**WEEK 1 HANDS-ON**

**DESIGN PATTERNS AND PRINCIPLES**

**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.

**Steps:**

1. **Create a New Java Project:**
   * Create a new Java project named **SingletonPatternExample**.
2. **Define a Singleton Class:**
   * Create a class named Logger that has a private static instance of itself.
   * Ensure the constructor of Logger is private.
   * Provide a public static method to get the instance of the Logger class.
3. **Implement the Singleton Pattern:**
   * Write code to ensure that the Logger class follows the Singleton design pattern.
4. **Test the Singleton Implementation:**
   * Create a test class to verify that only one instance of Logger is created and used across the application.

**CODE:**

**//Logger.java**

import java.util.\*;

public class Logger {

    private static Logger instance;

    private Logger() {

    }

    public static Logger getInstance() {

        if (instance == null) {

            instance = new Logger();

        }

        return instance;

    }

    public void log(String message) {

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

    }

}

**//Main.java**

import java.util.\*;

public class Main {

    public static void main(String[] args) {

        Logger logger1 = Logger.getInstance();

        Logger logger2 = Logger.getInstance();

        logger1.log("This is the first log message");

        logger2.log("This is the second log message");

        if (logger1 == logger2) {

            System.out.println("Both loggers are same");

        } else {

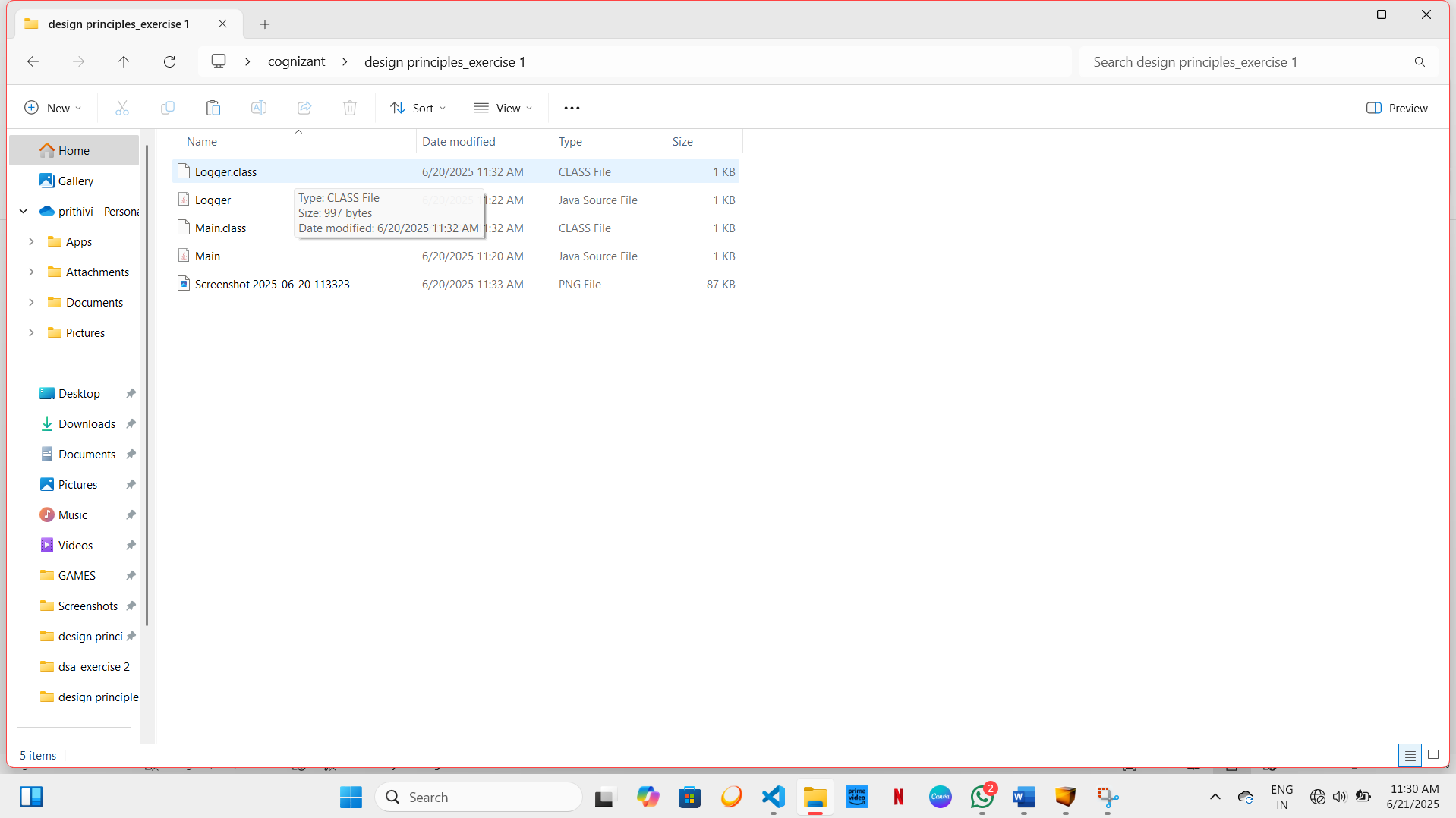
            System.out.println("Different instances exist");

