**Exercise 1: Implementing the Singleton Pattern**

**Logger.java**

package com.example.singleton;

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

public class Logger {

    private static Logger instance;

    private Logger() {

        System.out.println("Logger instance created at: " + getCurrentTimestamp());

    }

    public static synchronized Logger getInstance() {

        if (instance == null) {

            instance = new Logger();

        }

        return instance;

    }

    public void log(String message) {

        System.out.println("[" + getCurrentTimestamp() + "] LOG: " + message);

    }

    public void logError(String errorMessage) {

        System.err.println("[" + getCurrentTimestamp() + "] ERROR: " + errorMessage);

    }

    public void logWarning(String warningMessage) {

        System.out.println("[" + getCurrentTimestamp() + "] WARNING: " + warningMessage);

    }

    public void logInfo(String infoMessage) {

        System.out.println("[" + getCurrentTimestamp() + "] INFO: " + infoMessage);

    }

    private String getCurrentTimestamp() {

        return LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss"));

    }

    public String getInstanceDetails() {

        return "Logger instance: " + this.hashCode();

    }

}

**SingletonTest.java**

package com.example.singleton;

public class SingletonTest {

    public static void main(String[] args) {

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

        testSingletonBehavior();

        System.out.println("\n" + "=".repeat(50) + "\n");

        testLoggerFunctionality();

        System.out.println("\n" + "=".repeat(50) + "\n");

        testFromDifferentMethods();

    }

    private static void testSingletonBehavior() {

        System.out.println("Test 1: Verifying Singleton Behavior");

        System.out.println("-".repeat(40));

        Logger logger1 = Logger.getInstance();

        System.out.println("First instance: " + logger1.getInstanceDetails());

        Logger logger2 = Logger.getInstance();

        System.out.println("Second instance: " + logger2.getInstanceDetails());

        Logger logger3 = Logger.getInstance();

        System.out.println("Third instance: " + logger3.getInstanceDetails());

        if (logger1 == logger2 && logger2 == logger3) {

            System.out.println("✓ SUCCESS: All instances are the same object");

            System.out.println("✓ Singleton pattern is working correctly");

        } else {

            System.out.println("✗ FAILURE: Instances are different objects");

        }

        System.out.println("\nHash Code Verification:");

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

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

        System.out.println("logger3 hash: " + logger3.hashCode());

    }

    private static void testLoggerFunctionality() {

        System.out.println("Test 2: Testing Logger Functionality");

        System.out.println("-".repeat(40));

        Logger logger = Logger.getInstance();

        logger.log("Application started successfully");

        logger.logInfo("User login attempt");

        logger.logWarning("Memory usage is high");

        logger.logError("Database connection failed");

        logger.log("Application shutting down");

    }

    private static void testFromDifferentMethods() {

        System.out.println("Test 3: Testing from Different Methods");

        System.out.println("-".repeat(40));

        Logger loggerFromMethod1 = getLoggerFromMethod1();

        Logger loggerFromMethod2 = getLoggerFromMethod2();

        System.out.println("Logger from method 1: " + loggerFromMethod1.getInstanceDetails());

        System.out.println("Logger from method 2: " + loggerFromMethod2.getInstanceDetails());

        if (loggerFromMethod1 == loggerFromMethod2) {

            System.out.println("✓ SUCCESS: Same instance returned from different methods");

        } else {

            System.out.println("✗ FAILURE: Different instances returned from different methods");

        }

        loggerFromMethod1.log("Message from method 1 context");

        loggerFromMethod2.log("Message from method 2 context");

    }

    private static Logger getLoggerFromMethod1() {

        System.out.println("Getting logger instance from method 1...");

        return Logger.getInstance();

    }

    private static Logger getLoggerFromMethod2() {

        System.out.println("Getting logger instance from method 2...");

        return Logger.getInstance();

    }

}

**AdvancedLogger.java**

package com.example.singleton;

