+1));

            if (eleccion==1){
                char
minusculas=(char)((int)Math.floor(Math.random()*(123-97)+97));
                password+=minusculas;
            }else{
                if(eleccion==2){
                    char
mayusculas=(char)((int)Math.floor(Math.random()*(91-65)+65));
                    password+=mayusculas;
                }else{
                    char
numeros=(char)((int)Math.floor(Math.random()*(58-48)+48));
                    password+=numeros;
                }
            }
        }
        return password;
    }

    /**
     * Comprueba la fortaleza de la contraseña
     * @return
     */
    public boolean esFuerte(){
        int cuentanumeros=0;
        int cuentaminusculas=0;
        int cuentamayusculas=0;
        //Vamos caracter a caracter y comprobamos que tipo de caracter
es
        for (int i=0;i<contraseña.length();i++){
                if (contraseña.charAt(i)>=97 &&
contraseña.charAt(i)<=122){
```

```
                    cuentaminusculas+=1;
                }else{
                    if (contraseña.charAt(i)>=65 &&
contraseña.charAt(i)<=90){
                        cuentamayusculas+=1;
                }else{
                    cuentanumeros+=1;
                    }
                }
            }
            //Si la constraseña tiene mas de 5 numeros, mas de 1
minuscula y mas de 2 mayusculas
            if (cuentanumeros>=5 && cuentaminusculas>=1 &&
cuentamayusculas>=2){
            return true;
        }else{
            return false;
        }
    }

    //Constructores
    /**
     * Crea una contraseña al azar
     */
    public Password (){
        this(LONG_DEF);
    }

    /**
     * La contraseña sera la pasada por parametro
     * @param longitud
     */
    public Password (int longitud){
        this.longitud=longitud;
        contraseña=generaPassword();
    }
}


import javax.swing.JOptionPane;
public class PasswordApp {

    public static void main(String[] args) {

        //Introducimos el tamaño del array y la longitud del password
        String texto=JOptionPane.showInputDialog("Introduce un tamaño
para el array");
        int tamanio=Integer.parseInt(texto);

        texto=JOptionPane.showInputDialog("Introduce la longitud del
password");
        int longitud=Integer.parseInt(texto);

        //Creamos los arrays
        Password listaPassword[]=new Password[tamanio];
        boolean fortalezaPassword[]=new boolean[tamanio];

        //Creamos objetos, indicamos si es fuerte y mostramos la
contraseña y su fortaleza.
        for(int i=0;i<listaPassword.length;i++){
```

```
            listaPassword[i]=new Password(longitud);
            fortalezaPassword[i]=listaPassword[i].esFuerte();
            System.out.println(listaPassword[i].getContraseña()+"
"+fortalezaPassword[i]);
        }
    }

