## By Sachin Ray (SuperSet ID: 6364957)

## 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:**
   - Create a new Java project named **SingletonPatternExample**.
2. **Define a Singleton Class:**
   - Create a class named Logger that has a private static instance of itself.
   - Ensure the constructor of Logger is private.
   - Provide a public static method to get the instance of the Logger class.
3. **Implement the Singleton Pattern:**
   - Write code to ensure that the Logger class follows the Singleton design pattern.
4. **Test the Singleton Implementation:**
   - Create a test class to verify that only one instance of Logger is created and used across the application.

**SOLUTION** :

Logger.cs Code :

```
using System;

using System.IO;

namespace SingletonPatternExample

  {

  public sealed class Logger

  {

    private static Logger? _instance = null;

    private static readonly object _lock = new object();


    private Logger()

    {
```

```
    InitializeLogger();
}

public static Logger GetInstance()
{
    if (_instance == null)
    {
        lock (_lock)
        {
            if (_instance == null)
            {
                _instance = new Logger();
            }
        }
    }
    return _instance;
}

private void InitializeLogger()
{
    Console.WriteLine("Logger initialized successfully!");
}

public void LogInfo(string message)
{
    string logEntry = $"[INFO] {DateTime.Now:yyyy-MM-dd HH:mm:ss} - {message}";
    Console.WriteLine(logEntry);
    WriteToFile(logEntry);
}

public void LogError(string message)
```

```
    {
        string logEntry = $"[ERROR] {DateTime.Now:yyyy-MM-dd HH:mm:ss} - {message}";
        Console.WriteLine(logEntry);
        WriteToFile(logEntry);
    }

    public void LogWarning(string message)
    {
        string logEntry = $"[WARNING] {DateTime.Now:yyyy-MM-dd HH:mm:ss} - {message}";
        Console.WriteLine(logEntry);
        WriteToFile(logEntry);
    }

    private void WriteToFile(string logEntry)
    {
        try
        {
            string logFile = "application.log";
            File.AppendAllText(logFile, logEntry + Environment.NewLine);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Failed to write to log file: {ex.Message}");
        }
    }

    public override int GetHashCode()
    {
        return base.GetHashCode();
    }
}
```

```
}
```

SingletonTest.cs Code :

```
using System;
using System.Threading;

namespace SingletonPatternExample
{
    public class SingletonTest
    {
        public static void TestSingletonPattern()
        {
            Console.WriteLine("=== Testing Singleton Pattern ===\n");

            Logger logger1 = Logger.GetInstance();
            Logger logger2 = Logger.GetInstance();
            Logger logger3 = Logger.GetInstance();

            Console.WriteLine("Testing instance equality:");
            Console.WriteLine($"logger1 == logger2: {ReferenceEquals(logger1, logger2)}");
            Console.WriteLine($"logger2 == logger3: {ReferenceEquals(logger2, logger3)}");
            Console.WriteLine($"logger1 == logger3: {ReferenceEquals(logger1, logger3)}");

            Console.WriteLine($"\nInstance Hash Codes:");
            Console.WriteLine($"logger1 HashCode: {logger1.GetHashCode()}");
            Console.WriteLine($"logger2 HashCode: {logger2.GetHashCode()}");
            Console.WriteLine($"logger3 HashCode: {logger3.GetHashCode()}");

            Console.WriteLine("\n=== Testing Logger Functionality ===");
            logger1.LogInfo("Application started successfully");
            logger2.LogWarning("This is a warning message from logger2");
            logger3.LogError("This is an error message from logger3");
            logger1.LogInfo("All loggers are actually the same instance");

            Console.WriteLine("\n=== Testing Thread Safety ===");
            Thread[] threads = new Thread[5];

            for (int i = 0; i < 5; i++)
            {
                int threadId = i;
                threads[i] = new Thread(() =>
                {
                    Logger threadLogger = Logger.GetInstance();
                    threadLogger.LogInfo($"Message from Thread {threadId} - HashCode:
{threadLogger.GetHashCode()}");
                });
            }

            foreach (Thread thread in threads)
            {
                thread.Start();
            }

            foreach (Thread thread in threads)
            {
                thread.Join();
            }

            Console.WriteLine("\nSingleton Pattern test completed successfully!");
        }
    }
```

