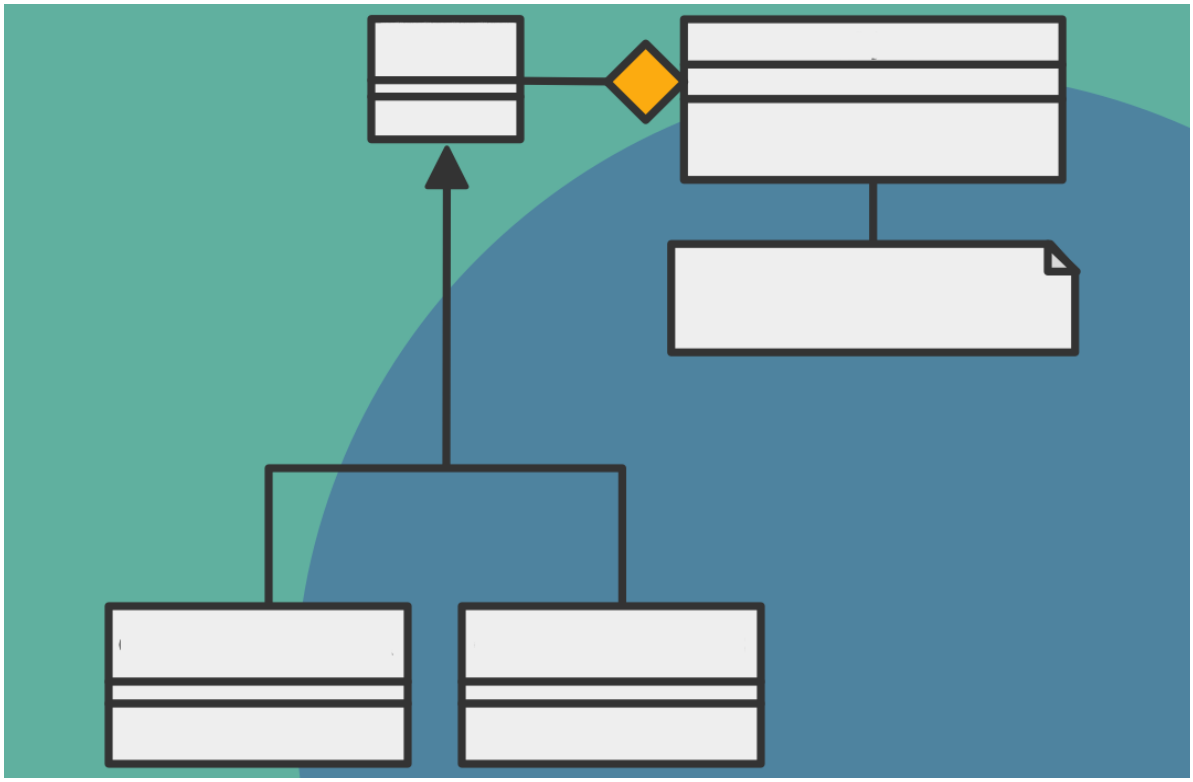
Alumna: Anahi Valdiviezo

INGENIERÍA DE SOFTWARE 3

PATRONES DE DISEÑO

Seguimiento del estado de pedidos



Introducción

Los patrones de diseño de software son soluciones generalizadas para problemas comunes que surgen durante el diseño y desarrollo de software. Son enfoques probados y estructuras reutilizables que ofrecen soluciones efectivas a desafíos recurrentes en el diseño de sistemas.

Estos patrones proporcionan un lenguaje común y un conjunto de mejores prácticas que los desarrolladores pueden utilizar para abordar problemas específicos en el diseño de software. Al aplicar patrones de diseño, se busca mejorar la flexibilidad, la mantenibilidad y la escalabilidad del código, así como promover la reutilización de código probado.

Problema planteado

Necesidad de un sistema de seguimiento de estado de pedidos, este no solo es una conveniencia para los clientes, sino que también es esencial para la eficiencia operativa y la competitividad de una plataforma de comercio electrónico.

Módulo de seguimiento de estados de pedidos

En este informe conoceremos más sobre los patrones de diseño implementados para la realización del módulo de seguimiento de pedidos, para dicha implementación hice uso del lenguaje de programación **JAVA**, haciendo uso de la herramienta **IntelliJ IDEA** como IDE.

