OOP Problem Statement

1) Create the CaesarCipher class with the following parts:

- Private fields for the alphabet and shiftedAlphabet
- Write a constructor CaesarCipher that has one int parameter key. This method should initialize all the private fields of the class.
- Write an encrypt method that has one String parameter named input. This method returns a String that is the input encrypted using shiftedAlphabet.
- Write a decrypt method that has one String parameter named input. This method returns a String that is the encrypted String decrypted using the key associated with this CaesarCipher object. One way to do this is to create another private field mainKey, which is initialized to be the value of key. Then you can create a CaesarCipher object within decrypt: CaesarCipher cc = new CaesarCipher(26 - mainKey); and call cc.encrypt(input).
- Create the TestCaesarCipher class with the following parts:
- Create a CaesarCipher object with key 18, encrypt the String read in using the Scanner class object, print the encrypted String, and decrypt the encrypted String using the decrypt method.

```
import java.util.Scanner;

public class CaesarCipher {
    private String alphabet;
    private String shiftedAlphabet;
    private int mainKey;

    // Constructor
    public CaesarCipher(int key) {
        alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
        shiftedAlphabet = alphabet.substring(key) + alphabet.substring(0,
key);
        mainKey = key;
    }

    // Encrypt method
    public String encrypt(String input) {
        StringBuilder encrypted = new StringBuilder();

        for (char c : input.toUpperCase().toCharArray()) {
            int idx = alphabet.indexOf(c);
            if (idx != -1) {
                encrypted.append(shiftedAlphabet.charAt(idx));
```

```
            } else {
                encrypted.append(c); // Leave non-alphabetic characters
unchanged
            }
        }

        return encrypted.toString();
    }

    // Decrypt method
    public String decrypt(String input) {
        CaesarCipher cc = new CaesarCipher(26 - mainKey);
        return cc.encrypt(input);
    }

    // Main method for testing
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a string to encrypt: ");
        String input = scanner.nextLine();

        CaesarCipher cipher = new CaesarCipher(18);
        String encrypted = cipher.encrypt(input);
        System.out.println("Encrypted string: " + encrypted);

        String decrypted = cipher.decrypt(encrypted);
        System.out.println("Decrypted string: " + decrypted);
    }
}
```

2) Create the URLFinder class with the following parts:

· Private fields for the url

· Write a constructor URLFinder that has one int parameter url. This method should initialize all the private fields of the class.

- Write an urlChecker method that has one String parameter named inputUrl. This method returns a Boolean “true” for valid url.
- Create the TestURLFinder class with the following parts:
- create a URLFinder object with String read in using the Scanner class

```
import java.util.Scanner;
```

```java
import java.net.URL;
import java.net.MalformedURLException;

public class URLFinder {
    private String url;

    // Constructor
    public URLFinder(String url) {
        this.url = url;
    }

    // Method to check if URL is valid
    public boolean urlChecker(String inputUrl) {
        try {
            new URL(inputUrl); // Try to create a URL object
            return true;
        } catch (MalformedURLException e) {
            return false;
        }
    }

    // Main method for testing
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a URL to check: ");
        String userInput = scanner.nextLine();

        URLFinder finder = new URLFinder(userInput);
        boolean isValid = finder.urlChecker(userInput);

        if (isValid) {
            System.out.println("The URL is valid.");
        } else {
            System.out.println("The URL is invalid.");
        }
    }
}
```

3) Perform the following operations on given matrix

M =

1	2	3
4	5	6
7	8	9

1. Transpose
2. Determinant
3. Inverse

```java
public class MatrixOperations {

    // Function to print a matrix
    public static void printMatrix(double[][] matrix) {
        for (double[] row : matrix) {
            for (double val : row)
                System.out.print(val + " ");
            System.out.println();
        }
    }

    // 1. Transpose of a matrix
    public static double[][] transpose(double[][] matrix) {
        double[][] result = new double[3][3];
        for (int i = 0; i < 3; i++)
            for (int j = 0; j < 3; j++)
                result[j][i] = matrix[i][j];
        return result;
    }

    // 2. Determinant of a 3x3 matrix
    public static double determinant(double[][] m) {
        return m[0][0]*(m[1][1]*m[2][2] - m[1][2]*m[2][1]) -
               m[0][1]*(m[1][0]*m[2][2] - m[1][2]*m[2][0]) +
               m[0][2]*(m[1][0]*m[2][1] - m[1][1]*m[2][0]);
    }

    // 3. Inverse of a 3x3 matrix (if determinant ≠ 0)
    public static double[][] inverse(double[][] m) {
        double det = determinant(m);
        if (det == 0) {
            return null;
```

```
        }

        double[][] inv = new double[3][3];

        inv[0][0] = (m[1][1]*m[2][2] - m[1][2]*m[2][1]) / det;
        inv[0][1] = -(m[0][1]*m[2][2] - m[0][2]*m[2][1]) / det;
        inv[0][2] = (m[0][1]*m[1][2] - m[0][2]*m[1][1]) / det;

        inv[1][0] = -(m[1][0]*m[2][2] - m[1][2]*m[2][0]) / det;
        inv[1][1] = (m[0][0]*m[2][2] - m[0][2]*m[2][0]) / det;
        inv[1][2] = -(m[0][0]*m[1][2] - m[0][2]*m[1][0]) / det;

        inv[2][0] = (m[1][0]*m[2][1] - m[1][1]*m[2][0]) / det;
        inv[2][1] = -(m[0][0]*m[2][1] - m[0][1]*m[2][0]) / det;
        inv[2][2] = (m[0][0]*m[1][1] - m[0][1]*m[1][0]) / det;

        return inv;
    }

    public static void main(String[] args) {
        double[][] M = {
            {1, 2, 3},
            {4, 5, 4},
            {7, 8, 3}
        };

        System.out.println("Original Matrix:");
        printMatrix(M);

        // Transpose
        System.out.println("\n1. Transpose:");
        printMatrix(transpose(M));

        // Determinant
        double det = determinant(M);
        System.out.println("\n2. Determinant: " + det);

        // Inverse
        System.out.println("\n3. Inverse:");
        double[][] inv = inverse(M);
        if (inv == null) {
            System.out.println("Inverse does not exist (determinant is
   0).");
        } else {
            printMatrix(inv);
```

```
80.         }
81.     }
82.}
83.
```

4) A) Write a Program to Print the Pascal's and Number Triangle in Java

```
Input : N = 5
Output:
          1
        1   1
      1   2   1
    1   3   3   1
  1   4   6   4   1
1   5   10   10   5   1
```

4) B) Write a Java program to print the number triangle

```
     1
    2 2
   3 3 3
  4 4 4 4
 5 5 5 5 5
6 6 6 6 6 6
```

```java
import java.util.Scanner;

public class TrianglePatterns {
    // Function to print Pascal's Triangle
    static void printPascalsTriangle(int rows) {
        for (int i = 0; i < rows; i++) {
            // Print spaces
            for (int space = 0; space < rows - i; space++) {
                System.out.print(" ");
            }
```

```
            int number = 1;
            for (int j = 0; j <= i; j++) {
                System.out.print(number + " ");
                number = number * (i - j) / (j + 1); // Formula for nCr
            }
            System.out.println();
        }
    }

    // Function to print Number Triangle
    static void printNumberTriangle(int rows) {
        int number = 1;
        for (int i = 1; i <= rows; i++) {
            // Print spaces
            for (int space = 1; space <= rows - i; space++) {
                System.out.print(" ");
            }
            for (int j = 1; j <= i; j++) {
                System.out.print(number + " ");
                number++;
            }
            System.out.println();
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter number of rows: ");
        int rows = sc.nextInt();

        System.out.println("\nPascal's Triangle:");
        printPascalsTriangle(rows);

        System.out.println("\nNumber Triangle:");
        printNumberTriangle(rows);
    }
}
```

5)

A. An ArrayList consists of 1-25 numbers. Write a Java program to remove prime numbers from an ArrayList using an iterator.

```
B. import java.util.ArrayList;
C. import java.util.Iterator;
D.
E. public class RemovePrimes {
F.
G.     // Function to check if a number is prime
H.     public static boolean isPrime(int num) {
I.         if (num <= 1){
J.             return false;
K.         }
L.
M.         for (int i = 2; i <= num / 2; i++) {
N.             if (num % i == 0){
O.
P.              return false;
Q.             }
R.
S.         }
T.         return true;
U.     }
V.
W.     public static void main(String[] args) {
X.         ArrayList<Integer> numbers = new ArrayList<>();
Y.
Z.         // Add numbers from 1 to 25
AA.        for (int i = 1; i <= 25; i++) {
BB.            numbers.add(i);
CC.        }
DD.
EE.        System.out.println("Original List: " + numbers);
FF.
GG.        // Use iterator to remove prime numbers
HH.        Iterator<Integer> it = numbers.iterator();
II.        while (it.hasNext()) {
JJ.            int num = it.next();
KK.            if (isPrime(num)) {
LL.                it.remove();
MM.            }
NN.        }
OO.
PP.        System.out.println("After removing prime numbers: " + numbers);
QQ.    }
RR.}
SS.
```

B.Write a Java program to

a. Create and traverse (or iterate) an ArrayList using a for-loop, iterator, and advance for-loop.

b. Check if the element(value) exists in the ArrayList?

C. Add element at the particular index of the ArrayList?

```
import java.util.ArrayList;
import java.util.Iterator;

public class ArrayListOperations {
    public static void main(String[] args) {
        // Create ArrayList
        ArrayList<String> fruits = new ArrayList<>();

        // Add elements to the ArrayList
        fruits.add("Apple");
        fruits.add("Banana");
        fruits.add("Mango");
        fruits.add("Orange");

        System.out.println("Original List: " + fruits);

        // a. Traverse using for loop
        System.out.println("\nTraverse using for loop:");
        for (int i = 0; i < fruits.size(); i++) {
            System.out.println(fruits.get(i));
        }

        // a. Traverse using iterator
        System.out.println("\nTraverse using Iterator:");
        Iterator<String> iterator = fruits.iterator();
        while (iterator.hasNext()) {
            System.out.println(iterator.next());
        }

        // a. Traverse using enhanced for-loop
        System.out.println("\nTraverse using enhanced for-loop:");
        for (String fruit : fruits) {
            System.out.println(fruit);
```

```
        }

        // b. Check if an element exists
        String search = "Mango";
        if (fruits.contains(search)) {
            System.out.println("\n" + search + " is in the list.");
        } else {
            System.out.println("\n" + search + " is NOT in the list.");
        }

        // c. Add element at specific index
        fruits.add(2, "Pineapple"); // Adding at index 2
        System.out.println("\nList after adding 'Pineapple' at index 2:");
        System.out.println(fruits);
    }
}
```

6)

- Write a Java program to find the largest and smallest elements in an array of integers.
- Implement a function to reverse an array in place.
- Given two arrays, write a method to merge them into a single sorted array.

HARDCODED

```
public class ArrayOperations {

    // Function to find largest and smallest elements in the array
    public static void findLargestAndSmallest(int[] arr) {
        int largest = arr[0];
        int smallest = arr[0];

        for (int i = 1; i < arr.length; i++) {
            if (arr[i] > largest)
                largest = arr[i];
            if (arr[i] < smallest)
                smallest = arr[i];
        }
```

```
        System.out.println("Largest element: " + largest);
        System.out.println("Smallest element: " + smallest);
    }

    // Function to reverse array in place
    public static void reverseArray(int[] arr) {
        int start = 0, end = arr.length - 1;
        while (start < end) {
            // Swap elements
            int temp = arr[start];
            arr[start] = arr[end];
            arr[end] = temp;

            start++;
            end--;
        }
    }

    // Function to merge two arrays and sort the result
    public static int[] mergeAndSortArrays(int[] arr1, int[] arr2) {
        int[] merged = new int[arr1.length + arr2.length];
        int index = 0;

        // Copy elements of arr1
        for (int num : arr1) {
            merged[index++] = num;
        }

        // Copy elements of arr2
        for (int num : arr2) {
            merged[index++] = num;
        }

        // Sort the merged array using simple bubble sort for clarity
        for (int i = 0; i < merged.length - 1; i++) {
            for (int j = 0; j < merged.length - i - 1; j++) {
                if (merged[j] > merged[j + 1]) {
                    int temp = merged[j];
                    merged[j] = merged[j + 1];
                    merged[j + 1] = temp;
                }
            }
        }

        return merged;
```

```
    }

    // Main method
    public static void main(String[] args) {
        // Hardcoded array
        int[] array1 = {12, 5, 8, 19, 1};
        int[] array2 = {3, 7, 6, 14};

        // 1. Find largest and smallest
        System.out.println("Original array1:");
        for (int num : array1) System.out.print(num + " ");
        System.out.println();
        findLargestAndSmallest(array1);

        // 2. Reverse array1
        reverseArray(array1);
        System.out.println("\nReversed array1:");
        for (int num : array1) System.out.print(num + " ");

        // 3. Merge and sort array1 and array2
        int[] merged = mergeAndSortArrays(array1, array2);
        System.out.println("\n\nMerged and Sorted Array:");
        for (int num : merged) System.out.print(num + " ");
    }
}
```

