JAVA PROGRAMMING LABORATORY

1) Program to implement stack operations (push, pop & display).

```
import java.util.*;

public class StackOperations
{
   private int maxSize;                                    // Maximum size of the stack
   private int top;                                        // Index of the top element in the stack
   private int[] stackArray;                               // Array to store the stack elements

   // Constructor to initialize the stack
   public StackOperations(int size)
   {
      maxSize = size;
      stackArray = new int[maxSize];
      top = -1;                                            // Stack is initially empty, so top is -1
   }

   // Method to push an element onto the stack
   public void push(int value)
   {
      if (isFull())
      {
         System.out.println("Stack is full. Cannot push " + value);
      }
        else
      {
         top++;                                            // Increment the top pointer
         stackArray[top] = value;                          // Place the value on top of the stack
         System.out.println("Pushed " + value + " onto the stack.");
      }
   }

   // Method to pop an element from the stack
   public int pop()
   {
      if (isEmpty())
      {
         System.out.println("Stack is empty. Cannot pop.");
         return -1;                                        // Return a default value to indicate failure
      }
        else
      {
         int poppedValue = stackArray[top];                // Get the top element
         top--;                                            // Decrement the top pointer
         System.out.println("Popped " + poppedValue + " from the stack.");
         return poppedValue;                               // Return the popped value
      }
   }
```

```
    // Method to check if the stack is empty
    public boolean isEmpty()
    {
        return top == -1;
    }

    // Method to check if the stack is full
    public boolean isFull()
    {
        return top == maxSize - 1;
    }

    // Method to display the elements in the stack
    public void display()
    {
        if (isEmpty())
        {
            System.out.println("Stack is empty.");
        } else {
            System.out.print("Stack elements: ");
            for (int i = 0; i <= top; i++)
            {
                System.out.print(stackArray[i] + " ");
            }
            System.out.println();
        }
    }

    public static void main(String[] args)
    {
        int stackSize = 5;
        StackOperations stack = new StackOperations(stackSize);

        stack.push(1);
        stack.push(2);
        stack.push(3);
        stack.push(4);
        stack.push(5);

        stack.display();

        int poppedValue = stack.pop();
        System.out.println("Popped value: " + poppedValue);

        stack.display();
    }
}
```

OUTPUT:

```
Pushed 1 onto the stack.
Pushed 2 onto the stack.
```

```
Pushed 3 onto the stack.
Pushed 4 onto the stack.
Pushed 5 onto the stack.
Stack elements: 1 2 3 4 5
Popped 5 from the stack.
Popped value: 5
Stack elements: 1 2 3 4
```

2) Program to print all the solutions of a quadratic equations.

```
import java.util.Scanner;

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

      System.out.println("Quadratic Equation Solver");
      System.out.println("Enter the coefficients of the quadratic equation (a, b, c):");

      double a = scanner.nextDouble();
      double b = scanner.nextDouble();
      double c = scanner.nextDouble();

      // Calculate the discriminant (b^2 - 4ac)
      double discriminant = b * b - 4 * a * c;

      if (discriminant > 0)
      {
         // Two real solutions
         double root1 = (-b + Math.sqrt(discriminant)) / (2 * a);
         double root2 = (-b - Math.sqrt(discriminant)) / (2 * a);

         System.out.println("Two real solutions:");
         System.out.println("Root 1: " + root1);
         System.out.println("Root 2: " + root2);
      }
         else if (discriminant == 0)
      {
         // One real solution (a repeated root)
         double root = -b / (2 * a);

         System.out.println("One real solution (repeated root):");
         System.out.println("Root: " + root);
      }
         else
      {
         // Complex solutions
         double realPart = -b / (2 * a);
         double imaginaryPart = Math.sqrt(-discriminant) / (2 * a);
```

```
            System.out.println("Complex solutions:");
            System.out.println("Root 1: " + realPart + " + " + imaginaryPart + "i");
            System.out.println("Root 2: " + realPart + " - " + imaginaryPart + "i");
        }

        scanner.close();
    }
}
```

OUTPUT:

```
Enter the coefficients of the quadratic equation (a, b, c):
1 -3 2
Two real solutions:
Root 1: 2.0
Root 2: 1.0

Enter the coefficients of the quadratic equation (a, b, c):
1 -4 4
One real solution (repeated root):
Root: 2.0

Enter the coefficients of the quadratic equation (a, b, c):
1 2 5
Complex solutions:
Root 1: -1.0 + 2.0i
Root 2: -1.0 - 2.0i
```

3) Program to read students name, register number, marks and display the student details with total marks using single & multilevel inheritance.

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

class Student {
    String name;
    int registerNumber;

    Student(String name, int registerNumber) {
        this.name = name;
        this.registerNumber = registerNumber;
    }

    void displayStudentInfo() {
        System.out.println("Name: " + name);
        System.out.println("Register Number: " + registerNumber);
    }
}

class MarksStudent extends Student {
    int[] marks;
```

```
    MarksStudent(String name, int registerNumber, int[] marks) {
        super(name, registerNumber);
        this.marks = marks;
    }

    void displayTotalMarks() {
        int totalMarks = 0;
        for (int mark : marks) {
            totalMarks += mark;
        }
        System.out.println("Total Marks: " + totalMarks);
    }

    void displayStudentDetails() {
        displayStudentInfo();
        System.out.println("Marks in Each Subject:");
        for (int i = 0; i < marks.length; i++) {
            System.out.println("Subject " + (i + 1) + ": " + marks[i]);
        }
        displayTotalMarks();
    }
}

public class StudentDetails {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        List<MarksStudent> students = new ArrayList<>();

        int numStudents;
        System.out.print("Enter the number of students: ");
        numStudents = scanner.nextInt();

        for (int i = 0; i < numStudents; i++) {
            scanner.nextLine(); // Consume the newline character
            System.out.println("Enter details for Student " + (i + 1));
            System.out.print("Name: ");
            String name = scanner.nextLine();
            System.out.print("Register Number: ");
            int registerNumber = scanner.nextInt();
            int[] marks = new int[5]; // Assuming 5 subjects
            System.out.println("Enter marks for 5 subjects:");
            for (int j = 0; j < 5; j++) {
                System.out.print("Subject " + (j + 1) + ": ");
                marks[j] = scanner.nextInt();
            }

            MarksStudent student = new MarksStudent(name, registerNumber, marks);
            students.add(student);
        }
```

```
        System.out.println("Student Details:");
        for (MarksStudent student : students) {
            student.displayStudentDetails();
            System.out.println("================");
        }

        scanner.close();
    }
}
```

OUTPUT:

```
Enter the number of students: 2

Enter details for Student 1
Name: Alice
Register Number: 101
Enter marks for 5 subjects:
Subject 1: 88
Subject 2: 92
Subject 3: 78
Subject 4: 85
Subject 5: 90

Enter details for Student 2
Name: Bob
Register Number: 102
Enter marks for 5 subjects:
Subject 1: 76
Subject 2: 84
Subject 3: 92
Subject 4: 89
Subject 5: 78
```

Student Details:

```
Name: Alice
Register Number: 101
Marks in Each Subject:
Subject 1: 88
Subject 2: 92
Subject 3: 78
Subject 4: 85
Subject 5: 90
Total Marks: 433
================

Name: Bob
Register Number: 102
Marks in Each Subject:
Subject 1: 76
Subject 2: 84
Subject 3: 92
```

```
Subject 4: 89
Subject 5: 78
Total Marks: 419
===============
```

4.a) Program to implement queue operations.

```
import java.util.LinkedList;
import java.util.Queue;
import java.util.Scanner;

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

      // Create an empty queue using LinkedList
      Queue<Integer> queue = new LinkedList<>();

      // Prompt the user to enqueue (add) elements to the queue
      System.out.println("Enter elements to enqueue (enter a non-integer to stop):");
      while (scanner.hasNextInt())
      {
         int element = scanner.nextInt();
         queue.add(element);
      }

      // Display the elements in the queue
      System.out.println("Queue elements: " + queue);

      // Dequeue (remove) an element from the queue
      if (!queue.isEmpty())
      {
         int removedElement = queue.poll();
         System.out.println("Removed element: " + removedElement);
      } else
      {
         System.out.println("Queue is empty. Nothing to remove.");
      }

      // Peek at the front element of the queue without removing it
      int frontElement = queue.peek();
      System.out.println("Front element (peek): " + frontElement);

      // Check if the queue is empty
      boolean isEmpty = queue.isEmpty();
      System.out.println("Is the queue empty? " + isEmpty);

      // Display the elements in the queue after dequeue
      System.out.println("Queue elements after dequeue: " + queue);
```

```
        scanner.close();
    }
}
```

OUTPUT:

Scenario 1: Entering elements and removing an element

```
Enter elements to enqueue (enter a non-integer to stop):
1
2
3
4
5
Queue elements: [1, 2, 3, 4, 5]
Removed element: 1
Front element (peek): 2
Is the queue empty? false
Queue elements after dequeue: [2, 3, 4, 5]
```

Scenario 2: Entering non-integer input immediately

```
Enter elements to enqueue (enter a non-integer to stop):
A
Queue elements: []
Queue is empty. Nothing to remove.
Front element (peek): null
Is the queue empty? true
Queue elements after dequeue: []
```

Scenario 3: Empty queue

```
Enter elements to enqueue (enter a non-integer to stop):

Queue elements: []
Queue is empty. Nothing to remove.
Front element (peek): null
Is the queue empty? true
Queue elements after dequeue: []
```

4.b) Program to read two integers, perform division of two numbers, display the result with appropriate messages using the concept of exception handling.

```
import java.util.Scanner;

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

        try
        {
            System.out.println("Enter the numerator:");
            int numerator = scanner.nextInt();
```

```
            System.out.println("Enter the denominator:");
            int denominator = scanner.nextInt();

            // Perform division
            double result = divide(numerator, denominator);

            System.out.println("Result of division: " + result);
        }
        catch (ArithmeticException ae)
        {
            System.out.println("Error: Division by zero is not allowed.");
        } catch (java.util.InputMismatchException ime) {
            System.out.println("Error: Please enter valid integers.");
        }
        finally
        {
            // Close the scanner to prevent resource leak
            scanner.close();
        }
    }

    public static double divide(int numerator, int denominator)
    {
        if (denominator == 0)
        {
            throw new ArithmeticException("Division by zero is not allowed.");
        }
        return (double) numerator / denominator;
    }
}
```

OUTPUT:

```
Enter the numerator:
10
Enter the denominator:
2
Result of division: 5.0

Enter the numerator:
5
Enter the denominator:
0
Error: Division by zero is not allowed.

Enter the numerator:
abc
Error: Please enter valid integers.
```

5) Program to implement different string operations like strcpy, strlength, strcat, strcmp, str.charAt(i) using method overloading.

```
import java.util.Scanner;

public class StringOperations
  {
  // Method to copy one string to another (strcpy)
  public static String copyString(String source) {
     return source;
  }

  // Method to get the length of a string (strlength)
  public static int getLength(String str)
  {
     return str.length();
  }

  // Method to concatenate two strings (strcat)
  public static String concatenateStrings(String str1, String str2)
  {
     return str1 + str2;
  }

  // Method to compare two strings (strcmp)
  public static int compareStrings(String str1, String str2)
  {
     return str1.compareTo(str2);
  }

  // Method to get a character at a specific index (str.charAt(i))
  public static char getCharAtIndex(String str, int index)
     {
     if (index >= 0 && index < str.length())
     {
        return str.charAt(index);
     }
        else
     {
        throw new IndexOutOfBoundsException("Index is out of bounds.");
     }
  }

  public static void main(String[] args)
  {
     String str1 = "Hello";
     String str2 = "World";
     int index = 2;

     // Copy string
     String copiedStr = copyString(str1);
```

```
        System.out.println("Copied String: " + copiedStr);

        // Get string length
        int length = getLength(str1);
        System.out.println("Length of str1: " + length);

