Informe del modelo conceptual orientado a objetos

Presentado a:

Inst. Millerlandy Becerra Chávez

Presentado por:

Carlos José Delgado González

Durley Sandrith Galván Jiménez

Yasser Leonardo Pacheco Cañizares

Programa:

PROGRAMACION DE APLICACIONES PARA DISPOSITIVOS MOVILES (2977832)

SENA - Servicio Nacional de Aprendizaje

2024

Introducción

La programación orientada a objetos (POO) se fundamenta en la creación de modelos que representan el problema a resolver dentro de los programas. Este enfoque de programación reduce los errores y fomenta la reutilización del código. La POO, conocida también como OOP por sus siglas en inglés, es un paradigma que se basa en "objetos", los cuales contienen datos en forma de campos (atributos o propiedades) y métodos (código).

Los objetos pueden interactuar y modificar los valores de sus campos a través de sus métodos. Muchos lenguajes de programación modernos permiten la agrupación de objetos en bibliotecas, y también ofrecen la posibilidad de que los usuarios creen sus propias bibliotecas. Las características clave de la POO incluyen herencia, cohesión, abstracción, polimorfismo, acoplamiento y encapsulamiento.

Conceptos Fundamentales de la Programación Orientada a Objetos

De Procedural a Orientado a Objetos

Lenguajes de programación antiguos como C, Basic o COBOL seguían un estilo procedimental, donde el código consistía en una serie de instrucciones ejecutadas secuencialmente. Estos lenguajes utilizaban procedimientos (subrutinas o funciones) para encapsular funcionalidad, pero dependían de datos globales, dificultando el aislamiento de datos específicos. Este enfoque se centraba más en la lógica que en los datos.

En contraste, lenguajes modernos como C#, Java y otros utilizan paradigmas diferentes, siendo el más popular la Programación Orientada a Objetos (POO).

Principales Conceptos de la POO

Clases, Objetos e Instancias

El primer y más importante concepto de la POO es la distinción entre clase y objeto.

- **Clase:** Es una plantilla que define de manera genérica cómo serán los objetos de un tipo específico. Por ejemplo, una clase Persona puede tener atributos como Nombre, Apellidos y Edad, y métodos como Hablar(), Caminar() y Comer().
- **Objeto:** Es una instancia concreta de una clase. Al instanciar una clase, se crea un objeto real con existencia en memoria. Por ejemplo, un objeto de la clase Persona podría ser una entidad llamada Cristina López, de 37 años, capaz de hablar, caminar o comer.

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

    void hablar() { }
    void caminar() { }
    void comer() { }
}
```

Los Cuatro Pilares de la POO

Para manejar eficientemente las clases y objetos generados con la POO, es necesario comprender cuatro principios fundamentales que ayudan a reducir la complejidad y evitar problemas.

1. Encapsulación

El concepto de encapsulamiento, aunque básico, es fundamental en la programación orientada a objetos (POO) y, a menudo, puede pasarse por alto debido a su simplicidad.

La encapsulación es una característica esencial en los lenguajes de POO que permite que toda la información relevante a un objeto permanezca dentro de él. Esto significa que los datos y detalles internos de un objeto están "encapsulados" y solo pueden ser accedidos mediante las propiedades y métodos definidos por la clase.

Por ejemplo, en una clase que representa a una persona, toda la información, como el nombre, los apellidos, y la edad, se mantiene dentro del objeto persona. Solo se puede acceder a estos datos a través de las propiedades públicas o métodos específicos proporcionados por la clase. Así, para obtener el nombre de una persona, se utilizaría una propiedad pública llamada Nombre.

En los lenguajes de programación tradicionales, sería necesario crear una estructura global para almacenar dicha información y luego acceder a ella de manera ordenada, evitando mezclar los datos de una persona con los de otra. En cambio, gracias a la encapsulación en POO, toda la información de un objeto está contenida y gestionada dentro del propio objeto.

Definición de Encapsulación

- **Encapsulación**: La encapsulación implica que un objeto agrupa tanto sus características (datos) como su comportamiento (métodos) en una unidad indivisible.
- **Objeto** = Características + Comportamiento
- **Objeto** = Información + Proceso
- **Objeto** = Atributos + Métodos

En resumen, el encapsulamiento se refiere a la consideración de un objeto como una unidad completa e indivisible que contiene tanto sus atributos como sus métodos, permitiendo así una gestión más eficiente y segura de los datos.

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

    public String getNombre() { return nombre; }
    public void setNombre(String nombre) { this.nombre = nombre; }
    // Otros getters y setters...
}
```

