WEEK 1 : EXCERCISES

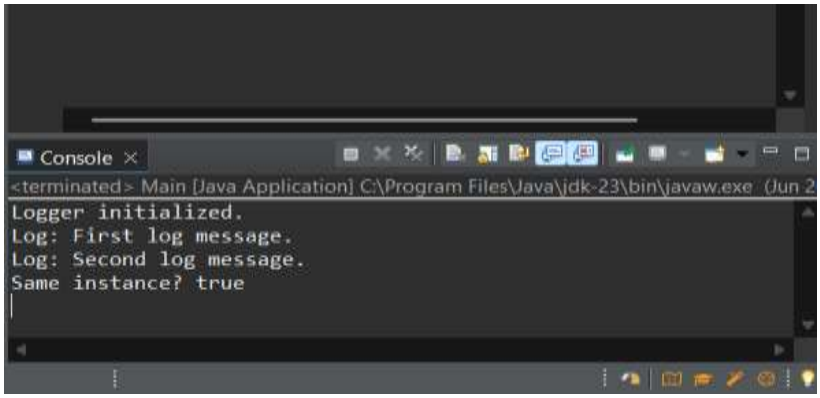
DESIGN PATTERNS

1. Implementing the Singleton Pattern

```
2. package ctsjavaexcer;
3. public class Logger {
4.     private static Logger instance;
5.
6.     private Logger() {
7.         System.out.println("Logger initialized.");
8.     }
9.
10.    public static Logger getInstance() {
11.        if (instance == null)
12.            instance = new Logger();
13.        return instance;
14.    }
15.
16.    public void log(String message) {
17.        System.out.println("Log: " + message);
18.    }
19. }
```

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

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

        System.out.println("Same instance? " + (logger1 == logger2));
    }
}
```

OUTPUT:



2. Implementing the Factory Method Pattern

```java
package ctsjavaexcer;
public interface Document {
    void open();
}
```

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

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

```java
package ctsjavaexcer;
public class ExcelFactory extends DocumentFactory {
    public Document createDocument() {
        return new ExcelDocument();
    }
}
```

```java
package ctsjavaexcer;
public class PdfDocument implements Document {
    public void open() {
        System.out.println("Opening PDF Document");
    }
}
```

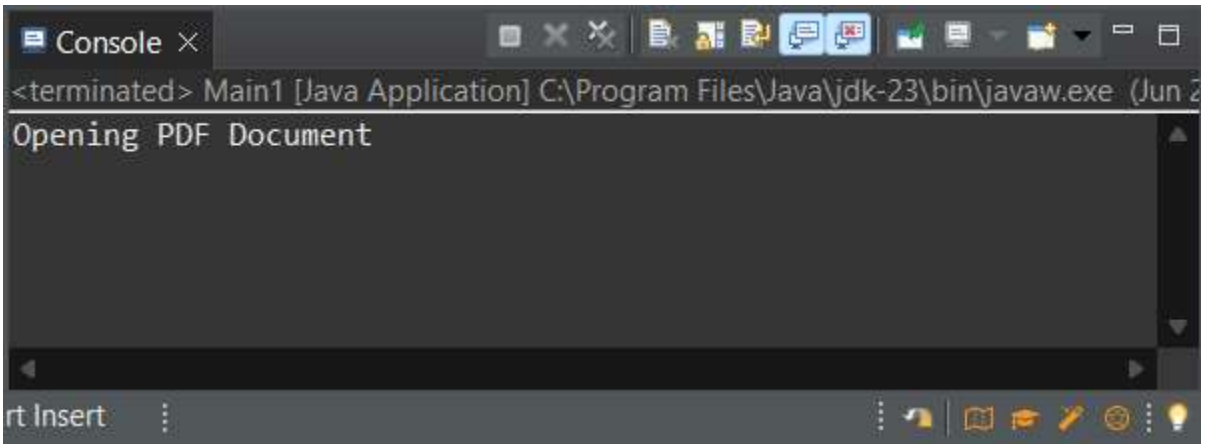
```java
package ctsjavaexcer;
public class PdfFactory extends DocumentFactory {
    public Document createDocument() {
        return new PdfDocument();
    }
}
```

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

```java
package ctsjavaexcer;
public class WordFactory extends DocumentFactory {
    public Document createDocument() {
        return new WordDocument();
    }
}
```

```java
package ctsjavaexcer;

public class Main1{
    public static void main(String[] args) {
        DocumentFactory factory = new PdfFactory();
        Document doc = factory.createDocument();
        doc.open();
    }
}
```

OUTPUT:



ALGORITHMS AND DATA STRUCTURE

1.

