Java Project Report
Arbitrary Arithmetic Precision

Gulla Mitra

CS24BTECH11027

May 2025

1 Introduction

This document provides a implementation of Arbitrary precision arithmetic in Java. The main motivation for this project is to overcome the limitations of built in numeric types which cannot handle large inputs as size is bounded. By representing numbers as strings we can perform fundamental arithmetic operations manually, this approach provides full control over precision, sign handling.

The system includes support for both integer and floating point operations,organized into seperate classes: **AInteger**, **AFloat**, to execute operations.There is main class **MyInfArith** which handles command line arguments and perform required arithmetic operations.

Operations Addition,Subtraction,Mutiplication,Divison can be performed in both integer and float types.

2 Project Folder

```
project_folder-
               javasrc-
                       arbitraryarithmetic-
                                           AInteger.java
                                           AInteger.class
                                           AFloat.java
                                           AFloat.class
                       MyInfArith.java
                       MyInfArith.class
```

```
build-
      arbitraryarithmetic-
                           AInteger.class
                           AFloat.class
      MyInfArith.class
arbitraryarithmetic-
                    aarithmetic.jar
build.xml
execute_java.py
latex_report.pdf
latex_report.tex
```

3 AInteger.java

The `AInteger` class provides functionality for representing and manipulating arbitrarily large integers using string-based internal storage. It supports arithmetic operations like addition, subtraction, multiplication and division with proper handling of signs.

Constructors

- `AInteger()`
  Default constructor AInteger() that initializes the instance with value "0".
- `AInteger(String str)`
  Constructor AInteger(String s) that initializes the instance by the number whose string representation is given by 's'.Removes leading zeros if present.
- `AInteger(AInteger other)`
  Copy constructor that creates an instance of AInteger.

Methods

- **removezeroes(String str)**
  This method removes the leading zeroes if present in the string.If zeroes are present while performing operations some Errors may occur.So we should remove zeroes using this method.
- **Sign(String str1,String str2)**
  Determines the sign of the result when subtracting two non-negative integer strings. If `str1` is less than `str2`, the method returns `"-"` indicating result as negative, otherwise, returns an empty string.This method is used in subtraction of two non negative integers.

- **less(String str1, String str2)**
  Determines the lesser string in the given two strings.This method is used in division to know which string is lesser in the given two strings.

- **parse(String str)**
  This static method creates and returns a new `AInteger` instance initialized with the given string `str`. It is equivalent to calling a constructor with a string.

- **toString()**
  Returns the internal string representation of the AInteger object.

- **addition(String str1,String str2)**
  Performs digit-wise addition of two non-negative integer strings `str1` and `str2`. The method iterates from right to left carrying the carry.
  This logic is same as manual addition. The final result is returned as a string after removing the leading zeroes.

- **subtract(String str1,String str2)**
  Performs the digit-wise subtraction of two non-negative integer strings. If first number `str1` is smaller than `str2`,then the method swaps the two numbers and stores the negative sign. It handles borrowing if needed and returns the result as a string after removing the leading zeroes.

- **multiply(String str1,String str2)**
  Performs the digit-wise mutiplication of two non-negative integer strings similar to the manual multiplication.for each digit in the `str2` starting from right side, it mutiples it with every digit in `str1` and forms partial products.
  These partial products are shifted properly by appending zeroes and are added using `addition` method.It returns the final result as a string after removing the leading zeroes.

- **AInteger add(AInteger other)**
  Adds the current `AInteger` to another instance. It handles the signs of both number strings and determines whether the operation should be an addition or subtraction and calls the required function. Prefixes the correct sign to final result and returns as a new `AInteger` instance containg sum.

- **AInteger sub(AInteger other)**
  Subtracts another `AInteger` instance from the current one. It handles the signs of both number strings and determines whether the operation should be an addition or subtraction and calls the required function. Prefixes the correct sign to the final result and returns a new `AInteger` instance containing difference.