USER INPUT

```
import java.util.Arrays;
import java.util.Scanner;

public class ArrayOperations {

    // Function to find the largest and smallest elements in an array
    public static void findLargestAndSmallest(int[] arr) {
        int largest = arr[0];
        int smallest = arr[0];

        for (int i = 1; i < arr.length; i++) {
            if (arr[i] > largest) {
                largest = arr[i];
            }
```

```
        if (arr[i] < smallest) {
            smallest = arr[i];
        }
    }

    System.out.println("Largest element: " + largest);
    System.out.println("Smallest element: " + smallest);
}

// Function to reverse an array in place
public static void reverseArray(int[] arr) {
    int left = 0;
    int right = arr.length - 1;

    while (left < right) {
        // Swap elements
        int temp = arr[left];
        arr[left] = arr[right];
        arr[right] = temp;

        // Move the pointers
        left++;
        right--;
    }
}

// Function to merge two arrays into a single sorted array
public static int[] mergeArrays(int[] arr1, int[] arr2) {
    int n = arr1.length + arr2.length;
    int[] mergedArray = new int[n];

    // Copy elements of arr1 and arr2 into the merged array
    System.arraycopy(arr1, 0, mergedArray, 0, arr1.length);
    System.arraycopy(arr2, 0, mergedArray, arr1.length, arr2.length);

    // Sort the merged array
    Arrays.sort(mergedArray);

    return mergedArray;
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
```

```
        // Task 1: Get user input for array 1
        System.out.println("Enter the number of elements for Array 1: ");
        int n1 = scanner.nextInt();
        int[] arr1 = new int[n1];
        System.out.println("Enter the elements for Array 1: ");
        for (int i = 0; i < n1; i++) {
            arr1[i] = scanner.nextInt();
        }

        // Task 2: Get user input for array 2
        System.out.println("Enter the number of elements for Array 2: ");
        int n2 = scanner.nextInt();
        int[] arr2 = new int[n2];
        System.out.println("Enter the elements for Array 2: ");
        for (int i = 0; i < n2; i++) {
            arr2[i] = scanner.nextInt();
        }

        // Task 3: Find the largest and smallest elements
        findLargestAndSmallest(arr1);

        // Task 4: Reverse the array
        reverseArray(arr1);
        System.out.println("Reversed Array 1: " + Arrays.toString(arr1));

        // Task 5: Merge two arrays and display the sorted array
        int[] mergedArray = mergeArrays(arr1, arr2);
        System.out.println("Merged and sorted array: " + Arrays.toString(mergedArray));

        scanner.close();  // Close the scanner to avoid resource leak
    }
}
```

7)

- Write a program to check if a given string is a palindrome.
- Implement a function to count the occurrences of a specific character in a string.
- Write a program to remove all whitespace from a string.

HARDCODED

```
public class StringOperations {

    // 1. Check if a string is a palindrome
    public static boolean isPalindrome(String str) {
        String cleaned = str.replaceAll("\\s+", "").toLowerCase(); // ignore spaces
and case
        int start = 0;
        int end = cleaned.length() - 1;

        while (start < end) {
            if (cleaned.charAt(start) != cleaned.charAt(end)) {
                return false;
            }
            start++;
            end--;
        }
        return true;
    }

    // 2. Count occurrences of a specific character in a string
    public static int countCharacter(String str, char target) {
        int count = 0;
        for (char c : str.toCharArray()) {
            if (c == target) {
                count++;
            }
        }
        return count;
    }

    // 3. Remove all whitespace from a string
    public static String removeWhitespace(String str) {
        return str.replaceAll("\\s+", "");
```

```
    }

    public static void main(String[] args) {
        String testStr = "A man a plan a canal Panama";
        char targetChar = 'a';

        System.out.println("Original String: \"" + testStr + "\"");

        // Palindrome check
        boolean isPalin = isPalindrome(testStr);
        System.out.println("Is palindrome? " + isPalin);

        // Character count
        int count = countCharacter(testStr.toLowerCase(),
Character.toLowerCase(targetChar));
        System.out.println("Occurrences of '" + targetChar + "': " + count);

        // Remove whitespace
        String noSpaces = removeWhitespace(testStr);
        System.out.println("String without whitespace: \"" + noSpaces + "\"");
    }
}
```

USER INPUT

```
import java.util.Scanner;

public class StringOperations {

    // Function to check if a string is a palindrome
    public static boolean isPalindrome(String str) {
        int left = 0;
        int right = str.length() - 1;

        // Compare characters from both ends
        while (left < right) {
            if (str.charAt(left) != str.charAt(right)) {
```

```
                return false; // Not a palindrome if characters don't
match
            }
            left++;
            right--;
        }
        return true; // It is a palindrome if all characters match
    }

    // Function to count occurrences of a specific character in a string
    public static int countOccurrences(String str, char ch) {
        int count = 0;

        // Loop through the string and count occurrences of the character
        for (int i = 0; i < str.length(); i++) {
            if (str.charAt(i) == ch) {
                count++;
            }
        }
        return count;
    }

    // Function to remove all whitespace from a string
    public static String removeWhitespace(String str) {
        return str.replaceAll("\\s+", ""); // Remove all spaces using
regular expression
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Task 1: Check if a given string is a palindrome
        System.out.print("Enter a string to check if it is a palindrome:
");
        String str1 = scanner.nextLine();
        if (isPalindrome(str1)) {
            System.out.println("The string is a palindrome.");
        } else {
            System.out.println("The string is not a palindrome.");
        }

        // Task 2: Count occurrences of a specific character
        System.out.print("Enter a string to count character occurrences:
");
        String str2 = scanner.nextLine();
```

```
            System.out.print("Enter the character to count: ");
            char ch = scanner.next().charAt(0);
            int count = countOccurrences(str2, ch);
            System.out.println("The character '" + ch + "' appears " + count +
    " times.");

            // Task 3: Remove all whitespace from a string
            System.out.print("Enter a string to remove all whitespaces: ");
            scanner.nextLine(); // Consume newline left by nextInt()
            String str3 = scanner.nextLine();
            String result = removeWhitespace(str3);
            System.out.println("String without whitespace: " + result);

            scanner.close(); // Close the scanner to avoid resource leak
        }
    }
```

8) Design a class BankAccount with methods for deposit, withdraw, and check balance. Implement exception handling for insufficient funds during withdrawal.

HARDCODED

```
// Custom exception for insufficient funds
class InsufficientFundsException extends Exception {
    public InsufficientFundsException(String message) {
        super(message);
    }
}

// BankAccount class
class BankAccount {
    private double balance;

    // Constructor
    public BankAccount(double initialBalance) {
        this.balance = initialBalance;
    }
```

```
    // Method to deposit money
    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
            System.out.println("Deposited Rs." + amount);
        } else {
            System.out.println("Deposit amount must be positive.");
        }
    }

    // Method to withdraw money with exception handling
    public void withdraw(double amount) throws InsufficientFundsException {
        if (amount <= 0) {
            System.out.println("Withdrawal amount must be positive.");
        } else if (amount > balance) {
            throw new InsufficientFundsException("Insufficient balance! Available: Rs." +
balance);
        } else {
            balance -= amount;
            System.out.println("Withdrew Rs." + amount);
        }
    }

    // Method to check balance
    public void checkBalance() {
        System.out.println("Current balance: Rs." + balance);
    }

    // Main method to test
    public static void main(String[] args) {
        BankAccount myAccount = new BankAccount(1000); // initial balance Rs.1000

        myAccount.checkBalance();
        myAccount.deposit(500);
        myAccount.checkBalance();
```

```
        try {
            myAccount.withdraw(300);
            myAccount.checkBalance();

            myAccount.withdraw(1500); // Should throw exception
        } catch (InsufficientFundsException e) {
            System.out.println("Error: " + e.getMessage());
        }

        myAccount.checkBalance();
    }
}
```

USER INPUT

```
import java.util.Scanner;

// Custom Exception for Insufficient Funds
class InsufficientFundsException extends Exception {
    public InsufficientFundsException(String message) {
        super(message);
    }
}

// BankAccount class
public class BankAccount {
    private double balance;

    // Constructor
    public BankAccount(double initialBalance) {
        this.balance = initialBalance;
    }

    // Method to deposit money
    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
            System.out.println("₹" + amount + " deposited successfully.");
        } else {
```

```
            System.out.println("Deposit amount must be positive.");
        }
    }

    // Method to withdraw money
    public void withdraw(double amount) throws InsufficientFundsException {
        if (amount <= 0) {
            System.out.println("Withdrawal amount must be positive.");
        } else if (amount > balance) {
            throw new InsufficientFundsException("Insufficient funds! Available
balance: ₹" + balance);
        } else {
            balance -= amount;
            System.out.println("₹" + amount + " withdrawn successfully.");
        }
    }

    // Method to check balance
    public void checkBalance() {
        System.out.println("Current balance: ₹" + balance);
    }

    // Main method with user input
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter initial balance: ");
        double initialBalance = scanner.nextDouble();

        BankAccount account = new BankAccount(initialBalance);
        int choice;

        do {
            System.out.println("\n--- Bank Menu ---");
            System.out.println("1. Deposit");
            System.out.println("2. Withdraw");
            System.out.println("3. Check Balance");
            System.out.println("4. Exit");
            System.out.print("Enter your choice: ");
            choice = scanner.nextInt();

            switch (choice) {
                case 1:
                    System.out.print("Enter amount to deposit: ₹");
                    double depositAmount = scanner.nextDouble();
                    account.deposit(depositAmount);
```

```
                    break;

                case 2:
                    System.out.print("Enter amount to withdraw: ₹");
                    double withdrawAmount = scanner.nextDouble();
                    try {
                        account.withdraw(withdrawAmount);
                    } catch (InsufficientFundsException e) {
                        System.out.println("Error: " + e.getMessage());
                    }
                    break;

                case 3:
                    account.checkBalance();
                    break;

                case 4:
                    System.out.println("Thank you for using our banking
system.");
                    break;

                default:
                    System.out.println("Invalid choice. Please try again.");
            }

        } while (choice != 4);

        scanner.close();
    }
}
```

9) Write a program to read data from a text file using IO Stream class and display the number of characters and words on the console.

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class ReadFileCount {
    public static void main(String[] args) {
        String fileName = "input.txt";
```

```
        FileInputStream fis = null;
        int charCount = 0;
        int wordCount = 0;

        try {
            // Check if file exists, if not create it with default content
            File file = new File(fileName);
            if (!file.exists()) {
                System.out.println("File not found. Creating " + fileName + "
with sample content...");
                FileOutputStream fos = new FileOutputStream(file);
                String defaultContent = "This is an input file.\nIt is made by
Radha.";
                fos.write(defaultContent.getBytes());
                fos.close();
            }

            fis = new FileInputStream(file);
            StringBuilder content = new StringBuilder();

            int byteRead;
            while ((byteRead = fis.read()) != -1) {
                char c = (char) byteRead;
                content.append(c);
                charCount++;
            }

            // Trim and split the content to count words
            String text = content.toString().trim();
            if (!text.isEmpty()) {
                String[] words = text.split("\\s+"); // Split on whitespace
                wordCount = words.length;
            }

            // Display results
            System.out.println("Number of characters: " + charCount);
            System.out.println("Number of words: " + wordCount);

        } catch (IOException e) {
            System.out.println("An error occurred while processing the file: " +
e.getMessage());
        } finally {
            try {
                if (fis != null)
                    fis.close();
```

```
            } catch (IOException e) {
                System.out.println("Failed to close the file input stream.");
            }
        }
    }
}
```

10)

A. Write a program to find the greatest common divisor (GCD) of two numbers.
B. Write a program to convert a decimal number to binary.

HARDOCDED:

```
public class MathOperations {

    // A. Method to find GCD using Euclidean algorithm
    public static int findGCD(int a, int b) {
        while (b != 0) {
            int temp = b;
            b = a % b;
            a = temp;
        }
        return a;
    }

    // B. Method to convert decimal to binary
    public static String decimalToBinary(int number) {
        if (number == 0) return "0";
        StringBuilder binary = new StringBuilder();

        while (number > 0) {
            binary.insert(0, number % 2);
            number /= 2;
        }

        return binary.toString();
```

```
    }

    public static void main(String[] args) {
        // Example inputs
        int num1 = 48;
        int num2 = 18;
        int decimalNumber = 25;

        // GCD
        int gcd = findGCD(num1, num2);
        System.out.println("GCD of " + num1 + " and " + num2 + " is: " + gcd);

        // Decimal to Binary
        String binary = decimalToBinary(decimalNumber);
        System.out.println("Binary of " + decimalNumber + " is: " + binary);
    }
}
```

USER INPUT

```
import java.util.Scanner;

public class GCDAndDecimalToBinary {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Input for GCD calculation
        System.out.print("Enter first number for GCD: ");
        int a = scanner.nextInt();

        System.out.print("Enter second number for GCD: ");
        int b = scanner.nextInt();

        // Calculate GCD
        int gcd = findGCD(a, b);
        System.out.println("GCD of " + a + " and " + b + " is: " + gcd);

        // Input for Decimal to Binary conversion
        System.out.print("Enter a decimal number to convert to binary: ");
        int decimal = scanner.nextInt();
```

```
        // Convert decimal to binary manually
        String binary = decimalToBinary(decimal);
        System.out.println("Binary representation: " + binary);

        scanner.close();
    }

    // Euclidean algorithm to find GCD
    static int findGCD(int a, int b) {
        while (b != 0) {
            int temp = b;
            b = a % b;
            a = temp;
        }
        return a;
    }

    // Manual method to convert decimal to binary
    static String decimalToBinary(int num) {
        StringBuilder binary = new StringBuilder();

        // Edge case when number is 0
        if (num == 0) {
            binary.append("0");
        } else {
            while (num > 0) {
                binary.insert(0, num % 2); // Insert the remainder (0 or 1) at
the start
                num = num / 2; // Update the number by dividing it by 2
            }
        }

        return binary.toString();
    }
}
```

11) Create the CarAssembly class which implements Runnable interface with the following parts:

- Private fields for the componentName(String) and timeToPrepare(int)
- Write a constructor CarAssembly that has two parameters componentName and timeToPrepare . This method should initialize all the private fields of the class.

- Write an run method that has sleep method which takes timeToPrepare parameter. The sleep method is invoveked between two print statements componentName is preparing & componentName is ready.
- Components names and their preparation times are as follows
- Engine-3000, Body-4000, Wheels-5000

Create three threads namely engineThread, bodyThread, wheelThread and use Join method for Sysnchronization.

