**Week-1(Design Patterns and Principles and Data Structures and Algorithms)**

**Hands-on 1: Implementing the Singleton Pattern**

* **Logger.java**

package ourcompany;

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;

    }

}

* **Main.java**

package ourcompany;

public class Main {

    public static void main(String[] args) {

        Logger logger1 = Logger.getInstance();

        Logger logger2 = Logger.getInstance();

        if (logger1 == logger2) {

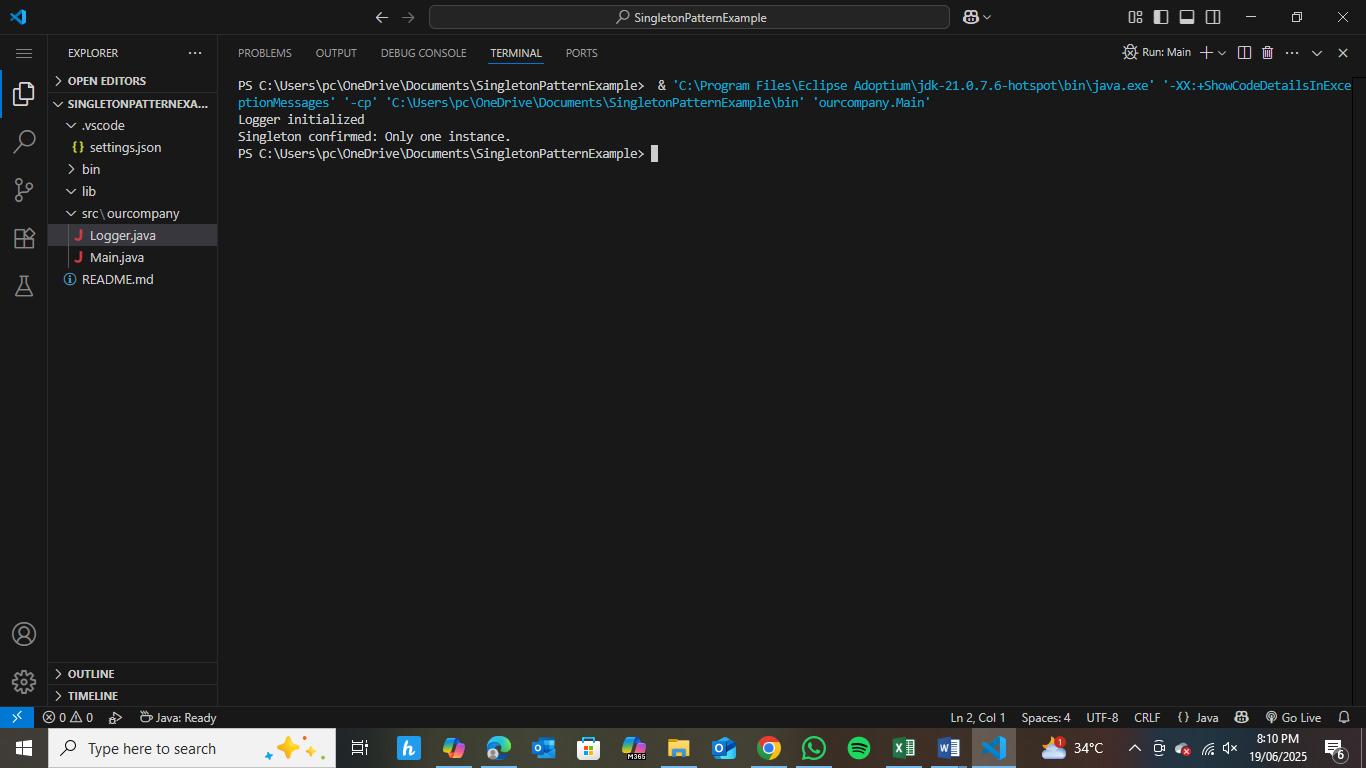
            System.out.println("Singleton confirmed: Only one instance.");

        }

    }

}

**Output:**



**Hands-on 2: Implementing the Factory Method Pattern**

* **Document.java**

package documents;

public interface Document {

    void open();

}

* **ExcelDocument.java**

package documents;

public class ExcelDocument implements Document {

    public void open() {

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

    }

}

* **PdfDocument.java**

package documents;

public class PdfDocument implements Document {

    public void open() {

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

    }

}

* **WordDocument.java**

package documents;

public class WordDocument implements Document {

    public void open() {

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

    }

}

* **DocumentFactory.java**

package documents;

public abstract class DocumentFactory {

    public abstract Document createDocument();

    public void openDocument() {

        Document doc = createDocument();

        doc.open();