        // Concatenate strings
        String concatenatedStr = concatenateStrings(str1, str2);
        System.out.println("Concatenated String: " + concatenatedStr);

        // Compare strings
        int comparison = compareStrings(str1, str2);
        System.out.println("Comparison Result: " + comparison);

        // Get character at index
        try
        {
            char charAtIndex = getCharAtIndex(str1, index);
            System.out.println("Character at index " + index + ": " + charAtIndex);
        } catch (IndexOutOfBoundsException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

OUTPUT:

```
String str1 = "Programming";
String str2 = "Language";
int index = 4;

Copied String: Programming
Length of str1: 11
Concatenated String: ProgrammingLanguage
Comparison Result: -1
Character at index 4: a

String str1 = "apple";
String str2 = "banana";
int index = 3;

Copied String: apple
Length of str1: 5
Concatenated String: applebanana
Comparison Result: -1
Character at index 3: l
```

6) Program to multiply given two matrices of size m x n.

```
import java.io.*;

// Driver Class

class GFG
{
   // Function to print Matrix
   static void printMatrix(int M[][], int rowSize, int colSize)
  {
      for (int i = 0; i < rowSize; i++)
      {
         for (int j = 0; j < colSize; j++)
            System.out.print(M[i][j] + " ");
         System.out.println();
      }

   }

   // Function to multiply two matrices A[][] and B[][]

   static void multiplyMatrix(int row1, int col1, int A[][], int row2, int col2, int B[][])
   {
      int i, j, k;

      // Print the matrices A and B

      System.out.println("\nMatrix A:");
      printMatrix(A, row1, col1);
      System.out.println("\nMatrix B:");
      printMatrix(B, row2, col2);

      // Check if multiplication is Possible

      if (row2 != col1)
      {
         System.out.println("\nMultiplication Not Possible");
         return;
      }

      // Matrix to store the result
      // The product matrix will be of size row1 x col2

      int C[][] = new int[row1][col2];

      // Multiply the two matrices

      for (i = 0; i < row1; i++)
      {
         for (j = 0; j < col2; j++)
        {
            for (k = 0; k < row2; k++)
               C[i][j] += A[i][k] * B[k][j];
         }
      }
```

```
        // Print the result
        System.out.println("\nResultant Matrix:");
        printMatrix(C, row1, col2);
    }

    // Driver code
    public static void main(String[] args)
    {
        int row1 = 4, col1 = 3, row2 = 3, col2 = 4;

        int A[][] =
        {
            {1, 1, 1},
            {2, 2, 2},
            {3, 3, 3},
            {4, 4, 4}
        };

        int B[][] =
        {
            {1, 1, 1, 1},
            {2, 2, 2, 2},
            {3, 3, 3, 3}
        };


        multiplyMatrix(row1, col1, A, row2, col2, B);

    }

}
```

OUTPUT:

Matrix A:
1 1 1
2 2 2
3 3 3
4 4 4

Matrix B:
1 1 1 1
2 2 2 2
3 3 3 3

Resultant Matrix:
6 6 6 6
12 12 12 12
18 18 18 18
24 24 24 24

7.a) Program to count the frequency of words (using string tokenizer).

```
import java.util.StringTokenizer;
import java.util.HashMap;
import java.util.Map;

public class WordFrequencyCounter
{
   public static void main(String[] args)
   {
      String text = "This is a sample text. This text contains some words. " +
                    "We will count the frequency of each word in this text.";

      // Create a StringTokenizer to tokenize the text by spaces and punctuation

      StringTokenizer tokenizer = new StringTokenizer(text, " .");

      // Create a map to store word frequencies

      Map<String, Integer> wordFrequency = new HashMap<>();

      // Tokenize and count word frequencies

      while (tokenizer.hasMoreTokens())
      {
         String word = tokenizer.nextToken().toLowerCase(); // Convert to lowercase
         if (wordFrequency.containsKey(word))
         {
            // If the word is already in the map, increment its count
            int count = wordFrequency.get(word);
            wordFrequency.put(word, count + 1);
         }
          else
         {
            // If the word is not in the map, add it with a count of 1
            wordFrequency.put(word, 1);
         }
      }

      // Print the word frequencies

      System.out.println("Word Frequencies:");
      for (Map.Entry<String, Integer> entry : wordFrequency.entrySet()) {
         System.out.println(entry.getKey() + ": " + entry.getValue());
      }
   }
}
```

