**Exercise 1: Implementing the Singleton Pattern**

Logger.java

public class Logger {

    private Logger() {

        System.out.println("Logger initialized.");

    }

    private static class LoggerHolder {

        private static final Logger INSTANCE = new Logger();

    }

    public static Logger getInstance() {

        return LoggerHolder.INSTANCE;

    }

    public void info(String message) {

        System.out.println("INFO: " + message);

    }

    public void warn(String message) {

        System.out.println("WARN: " + message);

    }

    public void error(String message) {

        System.err.println("ERROR: " + message);

    }

}

TestLogger.java

public class TestLogger {

    public static void main(String[] args) {

        System.out.println("Starting TestLogger...");

        Logger logger1 = Logger.getInstance();

        Logger logger2 = Logger.getInstance();

        if (logger1 == logger2) {

            System.out.println("Both logger1 and logger2 reference the same instance.");

        } else {

            System.out.println("Different instances exist! Singleton failed.");

        }

        logger1.info("This is an info message.");

        logger2.warn("This is a warning message.");

        logger1.error("This is an error message.");

        anotherMethod();

    }

   private static void anotherMethod() {

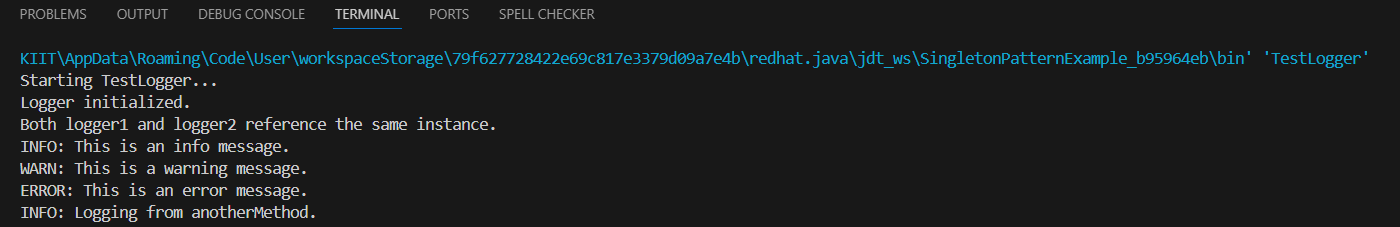
        Logger logger = Logger.getInstance();

        logger.info("Logging from anotherMethod.");

    }

}

Output



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

public class DocumentCreatorDemo {

    interface Doc {

        void open();

        void save();

        void close();

        String getName();

    }

    static class WordDoc implements Doc {

        @Override

        public void open() {

            System.out.println("[WordDoc] open()");

        }

        @Override

        public void save() {

            System.out.println("[WordDoc] save()");

        }

        @Override

        public void close() {

            System.out.println("[WordDoc] close()");

        }

        @Override

        public String getName() {

            return "Word Document";

        }

    }

    static class PdfDoc implements Doc {

        @Override

        public void open() {

            System.out.println("[PdfDoc] open()");

        }

        @Override

        public void save() {

            System.out.println("[PdfDoc] save()");

        }

        @Override

        public void close() {

            System.out.println("[PdfDoc] close()");

        }

        @Override

        public String getName() {

            return "PDF Document";

        }

    }

    static class ExcelDoc implements Doc {

        @Override

        public void open() {

            System.out.println("[ExcelDoc] open()");

        }

        @Override

        public void save() {

            System.out.println("[ExcelDoc] save()");

        }

        @Override

        public void close() {

            System.out.println("[ExcelDoc] close()");

        }

        @Override

        public String getName() {

            return "Excel Spreadsheet";

        }

    }

    abstract static class DocFactory {

        public abstract Doc makeDoc();

        public void process() {

            Doc d = makeDoc();

            System.out.println(">>> Processing a " + d.getName());

            d.open();

            d.save();

            d.close();

            System.out.println();

        }

    }

    static class WordFactory extends DocFactory {

        @Override

        public Doc makeDoc() {

            return new WordDoc();

        }

    }

    static class PdfFactory extends DocFactory {

        @Override

        public Doc makeDoc() {

            return new PdfDoc();

        }

    }

    static class ExcelFactory extends DocFactory {

        @Override

        public Doc makeDoc() {

            return new ExcelDoc();

        }

    }    public static void main(String[] args) {

        DocFactory[] factories = {

            new WordFactory(),

            new PdfFactory(),

            new ExcelFactory()

        };

        for (DocFactory f : factories) {

            f.process();

        }

    }

}

