Fundamentos de la Programación OO en Java
Segunda Parte

Herencia y Polimorfismo

Con base a los conceptos
del curso de Sun Microsystems
“Lenguaje de Programación Java”
(Java Programming Language. SL275).

Luis Ernesto Rubio.

La herencia

La **herencia**, o principio de “la clasificación de objetos”, permite la transferencia de las características y/o los comportamientos a través de las diferentes clases de objetos involucrados en la solución de un problema.

De ese modo, es posible crear jerarquías taxonómicas, que permiten organizar los objetos del sistema y hacer más fácil su control y su aprovechamiento.

Además la herencia, posibilita la primera de las dos formas en que se presenta el mecanismo conocido como reutilización de código, mediante el que se simplifica y abate el tiempo y esfuerzo de programación.

En Java la herencia se codifica defiendo una sub-clase que hereda los atributos y/o métodos de una súper-clase utilizando la palabra reservada *extends*.

Ejemplo:

```
class Cuenta{ //A esta clase se le denomina Super-Clase

    protected double    balance;   //el modificador protect hace accesible el balance en las Sub-Clases

    public boolean deposito(double cantidad){
        if(cantidad >0){
            balance = balance + cantidad;
            return true;
        } else {
            return false;
        }
    }

    public boolean retiro(double cantidad){
        if(balance >= cantidad){
            balance-=cantidad;
            return true;
        } else {
            return false;
        }
    }

    public double getBalance(){
        return balance;
    }

}
```


```
class CuentaCheques extends Cuenta{ //CuentaCheques se denomina la Sub-Clase de la Super-Clase Cuenta

    private double proteccionSobregiro = 1000.0; //Este atributo permite retirar cantidades mayores al balance

    //Este nuevo método retiro evita sobregirar la cuenta ante una cantidad mayor al balance
    public boolean retiro(double cantidad){ //Este método sustituye (override) al retiro de la clase Cuenta
        if(balance >= cantidad){   //el atributo protegido balance (protected) es aqui accesible
            balance-=cantidad;
            return true;
        } else if((balance + proteccionSobregiro) >= cantidad){
            proteccionSobregiro+=balance;
            balance = 0;
            proteccionSobregiro-=cantidad;
            return true;
        }else{
            return false;
        }
    }
}
```

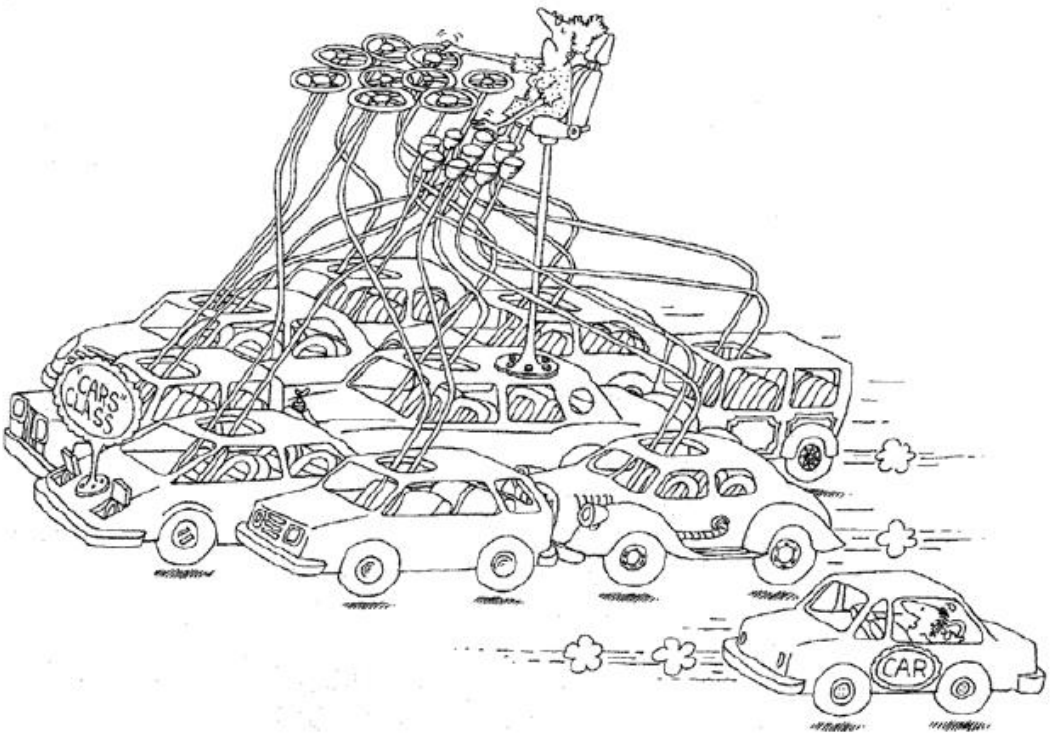
```
class TestHerencia {
   public static void main(String [ ] arguments){

      CuentaCheques unaChequera;
      unaChequera = new CuentaCheques();
      unaChequera.deposito(1000.0); //Reutiliza el depósito de la clase Cuenta

      //Despliega en pantalla: "Retirando $ 1500.00 : true"
      System.out.println(
          "Retirando $ 1500.00 de unaChequera: "
        + unaChequera.retiro(1500.0) //Ejecuta el método sustituido en CuentaCheques
      );

   }
}
```

