WEEK 1

Naren Karthick A

St. Joseph's Institute of Technology

Superset ID: 6377326

Design Pattern and Principles

1.Implementing the singleton pattern Code:

MyLogger.java

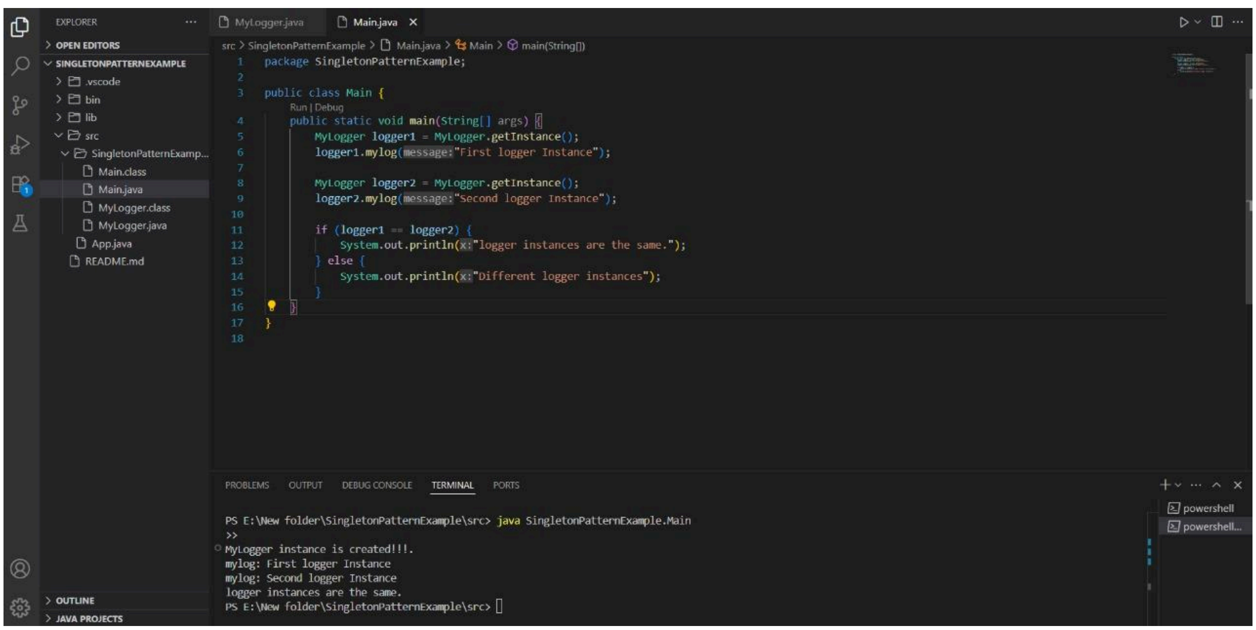
```
package SingletonPatternExample;

public class MyLogger {
    private static MyLogger instance;

    private MyLogger() {
        System.out.println("MyLogger instance is created!!!.");
    }

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

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

Main.java

```
package SingletonPatternExample;

public class Main {
    public static void main(String[] args) {
```

```
        MyLogger logger1 = MyLogger.getInstance();
        logger1.mylog("First logger Instance");

        MyLogger logger2 = MyLogger.getInstance();
        logger2.mylog("Second logger Instance");

        if (logger1 == logger2) {
            System.out.println("logger instances are the same.");
        } else {
            System.out.println("Different logger instances");
        }
    }
}
```

Output:



2.Factory Method Pattern

Code:

Main.java

```
package documentfactory;

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();
    }
}
```

Document.java

```
 package documentfactory;
public interface Document {
    void open();
}
```

DocumentFactory.java

```
 package documentfactory;
public abstract class DocumentFactory {
    public abstract Document createDocument();
}
```

ExcelDocument .java

```
package documentfactory;

public class ExcelDocument implements Document {

    public void open() {

        System.out.println("Opening the Excel Document!!!");

    }

}
```

ExcelDocumentFactory.java

```
package documentfactory;

public class ExcelDocumentFactory extends DocumentFactory {

    public Document createDocument() {

        return new ExcelDocument();

    }

}
```

PdfDocument.java

```
package documentfactory;

public class PdfDocument implements Document {

    public void open() {

        System.out.println("Opening the PDF Document!!!");

    }

}
```

PdfDocumentFactory.java

```
package documentfactory;
```

```
public class PdfDocumentFactory extends DocumentFactory {

    public Document createDocument() {

        return new PdfDocument();

