INSTITUTO POLITÉCNICO NACIONAL

Escuela Superior de Cómputo

Unidad de aprendizaje

"Programación Orientada a Objetos"

Grupo:

2CM1

Profesor:

Daniel Cruz García

Tarea 4:

"Herencia"

Alumna:

Luciano Espina Melisa

Fecha de entrega:

15/Abril/2018

Herencia

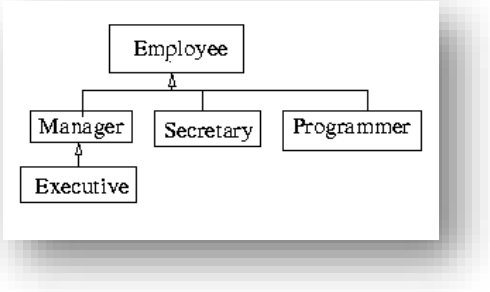
Java permite el empleo de la herencia, característica muy potente que permite definir una clase tomando como base a otra clase ya existente. Esto es una de las bases de la reutilización de código, en lugar de copiar y pegar.

En java, la herencia se especifica agregando la “cláusula extends” después del nombre de la clase. [1]

La idea de la herencia es permitir la creación de nuevas clases basadas en clases existentes. Cuando heredamos de una clase existente, reusamos (o heredamos) métodos y campos, y agregamos nuevos campos y métodos para cumplir con la situación nueva.

Cada vez que encontremos la relación "es-un" entre dos clases, estamos ante la presencia de herencia.

- La clase ya existente es llamada superclass, o clase base, o clase padre.
- La clase nueva es llamada subclase, clase derivada, o clase hija.
- A través de la herencia podemos agregar nuevos campos, y podemos agregar o sobre montar métodos (override). Sobre montar un método es redefinirlo en la case heredada.
- Hay que destacar uso de **super** para invocar al constructor de la clase base y para invocar a métodos sobremontados.



En java los atributos y métodos de la clase base pueden cambiar su modificador de visibilidad dentro de la clase derivada, la siguiente tabla recoge dichos cambios:

	Modificadores en la clase base			
	Public	Private	protected	paquete
En la derivada se transforma en	Public	Inaccesible	Protected	paquete

Inaccesible significa que, a pesar de haber sido heredado, no hay permisos en la clase derivada para poder acceder a dicho elemento inaccesible, pero aun así se pueden llamar a métodos de la clase base que si pueden acceder y modificar al elemento.

Recordemos que protected significa que es private, pero que al heredar no se hace inaccesible, es decir que desde la clase derivada se puede acceder.

¿Qué sucede con las clases en ese proceso?

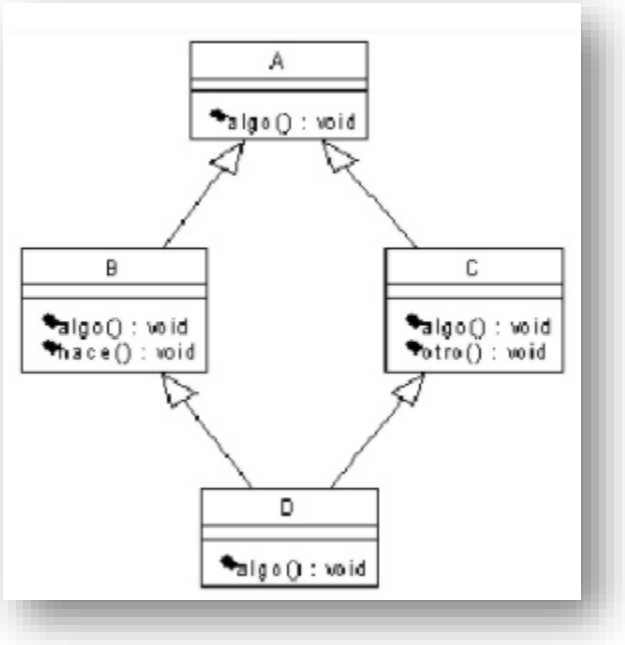
Al heredar de una clase base heredaremos tanto los atributos como los métodos, mientras que los constructores son utilizados, pero no heredados. [1]

Multiherencia

En algún momento, un programador puede estar tentado a derivar una clase única de varias clases. Esto se conoce como "herencia múltiple" y aunque parece útil, puede provocar problemas, como el importante "problema del diamante". Este problema ocurre cuando dos clases heredan de la misma clase (como la clase B y C derivan de la clase A), mientras que otra clase (D) deriva de B y C. Cuando se crea un objeto D, el sistema lo trata como un tipo de clase base (clase "Ball" o bola y "Sphere" o esfera, por ejemplo). En el problema del diamante, el sistema no puede determinar de forma decisiva qué clase D (¿es tipo A-B-D o tipo A-C-D?) es la que causa problemas.

