**DESIGN PATTERNS AND PRINCIPLES**

**Exercise 1: Implementing the Singleton Pattern**

Logger.cs

using System;

namespace SingletonPatternExample

{

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}");

}

}

}

Program.cs

using System;

namespace SingletonPatternExample

{

internal class Program

{

static void Main(string[] args)

{

Logger logger1 = Logger.GetInstance();

Logger logger2 = Logger.GetInstance();

logger1.Log("First log message");

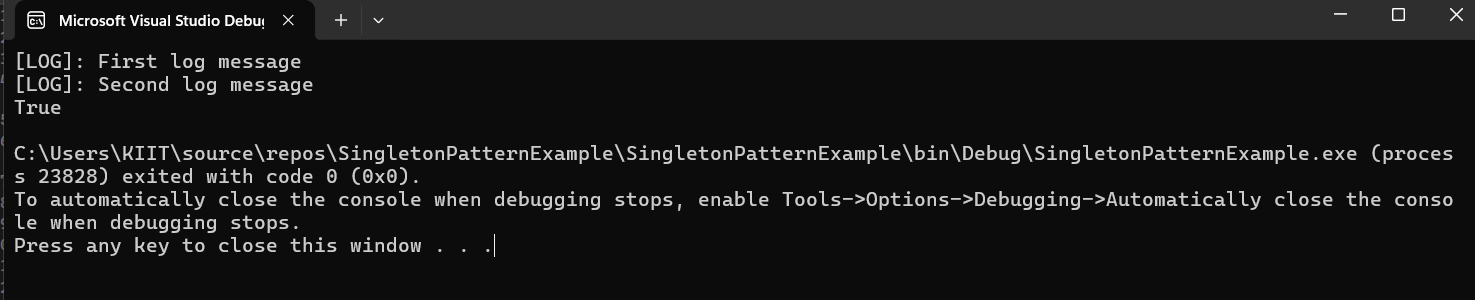
logger2.Log("Second log message");

Console.WriteLine(object.ReferenceEquals(logger1, logger2)); // True if same instance

}

}

}



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

**Document.cs**

namespace FactoryMethodPatternExample

{

public interface IDocument

{

string GetTypeName();

}

public class WordDocument : IDocument

{

public string GetTypeName() => "Word Document";

}

public class PdfDocument : IDocument

{

public string GetTypeName() => "PDF Document";

}

public class ExcelDocument : IDocument

{

public string GetTypeName() => "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();

}

}

**Program.cs**

using System;

namespace FactoryMethodPatternExample

{

internal class Program

{

static void Main(string[] args)

{

DocumentFactory wordFactory = new WordFactory();

IDocument wordDoc = wordFactory.CreateDocument();

Console.WriteLine(wordDoc.GetTypeName());

DocumentFactory pdfFactory = new PdfFactory();

IDocument pdfDoc = pdfFactory.CreateDocument();

Console.WriteLine(pdfDoc.GetTypeName());

DocumentFactory excelFactory = new ExcelFactory();

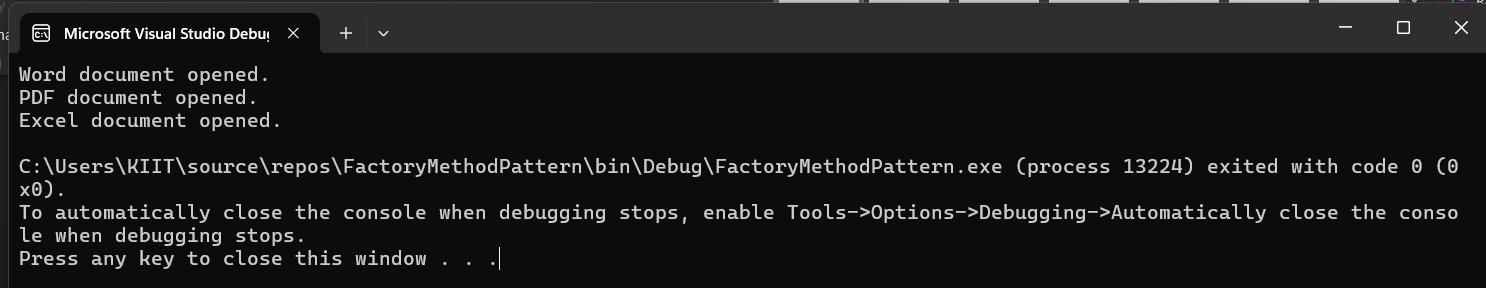
IDocument excelDoc = excelFactory.CreateDocument();