2. Abstracción

El principio de abstracción, como su nombre lo sugiere, implica que una clase debe representar las características visibles de una entidad, ocultando la complejidad interna asociada. Es decir, la abstracción nos permite utilizar una serie de atributos y comportamientos (propiedades y métodos) sin preocuparnos por los detalles internos de su implementación.

Una clase (y los objetos creados a partir de ella) debe exponer solo lo necesario para su uso. La forma en que se realizan las operaciones internamente no es relevante para los programas que utilizan estos objetos.

Por ejemplo, en una clase que representa a una persona en un juego, puede existir un atributo interno llamado "energía" que no es accesible directamente desde fuera de la clase. Cada vez que la persona anda o corre, este valor disminuye. Cuando la persona come, el valor de energía aumenta según la cantidad de comida ingerida.

Otro ejemplo claro podría ser el método hablar(). Este método podría generar una voz sintética a partir de un texto proporcionado como parámetro. Para lograr esto, pueden ocurrir varias operaciones internas: se podría llamar a un componente de síntesis de voz en la nube, lanzar la síntesis de voz en un dispositivo local y registrar la frase pronunciada en una base de datos para mantener un historial. Sin embargo, para el programa que llama al método hablar(), todos estos detalles son irrelevantes. El programa solo necesita llamar al método hablar() en un objeto, sin conocer la complejidad interna del proceso. Si en el futuro se cambia la forma de sintetizar la voz o cualquier otra acción interna, no afectará al programa que utiliza nuestros objetos de tipo Persona.

La abstracción está estrechamente relacionada con la encapsulación, pero va un paso más allá. No solo controla el acceso a la información, sino que también oculta la complejidad de los procesos implementados, permitiendo a los usuarios interactuar con objetos de manera sencilla y eficiente, sin necesidad de conocer los detalles internos.

```java
public class Persona {
    private int energia;

    public void caminar() {
        energia -= 10;
    }

    public void comer() {
        energia += 20;
    }
}
```

3. Herencia

En genética, se dice que una persona hereda ciertos rasgos de sus padres, como el color de los ojos o la piel, o incluso ciertas enfermedades genéticas. De manera similar, en la programación orientada a objetos (POO), cuando una clase hereda de otra, adquiere todos los atributos y métodos de la clase original.

Una clase actúa como un molde que define cómo es y cómo se comporta una entidad. Cuando una clase hereda de otra, adopta todos los atributos y comportamientos de la primera, pero también puede agregar nuevos y modificar los existentes. La clase de la que se hereda se llama clase base, mientras que la clase que hereda se llama clase derivada.

Por ejemplo, en un juego con personajes, podemos crear clases especializadas a partir de una clase general llamada Persona. Clases como Pirata, Piloto o Estratega heredarían de Persona. Todos los objetos de estas clases heredarían los atributos y métodos de Persona, pero podrían tener características y comportamientos adicionales únicos.

Por ejemplo, los objetos de la clase Pirata podrían tener un método adicional llamado abordar() para asaltar barcos enemigos y una propiedad exclusiva llamada sobrenombre, que es un apodo por el cual se les conoce. Un pirata podría llamarse Hızır bin Yakup, pero su sobrenombre sería Barba Roja.

