Programación Orientada a Objetos

By Álex.

# Conceptos importantes.

**COMPOSICIÓN**

La composición consiste en crear una clase nueva agrupando objetos de clases que ya existen.

La composición crea una relación ‘**tiene**’ o ‘**está compuesto por**’.

**Una composición agrupa uno o más objetos para construir una clase**, de manera que las instancias de esta nueva clase contienen uno o más objetos de otras clases. Normalmente los objetos contenidos se declaran con acceso private y se inicializan en el constructor de la clase.

Para hacer una composición utilizando objetos de una clase diferente de String, lo primero es definir una nueva clase.

En una relación de composición, hay atributos de la clase contenedora que son objetos que pertenecen a la clase contenida. Un objeto de la clase contenedora puede acceder a los métodos públicos de las clases contenidas.

public class VehiculoAlquilado {

private Cliente cliente;

private Vehiculo vehiculo;

…

**INTERFAZ**

Una **interface** es una construcción similar a una clase abstracta en Java, pero con las siguientes diferencias:

a) En el encabezado se usa la palabra clave interface en lugar de class o abstract class. Por ejemplo: public interface NombreDelInterface {…}

b) Todo método es abstracto y público sin necesidad de declararlo, es decir, no hace falta poner abstract public porque por defecto todos los métodos son abstract public. Por lo tanto, una interface en Java no implementa ninguno de los métodos que declara: **ninguno de sus métodos tiene cuerpo.**

c) **Las interfaces no tienen ningún constructor**.

d) Un interfaz solo admite campos de tipo “public static final”, es decir, campos de clase, públicos y constantes. No hace falta incluir las palabras public static final porque todos los campos serán tratados como si llevaran estas palabras. Recordemos que static equivalía a “de clase” y final a “constante”. Las interfaces pueden ser un lugar interesante para declarar constantes que van a ser usadas por diferentes clases en nuestros programas.

e) Una clase puede derivar de un interface de la misma manera en que puede derivar de otra clase. No obstante, se dice que el interface se implementa (implements), no se extiende (extends) por sus subclases. Por tanto, para declarar la herencia de una interface se usa la palabra clave implements en lugar de extends.

**HERENCIA**

En Java se utiliza la palabra reservada extends para indicar herencia.

Consiste en derivar o extender una clase nueva a partir de otra ya existente de forma que **la clase nueva hereda todos los atributos y métodos de la clase ya existente.**

A la clase ya existente se la denomina **superclase**, **clase base** o **clase padre**.

A la nueva clase se la denomina **subclase**, **clase derivada** o **clase hija**.

La clase(A) se debe relacionar mediante herencia con la clase (B) si “A ES-UN B”. Si la frase suena bien, entonces la situación de herencia es la más probable para ese caso.

El **constructor** de una clase derivada **debe encargarse de construir los atributos que estén definidos en la clase base además de sus propios atributos**. Dentro del constructor de la clase derivada, para llamar al constructor de la clase base se debe utilizar el método reservado super() pasándole como argumento los parámetros que necesite.

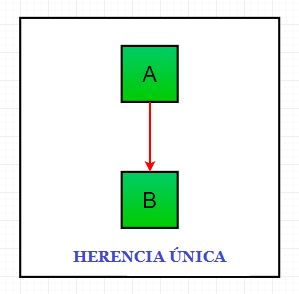
En una jerarquía, es posible que tanto las superclases como las subclases tengan sus propios constructores. Esto plantea una pregunta importante: ¿**qué constructor es responsable de construir un objeto de la subclase, el de la superclase, el de la subclase o ambos**? La respuesta es esta: **el constructor para la superclase construye la porción de la superclase del objeto, y el constructor para la subclase construye la parte de la subclase.**

Esto tiene sentido porque la superclase no tiene conocimiento ni acceso a ningún elemento en una subclase. Por lo tanto, su construcción debe estar separada. En la práctica, la mayoría de las clases tendrán constructores explícitos (no predeterminados).

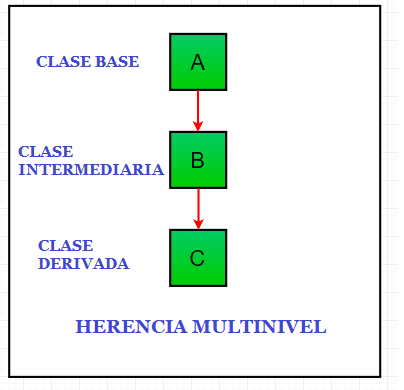
Puede ocurrir que alguno de los métodos que existen en la superclase no nos sirvan en la subclase (tal y como están programados) y necesitemos adecuarlos a las características de la subclase. Esto puede hacerse mediante la sobreescritura de métodos: **Un método está sobrescrito o reimplementado cuando se programa de nuevo en la clase derivada**. Por ejemplo el método mostrarPersona() de la clase Persona lo necesitaríamos sobrescribir en las clases Alumno y Profesor para mostrar también los nuevos atributos.

