**Design Patterns and Principles - Exercises**

**1.Singleton Pattern**

**Code:**

**using System;**

**class Logger**

**{**

**private static Logger instance;**

**// Private constructor to prevent instantiation**

**private Logger() {}**

**public static Logger GetInstance()**

**{**

**if (instance == null)**

**{**

**instance = new Logger();**

**}**

**return instance;**

**}**

**}**

**class SingletonTest**

**{**

**static void Main(string[] args)**

**{**

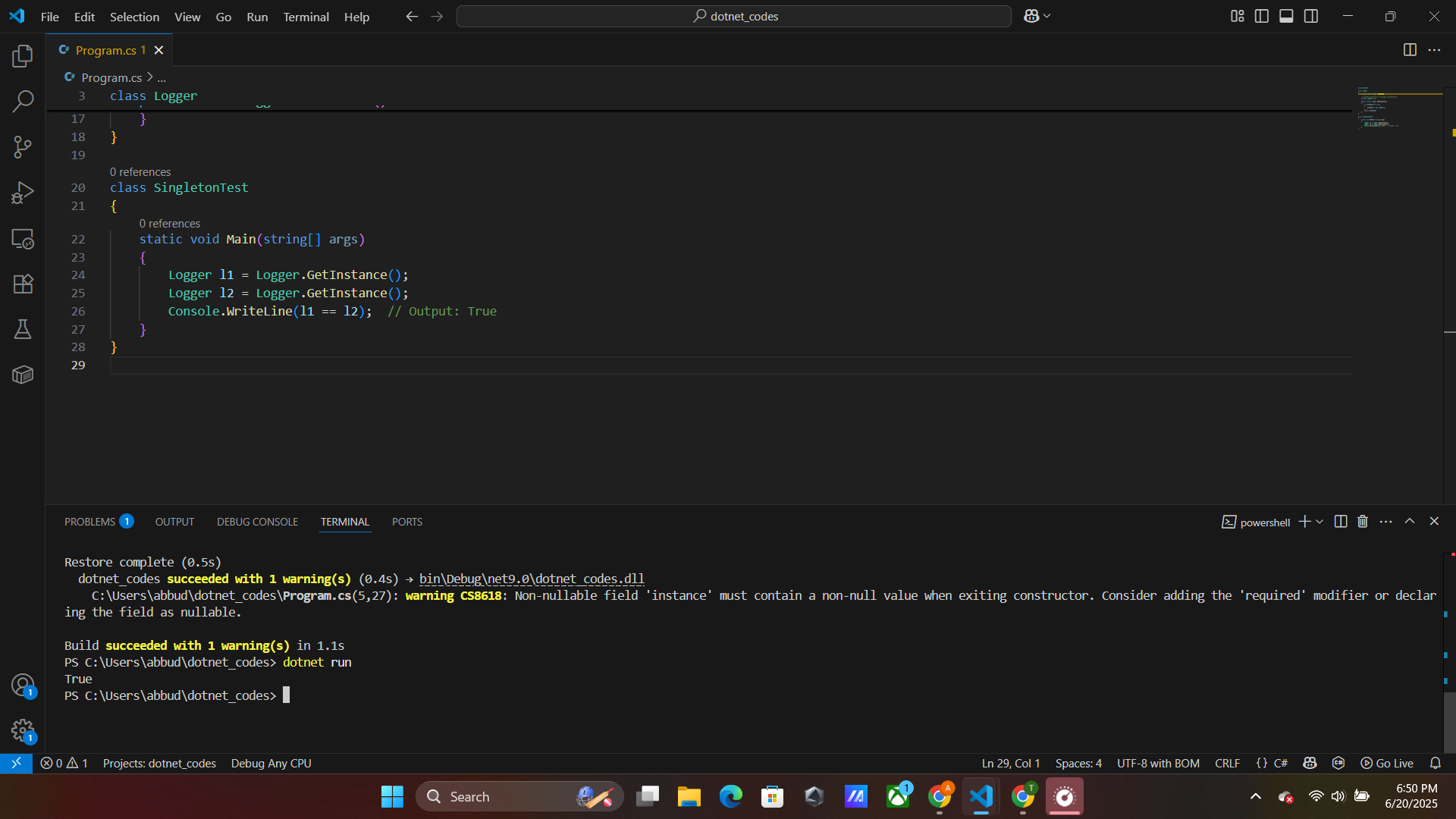
**Logger l1 = Logger.GetInstance();**

**Logger l2 = Logger.GetInstance();**

**Console.WriteLine(l1 == l2); // Output: True**

**}**

**}**

**Output** [**SingletonPattern.cs**](http://singletonpattern.cs)**:  
 **

**Exercise 2. Factory Method pattern**

**Code:**

**using System;**

**// Interface**

**interface IDocument**

**{**

**void Open();**

**}**

**// Concrete Products**

**class WordDocument : IDocument**

**{**

**public void Open()**

**{**

**Console.WriteLine("Word");**

**}**

**}**

**class PdfDocument : IDocument**

**{**

**public void Open()**

**{**

**Console.WriteLine("PDF");**

**}**

**}**

**class ExcelDocument : IDocument**

**{**

**public void Open()**

**{**

**Console.WriteLine("Excel");**

**}**

**}**

**// Abstract Factory**

**abstract class DocumentFactory**

**{**

**public abstract IDocument CreateDocument();**

**}**

**// Concrete Factories**

**class WordFactory : DocumentFactory**

**{**

**public override IDocument CreateDocument()**

**{**

**return new WordDocument();**

**}**

**}**

**class PdfFactory : DocumentFactory**

**{**

**public override IDocument CreateDocument()**

**{**

**return new PdfDocument();**

**}**

**}**

**class ExcelFactory : DocumentFactory**

**{**

**public override IDocument CreateDocument()**

**{**

**return new ExcelDocument();**

**}**

**}**

**// Test Class**

**class FactoryTest**

**{**

**static void Main(string[] args)**

**{**

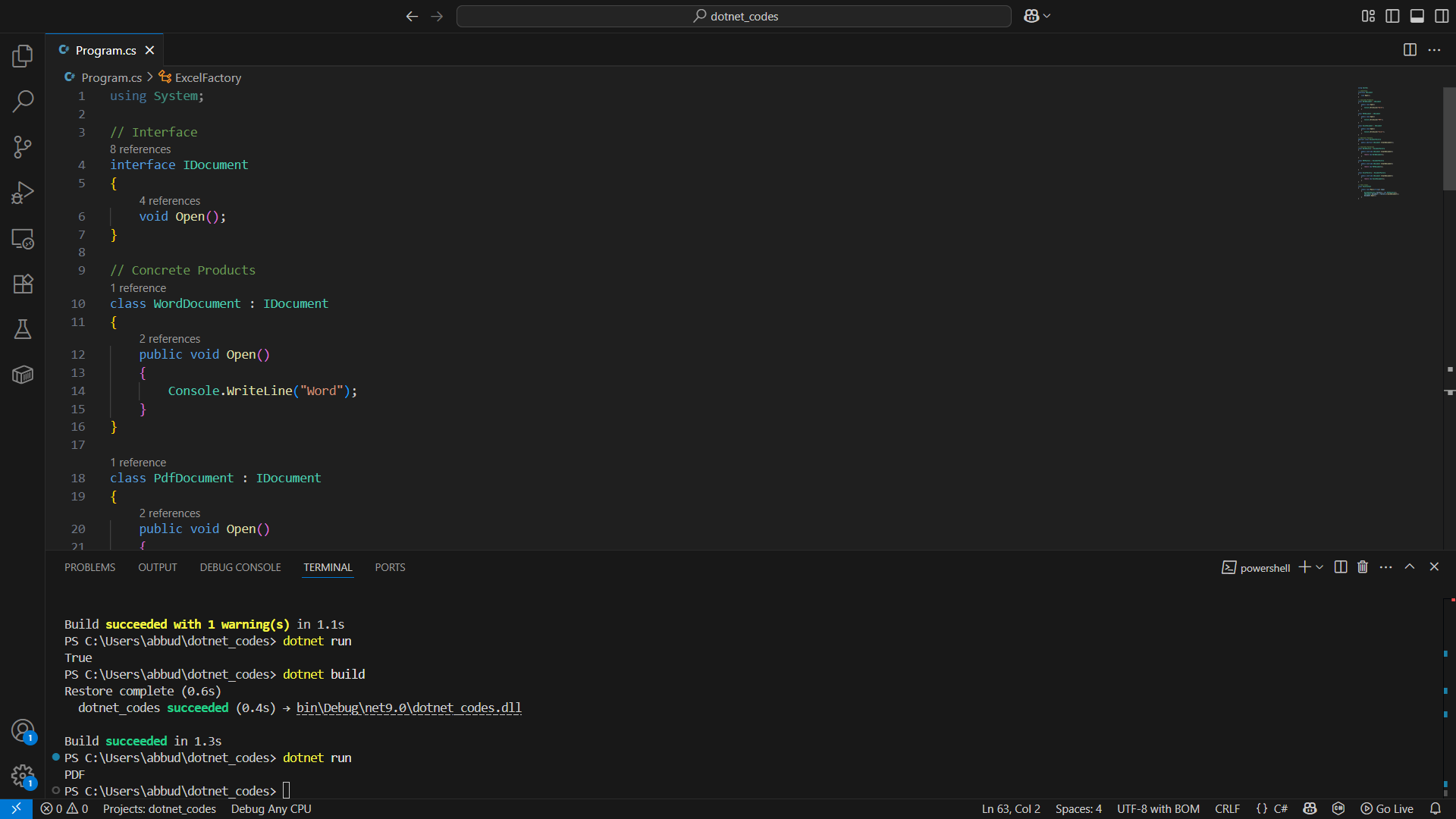
**DocumentFactory factory = new PdfFactory();**