```
public class CarAssembly implements Runnable {
    // Private fields
    private String componentName;
    private int timeToPrepare;

    // Constructor
    public CarAssembly(String componentName, int timeToPrepare) {
        this.componentName = componentName;
        this.timeToPrepare = timeToPrepare;
    }

    // Run method
    @Override
    public void run() {
        try {
            // Simulate preparation time with sleep
            System.out.println(componentName + " is preparing");
            Thread.sleep(timeToPrepare);
            System.out.println(componentName + " is ready");
        } catch (InterruptedException e) {
            System.out.println("Thread interrupted: " + e.getMessage());
        }
    }

    // Main method
    public static void main(String[] args) {
        // Create CarAssembly objects for different components
        CarAssembly engine = new CarAssembly("Engine", 3000);
        CarAssembly body = new CarAssembly("Body", 4000);
        CarAssembly wheels = new CarAssembly("Wheels", 5000);

        // Create threads for each component
        Thread engineThread = new Thread(engine);
        Thread bodyThread = new Thread(body);
        Thread wheelThread = new Thread(wheels);
```

```
        // Start the threads
        engineThread.start();
        bodyThread.start();
        wheelThread.start();

        try {
            // Use join to synchronize the threads (ensure each part is ready
before the next starts)
            engineThread.join();
            bodyThread.join();
            wheelThread.join();
        } catch (InterruptedException e) {
            System.out.println("Main thread interrupted: " + e.getMessage());
        }

        System.out.println("Car assembly completed!");
    }
}
```

12) Design a Java program to manage an ArrayList of integers that supports dynamic insertion at any position, deletion, updating values, and efficiently computing the sum of elements between two given indices after each modification.

Sample Input	Sample Output
insert 0 5	[5]
insert 1 10	[5, 10]
insert 1 15	[5, 15, 10]
update 2 20	[5, 15, 20]
sum 0 2	25
delete 1	[5, 20]
sum 0 1	25

HARDCODED

```
import java.util.*;

public class DynamicArrayListManager {
   public static void main(String[] args) {
      List<Integer> list = new ArrayList<>();

      // insert 0 5
      list.add(0, 5);
      System.out.println(list);  // [5]

      // insert 1 10
      list.add(1, 10);
      System.out.println(list);  // [5, 10]

      // insert 1 15
      list.add(1, 15);
      System.out.println(list);  // [5, 15, 10]

      // update 2 20
      list.set(2, 20);
      System.out.println(list);  // [5, 15, 20]

      // sum 0 2
      int sum1 = list.get(0) + list.get(1) + list.get(2);
      System.out.println(sum1);  // 25

      // delete 1
      list.remove(1);
      System.out.println(list);  // [5, 20]

      // sum 0 1
      int sum2 = list.get(0) + list.get(1);
      System.out.println(sum2);  // 25
   }
}
```

USER INPUT

```
import java.util.*;

public class ArrayListManager {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
```

```
        ArrayList<Integer> list = new ArrayList<>();

        while (sc.hasNextLine()) {
            String line = sc.nextLine().trim();
            if (line.isEmpty()) break;

            String[] parts = line.split(" ");
            String cmd = parts[0];

            try {
                switch (cmd) {
                    case "insert":
                        list.add(Integer.parseInt(parts[1]),
Integer.parseInt(parts[2]));
                        System.out.println(list);
                        break;

                    case "delete":
                        list.remove((int) Integer.parseInt(parts[1]));
                        System.out.println(list);
                        break;

                    case "update":
                        list.set(Integer.parseInt(parts[1]),
Integer.parseInt(parts[2]));
                        System.out.println(list);
                        break;

                    case "sum":
                        int from = Integer.parseInt(parts[1]);
                        int to = Integer.parseInt(parts[2]);
                        int sum = 0;
                        for (int i = from; i <= to; i++) sum += list.get(i);
                        System.out.println(sum);
                        break;

                    default:
                        System.out.println("Invalid command");
                }
            } catch (Exception e) {
                System.out.println("Error: " + e.getMessage());
            }
        }

        sc.close();
```

```
    }
}
```

13) Write a Java program using HashMap to add, remove, and track the frequency of words. You also need to find the most frequent word, and if there's a tie, return the smallest word alphabetically. The program should handle up to 10^5 operations efficiently.

Sample Input	Sample Output
add apple add banana add apple remove apple query apple mostFrequent	Frequency of 'apple': 1 Most frequent word: banana

```
import java.util.*;

public class Hashmap {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        HashMap<String, Integer> freqMap = new HashMap<>();

        while (sc.hasNextLine()) {
            String line = sc.nextLine().trim();
            if (line.isEmpty()) break;

            String[] parts = line.split(" ");
            String command = parts[0];

            switch (command) {
                case "add":
                    String wordToAdd = parts[1];
                    freqMap.put(wordToAdd, freqMap.getOrDefault(wordToAdd, 0) +
1);
                    break;

                case "remove":
```

```
                    String wordToRemove = parts[1];
                    freqMap.put(wordToRemove,
Math.max(freqMap.getOrDefault(wordToRemove, 0) - 1, 0));
                    break;

                case "query":
                    String wordToQuery = parts[1];
                    System.out.println("Frequency of '" + wordToQuery + "': " +
freqMap.getOrDefault(wordToQuery, 0));
                    break;

                case "mostFrequent":
                    int maxFreq = 0;
                    String mostFreqWord = "";
                    for (Map.Entry<String, Integer> entry : freqMap.entrySet()) {
                        String word = entry.getKey();
                        int freq = entry.getValue();
                        if (freq > maxFreq || (freq == maxFreq &&
word.compareTo(mostFreqWord) < 0)) {
                            maxFreq = freq;
                            mostFreqWord = word;
                        }
                    }
                    if (maxFreq > 0)
                        System.out.println("Most frequent word: " +
mostFreqWord);
                    else
                        System.out.println("No words found");
                    break;

                default:
                    System.out.println("Invalid command");
            }
        }

        sc.close();
    }
}
```

14) Design and implement a multi-threaded banking system in Java that simulates multiple users performing concurrent transactions on shared bank accounts. Each transaction can be a deposit, withdrawal, or transfer between accounts.

```
class BankAccount {
```

```
    private int balance;
    private final int accountId;

    public BankAccount(int accountId, int initialBalance) {
        this.accountId = accountId;
        this.balance = initialBalance;
    }

    public synchronized void deposit(int amount) {
        balance += amount;
        System.out.println("Account " + accountId + ": Deposited " + amount + ",
Balance = " + balance);
    }

    public synchronized void withdraw(int amount) {
        if (balance >= amount) {
            balance -= amount;
            System.out.println("Account " + accountId + ": Withdrawn " + amount +
", Balance = " + balance);
        } else {
            System.out.println("Account " + accountId + ": Insufficient balance
for withdrawal of " + amount);
        }
    }

    public synchronized int getBalance() {
        return balance;
    }

    public int getAccountId() {
        return accountId;
    }

    public static void transfer(BankAccount from, BankAccount to, int amount) {
        synchronized (from) {
            synchronized (to) {
                if (from.getBalance() >= amount) {
                    from.withdraw(amount);
                    to.deposit(amount);
                    System.out.println("Transferred " + amount + " from Account "
+ from.getAccountId() + " to Account " + to.getAccountId());
                } else {
                    System.out.println("Transfer failed: Insufficient funds in
Account " + from.getAccountId());
                }
```

```
            }
        }
    }
}

class Transaction implements Runnable {
    private final BankAccount from;
    private final BankAccount to;
    private final int amount;
    private final String type;

    public Transaction(BankAccount from, BankAccount to, int amount, String type)
{
        this.from = from;
        this.to = to;
        this.amount = amount;
        this.type = type;
    }

    @Override
    public void run() {
        switch (type) {
            case "deposit":
                from.deposit(amount);
                break;
            case "withdraw":
                from.withdraw(amount);
                break;
            case "transfer":
                BankAccount.transfer(from, to, amount);
                break;
        }
    }
}

public class BankSystem {
    public static void main(String[] args) {
        BankAccount acc1 = new BankAccount(101, 1000);
        BankAccount acc2 = new BankAccount(102, 500);

        Thread t1 = new Thread(new Transaction(acc1, null, 200, "deposit"));
        Thread t2 = new Thread(new Transaction(acc1, null, 300, "withdraw"));
        Thread t3 = new Thread(new Transaction(acc1, acc2, 150, "transfer"));
        Thread t4 = new Thread(new Transaction(acc2, acc1, 100, "transfer"));
```

```
        t1.start();
        t2.start();
        t3.start();
        t4.start();
    }
}
```

15)Design a shopping cart program where users can add items, apply discount codes, and check out. Use custom exceptions to handle scenarios like invalid coupon codes, out-of-stock items, and negative quantity inputs. (Exception handling)

```
// Custom Exceptions
class InvalidCouponException extends Exception {
    public InvalidCouponException(String message) {
        super(message);
    }
}

class OutOfStockException extends Exception {
    public OutOfStockException(String message) {
        super(message);
    }
}

class NegativeQuantityException extends Exception {
    public NegativeQuantityException(String message) {
        super(message);
    }
}

// Item class
class Item {
    String name;
    int price;
    int stock;

    public Item(String name, int price, int stock) {
        this.name = name;
        this.price = price;
        this.stock = stock;
    }
}

// Shopping Cart
```

```
class ShoppingCart {
    private int total = 0;
    private int discount = 0;

    public void addItem(Item item, int quantity) throws
NegativeQuantityException, OutOfStockException {
        if (quantity < 0) throw new NegativeQuantityException("Quantity cannot be
negative.");
        if (quantity > item.stock) throw new OutOfStockException("Not enough
stock for " + item.name);

        total += item.price * quantity;
        item.stock -= quantity;
        System.out.println(quantity + " x " + item.name + " added to cart.");
    }

    public void applyCoupon(String code) throws InvalidCouponException {
        if (code.equals("SAVE10")) {
            discount = 10;
            System.out.println("Coupon applied: 10% off");
        } else {
            throw new InvalidCouponException("Invalid coupon code.");
        }
    }

    public void checkout() {
        int discountedTotal = total - (total * discount / 100);
        System.out.println("Final total after discount: ₹" + discountedTotal);
    }
}

// Main class
public class ShoppingApp {
    public static void main(String[] args) {
        Item laptop = new Item("Laptop", 50000, 5);
        Item phone = new Item("Phone", 20000, 2);

        ShoppingCart cart = new ShoppingCart();

        try {
            cart.addItem(laptop, 1);
            cart.addItem(phone, 2);
            cart.applyCoupon("SAVE10");
            cart.checkout();
```

```
        } catch (InvalidCouponException | OutOfStockException |
NegativeQuantityException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