OUTPUT:

```
Word Frequencies:
in: 1
text: 3
is: 1
words: 2
contains: 1
will: 1
```

each: 1
a: 1
the: 1
sample: 1
we: 1
this: 2
word: 1
of: 1
frequency: 1
count: 1
some: 1

7.b) Program to read N values using an array and sort them using Bubble sort.

```
import java.io.*;

 class BUB
{
    // An optimized version of Bubble Sort
   static void bubbleSort(int arr[], int n)
   {
      int i, j, temp;
      boolean swapped;
      for (i = 0; i < n - 1; i++) {
         swapped = false;
         for (j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
               // Swap arr[j] and arr[j+1]
               temp = arr[j];
               arr[j] = arr[j + 1];
               arr[j + 1] = temp;
               swapped = true;
            }
         }
         // If no two elements were
         // swapped by inner loop, then break
         if (swapped == false)
            break;
      }
   }

   // Function to print an array

   static void printArray(int arr[], int size)
   {
      int i;
      for (i = 0; i < size; i++)
         System.out.print(arr[i] + " ");
      System.out.println();
   }

   // Driver program

   public static void main(String args[])
```

```
    {
        int arr[] = { 64, 34, 25, 12, 22, 11, 90 };
        int n = arr.length;
        bubbleSort(arr, n);
        System.out.println("Sorted array: ");
        printArray(arr, n);
    }
}
```

OUTPUT:

Sorted array:
11 12 22 25 34 64 90

8) Program to perform addition, subtraction, division, and multiplication of complex numbers (using method overloading).

```
class ComplexNumber
{
    double real;
    double imaginary;

    // Constructor to initialize complex numbers
    ComplexNumber(double real, double imaginary)
    {
        this.real = real;
        this.imaginary = imaginary;
    }

    // Method to add two complex numbers
    ComplexNumber add(ComplexNumber other)
    {
        return new ComplexNumber(this.real + other.real, this.imaginary + other.imaginary);
    }

    // Method to subtract two complex numbers

    ComplexNumber subtract(ComplexNumber other)
    {
        return new ComplexNumber(this.real - other.real, this.imaginary - other.imaginary);
    }

    // Method to multiply two complex numbers

    ComplexNumber multiply(ComplexNumber other)
    {
        double newReal = (this.real * other.real) - (this.imaginary * other.imaginary);
        double newImaginary = (this.real * other.imaginary) + (this.imaginary * other.real);
        return new ComplexNumber(newReal, newImaginary);
    }

    // Method to divide two complex numbers

    ComplexNumber divide(ComplexNumber other)
    {
        double denominator = (other.real * other.real) + (other.imaginary * other.imaginary);
```

```
        double newReal = ((this.real * other.real) + (this.imaginary * other.imaginary)) / denominator;
        double newImaginary = ((this.imaginary * other.real) - (this.real * other.imaginary)) / denominator;
        return new ComplexNumber(newReal, newImaginary);
    }

    // Method to display the complex number

    void display()
    {
        System.out.println(real + " + " + imaginary + "i");
    }

    public static void main(String[] args) {
        ComplexNumber num1 = new ComplexNumber(3.0, 4.0);
        ComplexNumber num2 = new ComplexNumber(1.0, 2.0);

        System.out.println("Complex Number 1: ");
        num1.display();

        System.out.println("Complex Number 2: ");
        num2.display();

        ComplexNumber sum = num1.add(num2);
        System.out.println("Addition Result: ");
        sum.display();

        ComplexNumber difference = num1.subtract(num2);
        System.out.println("Subtraction Result: ");
        difference.display();

        ComplexNumber product = num1.multiply(num2);
        System.out.println("Multiplication Result: ");
        product.display();

        ComplexNumber quotient = num1.divide(num2);
        System.out.println("Division Result: ");
        quotient.display();
    }
}
```

