Clases abstractas

ÍNDICE

1. Introducción

Una clase abstracta es una clase de la que no se pueden crear objetos. Su utilidad es permitir que otras clases deriven de ella, proporcionándoles un marco o modelo que deben seguir y algunos métodos de utilidad general. Las clases abstractas se declaran anteponiéndoles la palabra **`abstract`**. Por ejemplo, podemos crear la siguiente clase abstracta para modelizar figuras geométricas:

```
public abstract class FiguraGeometrica {
	// ...
}
```

Una clase **`abstract`** puede tener métodos declarados como **`abstract`**, en cuyo caso no se da definición del método. Si una clase tiene algún método abstracto es obligatorio que la clase sea abstracta.

En cualquier subclase, el método definido como abstracto en la superclase deberá ser implementado o volver a declararse como **`abstract`** (el método y la subclase).

Es importante resaltar que una clase abstracta puede tener métodos que no son abstractos, es decir, tener comportamiento definido. Aunque no se puedan crear objetos de esta clase, sus subclases heredarán el método completamente a punto para ser utilizado.

Importante. Como los métodos **`static`** no pueden ser redefinidos, un método **`abstract`** no puede ser **`static`**.

2. Ejemplo sencillo de clase abstracta

2.1 Declaración

Para entender mejor el trabajo con clases abstractas vamos a crear una clase llamada `FiguraGeometrica` que posteriormente será heredada por una clase llamada `Rectangulo` y otra clase llamada `Circulo`.

```
public abstract class FiguraGeometrica {
	private String identificador;

	public FiguraGeometrica(String identificador) {
		super();
		this.identificador = identificador;
	}

	private String getIdentificador() {
		return identificador;
	}

	public abstract double getArea();

	public boolean mayorQue(FiguraGeometrica fg) {
		return this.getArea() > fg.getArea();
	}
}
```

En la anterior clase, vemos que contamos con un método llamado *getArea()* que es abstracto, es decir, cada subclase o clase hija deberá implementar la lógica propia del cálculo del área para la figura geométrica que sea (rectángulo, círculo, etcétera). No obstante, la anterior clase también cuenta con un atributo *identificador* y un método *mayorQue()* que permite indicar si la figura geométrica tiene mayor área que otra pasada por parámetro, es decir, la clase abstracta tiene comportamiento común para cualquier `FiguraGeometrica`.

2.2 Subclases

¿Cómo podemos crear una clase `Rectangulo` que herede de `FiguraGeometrica` y tenga su propia lógica de cálculo del área? De la siguiente forma:

```
public class Rectangulo extends FiguraGeometrica {
	private double base;
	private double lado;

	public Rectangulo(String identificador, double base, double lado) {
		super(identificador);
		this.base = base;
		this.lado = lado;
	}

	public double getBase() {
		return base;
	}

	public void setBase(double base) {
		this.base = base;
	}

	public double getLado() {
		return lado;
	}

	public void setLado(double lado) {
		this.lado = lado;
	}

	@Override
	public double getArea() {
		return this.getBase() * this.getLado();
	}
}
```

2.3 Instanciación y uso de la clase hija

Vamos a ver un código en el que hagamos uso de nuestra clase `Rectangulo`:

```
Rectangulo r1;
r1 = new Rectangulo("R1", 13.5, 11.7);
System.out.println("Área de r1 = " + r1.getArea() + " cm2");

Rectangulo r2 = new Rectangulo("R2", 8.6, 33.1);
System.out.println("Área de r2 = " + r2.getArea() + " cm2");

if(r1.mayorQue(r2))
	System.out.println("El rectángulo de mayor área es r1");
else
	System.out.println("El rectángulo de mayor área es r2");
```

La salida por pantalla será la siguiente:

```
Área de r1 = 157.95 cm2
Área de r2 = 284.66 cm2
El rectángulo de mayor área es r2
```

2.4 Instanciación y uso como clase padre (abstracta)

Aunque no podemos instanciar una clase abstracta sí que podemos utilizarla para almacenar un tipo de una clase hija. Por ejemplo, podemos almacenar un Rectangulo en una FiguraGeometrica de la siguiente forma:

```
FiguraGeometrica fg = new Rectangulo("R3", 7.4, 13.8);
System.out.println("Área de fg = " + fg.getArea() + " cm2");
```

La salida por pantalla será la siguiente:

```
Área de fg = 102.12 cm2
```

3. Diferencia entre interfaz y clase abstracta

En este momento de nuestro aprendizaje es posible que nos preguntemos, ¿qué diferencia hay entre una interfaz y una clase abstracta? Aunque en un primer momento pueden parecer lo mismo, hay diferencias fundamentales de concepto entre ellas.

Ambas tienen en común que pueden contener varias declaraciones de métodos (pudiendo la clase abstracta incluso darles contenido/lógica) pero luego tienen varias diferencias. Las más destacables se indican a continuación.

- La diferencia principal es que una `interface`, siendo ortodoxos, no debe implementar comportamiento. Únicamente debe contener las firmas de los métodos (su declaración).
- Una clase no puede heredar de dos clases abstractas, pero sí puede heredar de una clase abstracta e implementar una `interface`, o bien implementar dos o más interfaces.
- Las interfaces permiten mucha más flexibilidad para conseguir que dos clases tengan el mismo comportamiento, independientemente de su situación en la jerarquía de clases de Java.
- Las interfaces permiten "publicar" el comportamiento de una clase desvelando un mínimo de información.
- Las interfaces tienen una jerarquía propia, independiente y más flexible que la de las clases. Clases sin relación de herencia pueden implementar la misma interfaz.
- Las interfaces no admiten más que los modificadores de acceso `public` y `package`. Si la interfaz no es `public` no será accesible desde fuera del paquete (tendrá la accesibilidad por defecto, que es `package`). Los métodos declarados en una interfaz son siempre `public` y `abstract`, de modo implícito.

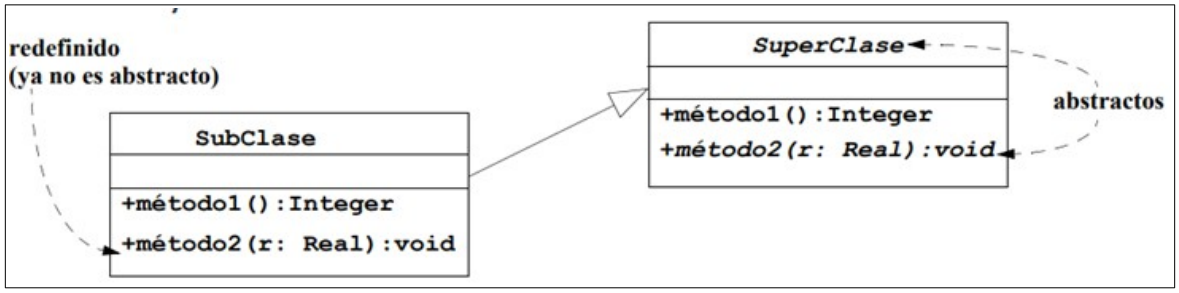
De cara al polimorfismo, las referencias de un tipo `interface` se pueden utilizar de modo similar a las clases abstractas.

Otras cuestiones que tenemos que tener en cuenta al trabajar con interfaces y herencia es lo siguiente:

- Una interfaz puede heredar de otra interfaz.
- Una interfaz NO puede heredar de una clase.
- Una clase (abstracta o no) puede implementar una o más interfaces.
- Una clase (abstracta o no) puede heredar de una única clase (siempre que no esté definida como `final`).
- Una clase puede heredar de otra clase e implementar una o más interfaces a la vez.

4. UML

En UML las clases y métodos abstractos se indican en cursiva.



También, en la definición de la clase o el método, podemos indicar la palabra *abstract*.