## Exercise 1: Implementing the Singleton Pattern

### Steps

#### Create a New Java Project

#### ****SingletonPatternExample****

#### 2. Define a Singleton Class

Develop a class named EventLogger with these requirements:

* Private static instance variable
* Private constructor to block external instantiation
* Public static accessor method

#### 3. Implement the Singleton Pattern

**EventLogger.java**

import java.text.SimpleDateFormat;

import java.util.Date;

public class EventLogger {

private static EventLogger uniqueInstance;

private int messageCounter;

// Private constructor prevents instantiation from outside

private EventLogger() {

messageCounter = 0;

System.out.println("EventLogger initialized - Ready for logging operations");

// Simulate initialization delay

try {

Thread.sleep(50);

} catch (InterruptedException ex) {

Thread.currentThread().interrupt();

System.err.println("Logger initialization interrupted: " + ex.getMessage());

}

}

// Thread-safe singleton instance retrieval

public static synchronized EventLogger getLoggerInstance() {

if (uniqueInstance == null) {

uniqueInstance = new EventLogger();

}

return uniqueInstance;

}

// Log message with timestamp and counter

public synchronized void recordEvent(String eventMessage) {

messageCounter++;

SimpleDateFormat formatter = new SimpleDateFormat("HH:mm:ss.SSS");

String timestamp = formatter.format(new Date());

System.out.println("[" + timestamp + "] Event #" + messageCounter + ": " + eventMessage);

}

// Get total number of logged messages

public int getEventCount() {

return messageCounter;

}

}

#### 4. Test the Singleton Implementation

**SingletonTest.java**

public class SingletonTest {

public static void main(String[] args) {

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

System.out.println("Objective: Validate single instance creation for EventLogger");

// Test 1: Basic singleton verification

System.out.println("\n--- Test 1: Instance Identity Check ---");

EventLogger logger1 = EventLogger.getLoggerInstance();

logger1.recordEvent("Application started successfully");

EventLogger logger2 = EventLogger.getLoggerInstance();

logger2.recordEvent("Configuration loaded");

System.out.println("Logger1 hash: " + logger1.hashCode());

System.out.println("Logger2 hash: " + logger2.hashCode());

if (logger1 == logger2) {

System.out.println("✓ SUCCESS: Both references point to same instance");

} else {

System.out.println("✗ FAILED: Multiple instances detected");

}

// Test 2: Concurrent access test

System.out.println("\n--- Test 2: Multi-threaded Access ---");

Thread worker1 = new Thread(() -> {

EventLogger logger = EventLogger.getLoggerInstance();

logger.recordEvent("Worker-1 task completed");

}, "WorkerThread-1");

Thread worker2 = new Thread(() -> {

EventLogger logger = EventLogger.getLoggerInstance();

logger.recordEvent("Worker-2 processing data");

}, "WorkerThread-2");

Thread worker3 = new Thread(() -> {

EventLogger logger = EventLogger.getLoggerInstance();

logger.recordEvent("Worker-3 finished cleanup");

}, "WorkerThread-3");

worker1.start();

worker2.start();

worker3.start();

try {

worker1.join();

worker2.join();

worker3.join();

System.out.println("All worker threads completed");

} catch (InterruptedException e) {

System.err.println("Thread synchronization error: " + e.getMessage());

}

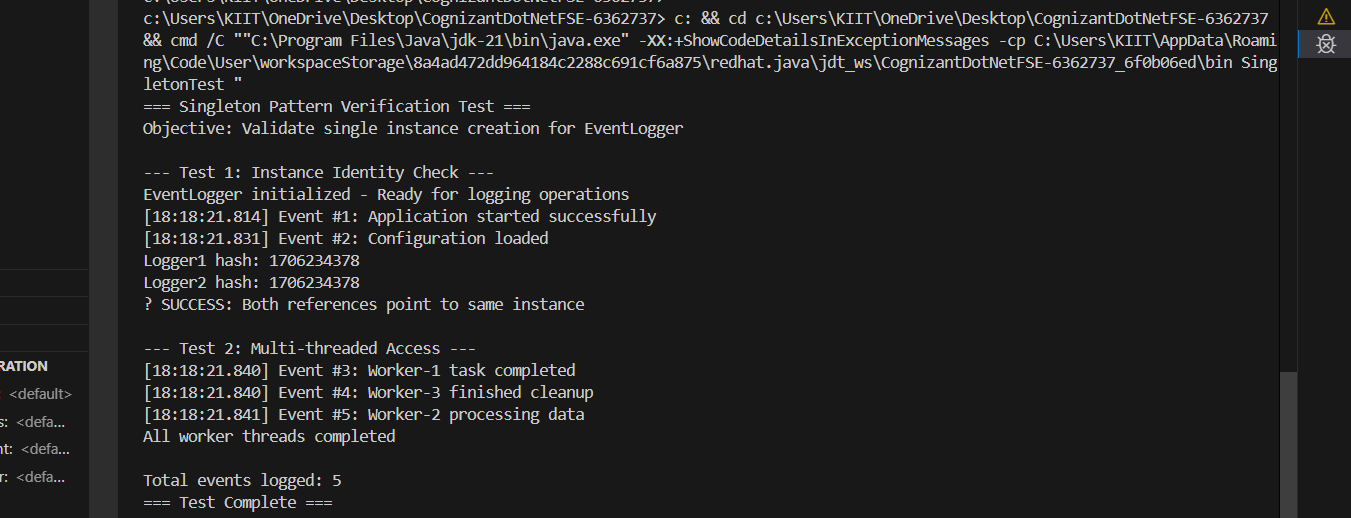
System.out.println("\nTotal events logged: " + logger1.getEventCount());

