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

Para cada uno de los siguientes ejercicios, en equipo:

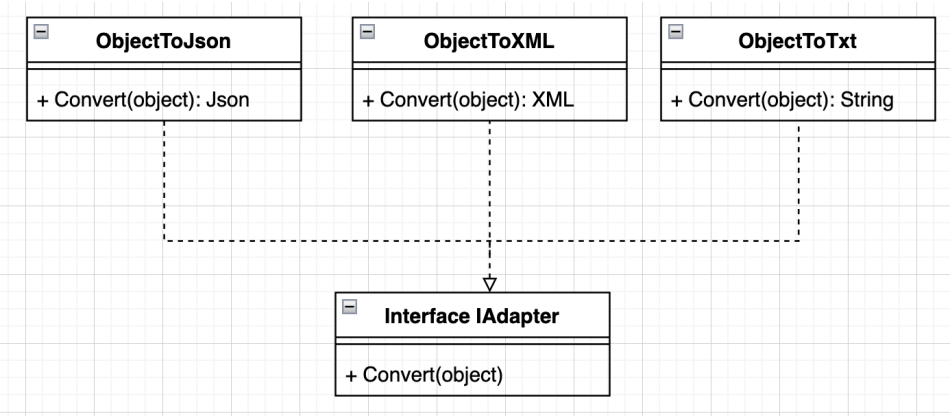
1- Determine que patrón puede resolver el problema de una forma más eficiente.

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));
    }}
```

El patrón para resolver sería Adapter ya que se requiere una interfaz para cada convertidor.



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 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.");
    }
}
```

Dado que 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" el patrón que soluciona es Adapter.

```
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.");
   }
}

public class SafePayAdapter : IQuickPay
{
   private SafePayService _safePayService;

   public SafePayAdapter(SafePayService safePayService)
   {
      _safePayService = safePayService;
   }

   public bool MakePayment(double amount, string currency)
   {
      _safePayService.Transact("Mi cuenta", "Cuenta de la tienda", currency, amount);
      return true;
   }
}
```

EJERCICIO 3

```
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}");
    }
}
```

Es Decorator

```
using System;

public interface INotification
```

```
{
    void Send(string message);
}

public class NotificationDecorator : INotification
{
    protected INotification _notification;

    public NotificationDecorator(INotification notification)
    {
        _notification = notification;
    }

    public virtual void Send(string message)
    {
        Console.WriteLine($"Enviando: {message}");
    }
}

public class EmailNotification : NotificationDecorator
{
    public EmailNotification(INotification notification) : base(notification) { }

    public override void Send(string message)
    {
        Console.WriteLine($"Enviando EmailNotification: {message}");
        if (_notification != null)
        {
            _notification.Send(message);
        }
    }
}

public class SMSNotification : NotificationDecorator
{
    public SMSNotification(INotification notification) : base(notification) { }

    public override void Send(string message)
    {
        Console.WriteLine($"Enviando SMS: {message}");
        if (_notification != null)
        {
            _notification.Send(message);
```

```
        }
    }
}

public class TwitterNotification : NotificationDecorator
{
    public TwitterNotification(INotification notification) : base(notification) { }

    public override void Send(string message)
    {
        Console.WriteLine($"Enviando Mensaje Directo en Twitter: {message}");
        if (_notification != null)
        {
            _notification.Send(message);
        }
    }
}

public class FacebookNotification : NotificationDecorator
{
    public FacebookNotification(INotification notification) : base(notification) { }

    public override void Send(string message)
    {
        Console.WriteLine($"Enviando Mensaje en Facebook: {message}");
        if (_notification != null)
        {
            _notification.Send(message);
        }
    }
}

class Program
{
    static void Main(string[] args)
    {
        INotification notification = new EmailNotification(null);
        notification = new SMSNotification(notification);
        notification = new TwitterNotification(notification);
        notification = new FacebookNotification(notification);

        notification.Send("¡Hola mundo!");
    }
```

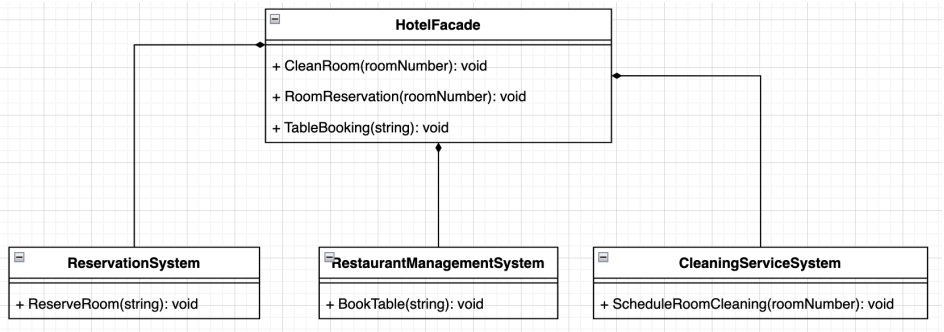
```
}
```

EJERCICIO 4

```
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
    }
}
```

Esto es tremendo facade, porque se tienen muchos subsistemas y conviene hacer una clase facade que centralice esos subsistemas para tener mayor control.



