Universidad Dr. Andrés Bello
La Universidad de la innovación

MÓDULO IV: PATRONES DE ESTRUCTURA DE SOFTWARE | PROYECTO FINAL

Docente: Jose Rosalio Serrano Carvajal

Integrantes (Regional Sonsonate):

- Mario Augusto Lúe Morales
- Wilber Denilson Lopez Perez
- Fredy Alberto Benitez Gomez
- Felix Gerardo Granadino Rauda

Indicaciones:

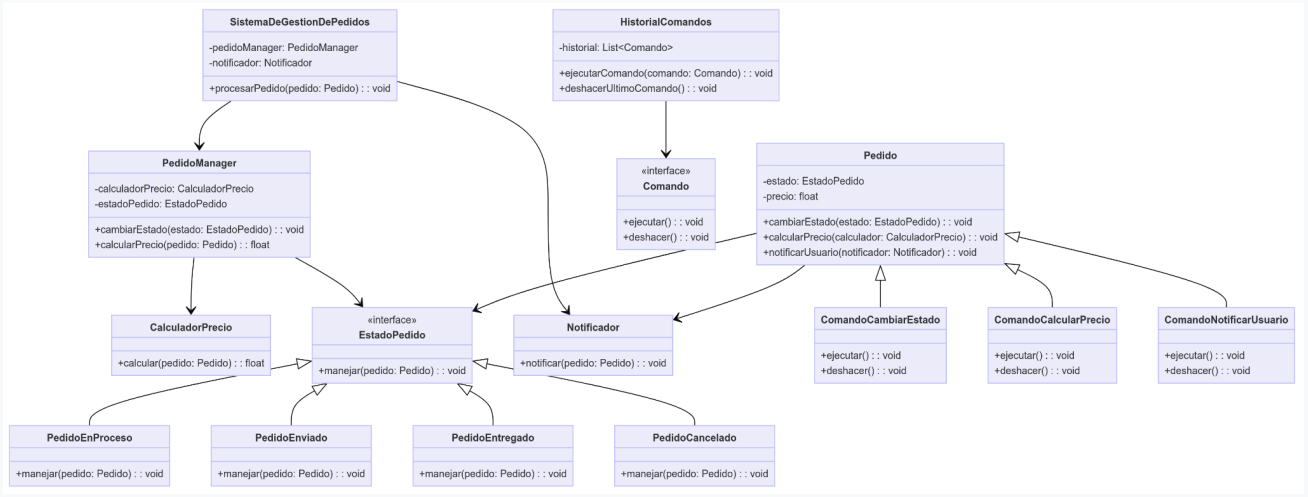
Los estudiantes desarrollarán un sistema de gestión de pedidos para una tienda en línea. El sistema deberá implementar varios patrones de diseño de comportamiento para manejar diferentes aspectos del proceso de pedidos, como el cálculo de precios, el estado de los pedidos, la notificación a los usuarios, y la gestión de comandos para operaciones como deshacer y rehacer cambios.

Se deben elaborar los diagramas de comportamiento para cada caso, basado en el mismo sistema con diferente comportamiento. Se solicita lo siguiente:

1. Diseño de clases **(por patrón de comportamiento)**
2. Diseño de diagrama de flujo **(por patrón de comportamiento)**
3. Diseño de diagrama de proceso **(por patrón de comportamiento)**
4. Diagrama general del sistema **(único)**

Lea y diseñe los diagramas de los siguientes casos específicos del sistema propiamente dicho:

Diagrama General del SIstema



Explicación del diagrama general

1. SistemaDeGestionDePedidos

- **Descripción**: Es la clase principal del sistema.
- **Atributos**:
  - -pedidoManager: PedidoManager: Un objeto que gestiona las operaciones relacionadas con los pedidos.
  - -notificador: Notificador: Un objeto encargado de enviar notificaciones a los usuarios.
- **Métodos**:
  - +procesarPedido(pedido: Pedido): void: Método que procesa un pedido utilizando PedidoManager y notifica al usuario usando Notificador.

2. PedidoManager

- **Descripción**: Clase responsable de manejar los pedidos, incluyendo el cálculo de precios y la gestión de estados.
- **Atributos**:
  - -calculadorPrecio: CalculadorPrecio: Un objeto que calcula el precio del pedido.
  - -estadoPedido: EstadoPedido: Un objeto que representa el estado actual del pedido.
- **Métodos**:
  - +cambiarEstado(estado: EstadoPedido): void: Cambia el estado del pedido.
  - +calcularPrecio(pedido: Pedido): float: Calcula el precio del pedido utilizando CalculadorPrecio.

3. CalculadorPrecio

- **Descripción**: Clase encargada de calcular el precio de un pedido.
- **Método**:
  - +calcular(pedido: Pedido): float: Calcula y devuelve el precio del pedido.

4. EstadoPedido (Interfaz)

- **Descripción**: Interfaz para los diferentes estados de un pedido.
- **Método**:
  - +manejar(pedido: Pedido): void: Método que gestiona el pedido en un estado particular.

5. Pedido

- **Descripción**: Clase que representa un pedido en el sistema.
- **Atributos**:
  - -estado: EstadoPedido: El estado actual del pedido.
  - -precio: float: El precio del pedido.
- **Métodos**:
  - +cambiarEstado(estado: EstadoPedido): void: Cambia el estado del pedido.
  - +calcularPrecio(calculador: CalculadorPrecio): void: Calcula el precio del pedido utilizando CalculadorPrecio.
  - +notificarUsuario(notificador: Notificador): void: Notifica al usuario sobre el pedido utilizando Notificador.

6. Notificador

- **Descripción**: Clase encargada de enviar notificaciones.
- **Método**:
  - +notificar(pedido: Pedido): void: Envía una notificación sobre el pedido.

7. Comando (Interfaz)

- **Descripción**: Interfaz para los comandos que se pueden ejecutar, deshacer o rehacer.
- **Métodos**:
  - +ejecutar(): void: Ejecuta el comando.
  - +deshacer(): void: Deshace el comando.

8. HistorialComandos

- **Descripción**: Clase que maneja un historial de comandos, permitiendo deshacer la última operación.
- **Atributos**:
  - -historial: List~Comando~: Lista de comandos ejecutados.
- **Métodos**:

- +ejecutarComando(comando: Comando): void: Ejecuta un comando y lo agrega al historial.
- +deshacerUltimoComando(): void: Deshace el último comando en el historial.

9. Estados de Pedido

- **Descripción**: Clases que implementan la interfaz EstadoPedido y representan los diferentes estados posibles de un pedido.
- **Clases**:
  - PedidoEnProceso
  - PedidoEnviado
  - PedidoEntregado
  - PedidoCancelado
- **Método**:
  - +manejar(pedido: Pedido): void: Implementa el manejo específico del pedido en ese estado.

10. Comandos

- **Descripción**: Clases que implementan la interfaz Comando para realizar operaciones específicas sobre los pedidos.
- **Clases**:
  - ComandoCambiarEstado: Cambia el estado del pedido.
  - ComandoCalcularPrecio: Calcula el precio del pedido.
  - ComandoNotificarUsuario: Notifica al usuario sobre el pedido.
- **Métodos**:
  - +ejecutar(): void: Ejecuta la acción del comando.
  - +deshacer(): void: Deshace la acción del comando.

Relaciones entre Clases

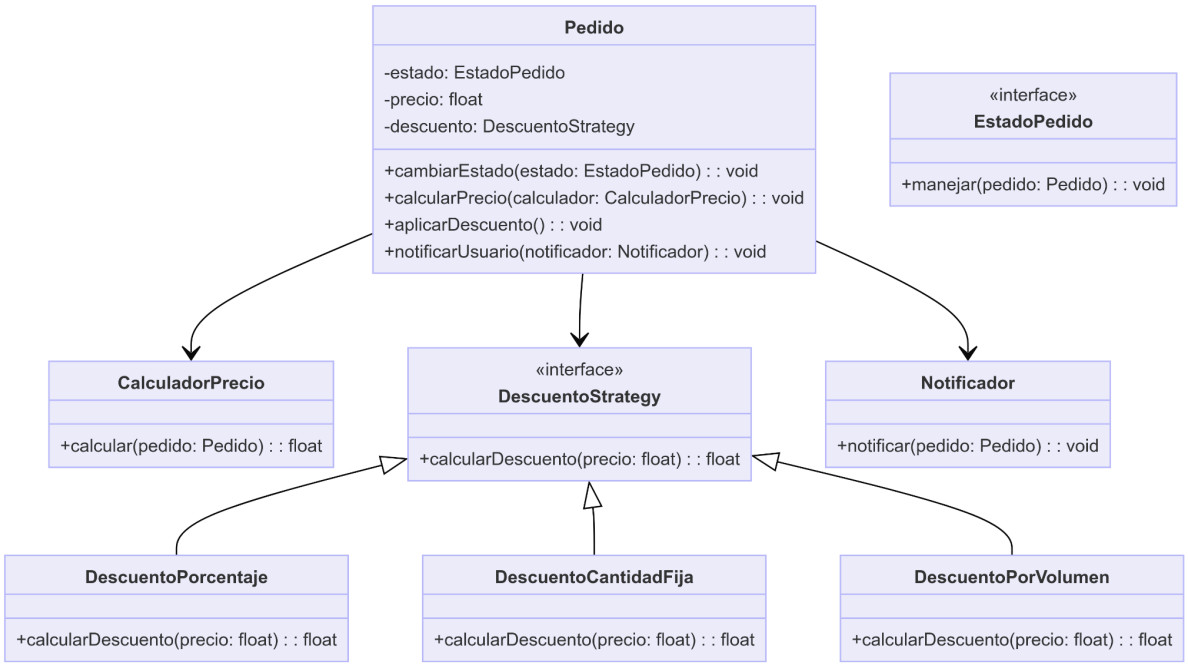
- **SistemaDeGestionDePedidos** utiliza **PedidoManager** y **Notificador** para manejar y notificar los pedidos.
- **PedidoManager** usa **CalculadorPrecio** para calcular precios y **EstadoPedido** para manejar los estados de los pedidos.
- **Pedido** está asociado a **EstadoPedido** para representar su estado actual y a **Notificador** para enviar notificaciones.
- **HistorialComandos** usa la interfaz **Comando** para gestionar y deshacer comandos.

Cada componente tiene una responsabilidad específica y se relaciona con otros para formar un sistema cohesivo para la gestión de pedidos en una tienda en línea.

Patrón Strategy para el Cálculo de Descuentos:

- Implementar diferentes estrategias de descuento (porcentaje, cantidad fija, por volumen de compra).

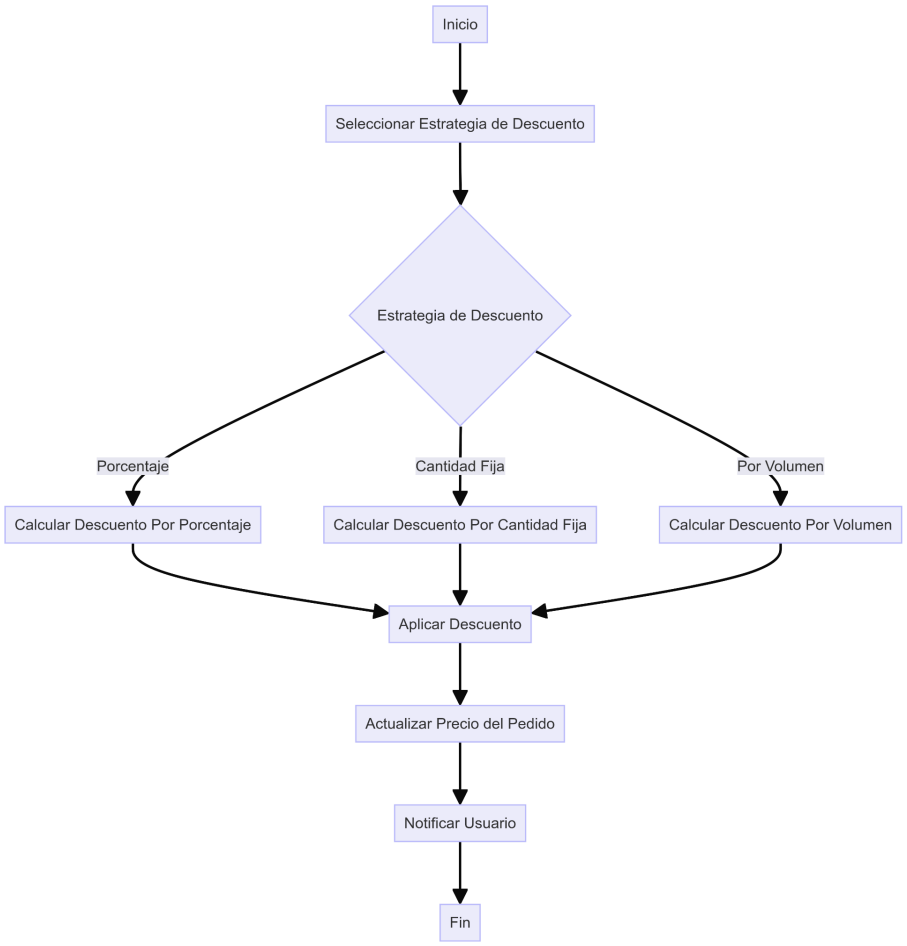
Diseño de Clases



Explicación del diagrama

- **Pedido**: Tiene un atributo `descuento` de tipo `DescuentoStrategy`. Dependiendo de la estrategia de descuento, este objeto puede ser `DescuentoPorcentaje`, `DescuentoCantidadFija`, o `DescuentoPorVolumen`.
- **DescuentoStrategy**: Interfaz que define el método `calcularDescuento`, que todas las estrategias de descuento deben implementar.
- **DescuentoPorcentaje**, **DescuentoCantidadFija**, **DescuentoPorVolumen**: Implementaciones concretas de `DescuentoStrategy`, cada una calculando el descuento de manera diferente.
- **EstadoPedido** y **Notificador** se mantienen igual que en el diagrama general.

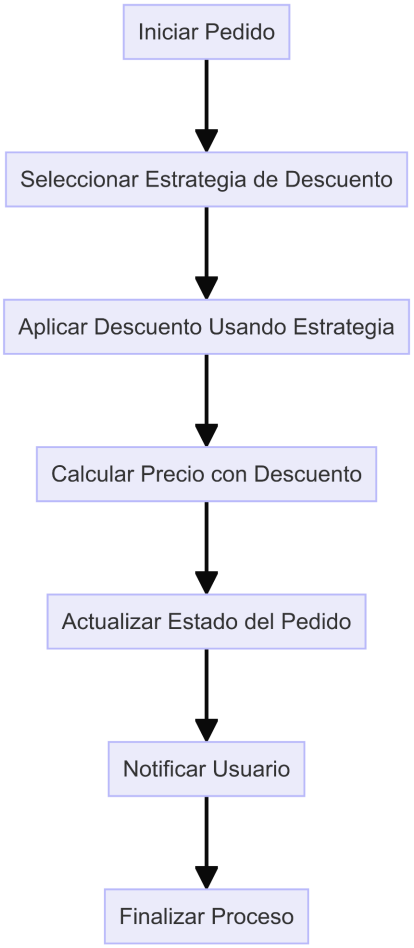
Diseño de Diagrama de Flujos



Explicación del diagrama

- **Inicio**: Comienza el proceso de descuento.
- **Seleccionar Estrategia de Descuento**: El usuario elige el tipo de descuento.
- **Calcular Descuento**: Dependiendo de la estrategia seleccionada, se calcula el descuento correspondiente.
- **Aplicar Descuento**: El descuento calculado se aplica al precio del pedido.
- **Actualizar Precio del Pedido**: El precio del pedido se actualiza con el descuento.
- **Notificar Usuario**: Se notifica al usuario sobre el cambio en el precio.

Diseño de Diagrama de Procesos



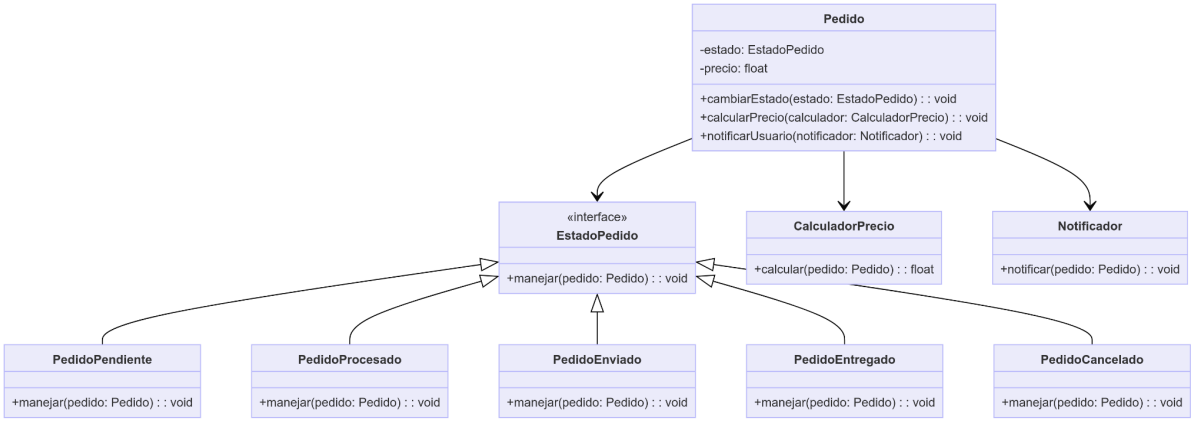
Explicación del diagrama

- **Iniciar Pedido**: Comienza el proceso con la creación de un pedido.
- **Seleccionar Estrategia de Descuento**: Se decide qué estrategia de descuento se aplicará.
- **Aplicar Descuento Usando Estrategia**: Se aplica la estrategia de descuento al pedido.
- **Calcular Precio con Descuento**: Se calcula el precio final después de aplicar el descuento.
- **Actualizar Estado del Pedido**: Se actualiza el estado del pedido en el sistema.
- **Notificar Usuario**: El usuario recibe una notificación sobre el estado actualizado y el precio del pedido.
- **Finalizar Proceso**: Se completa el proceso de gestión del pedido.

Patrón State para el Ciclo de Vida de los Pedidos:

- Manejar los diferentes estados de un pedido (pendiente, procesado, enviado, entregado, cancelado).

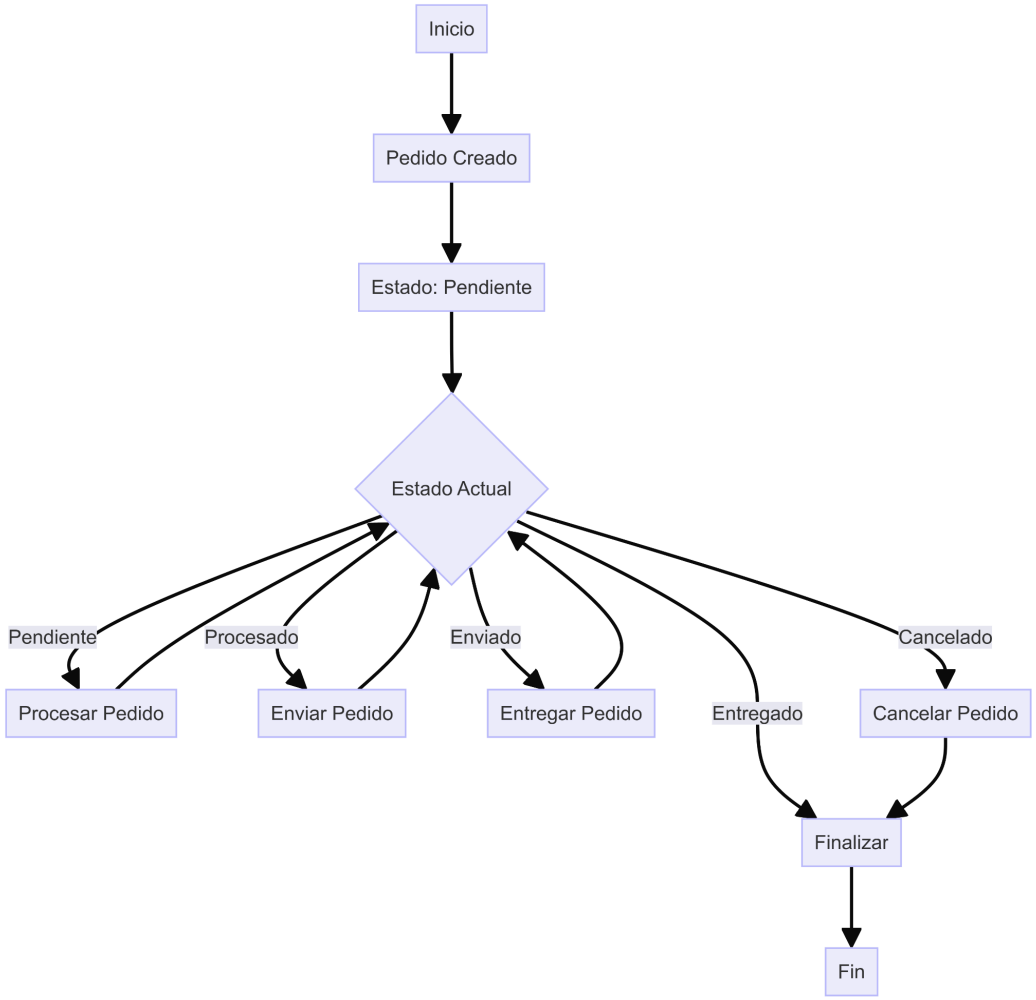
Diseño de Clases



Explicación del diagrama

Pedido: Clase principal que maneja el estado del pedido y realiza acciones relacionadas cómo calcular el precio y notificar al usuario.
EstadoPedido: Interfaz que define el método `manejar` que cada estado específico debe implementar.
PedidoPendiente, **PedidoProcesado**, **PedidoEnviado**, **PedidoEntregado**, **PedidoCancelado**: Implementaciones concretas de `EstadoPedido` que representan los diferentes estados posibles de un pedido.
CalculadorPrecio y **Notificador** se mantienen como en el diagrama general.

