educación IT
CODE YOUR FUTURE

Bootcamp Java Developer

Fase 1 - Java Analyst

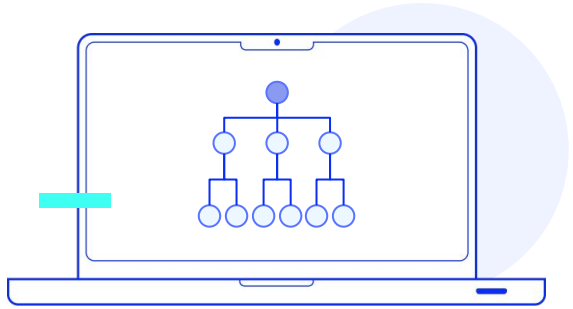
Módulo 8

Interfaces

Introducción a interfaces

Con la teoría y ejemplos que hemos realizado con la abstracción, específicamente con los métodos abstractos, vimos que al heredar de una clase con estas características, se obligaba a implementar estos métodos de la clase padre y así las clases hijas se convertirían en *clases concretas* con definición (cuerpo) en los métodos.

El concepto de **abstracción** se puede ampliar mucho más ya que se puede pensar en **clases totalmente abstractas** a la hora de analizar un problema y definir una serie de métodos genéricos que algunas clases podrían implementar.



Interface

Cuando hablamos de clases totalmente abstractas podemos recurrir a una variante de las clases llamadas ***interface***.

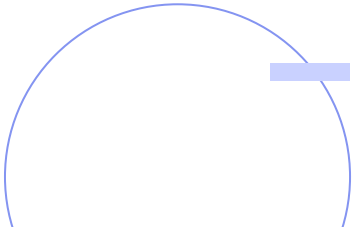
Recordemos que Java solo permite la herencia simple (heredar de una sola clase) pero **con las interfaces podemos simular la *herencia múltiple***.

```
public interface Archivo {

    String XLS = "XLS";
    String PDF = "PDF";
    String WORD = "WORD";
    String TXT = "TXT";

    boolean imprimir(String impresora);

    boolean guardar(String tipoArchivo);
}
```



- **Métodos:** todos los métodos deben ser abstractos y de acceso público.
- **Atributos:** todos los atributos deben ser miembros de clase (estáticos), constantes (finales) y de acceso público.

Ya que estos son requisitos para que no nos de errores en tiempo de compilación, se puede omitir el modificador de acceso **`public`**, la palabra reservada **`static`** y la palabra reservada **`final`** ya que **están implícitos en la `interface`.**

Implementación

Una de las grandes ventajas que ofrecen las interfaces, como vimos anteriormente, es que podemos simular la herencia múltiple. **De ser necesario, podemos heredar de una clase abstracta y de varias interfaces a la vez**. Se hace con la palabra reservada **`implements`** seguida del **nombre de las interfaces separadas por comas**, si llegaran a ser más de una.

> Veamos un ejemplo en la siguiente pantalla.

```
public abstract class Auto implements MantenimientoMecanico, Archivo{
```

Herencia en interfaces

Una `interface` puede heredar de otra. De esta manera, se modulariza más el software. *¿Por qué se realiza de esta manera y no se hace una sola interfaz con todos los métodos?*

Ejemplo

En el ejemplo del siguiente slide, se crea la interfaz llamada **`Mantenimiento Periódico`** que posee solo el método **`lavar`** y que se puede implementar en otras clases. También, una interfaz con métodos para **`Mantenimiento Mecánico`**.

Analizamos y llegamos a esta conclusión:

1. Cualquier objeto puede tener un **`Mantenimiento Periódico`** (Autos, Personas, Objetos) pero no todos los objetos van a tener **`Mantenimiento Mecánico`**. Esto justificaría la interfaz cuyo nombre es **`MantenimientoPeriodico`**.
2. Todos los objetos que pasan por un **`Mantenimiento Mecánico`** deben lavarse, pero no se crearía un método lavar con las mismas características que en la interfaz llamada **`MantenimientoPeriodico`**, sino que se reutilizaría el método heredando de esa interfaz.

```java
public interface MantenimientoPeriodico {

    // constantes
    String LAVADO_MANUAL = "Lavado Manual";
    String LAVADO_ECOLOGICO = "Lavado Ecológico";
    String LAVADO_PRESION = "Lavado a Presión";
    String LAVADO_TUNEL = "Lavado Túnel";

    // metodos
    void lavar(Date fecha, String tipo);
}
```

```java
public interface MantenimientoMecanico extends MantenimientoPeriodico {

    boolean reparar(Date fecha, String autoParte, String mecanico);

    boolean cambioPieza(Date fecha, String autoParte);

    String cambioAceite(Date fecha, String autoParte, String marca, String tipo, Float cantidad);

}
```

Interfaces en Java 8

La versión del **JDK 8 añade numerosas novedades importantes al lenguaje**. Entre ellas se destaca la **implementación de características del *Paradigma Funcional***, que veremos más adelante. Por ahora, nos enfocaremos en lo que trae en materia de interfaces.

Uno de los problemas que presentan las interfaces que son totalmente abstractas es que, a la hora de modificarlas y crear métodos nuevos, el JDK exige implementar esos métodos en todas las clases que heredaron de la interfaz.

Con Java 8 es posible, además de crear métodos abstractos, **crear métodos concretos, pero deben ser obligatoriamente *Miembros de Clase* (`static`) o *por Defecto* (`default`)**. De esta forma, no es necesario modificar las clases que hayan implementado la interfaz antes de la introducción de los nuevos métodos ofreciendo además una implementación por defecto de estos nuevos métodos, que será heredada por dichas clases, permitiendo su uso implícitamente de estos nuevos métodos.

```java
public interface Archivo {

    String XLS = "XLS";
    String PDF = "PDF";
    String WORD = "WORD";
    String TXT = "TXT";

    boolean imprimir(String impresora);

    boolean guardar(String tipoArchivo);

    default String cargar(String ruta) {
        return "Se ha cargado el archivo desde la ruta " + ruta;
    }

    static String[] listarImpresoras() {
        String[] lista = { "HP MODEL01", "Canon MODELP11" };
        return lista;
    }
}
```

```
// metodos por defecto

autoFamiliar1.cargar("C:/Archivos/auto.pdf");
autoFamiliar2.cargar("C:/Archivos/auto.pdf");
transporteCarga.cargar("C:/Archivos/auto.pdf");
transportePasajeros.cargar("C:/Archivos/auto.pdf");

// metodos estaticos

System.out.println(Arrays.toString(Archivo.listarImpresoras()));
```

De ser necesario un **cambio en el algoritmo por clase**, podemos seguir apoyándonos en la **sobreescritura de miembros.**

¡Sigamos trabajando!