

La clase base y la clase derivada

La herencia es una propiedad esencial de la Programación Orientada a Objetos que consiste en la creación de nuevas clases a partir de otras ya existentes. Este término ha sido prestado de la Biología donde afirmamos que un niño tiene la cara de su padre, que ha heredado ciertas facetas físicas o del comportamiento de sus progenitores.

La herencia es la característica fundamental que distingue un lenguaje orientado a objetos, como el C++ o Java, de otro convencional como C, BASIC, etc. Java permite heredar a las clases características y conductas de una o varias clases denominadas base. Las clases que heredan de clases base se denominan derivadas, estas a su vez pueden ser clases bases para otras clases derivadas. Se establece así una clasificación jerárquica, similar a la existente en Biología con los animales y las plantas.

La herencia ofrece una ventaja importante, permite la reutilización del código. Una vez que una clase ha sido depurada y probada, el código fuente de dicha clase no necesita modificarse. Su funcionalidad se puede cambiar derivando una nueva clase que herede la funcionalidad de la clase base y le añada otros comportamientos. Reutilizando el código existente, el programador ahorra tiempo y dinero, ya que solamente tiene que verificar la nueva conducta que proporciona la clase derivada.

La programación en los entornos gráficos, en particular Windows, con el lenguaje C++, es un ejemplo ilustrativo. Los compiladores como los de Borland y Microsoft proporcionan librerías cuyas clases describen el aspecto y la conducta de las ventanas, controles, menús, etc. Una de estas clases denominada *TWindow* describe el aspecto y la conducta de una ventana, tiene una función miembro denominada *Paint*, que no dibuja nada en el área de trabajo de la misma. Definiendo una clase derivada de *TWindow*, podemos redefinir en ella la función *Paint* para que dibuje una figura. Aprovechamos de este modo la ingente cantidad y complejidad del código necesario para crear una ventana en un entorno gráfico. Solamente, tendremos que añadir en la clase derivada el código necesario para dibujar un rectángulo, una elipse, etc.

En el lenguaje Java, todas las clases derivan implícitamente de la clase base *Object*, por lo que heredan las funciones miembro definidas en dicha clase. Las clases derivadas pueden redefinir algunas de estas funciones miembro como [*toString*](#) y definir otras nuevas.

Para crear un applet, solamente tenemos que definir una clase derivada de la clase base *Applet*, redefinir ciertas funciones como *init* o *paint*, o definir otras como las respuestas a las acciones sobre los controles.

Los programadores crean clases base:

1. Cuando se dan cuenta que diversos tipos tienen algo en común, por ejemplo en el juego del ajedrez peones, alfiles, rey, reina, caballos y torres, son piezas del juego. Creamos, por tanto, una clase base y derivamos cada pieza individual a partir de dicha clase base.

2. Cuando se precisa ampliar la funcionalidad de un programa sin tener que modificar el código existente.

La clase base

Vamos a poner un ejemplo del segundo tipo, que simule la utilización de librerías de clases para crear un interfaz gráfico de usuario como Windows.

Supongamos que tenemos una clase que describe la conducta de una ventana muy simple, aquella que no dispone de título en la parte superior, por tanto no puede desplazarse, pero si cambiar de tamaño actuando con el ratón en los bordes derecho e inferior.

La clase *Ventana* tendrá los siguientes miembros dato: la posición *x* e *y* de la ventana, de su esquina superior izquierda y las dimensiones de la ventana: *ancho* y *alto*.

```
public class Ventana {
    protected int x;
    protected int y;
    protected int ancho;
    protected int alto;
    public Ventana(int x, int y, int ancho, int alto) {
        this.x=x;
        this.y=y;
        this.ancho=ancho;
        this.alto=alto;
    }
    //...
}
```

Las funciones miembros, además del constructor serán las siguientes: la función *mostrar* que simula una ventana en un entorno gráfico, aquí solamente nos muestra la posición y las dimensiones de la ventana.

```
public void mostrar(){
    System.out.println("posición      : x="+x+", y="+y);
    System.out.println("dimensiones   : w="+ancho+", h="+alto);
}
```

La función *cambiarDimensiones* que simula el cambio en la anchura y altura de la ventana.

```
public void cambiarDimensiones(int dw, int dh){
    ancho+=dw;
    alto+=dh;
}
```

El código completo de la clase base *Ventana*, es el siguiente

```
public class Ventana {
    protected int x;
```

```

protected int y;
protected int ancho;
protected int alto;
public Ventana(int x, int y, int ancho, int alto) {
    this.x=x;
    this.y=y;
    this.ancho=ancho;
    this.alto=alto;
}
public void mostrar(){
    System.out.println("posición      : x="+x+", y="+y);
    System.out.println("dimensiones   : w="+ancho+", h="+alto);
}
public void cambiarDimensiones(int dw, int dh){
    ancho+=dw;
    alto+=dh;
}
}

