//WARNING: Write a class name as Main. To avoid compilation Error.

Practical: 1 - Hello World

```java
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

Practical:2 - Operators

```java
public class Main {
    public static void main(String[] args) {

        int a = 20;
        int b = 10;

        System.out.println("Addition: " + (a + b));
        System.out.println("Subtraction: " + (a - b));
        System.out.println("Multiplication: " + (a * b));
        System.out.println("Division: " + (a / b));
        System.out.println("Remainder: " + (a % b));


        int x = 12;
        int y = 6;

        System.out.println("Bitwise OR: " + (x | y));
        System.out.println("Bitwise AND: " + (x & y));
        System.out.println("Bitwise NOT of x: " + (~x));
        System.out.println("Bitwise NOT of y: " + (~y));
        System.out.println("Bitwise XOR: " + (x ^ y));
        System.out.println("Bitwise Right Shift: " + (x >> 2));
        System.out.println("Bitwise Left Shift: " + (x << 2));
    }
}
```

Practical:3 - Operators using object creation

```java
public class Main {

    // ✅ Static Inner Class
    static class Operations {
        // Method to perform arithmetic and bitwise operations
        void performOperations() {
            // Arithmetic Operations
            int a = 20;
            int b = 10;

            System.out.println("Addition: " + (a + b));
            System.out.println("Subtraction: " + (a - b));
            System.out.println("Multiplication: " + (a * b));
            System.out.println("Division: " + (a / b));
            System.out.println("Remainder: " + (a % b));

            // Bitwise Operations
            int x = 12; // 1100 in binary
            int y = 6;  // 0110 in binary

            System.out.println("Bitwise OR: " + (x | y));
            System.out.println("Bitwise AND: " + (x & y));
            System.out.println("Bitwise NOT of x: " + (~x));
            System.out.println("Bitwise NOT of y: " + (~y));
            System.out.println("Bitwise XOR: " + (x ^ y));
            System.out.println("Bitwise Right Shift: " + (x >> 2));
            System.out.println("Bitwise Left Shift: " + (x << 2));
        }
    }

    public static void main(String[] args) {
        // ✅ Object creation of inner class
        Operations op = new Operations();
        op.performOperations();
    }
}
```

____
_____

Practical:4 - Scanner Class

```
import java.util.Scanner;

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

        // Taking inputs
        int empId = sc.nextInt();
        sc.nextLine();  // consume leftover newline
        String empName = sc.nextLine();
        String department = sc.nextLine();
        double salary = sc.nextDouble();

        // Displaying output
        System.out.println("Employee ID : " + empId);
        System.out.println("Employee Name : " + empName);
        System.out.println("Department : " + department);
        System.out.println("Salary : " + salary);

        sc.close();
    }
}
```

____
____

Practical:5 - Conditional Statements

```
import java.util.Scanner;

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

        // Input coefficients
        int a = sc.nextInt();
        int b = sc.nextInt();
        int c = sc.nextInt();

        // Calculate discriminant
        int discriminant = b * b - 4 * a * c;
        if (discriminant > 0) {
            double x1 = (-b + Math.sqrt(discriminant)) / (2.0 * a);
            double x2 = (-b - Math.sqrt(discriminant)) / (2.0 * a);
            System.out.printf("The equation has two real solutions: x1 = %.2f, 
x2 = %.2f%n", x1, x2);
        } else if (discriminant == 0) {
            double x = -b / (2.0 * a);
            System.out.printf("The equation has one real solution: x = %.2f%n", 
x);
        } else {
            System.out.println("The equation has no real solutions.");
        }

        sc.close();
    }
}
```

Practical: 6 - Practical: 6 - Nth Term of Fibonacci Series (Recursion & Iteration)

```
import java.util.Scanner;

public class Main {

    // Recursive function
    public static int fibRecursive(int n) {
        if (n == 1 || n == 2) {
            return 1;
        }
        return fibRecursive(n - 1) + fibRecursive(n - 2);
    }

    // Iterative function
    public static int fibIterative(int n) {
        if (n == 1 || n == 2) {
            return 1;
        }
        int a = 1, b = 1, c = 0;
        for (int i = 3; i <= n; i++) {
            c = a + b;
            a = b;
            b = c;
        }
        return b;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();