**IDocument document = factory.CreateDocument();**

**document.Open();**

**}**

**}**

**Output:  
 **

**Exercise 3: Builder** [**Pattern.cs**](http://pattern.cs)

**Code:**

**using System;**

**class Computer**

**{**

**private string cpu, ram, storage;**

**private Computer(Builder builder)**

**{**

**this.cpu = builder.cpu;**

**this.ram = builder.ram;**

**this.storage = builder.storage;**

**}**

**public override string ToString()**

**{**

**return $"{cpu} {ram} {storage}";**

**}**

**public class Builder**

**{**

**public string cpu, ram, storage;**

**public Builder SetCpu(string cpu)**

**{**

**this.cpu = cpu;**

**return this;**

**}**

**public Builder SetRam(string ram)**

**{**

**this.ram = ram;**

**return this;**

**}**

**public Builder SetStorage(string storage)**

**{**

**this.storage = storage;**

**return this;**

**}**

**public Computer Build()**

**{**

**return new Computer(this);**

**}**

**}**

**}**

**class BuilderTest**

**{**

**static void Main(string[] args)**

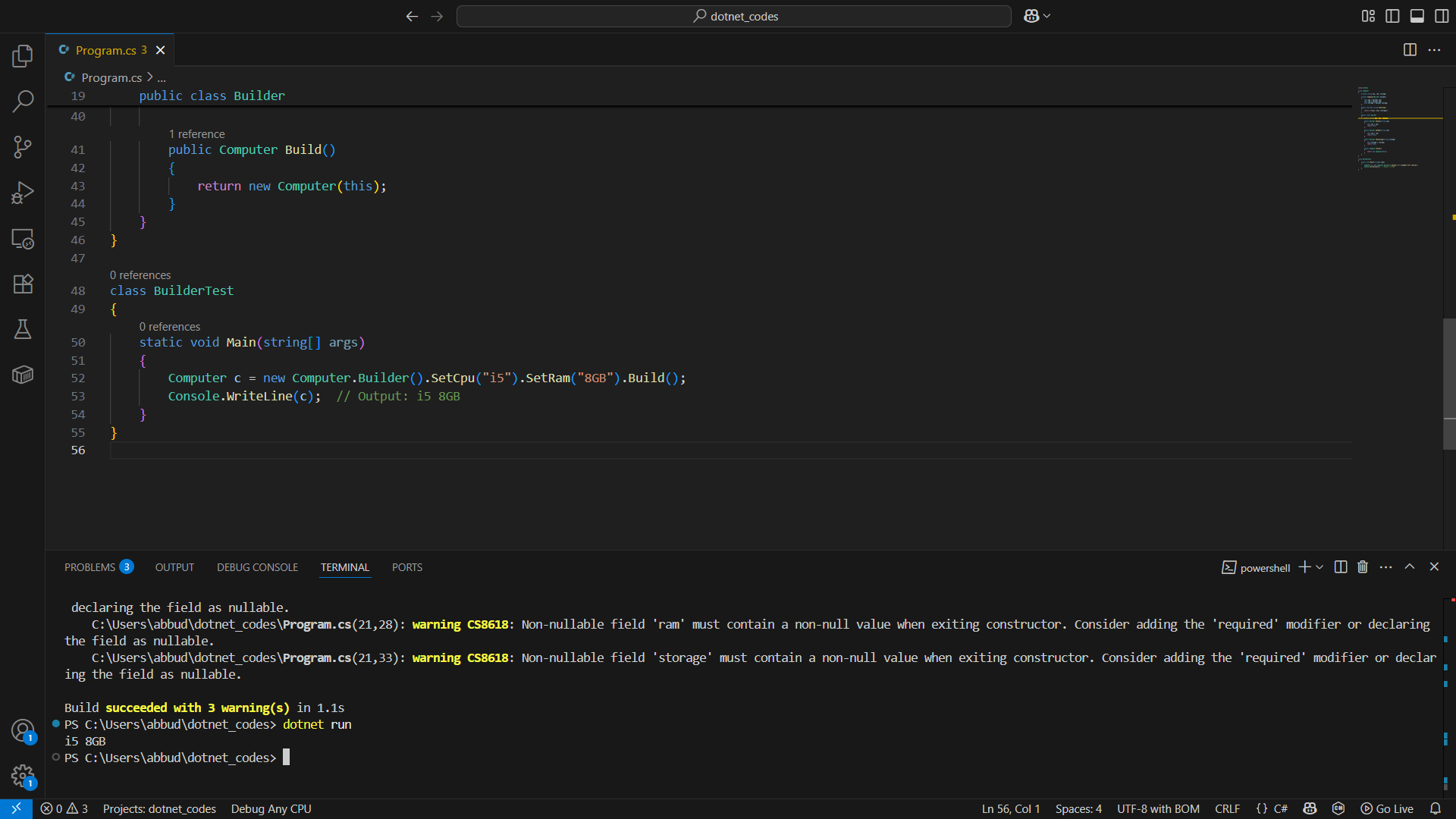
**{**

**Computer c = new Computer.Builder().SetCpu("i5").SetRam("8GB").Build();**

**Console.WriteLine(c); // Output: i5 8GB**

**}**

**}**

**Output:  
 **

**Exercise 4 : Adapter** [**Pattern**](http://pattern.cs)

**Code:**

**using System;**

**// Target interface**

**interface IPaymentProcessor**

**{**

**void Process();**

**}**

**// Adaptee 1**

**class PayPal**

**{**

**public void Make()**

**{**

**Console.WriteLine("PayPal");**

**}**

**}**

**// Adaptee 2**

**class Stripe**

**{**

**public void Execute()**

**{**

**Console.WriteLine("Stripe");**

**}**

**}**

**// Adapter 1**

**class PayPalAdapter : IPaymentProcessor**

**{**

**private PayPal p = new PayPal();**

**public void Process()**

**{**

**p.Make();**

**}**

**}**

**// Adapter 2**

**class StripeAdapter : IPaymentProcessor**

**{**

**private Stripe s = new Stripe();**

**public void Process()**

**{**

**s.Execute();**

**}**

**}**

**// Client**

**class AdapterTest**

**{**

**static void Main(string[] args)**

**{**

**IPaymentProcessor p = new PayPalAdapter();**

**p.Process(); // Output: PayPal**

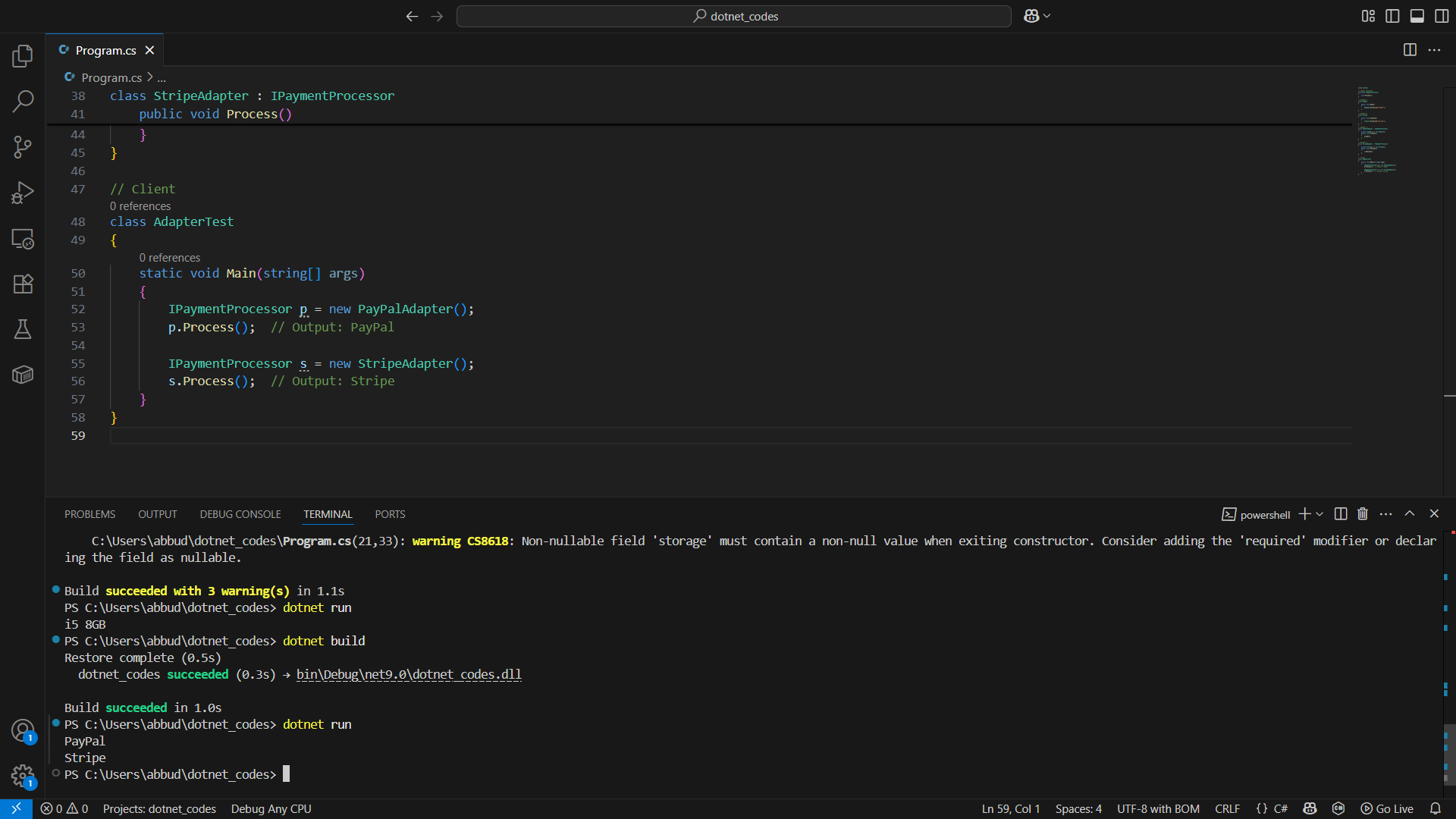
**IPaymentProcessor s = new StripeAdapter();**