16) Design an interface Vehicle with methods startRide(), endRide(), and calculateFare(int distance). Implement classes like Bike, Auto, and Cab, where each vehicle calculates fares differently. Use a PricingStrategy interface to dynamically adjust fares based on conditions like peak hours and holidays.

```
// Vehicle interface
interface Vehicle {
    void startRide();
    void endRide();
    int calculateFare(int distance);
}

// PricingStrategy interface
interface PricingStrategy {
    int adjustFare(int baseFare);
}

// PeakHourStrategy
class PeakHourStrategy implements PricingStrategy {
    public int adjustFare(int baseFare) {
        return baseFare + (baseFare * 20 / 100); // 20% extra
    }
}

// HolidayStrategy
class HolidayStrategy implements PricingStrategy {
    public int adjustFare(int baseFare) {
        return baseFare + 50; // flat ₹50 extra
    }
}

// No Extra Charges
class NormalStrategy implements PricingStrategy {
    public int adjustFare(int baseFare) {
        return baseFare; // no change
    }
}
```

```java
// Bike class
class Bike implements Vehicle {
    PricingStrategy strategy;

    public Bike(PricingStrategy strategy) {
        this.strategy = strategy;
    }

    public void startRide() {
        System.out.println("Bike ride started.");
    }

    public void endRide() {
        System.out.println("Bike ride ended.");
    }

    public int calculateFare(int distance) {
        int baseFare = distance * 5;
        return strategy.adjustFare(baseFare);
    }
}

// Auto class
class Auto implements Vehicle {
    PricingStrategy strategy;

    public Auto(PricingStrategy strategy) {
        this.strategy = strategy;
    }

    public void startRide() {
        System.out.println("Auto ride started.");
    }

    public void endRide() {
        System.out.println("Auto ride ended.");
    }

    public int calculateFare(int distance) {
        int baseFare = distance * 8;
        return strategy.adjustFare(baseFare);
    }
}
```

```java
// Cab class
class Cab implements Vehicle {
    PricingStrategy strategy;

    public Cab(PricingStrategy strategy) {
        this.strategy = strategy;
    }

    public void startRide() {
        System.out.println("Cab ride started.");
    }

    public void endRide() {
        System.out.println("Cab ride ended.");
    }

    public int calculateFare(int distance) {
        int baseFare = distance * 12;
        return strategy.adjustFare(baseFare);
    }
}

// Main class
public class RideSystem {
    public static void main(String[] args) {
        Vehicle bike = new Bike(new PeakHourStrategy());
        Vehicle auto = new Auto(new HolidayStrategy());
        Vehicle cab = new Cab(new NormalStrategy());

        bike.startRide();
        System.out.println("Bike Fare: ₹" + bike.calculateFare(10));
        bike.endRide();

        auto.startRide();
        System.out.println("Auto Fare: ₹" + auto.calculateFare(10));
        auto.endRide();

        cab.startRide();
        System.out.println("Cab Fare: ₹" + cab.calculateFare(10));
        cab.endRide();
    }
}
```

17) Design a University Staff Management System using a base class Staff and derived classes Professor, AdministrativeStaff, and MaintenanceStaff.
Override methods like displayDetails() and calculateBonus() differently in each subclass using polymorphism.
Use a list of base class pointers or references to manage multiple staff objects and demonstrate runtime polymorphism.
(Advanced) Implement a promote() method with different behaviors in each subclass.

```
import java.util.*;

// Base class
class Staff {
    String name;
    double salary;

    Staff(String name, double salary) {
        this.name = name;
        this.salary = salary;
    }

    void displayDetails() {
        System.out.println("Name: " + name + ", Salary: " + salary);
    }

    double calculateBonus() {
        return 0; // Base class has no bonus
    }

    void promote() {
        System.out.println("Promotion policy for general staff.");
    }
}

// Professor subclass
class Professor extends Staff {
    String subject;

    Professor(String name, double salary, String subject) {
        super(name, salary);
        this.subject = subject;
    }

    @Override
    void displayDetails() {
        System.out.println("Professor: " + name + ", Subject: " + subject + ",
Salary: " + salary);
```

```
    }

    @Override
    double calculateBonus() {
        return salary * 0.20; // 20% bonus
    }

    @Override
    void promote() {
        System.out.println(name + " is promoted to Senior Professor.");
    }
}

// AdministrativeStaff subclass
class AdministrativeStaff extends Staff {
    String department;

    AdministrativeStaff(String name, double salary, String department) {
        super(name, salary);
        this.department = department;
    }

    @Override
    void displayDetails() {
        System.out.println("Admin Staff: " + name + ", Department: " + department
+ ", Salary: " + salary);
    }

    @Override
    double calculateBonus() {
        return salary * 0.10; // 10% bonus
    }

    @Override
    void promote() {
        System.out.println(name + " is promoted to Senior Admin Officer.");
    }
}

// MaintenanceStaff subclass
class MaintenanceStaff extends Staff {
    String shift;

    MaintenanceStaff(String name, double salary, String shift) {
        super(name, salary);
```

```
        this.shift = shift;
    }

    @Override
    void displayDetails() {
        System.out.println("Maintenance Staff: " + name + ", Shift: " + shift +
", Salary: " + salary);
    }

    @Override
    double calculateBonus() {
        return salary * 0.05; // 5% bonus
    }

    @Override
    void promote() {
        System.out.println(name + " is promoted to Supervisor.");
    }
}

// Main class
public class University {
    public static void main(String[] args) {
        // List of base class references
        List<Staff> staffList = new ArrayList<>();

        staffList.add(new Professor("Dr. Ahuja", 90000, "AI & ML"));
        staffList.add(new AdministrativeStaff("Mr. Mehta", 50000, "Admissions"));
        staffList.add(new MaintenanceStaff("Ravi", 30000, "Night"));

        // Polymorphic behavior
        for (Staff staff : staffList) {
            staff.displayDetails();
            System.out.println("Bonus: " + staff.calculateBonus());
            staff.promote();
            System.out.println("------------------------");
        }
    }
}
```

18)

1. Create a class Rectangle with attributes length and breadth. Write a constructor that uses this keyword to initialize the attributes. Also, create a method compareArea(Rectangle r) that compares the area of two rectangles.
2. Create a class MathOperations with:
    - A static method square(int n) that returns the square of a number.
    - An instance method cube(int n) that returns the cube of a number. Write a Java program that demonstrates calling both static and instance methods properly.

```java
// Rectangle class
class Rectangle {
    int length;
    int breadth;

    // Constructor using 'this' keyword
    Rectangle(int length, int breadth) {
        this.length = length;
        this.breadth = breadth;
    }

    // Method to compare area with another rectangle
    void compareArea(Rectangle r) {
        int area1 = this.length * this.breadth;
        int area2 = r.length * r.breadth;

        if (area1 > area2) {
            System.out.println("Current rectangle has a larger area.");
        } else if (area1 < area2) {
            System.out.println("Given rectangle has a larger area.");
        } else {
            System.out.println("Both rectangles have the same area.");
        }
    }
}

// MathOperations class
class MathOperations {
    // Static method to return square of a number
    static int square(int n) {
        return n * n;
    }
```

```
    // Instance method to return cube of a number
    int cube(int n) {
        return n * n * n;
    }
}

// Main class to test both functionalities
public class RectangleAndMathOp {
    public static void main(String[] args) {
        // Test Rectangle comparison
        Rectangle rect1 = new Rectangle(5, 4);
        Rectangle rect2 = new Rectangle(6, 3);
        rect1.compareArea(rect2);

        // Test MathOperations
        int num = 3;

        // Call static method
        int squareResult = MathOperations.square(num);
        System.out.println("Square of " + num + " is: " +
squareResult);

        // Call instance method
        MathOperations mathOp = new MathOperations();
        int cubeResult = mathOp.cube(num);
        System.out.println("Cube of " + num + " is: " + cubeResult);
    }
}
```

19) Create a Java program for a seating system in a cinema hall represented by a 2D array (rows × columns).

- Mark all seats as available (e.g., 0) initially.
- Allow the user to book seats by marking them as booked (e.g., 1).
- Display the current seat map (matrix form).
- Add functionality to check if a given seat is available before booking.

HARDCODED

public class CinemaHall {

```
    public static void main(String[] args) {
        int rows = 5;
        int cols = 6;
        int[][] seats = new int[rows][cols]; // 0 = available, 1 = booked

        // Display initial seat map
        System.out.println("Initial Seat Map:");
        displaySeats(seats);

        // Hardcoded bookings
        bookSeat(seats, 2, 3);  // Book seat at row 2, col 3
        bookSeat(seats, 0, 0);  // Book seat at row 0, col 0
        bookSeat(seats, 4, 5);  // Book seat at row 4, col 5
        bookSeat(seats, 2, 3);  // Attempt to rebook the same seat (should be blocked)

        // Final seat map
        System.out.println("\nFinal Seat Map:");
        displaySeats(seats);
    }

    // Display the current seat matrix
    public static void displaySeats(int[][] seats) {
        for (int[] row : seats) {
            for (int seat : row) {
```

```
                System.out.print(seat + " ");
            }
            System.out.println();
        }
    }

    // Attempt to book a seat
    public static void bookSeat(int[][] seats, int row, int col) {
        System.out.println("\nBooking seat at row " + row + ", column " + col + ":");
        if (row < 0 || row >= seats.length || col < 0 || col >= seats[0].length) {
            System.out.println("Invalid seat coordinates.");
        } else if (seats[row][col] == 1) {
            System.out.println("Seat already booked.");
        } else {
            seats[row][col] = 1;
            System.out.println("Seat booked successfully.");
        }
    }
}
```

USER INPUT

```
import java.util.Scanner;

public class CinemaSeating {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

```

```
        // Define number of rows and columns in the cinema hall
        int rows = 5, cols = 5;
        int[][] seats = new int[rows][cols]; // All seats initialized to 0

        while (true) {
            System.out.println("\n--- Cinema Seating System ---");
            System.out.println("1. Book a seat");
            System.out.println("2. Show seat map");
            System.out.println("3. Exit");
            System.out.print("Enter your choice: ");
            int choice = sc.nextInt();

            if (choice == 1) {
                System.out.print("Enter row (0 to " + (rows - 1) + "): ");
                int row = sc.nextInt();
                System.out.print("Enter column (0 to " + (cols - 1) + "): ");
                int col = sc.nextInt();

                if (row >= 0 && row < rows && col >= 0 && col < cols) {
                    if (seats[row][col] == 0) {
                        seats[row][col] = 1;
                        System.out.println("Seat booked successfully!");
                    } else {
                        System.out.println("Seat already booked!");
                    }
                } else {
                    System.out.println("Invalid seat position!");
                }
            } else if (choice == 2) {
                System.out.println("\nSeat Map (0 = Available, 1 = Booked):");
                for (int i = 0; i < rows; i++) {
                    for (int j = 0; j < cols; j++) {
                        System.out.print(seats[i][j] + " ");
                    }
                    System.out.println();
                }
            } else if (choice == 3) {
                System.out.println("Exiting the system.");
                break;
```

```
            } else {
                System.out.println("Invalid choice! Try again.");
            }
        }

        sc.close();
    }
}

```

20)

Develop a Java program that takes a paragraph input from the user and:

- Remove all vowels (a, e, i, o, u) from the paragraph using StringBuilder.
- Efficiently update the paragraph after each deletion.

Finally, display the transformed paragraph along with the count of characters removed.

