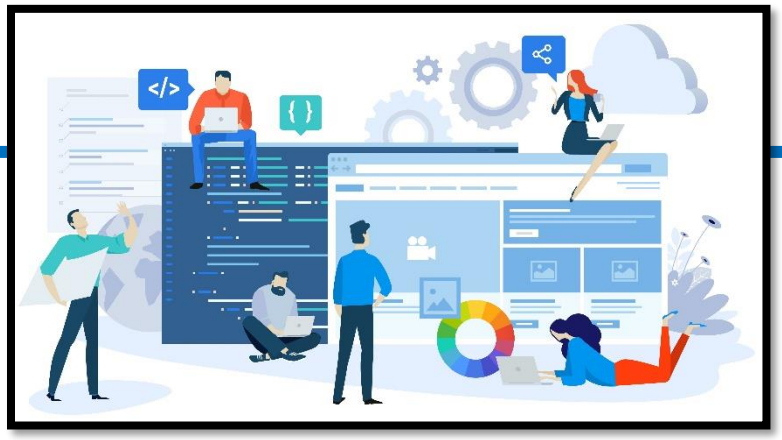
UNIVERSIDAD ISRAEL



TEMA: TAREA SEMANA 5

NOMBRE: VALVERDE PEDRO

CURSO: SEXTO A

FECHA: 05/05/2023

MODALIDAD: VIRTUAL

CARRERA: INGENIERÍA EN INFORMÁTICA

MATERIA: INGIENIERIA DEL SOFTWARE II

AÑO LECTIVO

2024 – A

Patrón Estructural

Patrón de Diseño Adapter

La implementación de estos métodos debe ser proporcionada por las clases que extienden la clase `Motor` y que representan tipos específicos de motores, como motores de automóviles, motores eléctricos, etc.

```
package backend;

public abstract class Motor {
    abstract public void encender();
    abstract public void acelerar();
    abstract public void apagar();
}
```

Los métodos simplemente imprimen mensajes en la consola para indicar lo que está haciendo el motor común en cada momento.

```java
public class MotorComun extends Motor {

    public MotorComun(){
        super();
        System.out.println("Creando motor común...");
    }

    @Override
    public void encender() {
        System.out.println("Encendiendo motor común.");
    }

    @Override
    public void acelerar() {
        System.out.println("Acelerando motor común.");
    }

    @Override
    public void apagar() {
        System.out.println("Apagando motor común.");
    }
}
```

Los métodos imprimen mensajes en la consola para indicar lo que está haciendo el motor económico en cada momento, de manera similar a la clase `MotorComun`.

```java
public class MotorEconomico extends Motor {

    public MotorEconomico(){
        super();
        System.out.println("Creando motor económico...");
    }

    @Override
    public void encender() {
        System.out.println("Encendiendo motor económico.");
    }

    @Override
    public void acelerar() {
        System.out.println("Acelerando el motor económico.");
    }

    @Override
    public void apagar() {
        System.out.println("Apagando motor económico.");
    }
}
```

Esto permite que la clase MotorElectrico sea utilizada como si fuera un `Motor` común, a pesar de tener una interfaz diferente. Cuando se llaman a los métodos `encender()`, `acelerar()` o `apagar()` en un objeto MotorElectricoAdapter, se delega la ejecución a los métodos correspondientes en un objeto `MotorElectrico`.

```java
public class MotorElectrico {

    private boolean conectado = false;

    public MotorElectrico() {
        System.out.println("Creando motor electrico...");
        this.conectado = false;
    }

    public void conectar() {
        System.out.println("Conectando motor eléctrico.");
        this.conectado = true;
    }

    public void activar() {
        if (!this.conectado) {
            System.out.println("No se puede activar porque no está conectado el motor eléctrico.");
        } else {
            System.out.println("Está conectado, activando motor eléctrico.");
        }
    }

    public void moverMasRapido() {
        if (!this.conectado) {
            System.out.println("No se puede mover rapido el motor eléctrico porque no está conectado.");
        } else {
            System.out.println("Moviendo más rapido, aumentando voltaje del motor eléctrico.");
        }
    }

    public void detener() {
        if (!this.conectado) {
            System.out.println("No se puede detener motor eléctrico porque no está conectado.");
        } else {
            System.out.println("Deteniendo motor eléctrico.");
        }
    }

    public void desconectar() {
        System.out.println("Desconectando motor eléctrico.");
        this.conectado = false;
    }
}
```

