# Week 1 - Module 1: Design Patterns and Principles

## Exercise 1: Singleton Pattern – Logger Class

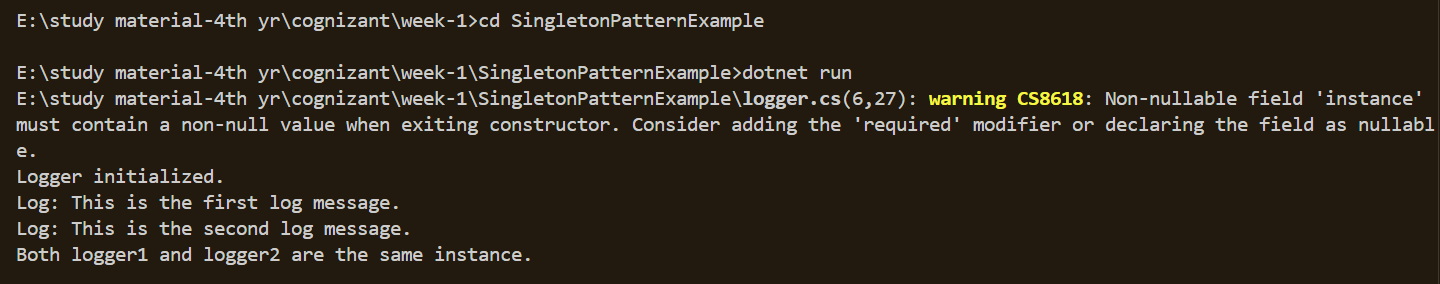
Logger.cs

public class Logger  
{  
 private static Logger instance;  
 private Logger()  
 {  
 Console.WriteLine("Logger instance created.");  
 }  
  
 public static Logger GetInstance()  
 {  
 if (instance == null)  
 {  
 instance = new Logger();  
 }  
 return instance;  
 }  
  
 public void Log(string message)  
 {  
 Console.WriteLine("Log message: " + message);  
 }  
}

Main.cs

class Program  
{  
 static void Main(string[] args)  
 {  
 Logger logger1 = Logger.GetInstance();  
 logger1.Log("First log");  
  
 Logger logger2 = Logger.GetInstance();  
 logger2.Log("Second log");  
  
 Console.WriteLine("Are both instances the same? " + (logger1 == logger2));  
 }  
}

**OUTPUT:**



## Exercise 2: Factory Method Pattern – Document Creation

IDocument.cs

public interface IDocument  
{  
 void Open();  
}

WordDocument.cs

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

PdfDocument.cs

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

ExcelDocument.cs

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

DocumentFactory.cs

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

WordDocumentFactory.cs

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

PdfDocumentFactory.cs

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

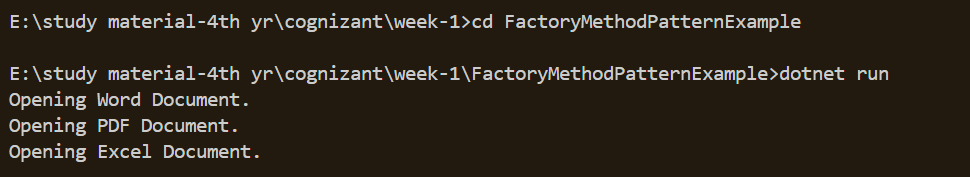
ExcelDocumentFactory.cs

public class ExcelDocumentFactory : DocumentFactory  
{  
 public override IDocument CreateDocument()  
 {  
 return new ExcelDocument();  
 }  
}

Main.cs

class Program  
{  
 static void Main(string[] args)  
 {  
 DocumentFactory wordFactory = new WordDocumentFactory();  
 IDocument wordDoc = wordFactory.CreateDocument();  
 wordDoc.Open();  
  
 DocumentFactory pdfFactory = new PdfDocumentFactory();  
 IDocument pdfDoc = pdfFactory.CreateDocument();  
 pdfDoc.Open();  
  
 DocumentFactory excelFactory = new ExcelDocumentFactory();  
 IDocument excelDoc = excelFactory.CreateDocument();  
 excelDoc.Open();  
 }  
}

**OUTPUT:**

**Week 1 - Module 2: Data Structures and Algorithms**

## Exercise 2: E-commerce Platform Search Function

using System;

using System.Linq;

public class Product

{

public int ProductId;

public string ProductName;

public string Category;

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

{

ProductId = id;

ProductName = name;

Category = category;

}

}

public class Search

{

public static Product LinearSearch(Product[] products, int id)

{

foreach (var product in products)

{

if (product.ProductId == id)

return product;

}

return null;

}

public static Product BinarySearch(Product[] products, int id)

{

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

while (left <= right)

{

int mid = (left + right) / 2;

if (products[mid].ProductId == id)

return products[mid];

else if (products[mid].ProductId < id)

left = mid + 1;

else

right = mid - 1;

}

return null;

}

}

class Program

{

static void Main(string[] args)

{

Product[] products = {

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

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

new Product(2, "Book", "Books")

};

Console.WriteLine(" Linear Search:");

var result1 = Search.LinearSearch(products, 2);

Console.WriteLine(result1 != null ? $"Found: {result1.ProductName}" : "Not Found");

Console.WriteLine("\n Binary Search:");

var sortedProducts = products.OrderBy(p => p.ProductId).ToArray();

var result2 = Search.BinarySearch(sortedProducts, 2);

Console.WriteLine(result2 != null ? $"Found: {result2.ProductName}" : "Not Found");

}

}  
  
**OUTPUT:**



## Exercise 7: Financial Forecasting

using System;

public class Finance

{

public static double PredictFutureValue(double currentValue, double growthRate, int years)

{

if (years == 0)

return currentValue;

return PredictFutureValue(currentValue \* (1 + growthRate), growthRate, years - 1);

}

}

class Program

{

static void Main(string[] args)

{

double current = 1000;

double growthRate = 0.10;

int years = 5;

double futureValue = Finance.PredictFutureValue(current, growthRate, years);

Console.WriteLine($" Future Value after {years} years: {futureValue:F2}");

}

}  
  
  
**OUTPUT:**

