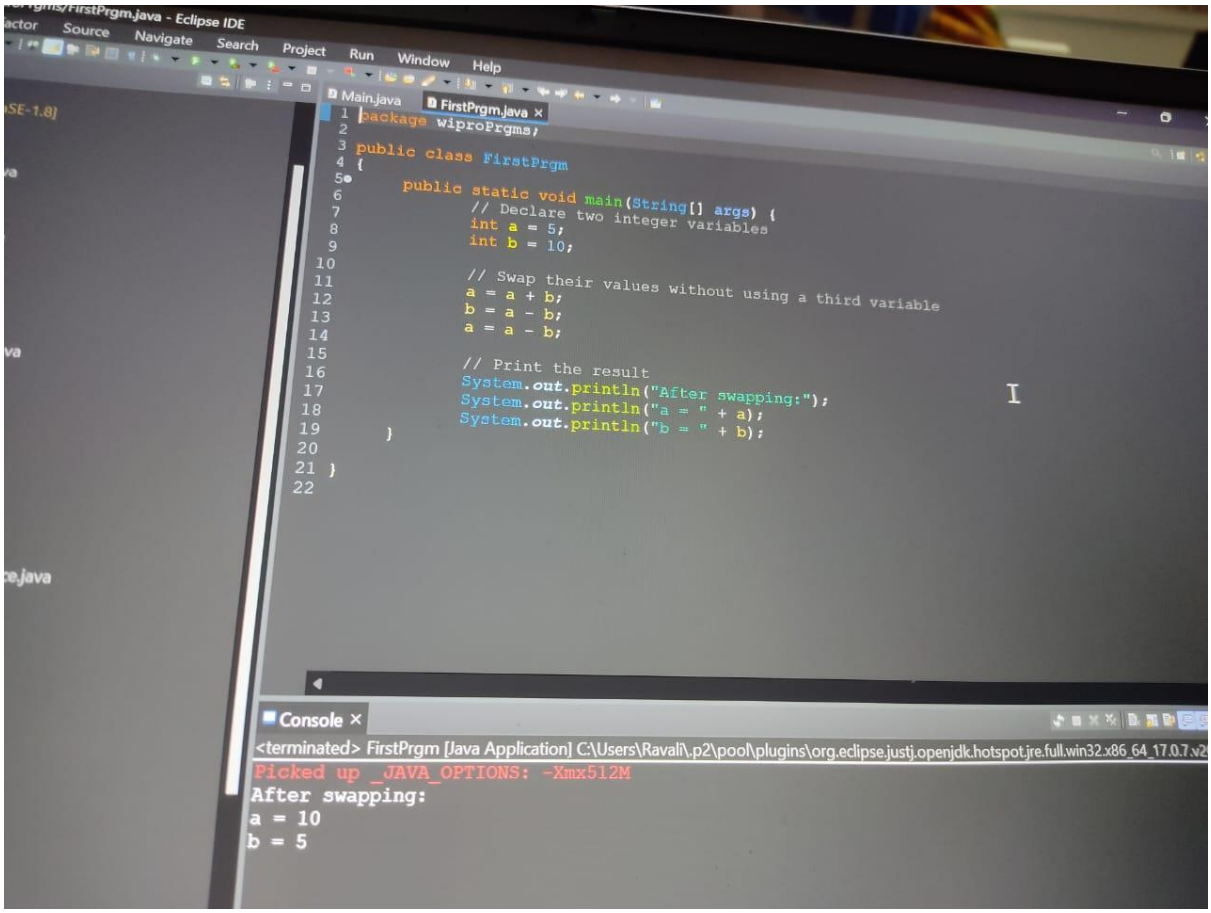
NAME :- **Jangili Ravali**

EMAIL:- **jangiliravali9@gmail.com**

Day 1and 2

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.



