UNIDAD TEMÁTICA 5 – Patrones de diseño– Trabajo de Aplicación 4

Patrón decorator:

S: Cumple con el patron SRP ya que cada decorator tiene una única responsabilidad, la cual es añadir una funcionalidad específica del objeto decorado.

O: En lugar de modificar la clase principal para añadir nuevas funcionalidades, se crean nuevos decoradores que extienden el comportamiento de la clase principal de forma dinámica. Esto permite extender el comportamiento sin modificar el código existente, cumpliendo así el patrón OCP

L: Este patrón permite que los decoradores sean intercambiables con los objetos que decoran, ya que ambos implementan la misma interfaz o heredan de la misma clase base.

I: Este principio no se aplica de manera directa al patrón decorator, ya que se centra en la creación de interfaces específicas y pequeñas, lo cual es más relevante para la definición de interfaces que para la estructura de clases y la composición dinámica de comportamientos que maneja este patrón.

D: Si los decoradores están diseñados para trabajar con implementaciones específicas en lugar de interfaces, se puede violar el principio de inversión de dependencia. Sin embargo, esta violación se puede evitar asegurando que tanto el componente base como los decoradores dependan de abstracciones.

Ejercicio 1)

public interface IDataExporter  
{  
 string Export(object data);  
}  
  
public class JsonDataExporter : IDataExporter  
{  
 public string Export(object data)  
 {  
 return JsonConvert.SerializeObject(data);  
 }  
}  
  
public class XmlDataExporter : IDataExporter  
{  
 public string Export(object data)  
 {  
 XmlSerializer xmlSerializer = new XmlSerializer(data.GetType());  
 using (StringWriter textWriter = new StringWriter())  
 {  
 xmlSerializer.Serialize(textWriter, data);  
 return textWriter.ToString();  
 }  
 }  
}  
  
public class TxtDataExporter : IDataExporter  
{  
 public string Export(object data)  
 {  
 return data.ToString();  
 }  
}  
  
public class DataService  
{  
 private readonly IDataExporter \_dataExporter;  
  
 public DataService(IDataExporter dataExporter)  
 {  
 \_dataExporter = dataExporter;  
 }  
  
 public string ExportData(object data)  
 {  
 return \_dataExporter.Export(data);  
 }  
}  
  
class Program  
{  
 static void Main()  
 {  
 *// Datos a exportar* var data = new { Name = "Juancito", Age = 30 };  
  
 *// Crear y utilizar servicio de datos con exportadores específicos* Console.WriteLine("Datos en formato JSON:");  
 DataService jsonDataService = new DataService(new JsonDataExporter());  
 Console.WriteLine(jsonDataService.ExportData(data));  
  
 Console.WriteLine("\nDatos en formato XML:");  
 DataService xmlDataService = new DataService(new XmlDataExporter());  
 Console.WriteLine(xmlDataService.ExportData(data));  
  
 Console.WriteLine("\nDatos en formato TXT:");  
 DataService txtDataService = new DataService(new TxtDataExporter());  
 Console.WriteLine(txtDataService.ExportData(data));  
 }  
}

Ejercicio 2)

public interface IQuickPay  
{  
 bool MakePayment(double amount, string currency);  
}  
public class QuickPayService : IQuickPay  
{  
 public bool MakePayment(double amount, string currency)  
 {  
 Console.WriteLine($"Pagado {amount} {currency} usando QuickPay.");  
 return true; *// Simular éxito* }  
}  
public class OnlineStore  
{  
 private IAdapter \_adapterService;  
 public OnlineStore(IAdapter adapterService)  
 {  
 \_adapterService = adapterService;  
 }  
 public void Checkout(double amount, string currency, bool useSafePay)  
 {  
 bool paymentSuccess;  
 if (useSafePay)  
 {  
 paymentSuccess = \_adapterService.MakePaymentWithSafePay(amount, currency);  
 }  
 else  
 {  
 paymentSuccess = \_adapterService.MakePaymentWithQuickPay(amount, currency);  
 }  
  
 if (paymentSuccess)  
 {  
 Console.WriteLine("Pago exitoso!");  
 }  
 else  
 {  
 Console.WriteLine("El pago ha fallado.");  
 }  
 }  
}  
  
public class SafePayService  
{  
 public void Transact(string fromAccount, string toAccount, string  
 currencyType, double amount)  
 {  
 Console.WriteLine($"Transfiriendo {amount} {currencyType} de  
 {fromAccount} a {toAccount} usando SafePay.");  
 }  
}  
  
public class IAdapter : QuickPayService, SafePayService  
{  
 private SafePayService \_safePayService;  
 private QuickPayService \_quickPayService;  
 private string \_fromAccount;  
 private string \_toAccount;  
  
 public Adapter(SafePayService safePayService, QuickPayService quickPayService string fromAccount, string toAccount)  
 {  
 \_safePayService = safePayService;  
 \_quickPayService = quickPayService;  
 \_fromAccount = fromAccount;  
 \_toAccount = toAccount;  
 }  
  
 public bool MakePaymentWithSafePay(double amount, string currency)  
 {  
 \_safePayService.Transact(\_fromAccount, \_toAccount, currency, amount);  
 return true; *// Simular éxito* }  
 public bool MakePaymentWithQuickPay(double amount, string currency)  
 {  
 return \_quickPayService.MakePayment(amount, currency);  
 }  
}