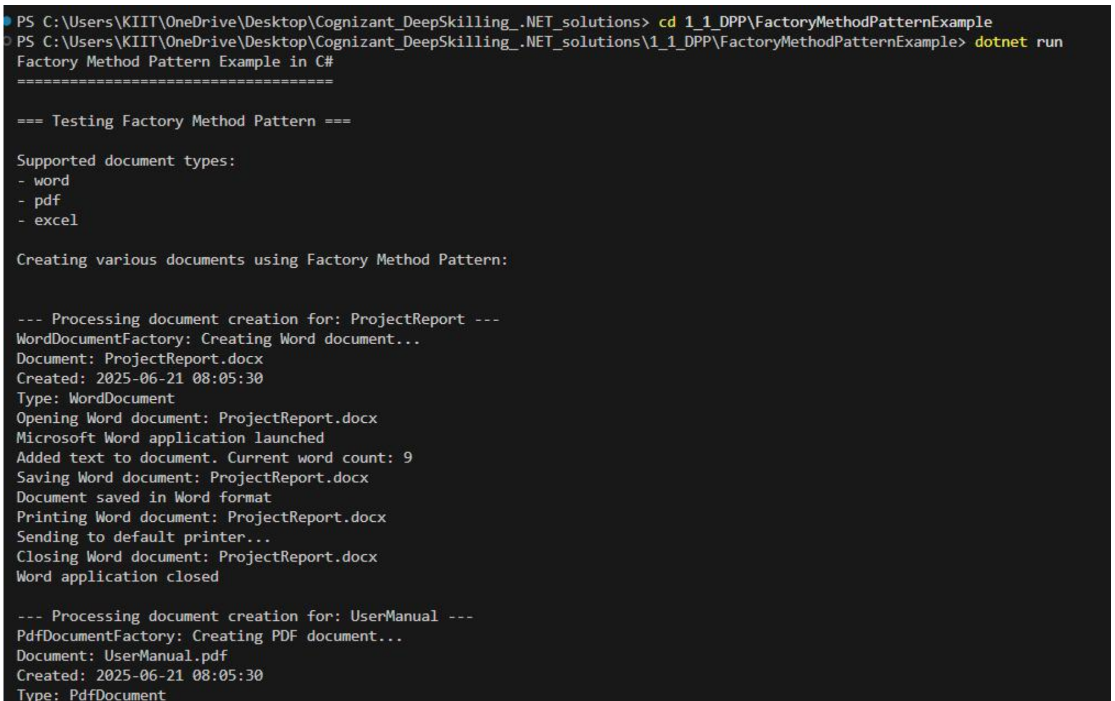
```
}
```

Main file:

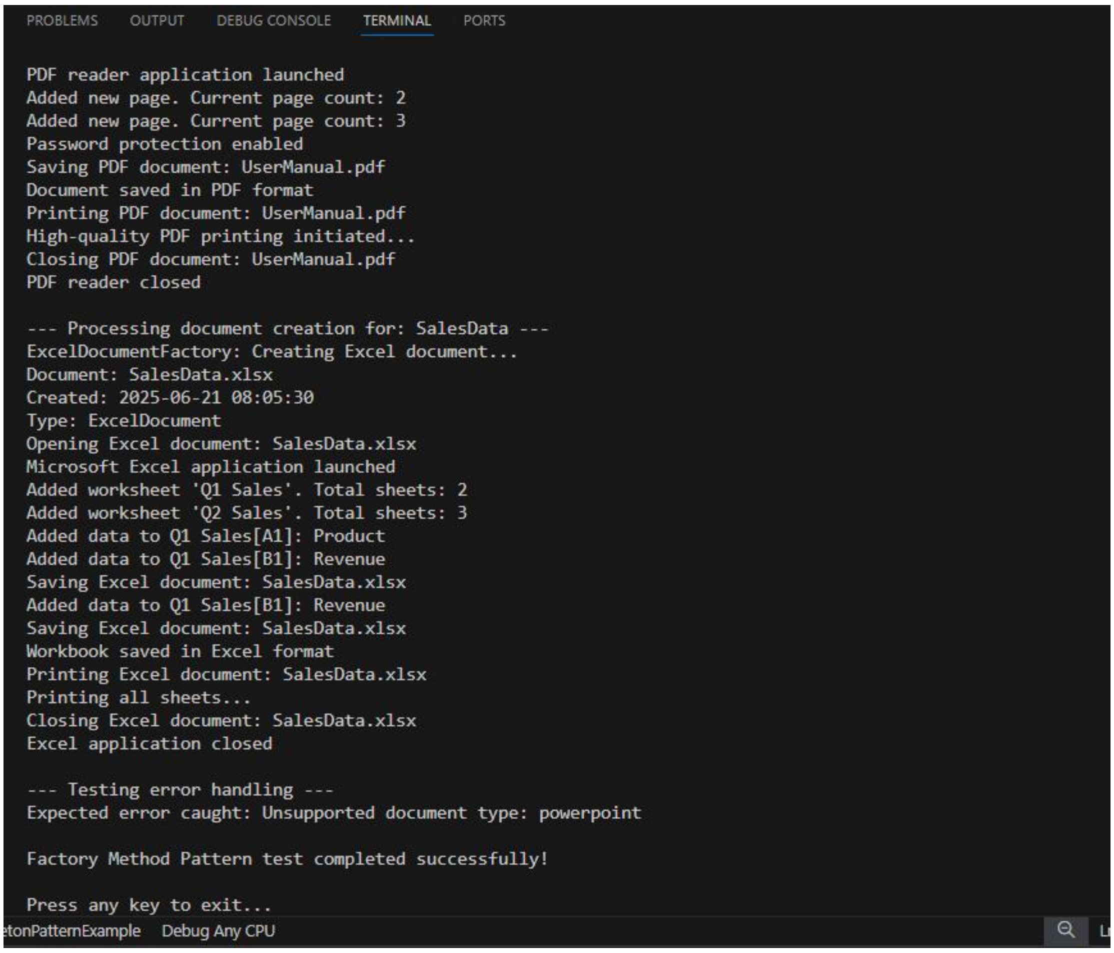
Program.cs Code :

```
using System;


namespace SingletonPatternExample

{

   class Program

   {

      static void Main(string[] args)

      {

         Console.WriteLine("Singleton Pattern Example in C#");

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


         SingletonTest.TestSingletonPattern();


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

         Console.ReadKey();

        }

   }

}
```