Dicho modulo esta disponible en mi repositorio de github : https://github.com/anissval/Patrones_diseno/tree/main/src/main/java/org/sistemaSeguimientoPedidos

Estados del sistema

Los posibles estados pueden variar según los requisitos específicos del negocio, por lo cual esto se puede adaptar en cualquier momento simplemente generando más estados en el proyecto, incluso podrías tener estados adicionales o subestados para manejar situaciones más detalladas, como devoluciones, reembolsos, o cambios en los pedidos.

Los estados implementados en este proyecto fueron los siguientes:

1. Pendiente de Confirmación:

   El pedido ha sido realizado pero aún no ha sido confirmado por el sistema.

2. Confirmado:

   El pedido ha sido confirmado y está pendiente de procesamiento.

3. En Proceso:

El pedido está siendo procesado, ya sea preparando los productos o coordinando la entrega.

4. Listo para Envío:

El pedido ha sido procesado y está listo para ser enviado o recogido.

5. En Tránsito:

El pedido ha sido enviado y está en camino hacia la dirección de entrega.

6. Entregado:

El pedido ha sido entregado con éxito al cliente.

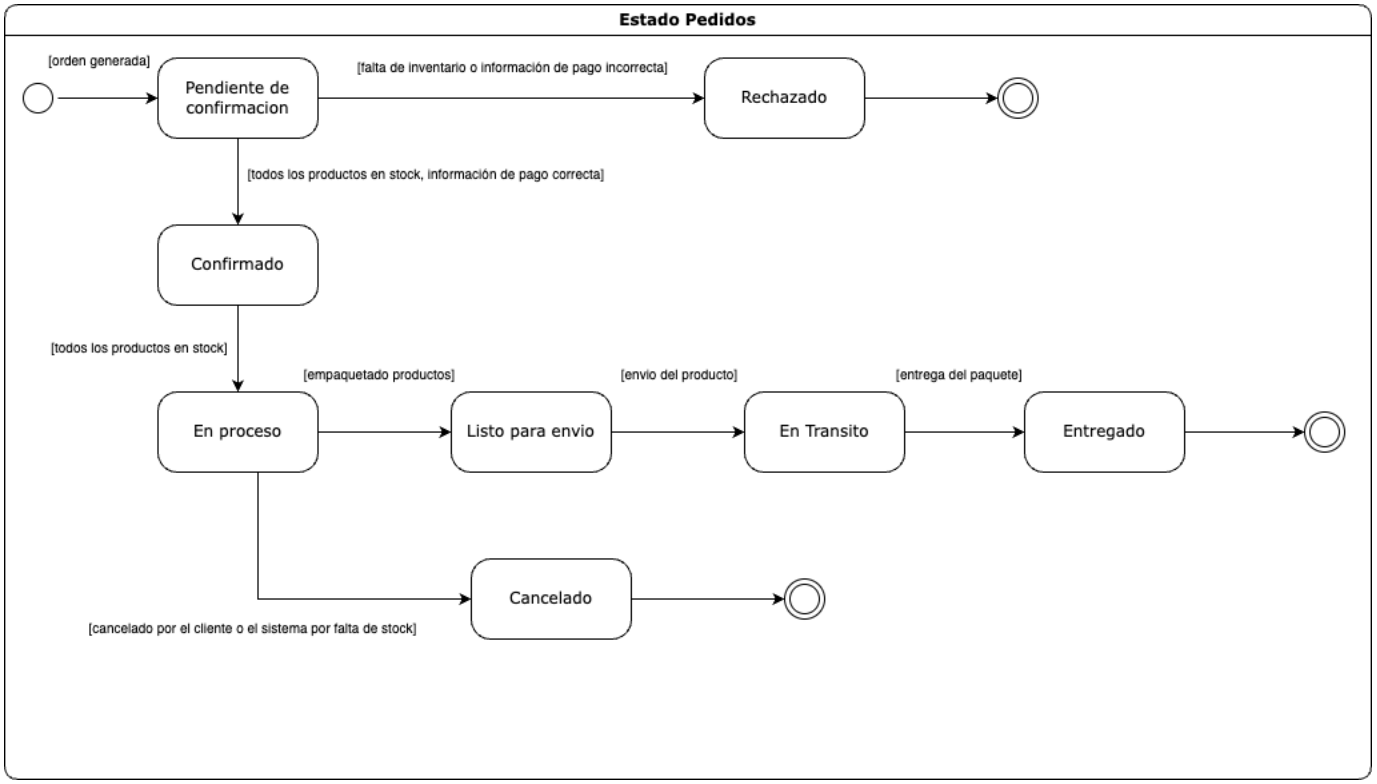
7. Cancelado:

El cliente o el sistema han cancelado el pedido antes de su procesamiento completo.

