-Primera Entrega TPO-

*Facultad de Ingeniería y Ciencias Exactas (**FAIN**)*

Ingeniería en Informática

Materia: Proceso de Desarrollo de Software

Día-Turno: Viernes - Mañana

Profesor: Ana Carolina Martínez

Alumnos: Charelli Franco - Sanchez Santiago - Tegli Santino

Fecha de Entrega: 7 de mayo del 2025

Introducción

Hemos diseñado una solución modular para cubrir todas las necesidades del restaurante: consulta de menú, toma de pedidos online, procesamiento de pagos, seguimiento de estados y envío de notificaciones. Para ello identificamos un conjunto de clases que abstraen claramente cada responsabilidad y facilitan futuras ampliaciones o modificaciones.

Gestión del menú:

La clase principal es MenuDigital, encargada de almacenar los productos organizados por categoría, así como de proporcionar métodos para agregar, eliminar y consultar platos. Para representar las categorías (Entradas, Platos principales, Postres y Bebidas) utilizamos una enumeración Categoria, que evita errores de escritura y garantiza la integridad de los datos. Cada elemento del menú está modelado por la clase Producto, que contiene la descripción, el precio, la indicación de posibles alérgenos y su categoría. Esta estructuración permite agregar en el futuro atributos como imágenes o estados de disponibilidad sin alterar la lógica existente.

Modelado del pedido:

Cada pedido se representa mediante la clase Pedido, que agrupa una lista de ItemPedido (cada uno enlaza un Producto con una cantidad y calcula su subtotal), el cliente que lo realiza, el mesero asignado (si corresponde), el método de pago seleccionado y el estado actual en que se encuentra. Pedido es además responsable de calcular el total, aplicar cupones de descuento y coordinar la generación de la factura.

Gestión de estados:

Para reflejar el ciclo de vida de un pedido (“En espera”, “En preparación”, “Listo para entregar” y “Entregado”) aplicamos el patrón State. Definimos una interfaz Estado cuyo método procesar() realiza la acción concreta de cada fase y, al cambiar de estado, dispara las notificaciones oportunas. Las implementaciones concretas de Estado encapsulan las reglas de transición y delegan la notificación a los suscriptores interesados (cliente y mesero), evitando condicionales complejos en la clase Pedido y permitiendo añadir futuros estados, como “Cancelado”, sin modificar el núcleo del sistema.

Procesamiento de pagos:

El pago se abstrae mediante la interfaz IPago, que define los métodos realizarPago(Pedido) y obtenerMonto(Pedido). De esta forma, las clases TarjetaDebito y TarjetaCredito implementan sus propios procesos de validación y cobro. Para incorporar descuentos mediante cupones sin duplicar lógica, aplicamos el patrón Decorator: PagoDecorator envuelve una estrategia IPago, y PagoCuponDecorator añade la reducción correspondiente tras validar el código de cupón. Este enfoque permite componer distintos elementos de precio (tarifa base, promociones, fidelidad) de forma dinámica y extensible.

Control de cupones:

La clase Cupon gestiona el código, el porcentaje o monto de descuento y su validación frente a una lista de códigos válidos. Al centralizar esta lógica en una única clase se facilita la incorporación de nuevos tipos de oferta y reglas de caducidad.

Notificaciones y facturación:

Para mantener informados a clientes y meseros se emplea el patrón Observer. La interfaz IPersona define el método notificar(mensaje, Pedido), implementado por Cliente (y su dirección de correo electrónico) y Mesero. Cuando el estado del pedido cambia, el objeto Estado invoca a Pedido para que notifique a todos sus suscriptores. Por su parte, la clase NotificadorEmail encapsula la funcionalidad de envío de correos, separando la lógica de comunicación de la de negocio. Finalmente, la clase Factura construye el detalle de venta (productos, cantidades, totales y datos del cliente) y se integra con NotificadorEmail para remitirla automáticamente al finalizar la entrega.

La propuesta combina patrones de diseño (State para gestionar estados, Strategy y Decorator para pagos, Observer para notificaciones) y una clara separación de responsabilidades. Esto garantiza flexibilidad para evolucionar el sistema (nuevos métodos de pago, canales de notificación, estados adicionales) y facilita su mantenimiento, a la vez que cumple de manera robusta con los requerimientos funcionales del cliente.