Práctica Estructurales

Tags

Ejercicio 1: Adapter

Descripción del ejercicio:
Crea una aplicación que simule un sistema de carga de vehículos eléctricos. Tendrás una clase `Charger` que proporciona energía en un formato específico (por ejemplo, 220V). Implementa una clase `USCharger` que representa un cargador estadounidense (110V). Utiliza el patrón Adapter para permitir que el `USCharger` funcione con el sistema que requiere 220V.

1. Define una interfaz `ICharger` con un método `Charge()`.
2. Implementa la clase `Charger` que proporciona la energía correcta.
3. Implementa la clase `USCharger` que debe ser adaptada.
4. Crea un adaptador `USChargerAdapter` que implemente `ICharger` y use `USCharger`.

Ejercicio 2: Decorator - Sistema de Gestión de Facturas

Descripción del ejercicio:
Crea una aplicación que simule un sistema de gestión de facturas para una tienda. Las facturas pueden incluir diferentes servicios adicionales (como envío y envoltura de regalo) y cada servicio tiene un costo adicional.

1. Define una interfaz `IFactura` que tenga métodos `GetDescription()` y `GetTotalCost()`.

2. Implementa una clase concreta `BasicInvoice` que represente una factura básica.
3. Crea decoradores para los servicios adicionales:
    - `ShippingDecorator` para agregar costos de envío.
    - `GiftWrappingDecorator` para agregar costos de envoltura de regalo.
    - `DiscountDecorator` para aplicar descuentos.
4. Permite al usuario crear una factura, agregar servicios y mostrar la descripción y el costo total.

Ejercicio 3: Bridge - Sistema de Procesamiento de Pagos

Descripción del ejercicio:
Crea una aplicación que simule un sistema de procesamiento de pagos que soporte múltiples métodos de pago (como Tarjeta de Crédito y PayPal) y diferentes plataformas de comercio (como Tienda en Línea y Aplicación Móvil). Utiliza el patrón Bridge para desacoplar la lógica de procesamiento de pagos de los métodos de pago.

1. Define una interfaz `IPaymentMethod` con métodos para `Pay(double amount)`.
2. Implementa clases concretas para los métodos de pago:
    - `CreditCardPayment` que implemente `IPaymentMethod`.
    - `PaypalPayment` que implemente `IPaymentMethod`.
3. Define una clase abstracta `PaymentProcessor` que contenga un campo de tipo `IPaymentMethod` y un método abstracto `ProcessPayment(double amount)`.
4. Implementa clases concretas que extiendan `PaymentProcessor` para diferentes plataformas de comercio:
    - `OnlineStorePaymentProcessor`
    - `MobileAppPaymentProcessor`
5. Permite al usuario seleccionar un método de pago y una plataforma para procesar un pago y muestra el resultado.