**s.Process(); // Output: Stripe**

**}**

**}**

**Output:**

****

**Exercise 5: Decorator Pattern**

**Code:**

**using System;**

**// Component interface**

**interface INotifier**

**{**

**void Send();**

**}**

**// Concrete component**

**class EmailNotifier : INotifier**

**{**

**public void Send()**

**{**

**Console.WriteLine("Email");**

**}**

**}**

**// Base Decorator**

**abstract class NotifierDecorator : INotifier**

**{**

**protected INotifier notifier;**

**public NotifierDecorator(INotifier notifier)**

**{**

**this.notifier = notifier;**

**}**

**public virtual void Send()**

**{**

**notifier.Send();**

**}**

**}**

**// Concrete Decorator**

**class SMSNotifier : NotifierDecorator**

**{**

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

**public override void Send()**

**{**

**base.Send();**

**Console.WriteLine("SMS");**

**}**

**}**

**// Test class**

**class DecoratorTest**

**{**

**static void Main(string[] args)**

**{**

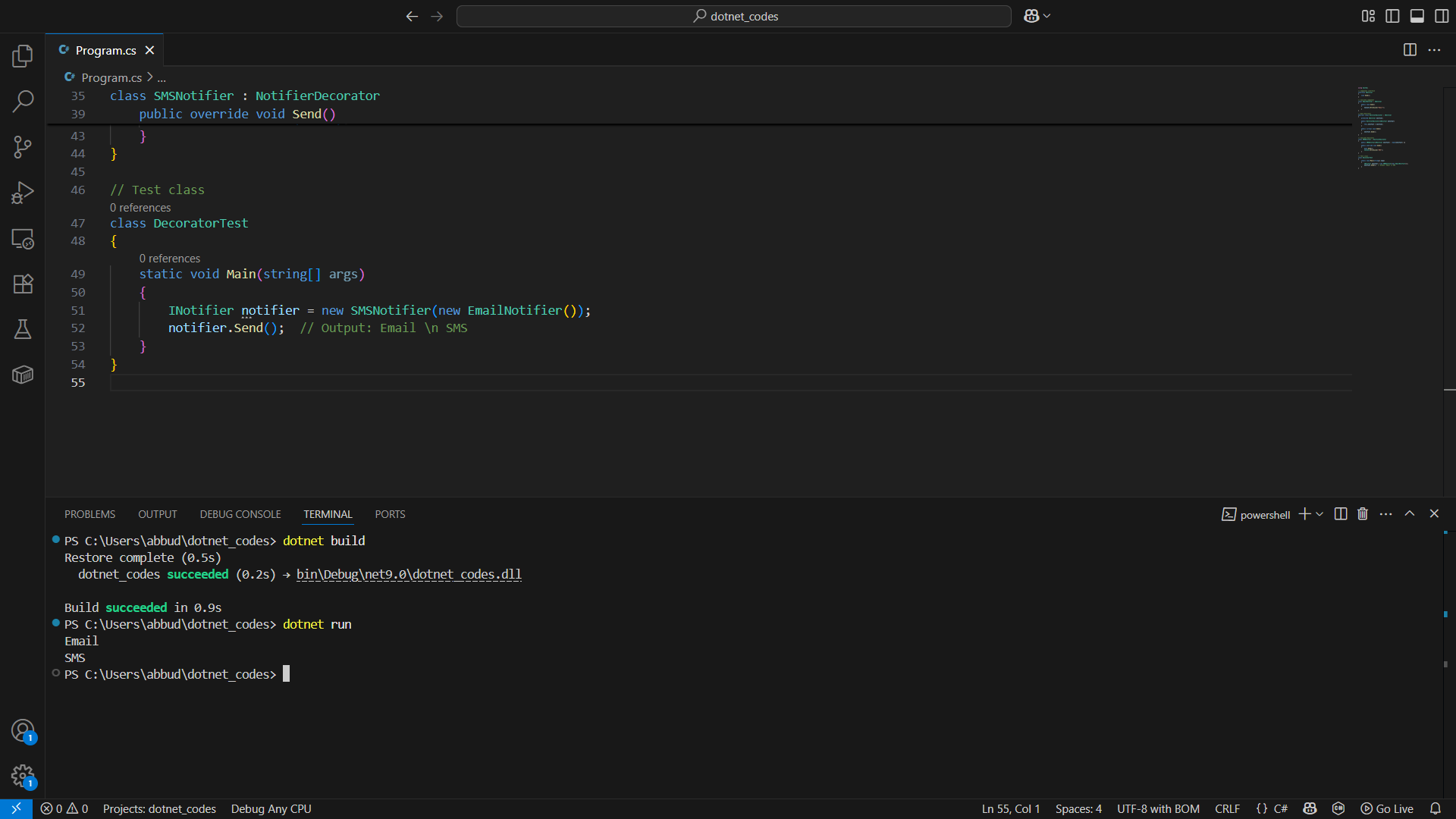
**INotifier notifier = new SMSNotifier(new EmailNotifier());**