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

public class AdvancedLogger {

    private AdvancedLogger() {

        System.out.println("AdvancedLogger instance created at: " + getCurrentTimestamp());

    }

    private static class SingletonHelper {

        private static final AdvancedLogger INSTANCE = new AdvancedLogger();

    }

    public static AdvancedLogger getInstance() {

        return SingletonHelper.INSTANCE;

    }

    public void log(String message) {

        System.out.println("[" + getCurrentTimestamp() + "] ADVANCED LOG: " + message);

    }

    public void logError(String errorMessage) {

        System.err.println("[" + getCurrentTimestamp() + "] ADVANCED ERROR: " + errorMessage);

    }

    public void logWarning(String warningMessage) {

        System.out.println("[" + getCurrentTimestamp() + "] ADVANCED WARNING: " + warningMessage);

    }

    public void logInfo(String infoMessage) {

        System.out.println("[" + getCurrentTimestamp() + "] ADVANCED INFO: " + infoMessage);

    }

    private String getCurrentTimestamp() {

        return LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss"));

    }

    public String getInstanceDetails() {

        return "AdvancedLogger instance: " + this.hashCode();

    }

    @Override

    protected Object clone() throws CloneNotSupportedException {

        throw new CloneNotSupportedException("Cannot clone singleton instance");

    }

}

**SingletonPatternDemo.java**

package com.example.singleton;

public class SingletonPatternDemo {

public static void main(String[] args) {

System.out.println("SINGLETON PATTERN DEMONSTRATION");

System.out.println("=" .repeat(60));

demonstrateBasicSingleton();

System.out.println("\n" + "=" .repeat(60) + "\n");

demonstrateAdvancedSingleton();

System.out.println("\n" + "=" .repeat(60) + "\n");

demonstrateThreadSafety();

}

private static void demonstrateBasicSingleton() {

System.out.println("DEMO 1: Basic Singleton Pattern (Synchronized)");

System.out.println("-" .repeat(50));

Logger logger1 = Logger.getInstance();

Logger logger2 = Logger.getInstance();

Logger logger3 = Logger.getInstance();

System.out.println("Instance 1 Hash: " + logger1.hashCode());

System.out.println("Instance 2 Hash: " + logger2.hashCode());

System.out.println("Instance 3 Hash: " + logger3.hashCode());

System.out.println("Are all instances equal? " + (logger1 == logger2 && logger2 == logger3));

logger1.log("Testing basic singleton pattern");

logger2.logInfo("This message comes from the same instance");

logger3.logWarning("All loggers are actually the same object");

}

private static void demonstrateAdvancedSingleton() {

System.out.println("DEMO 2: Advanced Singleton Pattern (Bill Pugh)");

System.out.println("-" .repeat(50));

AdvancedLogger advLogger1 = AdvancedLogger.getInstance();

AdvancedLogger advLogger2 = AdvancedLogger.getInstance();

AdvancedLogger advLogger3 = AdvancedLogger.getInstance();

System.out.println("Advanced Instance 1 Hash: " + advLogger1.hashCode());

System.out.println("Advanced Instance 2 Hash: " + advLogger2.hashCode());

System.out.println("Advanced Instance 3 Hash: " + advLogger3.hashCode());

System.out.println("Are all advanced instances equal? " + (advLogger1 == advLogger2 && advLogger2 == advLogger3));

advLogger1.log("Testing advanced singleton pattern");

advLogger2.logInfo("This is more efficient and thread-safe");

advLogger3.logWarning("No synchronization overhead!");

}

private static void demonstrateThreadSafety() {

System.out.println("DEMO 3: Thread Safety Test");

System.out.println("-" .repeat(50));

Thread thread1 = new Thread(() -> {

Logger logger = Logger.getInstance();

logger.log("Message from Thread 1 - Hash: " + logger.hashCode());

});

Thread thread2 = new Thread(() -> {

Logger logger = Logger.getInstance();

logger.log("Message from Thread 2 - Hash: " + logger.hashCode());

});

