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.

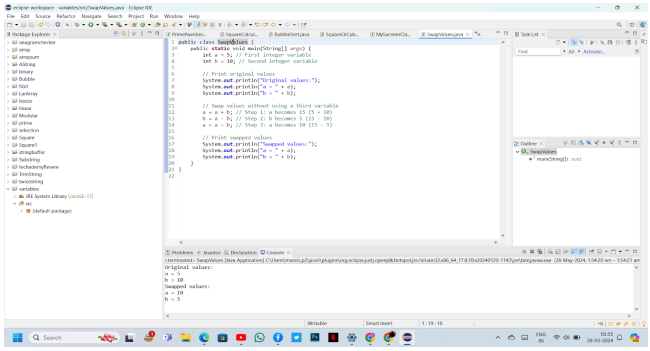
3. Variable Initialization:
- Two integer variables, `a` and `b`, are declared and initialized with values 5 and 10, respectively.

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



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.

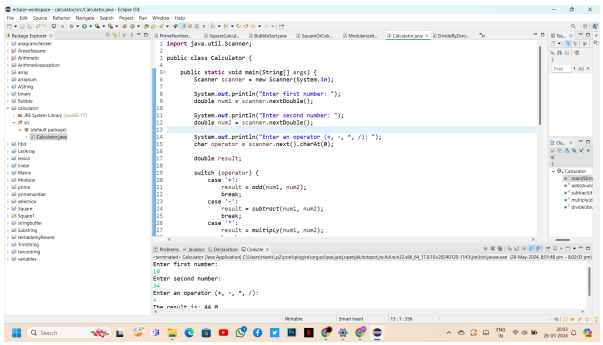
5. Example Interaction:

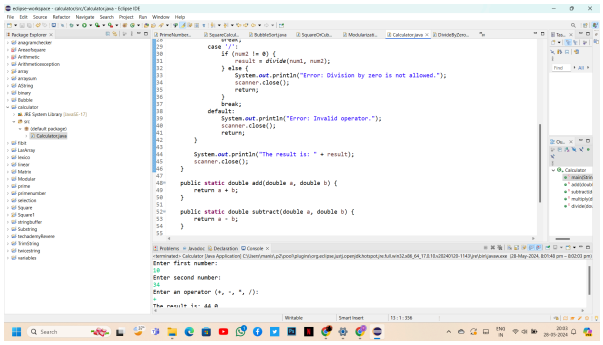
- The user enters two numbers (e.g., 10 and 5) and an operator (e.g., *).
- The program calculates the result (e.g., 50) and prints it.

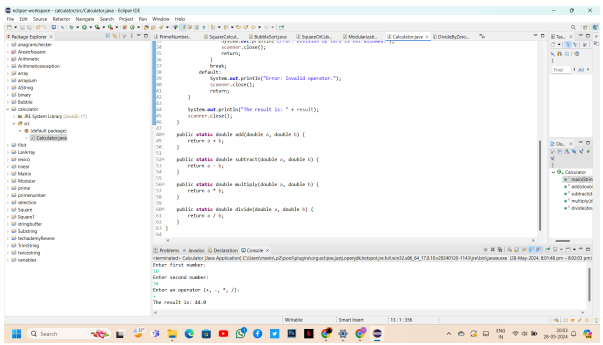
The program is designed to handle basic arithmetic operations and provide a clear result or error message based on the user's input.

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 👍







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`.

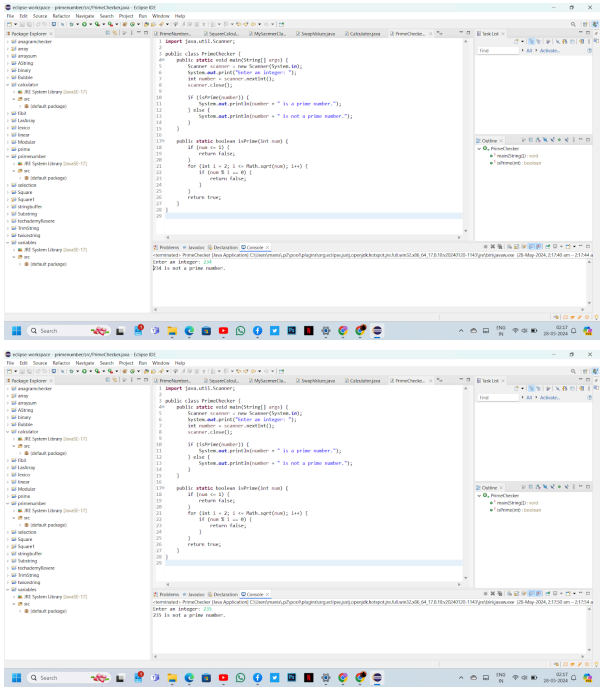
4. Prime Number Check:
- The `isPrime` method is called with the input number as an argument.
- The `isPrime` method checks if the number is a prime number:
  - If the number is less than or equal to 1, it returns `false`.
  - It iterates from `2` to the square root of the number. If any number in this 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:

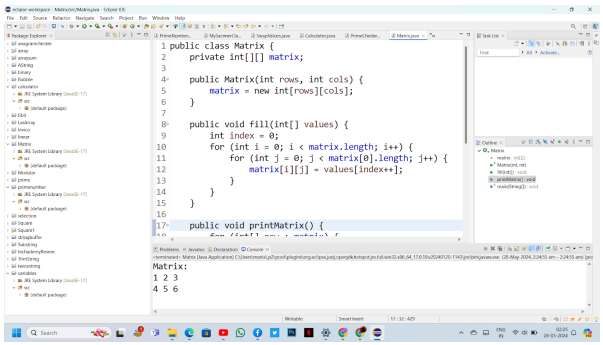
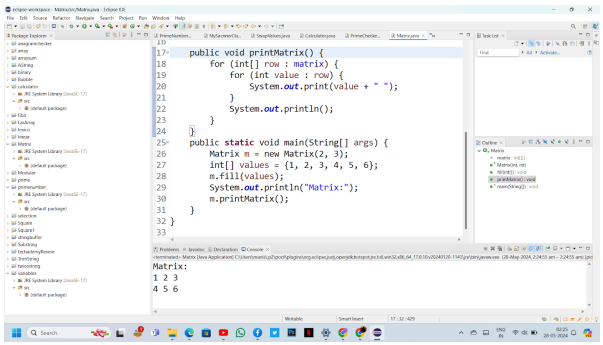
Enter an integer: 235
235 is not a prime number.

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.



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.**  

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.

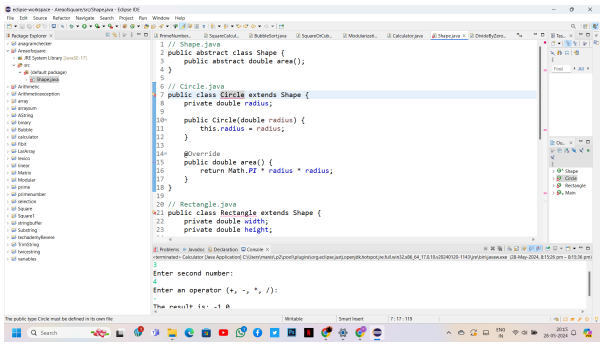
2. Circle Class:
   - `Circle` extends `Shape`.
   - It has a constructor to initialize its radius.
   - The `area` method calculates the area using the formula `π * radius^2`.

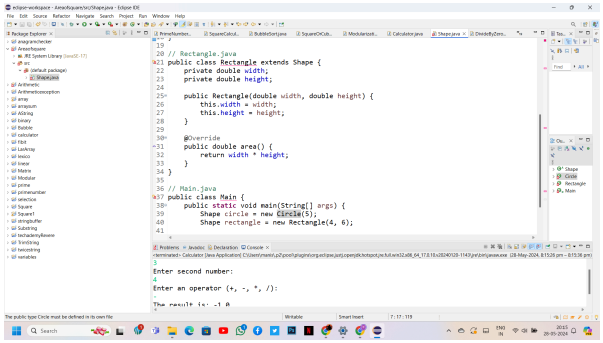
3. Rectangle Class:
   - `Rectangle` extends `Shape`.
   - It has a constructor to initialize its width and height.
   - The `area` method calculates the area using the formula `width * height`.

