# POO Tema 6. Interfaces

Martín González Dios

27 de noviembre de 2024

Los interfaces son uno de los componentes clave en el diseño de los programas, ya que con ellos se establecen de forma muy precisa los requisitos que deben cumplir las clases del programa.

- **Se definen los tipos de datos** que se deben crear en el programa, de modo que pueden existir otros tipos de datos, pero al menos deben estar definidos los indicados en las interfaces.
- **Se definen exactamente los métodos** que deben tener las clases del programa, indicando exactamente sus firmas.
- Cada clase puede tener más métodos públicos y privados, pero los **métodos de interés** son los indicados en los interfaces.

El objetivo de un interfaz es establecer los métodos que una clase debe implementar y que serán los **métodos visibles y accesibles** por las otras clases del programa.

El uso de interfaces facilita enormemente **el desarrollo y el mantenimiento de los programas**, puesto que con ellos se dividen las responsabilidades entre los programadores y permiten una implementación independiente de las clases.

- Un cambio en una clase en la que se implementan los métodos indicados en un interfaz **no afectará a las otras clases** que hace uso de dicho interfaz.
- Un cambio en el interfaz afectará de forma significativa tanto a las clases que implementan el interfaz como a las clases que lo usan. (las interfaces no se deberían modificar).

# 1. Interfaz en Java

- No se puede crear objetos de interfaces, con lo cual **no tienen constructores**.
- public class $< nombre\_clase >$ implements $< nombre\_interfaz >$
- Una vez la interfaz haya sido implementada, **se comportará como una clase**, pudiendo invocar a los métodos abstractos, con la implementación de clase, los métodos estáticos y por defecto.

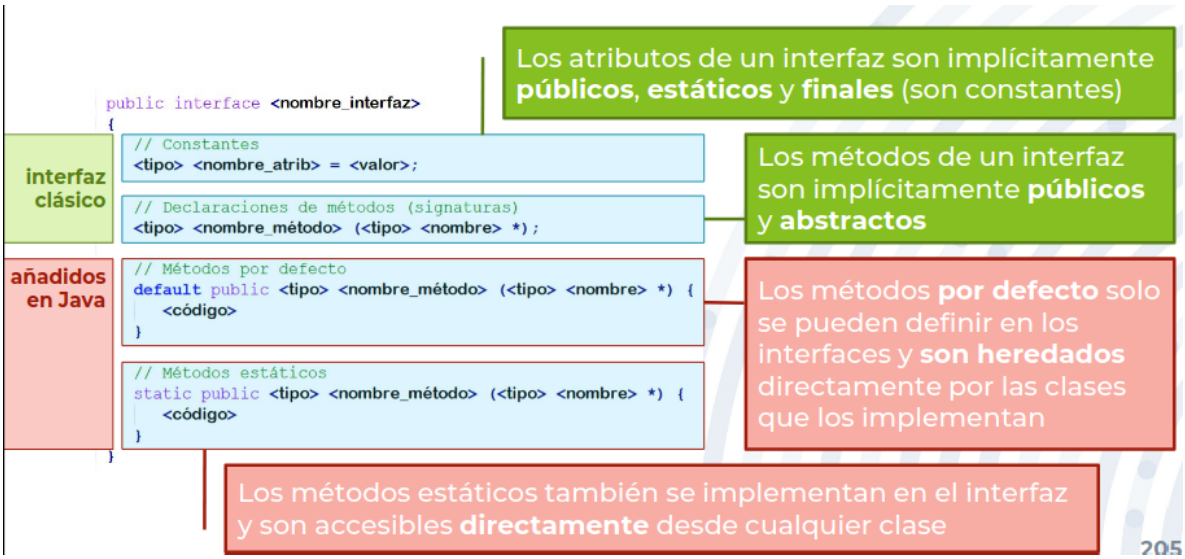


Figura 1: Ejemplo sobre interfaces en Java

Si una clase implementa una interfaz, entonces esa clase tendrá los mismos métodos que la interfaz, **del mismo modo** que una clase derivada tiene los mismos métodos que una clase base. La clase que implementa la interfaz se puede entender que **es como una clase derivada de la interfaz**, puesto que una interfaz es equivalente a una clase abstracta con las siguientes características: métodos abstractos, atributos constantes y constructores no implementados.

| Clases abstractas | Interfaces |
|---|---|
| Se definen cuando las clases derivadas tienen algunos métodos comunes | Se definen cuando las clases que los implementan tienen diferentes implementaciones para diferentes objetos |
| La herencia múltiple no es posible | Se puede entender que la herencia múltiple es posible, aunque solo puede haber una clase base |
| Pueden tener métodos abstractos y métodos concretos | Solamente tienen métodos abstractos, además de métodos estáticos y métodos por defecto |
| Los métodos abstractos pueden ser públicos y protegidos | Los métodos abstractos deben de ser públicos |
| Tienen constructores | No tienen constructores |
| Pueden tener cualquier tipo de atributo | Los únicos atributos que puede tener son de tipo estático y final (son constantes) |

Figura 2: Comparación entre clases abstractas e interfaces

## 2. Métodos por defecto

El principal problema de las interfaces es la incapacidad de las clases que los implementan para **adaptarse a cambios**. Una solución parcial son los métodos por defecto.