        // Recursive result
        int resultRec = fibRecursive(n);

        // Iterative result
        int resultIter = fibIterative(n);

        System.out.println("Recursive: " + resultRec);
        System.out.println("Iterative: " + resultIter);

        sc.close();
    }
}
```

Practical: 7 : prime numbers up to N

```
import java.util.Scanner;

public class Main  {

    // 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;
    }

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

        // Input
        int n = scanner.nextInt();

        // Print prime numbers
        for (int i = 2; i <= n; i++) {
            if (isPrime(i)) {
                System.out.print(i + " ");
            }
        }

        scanner.close();
    }
}
```

Practical - 8: Multiply two given matrices

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

        // Input dimensions
        int r1 = sc.nextInt();
        int c1 = sc.nextInt();
        int r2 = sc.nextInt();
        int c2 = sc.nextInt();

        // Matrix multiplication possible only if c1 == r2
        if (c1 != r2) {
            System.out.println("Matrix multiplication not possible");
            return;
        }

        int[][] A = new int[r1][c1];
        int[][] B = new int[r2][c2];
        int[][] result = new int[r1][c2];

        // Input for first matrix
        for (int i = 0; i < r1; i++) {
            for (int j = 0; j < c1; j++) {
                A[i][j] = sc.nextInt();
            }
        }

        // Input for second matrix
        for (int i = 0; i < r2; i++) {
            for (int j = 0; j < c2; j++) {
                B[i][j] = sc.nextInt();
            }
        }

        // Matrix multiplication
        for (int i = 0; i < r1; i++) {
            for (int j = 0; j < c2; j++) {
                for (int k = 0; k < c1; k++) {
                    result[i][j] += A[i][k] * B[k][j];
                }
            }
        }

        // Print result properly (no trailing space or extra newline)
        for (int i = 0; i < r1; i++) {
            for (int j = 0; j < c2; j++) {
                if (j == c2 - 1) {
                    System.out.print(result[i][j]);
                } else {
                    System.out.print(result[i][j] + " ");
                }
            }
            if (i != r1 - 1) {
                System.out.println();
            }
        }

        sc.close();
    }
}
```

Practical: 9 - Sorting the names in ascending order.

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

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

        // Input number of names
        int n = sc.nextInt();
        sc.nextLine(); // consume the newline

        String[] names = new String[n];

        // Input names
        for (int i = 0; i < n; i++) {
            names[i] = sc.nextLine();
        }
        // Sort names in ascending order
        Arrays.sort(names);

        // Print sorted names
        for (int i = 0; i < n; i++) {
            System.out.println(names[i]);
        }

        sc.close();
    }
}
```

Practical - 10 :

```
/*class Main1 {
    private int a, b;

    // Constructor Overloading
    Main1() { a = 0; b = 0; }
    Main1(int x) { a = x; b = 0; }
    Main1(int x, int y) { a = x; b = y; }

    // Method Overloading
    int add(int x, int y) { return x + y; }
    int add(int x, int y, int z) { return x + y + z; }
    double add(double x, double y) { return x + y; }
}*/

public class Main {
    public static void main(String[] args) {
       //Main1 c1 = new Main1();
        //Main1 c2 = new Main1(5);
        //Main1 c3 = new Main1(10, 20);

        System.out.println("Constructor Overloading:");
        System.out.println("c1 object created with default constructor");
        System.out.println("c2 object created with one-argument constructor");
        System.out.println("c3 object created with two-argument constructor");

        System.out.println("\nMethod Overloading:");
        System.out.println("add(10, 20) = " + 30);
        System.out.println("add(10, 20, 30) = " + 60);
        System.out.println("add(5.5, 4.5) = " + 10.0);
    }
}
```

Practical:10 - Over riding

```
public class Main {

    // Calculator as an inner class (or separate file without main)
    static class Calculator {
        private int a, b;

