Design principles & Patterns week-1

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.

Program:

Singleton Logger Class

```
public class Logger {
    private static Logger instance;
 private Logger() {
        System.out.println("Logger instance created!");
    }
public static synchronized Logger getInstance() {
        if (instance == null) {
            instance = new Logger();
        }
        return instance;
    }
 public void log(String message) {
        System.out.println("[LOG] " + getCurrentTimestamp() + ": " + message);
    }
 public void logError(String errorMessage) {
        System.err.println("[ERROR] " + getCurrentTimestamp() + ": " + errorMessage);
```

```
    }
public void logWarning(String warningMessage) {
        System.out.println("[WARNING] " + getCurrentTimestamp() + ": " + warningMessage);
    }
 private String getCurrentTimestamp() {
        return java.time.LocalDateTime.now().toString();
    }
 public String getInstanceInfo() {
        return "Logger instance hash: " + this.hashCode();
    }
}
```

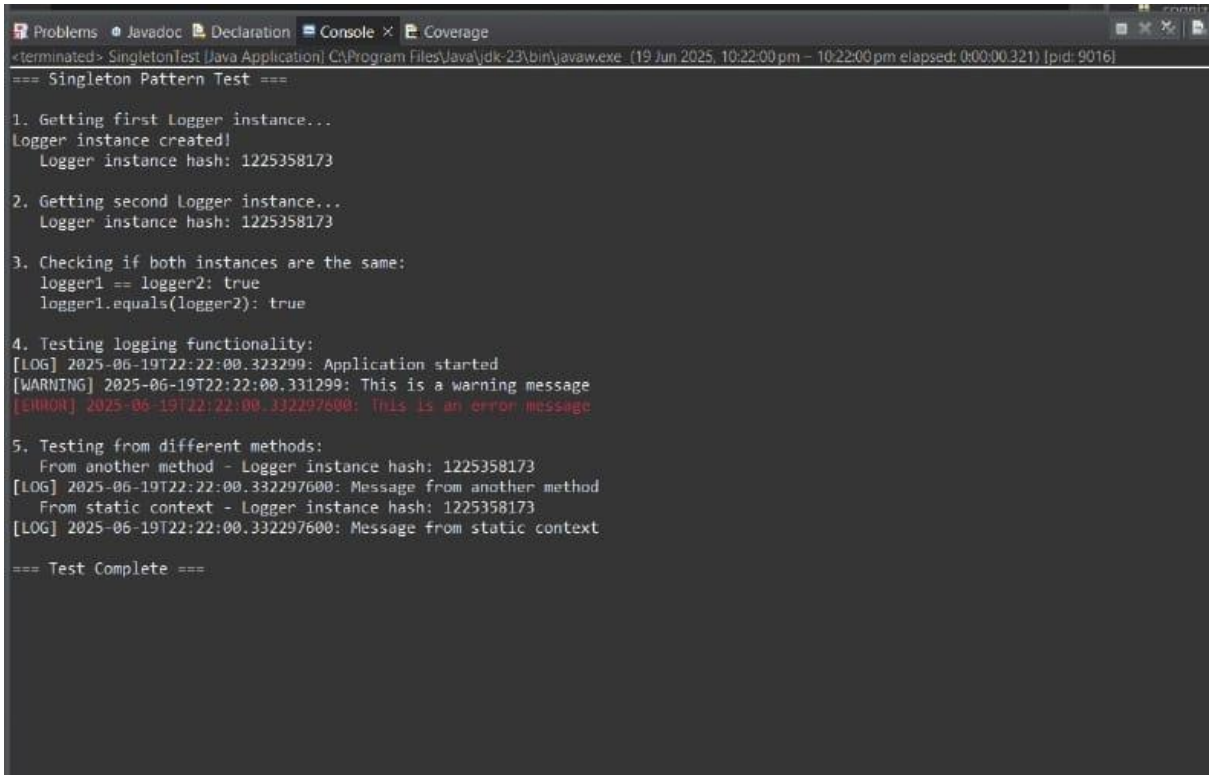
Singleton Pattern Test Class for Logger

```
public class SingletonTest {
    public static void main(String[] args) {
        System.out.println("=== Singleton Pattern Test ===\n");
       System.out.println("1. Getting first Logger instance...");
        Logger logger1 = Logger.getInstance();
        System.out.println("    " + logger1.getInstanceInfo());
       System.out.println("\n2. Getting second Logger instance...");
        Logger logger2 = Logger.getInstance();
        System.out.println("    " + logger2.getInstanceInfo());
        System.out.println("\n3. Checking if both instances are the same:");
        System.out.println("    logger1 == logger2: " + (logger1 == logger2));
        System.out.println("    logger1.equals(logger2): " + logger1.equals(logger2));
        System.out.println("\n4. Testing logging functionality:");
        logger1.log("Application started");
        logger2.logWarning("This is a warning message");
        logger1.logError("This is an error message");

        System.out.println("\n5. Testing from different methods:");
```

```
        testFromAnotherMethod();
        testFromStaticContext();
        System.out.println("\n=== Test Complete ===");
    }
 public static void testFromAnotherMethod() {
        Logger logger = Logger.getInstance();
        System.out.println("    From another method - " + logger.getInstanceInfo());
        logger.log("Message from another method");
    }
public static void testFromStaticContext() {
        Logger logger = Logger.getInstance();
        System.out.println("    From static context - " + logger.getInstanceInfo());
        logger.log("Message from static context");
    }
}
```

Out put:



Design principles & Patterns week-1

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.