Console.WriteLine(excelDoc.GetTypeName());

}

}

}



**Exercise 3: Implementing the Builder Pattern**

Computer.cs

using System;

namespace BuilderPatternExample

{

public class Computer

{

public string CPU { get; private set; }

public string RAM { get; private set; }

public string Storage { get; private set; }

public string GPU { get; private set; }

private Computer(Builder builder)

{

CPU = builder.CPU;

RAM = builder.RAM;

Storage = builder.Storage;

GPU = builder.GPU;

}

public class Builder

{

public string CPU { get; private set; }

public string RAM { get; private set; }

public string Storage { get; private set; }

public string GPU { get; private set; }

public Builder SetCPU(string cpu)

{

CPU = cpu;

return this;

}

public Builder SetRAM(string ram)

{

RAM = ram;

return this;

}

public Builder SetStorage(string storage)

{

Storage = storage;

return this;

}

public Builder SetGPU(string gpu)

{

GPU = gpu;

return this;

}

public Computer Build()

{

return new Computer(this);

}

}

public void DisplayConfig()

{

Console.WriteLine("Computer Configuration:");

Console.WriteLine($"CPU: {CPU}");

Console.WriteLine($"RAM: {RAM}");

Console.WriteLine($"Storage: {Storage}");

Console.WriteLine($"GPU: {GPU}");

}

}

}

Program.cs

using System;

namespace BuilderPatternExample

{

internal class Program

{

static void Main(string[] args)

{

Computer gamingPC = new Computer.Builder()

.SetCPU("Intel i9")

.SetRAM("32GB")

.SetStorage("1TB SSD")

.SetGPU("NVIDIA RTX 4090")

.Build();

gamingPC.DisplayConfig();

Computer officePC = new Computer.Builder()

.SetCPU("Intel i5")

.SetRAM("8GB")

.SetStorage("512GB SSD")

.Build();

officePC.DisplayConfig();

}

}

}



**Exercise 4: Implementing the Adapter Pattern**

Paymentclasses.cs

using System;

namespace AdapterPatternExample

{

public interface IPaymentProcessor

{

void ProcessPayment(string amount);

}

public class PhonePeGateway

{

public void PayViaPhonePe(string amount)

{

Console.WriteLine($"Paid {amount} using PhonePe");

}

}

public class GPayGateway

{

public void MakeGPayPayment(string amount)

{

Console.WriteLine($"Paid {amount} using GPay");

}

}

public class PhonePeAdapter : IPaymentProcessor

{

private PhonePeGateway phonepe;

public PhonePeAdapter()

{

phonepe = new PhonePeGateway();

}

public void ProcessPayment(string amount)

{

phonepe.PayViaPhonePe(amount);

}

}

public class GPayAdapter : IPaymentProcessor

{

private GPayGateway gpay;

public GPayAdapter()

{

gpay = new GPayGateway();

}

public void ProcessPayment(string amount)

{

gpay.MakeGPayPayment(amount);

}

}

}

Program.cs

using System;

namespace AdapterPatternExample

{

internal class Program

{

static void Main(string[] args{

IPaymentProcessor phonepe = new PhonePeAdapter();

IPaymentProcessor gpay = new GPayAdapter();

