HERENCIA

PROGRAMACIÓN

Herencia

Vimos en el concepto anterior que dos clases pueden estar relacionadas por la colaboración. Ahora veremos otro tipo de relaciones entre clases que es la Herencia.

La herencia significa que se pueden crear nuevas clases partiendo de clases existentes, que tendrá todas los atributos y los métodos de su 'superclase' o 'clase padre' y además se le podrán añadir otros atributos y métodos propios.

clase padre

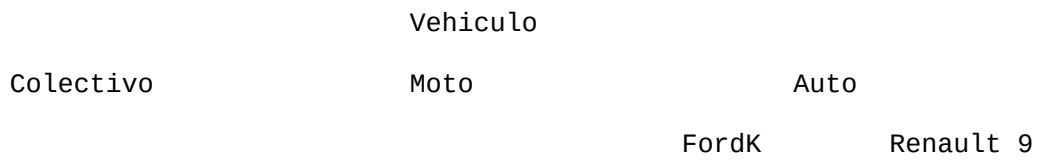
Clase de la que desciende o deriva una clase. Las clases hijas (descendientes) heredan (incorporan) automáticamente los atributos y métodos de la la clase padre.

Subclase

Clase desciendiente de otra. Hereda automáticamente los atributos y métodos de su superclase. Es una especialización de otra clase. Admiten la definición de nuevos atributos y métodos para aumentar la especialización de la clase.

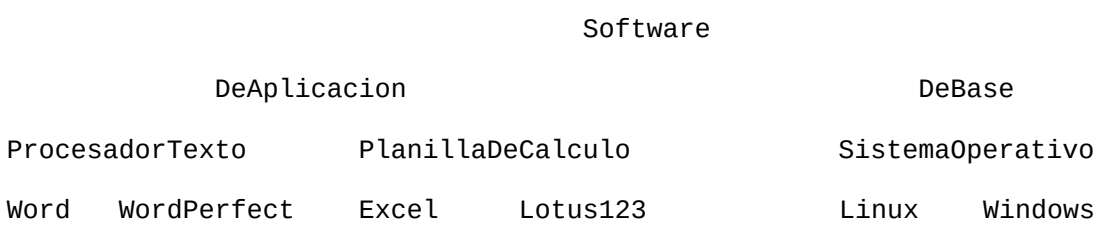
Veamos algunos ejemplos teóricos de herencia:

1) Imaginemos la clase Vehículo. Qué clases podrían derivar de ella?



Siempre hacia abajo en la jerarquía hay una especialización (las subclases añaden nuevos atributos y métodos.

2) Imaginemos la clase Software. Qué clases podrían derivar de ella?



El primer tipo de relación que habíamos visto entre dos clases, es la de colaboración. Recordemos que es cuando una clase contiene un objeto de otra clase como atributo. Cuando la relación entre dos clases es del tipo "...tiene un..." o "...es parte de...", no debemos implementar herencia. Estamos frente a una relación de colaboración de clases no de herencia.

Si tenemos una ClaseA y otra ClaseB y notamos que entre ellas existe una relacion de tipo "... tiene un...", no debe implementarse herencia sino declarar en la clase ClaseA un atributo de la clase ClaseB.

Por ejemplo: tenemos una clase Auto, una clase Rueda y una clase Volante. Vemos que la relación entre ellas es: Auto "...tiene 4..." Rueda, Volante "...es parte de..." Auto; pero la clase Auto no debe derivar de Rueda ni Volante de Auto porque la relación no es de tipo-subtipo sino de colaboración. Debemos declarar en la clase Auto 4 atributos de tipo Rueda y 1 de tipo Volante.

Luego si vemos que dos clase responden a la pregunta ClaseA "..es un.." ClaseB es posible que haya una relación de herencia.

Por ejemplo:

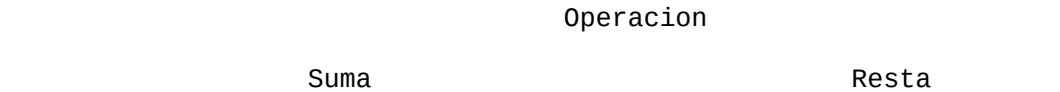
```
Auto "es un" Vehiculo
Circulo "es una" Figura
Mouse "es un" DispositivoEntrada
Suma "es una" Operacion
```

Problema 1:

Ahora plantearemos el primer problema utilizando herencia. Supongamos que necesitamos implementar dos clases que llamaremos Suma y Resta. Cada clase tiene como atributo valor1, valor2 y resultado. Los métodos a definir son cargar1 (que inicializa el atributo valor1), carga2 (que inicializa el atributo valor2), operar (que en el caso de la clase "Suma" suma los dos atributos y en el caso de la clase "Resta" hace la diferencia entre valor1 y valor2, y otro método mostrarResultado.

Si analizamos ambas clases encontramos que muchos atributos y métodos son idénticos. En estos casos es bueno definir una clase padre que agrupe dichos atributos y responsabilidades comunes.

