Fundamentos de la programación Orientada a Objetos

Definición Puntual:

Es un paradigma de programación basado en la jerarquía de clases y objetos bien definidos. Dicha terminología orientada a objetos está en constante evolución, muchos libros y profesionales dan diversas definiciones, pero básicamente debes comprender algunas cosas puntuales acerca de dicho paradigma.

Todo el escenario que se puede encontrar en la programación Orientada a Objetos envuelve los siguientes conceptos:

1. Abstracción
2. Encapsulamiento
3. Modularidad
4. Herencia
5. Polimorfismo
6. Persistencia
7. Identidad

Definiremos a continuación algunos conceptos fundamentales del paradigma.

Clases:

Una clase se define como una estructura que define los datos y los métodos para trabajar con esos datos. En Java, por ejemplo, todo el código de un programa está escrito en clases, ya sea que sea codificada por el programador o por ejemplo usando las librerías API de la plataforma de java. Para los primeros ejemplos usados en el anterior módulo, hemos generado en consola algunos mensajes, dicha tarea ha sido posible ya que hemos hecho uso dela clase java.lang.System para imprimir mensaje en consola.

```
class ExampleProgram
{
	public static void main(String[]args)
	{
		System.out.println("Soy un programa ejemplo de clase");
	}
}
```

Booch: "*Una clase es un set de objetos que comparten características y comportamientos similares".*

Tomando como base ese concepto, veamos la definición de una clase en PHP, la cual contiene un conjunto de propiedades y métodos que definirán su comportamiento:

```
<?php
class ClaseSencilla
{
   // Declaración de una propiedad
   public $var = 'un valor predeterminado';
   // Declaración de un método
   public function mostrarVar() {
      echo $this->var;
   }
}
?>
```

Una clase consiste principalmente en dos cosas:

Una interfaz y una implementación. La interfaz es la parte visible para el mundo exterior, y en la implementación definimos el código que representa el comportamiento de la clase.

Objetos:

Una instancia es una copia ejecutable de una clase, y una instancia es, un **objeto.** Los objetos nos permiten hacer referencia a definiciones de clase, podemos crear cualquier número de ellos al mismo tiempo para una clase determinada. Identifiquemos algunos objetos en el siguiente bloque de código:

```
class ExampleProgram
{
        public static void main(String[] args)
        {
                String text = newString("I'm a simple Program ");
                System.out.println(text);
                String text2 = text.concat("that uses classes andobjects");
                System.out.println(text2);
        }
}
```

Identificamos entonces algunos tipos de objetos String, en primer lugar, la declaración siguiente nos da uno:

*String **text** = new String ("I'm a simple Program ");*

El objeto **text** es creado para almacenar una cadena de texto.

El segundo objeto creado en el bloque es text2:

```
String text2 = text.concat("that uses classes and objects");
```

Herencia

En programación Orientada a objetos la característica que nos permite hacer trabajar a los objetos de forma conjunta y definir una relación entre clases es la herencia. Las clases en java descienden de *java.lang.Object* e implementa sus métodos.

Básicamente el objetivo fundamental de la herencia, hace posible la reutilización de métodos y atributos ya existentes. Una clase que es heredada de otra se la llama **subclase**, y esa clase de la cual las subclases son derivadas se llama **superclase**.

Ejemplo de Herencia:

```
public class Bicycle
{
        // Clase bicycle tiene tres atributos
        public int cadence;
        public int gear;
        public int speed;

        // constructor para la clase Bicycle
        public Bicycle(int startCadence, int startSpeed, int startGear)
        {
           gear = startGear;
           cadence = startCadence;
           speed = startSpeed;
        }

        // Métodos para la clase
        public void setCadence(int newValue)
        {
           cadence = newValue;
        }
```

```
    public void setGear(int newValue)
    {
       gear = newValue;
    }

    public void applyBrake(int decrement)
    {
       speed -= decrement;
    }

    public void speedUp(int increment)
    {
       speed += increment;
    }
}
```

Ahora, como podemos heredar la anterior clase y darle uso a los atributos creados en ella, en la clase siguiente, heredamos de Bicycle.