```

Objetos de la clase base

Como vemos en el código, el constructor de la clase base inicializa los cuatro miembros dato. Llamamos al constructor creando un objeto de la clase *Ventana*

```
Ventana ventana=new Ventana(0, 0, 20, 30);
```

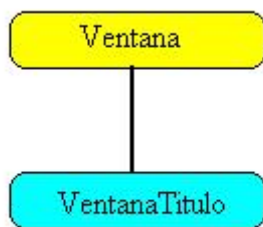
Desde el objeto *ventana* podemos llamar a las funciones miembro públicas

```

ventana.mostrar();
ventana.cambiarDimensiones(10, 10);
ventana.mostrar();

```

La clase derivada



Incrementamos la funcionalidad de la clase *Ventana* definiendo una clase derivada denominada *VentanaTitulo*. Los objetos de dicha clase tendrán todas las características de los objetos de la clase base, pero además tendrán un título, y se podrán desplazar (se simula el desplazamiento de una ventana con el ratón).

La clase derivada heredará los miembros dato de la clase base y las funciones miembro, y tendrá un miembro dato más, el título de la ventana.

```

public class VentanaTitulo extends Ventana{
    protected String titulo;
    public VentanaTitulo(int x, int y, int w, int h, String nombre) {
        super(x, y, w, h);
    }
}

```

```
        titulo=nombre;
    }
```

extends es la palabra reservada que indica que la clase *VentanaTitulo* deriva, o es una subclase, de la clase *Ventana*.

La primera sentencia del constructor de la clase derivada es una llamada al constructor de la clase base mediante la palabra reservada **super**. La llamada

```
super(x, y, w, h);
```

inicializa los cuatro miembros de la clase base *Ventana*: *x*, *y*, *ancho*, *alto*. A continuación, se inicializan los miembros de la clase derivada, y se realizan las tareas de inicialización que sean necesarias. Si no se llama explícitamente al constructor de la clase base Java lo realiza por nosotros, llamando al constructor por defecto si existe.

La función miembro denominada *desplazar* cambia la posición de la ventana, añadiéndoles el desplazamiento.

```
public void desplazar(int dx, int dy){
    x+=dx;
    y+=dy;
}
```

Redefine la función miembro *mostrar* para mostrar una ventana con un título.

```
public void mostrar(){
    super.mostrar();
    System.out.println("titulo      : "+titulo);
}
```

En la clase derivada se define una función que tiene el mismo nombre y los mismos parámetros que la de la clase base. Se dice que redefinimos la función *mostrar* en la clase derivada. La función miembro *mostrar* de la clase derivada *VentanaTitulo* hace una llamada a la función *mostrar* de la clase base *Ventana*, mediante

```
super.mostrar();
```

De este modo aprovechamos el código ya escrito, y le añadimos el código que describe la nueva funcionalidad de la ventana por ejemplo, que muestre el título.

Si nos olvidamos de poner la palabra reservada **super** llamando a la función *mostrar*, tendríamos una función recursiva. La función *mostrar* llamaría a *mostrar* indefinidamente.

```
public void mostrar(){ //¡ojo!, función recursiva
    System.out.println("titulo      : "+titulo);
    mostrar();
}
```

La definición de la clase derivada *VentanaTitulo*, será la siguiente.

```

public class VentanaTitulo extends Ventana{
    protected String titulo;
    public VentanaTitulo(int x, int y, int w, int h, String nombre) {
        super(x, y, w, h);
        titulo=nombre;
    }
    public void mostrar(){
        super.mostrar();
        System.out.println("titulo      : "+titulo);
    }
    public void desplazar(int dx, int dy){
        x+=dx;
        y+=dy;
    }
}

```

Objetos de la clase derivada

Creamos un objeto *ventana* de la clase derivada *VentanaTitulo*

```

VentanaTitulo ventana=new VentanaTitulo(0, 0, 20, 30,
"Principal");

```

Mostramos la ventana con su título, llamando a la función *mostrar*, redefinida en la clase derivada

```

ventana.mostrar();

```

Desde el objeto *ventana* de la clase derivada llamamos a las funciones miembro definidas en dicha clase

```

ventana.desplazar(4, 3);

```

Desde el objeto *ventana* de la clase derivada podemos llamar a las funciones miembro definidas en la clase base.

```

ventana.cambiarDimensiones(10, -5);

```

Para mostrar la nueva ventana desplazada y cambiada de tamaño escribimos

```

ventana.mostrar();

```

definición de la clase derivada *VentanaTitulo*, será la siguiente.

```

public class VentanaApp {

    public static void main(String[] args) {
        VentanaTitulo ventana=new VentanaTitulo(0, 0, 20, 30,
"Principal");
        ventana.mostrar();
    }
}

```

```

        ventana.cambiarDimensiones(10, -5);
        ventana.desplazar(4, 3);
        System.out.println("*****");
        ventana.mostrar();
    }
}

```

Modificadores de acceso (opcional)

Ya hemos visto el significado de los modificadores de acceso **public** y **private**, así como el control de acceso por defecto a nivel de paquete, cuando no se especifica nada. En la herencia, surge un nuevo control de acceso denominado **protected**.

