**Design Patterns and Principles:-**

**Question 1:-**

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 Java Project:

o Create a new Java project named SingletonPatternExample.

2. Define a Singleton Class:

o Create a class named Logger that has a private static instance of itself.

o Ensure the constructor of Logger is private.

o Provide a public static method to get the instance of the Logger class.

3. Implement the Singleton Pattern:

o Write code to ensure that the Logger class follows the Singleton design pattern.

4. Test the Singleton Implementation:

o Create a test class to verify that only one instance of Logger is created and used across the application.

**Solution:-**

**Program.cs**

using System;

class Program

{

    static void Main(string[] args)

    {

        Logger logger1 = Logger.Instance;

        Logger logger2 = Logger.Instance;

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

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

        if (ReferenceEquals(logger1, logger2))

        {

            Console.WriteLine("Logger is a singleton: Both instances are the same.");

        }

        else

        {

            Console.WriteLine("Logger is NOT a singleton: Instances are different.");

        }

    }

}

**Logger.cs**

using System;

public sealed class Logger

{

    private static Logger? \_instance = null;

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

    private Logger()

    {

        Console.WriteLine("Logger initialized.");

    }

    public static Logger Instance

    {

        get

        {

            if (\_instance == null)

            {

                lock (\_lock)

                {

                    if (\_instance == null)

                    {

                        \_instance = new Logger();

                    }

                }

            }

            return \_instance;

        }

    }

    public void Log(string message)

    {

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

    }

}

**Output:-**

**PS C:\Users\KIIT\SingletonPatternExample> dotnet run**

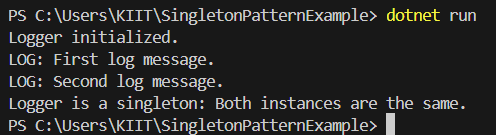
**Logger initialized.**

**LOG: First log message.**

**LOG: Second log message.**

**Logger is a singleton: Both instances are the same.**

**Screenshot:**

****

**Question 2:-**

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:

o Create a new Java project named FactoryMethodPatternExample.

2. Define Document Classes:

o Create interfaces or abstract classes for different document types such as WordDocument, PdfDocument, and ExcelDocument.

3. Create Concrete Document Classes:

o Implement concrete classes for each document type that implements or extends the above interfaces or abstract classes.

4. Implement the Factory Method:

o Create an abstract class DocumentFactory with a method createDocument().

o Create concrete factory classes for each document type that extends DocumentFactory and implements the createDocument() method.

5. Test the Factory Method Implementation:

o Create a test class to demonstrate the creation of different document types using the factory method.

**Solution:-**

**DocumentFactory.cs**

public abstract class DocumentFactory

{

    public abstract IDocument CreateDocument();

}

**ExcelDocument.cs**

public class ExcelDocument : IDocument

{

    public void Open()

    {

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

    }

}

**ExcelDocumentFactory.cs**

public class ExcelDocumentFactory : DocumentFactory

{

    public override IDocument CreateDocument()

    {

        return new ExcelDocument();

    }

}

**IDocument.cs**

public interface IDocument

{

    void Open();

}

**PdfDocument.cs**

public class PdfDocument : IDocument

{

    public void Open()

    {

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

    }

}

**PdfDocumentFactory.cs**

public class PdfDocumentFactory : DocumentFactory

{

    public override IDocument CreateDocument()

    {

        return new PdfDocument();

    }

}

**WordDocument.cs**

public class WordDocument : IDocument

{

    public void Open()

    {

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

    }

}

**WordDocumentFactory.cs**

public class WordDocumentFactory : DocumentFactory

{

    public override IDocument CreateDocument()

    {

        return new WordDocument();

    }

}

**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();

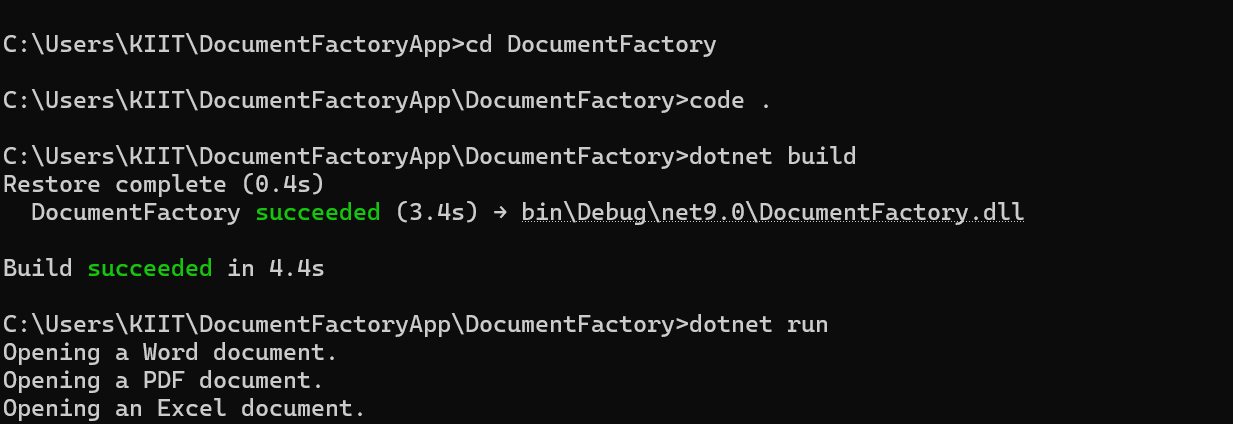
        IDocument excelDoc = excelFactory.CreateDocument();

        excelDoc.Open();

    }

}

**Output:**

****