**Suman Sourabh**

**6363866**

**Design principles & Patterns**

**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:**

using System;

namespace SingletonPatternExample

{

// Singleton Logger Class

public class Logger

{

private static Logger \_instance;

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

// Private constructor

private Logger()

{

Console.WriteLine("Logger initialized.");

}

// Public method to get the singleton instance

public static Logger GetInstance()

{

if (\_instance == null)

{

lock (\_lock)

{

if (\_instance == null)

{

\_instance = new Logger();

}

}

}

return \_instance;

}

// Log method

public void Log(string message)

{

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

}

}

// Main Program

class Program

{

static void Main(string[] args)

{

Logger logger1 = Logger.GetInstance();

logger1.Log("First log message.");

Logger logger2 = Logger.GetInstance();

logger2.Log("Second log message.");

if (logger1 == logger2)

{

Console.WriteLine("Logger is a Singleton. Both references point to the same instance.");

}

else

{

Console.WriteLine("Logger is NOT a Singleton. Different instances found.");

}

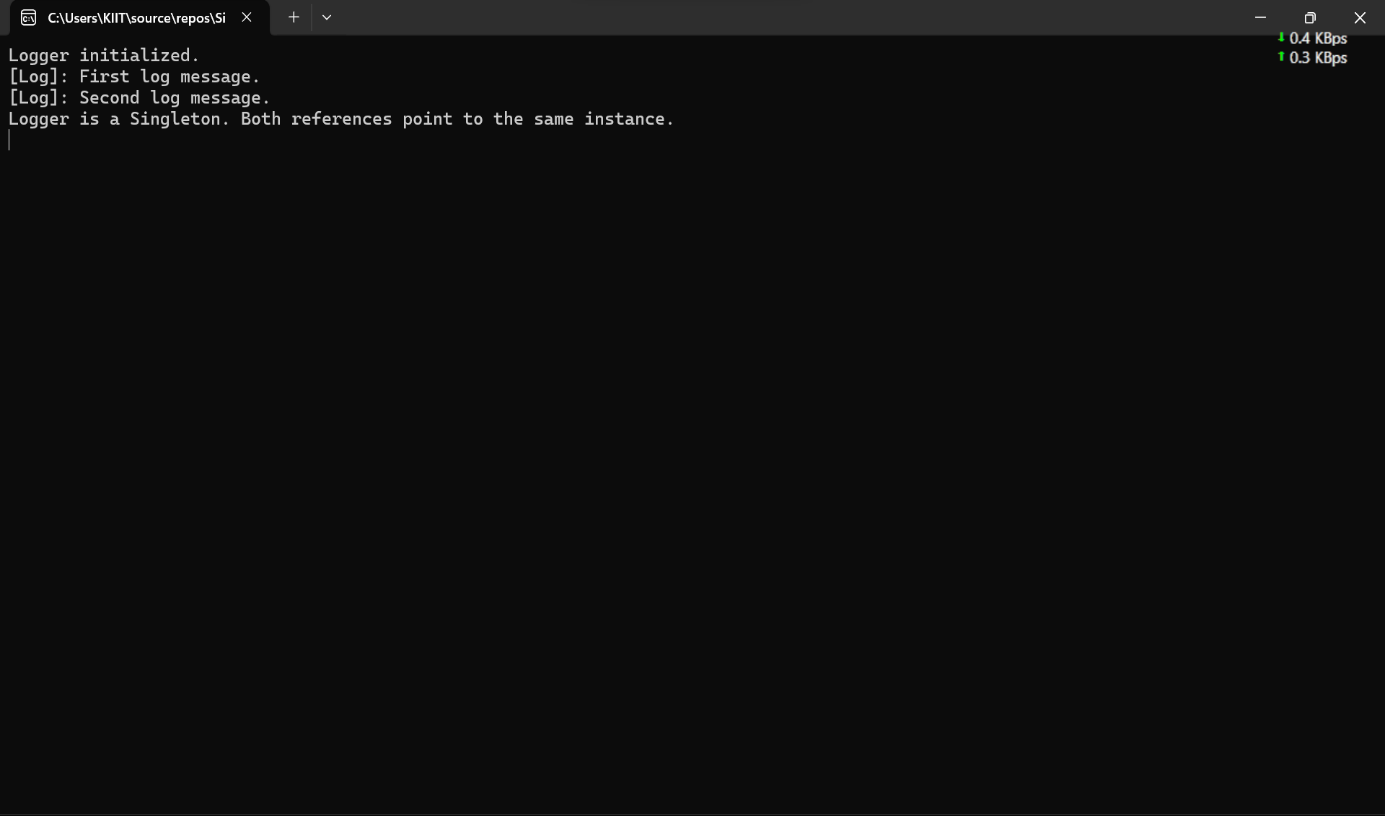
Console.ReadLine(); // Pause the console

