**Exercise 1: Implementing the Singleton Pattern**

using System;

public class Logger

{

    private static Logger instance;

    private static readonly object lockObj = new object();

    private Logger() {}

    public static Logger GetInstance()

    {

        if (instance == null)

        {

            lock (lockObj)

            {

                if (instance == null)

                    instance = new Logger();

            }

        }

        return instance;

    }

    public void Log(string message) => Console.WriteLine("Log: " + message);

}

public class Program

{

    public static void Main()

    {

        var logger1 = Logger.GetInstance();

        var logger2 = Logger.GetInstance();

        logger1.Log("Message 1");

        logger2.Log("Message 2");

        Console.WriteLine("Same instance: " + (logger1 == logger2));

    }

}

**Output:-**

**Screenshot 2025-06-19 225912**

**Exercise 2: Implementing the Factory Method Pattern**

using System;

public interface IDocument { void Open(); }

public class WordDocument : IDocument { public void Open() => Console.WriteLine("Opening Word Document"); }

public class PdfDocument : IDocument { public void Open() => Console.WriteLine("Opening PDF Document"); }

public class ExcelDocument : IDocument { public void Open() => Console.WriteLine("Opening Excel Document"); }

public abstract class DocumentFactory { public abstract IDocument CreateDocument(); }

public class WordFactory : DocumentFactory { public override IDocument CreateDocument() => new WordDocument(); }

public class PdfFactory : DocumentFactory { public override IDocument CreateDocument() => new PdfDocument(); }

public class ExcelFactory : DocumentFactory { public override IDocument CreateDocument() => new ExcelDocument(); }

public class Program

{

    public static void Main()

    {

        DocumentFactory factory = new WordFactory();

        factory.CreateDocument().Open();

        factory = new PdfFactory();

        factory.CreateDocument().Open();

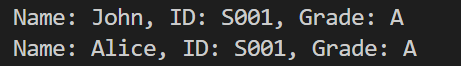
        factory = new ExcelFactory();

        factory.CreateDocument().Open();

    }

}

**Output:-**

****

**Exercise 3: Implementing the Builder Pattern**

using System;

public class Computer

{

    public string CPU { get; private set; }

    public string RAM { get; private set; }

    public string Storage { get; private set; }

    private Computer() { }

    public class Builder

    {

        private readonly Computer computer = new Computer();

        public Builder SetCPU(string cpu) { computer.CPU = cpu; return this; }

        public Builder SetRAM(string ram) { computer.RAM = ram; return this; }

        public Builder SetStorage(string storage) { computer.Storage = storage; return this; }

        public Computer Build() => computer;

    }

}

public class Program

{

    public static void Main()

    {

        var pc = new Computer.Builder().SetCPU("Intel i9").SetRAM("32GB").SetStorage("1TB SSD").Build();

        Console.WriteLine($"CPU={pc.CPU}, RAM={pc.RAM}, Storage={pc.Storage}");

    }

}

**Output:-**

**Screenshot 2025-06-19 225752**

**Exercise 4: Implementing the Adapter Pattern**

using System;

public interface IPaymentProcessor { void ProcessPayment(); }

public class OldGateway { public void MakePayment() => Console.WriteLine("OldGateway payment processed"); }

public class NewGateway { public void CompleteTransaction() => Console.WriteLine("NewGateway payment processed"); }

public class OldGatewayAdapter : IPaymentProcessor

{

    private readonly OldGateway old = new OldGateway();

    public void ProcessPayment() => old.MakePayment();

}

public class NewGatewayAdapter : IPaymentProcessor

{

    private readonly NewGateway nw = new NewGateway();

    public void ProcessPayment() => nw.CompleteTransaction();

}

public class Program

{

    public static void Main()

    {

        IPaymentProcessor oldAdapter = new OldGatewayAdapter();

        oldAdapter.ProcessPayment();

        IPaymentProcessor newAdapter = new NewGatewayAdapter();

        newAdapter.ProcessPayment();

    }

}

**Output:-**

**Screenshot 2025-06-19 225712**

**Exercise 5: Implementing the Decorator Pattern**

using System;

public interface INotifier { void Send(); }

public class EmailNotifier : INotifier { public void Send() => Console.WriteLine("Email sent"); }

public abstract class NotifierDecorator : INotifier

{

    protected INotifier wrappee;

    public NotifierDecorator(INotifier notifier) { wrappee = notifier; }

    public virtual void Send() => wrappee.Send();

}

public class SMSNotifier : NotifierDecorator

{

    public SMSNotifier(INotifier notifier) : base(notifier) {}

    public override void Send() { base.Send(); Console.WriteLine("SMS sent"); }

}

public class Program

{

    public static void Main()

    {

        INotifier notifier = new SMSNotifier(new EmailNotifier());

        notifier.Send();

    }

}

**Output:-**

**Screenshot 2025-06-19 225651**

**Exercise 6: Implementing the Proxy Pattern**

using System;

public interface IImage { void Display(); }

public class RealImage : IImage

{

    private string filename;

    public RealImage(string filename) { this.filename = filename; Load(); }

    private void Load() => Console.WriteLine("Loading " + filename);

    public void Display() => Console.WriteLine("Displaying " + filename);

}

public class ProxyImage : IImage

{

    private RealImage realImage;

    private string filename;

    public ProxyImage(string filename) { this.filename = filename; }

    public void Display()

