# Design Patterns and Principles(Hands-on)

Mandatory:

Deep-Skilling submission

**Exercise 1: Implementing the Singleton Pattern**

Code:

Logger.java

package singleton;  
  
public class Logger {  
 private static Logger *instance*;  
  
 private Logger() {  
 System.*out*.println("Logger initialized.");  
 }  
  
 public static Logger getInstance() {  
 if (*instance* == null) {  
 *instance* = new Logger();  
 }  
 return *instance*;  
 }  
  
 public void log(String message) {  
 System.*out*.println("LOG: " + message);  
 }  
}

Main.java

package singleton;  
  
public class Main {  
 public static void main(String[] args) {  
 Logger logger1 = Logger.*getInstance*();  
 logger1.log("First log message");  
  
 Logger logger2 = Logger.*getInstance*();  
 logger2.log("Second log message");  
  
 System.*out*.println("Same instance? " + (logger1 == logger2));  
 }  
}

Output:

A screen shot of a computer

AI-generated content may be incorrect.

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

Document.java

package factory;

public interface Document {

void open();

}

**WordDocument.java**

package factory;

public class WordDocument implements Document {

public void open() {

System.out.println("Opening Word document...");

}

}

**Create PdfDocument.java**

package factory;

public class PdfDocument implements Document {

public void open() {

System.out.println("Opening PDF document...");

}

}

**ExcelDocument.java**

package factory;

public class ExcelDocument implements Document {

public void open() {

System.out.println("Opening Excel document...");

}

}

**DocumentFactory.java (abstract class)**

package factory;

public abstract class DocumentFactory {

public abstract Document createDocument();

}

**WordDocumentFactory.java**

package factory;

public class WordDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new WordDocument();

}

}

**PdfDocumentFactory.java**

package factory;

public class PdfDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new PdfDocument();

}

}

**ExcelDocumentFactory.java**

package factory;

public class ExcelDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new ExcelDocument();

}

}

**Main.java**

package factory;

public class Main {

public static void main(String[] args) {

DocumentFactory wordFactory = new WordDocumentFactory();

Document word = wordFactory.createDocument();

word.open();

DocumentFactory pdfFactory = new PdfDocumentFactory();

Document pdf = pdfFactory.createDocument();

pdf.open();

DocumentFactory excelFactory = new ExcelDocumentFactory();

Document excel = excelFactory.createDocument();

excel.open();

}

}

Output:

A screen shot of a computer

AI-generated content may be incorrect.

Additional:

**Exercise 5: Implementing the Decorator Pattern**

**Notifier.java**

package decorator;

public interface Notifier {

void send(String message);

}

**EmailNotifier.java**

package decorator;

public class EmailNotifier implements Notifier {

public void send(String message) {

System.out.println("Sending Email: " + message);

}

}

**NotifierDecorator.java (Abstract Decorator)**

package decorator;

public abstract class NotifierDecorator implements Notifier {

protected Notifier wrappedNotifier;

public NotifierDecorator(Notifier notifier) {

this.wrappedNotifier = notifier;

}

public void send(String message) {

wrappedNotifier.send(message); // Delegate to the wrapped object

}

}

**SMSNotifierDecorator.java**

package decorator;

public class SMSNotifierDecorator extends NotifierDecorator {

public SMSNotifierDecorator(Notifier notifier) {

super(notifier);

}

@Override

public void send(String message) {

super.send(message);

System.out.println("Sending SMS: " + message);

}

}

**SlackNotifierDecorator.java**

package decorator;

public class SlackNotifierDecorator extends NotifierDecorator {

public SlackNotifierDecorator(Notifier notifier) {

super(notifier);

}

@Override

public void send(String message) {

super.send(message);

System.out.println("Sending Slack message: " + message);

}

}

**Main.java**

package decorator;

public class Main {

public static void main(String[] args) {

// Base notifier (Email only)

Notifier baseNotifier = new EmailNotifier();

// Add SMS on top of Email

Notifier smsNotifier = new SMSNotifierDecorator(baseNotifier);

// Add Slack on top of SMS and Email

Notifier fullNotifier = new SlackNotifierDecorator(smsNotifier);

// Send notification

fullNotifier.send("Your package has been shipped.");

}

}

A screenshot of a computer

AI-generated content may be incorrect.