HARDCODED

```
public class RemoveVowels {

    public static void main(String[] args) {
        // Hardcoded paragraph
        String paragraph = "This is a simple example paragraph where vowels will be removed.";

        // Use StringBuilder to modify the paragraph
        StringBuilder modifiedParagraph = new StringBuilder(paragraph);

        // Counter for vowels removed
        int vowelsRemoved = 0;

        // Vowels to remove
        String vowels = "aeiouAEIOU";

        // Traverse through the string and remove vowels
        for (int i = 0; i < modifiedParagraph.length(); i++) {
            char currentChar = modifiedParagraph.charAt(i);
            if (vowels.indexOf(currentChar) != -1) {
                modifiedParagraph.deleteCharAt(i);  // Remove the vowel
```

```
            vowelsRemoved++;  // Increment the removed vowels counter
            i--;  // Adjust the index due to the character shift
        }
    }

    // Output the transformed paragraph and the count of vowels removed
    System.out.println("Original Paragraph:");
    System.out.println(paragraph);
    System.out.println("\nTransformed Paragraph (without vowels):");
    System.out.println(modifiedParagraph.toString());
    System.out.println("\nTotal vowels removed: " + vowelsRemoved);
  }
}
```

- **USER INPUT**

```
import java.util.Scanner;

public class VowelRemoval {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Take input from the user
        System.out.println("Enter a paragraph:");
        String paragraph = scanner.nextLine();

        // Initialize StringBuilder with the original paragraph
        StringBuilder transformedParagraph = new
StringBuilder(paragraph);

        // Count of removed vowels
        int removedCount = 0;

        // Loop through the paragraph and remove vowels
        for (int i = 0; i < transformedParagraph.length(); i++) {
            char currentChar = transformedParagraph.charAt(i);

            // Check if the current character is a vowel (both
lowercase and uppercase)
            if (isVowel(currentChar)) {
                transformedParagraph.deleteCharAt(i); // Remove the
vowel
                removedCount++; // Increment the removed vowel count
                i--; // Adjust index after removal
```

```
            }
        }

        // Display the transformed paragraph and the count of
removed characters
        System.out.println("Transformed Paragraph: " +
transformedParagraph);
        System.out.println("Total Characters Removed: " +
removedCount);
    }

    // Helper method to check if a character is a vowel
    public static boolean isVowel(char c) {
        c = Character.toLowerCase(c); // Convert to lowercase for
uniform comparison
        return c == 'a' || c == 'e' || c == 'i' || c == 'o' || c ==
'u';
    }
}

```

21) Create a base class `Employee` with attributes: name, id, and basicSalary, along with methods `displayDetails()` and `calculateSalary()`. Derive two subclasses: `Manager` with an additional bonus attribute and `Developer` with a projectAllowance attribute. Override the `calculateSalary()` method in both subclasses to include their respective additional amounts. In the main method, create objects of Manager and Developer, and call `displayDetails()` for each. Demonstrate polymorphism by invoking `calculateSalary()` using base class references and display the total salary.

```
22)// Base class
23)class Employees {
24)    String name;
25)    int id;
26)    double basicSalary;
27)
28)    Employees(String name, int id, double basicSalary) {
29)        this.name = name;
30)        this.id = id;
31)        this.basicSalary = basicSalary;
32)    }
33)
34)    void displayDetails() {
35)        System.out.println("Name: " + name);
36)        System.out.println("ID: " + id);
```

```
37)         System.out.println("Basic Salary: " + basicSalary);
38)     }
39)
40)     double calculateSalary() {
41)         return basicSalary;
42)     }
43)}
44)
45)// Subclass: Manager
46)class Manager extends Employees {
47)     double bonus;
48)
49)     Manager(String name, int id, double basicSalary, double bonus) {
50)         super(name, id, basicSalary);
51)         this.bonus = bonus;
52)     }
53)
54)     @Override
55)     double calculateSalary() {
56)         return basicSalary + bonus;
57)     }
58)}
59)
60)// Subclass: Developer
61)class Developer extends Employees {
62)     double projectAllowance;
63)
64)     Developer(String name, int id, double basicSalary, double
   projectAllowance) {
65)         super(name, id, basicSalary);
66)         this.projectAllowance = projectAllowance;
67)     }
68)
69)     @Override
70)     double calculateSalary() {
71)         return basicSalary + projectAllowance;
72)     }
73)}
74)
75)// Main class
76)public class Company {
77)     public static void main(String[] args) {
78)         // Creating Manager and Developer objects
79)         Employees emp1 = new Manager("Alice", 101, 50000, 10000);
80)         Employees emp2 = new Developer("Bob", 102, 40000, 8000);
```

```
81)
82)         // Display details and calculate salary using polymorphism
83)         System.out.println("--- Manager Details ---");
84)         emp1.displayDetails();
85)         System.out.println("Total Salary: " + emp1.calculateSalary());
86)
87)         System.out.println("\n--- Developer Details ---");
88)         emp2.displayDetails();
89)         System.out.println("Total Salary: " + emp2.calculateSalary());
90)     }
91)}
92)
```

22)

Create a class BankAccount with the following attributes and methods:

Attributes:

accountNumber (String)

balance (double)

Methods:

A constructor that initializes the accountNumber and balance.

A method withdraw(double amount) that:

Throws an ArithmeticException if the withdrawal amount is greater than the current balance.

Throws an IllegalArgumentException if the withdrawal amount is less than or equal to zero.

In the main() method:

Create an object of the BankAccount class with an initial balance.

Use try-catch blocks to handle the following scenarios:

Catch the ArithmeticException and display a message "Insufficient funds for withdrawal."

Catch the IllegalArgumentException and display a message "Invalid withdrawal amount."

After handling the exception, allow the program to continue running.

Display the current balance after each operation.

```
class BankAccounts {
    String accountNumber;
    double balance;

    // Constructor
    BankAccounts(String accountNumber, double balance) {
        this.accountNumber = accountNumber;
        this.balance = balance;
    }

    // Withdraw method
    void withdraw(double amount) {
        if (amount <= 0) {
            throw new IllegalArgumentException("Amount must be greater than
zero.");
        }
        if (amount > balance) {
            throw new ArithmeticException("Not enough balance.");
        }
        balance -= amount;
        System.out.println("Withdrawal of " + amount + " successful.");
    }

    // Method to display current balance
    void displayBalance() {
        System.out.println("Current Balance: " + balance);
    }
}

// Main class
public class BankDemo {
    public static void main(String[] args) {
        BankAccounts myAccount = new BankAccounts("ACC123", 1000.0);

        // First attempt: invalid (negative amount)
        try {
            myAccount.withdraw(-200);
        } catch (IllegalArgumentException e) {
```

```
            System.out.println("Invalid withdrawal amount.");
        } catch (ArithmeticException e) {
            System.out.println("Insufficient funds for withdrawal.");
        }
        myAccount.displayBalance();

        // Second attempt: invalid (exceeds balance)
        try {
            myAccount.withdraw(1500);
        } catch (IllegalArgumentException e) {
            System.out.println("Invalid withdrawal amount.");
        } catch (ArithmeticException e) {
            System.out.println("Insufficient funds for withdrawal.");
        }
        myAccount.displayBalance();

        // Third attempt: valid withdrawal
        try {
            myAccount.withdraw(500);
        } catch (IllegalArgumentException e) {
            System.out.println("Invalid withdrawal amount.");
        } catch (ArithmeticException e) {
            System.out.println("Insufficient funds for withdrawal.");
        }
        myAccount.displayBalance();
    }
}
```

23)

Create a class Printer with a method printNumbers() that prints the numbers from 1 to 10 with a small delay between each number (use Thread.sleep(500) to simulate the delay).

Create two threads:

One thread will call the printNumbers() method and print numbers from 1 to 10.

The second thread will call the printNumbers() method and also print numbers from 1 to 10.

In the main() method:

Create two Printer objects.

Create two threads and start them to execute the printNumbers() method concurrently.

Ensure that the numbers from both threads are printed without any interruption.

- Additional attribute: department (String)

```
class Printer {
    String department;

    Printer(String department) {
        this.department = department;
    }

    // Synchronized method to avoid interruption
    synchronized void printNumbers() {
        System.out.println("Printing numbers for department: " +
department);
        for (int i = 1; i <= 10; i++) {
            System.out.println(department + " - " + i);
            try {
                Thread.sleep(500); // 0.5 second delay
            } catch (InterruptedException e) {
                System.out.println("Thread interrupted");
            }
        }
    }
}

// Thread class that uses Printer
class PrinterTask extends Thread {
    Printer printer;

    PrinterTask(Printer printer) {
        this.printer = printer;
    }

    @Override
    public void run() {
        printer.printNumbers();
    }
}

```

```java
// Main class
public class Printers {
    public static void main(String[] args) {
        // Creating two Printer objects with different departments
        Printer printer1 = new Printer("HR");
        Printer printer2 = new Printer("IT");

        // Creating two threads
        Thread t1 = new PrinterTask(printer1);
        Thread t2 = new PrinterTask(printer2);

        // Start the threads
        t1.start();
        t2.start();
    }
}
```

24) Create a Java program to demonstrate access modifiers (private, public, protected, and default).

1. Class Person:
    - Attributes:
        - name (private), age (public), address (protected), phoneNumber (default)
    - Methods:
        - Constructor to initialize all attributes.
        - displayDetails() (public) to display name, age, and address.
        - updatePhoneNumber() (public) to update phoneNumber.
2. Class Employee (extends Person):
    - Additional attribute: employeeId (public)
    - Override displayDetails() to include employeeId.

3. In the main() method:

   - Create an Employee object and demonstrate access to attributes and methods using different access modifiers.

```
// Base class
class Person {
    private String name;           // Private
    public int age;                // Public
    protected String address;      // Protected
    String phoneNumber;           // Default (package-private)

    // Constructor to initialize all attributes
    public Person(String name, int age, String address, String phoneNumber) {
        this.name = name;
        this.age = age;
        this.address = address;
        this.phoneNumber = phoneNumber;
    }

    // Public method to display name, age, and address
    public void displayDetails() {
        System.out.println("Name: " + name);  // Accessing private attribute inside class
        System.out.println("Age: " + age);
        System.out.println("Address: " + address);
    }

    // Public method to update phone number
    public void updatePhoneNumber(String newNumber) {
        phoneNumber = newNumber;
    }
}

// Derived class
class Employee extends Person {
    public String employeeId;  // Public attribute

    // Constructor
    public Employee(String name, int age, String address, String phoneNumber, String employeeId) {
        super(name, age, address, phoneNumber);
        this.employeeId = employeeId;
    }
```

```
41.
42.     // Override displayDetails method to include employeeId
43.     @Override
44.     public void displayDetails() {
45.         super.displayDetails(); // Call parent display
46.         System.out.println("Employee ID: " + employeeId);
47.     }
48.}
49.
50.// Main class
51.public class AccessModifiers {
52.     public static void main(String[] args) {
53.         // Create an Employee object
54.         Employee emp = new Employee("Alice", 30, "Pune", "9876543210",
   "EMP123");
55.
56.         // Accessing public method
57.         emp.displayDetails();
58.
59.         // Accessing and updating public field
60.         emp.age = 31;
61.         System.out.println("Updated Age: " + emp.age);
62.
63.         // Updating phone number using public method
64.         emp.updatePhoneNumber("9998887776");
65.
66.         // Accessing protected field (allowed since Employee is a
   subclass)
67.         System.out.println("Protected Address: " + emp.address);
68.
69.         // Accessing default (package-private) field
70.         System.out.println("Default Phone Number: " + emp.phoneNumber);
71.
72.         // ❌ The following would give error because `name` is private:
73.         // System.out.println(emp.name); // Not allowed
74.     }
75.}
76.
```

25)

1) Create an application for employee management with the following classes:

a) Create an Employee class with following attributes and behaviors:

i) int empId
ii) String empName
iii) String email
iv) String gender
v) float salary
vi) void GetEmployeeDetails() -> prints employee details

b) Create one more class EmployeeDB with the following attributes and behaviors:
i) ArrayList list;
ii) boolean addEmployee(Employee e) -> adds the employee object to the collection
iii) boolean deleteEmployee(int empId) -> delete the employee object from the collection with the given empid
iv) String showPaySlip(int empId) -> returns the payslip of the employee with the given empId

```java
import java.util.ArrayList;

// Employee class
class Employeee {
    int empId;
    String empName;
    String email;
    String gender;
    float salary;

    // Constructor
    public Employeee(int empId, String empName, String email, String gender,
float salary) {
        this.empId = empId;
        this.empName = empName;
        this.email = email;
        this.gender = gender;
        this.salary = salary;
    }

    // Method to print employee details
    public void getEmployeeDetails() {
        System.out.println("ID: " + empId);
        System.out.println("Name: " + empName);
        System.out.println("Email: " + email);
        System.out.println("Gender: " + gender);
        System.out.println("Salary: Rs. " + salary);
    }
```

```
}

// EmployeeDB class
class EmployeeDB {
    ArrayList<Employeee> list = new ArrayList<>();

    // Add an employee
    public boolean addEmployee(Employeee e) {
        return list.add(e);
    }

    // Delete an employee by ID
    public boolean deleteEmployee(int empId) {
        for (Employeee e : list) {
            if (e.empId == empId) {
                list.remove(e);
                return true;
            }
        }
        return false;
    }

    // Show pay slip
    public String showPaySlip(int empId) {
        for (Employeee e : list) {
            if (e.empId == empId) {
                return "PaySlip for Employee ID " + empId + ": Rs. " + e.salary;
            }
        }
        return "Employee not found.";
    }
}

// Main class
public class EmployeeManagementApp {
    public static void main(String[] args) {
        // Create Employee objects
        Employeee e1 = new Employeee(101, "Alice", "alice@example.com", "Female",
50000f);
        Employeee e2 = new Employeee(102, "Bob", "bob@example.com", "Male",
60000f);

        // Create the database
        EmployeeDB db = new EmployeeDB();
```

```
        // Add employees
        db.addEmployee(e1);
        db.addEmployee(e2);

        // Display employee details
        System.out.println("\nEmployee Details:");
        e1.getEmployeeDetails();
        System.out.println();
        e2.getEmployeeDetails();

        // Show payslip
        System.out.println("\n" + db.showPaySlip(101));

        // Delete employee
        boolean deleted = db.deleteEmployee(102);
        System.out.println("\nEmployee 102 deleted: " + deleted);

        // Try showing payslip for deleted employee
        System.out.println(db.showPaySlip(102));
    }
}
```

---

26)
You are given a sorted integer array nums in non-decreasing order. Your task is to remove the duplicate elements in-place such that each element appears only once, and return the new length of the modified array.

You must not allocate extra space for another array; you must do this by modifying the input array in-place with O(1) extra memory.

