**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.

**Source Code:**

/\*\*

\* Singleton Pattern Implementation - Logger Class

\* This class demonstrates the Singleton design pattern ensuring only one instance of the Logger class exists throughout the application lifecycle.

\*/

class Logger {

// Private static instance of the Logger class

private static Logger instance;

// Private constructor to prevent external instantiation

private Logger() {

System.out.println("Logger instance created");

System.out.println("Singleton pattern using logger");

}

/\*\*

\* Public static method to get the single instance of Logger

\* Uses lazy initialization - instance is created only when first requested

\* @return the single instance of Logger

\*/

public static Logger getInstance() {

if (instance == null) {

instance = new Logger();

}

return instance;

}

/\*\*

\* Method to log messages

\* @param message the message to be logged

\*/

public void log(String message) {

System.out.println("LOGGER MESSAGE: " + message);

}

}

/\*\*

\* Test class to demonstrate and verify the Singleton pattern implementation

\*/

public class SingletonPatternExample {

public static void main(String[] args) {

System.out.println("=== Testing Singleton Pattern Implementation ===\n");

// Get first instance of Logger

System.out.println("Getting first Logger instance:");

Logger logger1 = Logger.getInstance();

logger1.log("First log message");

System.out.println();

// Get second instance of Logger

System.out.println("Getting second Logger instance:");

Logger logger2 = Logger.getInstance();

logger2.log("Second log message");

System.out.println();

// Verify that both references point to the same instance

System.out.println("=== Singleton Verification ===");

System.out.println("Are both logger instances the same object? " + (logger1 == logger2));

System.out.println("logger1 hashCode: " + logger1.hashCode());

System.out.println("logger2 hashCode: " + logger2.hashCode());

// Additional test with more instances

System.out.println("\n=== Additional Testing ===");

Logger logger3 = Logger.getInstance();

Logger logger4 = Logger.getInstance();

System.out.println("logger1 == logger3: " + (logger1 == logger3));

System.out.println("logger2 == logger4: " + (logger2 == logger4));

System.out.println("All instances are the same: " +

(logger1 == logger2 && logger2 == logger3 && logger3 == logger4));

// Test logging from different references

System.out.println("\n=== Testing Logging from Different References ===");

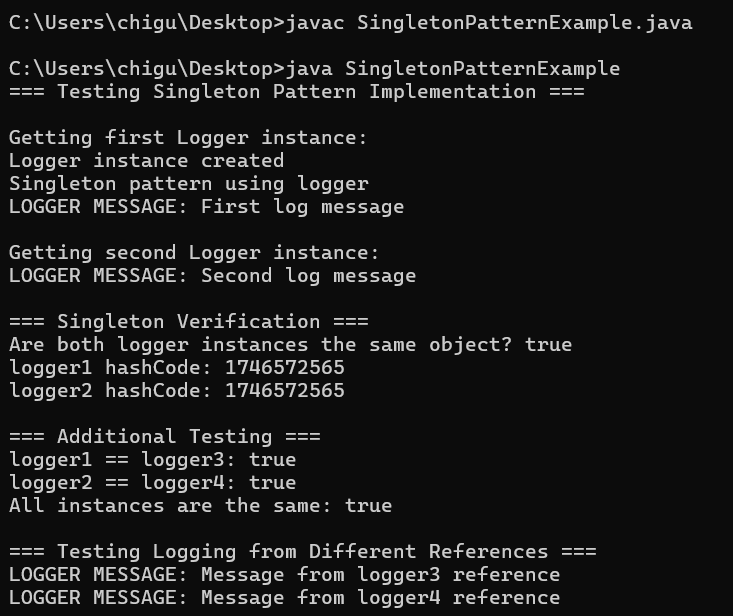
logger3.log("Message from logger3 reference");

logger4.log("Message from logger4 reference");

}

}

**Output:**



**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.

**Source Code:**

/\*\*

\* Factory Method Pattern Implementation - Document Management System

\* This demonstrates the Factory Method Pattern for creating different types of documents

\*/

// Step 1: Define the Document interface

interface Document {

void open();

void save();

void close();

String getType();

}

// Step 2: Create concrete document classes implementing the Document interface

/\*\*

\* Concrete implementation for Word Document

\*/

class WordDocument implements Document {

public WordDocument() {

System.out.println("WordDocument: Creating a new Word document");

}

@Override

public void open() {

System.out.println("WordDocument: Opening Word document with Microsoft Word");

}

@Override

public void save() {

System.out.println("WordDocument: Saving Word document as .docx file");

}

@Override

public void close() {

System.out.println("WordDocument: Closing Word document");

}

@Override

public String getType() {

return "Word Document (.docx)";

