Mateo Cante Garzón – m.canteg – 202223518

Taller 5 – Patrones

Link del repositorio: https://github.com/Jojan-Esteban-Serna/Practica-Patrones-de-Comportamiento---MVC-con-Command-y-Observer.git

Información general del proyecto:

Propósito

Este proyecto tiene como objetivo principal desarrollar un sistema de gestión de inventario para una tienda. La finalidad es optimizar la administración de productos, asegurar la disponibilidad de stock, y mejorar la eficiencia en la gestión de pedidos y entregas. +

Estructura General del Diseño

El diseño del sistema incluye los siguientes módulos:

- **Domain.service.Product.Service:** Permite la adición, modificación y eliminación de productos. Realiza un seguimiento en tiempo real de la cantidad de productos en stock.
- **Domain.Product**: Información del producto, lleva las intancias de estas.
- **Presentación.GUIProducts:** Sincroniza la información de productos y disponibilidad con la plataforma. Lógica de trasfondo de lo que está en las interfaces.

Retos de Diseño:

El desafío principal es gestionar un amplio conjunto de productos con diferentes características, tamaños y categorías de manera eficiente. Además de la de diseñar un sistema que optimice la disposición física de los productos en el almacén para facilitar la preparación de pedidos y reducir tiempos de procesamiento. Finalmente, asegurar una interfaz de usuario amigable para que el personal de la tienda pueda gestionar el inventario de manera eficiente.

Este proyecto busca mejorar significativamente la eficiencia operativa al optimizar la gestión de inventario. Por lo que se espera que el sistema garantice la disponibilidad de productos, reduzca los errores en la gestión de pedidos y mejore la experiencia general de compra para los clientes