After removing the duplicates, the first part of the array should contain the unique elements, and the remaining elements can be left as any value (underscores _ or any arbitrary values).

Example:

Input: nums = [0,0,1,1,1,2,2,3,3,4]
Output: 5, nums = [0,1,2,3,4,_,_,_,_,_]

```java
public class RemoveDuplicates {
    public static int removeDuplicates(int[] nums) {
        // Edge case: if the array is empty, return 0
        if (nums.length == 0) {
            return 0;
        }

        int i = 0; // Pointer to track the last unique element's index

        for (int j = 1; j < nums.length; j++) {
            // If we find a unique element
            if (nums[j] != nums[i]) {
                i++; // Move the pointer to the next unique element's position
                nums[i] = nums[j]; // Update the unique element at nums[i]
            }
        }

        // Return the length of the array after removing duplicates
        return i + 1;
    }

    public static void main(String[] args) {
        int[] nums = {0, 0, 1, 1, 1, 2, 2, 3, 3, 4};

        // Remove duplicates and get the new length
        int newLength = removeDuplicates(nums);

        // Print the new length
        System.out.println("New length: " + newLength);

        // Print the modified array (first part with unique elements)
        System.out.print("Modified array: ");
        for (int i = 0; i < newLength; i++) {
            System.out.print(nums[i] + " ");
        }

        // Optionally, print underscores for the rest of the array
        for (int i = newLength; i < nums.length; i++) {
            System.out.print("_ ");
        }
    }
}
```

27)

Write a class MathOperation which accepts 5 integers through the command line. Create an array using these parameters. Loop through the array and obtain the sum and average of all the elements and display the result.

Handle various exceptions that may arise such as:

- ArithmeticException
- NumberFormatException
- and other relevant exceptions.

```java
import java.util.Scanner;

public class Average {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        int[] nums = new int[5];
        int sum = 0;
        double average = 0.0;

        try {
            // Prompt the user for 5 integers
            System.out.println("Please enter 5 integers:");

            for (int i = 0; i < 5; i++) {
                System.out.print("Enter integer " + (i + 1) + ": ");
                nums[i] = scanner.nextInt();
            }

            // Calculate the sum
            for (int num : nums) {
                sum += num;
            }

            // Calculate the average
            if (nums.length != 0) {
                average = sum / (double) nums.length;
            }

```

```
            // Display results
            System.out.println("Sum: " + sum);
            System.out.println("Average: " + average);

        } catch (NumberFormatException e) {
            System.out.println("Error: Please enter valid integers.");
        } catch (ArithmeticException e) {
            System.out.println("Error: Arithmetic exception occurred.");
        } catch (Exception e) {
            System.out.println("An unexpected error occurred: " +
e.getMessage());
        } finally {
            scanner.close(); // Close the scanner to prevent resource leak
        }
    }
}

```

---

28)

Create a base class named Fruit with the following attributes:

- name (String)
- taste (String)
- size (String)

Define a method eat() in the Fruit class that prints the name and taste of the fruit.

Now, create two subclasses, Apple and Orange, that inherit from the Fruit class. Override the eat() method in each subclass to display the specific taste of that fruit.

```
// Base class Fruit
class Fruit {
    String name;
    String taste;
    String size;

    // Constructor for Fruit class to initialize attributes
```

```java
    public Fruit(String name, String taste, String size) {
        this.name = name;
        this.taste = taste;
        this.size = size;
    }

    // Method to print the name and taste of the fruit
    void eat() {
        System.out.println("Name of fruit is: " + name);
        System.out.println("Taste is: " + taste);
    }
}

// Subclass Apple inherits from Fruit
class Apple extends Fruit {

    // Constructor for Apple class
    public Apple(String name, String taste, String size) {
        super(name, taste, size);
    }

    @Override
    void eat() {
        System.out.println("Apples are sweet.");
    }
}

// Subclass Orange inherits from Fruit
class Orange extends Fruit {

    // Constructor for Orange class
    public Orange(String name, String taste, String size) {
        super(name, taste, size);
    }

    @Override
    void eat() {
        System.out.println("Oranges are tangy and sweet.");
    }
}

// Main class to test the implementation
public class Fruits {
    public static void main(String[] args) {
```

```
        // Create objects of Apple and Orange with specific attributes
        Fruit apple = new Apple("Apple", "Sweet", "Medium");
        Fruit orange = new Orange("Orange", "Tangy and Sweet", "Medium");

        // Call the eat method to display the fruit's taste
        apple.eat();
        orange.eat();
    }
}
```

---

29)
Given a number N, the task is to count the number of unique digits in the given number.
Examples:

Input: N = 22342 Output: 2

Explanation: The digits 3 and 4 occurs only once. Hence, the output is 2.

Input: N = 99677 Output:1
Explanation: The digit 6 occurs only once. Hence, the output is 1.

HARDCODED

```
public class UniqueDigits {


   public static void main(String[] args) {
       // Hardcoded number N
       int N = 22342;  // You can change this number for testing with different inputs


       // Convert the number to a string for easier manipulation
```

```
        String numberStr = Integer.toString(N);

        // Initialize an array to count occurrences of each digit (0 to 9)
        int[] digitCount = new int[10];

        // Loop through each character in the string and count the occurrences of
each digit
        for (char digitChar : numberStr.toCharArray()) {
            int digit = digitChar - '0';  // Convert char to int
            digitCount[digit]++;
        }

        // Count the number of unique digits
        int uniqueDigitCount = 0;
        for (int count : digitCount) {
            if (count == 1) {
                uniqueDigitCount++;
            }
        }

        // Output the result
        System.out.println("The number of unique digits is: " + uniqueDigitCount);
    }
}
```

USER INPUT

```java
import java.util.Scanner;

public class UniqueDigitCounter {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // User input
        System.out.print("Enter a number: ");
        String number = scanner.nextLine();

        // Array to count digit frequencies (0 to 9)
        int[] count = new int[10];

        // Count the frequency of each digit
        for (int i = 0; i < number.length(); i++) {
            char ch = number.charAt(i);
            if (Character.isDigit(ch)) {
                int digit = ch - '0';
                count[digit]++;
            }
        }

        // Count how many digits appeared only once
        int uniqueCount = 0;
        for (int i = 0; i < 10; i++) {
            if (count[i] == 1) {
                uniqueCount++;
            }
        }

        System.out.println("Number of unique digits: " + uniqueCount);
    }
}
```

30)

Given an array of positive integers nums and a positive integer target, return the minimal length of a subarray whose sum is greater than or equal to target. If there is no such subarray, return 0 instead.

Example 1: Input: target = 7, nums = [2,3,1,2,4,3] Output: 2

Example 2: Input: target = 11, nums = [1,1,1,1,1,1,1,1] Output: 0**HARDCODED**

```
public class MinSubArrayLength {
    public static int findMinLength(int target, int[] nums) {
        int start = 0;
        int currentSum = 0;
        int minLength = nums.length + 1;

        for (int end = 0; end < nums.length; end++) {
            currentSum += nums[end];

            while (currentSum >= target) {
                int windowLength = end - start + 1;
                if (windowLength < minLength) {
                    minLength = windowLength;
                }

                currentSum -= nums[start];
                start++;
            }
        }

        if (minLength == nums.length + 1) {
            return 0;
        }

        return minLength;
    }

    public static void main(String[] args) {
        int[] nums1 = {2, 3, 1, 2, 4, 3};
        int target1 = 7;
        System.out.println("Output: " + findMinLength(target1, nums1)); //
Output: 2

        int[] nums2 = {1, 1, 1, 1, 1, 1, 1};
        int target2 = 11;
        System.out.println("Output: " + findMinLength(target2, nums2)); //
Output: 0
    }
}
```

31)

Write a Java program to receive an integer number as a command-line argument, and print the binary, octal, and hexadecimal equivalents of the given number.

Given Number : 20
Binary equivalent : 10100
Octal equivalent : 24
Hexadecimal equivalent : 14

HARDCODED

```
public class NumberConverter {
   public static void main(String[] args) {
      // Hardcoded integer value
      int number = 123;  // You can change this to any integer

      // Convert to binary, octal, and hexadecimal
      String binary = Integer.toBinaryString(number);
      String octal = Integer.toOctalString(number);
      String hex = Integer.toHexString(number).toUpperCase();

      // Display the results
      System.out.println("Given Number: " + number);
      System.out.println("Binary equivalent: " + binary);
      System.out.println("Octal equivalent: " + octal);
      System.out.println("Hexadecimal equivalent: " + hex);
   }
}
```

USER INPUT

```
import java.util.Scanner;

public class NumberConverter {
    public static void main(String[] args) {
        // Create Scanner object to read input
        Scanner scanner = new Scanner(System.in);

        // Prompt the user to enter an integer
        System.out.print("Enter an integer: ");
        int number = scanner.nextInt();

        // Convert to binary, octal, and hexadecimal
        String binary = Integer.toBinaryString(number);
```

```
        String octal = Integer.toOctalString(number);
        String hex = Integer.toHexString(number).toUpperCase();

        // Display the results
        System.out.println("Given Number: " + number);
        System.out.println("Binary equivalent: " + binary);
        System.out.println("Octal equivalent: " + octal);
        System.out.println("Hexadecimal equivalent: " + hex);
    }
}
```

---

32) Develop an online payment system that supports different payment methods.

Requirements:

1. Create an abstract class Payment with:
    - Attributes: amount, transactionID.
    - Abstract method processPayment().
    - Concrete method showTransactionDetails().
2. Create subclasses:
    - CreditCardPayment (cardNumber, CVV, expiryDate).
    - PayPalPayment (email, password).
    - UPIPayment (UPI ID).
3. Implement the processPayment() method in each subclass to handle payments uniquely.
4. Create a PaymentGateway class to process transactions dynamically.

```
// Abstract Payment Class
abstract class Payment {
    double amount;
    String transactionID;

    public Payment(double amount, String transactionID) {
        this.amount = amount;
        this.transactionID = transactionID;
    }

    // Abstract method
    public abstract void processPayment();

```

```
    // Concrete method
    public void showTransactionDetails() {
        System.out.println("Transaction ID: " + transactionID);
        System.out.println("Amount: $" + amount);
    }
}

// Credit Card Payment
class CreditCardPayment extends Payment {
    String cardNumber;

    public CreditCardPayment(double amount, String transactionID, String cardNumber) {
        super(amount, transactionID);
        this.cardNumber = cardNumber;
    }

    @Override
    public void processPayment() {
        System.out.println("Processing Credit Card payment...");
        System.out.println("Card Number: **** **** **** " + cardNumber.substring(cardNumber.length() - 4));
        showTransactionDetails();
    }
}

// PayPal Payment
class PayPalPayment extends Payment {
    String email;

    public PayPalPayment(double amount, String transactionID, String email) {
        super(amount, transactionID);
        this.email = email;
    }

    @Override
    public void processPayment() {
        System.out.println("Processing PayPal payment...");
        System.out.println("PayPal Email: " + email);
        showTransactionDetails();
    }
}

// UPI Payment
```

```java
class UPIPayment extends Payment {
    String upiID;

    public UPIPayment(double amount, String transactionID, String upiID) {
        super(amount, transactionID);
        this.upiID = upiID;
    }

    @Override
    public void processPayment() {
        System.out.println("Processing UPI payment...");
        System.out.println("UPI ID: " + upiID);
        showTransactionDetails();
    }
}

// Payment Gateway
class PaymentGateway {
    public void processTransaction(Payment payment) {
        payment.processPayment();
        System.out.println("Payment successful!\n");
    }
}

// Main Class
public class OnlinePaymentSystem {
    public static void main(String[] args) {
        PaymentGateway gateway = new PaymentGateway();

        // Create different payment types
        Payment creditCard = new CreditCardPayment(150.75, "TXN1001", "1234567812345678");
        Payment paypal = new PayPalPayment(99.99, "TXN1002", "user@example.com");
        Payment upi = new UPIPayment(50.00, "TXN1003", "user@upi");

        // Process transactions
        gateway.processTransaction(creditCard);
        gateway.processTransaction(paypal);
        gateway.processTransaction(upi);
    }
}

```

33) Design a simple system to calculate the area of different 2D shapes using interfaces in Java.Define an interface named Shape with a method:

Create the following classes that implement the Shape interface:

- Circle with a field radius
- Rectangle with fields length and width
- Triangle with fields base and height

Each class must implement the calculateArea() method according to the respective formula:

- Circle: $\pi \times \text{radius}^2$
- Rectangle: length × width
- Triangle: 0.5 × base × height

In the main() method, use polymorphism to create an array of Shape references and call calculateArea() on each.

```
// Shape Interface
interface Shape {
    double calculateArea();
}

// Circle class
class Circle implements Shape {
    private double radius;

    // Constructor
    public Circle(double radius) {
        this.radius = radius;
    }

    // Area calculation for Circle
    public double calculateArea() {
        return Math.PI * radius * radius;
    }
}

// Rectangle class
class Rectangle implements Shape {
```

```
    private double length;
    private double width;
    // Constructor
    public Rectangle(double length, double width) {
        this.length = length;
        this.width = width;
    }

    // Area calculation for Rectangle
    public double calculateArea() {
        return length * width;
    }
}

// Triangle class
class Triangle implements Shape {
    private double base;
    private double height;

    // Constructor
    public Triangle(double base, double height) {
        this.base = base;
        this.height = height;
    }

    // Area calculation for Triangle
    public double calculateArea() {
        return 0.5 * base * height;
    }
}

// Main class
public class ShapeAreaCalculator {
    public static void main(String[] args) {
        // Create objects for each shape
        Shape circle = new Circle(5);  // Circle with radius 5
        Shape rectangle = new Rectangle(4, 6);  // Rectangle with length 4, width
6
        Shape triangle = new Triangle(3, 7);  // Triangle with base 3, height 7

        // Calculate and print area for each shape
        System.out.println("Circle Area: " + circle.calculateArea());
        System.out.println("Rectangle Area: " + rectangle.calculateArea());
        System.out.println("Triangle Area: " + triangle.calculateArea());
    }
```

```
}
```

34). simple banking system that handles user withdrawals, including proper use of exception handling and custom exceptions.