OUTPUT:

Complex Number 1:
3.0 + 4.0i

Complex Number 2:
1.0 + 2.0i

Addition Result:
4.0 + 6.0i

Subtraction Result:
2.0 + 2.0i

Multiplication Result:
-5.0 + 10.0i

Division Result:
2.2 + -0.4i

9) Program to find the values of unknowns in the polynomial degree of n.

```
import java.util.*;

class POLY

{
        // Function to calculate
       // factorial of N
       static int fact(int n)
       {
         // Base Case
         if (n == 1 || n == 0)
         return 1;
        // Otherwise, recursively
       // calculate the factorial
       else
        return n * fact(n - 1);
       }

// Function to find the value of
// P(n + r) for polynomial P(X)

static int findValue(int n, int r, int a)
       {
       // Stores the value of k
       int k = (a - 1) / fact(n);
       // Store the required answer
       int answer = k;
       // Iterate in the range [1, N] and
        // multiply (n + r - i) with answer
       for(int i = 1; i < n + 1; i++)
       answer = answer * (n + r - i);
        // Add the constant value C as 1
       answer = answer + 1;
        // Return the result
        return answer;
       }

// Driver Code

public static void main(String args[])
       {
       int N = 1;
       int A = 2;
       int R = 3;
       System.out.print(findValue(N, R, A));
       }
```

```
}
```

OUTPUT:

14

Explanation:

Using the formula:
P(n + r) = k * (n + r) * (n + r - 1) * ... * (n + 1) + 1,
where k is a constant calculated as (a - 1) / n! and n, r, and a are provided as inputs.
N (n) = 1
R (r) = 3
A (a) = 2
First, the program calculates the constant **k**:

- k = (a - 1) / fact(n) = (2 - 1) / fact(1) = 1 / 1 = 1

Then, it calculates the value of the polynomial function using the formula:

- P(1 + 3) = 1 * (1 + 3) * (1 + 3 - 1) + 1 = 1 * 4 * 3 + 1 = 13 + 1 = **14**

10) Program to create threads a,b and c for three different tasks using thread class with a thread for main method.

```
class TaskA extends Thread
{
   public void run()
  {
      for (int i = 1; i <= 5; i++)
     {
        System.out.println("Task A: " + i);
        try
        {
           Thread.sleep(1000);
        } catch (InterruptedException e)
        {
           e.printStackTrace();
        }
     }
   }
}

class TaskB extends Thread
{
   public void run()
  {
      for (int i = 1; i <= 5; i++)
     {
        System.out.println("Task B: " + i);
        try
        {
           Thread.sleep(1000);
        } catch (InterruptedException e)
        {
```

```
            e.printStackTrace();
        }
    }
  }
}
class TaskC extends Thread
{
   public void run()
  {
      for (int i = 1; i <= 5; i++)
     {
        System.out.println("Task C: " + i);
        try
        {
           Thread.sleep(1000);
        } catch (InterruptedException e)
        {
           e.printStackTrace();
        }
     }
  }
}
public class MultiThreadDemo
{
   public static void main(String[] args)
  {
      TaskA threadA = new TaskA();
      TaskB threadB = new TaskB();
      TaskC threadC = new TaskC();
      // Start the threads
      threadA.start();
      threadB.start();
      threadC.start();
      // Main thread
      for (int i = 1; i <= 5; i++)
     {
        System.out.println("Main Thread: " + i);
        try
        {
           Thread.sleep(1000);
        }
        catch (InterruptedException e)
        {
           e.printStackTrace();
        }
     }
      // Wait for threads A, B, and C to finish
      try {
        threadA.join();
        threadB.join();
```

```
            threadC.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("All threads have finished their tasks.");
    }
}
```

