

## Day 1 and 2: Manisha Assignment

### Task 1: Data Types/Variables

**Write a program that declares two integer variables, swaps their values without using a third variable, and prints the result.**

#### Simple Explanation of the Code

##### 1. Class Declaration:

- The program is defined within the class `SwapValues`.

##### 2. Main Method:

- The `main` method is the entry point of the program where execution begins.

##### 3. Variable Initialization:

- Two integer variables, `a` and `b`, are declared and initialized with values 5 and 10, respectively.

##### 4. Printing Original Values:

- The original values of `a` and `b` are printed to the console using `System.out.println`.

##### 5. Swapping Logic:

- Step 1: `a = a + b;`
  - The value of `a` is updated to the sum of `a` and `b` (15 in this case).
- \*\*Step 2: `b = a - b;`
  - The value of `b` is updated to `a` minus `b`, which gives the original value of `a` (5 in this case).
- Step 3: `a = a - b;`
  - The value of `a` is updated to `a` minus the new value of `b`, which gives the original value of `b` (10 in this case).

##### 6. Printing Swapped Values:

- The swapped values of `a` and `b` are printed to the console using `System.out.println`.

#### Output Here as follows

The screenshot shows an IDE interface with a code editor containing Java code. The code defines a class named SwapValues with a main method. It initializes two variables, a and b, to 5 and 10 respectively. It then swaps their values using a temporary variable c. Finally, it prints the swapped values. The output window shows the original values (a=5, b=10) and the swapped values (a=10, b=5).

```
public class SwapValues {
    public static void main(String[] args) {
        int a = 5;
        int b = 10;
        int c;
        // Swap values without using a third variable
        a = a + b;
        c = a - b;
        a = a - c;
        b = c;
        // Print swapped values
        System.out.println("a = " + a);
        System.out.println("b = " + b);
    }
}
```

```
a = 5
b = 10
a = 10
b = 5
```

## Task 2: Operators

Create a program that simulates a simple calculator using command-line arguments to perform and print the result of addition, subtraction, multiplication, and division..

### 1. Reading User Input:

- The program uses the `Scanner` class to read input from the user.
- It asks the user to enter two numbers and an operator (like +, -, \*, /).

### 2. Performing Operations:

- The program uses a `switch` statement to choose the correct operation based on the operator provided by the user.
- It has methods for addition, subtraction, multiplication, and division.
- If the operator is not valid, it prints an error message.

### 3. Handling Division by Zero:

- Before dividing, the program checks if the second number is zero.
- If it is zero, it prints a custom error message to avoid dividing by zero.

### 4. Displaying the Result:

- The program performs the chosen operation and prints the result to the console.

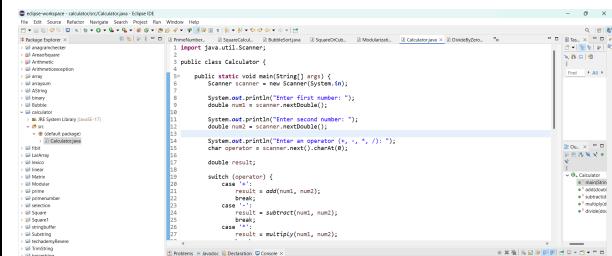
### 5. Example Interaction:

- The user enters two numbers (e.g., 10 and 5) and an operator (e.g., \*).
- The program calculates the result (e.g., 50) and prints it.

The program is designed to handle basic arithmetic operations and provide a clear result or error message based on the user's input.

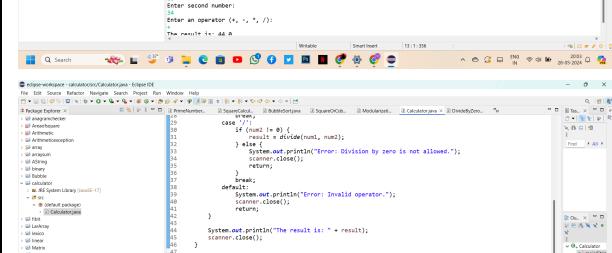
The program is designed to handle basic arithmetic operations and provide a clear result or error message based on the user's input.