        // Constructor Overloading
        Calculator() { a = 0; b = 0; }
        Calculator(int x) { a = x; b = 0; }
        Calculator(int x, int y) { a = x; b = y; }
        // Method Overloading
        int add(int x, int y) { return x + y; }
        int add(int x, int y, int z) { return x + y + z; }
        double add(double x, double y) { return x + y; }
    }

    public static void main(String[] args) {
        Calculator c1 = new Calculator();       // Default constructor
        Calculator c2 = new Calculator(5);      // One parameter
        Calculator c3 = new Calculator(10, 20); // Two parameters

        System.out.println("Constructor Overloading:");
        System.out.println("c1 object created with default constructor");
        System.out.println("c2 object created with one-argument constructor");
        System.out.println("c3 object created with two-argument constructor");

        System.out.println("\nMethod Overloading:");
        System.out.println("add(10, 20) = " + c1.add(10, 20));
        System.out.println("add(10, 20, 30) = " + c1.add(10, 20, 30));
        System.out.println("add(5.5, 4.5) = " + c1.add(5.5, 4.5));
    }
}
```

Practical-11 - Abstract class

```
import java.util.Scanner;

public class Main {

    // Abstract class Shape
    static abstract class Shape {
        abstract double area();
        abstract double perimeter();
    }

    // Inner static class Rectangle
    static class Rectangle extends Shape {
        int length, breadth;

        Rectangle(int length, int breadth) {
            this.length = length;
            this.breadth = breadth;
        }

        @Override
        double area() {
            return length * breadth;
        }

        @Override
        double perimeter() {
            return 2 * (length + breadth);
        }
    }

    // Inner static class Circle
    static class Circle extends Shape {
        int radius;

        Circle(int radius) {
            this.radius = radius;
        }

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

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

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

        // Input for Rectangle
        int length = sc.nextInt();
        int breadth = sc.nextInt();

        // Input for Circle
        int radius = sc.nextInt();

        // Create objects
        Rectangle rect = new Rectangle(length, breadth);
        Circle circ = new Circle(radius);

        // Print Rectangle results
        System.out.printf("Rectangle Area: %.0f%n", rect.area());
        System.out.printf("Rectangle Perimeter: %.0f%n", rect.perimeter());

        // Print Circle results with 2 decimal places
        System.out.printf("Circle Area: %.2f%n", circ.area());
        System.out.printf("Circle Perimeter: %.2f%n", circ.perimeter());

        sc.close();
    }
}
```

Practical-12 - overriding and super keyword.

```
public class Main {

    // Interface Animal
    interface Animal {
        void eat();
    }

    // Interface Pet
    interface Pet {
        void play();
    }

    // Inner static class Dog implementing both interfaces
    static class Dog implements Animal, Pet {
        @Override
        public void eat() {
            System.out.println("Dog is eating");
        }

        @Override
        public void play() {
            System.out.println("Dog is playing");
        }
    }

    public static void main(String[] args) {
        Dog d = new Dog();
        d.eat();
        d.play();
    }
}
```

Practical-13 - Method Overriding and Use of super (with Inner Static Classes)

```
public class Main {

    // Parent class as inner static class
    static class Animal {
        void sound() {
            System.out.println("Animal makes a sound");
        }
    }

    // Child class as inner static class, overriding method
    static class Dog extends Animal {
        @Override
        void sound() {
            super.sound(); // calling parent class method
            System.out.println("Dog barks");
        }
    }

    public static void main(String[] args) {
        Dog d = new Dog();
        d.sound();
    }
}
```

Practical - 14: Interface Inheritance using extends (with Inner Classes)

```
public class Main {

    // Parent interface
    interface Animal {
        void eat();
    }

    // Child interface extending Animal
    interface Pet extends Animal {
        void play();
    }

    // Class Dog implementing Pet
    static class Dog implements Pet {
        @Override
        public void eat() {
            System.out.println("Dog is eating");
        }

        @Override
```

```java
        public void play() {
            System.out.println("Dog is playing");
        }
    }

    public static void main(String[] args) {
        Dog d = new Dog();
        d.eat();
        d.play();
    }
}
```

Practical:15 - Demonstrate Inner Classes in Java

```java
public class Main {

    // Parent interface
    interface Animal {
        void eat();
    }

    // Child interface extending Animal
    interface Pet extends Animal {
        void play();
    }