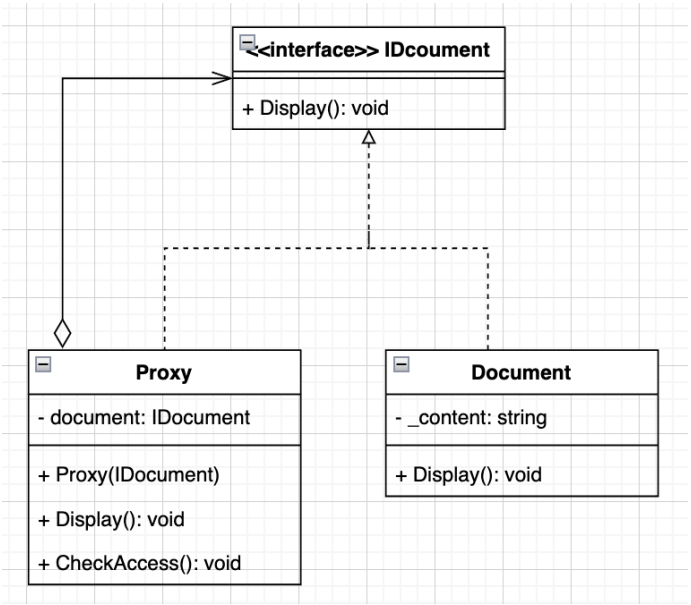

```
public class HotelFacade
{
   private ReservationSystem _roomBooking;
   private CleaningServiceSystem _roomCleaning;
   private RoomRestaurantManagementSystem _roomService;

   public HotelFacade()
   {
      _roomBooking = new ReservationSystem();
      _roomCleaning = new CleaningServiceSystem();
      _roomService = new RoomRestaurantManagementSystem();
   }

   public void RoomReservation(string roomType)
   {
      _roomBooking.ReserveRoom();
   }

   public void CleanRoom(string roomNumber)
   {
      _roomCleaning.ScheduleRoomCleaning(roomNumber);
   }

   public void TableBooking(string tableType)
   {
      _roomService.BookTable(tableType);
   }
}

public class ReservationSystem
{
   public void ReserveRoom()
   {
```

```
        Console.WriteLine("Reserving room...");
        // Logic to reserve the room
    }
}

public class RoomRestaurantManagementSystem
{
    public void BookTable(string tableType)
    {
        Console.WriteLine("Booking table...");
        // Logic to book the table
    }
}

public class CleaningServiceSystem
{
    public void ScheduleRoomCleaning(string roomNumber)
    {
        Console.WriteLine($"Scheduling cleaning for room {roomNumber}...");
        // Logic to schedule room cleaning
    }
}

public class Program
{
    public static void Main()
    {
        var hotelFacade = new HotelFacade();
        string roomType = "Deluxe";
        string roomNumber = "101";
        string tableType = "Window";
        hotelFacade.RoomReservation(roomType);
        hotelFacade.CleanRoom(roomNumber);
        hotelFacade.TableBooking(tableType);
    }
}
```

EJERCICIO 5

```
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();
    }
}
```

El patrón que resuelve el problema es Proxy debido a que este provee un control de acceso al objeto original, en este caso se desea un mecanismo de control de acceso a los documentos almacenados para evitar que todos puedan acceder a él.




```
public interface IDocument {
  void Display();
```

```
}

public class Proxy implements IDocument {
   private IDocument document;

   public Proxy(IDocument document) {
      this.document = document;
   }

   public boolean CheckAccess() {
      // Some real checks should be here.
      return true;
   }

   @Override
   public void Display() {
      if (CheckAccess()) {
         document.Display();
      }
   }
}

public class Document implements IDocument {
   private String content;

   public Document(String content) {
      this.content = content;
   }

   @Override
   public void Display() {
      System.out.println("Displaying " + content);
   }
}
```

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);
    }

}
```

Esto es tremendo facade, porque se tienen muchos subsistemas y conviene hacer una clase facade que centralice esos subsistemas para tener mayor control.

EJERCICIO 7

El patrón que resuelve el problema es Proxy debido a que este provee un control de acceso al objeto original, en este caso se desea un mecanismo de control de acceso a los documentos almacenados para evitar que todos puedan acceder a él.

EJERCICIO 8

Bridge o builder