**Design principles & Patterns**

**Name: Akshita Pattnaik**

**SUPERSET ID: 6360271**

**Exercise 1: Implementing the Singleton Pattern**

**Scenario:**

You need to ensure that a logging utility class in your application has only one instance throughout the application lifecycle to ensure consistent logging.

**Steps:**

1. **Create a New C# Project:**
   * Create a new C# project named **SingletonPatternExample**.
2. **Define a Singleton Class:**
   * Create a class named Logger that has a private static instance of itself.
   * Ensure the constructor of Logger is private.
   * Provide a public static method to get the instance of the Logger class.
3. **Implement the Singleton Pattern:**
   * Write code to ensure that the Logger class follows the Singleton design pattern.
4. **Test the Singleton Implementation:**
   * Create a test class to verify that only one instance of Logger is created and used across the application.

**CODE:**

**Program.cs:**

using System;

namespace SingletonPatternExample

{

internal class Program

{

static void Main(string[] args)

{

Logger logger1 = Logger.GetInstance();

logger1.Log("Logging first message...");

Logger logger2 = Logger.GetInstance();

logger2.Log("Logging second message...");

if (object.ReferenceEquals(logger1, logger2))

{

Console.WriteLine("✅ Both logger instances are the same (Singleton works!)");

}

else

{

Console.WriteLine("❌ Logger instances are different (Singleton failed!)");

}

}

}

}

**Logger.cs:**

using System;

namespace SingletonPatternExample

{

public class Logger

{

private static Logger \_instance;

private static readonly object \_lock = new object();

private Logger()

{

Console.WriteLine("Logger instance created.");

}

public static Logger GetInstance()

{

if (\_instance == null)

{

lock (\_lock)

{

if (\_instance == null)

{

\_instance = new Logger();

}

}

}

return \_instance;

}

public void Log(string message)

{

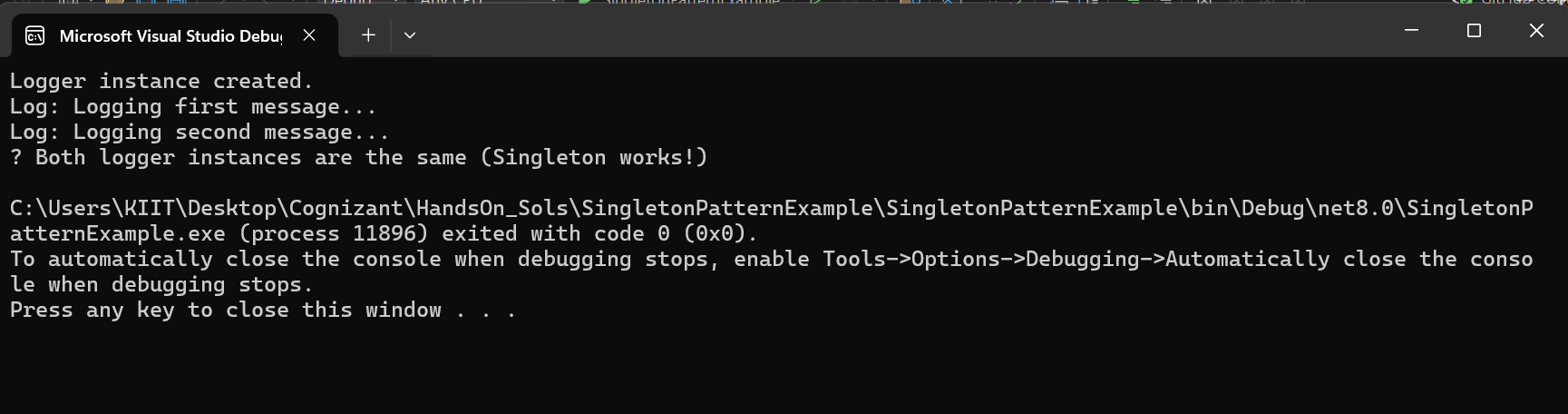
Console.WriteLine($"Log: {message}");

}

}

}

**OUTPUT:**



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

**Scenario:**

You are developing a document management system that needs to create different types of documents (e.g., Word, PDF, Excel). Use the Factory Method Pattern to achieve this.

**Steps:**

1. **Create a New Java Project:**
   * Create a new Java project named **FactoryMethodPatternExample**.
2. **Define Document Classes:**
   * Create interfaces or abstract classes for different document types such as **WordDocument**, **PdfDocument**, and **ExcelDocument**.
3. **Create Concrete Document Classes:**
   * Implement concrete classes for each document type that implements or extends the above interfaces or abstract classes.
4. **Implement the Factory Method:**
   * Create an abstract class **DocumentFactory** with a method **createDocument()**.
   * Create concrete factory classes for each document type that extends DocumentFactory and implements the **createDocument()** method.
5. **Test the Factory Method Implementation:**
   * Create a test class to demonstrate the creation of different document types using the factory method.

**CODE:**

**Program.cs:**

using System;

namespace FactoryMethodPatternExample

{

internal 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();

}

}

}

**IDocument.cs:**

namespace FactoryMethodPatternExample

{

public interface IDocument

{

void Open();

}

}

**WordDocument.cs:**

namespace FactoryMethodPatternExample

{

public class WordDocument : IDocument

{

public void Open()

{

Console.WriteLine("Opening a Word document...");

}

}

}

**ExcelDocument.cs:**

namespace FactoryMethodPatternExample

{

public class ExcelDocument : IDocument

{

public void Open()

{

Console.WriteLine("Opening an Excel document...");

}

}

}

**PdfDocument.cs:**

namespace FactoryMethodPatternExample

{

public class PdfDocument : IDocument

{

public void Open()

{

Console.WriteLine("Opening a PDF document...");

}

}

}

**DocumentFactory.cs:**

namespace FactoryMethodPatternExample

{

public abstract class DocumentFactory

{

public abstract IDocument CreateDocument();

}

}

**WordFactory.cs:**

namespace FactoryMethodPatternExample

{

public class WordFactory : DocumentFactory

{

public override IDocument CreateDocument()

{

return new WordDocument();

}

}

}

**ExcelFactory.cs:**

namespace FactoryMethodPatternExample

{

public class ExcelFactory : DocumentFactory

{

public override IDocument CreateDocument()

{

return new ExcelDocument();

}

}

}

**PdfFactory.cs:**

namespace FactoryMethodPatternExample

{

public class PdfFactory : DocumentFactory

{

public override IDocument CreateDocument()

{

return new PdfDocument();

}

}

}

**OUTPUT:**