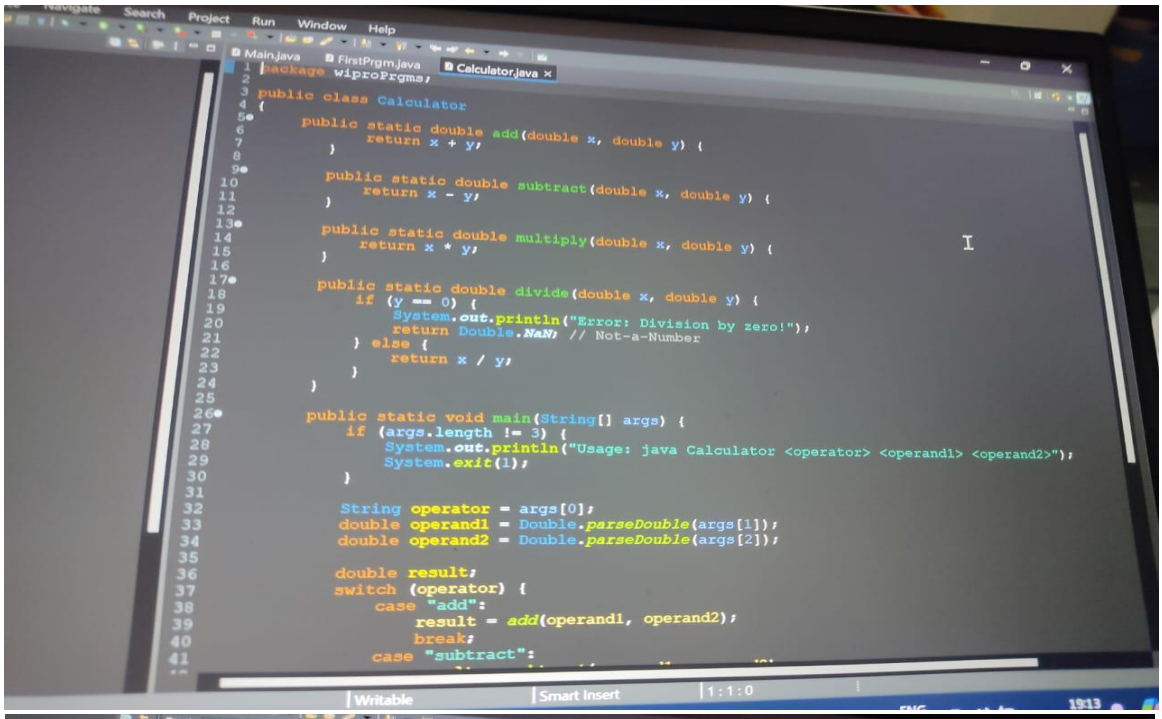
This Java program achieves the same result by swapping the values of a and b without using a third variable**.**

