Design Pattern and principles

Ex.1 Implementing the singleton pattern. (in C# dotnet)

Solution

>**Logger.cs**

public sealed class Logger

{

    private static readonly Logger \_instance = new Logger();

    private Logger()

    {

        Console.WriteLine("Logger instance created");

    }

    public static Logger Instance => \_instance;

    public void Log(string message)

    {

        Console.WriteLine($"[LOG] {DateTime.Now}: {message}");

    }

}

>**Program.cs**

class Program

{

    static void Main(string[] args)

    {

        Logger logger1 = Logger.Instance;

        logger1.Log("First log message");

        Logger logger2 = Logger.Instance;

        logger2.Log("Second log message");

        Console.WriteLine($"Same instance? {ReferenceEquals(logger1, logger2)}");

    }

}

>**SingletonPatternExample.csproj**

<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>

    <OutputType>Exe</OutputType>

    <TargetFramework>net9.0</TargetFramework>

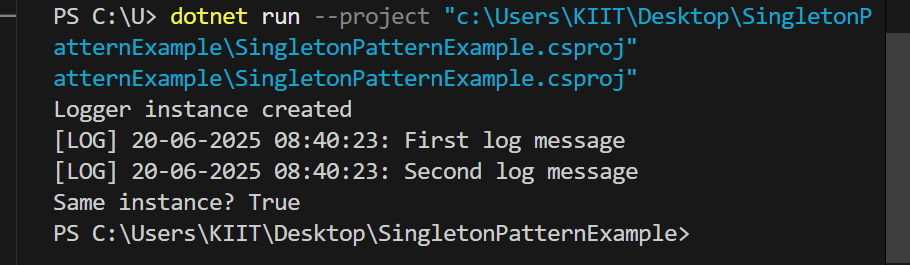
    <ImplicitUsings>enable</ImplicitUsings>

    <Nullable>enable</Nullable>

  </PropertyGroup>

</Project>

**Output:**



**Ex.2 Implementing the Factory method pattern:**

**Solution:**

>**Program.cs**

class Program

{

    static void Main()

    {

        Console.WriteLine("Document Management System designed by Vivek Prasad(SUPERSET ID-6362613)");

        Console.WriteLine("===========================");

        Console.WriteLine("\nCreating Word Document:");

        DocumentFactory wordFactory = new WordDocumentFactory();

        IDocument wordDoc = wordFactory.CreateDocument();

        wordDoc.Open();

        wordDoc.Close();

        Console.WriteLine("\nCreating PDF Document:");

        DocumentFactory pdfFactory = new PdfDocumentFactory();

        IDocument pdfDoc = pdfFactory.CreateDocument();

        pdfDoc.Open();

        pdfDoc.Close();

        Console.WriteLine("\nCreating Excel Document:");

        DocumentFactory excelFactory = new ExcelDocumentFactory();

        IDocument excelDoc = excelFactory.CreateDocument();

        excelDoc.Open();

        excelDoc.Close();

        Console.WriteLine("\nPress any key to exit...");

        Console.ReadKey();

    }

}

>**IDocument.cs**

public interface IDocument

{

    void Open();

    void Close();

}

>**Document.cs:**

public class WordDocument : IDocument

{

    public void Open()

    {

        Console.WriteLine("Opening Word Document");

    }

    public void Close()

    {

        Console.WriteLine("Closing Word Document");

    }

}

public class PdfDocument : IDocument

{

    public void Open()

    {

        Console.WriteLine("Opening PDF Document");

    }

    public void Close()

    {

        Console.WriteLine("Closing PDF Document");

    }

}

public class ExcelDocument : IDocument

{

    public void Open()

    {

        Console.WriteLine("Opening Excel Document");

    }

    public void Close()

    {

        Console.WriteLine("Closing Excel Document");

    }

}

>**concretefact.cs**

public class WordDocumentFactory : DocumentFactory

{

    public override IDocument CreateDocument() => new WordDocument();

}

public class PdfDocumentFactory : DocumentFactory

{

    public override IDocument CreateDocument() => new PdfDocument();

}

public class ExcelDocumentFactory : DocumentFactory

{

    public override IDocument CreateDocument() => new ExcelDocument();

}

>**AbstractFactory.cs**

public abstract class DocumentFactory

{

    public abstract IDocument CreateDocument();

}

**>factorymethodpattern.csproj**

<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>

    <OutputType>Exe</OutputType>

    <TargetFramework>net9.0</TargetFramework>

    <ImplicitUsings>enable</ImplicitUsings>

    <Nullable>enable</Nullable>

  </PropertyGroup>

**</Project>**

**Output of EX.2**