OUTPUT:

```
Task A: 1
Task B: 1
Main Thread: 1
Task C: 1
Main Thread: 2
Task B: 2
Task A: 2
Main Thread: 3
Task C: 2
Main Thread: 4
Task B: 3
Task A: 3
Main Thread: 5
Task C: 3
Task B: 4
Task A: 4
Task C: 4
Task B: 5
Task A: 5
Task C: 5
All threads have finished their tasks.
```

11) Program to overload the function search() and search an integer key value and key value of type double using binary search.

```
public class BinarySearch

{

   // Binary search for integers
   public static int search(int[] arr, int key)
   {
      int left = 0;
      int right = arr.length - 1;

      while (left <= right)
      {
         int mid = left + (right - left) / 2;

         if (arr[mid] == key)
         {
            return mid;
         } else if (arr[mid] < key)
         {
```

```
            left = mid + 1;
        }
            else
        {
            right = mid - 1;
        }
    }
    return -1;                                          // Key not found
}

// Binary search for doubles
public static int search(double[] arr, double key)
{
    int left = 0;
    int right = arr.length - 1;
    while (left <= right)
    {
        int mid = left + (right - left) / 2;
        if (arr[mid] == key)
        {
            return mid;
        }
            else if (arr[mid] < key)
        {
            left = mid + 1;
        }
        else
        {
            right = mid - 1;
        }
    }
    return -1;                                          // Key not found
}

public static void main(String[] args)
{
    int[] intArray = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    double[] doubleArray = {1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7, 8.8, 9.9, 10.0};

    int intKey = 8;
    double doubleKey = 7.7;

    int intResult = search(intArray, intKey);
    int doubleResult = search(doubleArray, doubleKey);

    if (intResult != -1)
    {
        System.out.println("Integer Key " + intKey + " found at index " + intResult);
    } else
    {
        System.out.println("Integer Key " + intKey + " not found.");
```

```
        }

        if (doubleResult != -1)
        {
            System.out.println("Double Key " + doubleKey + " found at index " + doubleResult);
        }
            else
        {
            System.out.println("Double Key " + doubleKey + " not found.");
        }
    }
}
```

OUTPUT:

1. Integer Key 8 found at index 7
   Double Key 7.7 found at index 6
2. Integer Key 5 found at index 4
   Double Key 5.5 found at index 4

12) Applet program that automatically display the text with font style, font type using getParameter(), getCodeBase(), and getDocumentBase().

```
import java.applet.Applet;
import java.awt.Font;
import java.awt.Graphics;

public class TextDisplayApplet extends Applet {

    private String text;
    private String fontName;
    private int fontStyle;
    private int fontSize;

    public void init() {

        // Get parameters from HTML

        text = getParameter("text");
        fontName = getParameter("fontname");
        fontStyle = Integer.parseInt(getParameter("fontstyle"));
        fontSize = Integer.parseInt(getParameter("fontsize"));
    }

    public void paint(Graphics g) {

        // Set font based on parameters

        Font font = new Font(fontName, fontStyle, fontSize);
        g.setFont(font);

        // Display the text

        g.drawString(text, 20, 50);
    }
```

```
}
```

HTML PART:

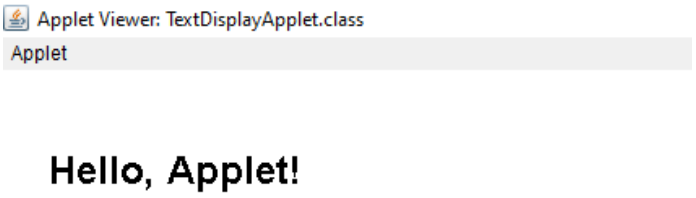
```
<!DOCTYPE html>
<html>
<head>
   <title>Text Display Applet</title>
</head>
<body>
   <applet code="TextDisplayApplet.class" width="400" height="100">
      <param name="text" value="Hello, Applet!"/>
      <param name="fontname" value="Arial"/>
      <param name="fontstyle" value="1"/> <!-- 0 for plain, 1 for bold, 2 for italic -->
      <param name="fontsize" value="24"/>
   </applet>
</body>
</html>
```

OUTPUT:

```
C:\Users\Dell>cd Desktop

C:\Users\Dell\Desktop>javac TextDisplayApplet.java

C:\Users\Dell\Desktop>appletviewer TextDisplayApplet.html
```



13) Create two classes 'Teacher' and 'Student' Perform interfacing of classes with appropriate attributes.