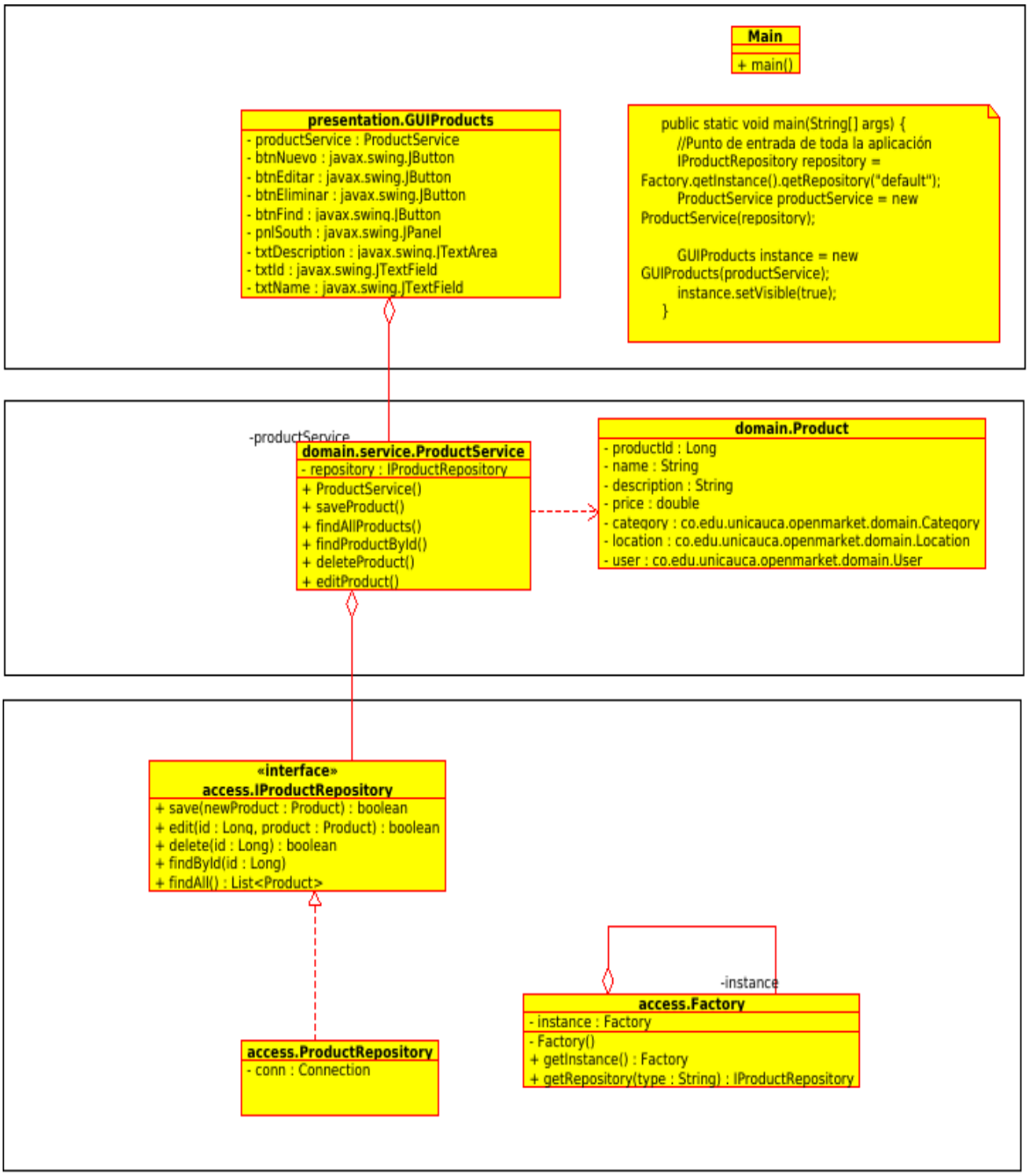

Imagen 1. UML general del proyecto - Patrón de comportamiento: Command.

Patrón de comportamiento: Command.

El patrón de diseño Command es un patrón de comportamiento que se utiliza para encapsular una solicitud como un objeto. Este objeto de comando permite parametrizar a los clientes con distintas solicitudes, encolar las solicitudes, y también proporcionar soporte para deshacer las operaciones.

Elementos clave del patrón Command:

Gracias a la ***imagen 1***, donde se nos muestra el UML del proyecto, es que podemos ver de manera mucho más clara y asertiva el uso de este patrón de comportamiento en el proyecto. En la representación UML, observamos las siguientes relaciones y componentes que ilustran la implementación del patrón Command:

- **Command :** Esta es una interfaz que declara un método abstracto. Este método encapsula la acción que debe realizarse cuando se invoca el comando.
- **ConcreteCommand (Acces.IProductoRepository):** Implementa la interfaz Command y define la acción específica que debe ejecutarse. También almacena una referencia al objeto receptor, que es responsable de llevar a cabo la acción.
- **Receiver (Domain.service.Product.Service):** Conoce cómo realizar la acción asociada con la solicitud. Cualquier clase puede actuar como un receptor.

¿Por qué tiene sentido haber utilizado el patrón en ese punto del proyecto?

- **Flexibilidad y Extensibilidad:** El uso del patrón Command proporciona flexibilidad al permitir la adición de nuevos comandos sin modificar el código existente. Por ejemplo, si en el futuro se desea introducir una nueva operación en el inventario, se puede crear un nuevo comando y asociarlo con un Receiver sin afectar otras partes del sistema.

- **Desacoplamiento:** Desacoplar el Invocador del Receiver facilita la gestión de cambios y actualizaciones en el sistema. El Invocador no necesita conocer los detalles internos de cómo se implementa una operación, simplemente invoca el comando y este se encarga de ejecutar la acción asociada.

- **Encolado de Operaciones:** El patrón Command permite encolar operaciones para su ejecución posterior. Esto podría ser útil para gestionar operaciones en lotes o para mantener un historial de acciones realizadas en el sistema de inventario.

¿Qué desventajas tiene haber utilizado el patrón en ese punto del proyecto?

