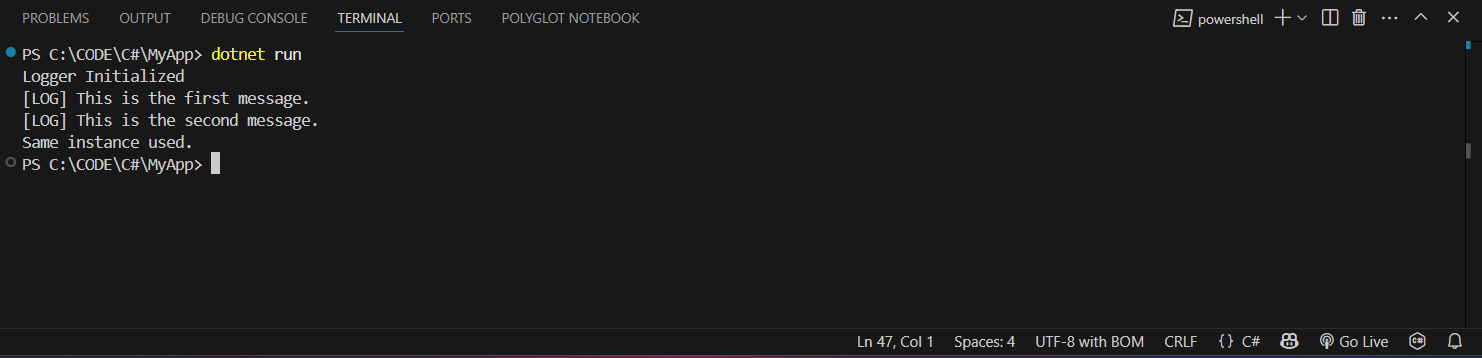
**Week 1: Design Patterns and Principles**

**Question 1 : Exercise:1 Implementing the Singleton Pattern**

**CODE :**

using System;  
public class Logger  
{  
 private static Logger \_instance;  
 private static readonly object \_lock = new object();  
 private Logger()  
 {  
 Console.WriteLine("Logger Initialized");  
 }  
 public static Logger GetInstance()  
 {  
 if (\_instance == null)  
 {  
 lock (\_lock)  
 {  
 if (\_instance == null)  
 {  
 \_instance = new Logger();  
 }  
 }  
 }  
 return \_instance;  
 }  
 public void Log(string message)  
 {  
 Console.WriteLine($"[LOG] {message}");  
 }  
}  
class Program  
{  
 static void Main(string[] args)  
 {  
 Logger logger1 = Logger.GetInstance();  
 Logger logger2 = Logger.GetInstance();  
 logger1.Log("This is the first message.");  
 logger2.Log("This is the second message.");  
 Console.WriteLine(object.ReferenceEquals(logger1, logger2)  
 ? "Same instance used." : "Different instances!");  
 }  
}

**OUTPUT :**



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

**CODE :**

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");  
 }  
}

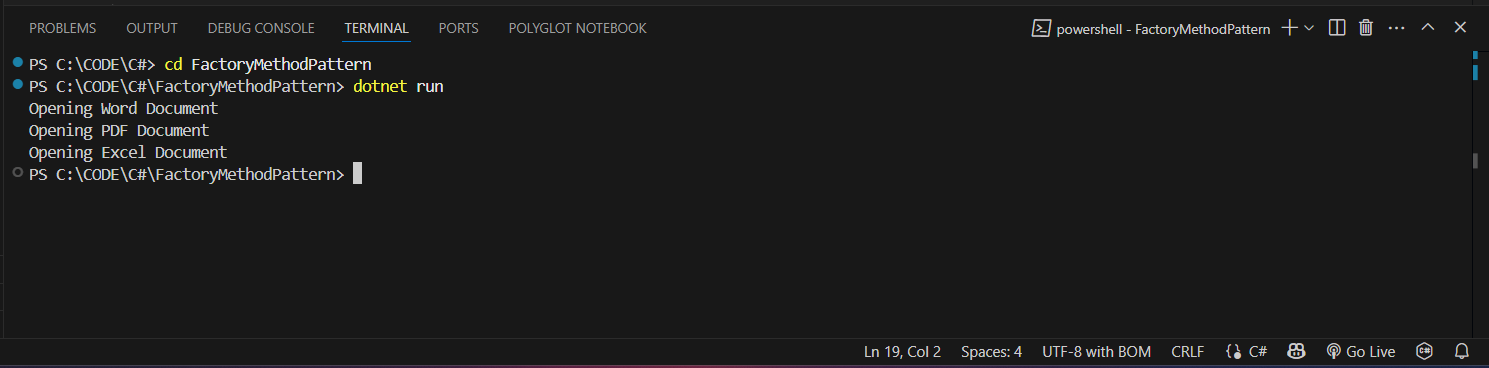
public abstract class DocumentFactory  
{  
 public abstract IDocument CreateDocument();  
}

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

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

public class ExcelFactory : DocumentFactory  
{  
 public override IDocument CreateDocument()  
 {  
 return new ExcelDocument();  
 }  
}  
class Program  
{  
 static void Main(string[] args)  
 {  
 DocumentFactory wordFactory = new WordFactory();  
 IDocument wordDoc = wordFactory.CreateDocument();  
 wordDoc.Open();  
 DocumentFactory pdfFactory = new PdfFactory();  
 IDocument pdfDoc = pdfFactory.CreateDocument();  
 pdfDoc.Open();  
 DocumentFactory excelFactory = new ExcelFactory();  
 IDocument excelDoc = excelFactory.CreateDocument();  
 excelDoc.Open();  
 }  
}