Hemos puesto **protected** delante de los miembros *x* e *y* de la clase base *Ventana*

```

public class Ventana {
    protected int x;
    protected int y;
    //...
}

```

En la clase derivada la función miembro *desplazar* accede a dichos miembros dato

```

public class VentanaTitulo extends Ventana{
    //...
    public void desplazar(int dx, int dy){
        x+=dx;
        y+=dy;
    }
}

```

Si cambiamos el modificador de acceso de los miembros *x* e *y* de la clase base *Ventana* de **protected** a **private**, veremos que el compilador se queja diciendo que los miembros *x* e *y* no son accesibles.

Los miembros *ancho* y *alto* se pueden poner con acceso **private** sin embargo, es mejor dejarlos como **protected** ya que podrían ser utilizados por alguna función miembro de otra clase derivada de *VentanaTitulo*. Dentro de una jerarquía pondremos un miembro con acceso **private**, si estamos seguros de que dicho miembro solamente va a ser usado por dicha clase.

Como vemos hay cuatro modificadores de acceso a los miembros dato y a los métodos: **private**, **protected**, **public** y **default** (por defecto, o en ausencia de cualquier modificador). La herencia complica aún más el problema de acceso, ya que las clases dentro del mismo paquete tienen diferentes accesos que las clases de distinto paquete

Los siguientes cuadros tratan de aclarar este problema

Clases dentro del mismo paquete		
Modificador de acceso	Heredado	Accesible
Por defecto (sin modificador)	Si	Si
private	No	No
protected	Si	Si
public	Si	Si

Clases en distintos paquetes		
Modificador de acceso	Heredado	Accesible
Por defecto (sin modificador)	No	No
private	No	No
protected	Si	No
public	Si	Si

Desde el punto de vista práctico, cabe reseñar que no se heredan los miembros privados, ni aquellos miembros (dato o función) cuyo nombre sea el mismo en la clase base y en la clase derivada.

La clase base *Object* (opcional)

La clase *Object* es la clase raíz de la cual derivan todas las clases. Esta derivación es implícita.

La clase *Object* define una serie de funciones miembro que heredan todas las clases. Las más importantes son las siguientes

```
public class Object {
    public boolean equals(Object obj) {
        return (this == obj);
    }
    protected native Object clone() throws CloneNotSupportedException;
    public String toString() {
        return getClass().getName() + "@" +
Integer.toHexString(hashCode());
    }
    protected void finalize() throws Throwable { }
    //otras funciones miembro...
```

```
}
```

Igualdad de dos objetos:

Hemos visto que el método [equals](#) de la clase *String* cuando compara un string y cualquier otro objeto. El método *equals* de la clase *Object* compara dos objetos uno que llama a la función y otro es el argumento de dicha función.

Representación en forma de texto de un objeto

El método *toString* imprime por defecto el nombre de la clase a la que pertenece el objeto y su código (hash). Esta función miembro se redefine en la clase derivada para mostrar la información que nos interese acerca del objeto. En la misma página, hemos mejorado la clase [Lista](#) para mostrar los datos que se guardan en los objetos de dicha clase, redefiniendo *toString*.

La función *toString* se llama automáticamente siempre que pongamos un objeto como argumento de la función *System.out.println* o concatenado con otro string.

Duplicación de objetos

El método [clone](#) crea un objeto duplicado (clónico) de otro objeto. Más adelante estudiaremos en detalle la redefinición de esta función miembro y pondremos ejemplos que nos muestren su utilidad.

Finalización

El método *finalize* se llama cuando va a ser liberada la memoria que ocupa el objeto por el recolector de basura (garbage collector). Normalmente, no es necesario redefinir este método en las clases, solamente en contados casos especiales. La forma en la que se redefine este método es el siguiente.

```
class CualquierClase{
    //...
    protected void finalize() throws Throwable{
        super.finalize();
        //código que libera recursos externos
    }
}
```

La primera sentencia que contenga la redefinición de *finalize* ha de ser una llamada a la función del mismo nombre de la clase base, y a continuación le añadimos cierta funcionalidad, habitualmente, la liberación de recursos, cerrar un archivo, etc.