Requirements:

1. Create a class BankAccount with the following:
    - Field: double balance
    - Constructor to initialize balance
    - Method:

      If the withdrawal amount is greater than the balance, throw a custom exception named InsufficientFundsException.
    - Otherwise, deduct the amount from the balance.

2. Define a custom exception class:
    - Include a constructor that accepts a custom error message.

      In the main() method:
    - Create a BankAccount object with an initial balance.
    - Try to withdraw different amounts (some valid, some invalid).
    - Catch the exception and display appropriate error messages.

```
// Custom exception class for Insufficient Funds
class InsufficientFundsException extends Exception {
    // Constructor accepting a custom error message
    public InsufficientFundsException(String message) {
        super(message);
    }
}

// BankAccount class
class BankAccount {
    private double balance;

    // Constructor to initialize the balance
    public BankAccount(double initialBalance) {
```

```
        balance = initialBalance;
    }

    // Method to withdraw an amount
    public void withdraw(double amount) throws InsufficientFundsException {
        if (amount > balance) {
            // This line throws the custom exception if funds are insufficient
            throw new InsufficientFundsException("Insufficient funds! Withdrawal
amount exceeds balance.");
        } else {
            balance -= amount;
            System.out.println("Withdrawal successful! New balance: $" +
balance);
        }
    }

    // Getter for balance (optional, for displaying the current balance)
    public double getBalance() {
        return balance;
    }
}

public class BankingSystem {
    public static void main(String[] args) {
        // Create a BankAccount object with an initial balance
        BankAccount account = new BankAccount(1000.0); // Initial balance of
$1000

        // Try to withdraw different amounts using try-catch
        try {
            // Valid withdrawal
            account.withdraw(500.0); // Withdraw $500
            // Invalid withdrawal (more than the balance)
            account.withdraw(600.0); // Try to withdraw $600, should throw
exception
        } catch (InsufficientFundsException e) {
            // This block is executed when an InsufficientFundsException is
thrown
            System.out.println("Error: " + e.getMessage());
        }

        // Final balance
        System.out.println("Final balance: $" + account.getBalance());
    }
}
```

35)
The Citizen class should have following attributes name, id, country, sex, maritalStatus, anualIncome, and economyStatus. Validate the fields if the age is below 18 and country is not 'India' throw NonEligibleException and give proper message. Use toString method to display the citizen object in proper format. Use separate packages for Exception and application classes

```
// Custom Exception class for non-eligibility
class NonEligibleException extends Exception {
    public NonEligibleException(String message) {
        super(message); // Pass the message to the parent exception class
    }
}

// Citizen class with necessary attributes
class Citizen {
    private String name;
    private int id;
    private String country;
    private String sex;
    private String maritalStatus;
    private double annualIncome;
    private String economyStatus;
    private int age;

    // Constructor to initialize Citizen object
    public Citizen(String name, int id, String country, String sex, String
maritalStatus, double annualIncome, String economyStatus, int age) {
        this.name = name;
        this.id = id;
        this.country = country;
        this.sex = sex;
        this.maritalStatus = maritalStatus;
        this.annualIncome = annualIncome;
        this.economyStatus = economyStatus;
        this.age = age;
    }

    // Method to check eligibility based on age and country
    public void checkEligibility() throws NonEligibleException {
        // Check if age is below 18 and country is not 'India'
        if (age < 18 && !country.equalsIgnoreCase("India")) {
```

```
            throw new NonEligibleException("Citizen is not eligible. Age below 18
and country is not India.");
        }
    }

    // Overriding toString method to display the citizen details in proper format
    @Override
    public String toString() {
        return "Citizen [Name: " + name + ", ID: " + id + ", Country: " + country
+ ", Sex: " + sex + ", Marital Status: " + maritalStatus +
                ", Annual Income: " + annualIncome + ", Economy Status: " +
economyStatus + ", Age: " + age + "]";
    }
}

// Main class to test the Citizen and exception
public class CitizenTest {
    public static void main(String[] args) {
        // Creating a Citizen object
        Citizen citizen = new Citizen("John Doe", 12345, "USA", "Male", "Single",
50000.0, "Middle-Class", 17);

        // Try to validate eligibility and print the Citizen details
        try {
            citizen.checkEligibility(); // This will throw exception if invalid
            System.out.println(citizen.toString()); // Display the citizen
details
        } catch (NonEligibleException e) {
            System.out.println(e.getMessage()); // Print exception message
        }
    }
}
```

36)

A. Write a Java program to remove prime numbers between 1 to 25 from ArrayList using an iterator.

B. Write a Java program to

a. create and traverse (or iterate) ArrayList using for-loop, iterator, and advance for-loop.

b. check if element(value) exists in ArrayList?

c. add element at particular index of ArrayList?

```java
import java.util.ArrayList;
import java.util.Iterator;

public class ArrayListIterator {

    // Part A: Method to check if a number is prime
    public static boolean isPrime(int number) {
        if (number <= 1) return false;
        for (int i = 2; i <= Math.sqrt(number); i++) {
            if (number % i == 0) return false;
        }
        return true;
    }

    public static void main(String[] args) {

        // ---------------- PART A ----------------
        System.out.println("Part A: Remove Prime Numbers from 1 to 25\n");

        ArrayList<Integer> numbers = new ArrayList<>();
        for (int i = 1; i <= 25; i++) {
            numbers.add(i);
        }

        // Remove prime numbers using Iterator
        Iterator<Integer> iterator = numbers.iterator();
        while (iterator.hasNext()) {
            int num = iterator.next();
            if (isPrime(num)) {
                iterator.remove();
            }
        }

        System.out.println("ArrayList after removing prime numbers: " + numbers);

        // ---------------- PART B ----------------
        System.out.println("\nPart B: ArrayList Operations\n");

        // a. Create ArrayList and traverse using:
        ArrayList<String> fruits = new ArrayList<>();
        fruits.add("Apple");
        fruits.add("Banana");
```

```
        fruits.add("Cherry");
        fruits.add("Date");

        // i. for-loop
        System.out.println("Traverse using for-loop:");
        for (int i = 0; i < fruits.size(); i++) {
            System.out.println(fruits.get(i));
        }

        // ii. Iterator
        System.out.println("\nTraverse using Iterator:");
        Iterator<String> fruitIterator = fruits.iterator();
        while (fruitIterator.hasNext()) {
            System.out.println(fruitIterator.next());
        }

        // iii. Enhanced for-loop
        System.out.println("\nTraverse using enhanced for-loop:");
        for (String fruit : fruits) {
            System.out.println(fruit);
        }

        // b. Check if element exists
        String searchFruit = "Banana";
        if (fruits.contains(searchFruit)) {
            System.out.println("\nElement '" + searchFruit + "' exists in the
ArrayList.");
        } else {
            System.out.println("\nElement '" + searchFruit + "' does not exist in
the ArrayList.");
        }

        // c. Add element at particular index
        fruits.add(2, "Mango"); // adding Mango at index 2
        System.out.println("\nArrayList after adding 'Mango' at index 2:");
        for (String fruit : fruits) {
            System.out.println(fruit);
        }
    }
}
```

37).

Write a Java program that handles various types of exceptions while performing different operations. The application should read data from a file specified by the user, handling potential `FileNotFoundException` and `IOException`. It should also allow the user to input values for arithmetic operations and handle division by zero using `ArithmeticException`. Additionally, implement exception handling for `InputMismatchException` when the user provides invalid input, `ArrayIndexOutOfBoundsException` for accessing invalid indices in arrays, and `NullPointerException` when performing operations on `null` values. The program should provide user-friendly error messages and ensure smooth execution even when exceptions occur.

```
import java.io.*;
import java.util.*;

public class ExceptionHandlingDemo {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // 1. File Reading with Exception Handling and Auto-Creation
        System.out.print("Enter the file name to read: ");
        String filename = scanner.nextLine();

        File file = new File(filename);

        if (!file.exists()) {
            try {
                if (file.createNewFile()) {
                    FileWriter writer = new FileWriter(file);
                    writer.write("Sample content: File created because it did not
exist.\n");
                    writer.close();
                    System.out.println("Note: File did not exist. A new file was
created with sample content.");
                } else {
                    System.out.println("Error: Unable to create file.");
                }
            } catch (IOException e) {
                System.out.println("Error: Failed to create the file due to an
I/O error.");
            }
        }
```

```
        // Reading the file
        try (BufferedReader reader = new BufferedReader(new
FileReader(filename))) {
            System.out.println("\nFile Content:");
            String line;
            while ((line = reader.readLine()) != null) {
                System.out.println(line);
            }
        } catch (IOException e) {
            System.out.println("Error: An I/O error occurred while reading the
file.");
        }

        // 2. Arithmetic Operation (Division)
        try {
            System.out.print("\nEnter numerator: ");
            int num = scanner.nextInt();

            System.out.print("Enter denominator: ");
            int denom = scanner.nextInt();

            int result = num / denom;
            System.out.println("Result: " + result);
        } catch (InputMismatchException e) {
            System.out.println("Error: Please enter valid integers.");
            scanner.nextLine(); // Clear the buffer
        } catch (ArithmeticException e) {
            System.out.println("Error: Division by zero is not allowed.");
        }

        // 3. Array Access
        int[] numbers = {10, 20, 30};
        try {
            System.out.print("\nEnter array index to access (0-2): ");
            int index = scanner.nextInt();
            System.out.println("Value at index " + index + ": " +
numbers[index]);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Error: Index out of bounds. Please enter a valid
index.");
        }

        // 4. Null Pointer Handling
        String text = null;
```

```
        try {

            System.out.println("Length of text: " + text.length());
        } catch (NullPointerException e) {
            System.out.println("Error: Cannot perform operations on a null
object.");
        }

        System.out.println("\nProgram completed successfully.");
        scanner.close();
    }
}
```

```
import java.io.*;
import java.util.*;

public class ExceptionHandlingDemo {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // 1. File Reading with Exception Handling
        System.out.print("Enter the file name to read: ");
        String filename = scanner.nextLine();

        try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {
            System.out.println("\nFile Content:");
            String line;
            while ((line = reader.readLine()) != null) {
                System.out.println(line);
            }
```

```
} catch (FileNotFoundException e) {
   System.out.println("Error: File not found. Please check the file name and path.");
} catch (IOException e) {
    System.out.println("Error: An I/O error occurred while reading the file.");
}

// 2. Arithmetic Operation (Division)
try {
    System.out.print("\nEnter numerator: ");
    int num = scanner.nextInt();

    System.out.print("Enter denominator: ");
    int denom = scanner.nextInt();

    int result = num / denom;
    System.out.println("Result: " + result);
} catch (InputMismatchException e) {
    System.out.println("Error: Please enter valid integers.");
    scanner.nextLine(); // Clear the buffer
} catch (ArithmeticException e) {
    System.out.println("Error: Division by zero is not allowed.");
}

// 3. Array Access
int[] numbers = {10, 20, 30};
try {
```

```
            System.out.print("\nEnter array index to access (0-2): ");
            int index = scanner.nextInt();
            System.out.println("Value at index " + index + ": " + numbers[index]);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Error: Index out of bounds. Please enter a valid index.");
        }

        // 4. Null Pointer Handling
        String text = null;
        try {
            System.out.println("\nLength of text: " + text.length());
        } catch (NullPointerException e) {
            System.out.println("Error: Cannot perform operations on a null object.");
        }

        System.out.println("\nProgram completed successfully.");
        scanner.close();
    }
}
```

38)
Create an abstract class Person with:

- Fields: name, age
- Constructor to initialize the fields
- Abstract method:
- Create a class Student that inherits from Person:
    - Additional fields: rollNumber, course

