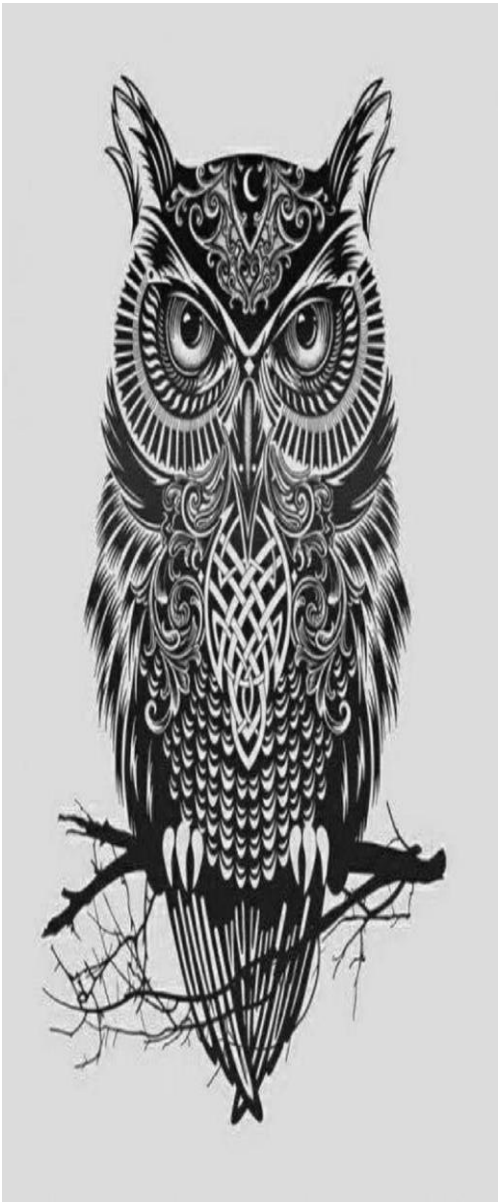
Visión General de los Elementos Básicos para la Programación OO con el Lenguaje Java



Clases y Objetos: Abstracción y Encapsulado en Java

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

Luis Ernesto Rubio.

Objetos

La tecnología Orientada a Objetos (OO), se apoya en los sólidos fundamentos de la ingeniería de sistemas, cuyos elementos reciben el nombre global de Modelo de Objetos. Esta tecnología se basa en la idea de la representación de objetos del "mundo real" y en utilizarlos para construir los sistemas, que estarán organizados en torno a dichos objetos.

En programación de sistemas, los objetos son unidades en la memoria de la computadora, capaces de almacenar información y efectuar procesos; al asociarse unos con otros, los objetos interactúan para llevar a cabo las actividades requeridas en una solución computacional.

Los objetos en una solución de dimensiones industriales, aprovechan sus características (datos), y su comportamiento (métodos), para relacionarse unos con otros (estableciendo enlaces).

Así los objetos logran interactuar de forma armónica y llevan a cabo las tareas que desembocarán en los efectos y resultados con los que las aplicaciones de hoy en día nos asombran, al usuario común, y aún a los mismos profesionales y expertos en tecnologías de frontera.

Programación OO (Orientada a Objetos)

La idea tradicional de programación: "escribir de forma organizada las órdenes que deberá ejecutar la computadora para llevar a cabo las tareas deseadas", resulta limitada, dadas las nuevas necesidades que las aplicaciones de la actualidad imponen.

Por ello, bajo el enfoque de la orientación a objetos, la nueva definición de programación es: "asociar los elementos (objetos), que en base a sus características (atributos/datos/propiedades) y a su comportamiento (métodos/funciones/algoritmos), interaccionan para realizar las tareas deseadas".

A semejanza del mundo real, en donde las agrupaciones sociales, y agregados de recursos, seres, máquinas y entes en general, hacen funcionar a nuestro universo, los objetos impulsan la dinámica del mundo digital.

Orientación a Objetos: Abstracción, Herencia y Polimorfismo

El paradigma OO determina una muy especial aproximación a la forma en que se expresa la solución a un problema. Esta aproximación es justificable, cuando la complejidad del problema es muy alta, y su solución conlleva a un software de dimensiones industriales.

