1.
    a. Adapter: En vez de tener una clase DataService, tenemos tres clases: JsonAdapter, XmlAdapter, y TxtAdapter, los cuales tienen la responsabilidad de exportar objetos a sus formatos correspondientes.
    b.

```
public class JsonAdapter {
      public string Export(object data)
      {
            // Code...
      }
}

public class XmlAdapter {
      public string Export(object data)
      {
            // Code...
      }
}

public class TxtAdapter {
      public string Export(object data)
      {
            // Code...
      }
}
```

2.

a. Adapter
b. Agrego la clase SafePayAdapter

```
public interface IQuickPay
{
      bool MakePayment(double amount, string currency);
}

public class SafePayAdapter : IQuickPay
{
      private SafePayService _safePayService;

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

      public bool MakePayment(double amount, string currency)
      {
            // Cuenta de origen predeterminada
            string fromAccount = "default";
            // Cuenta de destino predeterminada
            string toAccount = "default";

            _safePayService.Transact(fromAccount, toAccount,
currency, amount);

         // Supongamos que SafePay siempre retorna true para
simular éxito
         return true;
     }
}

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 QuickPayService : IQuickPay
{
      public bool MakePayment(double amount, string currency)
      {
            Console.WriteLine($"Pagado {amount} {currency} usando
QuickPay.");
            return true; // Simular éxito
      }
}

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.");
      }
}
```

3. Se puede utilizar el patrón Decorator, con clases decoradoras para cada tipo de notificación, que se extienden de la clase notificacion base. Solo se ejemplifica la instanciación del Decorador para SMS e Email

```
public interface INotification{
    void Send(string message);
}

public abstract class NotificationDecorator : INotification{
    private readonly INotification notification;

    public NotificationDecorator(INotification notification)
    {
        this.notification = notification;
    }

    public virtual void Send(string message)
    {
        notification.Send(message);
    }
}


public class SMSNotificationDecorator : NotificationDecorator{

    public SMSNotificationDecorator(INotification
notification){}

    public override void Send(string message){
        base.Send(message);
        SendSMS(message);
    }

    private void SendSMS(string message){
        Console.WriteLine($"Enviando SMS: {message}");
    }
}


public class EmailNotificationDecorator : NotificationDecorator{

    public EmailNotificationDecorator(INotification
notification){}

    public override void Send(string message){
        base.Send(message);
        SendEmail(message);
    }

    private void SendEmail(string message){
        Console.WriteLine($"Enviando Email: {message}");
    }
}
```

4. Dado que tenemos varios subsistemas independientes a los cuales queremos acceder, podemos utilizar el patrón Facade para exponer un solo objeto que agregue los distintos subsistemas y tenga los métodos necesarios en nuestro caso de uso

```
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}");
    }
}

public class HotelFacade
{
     private ReservationService _reservationSystem;
     private RestaurantManagementSystem 
_restaurantManagementSystem;
     private CleaningServiceSystem _cleaningServiceSystem;

     public HotelFacade()
     {
          this._reservationSystem = new ReservationService ();
          this._restaurantManagementSystem = new 
RestaurantManagementSystem ();
          this._cleaningServiceSystem = new CleaningServiceSystem 
();
     }
```

```
      public void ReserveRoom(string roomType)
      {
            this._reservationSystem.ReserveRoom(roomType);
      }

      public void BookTable(string tableType)
      {
            this._restaurantManagementSystem.BookTable(tableType);
      }

      public void ScheduleRoomCleaning(string roomNumber)
      {
            this._cleaningServiceSystem(roomNumber);
      }

}

class Program
{
    static void Main()
    {
        HotelFacade hotel = new HotelFacade();
        hotel.ReserveRoom("Deluxe");
        hotel.BookTable("VIP");
        hotel.ScheduleRoomCleaning("101");
        //… Do stuff… reservationSystem + restaurantSystem +
cleaningSystem
    }
}
```

5.

a. Proxy, más concretamente, proxy de control de acceso
b.

```
public interface IDisplayable
{
      void Display();
}

public class Document : IDisplayable
{
      private string _content;

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

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

public class DocProxy : IDisplayable
{
      private Document _doc;

      // Would be inserted as parameter of Display, but it’d
violate the IDisplayable interface
      private User _user = null;

      // Determines whether a user has permission
      private Func<User, bool> _pred;

      public DocProxy(Document doc, Func<User, bool> pred)
      {
            _doc = doc;
            _pred = pred;
      }



      private bool _allowed()
      {
            if (_user == null) return false;
            if (!_pred(_user)) return false;
```

```
            _user = null;
            return true;
      }

      public void SetUser(User user)
      {
            _user = user;
      }

      public void Display()
      {
            if (_allowed())
            {
                  _doc.Display();
            }
      }
}

class Program
{

      static void Main()
      {
            DocProxy protectedDoc = new DocProxy(new Document("Este
es un documento importante."), u => u.isAdmin());

            // Normally obtained via safer means
            User user = new AdminUser();

            protectedDoc.SetUser(user);
            protectedDoc.Display();
      }

}
```