NAME :- **Jangili Ravali**

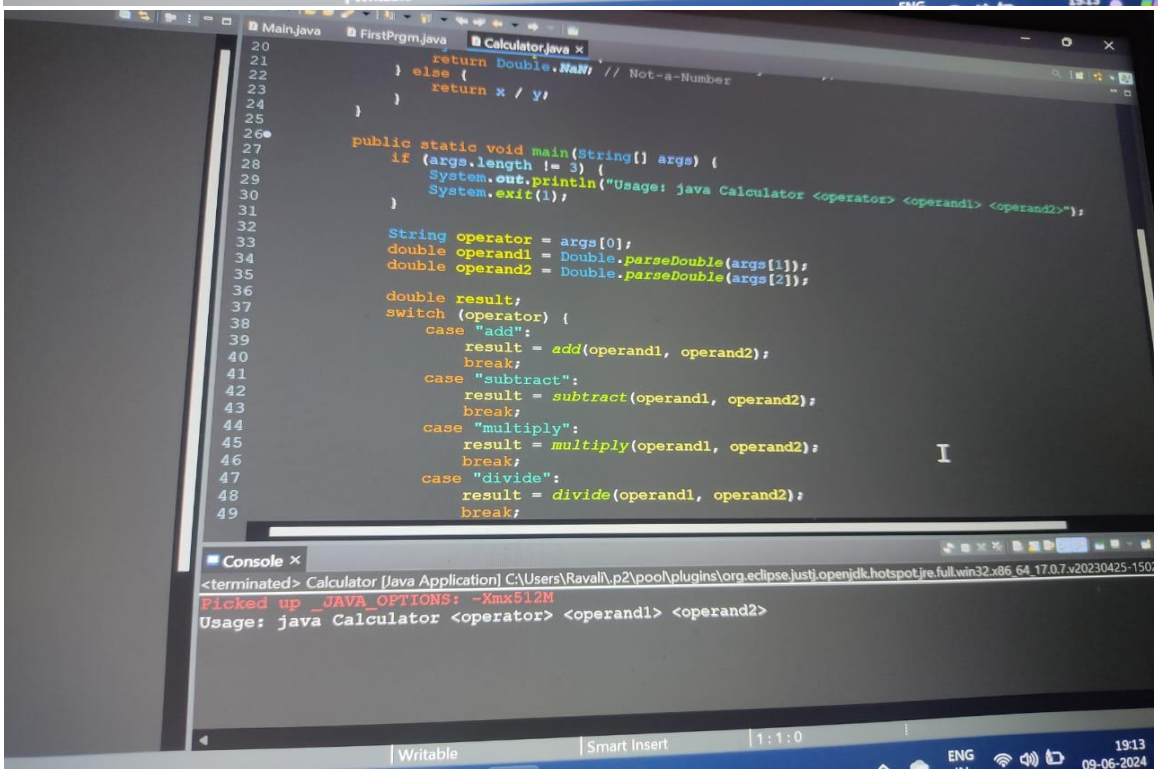
EMAIL:- **jangiliravali9@gmail.com**

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..



```
package wiproPrgms;

public class Calculator
{
    public static double add(double x, double y) {
        return x + y;
    }

    public static double subtract(double x, double y) {
        return x - y;
    }

    public static double multiply(double x, double y) {
        return x * y;
    }

    public static double divide(double x, double y) {
        if (y == 0) {
            System.out.println("Error: Division by zero!");
            return Double.NaN; // Not-a-Number
        } else {
            return x / y;
        }
    }

    public static void main(String[] args) {
        if (args.length != 3) {
            System.out.println("Usage: java Calculator <operator> <operand1> <operand2>");
            System.exit(1);
        }

        String operator = args[0];
        double operand1 = Double.parseDouble(args[1]);
        double operand2 = Double.parseDouble(args[2]);

        double result;
        switch (operator) {
            case "add":
                result = add(operand1, operand2);
                break;
            case "subtract":
```



```
        } else {
            return x / y;
        }
    }

    public static void main(String[] args) {
        if (args.length != 3) {
            System.out.println("Usage: java Calculator <operator> <operand1> <operand2>");
            System.exit(1);
        }

        String operator = args[0];
        double operand1 = Double.parseDouble(args[1]);
        double operand2 = Double.parseDouble(args[2]);

        double result;
        switch (operator) {
            case "add":
                result = add(operand1, operand2);
                break;
            case "subtract":
                result = subtract(operand1, operand2);
                break;
            case "multiply":
                result = multiply(operand1, operand2);
                break;
            case "divide":
                result = divide(operand1, operand2);
                break;
```

```
Picked up _JAVA_OPTIONS: -Xmx512M
Usage: java Calculator <operator> <operand1> <operand2>
```

NAME :- **Jangili Ravali**

EMAIL:- **jangiliravali9@gmail.com**

To compile and run this Java program, save it to a file named Calculator.java, then compile it using javac:

Javac Calculator.java

And run it using java with the desired operation and operands, like this:

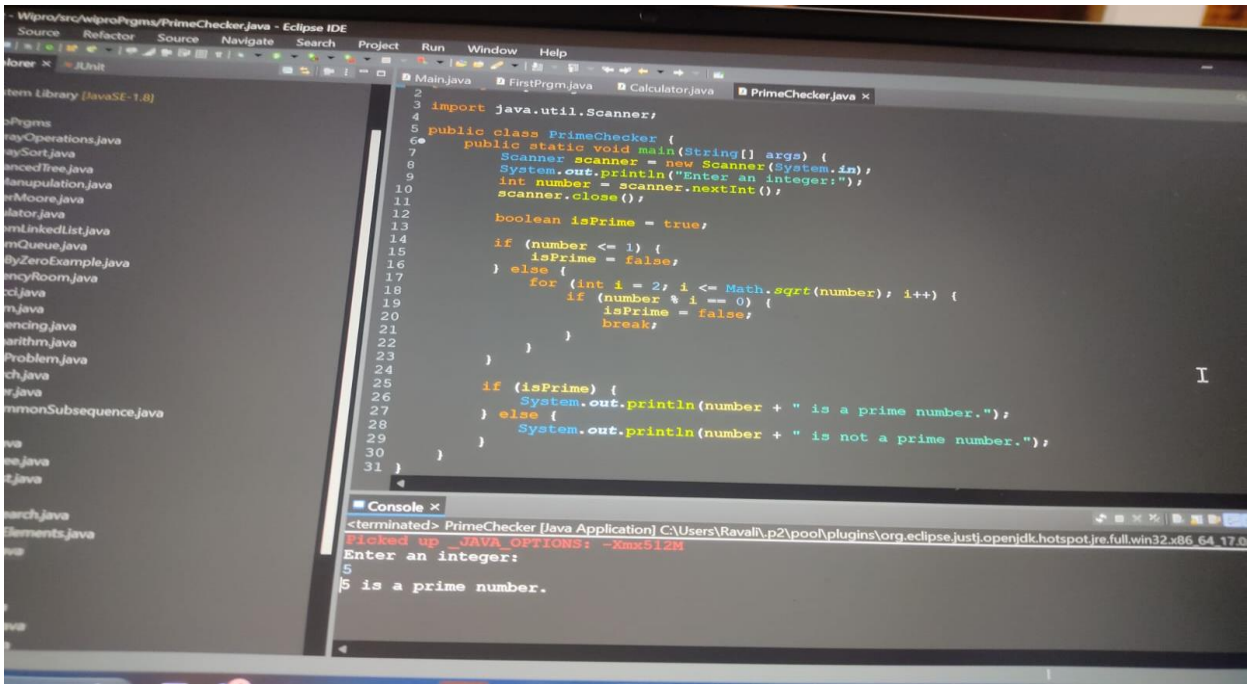
Java Calculator add 5 3

Output

Result: 8.0

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.



NAME :- **Jangili Ravali**

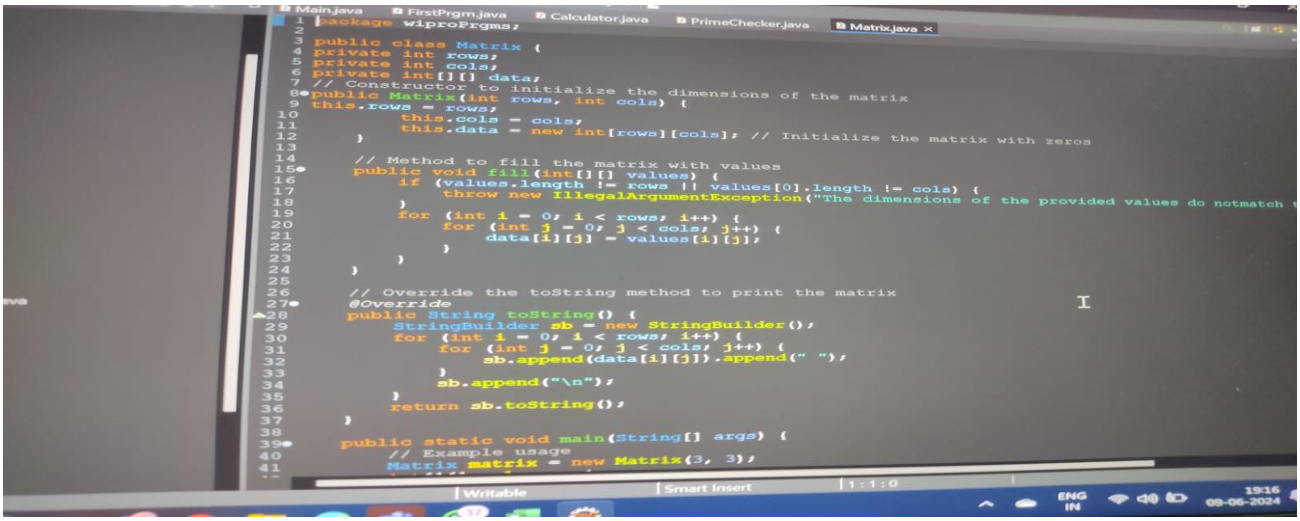
EMAIL:- **jangiliravali9@gmail.com**

How the program works:

1. It prompts the user to enter an integer.
2. It reads the integer from the user.
3. It checks if the entered number is less than or equal to 1. If it is, the number is not prime.
4. If the number is greater than 1, it iterates from 2 to the square root of the entered number.
5. Within the loop, it checks if the entered number is divisible by any number between 2 and the square root of the entered number.
6. If the entered number is divisible by any of these numbers, it sets the isPrime flag to false and breaks out of the loop.
7. After the loop, it checks the value of isPrime. If it's true, the entered number is prime; otherwise, it's not prime.
8. Finally, it prints the result accordingly