Thread thread3 = new Thread(() -> {

AdvancedLogger logger = AdvancedLogger.getInstance();

logger.log("Message from Thread 3 (Advanced) - Hash: " + logger.hashCode());

});

Thread thread4 = new Thread(() -> {

AdvancedLogger logger = AdvancedLogger.getInstance();

logger.log("Message from Thread 4 (Advanced) - Hash: " + logger.hashCode());

});

thread1.start();

thread2.start();

thread3.start();

thread4.start();

try {

thread1.join();

thread2.join();

thread3.join();

thread4.join();

} catch (InterruptedException e) {

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

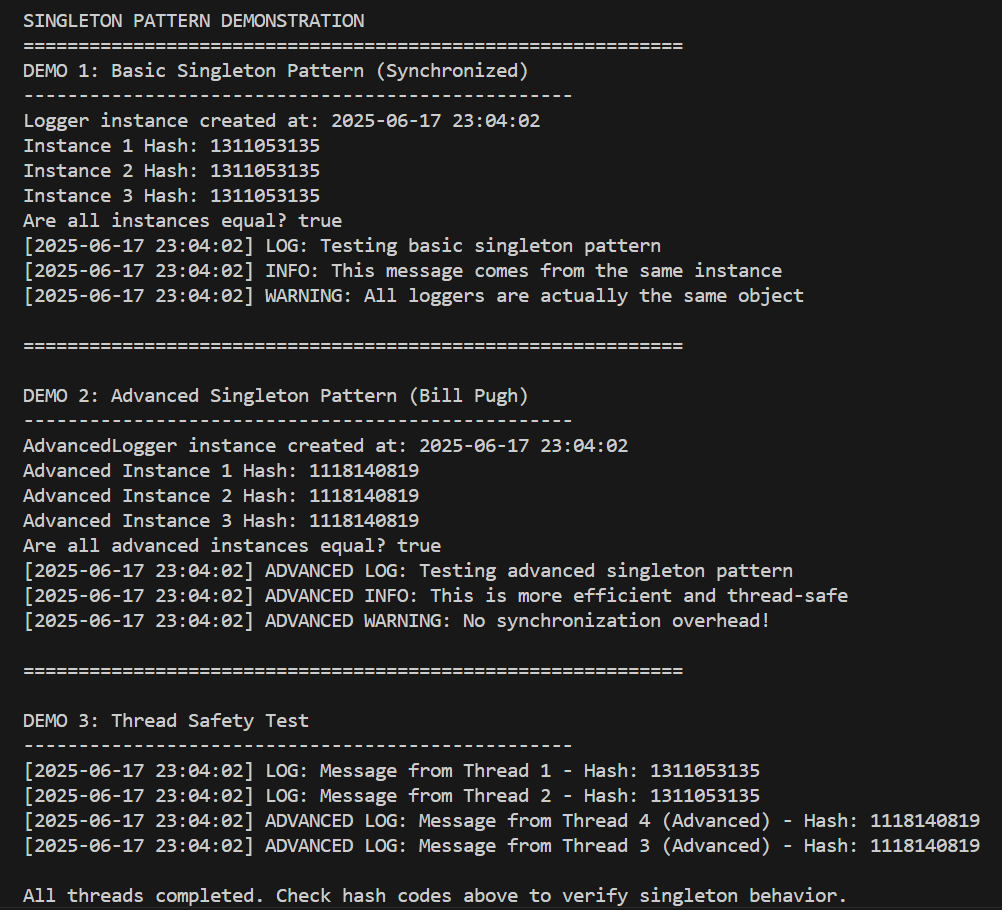
}

System.out.println("\nAll threads completed. Check hash codes above to verify singleton behavior.");

}

}

**Output Screenshot**



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

**Document.java**

package com.example.factory.documents;

import java.time.LocalDateTime;

public interface Document {

    void open();

    void save();

    void close();

    void print();

    String getDocumentType();

    LocalDateTime getCreationTime();

}

**ExcelDocument.java**

package com.example.factory.documents;

