# Design Patterns & Principles Exercise Solutions

## Exercise 1: Implementing the Singleton Pattern

### Code & Output Screenshots:

Logger.cs

using System;

namespace Exercise

{

public class Logger

{

private static Logger instance;

private static readonly object lockObj = new object();

private Logger() { }

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);

}

}

}

LoggerTest.cs

using System;

namespace Exercise

{

public class LoggerTest

{

public static void Main(string[] args)

{

Logger logger1 = Logger.GetInstance();

Logger logger2 = Logger.GetInstance();

logger1.Log("This is from logger1");

logger2.Log("This is from logger2");

if (object.ReferenceEquals(logger1, logger2))

{

Console.WriteLine("Both logger1 and logger2 are the same instance.");

}

else

{

Console.WriteLine("logger1 and logger2 are different instances.");

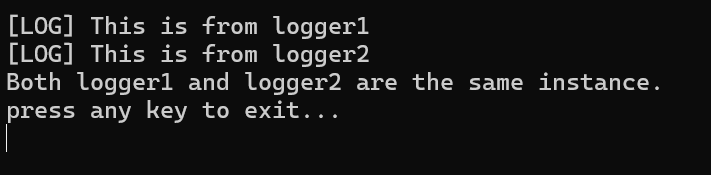
}

}

}

}

Output



## Exercise 2: Implementing the Factory Method Pattern

### Code & Output Screenshots:

*IDocument.cs*

namespace Exercise.FactoryMethods

{

public interface IDocument

{

void Open();

void Close();

void Save();

}

}

*WordDocument.cs*

using System;

namespace Exercise.FactoryMethods

{

public class WordDocument : IDocument

{

public void Open() => Console.WriteLine("Opening Word Document");

public void Close() => Console.WriteLine("Closing Word Document");

public void Save() => Console.WriteLine("Saving Word Document");

}

}

*PdfDocument.cs*

using System;

namespace Exercise.FactoryMethods

{

public class PdfDocument : IDocument

{

public void Open() => Console.WriteLine("Opening PDF Document");

public void Close() => Console.WriteLine("Closing PDF Document");

public void Save() => Console.WriteLine("Saving PDF Document");

}

}

*ExcelDocument.cs*

using System;

namespace Exercise.FactoryMethods

{

public class ExcelDocument : IDocument

{

public void Open() => Console.WriteLine("Opening Excel Document");

public void Close() => Console.WriteLine("Closing Excel Document");

public void Save() => Console.WriteLine("Saving Excel Document");

}

}

*WordDocumentFactory.cs*

namespace Exercise.FactoryMethods

{

public class WordDocumentFactory : DocumentFactory

{

public override IDocument CreateDocument()

{

return new WordDocument();

}

}

}

*PdfDocumentFactory.cs*

namespace Exercise.FactoryMethods

{

public class PdfDocumentFactory : DocumentFactory

{

public override IDocument CreateDocument()

{

return new PdfDocument();

}

}

}

*ExcelDocumentFactory.cs*

namespace Exercise.FactoryMethods

{

public class ExcelDocumentFactory : DocumentFactory

{

public override IDocument CreateDocument()

{

return new ExcelDocument();

}

}

}

*DocumentFactory.cs*

namespace Exercise.FactoryMethods

{

public abstract class DocumentFactory

{

public abstract IDocument CreateDocument();

}

}

*Program.cs*

using System;

using Exercise.FactoryMethods;

namespace Exercise

{

public class Program

{

public static void Main(string[] args)

{

DocumentFactory wordFactory = new WordDocumentFactory();

IDocument wordDoc = wordFactory.CreateDocument();

wordDoc.Open();

wordDoc.Save();

wordDoc.Close();

Console.WriteLine("--------------------");

DocumentFactory pdfFactory = new PdfDocumentFactory();

IDocument pdfDoc = pdfFactory.CreateDocument();

pdfDoc.Open();

pdfDoc.Save();

pdfDoc.Close();

Console.WriteLine("--------------------");

DocumentFactory excelFactory = new ExcelDocumentFactory();

IDocument excelDoc = excelFactory.CreateDocument();

excelDoc.Open();

excelDoc.Save();

excelDoc.Close();

}

}

}

*Output*