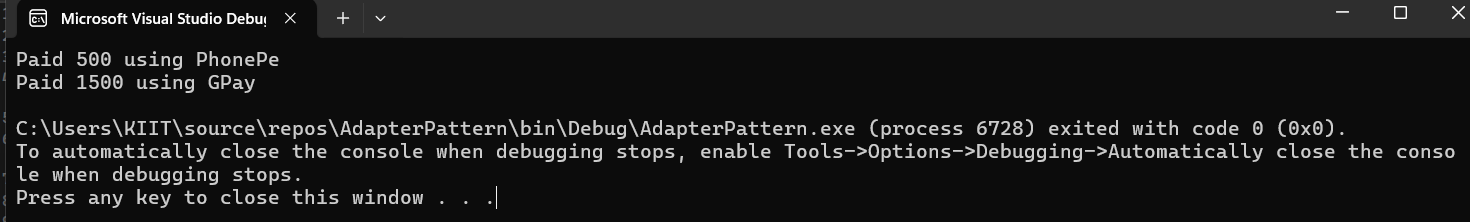
phonepe.ProcessPayment("500");

gpay.ProcessPayment("1500");

}

}

}



**Exercise 5: Implementing the Decorator Pattern**

using System;

namespace DecoratorPatternExample

{

// Step 1: Define Component Interface

public interface INotifier

{

void Send(string message);

}

// Step 2: Concrete Component

public class EmailNotifier : INotifier

{

public void Send(string message)

{

Console.WriteLine("Sending Email: " + message);

}

}

// Step 3: Abstract Decorator

public abstract class NotifierDecorator : INotifier

{

protected INotifier notifier;

public NotifierDecorator(INotifier notifier)

{

this.notifier = notifier;

}

public virtual void Send(string message)

{

notifier.Send(message);

}

}

// Step 4: Concrete Decorator - SMS

public class SMSNotifierDecorator : NotifierDecorator

{

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

public override void Send(string message)

{

base.Send(message); // Email first

Console.WriteLine("Sending SMS: " + message);

}

}

// Step 4: Concrete Decorator - Slack

public class SlackNotifierDecorator : NotifierDecorator

{

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

public override void Send(string message)

{

base.Send(message); // Previous (email/sms)

Console.WriteLine("Sending Slack Message: " + message);

}

}

// Step 5: Test class

class Program

{

static void Main(string[] args)

{

INotifier notifier = new EmailNotifier();

// Wrap with SMS and then Slack

notifier = new SMSNotifierDecorator(notifier);

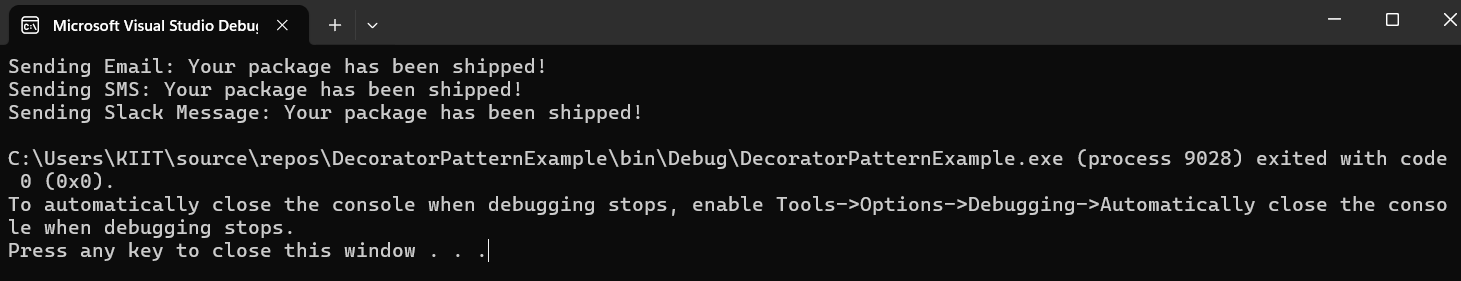
notifier = new SlackNotifierDecorator(notifier);

notifier.Send("Your package has been shipped!");

