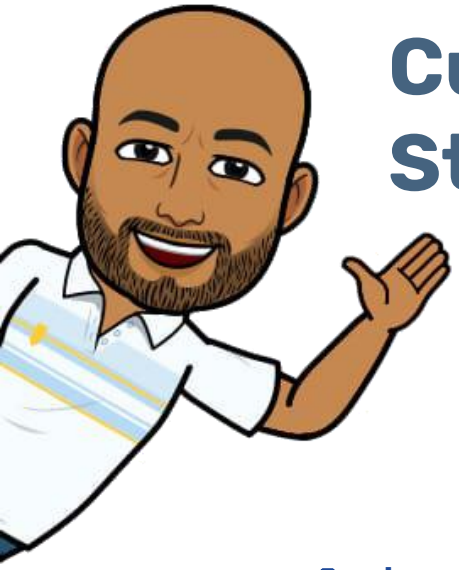


Curso de Java Standard

Ing. Octavio Robleto

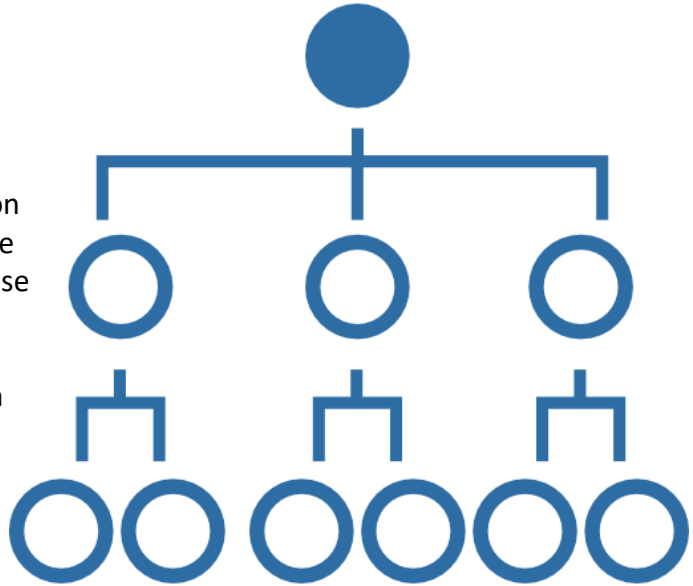
octavio.robleto@gmail.com

https://octaviorobleto.com

Introducción

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 que la clase padre tenga y así las clases hijas se convertirían en clases concretas con definición (cuerpo) en los métodos.

El concepto de Abstracción se puede ampliar mucho mas ya que podemos 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 publico.

Atributos: todos los atributos deben ser miembros de clase (estáticos) , constantes (finales) y de acceso publico.

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 esta implícito en la interface.

Implementación

Una de las grandes ventajas que ofrecen las interfaces como hablamos anteriormente es que podemos simular la herencia múltiple, por lo que de ser necesario podemos heredar de una clase abstracta y de varias interfaces a la vez.

Podemos lograr esto a través de la palabra reservada **implements** seguida del nombre de las interfaces separadas por comas si llega a ser mas de una.

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

Y al igual que pasa cuando heredas de una clase abstracta las clases que implementes dichas interfaces se verán obligadas a implementar los métodos que posea dicha interface.

Herencia en Interfaces

Además una interface puede heredar de otra y así modularizamos mas nuestro software.

¿Porque haríamos esto y no hacemos una sola interfaz con todos los métodos?

En este ejemplo creamos una interfaz **Mantenimiento Periódico** que posee solo el método **lavar** y que podemos implementar en otras clases y una interfaz con métodos para **Mantenimiento Mecánico** en nuestro caso analizamos y llegamos a la siguiente conclusión:

1. Cualquier objeto puede tener un **Mantenimiento Periódico** (Autos, Personas, Objetos) pero no todos los objetos van a tener **Mantenimiento Mecánico** , lo que justificaría la interfaz **MantenimientoPeriodico**.
2. Todos los objetos que pasan por un **Mantenimiento Mecánico** deben lavarse, pero no crearíamos un método lavar con las mismas características que está en la interfaz **MantenimientoPeriodico,** si no, que reutilizaríamos el método heredando de dicha 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 las cuales destaca la implementación de características del Paradigma Funcional, dicho paradigma lo veremos mas adelante, por ahora veremos que nos trae en este tema 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 te exige implementar dichos métodos en todas las clases que heredaron de la interfaz.

Con Java 8 podemos además de crear métodos abstractos podemos crear métodos concretos pero deben ser obligatoriamente Miembros de Clase (**static**) o por Defecto (**default**), de esta forma no tenemos la necesidad de 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;
    }
}
```

```java
// 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.