}
```

4) Crearemos una supeclase llamada Electrodomestico con las siguientes características:

- Sus atributos son precio base, color, consumo energético (letras entre A y F) y peso. Indica que se podrán heredar.
- Por defecto, el color sera blanco, el consumo energético sera F, el precioBase es de 100 € y el peso de 5 kg. Usa constantes para ello.
- Los colores disponibles son blanco, negro, rojo, azul y gris. No importa si el nombre esta en mayúsculas o en minúsculas.
- Los constructores que se implementaran serán
    - Un constructor por defecto.
    - Un constructor con el precio y peso. El resto por defecto.
    - Un constructor con todos los atributos.
- Los métodos que implementara serán:
    - Métodos get de todos los atributos.
    - comprobarConsumoEnergetico(char letra): comprueba que la letra es correcta, sino es correcta usara la letra por defecto. Se invocara al crear el objeto y no sera visible.
    - comprobarColor(String color): comprueba que el color es correcto, sino lo es usa el color por defecto. Se invocara al crear el objeto y no sera visible.
    - precioFinal(): según el consumo energético, aumentara su precio, y según su tamaño, también. Esta es la lista de precios:

| Letra | Precio |
|---|---|
| A | 100 € |

| Letra | Precio |
|---|---|
| B | 80 € |
| C | 60 € |
| D | 50 € |
| E | 30 € |
| F | 10 € |

| Tamaño | Precio |
|---|---|
| Entre 0 y 19 kg | 10 € |
| Entre 20 y 49 kg | 50 € |
| Entre 50 y 79 kg | 80 € |
| Mayor que 80 kg | 100 € |

Crearemos una subclase llamada Lavadora con las siguientes características:

- Su atributo es carga, ademas de los atributos heredados.
- Por defecto, la carga es de 5 kg. Usa una constante para ello.
- Los constructores que se implementaran serán:
    - Un constructor por defecto.
    - Un constructor con el precio y peso. El resto por defecto.
    - Un constructor con la carga y el resto de atributos heredados. Recuerda que debes llamar al constructor de la clase padre.

- Los métodos que se implementara serán:
    - Método get de carga.
    - precioFinal():, si tiene una carga mayor de 30 kg, aumentara el precio 50 €, sino es así no se incrementara el precio. Llama al método padre y añade el código necesario. Recuerda que las condiciones que hemos visto en la clase Electrodomestico también deben afectar al precio.

Crearemos una subclase llamada Television con las siguientes características:

- Sus atributos son resolución (en pulgadas) y sintonizador TDT (booleano), ademas de los atributos heredados.
- Por defecto, la resolución sera de 20 pulgadas y el sintonizador sera false.
- Los constructores que se implementaran serán:
    - Un constructor por defecto.
    - Un constructor con el precio y peso. El resto por defecto.
    - Un constructor con la resolución, sintonizador TDT y el resto de atributos heredados. Recuerda que debes llamar al constructor de la clase padre.
- Los métodos que se implementara serán:
    - Método get de resolución y sintonizador TDT.
    - precioFinal(): si tiene una resolución mayor de 40 pulgadas, se incrementara el precio un 30% y si tiene un sintonizador TDT incorporado, aumentara 50 €. Recuerda que las condiciones que hemos visto en la clase Electrodomestico también deben afectar al precio.

Ahora crea una clase ejecutable que realice lo siguiente:

- Crea un array de Electrodomesticos de 10 posiciones.
- Asigna a cada posición un objeto de las clases anteriores con los valores que desees.
- Ahora, recorre este array y ejecuta el método precioFinal().
- Deberás mostrar el precio de cada clase, es decir, el precio de todas las televisiones por un lado, el de las lavadoras por otro y la suma de los Electrodomesticos (puedes crear objetos Electrodomestico, pero recuerda que Television y Lavadora también son electrodomésticos). Recuerda el uso operador instanceof.

Por ejemplo, si tenemos un Electrodomestico con un precio final de 300, una lavadora de 200 y una televisión de 500, el resultado final sera de 1000 (300+200+500) para electrodomésticos, 200 para lavadora y 500 para televisión.

```
public class Electrodomestico {

    //Constantes

    /**
     * Color por defecto
     */
    protected final static String COLOR_DEF="blanco";

    /**
     * Consumo energetico por defecto
     */
    protected final static char CONSUMO_ENERGETICO_DEF='F';

    /**
     * Precio base por defecto
     */
    protected final static double PRECIO_BASE_DEF=100;

    /**
     * Peso por defecto
     */
    protected final static double PESO_DEF=5;

    //Atributos

    /**
     * El precio base del electrodomestico
     */
    protected double precioBase;

    /**
     * Color del electrodomestico
     */
    protected String color;

    /**
     * Indica el consumo energetico del electrodomestico
     */
    protected char consumoEnergetico;

    /**
     * Peso del electrodomestico
     */
    protected double peso;

    //Métodos privados

    private void comprobarColor(String color){

        //Colores disponibles
        String colores[]={"blanco", "negro", "rojo", "azul", "gris"};
        boolean encontrado=false;

        for(int i=0;i<colores.length && !encontrado;i++){

            if(colores[i].equals(color)){
                encontrado=true;
```

```
        }

    }

    if(encontrado){
        this.color=color;
    }else{
        this.color=COLOR_DEF;
    }


}

/**
 * Comprueba el consumo energetico
 * Solo mayusculas, si es una 'a' no lo detecta como una 'A'
 * @param consumoEnergetico
 */
public void comprobarConsumoEnergetico(char consumoEnergetico){

    if(consumoEnergetico>=65 && consumoEnergetico<=70){
        this.consumoEnergetico=consumoEnergetico;
    }else{
        this.consumoEnergetico=CONSUMO_ENERGETICO_DEF;
    }