El polimorfismo



También la herencia, hace posible la aplicación del tercer principio del paradigma OO, el polimorfismo.

El **polimorfismo** es el principio de “la usabilidad de los objetos”, en sí mismo, constituye una “cualidad”, deseable en todos los sistemas actuales.

En el paradigma OO, el **polimorfismo** se define como la capacidad de objetos de clases distintas, para comportarse, de acuerdo a ellas, ante una misma orden o solicitud.

Ejemplo:

```
class TestPolimorfismo{
    public static void main(String [ ] arguments){

        Cuenta poliCuenta;  //esta variable funciona como Super-Referencia:
                            //referencia de tipo super-clase (Cuenta) que refiere o nombra
                            //a un objeto de tipo sub-clase (CuentaCheques)

        poliCuenta = new CuentaCheques();  //la super-referencia refiere al objeto de tipo sub-clase (CuentaCheques)
        poliCuenta.deposito(1000.0);

        //Aqui, la instrucción: poliCuenta.retiro(1500.0); SI permite retirar $1500.00 de una cuenta de cheques
        System.out.println(
            "Retirando $ 1500.00 de poliCuenta: "
          + poliCuenta.retiro(1500.0)    //Polimorfismo: Despliega en pantalla: "Retirando $ 1500.00 : true"
                                         //Aqui funciona el mecanismo de Invocación Virtual de Métodos:
                                         //invocar un método sustituido mediante una super-referencia
        );

        poliCuenta = new Cuenta();  //poliCuenta ahora refiere a un objeto de distinta clase (Cuenta)
        poliCuenta.deposito(1000.0);

        //Pero aqui, la MISMA instrucción: poliCuenta.retiro(1500.0); NO permite retirar los $1500.00 de una cuenta
        System.out.println(
            "Retirando $ 1500.00 de poliCuenta: "
          + poliCuenta.retiro(1500.0)    //Polimorfismo: Despliega en pantalla: "Retirando $ 1500.00 : false"
                                         //Aqui se invoca normalmente el método retiro de la clase Cuenta
                                         //pues poliCuenta se definió como de tipo clase Cuenta
        );

        //El polimorfismo en Java es el mecanismo por el cual la misma instrucción arroja distintos resultados
        //Este mecanismo promueve la usabilidad y legibilidad del código
    }
}
```

En Java el polimorfismo es consecuencia de varios mecanismos: el uso de súper-referencias, la sustitución de métodos (override) y la invocación virtual de métodos.