**OUTPUT** :



```
PS C:\Users\KIIT\OneDrive\Desktop\Cognizant_DeepSkilling_.NET_solutions> cd 1_1_DPP\FactoryMethodPatternExample
PS C:\Users\KIIT\OneDrive\Desktop\Cognizant_DeepSkilling_.NET_solutions\1_1_DPP\FactoryMethodPatternExample> dotnet run
Factory Method Pattern Example in C#
====================================

=== Testing Factory Method Pattern ===

Supported document types:
- word
- pdf
- excel

Creating various documents using Factory Method Pattern:


--- Processing document creation for: ProjectReport ---
WordDocumentFactory: Creating Word document...
Document: ProjectReport.docx
Created: 2025-06-21 08:05:30
Type: WordDocument
Opening Word document: ProjectReport.docx
Microsoft Word application launched
Added text to document. Current word count: 9
Saving Word document: ProjectReport.docx
Document saved in Word format
Printing Word document: ProjectReport.docx
Sending to default printer...
Closing Word document: ProjectReport.docx
Word application closed

--- Processing document creation for: UserManual ---
PdfDocumentFactory: Creating PDF document...
Document: UserManual.pdf
Created: 2025-06-21 08:05:30
Type: PdfDocument
```



```
PDF reader application launched
Added new page. Current page count: 2
Added new page. Current page count: 3
Password protection enabled
Saving PDF document: UserManual.pdf
Document saved in PDF format
Printing PDF document: UserManual.pdf
High-quality PDF printing initiated...
Closing PDF document: UserManual.pdf
PDF reader closed

--- Processing document creation for: SalesData ---
ExcelDocumentFactory: Creating Excel document...
Document: SalesData.xlsx
Created: 2025-06-21 08:05:30
Type: ExcelDocument
Opening Excel document: SalesData.xlsx
Microsoft Excel application launched
Added worksheet 'Q1 Sales'. Total sheets: 2
Added worksheet 'Q2 Sales'. Total sheets: 3
Added data to Q1 Sales[A1]: Product
Added data to Q1 Sales[B1]: Revenue
Saving Excel document: SalesData.xlsx
Added data to Q1 Sales[B1]: Revenue
Saving Excel document: SalesData.xlsx
Workbook saved in Excel format
Printing Excel document: SalesData.xlsx
Printing all sheets...
Closing Excel document: SalesData.xlsx
Excel application closed

--- Testing error handling ---
Expected error caught: Unsupported document type: powerpoint

Factory Method Pattern test completed successfully!

Press any key to exit...
```

---

## 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:**

- Create a new Java project named **FactoryMethodPatternExample**.

2. **Define Document Classes:**
   - Create interfaces or abstract classes for different document types such as **WordDocument**, **PdfDocument**, and **ExcelDocument**.
3. **Create Concrete Document Classes:**
   - Implement concrete classes for each document type that implements or extends the above interfaces or abstract classes.
4. **Implement the Factory Method:**
   - Create an abstract class **DocumentFactory** with a method **createDocument()**.
   - Create concrete factory classes for each document type that extends DocumentFactory and implements the **createDocument()** method.
5. **Test the Factory Method Implementation:**
   - Create a test class to demonstrate the creation of different document types using the factory method.

