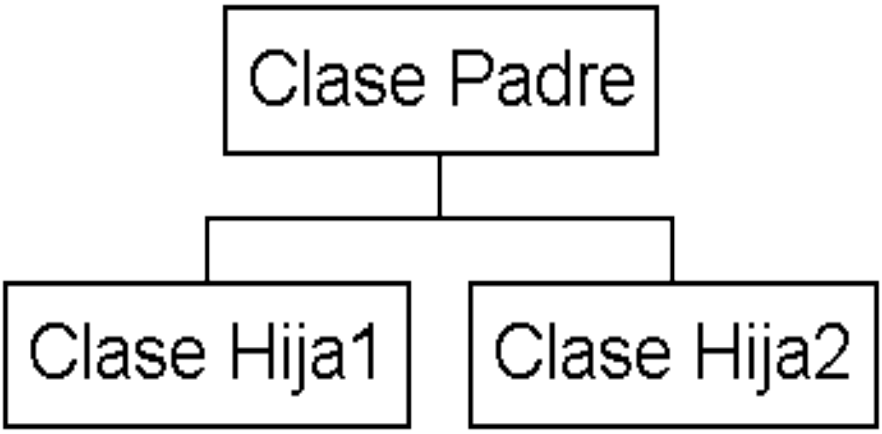
UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS



Herencia simple.

2013
Transversal Programación Orientada a Objetos
Proyecto Curricular de Ingeniería de Sistemas

Introducción

No solo basta con abstraer para consolidar objetos, se requiere indagar en las relaciones de dichas abstracciones, pues puede surgir alguna de utilidad.

Cuando se piensa en la solución de un problema por medio de la programación orientada a objetos, se debe considerar la existencia de diferentes mecanismos. Generalmente se puede encontrar cierta similitud entre objetos que aunque no ejemplifican la misma clase, coinciden en sus atributos y comportamientos, lo cual nos lleva a desear una generalización de los mismos para que ambos objetos los usen sin tener que repetir en diferentes clases los mismos segmentos de código, surge la necesidad de reutilización. Para esta situación existe un mecanismo específico que se explicará posteriormente.

1. ¿Herencia?

Básicamente se busca crear nuevas clases en base a clases existentes, facilitando la reutilización de código. Para ser más claros, se tiene una clase padre (clase base) y una clase hija (clase derivada), la cual como sucede con los seres humanos, hereda las características de su padre y podrá tener más características a parte de las de este.
Miremos algunos ejemplos para dejar claro lo que es herencia:

Fuente:www.iconfinder.com

Si deseáramos crear clases correspondientes a uva, fresa, banano, manzana y naranja, notaríamos que para cada una de esas clases, requeriríamos probablemente los siguientes atributos:

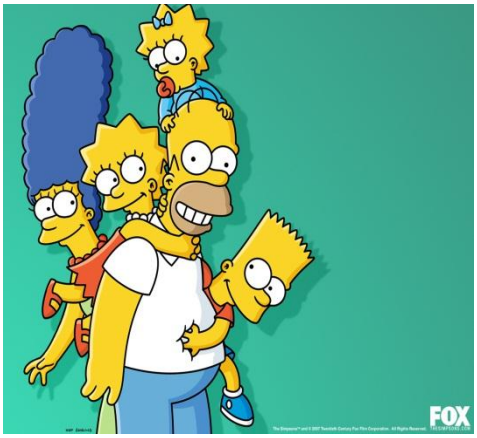
- Color.
- Tamaño.
- Estado.
- Sabor (cítrico o dulce).
- Función medicinal.

Suponiendo además que con cada uno se desea saber si se puede hacer jugo, considerando su estado y tamaño, todas la clases tendrían el método **verficarEstado().**

¿Qué podemos hacer para no escribir los mismos atributos y el mismo método en cada clase? Sencillamente consideraremos que ante todo esas clases hacen referencia a frutas; el banano es una fruta, la uva también al igual que la manzana, fresa y naranja, así que podemos decir que la clase base será fruta la cual tendrá atributos: Color, tamaño, estado, sabor y función medicinal y el método verificarEstado(). Las clases derivadas serán fresa, banano, manzana uva y naranja, las cuales al heredar de fruta, tendrán esas mismas características y ese método, aun cuando no se visualice en las clases de ellas explícitamente.
Además si quisiéramos, podríamos hacer que una de esas clases derivadas se convirtiera en clase base para otras clases, como sería decir que naranja es clase base para NaranjaNavel,

NaranjaLeng, las cuales son clases de naranjas y tendrán las mismas características que la clase naranja, las cuales le fueron heredadas de fruta.

Otro ejemplo podría ser empleando Los Simpsons. Homero, Marge, Lisa, Bart y Maggie.



Fuente: http://www.thesimpsons.com

Podríamos crear una clase base que se llame personaje, que tenga los atributos:

- Sexo.
- Estatura.
- Carácter.
- Color_de_cabello.
- Color_de_ojos.
- Contextura.

Y métodos que todos tendrían en común como:

- hablar()
- caminar()
- saltar()
- reir()

En ese caso Lisa tendría dichas características, pero si deseamos puede tener métodos adicionales como tocarSaxofon(), estudiar() y leer().

2. ¿Cómo la uso?

Una vez que se tiene claro lo que representa la herencia, se ilustra cómo implementarla en el código y su representación en un diagrama.