}

}

}



**Exercise 6: Implementing the Proxy Pattern**

using System;

namespace ProxyPatternExample

{

// Step 1: Subject Interface

public interface IImage

{

void Display();

}

// Step 2: Real Subject

public class RealImage : IImage

{

private string filename;

public RealImage(string filename)

{

this.filename = filename;

LoadFromDisk();

}

private void LoadFromDisk()

{

Console.WriteLine("Loading image from disk: " + filename);

}

public void Display()

{

Console.WriteLine("Displaying image: " + filename);

}

}

// Step 3: Proxy

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); // Lazy loading

}

realImage.Display();

}

}

// Step 4: Test Class

class Program

{

static void Main(string[] args)

{

IImage image = new ProxyImage("cat\_picture.jpg");

Console.WriteLine("Image object created.");

Console.WriteLine("Calling Display for the first time...");

image.Display(); // Loads and displays

Console.WriteLine("\nCalling Display again...");

image.Display(); // Displays only, no reloading

}

}

}



**Exercise 7: Implementing the Observer Pattern**

using System;

using System.Collections.Generic;

namespace ObserverPatternExample

{

// Step 1: Observer Interface

public interface IObserver

{

void Update(string stockName, double price);

}

// Step 2: Subject Interface

public interface IStock

{

void RegisterObserver(IObserver observer);

void RemoveObserver(IObserver observer);

void NotifyObservers();

}

// Step 3: Concrete Subject

public class StockMarket : IStock

{

private List<IObserver> observers = new List<IObserver>();

private string stockName;

private double stockPrice;

public void SetStock(string name, double price)

{

stockName = name;

stockPrice = price;

NotifyObservers();

}

public void RegisterObserver(IObserver observer)

{

observers.Add(observer);

}

public void RemoveObserver(IObserver observer)

{

observers.Remove(observer);

}

public void NotifyObservers()

{

foreach (var observer in observers)

{

observer.Update(stockName, stockPrice);

}

}

}

// Step 4: Concrete Observers

public class MobileApp : IObserver

{

public void Update(string stockName, double price)

{

Console.WriteLine($"Mobile App: {stockName} is now Rs {price}");

}

}

public class WebApp : IObserver

{

public void Update(string stockName, double price)

{

Console.WriteLine($"Web App: {stockName} updated to Rs {price}");

}

}

// Step 5: Test Class

class Program

{

static void Main(string[] args)

{

StockMarket market = new StockMarket();

IObserver mobileApp = new MobileApp();

IObserver webApp = new WebApp();

market.RegisterObserver(mobileApp);

market.RegisterObserver(webApp);

market.SetStock("TCS", 3480.25);

market.SetStock("INFY", 1575.50);

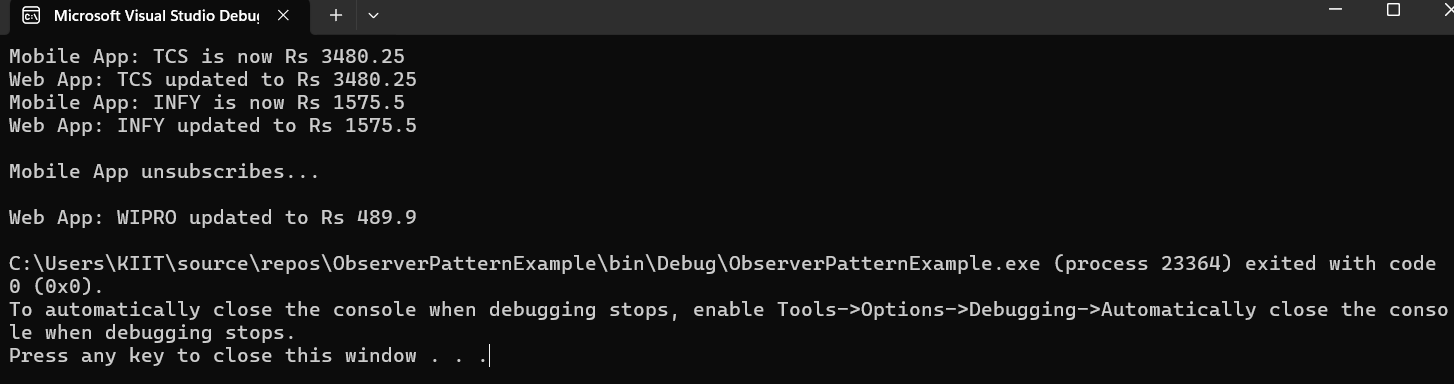
Console.WriteLine("\nMobile App unsubscribes...\n");

