UNIVERSIDAD TECNOLÓGICA DEL PERÚ

Facultad de Ingeniería

Programación Orientada a Objetos

Sesión 9: Diseño de diagrama de clases del UML usando herencia simple y herencia múltiple

Recordando:

¿Que vimos la clase pasada?

Universidad Tecnológica del Perú

Logro de aprendizaje

Al finalizar la sesión, el estudiante comprende la naturaleza y propósito de las clases abstractas e interfaces en la solución de problemas usando Java con ejemplos prácticos.

Agenda

- Introducción a las clases abstractas
- Definición y características de las clases abstractas.
- Diferencias clave entre clases abstractas e interfaces.

Universidad Tecnológica del Perú

Saberes previos

- Concepto de programación orientada a objetos (POO).
- Definición de clases y objetos.
- Encapsulamiento, herencia y polimorfismo.

¿Qué es una clase abstracta?

- Una clase abstracta es un tipo especial de clase en la programación orientada a objetos que no puede ser instanciada directamente. Está diseñada para ser una clase base de la cual otras clases pueden heredar.
- La principal razón para usar clases abstractas es proporcionar una base común para otras clases, definiendo ciertos comportamientos (métodos) que deben ser implementados por las clases derivadas.

Propósito y uso de clases abstractas

El propósito principal de una clase abstracta es:

Proporcionar una base común:

Las clases abstractas permiten definir métodos que deben ser implementados por todas las clases derivadas, asegurando que todas las clases hijas tengan ciertos métodos en común.

Evitar la instanciación directa:

Dado que una clase abstracta es, por definición, abstracta, no tiene sentido crear un objeto de ella.

Su propósito es ser extendida por otras clases.

Métodos abstractos y concretos

Métodos abstractos:

Son métodos que se declaran en la clase abstracta pero no tienen una implementación. Las clases derivadas deben proporcionar una implementación para estos métodos. Si una clase derivada no proporciona una implementación para todos los métodos abstractos, entonces esa clase derivada también se considera abstracta.

Métodos concretos:

Aunque el propósito principal de una clase abstracta es proporcionar métodos abstractos para sus clases hijas, también puede tener métodos concretos (métodos con implementación). Esto permite que las clases abstractas proporcionen un comportamiento predeterminado, que las clases derivadas pueden usar o sobrescribir según sea necesario.

Limitaciones y ventajas

Limitaciones:

- No se pueden crear instancias de una clase abstracta.
- Puede requerir más esfuerzo inicial al diseñar la arquitectura del software, ya que se debe pensar en las clases base y en cómo se relacionarán con las clases derivadas.

Ventajas:

- Proporciona una estructura clara y una base común para las clases derivadas.
- Fomenta la reutilización de código, ya que las clases derivadas pueden heredar métodos y propiedades de la clase abstracta.
- Ayuda a mantener la consistencia en el diseño, ya que todas las clases derivadas deben seguir la estructura definida por la clase abstracta.

Diferencias clave entre clases abstractas e interfaces

Clases abstractas:

- Permiten la herencia simple, lo que significa que una clase derivada solo puede heredar de una única clase abstracta.
- Pueden tener miembros con implementaciones, lo que significa que las clases derivadas pueden heredar tanto la definición como la implementación de esos miembros.

Interfaces:

- Permiten la herencia múltiple, lo que significa que una clase puede implementar múltiples interfaces.
- Solo contienen definiciones de métodos sin implementación.
- Las clases que implementan una interfaz deben proporcionar la implementación de todos los métodos definidos en la interfaz.

Acceso a variables y modificadores

Clases abstractas:

- Pueden tener variables de instancia (campos) y estas pueden tener modificadores de acceso (privado, protegido, público).
- Pueden tener métodos con modificadores de acceso y métodos con implementación.

Interfaces:

- No pueden tener variables de instancia. Sin embargo, pueden tener constantes (variables estáticas y finales).
- Todos los métodos en una interfaz son implícitamente públicos y abstractos (esto puede variar según el lenguaje de programación; por ejemplo, en versiones más recientes de Java, las interfaces pueden tener métodos con implementación predeterminada)

Implementación de métodos

Clases abstractas:

- Pueden tener métodos con implementación (métodos concretos) y métodos sin implementación (métodos abstractos).
- Las clases derivadas pueden optar por usar la implementación predeterminada de la clase abstracta o sobrescribirla.

Interfaces:

- Tradicionalmente, solo contienen definiciones de métodos sin implementación. Las clases que implementan una interfaz deben proporcionar la implementación de todos los métodos.
- En algunos lenguajes modernos, como Java (a partir de Java 8), las interfaces pueden contener métodos con implementación predeterminada.

Uso y aplicaciones prácticas

Clases abstractas:

- Son útiles cuando se desea proporcionar una base común para un grupo de clases relacionadas, donde algunas implementaciones pueden ser compartidas.
- Son ideales cuando hay una relación "es un tipo de" entre la clase base y las clases derivadas.

Interfaces:

- Son ideales para definir contratos que varias clases pueden implementar, especialmente cuando estas clases no están relacionadas entre sí en términos de jerarquía de herencia.
- Son útiles cuando se desea garantizar que ciertas clases tengan ciertos métodos, independientemente de su árbol de herencia.