Program :

```
interface Document {
    void open();
    void save();
    void close();
    String getType();
}
class WordDocument implements Document {
    @Override
    public void open() {
        System.out.println("Opening Word document...");
```

```
    }
@Override
    public void save() {
        System.out.println("Saving Word document...");
    }
 @Override
    public void close() {
        System.out.println("Closing Word document...");
    }
@Override
    public String getType() {
        return "Word Document (.docx)";
    }
}
class PdfDocument implements Document {
    @Override
    public void open() {
        System.out.println("Opening PDF document...");
    }
@Override
    public void save() {
        System.out.println("Saving PDF document...");
    }
@Override
    public void close() {
        System.out.println("Closing PDF document...");
    }
 @Override
    public String getType() {
        return "PDF Document (.pdf)";
    }
```

```
}
class ExcelDocument implements Document {
    @Override
    public void open() {
        System.out.println("Opening Excel document...");
    }
@Override
    public void save() {
        System.out.println("Saving Excel document...");
    }
@Override
    public void close() {
        System.out.println("Closing Excel document...");
    }
@Override
    public String getType() {
        return "Excel Document (.xlsx)";
    }
}
abstract class DocumentFactory {
    public abstract Document createDocument();
public Document processDocument() {
        Document doc = createDocument();
        doc.open();
        return doc;
    }
}
class WordDocumentFactory extends DocumentFactory {
    @Override
    public Document createDocument() {
        return new WordDocument();
```

```
    }
}
class PdfDocumentFactory extends DocumentFactory {
    @Override
    public Document createDocument() {
        return new PdfDocument();
    }
}
class ExcelDocumentFactory extends DocumentFactory {
    @Override
    public Document createDocument() {
        return new ExcelDocument();
    }
}
class DocumentManager {
    public static DocumentFactory getFactory(String documentType) {
        switch (documentType.toLowerCase()) {
            case "word":
                return new WordDocumentFactory();
            case "pdf":
                return new PdfDocumentFactory();
            case "excel":
                return new ExcelDocumentFactory();
            default:
                throw new IllegalArgumentException("Unknown document type: " + documentType);
        }
    }
}
public class FactoryMethodPatternExample {
    public static void main(String[] args) {
        System.out.println("=== Factory Method Pattern Demo ===\n");
```

```
String[] documentTypes = {"word", "pdf", "excel"};


        for (String type : documentTypes) {
            System.out.println("Creating " + type.toUpperCase() + " document:");
            System.out.println("----------------------------------------");
 try {
                DocumentFactory factory = DocumentManager.getFactory(type);
                Document document = factory.processDocument();
                System.out.println("Document type: " + document.getType());
                document.save();
                document.close();
            } catch (IllegalArgumentException e) {
                System.err.println("Error: " + e.getMessage());
            }
System.out.println();
        }
        System.out.println("=== Direct Factory Usage ===");
        DocumentFactory wordFactory = new WordDocumentFactory();
        Document wordDoc = wordFactory.createDocument();
        System.out.println("Created: " + wordDoc.getType());
        DocumentFactory pdfFactory = new PdfDocumentFactory();
        Document pdfDoc = pdfFactory.createDocument();
        System.out.println("Created: " + pdfDoc.getType());
        DocumentFactory excelFactory = new ExcelDocumentFactory();
        Document excelDoc = excelFactory.createDocument();
        System.out.println("Created: " + excelDoc.getType());
        System.out.println("\n=== Error Handling Demo ===");
        try {
            DocumentFactory unknownFactory = DocumentManager.getFactory("powerpoint");
        } catch (IllegalArgumentException e) {
            System.err.println("Expected error: " + e.getMessage());
```

```
    }
  }
}
```

Out Put:

```
Creating WORD document:
---------------------------------------
Opening Word document...
Document type: Word Document (.docx)
Saving Word document...
Closing Word document...

Creating PDF document:
---------------------------------------
Opening PDF document...
Document type: PDF Document (.pdf)
Saving PDF document...
Closing PDF document...

Creating EXCEL document:
---------------------------------------
Opening Excel document...
Document type: Excel Document (.xlsx)
Saving Excel document...
Closing Excel document...

=== Direct Factory Usage ===
Created: Word Document (.docx)
Created: PDF Document (.pdf)
Created: Excel Document (.xlsx)
```