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

Realicen el siguientes ejercicio en equipo.

EJERCICIO 0

En el patrón Decorator, ¿qué principios SOLID se aplican? Justifique su respuesta.

¿Qué principios se violan, si los hubiera? Justifiquen su respuesta.

Para cada uno de los siguientes ejercicios, en equipo:

1- Determine que patrón puede resolver el problema de una forma más eficiente.
2- Agregue las clases, interfaces, métodos que considere necesarios para remediar la situación.

EJERCICIO 1

```
public class DataService
{
    public string ExportAsJson(object data)
    {
        return JsonConvert.SerializeObject(data);
    }

    public string ExportAsXml(object data)
    {
        XmlSerializer xmlSerializer = new XmlSerializer(data.GetType());
        using (StringWriter textWriter = new StringWriter())
        {
            xmlSerializer.Serialize(textWriter, data);
            return textWriter.ToString();
        }
    }

    public string ExportAsTxt(object data)
    {
        return data.ToString();
    }
}


class Program
{
    static void Main()
    {
        // Datos a exportar
        var data = new { Name = "Juancito", Age = 30 };

        // Crear servicio de datos
        DataService dataService = new DataService();

        // Exportar y mostrar datos en formato JSON
        Console.WriteLine("Datos en formato JSON:");
        Console.WriteLine(dataService.ExportAsJson(data));
```

```
        // Exportar y mostrar datos en formato XML
        Console.WriteLine("\nDatos en formato XML:");
        Console.WriteLine(dataService.ExportAsXml(data));

        // Exportar y mostrar datos en formato TXT
        Console.WriteLine("\nDatos en formato TXT:");
        Console.WriteLine(dataService.ExportAsTxt(data));
    }}
```

EJERCICIO 2

Imagina que eres un desarrollador en una tienda en línea. Tu tienda actualmente solo soporta pagos a través de un proveedor de servicios de pago llamado "QuickPay". Ahora, la tienda quiere soportar un nuevo proveedor de servicios de pago llamado "SafePay". El problema es que "SafePay" tiene una interfaz de programación completamente diferente a la de "QuickPay".

```
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 IQuickPay _paymentService;

    public OnlineStore(IQuickPay paymentService)
    {
        _paymentService = paymentService;
    }

    public void Checkout(double amount, string currency)
    {
        if (_paymentService.MakePayment(amount, currency))
        {
            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.");
```

```
        }
}
```

El objetivo es hacer que la tienda en línea sea compatible con "SafePay" sin cambiar el código existente de las clases que procesan pagos con "QuickPay".

EJERCICIO 3

Una aplicación que permite a los usuarios recibir notificaciones. Inicialmente, la aplicación solo soportaba notificaciones por correo electrónico. Ahora, se planea introducir nuevas formas de envío de notificaciones como SMS, Mensajes Directos en Twitter y Mensajes en Facebook.

Actualmente tienes la siguiente estructura en tu aplicación:

```
public abstract class Notification
{
    public abstract void Send(string message);
}

public class EmailNotification : Notification
{
    public override void Send(string message)
    {
        Console.WriteLine($"Enviando correo electrónico: {message}");
    }
}
```

Refactoriza el código existente e introduce nuevas clases o interfaces si es necesario para soportar las nuevas formas de notificación (SMS, Mensajes Directos en Twitter, Mensajes en Facebook).

EJERCICIO 4

Un sistema de gestión de hotel que tiene múltiples subsistemas como el sistema de reservas, sistema de gestión de restaurantes, sistema de gestión de limpieza, etc. Cada subsistema tiene su propia interfaz compleja y son independientes entre sí.

Actualmente tienes la siguiente estructura en tu aplicación:

```
public class ReservationSystem
{
    public void ReserveRoom(string roomType)
    {
        Console.WriteLine($"Reservando una habitación de tipo:
{roomType}");
    }
}

public class RestaurantManagementSystem
{
    public void BookTable(string tableType)
    {
        Console.WriteLine($"Reservando una mesa de tipo: {tableType}");
    }
}

public class CleaningServiceSystem
{
    public void ScheduleRoomCleaning(string roomNumber)
    {
        Console.WriteLine($"Programando la limpieza para la habitación
número: {roomNumber}");
    }
}
```

```
class Program
{
    static void Main()
    {
        ReservationSystem reservationSystem = new ReservationSystem();
        reservationSystem.ReserveRoom("Deluxe");

        RestaurantManagementSystem restaurantSystem = new
RestaurantManagementSystem();
        restaurantSystem.BookTable("VIP");

        CleaningServiceSystem cleaningSystem = new CleaningServiceSystem();
        cleaningSystem.ScheduleRoomCleaning("101");

      //… Do stuff… reservationSystem + restaurantSystem + cleaningSystem
    }
}
```

EJERCICIO 5

Un sistema de gestión de documentos y se te ha pedido que implementes un mecanismo de control de acceso a los documentos almacenados. Los usuarios solo deben poder acceder a un documento si tienen el permiso adecuado.