**notifier.Send(); // Output: Email \n SMS**

**}**

**}**

**Output:**

****

**Exercise 6 Proxy Pattern**

**Code:**

**using System;**

**// Subject interface**

**interface IImage**

**{**

**void Display();**

**}**

**// Real subject**

**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);**

**}**

**}**

**// Proxy**

**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();**

**}**

**}**

**// Test class**

**class ProxyTest**

**{**

**static void Main(string[] args)**

**{**

**IImage img = new ProxyImage("test.jpg");**

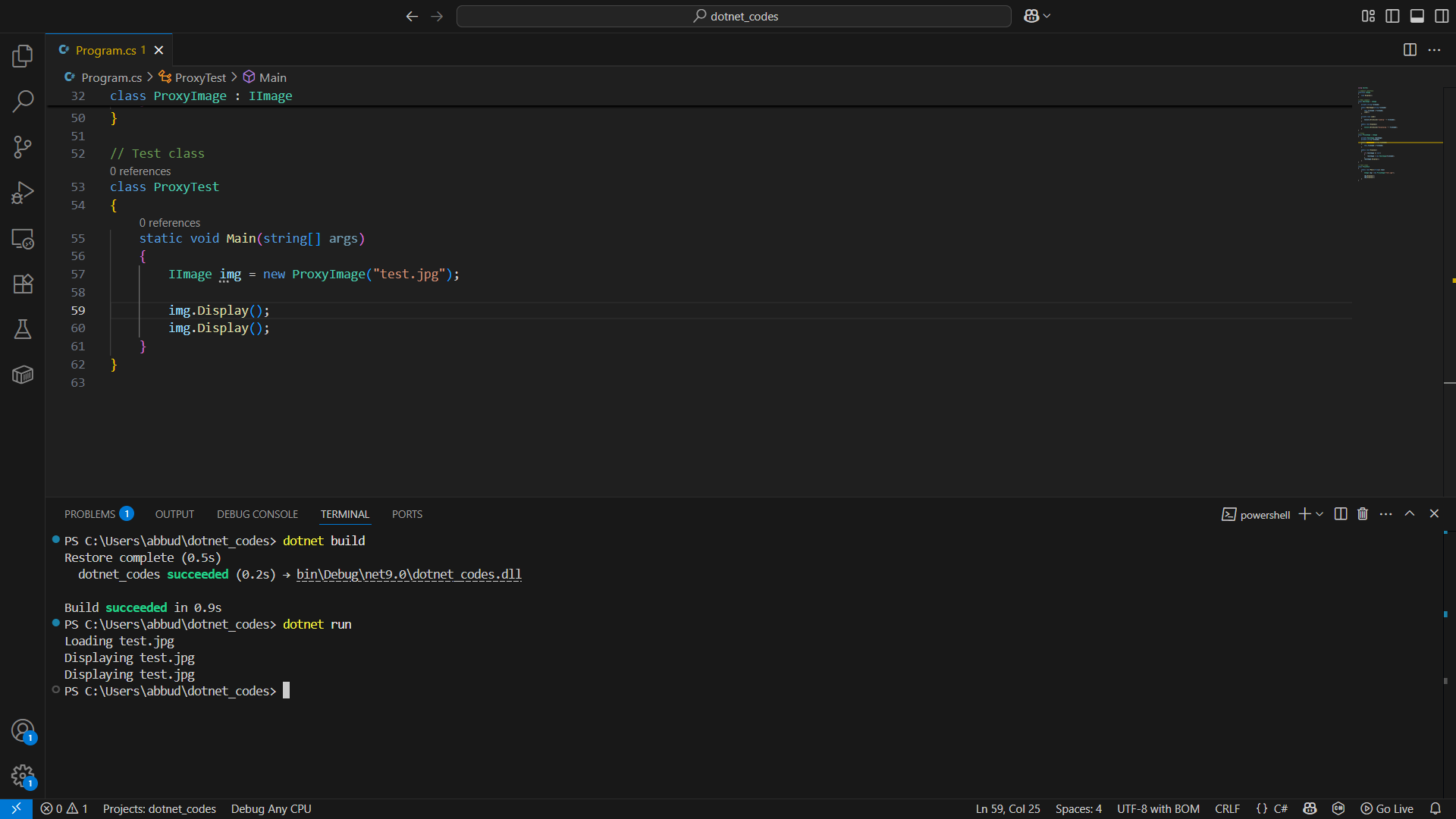
**img.Display();**

**img.Display();**

**}**

**}**

**Output:**

****

**Exercise 7 Observer Pattern**

**Code:**

**using System;**

**using System.Collections.Generic;**

**// Observer interface**

**interface IObserver**

**{**

**void Update(string stock);**

**}**

**// Subject interface**

**interface IStock**

**{**

**void Register(IObserver o);**

**void Deregister(IObserver o);**

**void NotifyObservers();**

**}**

**// Concrete Subject**

**class StockMarket : IStock**

**{**

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

**private string stock;**

**public void Register(IObserver o)**

**{**

**observers.Add(o);**

**}**

**public void Deregister(IObserver o)**

**{**

**observers.Remove(o);**

**}**

**public void SetStock(string stock)**

**{**

**this.stock = stock;**

**NotifyObservers();**

**}**

**public void NotifyObservers()**

**{**

**foreach (IObserver o in observers)**

**{**

**o.Update(stock);**

**}**

**}**

**}**

**// Concrete Observers**

**class MobileApp : IObserver**

**{**

**public void Update(string stock)**

**{**

**Console.WriteLine("Mobile: " + stock);**

**}**

**}**

**class WebApp : IObserver**

**{**

**public void Update(string stock)**

**{**

**Console.WriteLine("Web: " + stock);**

**}**

**}**

**// Test class**

**class ObserverTest**

**{**

**static void Main(string[] args)**

**{**

**StockMarket market = new StockMarket();**

**IObserver mob = new MobileApp();**

**IObserver web = new WebApp();**

**market.Register(mob);**

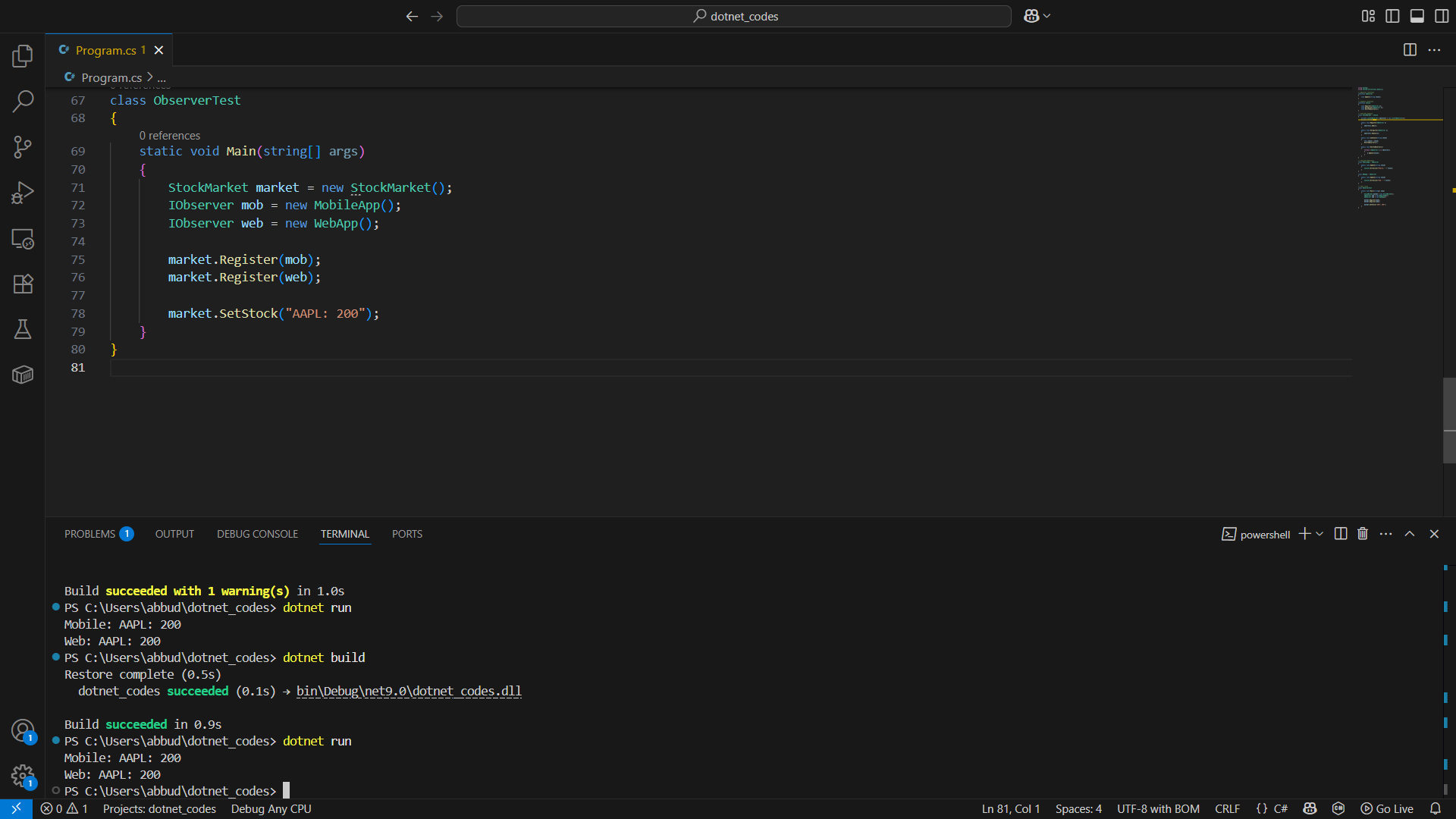
**market.Register(web);**

**market.SetStock("AAPL: 200");**

**}**

**}**

**Output:**

****

**Exercise 8 Strategy Pattern**

**Code:**

**using System;**

**// Strategy interface**

**interface IPaymentStrategy**

**{**

**void Pay();**

**}**

**// Concrete strategies**

**class CreditCardPayment : IPaymentStrategy**

**{**

**public void Pay()**

**{**

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

**}**

**}**

**class PayPalPayment : IPaymentStrategy**

**{**

**public void Pay()**

**{**

**Console.WriteLine("PayPal");**

**}**

**}**

**// Context**

**class PaymentContext**

**{**

**private IPaymentStrategy strategy;**

**public void SetStrategy(IPaymentStrategy strategy)**

**{**

**this.strategy = strategy;**

**}**

**public void Pay()**

**{**

**strategy.Pay();**

**}**