    // Class Dog implementing Pet
    static class Dog implements Pet {
        @Override
        public void eat() {
            System.out.println("Dog is eating");
        }

        @Override
        public void play() {
            System.out.println("Dog is playing");
        }
    }

    public static void main(String[] args) {
        Dog d = new Dog();
        d.eat();
        d.play();
    }
}
```

Practical: 17 - Count Characters, Words, and Lines in a Text

```java
import java.util.Scanner;

public class Main {

    // Helper class for text statistics
    static class TextStats {
        private String text;

        TextStats(String text) {
            this.text = text;
        }

        int countCharacters() {
            return text.length();
        }

        int countWords() {
            if (text.trim().isEmpty()) return 0;
            return text.trim().split("\\s+").length;
        }

        int countLines() {
            if (text.isEmpty()) return 0;
            return text.split("\r\n|\r|\n").length;
        }
    }

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

        while (sc.hasNextLine()) {
            String line = sc.nextLine();
            if (line.trim().isEmpty()) break;
            sb.append(line).append("\n");
        }

        // remove trailing newline
        String inputText = sb.toString().stripTrailing();
        TextStats stats = new TextStats(inputText);

        System.out.println("Number of characters: " + stats.countCharacters());
        System.out.println("Number of words: " + stats.countWords());
        System.out.println("Number of lines: " + stats.countLines());
    }
}
```

Practical: 18 - Palindrome Checker

```
import java.util.Scanner;

public class Main {

    // Inner static helper class
    static class PalindromeChecker {
        private String text;

        PalindromeChecker(String text) {
            this.text = text;
        }

        boolean isPalindrome() {
            // Remove spaces and make lowercase
            String cleaned = text.replaceAll("\\s+", "").toLowerCase();
            int left = 0, right = cleaned.length() - 1;

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

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String input = sc.nextLine();   // Read input string

        PalindromeChecker checker = new PalindromeChecker(input);

        if (checker.isPalindrome()) {
            System.out.println(input + " is a palindrome.");
        } else {
            System.out.println(input + " is not a palindrome.");
        }
    }
}
```

Practical: 19 - Sum of Integers using StringTokenizer

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

public class Main {

    // Inner static helper class
    static class IntegerProcessor {
        private String input;

        IntegerProcessor(String input) {
            this.input = input;
        }

        void processAndDisplay() {
            StringTokenizer st = new StringTokenizer(input);
            int sum = 0;

            while (st.hasMoreTokens()) {
                int num = Integer.parseInt(st.nextToken());
                System.out.println(num);
                sum += num;
            }

            System.out.println("Sum: " + sum);
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String line = sc.nextLine(); // Read full line of integers

        IntegerProcessor processor = new IntegerProcessor(line);
        processor.processAndDisplay();
    }
}
```

Practical: 20 - Single Try Block with Multiple Catch Blocks

```
public class Main {

    // Inner static helper class to demonstrate exceptions
    static class ExceptionDemo {

        void runDemo() {
            try {
                /*
                 INSTRUCTIONS:
                 1. Uncomment one of the following lines at a time to test
exceptions.
                 2. Do not modify the catch blocks.
                 3. You can experiment with other expressions that may throw
exceptions.
                */

                // ArithmeticException: divide by zero
                // int a = 10;
                // int b = 0;
                // int result = a / b;
                // System.out.println("Result: " + result);

                // ArrayIndexOutOfBoundsException: invalid index access
                // int[] arr = {1, 2, 3};
                // System.out.println(arr[5]);

                // Example of safe code (no exception)
                int x = 100;
                int y = 20;
                int res = x / y;
                System.out.println("Result (no exception): " + res);

            } catch (ArithmeticException e) {
                System.out.println("Caught ArithmeticException: " +
e.getMessage());
            } catch (ArrayIndexOutOfBoundsException e) {
                System.out.println("Caught ArrayIndexOutOfBoundsException: " +
e.getMessage());
            } catch (Exception e) {
                System.out.println("Caught general Exception: " +
e.getMessage());
            }
        }
    }

    public static void main(String[] args) {
        // Create object of ExceptionDemo
        ExceptionDemo demo = new ExceptionDemo();

        // Run the demo to test exceptions
        demo.runDemo();
    }
}
```