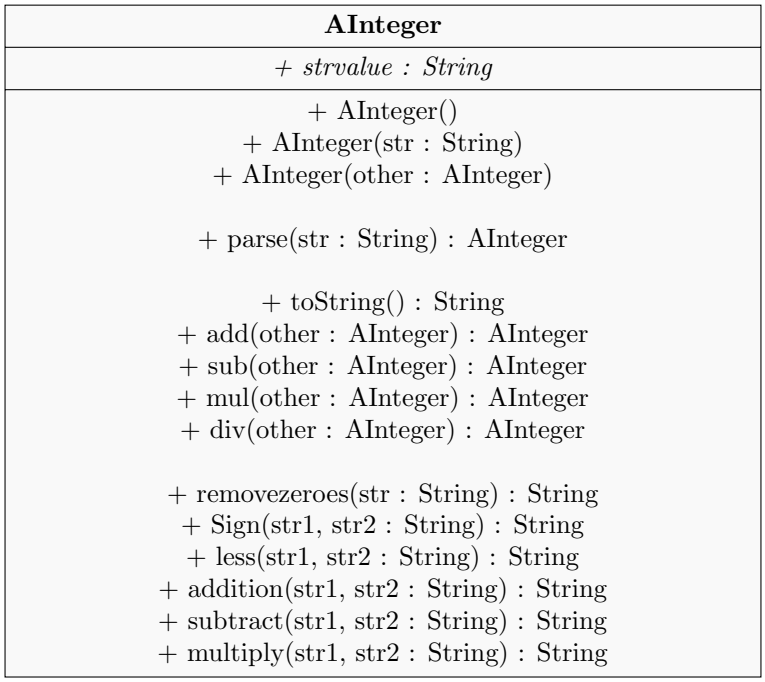
- **AInteger mul(AInteger other)**
  Multiplies the current `AInteger` with another instance. This method first

checks the signs of both number strings to determine the sign of the final result using `XOR` logic (XOR-the result is false if exactly one is false). After checking sign,it removes negative signs if present,then calls `multiply` method.Prefixes the sign to the final result if needed and returns a new `AInteger` instance containing the product.

- **AInteger div(AInteger other)**
  Divides the current `AInteger` by another instance and returns the quotient as a new `AInteger` instance. This method first checks the signs of both number strings to determine the sign of the final result using `XOR` logic (XOR-the result is false if exactly one is false).After checking for sign it removes negative signs if present.

  The division is performed using repeated subtraction. It appends each digit of the dividend from left to right to a temporary string "temp" and when this temp is greater than the divisor then it keeps subtracting the divisor from it as many times as possible. The number of subtractions at each step are added to the quotient which is the final result. Leading zeros are removed from the result before returning.
  If divisor is zero then the method throws an `ArithmeticException` to prevent division by zero. Prefixes the sign to the final result if needed and returns a new `AInteger` instance containing the final result.

UML Diagram for AInteger



4 AFloat.java

The `AFloat` class provides functionality for representing and manipulating very large floating point numbers with arbitrary precision. It stores the number as a string and supports arithmetic operations like addition, subtraction, multiplication and division with proper handling of signs and decimal places.

Constructors

- `AFloat()`
  Default constructor AFloat() that initializes the instance with value "0".
- `AFloat(String str)`
  Constructor AFloat(String s) that initializes the instance by the number whose string representation is given by 's'.
- `AFloat(AFloat other)`
  Copy constructor that creates an instance of AFloat.

Methods

- **removezeroesatstart(String str)**
  This method removes the leading zeroes if present in the string. If the input number string starts with a decimal point, then "0" is added before the point.
- **removezeroesatend(String str)**
  This method removes unnecessary trailing zeroes after the decimal point. If all digits after decimal point are zero,then one zero is added after the decimal.
- **Sign(String str1,String str2)**
  Determines the sign of the result when subtracting two non-negative floating point strings. If `str1` is less than `str2`, then the method returns `"-"` indicating result as negative, otherwise returns an empty string. This method is used in subtraction.
- **less(String str1, String str2)**
  Determines the lesser string in the given two. This method is used in division to know which string is lesser in the given two strings.
- **decimalindex(String str)**
  This method returns the decimal point index in the string. If decimal is not present it returns the length of string.
- **addzeroes(int index, String str)**
  If decimal is not present in the string, this method appends ".0" to the string.

- **thirtydecimals(String str)**
  This method truncates the decimal part to at most 30 digits after decimal point and returns the result string.

- **parse(String str)**
  This static method creates and returns a new `AFloat` instance initialized with the given string `str`. It is equivalent to calling a constructor with a string.

