

Interfaces

Interfaces

Los métodos abstractos son útiles cuando se quiere que cada clase que pueda heredar de la clase abstracta parezca y funcione igual. Pero para ello estamos obligados a crear una nueva clase en la que poder utilizar los métodos abstractos.

Las interfaces proporcionan un mecanismo para abstraer los métodos a un nivel superior de tal forma que en las interfaces se especifica qué se debe hacer pero no su desarrollo o implementación. Serán las clases que implementen (que implementen=que tengan) estas interfaces las que describan la lógica del comportamiento de los métodos. Es decir, una clase que implemente una interfaz estará obligada a desarrollar todos los métodos que contenga dicha interfaz.

Resumiendo, podemos decir que una interfaz contiene una colección de métodos que se implementan en otro lugar. Los métodos de una interfaz son **public**, **static** o **final**.

La principal diferencia entre interfaz y una clase abstracta es que la interfaz proporciona un mecanismo de encapsulación de los protocolos de los métodos sin forzar al usuario a utilizar la herencia. Esto proporciona al programador cierta flexibilidad a la hora de programar ya que como sabemos Java no soporta la herencia múltiple, sin embargo, una clase de Java puede implementar todas las interfaces que se quieran.

- Ejemplo de declaración de interfaz:

```
package es.pildorasinformaticas.pooAbstractas;

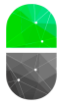
public interface VideoClip {

    // comienza la reproduccion del video
    void play();

    // reproduce el clip en un bucle
    void bucle();

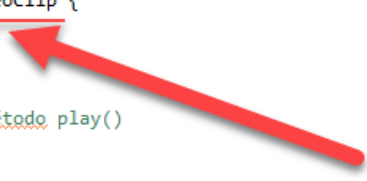
    // detiene la reproducción
    void stop();

}
```



Las clases que quieran utilizar la interfaz VideoClip utilizarán la palabra **implements** y proporcionarán el código necesario para implementar los métodos que se han definido para la interfaz:

```
1 package es.pildorasinformaticas.pooAbstractas;
2
3 public class UsoVideo implements VideoClip {
4
5     @Override
6     public void play() {
7         // Aquí iría el código del método play()
8     }
9
10    @Override
11    public void bucle() {
12        // Aquí iría el código del método bucle()
13    }
14
15    @Override
16    public void stop() {
17        // Aquí iría el código del método stop()
18    }
19 }
20
21
22
23
24
25
26
27
28
29
30
31
```



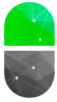
Al utilizar **implements** para la interfaz es como si se hiciese una acción de copiar-y-pegar del código de la interfaz, con lo cual no se hereda nada, solamente se pueden usar los métodos.

La ventaja principal del uso de interfaces es que una interfaz puede ser implementada por cualquier número de clases, permitiendo a cada clase compartir la interfaz de programación sin tener que ser consciente de la implementación que hagan las otras clases que implementen el interface. Dicho de otra forma: la interfaz dice qué métodos deben usar las clases que la implementen, pero son estas clases las que eligen como desarrollar (construir) esos métodos.

Diferencias entre clases abstractas e Interfaces

Una interfaz es simplemente una lista de métodos no implementados y además puede incluir la declaración de constantes. Una clase abstracta puede incluir métodos implementados y no implementados o abstractos, miembros dato constantes y otros no constantes.

Una clase solamente puede derivar (extends) de una clase base, pero puede implementar varios interfaces. Los nombres de los interfaces se colocan separados por una coma después de la palabra reservada **implements**.



El lenguaje Java no fuerza, por tanto, una relación jerárquica, simplemente permite que clases no relacionadas puedan tener algunas características de su comportamiento similares siempre que estas clases implementen las mismas interfaces.

- Ejemplo:

Declaración de interfaz “ParaTrabajadores”

```
package es.pildorasinformaticas.pooAbstractas;

public interface ParaTrabajadores {

    double setBonus(double gratificacion);

    double bonus=200;

}
```

```
class Empleados extends Personas implements Comparable, ParaTrabajadores{

    public Empleados(String nom, Date fechaAlta, double sueldo) {
        super(nom);
        // TODO Auto-generated constructor stub

        this.fechaAlta=fechaAlta;

        this.sueldo=sueldo;
    }

    public double getSuelo(){

        return sueldo;
    }

    @Override
    public String getDescripcion() {
        // TODO Auto-generated method stub
        return "El empleado " + this.getNombre() + " tiene un sueldo"
            + " de " + sueldo + " € " + " y entró a trabajar en " + fechaAlta;
    }

    private double sueldo;

    private Date fechaAlta;

    @Override
    public int compareTo(Object o) {
        // TODO Auto-generated method stub

        Empleados otroEmpleado=(Empleados)o;

        if(this.sueldo<otroEmpleado.sueldo) return -1;

        if(this.sueldo>otroEmpleado.sueldo) return 1;

        return 0;
    }

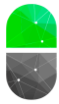
    @Override
    public double setBonus(double gratificacion) {
        // TODO Auto-generated method stub
        return ParaTrabajadores.bonus + gratificacion;
    }

}
```

Método que implementa
la interfaz Comparable de
la API de Java

Método que implementa la
interfaz ParaTrabajadores
creada por el programador

Como se observa en el ejemplo anterior, la clase Empleados hereda de una clase base llamada “Personas” y a su vez implementa dos interfaces: “Comparable” y “ParaTrabajadores”. La



interfaz Comparable viene en la biblioteca de clases de Java (API) mientras que la interfaz "ParaTrabajadores" la ha creado el propio programador. Varias cosas que anotar viendo este ejemplo:

- Una clase puede implementar más de una interfaz (en caso de implementar más de una interfaz estas se separarán con comas).
- La clase se ve obligada a desarrollar todos los métodos de las interfaces implementadas. En el ejemplo no son demasiados afortunadamente ya que tanto la interfaz Comparable como la interfaz ParaTrabajadores tienen solo un método cada una (CompareTo y setBonus). En el caso de que cada una hubiera tenido cinco métodos, la clase tendría que haber desarrollado diez métodos.
- Java no admite la herencia múltiple pero si es posible heredar de una clase base o padre e implementar varias interfaces.

La pregunta recurrente que suele hacerse la persona que está aprendiendo el lenguaje Java es ¿Y por qué voy a utilizar interfaces habiendo clases abstractas cuya función es la misma prácticamente? La respuesta la da una palabra clave: **herencia**. Las interfaces no nos obligan a heredar de ellas. Las clases abstractas sí.