market.RemoveObserver(mobileApp);

market.SetStock("WIPRO", 489.90);

}

}



**Exercise 8: Implementing the Strategy Pattern**

using System;

namespace StrategyPatternExample

{

// Step 1: Strategy Interface

public interface IPaymentStrategy

{

void Pay(double amount);

}

// Step 2: Concrete Strategy - Credit Card

public class CreditCardPayment : IPaymentStrategy

{

public void Pay(double amount)

{

Console.WriteLine("Paid Rs " + amount + " using Credit Card.");

}

}

// Step 2: Concrete Strategy - PayPal

public class PayPalPayment : IPaymentStrategy

{

public void Pay(double amount)

{

Console.WriteLine("Paid Rs " + amount + " using PayPal.");

}

}

// Step 2: Concrete Strategy - UPI

public class UpiPayment : IPaymentStrategy

{

public void Pay(double amount)

{

Console.WriteLine("Paid Rs " + amount + " using UPI.");

}

}

// Step 3: Context Class

public class PaymentContext

{

private IPaymentStrategy paymentStrategy;

public void SetPaymentStrategy(IPaymentStrategy strategy)

{

paymentStrategy = strategy;

}

public void PayAmount(double amount)

{

if (paymentStrategy == null)

{

Console.WriteLine("Please select a payment method first.");

}

else

{

paymentStrategy.Pay(amount);

}

}

}

// Step 4: Test class

class Program

{

static void Main(string[] args)

{

PaymentContext payment = new PaymentContext();

Console.WriteLine("Choose Payment Method:");

Console.WriteLine("1. Credit Card");

Console.WriteLine("2. PayPal");

Console.WriteLine("3. UPI");

Console.Write("Enter option: ");

int choice = Convert.ToInt32(Console.ReadLine());

switch (choice)

{

case 1:

payment.SetPaymentStrategy(new CreditCardPayment());

break;

case 2:

payment.SetPaymentStrategy(new PayPalPayment());

break;

case 3:

payment.SetPaymentStrategy(new UpiPayment());

break;

default:

Console.WriteLine("Invalid choice.");

return;

}

Console.Write("Enter amount (in Rs): ");

