Module 1 - Design Patterns and Principles

Exercise 1: Implementing the Singleton Pattern(mandatory)

```
public class SingletonPatternExample {

    static 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 static void main(String[] args) {
        Logger logger1 = Logger.getInstance();
        logger1.log("Starting the application");

        Logger logger2 = Logger.getInstance();
```

```
        logger2.log("Continuing the application");

        if (logger1 == logger2) {
            System.out.println("Both logger instances are the same (Singleton Verified)");
        } else {
            System.out.println("Different instances");
        }
    }
}
```

Exercise 2: Implementing the Factory Method Pattern(mandatory)

```
public class FactoryMethodPatternDemo  {

    interface Document {
        void open();
    }

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

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

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

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

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

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

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

```
    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();
    }
}
```

Exercise 6: Implementing the Proxy Pattern(additional)

```
public class ProxyPatternExample {

    interface Image {
        void display();
    }

    static class RealImage implements Image {
        private String filename;

        public RealImage(String filename) {
            this.filename = filename;
            loadFromServer();
        }

        private void loadFromServer() {
            System.out.println("Loading image from remote server: " + filename);
```

```
    }

    public void display() {
      System.out.println("Displaying image: " + filename);
    }
  }

  static class ProxyImage implements Image {
    private RealImage realImage;
    private String filename;

    public ProxyImage(String filename) {
      this.filename = filename;
    }

    public void display() {
      if (realImage == null) {
        realImage = new RealImage(filename);

      }
else {
        System.out.println("Using cached image for: " + filename);
      }
      realImage.display();
    }
  }


  public static void main(String[] args) {
    Image image1 = new ProxyImage("sunset.jpg");
    Image image2 = new ProxyImage("mountain.jpg");

    image1.display();
    System.out.println();

    image1.display();
    System.out.println();

    image2.display();
  }
}
```

Module 2 - Data Structures and Algorithms

Exercise 2: E-commerce Platform Search Function(mandatory)

```
import java.util.*;
class Product {
    int productId;
    String productName;
    String category;

    Product(int productId, String productName, String category) {
        this.productId = productId;
        this.productName = productName;
        this.category = category;
    }
}

class LinearSearch {
    static int search(Product[] products, String key) {
        for (int i = 0; i < products.length; i++) {
            if (products[i].productName.equalsIgnoreCase(key)) {
                return i;
            }
        }
        return -1;
    }
}

class BinarySearch {
    static int search(Product[] products, String key) {
```

```
        Arrays.sort(products, Comparator.comparing(p -> p.productName));
        int left = 0, right = products.length - 1;
        while (left <= right) {
            int mid = (left + right) / 2;
            int comp = products[mid].productName.compareToIgnoreCase(key);
            if (comp == 0)
                return mid;
            else if (comp < 0)
                left = mid + 1;
            else
                right = mid - 1;
        }
        return -1;
    }
}

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

        int index1 = LinearSearch.search(products, "Phone");
        System.out.println("Linear Search found at index: " + index1);

        int index2 = BinarySearch.search(products, "Phone");
```

```
        System.out.println("Binary Search found at index: " + index2);
    }
}
```

Exercise 7: Financial Forecasting(mandatory)

```
class Forecast {
    static double predictRecursive(double initial, double rate, int years) {
        if (years == 0)
            return initial;
        return predictRecursive(initial, rate, years - 1) * (1 + rate);
    }

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

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

        double futureValueRecursive = Forecast.predictRecursive(initialAmount,
annualGrowthRate, years);
        double futureValueIterative = Forecast.predictIterative(initialAmount,
annualGrowthRate, years);
```

```
        System.out.println("Future value using recursion: " + futureValueRecursive);
        System.out.println("Future value using iteration: " + futureValueIterative);
    }
}
```

Exercise 6: Library Management System(additional)

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

class Book {
    int bookId;
    String title;
    String author;

    Book(int bookId, String title, String author) {
        this.bookId = bookId;
        this.title = title;
        this.author = author;
    }
}

class LinearSearch {
    static int search(Book[] books, String key) {
        for (int i = 0; i < books.length; i++) {
            if (books[i].title.equalsIgnoreCase(key)) {
                return i;
            }
```

```
        }
        return -1;
    }
}

class BinarySearch {
    static int search(Book[] books, String key) {
        Arrays.sort(books, Comparator.comparing(b -> b.title.toLowerCase()));
        int left = 0, right = books.length - 1;
        while (left <= right) {
            int mid = (left + right) / 2;
            int cmp = books[mid].title.compareToIgnoreCase(key);
            if (cmp == 0)
                return mid;
            else if (cmp < 0)
                left = mid + 1;
            else
                right = mid - 1;
        }
        return -1;
    }
}

public class LibraryManagementSystem {
    public static void main(String[] args) {
        Book[] books = {
            new Book(1, "The Alchemist", "Paulo Coelho"),
            new Book(2, "Clean Code", "Robert Martin"),
            new Book(3, "1984", "George Orwell"),
```

```
            new Book(4, "To Kill a Mockingbird", "Harper Lee")
        };

        int index1 = LinearSearch.search(books, "Clean Code");
        System.out.println("Linear Search found at index: " + index1);

        int index2 = BinarySearch.search(books, "Clean Code");
        System.out.println("Binary Search found at index: " + index2);
    }
}
```