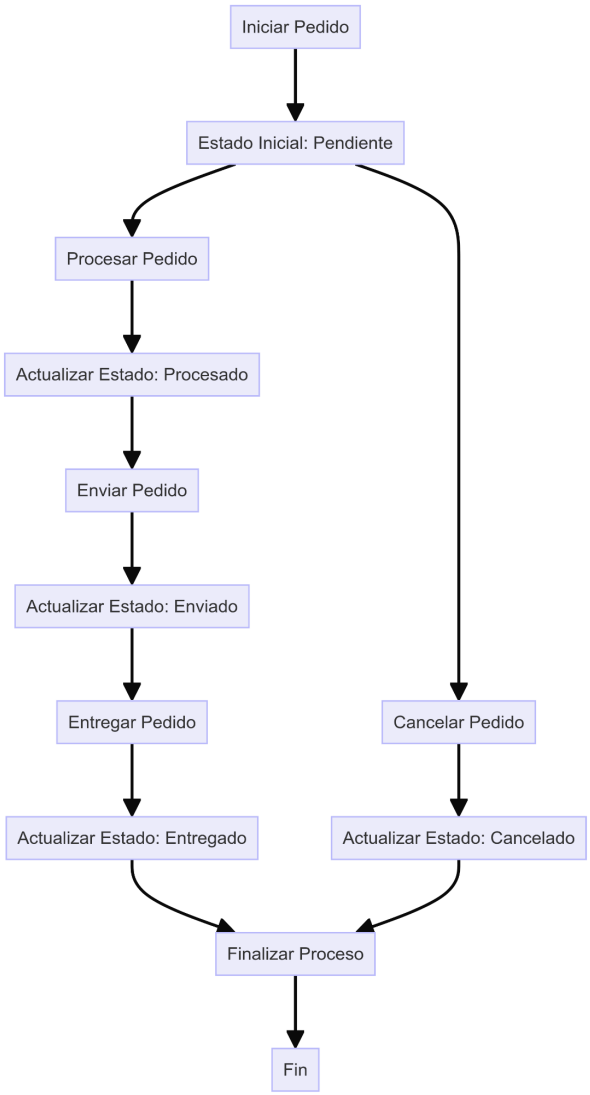
Diseño de Diagrama de Flujos



Explicación del diagrama

- **Inicio**: Comienza el proceso de manejo del pedido.
- **Pedido Creado**: El pedido es creado y se establece en el estado "Pendiente".
- **Estado: Pendiente**: El estado inicial del pedido.
- **Procesar Pedido**: El pedido se procesa y cambia a "Procesado".
- **Enviar Pedido**: El pedido se envía y cambia a "Enviado".
- **Entregar Pedido**: El pedido se entrega y cambia a "Entregado".
- **Cancelar Pedido**: Alternativa para cancelar el pedido en cualquier estado.
- **Finalizar**: Finaliza el proceso de gestión del pedido.
- **Fin**: El proceso de gestión del pedido ha concluido.

Diseño de Diagrama de Procesos



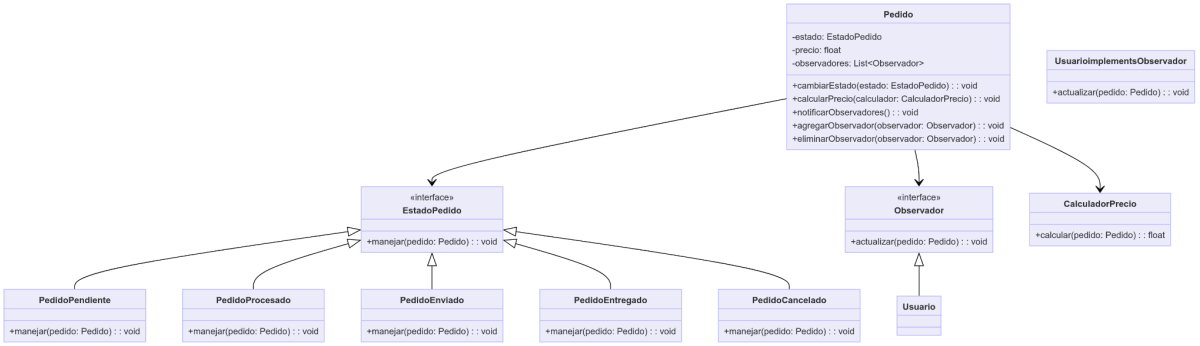
Explicación del diagrama

- **Iniciar Pedido**: Comienza el proceso con la creación de un pedido.
- **Estado Inicial: Pendiente**: El pedido comienza en el estado "Pendiente".
- **Procesar Pedido**: Se procesa el pedido y se actualiza su estado a "Procesado".
- **Enviar Pedido**: Se envía el pedido y se actualiza su estado a "Enviado".
- **Entregar Pedido**: Se entrega el pedido y se actualiza su estado a "Entregado".
- **Finalizar Proceso**: Concluye el proceso de gestión del pedido.
- **Cancelar Pedido**: Alternativa para cancelar el pedido en cualquier momento.
- **Actualizar Estado: Cancelado**: Actualiza el estado del pedido a "Cancelado" si se cancela.

Patrón Observer para las Notificaciones de Pedidos:

- Notificar a los usuarios sobre cambios en el estado de sus pedidos.

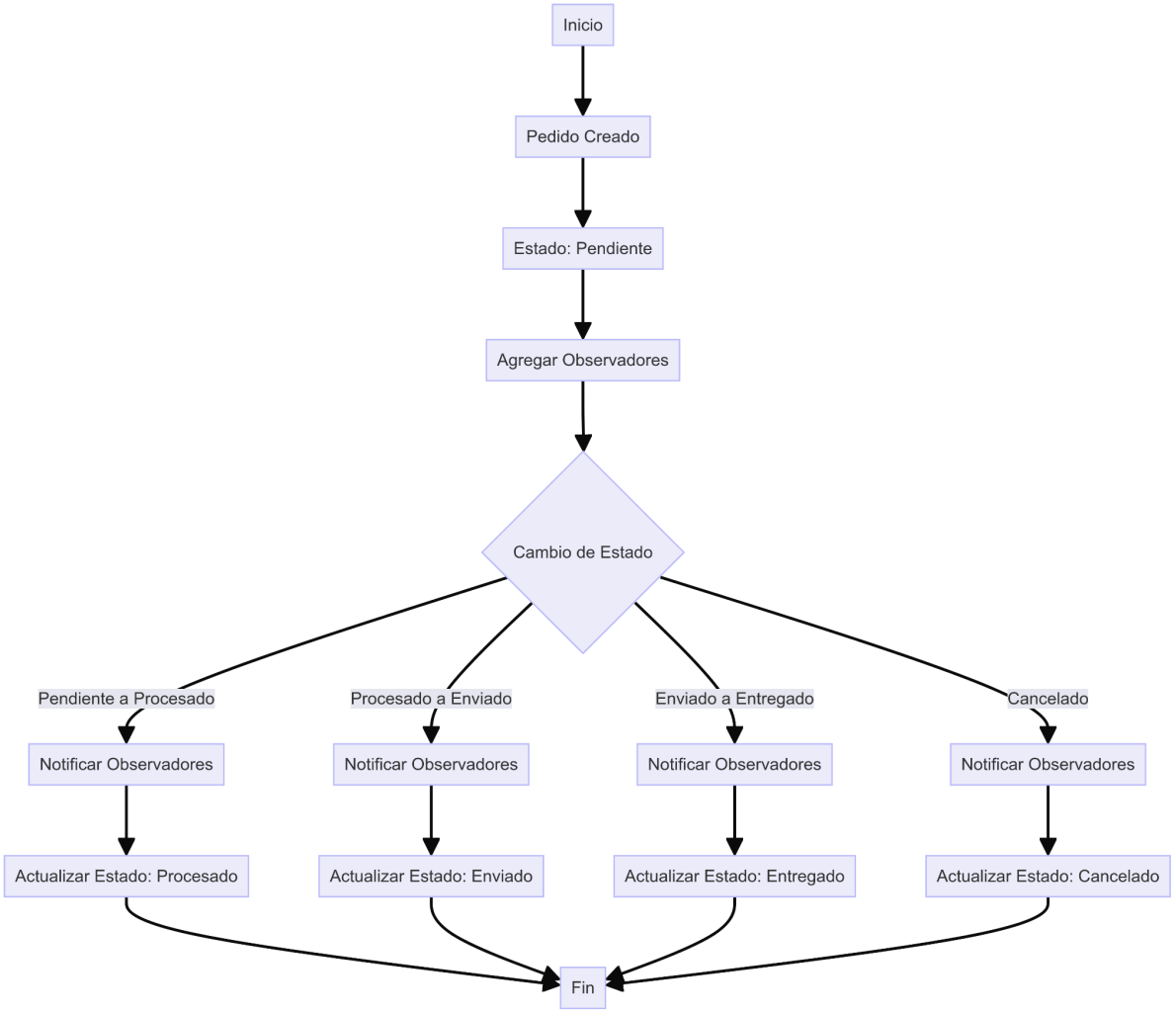
Diseño de Clases



Explicación del diagrama

- **Pedido**: Clase principal que mantiene una lista de observadores (usuarios). Proporciona métodos para agregar, eliminar y notificar a los observadores.
- **EstadoPedido**: Interfaz que define el método `manejar`, implementado por los estados específicos del pedido.
- **PedidoPendiente**, **PedidoProcesado**, **PedidoEnviado**, **PedidoEntregado**, **PedidoCancelado**: Implementaciones concretas de `EstadoPedido`.
- **Observador**: Interfaz que define el método `actualizar`, que los observadores deben implementar para recibir actualizaciones.
- **Usuario**: Implementa la interfaz `Observador` y recibe actualizaciones sobre los cambios en el estado del pedido.
- **CalculadorPrecio**: Clase que calcula el precio del pedido.