    }

}

* **ExcelDocumentFactory.java**

package documents;

public class ExcelDocumentFactory extends DocumentFactory {

    public Document createDocument() {

        return new ExcelDocument();

    }

}

* **PdfDocumentFactory.java**

package documents;

public class PdfDocumentFactory extends DocumentFactory {

    public Document createDocument() {

        return new PdfDocument();

    }

}

* **WordDocumentFactory.java**

package documents;

public class WordDocumentFactory extends DocumentFactory {

    public Document createDocument() {

        return new WordDocument();

    }

}

* **Main.java**

package documents;

public class Main {

    public static void main(String[] args) {

        DocumentFactory wordFactory = new WordDocumentFactory();

        wordFactory.openDocument();  // Output: Opening Word document...

        DocumentFactory pdfFactory = new PdfDocumentFactory();

        pdfFactory.openDocument();   // Output: Opening PDF document...

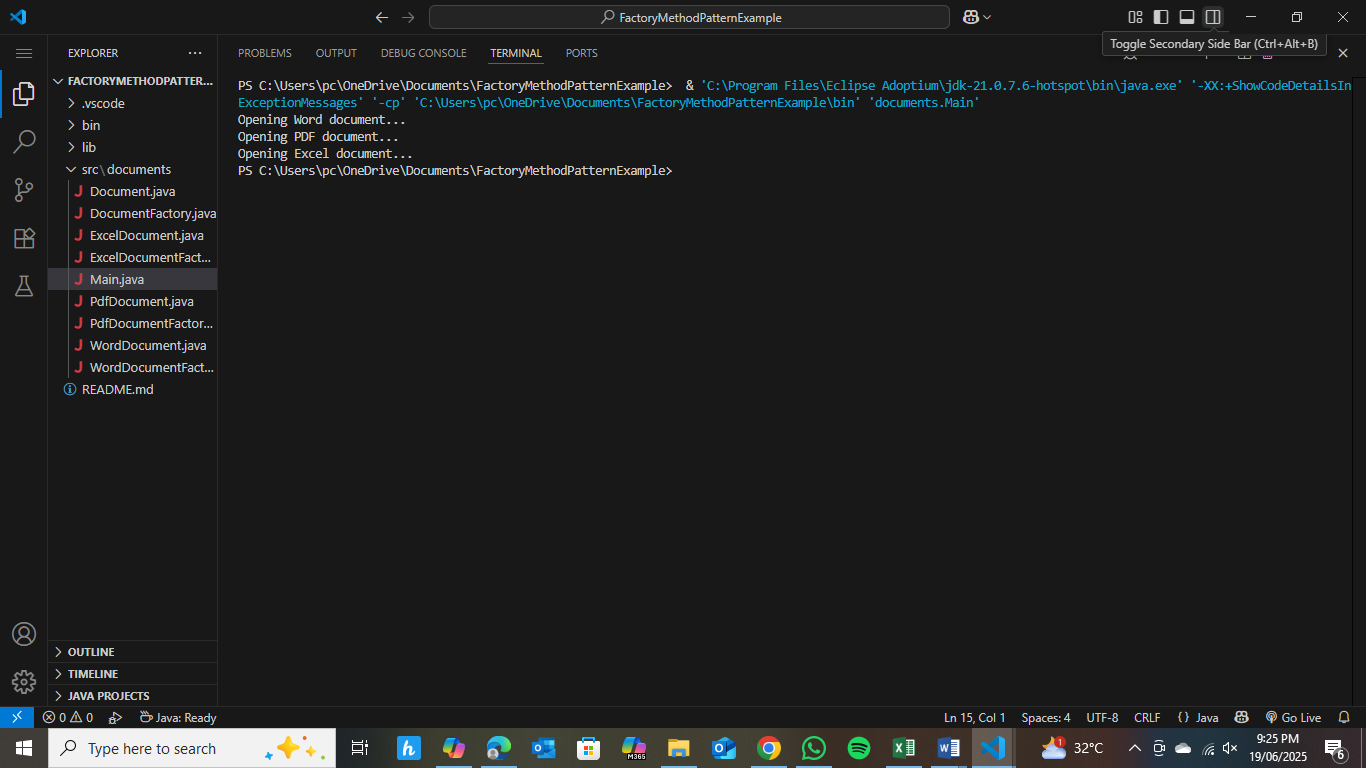
        DocumentFactory excelFactory = new ExcelDocumentFactory();

        excelFactory.openDocument(); // Output: Opening Excel document...