Permite al usuario interactuar con diferentes tipos de motores (común, económico o eléctrico) a través de un menú de opciones. Dependiendo de la opción seleccionada, se crea el tipo de motor correspondiente y se utilizan sus funcionalidades.

```
public class MotorElectricoAdapter extends Motor{
    private MotorElectrico motorElectrico;

    public MotorElectricoAdapter(){
        super();
        System.out.println("Creando motor eléctrico adapter...");
        this.motorElectrico = new MotorElectrico();
    }
    @Override
    public void encender() {
        System.out.println("Encendiendo motor eléctrico adapter.");
        this.motorElectrico.conectar();
        this.motorElectrico.activar();
    }

    @Override
    public void acelerar() {
        System.out.println("Acelerando motor eléctrico adapter.");
        this.motorElectrico.moverMasRapido();
    }

    @Override
    public void apagar() {
        System.out.println("Apagando motor eléctrico adapter.");
        this.motorElectrico.detener();
        this.motorElectrico.desconectar();
    }
}
```

Utilizaremos nuestra clase para imprimir por consola utilizaremos un do while para que se repetitivo y un switch case para que se pueda ir seleccionando y sobre todo se dinámico y controlado.

```java
public class ArquitecturaAdapter {

    private static Scanner S = new Scanner(System.in);
    private static Motor motor;

    public static void main(String[] args) {
        System.out.println("");
        int opcion;
        do{
            opcion = preguntarOpcion();
            switch(opcion) {
                case 1:
                    motor = new MotorComun();
                    usarMotor();
                    break;
                case 2:
                    motor = new MotorEconomico();
                    usarMotor();
                    break;
                case 3:
                    motor = new MotorElectricoAdapter();
                    usarMotor();
                    break;
                case 4:
                    System.out.println("¡Cerrando programa!");
                    break;
                default:
                    System.out.println("La opción ingresada NO es válida.");
            }
            System.out.print("\n\n");
        }while(opcion!=4);
    }

    private static int preguntarOpcion() {
        System.out.print(
                "MENÚ DE OPCIONES\n"
               +"---- -- --------\n"
               +"1. Encender motor común.\n"
               +"2. Encender motor económico.\n"
               +"3. Encender motor eléctrico.\n"
               +"4. Salir.\n"
               +"Seleccione opción: "
        );
        return Integer.parseInt( S.nextLine() );
    }
```

Ejecución

En el menú mostraremos cual la opción que desea realizar en este caso mostraremos la opción 1 y nos mostrara que el motor se creo enciende acelera y apaga y como es dinámico nos volverá a preguntar en el menú que opción deseamos.

```
MENÚ DE OPCIONES
---- -- --------
1. Encender motor común.
2. Encender motor económico.
3. Encender motor eléctrico.
4. Salir.
Seleccione opción: 1
Creando motor común...
Encendiendo motor común.
Acelerando motor común.
Apagando motor común.


MENÚ DE OPCIONES
---- -- --------
1. Encender motor común.
2. Encender motor económico.
3. Encender motor eléctrico.
4. Salir.
Seleccione opción: 2
Creando motor económico...
Encendiendo motor económico.
Acelerando el motor económico.
Apagando motor económico.


MENÚ DE OPCIONES
---- -- --------
1. Encender motor común.
2. Encender motor económico.
3. Encender motor eléctrico.
4. Salir.
Seleccione opción: 3
Creando motor eléctrico adapter...
Creando motor electrico...
```

Conclusión

En conclusión, el patrón de diseño adapter es una herramienta valiosa para integrar componentes con interfaces incompatibles y facilitar la interoperabilidad entre diferentes partes de un sistema. Su capacidad para adaptar interfaces existentes sin modificar su código subyacente lo hace especialmente útil en entornos donde la flexibilidad y la reutilización del código son importantes.

Nota: Se adjunta el proyecto por la herramienta git hub https://github.com/Eduardo199614/PatronEstrucuralAdapter.git