8. Rechazado:

El sistema ha rechazado el pedido debido a problemas como falta de inventario o información de pago incorrecta.

Diagrama de estados



Patrones implementados

Este proyecto incorpora los patrones de diseño **Singleton, Observer y State** en un sistema de seguimiento de estado de pedidos.

El patrón Singleton asegura que solo haya una instancia del sistema, el Observer notifica a los usuarios sobre cambios en el estado del pedido, y el State gestiona los diferentes estados posibles de un pedido.

1. Singleton

Singleton es un patrón de diseño creacional que nos permite asegurarnos de que una clase tenga una única instancia, a la vez que proporciona un punto de acceso global a dicha instancia.

Garantía de una Única Instancia

El patrón Singleton garantiza que solo haya una instancia de la clase **SistemaSeguimientoPedidos**. Esto es esencial para mantener una única fuente de gestión del sistema, evitando problemas de concurrencia y asegurando la coherencia en el manejo de los pedidos, así también aplique este patrón en las instancias de las implementaciones de los diferentes estados de pedido.

2. Observer

El patrón Observer pertenece a la categoría de patrones de comportamiento. Este patrón define una relación de uno a muchos entre objetos de modo que cuando un objeto cambia de estado, todos sus dependientes, llamados **observadores**, son notificados y actualizados automáticamente.

Notificación Eficiente

El patrón Observer permite notificar a múltiples observadores (usuarios) sobre cambios en el estado del pedido. Esto mejora la eficiencia al evitar la necesidad de un acoplamiento directo entre el sujeto (pedido) y los observadores, permitiendo una extensión fácil de nuevos observadores sin modificar el sujeto.

Características del Patrón Observer:

Desacoplamiento entre Sujeto y Observadores

Permite que el sujeto (objeto observado) y los observadores (objetos que necesitan ser notificados) operen de forma independiente.

Soporte para Múltiples Observadores

Puedes tener múltiples observadores registrados para un único sujeto.

Notificación Automática

Cuando el sujeto cambia su estado, todos los observadores registrados son notificados automáticamente sin que el sujeto tenga que conocer la identidad de los observadores.

Fácil Extensión

Puedes añadir nuevos observadores o cambiar el comportamiento de los existentes sin modificar el sujeto.

Participantes del Patrón Observer

Sujeto (Observable)

Mantiene una lista de observadores y notifica a estos observadores sobre cualquier cambio de estado.

Observador

Define una interfaz de actualización para los objetos que deben ser notificados sobre los cambios de estado en el sujeto.

Observadores Concretos

Implementan la interfaz de observador y reciben notificaciones del sujeto cuando se produce un cambio.

3. State

El patrón de diseño State pertenece a la categoría de patrones de comportamiento. Este patrón se utiliza cuando un objeto debe cambiar su comportamiento cuando su estado interno cambia. En lugar de tener múltiples condicionales que verifican el estado actual, el patrón State encapsula los diferentes estados en clases separadas y delega el comportamiento a estos objetos de estado.

Manejo Modular de Estados

El patrón State facilita la gestión modular de los estados del pedido. Cada estado (pendiente, en proceso, entregado, etc) está encapsulado en una clase separada, lo que simplifica la extensión del sistema al agregar nuevos estados. Además, evita la necesidad de condicionales múltiples para gestionar el comportamiento en función del estado.

Fácil Extensión

Agregar nuevos estados o modificar el comportamiento de un estado existente es más sencillo con el patrón State. Puedes crear nuevas clases de estado sin cambiar las clases existentes, lo que facilita la escalabilidad y el mantenimiento del sistema.

Claridad en la Lógica del Estado

Cada clase de estado encapsula su propia lógica de procesamiento, lo que mejora la claridad y mantenibilidad del código. Las transiciones de estado se gestionan de manera centralizada en el sujeto (pedido), simplificando la comprensión y la modificación del comportamiento del sistema.