**}**

**// Test class**

**class StrategyTest**

**{**

**static void Main(string[] args)**

**{**

**PaymentContext ctx = new PaymentContext();**

**ctx.SetStrategy(new CreditCardPayment());**

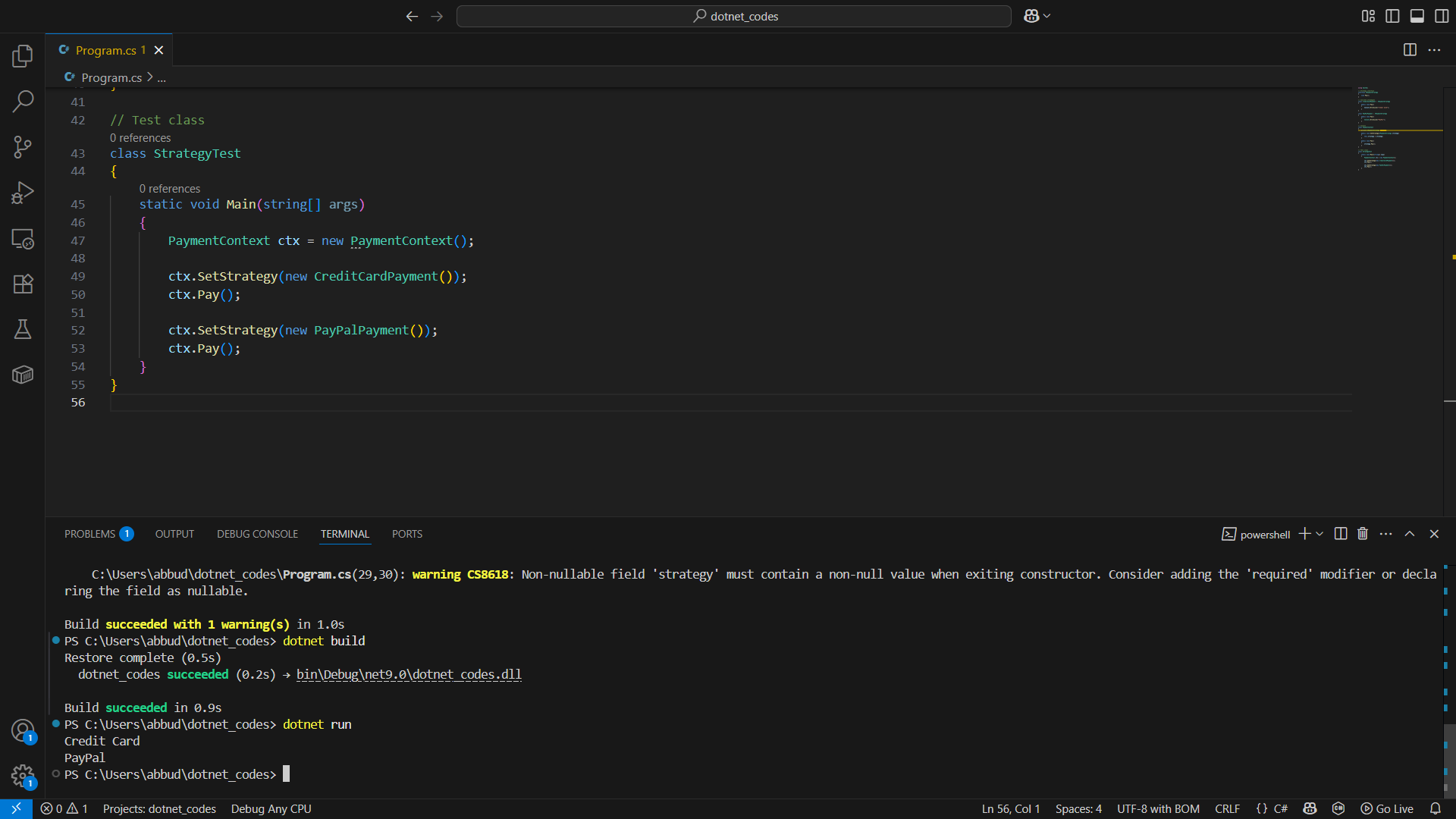
**ctx.Pay();**

**ctx.SetStrategy(new PayPalPayment());**

**ctx.Pay();**

**}**

**}**

**Output:  
  
**

**Exercise 9 Command Pattern**

**Code:**

**using System;**

**// Command interface**

**interface ICommand**

**{**

**void Execute();**

**}**

**// Receiver**

**class Light**

**{**

**public void On()**

**{**

**Console.WriteLine("Light On");**

**}**

**public void Off()**

**{**

**Console.WriteLine("Light Off");**

**}**

**}**

**// Concrete Commands**

**class LightOnCommand : ICommand**

**{**

**private Light light;**

**public LightOnCommand(Light light)**

**{**

**this.light = light;**

**}**

**public void Execute()**

**{**

**light.On();**

**}**

**}**

**class LightOffCommand : ICommand**

**{**

**private Light light;**

**public LightOffCommand(Light light)**

**{**

**this.light = light;**

**}**

**public void Execute()**

**{**

**light.Off();**

**}**

**}**

**// Invoker**

**class RemoteControl**

**{**

**private ICommand command;**

**public void SetCommand(ICommand command)**

**{**

**this.command = command;**

**}**

**public void PressButton()**

**{**

**command.Execute();**

**}**

**}**

**// Client**

**class CommandTest**

**{**

**static void Main(string[] args)**

**{**