Here Output as follows 



```
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Enter first number: ");
    double num1 = scanner.nextDouble();
    System.out.print("Enter second number: ");
    double num2 = scanner.nextDouble();
    System.out.print("Enter an operator (+, -, *, /): ");
    char operator = scanner.next().charAt(0);

    double result;
    switch(operator) {
        case '+':
            result = add(num1, num2);
            break;
        case '-':
            result = subtract(num1, num2);
            break;
        case '*':
            result = multiply(num1, num2);
            break;
        case '/':
            result = divide(num1, num2);
            break;
        default:
            System.out.println("Error: Invalid operator.");
            scanner.close();
    }
    System.out.println("The result is: " + result);
}
```

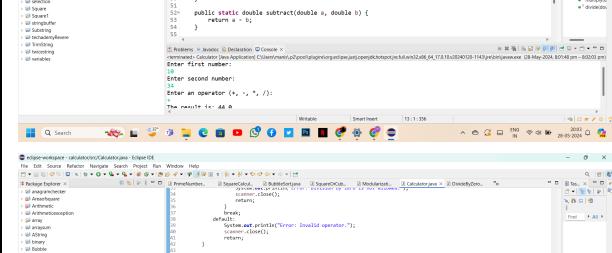


```
public static double add(double a, double b) {
    return a + b;
}

public static double subtract(double a, double b) {
    return a - b;
}

public static double multiply(double a, double b) {
    return a * b;
}

public static double divide(double a, double b) {
    if (b == 0) {
        System.out.println("Error: Division by zero is not allowed.");
        scanner.close();
    } else {
        return a / b;
    }
}
```



```
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Enter first number: ");
    double num1 = scanner.nextDouble();
    System.out.print("Enter second number: ");
    double num2 = scanner.nextDouble();
    System.out.print("Enter an operator (+, -, *, /): ");
    char operator = scanner.next().charAt(0);

    double result;
    switch(operator) {
        case '+':
            result = add(num1, num2);
            break;
        case '-':
            result = subtract(num1, num2);
            break;
        case '*':
            result = multiply(num1, num2);
            break;
        case '/':
            result = divide(num1, num2);
            break;
        default:
            System.out.println("Error: Invalid operator.");
            scanner.close();
    }
    System.out.println("The result is: " + result);
}
```



```
public static double add(double a, double b) {
    return a + b;
}

public static double subtract(double a, double b) {
    return a - b;
}

public static double multiply(double a, double b) {
    return a * b;
}

public static double divide(double a, double b) {
    if (b == 0) {
        System.out.println("Error: Division by zero is not allowed.");
        scanner.close();
    } else {
        return a / b;
    }
}
```

## Task 3: Control Flow

**Write a Java program that reads an integer and prints whether it is a prime number using a for loop and if statements.**

## 1. Import Statement:

- The program imports the `java.util.Scanner` class to read input from the user.

## **2. Main Class and Method:**

- The `PrimeChecker` class contains the `main` method which is the entry point of the program.

### 3. Reading Input:

- A `Scanner` object is created to read input from the user.
    - The program prompts the user to enter an integer with `System.out.print("Enter an integer: ")`.
    - The entered integer is stored in the variable `number`.

## 4. Prime Number Check:

- The `isPrime` method is called with the input number as an argument.
  - The `isPrime` method checks if the number is a prime number:
    - If the number is less than or equal to 1, it returns `false`.
    - It iterates from `2` to the square root of the number. If any number in this range divides the input number without a remainder, it is not a prime number and returns `false`.
    - If no divisors are found, it returns `true`.

## **5. Printing the Result:**

- Based on the result from the `isPrime` method, the program prints whether the number is a prime number or not using `System.out.println`.

## Output:

- If the user enters `235`, the output is:

**Enter an integer: 235**  
**235 is not a prime number.**

This program effectively checks and prints whether a given integer is a prime number by iterating up to the square root of the number and checking for divisors.