}

//Métodos publicos
/**
 * Devuelve el precio base del electrodomestico
 * @return precio base del electrodomestico
 */
public double getPrecioBase() {
    return precioBase;
}
/**
 * Devuelve el color del electrodomestico
 * @return color del elesctrodomestico
 */
public String getColor() {
    return color;
}

 /**
 * Devuelve el consumo energetico del electrodomestico
 * @return consumo energetico del electrodomestico
 */
public char getConsumoEnergetico() {
    return consumoEnergetico;
}
/**
 * Devuelve el peso del electrodomestico
 * @return peso del electrodomestico
 */
public double getPeso() {
    return peso;
}
/**
 * Precio final del electrodomestico
```

```
     * @return precio final del electrodomestico
     */
    public double precioFinal(){
        double plus=0;
        switch(consumoEnergetico){
            case 'A':
                plus+=100;
                break;
            case 'B':
                plus+=80;
                break;
            case 'C':
                plus+=60;
                break;
            case 'D':
                plus+=50;
                break;
            case 'E':
                plus+=30;
                break;
            case 'F':
                plus+=10;
                break;
        }

        if(peso>=0 && peso<19){
            plus+=10;
        }else if(peso>=20 && peso<49){
            plus+=50;
        }else if(peso>=50 && peso<=79){
            plus+=80;
        }else if(peso>=80){
            plus+=100;
        }

        return precioBase+plus;
    }

    //Constructores

    /**
     * Contructor por defecto
     */
    public Electrodomestico(){
        this(PRECIO_BASE_DEF, PESO_DEF, CONSUMO_ENERGETICO_DEF,
COLOR_DEF);
    }

    /**
     * Contructor con 2 parametros
     * @param precioBase del electrodomestico
     * @param peso del electrodomestico
     */
    public Electrodomestico(double precioBase, double peso){
        this(precioBase, peso, CONSUMO_ENERGETICO_DEF, COLOR_DEF);
    }

    /**
     * Constructor con 4 parametros
     * @param precioBase
```

```
     * @param peso
     * @param consumoEnergetico
     * @param color
     */
    public Electrodomestico(double precioBase, double peso, char
consumoEnergetico, String color){
        this.precioBase=precioBase;
        this.peso=peso;
        comprobarConsumoEnergetico(consumoEnergetico);
        comprobarColor(color);
    }

}


public class Lavadora extends Electrodomestico{

    //Constantes

    /**
     * Carga por defecto
     */
    private final static int CARGA_DEF=5;

    //Atributos

    /**
     * Carga de la lavadora
     */
    private int carga;

    //Métodos publicos

    /**
     * Devuelve la carga de la lavadora
     * @return
     */
    public int getCarga() {
        return carga;
    }

    /**
     * Precio final de la lavadora
     * @return precio final de la lavadora
     */
    public double precioFinal(){
        //Invocamos el método precioFinal del método padre
        double plus=super.precioFinal();

        //añadimos el código necesario
        if (carga>30){
            plus+=50;
        }

        return plus;
    }

    //Constructor
```

```
    /**
     * Contructor por defecto
     */
    public Lavadora(){
        this(PRECIO_BASE_DEF, PESO_DEF, CONSUMO_ENERGETICO_DEF,
COLOR_DEF, CARGA_DEF);
    }

    /**
     * Constructor con 2 parametros
     * @param precioBase
     * @param peso
     */
    public Lavadora(double precioBase, double peso){
        this(precioBase, peso, CONSUMO_ENERGETICO_DEF, COLOR_DEF,
CARGA_DEF);
    }

    /**
     * Constructor con 5 parametros
     * @param precioBase
     * @param peso
     * @param consumoEnergetico
     * @param color
     * @param carga
     */
    public Lavadora(double precioBase, double peso, char
consumoEnergetico, String color, int carga){
        super(precioBase,peso, consumoEnergetico,color);
        this.carga=carga;
    }
}


public class Television extends Electrodomestico{

    //Constantes

    /**
     * Resolucion por defecto
     */
    private final static int RESOLUCION_DEF=20;

    //Atributos

    /**
     * Resolucion del televisor
     */
    private int resolucion;