**Light light = new Light();**

**ICommand on = new LightOnCommand(light);**

**ICommand off = new LightOffCommand(light);**

**RemoteControl remote = new RemoteControl();**

**remote.SetCommand(on);**

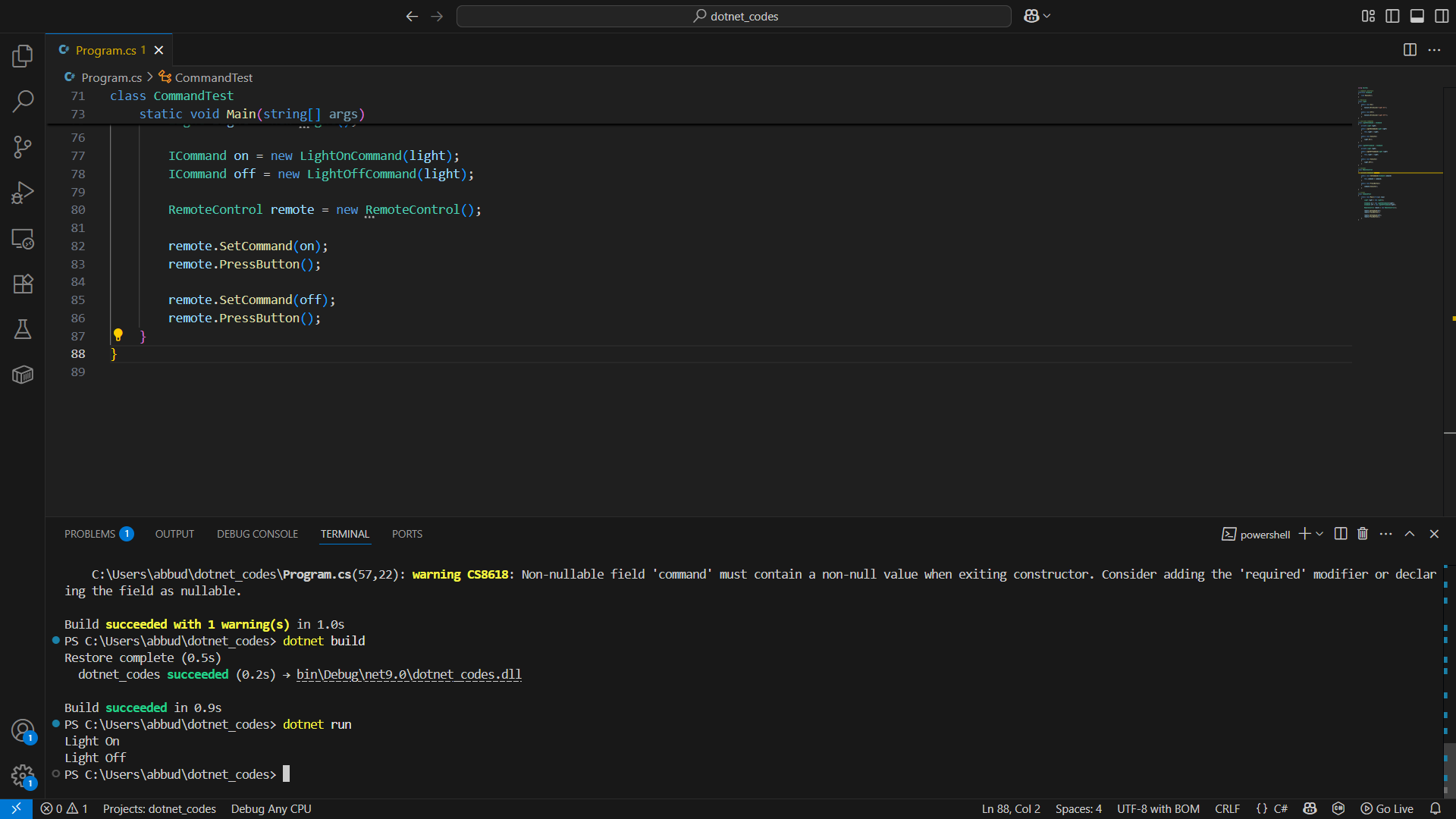
**remote.PressButton();**

**remote.SetCommand(off);**

**remote.PressButton();**

**}**

**}**

**Output:  
**

**Exercise 10 : MVC Pattern**

**Code :**

**using System;**

**// Model**

**class Student**

**{**

**public string Name { get; set; }**

**public int Id { get; set; }**

**public string Grade { get; set; }**

**public Student(string name, int id, string grade)**

**{**

**Name = name;**

**Id = id;**

**Grade = grade;**

**}**

**}**

**// View**

**class StudentView**

**{**

**public void Display(Student student)**

**{**

**Console.WriteLine($"{student.Name} {student.Id} {student.Grade}");**

**}**

**}**

**// Controller**

**class StudentController**

**{**

**private Student student;**

**private StudentView view;**

**public StudentController(Student student, StudentView view)**

**{**

**this.student = student;**

**this.view = view;**

**}**

**public void UpdateView()**

**{**

**view.Display(student);**

**}**

**}**

**// Entry Point**

**class MVCTest**

**{**

**static void Main(string[] args)**

**{**

**Student s = new Student("John", 1, "A");**

**StudentView v = new StudentView();**

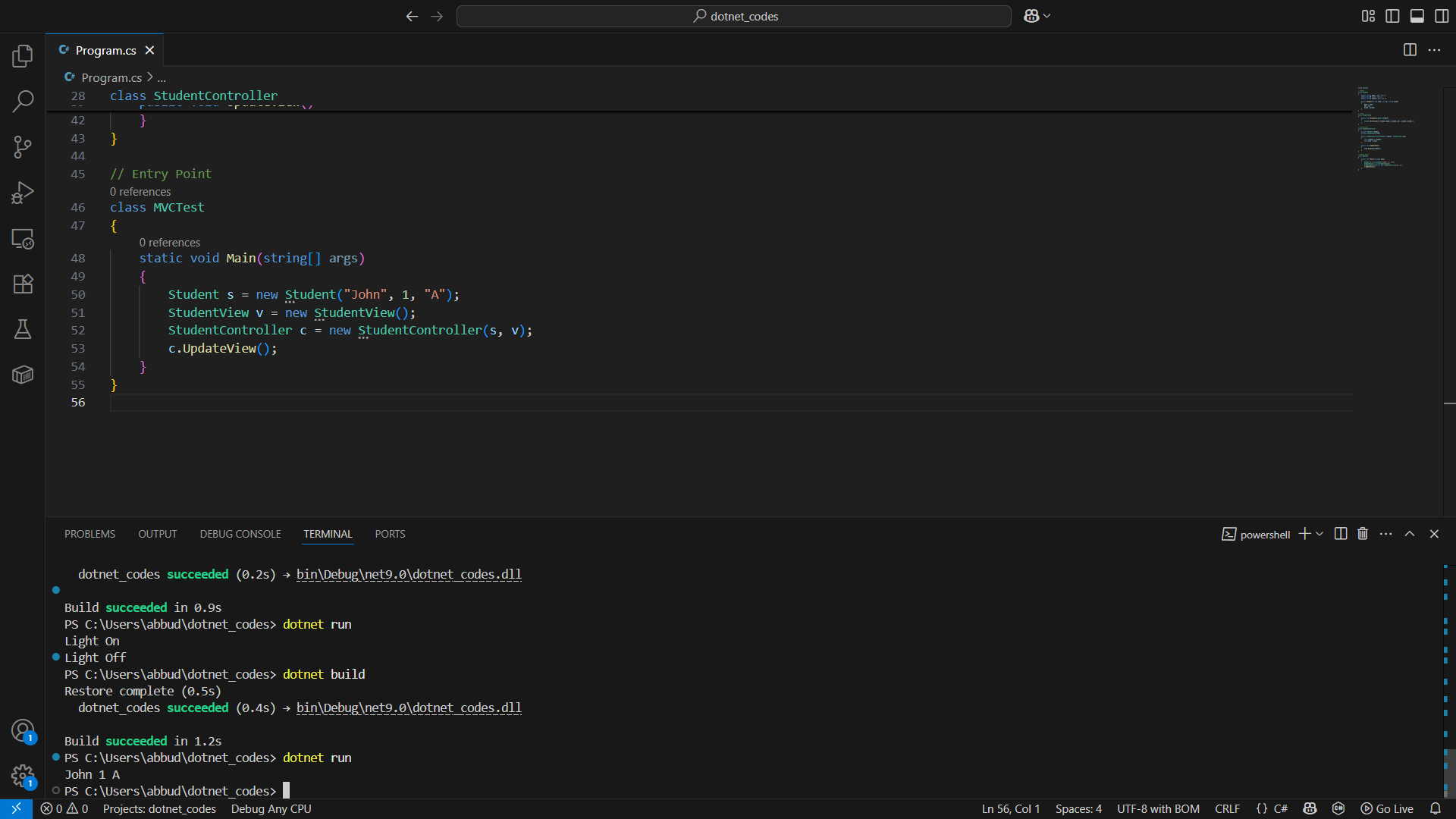
**StudentController c = new StudentController(s, v);**

**c.UpdateView();**

**}**

**}**

**Output:**

****

**Exercise 11 Dependency Injection**

**Code:**

**using System;**

**// Interface**

**public interface ICustomerRepository**

**{**

**string FindCustomerById(int id);**

**}**

**// Concrete repository implementation**

**class CustomerRepositoryImpl : ICustomerRepository**

**{**

**public string FindCustomerById(int id)**

**{**

**if (id <= 0)**

**{**

**throw new ArgumentException("Invalid customer ID: " + id);**

**}**

**return $"Customer {id}";**

**}**

**}**

**// Service class using constructor injection**

**class CustomerService**

**{**

**private readonly ICustomerRepository repo;**

**public CustomerService(ICustomerRepository repo)**

**{**

**this.repo = repo;**

**}**

**public void Find(int id)**

**{**

**Console.WriteLine(repo.FindCustomerById(id));**

**}**

**}**

**// Main class to test DI**

**class DIExample**

**{**

**static void Main(string[] args)**

**{**

**try**

**{**

**ICustomerRepository repo = new CustomerRepositoryImpl();**

**CustomerService service = new CustomerService(repo);**

**service.Find(101); // Output: Customer 101**

**service.Find(-1); // Will throw exception**

**}**

**catch (ArgumentException e)**

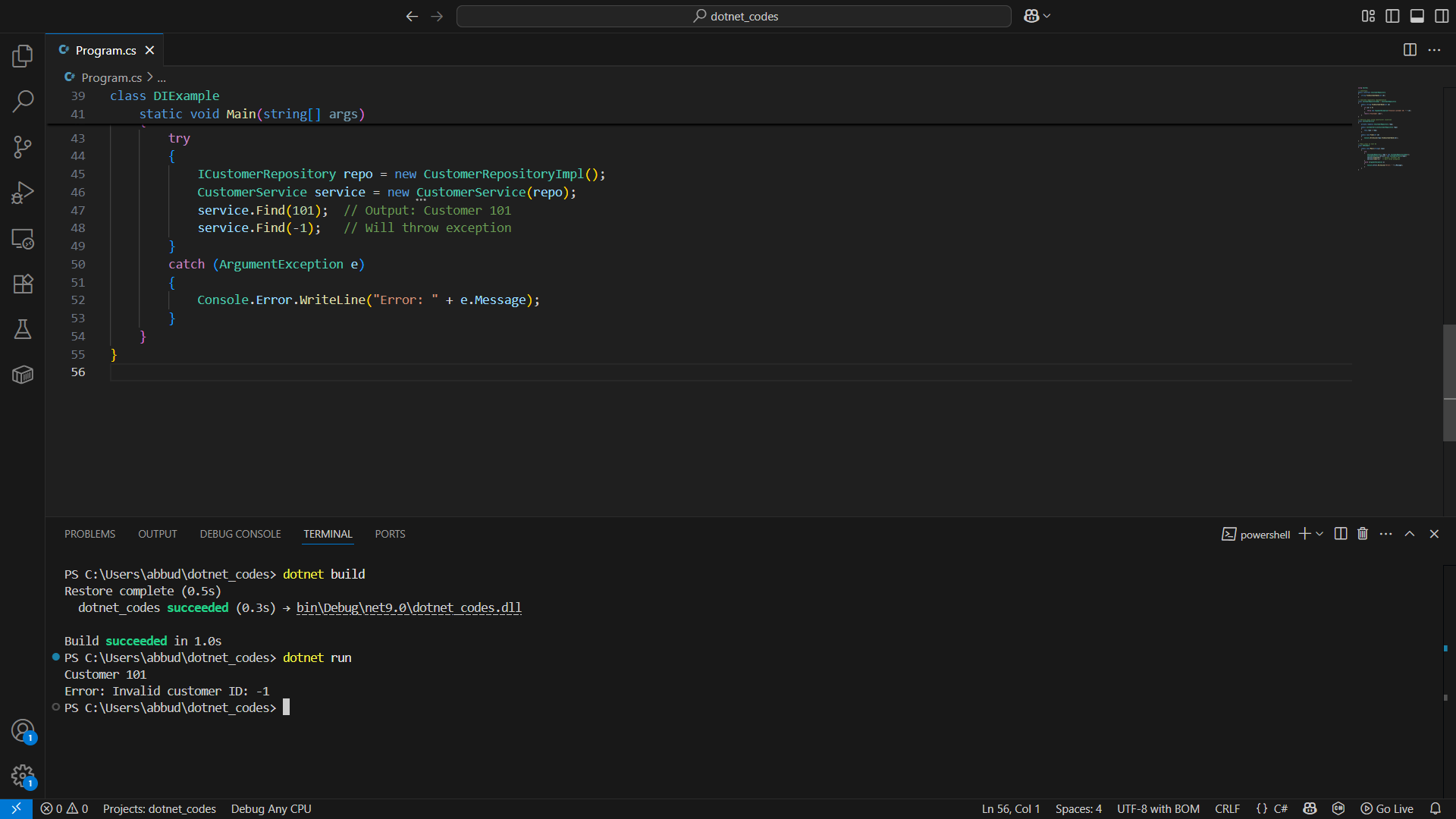
**{**

**Console.Error.WriteLine("Error: " + e.Message);**

**}**

**}**

**}**

**Output:  
**