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows the project structure with files like `primechecker.java`, `primechecker.class`, and `primechecker.html`.
- Java Editor:** Displays the Java code for the `PrimeChecker` class.

```
public class PrimeChecker {  
    public static void main(String[] args) {  
        System.out.println("Section and write either an integer:");  
        int number = Integer.parseInt(args[0]);  
        boolean classmate = true;  
        for (int i = 2; i < number; i++) {  
            if (number % i == 0) {  
                classmate = false;  
            }  
        }  
        System.out.println(number + " is a prime number.");  
        System.out.println(number + " is not a prime number.");  
    }  
}  
  
public static boolean isPrime(int num) {  
    if (num < 2) {  
        return false;  
    }  
    for (int i = 2; i < Math.sqrt(num); i++) {  
        if (num % i == 0) {  
            return false;  
        }  
    }  
    return true;  
}
```
- Output View:** Shows the output of the program: `235 is not a prime number.`
- Help View:** Shows the help documentation for the `isPrime` method.

## Task 4: Constructors

Implement a Matrix class that has a constructor which initializes the dimensions of a matrix and a method to fill the matrix with values.

'Matrix' object that can be filled with numbers and then printed out. Here's a straightforward breakdown:

### 1. Creating the Matrix:

- The 'Matrix' class is defined to hold a 2D array called 'matrix' to store numbers.

### 2. Initializing the Matrix:

- When a 'Matrix' object is created using 'Matrix(int rows, int cols)', it initializes an empty matrix of size 'rows' by 'cols'.

### 3. Filling the Matrix:

- The 'fill(int[] values)' method fills the matrix with numbers provided in the 'values' array. It's assumed that the array is large enough to fill the entire matrix.

### 4. Printing the Matrix:

- The 'printMatrix()' method simply prints out the numbers in the matrix in a readable format.

### 5. Example Usage in 'main()':

- In the 'main' method, a 'Matrix' object is created with 2 rows and 3 columns.
  - An array of values '{1, 2, 3, 4, 5, 6}' is filled into the matrix.
  -

The filled matrix is then printed out.

The key idea is that this code lets you create a matrix, put numbers into it, and then see those numbers displayed neatly.

```
public class Matrix {
    private int[][] matrix;
    public Matrix(int rows, int cols) {
        matrix = new int[rows][cols];
    }
    public void fill(int[] values) {
        int index = 0;
        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix[i].length; j++) {
                matrix[i][j] = values[index++];
            }
        }
    }
    public void printMatrix() {
        for (int[] row : matrix) {
            for (int value : row) {
                System.out.print(value + " ");
            }
            System.out.println();
        }
    }
}
```

```
public static void main(String[] args) {
    Matrix m = new Matrix(2, 3);
    int[] values = {1, 2, 3, 4, 5, 6};
    m.fill(values);
    System.out.println("Matrix:");
    m.printMatrix();
}
```

```
Matrix:
1 2 3
4 5 6
```

## Task 5: Inheritance

Create a Shape class with a method area() and extend it with Circle and Rectangle classes overriding the area() method appropriately.

### 1. Shape Class:

- 'Shape' is an abstract class with an abstract method 'area()'. This means that any class inheriting from 'Shape' must provide an implementation for the 'area' method.

### 2. Circle Class:

- 'Circle' extends 'Shape'.
- It has a constructor to initialize its radius.
- The 'area' method calculates the area using the formula ' $\pi * \text{radius}^2$ '.