    }

}
```

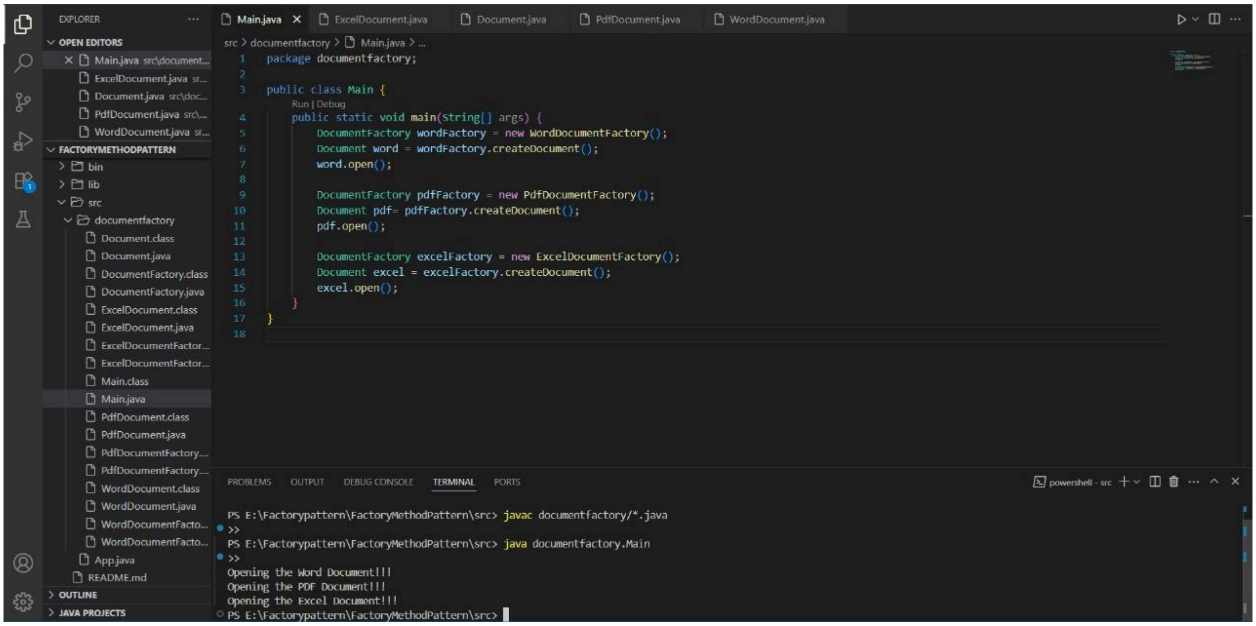
WordDocument.java

```
package documentfactory;

public class WordDocument implements Document {

    public void open() {

        System.out.println("Opening the Word Document!!!");

    }

}
```

WordDocumentFactory

```
package documentfactory;

public class WordDocumentFactory extends DocumentFactory {

    public Document createDocument() {

        return new WordDocument();

    }

}
```

Output:


EXPLORER
OPEN EDITORS
Main.java
ExcelDocument.java
Document.java
PdfDocument.java
WordDocument.java
FACTORYMETHODPATTERN
bin
lib
src
documentfactory
Document.class
Document.java
DocumentFactory.class
DocumentFactory.java
ExcelDocument.class
ExcelDocument.java
Main.class
Main.java
PdfDocument.class
PdfDocument.java
WordDocument.class
WordDocument.java
App.java
README.md
OUTLINE
JAVA PROJECTS
src > documentfactory > Main.java > ...
package documentfactory;
public class Main {
Run | Debug
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();
}
}
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
powershell - src
PS E:\Factorypattern\FactoryMethodPattern\src> javac documentfactory/*.java
>>
PS E:\Factorypattern\FactoryMethodPattern\src> java documentfactory.Main
>>
Opening the Word Document!!!
Opening the PDF Document!!!
Opening the Excel Document!!!
PS E:\Factorypattern\FactoryMethodPattern\src>

Data Structure and Algorithm

3. E_Commerce Platform Search Function

Big O notation: Big O notation describes the upper bound of an algorithms space and time complexity based on the input size n. It helps us to easily understand the performance and efficiency of the algorithm with respective to time and space without running the actual code.

There are three cases here: best, average and worst cases.

Linear Search:

Best case: Target element is the first element

Average case: Target element in the middle or not present.

Worst Case: Target element is the last element or not present

Binary Search:

Best case: Target element is the middle element

Average case: target element is found after repeatedly halving the search space.

Worst Case: target element is found after repeatedly halving the search space.

Code:

Main.java

```
package search;

public class Main {

    public static void main(String[] args) {

        Product[] products = {

            new Product(101, "Jeans", "Fashion"),

            new Product(102, "Washing Machine", "Appliances"),

            new Product(103, "Sofa", "Home"),

            new Product(104, "Mobile", "Electronics"),

            new Product(105, "Lipstick", "Cosmetics")

        };
```

```java
        String searchName = "Sofa";
        Product resultLinear = ProductSearchAlgorithms.linearSearch(products, searchName);
        System.out.println(" Linear Search Result:");
        System.out.println(resultLinear != null ? resultLinear : "Product not found");
        Product resultBinary = ProductSearchAlgorithms.binarySearch(products, searchName);
        System.out.println(" Binary Search Result:");
        System.out.println(resultBinary != null ? resultBinary : "Product not found");
    }
}
```

Product.java

```java
package search;
public class Product {
    int productId;
    String productName;
    String category;
    public Product(int productId, String productName, String category) {
        this.productId = productId;
        this.productName = productName;
        this.category = category;
    }
    @Override
    public String toString() {
        return "ProductID: " + productId + ", Name: " + productName + ", Category: " + category;
    }
```

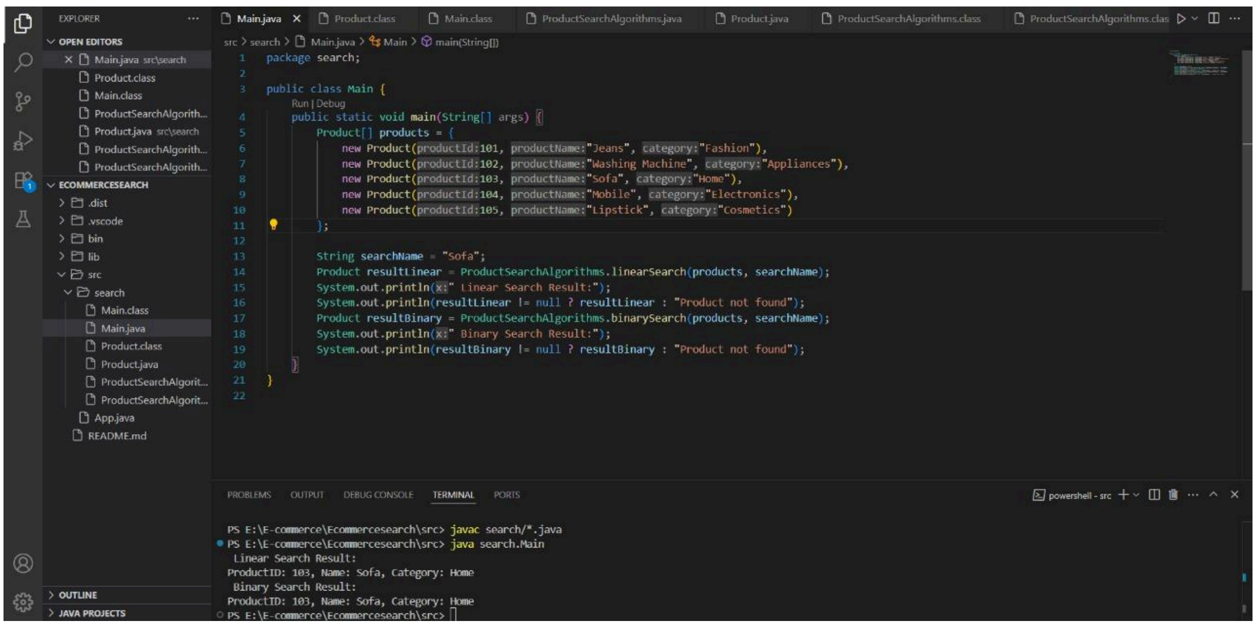
```
}
```

ProductSearchAlgorithms.java

```
package search;

import java.util.Arrays;

import java.util.Comparator;

public class ProductSearchAlgorithms {

    public static Product linearSearch(Product[] products, String targetName) {

        for (Product product : products) {

            if (product.productName.equalsIgnoreCase(targetName)) {

                return product;

            }

        }

        return null;

    }

    public static Product binarySearch(Product[] products, String targetName) {

        Arrays.sort(products, Comparator.comparing(p -> p.productName.toLowerCase()));

        int low = 0, high = products.length - 1;

        while (low <= high) {

            int mid = (low + high) / 2;

            int compare = products[mid].productName.compareToIgnoreCase(targetName);

            if (compare == 0) {

                return products[mid];

