ESPE

UNIVERSIDAD DE LAS FUERZAS ARMADAS

INNOVACIÓN PARA LA EXCELENCIA

UNIVERSIDAD DE LAS FUERZAS ARMADAS

INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN

MATERIA

PROGRAMACION ORIENTADA A OBJETOS

TEMA

patrón de diseño Modelo Vista Controlador (MVC)

Estudiantes

- **Bryan Roger Camacho Ramírez**
- **Eduardo Anibal Mejía Catillo**
- **Israel Fernando Portilla Santamaria**
- **Jenifer Andrea Castro Cuevas**
- **Julio Emerson Coro Pineida**
- **Alexis Sebastián Murminacho Cabascango**

Link del repositorio:
https://github.com/IsraelPortilla/Repositorio_Israel_Portilla/tree/main/Grupo%209/Control%20de%20lectura%202

Link del video: https://drive.google.com/file/d/18F3A2u8jh4-r9ZeSfSCYePRiBbiNGHQf/view?usp=drivesdk

Objetivos

- Garantizar que el código sea sólido, fácil de mantener y ampliable mediante el uso de patrones de diseño que mejoren su estructura y modularidad.
- Fomentar el desarrollo de componentes reutilizables que puedan integrarse y adaptarse con facilidad en diversos proyectos, optimizando la eficiencia y coherencia en el proceso de desarrollo de software.

Organización del Código mediante Patrones de Diseño

En el desarrollo de software, los patrones de diseño representan soluciones establecidas para abordar problemas recurrentes. Su aplicación contribuye a mejorar la estructura, el mantenimiento y la adaptabilidad del código. Uno de los patrones más utilizados es el Modelo Vista Controlador (MVC).

El patrón MVC segmenta una aplicación en tres partes fundamentales. El Modelo gestiona la lógica de negocio y los datos, asegurando la interacción con la base de datos y las reglas del negocio. La Vista se encarga de mostrar la información al usuario y reflejar cualquier cambio en los datos. Por su parte, el Controlador actúa como intermediario entre el modelo y la vista, procesando las entradas del usuario y actualizando ambos componentes según sea necesario. Esta división facilita la administración de proyectos complejos, separando la lógica de negocio, la interfaz gráfica y el flujo de control. Además, mejora la capacidad de prueba del código

mediante pruebas unitarias e integraciones, lo que resulta en sistemas más sólidos y menos propensos a errores.

Otro patrón esencial es el Singleton, que garantiza que una clase tenga una única instancia y ofrece un punto de acceso global. Esto es útil cuando se requiere gestionar recursos compartidos de manera eficiente.

El Factory Method permite la creación de objetos sin definir explícitamente la clase instanciada, promoviendo flexibilidad y reutilización del código. De manera similar, el Abstract Factory facilita la generación de grupos de objetos relacionados sin concretar sus clases, asegurando coherencia en el diseño y favoreciendo la interacción entre ellos.

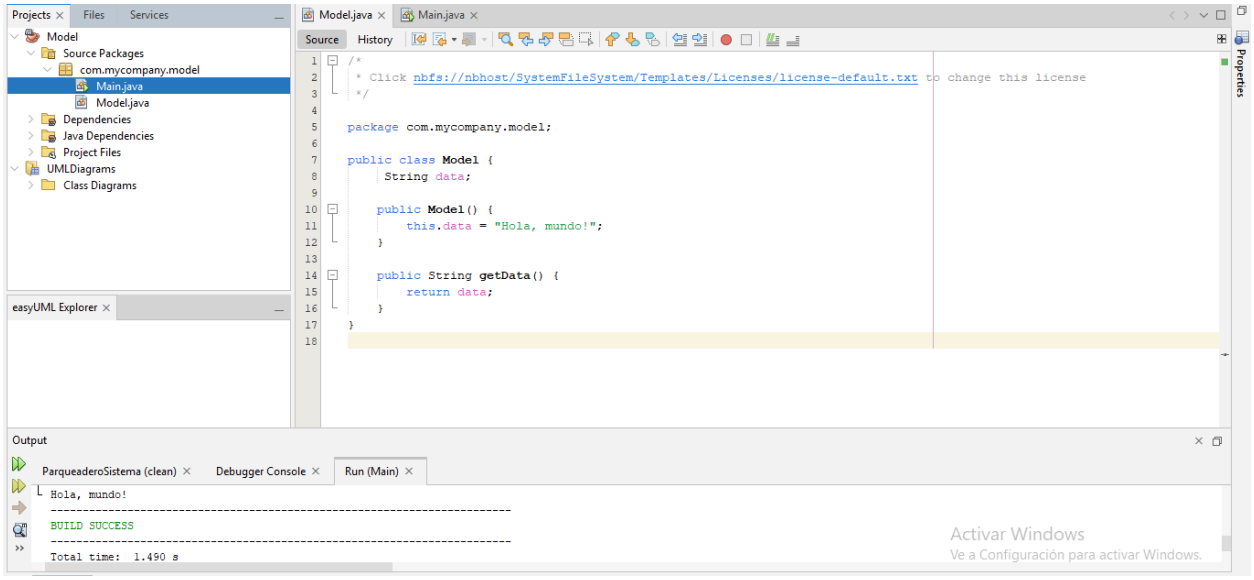
El Adapter es clave para compatibilizar interfaces incompatibles, ya que envuelve una clase dentro de una interfaz existente, facilitando la integración y reutilización de código en distintos contextos. Por otro lado, el Composite permite tratar de manera uniforme objetos individuales y estructuras compuestas mediante una jerarquía en árbol, mientras que el Decorator añade funcionalidades adicionales a un objeto de forma dinámica, proporcionando una alternativa más flexible a la herencia.

Entre los patrones de comportamiento, el Observer establece una relación de dependencia de uno a muchos entre objetos, asegurando que cuando uno cambia de estado, todos sus dependientes sean notificados automáticamente. Esto mejora la comunicación y la escalabilidad del sistema. El **Strategy** encapsula un conjunto de algoritmos intercambiables, permitiendo cambiar su comportamiento sin afectar a los clientes que los emplean, lo que favorece la reutilización y flexibilidad. Finalmente, el Command encapsula una acción dentro de un objeto, lo que facilita la

parametrización de clientes con diferentes solicitudes y permite gestionar operaciones reversibles, brindando un enfoque extensible para el manejo de comandos.

ejemplos

ejemplo Model



Ejemplo de Singleton

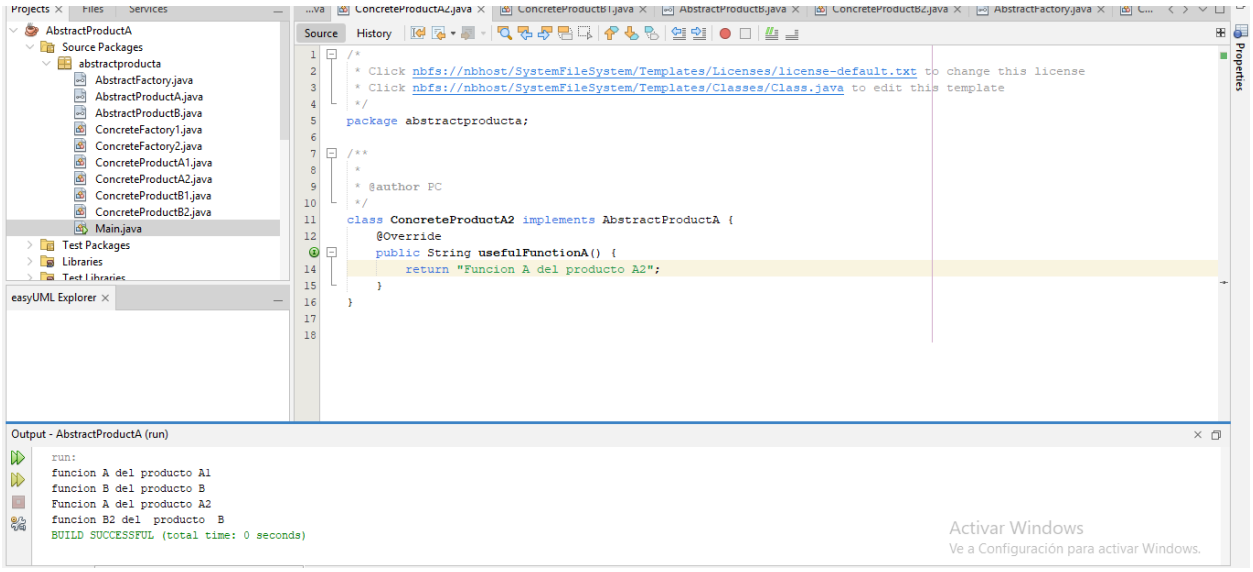
Ejemplo Factory Method

```java
public class Singleton {
    static Singleton instance;
    String value;

    Singleton(String value) {
        this.value = value;
    }

    public static Singleton getInstance(String value) {
        if (instance == null) {
            instance = new Singleton(value);
        }
        return instance;
    }

    public String getValue() {
        return value;
    }

    public static void main(String[] args) {
        Singleton singleton1 = Singleton.getInstance("primer instancia");
        Singleton singleton2 = Singleton.getInstance("segunda instancia");

        System.out.println(singleton1.getValue());
        System.out.println(singleton2.getValue());
        System.out.println(singleton1 == singleton2);
    }
}
```

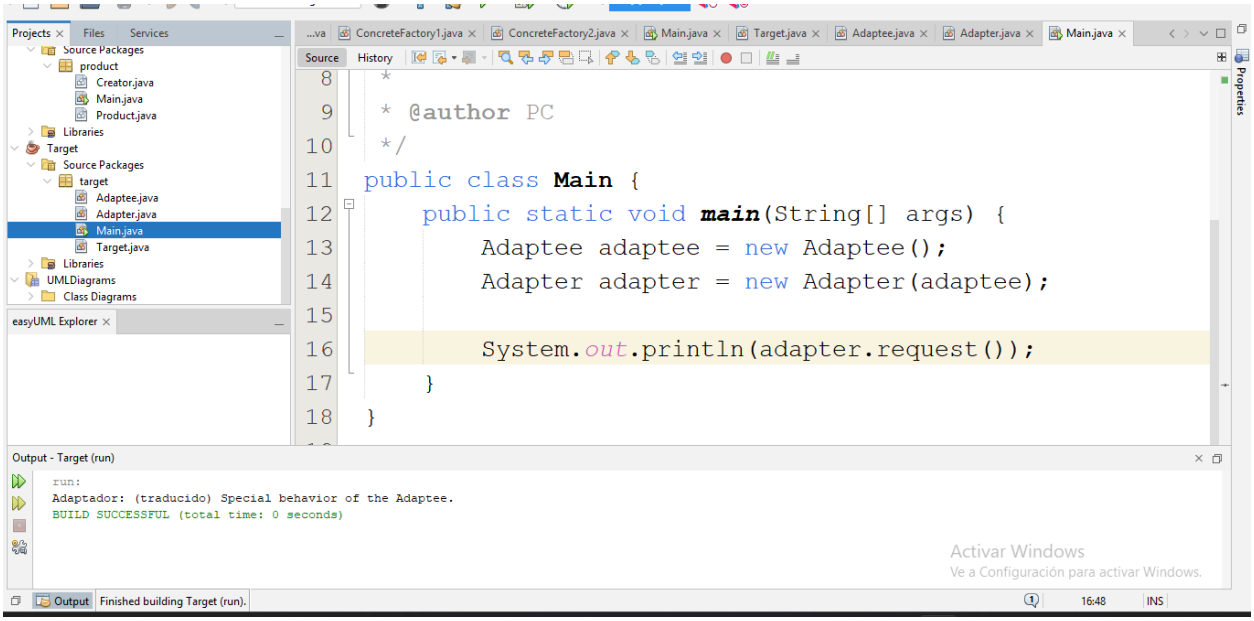
```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit this t
 */
package product;

/**
 *
 * @author PC
 */
abstract class Product {
    public abstract String operation();
}

class ConcreteProductA extends Product {
    @Override
    public String operation() {
        return "Productob aperacion";
    }
}

class ConcreteProductB extends Product {
    @Override
    public String operation() {
        return "ProductoB operacion";
    }
}
```

```
run:
Productob aperacion
ProductoB operacion
BUILD SUCCESSFUL (total time: 0 seconds)
```

Ejemplo Abstract Factory



ejemplo Adapter



BIBLIOGRAFIA

Gamma, E., Helm, R., Johnson, R., Vlissides, J., & Patterns, D. (1995). Elements of reusable object-oriented software. *Design Patterns*.

Bascón Pantoja, E. (2004). El patrón de diseño Modelo-Vista-Controlador (MVC) y su implementación en Java Swing. *Acta Nova*, *2*(4), 493-507.

Alvarez, O. D. G., Larrea, N. P. L., & Valencia, M. V. R. (2022). Análisis comparativo de Patrones de Diseño de Software. *Polo del Conocimiento: Revista científico-profesional*, *7*(7), 2146-2165.

Construir una base de código modular con patrones de programación MVC y MVP. (s. f.). Unity. https://unity.com/es/how-to/build-modular-codebase-mvc-and-mvp-programming-patterns

Ponce Romero, J. M., & Valderrama Bacilio, L. B. (2013). Desarrollo de un Sistema basado en Tecnología web usando el Patrón de Diseño Modelo Vista Controlador para mejorar la Actualización del Inventario de equipos dslam de Telefónica del Perú.