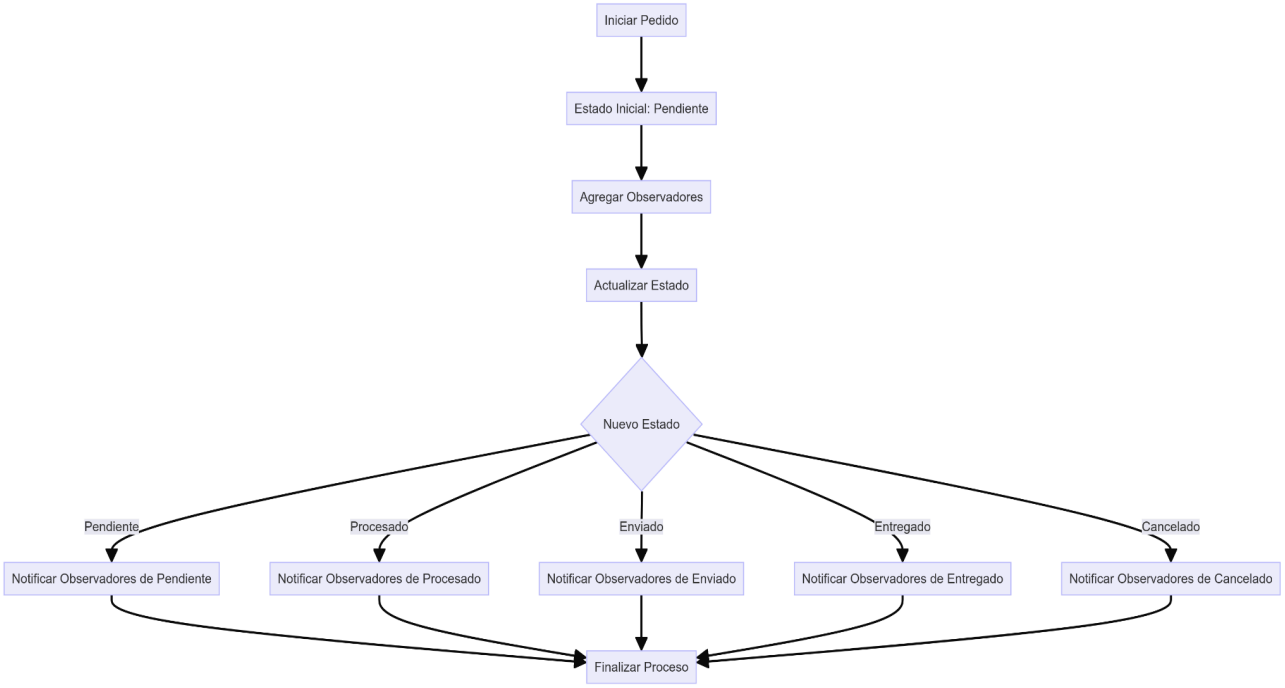
Diseño de Diagrama de Flujos



Explicación del diagrama

- **Inicio**: Comienza el proceso de manejo del pedido.
- **Pedido Creado**: Se crea el pedido y se establece en el estado "Pendiente".
- **Estado: Pendiente**: El estado inicial del pedido.
- **Agregar Observadores**: Se agregan observadores (usuarios) al pedido.
- **Cambio de Estado**: Dependiendo del cambio de estado, se notifica a los observadores.
- **Notificar Observadores**: Se envía una notificación a los observadores sobre el cambio de estado.
- **Actualizar Estado**: El estado del pedido se actualiza.
- **Fin**: El proceso de notificación y actualización del estado del pedido se completa.

Diseño de Diagrama de Procesos



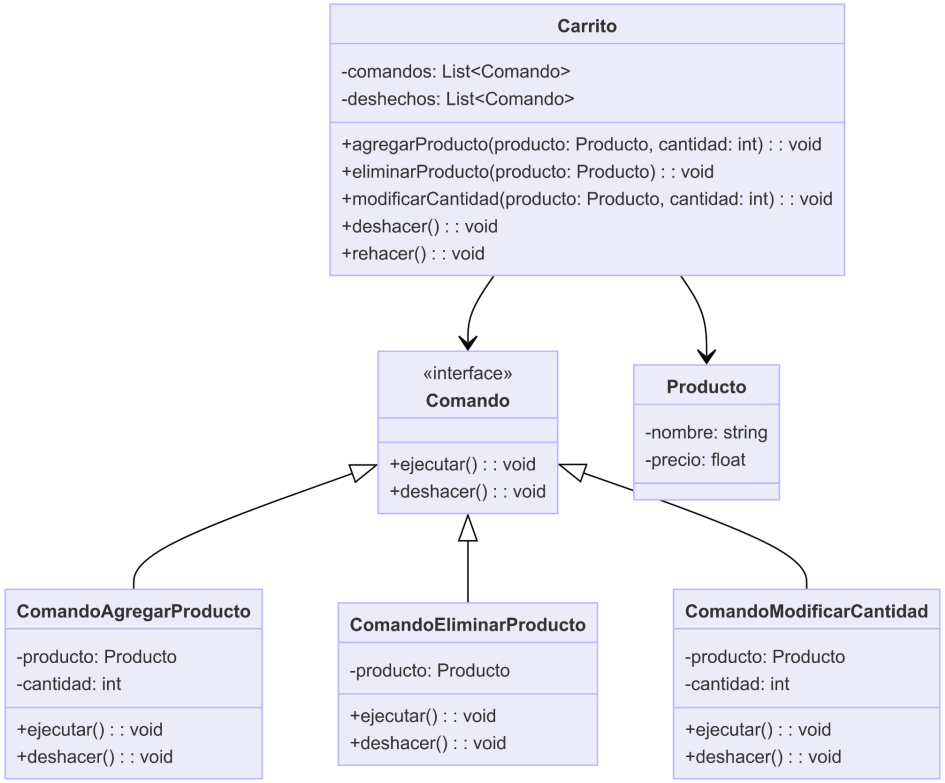
Explicación del diagrama

- **Iniciar Pedido**: Comienza el proceso con la creación de un pedido.
- **Estado Inicial: Pendiente**: El pedido comienza en el estado "Pendiente".
- **Agregar Observadores**: Se agregan observadores (usuarios) para recibir notificaciones.
- **Actualizar Estado**: El estado del pedido cambia.
- **Notificar Observadores**: Los observadores reciben notificaciones sobre el nuevo estado del pedido.
- **Finalizar Proceso**: Concluye el proceso de gestión del pedido y notificación.

Patrón Command para Operaciones de Deshacer y Rehacer:

- Implementar comandos para operaciones como agregar productos al carrito, eliminar productos del carrito, y modificar la cantidad de productos.
- Implementar una funcionalidad de deshacer y rehacer utilizando una pila de comandos.

