UNIVERSIDAD DE LIMA

FACULTAD DE INGENIERÍA
CARRERA DE INGENIERÍA DE SISTEMAS

Ingeniería de Software II

Laboratorio N°3

Patrones Creacionales

Repositorio: https://github.com/anderalarcon/ing-sw-2-2024-1-java

Singleton

En simples palabras, el objetivo es garantizar que una clase tenga solo una instancia y proporcionar un punto de acceso global a esa instancia.

Descripción:

Estás trabajando en un proyecto de desarrollo de software que requiere la conexión a una base de datos para almacenar y recuperar datos relevantes para la aplicación. Se te ha proporcionado un bloque de código que se encarga de establecer la conexión a la base de datos, pero parece que está teniendo problemas al crear múltiples conexiones innecesarias.

Tareas a Realizar:

1. Se te proporcionará un bloque de código que establece una conexión a la base de datos, pero no aplica el patrón Singleton correctamente.

DatabaseConnector:

```
import java.sql.Connection;

public class DatabaseConnector {

    private Connection connection;

    public DatabaseConnector() {
```

```
        // Lógica de conexión a la base de datos
        try {
            System.out.println("Database connected");
        } catch (Error e) {
            System.out.println("Error:" + e.getMessage());
        }
    }
}
```

Main:

```
public class Main {

    public static void main(String[] args) {
        // Crear una nueva instancia de DatabaseConnector y conectar a la
base de datos
        DatabaseConnector connector1 = new DatabaseConnector();

        // Crear otra instancia de DatabaseConnector y conectar a la base de
datos nuevamente
        DatabaseConnector connector2 = new DatabaseConnector();
    }

}
```

2. Tu tarea es corregir este bloque de código para que aplique el patrón Singleton correctamente, asegurando que solo haya una instancia de la clase de conexión a la base de datos en toda la aplicación.

3. Proporciona una versión corregida del bloque de código, aplicando el patrón Singleton de manera adecuada.

Factory Method

- ***Método de fábrica**: Debe haber un método en una clase (llamada la clase creadora) que se encargue de crear objetos.*
- ***Objetos producidos por subclases:** El método de fábrica debe ser abstracto.*
- ***Desacoplamiento entre el cliente y las clases concretas***
- ***Creación de un solo objeto o familia de objetos***

Descripción:

Estás revisando el código de una tienda online que tiene problemas en la creación de productos. La implementación actual no sigue las mejores prácticas de diseño y está generando un acoplamiento innecesario entre la lógica de creación de productos y la tienda en sí. Tu tarea es corregir esta implementación aplicando el patrón Factory Method.

Tareas a realizar:

1. Identifica los problemas en la implementación actual y comprende por qué no sigue las mejores prácticas de diseño.

Producto:

```
public class Producto {

    private String nombre;
    private double precio;
    private double costoEnvio;

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

    public String obtenerNombre() {
        return nombre;
    }

    public double obtenerPrecio() {
        return precio;
    }

    public double calcularEnvio() {
```

```
        return costoEnvio;
    }
}
```

Tienda:

```
public class Tienda {

    public Producto crearProducto(String tipo) {
        if (tipo.equals("electronico")) {
            return new Producto("Smartphone", 500.0, 10.0);
        } else if (tipo.equals("ropa")) {
            return new Producto("Camisa", 30.0, 5.0);
        } else {
            return null;
        }
    }
}
```

Main:

```
public class Main {

    public static void main(String[] args) {
        // Crear una instancia de la tienda
        Tienda tienda = new Tienda();

        // Crear productos utilizando la tienda
        Producto productoElectronico = tienda.crearProducto("electronico");
        Producto productoRopa = tienda.crearProducto("ropa");

        // Mostrar detalles de los productos
        System.out.println("Producto electrónico:");
        mostrarDetallesProducto(productoElectronico);
        System.out.println();

        System.out.println("Producto de ropa:");
        mostrarDetallesProducto(productoRopa);
        System.out.println();
    }

    // Método para mostrar los detalles de un producto
    public static void mostrarDetallesProducto(Producto producto) {
        System.out.println("Nombre: " + producto.obtenerNombre());
        System.out.println("Precio: $" + producto.obtenerPrecio());
        System.out.println("Costo de envío: $" + producto.calcularEnvio());
    }
}
```

2. Utiliza el patrón Factory Method para refactorizar la implementación y corregir los problemas identificados.
3. Implementa una estructura de clases que permita la creación de diferentes tipos de productos de manera uniforme y extensible.

Abstract Factory Method

Para que un diseño cumpla con el patrón Abstract Factory Method, debe tener dos características principales:

1. ***Familias de objetos relacionados****: Debe definir familias completas de objetos relacionados, como diferentes tipos de productos que comparten una funcionalidad o propósito común.*
2. ***Fábricas abstractas e intercambiables****: Debe tener una interfaz o clase base común para las fábricas, que define métodos para crear cada producto de la familia. Las fábricas concretas implementan esta interfaz o heredan de la clase base, proporcionando implementaciones específicas para crear los productos de la familia correspondiente.*

Descripción:
Se te ha encomendado la tarea de corregir una implementación incorrecta en el sistema de fabricación de muebles de una fábrica. La fábrica produce diferentes estilos de muebles, como muebles modernos y muebles clásicos. Sin embargo, la implementación actual no sigue las mejores prácticas de diseño y necesita ser corregida.

Tareas a realizar:

1. Identifica qué aspectos de la implementación actual no siguen las mejores prácticas de diseño y por qué.
2. Utiliza el patrón Abstract Factory Method para corregir la implementación incorrecta.
3. Implementa clases adicionales y reestructura el código según sea necesario para aplicar el patrón correctamente.