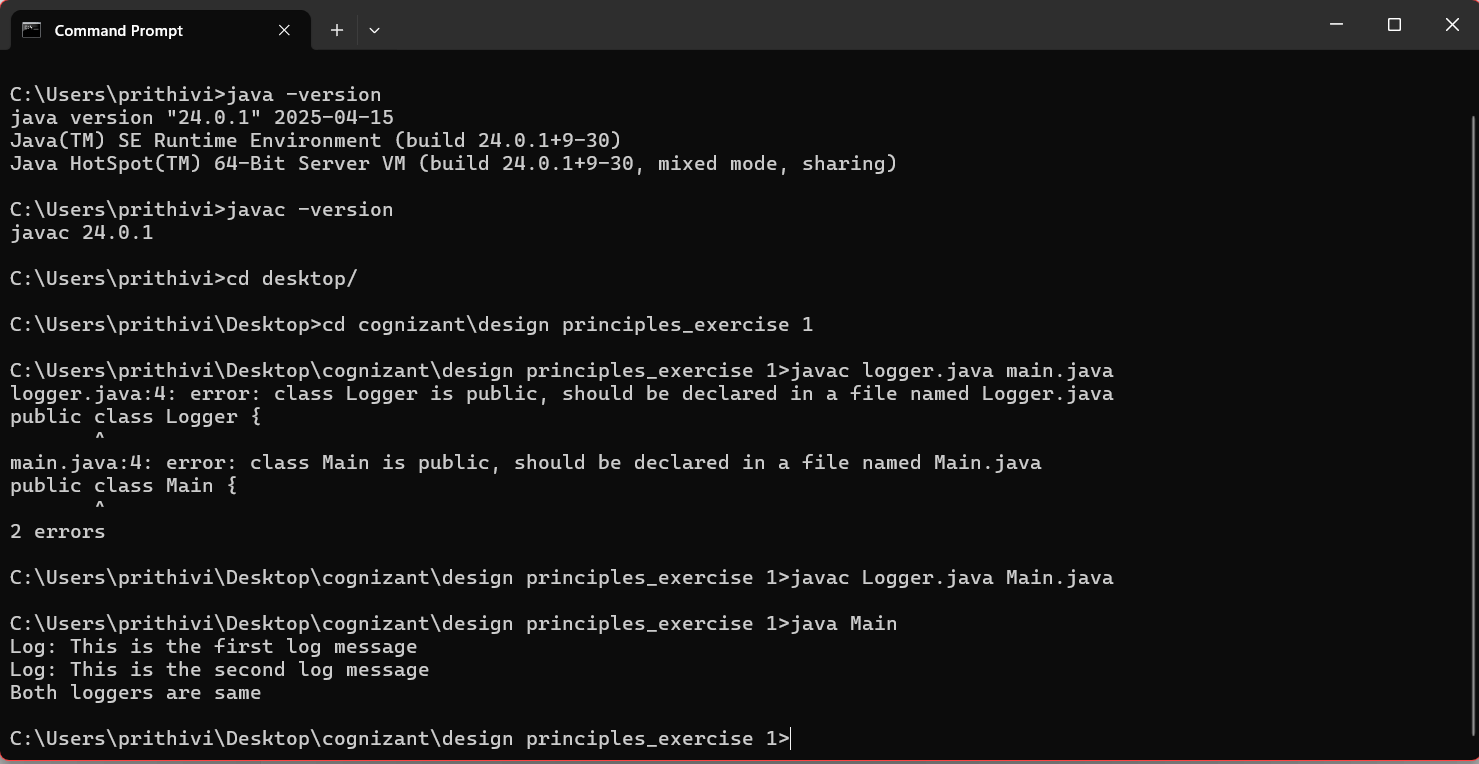
        }

    }

}

**OUTPUT:**

****



**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.

**Steps:**

1. **Create a New Java Project:**
   * Create a new Java project named **FactoryMethodPatternExample**.
2. **Define Document Classes:**
   * Create interfaces or abstract classes for different document types such as **WordDocument**, **PdfDocument**, and **ExcelDocument**.
3. **Create Concrete Document Classes:**
   * Implement concrete classes for each document type that implements or extends the above interfaces or abstract classes.
4. **Implement the Factory Method:**
   * Create an abstract class **DocumentFactory** with a method **createDocument()**.
   * Create concrete factory classes for each document type that extends DocumentFactory and implements the **createDocument()** method.
5. **Test the Factory Method Implementation:**
   * Create a test class to demonstrate the creation of different document types using the factory method.

**CODE:**

**//Main.java**

import java.util.\*;

public class Main {

    interface Document {

        void open();

    }

    static class WordDocument implements Document

    {

        public void open() {

            System.out.println("opening word doc");

        }

    }

    static class PdfDocument implements Document {

        public void open() {

            System.out.println("opening PDF doc");

        }

    }

    static class ExcelDocument implements Document {

        public void open() {

            System.out.println("opening excel doc");

        }

    }

    abstract static class DocumentFactory {

        public abstract Document createDocument();

    }

    static class WordDocumentFactory extends DocumentFactory {

        public Document createDocument() {

            return new WordDocument();

        }

    }

    static class PdfDocumentFactory extends DocumentFactory {

        public Document createDocument() {

            return new PdfDocument();

        }

    }

    static class ExcelDocumentFactory extends DocumentFactory {

        public Document createDocument() {

            return new ExcelDocument();

        }

    }

    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();

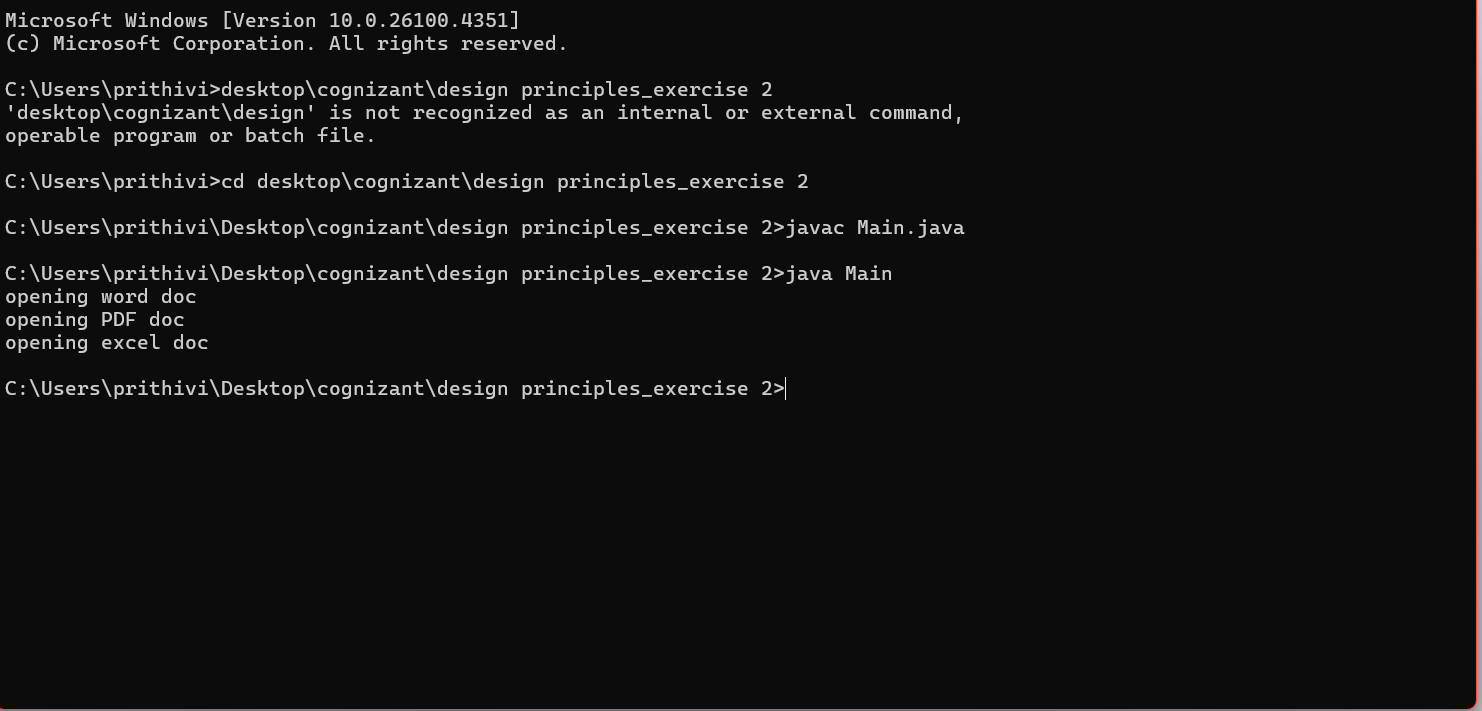
        Document excel = excelFactory.createDocument();

