# 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));  
 }  
}

## 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();  
 }  
}

# Week 1 - Module 2: Data Structures and Algorithms

## Exercise 2: E-commerce Platform Search Function

public class Product  
{  
 public int ProductId { get; set; }  
 public string ProductName { get; set; }  
 public string Category { get; set; }  
  
 public Product(int id, string name, string category)  
 {  
 ProductId = id;  
 ProductName = name;  
 Category = category;  
 }  
}  
  
public class Search  
{  
 public static Product LinearSearch(Product[] products, string name)  
 {  
 foreach (var product in products)  
 {  
 if (product.ProductName.Equals(name, StringComparison.OrdinalIgnoreCase))  
 {  
 return product;  
 }  
 }  
 return null;  
 }  
  
 public static Product BinarySearch(Product[] products, string name)  
 {  
 int left = 0, right = products.Length - 1;  
 while (left <= right)  
 {  
 int mid = (left + right) / 2;  
 int cmp = string.Compare(products[mid].ProductName, name, true);  
 if (cmp == 0) return products[mid];  
 if (cmp < 0) left = mid + 1;  
 else right = mid - 1;  
 }  
 return null;  
 }  
}

## Exercise 7: Financial Forecasting

public class Forecast  
{  
 public static double FutureValueRecursive(double initial, double rate, int years)  
 {  
 if (years == 0) return initial;  
 return (1 + rate) \* FutureValueRecursive(initial, rate, years - 1);  
 }  
  
 public static double FutureValueIterative(double initial, double rate, int years)  
 {  
 double value = initial;  
 for (int i = 0; i < years; i++)  
 {  
 value \*= (1 + rate);  
 }  
 return value;  
 }  
}