**SOLUTION :**

CODES Folder-Wise:

For Documents Folder:

1. Document.cs code :

```
using System;


namespace FactoryMethodPatternExample

{


   public abstract class Document

   {

      public string Name { get; protected set; }

      public string FileExtension { get; protected set; }

      public DateTime CreatedDate { get; protected set; }
```

```
        protected Document(string name)
        {
            Name = name;
            CreatedDate = DateTime.Now;
        }

        // Abstract methods that concrete documents must implement
        public abstract void Open();
        public abstract void Save();
        public abstract void Close();
        public abstract void Print();

        // Common method for all documents
        public virtual void DisplayInfo()
        {
            Console.WriteLine($"Document: {Name}{FileExtension}");
            Console.WriteLine($"Created: {CreatedDate:yyyy-MM-dd HH:mm:ss}");
            Console.WriteLine($"Type: {GetType().Name}");
        }
    }
}
```

2. ExcelDocument.cs  code :

```
 using System;
using System.Collections.Generic;

namespace FactoryMethodPatternExample
{

    public class ExcelDocument : Document
    {
```

```
public int SheetCount { get; private set; }
public List<string> SheetNames { get; private set; }

public ExcelDocument(string name) : base(name)
{
   FileExtension = ".xlsx";
   SheetCount = 1;
   SheetNames = new List<string> { "Sheet1" };
}

public override void Open()
{
   Console.WriteLine($"Opening Excel document: {Name}{FileExtension}");
   Console.WriteLine("Microsoft Excel application launched");
}

public override void Save()
{
   Console.WriteLine($"Saving Excel document: {Name}{FileExtension}");
   Console.WriteLine("Workbook saved in Excel format");
}

public override void Close()
{
   Console.WriteLine($"Closing Excel document: {Name}{FileExtension}");
   Console.WriteLine("Excel application closed");
}

public override void Print()
{
   Console.WriteLine($"Printing Excel document: {Name}{FileExtension}");
```

```
            Console.WriteLine("Printing all sheets...");
        }

        public void AddWorksheet(string sheetName)
        {
            SheetCount++;
            SheetNames.Add(sheetName);
            Console.WriteLine($"Added worksheet '{sheetName}'. Total sheets: {SheetCount}");
        }

        public void AddData(string sheetName, string cellReference, object value)
        {
            Console.WriteLine($"Added data to {sheetName}[{cellReference}]: {value}");
        }
    }
}
```

3. PdfDocument.cs Code :

```
using System;

namespace FactoryMethodPatternExample
{

    public class PdfDocument : Document
    {
        public int PageCount { get; private set; }
        public bool IsPasswordProtected { get; private set; }

        public PdfDocument(string name) : base(name)
```

```
{
    FileExtension = ".pdf";
    PageCount = 1;
    IsPasswordProtected = false;
}

public override void Open()
{
    Console.WriteLine($"Opening PDF document: {Name}{FileExtension}");
    Console.WriteLine("PDF reader application launched");
}

public override void Save()
{
    Console.WriteLine($"Saving PDF document: {Name}{FileExtension}");
    Console.WriteLine("Document saved in PDF format");
}

public override void Close()
{
    Console.WriteLine($"Closing PDF document: {Name}{FileExtension}");
    Console.WriteLine("PDF reader closed");
}

public override void Print()
{
    Console.WriteLine($"Printing PDF document: {Name}{FileExtension}");
    Console.WriteLine("High-quality PDF printing initiated...");
}

public void AddPage()
```

```
        {
            PageCount++;
            Console.WriteLine($"Added new page. Current page count: {PageCount}");
        }

        public void SetPasswordProtection(bool enabled)
        {
            IsPasswordProtected = enabled;
            Console.WriteLine($"Password protection {(enabled ? "enabled" : "disabled")}");
        }
    }
}
```

4. wordDocument.cs code :

```
using System;

namespace FactoryMethodPatternExample
{

    public class WordDocument : Document
    {
        public int WordCount { get; private set; }

        public WordDocument(string name) : base(name)
        {
            FileExtension = ".docx";
            WordCount = 0;
        }

        public override void Open()
        {
            Console.WriteLine($"Opening Word document: {Name}{FileExtension}");
            Console.WriteLine("Microsoft Word application launched");
        }

        public override void Save()
        {
            Console.WriteLine($"Saving Word document: {Name}{FileExtension}");
            Console.WriteLine("Document saved in Word format");
        }

        public override void Close()
        {
            Console.WriteLine($"Closing Word document: {Name}{FileExtension}");
            Console.WriteLine("Word application closed");
        }

        public override void Print()
        {
            Console.WriteLine($"Printing Word document: {Name}{FileExtension}");
            Console.WriteLine("Sending to default printer...");
        }

        public void AddText(string text)
        {
            WordCount += text.Split(' ').Length;
            Console.WriteLine($"Added text to document. Current word count: {WordCount}");
        }
    }
}
```

For Factories Folder:

5. DocumentFactory.cs code :

```
 using System;


namespace FactoryMethodPatternExample

{


   public abstract class DocumentFactory

   {

      // Factory method - to be implemented by concrete factories

      public abstract Document CreateDocument(string name);


      // Template method that uses the factory method

      public Document ProcessDocument(string name)

      {

         Console.WriteLine($"\n--- Processing document creation for: {name} ---");

         Document document = CreateDocument(name);


         // Common processing steps

         document.DisplayInfo();

         document.Open();


         return document;

      }

   }

}
```

