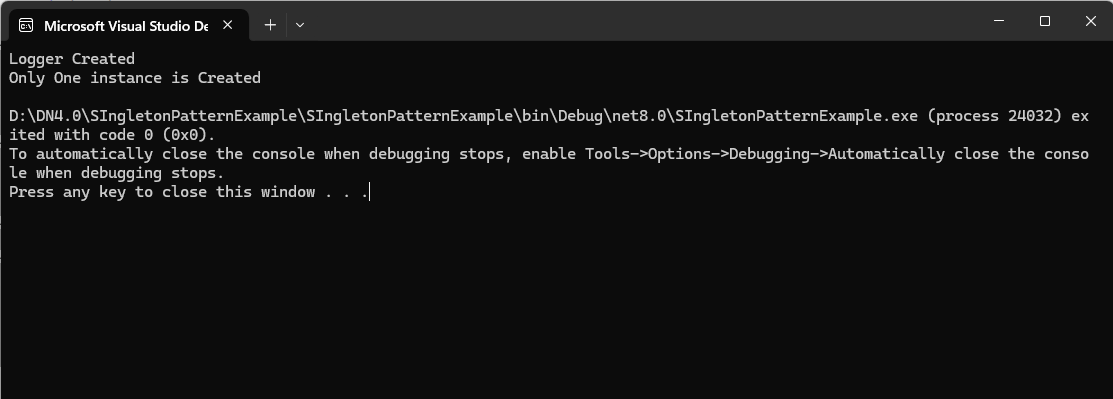
**Exercise 1: Implementing the Singleton Pattern**

|  |
| --- |
| using System;  public class Logger  {  private static Logger obj;  private Logger()  {  Console.WriteLine("Logger Created");  }  public static Logger GetInstance()  {  if (obj == null)  {  obj = new Logger();  }  return obj;  }  }  class SingletonPatternExample  {  static void Main(string[] args)  {  var user1 = Logger.GetInstance();  var user2 = Logger.GetInstance();  if (user1 == user2)  {  Console.WriteLine("Only One instance is Created");  }  }  } |



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

|  |
| --- |
| using System;  interface IDocumentTypes  {  void Spec();  }  class WordDocument : IDocumentTypes  {  public void Spec()  {  Console.WriteLine("Word Document Created");  }  }  class ExcelDocument : IDocumentTypes  {  public void Spec()  {  Console.WriteLine("Excel Document Created");  }  }  class PDFDocument : IDocumentTypes  {  public void Spec()  {  Console.WriteLine("PDF Document Created");  }  }  class DocumentFactory  {  public IDocumentTypes CreateDocument(string str)  {  return str switch  {  "Word" => new WordDocument(),  "Pdf" => new PDFDocument(),  \_ => new ExcelDocument()  };  }  }  class FactoryMethodPatternExample  {  static void Main(string[] args)  {  var factory = new DocumentFactory();  factory.CreateDocument("Pdf").Spec();  factory.CreateDocument("Word").Spec();  factory.CreateDocument("Excel").Spec();  }  } |

A screenshot of a computer

AI-generated content may be incorrect.

**Exercise 3: Implementing the Builder Pattern**

|  |
| --- |
| using System;  class Computer  {  public string CPU { get; }  public int Ram { get; }  public int Storage { get; }  private Computer(Builder builder)  {  CPU = builder.CPU;  Ram = builder.Ram;  Storage = builder.Storage;  }  public class Builder  {  public string CPU;  public int Ram;  public int Storage;  public Builder SetCPU(string cpu)  {  this.CPU = cpu;  return this;  }  public Builder SetRam(int ram)  {  this.Ram = ram;  return this;  }  public Builder SetStorage(int storage)  {  this.Storage = storage;  return this;  }  public Computer Build()  {  return new Computer(this);  }  }  }  class BuilderPatternExample  {  static void Main(string[] args)  {  var myPc = new Computer.Builder()  .SetCPU("Ryzen 7 7600X")  .SetRam(16)  .SetStorage(1024)  .Build();  Console.WriteLine("CPU: " + myPc.CPU);  Console.WriteLine("RAM: " + myPc.Ram);  Console.WriteLine("Storage: " + myPc.Storage + " GB");  }  } |

A screenshot of a computer

AI-generated content may be incorrect.

