# WEEK:1

## Exercise 1: Implementing the Singleton Pattern

Logger.java

```
public class Logger {
    private static Logger instance;
    private Logger() {
        System.out.println("Logger instance created.");
    }
    public static Logger getInstance() {
        if (instance == null) {
            instance = new Logger();
        }
        return instance;
    }
    public void log(String message) {
        System.out.println("Log: " + message);
    }
}
```

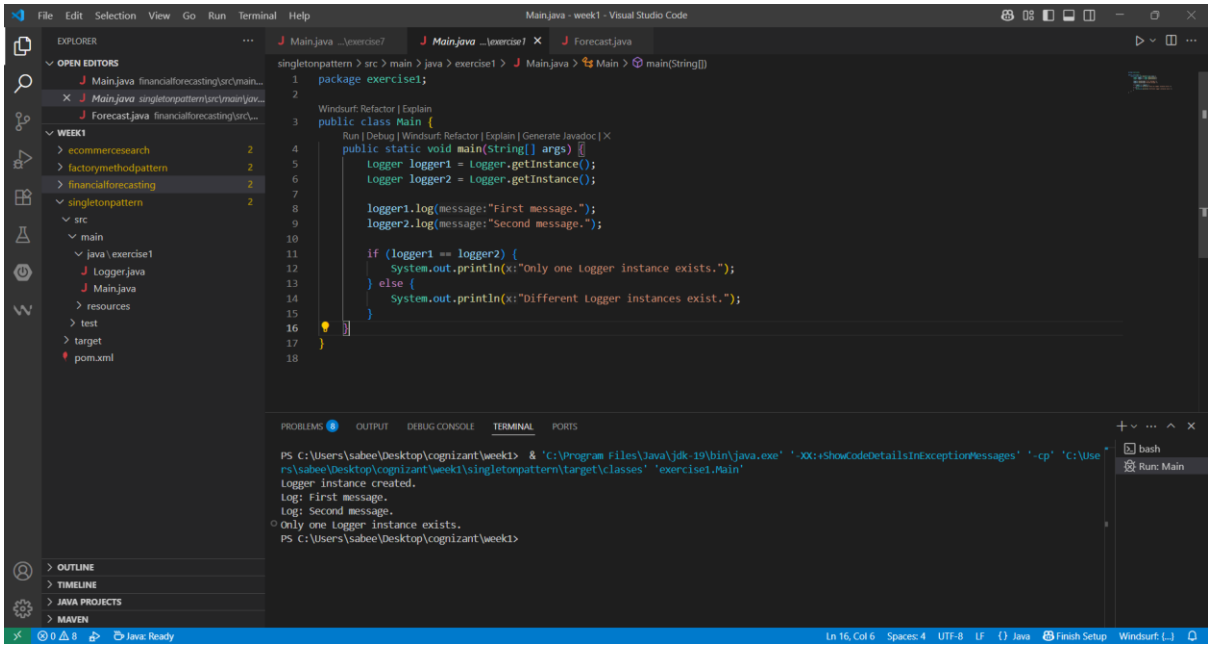
Main.java

```
public class Main {
    public static void main(String[] args) {
        Logger logger1 = Logger.getInstance();
        Logger logger2 = Logger.getInstance();
```

```
        logger1.log("First message.");
        logger2.log("Second message.");

        if (logger1 == logger2) {
            System.out.println("Only one Logger instance exists.");
        } else {
            System.out.println("Different Logger instances exist.");
        }
    }
}
```

**OUTPUT:**




## Exercise 2: Implementing the Factory Method Pattern

Document.java

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

```
}
```

DocumentFactory.java

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

ExcelDocument.java

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

ExcelDocumentFactory.java

```
public class ExcelDocumentFactory extends DocumentFactory {
    public Document createDocument() {
        return new ExcelDocument();
    }
}
```