La relación de herencia que podemos disponer para este problema es:



Solamente el método operar es distinto para las clases Suma y Resta (esto hace que no lo podamos disponer en la clase Operacion), luego los métodos cargar1, cargar2 y mostrarResultado son idénticos a las dos clases, esto hace que podamos disponerlos en la clase Operacion. Lo mismo los atributos valor1, valor2 y resultado se definirán en la clase padre Operacion.

Crear un proyecto y luego crear cuatro clases llamadas: Operacion, Suma, Resta y Prueba

Programa:

```
import java.util.Scanner;
public class Operacion {
    protected Scanner teclado;
    protected int valor1;
```

```
    protected int valor2;
    protected int resultado;
    public Operacion() {
        teclado=new Scanner(System.in);
    }

    public void cargar1() {
        System.out.print("Ingrese el primer valor:");
        valor1=teclado.nextInt();
    }

    public void cargar2() {
        System.out.print("Ingrese el segundo valor:");
        valor2=teclado.nextInt();
    }

    public void mostrarResultado() {
        System.out.println(resultado);
    }
}



public class Suma extends Operacion{
    void operar() {
        resultado=valor1+valor2;
    }
}



public class Resta extends Operacion {
    public void operar(){
        resultado=valor1-valor2;
    }
}



public class Prueba {
    public static void main(String[] ar) {
        Suma suma1=new Suma();
        suma1.cargar1();
        suma1.cargar2();
        suma1.operar();
        System.out.print("El resultado de la suma es:");
        suma1.mostrarResultado();
        Resta resta1=new Resta();
        resta1.cargar1();
        resta1.cargar2();
        resta1.operar();
        System.out.print("El resultado de la resta es:");
        resta1.mostrarResultado();
    }
}
```

La clase Operación define los cuatro atributos:

```
import java.util.Scanner;
public class Operacion {
    protected Scanner teclado;
    protected int valor1;
    protected int valor2;
    protected int resultado;
```

Ya veremos que definimos los atributos con este nuevo modificador de acceso (protected) para que la subclase tenga acceso a dichos atributos. Si los definimos private las subclases no pueden acceder a dichos atributos.

Los métodos de la clase Operacion son:

```
    public Operacion() {
        teclado=new Scanner(System.in);
    }

    public void cargar1() {
        System.out.print("Ingrese el primer valor:");
        valor1=teclado.nextInt();
    }

    public void cargar2() {
        System.out.print("Ingrese el segundo valor:");
        valor2=teclado.nextInt();
    }

    public void mostrarResultado() {
        System.out.println(resultado);
    }
```

Ahora veamos como es la sintaxis para indicar que una clase hereda de otra:

```
public class Suma extends Operacion{
```

Utilizamos la palabra clave extends y seguidamente el nombre de la clase padre (con esto estamos indicando que todos los métodos y atributos de la clase Operación son también métodos de la clase Suma.

Luego la característica que añade la clase Suma es el siguiente método:

```
    void operar() {
        resultado=valor1+valor2;
    }
```

El método operar puede acceder a los atributos heredados (siempre y cuando los mismos se declaren protected, en caso que sean private si bien lo hereda de la clase padre solo los pueden modificar métodos de dicha clase padre.

Ahora podemos decir que la clase Suma tiene cinco métodos (cuatro heredados y uno propio) y 3 atributos (todos heredados)

Luego en otra clase creamos un objeto de la clase Suma:

```
public class Prueba {
```

```
    public static void main(String[] ar) {
        Suma suma1=new Suma();
        suma1.cargar1();
        suma1.cargar2();
        suma1.operar();
        System.out.print("El resultado de la suma es:");
        suma1.mostrarResultado();
        Resta resta1=new Resta();
        resta1.cargar1();
        resta1.cargar2();
        resta1.operar();
        System.out.print("El resultado de la resta es:");
        resta1.mostrarResultado();
    }
}
```

Podemos llamar tanto al método propio de la clase Suma "operar()" como a los métodos heredados. Quien utilice la clase Suma solo debe conocer que métodos públicos tiene (independientemente que pertenezcan a la clase Suma o a una clase superior)

La lógica es similar para declarar la clase Resta.

La clase Operación agrupa en este caso un conjunto de atributos y métodos comunes a un conjunto de subclases (Suma, Resta). No tiene sentido definir objetos de la clase Operacion.

El planteo de jerarquías de clases es una tarea compleja que requiere un perfecto entendimiento de todas las clases que intervienen en un problema, cuales son sus atributos y responsabilidades.

Problemas propuestos

1. Confeccionar una clase Persona que tenga como atributos el nombre y la edad. Definir como responsabilidades un método que cargue los datos personales y otro que los imprima.
   Plantear una segunda clase Empleado que herede de la clase Persona. Añadir un atributo sueldo y los métodos de cargar el sueldo e imprimir su sueldo. Definir un objeto de la clase Persona y llamar a sus métodos. También crear un objeto de la clase Empleado y llamar a sus métodos.