Tipos de Herencia

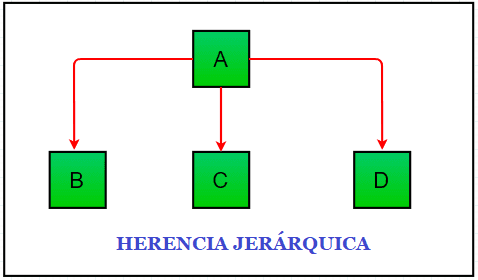
* **Herencia única**: en la herencia única, las subclases heredan las características de solo una superclase. En la imagen a continuación, la clase A sirve como clase base para la clase derivada B.

[](https://www.javadesdecero.es/wp-content/uploads/2018/02/Herencia-%C3%BAnica-Java.png)

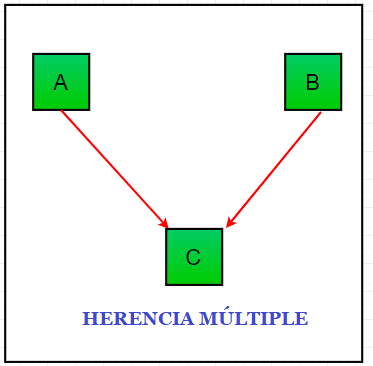
* **Herencia Multinivel**: en la herencia multinivel, una clase derivada heredará una clase base y, además, la clase derivada también actuará como la clase base de otra clase. En la imagen inferior, la clase A sirve como clase base para la clase derivada B, que a su vez sirve como clase base para la clase derivada C. En Java, una clase no puede acceder directamente a los miembros de los “abuelos”.

[](https://www.javadesdecero.es/wp-content/uploads/2018/02/Herencia-Multinivel-Java.png)

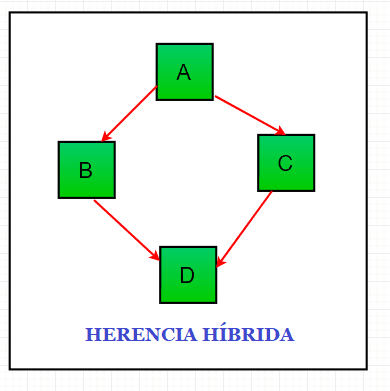
* **Herencia Jerárquica**: en la herencia jerárquica, una clase sirve como una superclase (clase base) para más de una subclase. En la imagen inferior, la clase A sirve como clase base para la clase derivada B, C y D.

[](https://www.javadesdecero.es/wp-content/uploads/2018/02/Herencia-Jer%C3%A1rquica-Java.png)

* **Herencia Múltiple (a través de interfaces)**: en Herencia múltiple, una clase puede tener más de una superclase y heredar características de todas las clases principales. Tenga en cuenta que Java no admite herencia múltiple con clases. En Java, podemos lograr herencia múltiple **solo a través de Interfaces**. En la imagen a continuación, la Clase C se deriva de la interfaz A y B.

[](https://www.javadesdecero.es/wp-content/uploads/2018/02/Herencia-M%C3%BAltiple-Java.png)

* **Herencia Híbrida (a través de Interfaces)**: Es una mezcla de dos o más de los tipos de herencia anteriores. Como Java no admite herencia múltiple con clases, la herencia híbrida tampoco es posible con clases. En Java, podemos lograr **herencia híbrida solo a través de Interfaces**.

[](https://www.javadesdecero.es/wp-content/uploads/2018/02/Herencia-H%C3%ADbrida-Java.png)

POLIFORMISMO

**CLASE ABSTRACTA**

Una clase abstracta **es una clase que declara la existencia de algunos métodos, pero no su implementación** (es decir, contiene la cabecera del método, pero no su código**). Los métodos sin implementar son métodos abstractos**.

* + abstract void método();

Una clase abstracta **no se puede instanciar**, pero sí heredar. Las subclases tendrán que implementar obligatoriamente el código de los métodos abstractos (a no ser que también se declaren como abstractas).

Para declarar que una clase hereda de una clase abstracta, se utiliza extends ….

* + class claseHeredada extends claseAbstracta {}

Instance Of

¿Constructor?

Sí, una clase abstracta **siempre tiene un constructor**. Si no define su propio constructor, el compilador le dará un constructor predeterminado a la clase Abstract.