Módulo de programación orientada a objetos (POO) en Java

Alejandro Héctor González

1. Concepto de Programación Orientada a Objetos

La orientación a objetos surge como alternativa para mejorar algunos inconvenientes de la programación imperativa como ser la falta de portabilidad del código y su escasa reusabilidad, el código estructurado es difícil de modificar, ciclos de desarrollo largos, Técnicas de codificación no intuitivas (Llobet 2009).

El concepto de programación orientada a objetos (POO) es utilizado por diversos lenguajes, por ejemplo SmallTalk, Java, Delphi entre otros. Dado que la POO se basa en la idea natural de la existencia de un mundo lleno de objetos y que la resolución del problema se realiza en términos de objetos, un lenguaje se dice que está orientado a objetos si puede manejar objeto, clases y relaciones de herencia entre ellas como mínimo.

Se entiende paradigma como un conjunto de teorías, estándares y métodos que juntos representan una forma de organizar el conocimiento, una forma de ver el mundo.
La programación orientada a objetos es un paradigma de programación donde los objetos manipulan los datos de entrada para la obtención de datos de salida específicos, donde cada objeto ofrece una funcionalidad especial.

2. Objetos

Podemos definir un objeto como un conjunto complejo de datos y programas que poseen estructura y forman parte de una organización. En este caso las estructuras de datos y los algoritmos usados para administrarlos están encapsulados en una idea común llamada objeto.

Un objeto no es un dato simple, sino que contiene en su interior cierto número de componentes bien estructurados. Cada objeto no es un ente aislado, sino que forma parte de una organización jerárquica o de otro tipo. Un objeto es una instancia particular de una clase. Todos los objetos son instancias de una clase, es decir, ejemplos concretos de la clase

3. Clase

Una clase es la descripción de una familia de objetos que tienen la misma estructura (atributos) y el mismo comportamiento (métodos).

Las clases son el patrón habitual que proporcionan los lenguajes orientados a objetos para definir la implementación de los objetos. Por ejemplo, supongamos que en nuestro sistema detectamos la necesidad de tener personas, por tanto definimos las características comunes de toda persona:

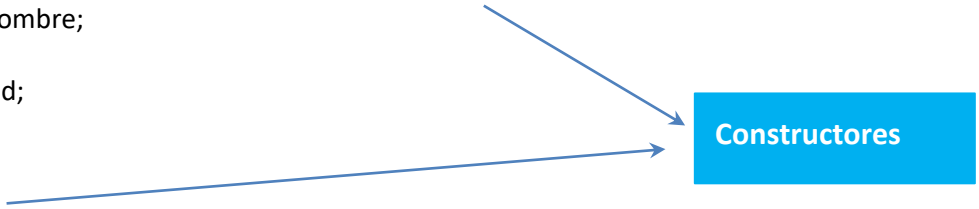
```
public class Persona {
   private String nombre;
   private int DNI;
   private int edad;

 public Persona(String unNombre, int unDNI, int unaEdad){
      nombre = unNombre;
      DNI = unDNI;
      edad = unaEdad;
   }

 public Persona(){
   }

 public int getDNI() {
      return DNI;
   }
   public int getEdad() {
      return edad;
   }
   public String getNombre() {
      return nombre;
   }
   public void setDNI(int unDNI) {
      DNI = unDNI;
   }

   public void setEdad(int unaEdad) {
      edad = unaEdad;
   }
   public void setNombre(String unNombre) {
```



```
        nombre = unNombre;
    }
        public String toString(){
        String aux;
        aux = "Mi nombre es " + nombre + ", mi DNI es " + DNI + " y tengo " + edad + " años.";
        return aux;
    }
}
```

La clase lo define todo, es una entidad autocontenida en cuanto que contiene todo lo necesario para definir la estructura o los datos (atributos) que cada objeto tendrá y la manera de manejarse esos objetos a través de los métodos.

Tener en cuenta que se debe anteponer a la declaración de cada atributo la palabra ***private*** para lograr ocultamiento de la información.

¿Cómo crear un objeto a partir de una clase?

1.**Declarar variable para mantener la referencia**:

NombreDeClase miVariable;

2.**Enviar a la clase el mensaje de creación y guardar referencia**: Debemos realizar la instanciación de la clase utilizando el constructor. Si tiene más de uno puedo utilizar el que mejor se adapte a lo que necesito.

miVariable= new NombreDeClase(valores para inicialización);

Estos dos pasos pueden realizarse en un solo paso de la siguiente manera:

NombreDeClase miVariable= new NombreDeClase(alores para la inicialización);

Ejemplo 1 de uso de constructores:

```
/**   Asigno valores por teclado a las variables que necesito para el objeto    **/
Scanner in = new Scanner (System.in);
System.out.println("Ingrese nombre: ");
String nom = in.next();
System.out.println("Ingrese dni: ");
int dni = in.nextInt();
System.out.println("Ingrese edad: ");
int edad = in.nextInt();
/**   Crea una referencia a una persona con 3 atributos    **/

 Persona p= new Persona(nom,dni,edad);
```

o

```
/**  Crea una referencia a un objeto persona sin atributos cargados  **/
Persona pnue = new Persona();

/**  Agregamos luego los 3 atributos **/
pnue.setNombre(nom);
pnue.setEdad(edad);
pnue.setDNI(dni);
```