Ejercicio 1

Escenario:

Supongamos que en una farmacia se venden diferentes tipos de productos. Todos los productos tienen un nombre, un precio y una fecha de caducidad. Sin embargo, algunos productos requieren receta médica y otros no. Para este ejercicio, vamos a considerar dos tipos de productos: medicamentos y productos de cuidado personal.

Diseño

Clase abstracta Producto:

- Esta clase representará cualquier producto vendido en la farmacia.
- Tendrá atributos comunes como nombre, precio y fecha de caducidad.
- Tendrá métodos abstractos que las clases derivadas deben implementar, como requiereReceta().

Clase Medicamento:

- Esta clase heredará de Producto.
- Tendrá un atributo adicional para indicar si el medicamento requiere receta o no.
- Implementará el método requiereReceta().

Clase ProductoCuidadoPersonal:

- Esta clase también heredará de Producto.
- Los productos de cuidado personal generalmente no requieren receta, por lo que el método requiereReceta() simplemente devolverá false

```java
// Clase abstracta Producto
public abstract class Producto {
    protected String nombre;
    protected double precio;
    protected String fechaCaducidad;

    public Producto(String nombre, double precio, String fechaCaducidad) {
        this.nombre = nombre;
        this.precio = precio;
        this.fechaCaducidad = fechaCaducidad;
    }

    // Método abstracto que las clases derivadas deben implementar
    public abstract boolean requiereReceta();
}
```

```java
// Clase Medicamento que hereda de Producto
public class Medicamento extends Producto {
    private boolean requiereRecetaMedica;

    public Medicamento(String nombre, double precio, String fechaCaducidad, boolean requiereRecetaMedica) {
        super(nombre, precio, fechaCaducidad);
        this.requiereRecetaMedica = requiereRecetaMedica;
    }

    @Override
    public boolean requiereReceta() {
        return requiereRecetaMedica;
    }
}
```

```
// Clase ProductoCuidadoPersonal que hereda de Producto
public class ProductoCuidadoPersonal extends Producto {

    public ProductoCuidadoPersonal(String nombre, double
precio, String fechaCaducidad) {
        super(nombre, precio, fechaCaducidad);
    }

    @Override
    public boolean requiereReceta() {
        return false; // Los productos de cuidado personal
generalmente no requieren receta
    }
}
```

Con este diseño, puedes crear instancias de Medicamento y ProductoCuidadoPersonal, y cada una de ellas tendrá la capacidad de indicar si requiere receta médica o no, gracias al método requiereReceta().

```
public class FarmaciaMain {
    public static void main(String[] args) {
        //Crear un medicamento que requiere receta
        Medicamento paracetamol = new Medicamento("Paracetamol", 5.50, "31/12/2024", true);

        //Crear un medicamento que no requiere receta
        Medicamento ibuprofeno = new Medicamento("Ibuprofeno", 7.00, "31/12/2024", false);

        //Crear un producto de cuidado personal
        ProductoCuidadoPersonal champu = new ProductoCuidadoPersonal("Champú Herbal", 3.50, "31/12/2025");

        //Mostrar información
        mostrarInfoProducto(paracetamol);
        mostrarInfoProducto(ibuprofeno);
        mostrarInfoProducto(champu);
    }

    public static void mostrarInfoProducto(Producto producto) {
        System.out.println("Nombre del producto: " + producto.nombre);
        System.out.println("Precio: $" + producto.precio);
        System.out.println("Fecha de caducidad: " + producto.fechaCaducidad);
        if (producto.requiereReceta()) {
            System.out.println("Este producto requiere receta médica.");
        } else {
            System.out.println("Este producto no requiere receta médica.");
        }
        System.out.println("------------------------------");
    }
}
```

Practica:

- Implementar el ejercicio anterior y adicionar una clase adicional (a su criterio) que herede de producto.
- No olvidar sobrescribir el método de acuerdo a la clase implementada.

Conclusiones

- Las clases abstractas en programación orientada a objetos sirven como una base común para otras clases, permitiendo definir comportamientos (métodos) que las clases derivadas deben implementar. Estas clases no pueden ser instanciadas directamente, lo que significa que su principal función es ser extendidas por otras clases.
- Mientras que las clases abstractas se centran en proporcionar una base común y permiten la herencia simple, las interfaces definen contratos que las clases pueden implementar, permitiendo la herencia múltiple. Esto hace que las interfaces sean ideales para definir comportamientos que pueden ser compartidos por clases que no están necesariamente relacionadas en términos de jerarquía de herencia.

Cierre

¿Qué hemos aprendido hoy?

Bibliografía

- MORENO PÉREZ, J. “Programación orientada a objetos”. RA-MA Editorial. https://tubiblioteca.utp.edu.pe/cgi-bin/koha/opac-detail.pl?biblionumber=31933
- Vélez Serrano, José. “Diseñar y programar, todo es empezar: una introducción a la Programación Orientada a Objetos usando UML y Java”. Dykinson. https://tubiblioteca.utp.edu.pe/cgi-bin/koha/opac-detail.pl?biblionumber=36368

Universidad Tecnológica del Perú