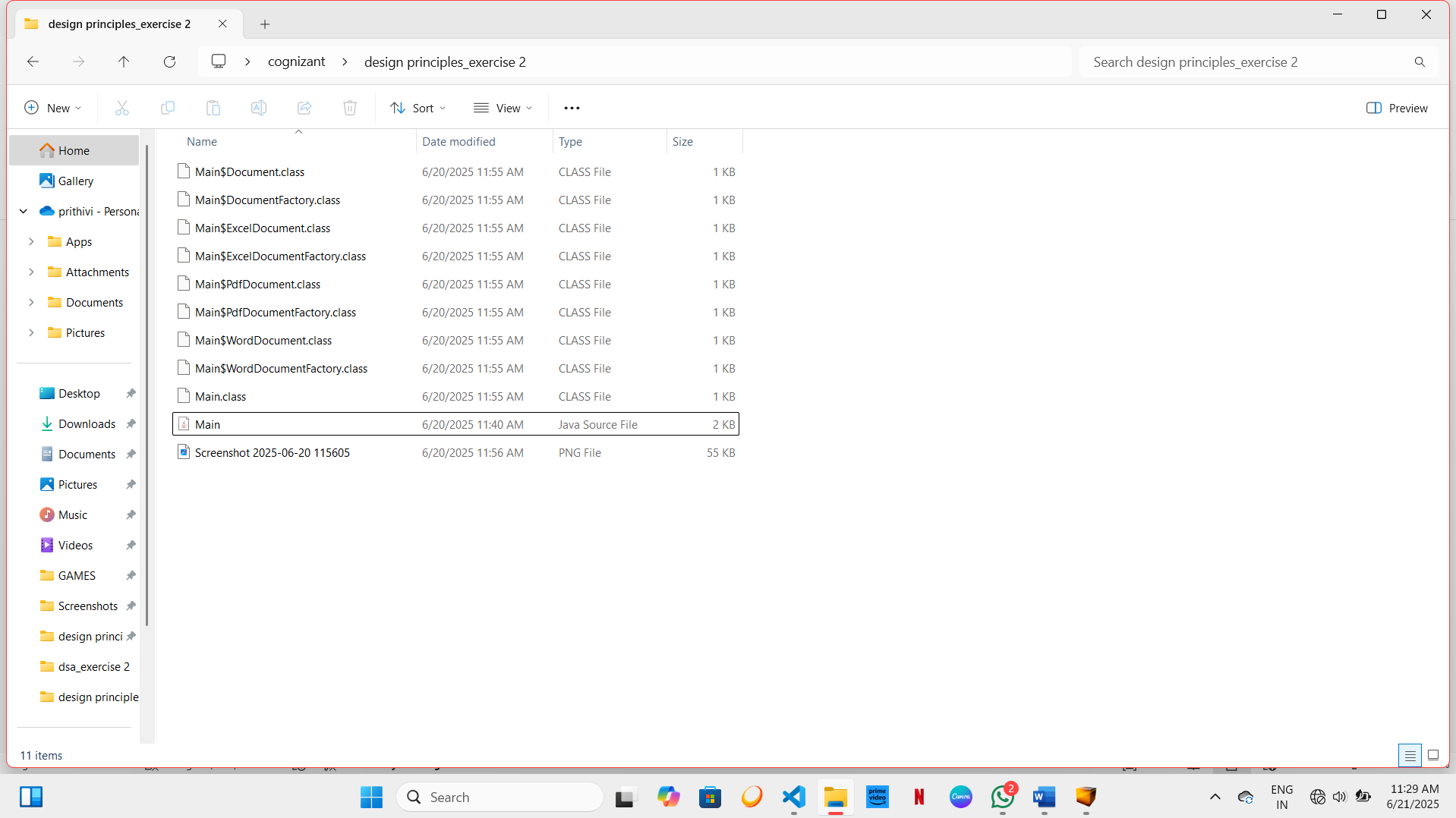
        excel.open();

