Bug 1: Rounding error

- **Summary**: Results are guaranteed exact up to 8 decimal places in all the functions (Add, Multiply, Subtract and Divide).
- **Steps to Reproduce**:
    1. Open a terminal.
    2. Run the following Docker command:

       ```
       docker run --rm public.ecr.aws/l4q9w4c5/loanpro-calculator-cli add
       1.00000001 1.00000001
       ```

    3. Observe the result.
- **Expected Result**: Result: 2.0
- **Actual Result**: Result: 2.00000002
- **Potential Cause**: Yields the expected 2.0000002, but
- 1.00000001 + 1.00000001 (seven 0's) results in 2.0. Same behavior for all the functions.
- **Automation Code**:

  ```
  java
  runDockerCommand("add", 1.00000001, 1.00000001, "Result: 2.0");
  ```

Bug 2: Commands take exactly two numbers

- **Summary**: Attempting to use more or less operands will result in an error message; this is expected behavior in all the functions (Add, Multiply, Subtract and Divide).
- **Steps to Reproduce**:
    1. Open a terminal.
    2. Run the following Docker command:

       ```
       docker run --rm public.ecr.aws/l4q9w4c5/loanpro-calculator-cli add
       docker run --rm public.ecr.aws/l4q9w4c5/loanpro-calculator-cli add 8
       ```

    3. Observe the result.
- **Expected Result**: Error: Commands take exactly two numbers.
- **Actual Result**: Usage: cli-calculator operation operand1 operand2
  Supported operations: add, subtract, multiply, divide
- **Potential Cause**: The application is expecting only 2 operators and is receiving less than 2. Same behavior for all the functions.
- **Automation Code**:

java
runDockerCommandWithInvalidOperands("add ", "Error: Commands take exactly two numbers.");

Bug 3: More than 3 operator's error.

- **Summary**: The application does not handle cases where are send more than 2 number operators in all the functions (Add, Multiply, Subtract and Divide)..
- **Steps to Reproduce**:
    1. Open a terminal.
    2. Run the following Docker command:

       docker run --rm public.ecr.aws/l4q9w4c5/loanpro-calculator-cli add 8 8 8

    3. Observe the result.
- **Expected Result**: Error: Invalid number of operands
- **Actual Result** Result: 16
- **Potential Cause**: The application is using the first 2 numbers as arguments of the functions.
- **Automation Code**:

  java
  runDockerCommandWithInvalidOperands("add 8 8 8", "Error: Invalid number of operands");