    {

        if (realImage == null)

            realImage = new RealImage(filename);

        realImage.Display();

    }

}

public class Program

{

    public static void Main()

    {

        IImage img = new ProxyImage("photo.jpg");

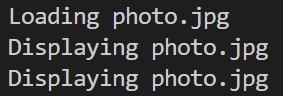
        img.Display();

        img.Display();

    }

}

**Output:-**

****

**Exercise 7: Implementing the Observer Pattern**

using System;

using System.Collections.Generic;

public interface IObserver { void Update(float price); }

public interface IStock { void Register(IObserver o); void Remove(IObserver o); void Notify(); }

public class StockMarket : IStock

{

    private List<IObserver> observers = new();

    private float price;

    public void Register(IObserver o) => observers.Add(o);

    public void Remove(IObserver o) => observers.Remove(o);

    public void SetPrice(float p) { price = p; Notify(); }

    public void Notify() { foreach (var o in observers) o.Update(price); }

}

public class MobileApp : IObserver { public void Update(float price) => Console.WriteLine("MobileApp: " + price); }

public class WebApp : IObserver { public void Update(float price) => Console.WriteLine("WebApp: " + price); }

public class Program

{

    public static void Main()

    {

        var market = new StockMarket();

        var mobile = new MobileApp();

        var web = new WebApp();

        market.Register(mobile);

        market.Register(web);

        market.SetPrice(120.75f);

    }

}

**Output:-**

**Screenshot 2025-06-19 225543**

**Exercise 8: Implementing the Strategy Pattern**

using System;

public interface IPaymentStrategy { void Pay(); }

public class CreditCardPayment : IPaymentStrategy { public void Pay() => Console.WriteLine("Paid by Credit Card"); }

public class PayPalPayment : IPaymentStrategy { public void Pay() => Console.WriteLine("Paid by PayPal"); }

public class PaymentContext

{

    private IPaymentStrategy strategy;

    public void SetStrategy(IPaymentStrategy strategy) { this.strategy = strategy; }

    public void Pay() { strategy.Pay(); }

}

public class Program

{

    public static void Main()

    {

        var context = new PaymentContext();

        context.SetStrategy(new CreditCardPayment());

        context.Pay();

        context.SetStrategy(new PayPalPayment());

        context.Pay();

    }

}

**Output:-**

**Screenshot 2025-06-19 225459**

**Exercise 9: Implementing the Command Pattern**

using System;

public interface ICommand { void Execute(); }

public class Light { public void On() => Console.WriteLine("Light ON"); public void Off() => Console.WriteLine("Light OFF"); }

public class LightOnCommand : ICommand

{

    private Light light;

    public LightOnCommand(Light light) { this.light = light; }

    public void Execute() => light.On();

}

public class LightOffCommand : ICommand

{

    private Light light;

    public LightOffCommand(Light light) { this.light = light; }

    public void Execute() => light.Off();

}

public class RemoteControl

{

    private ICommand command;

    public void SetCommand(ICommand command) { this.command = command; }

    public void PressButton() => command.Execute();

}

public class Program

{

    public static void Main()

    {

        var light = new Light();

        var on = new LightOnCommand(light);

        var off = new LightOffCommand(light);

        var remote = new RemoteControl();

        remote.SetCommand(on);

        remote.PressButton();

        remote.SetCommand(off);

        remote.PressButton();

    }

}

**Output:-**

**Screenshot 2025-06-19 224959**

**Exercise 10: Implementing the MVC Pattern**

using System;

public interface ICommand { void Execute(); }

public class Light { public void On() => Console.WriteLine("Light ON"); public void Off() => Console.WriteLine("Light OFF"); }

public class LightOnCommand : ICommand

{

    private Light light;

    public LightOnCommand(Light light) { this.light = light; }

    public void Execute() => light.On();

}

public class LightOffCommand : ICommand

{

    private Light light;

    public LightOffCommand(Light light) { this.light = light; }

    public void Execute() => light.Off();

}

public class RemoteControl

{

    private ICommand command;

    public void SetCommand(ICommand command) { this.command = command; }

    public void PressButton() => command.Execute();

}

public class Program

{

    public static void Main()

    {

        var light = new Light();

        var on = new LightOnCommand(light);

        var off = new LightOffCommand(light);

        var remote = new RemoteControl();

        remote.SetCommand(on);

        remote.PressButton();

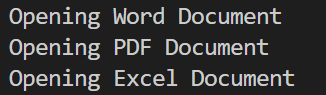
        remote.SetCommand(off);

        remote.PressButton();

    }

}

**Output:-**

****

**Exercise 11: Implementing Dependency Injection**

using System;

public interface ICustomerRepository { string GetCustomer(int id); }

public class CustomerRepository : ICustomerRepository

{

    public string GetCustomer(int id) => $"Customer with ID {id}";

}

public class CustomerService

{

    private readonly ICustomerRepository repo;

    public CustomerService(ICustomerRepository repo) { this.repo = repo; }

    public void Display(int id) => Console.WriteLine(repo.GetCustomer(id));

}

public class Program

{

    public static void Main()

    {

        ICustomerRepository repo = new CustomerRepository();

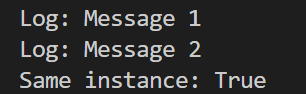
        var service = new CustomerService(repo);

        service.Display(101);

    }

}

**Output:-**

****