6. ExcelDocumentFactory.cs  code :

```
using System;

namespace FactoryMethodPatternExample
{

    public class ExcelDocumentFactory : DocumentFactory
    {
        public override Document CreateDocument(string name)
        {
            Console.WriteLine("ExcelDocumentFactory: Creating Excel document...");
            return new ExcelDocument(name);
        }
    }
}
```

7. PdfDocumentFactory.cs Code :

```
using System;

namespace FactoryMethodPatternExample
{

    public class PdfDocumentFactory : DocumentFactory
    {
        public override Document CreateDocument(string name)
        {
            Console.WriteLine("PdfDocumentFactory: Creating PDF document...");
            return new PdfDocument(name);
        }
    }
}
```

8. wordDocumentFactory.cs code :

```
using System;

namespace FactoryMethodPatternExample
{

  public class WordDocumentFactory : DocumentFactory
  {
    public override Document CreateDocument(string name)
    {
      Console.WriteLine("WordDocumentFactory: Creating Word document...");
      return new WordDocument(name);
    }
  }
}
```

For FactoryMethodPatternExample Folder:

9. DocumentManager.cs code :

```
using System;
using System.Collections.Generic;

namespace FactoryMethodPatternExample
{

  public class DocumentManager
  {
    private Dictionary<string, DocumentFactory> _factories;

    public DocumentManager()
```

```
{
    _factories = new Dictionary<string, DocumentFactory>
    {
        { "word", new WordDocumentFactory() },
        { "pdf", new PdfDocumentFactory() },
        { "excel", new ExcelDocumentFactory() }
    };
}

public Document CreateDocument(string type, string name)
{
    type = type.ToLower();

    if (_factories.ContainsKey(type))
    {
        return _factories[type].ProcessDocument(name);
    }
    else
    {
        throw new ArgumentException($"Unsupported document type: {type}");
    }
}

public void ListSupportedTypes()
{
    Console.WriteLine("Supported document types:");
    foreach (var type in _factories.Keys)
    {
        Console.WriteLine($"- {type}");
    }
}
```

```
    }
}
```

10. FactoryMethodTest.cs  code :

```
using System;

namespace FactoryMethodPatternExample
        {

    public class FactoryMethodTest
    {
        public static void TestFactoryMethod()
        {
            Console.WriteLine("=== Testing Factory Method Pattern ===\n");

            DocumentManager manager = new DocumentManager();

            // Display supported types
            manager.ListSupportedTypes();
            Console.WriteLine();

            try
            {
                // Test creating different types of documents
                Console.WriteLine("Creating various documents using Factory Method Pattern:\n");

                // Create Word document
                Document wordDoc = manager.CreateDocument("word", "ProjectReport");
                if (wordDoc is WordDocument wd)
                {
```