Una **súper-referencia** es una variable de tipo súper-clase (clase base) que refiere a un objeto de tipos sub-clase.

Un **método sustituto** (overriden), es un método en una sub-clase, que sobre-escribe la lógica de un método del mismo nombre en la súper-clase.

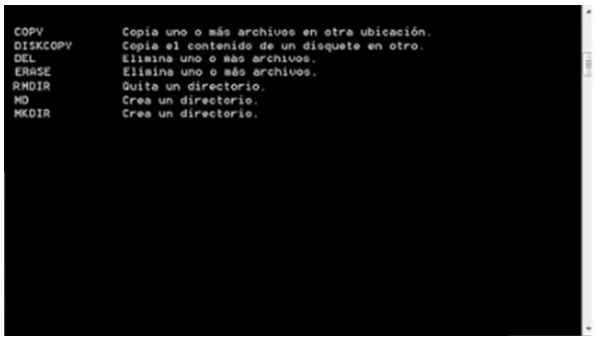
La **invocación virtual de métodos** es un mecanismo que consiste en llamar a un método sustituido mediante una súper-referencia.

Obsérvese el polimorfismo aparece cuándo diferentes instancias responden al mismo estímulo, de acuerdo a su particular naturaleza. Por ejemplo, si se le pide bailar a una pareja; el hombre y la mujer aunque, estando juntos, bailan de modo distinto, cada cual de acuerdo a su género, no obstante que el mensaje "bailar", haya sido el mismo para ambos.

Esta cualidad está presente en muchos de los sistemas y artefactos de nuestra vida diaria. Por ejemplo, los actuales reproductores de música y video, incluyen controles polimorfos (el botón "play" de una "grabadora" actual, sirve para tocar tanto, cintas, como discos compactos y hasta para reproducir películas).

Esta "comodidad", no era posible en los aparatos de antaño, antes un disco LP se "tocaba" colocando el brazo con la aguja del tocadiscos, sobre los surcos del disco giratorio, mientras que un "cassete", por su parte, se "tocaba" en un walkman oprimiendo botones. Estos aparatos no ofrecían el beneficio del polimorfismo.

Los sistemas de cómputo actuales, también incluyen en su funcionamiento esta característica. Hoy en día, igualmente se puede “tirar” al “bote de basura” (o papelera de reciclaje), con solo arrastralos allá, un archivo de datos, o toda una carpeta repleta de más archivos y/o más carpetas (“drag & drop”).



Antes del paradigma OO, no había polimorfismo y los sistemas eran engorrosos y difíciles de aprender a usar. Así, hace años, cuando solamente había sistemas al estilo MS DOS, se usaba el comando “del” para borrar archivos y otro comando “rmdir” para borrar carpetas.

Pero ahora, la función polimorfa del “drag & drop”, facilita enormemente el “diálogo” con la “máquina”, esto es, el empleo y comprensión del funcionamiento del sistema operativo.

Las aplicaciones de hoy en día, deben integrar controles y modos de operación polimorfos de forma extensiva, para poder ingresar y permanecer en los mercados del software, pues los sistemas poco amigables, no tienen ya demanda.

Desde el punto de vista del usuario final, el polimorfismo permite que un sistema resulte “amigable”, es decir, fácil de emplear, de hecho la ergonomía en las aplicaciones se apoya de este principio.

Desde la perspectiva del programador, el polimorfismo es un medio para facilitar la legibilidad y escritura de código, es decir, para comunicar de forma escrita, ordenes generales a otros componentes del sistema, de una forma más sencilla, clara y comprensible.

Como el polimorfismo es una consecuencia directa de la herencia, conlleva a la segunda y más sofisticada forma del mecanismo de reutilización de los componentes escritos del software: la definición de métodos algorítmicos para el procesamiento genérico de objetos de clases “semejantes”.