Ejercicio 3)

public abstract class NotificationDecorator : Notification  
{  
 protected Notification \_notification;  
  
 public NotificationDecorator(Notification notification)  
 {  
 \_notification = notification;  
 }  
  
 public override void Send(string message)  
 {  
 \_notification.Send(message);  
 }  
}  
public class SmsNotificationDecorator : NotificationDecorator  
{  
 public SmsNotificationDecorator(Notification notification) : base(notification)  
 {  
 }  
  
 public override void Send(string message)  
 {  
 base.Send(message);  
 Console.WriteLine($"Enviando SMS: {message}");  
 }  
}  
  
public class PushNotificationDecorator : NotificationDecorator  
{  
 public PushNotificationDecorator(Notification notification) : base(notification)  
 {  
 }  
  
 public override void Send(string message)  
 {  
 base.Send(message);  
 Console.WriteLine($"Enviando notificación push: {message}");  
 }  
}  
  
class Program  
{  
 static void Main()  
 {  
 Notification emailNotification = new EmailNotification();  
 emailNotification.Send("Mensaje básico por correo electrónico.");  
  
 Notification smsAndEmailNotification = new SmsNotificationDecorator(emailNotification);  
 smsAndEmailNotification.Send("Mensaje con correo electrónico y SMS.");  
  
 Notification pushSmsAndEmailNotification = new PushNotificationDecorator(smsAndEmailNotification);  
 pushSmsAndEmailNotification.Send("Mensaje con correo electrónico, SMS y notificación push.");  
 }  
}

Ejercicio 4)

public class HotelManagementFacade  
{  
 private readonly ReservationSystem \_reservationSystem;  
 private readonly RestaurantManagementSystem \_restaurantSystem;  
 private readonly CleaningServiceSystem \_cleaningSystem;  
  
 public HotelManagementFacade()  
 {  
 \_reservationSystem = new ReservationSystem();  
 \_restaurantSystem = new RestaurantManagementSystem();  
 \_cleaningSystem = new CleaningServiceSystem();  
 }  
  
 public void ReserveRoom(string roomType)  
 {  
 \_reservationSystem.ReserveRoom(roomType);  
 }  
  
 public void BookRestaurantTable(string tableType)  
 {  
 \_restaurantSystem.BookTable(tableType);  
 }  
  
 public void ScheduleRoomCleaning(string roomNumber)  
 {  
 \_cleaningSystem.ScheduleRoomCleaning(roomNumber);  
 }  
}  
class Program  
{  
 static void Main()  
 {  
 HotelManagementFacade hotelFacade = new HotelManagementFacade();  
  
 hotelFacade.ReserveRoom("Deluxe");  
 hotelFacade.BookRestaurantTable("VIP");  
 hotelFacade.ScheduleRoomCleaning("101");  
  
 *// ... Realizar otras operaciones ...* }  
}

Ejercicio 5)

public interface IDocument  
{  
 void Display();  
}  
  
public class Document : IDocument  
{  
 private string \_content;  
  
 public Document(string content)  
 {  
 \_content = content;  
 }  
  
 public void Display()  
 {  
 Console.WriteLine($"Contenido del documento: {\_content}");  
 }  
}  
  
public class DocumentProxy : IDocument  
{  
 private Document \_document;  
 private string \_userRole;  
  
 public DocumentProxy(string content, string userRole)  
 {  
 \_document = new Document(content);  
 \_userRole = userRole;  
 }  
  
 public void Display()  
 {  
 if (HasAccess())  
 {  
 \_document.Display();  
 }  
 else  
 {  
 Console.WriteLine("Acceso denegado. No tiene los permisos necesarios para ver este documento.");  
 }  
 }  
  
 private bool HasAccess()  
 {  
 *// Aquí se define la lógica para verificar los permisos del usuario  
 // Por simplicidad, asumimos que solo los usuarios con rol "admin" tienen acceso* return \_userRole == "admin";  
 }  
}