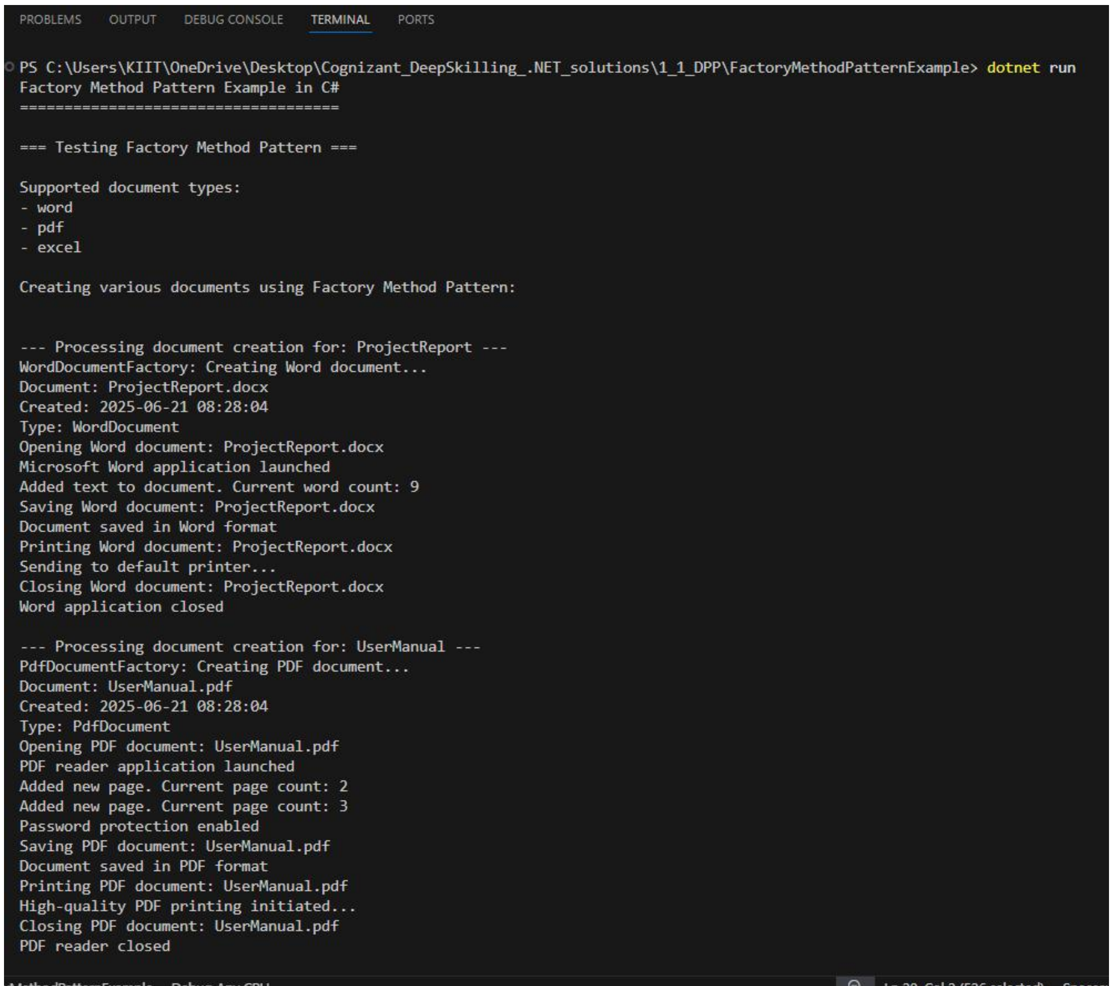
```
    wd.AddText("This is a sample project report with multiple paragraphs.");
    wd.Save();
    wd.Print();
    wd.Close();
}

// Create PDF document
Document pdfDoc = manager.CreateDocument("pdf", "UserManual");
if (pdfDoc is PdfDocument pd)
{
    pd.AddPage();
    pd.AddPage();
    pd.SetPasswordProtection(true);
    pd.Save();
    pd.Print();
    pd.Close();
}

// Create Excel document
Document excelDoc = manager.CreateDocument("excel", "SalesData");
if (excelDoc is ExcelDocument ed)
{
    ed.AddWorksheet("Q1 Sales");
    ed.AddWorksheet("Q2 Sales");
    ed.AddData("Q1 Sales", "A1", "Product");
    ed.AddData("Q1 Sales", "B1", "Revenue");
    ed.Save();
    ed.Print();
    ed.Close();
}
```

```
            // Test error handling
            Console.WriteLine("\n--- Testing error handling ---");
            try
            {
                manager.CreateDocument("powerpoint", "Presentation");
            }
            catch (ArgumentException ex)
            {
                Console.WriteLine($"Expected error caught: {ex.Message}");
            }

        }
        catch (Exception ex)
        {
            Console.WriteLine($"Unexpected error: {ex.Message}");
        }

        Console.WriteLine("\nFactory Method Pattern test completed successfully!");
    }
  }
}
```