            } else if (compare < 0) {

                low = mid + 1;
```

```
            } else {
                high = mid - 1;
            }
        }
        return null;
    }
}
```

Output:



Time Complexity of linear search and binary search

Linear Search:

Best case: O(1)

Average case: O(n/2)

Worst case: O(n)

Binary Search:

Best case: O(1)

Average case: O(log n)

Worst case: O(log n)

Best Algorithm for E-commerce:Best Algorithm for E-commerce is the Binary Search as it is faster with time complexity O(log n),where e-commerce is a platform with millions of product,checking with linear search is a slower process. The linear search process is not efficient for large data as e-commerce is a large data site, binary search is highly efficient for the large data.

4.Financial Forecasting

Recursion

Recursion is a programming technique where a function calls itself to solve a problem by breaking it down into smaller subproblems until the base case is reached.Recursion has 2 parts

Base Case: The condition to stop recursion

Recursive call: The function calls itself with smaller input

Code:

Main.java

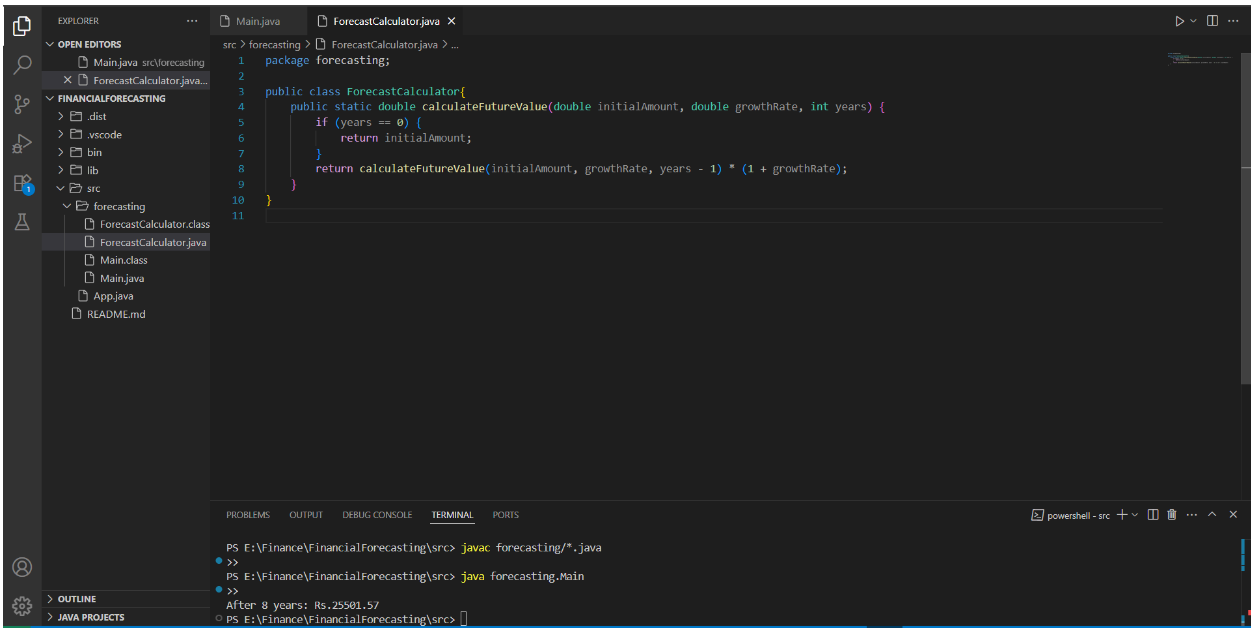
```
package forecasting;

public class Main {

    public static void main(String[] args) {

        double initialAmount=16000;

        double growthRate=0.06;

        int years=8;

        double futureValue = ForecastCalculator.calculateFutureValue(initialAmount, growthRate, years);

        System.out.printf("After %d years: Rs.%.2f\n", years, futureValue);

    }

}
```

ForecastCalculator.java

```
package forecasting;

public class ForecastCalculator{

    public static double calculateFutureValue(double initialAmount, double growthRate, int years) {
```

```
        if (years == 0) {
            return initialAmount;
        }
        return calculateFutureValue(initialAmount, growthRate, years - 1) * (1 + growthRate);
    }
}
```

Output:



Time Complexity:

O(n) where n is no of years

One call for each year

Optimization Approach:

We can use mathematical formula as best approach where is result time complexity of O(1).

The formula:
PredictValue = Currentvalue*(1+growth rate)^years.