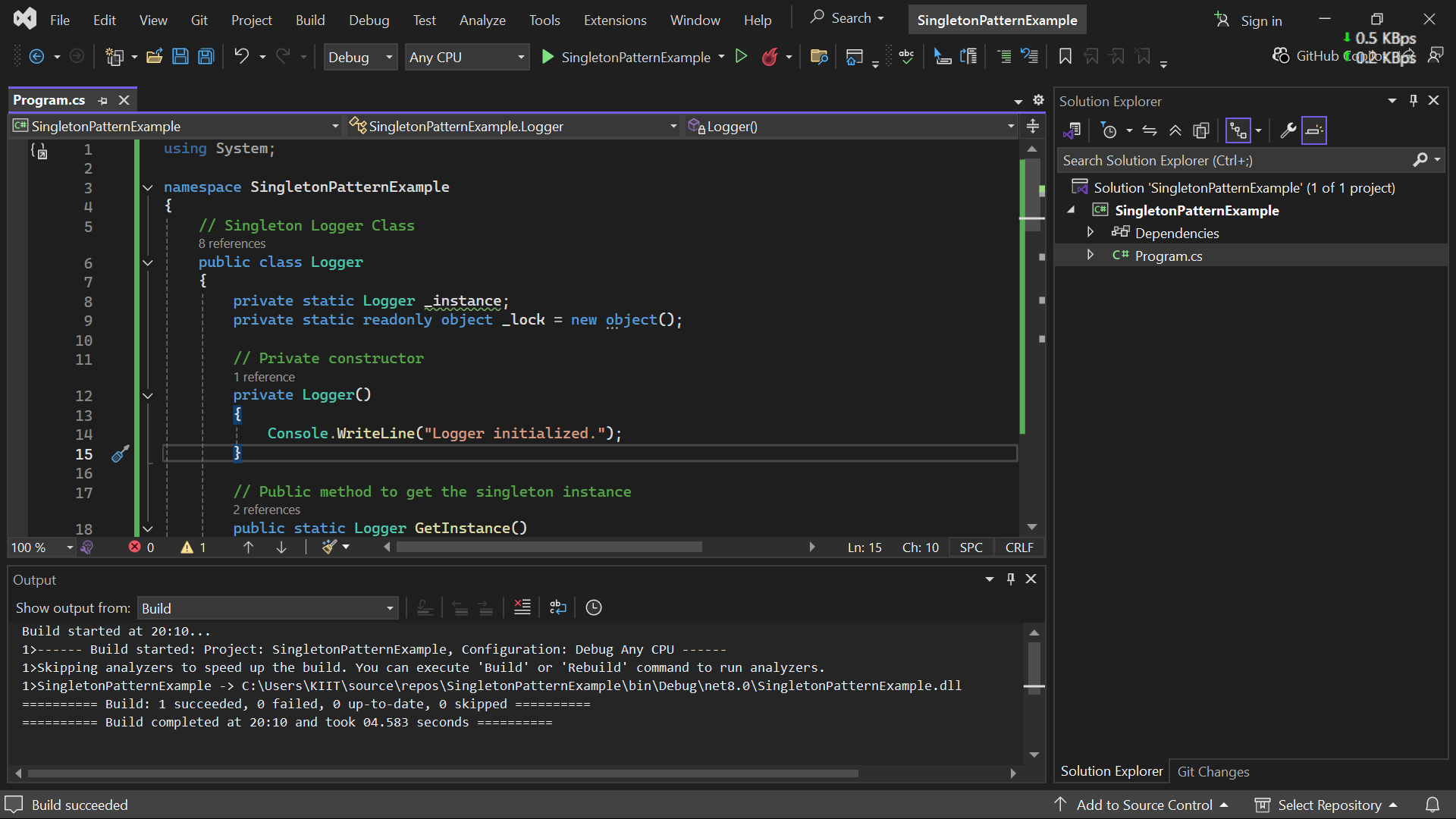
}

}

}

**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:**

using System;

namespace FactoryMethodPatternExample

{

// Step 2: Define Document Interface

public interface IDocument

{

void Open();