    }

}

**OUTPUT**



****

**DATA STRUCTURES AND ALGORITHM**

**Exercise 2: E-commerce Platform Search Function**

**Scenario:**

You are working on the search functionality of an e-commerce platform. The search needs to be optimized for fast performance.

**Steps:**

1. **Understand Asymptotic Notation:**
   * Explain Big O notation and how it helps in analyzing algorithms.
   * Describe the best, average, and worst-case scenarios for search operations.
2. **Setup:**
   * Create a class **Product** with attributes for searching, such as **productId, productName**, and **category**.
3. **Implementation:**
   * Implement linear search and binary search algorithms.
   * Store products in an array for linear search and a sorted array for binary search.
4. **Analysis:**
   * Compare the time complexity of linear and binary search algorithms.
   * Discuss which algorithm is more suitable for your platform and why.

**CODE:**

**//Main.java**

import java.util.\*;

public class Main {

    static class Product {

        int id;

        String name;

        String category;

        Product(int id, String name, String category) {

            this.id = id;

            this.name = name;

            this.category = category;

        }

        void showDetails() {

            System.out.println("ID: " + id + ", Name: " + name + ", Category: " + category);

        }

    }

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

        for (int i = 0; i < products.length; i++) {

            if (products[i].name.equalsIgnoreCase(targetName)) {

                return products[i];

            }

        }

        return null;

    }

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

        int start = 0;

        int end = products.length - 1;

        while (start <= end) {

            int mid = (start + end) / 2;

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

            if (compare == 0) {

                return products[mid];

            } else if (compare < 0) {

                end = mid - 1;

            } else {

                start = mid + 1;

            }

        }

        return null;

    }

    static void sortByName(Product[] products)

    {

        for (int i = 0; i < products.length - 1; i++)

        {

            for (int j = 0; j < products.length - i - 1; j++)

            {

                if (products[j].name.compareToIgnoreCase(products[j + 1].name) > 0)

                {

                    Product temp = products[j];

                    products[j] = products[j + 1];

                    products[j + 1] = temp;

                }

            }

        }

    }

    public static void main(String[] args)

 {

        Product[] productList = new Product[5];

        productList[0] = new Product(1, "Laptop", "Electronics");

        productList[1] = new Product(2, "Shoes", "Fashion");

        productList[2] = new Product(3, "Mobile", "Electronics");

        productList[3] = new Product(4, "Watch", "Accessories");

        productList[4] = new Product(5, "Camera", "Electronics");

        System.out.println("Linear Search");

        Product found1 = linearSearch(productList, "Camera");

        if (found1 != null) {

            found1.showDetails();

        }

        else {

            System.out.println("Product not found.");

        }

        System.out.println("\nSorting products in binary search");

        sortByName(productList);

        System.out.println("Binary Search");

        Product found2 = binarySearch(productList, "Camera");

        if (found2 != null) {

            found2.showDetails();