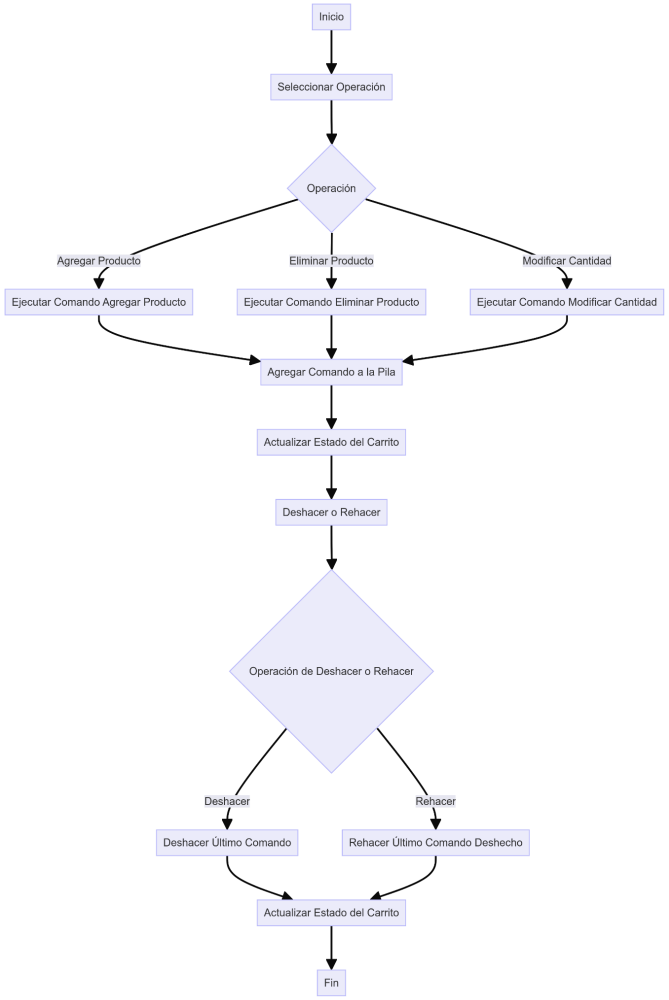
Diseño de Clases



Explicación del diagrama

- **Carrito**: Clase principal que maneja los productos en el carrito y las operaciones. Mantiene pilas de comandos y comandos deshechos para realizar deshacer y rehacer.
- **Comando**: Interfaz que define los métodos `ejecutar` y `deshacer`, que deben implementar todos los comandos concretos.
- **ComandoAgregarProducto**: Implementación concreta de `Comando` para agregar productos al carrito.
- **ComandoEliminarProducto**: Implementación concreta de `Comando` para eliminar productos del carrito.
- **ComandoModificarCantidad**: Implementación concreta de `Comando` para modificar la cantidad de un producto en el carrito.
- **Producto**: Clase que representa un producto con atributos como nombre y precio.

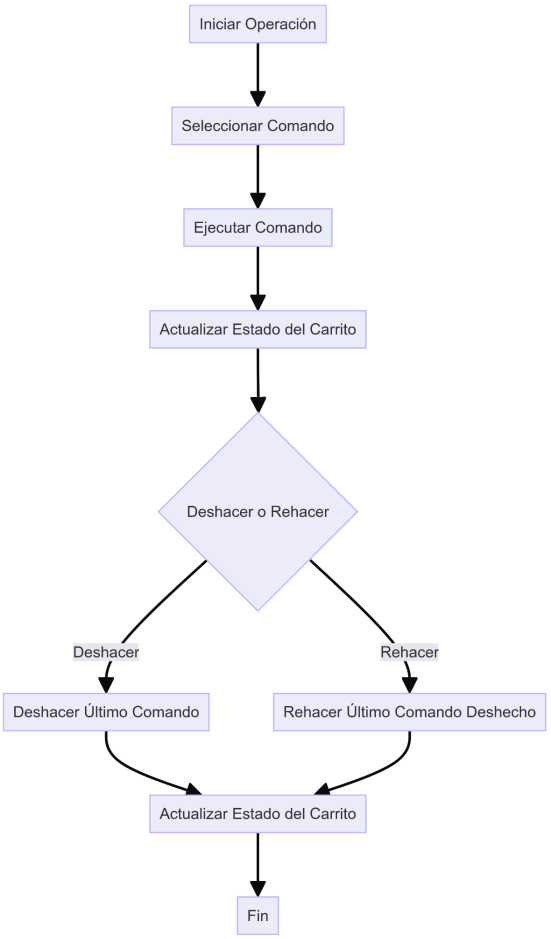
Diseño de Diagrama de Flujos



Explicación del diagrama

- **Inicio**: Comienza el proceso de manejo de operaciones en el carrito.
- **Seleccionar Operación**: El usuario selecciona la operación que desea realizar (agregar, eliminar o modificar productos).
- **Ejecutar Comando**: Se ejecuta el comando correspondiente para la operación seleccionada.
- **Agregar Comando a la Pila**: El comando ejecutado se agrega a la pila de comandos para posibles operaciones de deshacer.
- **Actualizar Estado del Carrito**: El estado del carrito se actualiza en función de la operación realizada.
- **Deshacer o Rehacer**: El usuario puede deshacer o rehacer la última operación.
- **Deshacer Último Comando**: Se deshace el último comando ejecutado y se actualiza el estado del carrito.
- **Rehacer Último Comando Deshecho**: Se rehace el último comando deshecho y se actualiza el estado del carrito.
- **Fin**: El proceso de manejo de operaciones y deshacer/rehacer se completa.

Diseño de Diagrama de Procesos



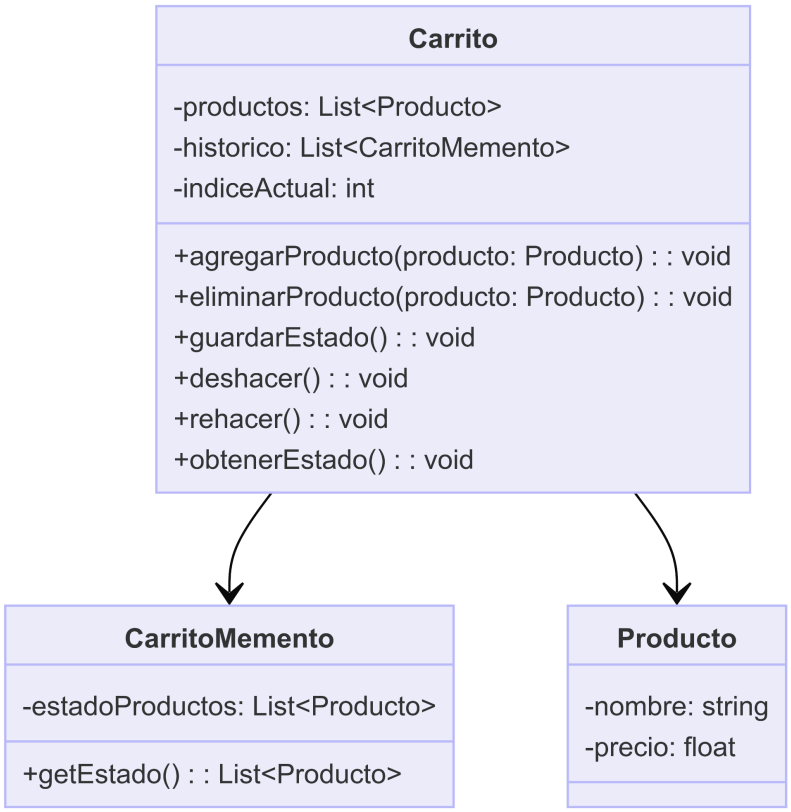
Explicación del diagrama

- **Iniciar Operación**: Comienza el proceso con la selección de una operación (agregar, eliminar o modificar productos).
- **Seleccionar Comando**: Se selecciona el comando que representa la operación deseada.
- **Ejecutar Comando**: Se ejecuta el comando seleccionado.
- **Actualizar Estado del Carrito**: El estado del carrito se actualiza en función de la operación realizada.
- **Deshacer o Rehacer**: El usuario puede optar por deshacer o rehacer la última operación.
- **Deshacer Último Comando**: Se deshace la última operación y se actualiza el estado del carrito.
- **Rehacer Último Comando Deshecho**: Se rehace la última operación deshecha y se actualiza el estado del carrito.
- **Fin**: Concluye el proceso de gestión de operaciones en el carrito.

Patrón Memento para el Estado del Carrito de Compras:

- Guardar el estado del carrito de compras y permitir al usuario deshacer cambios.

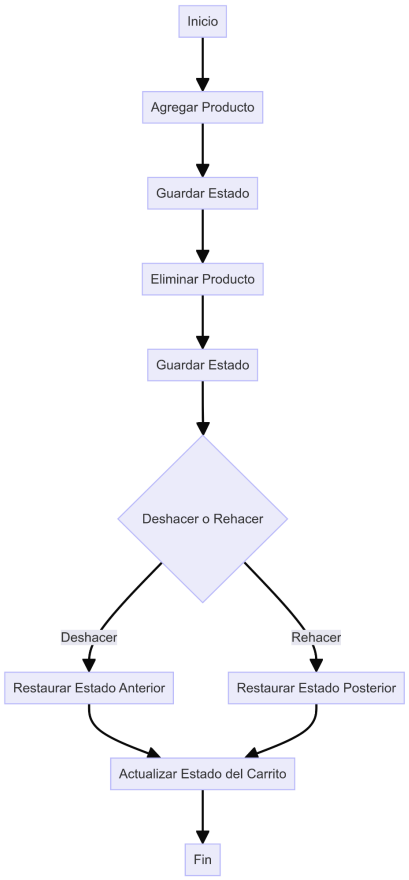
Diseño de Clases



Explicación del diagrama

- **Carrito**: Clase principal que maneja los productos en el carrito y el historial de estados. Proporciona métodos para agregar, eliminar, guardar el estado actual, deshacer y rehacer.
- **Carrito Memento**: Clase que guarda el estado del carrito en un momento específico. Proporciona un método para obtener el estado guardado.
- **Producto**: Clase que representa un producto con atributos como nombre y precio.

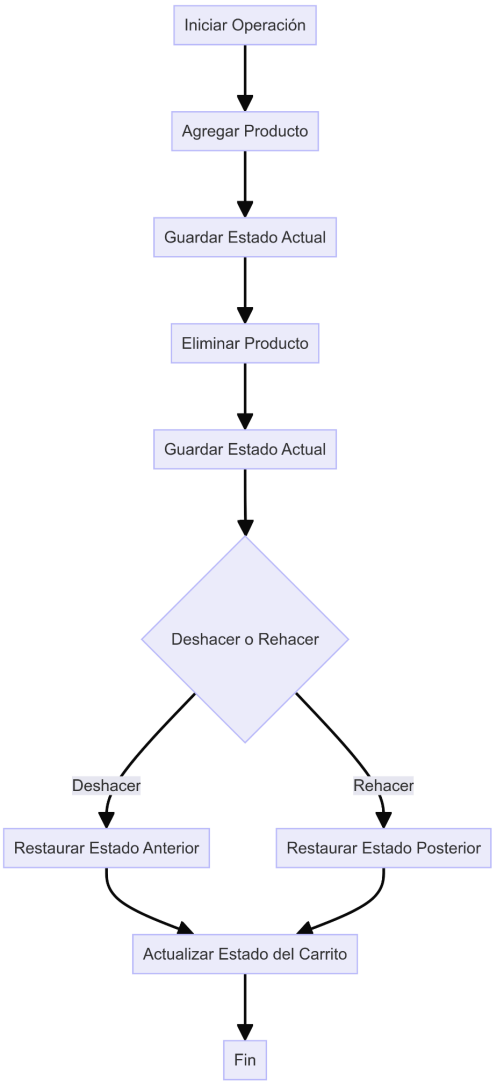
Diseño de Diagrama de Flujos



Explicación del diagrama

- **Inicio**: Comienza el proceso de manejo de operaciones en el carrito.
- **Agregar Producto**: Se agrega un producto al carrito.
- **Guardar Estado**: Se guarda el estado actual del carrito.
- **Eliminar Producto**: Se elimina un producto del carrito.
- **Guardar Estado**: Se guarda el estado actual del carrito después de eliminar un producto.
- **Deshacer o Rehacer**: El usuario puede optar por deshacer o rehacer la última operación.
- **Restaurar Estado Anterior**: Se restaura el estado del carrito al momento anterior a la última operación.
- **Restaurar Estado Posterior**: Se restaura el estado del carrito al momento posterior a la última operación deshecha.
- **Actualizar Estado del Carrito**: El estado del carrito se actualiza.
- **Fin**: El proceso de manejo de operaciones y deshacer/rehacer se completa.

Diseño de Diagrama de Procesos



Explicación del diagrama

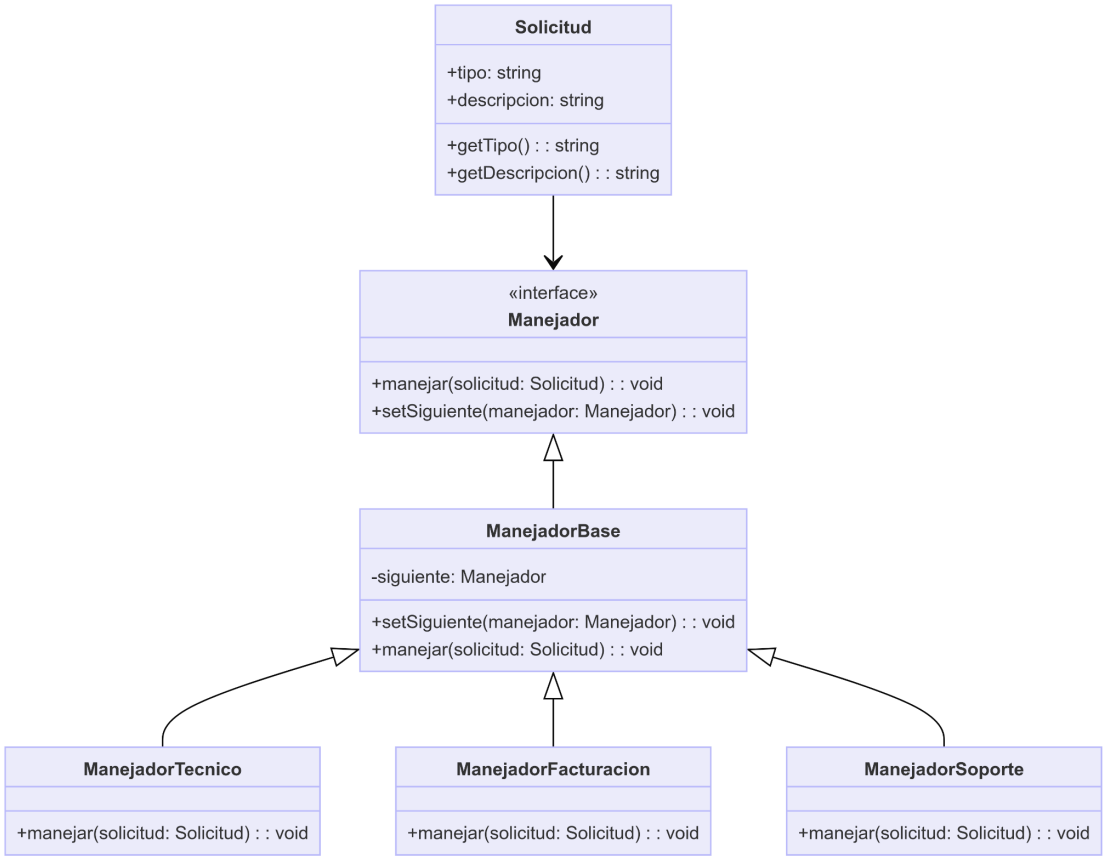
- **Iniciar Operación**: Comienza el proceso con la selección de una operación (agregar, eliminar productos).
- **Agregar Producto**: Se agrega un producto al carrito.
- **Guardar Estado Actual**: Se guarda el estado actual del carrito.
- **Eliminar Producto**: Se elimina un producto del carrito.
- **Guardar Estado Actual**: Se guarda el estado actual del carrito después de eliminar un producto.
- **Deshacer o Rehacer**: El usuario puede optar por deshacer o rehacer la última operación.
- **Restaurar Estado Anterior**: Se restaura el estado del carrito al momento anterior a la última operación.
- **Restaurar Estado Posterior**: Se restaura el estado del carrito al momento posterior a la última operación deshecha.

