HERENCIA

La herencia es la capacidad que tienen los lenguajes orientados a objetos para extender clases. Es decir, mediante esta propiedad, los atributos y métodos de una clase pueden ser reutilizados en otras clases, las que vendrían a llamarse clases hijas.

Ahora bien, esto se asemeja al término herencia en los seres humanos, ya que en los seres humanos un hijo hereda una serie de rasgos físicos, mientras que un padre, por el simple hecho de ser padre no hereda de su hijo ningún rasgo físico. De ahí que el término de herencia en java, se utiliza para explicar que una clase hijo, al heredar sus atributos y métodos de una clase padre, puede usar estos miembros heredados como si fuesen suyo.

La herencia nos sirve mucho cuando queramos hacer un conjunto de clases que tengan atributos o métodos en común, ya que en ese caso nosotros podremos realizar una clase genérica que contenga todos esos atributos y métodos que tengan estas clases en común, para luego usar la herencia con el objetivo que este conjunto de clases se conviertan en clases hijas de la clase genérica, la cual sería la clase padre. Esto nos ahorraría mucho tiempo y espacio, además trabajaríamos de una forma más eficiente y ordenada.

La notación de una herencia se hace usando la palabra reservada "extends", de la siguiente forma:

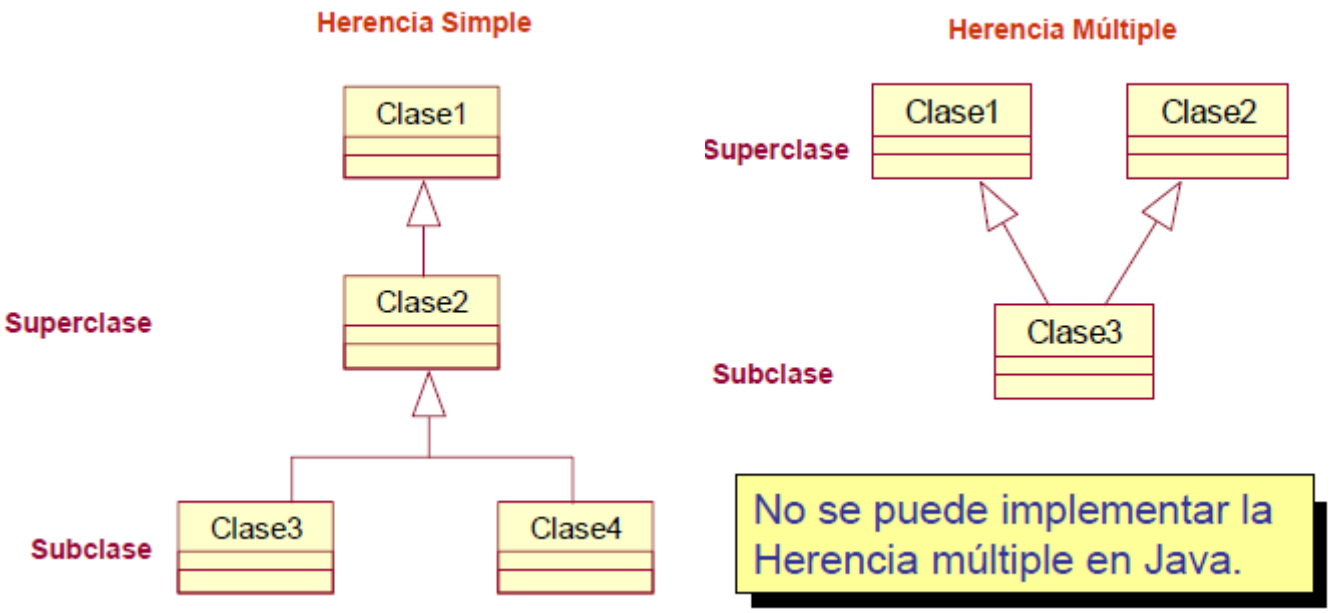
class nombre_clase_hija **extends** nombre_clase_padre{ //atributos y métodos }

Recordando el tema de los modificadores de acceso, durante las primeras clases, la herencia tiene un límite con respecto a estos modificadores. Pues esta no se cumple para aquellos atributos y/o métodos que tengan el modificador "private" en la clase padre, solo se cumplirá para el resto de atributos y métodos que tengan otro tipo de modificador, como es el caso del public, default y protected, en el caso del default, se cumplirá siempre y cuando ambas clases, tanto padre como hija estén en el mismo paquete, si están fuera de este paquete el default se comportará como un private.