// Create an interface for common attributes between Teacher and Student

```
interface Person {

   String getName();
   int getAge();
}
```

// Define the Teacher class implementing the Person interface

```
class Teacher implements Person {

   private String name;
   private int age;
```

```
    private String subject;

    public Teacher(String name, int age, String subject) {

        this.name = name;
        this.age = age;
        this.subject = subject;
    }

    public String getName() {

        return name;
    }

    public int getAge() {

        return age;

    }

    public String getSubject() {

        return subject;

    }

    public void teach() {

        System.out.println(name + " is teaching " + subject);

    }

}
// Define the Student class implementing the Person interface
class Student implements Person {

    private String name;
    private int age;
    private String course;

    public Student(String name, int age, String course) {

        this.name = name;
        this.age = age;
        this.course = course;
    }

    public String getName() {

        return name;

    }

    public int getAge() {

        return age;

    }

    public String getCourse() {

        return course;
```

```
    }
    public void study() {
        System.out.println(name + " is studying " + course);
    }
}
public class Main {
    public static void main(String[] args) {
        Teacher teacher = new Teacher("Mr. Smith", 35, "Math");
        Student student = new Student("Alice", 20, "Computer Science");
        // Using common attributes through the Person interface
        System.out.println("Teacher: " + teacher.getName() + ", Age: " + teacher.getAge());
        System.out.println("Student: " + student.getName() + ", Age: " + student.getAge());
        // Calling specific methods
        teacher.teach();
        student.study();
    }
}
```

OUTPUT:

```
Teacher: Mr. Smith, Age: 35
Student: Alice, Age: 20
Mr. Smith is teaching Math
Alice is studying Computer Science
```

14) Develop an applet that receives an integer in one text field and computes its factorial value and returns it in another text field, when button named “Computes” is clicked.

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class FactorialCalculatorApplet extends Applet implements ActionListener {
    TextField inputField, resultField;
    Button computeButton;

    public void init() {
        inputField = new TextField(10);
        resultField = new TextField(10);
        computeButton = new Button("Compute");
```

```
        Label inputLabel = new Label("Enter an integer:");
        Label resultLabel = new Label("Factorial:");

        add(inputLabel);
        add(inputField);
        add(computeButton);
        add(resultLabel);
        add(resultField);

        computeButton.addActionListener(this);

    }

    public void actionPerformed(ActionEvent event) {

        if (event.getSource() == computeButton) {
            try {
                int input = Integer.parseInt(inputField.getText());
                long factorial = calculateFactorial(input);
                resultField.setText(Long.toString(factorial));
            } catch (NumberFormatException e) {
                resultField.setText("Invalid input");
            }
        }
    }

    private long calculateFactorial(int n) {

        if (n < 0) {
            return -1; // Return -1 for invalid input
        } else if (n == 0 || n == 1) {
            return 1;
        } else {
            long factorial = 1;
            for (int i = 2; i <= n; i++) {
                factorial *= i;
            }
            return factorial;
        }
    }
}
```

HTML PART:

```
<!DOCTYPE html>
<html>
<head>
    <title>Text Display Applet</title>
</head>
<body>
    <applet code="FactorialCalculatorApplet.class" width="400" height="100">
        <param name="text" value="Hello, Applet!"/>
        <param name="fontname" value="Arial"/>
        <param name="fontstyle" value="1"/> <!-- 0 for plain, 1 for bold, 2 for italic -->
        <param name="fontsize" value="24"/>
```

```
   </applet>
</body>
</html>
```

OUTPUT:

```
C:\Users\Dell\Desktop>cd ..

C:\Users\Dell>cd Desktop

C:\Users\joshua\Desktop>javac FactorialCalculatorApplet.java

C:\Users\joshua\Desktop>appletviewer TextDisplayApplet.html
```