Source Code for Automation

Java Test Code

```
java

package DockerCalculator;

import org.testng.Assert;
import org.testng.annotations.AfterClass;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.Test;

import java.io.BufferedReader;
import java.io.InputStreamReader;

public class DockerCalculatorTest {

  @BeforeClass
```

```
    public void setUp() {
        System.out.println("Setting up the test environment...");
    }

    @AfterClass
    public void tearDown() {
        System.out.println("Cleaning up the test environment...");
    }

   @Test
    public void testAdd() {
        runDockerCommand("add", 8, 5, "Result: 13");
        runDockerCommand("add", 1.0000001, 1.0000001, "Result: 2.0000002");
        runDockerCommand("add", 1.00000001, 1.00000001, "Result: 2.0"); // Rounding
error
    }

    @Test
    public void testSubtract() {
        runDockerCommand("subtract", 8, 5, "Result: 3");
        runDockerCommand("subtract", 5, 8, "Result: -3");
    }

    @Test
    public void testMultiply() {
        runDockerCommand("multiply", 8, 5, "Result: 40");
        runDockerCommand("multiply", 1e8, 1e8, "Result: 10000000000000000"); //
Scientific notation
    }

    @Test
    public void testDivide() {
        runDockerCommand("divide", 10, 5, "Result: 2");
        runDockerCommand("divide", 1, 3, "Result: 0.33333333");
        runDockerCommand("divide", 1, 0, "Error: Cannot divide by zero"); // Division by
zero
        runDockerCommand("divide", 1, 10000000, "Result: 0.0000001");
        runDockerCommand("divide", 1, 100000000, "Result: 0"); // Rounding error
    }

    @Test
    public void testInvalidOperands() {
        runDockerCommandWithInvalidOperands("add a b", "Invalid argument. Must be a
numeric value.");
        runDockerCommandWithInvalidOperands("add ", "Error: Commands take exactly
two numbers.");
        runDockerCommandWithInvalidOperands("add 8", "Error: Commands take exactly
two numbers.");
        runDockerCommandWithInvalidOperands("add 8 8 8", "Error: Commands take
exactly two numbers.");
        runDockerCommandWithInvalidOperands("subtract ", "Error: Commands take
exactly two numbers.");
        runDockerCommandWithInvalidOperands("subtract a b", "Error: Commands take
exactly two numbers.");
        runDockerCommandWithInvalidOperands("subtract 8", "Error: Commands take
exactly two numbers.");
        runDockerCommandWithInvalidOperands("subtract 8 8 8", "Error: Commands take
exactly two numbers.");
        runDockerCommandWithInvalidOperands("multiply ", "Error: Commands take
exactly two numbers.");
        runDockerCommandWithInvalidOperands("multiply a b", "Error: Commands take
exactly two numbers.");
        runDockerCommandWithInvalidOperands("multiply 8", "Error: Commands take
exactly two numbers.");
        runDockerCommandWithInvalidOperands("multiply 8 8 8", "Error: Commands take
exactly two numbers.");
        runDockerCommandWithInvalidOperands("divide a b", "Error: Commands take
exactly two numbers.");
        runDockerCommandWithInvalidOperands("divide ", "Error: Commands take exactly
two numbers.");
        runDockerCommandWithInvalidOperands("divide 8", "Error: Commands take exactly
two numbers.");
        runDockerCommandWithInvalidOperands("divide 8 8 8", "Error: Commands take
exactly two numbers.");
    }

    private void runDockerCommand(String operation, double num1, double num2, String
expectedMessage) {
        try {
            System.out.println("docker run --rm public.ecr.aws/l4q9w4c5/loanpro-calculator-cli
" + operation + " " + num1 + " " + num2);
            ProcessBuilder processBuilder = new ProcessBuilder();
            processBuilder.command("cmd", "/c", "docker run --rm
public.ecr.aws/l4q9w4c5/loanpro-calculator-cli " + operation + " " + num1 + " " + num2);
            Process process = processBuilder.start();
            BufferedReader reader = new BufferedReader(new
InputStreamReader(process.getInputStream()));
            StringBuilder output = new StringBuilder();
            String line;
            while ((line = reader.readLine()) != null) {
                output.append(line);
            }
            System.out.println("(Output) " + output);
            System.out.println("(Expected Message) " + expectedMessage + "\n");
            int exitCode = process.waitFor();
            Assert.assertTrue(output.toString().contains(expectedMessage), "The expected
result is not present in the output.");

        } catch (Exception e) {
            e.printStackTrace();
            Assert.fail("An error occurred while executing the Docker command.");
        }
    }

    private void runDockerCommandWithInvalidOperands(String invalidOperation, String
expectedMessage) {
        try {
            System.out.println("docker run --rm public.ecr.aws/l4q9w4c5/loanpro-calculator-cli
" + invalidOperation);
            ProcessBuilder processBuilder = new ProcessBuilder();
            processBuilder.command("cmd", "/c", "docker run --rm
public.ecr.aws/l4q9w4c5/loanpro-calculator-cli " + invalidOperation);
            Process process = processBuilder.start();
            BufferedReader reader = new BufferedReader(new
InputStreamReader(process.getInputStream()));
            StringBuilder output = new StringBuilder();
            String line;
            while ((line = reader.readLine()) != null) {
                output.append(line);
            }
            System.out.println("(Output) " + output);
            System.out.println("(Expected Message) " + expectedMessage + "\n");
            int exitCode = process.waitFor();
            Assert.assertTrue(output.toString().contains(expectedMessage), "The expected
result is not present in the output.");

        } catch (Exception e) {
            e.printStackTrace();
            Assert.fail("An error occurred while executing the Docker command.");
        }
    }
}
```

Instructions to Run the Tests

1. **Ensure Docker is installed**: Verify that Docker is installed and running on your machine.
2. **Setup the project**:
    - Create a Maven project if you don't have one.
    - Add the provided Java test code to your project.
3. **Update pom.xml**: Ensure your pom.xml has the necessary dependencies:

xml
```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.example</groupId>
    <artifactId>docker-test</artifactId>
    <version>1.0-SNAPSHOT</version>

    <dependencies>
        <dependency>
            <groupId>org.testng</groupId>
            <artifactId>testng</artifactId>
            <version>7.4.0</version>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-plugin</artifactId>
                <version>2.22.2</version>
                <configuration>
                    <includes>
                        <include>**/*Test.java</include>
                    </includes>
                    <testFailureIgnore>false</testFailureIgnore>
                    <printSummary>true</printSummary>
                    <redirectTestOutputToFile>false</redirectTestOutputToFile>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-report-plugin</artifactId>
                <version>2.22.2</version>
            </plugin>
        </plugins>
    </build>
</project>
```

4. **Run the tests**: Execute the tests using Maven:

   ```
   mvn test -X
   ```

This document should help developers replicate the bugs in their own environment and provide insight into what might be causing these issues. The provided automation code and instructions ensure that they can reproduce the tests accurately.