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 Logger {
   private static Logger instance;

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


   public static Logger getInstance()
      { if (instance == null) {
         instance = new Logger();
      }
      return instance;
   }

   public void log(String message) {
      System.out.println("Log: " + message);
   }
}
public class Main {
   public static void main(String[] args) {
      Logger logger1 = Logger.getInstance();
      logger1.log("First message");

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

      System.out.println("Are both loggers the same instance? " + (logger1 == logger2));
   }
}
```

Output:

Main.java

```
        }
        return instance;
    }

    public void log(String message) {
        System.out.println("Log: " + message);
    }
}
public class Main {
    public static void main(String[] args) {
        Logger logger1 = Logger.getInstance();
        logger1.log("First message");

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

        System.out.println("Are both loggers the same instance? " + (logger1 == logger2));
    }
}
```

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 Document {
   void open();
}
 class WordDocument implements Document {
```

```
    @Override
    public void open() {
        System.out.println("Opening Word document.");
    }
}
 class PdfDocument implements Document {
    @Override
    public void open() {
        System.out.println("Opening PDF document.");
    }
}
class ExcelDocument implements Document
    { @Override
    public void open() {
        System.out.println("Opening Excel document.");
    }
}
 abstract class DocumentFactory {
    public abstract Document createDocument();
}
 class WordDocumentFactory extends DocumentFactory {
    @Override
    public Document createDocument()
        { return new WordDocument();
    }
}
 class PdfDocumentFactory extends DocumentFactory {
    @Override
    public Document createDocument()
        { return new PdfDocument();
    }
}
 class ExcelDocumentFactory extends DocumentFactory {
    @Override
    public Document createDocument()
        { return new ExcelDocument();
    }
}
public class Main {
    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:

```
Main.java
}
class ExcelDocument implements Document {
    @Override
    public void open() {
        System.out.println("Opening Excel document.");
    }
}
abstract class DocumentFactory {
    public abstract Document createDocument();
}
class WordDocumentFactory extends DocumentFactory {
    @Override
    public Document createDocument() {
        return new WordDocument();
    }
}
class PdfDocumentFactory extends DocumentFactory {
    @Override
    public Document createDocument() {
        return new PdfDocument();
```

```
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.

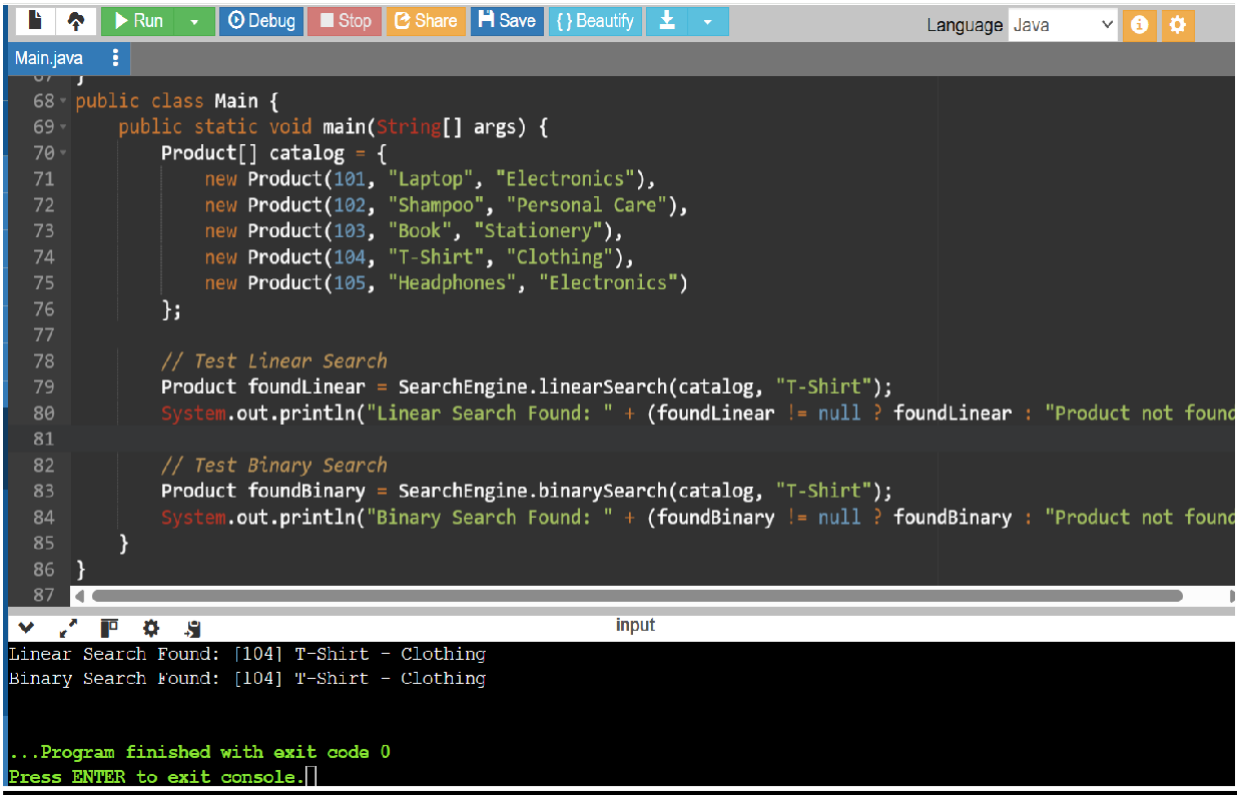
Code:

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

```
class SearchEngine {

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

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

    Arrays.sort(products, Comparator.comparing(Product::getProductName,
String.CASE_INSENSITIVE_ORDER));

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

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

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

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

```
        };

        Product foundLinear = SearchEngine.linearSearch(catalog, "T-Shirt");
        System.out.println("Linear Search Found: " + (foundLinear != null ? foundLinear
: "Product not found"));    Product foundBinary
=    SearchEngine.binarySearch(catalog,    "T-
Shirt");
        System.out.println("Binary Search Found: " + (foundBinary != null ?
foundBinary : "Product 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 predictFutureValue(double amount, double rate, int
      years) { if (years == 0) {
         return amount;
      }

      // Recursive step
      return predictFutureValue(amount, rate, years - 1) * (1 + rate);
   }

   public static double predictIteratively(double amount, double rate, int
      years) { for (int i = 0; i < years; i++) {
         amount *= (1 + rate);
      }
      return amount;
   }

   public static void main(String[]
      args) { double initialInvestment =
      10000.0 double
      annualGrowthRate = 0.07; int
      forecastYears = 5;


      double futureValueRecursive = predictFutureValue(initialInvestment,
annualGrowthRate, forecastYears);
      System.out.printf("Recursive: Predicted value after %d years: ₹%.2f\n",
forecastYears, futureValueRecursive);


      double futureValueIterative = predictIteratively(initialInvestment,
annualGrowthRate, forecastYears);
      System.out.printf("Iterative: Predicted value after %d years: ₹%.2f\n",
forecastYears, futureValueIterative);
   }
}
```

Output:

Run Debug Stop Share Save {} Beautify Language Java

Main.java

```java
public class Main {

    // Recursive method to calculate future value
    public static double predictFutureValue(double amount, double rate, int years) {
        if (years == 0) {
            return amount;
        }

        // Recursive step
        return predictFutureValue(amount, rate, years - 1) * (1 + rate);
    }

    // Iterative method to calculate future value
    public static double predictIteratively(double amount, double rate, int years) {
        for (int i = 0; i < years; i++) {
            amount *= (1 + rate);
        }
        return amount;
    }

```

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.
```