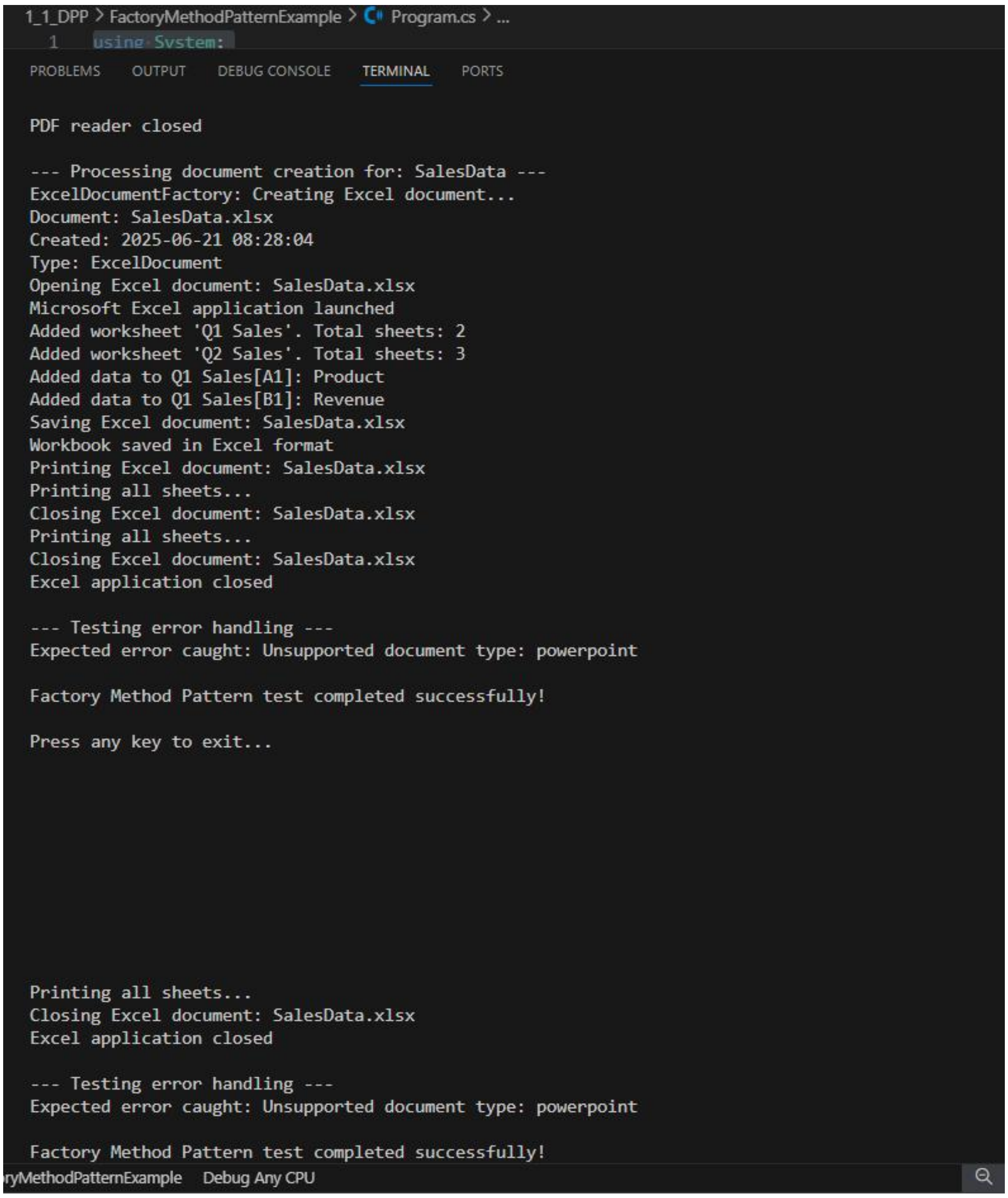
11. Program.cs Code :

```
using System;

namespace FactoryMethodPatternExample
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Factory Method Pattern Example in C#");
            Console.WriteLine("====================================\n");

            // Run the factory method tests
            FactoryMethodTest.TestFactoryMethod();

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

**OUTPUT** **:**

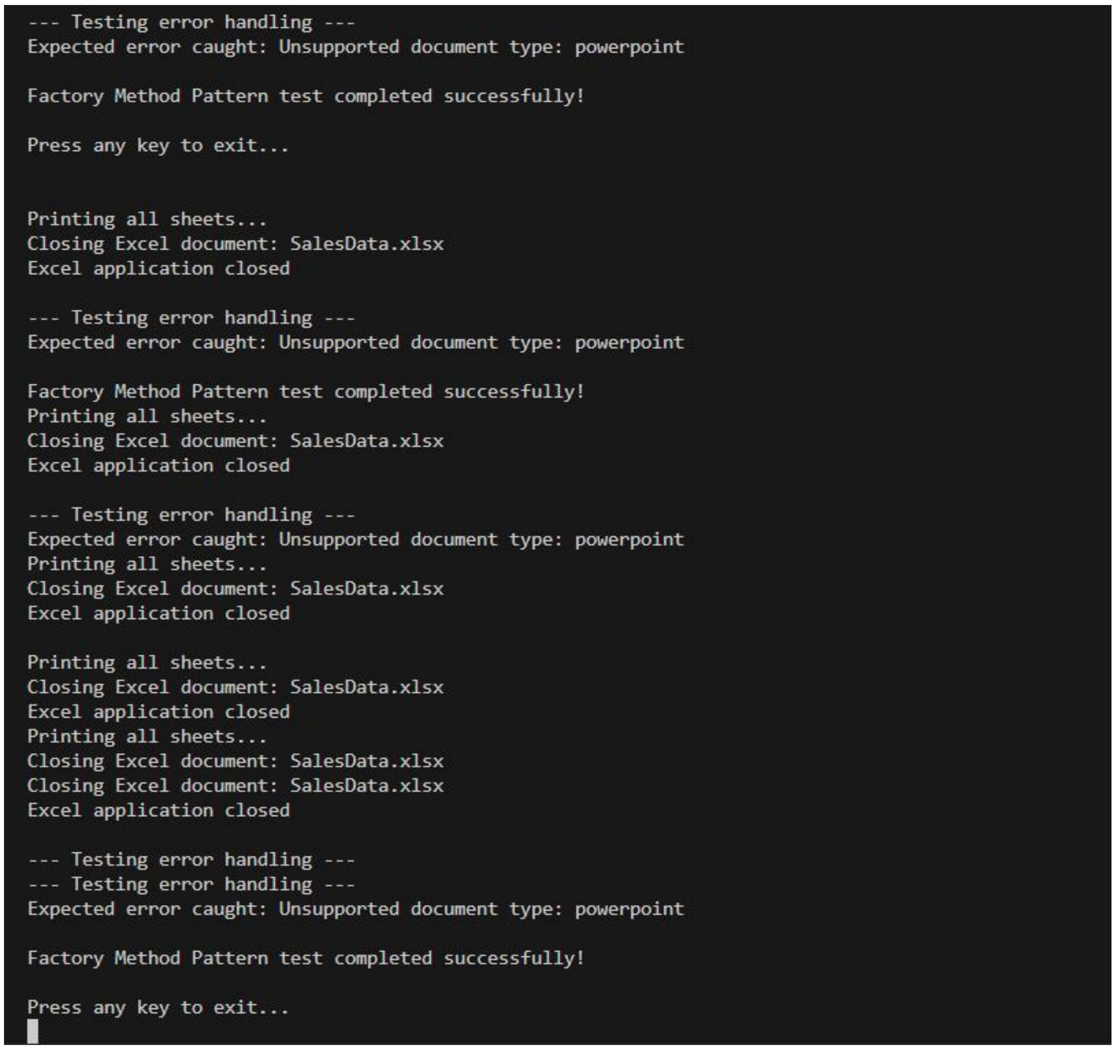


