Infinite Precision Calculator

Final Project Report for CS1023
Software Development Fundamentals

Submitted by:

Devansh Tripathi
CS24BTECH11022

Date: April 13, 2025

Indian Institute of Technology Hyderabad

1 Abstract

This Java library contains classes and methods to enable infinite precision arithmetic operations for integers and float types. This solves the problem of doing addition, subtraction, multiplication and division on large numbers that may overflow the bounds of a regular `int` or `float`, or even `long` or `double`, or it may be the case with their result (eg: multiplying two very big numbers or infinite division).

The main challenges in the project were:

- Performing the operations on numbers with different number of digits.
- Operations on float types as they have integral and fractional parts.
- Logic for handling position of decimal points in the case of multiplication or division of float types and acheiving precision upto n places.
- Dealing with negative numbers and handling cases for each operation.

2 Introduction

Precision is quite important for complex calculations where even small changes can drastically change the final result. Hence, this library tries to present a solution to this problem of bounded values of standard `int`, `long`, `float` and `double` types in Java by storing the numbers as Strings, and then doing the required operation digit-wise and storing the result into a String as well. Since there is no explicit bound on the size of a String in Java, this solves the problem of precision as the result can be as long as desired.

Data range of the standard data types in Java:

- int : -2147483648 to 2147483647
- long : -9223372036854775808 to 9223372036854775807
- float : 1.40239846e-45 to 3.40282347e+38
- double : 4.94065645841246544e-324 to 1.79769313486231570e+308

What this library supports:

- Addition, Subtraction, Multiplication and Divison of numbers of arbitrary length, even those exceeding the data range of the standard data types.
- Division producing a result capable of having an arbitrary amount of decimal places, it has been set to 1000 but can be extended.

3 Features

A brief description of the main components of this library:

- `AInteger`
  - This class handles parsing integers(small and large) and all the integer-related operations.
  - Each `AInteger` object stores the number given to it as a String.
  - Since the numbers are stored as Strings, the length of the numbers can be arbitrary.
  - Supports addition, subtraction, multiplication, and division.
- `AFloat`
  - This class handles all the float-related operations and parsing.
  - Each `AFloat` object stores the floating point number given to it as a String.
  - The number can be arbitrary precision- there is no bound on it from either side.
  - Supports addition, subtraction, multiplication and division upto arbitrary precision.
- `MyInfArith`
  - Although this class is not a part of the package, it is still a convenient way to use the above classes via the command line.
  - Usage of this class is given in the **Build and Run** section.

4 Implementation

- Language: **Java 21**
- Build Tool used: **Maven**
- Structure: The main source files are located in the `src/main/java/arbitraryarithmetic` directory, namely `AInteger.java` and `AFloat.java`.
- Each number is stored in an object which is either `AInteger` or `AFloat`. The number is stored as a private variable `value`, which can be accessed by using the accessor function `getValue()`.
- Operations are implemented as methods, so the four operations are implemented as `add`, `subtract`, `multiply` and `divide` in either class.
- To perform an operation between `x1` and `x2`, use the statment `x1.operation_name(x2)`, for example if we want to add two `AInteger` objects `a` and `b`, we would use `a.add(b)`.

- Performing an operation will return the result of the operation as an object of the same type. The value of the result can be accessed by the `getValue()` function.
- The main challenges encountered in the implementation were:
  - Performing the operations on numbers with different number of digits.
  - Operations on float types as they have integral and fractional parts.
  - Logic for handling position of decimal points in the case of multiplication or division of float types and acheiving precision upto n places.
  - Dealing with negative numbers and handling cases for each operation.

5 Testing

Correctness of the functions of these classes were mainly testes manually, and all the cases used can be found in `raw_source/test_files/` directory.

Here are some example cases:

5.1 AInteger

1. 123 + 456 = 579
2. -123 + 456 = 333
3. 123 + -456 = -333
4. -123 + -456 = -579
5. 1000000000000000000000000 + 1 = 1000000000000000000000001
6. 999999999999999999999999 + 1 = 1000000000000000000000000
7. 1000000000000000000000000 - 999999999999999999999999 = 1
8. -999999999999999999999999 + -1 = -1000000000000000000000000
9. 0 + 0 = 0
10. 0 - 0 = 0
11. 0 - 123 = -123
12. 123 - 0 = 123

5.2 AFloat

1. 1.0 / 3.0 = 0.333333.... (1000 places)
2. 2.0 / 3.0 = 0.666666.... (1000 places)
3. 0.0000000000001 + 0.0000000000001 = 0.0000000000002
4. 999999999999.999999999999 + 0.000000000001 = 1000000000000.0
5. 999999999999.999 * 1.001 = 1000999999999.998999
6. -123456.789 * -1.0 = 123456.789
7. 1.000000 / 1000000.0 = 0.000001
8. -0.0 + 0.0 = 0.0
9. -0.0 - -0.0 = 0.0
10. 0.0 * 0.1 = 0.0
11. 0.0 / 0.1 = 0.0
12. 1.0000000000001 - 1.0000000000000 = 0.0000000000001
13. 1000.0000001 - 999.9999999 = 0.0000002

All these answers can be verified to be correct.

6 Build and Run

- Dependecies: **Java 21**, **Maven**
- How to build the package (.JAR):
  Run the following command in the terminal from the project directory:
  `mvn clean package`
- How to link the package to your file:
  - First, compile your file using the command:
    `javac -cp .:<add the absolute path to the aarithmetic.jar package here> <relative path to your desired file>`
    After this, you will find the package `aarithmetic.jar` in the `target/` directory.
  - Next, run your file using:
    `java -cp .:<absolute path to aarithmetic.jar> <relative path to your file>`
- If you just want to test out the package functions or perform simple operations, there are two ways to do this:

  - Link the package to `MyInfArith/MyInfArith.java` and then run it with the following command:
    ```
    java -cp .:<absolute path to aarithmetic.jar> MyInfArith.MyInfArith
    <int/float> <add/sub/mul/div> <operand_1> <operand_2>
    ```
  - Configure the `my_exe` Python script, editing the path to the .JAR file, and then run it with the four arguments, which will automate the above process:
    ```
    <python3/python> my_exe <int/float> <add/sub/mul/div> <operand_1>
    <operand_2>
    ```
    or
    ```
    ./my_exe <int/float> <add/sub/mul/div> <operand_1> <operand_2>
    ```
  - Example Input:
    ```
    ./my_exe float sub 840196454.51725 712586963.70283
    ```
    Output:
    ```
    127609490.81442
    ```

7 Conclusion

7.1 Key Learnings

- Handling large numbers as Strings and performing arithmetic operations on them.
- Object Oriented Programming with Java.
- Logic of basic arithmetic operations digit-wise.
- Build systems in Java.
- How to use .JAR packages in files.
- Using Python to build execution scripts.

7.2 Areas of Improvement

- A lot of code is reused within the same class for different cases. For example, trimming zeroes for integers and floats in the `AFloat` class has the same core logic but slightly different implementation.
- Right now I am not handling the operations universally, i.e., I am checking for the positivity of the operands and making cases for each operation.