System.out.println("=== Test Complete ===");

}

}

**OUTPUT:**



**Exercise 2: Implementing the Factory Method Pattern**

### Steps

#### 1. Create a New Java Project

**FactoryMethodPatternExample**

#### 2. Define Document Interface

// Common interface for all document types

interface FileDocument {

void initialize();

void processContent();

void finalize();

String getDocumentType();

}

#### 3. Create Concrete Document Classes

**TextDocument.java**

// Implementation for text-based documents

class TextDocument implements FileDocument {

private String content;

public TextDocument() {

this.content = "";

}

public void initialize() {

System.out.println("→ TextDocument: Preparing text editor environment");

}

public void processContent() {

System.out.println("→ TextDocument: Processing text formatting and content");

content = "Sample text content processed";

}

public void finalize() {

System.out.println("→ TextDocument: Saving to .txt format and closing editor");

}

public String getDocumentType() {

return "Text Document (.txt)";

}

}

**PresentationDocument.java**

// Implementation for presentation documents

class PresentationDocument implements FileDocument {

private int slideCount;

public PresentationDocument() {

this.slideCount = 0;

}

public void initialize() {

System.out.println("→ PresentationDocument: Loading presentation software");

}

public void processContent() {

System.out.println("→ PresentationDocument: Creating slides and animations");

slideCount = 5;

}

public void finalize() {

System.out.println("→ PresentationDocument: Exporting to .pptx and closing application");

}

public String getDocumentType() {

return "Presentation Document (.pptx) - " + slideCount + " slides";

}

}

**SpreadsheetDocument.java**

// Implementation for spreadsheet documents

class SpreadsheetDocument implements FileDocument {

private String[][] dataGrid;

public SpreadsheetDocument() {

this.dataGrid = new String[10][5]; // 10 rows, 5 columns

}

public void initialize() {

System.out.println("→ SpreadsheetDocument: Starting spreadsheet application");

}

public void processContent() {

System.out.println("→ SpreadsheetDocument: Calculating formulas and organizing data");

// Simulate data processing

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

dataGrid[i][j] = "Cell[" + i + "," + j + "]";

}

}

}

public void finalize() {

System.out.println("→ SpreadsheetDocument: Saving workbook to .xlsx format");

}

public String getDocumentType() {

return "Spreadsheet Document (.xlsx)";

}

}

#### 4. Implement the Factory Method

**Document Creator Base Class**

// Abstract factory for creating documents

abstract class DocumentCreator {

// Factory method to be implemented by subclasses

public abstract FileDocument buildDocument();

// Template method that uses the factory method

public void executeDocumentWorkflow() {

System.out.println("\n\*\*\* Starting Document Workflow \*\*\*");

FileDocument doc = buildDocument();

System.out.println("Document Type: " + doc.getDocumentType());

doc.initialize();

doc.processContent();

doc.finalize();

System.out.println("\*\*\* Workflow Completed \*\*\*");

}

}

**Concrete Factory Classes**

**TextDocumentCreator.java**

// Factory for creating text documents

class TextDocumentCreator extends DocumentCreator {

public FileDocument buildDocument() {

System.out.println("Factory: Constructing new TextDocument instance");

return new TextDocument();

}

}

**PresentationDocumentCreator.java**

// Factory for creating presentation documents

class PresentationDocumentCreator extends DocumentCreator {

public FileDocument buildDocument() {

System.out.println("Factory: Constructing new PresentationDocument instance");

return new PresentationDocument();

}

}

**SpreadsheetDocumentCreator.java**

// Factory for creating spreadsheet documents

class SpreadsheetDocumentCreator extends DocumentCreator {

public FileDocument buildDocument() {

System.out.println("Factory: Constructing new SpreadsheetDocument instance");

return new SpreadsheetDocument();

}

}

#### 5. Test the Factory Method Implementation

**FactoryMethodTest.java**

// Test class demonstrating factory method pattern

public class FactoryMethodTest {

public static void main(String[] args) {

System.out.println("=== Factory Method Pattern Demonstration ===");

System.out.println("Purpose: Show flexible document creation using factory pattern");

// Test Case 1: Direct document creation

System.out.println("\n--- Direct Document Creation Test ---");

DocumentCreator textFactory = new TextDocumentCreator();

FileDocument textDoc = textFactory.buildDocument();

demonstrateDocument(textDoc);

DocumentCreator presentationFactory = new PresentationDocumentCreator();

FileDocument slideDoc = presentationFactory.buildDocument();

demonstrateDocument(slideDoc);

DocumentCreator spreadsheetFactory = new SpreadsheetDocumentCreator();

FileDocument calcDoc = spreadsheetFactory.buildDocument();

demonstrateDocument(calcDoc);

// Test Case 2: Using template method workflow

System.out.println("\n--- Template Method Workflow Test ---");

System.out.println("Processing text document workflow:");

textFactory.executeDocumentWorkflow();

System.out.println("\nProcessing presentation document workflow:");

presentationFactory.executeDocumentWorkflow();

System.out.println("\n--- Demonstration Complete ---");

}

// Helper method to demonstrate document operations

private static void demonstrateDocument(FileDocument document) {

System.out.println("Demonstrating: " + document.getDocumentType());

document.initialize();

document.processContent();

document.finalize();

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

}

}

Output:

