Project Documentation: Console-Based Calculator in Java

Project Overview

This project is a console-based calculator developed in Java, designed to perform basic arithmetic operations (addition, subtraction, multiplication, and division) on two real numbers. The user can execute the calculator through the command line, specifying the operator and two operands as arguments.

The project adheres to object-oriented programming (OOP) principles by utilizing an abstract class, `Calculator`, which defines the structure for various operation classes (`Addition`, `Subtraction`, `Multiplication`, `Division`). Each class implements a `compute` method specific to the operation.

Key Features:

- Command-line based usage
- Supports four operations: addition (+), subtraction (-), multiplication (*), and division (/)
- Modular design using OOP with an abstract class and concrete subclasses
- Error handling for division by zero

Usage Instructions

To run the calculator, use the following command in your terminal or command prompt:

bash

Copy code

```
java MySoftware <operator> <A> <B>
```

Where:

- `<operator>` can be one of the following: `+`, `-`, `*`, `/`
- `<A>` and `<B>` are two real numbers on which the operation is to be performed

Example 1: Addition

To add two numbers (10 and 5), use the following command:

bash

Copy code

```
java MySoftware + 10 5
```

The output will be:

bash

Copy code

```
15.0
```

Example 2: Subtraction

To subtract 5 from 10:

bash

Copy code

```
java MySoftware - 10 5
```

The output will be:

bash

Copy code

```
5.0
```

Example 3: Multiplication

To multiply 10 by 5:

bash

Copy code

```
java MySoftware * 10 5
```

The output will be:

bash

Copy code

```
50.0
```

Example 4: Division

To divide 10 by 5:

bash

Copy code

```
java MySoftware / 10 5
```

The output will be:

bash

Copy code

```
2.0
```

Note: If you attempt to divide by zero, the program will throw an `ArithmeticException` with the message "Cannot divide by zero."

Project Structure

1. Abstract Class: `Calculator`

The abstract class `Calculator` defines a method signature for `compute(double A, double B)`. It enforces the structure for subclasses to implement specific operations.

java

Copy code

```
public abstract class Calculator {

    public abstract double compute(double A, double B);

}
```

2. Subclasses

Each subclass inherits from `Calculator` and implements the `compute` method for specific operations.

Addition Class

java

Copy code

```
public class Addition extends Calculator {
    @Override
    public double compute(double A, double B) {
        return A + B;
    }
}
```

Subtraction Class

java

Copy code

```
public class Subtraction extends Calculator {
    @Override
    public double compute(double A, double B) {
        return A - B;
    }
}
```

Multiplication Class

java

Copy code

```
public class Multiplication extends Calculator {
    @Override
    public double compute(double A, double B) {
        return A * B;
    }
}
```

Division Class

java

Copy code

```
public class Division extends Calculator {
    @Override
    public double compute(double A, double B) {
        if (B != 0) {
            return A / B;
        } else {
            throw new ArithmeticException("Cannot divide by
     zero.");
        }
    }
}
```

3. Main Class: `MainCalculator`

The `MainCalculator` class handles command-line input, determines the appropriate operator class to instantiate, and calls the `compute` method.

java

Copy code

```
public class MainCalculator {
    public static void main(String[] args) {
        if (args.length != 3) {
            System.out.println("Usage: java MySoftware
     <operator> <A> <B>");
            return;
        }

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

        Calculator calculator = null;

        switch (operator) {
```

```
            case "+":
                calculator = new Addition();
                break;
            case "-":
                calculator = new Subtraction();
                break;
            case "*":
                calculator = new Multiplication();
                break;
            case "/":
                calculator = new Division();
                break;
            default:
                System.out.println("Unknown operator: " +
     operator);
                return;
        }

        double result = calculator.compute(A, B);
        System.out.println("Result: " + result);
    }
}
```

Error Handling

- **Division by Zero**: The program checks for division by zero in the `Division` class. If the denominator (`B`) is zero, the program throws an `ArithmeticException` with the message `"Cannot divide by zero."`
- **Invalid Operator**: If an invalid operator is passed in the command line (e.g., `%`), the program prints `"Unknown operator: <operator>"` and exits.

Testing the Program

To test the program, you can create test cases using the same command-line structure. For example:

Test Case 1: Addition

bash

Copy code

```
java MySoftware + 10 15
```

Expected Output:

bash

Copy code

```
25.0
```

Test Case 2: Division by Zero

bash

Copy code

```
java MySoftware / 10 0
```

Expected Output:

bash

Copy code

```
Cannot divide by zero.
```

Build and Run the Project

1. Compile the Java Files

To compile the files, open the terminal in the project directory and run:

bash

Copy code

```
javac *.java
```

2. Run the Program

To run the program, use the following command:

bash

Copy code

```
java MainCalculator + 5 10
```

3. Package as a JAR File

You can package the project as a JAR file:

bash

Copy code

```
jar cf MySoftware.jar *.class
```

To run the JAR file:

bash

Copy code

```
java -jar MySoftware.jar + 5 10
```

Version Control

The project is maintained using Git for version control. Team members are responsible for creating and reviewing pull requests, ensuring code quality, and adhering to coding standards.

Future Enhancements

- Add support for additional operations (e.g., modulus, exponentiation).
- Implement user-friendly error messages and input validation.
- Extend the calculator to support more complex operations like functions (e.g., square root, logarithm).