Por otro lado, formalizando el tipo de clases en una herencia, la clase hija sería llamada, subclase mientras que la clase padre se llamará superclase. Usando estos términos, diremos que una "cadena de herencia" se refiere a que una clase puede tener una subclase y ésta última, puede tener otra subclase, y así sucesivamente.

A parte de esto, lo que averigüé es que existen dos tipos de herencias, múltiple y simple, en nuestro caso, ya que nuestro curso se basa en JAVA, solo se podrá usar la herencia simple, ya que esta es la única que se puede usar en JAVA. Dicho de otro modo, lo único que podemos hacer es que una superclase

tenga varias subclases pero no podemos hacer lo inverso, es decir, que una subclase tenga varias superclases (clases en el mismo nivel jerárquico). Para ejemplos didácticos, veremos el siguiente gráfico:



Algo más por añadir sería explicar la palabra reservada "super", esto se usa en subclase ya que ésta llama al constructor de la superclase de esta subclase anteriormente mencionada. Este enunciado deberá ir en la primera línea del cuerpo del constructor de la subclase.

Veremos un ejemplo para explicarlo mejor:

public ConstructorSubclase (Integer a, String b){ super(1, 2); }

En este ejemplo el enunciado “super(1, 2)”, está llamando al constructor de la superclase cuyos argumentos son Integer, notar que los argumentos de la clase “super” no necesariamente tienen que ser iguales a los argumentos del constructor de la subclase.

Por último, ya que hemos hablado de una palabra reservada, hablaremos de otra palabra reservada llamada “final”, la cual, si es que se aplica a una clase, indica que de esta clase no se puede heredar nada. La notación de esta palabra reservada sería:

public **final** class ClaseNoGeneraHerencia { //métodos y atributos }

POLIMORFISMO

Polimorfismo es una propiedad de “Java” que está conformada por otras 2 propiedades, la sobrecarga y la redefinición.

Pues bien, la sobrecarga es la propiedad de poder definir un método con el mismo nombre pero con diferentes argumentos, ya sea que se diferencien en la cantidad de argumentos o en el tipo de argumentos, sin que “Java” los confunda o al momento de compilar se ejecute como error.

Por ejemplo:

```
public Class ClaseA {       //definiremos los atributos con el modificador public,
                            //para una mejor comprensión y mayor rapidez.

public Integer a;

public String b;

//Métodos sobrecargados:

public void print (Integer a) {      this.a  =  a;
System.out.println(“Ingresó al método print:  ”+ toString(a) ;
                                }

public void print (String b) { System.out.println(“Ingresó al método print:  ”+ b );
        }

}
```

En este ejemplo ese método print, es un método que cumple la propiedad de sobrecarga ya que se puede definir cuantas veces quieras varios métodos con el mismo nombre, siempre y cuando se diferencien en sus argumentos, ya sea en la cantidad o en el tipo. En el ejemplo los argumentos del método print se diferencian en el tipo de argumento en uno es Integer y en el otro es String.

Luego de haber creado el método “main” y dentro de este haber declarado e inicializado una variable de tipo ClaseA mediante la instancia de un constructor, podremos invocar al método “print”, el cual se ejecutará de acuerdo a lo que nosotros coloquemos en el argumento, es decir, si es que nosotros colocamos dentro de los paréntesis un número entero, entonces Java, buscará primero si es que existe el método “print”, luego de esto se fijará si es que el argumento que ha ingresado (entero) ha sido definido dentro del método “print”, si es así ahí recién ejecuta todo el código que existe dentro de este método; los demás métodos “print”, Java los ignora, pues para el caso de haber ingresado un entero, a Java solo le interesa la definición del método con argumento “Integer”.

Si nosotros colocamos otro argumento en el “print”, que no haya sido declarado, por ejemplo colocamos uno de tipo “Double”, Java dirá que existe un

error, puesto que solo han sido definidos los métodos “print” con argumento “Integer” y otro, con el mismo nombre, pero con argumento “String”.

Por otro lado, hablando de la otra propiedad llamada redefinición, esta propiedad sirve más que nada para aquellas subclases que utilizan el método, con el mismo nombre y la misma cantidad y tipo de argumentos o parámetros, de su superclase. En ese caso se está redefiniendo el método.

Las variables no se pueden redefinir, a menos que hayamos usado los métodos “get y set” para encapsularlas, y así poderlas redefinir mediante los métodos “get y set”, lo cual se había dicho en el párrafo anterior, respecto a que los métodos sí se pueden redefinir. De ahí que para la propiedad de herencia, solo se usan métodos heredados para poder acceder a los atributos de la superclase.