Además, la herencia permite reutilizar el código de la clase base. Supongamos que los piratas en el juego tienen una forma particular de hablar. Podríamos modificar el método hablar() para que añada frases típicas de piratas, como "¡Arrrr!" o "¡Por todos los demonios!" a sus diálogos. No sería necesario reescribir todo el código del método hablar() porque la clase Pirata ya hereda esta funcionalidad de la clase Persona. Solo necesitaríamos agregar las frases adicionales y delegar el resto del trabajo a la clase base. Esto nos proporciona consistencia y facilita la personalización de comportamientos.

La herencia es una de las características más poderosas de la POO porque promueve la reutilización del código y permite la especialización y personalización de clases de manera eficiente.

```
public class Pirata extends Persona {
    private String sobrenombre;

    public void abordar() { }
}
```

4. Polimorfismo

El término polimorfismo proviene del griego "polys" (muchos) y "morfo" (forma), y significa "capacidad de tomar muchas formas". En programación orientada a objetos (POO), polimorfismo se refiere a la habilidad de usar objetos de diferentes clases, con una base común, de manera intercambiable, sin necesidad de conocer su clase específica.

Imaginemos un juego con diversos personajes compartiendo el mismo escenario: varios piratas, algunos estrategas y otros tipos de personas. En un momento determinado, necesitamos que todos hablen. Cada personaje tiene su propia forma de hablar, lo que podría complicar la implementación si tuviéramos que identificar cada tipo de personaje por separado para hacerlos hablar. La idea detrás del polimorfismo es tratar a todos estos personajes como instancias de la clase Persona y simplemente invocar el método hablar() en cada uno, sin importar su tipo específico.

Al derivar todos de la clase Persona, todos los personajes tienen el método hablar(). Cuando este método es llamado, cada personaje hablará de acuerdo a su tipo específico: los piratas usarán expresiones adicionales como "¡Arrrr!", los pilotos podrían decir "Entrando en pista", y los estrategas agregarán "Déjame que lo piense bien". Todo esto sucede de manera transparente para el programador. Esto es el polimorfismo.

El polimorfismo puede ser más complejo, ya que también incluye la sobrecarga de métodos y el uso de interfaces. Sin embargo, el concepto básico es que permite utilizar objetos de manera generalizada, mientras que internamente se comportan según su tipo específico.

```
public class Juego {
    public void hacerHablar(Persona persona) {
        persona.hablar();
    }
}
```

Ejemplo de clases en POO

Encapsulación: Clase “Cuenta Bancaria”

```java
public class CuentaBancaria {
    private double saldo;

    public CuentaBancaria(double saldoInicial) {
        this.saldo = saldoInicial;
    }

    public double getSaldo() {
        return saldo;
    }

    public void depositar(double cantidad) {
        if (cantidad > 0) {
            saldo += cantidad;
        }
    }

    public void retirar(double cantidad) {
        if (cantidad > 0 && cantidad <= saldo) {
            saldo -= cantidad;
        }
    }
}
```

Abstracción: Clase “Vehículo”

```java
abstract class Vehiculo {
    abstract void arrancar();
    abstract void detener();
}

class Coche extends Vehiculo {
    @Override
    void arrancar() {
        System.out.println("El coche arranca.");
    }

    @Override
    void detener() {
        System.out.println("El coche se detiene.");
    }
}
```

Herencia: Clase “animal” Subclase “perro”

```java
class Animal {
    void hacerSonido() {
        System.out.println("El animal hace un sonido.");
    }
}

class Perro extends Animal {
    @Override
    void hacerSonido() {
        System.out.println("El perro ladra.");
    }
}
```

Polimorfismo: Clase “animal”

```java
class Animal {
    void hacerSonido() {
        System.out.println("El animal hace un sonido.");
    }
}

class Perro extends Animal {
    @Override
    void hacerSonido() {
        System.out.println("El perro ladra.");
    }
}

class Gato extends Animal {
    @Override
    void hacerSonido() {
        System.out.println("El gato maúlla.");
    }
}
```

```java
public class Main {
    public static void main(String[] args) {
        Animal[] animales = {new Perro(), new Gato()};

        for (Animal animal : animales) {
            animal.hacerSonido();
        }
    }
}
```