**Design Principles and Patterns:**

**Exercise 1: Implementing the Singleton Pattern**

**Logger.java**  
  
public class Logger {

private static Logger onlyLogger;

private Logger() {

System.out.println("Logger Created");

}

public static Logger getInstance() {

if (onlyLogger == null) {

onlyLogger = new Logger();

}

return onlyLogger;

}

public void log(String message) {

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

}

}  
  
**Main.java**  
  
public class Main {

public static void main(String[] args) {

Logger l1 = Logger.getInstance();

l1.log("First message");

Logger l2 = Logger.getInstance();

l2.log("Second message");

if (l1 == l2) {

System.out.println("Both are the SAME Logger!");

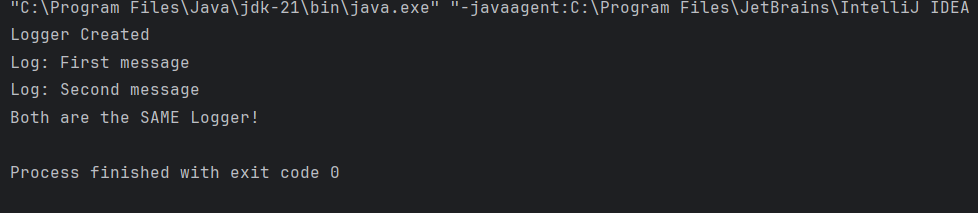
} else {

System.out.println("They are different");

}

}

}

OUTPUT:  
  


**Design Principles and Patterns:**

**Exercise 2: Implementing the Factory Method Pattern**

**Document.java**  
  
public interface Document {

void open();

}  
  
**WordDocument.java**  
  
public class WordDocument implements Document {

@Override

public void open() {

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

}

}  
  
**PdfDocument.java**  
  
public class PdfDocument implements Document {

@Override

public void open() {

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

}

}

**ExcelDocument.java**  
  
public class ExcelDocument implements Document {

@Override

public void open() {

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

}

}

**DocumentFactory.java**  
  
public abstract class DocumentFactory {

public abstract Document createDocument();

}

**WordDocumentFactory.java**  
  
public class WordDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new WordDocument();

}

}  
  
**PdfDocumentFactory.java**  
  
public class PdfDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new PdfDocument();

}

}

**ExcelDocumentFactory.java**  
  
public class ExcelDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new ExcelDocument();

}

}

**Main.java**  
  
public class Main {

public static void main(String[] args) {

// Word Document

DocumentFactory wordFactory = new WordDocumentFactory();

Document wordDoc = wordFactory.createDocument();

wordDoc.open();

DocumentFactory pdfFactory = new PdfDocumentFactory();

Document pdfDoc = pdfFactory.createDocument();

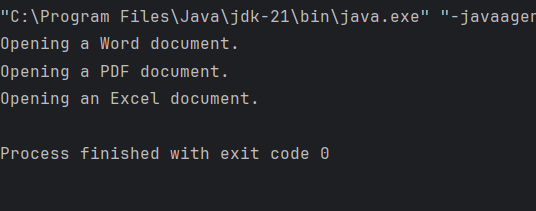
pdfDoc.open();

DocumentFactory excelFactory = new ExcelDocumentFactory();

Document excelDoc = excelFactory.createDocument();

excelDoc.open();

}

}  
  
  
**OUTPUT:**  


**Data Structures and Algorithms:**

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

**Product.java**  
  
public class Product {

int productId;

String productName;

String category;

public Product(int productId, String productName, String category) {

this.productId = productId;

this.productName = productName;

this.category = category;

}

public String toString() {

return productId + " - "+ productName + "(" + category +")";

}

}

**ProductSearch.java**  
  
import java.util.Arrays;

import java.util.Comparator;

public class ProductSearch

public static Product linearSearch(Product[] products, String name) {

for (Product p : products) {

if (p.productName.equalsIgnoreCase(name)) {

return p;

}

}

return null;