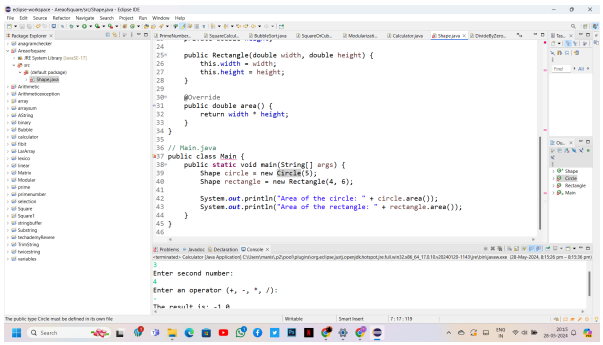
4. Main Class:
   - Creates instances of `Circle` and `Rectangle`.
   - Prints the area of each shape using the overridden `area` method.

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

Here Output as Follows 👍







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).

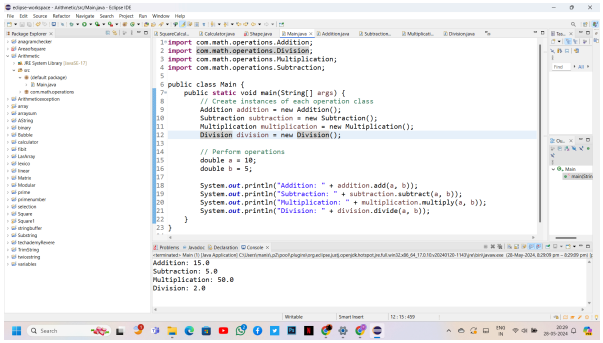
4. Run the Main Class:

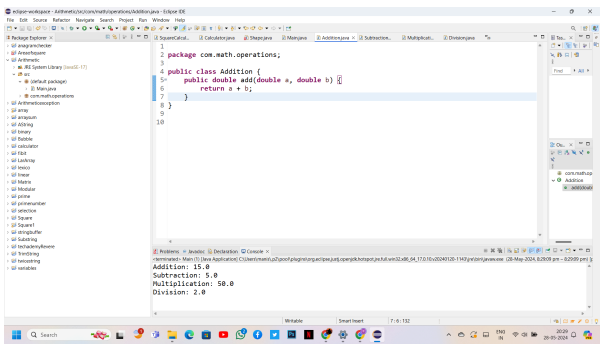
- Right-click on the `Main` class, select `Run As > Java Application`.

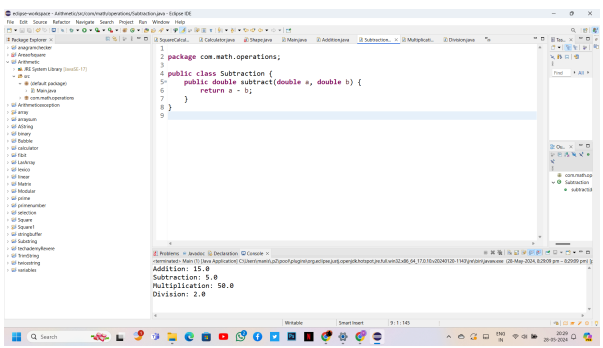
When you run the `Main` class, you should see the following output:

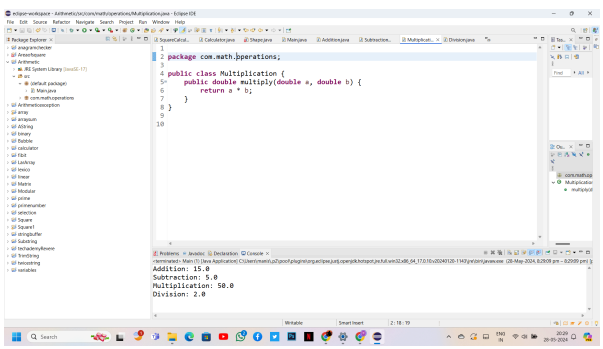
Addition: 15.0
Subtraction: 5.0
Multiplication: 50.0
Division: 2.0

Here output as follows 👍









Task 7: Basic Exception Handling

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

Java program that attempts to divide by zero, catches the `ArithmeticException`, 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 `ArithmeticException` (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 `ArithmeticException`

- 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 `ArithmeticException` and providing a meaningful error message to the user.

Output as Follows