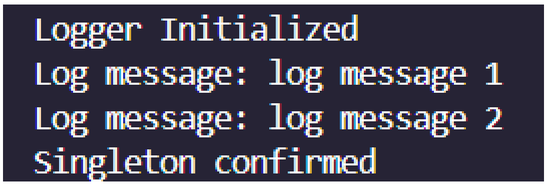
Design Patterns and Principles

Exercise 1: Implementing the Singleton Pattern

```
class Logger {
    private static Logger instance;

    private Logger() {
        System.out.println("Logger Initialized");
    }
    public static Logger getInstance() {
        if (instance == null) {
            instance = new Logger();
        }
        return instance;
    }

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

public class Singleton {

    public static void main(String[] args) {
        Logger logger1 = Logger.getInstance();
        logger1.log("log message 1");

        Logger logger2 = Logger.getInstance();
        logger2.log("log message 2");

        if (logger1 == logger2) {
            System.out.println("Singleton confirmed");
        } else {
            System.out.println("Different instances");
        }
    }
}
```

Output:



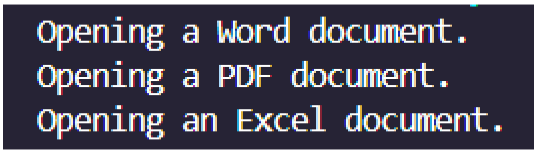
```
Logger Initialized
Log message: log message 1
Log message: log message 2
Singleton confirmed
```

Exercise 2: Implementing the Factory Method Pattern

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

class WordDocument implements Document {
  public void open() {
    System.out.println("Opening a Word document.");
  }
}

class PdfDocument implements Document {
  public void open() {
    System.out.println("Opening a PDF document.");
  }
}

class ExcelDocument implements Document {
  public void open() {
    System.out.println("Opening an Excel document.");
  }
}

abstract class DocumentFactory {
  public abstract Document createDocument();
}

class WordDocumentFactory extends DocumentFactory {
  public Document createDocument() {
    return new WordDocument();
  }
}
```

```
class PdfDocumentFactory extends DocumentFactory {
    public Document createDocument() {
        return new PdfDocument();
    }
}

class ExcelDocumentFactory extends DocumentFactory {
    public Document createDocument() {
        return new ExcelDocument();
    }
}

public class FileFactory{
    public static void main(String[] args) {
        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:



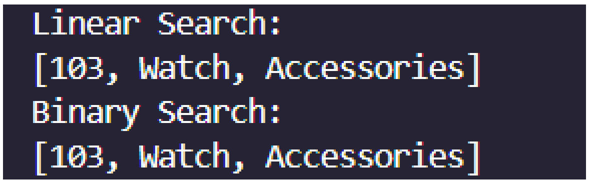
```
Opening a Word document.
Opening a PDF document.
Opening an Excel document.
```

Algorithms and Data Structures

Exercise 2: E-commerce Platform Search Function

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

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;
    }

    @Override
    public String toString() {
        return "[" + productId + ", " + productName + ", " + category + "]";
    }
}

public class ECommerceSearch {

    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) {
        int low = 0, high = products.length - 1;

        while (low <= high) {
            int mid = (low + high) / 2;
            int compare = products[mid].productName.compareToIgnoreCase(name);

            if (compare == 0) return products[mid];
            else if (compare < 0) low = mid + 1;
            else high = mid - 1;
        }
        return null;
    }

    public static void main(String[] args) {
        Product[] products = {
            new Product(101, "Laptop", "Electronics"),
            new Product(102, "Shoes", "Footwear"),
            new Product(103, "Watch", "Accessories"),
            new Product(104, "Book", "Stationery"),
            new Product(105, "Phone", "Electronics")
        };

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

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

        System.out.println("Binary Search:");
        Product result2 = binarySearch(products, "Watch");
        System.out.println(result2 != null ? result2 : "Product not found.");
    }
}
```

Output:

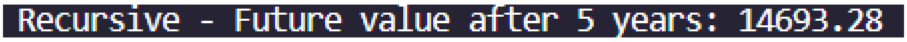


```
Linear Search:
[103, Watch, Accessories]
Binary Search:
[103, Watch, Accessories]
```

Exercise 7: Financial Forecasting

```
public class FinancialForecasting {

    public static double futureValue(double principal, double rate, int years) {
        if (years == 0) {
            return principal;
        } else {
            return futureValue(principal, rate, years - 1) * (1 + rate);
        }
    }

    public static void main(String[] args) {
        double initialAmount = 10000;
        double growthRate = 0.08;
        int years = 5;

        double recursiveResult = futureValue(initialAmount, growthRate, years);

        System.out.printf("Recursive - Future value after %d years: %.2f%n", years, recursiveResult);
    }
}
```

Output:



```
Recursive - Future value after 5 years: 14693.28
```