Ejemplo 2 creación de un vector de objetos:

¿Cómo puedo cargar un vector de 6 elementos con objetos Persona?

```
int N=6;
Persona personas[]=new Persona[N];
Scanner in = new Scanner (System.in);
int i;
for (i=0;i<N;i++){
   System.out.println("Ingrese nombre: ");
   String nom = in.next();
   System.out.println("Ingrese dni: ");
   int dni = in.nextInt();
   System.out.println("Ingrese edad: ");
   int edad = in.nextInt();
   Persona p= new Persona(nom,dni,edad);
   personas[i]=p;
}
```

¿Qué ocurre durante la creación del objeto?

a.**Alocación de Memoria.** Las variables de instancia se inicializan a valores por defecto.

b.**Inicialización Explícita** (si hubiese) de las variables de instancia.

c.**Ejecución del Constructor** (código para inicializar variables de instancia con los valores que enviamos en el mensaje de creación).

d.**Asignación de la referencia a la variable**.

Para pensar:

¿Qué estamos haciendo en esta línea, cúal es la clase y cúal es el objeto?
Scanner in = new Scanner (System.in);

4. Método

Es una subrutina o módulo cuyo código es definido en una clase y puede pertenecer tanto a una clase, como es el caso de los métodos de clase o estáticos, como a un objeto, como es el caso de los métodos de instancia.

Tipos de métodos:

public hace que una declaración sea accesible por cualquier clase.
protected hace una declaración accesible por cualquier subclase de la clase que se declara, o a cualquier clase dentro del mismo paquete.
private hace una declaración accesible sólo dentro de la clase en que se declara.

Si no se provee ninguna de estas tres palabras clave, se dice que la declaración tiene accesibilidad por defecto (default accesibility), lo que significa que es accesible por cualquier otra clase dentro del mismo paquete.

Los métodos de retorno vacío (o métodos void) se declaran con tipo de retorno void.

El cuerpo de un método void simplemente realiza un procesamiento, que tendrá el efecto colateral de cambiar el estado del objeto para el cual fue invocado, y termina sin explícitamente retornar un valor.

Parámetros:

Únicamente se utiliza pasaje por valor

Parámetro dato primitivo

•Copia del parámetro actual .

•Si se modifica el parámetro formal, no altera el parámetro actual.

Parámetro objeto

•Copia de la referencia al objeto pasado como parámetro actual.

•Si se modifica el estado interno del objeto, el cambio es visible fuera.

•Si se modifica la referencia, el parámetro actual sigue referenciando al mismo objeto.

Return

La palabra return puede aparecer en cualquier parte del cuerpo de un método y en múltiples lugares. Un método no-vacío debe contener al menos un enunciado return.

El siguiente ejemplo incluye un método min que retorna el mínimo de dos argumentos:

```
public class EjemploReturn {
  private static int min (int a, int b) {
    if (a < b) return a;
    else return b; }
   public static void main( String[] args) {
      System.out.println("El menor de 5 y 8 es: " + min(5,8));
   }
}
```

5. Constructores

Un constructor se declara como cualquier método, contando con una lista de parámetros.
Sin embargo, la declaración de un constructor no tiene tipo de retorno, y su nombre debe coincidir con el nombre de la clase dentro de la cual es declarado.
El constructor tiene la siguiente forma:
Nombre del constructor (listadeparámetros) {
Secuencia de enunciados
}
Un constructor puede ser declarado opcionalmente como public, protected o private.
Puede haber varios constructores para la clase (se lo conoce como sobrecarga). Java identifica cuál está siendo invocado por el número y tipo de sus parámetros.

Recomendación: siempre incluir un constructor sin parámetros.

6. Pautas para una buena programación

Convención para nombres: comenzar con minúscula, luego cada palabra en mayúscula.
Por convención, en Java se utilizan las siguientes reglas para identificadores:
Clases e interfaces: Mayúsculas inicial cada palabra
Ejemplo:

class MiClaseRectangulo {...}

Miembros, variables y parámetros: Minúsculas la primera palabra, mayúsculas las siguientes
Ejemplo:

void escribeEntero(int entero) {...}
int numLinea;

Constantes: Todo mayúsculas (con subrayados entre palabras)
Ejemplo

final double PI = 3.141592;
final int VELOCIDAD_LUZ = 299792458;

Bibliografía

Barnes, David; Kölling, Michael (2013). *Programación orientada a objetos con JAVA usando BlueJ*. Editorial Prentice Hall .

Booch Grady. (1996). *Análisis y Diseño Orientado a Objetos con Aplicaciones*. Editorial Addison Wesley.

Llobet Azpitarte Rafael, et al (2009). *Introducción a la Programación Orientada a Objetos con Java.* Primera edición.Rev. 1.0.1.. Departamento de Sistemas Informáticos y Computación. Universidad Politécnica de Valencia. ISBN: 978-84-613-0411-0

Lopez Roman Leobardo (2011). *Programación estructurada y orientada a objetos*. Editorial: Alfa Omega Grupo editor

Román Martínez, Elda Quiroga (2002). *Estructuras de datos: referencia práctica con orientación a objetos***.** Editorial Thomson