Son métodos que se **deben declarar en implementar en la interfaz**, de modo que se **heredarán** por las clases que implementan dicha interfaz.

La implementación debe operar únicamente con los **argumentos del método por defecto**, ya que en el interfaz no existen atributos que no sean constantes. Los métodos por defecto **pueden ser sobrescritos** en las clases que implementan el interfaz o en las clases derivadas de ellas. En la sobreescritura se puede hacer uso de super (nombre_interfaz.super.método)

Se debe de valorar la **conveniencia** de definir métodos por defecto, sobre todo si necesitan **argumentos asociados a atributos** de las clases que implementan los interfaces que contienen dichos métodos.

## 3. Métodos estáticos

Métodos que para **invocarlos no hace falta** crear ningún objeto de la clase en la que se han definido. Se **cargan automáticamente** en memoria al iniciar el programa.

En un método estático **únicamente se pueden invocar métodos** de la clase o del interfaz que sean estáticos, puesto que tienen que estar disponibles desde el arranque del programa. (**No son heredados** por las clases e interfaces derivados de ellos)

| Métodos por defecto | Métodos estáticos |
|---|---|
| Solamente pueden estar definidos en los interfaces | Se pueden definir en interfaces y en clases |
| Pueden invocar cualquier tipo de método, incluyendo métodos por defecto, estáticos y abstractos | Solamente pueden invocar métodos que, a su vez, sean métodos estáticos |
| Son heredados por las clases y por los interfaces derivados del interfaz que los contiene | No pueden ser heredados, siendo métodos que son específicos de las clases o de los interfaces en los que están implementados |
| No están disponibles en el arranque del programa, es necesario crear un objeto para invocarlos | Están disponibles al arrancar el programa |

Figura 3: Compración entre métodos estáticos y por defecto

# 4. Herencia e interfaces

Si una **clase abstracta cumple las mismas condiciones** que una interfaz "clásica", es decir, solamente tiene métodos abstractos y no tiene ningún atributo que no sea constante, entonces se podría pensar que una clase derivada tiene varias clase base, o lo que es lo mismo, que **existe herencia múltiple**.

Una clase debe de **implementar todos los métodos abstractos** de todas las interfaces que implementa y **hereda todos los métodos por defecto** de todos esas interfaces.

- Si varias interfaces tienen en **común algún método abstracto**, la implementación de ese método en la clase será válida para todos los interfaces.
- Si varias interfaces tienen en **común algún método estático**, **no existirá ningún conflicto** en la clase, ya que **ese tipo de métodos no serán heredan**.
- Si varias interfaces tienen en **común algún método por defecto**, existirá una **colisión entre dichos métodos** en relación a cuál de ellos será el que heredará la clase.

Para resolver el conflicto de los métodos por defecto, la clase que implementa las interfaces **deberá sobrescribir los métodos por defecto** que son comunes a dichas interfaces.

Se pueden crear **jerarquías de interfaces** siguiendo las mismas reglas de herencia que en el caso de las jerarquías de clases. (**Existe herencia múltiple entre interfaces**). Puesto que en una interfaz tiene todos sus métodos implícitamente públicos, las interfaces derivadas **heredarán todos los métodos** de las interfaces base, **excepto los métodos estáticos**. Existe **sobreescritura de los métodos por defecto**, de manera que las interfaces derivadas pueden usar super para invocar la ejecución de los métodos por defecto de las interfaces base. (**Una clase no puede extender un interfaz ni viceversa**)

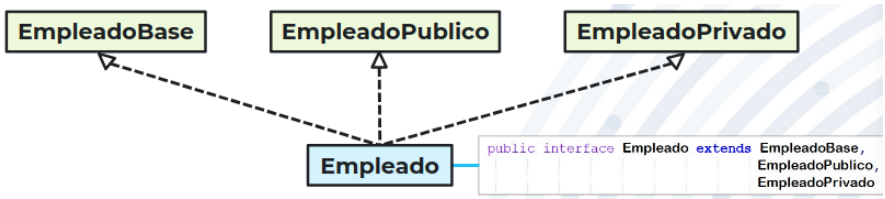


Figura 4: Herencia de interfaces

Buenas prácticas:

Se deben definir **interfaces** cuando se quieren **establecer de forma clara e inequívoca** los métodos de que se van a usar en las restantes clases del programa.

Se deben usar **clases abstractas** cuando se quiere hacer **énfasis en la reutilización de código**, incluyendo constructores que se invocan desde las clases base.