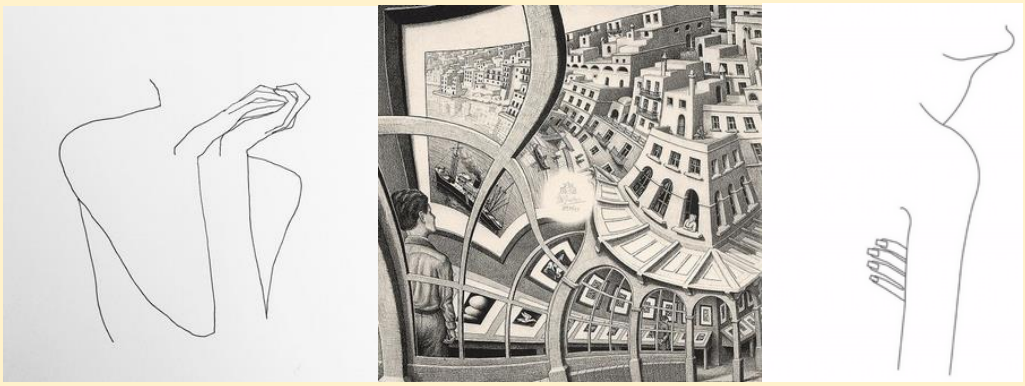
En el paradigma OO, la solución a un problema complejo, se expresa en términos del comportamiento y características de los objetos que, lógicamente enlazados, se asocian para resolverlo.

La dinámica en los sistemas OO, es semejante a la de un organismo vivo, o a una gran corporación industrial, en donde los recursos (u objetos); biológicos (como los órganos y tejidos) o empresariales (tales como los recursos materiales, financieros, humanos y/o tecnológicos), cooperan entre sí para lograr un propósito común; ya sea mantener vivo al organismo, o bien, cumplir con los objetivos y metas corporativas, produciendo bienes y/o servicios útiles a la sociedad.

Son tres los principios que se aplican para expresar soluciones OO: el principio de abstracción, el de herencia y el principio de polimorfismo; que pueden considerarse, como pautas para guían el proceso de construcción de un sistema.

La abstracción

La **abstracción**, constituye el principio de “la representación de objetos”. Permite modelar cómo serán los objetos que intervendrán en la resolución de un problema, mediante moldes o plantillas de código denominados clases (*class* en Java).



La abstracción sirve para representar los enlaces que conectan a los objetos en un sistema, así como también para representar, sus características, o sea los datos de los objetos y el comportamiento de ellos, es decir, los procesos algorítmicos o métodos, con que llevarán a cabo sus tareas.

Ejemplo:

```
class Cuenta{ //Aquí comienza la clase (abstracción): Cuenta

  //A continuación se definen algunos atributos (datos / características / variables):

  int       numeroCuenta;  //Este es el atributo numeroCuenta de tipo int
  double  balance;          //Este es el atributo balance de tipo double

  //A continuación se definen algunos métodos (funciones / órdenes  / comportamiento):

  void deposito(double cantidad){  //Aquí comienza el el método deposito
     balance = balance + cantidad;
  } //Fin del método depósito

  void retiro(double cantidad){  ///Aquí comienza el el método retiro
     balance-=cantidad;
  }//Fin del método retiro

} // Aqui termina la clase Cuenta
```

En el entorno Java, una solución de software, está conformada normalmente por varias clases, una de las cuales, denominada la aplicación, será la encargada de la organización lógica del sistema, determinando la creación de los objetos y el flujo del proceso que se llevará a cabo.

La **aplicación**, es la clase inicial de un sistema en Java, la cual requiere de las otras clases para su funcionamiento. El método principal de una aplicación, que sirve como punto de arranque en la ejecución de un programa, tiene la siguiente firma en Java: *public static void main(String [ ] arguments)* { //code }

Ejemplo:

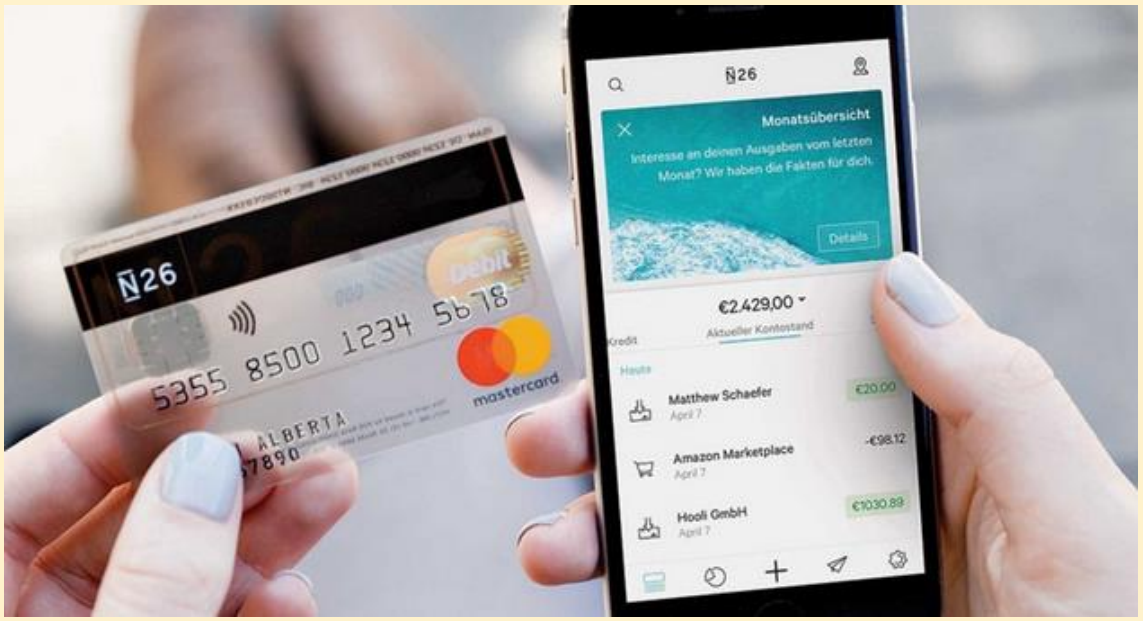
```
class TestCuenta{ //Aplicación (contiene el programa principal que lleva la lógica de la solución)
   public static void main(String [ ] arguments){ //inicia el método principal

      //A continuación se crea un objeto :
      Cuenta miCuenta; //variable referencia de tipo Cuenta (sirve para nombrar el objeto)
      miCuenta = new Cuenta(); //new crea el objeto de la clase Cuenta en memoria

      //A continuación se le dan órdenes al objeto referedo como miCuenta
      //A esto se le conoce como el envío de mensajes:

      miCuenta.deposito(1000.0); //se invoca al método deposito
      miCuenta.retiro(750.0);       //se invoca al método retiro

      //Se despliega en pantalla: El balance de miCuenta es: 250.0 :
      System.out.println("El balance de miCuenta es: " + miCuenta.balance );
   }
}
```



El encapsulado

El **encapsulado** de datos es considerado por algunos autores, otro de los principios de la programación orientada a objetos, e incluso lo enuncian en vez de la abstracción.

El encapsulado es el mecanismo a través del cual se protege y mantiene la integridad de la información (estado) de los objetos, ocultando los atributos, haciéndolos inaccesibles mediante el modificar *private*, y proporcionado métodos de acceso y/o validación que aseguran la consistencia de los datos almacenados en los atributos, a través del modificador *public*.

Ejemplo sin encapsulado:

```
class Cuenta{

   double  balance;

   void deposito(double cantidad){
      balance = balance + cantidad;
   }

   void retiro(double cantidad){
      balance-=cantidad;
   }
}

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

      Cuenta otraCuenta;
      otraCuenta = new Cuenta();
      otraCuenta.deposito(1000.0);
      otraCuenta.retiro(2750.0);   //Se provoca un sobregiro
      otraCuenta.balance+= (-1500.0);  //Mala práctica: se accede directamente al atrbuto balance

      //Se despliega en pantalla :    El balance de otraCuenta es: -3250.0 ; con un balance inconsistente
      System.out.println("El balance de otraCuenta es: " + otraCuenta.balance );

   }
}
```

Ejemplo con encapsulado:

```
class Cuenta{
   /*En esta clase se aplica el Encapsulado:
    *mecanismo que asegura la integridad de los datos de un objeto,
    *haciendo los atributos privados y proporcionando métodos de
    *acceso y validación públicos
    */

   private double  balance;  //atributo oculto, es inaccesible fuera de la clase Cuenta

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

   public boolean retiro(double cantidad){  //método de acceso retiro
      if(balance >= cantidad){  //Valida el contenido del balance
         balance-=cantidad;
         return true;
      } else {
         return false;
      }
   }

   public double getBalance(){
      return balance;
   }

}
```