

Design Pattern and Principles

Exercise 1: Implementing Singleton Pattern

Code:

```
MyLogger.cs X Program.cs SingletonPatternExample.csproj
C# MyLogger.cs
1  Using System;
2
3  public class MyLogger
4  {
5      private static MyLogger _instance;
6      private static readonly object _lock = new object();
7
8      private MyLogger()
9      {
10         Console.WriteLine("Logger instance created.");
11     }
12
13     public static MyLogger GetInstance()
14     {
15         if (_instance == null)
16         {
17             lock (_lock)
18             {
19                 if (_instance == null)
20                 {
21                     _instance = new MyLogger();
22                 }
23             }
24         }
25
26         return _instance;
27     }
28
29     public void Log(string message)
```

```
28  
29     public void Log(string message)  
30     {  
31         Console.WriteLine("[LOG]: " + message);  
32     }  
33 }  
34
```

The screenshot shows a Windows application window with three tabs at the top: 'MyLogger.cs', 'Program.cs X', and 'SingletonPatternExample.csproj'. The 'Program.cs' tab is selected, displaying the following C# code:

```
1  using System;
2
3  class Program
4  {
5      static void Main(string[] args)
6      {
7          MyLogger logger1 = MyLogger.GetInstance();
8          logger1.Log("Application started.");
9
10         MyLogger logger2 = MyLogger.GetInstance();
11         logger2.Log("Another message logged.");
12
13         if (logger1 == logger2)
14         {
15             Console.WriteLine("logger1 and logger2 are the same instance.");
16         }
17         else
18         {
19             Console.WriteLine("Different instances! Singleton failed.");
20         }
21     }
22 }
```

MyLogger.cs	Program.cs	SingletonPatternExample.csproj
-------------	------------	--------------------------------

```
 SingletonPatternExample.csproj
 1 <Project Sdk="Microsoft.NET.Sdk">
 2
 3   <PropertyGroup>
 4     <OutputType>Exe</OutputType>
 5     <TargetFramework>net8.0</TargetFramework>
 6     <ImplicitUsings>enable</ImplicitUsings>
 7     <Nullable>enable</Nullable>
 8   </PropertyGroup>
```

Output:

A screenshot of a terminal window from a development environment like Visual Studio Code. The window has tabs at the top: PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is selected), and PORTS. On the right side, there are icons for powershell, a plus sign, a refresh, a trash can, and three dots. The main area of the terminal shows the following text:

```
Logger instance created.  
[LOG]: Application started.  
[LOG]: Another message logged.  
logger1 and logger2 are the same instance.  
PS C:\Users\KIIT\Desktop\DotNet FSE\Engineering concepts\Design patterns and principles\SingletonPatternExample> 
```

Design Pattern and Principles

Exercise 2: Implementing the Factory Method Pattern

Code:

```
C# Program.cs X
C# Program.cs
1  using System;
2
3  // Document interface
4  public interface IDocument
5  {
6      void Open();
7      void Save();
8  }
9
10 // Concrete Documents
11 public class WordDocument : IDocument
12 {
13     public void Open()
14     {
15         Console.WriteLine("Opening Word document...");
16     }
17
18     public void Save()
19     {
20         Console.WriteLine("Saving Word document...");
21     }
22 }
23
24 public class PdfDocument : IDocument
25 {
26     public void Open()
27     {
28         Console.WriteLine("Opening PDF document...");
29     }
}
```

```
C# Program.cs X
C# Program.cs
25  {
26
27     public void Save()
28     {
29         Console.WriteLine("Saving PDF document...");
30     }
31
32     public class ExcelDocument : IDocument
33     {
34         public void Open()
35         {
36             Console.WriteLine("Opening Excel document...");
37         }
38
39         public void Save()
40         {
41             Console.WriteLine("Saving Excel document...");
42         }
43
44     }
45
46     // Abstract Factory
47     public abstract class DocumentFactory
48     {
49         public abstract IDocument CreateDocument();
50     }
51
52     // Concrete Factories
53     public class WordDocumentFactory : DocumentFactory
54     {
55
56         public IDocument CreateDocument()
57         {
58             return new WordDocument();
59         }
60
61     }
62 }
```

```
C# Program.cs X
C# Program.cs
56     // Concrete factories
57     public class WordDocumentFactory : DocumentFactory
58     {
59         public override IDocument CreateDocument()
60         {
61             return new WordDocument();
62         }
63
64     }
65
66     public class PdfDocumentFactory : DocumentFactory
67     {
68         public override IDocument CreateDocument()
69         {
70             return new PdfDocument();
71         }
72
73     }
74
75     public class ExcelDocumentFactory : DocumentFactory
76     {
77         public override IDocument CreateDocument()
78         {
79             return new ExcelDocument();
80         }
81
82     }
83
84     class Program
85     {
86         static void Main(string[] args)
87         {
88             DocumentFactory factory;
89             IDocument document;
90
91             // Create a Word Document
92             factory = new WordDocumentFactory();
93             document = factory.CreateDocument();
94             document.Open();
95             document.Save();
96
97             // Create a PDF Document
98             factory = new PdfDocumentFactory();
99             document = factory.CreateDocument();
100            document.Open();
101            document.Save();
102        }
103    }
104 }
```

```
C# Program.cs X
C# Program.cs
82    {
83        static void Main(string[] args)
84        {
85            DocumentFactory factory;
86            IDocument document;
87
88            // Create a Word Document
89            factory = new WordDocumentFactory();
90            document = factory.CreateDocument();
91            document.Open();
92            document.Save();
93
94            Console.WriteLine();
95
96            // Create a PDF Document
97            factory = new PdfDocumentFactory();
98            document = factory.CreateDocument();
99            document.Open();
100           document.Save();
101       }
102   }
103 }
```

```
100     document.Save();
101
102     Console.WriteLine();
103
104     // Create an Excel Document
105     factory = new ExcelDocumentFactory();
106     document = factory.CreateDocument();
107     document.Open();
108     document.Save();
109 }
110 }
```

OUTPUT :



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS powershell + □
● >> nExample>
Opening Word document...
Saving Word document...

Opening PDF document...
Saving PDF document...

Opening Excel document...
Saving Excel document...
○ PS C:\Users\KIIT\Desktop\DotNet FSE\Engineering concepts\Design patterns and principles\FactoryMethodPatternExample>
```