6.

a. Facade
b. Creo la clase SystemApi que implementa los 3 métodos que se llaman en el program para no permitir que se interactúe con instancias de clases que en la mayoría de métodos no voy a usar

```
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}.");
		}
	}
public class ApiSystem{

	BillingSystem billingSystem = new BillingSystem()
	InventorySystem inventorySystem = new InventorySystem()
	CartSystem cartSystem = new CartSystem()

	public void ReduceStock(string product, int quantity)
	{
		inventorySystem.ReduceStock( product, quantity )
	}

	public void GenerateInvoice(string product, int quantity)
	{
		billingSystem.GenerateInvoice( product, quantity )
	}

	public void AddToCart(string product, int quantity)
	 {
		cartSystem.AddToCart(product, quantity)
	 }

}


}
class Program
{
	static void Main()
	 {
		ApiSystem apiSystem = new ApiSystem();
		string product = "Libro";
		int quantity = 2;
```

```
            ApiSystem.AddToCart(product, quantity);
            ApiSystem.ReduceStock(product, quantity);

            ApiSystem.GenerateInvoice(product, quantity);
        }
}
```

7.

a. Facade

b.

```
public class TwitterService
{

	private string _accessToken = null;
	private string _apiKey;
	private string _apiSecret;

	public TwitterService(string apiKey, string apiSecret)
	{
		_apiKey = apiKey;
		_apiSecret = apiSecret;
	}


	private string getAccessToken()
	{
		if (_accessToken == null)
		{
			_accessToken = _authenticator.Authenticate(_apiKey,
_apiSecret);
		}

		return _accessToken;
	}

	public int GetNumberOfPostsOfUser(string username)
	{
		string accessToken = getAccessToken();
		string jsonResponse =
_twitterApi.MakeApiRequest($"https://api.twitter.com/users/{username}",
accessToken);
		int postCount = _dataParser.ParsePostCount(jsonResponse);
		return postCount;
	}
}

class Program
{
	static void Main()
	{
		TwitterService twitterService = new TwitterService(“api_key”,
“api_secret”);
		int postCount =
twitterService.GetNomberOfPostsOfUser(“john_doe”);
		Console.WriteLine($"Cantidad de posts del usuario john_doe:
	{postCount}");
	}
}
```

8. Utilizamos Decorator para poder añadir estilos al texto de forma dinámica en runtime.

```
public interface IElementoTexto
{
        string ObtenerTexto();
        void SetEstiloFuente(string estiloFuente);
        void SetDecoracion(string decoracion);
        void SetColor(string color);
}

public class ElementoTexto : IElementoTexto
{
    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;
    }
}

public abstract class ElementoTextoDecorator : IElementoTexto
{
    protected IElementoTexto _wrappee;

    public ElementoTextoDecorator() {}
```

```
    public abstract string ObtenerTexto();

    public void SetWrappee(IElementoTexto wrappee)
    {
        this._wrappee = wrappee;
    }

    public void SetEstiloFuente(string estiloFuente)
    {
        this._wrappee.SetEstiloFuente(estiloFuente);
    }

    public void SetDecoracion(string decoracion)
    {
        this._wrappee.SetDecoracion(decoracion);
    }

    public void SetColor(string color)
    {
        this._wrappee.SetColor(color);
    }

}


public class ElementoTextoArial : ElementoTextoDecorator
{
    protected IElementoTexto _wrappee;

    public ElementoTextoDecorator() {}

    public string ObtenerTexto()
    {
        this._wrappee.SetEstiloFuente("Arial");
        return this._wrappee.ObtenerTexto();
    }
}

public class ElementoTextoRed : ElementoTextoDecorator
{
    protected IElementoTexto _wrappee;

    public ElementoTextoDecorator() {}

    public string ObtenerTexto()
    {
        this._wrappee.SetColor("red");
        return this._wrappee.ObtenerTexto();
    }
}
public class ElementoTextoUnderlined : ElementoTextoDecorator
{
    protected IElementoTexto _wrappee;

    public ElementoTextoDecorator() {}

    public string ObtenerTexto()
    {
        this._wrappee.SetDecoracion("underline");
        return this._wrappee.ObtenerTexto();
    }
}

class Program
{
```

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

         IElementoTexto arial = new ElementoTextoArial();
         IElementoTexto red = new ElementoTextoRed();
         IElementoTexto underlined = new ElementoTextoUnderlined();

         underlined.SetWrappee(red.SetWrappee(arial.SetWrappee(elementoTexto)))

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

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