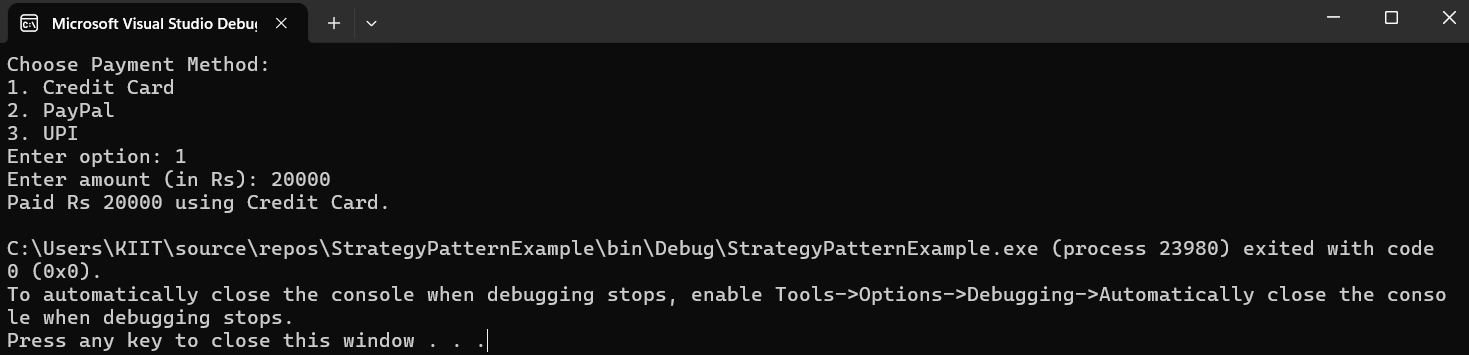
double amount = Convert.ToDouble(Console.ReadLine());

payment.PayAmount(amount);

}

}

}



**Exercise 9: Implementing the Command Pattern**

using System;

namespace CommandPatternExample

{

// Step 1: Command Interface

public interface ICommand

{

void Execute();

}

// Step 2: Receiver Class (Light)

public class Light

{

public void TurnOn()

{

Console.WriteLine("Light is ON");

}

public void TurnOff()

{

Console.WriteLine("Light is OFF");

}

}

// Step 3: Concrete Command - Light On

public class LightOnCommand : ICommand

{

private Light light;

public LightOnCommand(Light light)

{

this.light = light;

}

public void Execute()

{

light.TurnOn();

}

}

// Step 3: Concrete Command - Light Off

public class LightOffCommand : ICommand

{

private Light light;

public LightOffCommand(Light light)

{

this.light = light;

}

public void Execute()

{

light.TurnOff();

}

}

// Step 4: Invoker Class

public class RemoteControl

{

private ICommand command;

public void SetCommand(ICommand command)

{

this.command = command;

}

public void PressButton()

{

if (command != null)

{

command.Execute();

}

else

{

Console.WriteLine("No command set.");

}

}

}

// Step 5: Test class

class Program

{

static void Main(string[] args)

{

Light livingRoomLight = new Light();

ICommand lightOn = new LightOnCommand(livingRoomLight);

ICommand lightOff = new LightOffCommand(livingRoomLight);

RemoteControl remote = new RemoteControl();

Console.WriteLine("Choose Command:");

Console.WriteLine("1. Turn Light ON");

Console.WriteLine("2. Turn Light OFF");

Console.Write("Enter choice: ");

int choice = Convert.ToInt32(Console.ReadLine());

if (choice == 1)

{

remote.SetCommand(lightOn);

}

else if (choice == 2)

{

remote.SetCommand(lightOff);

}

else

{

Console.WriteLine("Invalid choice.");

return;

}

Console.WriteLine("Pressing button on remote...");

remote.PressButton();

}

}

}

**Exercise 10: Implementing the MVC Pattern**

using System;

namespace MVCPatternExample

{

// Step 1: Model Class

public class Student

{

public string Name { get; set; }

public int Id { get; set; }

public string Grade { get; set; }

}

// Step 2: View Class

public class StudentView

{

public void DisplayStudentDetails(Student student)

{

Console.WriteLine("Student Details:");

Console.WriteLine("ID: " + student.Id);

Console.WriteLine("Name: " + student.Name);

Console.WriteLine("Grade: " + student.Grade);

}

}

// Step 3: Controller Class

public class StudentController

{

private Student model;

private StudentView view;

public StudentController(Student model, StudentView view)

{

this.model = model;

this.view = view;

}

public void SetStudentName(string name)

{

model.Name = name;

}

public void SetStudentGrade(string grade)

{

model.Grade = grade;

}

public void UpdateView()

{

view.DisplayStudentDetails(model);

}

}

// Step 4: Test Class

class Program

{

static void Main(string[] args)