Características del Patrón State

Encapsulamiento de Estados:

Cada estado es representado por una clase separada, encapsulando su comportamiento único.

Desacoplamiento:

El objeto contexto no tiene que conocer los detalles internos de cada estado. Se comunica con el estado a través de una interfaz común.

Añadir Nuevos Estados Fácilmente:

Se pueden agregar nuevos estados simplemente creando una nueva clase de estado y agregando la lógica específica.

Facilita la Transición de Estados:

Permite que un objeto cambie su comportamiento alterando su estado interno sin cambiar su interfaz.

Participantes del Patrón State

Contexto

Mantiene una referencia a la instancia de estado actual y delega a esa instancia cuando se requiere un comportamiento específico.

Estado

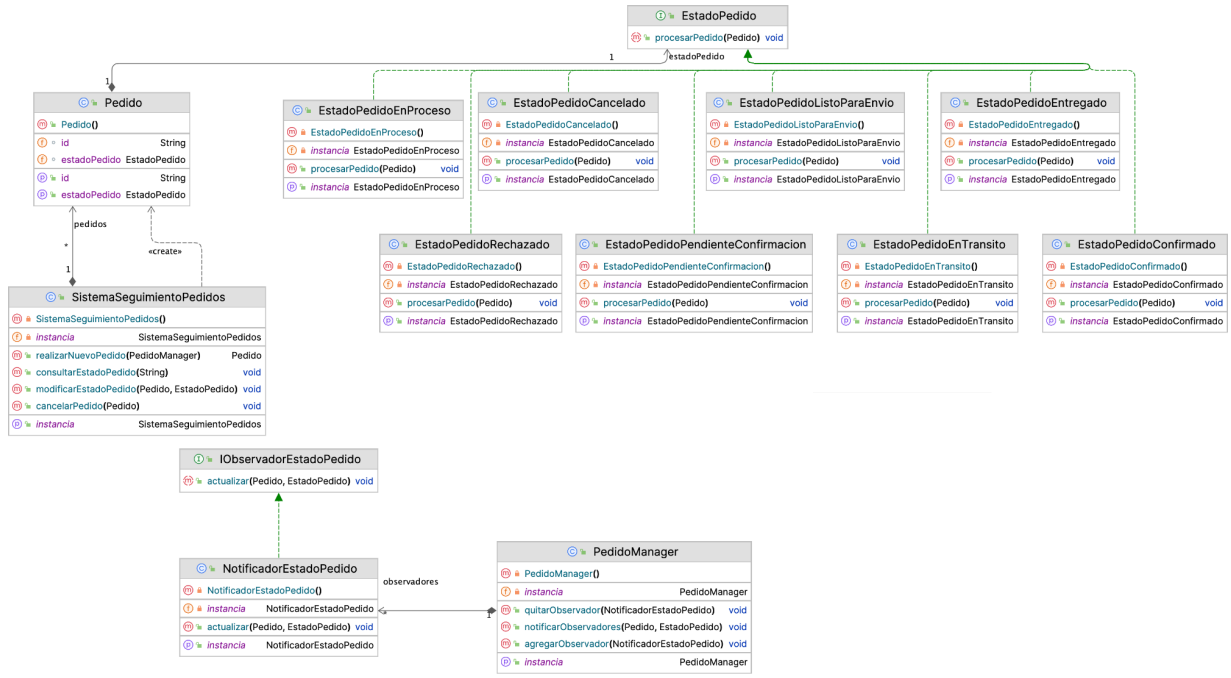
Define una interfaz para encapsular el comportamiento asociado con un estado particular del contexto.

Estados Concretos

Implementan la interfaz de estado y proporcionan la implementación específica del comportamiento asociado con el estado.

La aplicación de estos patrones en el sistema aporta claridad, modularidad, reutilización y mantenibilidad al diseño del sistema de seguimiento de estado de pedidos. Además, facilita la extensión del sistema para adaptarse a futuros cambios o requerimientos.

Diagrama UML Sistema seguimiento de estados de pedidos



Bibliografía

https://refactoring.guru/es/design-patterns/state

https://refactoring.guru/es/design-patterns/observer

https://refactoring.guru/es/design-patterns/singleton

https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/que-es-el-patron-observer/

https://es.wikipedia.org/