**OUTPUT :**

****

**Algorithms and Data Structures**

**Question 3: Exercise: 2 E-commerce Platform Search Function**

**CODE :**

using System;

class Product

{

public int Id;

public string Name;

public string Category;

public Product(int id, string name, string category)

{

Id = id;

Name = name;

Category = category;

}

}

class Program

{

static Product LinearSearch(Product[] products, int searchId)

{

foreach (var p in products)

if (p.Id == searchId)

return p;

return null;

}

static Product BinarySearch(Product[] products, int searchId)

{

int left = 0, right = products.Length - 1;

while (left <= right)

{

int mid = (left + right) / 2;

if (products[mid].Id == searchId)

return products[mid];

if (products[mid].Id < searchId)

left = mid + 1;

else

right = mid - 1;

}

return null;

}

static void Main()

{

Product[] products = {

new Product(3, "Mouse", "Electronics"),

new Product(1, "Laptop", "Electronics"),

new Product(2, "Shoes", "Fashion")

};

Console.WriteLine("Linear Search for ID = 2:");

var result1 = LinearSearch(products, 2);

if (result1 != null)

Console.WriteLine($"{result1.Id} - {result1.Name} ({result1.Category})");

else

Console.WriteLine("Product not found.");

Array.Sort(products, (a, b) => a.Id.CompareTo(b.Id));

Console.WriteLine("\nBinary Search for ID = 1:");

var result2 = BinarySearch(products, 1);

if (result2 != null)

Console.WriteLine($"{result2.Id} - {result2.Name} ({result2.Category})");

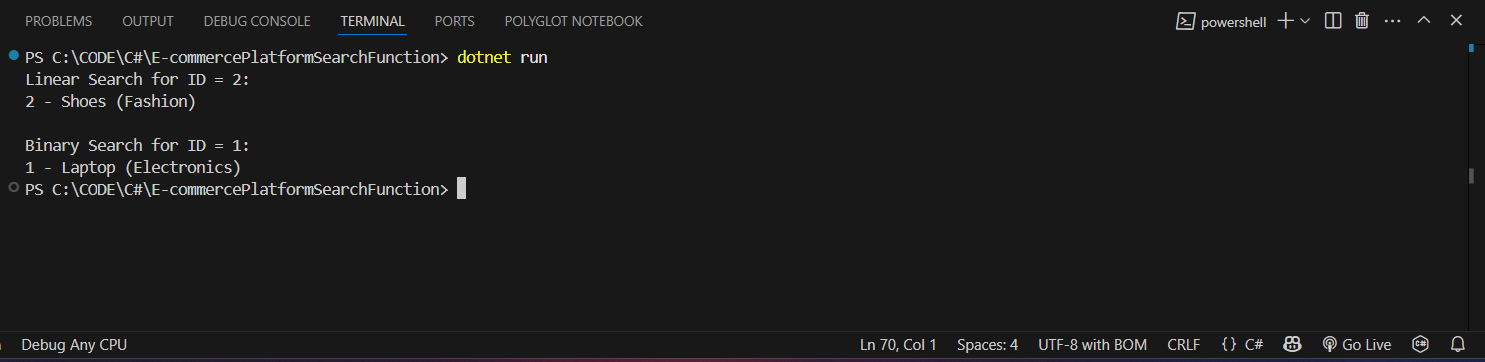
else

Console.WriteLine("Product not found.");

}

}

**OUTPUT :**



**Question 4: Exercise: 7 Financial Forecasting**

**CODE :**

using System;

class Program

{

static double Predict(int year, double initial, double growth){

if (year == 0) return initial;

return Predict(year - 1, initial, growth) \* (1 + growth);

}

static void Main()

{

double initial = 1000;

double rate = 0.10;

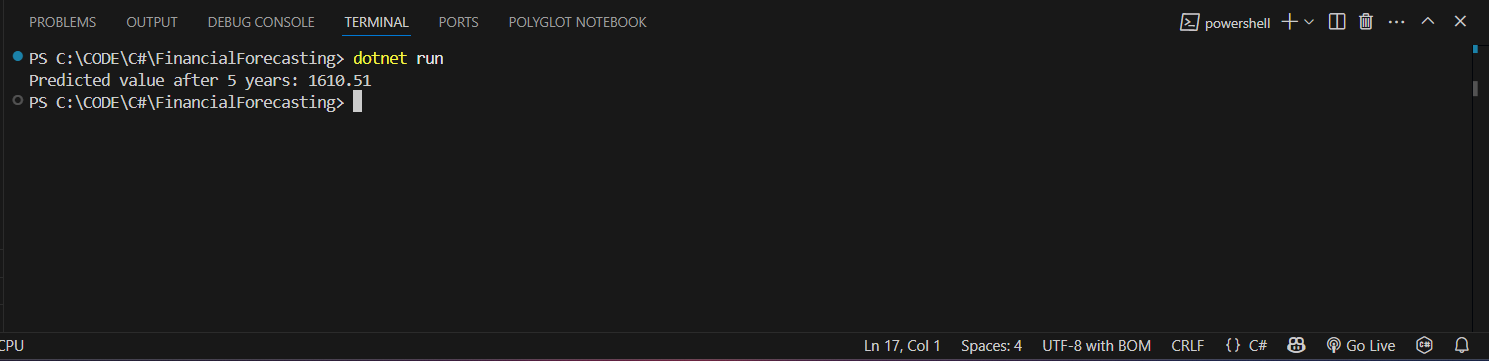
int years = 5;

double futureValue = Predict(years, initial, rate);

Console.WriteLine($"Predicted value after {years} years: {futureValue:F2}");

}}

**OUTPUT :**

****