**Exercise 1: Implementing the Singleton Pattern**

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

Program.cs:

using System;

class Program

{

    static void Main(string[] args)

    {

        Logger logger1 = Logger.GetInstance();

        logger1.Log("This is the first log message.");

        Logger logger2 = Logger.GetInstance();

        logger2.Log("This is the second log message.");

        Console.WriteLine(object.ReferenceEquals(logger1, logger2)

            ? "\n Only one Logger instance exists."

            : "\n Different Logger instances exist.");

    }

}

Logger.cs:

using System;

public class Logger

{

    private static Logger? instance = null;

    private static readonly object lockObj = new object();

    private Logger()

    {

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

    }

    public static Logger GetInstance()

    {

        if (instance == null)

        {

            lock (lockObj)

            {

                if (instance == null)

                {

                    instance = new Logger();

                }

            }

        }

        return instance;

    }

    public void Log(string message)

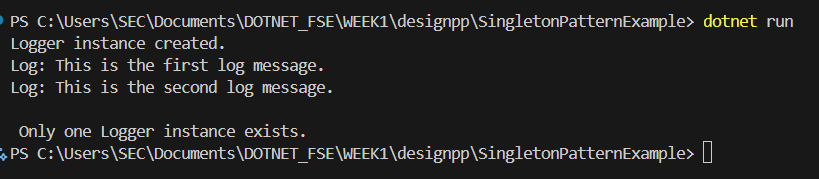
    {

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

    }

}

Output:



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

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.

IDocument.cs :

public interface IDocument

{

    void Open();

}

WordDocument.cs:

using System;

public class WordDocument : IDocument

{

    public void Open()

    {

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

    }

}

PdfDocument.cs:

using System;

public class PdfDocument : IDocument

{

    public void Open()

    {

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

    }

}

ExcelDocument.cs:

using System;

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();

    }

}

Program.cs:

using System;

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:

