**Definicion**

La herencia permite que se puedan definir nuevas clases basadas de unas ya existentes a fin de reutilizar el código, generando así una jerarquía de clases dentro de una aplicación. Si una clase deriva de otra, esta hereda sus atributos y métodos y puede añadir nuevos atributos, métodos o redefinir los heredados.

La herencia, junto con la encapsulación y el polimorfismo, es una de las tres características principales de la programación orientada a objetos. La herencia permite crear clases que reutilizan, extienden y modifican el comportamiento definido en otras clases. La clase cuyos miembros se heredan se denomina *clase base* y la clase que hereda esos miembros se denomina *clase derivada*.

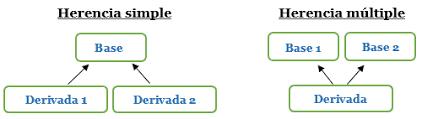
**Herencia Simple**

La herencia es simple cuando la clase derivada que estamos considerando sólo tiene una clase base. Una clase solo puede heredar de una clase base y de ninguna otra.

**Herencia Múltiple**

Cuando se deriva una clase usando varias clases bases la misma se denomina herencia múltiple. Usando la herencia múltiple una clase derivada recibe los atributos de dos o más clases bases. Sí se utiliza la herencia múltiple para derivar una clase el constructor de la clase derivada deberá llamar a los constructores de cada una de las clases bases.

Una clase puede heredar las características de varias clases base, es decir, puede tener varios padres.



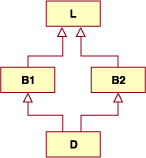
**Clases Bases Virtuales**

Las clases base virtuales son una técnica de programación orientada a objetos que se utiliza en la herencia virtual para verificar múltiples instancias de una clase en una jerarquía de herencias

Las clases base virtuales se utilizan para resolver algunos problemas de la herencia múltiple, como el "problema del diamante". Este problema se produce cuando no está claro qué miembros de las clases padre usar.

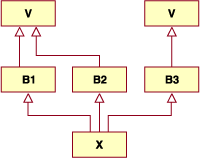
Supongamos que tiene dos clases derivadas B y C que tienen una clase base común A, y también tiene otra clase D que hereda de B y C. Puede declarar la clase base A como *virtual* para asegurarse de que B y C comparten el mismo subobjeto de A.

En el ejemplo siguiente, un objeto de la clase D tiene dos subobjetos distintos de la clase L, uno a través de la clase B1 y otro a través de la clase B2. Puede utilizar la palabra clave virtual delante de los especificadores de clase base en las *listas base* de clases B1 y B2 para indicar que sólo existe un subobjeto de tipo L, compartido por la clase B1 y la clase B2.



El uso de la palabra clave virtual en este ejemplo garantiza que un objeto de clase D hereda sólo un subobjeto de clase L.

Una clase derivada puede tener clases base virtuales y no virtuales. Por ejemplo:



**Visibilidad de la Herencia**

La visibilidad de la herencia en programación es la característica que determina qué atributos y métodos de una clase son accesibles desde otras clases.

La visibilidad de los miembros de una clase depende de su accesibilidad. Por ejemplo, en Java, los miembros de una clase pueden ser públicos, protegidos o privados:

* **Públicos**: Se puede acceder a ellos desde cualquier lugar.
* **Protegidos**: Solo se puede acceder a ellos desde la propia clase o desde una clase que herede de ella.
* **Privados**: Solo se puede acceder a ellos desde la propia clase.

**Clases Abstractas**

Clases definidas en la jerarquía que simplemente recogen las características comunes de otra serie de clases (sus descendientes), pero que no se van a o no se deben utilizar para crear ejemplares.

Una clase que declara la existencia de métodos, pero no la implementación de dichos métodos, estos métodos no hacen nada, solo son declarados, se considera una clase abstracta.

**Metodos Virtuales**

Los métodos virtuales son aquellos métodos definidos en una clase base **que pueden ser sobrescritos por las clases derivadas**. Así de sencillo.

Al marcar un método como virtual en la clase base, se indica que este método puede ser reemplazado por una implementación diferente en las clases derivadas *(proporcionando su propia implementación)*

**Constructores y Destructores en la Herencia**

Construcción objeto clase derivada:

• Primero se construye la parte heredada de la clase(s) base: Se ejecutan constructores de las clases base.

• Por último se ejecuta el código del constructor de la clase derivada.

Destrucción objeto clase derivada:

• el proceso es a la inversa que en la construcción.

• Se ejecuta primero el destructor de la clase derivada y a continuación los de las clases base.

Teniendo varios niveles de herencia, y asumiendo la siguiente jerarquía: *A* es la clase base, *B* hereda de *A*, *C* hereda de *B*. El orden de construcción de un objeto es A-B-C. El orden de destrucción es el inverso: C-B-A. El primer sub-objeto en construirse es de la clase base que está más arriba *A*, a partir de este se construye el de la siguiente en jerarquía *B*, y finalmente el de la última clase derivada *C*.