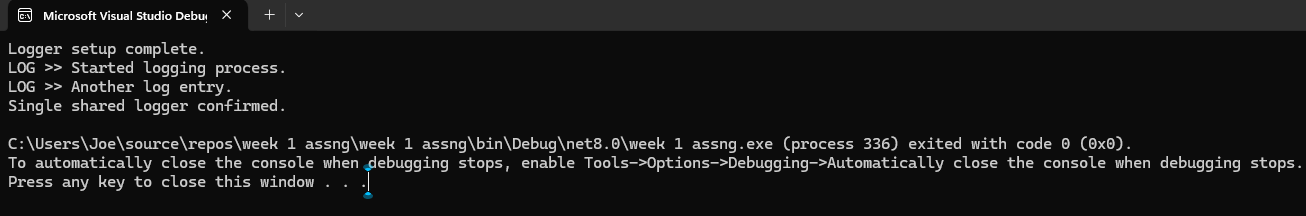
**Week 1: Design Patterns and Principles**

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

**Logger.cs**

using System;  
  
 public class Logger  
 {  
 private static Logger \_onlyInstance;  
 private Logger()  
 {  
 Console.WriteLine("Logger setup complete.");  
 }  
  
 public static Logger GetLogger()  
 {  
 if (\_onlyInstance == null)  
 {  
 \_onlyInstance = new Logger();  
 }  
 return \_onlyInstance;  
 }  
  
 public void WriteLog(string message)  
 {  
 Console.WriteLine("LOG >> " + message);  
 }  
 }  
  
 class Program  
 {  
 static void Main(string[] args)  
 {  
 Logger logA = Logger.GetLogger();  
 Logger logB = Logger.GetLogger();  
  
 logA.WriteLog("Started logging process.");  
 logB.WriteLog("Another log entry.");  
  
 Console.WriteLine(logA == logB ? "Single shared logger confirmed." : "Different loggers created.");  
 }  
 }

**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");  
 }  
}

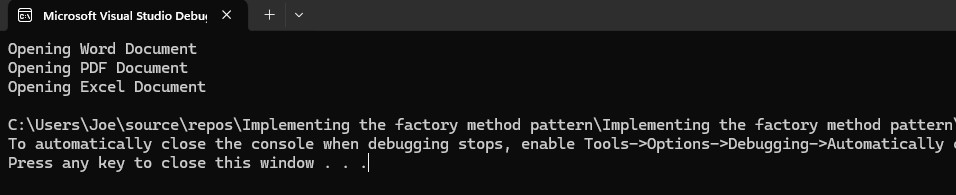
public abstract class DocumentFactory  
{  
 public abstract IDocument CreateDocument();  
}

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:**

**Program.cs:**

using System;

class Item

{

public int Id;

public string Title;

public string Group;

public Item(int id, string title, string group)

{

Id = id;

Title = title;

Group = group;

}

}

class Program

{

static Item FindByLinear(Item[] items, int id)

{

foreach (var i in items)

if (i.Id == id) return i;

return null;

}

static Item FindByBinary(Item[] items, int id)

{

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

while (left <= right)

{

int mid = (left + right) / 2;

if (items[mid].Id == id)

return items[mid];

if (items[mid].Id < id)

left = mid + 1;

else

right = mid - 1;

}

return null;

}

static void Main()

{

Item[] items = {

new Item(4, "Keyboard", "Tech"),

new Item(2, "Phone", "Tech"),

new Item(3, "Bag", "Style")

};

Console.WriteLine("Linear Search for ID = 3:");

var result1 = FindByLinear(items, 3);

if (result1 != null)

Console.WriteLine($"{result1.Id} - {result1.Title} ({result1.Group})");

else

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

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

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

var result2 = FindByBinary(items, 2);

if (result2 != null)

Console.WriteLine($"{result2.Id} - {result2.Title} ({result2.Group})");

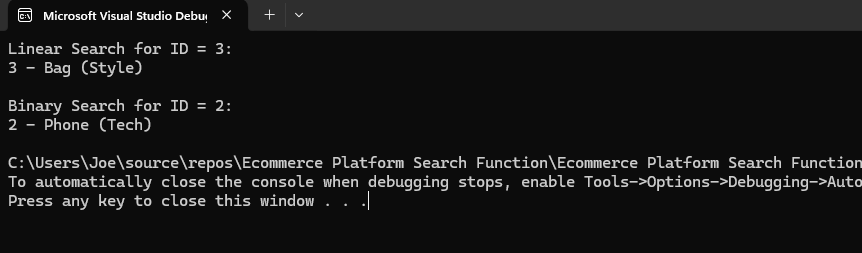
else

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

}

}

**Output:**



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

**CODE:**

**Program.cs**

using System;

class Program

{

static double Forecast(int time, double start, double rate)

{

return time == 0 ? start : Forecast(time - 1, start, rate) \* (1 + rate);

}

static void Main()

{

double baseValue = 1500;

double growth = 0.08;

int period = 6;

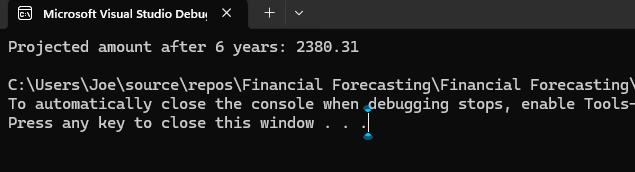
double estimate = Forecast(period, baseValue, growth);

Console.WriteLine($"Projected amount after {period} years: {estimate:F2}");

}

}

**Output:**

****