}

}

/\*\*

\* Concrete implementation for PDF Document

\*/

class PdfDocument implements Document {

public PdfDocument() {

System.out.println("PdfDocument: Creating a new PDF document");

}

@Override

public void open() {

System.out.println("PdfDocument: Opening PDF document with PDF reader");

}

@Override

public void save() {

System.out.println("PdfDocument: Saving PDF document as .pdf file");

}

@Override

public void close() {

System.out.println("PdfDocument: Closing PDF document");

}

@Override

public String getType() {

return "PDF Document (.pdf)";

}

}

/\*\*

\* Concrete implementation for Excel Document

\*/

class ExcelDocument implements Document {

public ExcelDocument() {

System.out.println("ExcelDocument: Creating a new Excel document");

}

@Override

public void open() {

System.out.println("ExcelDocument: Opening Excel document with Microsoft Excel");

}

@Override

public void save() {

System.out.println("ExcelDocument: Saving Excel document as .xlsx file");

}

@Override

public void close() {

System.out.println("ExcelDocument: Closing Excel document");

}

@Override

public String getType() {

return "Excel Document (.xlsx)";

}

}

// Step 3: Create the abstract DocumentFactory class

/\*\*

\* Abstract factory class that defines the factory method

\*/

abstract class DocumentFactory {

/\*\*

\* Factory method to be implemented by concrete factories

\* @return Document instance

\*/

public abstract Document createDocument();

/\*\*

\* Template method that uses the factory method

\* @return Document instance after performing common operations

\*/

public Document getDocument() {

System.out.println("DocumentFactory: Preparing to create document...");

Document document = createDocument();

System.out.println("DocumentFactory: Document created successfully - " + document.getType());

return document;

}

}

// Step 4: Create concrete factory classes

/\*\*

\* Concrete factory for creating Word documents

\*/

class WordDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new WordDocument();

}

}

/\*\*

\* Concrete factory for creating PDF documents

\*/

class PdfDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new PdfDocument();

}

}

/\*\*

\* Concrete factory for creating Excel documents

\*/

class ExcelDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new ExcelDocument();

}

}

// Step 5: Utility class for getting appropriate factory based on document type

class DocumentFactoryProvider {

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);

}

}

}

// Step 6: Test class demonstrating the Factory Method Pattern

public class FactoryMethodPatternExample {

public static void main(String[] args) {

System.out.println("=== Factory Method Pattern - Document Management System ===\n");

try {

// Test creating different types of documents

System.out.println("1. Creating Word Document:");

System.out.println("---------------------------");

DocumentFactory wordFactory = DocumentFactoryProvider.getFactory("word");

Document wordDoc = wordFactory.getDocument();

demonstrateDocumentOperations(wordDoc);

System.out.println("\n2. Creating PDF Document:");

System.out.println("---------------------------");

DocumentFactory pdfFactory = DocumentFactoryProvider.getFactory("pdf");

Document pdfDoc = pdfFactory.getDocument();

demonstrateDocumentOperations(pdfDoc);

System.out.println("\n3. Creating Excel Document:");

System.out.println("----------------------------");

DocumentFactory excelFactory = DocumentFactoryProvider.getFactory("excel");

Document excelDoc = excelFactory.getDocument();

demonstrateDocumentOperations(excelDoc);

// Demonstrate direct factory usage

System.out.println("\n4. Direct Factory Usage:");

System.out.println("-------------------------");

DocumentFactory directWordFactory = new WordDocumentFactory();

Document directWordDoc = directWordFactory.createDocument();

System.out.println("Direct creation result: " + directWordDoc.getType());

// Test error handling

System.out.println("\n5. Error Handling Test:");

System.out.println("-----------------------");

try {

DocumentFactory invalidFactory = DocumentFactoryProvider.getFactory("powerpoint");

} catch (IllegalArgumentException e) {

System.out.println("Caught expected error: " + e.getMessage());

}

// Demonstrate polymorphism

System.out.println("\n6. Polymorphism Demonstration:");

System.out.println("-------------------------------");

Document[] documents = {

new WordDocumentFactory().createDocument(),

new PdfDocumentFactory().createDocument(),

new ExcelDocumentFactory().createDocument()

};

for (Document doc : documents) {

System.out.println("Processing: " + doc.getType());

doc.open();

}

} catch (Exception e) {

System.err.println("An error occurred: " + e.getMessage());

}

}

/\*\*

\* Helper method to demonstrate document operations

\* @param document Document instance to operate on

\*/

private static void demonstrateDocumentOperations(Document document) {

document.open();

document.save();

document.close();

System.out.println("Document type: " + document.getType());

}

}

**Output:**

