Programación II
Herencia en Java (Parte2)

Composición

En anteriores ejemplos se ha visto que una clase tiene datos miembro que son instancias de otras clases. Por ejemplo:

```
class Circulo {
        Punto centro;
        int radio;
        float superficie() {
                return 3.14 * radio * radio;
        }
}
```

Esta técnica en la que una clase se compone o contiene instancias de otras clases se denomina composición. Es una técnica muy habitual cuando se diseñan clases. En el ejemplo diríamos que un Circulo tiene un Punto (centro) y un radio.

Herencia

Pero además de esta técnica de composición es posible pensar en casos en los que una clase es una extensión de otra. Es decir una clase es como otra y además tiene algún tipo de característica propia que la distingue. Por ejemplo podríamos pensar en la clase Empleado y definirla como:

```
class Empleado {
    String nombre;
    int numEmpleado , sueldo;

    static private int contador = 0;


      //constructor no-args
    //Empleado() {      }


    Empleado(String nombre, int sueldo) {
        this.nombre = nombre;
        this.sueldo = sueldo;
        numEmpleado = ++contador;
    }

    public void aumentarSueldo(int porcentaje) {
        sueldo += (int)(sueldo * porcentaje / 100);
    }

    public String toString() {
        return "Num. empleado " + numEmpleado + " Nombre: " + nombre +
                " Sueldo: " + sueldo;
    }
}
```

Con esta representación podemos pensar en otra clase que reuna todas las características de Empleado y añada alguna propia. Por ejemplo, la clase Ejecutivo. A los objetos de esta clase se les podría aplicar todos los datos y métodos de la clase Empleado y añadir algunos, como por ejemplo el hecho de que un Ejecutivo tiene un presupuesto.

Así diriamos que la clase Ejecutivo extiende o hereda la clase Empleado. Esto en Java se hace con la clausula extends que se incorpora en la definición de la clase, de la siguiente forma:

```
class Ejecutivo extends Empleado {
    int presupuesto;
    void asignarPresupuesto(int p) {
        presupuesto = p;
    }
}
```

Con esta definición un Ejecutivo es un Empleado que además tiene algún rasgo distintivo propio. El cuerpo de la clase Ejecutivo incorpora sólo los miembros que son específicos de esta clase, pero implícitamente tiene todo lo que tiene la clase Empleado.

A Empleado se le llama clase base o superclase y a Ejecutivo clase derivada o subclase.

Los objetos de las clases derivadas se crean igual que los de la clase base y pueden acceder tanto sus datos y métodos como a los de la clase base. Por ejemplo:

```
Ejecutivo jefe = new Ejecutivo( "Armando Mucho", 1000);
jefe.asignarPresupuesto(1500);
jefe.aumentarSueldo(5);
```

Nota: La discusión acerca de los constructores la veremos un poco más adelante.
Atención!: **Un Ejecutivo ES un Empleado, pero lo contrario no es cierto**.

```
Empleado e = new Empleado ( "Esteban " , 100) ;
```

Si escribimos:

```
e.asignarPresupuesto(5000);  // error
```

Se producirá un error de compilación pues en la clase Empleado no existe ningún método llamado asignarPresupuesto.

Redefinición de métodos. El uso de *super*.

Además se podría pensar en redefinir algunos métodos de la clase base pero haciendo que métodos con el mismo nombre y características se comporten de forma distinta.

Por ejemplo podríamos pensar en rediseñar el método toString de la clase Empleado añadiendo las características propias de la clase Ejecutivo. Así se podría poner:

```
class Ejecutivo extends Empleado {
    int presupuesto;

    void asignarPresupuesto(int p) {
        presupuesto = p;
    }

    public String toString() {
        String s = super.toString();
        s = s + " Presupuesto: " + presupuesto;
        return s;
    }
}
```

De esta forma cuando se invoque `jefe.toString()` se usará el método toString de la clase Ejecutivo en lugar del existente en la clase Empleado.

Observese en el ejemplo el uso de **super**, que representa referencia interna implícita a la clase base (superclase). Mediante `super.toString()` se invoca el método toString de la clase Empleado

Inicialización de clases derivadas

Cuando se crea un objeto de una clase derivada se crea implicitamente un objeto de la clase base que se inicializa con su constructor correspondiente.

Si en la creación del objeto se usa el constructor no-args, entonces se produce una llamada implicita al constructor no-args para la clase base.

Pero si se usan otros constructores es necesario invocarlos explicitamente.

En nuestro ejemplo dado que la clase método define un constructor, necesitaremos también un constructor para la clase Ejecutivo, que podemos completar así:

```
class Ejecutivo extends Empleado {
    int presupuesto;

      Ejecutivo () { }        //constructor no-args

    Ejecutivo (String n, int s) {
        super(n,s);
    }

    void asignarPresupuesto(int p) {
        presupuesto = p;
    }

    public String toString() {
        String s = super.toString();
        s = s + " Presupuesto: " + presupuesto;
        return s;
    }
}
```

Observese que el constructor de Ejecutivo invoca directamente al constructor de Empleado mediante *super(argumentos)*.

En caso de resultar necesaria la invocación al constructor de la superclase debe ser la primera sentencia del constructor de la subclase.

Referencias:

http://www.arrakis.es/~abelp/ApuntesJava/Herencia.htm