- **Código**

La palabra clave en este asunto es **extends**. Básicamente como su nómbrelo indica extiende las características de la clase base a las clases derivadas.
Como ejemplo, tomaremos la clase base Dispositivo Movil.

```
class DispositivoMovil{
      private String S= new String("Dispositivo Movil");
      public void anadir(String a) { S += a; }
      public void conectar() { anadir(" conectar "); }
      public void reconocer () { anadir (" reconocer") ; }
      public void almacenar () { anadir (" almacenar () "); }
      public void escribir() { System.out.println(S); }
      public static void main(String[] args) {
            Dispositivo x = new Dispositivo();
            x. conectar(); x. reconocer() ; x. almacenar() ;
            x. escribir();

}
```

Esta clase posee un atributo String al cual se le podrá añadir diferentes características, al invocar el método escribir(), se imprime:

```
DispositivoMovil conectar reconocer almacenar
```

Ahora la clase derivada será Celular y para heredar de DispositivoMovil deberá emplear extends:

```
class Celular extends DispositivoMovil {
 // Cambiar un método:
      public void conectar () {
            anadir (" Celular conectar ") ;
            super.conectar();//Llama a la versión de la clase base
      }
// Añadir métodos:
      public void llamar () { anadir(" llamar "); }
// Probar la nueva clase:

      public static void main(String[] args) {
            Celular x = new Celular();
            x.conectar();
            x.reconocer();
            x.almacenar();
            x.llamar();
            x.escribir();

}
}
```

Observe que los atributos son privados y los métodos públicos, esto se debe a que si una clase desea heredar de una clase base y esta no pertenece al mismo paquete, no podría emplear sus métodos. También, aunque no se vean explícitamente en el código, Celular posee todos los métodos y atributos de Dispositivo Movil.

Además, se dan dos situaciones:

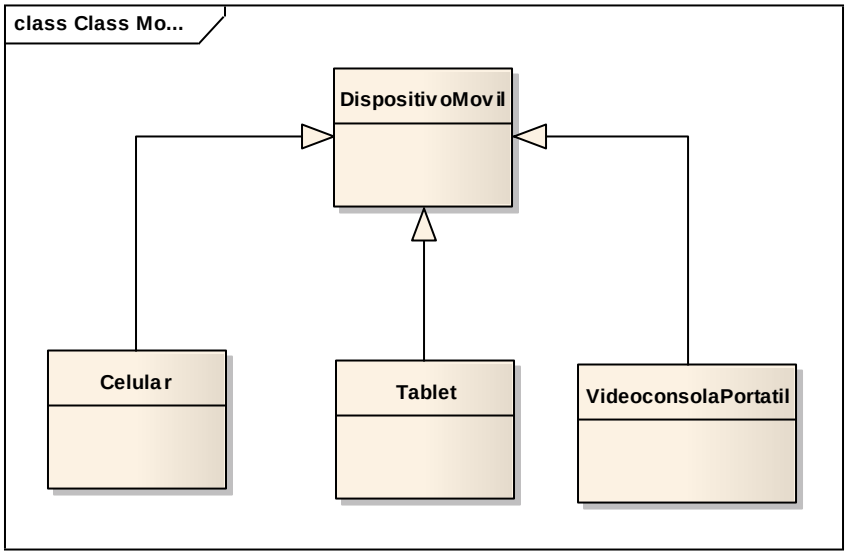
- El uso de **super,** ¿qué permite?
  Como se observa una clase derivada puede cambiar (sobrecargar) un método, es decir, acoplar el método del que dispone de la clase base y cambiar su contenido por lo que necesita y en caso de requerir llamar al método de la clase base (el original) se emplea el super, para que no se de una recursión. Para este caso, conectar() ha sido modificado por la clase celular, pero dentro de conectar() requiere llamar a conectar() de la clase DispositivoMovil, el cual no ha sido modificado, entonces emplea super para que no se piense que se esgtá tratando de invocar el conectar() de la clase Celular.

- Hay un nuevo método… llamar().

Como se observa se pueden añadir más métodos, no se tiene restricción a los que se heredan.
Teniendo esto claro, podemos observar que en la salida se imprimirá celular conectar y conectar seguidos, en vista de que en el método conectar de Celular se modificó y además se invocó el de DispositivoMovil por medio de super.

```
Celular conectar conectar reconocer almacenar llamar
```

- **Diagrama**
  Al expresarlo como diagrama se emplea una flecha que indica generalización y especificación. La clase derivada es una especialización de la clase base y la clase base es una generalización de la derivada.



Fuente: Autor

3. Inicialización de la clase base.

La herencia no es una simple copia de contenido y posibilidad de añadir. La creación de un objeto de la clase derivada lleva de por si un sub ojeto de la clase base (tal cual como si se generara un objeto normal de esta clase) generando una envoltura.

El constructor de la clase derivada genera una llamada al constructor de la clase base, garantizando que el subojeto de la clase base se inicialice correctamente.

Como ejemplo:

```
//Llamadas al constructor durante la herencia
class Animal {
      Animal() {
            System.out.println("Constructor de animal");
      }
}
class Mamifero extends Animal {
      Mamifero () {
            System.out.println ("Constructor de mamifero");
      }
}
public class Perro extends Mamifero {
      Perro () {
            System.out.println("Constructor de perro");
      }
}

public static void main (String[] args) {
                        Perro x = new Perro ();

}
```

La salida de este programa sería:

```
Constructor de animal
Constructor de mamifero
Constructor de perro
```

Como se observa para generar el objeto de la clase Perro, se invocan los constructores de manera interna - externa, es decir, primero se inicializa la clase base antes de que los constructores de las clases derivadas accedan.

Bibliografía

- Eckel, Bruce. Thinking in Java. Prentice Hall. 1998. Estados Unidos.