Sobrecarga de Métodos

Retomamos el ejercicio: Realizar una clase que represente Alumnos que tenga como atributos: nombre, apellido, edad, curso. Realizar el constructor de la clase y un método que muestre los datos de los alumnos.

Vamos agregar un atributo más: el dni de los alumnos.

Define una clase Bombero considerando los siguientes **atributos de clase**: nombre (String), apellidos (String), edad (int), especialista (boolean). Define **un constructor** que reciba los parámetros necesarios para la inicialización y un método para mostrar los datos y además que si el bombero creado es especialista que muestre un cartel indicando que lo es. Compila el código para comprobar que no presenta errores, crea un objeto y comprueba que se inicializa correctamente consultando el valor de sus atributos después de haber creado el objeto.

Modificadores de acceso

Los modificadores de acceso, como su nombre indica, determinan desde qué clases se puede acceder a un determinado elemento.

En Java tenemos 4 tipos: **public, private, protected y el tipo por defecto,** que no tiene ninguna palabra clave asociada, pero se suele conocer como default o package-private.

Si no especificamos ningún modificador de acceso se utiliza el nivel de acceso por defecto, que consiste en que el elemento puede ser accedido sólo desde las clases que pertenezcan al mismo paquete.

El nivel de acceso **public** permite a acceder al elemento desde cualquier clase, independientemente de que esta pertenezca o no al paquete en que se encuentra el elemento.

private, por otro lado, es el modificador más restrictivo y especifica que los elementos que lo utilizan sólo pueden ser accedidos desde la clase en la que se encuentran. Este modificador sólo puede utilizarse sobre los miembros de una clase y sobre interfaces y clases internas, no sobre clases o interfaces de primer nivel, dado que esto no tendría sentido.

Es importante destacar también que private convierte los elementos en privados para otras clases, no para otras instancias de la clase. Es decir, un objeto de una determinada clase puede acceder a los miembros privados de otro objeto de la misma clase, por lo que algo como lo siguiente sería perfectamente válido:

```
class MiObjeto {
  private short valor = 0;

  MiObjeto(MiObjeto otro) {
    valor = otro.valor;
  }
```

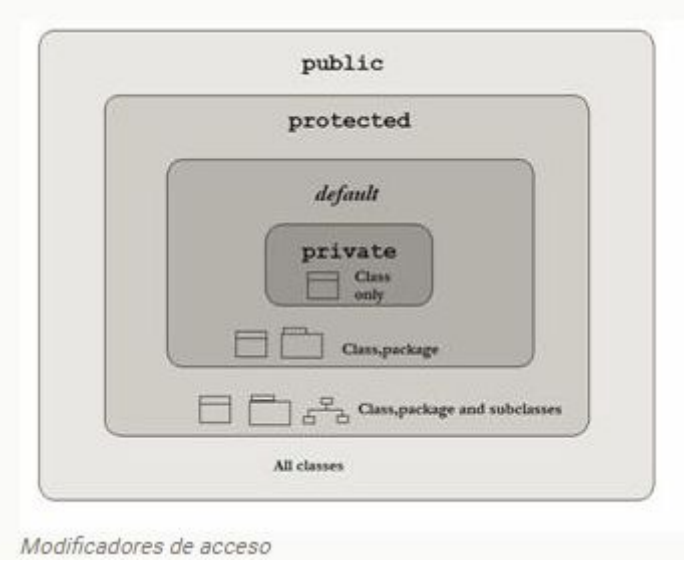
```
}
```

El modificador **protected**, por último, indica que los elementos sólo pueden ser accedidos desde su mismo paquete (como el acceso por defecto) y desde cualquier clase que extienda la clase en que se encuentra, independientemente de si esta se encuentra en el mismo paquete o no. Este modificador, como private, no tiene sentido a nivel de clases o interfaces no internas.

Los distintos modificadores de acceso quedan resumidos en la siguiente tabla:

Modificadores de acceso

	La misma clase	Otra clase del mismo paquete	Subclase de otro paquete	Otra clase de otro paquete
public	X	X	X	X
protected	X	X	X	
default	X	X		
private	X			



Modificadores de acceso

Visibilidad	Significado	Java	UML
Pública	Se puede acceder al miembro de la clase desde cualquier lugar.	`public`	+
Protegida	Sólo se puede acceder al miembro de la clase desde la propia clase o desde una clase que herede de ella.	`protected`	#
Por defecto	**Se puede acceder a los miembros de una clase desde cualquier clase en el mismo paquete**		~
Privada	Sólo se puede acceder al miembro de la clase desde la propia clase.	`private`	-

Vamos hacer un Ejemplo:

Primero vamos a crear un proyecto en el cual definimos dos paquetes paquete1 y paquete2. En el paquete1 creamos dos clases: Clase1 y Clase2. En el segundo paquete, paquete2, creamos una tercer clase: Clase3.

En la Clase1 vamos a crear un atributo del tipo int: primero vamos a definir nuestro atributo como **public, con esto decimos que nuestro atributo puede ser accesado por cualquier clase del mismo paquete,** en nuestro ejemplo quiere decir que podemos acceder desde cualquier clase que este en el paquete1:

```java
package paquete1;

// Ejemplo de  Encapsulamiento
public class Clase1 {
    public int edad;

}
```

Entonces en nuestra Clase2 podemos crear un objeto un de la Clase1:

```java
package paquete1;

public class Clase2 {
    public static void main(String [] args){
        Clase1 objeto1 = new Clase1();
        objeto1.edad = -5;
    }
}
```

Si definimos nuestros atributos como **public** corremos el riesgo que desde otra clase nos modifiquen los valores de los atributos, provocando errores en la lógica de nuestros programas, por tal motivo podemos definir nuestros atributos como **private**:

```java
package paquete1;

// Ejemplo de  Encapsulamiento
public class Clase1 {
    private int edad;

}
```

```java
package paquete1;

public class Clase2 {
    public static void main(String [] args){
        Clase1 objeto1 = new Clase1();
        objeto1.edad = -5;
    }
}
```

A definir nuestro atributo como private, java nos marca el error que no podemos acceder al atributo, y de esta manera evitar futuros problemas, por ejemplo.
Nuestro atributo edad, ahora solo puede ser accedido desde métodos que pertenezcan a la Clase1 solamente.

Entonces, el encapsulamiento nos permite ocultar atributos y métodos.

Sigamos con nuestro ejemplo:
Ahora si a nuestro atributo le dejamos el modificador de acceso por defecto, vamos a ver que podemos acceder desde la Clase2:

```java
package paquete1;

// Ejemplo de  Encapsulamiento
public class Clase1 {
    int edad;

}
```

```java
package paquete1;

public class Clase2 {
    public static void main(String [] args){
        Clase1 objeto1 = new Clase1();
        objeto1.edad = -5;
    }
}
```

Entonces al definirlo así podemos accederlo desde cualquier clase del mismo paquete.

En la Clase3, también creamos un método main, he intentamos acceder al atributo de la Clase1:

```java
package paquete2;

import paquete1.Clase1;

public class Clase3 {
    public static void main(String [] args){
        Clase1 objeto1 = new Clase1();
        objeto1.edad = -5;
    }
}
```

Como vemos java nos indica que no se puede, porque nuestro atributo tiene el modificador de acceso por defecto y solo puede ser accedido desde clases del mismo paquete, y la Clase3 pertenece a otro paquete: en nuestro ejemplo al paquete2.

Ahora si lo definimos como public:

```java
package paquete1;

// Ejemplo de  Encapsulamiento
public class Clase1 {
    public int edad;

}
```

vamos a ver que :

```java
package paquete2;

import paquete1.Clase1;

public class Clase3 {
    public static void main(String [] args){
        Clase1 objeto1 = new Clase1();
        objeto1.edad = -5;
    }
}
```

Ahora si podemos acceder desde clases de otro paquete.
En el caso de que definamos al atributo como **private**, como habíamos visto anteriormente sólo podemos acceder al mismo a través de métodos definidos en la misma clase.

```java
package paquete2;

import paquete1.Clase1;

public class Clase3 {
    public static void main(String [] args){
        Clase1 objeto1 = new Clase1();
        objeto1.edad = -5;
    }
}
```

- Que los miembros de una clase sean privados quiere decir que no se puede acceder a ellos desde el exterior de la clase (ni siquiera desde sus propias subclases), lo que permite mantener la encapsulación de los objetos.

Métodos accesores:
Son los getters y los setters para poder inicializar y obtener los valores de nuestros atributos.

```java
//Encapsulamiento y métodos accesores (Setters y Getters)
package paquete1;

public class Clase1 {
    private int edad;

    //Metodo Setter: Establecemos la edad
    public void SetEdad(int edad){
        this.edad = edad;
    }

    //Método Getter: Mostramos la edad
    public int GetEdad(){
        return edad;
    }
}
```

```java
package paquete1;

public class Clase2 {
    public static void main(String[] args){
        Clase1 objeto1 = new Clase1();

        objeto1.SetEdad(10);
        System.out.println("La edad es: "+objeto1.GetEdad());
    }
}
```