

Desafío de Arquitectura de Software

Sistema de Procesamiento de Pedidos Modulares

1. Descripción General y Problema

🎯 Objetivo del Taller

Diseñar e implementar un sistema que gestione el flujo de pedidos utilizando Patrones de Diseño (Creacionales, Estructurales y Comportamentales) para lograr máxima flexibilidad y desacoplamiento. El objetivo es construir un motor de software que sea fácil de mantener y extender en el futuro.

🚀 El Problema: El Motor de Pedidos Flexible

La empresa "E-Commerce Súper Ágil" necesita un motor de pedidos que pueda manejar diferentes tipos de productos, aplicar descuentos y procesar el pago y la notificación al cliente, de forma que añadir nuevas funcionalidades o métodos de pago sea trivial sin tener que reescribir el código existente.

📋 Requisitos Funcionales

- Productos Variados:** Manejar al menos dos tipos de productos ('ProductoSimple' y 'ProductoPersonalizable').
- Modificadores Dinámicos:** Poder aplicar descuentos o servicios adicionales (envoltura de regalo, seguro) al costo final de forma dinámica.
- Pagos Flexibles:** Procesar pagos usando diferentes métodos (Tarjeta de Crédito, PayPal, Transferencia Bancaria).
- Notificaciones:** Informar al cliente una vez que el pedido ha sido procesado exitosamente.

2. Patrones de Diseño Requeridos

Haz clic en cada patrón para ver los detalles de implementación específicos para este taller.

Factory Method

Para la creación de diferentes tipos de productos.

Decorator

Para aplicar costos o servicios adicionales dinámicamente.

Strategy

Para manejar diferentes algoritmos de pago.

Descripción General

Patrones Requeridos

Evaluación y Entrega

Arquitectura Visual

Strategy

Contexto: El proceso de pago es altamente variable. El sistema debe poder cambiar el algoritmo de pago sin modificar la clase principal del pedido.

Implementación Sugerida:

- Crear una interfaz `EstrategiaDePago` con un método `pagar(monto)`.
- Crear clases concretas como `PagoConTarjeta`, `PagoConPayPal`, etc., que implementen la interfaz.
- La clase `Pedido` (o un `MotorDePedidos`) debe tener un atributo para almacenar la estrategia de pago actual.
- El método para procesar el pedido debe delegar la ejecución del pago a la estrategia seleccionada.

3. Evaluación y Entrega

Rúbrica de Evaluación

Criterio	Puntos
Implementación del Factory Method	30
Implementación del Decorator	30
Implementación del Strategy	30
Clean Code y POO (abstracción, S.R.P.)	10
TOTAL	100

Instrucciones para la Presentación

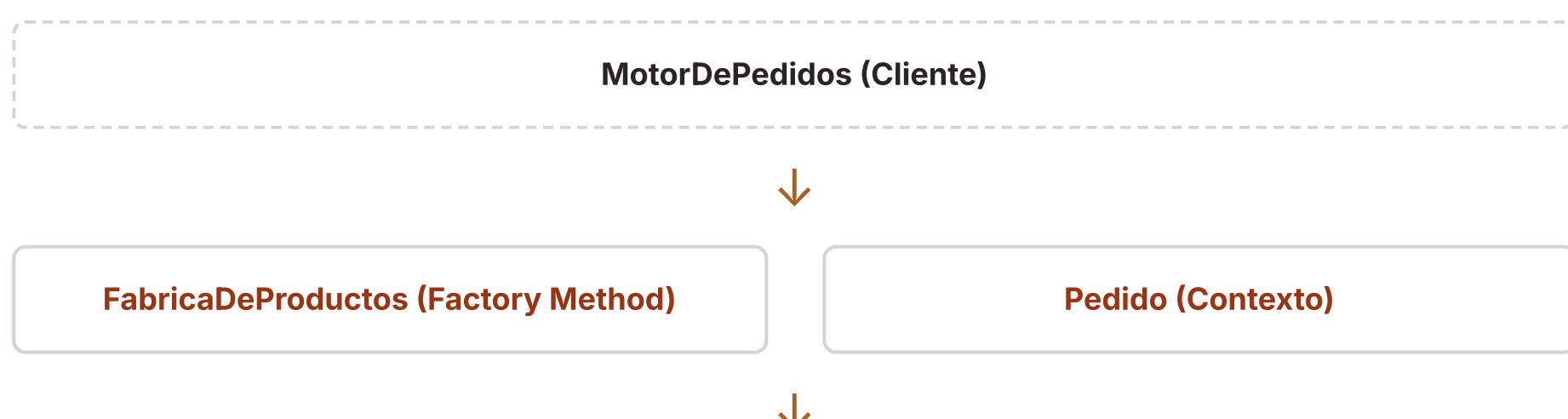
1. Explicar la Arquitectura: Muestren un diagrama de clases o expliquen verbalmente cómo se relacionan los tres patrones en su sistema.

2. Demostración de Código: Ejecuten su programa y muestren los escenarios requeridos, explicando cómo los patrones permiten la flexibilidad.

Pregunta de Reflexión (Opcional): Si tuvieran que añadir un nuevo método de notificación (ej. WhatsApp), ¿qué principio S.O.L.I.D. y qué patrón estructural les facilitaría esta tarea y por qué?

4. Arquitectura Visual de Referencia

Usa este diagrama conceptual como guía. Muestra cómo los diferentes componentes del sistema interactúan. Pasa el cursor sobre cada elemento para ver su descripción.



Decoradores de Pedido (Decorator)**Estrategias de Pago (Strategy)**