**Exercise 4: Implementing the Adapter Pattern**

|  |
| --- |
| using System;  interface IPaymentProcessor  {  void MakePayment(double amt);  }  class IciciGateway  {  public void Pay(double amt)  {  Console.WriteLine("Paid " + amt + " using Icici Gateway.");  }  }  class PaytmGateway  {  public void Payment(double amt)  {  Console.WriteLine("Paid " + amt + " using Paytm.");  }  }  class IciciTranslator : IPaymentProcessor  {  private readonly IciciGateway icici;  public IciciTranslator(IciciGateway icici)  {  this.icici = icici;  }  public void MakePayment(double amt)  {  icici.Pay(amt);  }  }  class PaytmTranslator : IPaymentProcessor  {  private readonly PaytmGateway paytm;  public PaytmTranslator(PaytmGateway paytm)  {  this.paytm = paytm;  }  public void MakePayment(double amt)  {  paytm.Payment(amt);  }  }  // Client code  class AdapterPatternExample  {  static void Main(string[] args)  {  IPaymentProcessor icici = new IciciTranslator(new IciciGateway());  icici.MakePayment(2000);  IPaymentProcessor paytm = new PaytmTranslator(new PaytmGateway());  paytm.MakePayment(200);  }  } |

A screenshot of a computer program

AI-generated content may be incorrect.

**Exercise 5: Implementing the Decorator Pattern**

|  |
| --- |
| using System;  interface INotifier  {  void Send();  }  class EmailNotifier : INotifier  {  public void Send()  {  Console.WriteLine("Email has been sent to your registered mail ID.");  }  }  class NotifierDecorator : INotifier  {  protected readonly INotifier notifier;  public NotifierDecorator(INotifier notifier)  {  this.notifier = notifier;  }  public virtual void Send()  {  notifier.Send();  }  }  class SMSNotifierDecorator : NotifierDecorator  {  public SMSNotifierDecorator(INotifier notifier) : base(notifier) { }  public override void Send()  {  base.Send();  Console.WriteLine("SMS has been sent.");  }  }  class SlackNotifierDecorator : NotifierDecorator  {  public SlackNotifierDecorator(INotifier notifier) : base(notifier) { }  public override void Send()  {  base.Send();  Console.WriteLine("Slack has been sent.");  }  }  class DecoratorPatternExample  {  static void Main(string[] args)  {  INotifier notifier = new EmailNotifier();  notifier = new SMSNotifierDecorator(notifier);  notifier = new SlackNotifierDecorator(notifier);  notifier.Send(); // Should print all three  }  } |

A screenshot of a computer program

AI-generated content may be incorrect.

**Exercise 7: Implementing the Observer Pattern**

|  |
| --- |
| using System;  using System.Collections.Generic;  interface IStock  {  void RegisterObserver(IObserver o);  void DeregisterObserver(IObserver o);  void NotifyObservers();  }  class StockMarket : IStock  {  private readonly List<IObserver> observers = new List<IObserver>();  private double stockPrice;  public void RegisterObserver(IObserver o)  {  observers.Add(o);  }  public void DeregisterObserver(IObserver o)  {  observers.Remove(o);  }  public void NotifyObservers()  {  foreach (var observer in observers)  {  observer.Update(stockPrice);  }  }  public void SetStockPrice(double price)  {  this.stockPrice = price;  NotifyObservers();  }  }  interface IObserver  {  void Update(double price);  }  // Concrete Observer: Mobile  class MobileApp : IObserver  {  private readonly string appName;  public MobileApp(string name)  {  appName = name;  }  public void Update(double stockPrice)  {  Console.WriteLine($"{appName} - Mobile App received stock price update: {stockPrice}");  }  }  class WebApp : IObserver  {  private readonly string appName;  public WebApp(string name)  {  appName = name;  }  public void Update(double stockPrice)  {  Console.WriteLine($"{appName} - Web App received stock price update: {stockPrice}");  }  }  class ObserverPatternExample  {  static void Main(string[] args)  {  StockMarket stockMarket = new StockMarket();  IObserver mobileObserver = new MobileApp("Zerodha");  IObserver webObserver = new WebApp("Growww");  stockMarket.RegisterObserver(webObserver);  stockMarket.RegisterObserver(mobileObserver);  stockMarket.SetStockPrice(10000);  }  } |

A screenshot of a computer program

AI-generated content may be incorrect.

**Exercise 8: Implementing the Strategy Pattern**

|  |
| --- |
| using System;  interface IPaymentStrategy  {  void Pay(int amt);  }  class CreditCardPayment : IPaymentStrategy  {  public void Pay(int amt)  {  Console.WriteLine($"The Amount of Rupees {amt} has been deducted from your Credit Card.");  }  }  class PayPalPayment : IPaymentStrategy  {  public void Pay(int amt)  {  Console.WriteLine($"The Amount of Rupees {amt} has been deducted from your PayPal Account.");  }  }  class PaymentContext  {  private IPaymentStrategy strategy;  public void SetPaymentStrategy(IPaymentStrategy strategy)  {  this.strategy = strategy;  }  public void Pay(int amt)  {  if (strategy != null)  strategy.Pay(amt);  else  Console.WriteLine("No payment strategy selected.");  }  }  class StrategyPatternExample  {  static void Main(string[] args)  {  PaymentContext context = new PaymentContext();  context.SetPaymentStrategy(new CreditCardPayment());  context.Pay(1500);  context.SetPaymentStrategy(new PayPalPayment());  context.Pay(20000);  }  } |

A screenshot of a computer

AI-generated content may be incorrect.