Aunque el patrón Command proporciona varias ventajas en términos de desacoplamiento, flexibilidad y extensibilidad, también tiene algunas posibles desventajas y consideraciones que se deben tener en cuenta:

- **Complejidad Adicional:** La implementación del patrón Command puede agregar una capa adicional de complejidad al código, especialmente si el proyecto es pequeño o la lógica de negocio no es muy compleja. En algunos casos, la introducción de este patrón podría considerarse excesiva.

- **Aumento de la Cantidad de Clases:** Cada comando y clase relacionada agrega una nueva clase al código. Si hay muchos comandos y operaciones, podría haber un aumento significativo en la cantidad de clases, lo que puede hacer que el código sea más difícil de entender y mantener.

- **Memoria Adicional Utilizada:** En algunos casos, la implementación del patrón Command puede resultar en un uso adicional de memoria, ya que se están creando instancias de objetos Command y almacenándolas para su ejecución posterior.

- **Sobrecarga de Implementación:** En proyectos pequeños o simples, la introducción del patrón Command puede considerarse una sobrecarga innecesaria. Si las operaciones son directas y la lógica de negocio es sencilla, otras opciones más simples podrían ser más apropiadas.

- **Potencial para Confusión en Grandes Proyectos:** En proyectos grandes, si no se organiza correctamente, la introducción excesiva de comandos y clases relacionadas podría llevar a una complejidad excesiva y a posibles confusiones en la gestión de clases.

¿De qué otras formas se le ocurre que se podrían haber solucionado, en este caso particular, los problemas que resuelve el patrón?

El enfoque basado en eventos y observadores es una alternativa al patrón de diseño Command y se basa en la idea de que los cambios en el sistema generan eventos, y los componentes interesados (observadores) pueden reaccionar a estos eventos sin necesidad de comandos explícitos.

Componentes clave en este enfoque:

- **Eventos:** Representan un cambio significativo en el sistema, en este caso, un cambio en el inventario. Cada vez que ocurre una operación importante (como agregar, eliminar o actualizar un producto), se emite un evento correspondiente.

- **Observadores:** Son componentes que están interesados en ciertos tipos de eventos. Se registran para recibir notificaciones cuando ocurre un evento específico. En el contexto del inventario, los observadores podrían ser, por ejemplo, componentes encargados de actualizar la interfaz de usuario, generar informes o realizar otras acciones en respuesta a cambios en el inventario.

Cómo funciona este enfoque:

- Generación de Eventos: Cuando ocurre una operación en el inventario, se genera un evento correspondiente. Por ejemplo, si se agrega un nuevo producto, se genera un

evento de "Producto Agregado". Este evento contiene información relevante sobre la operación.

- Notificación a Observadores: Cuando ocurre un evento, el sistema notifica a todos los observadores registrados para ese tipo de evento. Cada observador reacciona de acuerdo con su lógica específica.

Ventajas de este enfoque:

- **Desacoplamiento:** Los componentes que generan eventos no necesitan conocer a los observadores ni sus detalles de implementación. Esto promueve el desacoplamiento y facilita la modificación y extensión del sistema.

- **Flexibilidad:** Se pueden agregar nuevos observadores o tipos de eventos sin afectar a los componentes existentes. Esto facilita la introducción de nuevas funcionalidades sin modificar el código existente.

- **Escalabilidad:** Este enfoque puede escalar bien, especialmente en sistemas grandes con múltiples componentes que necesitan reaccionar a eventos específicos.

Limitaciones y Consideraciones:

- **Complejidad en la Lógica de Observadores:** la lógica de estos componentes podría volverse compleja.

- **Control sobre el Flujo de Programa:** puede ser menos controlable en ciertos casos.

- **Identificación de Eventos Relevantes:** Identificar y definir correctamente los eventos relevantes para el sistema puede requerir una cuidadosa planificación y diseño.