- **Actualizar Estado del Carrito**: El estado del carrito se actualiza.
- **Fin**: Concluye el proceso de gestión de operaciones en el carrito utilizando el patrón Memento.

Patrón Chain of Responsibility para el Manejo de Solicitudes:

- Manejar diferentes tipos de solicitudes de los clientes (técnicas, de facturación, de soporte).

Diseño de Clases



Explicación del diagrama

- **Solicitud**: Clase que representa una solicitud del cliente con atributos como tipo y descripción. Proporciona métodos para obtener el tipo y la descripción de la solicitud.

- **Manejador**: Interfaz que define los métodos `manejar` y `setSiguiente`, que deben implementar todos los manejadores concretos.
- **ManejadorBase**: Clase base que implementa la interfaz Manejador y proporciona la funcionalidad para establecer el siguiente manejador en la cadena.
- **ManejadorTecnico**: Implementación concreta de `ManejadorBase` para manejar solicitudes técnicas.
- **ManejadorFacturacion**: Implementación concreta de `ManejadorBase` para manejar solicitudes de facturación.
- **ManejadorSoporte**: Implementación concreta de `ManejadorBase` para manejar solicitudes de soporte.

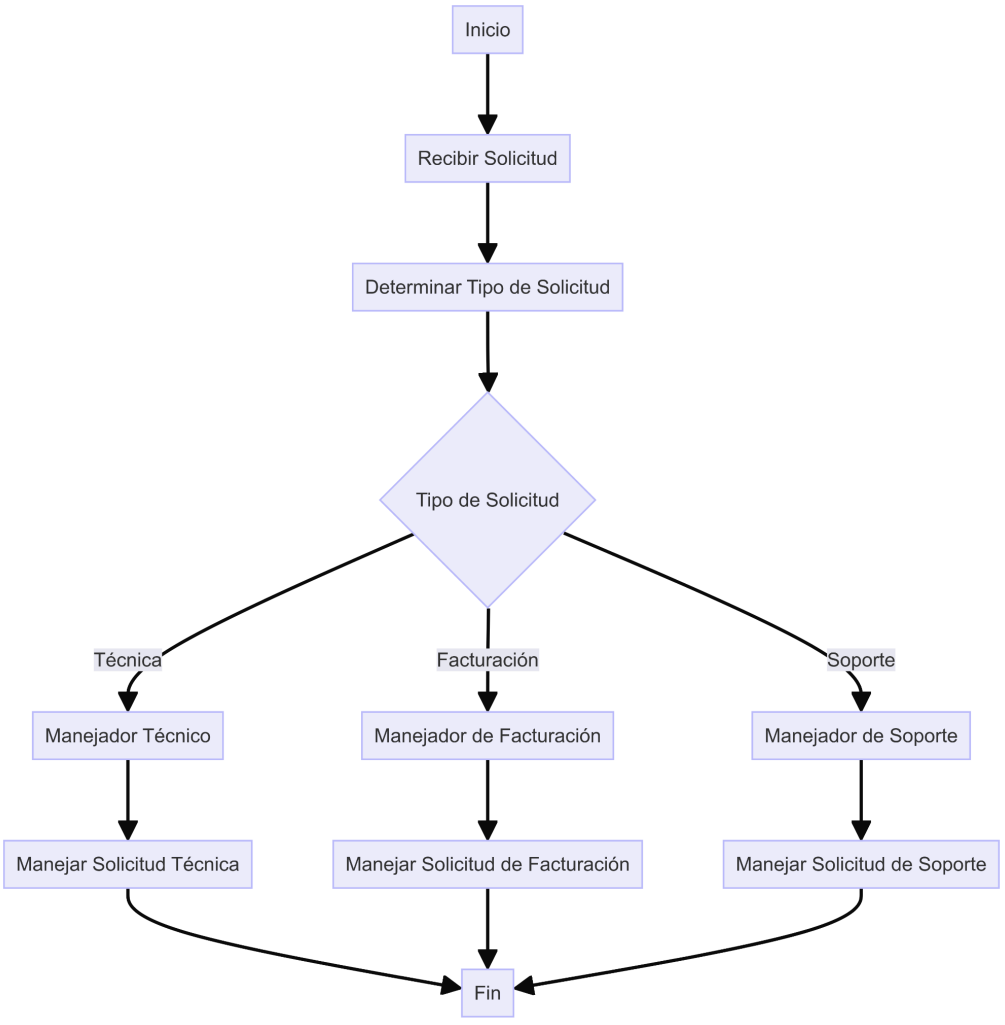
Diseño de Diagrama de Flujos

Explicación del diagrama

- **Inicio**: Comienza el proceso de manejo de solicitudes de clientes.

- **Recibir Solicitud**: Se recibe una solicitud de un cliente.
- **Determinar Tipo de Solicitud**: Se determina el tipo de solicitud.
- **Tipo de Solicitud**: Se evalúa el tipo de solicitud para decidir qué manejador debe encargarse.
- **Manejador Técnico**: Se maneja la solicitud técnica.
- **Manejador de Facturación**: Se maneja la solicitud de facturación.
- **Manejador de Soporte**: Se maneja la solicitud de soporte.
- **Fin**: El proceso de manejo de la solicitud se completa.

Diseño de Diagrama de Procesos



Explicación del diagrama

- **Inicio**: Comienza el proceso de manejo de solicitudes de clientes.
- **Recibir Solicitud**: Se recibe una solicitud de un cliente.
- **Determinar Tipo de Solicitud**: Se determina el tipo de solicitud.

- **Tipo de Solicitud**: Se evalúa el tipo de solicitud para decidir qué manejador debe encargarse.
- **Manejador Técnico**: Se maneja la solicitud técnica.
- **Manejador de Facturación**: Se maneja la solicitud de facturación.
- **Manejador de Soporte**: Se maneja la solicitud de soporte.
- **Fin**: El proceso de manejo de la solicitud se completa.

Conclusiones

En el desarrollo del sistema de gestión de pedidos para una tienda en línea, hemos implementado y diagramado varios patrones de diseño de comportamiento fundamentales para optimizar y estructurar diversos aspectos del proceso de pedidos. Utilizamos el patrón Strategy para permitir una lógica flexible y dinámica en el cálculo de descuentos, adaptándose a diferentes necesidades sin modificar el código cliente. El patrón State nos permitió gestionar eficazmente el ciclo de vida de los pedidos, asegurando una transición clara y manejable entre estados como pendiente, procesado, enviado, entregado y cancelado.

Para mejorar la comunicación con los usuarios, implementamos el patrón Observer, garantizando que las notificaciones sobre cambios en el estado de sus pedidos sean recibidas en tiempo real. Asimismo, adoptamos el patrón Command para gestionar operaciones en el carrito de compras, como agregar, eliminar y modificar productos, con capacidades de deshacer y rehacer, asegurando así una experiencia de usuario fluida y controlada. El patrón Memento fue utilizado para guardar y restaurar el estado del carrito de compras, permitiendo a los usuarios revertir cambios cuando sea necesario.

Finalmente, el patrón Chain of Responsibility fue clave para manejar diferentes tipos de solicitudes de clientes, tales como técnicas, de facturación y de soporte, permitiendo una distribución flexible y escalable de estas solicitudes a través de una cadena de responsabilidad bien definida.

En conjunto, estos patrones no solo resuelven problemas específicos del negocio, sino que también facilitan el mantenimiento y la extensibilidad del sistema, proporcionando una base sólida, robusta y escalable para manejar pedidos, notificaciones y solicitudes de clientes de manera eficiente.