}

public static Product binarySearch(Product[] products, String name) {

Arrays.sort(products, Comparator.comparing(p -> p.productName.toLowerCase()));

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

while (left <= right) {

int mid = (left + right) / 2;

int cmp = name.compareToIgnoreCase(products[mid].productName);

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

else if (cmp < 0) right = mid - 1;

else left = mid + 1;

}

return null;

}

}  
  
**Main.java**  
  
public class Main {

public static void main(String[] args) {

Product[] products = {

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

new Product(102, "Shoe", "Fashion"),

new Product(103, "Laptops", "Electronics"),

new Product(104, "T-Shirts", "Fashion"),

new Product(105, "Headphone", "Electronics")

};

String searchName = "Laptops";

System.out.println("LinearSearch Result:");

Product linearResult = ProductSearch.linearSearch(products, searchName);

System.out.println(linearResult != null ? linearResult : "Not Found");

System.out.println("\n BinarySearch Result:");

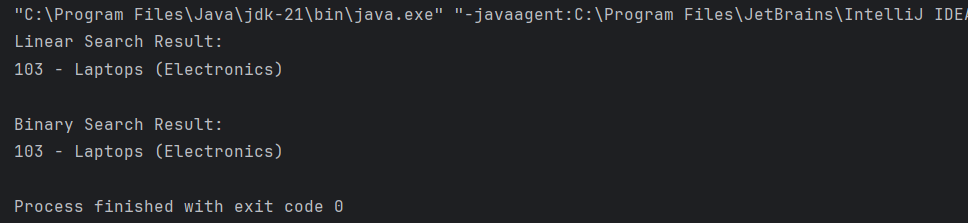
Product binaryResult = ProductSearch.binarySearch(products, searchName);

System.out.println(binaryResult != null ? binaryResult : "Not Found");

}

}

**OUTPUT:**

  
- Big O notation is a mathematical way to describe how fast or slow an algorithmbased on the size of the input, it helps us know whether our program is efficient or not.  
-It helps us know whether our approach is optimized or not even before we implement it.  
- Linear search has **O(1)** best case, **O(n)** average and worst case.

- Binary search, which requires sorted data, performs much better with **O(1)** best case and **O(log n)** average and worst case  
- Binary search is more efficient for large datasets, while linear search is simpler and works without sorting  
- Linear search has a time complexity of O(n), while binary search is more efficient with O(log n), provided the list is sorted. For large-scale e-commerce platforms with thousands of products, binary search is more optimized and scalable

-Hence, binary search is more suitable when performance is a priority, and the data can be maintained in a sorted format

**Data Structures and Algorithms:**

**Exercise 7: Financial Forecasting**

**FinancialForecast.java**  
public class FinancialForecast {

// Recursive

public static double forecastValue(double initialAmount, double growthRate, int years) {

// Base case

if (years == 0) return initialAmount;

// Recursive case

return forecastValue(initialAmount, growthRate, years - 1) \* (1 + growthRate);

}

public static void main(String[] args) {

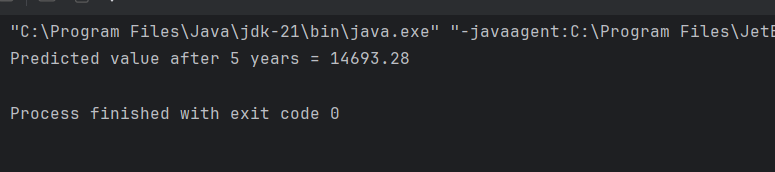
double initialAmount = 10000.0;

double growthRate = 0.08;

int years = 5;

double result = forecastValue(initialAmount, growthRate, years);

System.out.printf("Predicted value after %d years = %.2f\n", years, result);}}

OUTPUT:  
  
  
  
-Here,in the recursive approach the time and space complexities are both O(n)  
-Now,the problem here using the recurive approch is that it has the possiblity of generating risks like function call overhead or stack overfloa for large values of n  
-To tackle this we can use iterative approch where the time complexity is O(n) and space complexity is O(1).