**WEEK – 1 (Mandatory Hands On)**

**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 1:**

package Singleton;

public class Logger {

private static Logger *instance* =new Logger();

private Logger() {

}

public static Logger getInstance(){

return *instance*;}

public static void main(String args[]) {

Logger logger1=Logger.*getInstance*();

Logger logger2=Logger.*getInstance*();

if(logger1==logger2) {

System.***out***.println("Singleton Patter Created");

}

else {

System.***out***.println("Not a Singleton Pattern");

}

}

}

**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 2**

**FactoryMethodPatternExample.java**

package FactoryPattern;

public class FactoryMethodPatternExample {

public static void main(String[] args) {

FactoryPattern factoryPattern=new FactoryPattern();

DocumentFactory word =factoryPattern.Document("WordDocument");

word.createDocument();

DocumentFactory pdf =factoryPattern.Document("PdfDocument");

pdf.createDocument();

DocumentFactory excel =factoryPattern.Document("ExcelDocument");

excel.createDocument();

}

}

**FactoryPattern.java**

package FactoryPattern;

public class FactoryPattern {

public DocumentFactory Document(String type) {

if(type.equalsIgnoreCase("WordDocument")) {

return new WordDocument();

}

else if(type.equalsIgnoreCase("PdfDocument")) {

return new PdfDocument();

}

else if(type.equalsIgnoreCase("ExcelDocument")) {

return new ExcelDocument();

}

return null;

}

}

**DocumentFactory.java**

package FactoryPattern;

public interface DocumentFactory {

public void createDocument();

}

**WordDocument.java**

package FactoryPattern;

public class WordDocument implements DocumentFactory {

*@Override*

public void createDocument() {

System.***out***.println("This is a word Document");

}}

**ExcelDocument.java**

package FactoryPattern;

public class ExcelDocument implements DocumentFactory {

*@Override*

public void createDocument() {

System.***out***.println("This is a Excel Document");

}

}

**PdfDocument.java**

package FactoryPattern;

public class PdfDocument implements DocumentFactory {

*@Override*

public void createDocument() {

System.***out***.println("This is a PDF Document");

}

}