Debido a los problemas con la herencia múltiple, Java no la permite. Pero en realidad las clases derivadas de varias clases base pueden conseguirse de una forma segura usando "interfaces". Una interfaz es parecida a una clase, pero no sólo define la estructura de la clase, sino también su código real. Una clase base que implementa una interfaz no "hereda" necesariamente la funcionalidad de la interfaz: sólo promete el uso de su estructura. Puesto que una clase que implementa una interfaz no hereda de otra clase (y no es un tipo de la clase base), un programador puede implementar varias interfaces usando la misma clase. [4]

Analicemos un caso de implementación en Java, basado en el esquema propuesto en la Figura 4, donde suponemos la existencia de un método denominado algo(), que es redefinido para cada clase con distintos comportamientos.



```
class A {  public void algo() {
System.out.println("A");
}
}
class B extends A {
public void algo() {
super.algo(); System.out.println("B");
}  public void hace()
{ System.out.println("Hace");
}
}
class C extends A
{  public void algo()
{ super.algo(); System.out.println("C"); }  public void otro() {
System.out.println("Otro"); } } class D
extends C, B
 {
public void algo() {
super.hace();
super.otro();
System.out.println("D");
}
 }
```

el planteo realizado en el código correspondiente a la Figura 4; el problema es que Java no provee medios para implementar herencia múltiple, por esto el código provoca un error durante el proceso de compilación, en el enunciado donde se indica que D hereda de C y B; como consecuencia de esto las activaciones super.otro() y super.hace() no son posibles. La alternativa a lo anterior es “simular” el esquema de herencia múltiple propuesto. Esta implementación, tiene que realizarse respetando las características de reusabilidad, redefinición de métodos y polimorfismo, que presenta el modelo originalmente propuesto. Para llevar adelante esto, se aprovecha una característica de Java, que permite “heredar”

en forma simultánea de una clase y una interface. Estrictamente, no se trata de herencia múltiple, sino de heredar de una clase e implementar una o más interfaces, en las cuales por definición todos los atributos son constantes y todos los métodos son abstractos (Cornell y Horstmann, 1996) [5]

Ejemplo sencillo entre clases

Herencia

```
Class Punto2D {
 Private int x, y;
 Punto2D(int x, int y){
        this.x=x; this.y=y;
 }

   Public void pintar () {
        …
   }
 }

 Final class Punto3D extends Punto2D {
 Private int z;

 Punto3D(int x, int y, int z ){

 Super(x, y); this z = z;
 }

 Public void pintar (){
 Super.pintar();
 }

```

Multiherencia

```
Interface Basic {
        Int doubleA();
 }
        Public class B implements Basic{
        Public int a;

   Public int doubleA()
      Return a * 2;
   }
 }

```

Bibliografía

[1] S/A 4.4.6 La herencia en java | Curso de Introducción a Java| Mundojava.net [Online] Available: http://www.mundojava.net/la-herencia-en-java.html?Pg=java_inicial_4_4_6.html

[2] S/A Herencia en java | Profesores.elo.utfsm.cl [Online] Available: http://profesores.elo.utfsm.cl/~agv/elo330/2s04/lectures/JAVA/Herencia_en_Java.html

[3] Gonzalo Méndez Pozo año: 2006 - 2007| Fdi.ucm.es| [Online] Available: https://www.fdi.ucm.es/profesor/gmendez/docs/prog0607/Tema4-Herencia.pdf

[4] S/A ¿Qué es la herencia múltiple en Java? | Techlandia [Online] Available: https://techlandia.com/herencia-multiple-java-info_269416/

[5] Eualio Gonzáles | Herencia múltiple en Java | Academia.edu [Online] Available: http://www.academia.edu/15830452/HERENCIA_M%C3%9ALTIPLE_EN_JAVA