    /**
     * Indica si tiene o no sintonizadorTDT
     */
    private boolean sintonizadorTDT;

    //Métodos publicos

    /**
     * Precio final de la television
     * @return precio final de la television
```

```
     */
    public double precioFinal(){
        //Invocamos el método precioFinal del método padre
        double plus=super.precioFinal();

        //Añadimos el codigo necesario
        if (resolucion>40){
            plus+=precioBase*0.3;
        }
        if (sintonizadorTDT){
            plus+=50;
        }

        return plus;
    }

    //Constructor

    /**
     * Constructor por defecto
     */
    public Television(){
        this(PRECIO_BASE_DEF, PESO_DEF, CONSUMO_ENERGETICO_DEF,
COLOR_DEF, RESOLUCION_DEF, false);
    }

    /**
     * Constructor con 2 parametros
     * @param precioBase
     * @param peso
     */
    public Television(double precioBase, double peso){
        this(precioBase, peso, CONSUMO_ENERGETICO_DEF, COLOR_DEF,
RESOLUCION_DEF, false);
    }

    /**
     * Contructor con 6 parametros
     * @param precioBase
     * @param peso
     * @param consumoEnergetico
     * @param color
     * @param resolucion
     * @param sintonizadorTDT
     */
    public Television(double precioBase, double peso, char
consumoEnergetico, String color, int resolucion, boolean
sintonizadorTDT){
        super(precioBase, peso, consumoEnergetico, color);
        this.resolucion=resolucion;
        this.sintonizadorTDT=sintonizadorTDT;
    }
}



public class ElectrodomesticosApp {

    public static void main(String[] args) {
```

```
        //Creamos un array de Electrodomesticos
        Electrodomestico listaElectrodomesticos[]=new
Electrodomestico[10];

        //Asignamos cada una de las posiciones como queramos
        listaElectrodomesticos[0]=new Electrodomestico(200, 60, 'C',
"Verde");
        listaElectrodomesticos[1]=new Lavadora(150, 30);
        listaElectrodomesticos[2]=new Television(500, 80, 'E',
"negro", 42, false);
        listaElectrodomesticos[3]=new Electrodomestico();
        listaElectrodomesticos[4]=new Electrodomestico(600, 20, 'D',
"gris");
        listaElectrodomesticos[5]=new Lavadora(300, 40, 'Z', "blanco",
40);
        listaElectrodomesticos[6]=new Television(250, 70);
        listaElectrodomesticos[7]=new Lavadora(400, 100, 'A', "verde",
15);
        listaElectrodomesticos[8]=new Television(200, 60, 'C',
"naranja", 30, true);
        listaElectrodomesticos[9]=new Electrodomestico(50, 10);

        //Creamos las variables que usaremos para almacenar la suma de
los precios
        double sumaElectrodomesticos=0;
        double sumaTelevisiones=0;
        double sumaLavadoras=0;

        //Recorremos el array invocando el metodo precioFinal
        for(int i=0;i<listaElectrodomesticos.length;i++){
            /*
             * Cuando una Television o una Lavadora este en la
posicion del array actual,
             * pasara por su clase y por la de electrodomestico, ya
que una television es un electrodomestico.
             * Ejecutamos en cada uno su propia version del metodo
precioFinal
             */

            if(listaElectrodomesticos[i] instanceof Electrodomestico){
                sumaElectrodomesticos+=listaElectrodomesticos[i].preci
oFinal();
            }
            if(listaElectrodomesticos[i] instanceof Lavadora){
                sumaLavadoras+=listaElectrodomesticos[i].precioFinal()
;
            }
            if(listaElectrodomesticos[i] instanceof Television){
                sumaTelevisiones+=listaElectrodomesticos[i].precioFina
l();
            }
        }

        //Mostramos los resultados
        System.out.println("La suma del precio de los
electrodomesticos es de "+sumaElectrodomesticos);
        System.out.println("La suma del precio de las lavadoras es de
"+sumaLavadoras);
```

```
        System.out.println("La suma del precio de las televisiones es
de "+sumaTelevisiones);

    }

}
```

URL

https://www.discoduroderoer.es/ejercicios-propuestos-y-resueltos-programacion-orientado-a-objetos-java/