    }

}

**Output:**

****

**Hands-on 3: E-commerce Platform Search Function**

* **SearchProducts.java**

import java.util.Arrays;

import java.util.Comparator;

public class SearchProducts{

    public static void main(String[] args) {

        Product[] products = {

            new Product(101, "Laptop", "Electronics"),

            new Product(305, "Headphones", "Electronics"),

            new Product(202, "Smartphone", "Electronics"),

            new Product(150, "Keyboard", "Accessories"),

            new Product(410, "Monitor", "Electronics")

        };

        Product resultLinear = linearSearch(products, 202);

        System.out.println("Linear Search Result: " + resultLinear);

        Arrays.sort(products, new Comparator<Product>() {

            @Override

            public int compare(Product p1, Product p2) {

                return p1.getProductId() - p2.getProductId();

            }

        });

        System.out.println("\nProducts after sorting by ID:");

        for (Product p : products) {

            System.out.println(p);

        }

        Product resultBinary = binarySearch(products, 202);

        System.out.println("\nBinary Search Result: " + resultBinary);

    }

    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 left = 0, right = products.length - 1;

        while (left <= right) {

            int mid = left + (right - left) / 2;

            if (products[mid].getProductId() == id) {

                return products[mid];

            } else if (products[mid].getProductId() < id) {

                left = mid + 1;

            } else {

                right = mid - 1;

            }

        }

        return null;

    }

    static 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() {

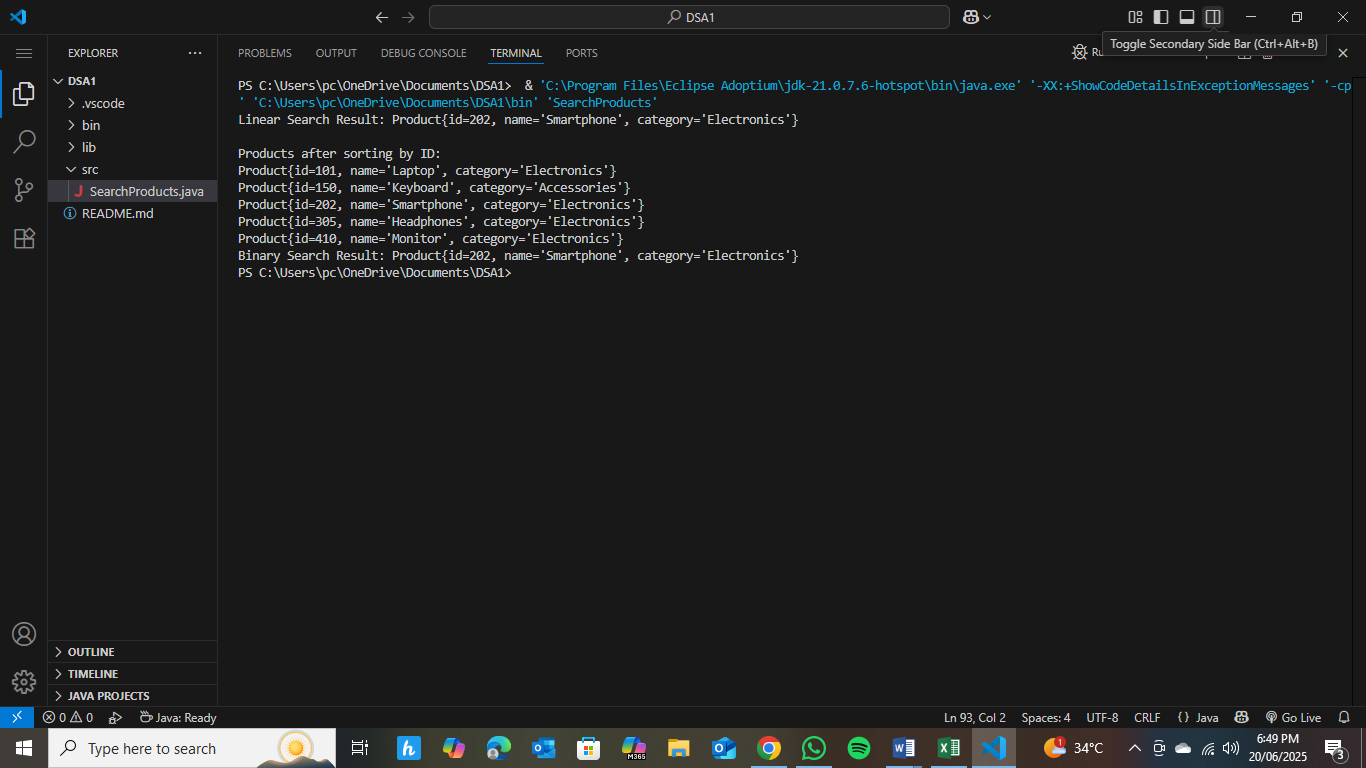
            return "Product{id=" + productId + ", name='" + productName + "', category='" + category + "'}";