        } else {

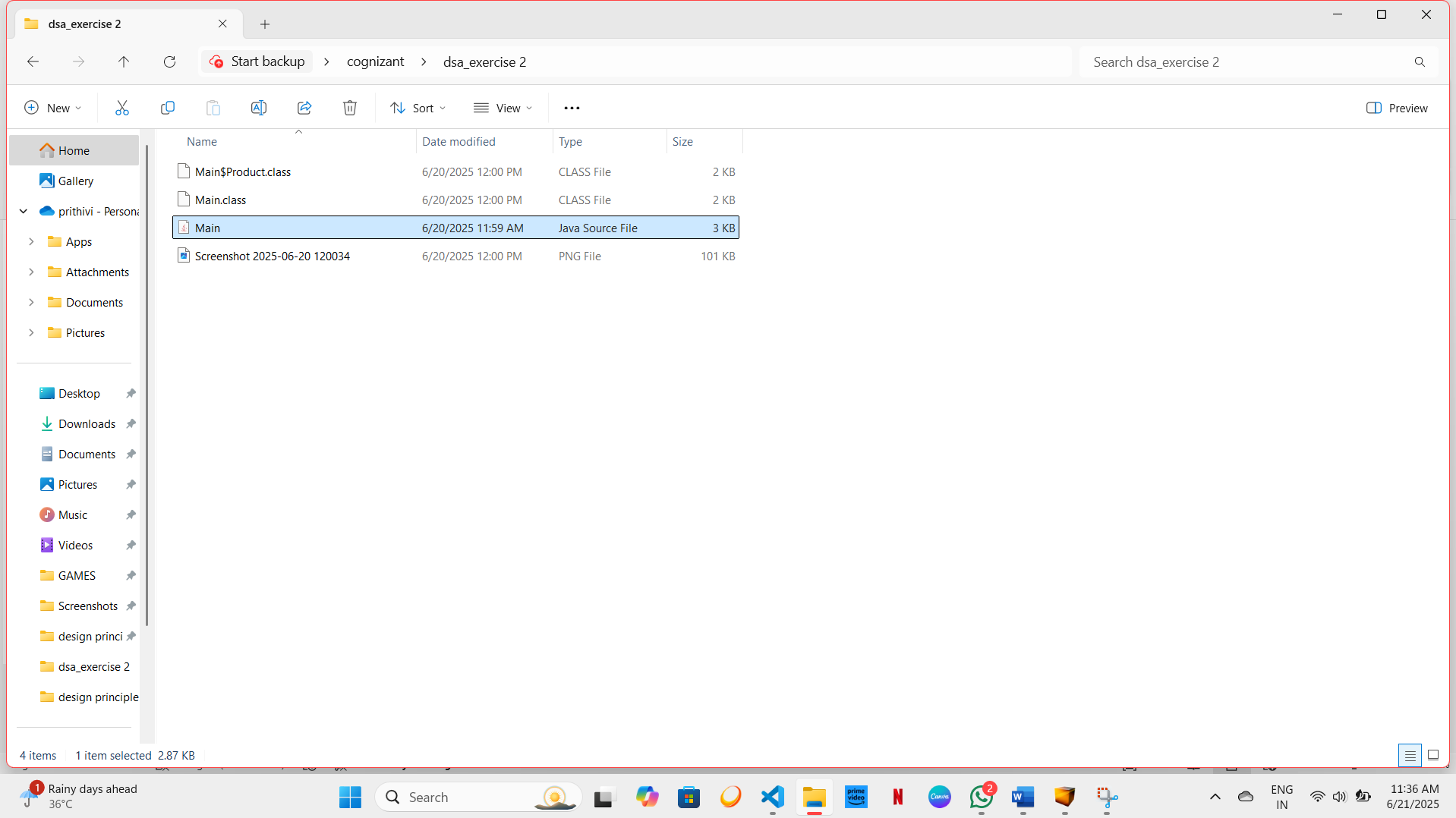
            System.out.println("Product not found.");

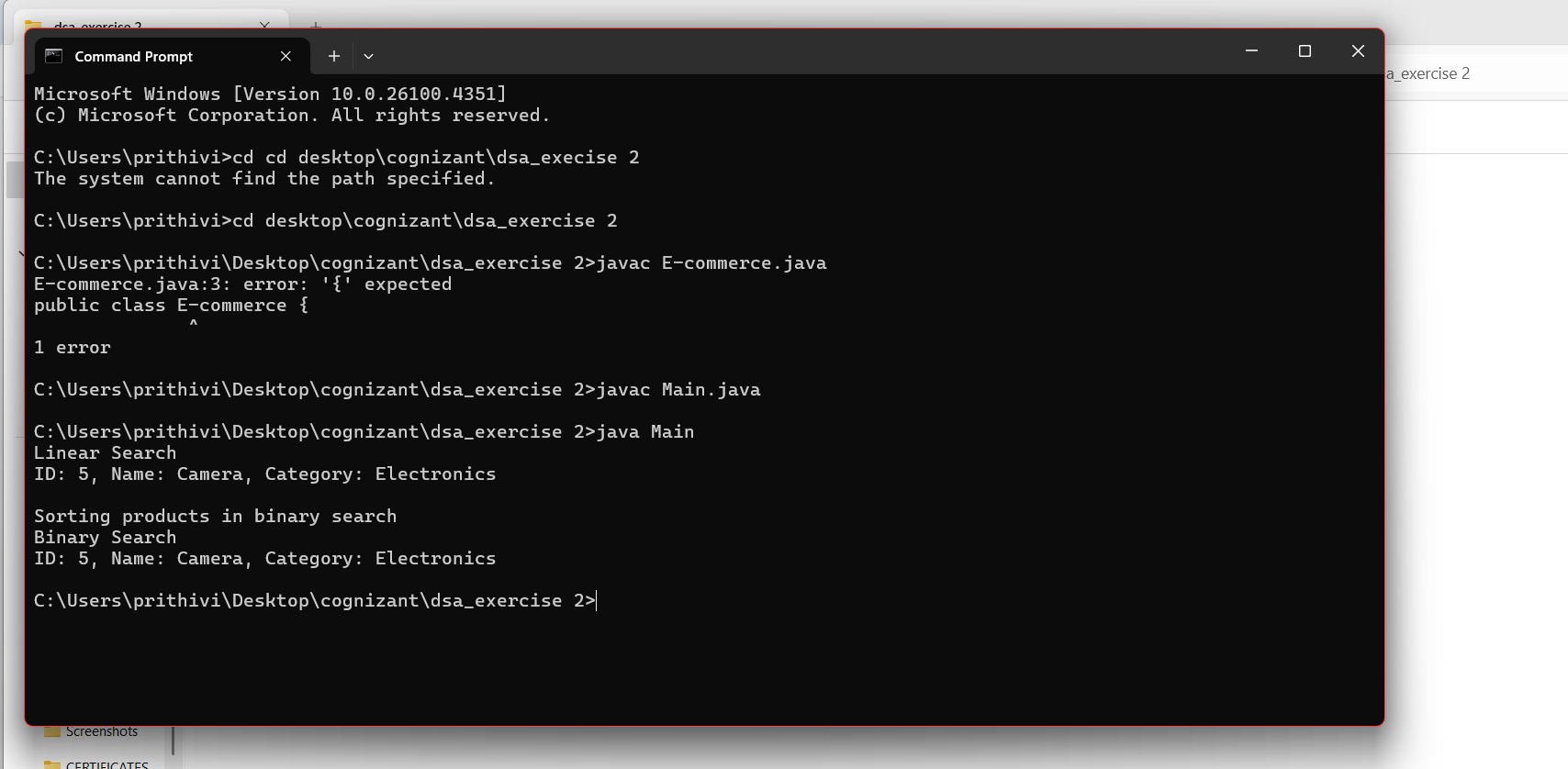
        }

    }

}

**OUTPUT:**

****



**Exercise 7: Financial Forecasting**

**Scenario:**

You are developing a financial forecasting tool that predicts future values based on past data.

**Steps:**

1. **Understand Recursive Algorithms:**
   * Explain the concept of recursion and how it can simplify certain problems.
2. **Setup:**
   * Create a method to calculate the future value using a recursive approach.
3. **Implementation:**
   * Implement a recursive algorithm to predict future values based on past growth rates.
4. **Analysis:**
   * Discuss the time complexity of your recursive algorithm.
   * Explain how to optimize the recursive solution to avoid excessive computation.

**CODE:**

**//Main.java**import java.util.\*;

public class Main {

    public static double calculateFutureValue(double presentValue, double rate, int years)

    {

        if (years == 0) {

            return presentValue;

        }

        else {

            return calculateFutureValue(presentValue \* (1 + rate), rate, years - 1);

        }

    }

    public static double calculateFutureValueOptimized(double presentValue, double rate, int years) {

        return presentValue \* Math.pow(1 + rate, years);

    }

    public static void main(String[] args) {

        double presentValue = 1000;

        double growthRate = 0.10;

        int years = 5;

        double futureValue = calculateFutureValue(presentValue, growthRate, years);

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

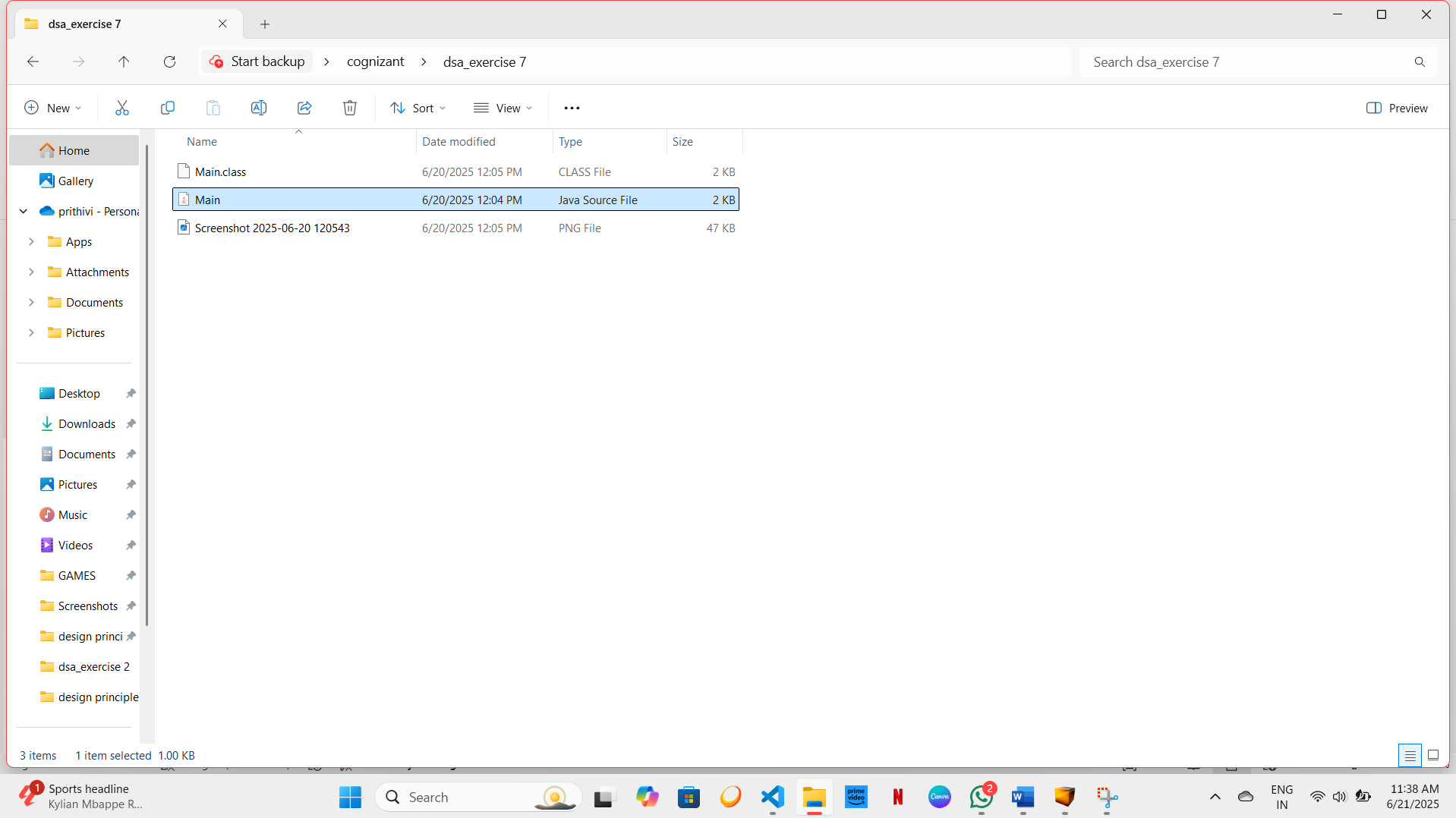
        double optimizedValue = calculateFutureValueOptimized(presentValue, growthRate, years);

        System.out.println("Future Value Optimized: " + optimizedValue);

    }

}

**OUTPUT:**

****