{

// Creating student data

Student student = new Student { Id = 1, Name = "Ravi", Grade = "A" };

StudentView view = new StudentView();

StudentController controller = new StudentController(student, view);

// Display initial details

controller.UpdateView();

Console.WriteLine("\nUpdating student name and grade...\n");

controller.SetStudentName("Priya");

controller.SetStudentGrade("B+");

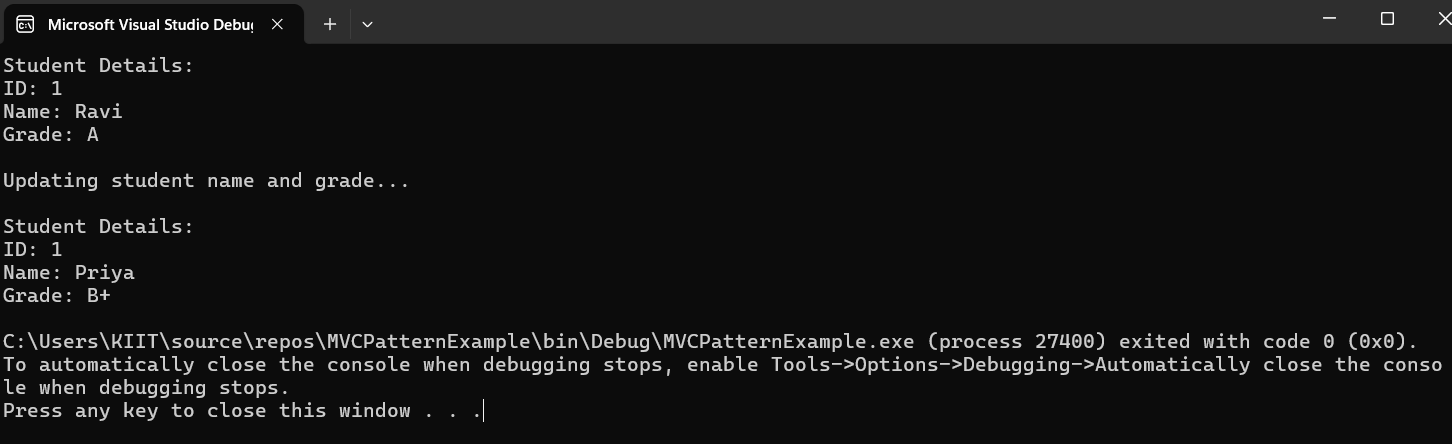
// Display updated details

controller.UpdateView();

}

}

}



**Exercise 11: Implementing Dependency Injection**

using System;

namespace DependencyInjectionExample

{

// Step 1: Repository Interface

public interface ICustomerRepository

{

string FindCustomerById(int id);

}

// Step 2: Concrete Repository

public class CustomerRepositoryImpl : ICustomerRepository

{

public string FindCustomerById(int id)

{

// Dummy data — normally this comes from a database

if (id == 1)

return "Customer: Rahul, Phone: 9876543210";

else if (id == 2)

return "Customer: Neha, Phone: 9123456789";

else

return "Customer not found.";

}

}

// Step 3: Service Class (depends on repository)

public class CustomerService

{

private ICustomerRepository repository;

// Constructor Injection

public CustomerService(ICustomerRepository repository)

{

this.repository = repository;

}

public void DisplayCustomerDetails(int customerId)

{

string customerInfo = repository.FindCustomerById(customerId);

Console.WriteLine(customerInfo);

}

}

// Step 4: Test Class

class Program

{

static void Main(string[] args)

{

// Injecting the dependency via constructor

ICustomerRepository repo = new CustomerRepositoryImpl();

CustomerService service = new CustomerService(repo);

Console.Write("Enter customer ID to search (1 or 2): ");

int id = Convert.ToInt32(Console.ReadLine());

service.DisplayCustomerDetails(id);

}

}

}