        }

    }

}

**Output:**

****

**Hands-on 4: Financial Forecasting**

* **Forecasting.java**

public class forecasting {

    public static void main(String[] args) {

        double initialValue = 1000.0;

        double growthRate = 0.05;

        int periods = 5;

        double futureValue = calculateFutureValue(initialValue, growthRate, periods);

        System.out.println("Future Value after " + periods + " periods: $" + futureValue);

    }

    public static double calculateFutureValue(double value, double rate, int periods) {

        if (periods == 0) {

            return value;

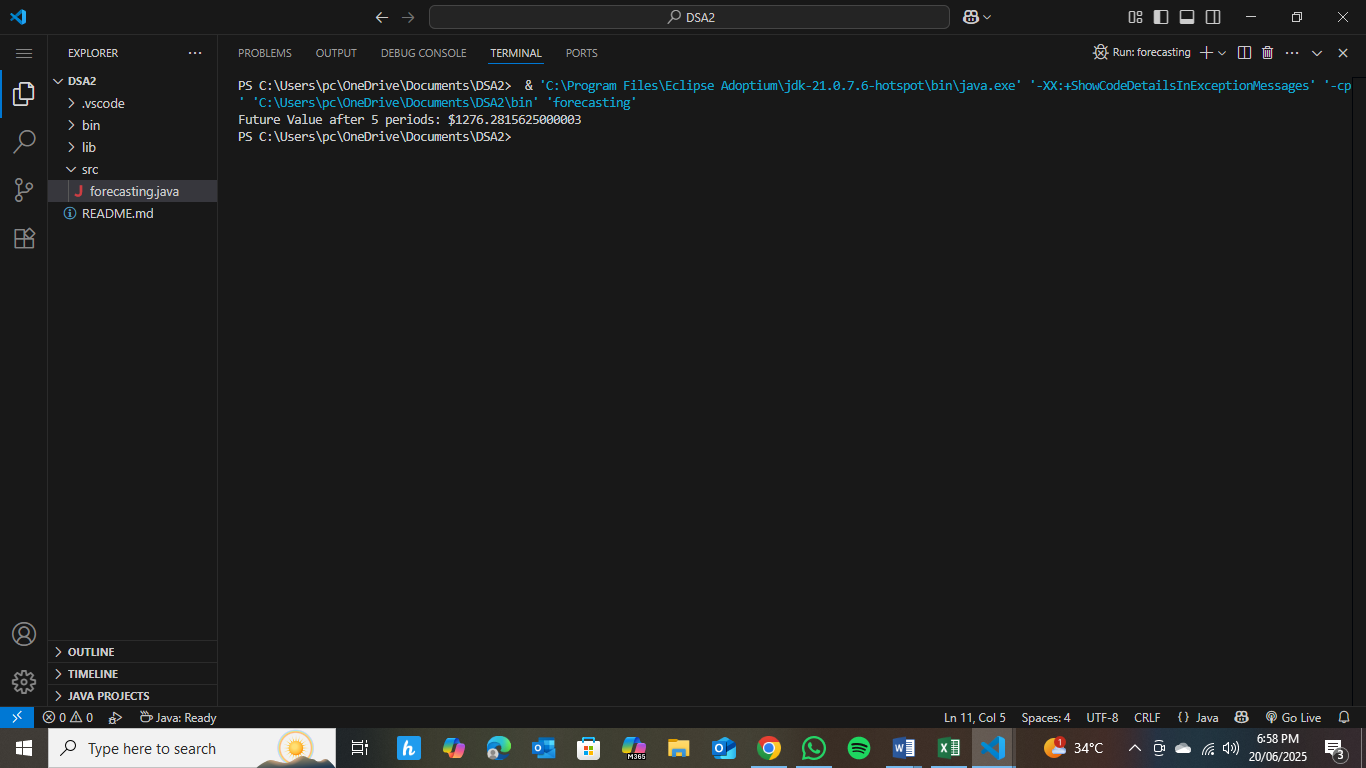
        }

        return calculateFutureValue(value \* (1 + rate), rate, periods - 1);

    }

}

**Output:**

****