UT3-TA4

Ejercicio 1

Patrón adapter

CÓDIGO:

public interface IDataExporter

{

string Export(object data);

}

// Adapter JSON

public class JsonExporter : IDataExporter

{

public string Export(object data)

{

return JsonConvert.SerializeObject(data);

}

}

// Adapter XML

public class XmlExporter : IDataExporter

{

public string Export(object data)

{

XmlSerializer xmlSerializer = new XmlSerializer(data.GetType());

using (StringWriter textWriter = new StringWriter())

{

xmlSerializer.Serialize(textWriter, data);

return textWriter.ToString();

}

}

}

// Adapter for TXT

public class TxtExporter : IDataExporter

{

public string Export(object data)

{

return data.ToString();

}

}

// Client

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()

{

var data = new { Name = "Juancito", Age = 30 };

DataService jsonService = new DataService(new JsonExporter());

Console.WriteLine("Datos en formato JSON:");

Console.WriteLine(jsonService.ExportData(data));

DataService xmlService = new DataService(new XmlExporter());

Console.WriteLine("\nDatos en formato XML:");

Console.WriteLine(xmlService.ExportData(data));

DataService txtService = new DataService(new TxtExporter());

Console.WriteLine("\nDatos en formato TXT:");

Console.WriteLine(txtService.ExportData(data));

}

}

Ejercicio 2

Patrón Adapter

CÓDIGO:

// Target interface

public interface IPaymentService

{

bool MakePayment(double amount, string currency);

}

// Adapter para SafePay

public class SafePayAdapter : IPaymentService

{

private readonly SafePayService \_safePayService;

public SafePayAdapter(SafePayService safePayService)

{

\_safePayService = safePayService;

}

public bool MakePayment(double amount, string currency)

{

\_safePayService.Transact("store\_account", "customer\_account", currency, amount);

return true; // Simulate success

}

}

public class OnlineStore

{

private readonly IPaymentService \_paymentService;

public OnlineStore(IPaymentService 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.");

}

}

}

class Program

{

static void Main()

{

IPaymentService quickPayService = new QuickPayService();

OnlineStore storeWithQuickPay = new OnlineStore(quickPayService);

storeWithQuickPay.Checkout(100, "USD");

SafePayService safePay = new SafePayService();

IPaymentService safePayAdapter = new SafePayAdapter(safePay);

OnlineStore storeWithSafePay = new OnlineStore(safePayAdapter);

storeWithSafePay.Checkout(200, "EUR");

}

}

Ejercicio 3

Patrón Decorator

CÓDIGO:

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}");

}

}

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); // Llama al método Send de la notificación original

SendSMS(message); // Añade comportamiento extra

}

private void SendSMS(string message)

{

Console.WriteLine($"Enviando SMS: {message}");

}

}

public class WhatsAppNotificationDecorator : NotificationDecorator

{

public WhatsAppNotificationDecorator(Notification notification) : base(notification)

{

}

public override void Send(string message)

{

base.Send(message); // Llama al método Send de la notificación original

SendWhatsApp(message); // Añade comportamiento extra

}

private void SendWhatsApp(string message)

{

Console.WriteLine($"Enviando WhatsApp: {message}");

}

}

//Ahora vamos a ver un ejemplo de como usarlo  
class Program

{

static void Main(string[] args)

{

Notification emailNotification = new EmailNotification();

// Decoramos con notificaciones adicionales

Notification smsAndEmailNotification = new SMSNotificationDecorator(emailNotification);

Notification allNotifications = new WhatsAppNotificationDecorator(smsAndEmailNotification);

allNotifications.Send("Hola mundo!");

// Esto en TEoRÍA enviará el mensaje por correo, SMS y WhatsApp

}

}

Ejercicio 4

Patrón: Facade

CÓDIGO:

//ESTE ME DA DUDA

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 ReserveRoomAndBookTable(string roomType, string tableType)

{

\_reservationSystem.ReserveRoom(roomType);

\_restaurantSystem.BookTable(tableType);

}

public void ScheduleCleaning(string roomNumber)

{

\_cleaningSystem.ScheduleRoomCleaning(roomNumber);

}

}

class Program

{

static void Main()

{

HotelManagementFacade hotelFacade = new HotelManagementFacade();

hotelFacade.ReserveRoomAndBookTable("Deluxe", "VIP");

hotelFacade.ScheduleCleaning("101");

}

}

Ejercicio 5

Patrón: Proxy

CÓDIGO:

// Subject

public interface IDocument

{

void Display();