### 3. Rectangle Class:

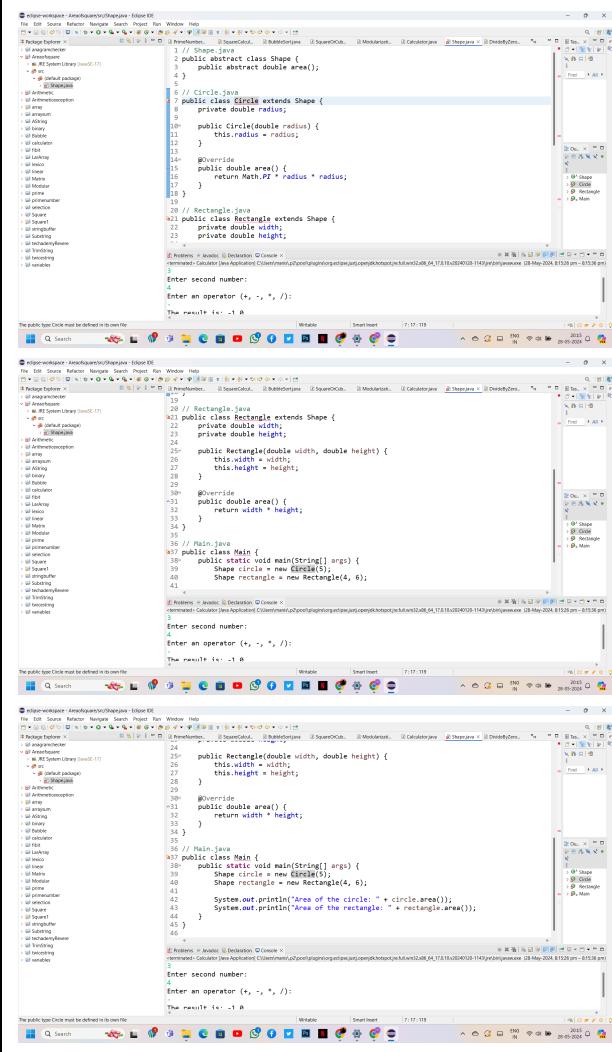
- 'Rectangle' extends 'Shape'.
- It has a constructor to initialize its width and height.
- The 'area' method calculates the area using the formula 'width \* height'.

### 4. Main Class:

- Creates instances of 'Circle' and 'Rectangle'.
- Prints the area of each shape using the overridden 'area' method.

This corrected and simplified code should work correctly in an Eclipse environment and provide the expected output.

### Here Output as Follows



The screenshots show the Eclipse IDE interface with three open files:

- Shape.java**: An abstract class with an abstract method `area()`.
- Circle.java**: Extends `Shape` and overrides `area()` to calculate  $\pi * \text{radius}^2$ .
- Rectangle.java**: Extends `Shape` and overrides `area()` to calculate `width * height`.
- Main.java**: The main application that creates `Circle` and `Rectangle` objects and prints their areas.

The output window shows the expected results for each operation (+, -, \*, /) entered.

## Task 6: Packages/Classpath

Create a package com.math.operations and include classes for various arithmetic operations. Demonstrate how to compile and run these using the classpath.

### 1. Create the Project:

- Open Eclipse and create a new Java project named 'MathOperations'.

### 2. Create the Package:

- Right-click on the 'src' folder, select 'New > Package', and name it 'com.math.operations'.

### 3. Add the Classes:

- Right-click on the 'com.math.operations' package, select 'New > Class', and create the 'Addition', 'Subtraction', 'Multiplication', and 'Division' classes with the respective code provided above.

- Create the 'Main' class directly under the 'src' folder (it doesn't need to be in the package).

### 4. Run the Main Class:

- Right-click on the 'Main' class, select 'Run As > Java Application'.

When you run the 'Main' class, you should see the following output:

Addition: 15.0

Subtraction: 5.0

Multiplication: 50.0

Division: 2.0

### Here output as follows

Output window - ArithmeticOperations [Java Application]

```
1 import com.math.operations.Addition;
2 import com.math.operations.Division;
3 import com.math.operations.Multiplication;
4 import com.math.operations.Subtraction;
5
6 public class Main {
7     public static void main(String[] args) {
8         Addition addition = new Addition();
9         Subtraction subtraction = new Subtraction();
10        Multiplication multiplication = new Multiplication();
11        Division division = new Division();
12
13        // Perform operations
14        double a = 10;
15        double b = 5;
16        double c = a + b;
17
18        System.out.println("Addition: " + addition.add(a, b));
19        System.out.println("Subtraction: " + subtraction.subtract(a, b));
20        System.out.println("Multiplication: " + multiplication.multiply(a, b));
21        System.out.println("Division: " + division.divide(a, b));
22    }
23 }
```

