DESIGN PATTERN AND PRINCIPLES

Exercise 1: Implementing the Singleton pattern

```
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 ");

      Logger logger2 = Logger.getInstance();
      logger2.log("Continuing ");

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

Exercise 2: Implementing the Factory Method Pattern:

```
public class FactoryMethodPatternDemo {
```

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

static class WordDocument implements Document {
   public void open() {
      System.out.println("Word Document is being opened” );
   }
}

static class PdfDocument implements Document {
   public void open() {
      System.out.println("PDF Document is being opened”);
   }
}

static class ExcelDocument implements Document {
   public void open() {
      System.out.println(" Excel Document is being opened”);
   }
}

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();
    }
}
```

DATA STRUCTURES AND ALGORITHMS:

Exercise 2: E-commerce Platform Search Function

```
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 i1 = LinearSearch.search(products, "Phone");
        System.out.println("Linear Search found at index: " + i1);

        int i2 = BinarySearch.search(products, "Phone");
        System.out.println("Binary Search found at index: " + i2);
    }
}
```

Exercise 7: Financial Forecasting

```
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);
    }
}
```

Additional:

Exercise 4: Implementing the Adapter Pattern

```
public class AdapterPatternExample {

    interface PaymentProcessor {
        void processPayment(double amount);
    }

    static class PayPalGateway {
        public void makePayment(double amount) {
            System.out.println("Processing payment through PayPal: ₹" + amount);
        }
    }

    static class StripeGateway {
        public void pay(double amountInINR) {
            System.out.println("Processing payment through Stripe: ₹" + amountInINR);
        }
    }


    static class PayPalAdapter implements PaymentProcessor {
        private PayPalGateway paypal;

        public PayPalAdapter() {
            this.paypal = new PayPalGateway();
        }

        public void processPayment(double amount) {
            paypal.makePayment(amount);
        }
    }

    static class StripeAdapter implements PaymentProcessor {
        private StripeGateway stripe;

        public StripeAdapter() {
            this.stripe = new StripeGateway();
        }

        public void processPayment(double amount) {
            stripe.pay(amount);
        }
    }
    public static void main(String[] args) {
        PaymentProcessor paypalProcessor = new PayPalAdapter();
        PaymentProcessor stripeProcessor = new StripeAdapter();

        System.out.println("=== Using PayPal ===");
        paypalProcessor.processPayment(2500.00);

        System.out.println("\n=== Using Stripe ===");
        stripeProcessor.processPayment(4500.00);
    }
}
```

Exercise 6: Library Management System

```
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 i1 = LinearSearch.search(books, "Clean Code");
        System.out.println("Linear Search found at index: " + i1);

        int i2 = BinarySearch.search(books, "Clean Code");
        System.out.println("Binary Search found at index: " + i2);
    }
}
```