}

// Real Subject

public class Document : IDocument

{

private readonly string \_content;

public Document(string content)

{

\_content = content;

}

public void Display()

{

Console.WriteLine($"Contenido del documento: {\_content}");

}

}

// Proxy

public class DocumentProxy : IDocument

{

private readonly Document \_document;

private readonly bool \_hasAccess;

public DocumentProxy(string content, bool hasAccess)

{

\_document = new Document(content);

\_hasAccess = hasAccess;

}

public void Display()

{

if (\_hasAccess)

{

\_document.Display();

}

else

{

Console.WriteLine("Acceso denegado. No tienes permiso para ver este documento.");

}

}

}

// Client

class Program

{

static void Main()

{

DocumentProxy document = new DocumentProxy("Este es un documento importante.", true);

document.Display();

DocumentProxy restrictedDocument = new DocumentProxy("Este es un documento importante.", false);

restrictedDocument.Display();

}

}

Ejercicio 6

Patrón: Facade

CÓDIGO:

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)

{

\_cartSystem.AddToCart(product, quantity);

\_inventorySystem.ReduceStock(product, quantity);

\_billingSystem.GenerateInvoice(product, quantity);

}

}

class Program

{

static void Main()

{

OrderFacade orderFacade = new OrderFacade();

orderFacade.PlaceOrder("Libro", 2);

}

}

Ejercicio 7

Patrón: Facade

CÓDIGO:

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 user)

{

string accessToken = \_authenticator.Authenticate(apiKey, apiSecret);

string jsonResponse = \_twitterApi.MakeApiRequest($"https://api.twitter.com/users/{user}", accessToken);

return \_dataParser.ParsePostCount(jsonResponse);

}

}

class Program

{

static void Main()

{

TwitterFacade twitterFacade = new TwitterFacade();

int postCount = twitterFacade.GetPostCount("api\_key", "api\_secret", "john\_doe");

Console.WriteLine($"Cantidad de posts del usuario john\_doe: {postCount}");

}

}

Ejercicio 8

Patrón: Decorator

CÓDIGO:

// Clase base

public abstract class ElementoTexto

{

public abstract string ObtenerTexto();

}

// Clase concreta que contiene el texto

public class TextoSimple : ElementoTexto

{

private string \_texto;

public TextoSimple(string texto)

{

\_texto = texto;

}

public override string ObtenerTexto()

{

return \_texto;

}

}

// Clase base para los decoradores

public abstract class TextoDecorador : ElementoTexto

{

protected ElementoTexto \_elementoTexto;

public TextoDecorador(ElementoTexto elementoTexto)

{

\_elementoTexto = elementoTexto;

}

public override string ObtenerTexto()

{

return \_elementoTexto.ObtenerTexto();

}

}

// Decorador concreto para el estilo de fuente

public class EstiloFuenteDecorador : TextoDecorador

{

private string \_estiloFuente;

public EstiloFuenteDecorador(ElementoTexto elementoTexto, string estiloFuente)

: base(elementoTexto)

{

\_estiloFuente = estiloFuente;

}

public override string ObtenerTexto()

{

return $"<span style=\"font-family: {\_estiloFuente}\">{base.ObtenerTexto()}</span>";

}

}

// Decorador concreto para el color

public class ColorDecorador : TextoDecorador

{

private string \_color;

public ColorDecorador(ElementoTexto elementoTexto, string color)

: base(elementoTexto)

{

\_color = color;

}

public override string ObtenerTexto()

{

return $"<span style=\"color: {\_color}\">{base.ObtenerTexto()}</span>";

}

}

// Decorador concreto para la decoración

public class DecoracionDecorador : TextoDecorador

{

private string \_decoracion;

public DecoracionDecorador(ElementoTexto elementoTexto, string decoracion)

: base(elementoTexto)

{

\_decoracion = decoracion;

}

public override string ObtenerTexto()

{

return $"<span style=\"text-decoration: {\_decoracion}\">{base.ObtenerTexto()}</span>";

}

}

// Clase Program

class Program

{

static void Main()

{

// Crear una instancia de un elemento de texto

ElementoTexto elementoTexto = new TextoSimple("Hola, mundo!");

// Decorar el texto con diferentes estilos

elementoTexto = new EstiloFuenteDecorador(elementoTexto, "Arial");

elementoTexto = new ColorDecorador(elementoTexto, "red");

elementoTexto = new DecoracionDecorador(elementoTexto, "underline");

// Obtener el texto con la apariencia personalizada

string textoPersonalizado = elementoTexto.ObtenerTexto();

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

}

}