```
PS C:\Users\KIIT\OneDrive\Desktop\Cognizant_DeepSkilling_.NET_solutions\1_1_DPP\FactoryMethodPatternExample> dotnet run
Factory Method Pattern Example in C#
====================================

=== Testing Factory Method Pattern ===

Supported document types:
- word
- pdf
- excel

Creating various documents using Factory Method Pattern:


--- Processing document creation for: ProjectReport ---
WordDocumentFactory: Creating Word document...
Document: ProjectReport.docx
Created: 2025-06-21 08:28:04
Type: WordDocument
Opening Word document: ProjectReport.docx
Microsoft Word application launched
Added text to document. Current word count: 9
Saving Word document: ProjectReport.docx
Document saved in Word format
Printing Word document: ProjectReport.docx
Sending to default printer...
Closing Word document: ProjectReport.docx
Word application closed

--- Processing document creation for: UserManual ---
PdfDocumentFactory: Creating PDF document...
Document: UserManual.pdf
Created: 2025-06-21 08:28:04
Type: PdfDocument
Opening PDF document: UserManual.pdf
PDF reader application launched
Added new page. Current page count: 2
Added new page. Current page count: 3
Password protection enabled
Saving PDF document: UserManual.pdf
Document saved in PDF format
Printing PDF document: UserManual.pdf
High-quality PDF printing initiated...
Closing PDF document: UserManual.pdf
PDF reader closed
```



```
PDF reader closed

--- Processing document creation for: SalesData ---
ExcelDocumentFactory: Creating Excel document...
Document: SalesData.xlsx
Created: 2025-06-21 08:28:04
Type: ExcelDocument
Opening Excel document: SalesData.xlsx
Microsoft Excel application launched
Added worksheet 'Q1 Sales'. Total sheets: 2
Added worksheet 'Q2 Sales'. Total sheets: 3
Added data to Q1 Sales[A1]: Product
Added data to Q1 Sales[B1]: Revenue
Saving Excel document: SalesData.xlsx
Workbook saved in Excel format
Printing Excel document: SalesData.xlsx
Printing all sheets...
Closing Excel document: SalesData.xlsx
Printing all sheets...
Closing Excel document: SalesData.xlsx
Excel application closed

--- Testing error handling ---
Expected error caught: Unsupported document type: powerpoint

Factory Method Pattern test completed successfully!

Press any key to exit...
```

```
Printing all sheets...
Closing Excel document: SalesData.xlsx
Excel application closed

--- Testing error handling ---
Expected error caught: Unsupported document type: powerpoint

Factory Method Pattern test completed successfully!
```



```
--- Testing error handling ---
Expected error caught: Unsupported document type: powerpoint

Factory Method Pattern test completed successfully!

Press any key to exit...


Printing all sheets...
Closing Excel document: SalesData.xlsx
Excel application closed

--- Testing error handling ---
Expected error caught: Unsupported document type: powerpoint

Factory Method Pattern test completed successfully!
Printing all sheets...
Closing Excel document: SalesData.xlsx
Excel application closed

--- Testing error handling ---
Expected error caught: Unsupported document type: powerpoint
Printing all sheets...
Closing Excel document: SalesData.xlsx
Excel application closed

Printing all sheets...
Closing Excel document: SalesData.xlsx
Excel application closed
Printing all sheets...
Closing Excel document: SalesData.xlsx
Closing Excel document: SalesData.xlsx
Excel application closed

--- Testing error handling ---
--- Testing error handling ---
Expected error caught: Unsupported document type: powerpoint

Factory Method Pattern test completed successfully!

Press any key to exit...
```

---