Tienes la siguiente clase que representa un documento:

```
public class Document
{
    private string _content;

    public Document(string content)
    {
        _content = content;
    }

    public void Display()
    {
        Console.WriteLine($"Contenido del documento: {_content}");
    }
}

class Program
{
    static void Main()
    {
        Document document = new Document("Este es un documento
importante.");
        document.Display();
    }
}
```

EJERCICIO 6

```
public class CartSystem
{
    public void AddToCart(string product, int quantity)
    {
        // Simular llamada a la API del sistema de carrito de compras
        Console.WriteLine($"API llamada: Agregando {quantity} de {product}
al carrito.");
    }
}

public class InventorySystem
{
    public void ReduceStock(string product, int quantity)
    {
        // Simular llamada a la API del sistema de inventario
        Console.WriteLine($"API llamada: Reduciendo el stock de {product}
en {quantity}.");
    }
}

public class BillingSystem
{
    public void GenerateInvoice(string product, int quantity)
    {
        // Simular llamada a la API del sistema de facturación
        Console.WriteLine($"API llamada: Generando factura para {quantity}
de {product}.");
    }
}

class Program
{
    static void Main()
    {
        // Crear instancias de cada subsistema
        CartSystem cartSystem = new CartSystem();
        InventorySystem inventorySystem = new InventorySystem();
        BillingSystem billingSystem = new BillingSystem();

        // Definir los parámetros del pedido
        string product = "Libro";
        int quantity = 2;

        // Llamar a la API del sistema de carrito de compras para agregar
el producto al carrito
        cartSystem.AddToCart(product, quantity);

        // Llamar a la API del sistema de inventario para reducir el stock
        inventorySystem.ReduceStock(product, quantity);

        // Llamar a la API del sistema de facturación para generar la
factura
        billingSystem.GenerateInvoice(product, quantity);
    }

}
```

EJERCICIO 7

```
public class TwitterAuthenticator
{
    public string Authenticate(string apiKey, string apiSecret)
    {
        // Lógica para autenticarse en la API de Twitter y obtener un token
de acceso.
        // Para simplificar el ejemplo, vamos a suponer que siempre
recibimos un token "ABC123".
        Console.WriteLine("Autenticando en la API de Twitter...");
        return "ABC123";
    }
}

public class TwitterApi
{
    public string MakeApiRequest(string endpoint, string accessToken)
    {
        // Lógica para hacer una solicitud a la API de Twitter.
        // Para simplificar el ejemplo, vamos a suponer que siempre
recibimos un JSON de respuesta.
        Console.WriteLine($"Haciendo una solicitud a {endpoint}...");
        return "{\"user\": \"john_doe\", \"post_count\": 42}";
    }
}

public class TwitterDataParser
{
    public int ParsePostCount(string jsonResponse)
    {
        // Lógica para parsear el JSON y extraer la cantidad de posts.
        // Para simplificar, vamos a suponer que siempre recibimos 42.
        Console.WriteLine("Parseando la respuesta de la API...");
        return 42;
    }
}

class Program
{
    static void Main()
    {
        // Crear instancias de las clases necesarias
        TwitterAuthenticator authenticator = new TwitterAuthenticator();
        TwitterApi twitterApi = new TwitterApi();
        TwitterDataParser dataParser = new TwitterDataParser();

        // Autenticarse en la API de Twitter
        string accessToken = authenticator.Authenticate("api_key",
"api_secret");

        // Hacer una solicitud a la API de Twitter para obtener información
del usuario
        string jsonResponse =
twitterApi.MakeApiRequest($"https://api.twitter.com/users/john_doe",
accessToken);

        // Parsear la respuesta JSON para extraer la cantidad de posts
        int postCount = dataParser.ParsePostCount(jsonResponse);

        // Mostrar la cantidad de posts
```

```
        Console.WriteLine($"Cantidad de posts del usuario john_doe:
{postCount}");
    }
}
```

EJERCICIO 8

```
public class ElementoTexto
{
    private string _texto;
    private string _estiloFuente;
    private string _color;
    private string _decoracion;

    public ElementoTexto(string texto)
    {
        _texto = texto;
    }

    public void SetEstiloFuente(string estiloFuente)
    {
        _estiloFuente = estiloFuente;
    }

    public void SetColor(string color)
    {
        _color = color;
    }

    public void SetDecoracion(string decoracion)
    {
        _decoracion = decoracion;
    }

    public string ObtenerTexto()
    {
        string textoDecorado = _texto;

        if (!string.IsNullOrEmpty(_estiloFuente))
        {
            textoDecorado = $"<span style=\"font-family:
{_estiloFuente}\">{textoDecorado}</span>";
        }

        if (!string.IsNullOrEmpty(_color))
        {
            textoDecorado = $"<span style=\"color:
{_color}\">{textoDecorado}</span>";
        }

        if (!string.IsNullOrEmpty(_decoracion))
        {
            textoDecorado = $"<span style=\"text-decoration:
{_decoracion}\">{textoDecorado}</span>";
        }

        return textoDecorado;
    }
}

class Program
```

```
{
    static void Main()
    {
        // Crear una instancia de un elemento de texto
        ElementoTexto elementoTexto = new ElementoTexto("Hola, mundo!");

        // Personalizar la apariencia del elemento de texto
        elementoTexto.SetEstiloFuente("Arial");
        elementoTexto.SetColor("red");
        elementoTexto.SetDecoracion("underline");

        // Obtener el texto con la apariencia personalizada
        string textoPersonalizado = elementoTexto.ObtenerTexto();

        Console.WriteLine(textoPersonalizado);  // <span style="font-
family: Arial; color: red; text-decoration: underline">Hola, mundo!</span>
    }
}
```