

Design principles & Patterns:

Exercise 1: Implementing the Singleton Pattern

Scenario:

You need to ensure that a logging utility class in your application has only one instance throughout the application lifecycle to ensure consistent logging.

Code:

```
class Logger {
    private static Logger instance;

    private Logger() {
        System.out.println("Logger instance created.");
    }

    public static synchronized Logger getInstance() {
        if (instance == null) {
            instance = new Logger();
        }
        return instance;
    }

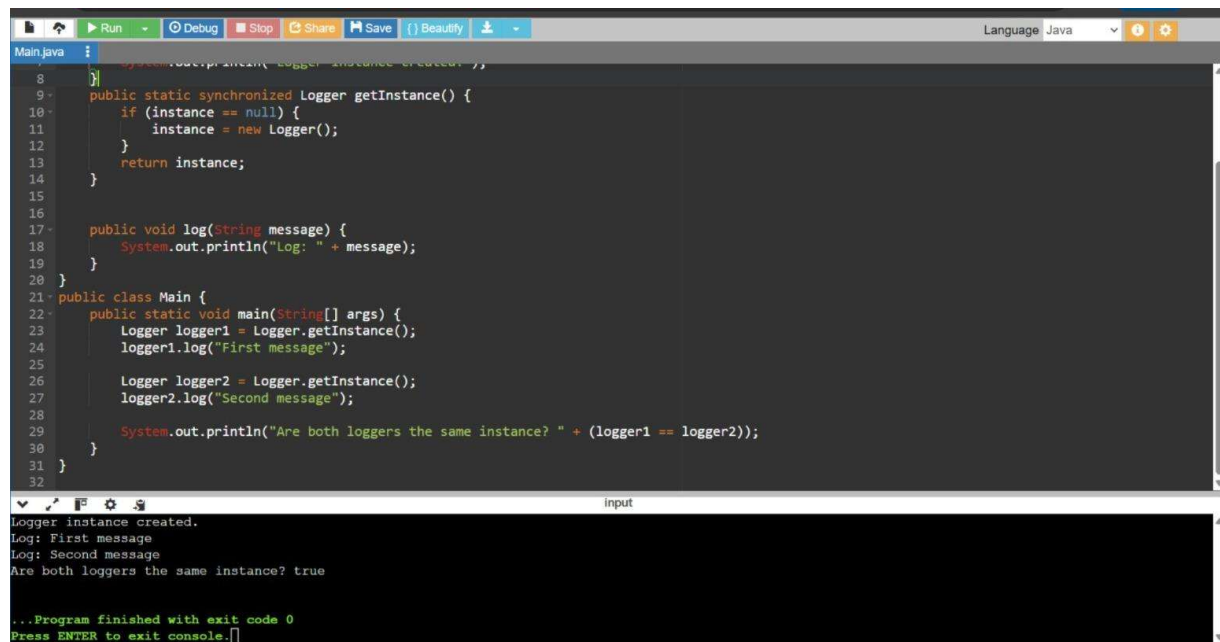
    public void log(String message) {
        System.out.println("Log: " + message);
    }
}

public class Main {
    public static void main(String[] args) {
        Logger logger1 = Logger.getInstance();
        logger1.log("First message");

        Logger logger2 = Logger.getInstance();
        logger2.log("Second message");

        System.out.println("Are both loggers the same instance? " + (logger1 == logger2));
    }
}
```

Output:



```
8      System.out.println("Logger instance created.");
9  }
10 public static synchronized Logger getInstance() {
11     if (instance == null) {
12         instance = new Logger();
13     }
14     return instance;
15 }
16
17 public void log(String message) {
18     System.out.println("Log: " + message);
19 }
20 }
21 public class Main {
22     public static void main(String[] args) {
23         Logger logger1 = Logger.getInstance();
24         logger1.log("First message");
25
26         Logger logger2 = Logger.getInstance();
27         logger2.log("Second message");
28
29         System.out.println("Are both loggers the same instance? " + (logger1 == logger2));
30     }
31 }
32
```

Logger instance created.
Log: First message
Log: Second message
Are both loggers the same instance? true
...Program finished with exit code 0
Press ENTER to exit console.

Exercise 2: Implementing the Factory Method Pattern

Scenario:

You are developing a document management system that needs to create different types of documents (e.g., Word, PDF, Excel). Use the Factory Method Pattern to achieve this.