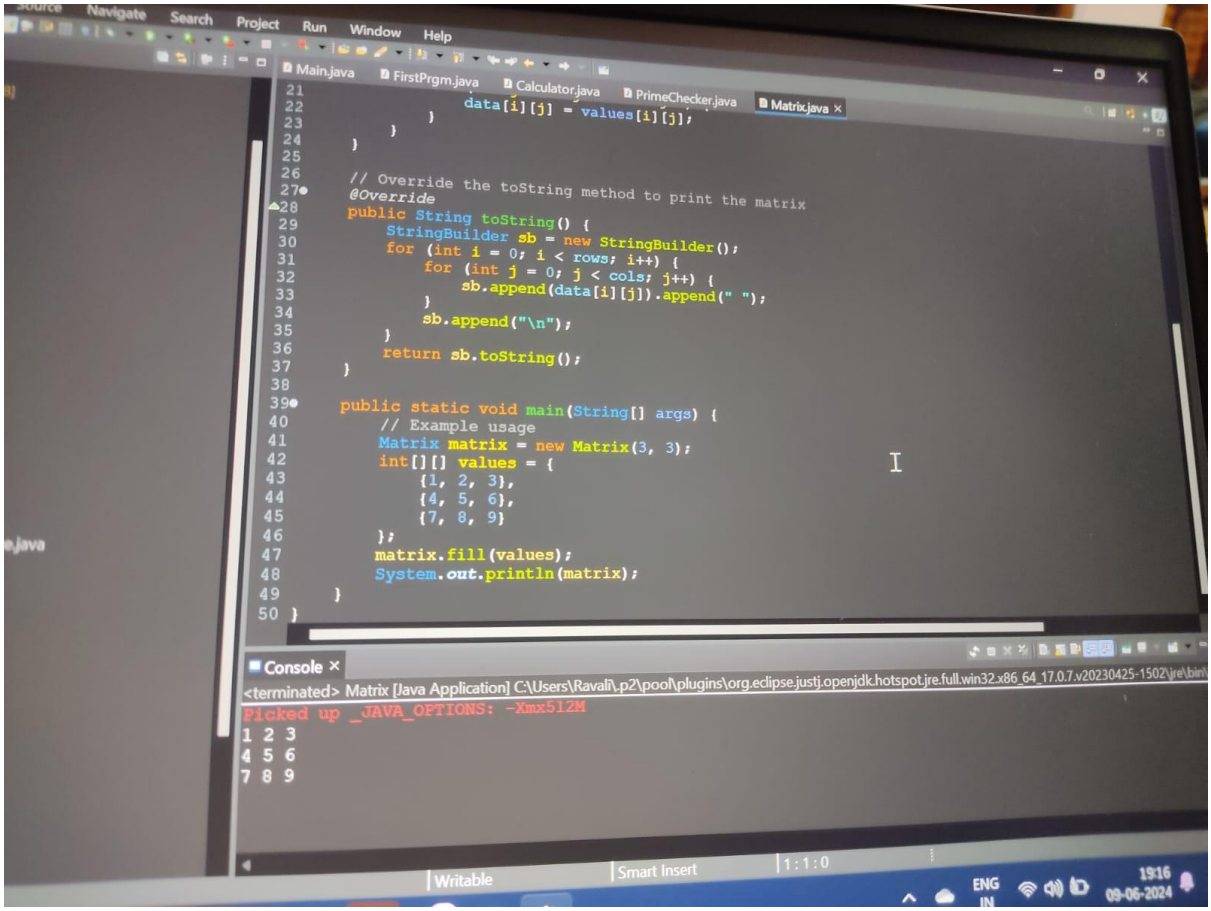
Task 4: Constructors

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



NAME :- **Jangili Ravali**

EMAIL:- **jangiliravali9@gmail.com**



Explanation:

1. Constructor (__init__ method):

- The constructor takes two parameters, rows and cols, to define the dimensions of the matrix.
- It initializes the matrix with zeros using a list comprehension.

2. Fill Method (fill method):

- The fill method takes a list of lists (values) as input.
- It first checks if the dimensions of the provided values match the matrix dimensions. If not, it raises a ValueError.
- If the dimensions match, it assigns the provided values to the matrix.

3. String Representation (__str__ method):

NAME :- **Jangili Ravali**

EMAIL:- **jangiliravali9@gmail.com**

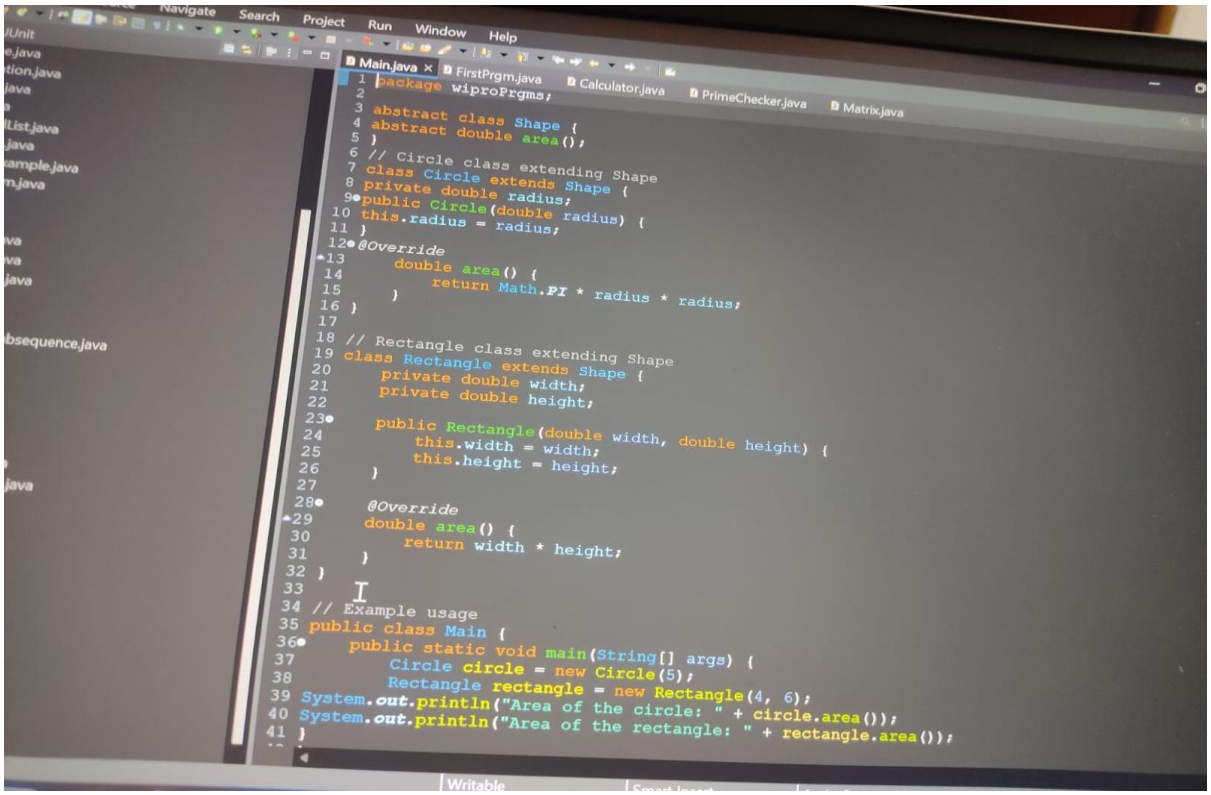
● The __str__ method returns a string representation of the matrix for easy printing. It joins each row's elements with spaces and each row with newlines.

You can use this Matrix class to create a matrix of any dimensions, fill it with values, and print it out. The fill method ensures that the input values match the matrix's dimensions, raising an error if they do not.

Task 5: Inheritance

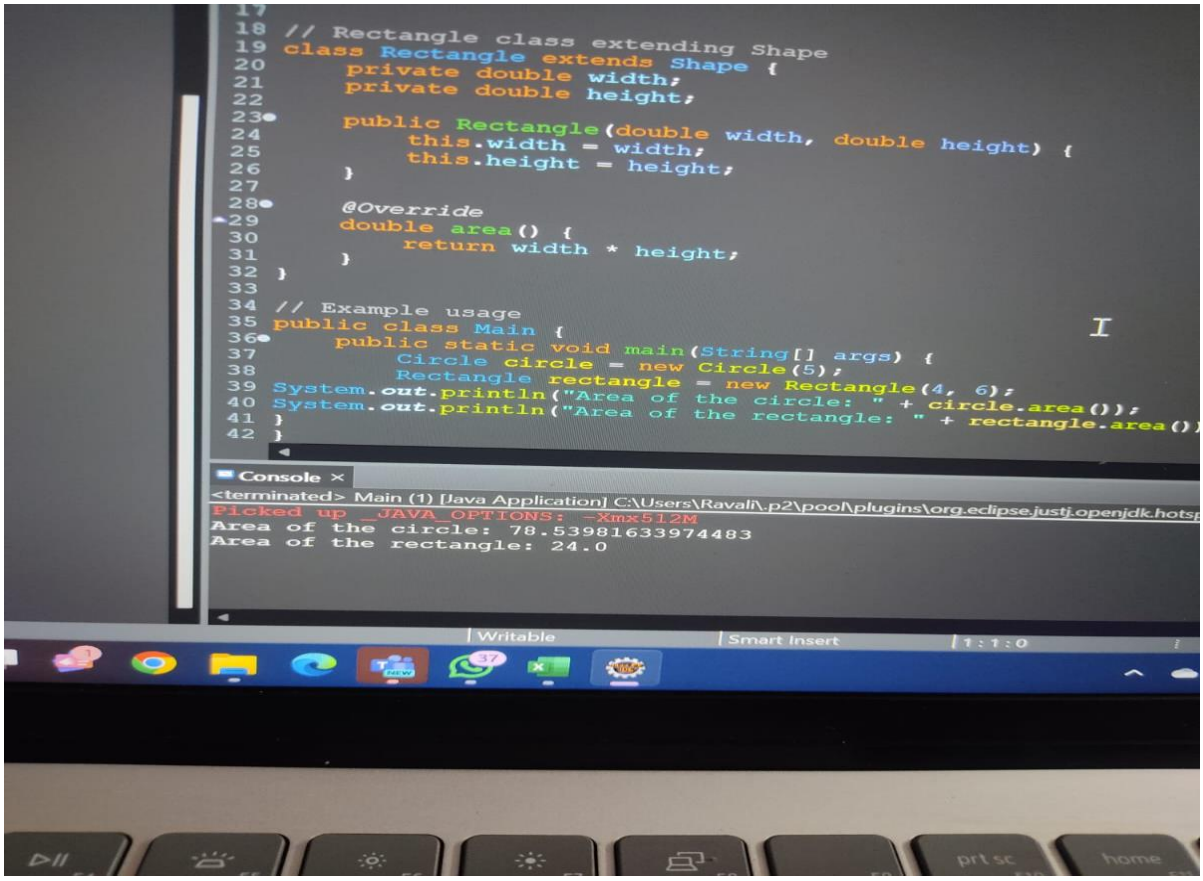
Create a Shape class with a method area() and extend it with Circle and Rectangle classes overriding the area() method appropriately.

Java implementation of the task where we create a Shape class with an area() method and then extend it with Circle and Rectangle classes, each overriding the area() method appropriately.



NAME :- **Jangili Ravali**

EMAIL:- **jangiliravali9@gmail.com**



In this Java code:

1. Shape is an abstract base class with an abstract area() method.

2. Circle is a class extending Shape and implementing the area() method to calculate the area of a circle.

3. Rectangle is a class extending Shape and implementing the area() method to calculate the area of a rectangle.

NAME :- **Jangili Ravali**

EMAIL:- **jangiliravali9@gmail.com**

The Main class contains the main method where instances of Circle and Rectangle are created and their areas are printed.

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.

To create a package com.math.operations and include classes for various arithmetic operations, you can follow these steps:

Step 1: Create the Package and Classes

1. Create Directory Structure

● Create a directory structure to reflect the package name. For example:

com/

math/

operations/

2. Create Java Classes

● Inside the operations directory, create Java classes for different arithmetic operations.

Here are examples for addition and subtraction: Addition.java package

com.math.operations;

```
public class Addition {
public int add(int a, int b) {
return a + b;
}
```

NAME :- **Jangili Ravali**

EMAIL:- **jangiliravali9@gmail.com**

```
}
```

Subtraction.java package

```
com.math.operations;


public    class    Subtraction    {
public int subtract(int a, int b) {
return a - b;
   }
}
```

Step 2: Compile the Classes

1. Open a terminal or command prompt and navigate to the parent directory of com (the root directory where com is located)
2. Compile the Java classes using the javac command:

```
javac com/math/operations/Addition.java com/math/operations/Subtraction.java
```

Step 3: Create a Main Class to Test the Operations

1. Create a main class to use the arithmetic operations. For example:

Main.java import com.math.operations.Addition; import

com.math.operations.Subtraction;

```
public class Main {
   public static void main(String[] args) {
Addition add = new Addition();
      Subtraction sub = new Subtraction();
```

NAME :- **Jangili Ravali**

EMAIL:- **jangiliravali9@gmail.com**

```
        int sum = add.add(5, 3);        int
difference = sub.subtract(5, 3);


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

Step 4: Compile and Run the Main Class

1. Compile the Main Class:

```
javac Main.java
```

2. Run the Main Class using the Classpath:

```
java -cp . Main
```

Directory Structure Example:

Your directory structure should look like this after creating the classes:

```
project_root/
├── Main.java
└── com/
    └── math/
        └── operations/
            ├── Addition.java
└── Subtraction.java Full
```

Example:

Assuming the following file contents:

NAME :- **Jangili Ravali**

EMAIL:- **jangiliravali9@gmail.com**

com/math/operations/Addition.java

```
package com.math.operations;

public class Addition {
public int add(int a, int b) {
return a + b;
  }
}
```

com/math/operations/Subtraction.java

```
package com.math.operations;

public class Subtraction {
public int subtract(int a, int b) {
return a - b;

  }
}
```

Main.java import

```
com.math.operations.Addition; import
com.math.operations.Subtraction;

public class Main {
  public static void main(String[] args) {
Addition add = new Addition();
    Subtraction sub = new Subtraction();
```

NAME :- **Jangili Ravali**

EMAIL:- **jangiliravali9@gmail.com**

```
        int sum = add.add(5, 3);        int
difference = sub.subtract(5, 3);


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

Compilation and Execution Commands:

Navigate to the project root directory where Main.java and com/ are located

javac com/math/operations/Addition.java com/math/operations/Subtraction.java

javac Main.java java -cp . Main

This should output:

Sum: 8

Difference: 2

By following these steps, you create a package com.math.operations, compile the Java classes, and run the main class using the classpath.

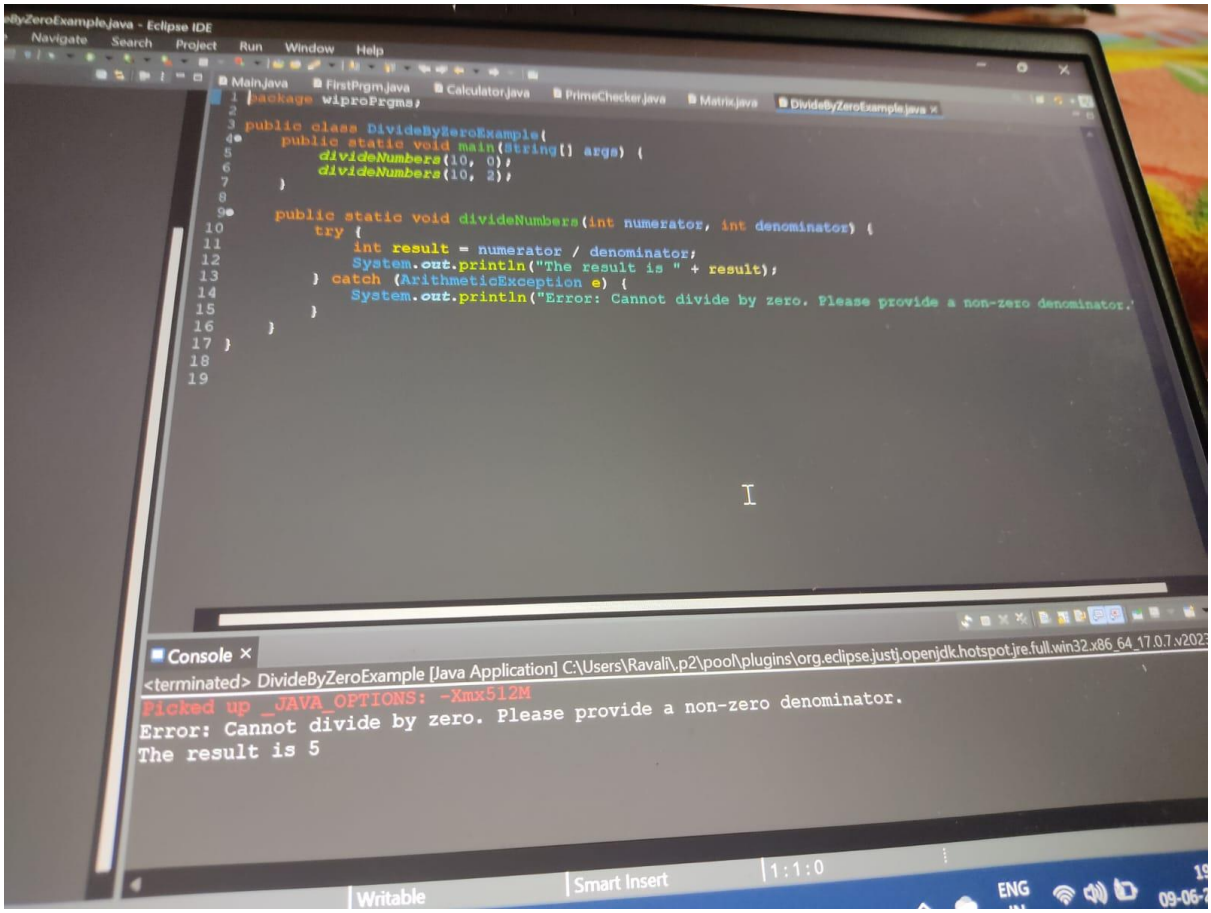
Task 7: Basic Exception Handling

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

Java program that demonstrates basic exception handling for a division by zero error, catching the ArithmeticException, and providing a custom error message:

NAME :- **Jangili Ravali**

EMAIL:- **jangiliravali9@gmail.com**



In this program:

- The main method demonstrates the usage of the divideNumbers method with two examples: one with a denominator of zero and one with a valid non-zero denominator.
- The divideNumbers method attempts to divide the numerator by the denominator inside a try block
- If an ArithmeticException occurs (which happens in case of division by zero), it catches the exception in the catch block and prints a custom error message.
