# Design Patterns & Principles Exercise Solutions

## Exercise 1: Implementing the Singleton Pattern

### Code & Output Screenshots:

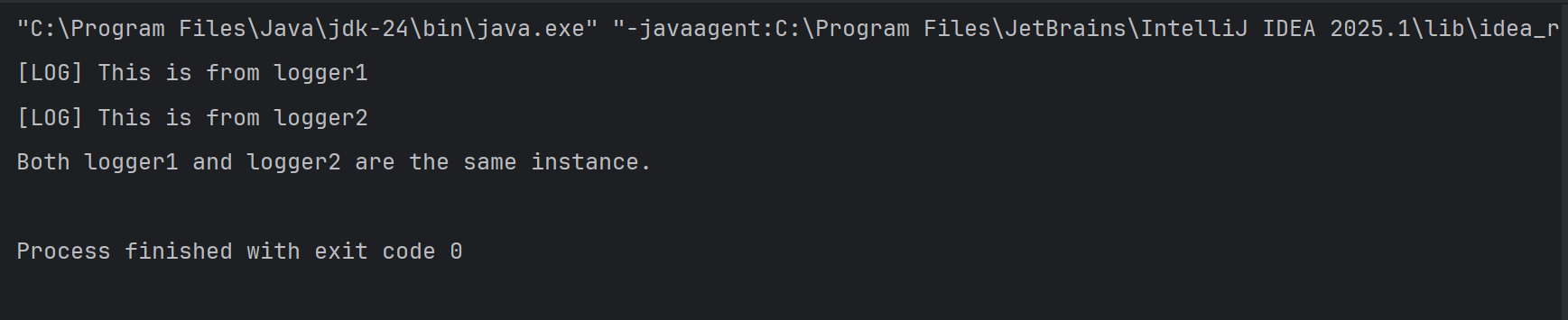
Logger.java

package com.exercise;  
  
public class Logger {  
 private static Logger *instance*;  
 private Logger(){}  
  
 public static Logger getInstance(){  
 if (*instance* == null) {  
 synchronized (Logger.class){  
 if(*instance* == null){  
 *instance* = new Logger();  
 }  
 }  
 }  
 return *instance*;  
 }  
  
 public void log(String message) {  
 System.*out*.println("[LOG] " + message);  
 }  
}

LoggerTest.java

package com.exercise;  
  
public class LoggerTest {  
 public static void main(String[] args) {  
 Logger logger1 = Logger.*getInstance*();  
 Logger logger2 = Logger.*getInstance*();  
  
 logger1.log("This is from logger1");  
 logger2.log("This is from logger2");  
  
 if (logger1 == logger2) {  
 System.*out*.println("Both logger1 and logger2 are the same instance.");  
 } else {  
 System.*out*.println("logger1 and logger2 are different instances.");  
 }  
 }  
}

Output



## Exercise 2: Implementing the Factory Method Pattern

### Code & Output Screenshots:

*Document.java*

package com.exercise.factorymethods;  
  
public interface Document {  
 void open();  
 void close();  
 void save();  
}

*WordDocument.java*

package com.exercise.factorymethods;  
  
public class WordDocument implements Document{  
 @Override  
 public void open() {  
 System.*out*.println("Opening Word Document");  
 }  
 @Override  
 public void close() {  
 System.*out*.println("Closing Word Document");  
 }  
 @Override  
 public void save() {  
 System.*out*.println("Saving Word Document");  
 }  
}

*PdfDocument.java*

package com.exercise.factorymethods;  
  
public class PdfDocument implements Document{  
 @Override  
 public void open() {  
 System.*out*.println("Opening PDF Document");  
 }  
 @Override  
 public void close() {  
 System.*out*.println("Closing PDF Document");  
 }  
 @Override  
 public void save() {  
 System.*out*.println("Saving PDF Document");  
 }  
}

*ExcelDocument.java*

package com.exercise.factorymethods;  
  
public class ExcelDocument implements Document{  
 @Override  
 public void open() {  
 System.*out*.println("Opening Excel Document");  
 }  
 @Override  
 public void close() {  
 System.*out*.println("Closing Excel Document");  
 }  
 @Override  
 public void save() {  
 System.*out*.println("Saving Excel Document");  
 }  
}

*WordDocumentFactory.java*

package com.exercise.factorymethods;  
  
public class WordDocumentFactory extends DocumentFactory {  
 @Override  
 public Document createDocument() {  
 return new WordDocument();  
 }  
}

*PdfDocumentFactory.java*

package com.exercise.factorymethods;  
  
public class PdfDocumentFactory extends DocumentFactory {  
 @Override  
 public Document createDocument() {  
 return new PdfDocument();  
 }  
}

*ExcelDocumentFactory.java*

package com.exercise.factorymethods;  
  
public class ExcelDocumentFactory extends DocumentFactory {  
 @Override  
 public Document createDocument() {  
 return new ExcelDocument();  
 }  
}

*DocumentFactory.java*

package com.exercise.factorymethods;  
  
public abstract class DocumentFactory {  
 public abstract Document createDocument();  
}

*Main.java*

package com.exercise;  
  
import com.exercise.factorymethods.\*;  
  
public class Main {  
 public static void main(String[] args) {  
 DocumentFactory wordFactory = new WordDocumentFactory();  
 Document wordDoc = wordFactory.createDocument();  
 wordDoc.open();  
 wordDoc.save();  
 wordDoc.close();  
  
 System.*out*.println("--------------------");  
  
 DocumentFactory pdfFactory = new PdfDocumentFactory();  
 Document pdfDoc = pdfFactory.createDocument();  
 pdfDoc.open();  
 pdfDoc.save();  
 pdfDoc.close();  
  
 System.*out*.println("--------------------");  
  
 DocumentFactory excelFactory = new ExcelDocumentFactory();  
 Document excelDoc = excelFactory.createDocument();  
 excelDoc.open();  
 excelDoc.save();  
 excelDoc.close(); }  
}

*Output*

