

**Universidad Tecnológica del Perú**

**PROYECTO FINAL**

**Asignatura:**

# Diseño de patrones

**Sección:**

**54150**

**Integrantes:**

# Luis Angel Oros Sicha

# Sandro Piero Salazar Camacho

**Docente:**

# Edwin Jesus Alonso Ledes

# Lima, Perú 202

**INTRODUCCIÓN**

El presente informe da cuenta del desarrollo de una aplicación de consola desarrollada en Apache NetBeans, cuya finalidad se centra en realizar una simulación de un proceso de selección de productos por parte de un cliente, el cual puede especificar el nombre del producto, su tipo y el precio de este.

Durante el desarrollo se implementaron diferentes patrones de diseño con el objetivo de mejorar la estructura del código, hacerlo más fácil de mantener y, a su vez, seguir buenas prácticas de programación orientada a objetos. Se emplearon patrones de diseño de carácter creacional como de carácter estructural y de comportamiento. Este proyecto es el punto inicial para entender el uso práctico de los patrones de diseño en aplicaciones de la vida real y de los beneficios que se citan en cuanto a la mejora de la calidad del software.

**OBJETIVO GENERAL**

Desarrollar una aplicación de consola en que permita la simulación de un sistema de ventas de productos, aplicando patrones de diseño de software con el fin de lograr un código modular, reutilizable y fácil de mantener.

**OBJETIVOS ESPECIFICOS**

- Implementar patrones de diseño creacionales (Builder y Factory Method) para la creación deobjetos tipo producto que sea flexible.

- Aplicar el patrón estructural Facade para simplificar la interacción entre las clases del sistema.

- Aplicar patrones de comportamiento (Observer, Command y Strategy) para facilitar que la lógica del sistema sea dinámica y desacoplada.

- Implementar una interfaz de consola que permite al usuario seleccionar productos, tipos y precios de una forma clara e interactiva.

- Fomentar buenas prácticas de programación por medio de la correcta aplicación de los patrones de diseño.

**MARCO TEORICO**

Los patrones de diseño de software son soluciones reutilizables a problemas comunes que surgen en el desarrollo de sistemas orientados a objetos, aunque no se trate de trozos de código reutilizable en forma directa, sino más bien de una guía que forma parte de un conjunto de buenas prácticas que describen cómo organizar clases y objetos para resolver determinados problemas de diseño. Su utilización facilita la mantenibilidad, la escalabilidad y la claridad del software.

Los patrones de diseño pueden ser clasificados en tres grandes grupos:

Patrones Creacionales: Se centran en la forma de construir objetos y permiten ampliar la flexibilidad en quién, cómo y cuándo se crean.

Patrones Estructurales: Se ocupan de las clases y objetos para formar estructuras más complejas.

Patrones de Comportamiento: Definé la forma en que se llevan a cabo la comunicación y las responsabilidades entre objetos.

**JUSTIFICACIÓN**

Un software aceptable y mantenible necesita algo más que simplemente funcionamiento; también debe tener una estructura lógica, clara y capaz de adaptarse al cambio. Este proyecto se basa en patrones de diseño porque proporciona una manera de resolver problemas de programación que se repiten de forma conocida y documentada.

El uso de patrones como el patrón Builder, el patrón Factory Method, el patrón Facade, el patrón Observer, el patrón Command y el patrón Strategy, permitió diseñar un sistema escalable, ayudando a la organización del código de manera que las responsabilidades de cada clase quedaran definidas, disminuyendo de esta forma la dependencia entre componentes y mejorando la legibilidad del programa.

Todo lo anterior a su vez facilita futuras ampliaciones o mejoras del sistema, sin necesidad de reescribir todo el código, lo cual es imprescindible en un entorno real de desarrollo. Al final, el resultado no es solo un diseño técnico, sino una buena oportunidad para aprender un uso adecuado de las buenas prácticas de la programación orientada a objetos.

**DESARROLLO**

**Estructura del Proyecto**

**Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.**

**Diagrama de clases UML**

**Diagrama

El contenido generado por IA puede ser incorrecto.Main.java:**

**Texto

El contenido generado por IA puede ser incorrecto.**

**OrdenBuilder.java:**

**Texto

El contenido generado por IA puede ser incorrecto.**

**VentaFacade.java:**

**Texto

El contenido generado por IA puede ser incorrecto.**

**ProductFactory.java:**

**Texto

El contenido generado por IA puede ser incorrecto.**

**Pedido.java:**

**Texto

El contenido generado por IA puede ser incorrecto.**

**Producto.java:**

**Texto

El contenido generado por IA puede ser incorrecto.**

**ProductoElectronico.java:**

**Texto

El contenido generado por IA puede ser incorrecto.**

**ProductoRopa.java:**

**Texto

El contenido generado por IA puede ser incorrecto.**

**Inventario.java**

**Texto

El contenido generado por IA puede ser incorrecto.**

# InventarioObserver.java

# Texto El contenido generado por IA puede ser incorrecto.

**CONCLUSIONES**

- Gracias a la implementación de patrones de diseño en la aplicación de consola desarrollada en Java, se ha desarrollado un sistema ordenado, modular y sencillo de mantener.

- La aplicación de los patrones, tal como son eBuilder, Factor Method, Facade, Observer, Command y Strategy han permitido aislar responsabilidades e implementar la reutilización del código, cumpliendo así principios de la programación orientada a objetos.

- A partir de la realización de este proyecto se ha conseguido entender de forma práctica como los patrones de diseño pueden dar respuesta a problemas comunes de desarrollo y mejorar la calidad del software desde los inicios del diseño.

**RECOMENDACIONES**

- Seguir practicando la utilización de patrones de diseño en proyectos más ambiciosos para reafirmar los aprendizajes adquiridos y desarrollar soluciones más complejas.

- Documentar siempre el uso de cada patrón en el código fuente para que los demás desarrolladores puedan entenderlo rápidamente.

- Explorar el uso de pruebas unitarias junto con los patrones aplicados para intentar asegurar que el sistema siga funcionando en el futuro.

- Considerar una interfaz gráfica (GUI) para futuras versiones del programa, mejorando de esta manera la experiencia del usuario, reutilizando así también la lógica que haya sido implementada desde el diseño modular del sistema.

**ANEXOS**

**-** [**https://refactoring.guru/es/design-patterns**](https://refactoring.guru/es/design-patterns)

**-** [**https://www.freecodecamp.org/espanol/news/los-principios-solid-explicados-en-espanol/**](https://www.freecodecamp.org/espanol/news/los-principios-solid-explicados-en-espanol/)

**-** [**https://codigofacilito.com/articulos/listado\_patrones**](https://codigofacilito.com/articulos/listado_patrones)