Java Problems - Java Application [Java Application]

Addition: 15.0  
Subtraction: 5.0  
Multiplication: 50.0  
Division: 2.0

Output window - ArithmeticOperations [Java Application]

```
1 package com.math.operations;
2
3 public class Addition {
4     public double add(double a, double b) {
5         return a + b;
6     }
7 }
8
```

Java Problems - Java Application [Java Application]

Addition: 15.0  
Subtraction: 5.0  
Multiplication: 50.0  
Division: 2.0

Output window - ArithmeticOperations [Java Application]

```
1 package com.math.operations;
2
3 public class Subtraction {
4     public double subtract(double a, double b) {
5         return a - b;
6     }
7 }
8
```

Java Problems - Java Application [Java Application]

Addition: 15.0  
Subtraction: 5.0  
Multiplication: 50.0  
Division: 2.0

Output window - ArithmeticOperations [Java Application]

```
1 package com.math.operations;
2
3 public class Multiplication {
4     public double multiply(double a, double b) {
5         return a * b;
6     }
7 }
8
```

Java Problems - Java Application [Java Application]

Addition: 15.0  
Subtraction: 5.0  
Multiplication: 50.0  
Division: 2.0

Output window - ArithmeticOperations [Java Application]

```
1 package com.math.operations;
2
3 public class Division {
4     public double divide(double a, double b) {
5         return a / b;
6     }
7 }
8
```

Java Problems - Java Application [Java Application]

Addition: 15.0  
Subtraction: 5.0  
Multiplication: 50.0  
Division: 2.0

## Task 7: Basic Exception Handling

**Write a program that attempts to divide by zero, catches the `ArithmaticException`, and provides a custom error message.**

Java program that attempts to divide by zero, catches the `ArithmaticException`, and provides a custom error message:

### 1. Main Method:

- In the `main` method, we initialize two integer variables: `numerator` with a value of 10 and `denominator` with a value of 0.

- We then call the `divideByZero` method with these values to attempt division by zero.

### 2. divideByZero Method:

- The `divideByZero` method takes two integer parameters: `numerator` and `denominator`.

- It tries to divide `numerator` by `denominator` and returns the result.

### 3. Try-Catch Block:

- We use a `try-catch` block to handle any exceptions that may occur during the division operation.

- If an `ArithmaticException` (division by zero) occurs, the `catch` block is executed.

### 4. Custom Error Message:

- Inside the `catch` block, we use `System.err.println` to print a custom error message stating that division by zero is not allowed.

### 5. Output:

- When you run the program, it attempts to divide by zero and encounters an `ArithmaticException`.

- The `catch` block catches this exception and prints the custom error message to the standard error stream (`System.err`).

So, the program demonstrates how to handle division by zero by catching the `ArithmaticException` and providing a meaningful error message to the user.

### Output as Follows

The screenshot shows an IDE interface with a code editor containing Java code. The code defines a class `DividebyZeroExample` with a `main` method that initializes `numerator` to 10 and `denominator` to 0. It then calls the `divideByZero` method with these values. A try-catch block is used to catch an `ArithmaticException`. If caught, it prints a custom error message to `System.err`. If no exception occurs, it prints the result of the division. Below the code editor is a terminal window showing the output of running the program, which includes the custom error message "Error: Division by zero is not allowed."

```
1 public class DividebyZeroExample {
2     public static void main(String[] args) {
3         int numerator = 10;
4         int denominator = 0;
5         try {
6             int result = divideByZero(numerator, denominator);
7             System.out.println("Result: " + result);
8         } catch (ArithmaticException e) {
9             System.err.println("Error: Division by zero is not allowed.");
10        }
11    }
12
13    public static int divideByZero(int numerator, int denominator) {
14        return numerator / denominator;
15    }
16 }
17
```

```
2 Problems = Javac 0 Declarat 0 Errors > 0
12:11:45 20 May 2014, 2:15 PM
Error: Division by zero is not allowed.
```