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 LogManager {
    private static LogManager uniqueInstance;

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

    public static LogManager getInstance() {
        if (uniqueInstance == null) {
            uniqueInstance = new LogManager();
        }
        return uniqueInstance;
    }

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

public class Main {
    public static void main(String[] args) {
        LogManager log1 = LogManager.getInstance();
        log1.log("First message");
        LogManager log2 = LogManager.getInstance();
        log2.log("Second message");
        System.out.println("Are both loggers the same instance? " + (log1 == log2));
    }
}
```

Output:

Main.java

C: > Users > keert > OneDrive > Desktop > React > new > src > Main.java

```
 1   class LogManager {
 8       public static LogManager getInstance() {
11           }
12           return uniqueInstance;
13       }
14
15       public void log(String message) {
16           System.out.println("Log: " + message);
17       }
18   }
19
20   public class Main {
21       public static void main(String[] args) {
22           LogManager log1 = LogManager.getInstance();
23           log1.log("First message");
24           LogManager log2 = LogManager.getInstance();
25           log2.log("Second message");
26           System.out.println("Are both loggers the same instance? " + (log1 == log2));
27       }
28   }
29
```

input

```
Logger instance created.
Log: First message
Log: Second message
Are both loggers the same instance? true
```

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.

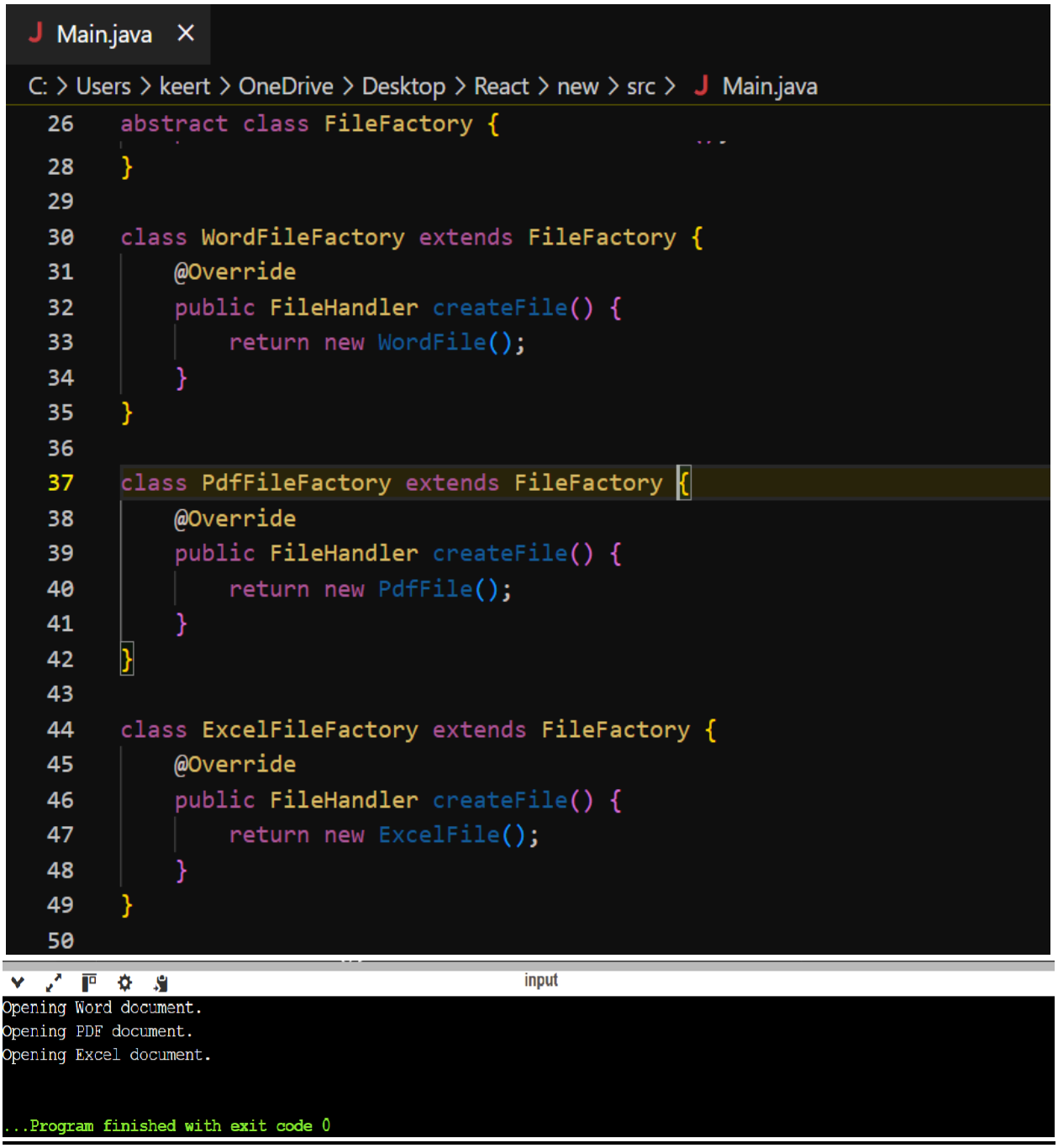
Code:

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

class WordFile implements FileHandler {
    @Override
    public void open() {
```

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

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

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

abstract class FileFactory {
    public abstract FileHandler createFile();
}

class WordFileFactory extends FileFactory {
    @Override
    public FileHandler createFile() {
        return new WordFile();
    }
}

class PdfFileFactory extends FileFactory {
    @Override
    public FileHandler createFile() {
        return new PdfFile();
    }
}

class ExcelFileFactory extends FileFactory {
    @Override
    public FileHandler createFile() {
        return new ExcelFile();
    }
}

public class Main {
    public static void main(String[] args) {
        FileFactory wordCreator = new WordFileFactory();
        FileHandler word = wordCreator.createFile();
        word.open();

        FileFactory pdfCreator = new PdfFileFactory();
```

```
        FileHandler pdf = pdfCreator.createFile();
        pdf.open();

        FileFactory excelCreator = new ExcelFileFactory();
        FileHandler excel = excelCreator.createFile();
        excel.open();
    }
}
```

Output:



Data structures and Algorithms

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.

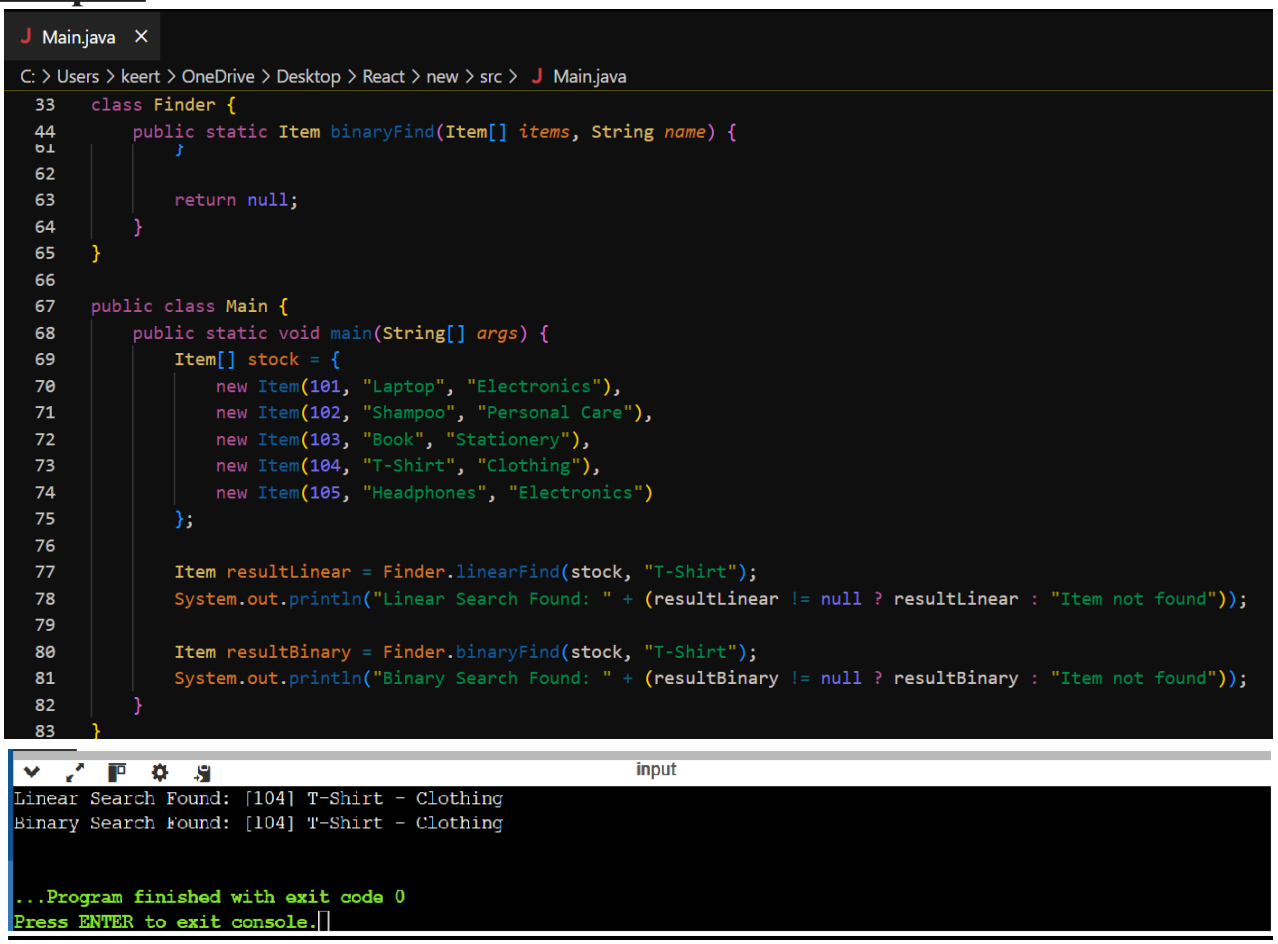
Code:

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

class Item {
    private int itemId;
    private String itemName;
    private String type;

    public Item(int itemId, String itemName, String type) {
        this.itemId = itemId;
        this.itemName = itemName;
        this.type = type;
    }

    public int getItemId() {
        return itemId;
    }

    public String getItemName() {
        return itemName;
    }

    public String getType() {
        return type;
    }

    @Override
    public String toString() {
        return "[" + itemId + "] " + itemName + " - " + type;
    }
}

class Finder {
```

```
    public static Item linearFind(Item[] items, String name) {
        for (Item i : items) {
            if (i.getItemName().equalsIgnoreCase(name)) {
                return i;
            }
        }
        return null;
    }

    public static Item binaryFind(Item[] items, String name) {
        Arrays.sort(items, Comparator.comparing(Item::getItemName,
String.CASE_INSENSITIVE_ORDER));

        int low = 0;
        int high = items.length - 1;

        while (low <= high) {
            int mid = (low + high) / 2;
            int cmp = items[mid].getItemName().compareToIgnoreCase(name);

            if (cmp == 0) {
                return items[mid];
            } else if (cmp < 0) {
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }

        return null;
    }
}

public class Main {
    public static void main(String[] args) {
        Item[] stock = {
            new Item(101, "Laptop", "Electronics"),
            new Item(102, "Shampoo", "Personal Care"),
            new Item(103, "Book", "Stationery"),
            new Item(104, "T-Shirt", "Clothing"),
            new Item(105, "Headphones", "Electronics")
        };

        Item resultLinear = Finder.linearFind(stock, "T-Shirt");
        System.out.println("Linear Search Found: " + (resultLinear != null ? resultLinear : "Item
not found"));
```

```
        Item resultBinary = Finder.binaryFind(stock, "T-Shirt");
        System.out.println("Binary Search Found: " + (resultBinary != null ? resultBinary :
"Item not found"));
    }
}
```

Output:



Exercise 7: Financial Forecasting

Scenario:

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

Code:

```
public class Main {

    public static double calculateFutureRecursive(double principal, double growthRate, int
duration) {
        if (duration == 0) {
            return principal;
        }
        return calculateFutureRecursive(principal, growthRate, duration - 1) * (1 +
growthRate);
    }

    public static double calculateFutureIterative(double principal, double growthRate, int
duration) {
        for (int i = 0; i < duration; i++) {
            principal *= (1 + growthRate);
        }
        return principal;
    }

    public static void main(String[] args) {
        double baseAmount = 10000.0;
        double yearlyRate = 0.07;
        int yearsAhead = 5;

        double valueByRecursion = calculateFutureRecursive(baseAmount, yearlyRate,
yearsAhead);
        System.out.printf("Recursive: Predicted value after %d years: ₹%.2f\n", yearsAhead,
valueByRecursion);

        double valueByIteration = calculateFutureIterative(baseAmount, yearlyRate,
yearsAhead);
        System.out.printf("Iterative: Predicted value after %d years: ₹%.2f\n", yearsAhead,
valueByIteration);
    }
}
```

Output:

Main.java

C: > Users > keert > OneDrive > Desktop > React > new > src > Main.java

```java
public class Main {

    public static double calculateFutureRecursive(double principal, double growthRate, int duration) {
        if (duration == 0) {
            return principal;
        }
        return calculateFutureRecursive(principal, growthRate, duration - 1) * (1 + growthRate);
    }

    public static double calculateFutureIterative(double principal, double growthRate, int duration) {
        for (int i = 0; i < duration; i++) {
            principal *= (1 + growthRate);
        }
        return principal;
    }

    public static void main(String[] args) {
        double baseAmount = 10000.0;
        double yearlyRate = 0.07;
        int yearsAhead = 5;

        double valueByRecursion = calculateFutureRecursive(baseAmount, yearlyRate, yearsAhead);
        System.out.printf("Recursive: Predicted value after %d years: ₹%.2f\n", yearsAhead, valueByRecursion);

```

input

```
Recursive: Predicted value after 5 years: ₹14025.52
Iterative: Predicted value after 5 years: ₹14025.52


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