PdfDocument.java

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

```
    }
}
```

PdfDocumentFactory.java

```
public class PdfDocumentFactory extends DocumentFactory {
    public Document createDocument() {
        return new PdfDocument();
    }
}
```

WordDocument.java

```
public class WordDocument implements Document {
    public void open() {
        System.out.println("Opening a Word document.");
    }
}
```

WordDocumentFactory.java

```
public class WordDocumentFactory extends DocumentFactory {
    public Document createDocument() {
        return new WordDocument();
    }
}
```

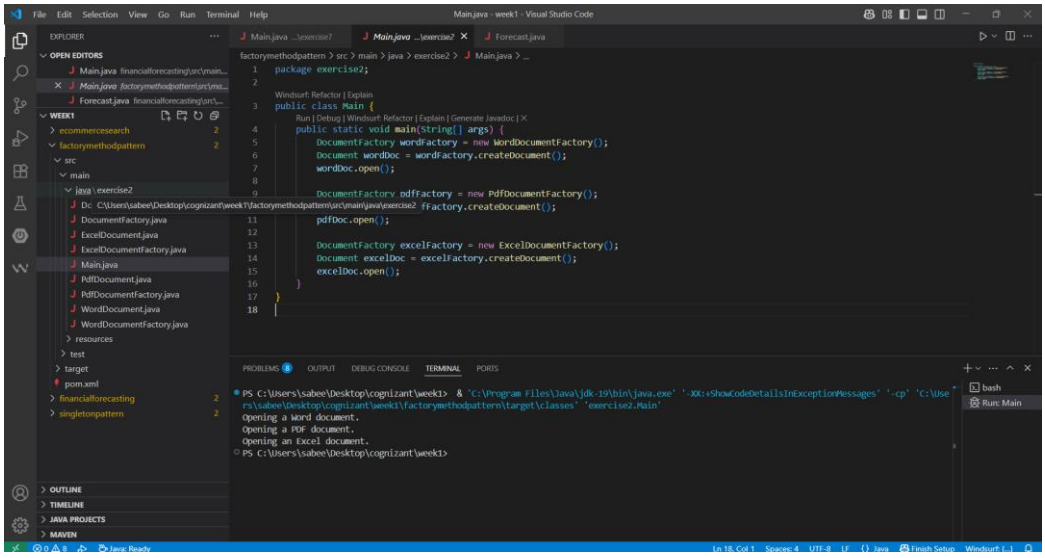
Main.java

```
public class Main {
    public static void main(String[] args) {
        DocumentFactory wordFactory = new WordDocumentFactory();
        Document wordDoc = wordFactory.createDocument();
        wordDoc.open();

        DocumentFactory pdfFactory = new PdfDocumentFactory();
        Document pdfDoc = pdfFactory.createDocument();
        pdfDoc.open();

        DocumentFactory excelFactory = new ExcelDocumentFactory();
        Document excelDoc = excelFactory.createDocument();
        excelDoc.open();
    }
}
```

**OUTPUT:**



## Exercise 3: E-commerce Platform Search Function

Product.java

```
public class Product {
    private int productId;
    private String productName;
    private String category;

    public Product(int productId, String productName, String category) {
        this.productId = productId;
        this.productName = productName;
        this.category = category;
    }

    public int getProductId() {
        return productId;
    }

    public String getProductName() {
        return productName;
    }

    public String getCategory() {
        return category;
    }
```

```
    @Override
    public String toString() {
        return "[" + productId + "] " + productName + " - " + category;
    }
}
```

SearchUtils.java

```
import java.util.Arrays;
import java.util.Comparator;

public class SearchUtils {

    public static Product linearSearch(Product[] products, int id) {
        for (Product p : products) {
            if (p.getProductId() == id) {
                return p;
            }
        }
        return null;
    }

    public static Product binarySearch(Product[] products, int id) {
        int low = 0;
        int high = products.length - 1;
```

```
        while (low <= high) {
            int mid = (low + high) / 2;
            int midId = products[mid].getProductId();

            if (midId == id) {
                return products[mid];
            } else if (midId < id) {
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }
        return null;
    }

    public static void sortByProductId(Product[] products) {
        Arrays.sort(products, Comparator.comparingInt(Product::getProductId));
    }
}
```