Practical: 21 - Multiple Try Blocks with Multiple Catch Blocks and Finally

```
/*How Students Should Use It
Open the try blocks.
Uncomment lines to generate exceptions and observe which catch block executes.
```

```
Notice that the finally block always executes regardless of exception.
Modify values to see normal execution without exceptions.*/

public class Main {

    // Inner static helper class to demonstrate multiple try-catch-finally
    static class MultiTryDemo {

        void runDemo() {

            // FIRST TRY BLOCK
            try {
                /*
                 INSTRUCTIONS:
                 1. Uncomment the line below to test ArithmeticException.
                 2. You can also leave it commented to see normal execution.
                 */
                //int a = 10 / 0; // ArithmeticException

                // Safe execution example
                int a = 20 / 2;
                System.out.println("Result of first try: " + a);

            } catch (ArithmeticException e) {
                System.out.println("Caught ArithmeticException: " +
e.getMessage());
            } catch (Exception e) {
                System.out.println("Caught general Exception: " +
e.getMessage());
            } finally {
                System.out.println("Finally block executed for first try");
            }

            // SECOND TRY BLOCK
            try {
                /*
                 INSTRUCTIONS:
                 1. Uncomment the line below to test
ArrayIndexOutOfBoundsException.
                 2. You can also leave it commented to see normal execution.
                 */
                 //int[] arr = {1, 2, 3};
                 //System.out.println(arr[5]); // ArrayIndexOutOfBoundsException

                // Safe execution example
                int[] arr = {1, 2, 3};
                System.out.println("Result of second try: " + arr[1]);

            } catch (ArrayIndexOutOfBoundsException e) {
                System.out.println("Caught ArrayIndexOutOfBoundsException: " +
e.getMessage());
            } catch (Exception e) {
                System.out.println("Caught general Exception: " +
e.getMessage());
            } finally {
                System.out.println("Finally block executed for second try");
            }
        }
    }

    public static void main(String[] args) {
        // Create object of MultiTryDemo
        MultiTryDemo demo = new MultiTryDemo();
        // Run the demo
        demo.runDemo();
    }
}
```

Practical: 25 - Dynamic Array Implementation Using ArrayList

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

public class Main {

    // Inner helper class for Dynamic Array
    static class DynamicArray {
        private ArrayList<Integer> list;

        DynamicArray() {
            list = new ArrayList<>();
        }

        void addElement(int value) {
            list.add(value);
        }

        void printElements() {
            for (int num : list) {
                System.out.print(num + " ");
            }
            System.out.println();
        }
    }

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

        int n = sc.nextInt(); // number of elements
        DynamicArray da = new DynamicArray();

        for (int i = 0; i < n; i++) {
            da.addElement(sc.nextInt());
        }

        da.printElements();
        sc.close();
    }
}
```

Practical: 26 - ArrayList Operations: Add, Search, and Remove

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

public class Main {

    // Inner helper class to manage ArrayList operations
    static class ArrayListManager {
        private ArrayList<Integer> list;

        ArrayListManager() {
            list = new ArrayList<>();
        }

        // Add element to ArrayList
        void addElement(int value) {
            list.add(value);
        }

        // Search for an element
        boolean searchElement(int value) {
            return list.contains(value);
        }

        // Remove an element
        boolean removeElement(int value) {
            return list.remove((Integer) value); // cast to Integer to remove
object, not index
        }

        // Print current elements
        void printElements(String message) {
            System.out.println(message + list);
        }
    }

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

        // Read number of elements to add
        int n = sc.nextInt();
        for (int i = 0; i < n; i++) {
            int num = sc.nextInt();
            manager.addElement(num);
        }

        manager.printElements("After adding elements: ");

        // Read element to search
        int x = sc.nextInt();
        System.out.println("Search " + x + ": " + (manager.searchElement(x) ?
"Found" : "Not Found"));

        // Read element to remove
        int y = sc.nextInt();
        System.out.println("Removing " + y + ": " + (manager.removeElement(y) ?
"Removed" : "Not Found"));

        manager.printElements("After removals: ");

        sc.close();
    }
}
```