

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:

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

The screenshot shows an IDE window titled 'Main.java'. The code defines an abstract class 'FileFactory' and three concrete subclasses: 'WordFileFactory', 'PdfFileFactory', and 'ExcelFileFactory'. Each subclass overrides the 'createFile()' method to return a new instance of 'WordFile', 'PdfFile', or 'ExcelFile' respectively. The output console at the bottom shows the program's execution: 'Opening Word document.', 'Opening PDF document.', 'Opening Excel document.', and finally '...Program finished with exit code 0'.

```

J Main.java X
C: > Users > keert > OneDrive > Desktop > React > new > src > J Main.java
26  abstract class FileFactory {
27      ...
28  }
29
30  class WordFileFactory extends FileFactory {
31      @Override
32      public FileHandler createFile() {
33          return new WordFile();
34      }
35  }
36
37  class PdfFileFactory extends FileFactory {
38      @Override
39      public FileHandler createFile() {
40          return new PdfFile();
41      }
42  }
43
44  class ExcelFileFactory extends FileFactory {
45      @Override
46      public FileHandler createFile() {
47          return new ExcelFile();
48      }
49  }
50
input
Opening Word document.
Opening PDF document.
Opening Excel document.
...Program finished with exit code 0

```

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"));
    }
}

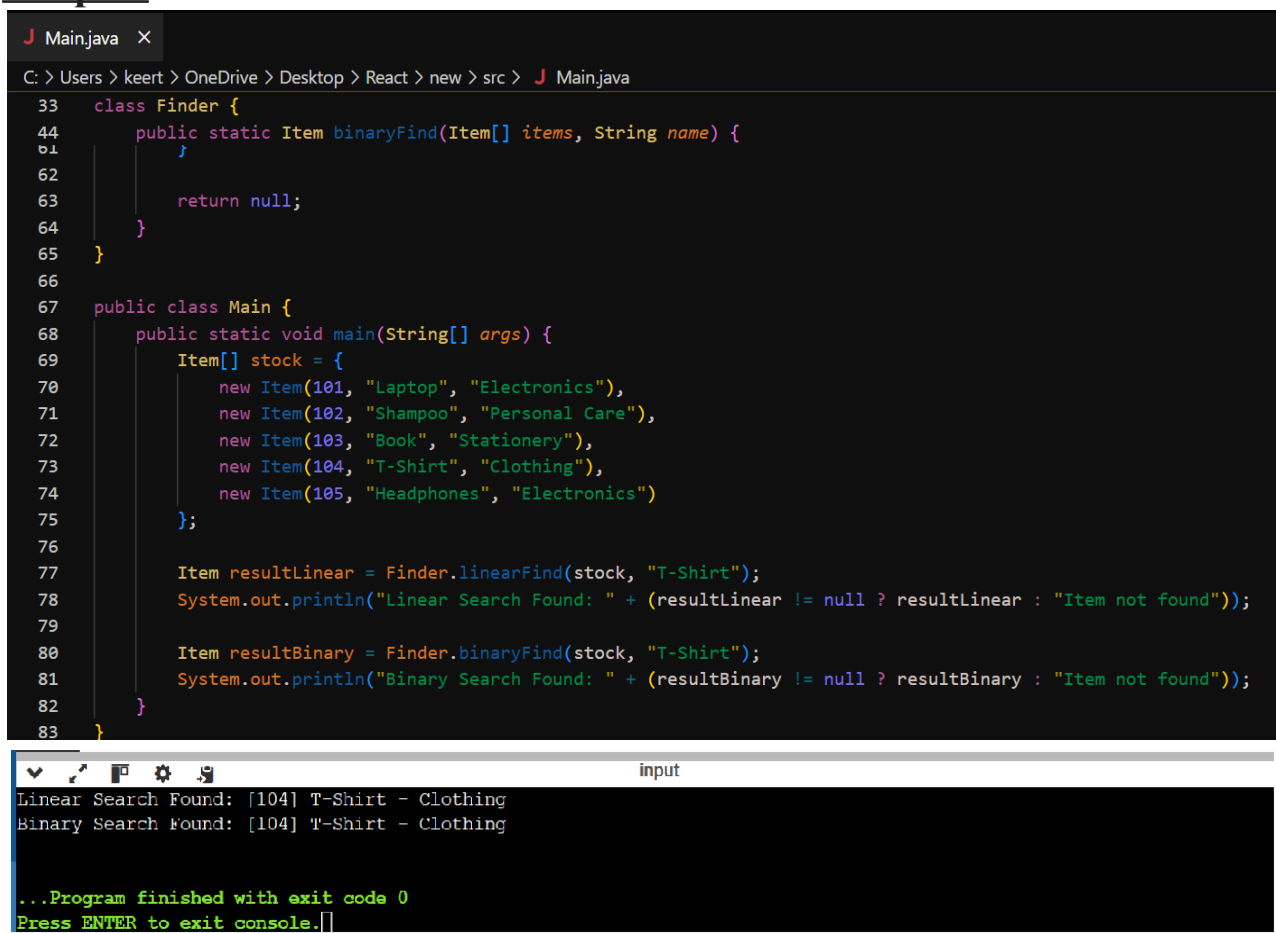
```

```

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

```

Output:



The screenshot shows a Java IDE with a file named `Main.java`. The code defines a `Finder` class with a `binaryFind` method and a `Main` class with a `main` method. The `main` method creates an array of `Item` objects and performs both linear and binary searches for a "T-Shirt". The console output shows the results of these searches.

```

J Main.java X
C: > Users > keert > OneDrive > Desktop > React > new > src > J Main.java
33  class Finder {
44      public static Item binaryFind(Item[] items, String name) {
61          }
62
63      return null;
64  }
65  }
66
67  public class Main {
68      public static void main(String[] args) {
69          Item[] stock = {
70              new Item(101, "Laptop", "Electronics"),
71              new Item(102, "Shampoo", "Personal Care"),
72              new Item(103, "Book", "Stationery"),
73              new Item(104, "T-Shirt", "Clothing"),
74              new Item(105, "Headphones", "Electronics")
75          };
76
77          Item resultLinear = Finder.linearFind(stock, "T-Shirt");
78          System.out.println("Linear Search Found: " + (resultLinear != null ? resultLinear : "Item not found"));
79
80          Item resultBinary = Finder.binaryFind(stock, "T-Shirt");
81          System.out.println("Binary Search Found: " + (resultBinary != null ? resultBinary : "Item not found"));
82      }
83  }

```

input

```

Linear Search Found: [104] T-Shirt - Clothing
Binary Search Found: [104] T-Shirt - Clothing

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

```

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:

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

1 public class Main {
2
3     public static double calculateFutureRecursive(double principal, double growthRate, int duration) {
4         if (duration == 0) {
5             return principal;
6         }
7         return calculateFutureRecursive(principal, growthRate, duration - 1) * (1 + growthRate);
8     }
9
10    public static double calculateFutureIterative(double principal, double growthRate, int duration) {
11        for (int i = 0; i < duration; i++) {
12            principal *= (1 + growthRate);
13        }
14        return principal;
15    }
16
17    public static void main(String[] args) {
18        double baseAmount = 10000.0;
19        double yearlyRate = 0.07;
20        int yearsAhead = 5;
21
22        double valueByRecursion = calculateFutureRecursive(baseAmount, yearlyRate, yearsAhead);
23        System.out.printf("Recursive: Predicted value after %d years: ₹%.2f\n", yearsAhead, valueByRecursion);
24    }
25 }
```

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.
```