Code:

```
interface Document {  
    void open();  
}
```

```
class WordDocument implements Document {  
    @Override
```

```
public void open() {  
    System.out.println("Opening Word document.");  
}  
}
```

```
class PdfDocument implements Document {  
    @Override  
    public void open() {  
        System.out.println("Opening PDF document.");  
    }  
}
```

```
class ExcelDocument implements Document {  
    @Override  
    public void open() {  
        System.out.println("Opening Excel document.");  
    }  
}
```

```
enum DocumentType {  
    WORD, PDF, EXCEL  
}
```

```
class DocumentFactory {  
    public static Document createDocument(DocumentType type) {  
        switch (type) {  
            case WORD:
```

```

        return new WordDocument();
    case PDF:
        return new PdfDocument();
    case EXCEL:
        return new ExcelDocument();
    default:
        throw new IllegalArgumentException("Invalid document type");
    }
}
}

```

```

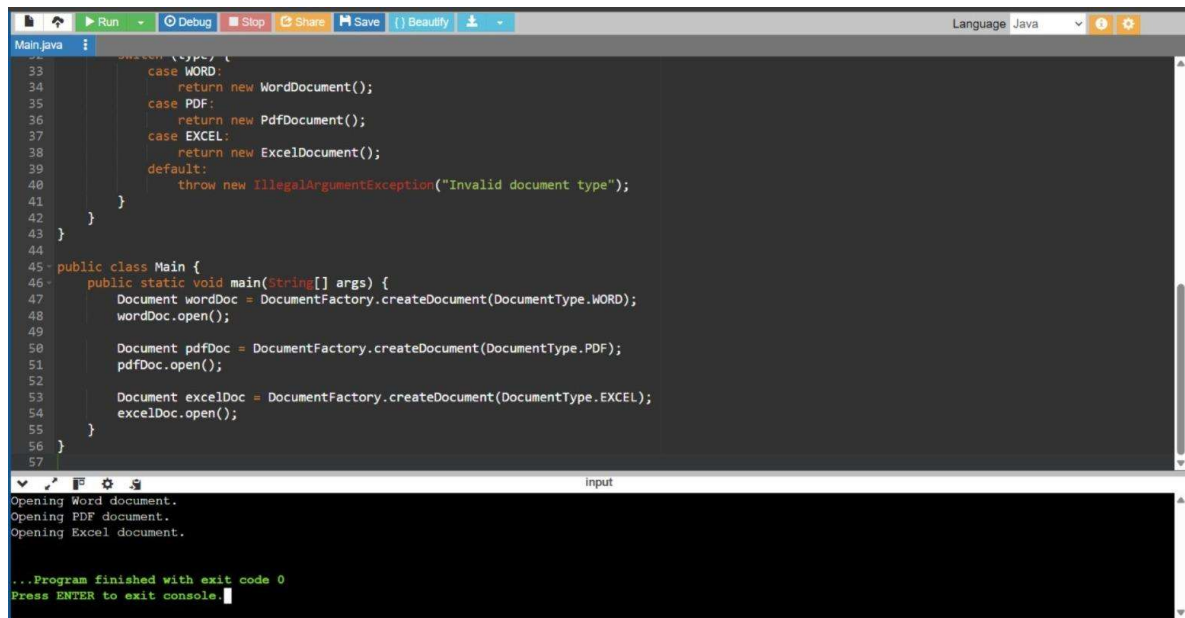
public class Main {
    public static void main(String[] args) {
        Document wordDoc =
        DocumentFactory.createDocument(DocumentType.WORD);
        wordDoc.open();

        Document pdfDoc =
        DocumentFactory.createDocument(DocumentType.PDF);
        pdfDoc.open();

        Document excelDoc =
        DocumentFactory.createDocument(DocumentType.EXCEL);
        excelDoc.open();
    }
}

```

Output:



The image shows a screenshot of an IDE window. The top part displays a Java source file named 'Main.java'. The code defines a switch statement for 'DocumentType' with cases for WORD, PDF, and EXCEL, each returning a new instance of WordDocument, PdfDocument, or ExcelDocument respectively. A default case throws an 'IllegalArgumentException' with the message 'Invalid document type'. Below the switch statement, a 'Main' class contains a 'main' method that creates and opens documents of each type using 'DocumentFactory.createDocument()' and 'open()' methods.

```
33     case WORD:
34         return new WordDocument();
35     case PDF:
36         return new PdfDocument();
37     case EXCEL:
38         return new ExcelDocument();
39     default:
40         throw new IllegalArgumentException("Invalid document type");
41     }
42 }
43 }
44
45 public class Main {
46     public static void main(String[] args) {
47         Document wordDoc = DocumentFactory.createDocument(DocumentType.WORD);
48         wordDoc.open();
49
50         Document pdfDoc = DocumentFactory.createDocument(DocumentType.PDF);
51         pdfDoc.open();
52
53         Document excelDoc = DocumentFactory.createDocument(DocumentType.EXCEL);
54         excelDoc.open();
55     }
56 }
57
```

The bottom part of the IDE shows the console output. It displays the messages 'Opening Word document.', 'Opening PDF document.', and 'Opening Excel document.' followed by '...Program finished with exit code 0' and a prompt 'Press ENTER to exit console.'.

```
Opening Word document.
Opening PDF document.
Opening Excel document.

...Program finished with exit code 0
Press ENTER to exit console.
```