Main.java

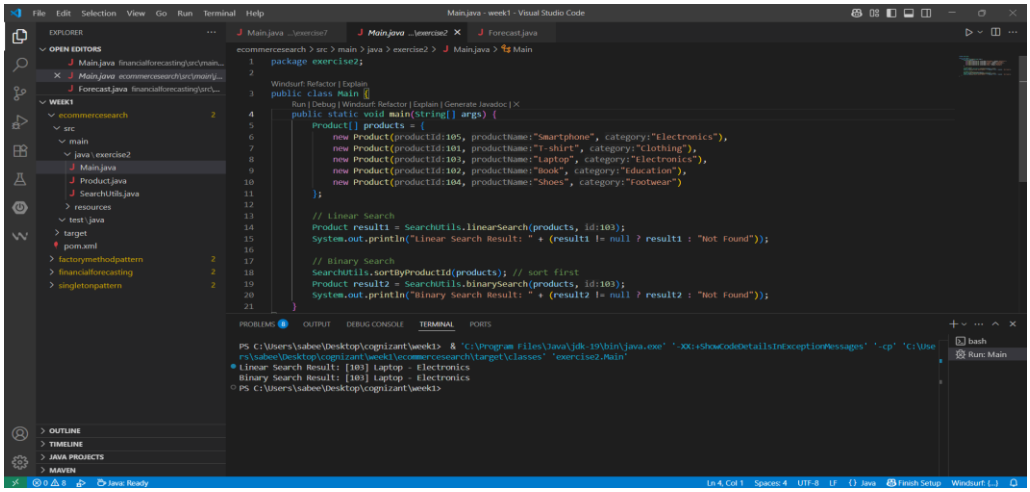
```
public class Main {
    public static void main(String[] args) {
        Product[] products = {
            new Product(105, "Smartphone", "Electronics"),
```

```
            new Product(101, "T-shirt", "Clothing"),
            new Product(103, "Laptop", "Electronics"),
            new Product(102, "Book", "Education"),
            new Product(104, "Shoes", "Footwear")
        };

        // Linear Search
        Product result1 = SearchUtils.linearSearch(products, 103);
        System.out.println("Linear Search Result: " + (result1 != null ? result1 : "Not Found"));

        // Binary Search
        SearchUtils.sortByProductId(products); // sort first
        Product result2 = SearchUtils.binarySearch(products, 103);
        System.out.println("Binary Search Result: " + (result2 != null ? result2 : "Not Found"));
    }
}
```

**OUTPUT:**



## Exercise 4: Financial Forecasting

Forecast.java

```
public class Forecast {


    // Recursive method
    public static double futureValueRecursive(double currentValue, double
growthRate, int years) {
        if (years == 0) {
            return currentValue;
        }
        return futureValueRecursive(currentValue, growthRate, years - 1) * (1 +
growthRate);
    }


    // Optimized (Iterative) method
    public static double futureValueIterative(double currentValue, double
growthRate, int years) {
        for (int i = 0; i < years; i++) {
            currentValue *= (1 + growthRate);
        }
        return currentValue;
    }
}
```

Main.java

```
public class Main {
```

```
    public static void main(String[] args) {
        double currentValue = 10000; // Rs.10,000
        double growthRate = 0.08;    // 8% annual growth
        int years = 5;

        double recursiveValue = Forecast.futureValueRecursive(currentValue, growthRate, years);
        double iterativeValue = Forecast.futureValueIterative(currentValue, growthRate, years);

        System.out.printf("Recursive Forecast after %d years: ₹%.2f%n", years, recursiveValue);
        System.out.printf("Iterative Forecast after %d years: ₹%.2f%n", years, iterativeValue);
    }
}
```

**OUTPUT:**