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

public class ExcelDocument implements Document {

    private LocalDateTime creationTime;

    private String fileName;

    public ExcelDocument() {

        this.creationTime = LocalDateTime.now();

        this.fileName = "Spreadsheet\_" + creationTime.format(DateTimeFormatter.ofPattern("yyyyMMdd\_HHmmss")) + ".xlsx";

    }

    @Override

    public void open() {

        System.out.println("Opening Excel document: " + fileName);

        System.out.println("Loading Microsoft Excel application...");

    }

    @Override

    public void save() {

        System.out.println("Saving Excel document: " + fileName);

        System.out.println("Workbook saved in .xlsx format");

    }

    @Override

    public void close() {

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

        System.out.println("Microsoft Excel application closed");

    }

    @Override

    public void print() {

        System.out.println("Printing Excel document: " + fileName);

        System.out.println("Printing all worksheets...");

    }

    @Override

    public String getDocumentType() {

        return "Microsoft Excel Spreadsheet";

    }

    @Override

    public LocalDateTime getCreationTime() {

        return creationTime;

    }

}

**PdfDocument.java**

package com.example.factory.documents;

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

public class PdfDocument implements Document {

    private LocalDateTime creationTime;

    private String fileName;

    public PdfDocument() {

        this.creationTime = LocalDateTime.now();

        this.fileName = "Document\_" + creationTime.format(DateTimeFormatter.ofPattern("yyyyMMdd\_HHmmss")) + ".pdf";

    }

    @Override

    public void open() {

        System.out.println("Opening PDF document: " + fileName);

        System.out.println("Loading PDF reader application...");

    }

    @Override

    public void save() {

        System.out.println("Saving PDF document: " + fileName);

        System.out.println("Document saved in .pdf format");

    }

    @Override

    public void close() {

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

        System.out.println("PDF reader application closed");

    }

    @Override

    public void print() {

        System.out.println("Printing PDF document: " + fileName);

        System.out.println("High-quality print mode enabled...");

    }

    @Override

    public String getDocumentType() {

        return "Portable Document Format";

    }

    @Override

    public LocalDateTime getCreationTime() {

        return creationTime;

    }

}

**WordDocument.java**

package com.example.factory.documents;

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

public class WordDocument implements Document {

    private LocalDateTime creationTime;

    private String fileName;

    public WordDocument() {

        this.creationTime = LocalDateTime.now();

        this.fileName = "Document\_" + creationTime.format(DateTimeFormatter.ofPattern("yyyyMMdd\_HHmmss")) + ".docx";

    }

    @Override

    public void open() {

        System.out.println("Opening Word document: " + fileName);

        System.out.println("Loading Microsoft Word application...");

    }

    @Override

    public void save() {

        System.out.println("Saving Word document: " + fileName);

        System.out.println("Document saved in .docx format");

    }

    @Override

    public void close() {

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

        System.out.println("Microsoft Word application closed");

    }

    @Override

    public void print() {

        System.out.println("Printing Word document: " + fileName);

        System.out.println("Sending to default printer...");

    }

    @Override

    public String getDocumentType() {

        return "Microsoft Word Document";

    }

    @Override

    public LocalDateTime getCreationTime() {

        return creationTime;

    }

}

**DocumentFactory.java**

package com.example.factory.factories;

import com.example.factory.documents.Document;

public abstract class DocumentFactory{

    public abstract Document createDocument();

    public Document processDocument()

    {

        System.out.println("=== Document Processing Started ===");

        Document document = createDocument();

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

        System.out.println("Creation Time: " + document.getCreationTime());

        document.open();

        document.save();

        System.out.println("=== Document Processing Completed ===");

        return document;

    }

}

**ExcelDocumentFactory.java**

package com.example.factory.factories;

import com.example.factory.documents.Document;

import com.example.factory.documents.ExcelDocument;

public class ExcelDocumentFactory extends DocumentFactory {

    @Override

    public Document createDocument() {

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

        return new ExcelDocument();

