Introducción a la Programación Orientada a Objetos (POO)

La Programación Orientada a Objetos (POO) es un paradigma de programación que se centra en organizar el código en torno a objetos que representan entidades del mundo real o conceptual. A continuación, exploraremos los conceptos fundamentales de este enfoque y los beneficios que aporta al desarrollo de software.

1. ¿Qué es la Programación Orientada a Objetos?

- Es un paradigma que permite estructurar el código en torno a **clases** y **objetos**.
- Los objetos combinan **datos** (atributos) y **comportamientos** (métodos) en una sola entidad.
- Este enfoque refleja la forma en que percibimos y modelamos el mundo real, facilitando la comprensión y mantenimiento del código.

2. Conceptos básicos

2.1 Clase

- Una clase es un modelo o plantilla que define las propiedades (atributos) y acciones (métodos) comunes a un conjunto de objetos.
- Ejemplo:

```
public class Coche {
    // Atributos
    String marca;
    String color;

    // Constructor
    public Coche(String marca, String color) {
        this.marca = marca;
        this.color = color;
    }

    // Método
    public void acelerar() {
        System.out.println("El coche está acelerando.");
    }
}
```

2.2 Objeto

- Un objeto es una instancia de una clase. Representa un elemento específico del mundo real o conceptual definido por la clase.
- Ejemplo:

```
Coche miCoche = new Coche("Toyota", "Rojo");
System.out.println("Marca: " + miCoche.marca);
System.out.println("Color: " + miCoche.color);
miCoche.acelerar();
```

2.3 Atributos

- Son las propiedades que describen el estado o las características de un objeto.
- En el ejemplo anterior, `marca` y `color` son atributos de la clase `Coche`.

2.4 Constructores

- Los constructores son métodos especiales que se ejecutan cuando se crea un objeto.
- Tienen el mismo nombre que la clase y no tienen un tipo de retorno.
- Se utilizan para inicializar los atributos de un objeto.
- Ejemplo:

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

    // Constructor
    public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }

    // Método para mostrar información
    public void mostrarInformacion() {
        System.out.println("Nombre: " + nombre + ", Edad: " + edad);
    }
}

// Crear un objeto utilizando el constructor
Persona persona1 = new Persona("Ana", 30);
persona1.mostrarInformacion();
```

2.5 Métodos

- Son las acciones que un objeto puede realizar.
- En el ejemplo, el método `acelerar()` representa una acción que un coche puede realizar.

2.6 Encapsulamiento

- Consiste en proteger los datos de un objeto para que sólo puedan ser accedidos o modificados a través de métodos específicos.
- Para ello, los atributos se declaran como **privados** y se crean métodos públicos llamados **getters** y **setters** para acceder a ellos.

Ventajas del encapsulamiento:

1. **Protección de datos sensibles:** Evita accesos indebidos o modificaciones directas a los atributos.
2. **Control sobre los datos:** Permite validar o transformar los datos antes de asignarlos a los atributos.
3. **Mantenimiento fácil:** Los cambios en la implementación interna de los atributos no afectan a las partes del código que los utilizan.

4. **Reutilización y modularidad:** Los métodos públicos pueden ser reutilizados en diferentes contextos, proporcionando un punto centralizado para acceder o modificar los datos.

Getters

- Son métodos que permiten obtener el valor de un atributo.
- Ejemplo:

```
public String getMarca() {
    return marca;
}
```

Setters

- Son métodos que permiten modificar el valor de un atributo.
- Ejemplo:

```
public void setMarca(String marca) {
    this.marca = marca;
}
```

Ejemplo completo de encapsulamiento:

```
public class Coche {
    // Atributos privados
    private String marca;
    private String color;

    // Constructor
    public Coche(String marca, String color) {
        this.marca = marca;
        this.color = color;
    }

    // Getter y Setter para marca
    public String getMarca() {
        return marca;
    }

    public void setMarca(String marca) {
        this.marca = marca;
    }

    // Getter y Setter para color
    public String getColor() {
        return color;
    }

    public void setColor(String color) {
        this.color = color;
    }

    // Método para mostrar información
    public void mostrarInformacion() {
        System.out.println("Marca: " + marca + ", Color: " + color);
    }
```

```
}

// Uso de la clase encapsulada
public class Main {
    public static void main(String[] args) {
        Coche coche1 = new Coche("Toyota", "Rojo");

        // Accediendo a los atributos mediante getters y setters
        System.out.println("Marca inicial: " + coche1.getMarca());
        coche1.setColor("Azul");
        coche1.mostrarInformacion();
    }
}
```

Con este enfoque, los atributos `marca` y `color` están protegidos del acceso directo y sólo pueden ser manipulados de manera controlada mediante los métodos definidos.

- Consiste en proteger los datos de un objeto para que sólo puedan ser accedidos o modificados a través de métodos específicos.
- Para ello, los atributos se declaran como **privados** y se crean métodos públicos llamados **getters** y **setters** para acceder a ellos.

3. Principios fundamentales de la POO

3.1 Abstracción

- Permite enfocarse en los aspectos relevantes de un objeto, ignorando los detalles innecesarios.
- Ejemplo: Un coche tiene muchas partes, pero sólo nos interesa modelar `marca` y `color` para una aplicación específica.

4. Beneficios de la POO

- **Reutilización del código:** La organización en clases permite usar el mismo código en diferentes proyectos.
- **Mantenimiento sencillo:** Los cambios se concentran en clases específicas.
- **Organización y modularidad:** Los programas son más fáciles de entender y modificar.

5. Ejemplo práctico completo

```
// Clase Persona
public class Persona {
    private String nombre;
    private int edad;

    // Constructor
    public Persona(String nombre, int edad) {
```

```
        this.nombre = nombre;
        this.edad = edad;
    }

    // Métodos
    public void saludar() {
        System.out.println("Hola, mi nombre es " + nombre + ".");
    }

    public void cumplirAnios() {
        edad++;
        System.out.println("Ahora tengo " + edad + " años.");
    }
}

// Programa principal
public class Main {
    public static void main(String[] args) {
        Persona persona1 = new Persona("Juan", 25);
        persona1.saludar();
        persona1.cumplirAnios();
    }
}
```