**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](http://program.cs)

// See https://aka.ms/new-console-template for more information

// Console.WriteLine("Hello, World!");

using System;

namespace SingletonPatternExample

{

class Program

{

static void Main(string[] args)

{

// two instances of Logger

Logger logger1 = Logger.Instance;

Logger logger2 = Logger.Instance;

// logger to log messages

logger1.Log("First logger instance message");

logger2.Log("Second logger instance message");

// they are the same instance

if (logger1 == logger2)

{

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

}

else

{

Console.WriteLine("They are different instances — Singleton failed.");

}

}

}

}

[Logger.cs](http://logger.cs)

using System;

namespace SingletonPatternExample

{

public class Logger

{

// Create a private static instance of Logger

private static Logger? \_instance;

// Lock object to make thread-safe

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

// constructor

private Logger()

{

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

}

// global access to the instance

public static Logger Instance

{

get

{

// Double-checked locking

if (\_instance == null)

{

lock (\_lock)

{

if (\_instance == null)

{

\_instance = new Logger();

}

}

}

return \_instance;

}

}

// log method

public void Log(string message)

{

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

}

}

}

**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

namespace FactoryMethodPattern

{

public interface IDocument

{

void Create(string filename); // really creates the file

}

}

WordDocument.cs

using DocumentFormat.OpenXml.Packaging;

using DocumentFormat.OpenXml.Wordprocessing;

using System;

namespace FactoryMethodPattern

{

public class WordDocument : IDocument

{

public void Create(string filename)

{

string filepath = filename + ".docx";

// Create a Wordprocessing document.

using (WordprocessingDocument wordDocument =

WordprocessingDocument.Create(filepath, DocumentFormat.OpenXml.WordprocessingDocumentType.Document))

{

// Add a main document part.

MainDocumentPart mainPart = wordDocument.AddMainDocumentPart();

// Create the document structure and add some text.

mainPart.Document = new Document();

Body body = new Body();

Paragraph para = new Paragraph();

Run run = new Run();

Text text = new Text("This is a Word document created using Open XML SDK.");

run.Append(text);

para.Append(run);

body.Append(para);

mainPart.Document.Append(body);

mainPart.Document.Save();

}

Console.WriteLine(" Word document created using Open XML SDK.");

}

}

}

PdfDocument.cs

using PdfSharp.Pdf;

using PdfSharp.Drawing;

using System.Diagnostics;

namespace FactoryMethodPattern

{

public class PdfDocumentCreator : IDocument

{

public void Create(string filename)

{

PdfDocument document = new PdfDocument(); // This now refers to PdfSharp.Pdf.PdfDocument

document.Info.Title = "Created with PdfSharp";

PdfPage page = document.AddPage();

XGraphics gfx = XGraphics.FromPdfPage(page);

gfx.DrawString("This is a PDF document.",

new XFont("Verdana", 20, XFontStyle.Bold),

XBrushes.Black,

new XRect(0, 0, page.Width, page.Height),

XStringFormats.Center);

document.Save(filename + ".pdf");

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

}

}

}

ExcelDocument.cs

using OfficeOpenXml;

using System.IO;

namespace FactoryMethodPattern

{

public class ExcelDocument : IDocument

{

public void Create(string filename)

{

ExcelPackage.LicenseContext = LicenseContext.NonCommercial;

using (ExcelPackage package = new ExcelPackage())

{

var sheet = package.Workbook.Worksheets.Add("Sheet1");

sheet.Cells[1, 1].Value = "This is an Excel document.";

File.WriteAllBytes(filename + ".xlsx", package.GetAsByteArray());

Console.WriteLine("Excel document created.");

}

}

}

}

DocumentFacotry.cs

namespace FactoryMethodPattern

{

public abstract class DocumentFactory

{

public abstract IDocument CreateDocument();

}

}

WordFactory.cs

namespace FactoryMethodPattern

{

public abstract class DocumentFactory

{

public abstract IDocument CreateDocument();

}

}

PdfFactory.cs

namespace FactoryMethodPattern

{

public class PdfFactory : DocumentFactory

{

public override IDocument CreateDocument()

{

return new PdfDocumentCreator();

}

}

}

ExcelFactory.cs

namespace FactoryMethodPattern

{

public class ExcelFactory : DocumentFactory

{

public override IDocument CreateDocument()

{

return new ExcelDocument();

}

}

}

Program.cs

using System;

using System.Text; // Needed for Encoding

namespace FactoryMethodPattern

{

class Program

{

static void Main(string[] args)

{

Encoding.RegisterProvider(CodePagesEncodingProvider.Instance);

Console.WriteLine("Select document type to create: Word / PDF / Excel");

string input = Console.ReadLine()?.Trim().ToLower();

DocumentFactory factory = input switch

{

"word" => new WordFactory(),

"pdf" => new PdfFactory(),

"excel" => new ExcelFactory(),

\_ => throw new Exception("Invalid type")

};

Console.Write("Enter filename (without extension): ");

string filename = Console.ReadLine();

IDocument document = factory.CreateDocument();

document.Create(filename);

}

}