Patricia Alonso Ayllón

Interfaces como contrato: con métodos default y privados

¿Se pueden utilizar métodos privados en una interfaz? Aunque nos parezca una pregunta sin sentido ya que las interfaces están formadas por métodos públicos, a partir de Java 8 se pueden usar métodos con código y estos contener métodos privados.

Los métodos con código pueden ser de dos tipos:

- **Static Methods:** Permiten invocar a un método en cualquier clase independientemente de la instancia y sin tener que crear un objeto.
- **Default Methods**: Son métodos con cuerpo en la interfaz que permiten que las clases que hereden la interfaz no tendrán que sobreescribirlos ya que vienen por "defecto".

Vamos usar un ejemplo para explicarlos:

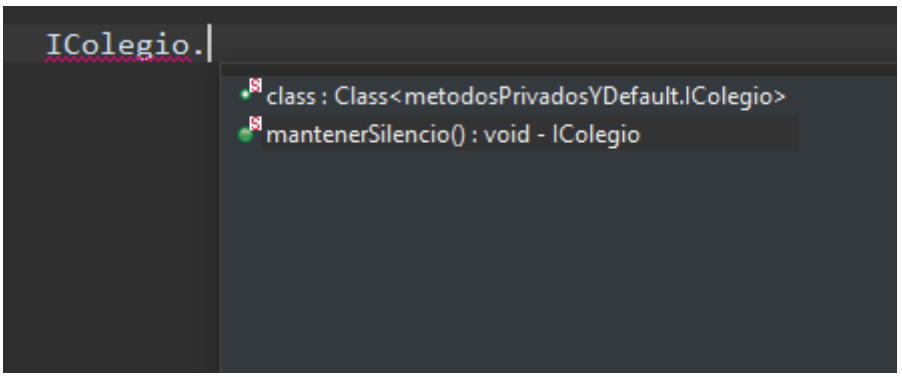
Tenemos un colegio, que está formado por personas (profesores, becarios y estudiantes)

```java
package metodosPrivadosYDefault;

public interface IColegio {

    //
    public static void mostrarSilencio() {

        System.out.println("Todos se callan");

    }

}

```

En la interfaz IColegio tenemos el método estático mostrarSilencio()..

¿Qué es un método estático?

> Que un método sea estático nos permite llamar al método mediante la clase(en este caso una interfaz ya que las clases la implementan) sin tener que llamarlo instanciando un objeto.



El static nos permite que el método tenga cuerpo ya que se llama directamente desde la interfaz.

Creamos las otras dos interfaces: IEstudios e ITrabajador

```java
package metodosPrivadosYDefault;

public interface IEstudios{

    public default double hacerExamen (double notaRandom){

        return notaRandom;
    }

    public default void mostrarAprobado(double notaRandom) {

        if(notaRandom>5){
            System.out.println("El alumno ha aprobado");
        }else {
            System.out.println("El alumno ha suspendido");
        }
    }

    
}
```

```java
package metodosPrivadosYDefault;

import java.util.List;

public interface ITrabajador {

	public default int pasarLista(List<Persona> listaAlumnos) {
		int numeroAlumnos=0;
		numeroAlumnos=listaAlumnos.size();
		return numeroAlumnos;
	}

	public default void darClase(String dni) {
		System.out.println("Ha comenzado a explicar");
	}

}

```

Aunque sean interfaces los métodos tienen cuerpos ¿cómo puede ser esto si los métodos abstractos no permiten tener cuerpo? Esto se debe a que son default, al incluirlo en un método abstracto en una interfaz "engañamos" a java para que aunque se reconozca como una interfaz actúen como clases madres, así "pudiendo" una clase heredar de más de una (en realidad implementa una interfaz con métodos default).

Vamos a comprobarlo:

Primero vamos a terminar de crear las clases restantes: persona, alumno, profesor, practicante.

```java
package metodosPrivadosYDefault;

public class Persona {

	private String nombre;
	private String apellidos;
	private int edad;
	private String dni;

}

```

```java
package metodosPrivadosYDefault;

public class Alumno extends Persona implements IEstudios, IColegio{

    public Alumno(String nombre, String apellidos, int edad, String dni) {
        super(nombre, apellidos, edad, dni);
        // TODO Auto-generated constructor stub
    }

    @Override
    public String toString() {
        return super.toString()+"Alumno";
    }


    public double hacerExamen(double notaRandom) {
        // TODO Auto-generated method stub
        return IEstudios.super.hacerExamen(notaRandom);
    }


    public void mostrarAprobado(double nota) {
        // TODO Auto-generated method stub
        IEstudios.super.mostrarAprobado(nota);
    }
```

```java
package metodosPrivadosYDefault;

import java.util.List;

public class Profesor extends Persona implements ITrabajador, IColegio{

    public Profesor(String nombre, String apellidos, int edad, String dni) {
        super(nombre, apellidos, edad, dni);
        // TODO Auto-generated constructor stub
    }

    @Override
    public String toString() {
        return super.toString()+"Profesor";
    }

    public int pasarLista(List<Persona> listaAlumnos) {
        // TODO Auto-generated method stub
        return ITrabajador.super.pasarLista(listaAlumnos);
    }


    public void darClase(String dni) {
        // TODO Auto-generated method stub
        ITrabajador.super.darClase(dni);
```

```java
import java.util.List;

public class Practicante extends Persona implements IEstudios, ITrabajador, IColegio{

    public Practicante(String nombre, String apellidos, int edad, String dni) {
        super(nombre, apellidos, edad, dni);
        // TODO Auto-generated constructor stub
    }
    @Override
    public String toString() {
        return super.toString()+"Practicante";
    }


    public double hacerExamen(double notaRandom) {
        // TODO Auto-generated method stub
        return IEstudios.super.hacerExamen(notaRandom);
    }


    public void mostrarAprobado(double nota) {
        // TODO Auto-generated method stub
        IEstudios.super.mostrarAprobado(nota);
```

Una vez creadas vemos que profesor, practicante y alumno heredan de la clase Persona todos sus atributos, reescribimos los métodos porque los objetos son tipo Persona, si fueran Alumno, Practicante o Profesor no haría falta ya que vendrían por “defecto”

Después de esto crearemos la clase crud para crear los métodos como agregar, borrar… que utilizaremos en el main. Vamos a crear un menú para probar a llamar a los métodos:

```java
Persona p1=new Profesor("Patty", "Ayllón", 21, "1A");
Persona p2=new Practicante("Iván", "Machuca", 24, "2B");
Persona p3=new Alumno("Pedro", "Laffon", 12, "3C");
Persona p4=new Profesor("Carmen", "López", 43, "4D");
Persona p5=new Practicante("Dani", "Martínez", 19, "5I");
Persona p6=new Alumno("Maria", "Sánchez", 11, "6G");
```

Los objetos ya creados

```java
System.out.println("Indique el dni del alumno:");
dni=sc.nextLine();
System.out.println(c.buscarEnListaProfesores(dni, listaAlumnos));
notaRandom=rnd.nextDouble(hasta-desde+1)+desde;
System.out.println(p.hacerExamen(notaRandom));
IColegio.mostrarSilencio();
p.mostrarAprobado(notaRandom);
```

Para llamar al método static utilizamos la interfaz y para llamar al método default lo hacemos mediante un objeto.