}

// Step 3: Create Concrete Document Classes

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.");

}

}

// Step 4: Define Abstract Factory

public abstract class DocumentFactory

{

public abstract IDocument CreateDocument();

}

// Concrete Factories

public class WordDocumentFactory : DocumentFactory

{

public override IDocument CreateDocument()

{

return new WordDocument();

}

}

public class PdfDocumentFactory : DocumentFactory

{

public override IDocument CreateDocument()

{

return new PdfDocument();

}

}

public class ExcelDocumentFactory : DocumentFactory

{

public override IDocument CreateDocument()

{

return new ExcelDocument();

}

}

// Step 5: Test the Factory Method Implementation

class Program

{

static void Main(string[] args)

{

DocumentFactory wordFactory = new WordDocumentFactory();

IDocument word = wordFactory.CreateDocument();

word.Open();

DocumentFactory pdfFactory = new PdfDocumentFactory();

IDocument pdf = pdfFactory.CreateDocument();

pdf.Open();

DocumentFactory excelFactory = new ExcelDocumentFactory();

IDocument excel = excelFactory.CreateDocument();

excel.Open();

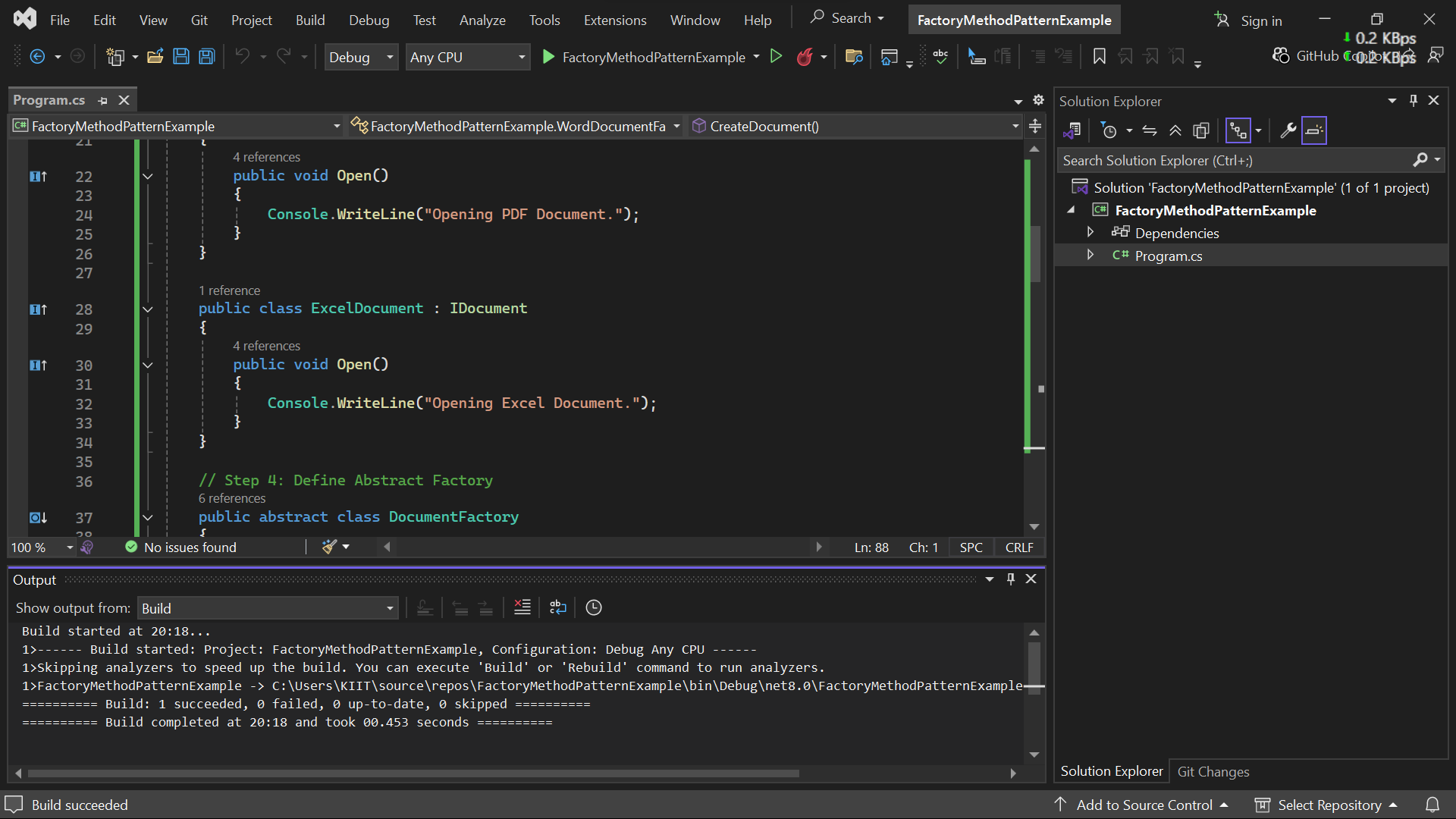
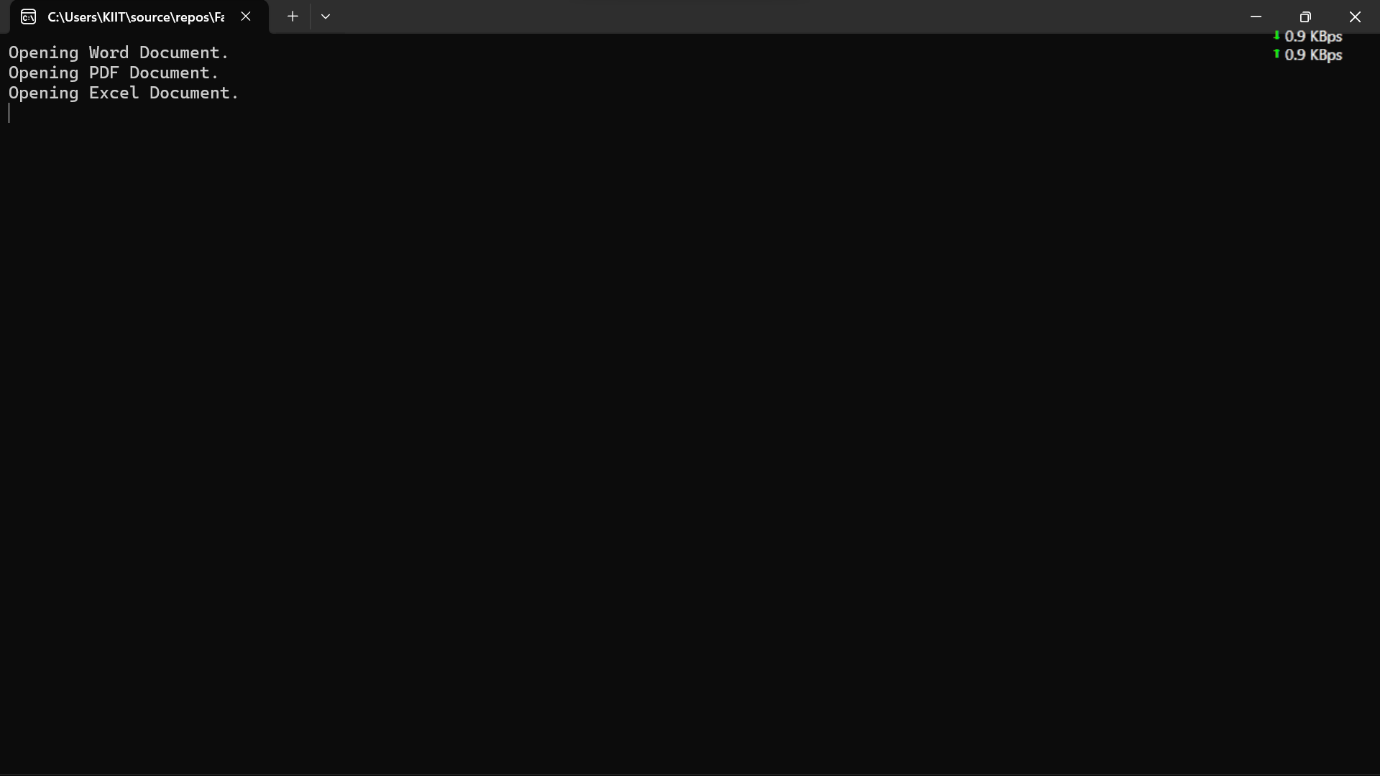
Console.ReadLine(); // Keep console open

}

}

}

**OUTPUT:**

****