    }

}

**PdfDocumentFactory.java**

package com.example.factory.factories;

import com.example.factory.documents.Document;

import com.example.factory.documents.PdfDocument;

public class PdfDocumentFactory extends DocumentFactory {

    @Override

    public Document createDocument() {

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

        return new PdfDocument();

    }

}

**WordDocumentFactory.java**

package com.example.factory.factories;

import com.example.factory.documents.Document;

import com.example.factory.documents.WordDocument;

public class WordDocumentFactory extends DocumentFactory {

    @Override

    public Document createDocument() {

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

        return new WordDocument();

    }

}

**DocumentFactoryRegistry.java**

package com.example.factory.factories;

import java.util.HashMap;

import java.util.Map;

public class DocumentFactoryRegistry {

    private static final Map<String, DocumentFactory> factories = new HashMap<>();

    static {

        factories.put("WORD", new WordDocumentFactory());

        factories.put("PDF", new PdfDocumentFactory());

        factories.put("EXCEL", new ExcelDocumentFactory());

    }

    public static DocumentFactory getFactory(String documentType) {

        DocumentFactory factory = factories.get(documentType.toUpperCase());

        if (factory == null) {

            throw new IllegalArgumentException("Unknown document type: " + documentType);

        }

        return factory;

    }

    public static String[] getSupportedTypes() {

        return factories.keySet().toArray(new String[0]);

    }

}

**DocumentFactoryTest.java**

package com.example.factory.test;

import com.example.factory.documents.Document;

import com.example.factory.factories.\*;

public class DocumentFactoryTest {

    public static void main(String[] args) {

        System.out.println("=== Factory Method Pattern Demo ===\n");

        testBasicFactoryUsage();

        System.out.println("\n" + "=".repeat(50) + "\n");

        testFactoryRegistry();

        System.out.println("\n" + "=".repeat(50) + "\n");

        testDocumentOperations();

    }

    private static void testBasicFactoryUsage() {

        System.out.println("1. Basic Factory Usage:");

        DocumentFactory wordFactory = new WordDocumentFactory();

        Document wordDoc = wordFactory.processDocument();

        wordDoc.close();

        System.out.println();

        DocumentFactory pdfFactory = new PdfDocumentFactory();

        Document pdfDoc = pdfFactory.processDocument();

        pdfDoc.close();

        System.out.println();

        DocumentFactory excelFactory = new ExcelDocumentFactory();

        Document excelDoc = excelFactory.processDocument();

        excelDoc.close();

    }

    private static void testFactoryRegistry() {

        System.out.println("2. Factory Registry Usage:");

        System.out.println("Supported document types: " + String.join(", ", DocumentFactoryRegistry.getSupportedTypes()));

        String[] documentTypes = {"WORD", "PDF", "EXCEL"};

        for (String type : documentTypes) {

            try {

                DocumentFactory factory = DocumentFactoryRegistry.getFactory(type);

                Document doc = factory.createDocument();

                doc.open();

                doc.save();

                doc.close();

                System.out.println();

            } catch (IllegalArgumentException e) {

                System.err.println("Error: " + e.getMessage());

            }

        }

    }

    private static void testDocumentOperations() {

        System.out.println("3. Full Document Operations:");

        DocumentFactory[] factories = {

            new WordDocumentFactory(),

            new PdfDocumentFactory(),

            new ExcelDocumentFactory()

        };

        for (DocumentFactory factory : factories) {

            Document doc = factory.createDocument();

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

            doc.open();

            doc.save();

            doc.print();

            doc.close();

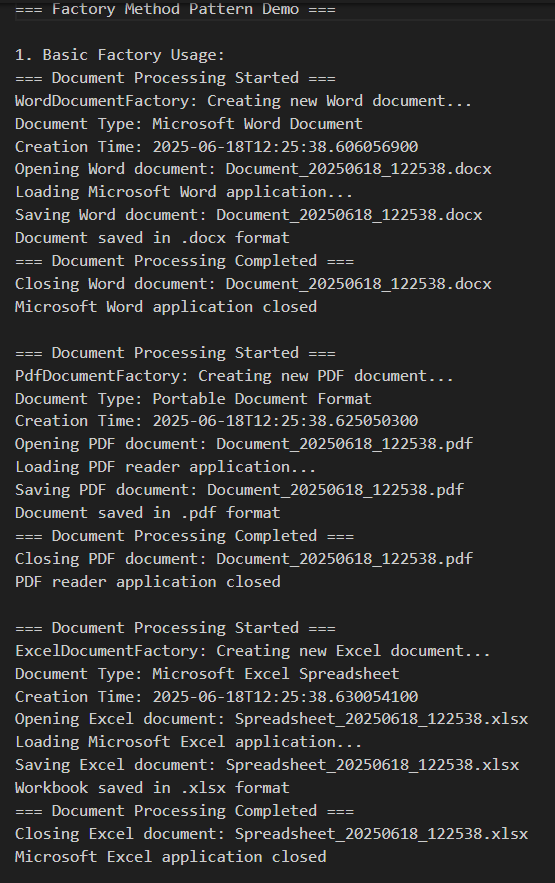
            System.out.println("-".repeat(30));

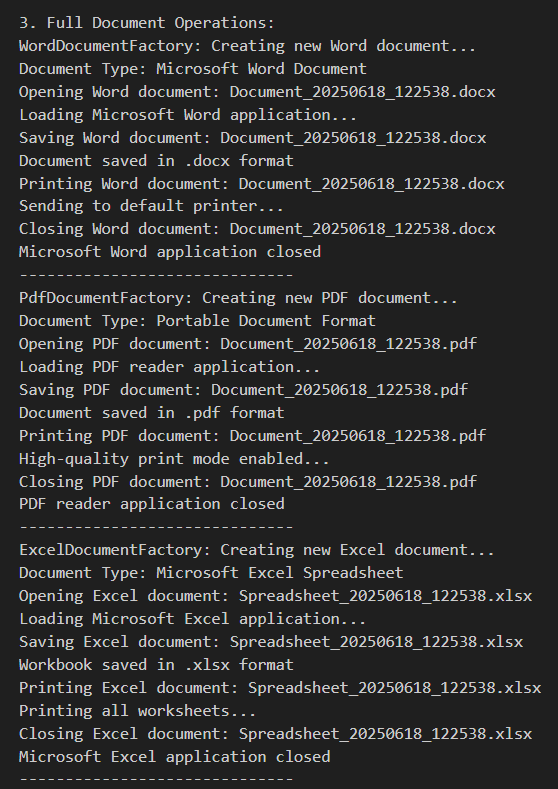
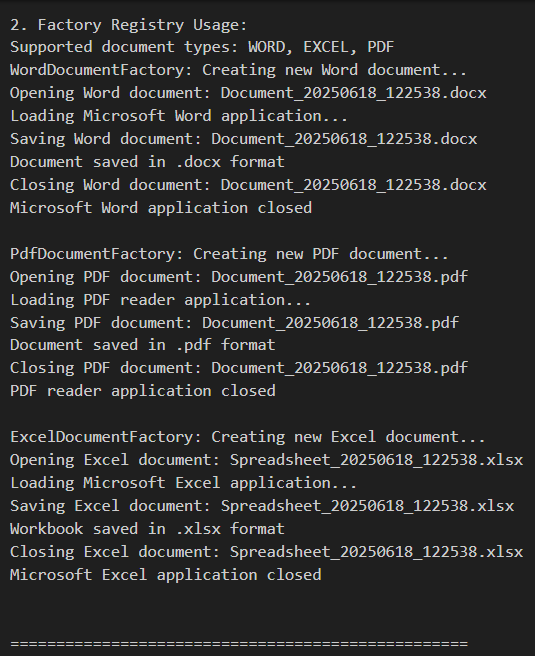
        }

    }

}

**Output**

****

****