```java
package ctsjavaexcer;
public class FinancialForecast {

 // Recursive method to calculate future value
 public static double calculateFutureValue(double presentValue, double growthRate,
int years) {
     if (years == 0)
         return presentValue;
```

```java
    return calculateFutureValue(presentValue * (1 + growthRate), growthRate, years -
1);
 }

 public static double calculateFutureValueMemo(double presentValue, double
growthRate, int years, Double[] memo) {
    if (years == 0)
        return presentValue;
    if (memo[years] != null)
        return memo[years];

    memo[years] = calculateFutureValueMemo(presentValue, growthRate, years - 1,
memo) * (1 + growthRate);
    return memo[years];
 }

 public static void main(String[] args) {
    double presentValue = 10000;
    double growthRate = 0.10; // 10%
    int years = 5;

    System.out.println(" Recursive Forecast:");
    double result = calculateFutureValue(presentValue, growthRate, years);
    System.out.printf("Value after %d years: ₹%.2f%n", years, result);

    System.out.println("\n⚡ Optimized Recursive Forecast with Memoization:");
    Double[] memo = new Double[years + 1];
    double memoResult = calculateFutureValueMemo(presentValue, growthRate, years,
memo);
    System.out.printf("Value after %d years (Memoized): ₹%.2f%n", years,
memoResult);
 }
}
```

OUTPUT:

```
Console
<terminated> FinancialForecast [Java Application] C:\Program Files\Java\jdk-23\bin\javaw
Value after 5 years: ₹16105.10

 Optimized Recursive Forecast with Memoization:
Value after 5 years (Memoized): ₹16105.10
```

2.

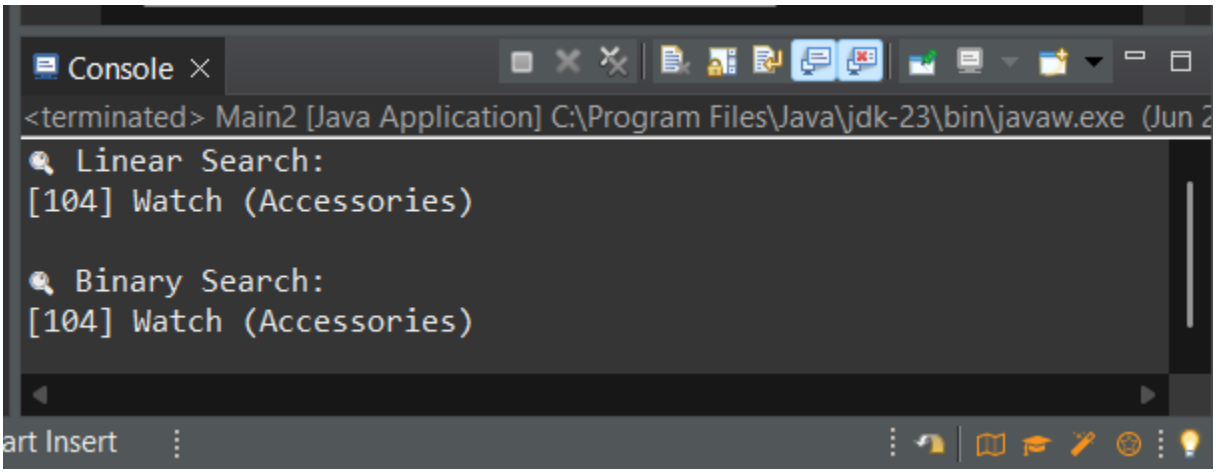
```java
package ctsjavaexcer;
//Product.java
public class Product {
 private int productId;
 private String productName;
 private String category;

 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 + ")";
 }
}
```

```java
package ctsjavaexcer;
//SearchEngine.java
import java.util.Arrays;
import java.util.Comparator;

public class SearchEngine {

 // Linear Search by product name
 public static Product linearSearch(Product[] products, String targetName) {
     for (Product p : products) {
         if (p.getProductName().equalsIgnoreCase(targetName)) {
             return p;
         }
     }
     return null;
 }

 // Binary Search by product name (array must be sorted)
 public static Product binarySearch(Product[] products, String targetName) {
     int left = 0;
     int right = products.length - 1;

     while (left <= right) {
         int mid = (left + right) / 2;
         int cmp = products[mid].getProductName().compareToIgnoreCase(targetName);

         if (cmp == 0)
             return products[mid];
         else if (cmp < 0)
```

```
            left = mid + 1;
        else
            right = mid - 1;
    }
    return null;
 }
}
```

```
package ctsjavaexcer;

//Main.java
import java.util.Arrays;
import java.util.Comparator;

public class Main2 {
 public static void main(String[] args) {
     Product[] products = {
         new Product(101, "Laptop", "Electronics"),
         new Product(102, "Shoes", "Fashion"),
         new Product(103, "Mobile", "Electronics"),
         new Product(104, "Watch", "Accessories"),
         new Product(105, "Tablet", "Electronics")
     };

     System.out.println("🔍 Linear Search:");
     Product result1 = SearchEngine.linearSearch(products, "Watch");
     System.out.println(result1 != null ? result1 : "Product not found");

     System.out.println("\n🔍 Binary Search:");
     Arrays.sort(products, Comparator.comparing(Product::getProductName)); //
Important
     Product result2 = SearchEngine.binarySearch(products, "Watch");
     System.out.println(result2 != null ? result2 : "Product not found");
 }
}
```

OUTPUT:


Console
<terminated> Main2 [Java Application] C:\Program Files\Java\jdk-23\bin\javaw.exe (Jun 2
Linear Search:
[104] Watch (Accessories)
Binary Search:
[104] Watch (Accessories)
art Insert