Creamos otra clase que defina a una bicicleta montañesa, en la cual podemos agregar otros atributos diferentes a los de la superclase pero heredando todos sus atributos y métodos.

```
public class MountainBike extends Bicycle
{
    // Agrega una característica más
    public int seatHeight;

    // Constructor
    public MountainBike(int startHeight, int startCadence, int startSpeed, int startGear)
    {
       super(startCadence, startSpeed, startGear);
       seatHeight = startHeight;
    }

    // Y un método más
    public void setHeight(int newValue)
    {
       seatHeight = newValue;
    }
}
```

Encapsulamiento

La idea fundamental del encapsulamiento es poder mantener de forma privada los detalles de una clase, lo que hace imposible que los clientes que hacen uso de dicha clase, conozcan sus detalles.

Algunas de las razones para aplicar encapsulamiento es para mantener el código de forma modular, que se hace básicamente con dos piezas separadas para cada clase, una implementación y una interfaz.

Modularidad

La modularidad o programación modular es una técnica de diseño de software que enfatiza la separación de la funcionalidad de un programa en módulos independientes e intercambiables, de manera que cada uno contenga todo lo necesario para ejecutar solo un aspecto de la funcionalidad deseada.

La programación modular está estrechamente relacionada con la programación estructurada y la programación orientada a objetos, todas con el mismo objetivo de facilitar la construcción de grandes programas de software y sistemas por descomposición en piezas más pequeñas, "divide y vencerás".

Polimorfismo

El polimorfismo es la capacidad para interpretar objetos de diferentes maneras en función de su clase o tipo de datos. En esencia, es la capacidad de aplicar un único método a las clases derivadas y lograr un resultado adecuado.

Tipos

1. Polimorfismo **ad hoc** es cuando una función o método se implementa de manera diferente, dependiendo de un número limitado de tipos especificados y combinaciones de parámetros de entrada. Un ejemplo de polimorfismo ad hoc es la sobrecarga de funciones.
2. Polimorfismo **paramétrico** es cuando el código se escribe sin ninguna especificación de tipo, y así se puede usar con cualquier cantidad de tipos diferentes especificados más adelante. En la programación orientada a objetos, esto a menudo se llama programación genérica.
3. Polimorfismo de **inclusión**, también conocido como sub-tipificación, se produce cuando un solo nombre puede referirse a instancias de cualquier cantidad de clases diferentes, siempre que compartan la misma superclase.

Los resultados son diferentes en función de los parámetros de entrada, aunque el nombre de clase y método invocado sea el mismo.

1 Instancias y objetos

1.1 Clases y métodos

1.1.1 Métodos

La declaración de métodos en Java se hace de la siguiente manera:

```
public double calcularPromedio(double val1, double val2, double val3)
{
   //Bloque de código para el promedio
}
```

Componentes en la declaración de métodos.

1. **Modificadores:** public - private
2. **Tipo de retorno:** El tipo de dato devuelto por el método, dependiendo de la operación o función que realice su bloque de código.
3. **Nombre del método:** El nombre que lo identifica.
4. **Los parámetros dentro de paréntesis:** Definir los parámetros de entrada que servirán para condicionar elementos en el bloque, separados por coma.
5. **Una lista de excepción**.
6. **El cuerpo del método:** El bloque que define la lógica, declaración de variables, condicionantes, etc.

Sobrecarga de métodos.

Java soporta la sobrecarga de métodos, significa que podemos tener varios métodos en una clase, con parámetros de entrada que sean de diferente tipo. Veamos un ejemplo muy claro para entender la sobrecarga:

```
public class Dibujos
{
        ...
        public void dibujar(String s)
        {
            ...
        }
        public void dibujar(int i)
        {
                ...
        }
        public void dibujar(double f)
        {
                ...
        }
        public void dibujar(int i, double f)
        {
                ...
        }
}
```

Tenemos en la clase, el mismo nombre de método, pero con diferentes tipos de datos en sus parámetros.

Ejemplo de uso de métodos:

Podemos crear métodos que realicen diferentes funciones en nuestras clases, veamos algunos ejemplos sencillos:

```
 public class Program
{
        public static void main(String arg[ ])
        {
            Mensajes mensajes = new Mensajes(); //Instancia de la clase Mensajes
            mensajes.Saludo();              //Llamar los métodos en esa clase por medio de objeto mensajes
            mensajes.dibujar(10);            //Llamar el segundo método
        }

        public static class Mensajes
        {
                //Método que no recibe parámetro de entrada ni retorna valor
                public void Saludo()
                {
                        System.out.println("Hola a todos");
                }

                //Método que  recibe parámetro entero para imprimir un valor
                public void dibujar(int edad)
                {
                        if (edad > 18)
                        {
                                System.out.println("Eres mayor de edad");
                        }
                        Else
                        {
                                System.out.println("Eres menor de edad");
                        }
                }
        }
 }
```

En el anterior programa tenemos una clase llamada Mensajes que contiene dos métodos que hacen una tarea determinada, imprimir algunos valores en pantalla. **Un objeto es una instancia de una clase**

Convenciones de nombres en la programación orientada a objetos:

Cuando escribimos códigos de programas, es importante que el todo lo que estamos generando sea limpio y ordenado, eso ayuda a que tengamos una estructura de código fácil de entender, para que otros programadores que den continuidad a los proyectos puedan entender la lógica de aplicación, y en general, para seguir los estándares internacionales de programación. Los siguientes son dos opciones de nomenclatura de clases, objetos, variables de instancia y métodos.

Nomenclatura 1:

Para los nombres de clase:

Iniciamos nombrando la clase con mayúscula, y en caso de ser dos palabras o más, las colocamos de forma consecutiva, siempre iniciando cada una de ellas con mayúscula, ejemplos:

- Empleados
- Estudiantes
- NumeroCuenta
- NombreEstudiante
- EstadoCivilEstudiante
-

Objetos, instancias y métodos:

Siguiendo la misma línea para las clases, nombres concatenados, a diferencia que vamos a declarar los nombres de estos elementos iniciando con una letra minúscula, todas las demás palabras serán con letra mayúscula.

Algunos ejemplos de esas declaraciones serían:

- empleados
- estudiantes
- listas
- numeroCuenta
- nombreEstudiante
- estadoCivilEstudiante
- obtenerNombres

Nomenclatura 2

Clases

Para nombrar las clases usaríamos la misma estructura anterior, iniciando con letra mayúscula para cada palabra que componga el nombre:

- Empleados
- Estudiantes
- NumeroCuenta

Ahora, para definir los nombres de los objetos, lo haríamos de la siguiente manera, palabras en:

- empleados
- estudiantes
- numeros_cuenta
- listas_matricula

Para declarar una instancia: cada palabra debe iniciarse con minúscula y debe separarse por guion bajo.

- cuentas_estudiante
- nombres_campus
- direcciones_ciudades

El siguiente es un ejemplo de la documentación Java del uso delas anteriores nomenclaturas:

```
public class Bicycle
{
        // **Definición de campos**
        // **3 variables enteras**
        public int cadence;
        public int gear;
        public int speed;

        // **Los métodos para la clase definidos**
        //**La nomenclatura para los métodos usada es nombreMetodo, la primera en
        minúscula y la segunda palabra en mayúscula.**

        public void setCadence(int newValue)
        {
                cadence = newValue;
        }

        public void setGear(int newValue)
        {
                gear = newValue;
        }

        public void applyBrake(intdecrement)
        {
                speed -= decrement;
        }

        public void speedUp(intincrement)
        {
                speed += increment;
        }
}
```

El ejemplo anterior muestra la estructura que deberíamos seguir para nombrar los nombres de los elementos.