Project Documentation: Console-Based Calculator in Java

Project Overview

This project is a console-based calculator developed in Java, designed to perform basic arithmetic operations (addition, subtraction, multiplication, and division) on two real numbers. The user can execute the calculator through the command line, specifying the operator and two operands as arguments.

The project adheres to object-oriented programming (OOP) principles by utilizing an abstract class, `Calculator`, which defines the structure for various operation classes (`Addition`, `Subtraction`, `Multiplication`, `Division`). Each class implements a `compute` method specific to the operation.

Key Features:

- Command-line based usage
- Supports four operations: addition (+), subtraction (-), multiplication (*), and division (/)
- Modular design using OOP with an abstract class and concrete subclasses
- Error handling for division by zero

Usage Instructions

To run the calculator, use the following command in your terminal or command prompt:

bash

Copy code

```
java MySoftware <operator> <A> <B>
```

Where:

- `<operator>` can be one of the following: `+`, `-`, `*`, `/`
- `<A>` and `<B>` are two real numbers on which the operation is to be performed

Example 1: Addition

To add two numbers (10 and 5), use the following command:

bash

Copy code

```
java MySoftware + 10 5
```

The output will be:

bash

Copy code

```
15.0
```

Example 2: Subtraction

To subtract 5 from 10:

bash

Copy code

```
java MySoftware - 10 5
```

The output will be:

bash

Copy code

```
5.0
```

Example 3: Multiplication

To multiply 10 by 5:

bash

Copy code

```
java MySoftware * 10 5
```

The output will be:

bash

Copy code

```
50.0
```

Example 4: Division

To divide 10 by 5:

bash

Copy code

```
java MySoftware / 10 5
```

The output will be:

bash

Copy code

```
2.0
```

Note: If you attempt to divide by zero, the program will throw an `ArithmeticException` with the message "Cannot divide by zero."

Project Structure

1. Abstract Class: `Calculator`

The abstract class `Calculator` defines a method signature for `compute(double A, double B)`. It enforces the structure for subclasses to implement specific operations.

java

Copy code

```
public abstract class Calculator {
    public abstract double compute(double A, double B);
}
```

2. Subclasses

Each subclass inherits from `Calculator` and implements the `compute` method for specific operations.

Addition Class

java

Copy code

```
public class Addition extends Calculator {
    @Override
```

```
    public double compute(double A, double B) {
        return A + B;
    }
}
```

Subtraction Class

java

Copy code

```
public class Subtraction extends Calculator {
    @Override
    public double compute(double A, double B) {
        return A - B;
    }
}
```

Multiplication Class

java

Copy code

```
public class Multiplication extends Calculator {
    @Override
    public double compute(double A, double B) {
```

```
        return A * B;
    }
}
```

Division Class

java

Copy code

```
public class Division extends Calculator {
    @Override
    public double compute(double A, double B) {
        if (B != 0) {
            return A / B;
        } else {
            throw new ArithmeticException("Cannot divide by
     zero.");
        }
    }
}
```

3. Main Class: `MainCalculator`

The `MainCalculator` class handles command-line input, determines the appropriate operator class to instantiate, and calls the `compute` method.

java

Copy code

```
public class MainCalculator {
    public static void main(String[] args) {
        if (args.length != 3) {
            System.out.println("Usage: java MySoftware
     <operator> <A> <B>");
            return;
        }

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

        Calculator calculator = null;

        switch (operator) {
            case "+":
                calculator = new Addition();
                break;
```

```
            case "-":
                calculator = new Subtraction();
                break;
            case "*":
                calculator = new Multiplication();
                break;
            case "/":
                calculator = new Division();
                break;
            default:
                System.out.println("Unknown operator: " +
     operator);
                return;
        }

        double result = calculator.compute(A, B);
        System.out.println("Result: " + result);
    }
}
```

Error Handling

- **Division by Zero**: The program checks for division by zero in the `Division` class. If the denominator (`B`) is zero, the program throws an `ArithmeticException` with the message `"Cannot divide by zero."`
- **Invalid Operator**: If an invalid operator is passed in the command line (e.g., `%`), the program prints `"Unknown operator: <operator>"` and exits.

Testing the Program

To test the program, you can create test cases using the same command-line structure. For example:

Test Case 1: Addition

bash

Copy code

```
java MySoftware + 10 15
```

Expected Output:

bash

Copy code

```
25.0
```

Test Case 2: Division by Zero

bash

Copy code

```
java MySoftware / 10 0
```

Expected Output:

bash

Copy code

```
Cannot divide by zero.
```

Build and Run the Project

1. Compile the Java Files

To compile the files, open the terminal in the project directory and run:

bash

Copy code

```
javac *.java
```

2. Run the Program

To run the program, use the following command:

bash

Copy code

```
java MainCalculator + 5 10
```

3. Package as a JAR File

You can package the project as a JAR file:

bash

Copy code

```
jar cf MySoftware.jar *.class
```

To run the JAR file:

bash

Copy code

```
java -jar MySoftware.jar + 5 10
```

Version Control

The project is maintained using Git for version control. Team members are responsible for creating and reviewing pull requests, ensuring code quality, and adhering to coding standards.

Future Enhancements

- Add support for additional operations (e.g., modulus, exponentiation).
- Implement user-friendly error messages and input validation.
- Extend the calculator to support more complex operations like functions (e.g., square root, logarithm).