    - Override the displayDetails() method to print all student details
- Create another class Teacher that also extends Person:
    - Additional fields: employeeId, subject
    - Override the displayDetails() method to print all teacher details
- Demonstrate encapsulation by making all fields private and using getter and setter methods.
- In the main() method:
    - Create an array of Person references (use polymorphism).
    - Store both Student and Teacher objects.
    - Call the displayDetails() method for each object using a loop.

```
abstract class Person {
    private String name;
    private int age;

    public Person(String name, int age){
        this.name = name;
        this.age = age;
    }

    public String getName(){
        return name;
    }

    public void setName(String name){
        this.name = name;
    }

    public int getAge(){
        return age;
    }

    public void setAge(int age){
        this.age = age;
    }

    public abstract void displayDetails();
}
```

```java
class Student extends Person {
    private int rollNumber;
    private String course;

    public Student(String name, int age, int rollNumber, String course){
        super(name, age);
        this.rollNumber = rollNumber;
        this.course = course;
    }

    public int getRollNumber(){
        return rollNumber;
    }

    public void setRollNumber(int rollNumber){
        this.rollNumber = rollNumber;
    }

    public String getCourse(){
        return course;
    }

    public void setCourse(String course){
        this.course = course;
    }

    @Override
    public void displayDetails(){
        System.out.println("Student Details:");
        System.out.println("Name: " + getName());
        System.out.println("Age: " + getAge());
        System.out.println("Roll Number: " + rollNumber);
        System.out.println("Course: " + course);
        System.out.println();
    }
}

class Teacher extends Person {
    private int employeeId;
    private String subject;

    public Teacher(String name, int age, int employeeId, String subject){
        super(name, age);
        this.employeeId = employeeId;
```

```
        this.subject = subject;
    }

    public int getEmployeeId(){
        return employeeId;
    }

    public void setEmployeeId(int employeeId){
        this.employeeId = employeeId;
    }

    public String getSubject(){
        return subject;
    }

    public void setSubject(String subject){
        this.subject = subject;
    }

    @Override
    public void displayDetails(){
        System.out.println("Teacher Details:");
        System.out.println("Name: " + getName());
        System.out.println("Age: " + getAge());
        System.out.println("Employee ID: " + employeeId);
        System.out.println("Subject: " + subject);
        System.out.println();
    }
}

public class StudentTeacher {
    public static void main(String[] args) {
        // Polymorphism: using Person references
        Person[] people = new Person[2];

        people[0] = new Student("Alice", 20, 101, "Computer Science");
        people[1] = new Teacher("Dr. Bob", 45, 5001, "Mathematics");

        for (int i = 0; i < people.length; i++) {
            people[i].displayDetails(); // Dynamic method dispatch
        }
    }
}
```

39) A) Write a Java program to create a class called Employee with methods called work() and getSalary(). Create a subclass called HRManager that overrides the work() method and adds a new method called addEmployee().

B) Develop a JAVA program to create a class named shape. Create three sub classes namely: circle, triangle and square, each class has two member functions named draw () and erase (). Demonstrate polymorphism concepts by developing suitable methods, defining member data and main program.

```java
// Employee.java
class Employee {
    void work() {
        System.out.println("Employee is working.");
    }

    double getSalary() {
        return 30000.0; // base salary
    }
}

// HRManager.java
class HRManager extends Employee {
    // Overriding the work() method
    @Override
    void work() {
        System.out.println("HR Manager is recruiting new employees.");
    }

    // New method in HRManager
    void addEmployee() {
        System.out.println("New employee has been added to the system.");
    }
}

// Main.java
public class PartA {
    public static void main(String[] args) {
        HRManager hr = new HRManager();

        // Calls overridden work method
        hr.work();

        // Calls inherited getSalary method
        System.out.println("Salary: " + hr.getSalary());
```

```
        // Calls HR-specific method
        hr.addEmployee();
    }
}
```

```
// Shape.java
class Shape {
    void draw() {
        System.out.println("Drawing a shape.");
    }

    void erase() {
        System.out.println("Erasing a shape.");
    }
}

// Circle.java
class Circle extends Shape {
    @Override
    void draw() {
        System.out.println("Drawing a circle.");
    }

    @Override
    void erase() {
        System.out.println("Erasing a circle.");
    }
}

// Triangle.java
class Triangle extends Shape {
    @Override
    void draw() {
        System.out.println("Drawing a triangle.");
    }

    @Override
    void erase() {
        System.out.println("Erasing a triangle.");
    }
}

// Square.java
```

```java
class Square extends Shape {
    @Override
    void draw() {
        System.out.println("Drawing a square.");
    }

    @Override
    void erase() {
        System.out.println("Erasing a square.");
    }
}

// Main.java
public class PartB {
    public static void main(String[] args) {
        // Polymorphism: Using Shape reference to refer to subclass objects
        Shape s;

        s = new Circle();
        s.draw();
        s.erase();

        s = new Triangle();
        s.draw();
        s.erase();

        s = new Square();
        s.draw();
        s.erase();
    }
}
```

40) Develop a JAVA program to create an abstract class Shape with abstract methods calculateArea() and calculatePerimeter(). Create subclasses Circle and Triangle that extend the Shape class and implement the respective methods to calculate the area and perimeter of each shape.

```java
// Abstract class Shape
abstract class Shapee {
    // Abstract methods
    abstract double calculateArea();
    abstract double calculatePerimeter();
```

```
}

// Circle class extending Shape
class Circlee extends Shapee {
    double radius;

    // Constructor
    Circlee(double radius) {
        this.radius = radius;
    }

    // Implementing abstract methods
    @Override
    double calculateArea() {
        return Math.PI * radius * radius;
    }

    @Override
    double calculatePerimeter() {
        return 2 * Math.PI * radius;
    }
}

// Triangle class extending Shape
class Trianglee extends Shapee {
    double sideA, sideB, sideC;

    // Constructor
    Trianglee(double a, double b, double c) {
        this.sideA = a;
        this.sideB = b;
        this.sideC = c;
    }

    // Implementing abstract methods
    @Override
    double calculateArea() {
        double s = (sideA + sideB + sideC) / 2; // Semi-perimeter
        return Math.sqrt(s * (s - sideA) * (s - sideB) * (s - sideC)); // Heron's
formula
    }

    @Override
    double calculatePerimeter() {
        return sideA + sideB + sideC;
```

```
    }
}

// Main class
public class CircleTriangle {
    public static void main(String[] args) {
        // Create a Circle object
        Shapee circle = new Circlee(5); // Radius = 5
        System.out.println("Circle:");
        System.out.println("Area = " + circle.calculateArea());
        System.out.println("Perimeter = " + circle.calculatePerimeter());

        System.out.println();

        // Create a Triangle object
        Shapee triangle = new Trianglee(3, 4, 5); // Sides: 3, 4, 5
        System.out.println("Triangle:");
        System.out.println("Area = " + triangle.calculateArea());
        System.out.println("Perimeter = " + triangle.calculatePerimeter());
    }
}
```

41) A) Write a java program to Move all zeroes to end of array

Input: arr[] = {1, 2, 0, 4, 3, 0, 5, 0};

Output: arr[] = {1, 2, 4, 3, 5, 0, 0, 0};

```
public class MoveZeros {
    public static void moveZeroesToEnd(int[] arr) {
        int count = 0; // Index to keep track of non-zero elements

        // Step 1: Copy all non-zero elements to the front
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] != 0) {
                arr[count] = arr[i];
                count++;
            }
        }

        // Step 2: Fill remaining positions with 0
```

```
        while (count < arr.length) {
            arr[count] = 0;
            count++;
        }
    }

    public static void main(String[] args) {
        int[] arr = {1, 2, 0, 4, 3, 0, 5, 0};

        System.out.println("Before:");
        for (int num : arr) {
            System.out.print(num + " ");
        }

        moveZeroesToEnd(arr);

        System.out.println("\nAfter:");
        for (int num : arr) {
            System.out.print(num + " ");
        }
    }
}
```

B)Write a Java program to create an interface Sortable with a method sort() that sorts an array of integers in ascending order. Create two classes BubbleSort and SelectionSort that implement the Sortable interface and provide their own implementations of the sort() method.

```
// Define the Sortable interface
interface Sortable {
    void sort(int[] arr);  // Method to sort an array
}

// Implement BubbleSort class that implements Sortable interface
class BubbleSort implements Sortable {
    @Override
    public void sort(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            // Last i elements are already in place
            for (int j = 0; j < n - i - 1; j++) {
                // Swap if the element found is greater than the next element
```

```
                if (arr[j] > arr[j + 1]) {
                    // Swap arr[j] and arr[j + 1]
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
            }
        }
    }
}

// Implement SelectionSort class that implements Sortable interface
class SelectionSort implements Sortable {
    @Override
    public void sort(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            // Find the minimum element in unsorted array
            int minIndex = i;
            for (int j = i + 1; j < n; j++) {
                if (arr[j] < arr[minIndex]) {
                    minIndex = j;
                }
            }
            // Swap the found minimum element with the first element
            int temp = arr[minIndex];
            arr[minIndex] = arr[i];
            arr[i] = temp;
        }
    }
}

// Main class to test both sorting algorithms
public class Sort {
    public static void printArray(int[] arr) {
        for (int num : arr) {
            System.out.print(num + " ");
        }
        System.out.println();
    }

    public static void main(String[] args) {
        // Test array
        int[] arr = {64, 25, 12, 22, 11};
```

```
        System.out.println("Original Array:");
        printArray(arr);

        // Using BubbleSort
        Sortable bubbleSort = new BubbleSort();
        bubbleSort.sort(arr);
        System.out.println("Sorted Array using BubbleSort:");
        printArray(arr);

        // Reset the array to original for SelectionSort
        arr = new int[]{64, 25, 12, 22, 11};

        // Using SelectionSort
        Sortable selectionSort = new SelectionSort();
        selectionSort.sort(arr);
        System.out.println("Sorted Array using SelectionSort:");
        printArray(arr);
    }
}
```

42) A) Write a Java program to create a class called "Book" with instance variables title, author, and price. Implement a default constructor and two parameterized constructors:

One constructor takes the title and author as parameters.
The other constructor takes title, author, and price as parameters.
Print the values of the variables for each constructor.

```
// Class definition for Book
class Book {
    // Instance variables
    String title;
    String author;
    double price;

    // Default constructor
    public Book() {
        this.title = "Unknown Title";
        this.author = "Unknown Author";
        this.price = 0.0;
    }
```

```java
    // Constructor with title and author as parameters
    public Book(String title, String author) {
        this.title = title;
        this.author = author;
        this.price = 0.0;  // Default price if not provided
    }

    // Constructor with title, author, and price as parameters
    public Book(String title, String author, double price) {
        this.title = title;
        this.author = author;
        this.price = price;
    }

    // Method to print book details
    public void printBookDetails() {
        System.out.println("Title: " + this.title);
        System.out.println("Author: " + this.author);
        System.out.println("Price: " + this.price);
        System.out.println("----------------------------");
    }
}

// Main class to test the Book class
public class BookTest {
    public static void main(String[] args) {
        // Creating objects using different constructors

        // Using the default constructor
        Book book1 = new Book();
        System.out.println("Using default constructor:");
        book1.printBookDetails();

        // Using the constructor with title and author
        Book book2 = new Book("The Great Gatsby", "F. Scott Fitzgerald");
        System.out.println("Using constructor with title and author:");
        book2.printBookDetails();

        // Using the constructor with title, author, and price
        Book book3 = new Book("1984", "George Orwell", 12.99);
        System.out.println("Using constructor with title, author, and price:");
        book3.printBookDetails();
    }
}
```

B) Write a Java program to create a class called "TrafficLight" with attributes for color and duration, and methods to change the color and check for red or green.

```java
// TrafficLight class with basic attributes
class TrafficLight {
    private String color;  // Traffic light color
    private int duration;  // Duration in seconds

    // Constructor to initialize color and duration
    public TrafficLight(String color, int duration) {
        this.color = color;
        this.duration = duration;
    }

    // Method to change the color of the traffic light
    public void changeColor(String color) {
        this.color = color;
    }

    // Method to check if the light is red
    public boolean isRed() {
        return color.equals("Red");
    }

    // Method to check if the light is green
    public boolean isGreen() {
        return color.equals("Green");
    }

    // Method to display the current state
    public void displayStatus() {
        System.out.println(color + " light for " + duration + " seconds.");
    }
}

// Main class to test TrafficLight
public class TrafficLights {
    public static void main(String[] args) {
        // Creating a TrafficLight object with initial color and duration
        TrafficLight light = new TrafficLight("Red", 60);
```

```
        // Displaying the current light status
        light.displayStatus();

        // Change the color to green
        light.changeColor("Green");
        light.displayStatus();

        // Check the light status
        if (light.isRed()) {
            System.out.println("The light is Red.");
        } else if (light.isGreen()) {
            System.out.println("The light is Green.");
        }
    }
}
```