Ejercicio 6)

public class OrderFacade  
{  
 private readonly CartSystem \_cartSystem;  
 private readonly InventorySystem \_inventorySystem;  
 private readonly BillingSystem \_billingSystem;  
  
 public OrderFacade()  
 {  
 \_cartSystem = new CartSystem();  
 \_inventorySystem = new InventorySystem();  
 \_billingSystem = new BillingSystem();  
 }  
  
 public void PlaceOrder(string product, int quantity)  
 {  
 *// Agregar producto al carrito* \_cartSystem.AddToCart(product, quantity);  
 *// Reducir el stock del producto* \_inventorySystem.ReduceStock(product, quantity);  
 *// Generar la factura del producto* \_billingSystem.GenerateInvoice(product, quantity);  
 }  
}  
  
class Program  
{  
 static void Main()  
 {  
 *// Crear instancia de la fachada* OrderFacade orderFacade = new OrderFacade();  
 *// Definir los parámetros del pedido* string product = "Libro";  
 int quantity = 2;  
 *// Realizar el pedido a través de la fachada* orderFacade.PlaceOrder(product, quantity);  
 }  
}

Ejercicio 7)

public class TwitterFacade  
{  
 private readonly TwitterAuthenticator \_authenticator;  
 private readonly TwitterApi \_twitterApi;  
 private readonly TwitterDataParser \_dataParser;  
  
 public TwitterFacade()  
 {  
 \_authenticator = new TwitterAuthenticator();  
 \_twitterApi = new TwitterApi();  
 \_dataParser = new TwitterDataParser();  
 }  
  
 public int GetPostCount(string apiKey, string apiSecret, string username)  
 {  
 *// Autenticarse en la API de Twitter* string accessToken = \_authenticator.Authenticate(apiKey, apiSecret);  
 *// Hacer una solicitud a la API de Twitter para obtener información del usuario* string jsonResponse = \_twitterApi.MakeApiRequest($"https://api.twitter.com/users/{username}", accessToken);  
 *// Parsear la respuesta JSON para extraer la cantidad de posts* int postCount = \_dataParser.ParsePostCount(jsonResponse);  
 return postCount;  
 }  
}  
  
class Program  
{  
 static void Main()  
 {  
 *// Crear instancia del facade* TwitterFacade twitterFacade = new TwitterFacade();  
 *// Definir las credenciales y el nombre de usuario* string apiKey = "api\_key";  
 string apiSecret = "api\_secret";  
 string username = "john\_doe";  
 *// Obtener la cantidad de posts a través del facade* int postCount = twitterFacade.GetPostCount(apiKey, apiSecret, username);  
 *// Mostrar la cantidad de posts* Console.WriteLine($"Cantidad de posts del usuario {username}: {postCount}");  
 }  
}

Ejercicio 8)

public abstract class TextDecorator : ElementoTexto  
{  
 protected ElementoTexto \_elementoTexto;  
  
 public TextDecorator(ElementoTexto elementoTexto) : base(elementoTexto.ObtenerTexto())  
 {  
 \_elementoTexto = elementoTexto;  
 }  
  
 public override abstract string ObtenerTexto();  
}  
public class ElementoTexto  
{  
 private string \_texto;  
  
 public ElementoTexto(string texto)  
 {  
 \_texto = texto;  
 }  
  
 public virtual string ObtenerTexto()  
 {  
 return \_texto;  
 }  
}  
  
public class EstiloFuenteDecorator : TextDecorator  
{  
 private string \_estiloFuente;  
  
 public EstiloFuenteDecorator(ElementoTexto elementoTexto, string estiloFuente) : base(elementoTexto)  
 {  
 \_estiloFuente = estiloFuente;  
 }  
  
 public override string ObtenerTexto()  
 {  
 return $"<span style=\"font-family: {\_estiloFuente}\">{\_elementoTexto.ObtenerTexto()}</span>";  
 }  
}  
  
public class ColorDecorator : TextDecorator  
{  
 private string \_color;  
  
 public ColorDecorator(ElementoTexto elementoTexto, string color) : base(elementoTexto)  
 {  
 \_color = color;  
 }  
  
 public override string ObtenerTexto()  
 {  
 return $"<span style=\"color: {\_color}\">{\_elementoTexto.ObtenerTexto()}</span>";  
 }  
}  
  
public class DecoracionDecorator : TextDecorator  
{  
 private string \_decoracion;  
  
 public DecoracionDecorator(ElementoTexto elementoTexto, string decoracion) : base(elementoTexto)  
 {  
 \_decoracion = decoracion;  
 }  
  
 public override string ObtenerTexto()  
 {  
 return $"<span style=\"text-decoration: {\_decoracion}\">{\_elementoTexto.ObtenerTexto()}</span>";  
 }  
}  
class Program  
{  
 static void Main()  
 {  
 *// Crear una instancia de un elemento de texto* ElementoTexto elementoTexto = new ElementoTexto("Hola, mundo!");  
 *// Aplicar decoradores para personalizar la apariencia del elemento de texto* elementoTexto = new EstiloFuenteDecorator(elementoTexto, "Arial");  
 elementoTexto = new ColorDecorator(elementoTexto, "red");  
 elementoTexto = new DecoracionDecorator(elementoTexto, "underline");  
 *// Obtener el texto con la apariencia personalizada* string textoPersonalizado = elementoTexto.ObtenerTexto();  
 Console.WriteLine(textoPersonalizado);  
 }  
}