- **toString()**
  Returns the internal string representation of the AFloat object.

- **addition(String str1,String str2)**
  Performs digit-wise addition of two non negative integer strings which are aligned properly. The method iterates from right to left carrying the carry. This logic is same as manual addition. The final result is returned as a string after removing the leading zeroes.

- **subtract(String str1,String str2)**
  Performs the digit-wise subtraction of two non-negative integer strings. If first number `str1` is smaller than `str2`,then it swaps the two numbers and stores the negative sign. It handles borrowing if needed and returns the result as a string after removing the leading zeroes.

- **multiply(String str1,String str2)**
  Performs the digit-wise multiplication of two numeric strings. For each digit in `str2` starting from right, it multiplies with every digit in `str1` and forms partial products. These partial products are shifted properly by appending zeroes and added using `addition` method. It returns final result as a string after removing the leading zeroes.

- **AFloat add(AFloat other)**
  Adds the current `AFloat` to another instance. It aligns the decimals of both numbers and determines whether the operation should be an addition or subtraction based on the signs of two numeric strings and calls the required function.Adds the decimal correctly to the string in the correct position.Prefixes the correct sign to final result and returns as a new `AFloat` instance containing sum with 30 digits after decimal.

- **AFloat sub(AFloat other)**
  Subtracts another `AFloat` instance from the current one. It aligns the decimals and handles the signs of both strings. Determines whether to call addition or subtraction based on the signs of two numbers.Adds the decimal correctly to the string in the correct position. Prefixes the correct sign to the final result and returns a new `AFloat` instance containing difference with 30 digits after decimal.

- **AFloat mul(AFloat other)**
  Multiplies the current `AFloat` with another instance. This method removes the decimal points, performs multiplication using `multiply` method, and then re-inserts the decimal at the correct position. Sign is handled using `XOR` logic. Prefixes the sign to the final result if needed and returns the result as a new `AFloat` instance containing product with 30 digits after decimal.

- **AFloat div(AFloat other)**
  Divides the current `AFloat` by another instance and returns the quotient as a new `AFloat` instance. It removes the decimal points and balances the digits after decimal. Division is performed using repeated subtraction logic similar to integer division. It adds precision up to 30 decimal places. If divisor is zero then method throws an `ArithmeticException` to prevent division by zero.

AFloat
+ strvalue : String
+ AFloat() + AFloat(str : String) + AFloat(other : AFloat) + parse(str : String) : AFloat + toString() : String + add(other : AFloat) : AFloat + sub(other : AFloat) : AFloat + mul(other : AFloat) : AFloat + div(other : AFloat) : AFloat + removezeroesatstart(str : String) : String + removezeroesatend(str : String) : String + Sign(str1, str2 : String) : String + less(str1, str2 : String) : String + decimalindex(str : String) : int + addzeroes(index : int, str : String) : String + thirtydecimals(str : String) : String + addition(str1, str2 : String) : String + subtract(str1, str2 : String) : String + multiply(str1, str2 : String) : String

5 MyInfArith.java

The `MyInfArith` class contains the `main()` method and acts as the entry point for the program. It is responsible for taking input arguments from the command line and calling the appropriate arithmetic operations based on the data type and operation.

Method

- **public static void main(String[] args)**
  This is the main method of the program. It takes exactly four command-line arguments in the following order:
  - `args[0]`: Specifies the type . It can be `"int"` for integer or `"float"` for floating-point.
  - `args[1]`: Specifies the operation to perform. It can be `"add"`, `"sub"`, `"mul"`, or `"div"`.
  - `args[2]`: The first operand (number in string format).
  - `args[3]`: The second operand (number in string format).

  The method first checks if the correct number of arguments is provided. If not, it prints an error message.

  If the type is `"int"`, it verifies that both numbers do not contain decimal points. Then it creates instances of `AInteger` using the input strings and performs the required operation by calling the corresponding method

```
int-add
    sub
    mul
    div
```

  Final result is printed to the console.

  If the type is `"float"`, it directly creates instances of `AFloat` using the input strings and performs the required operation by calling the corresponding method.

```
float-add
      sub
      mul
      div
```

The result is printed to the console.

If the operation is not one of the allowed operations, an error message is printed indicating that the operation is not supported.

Exmaple

javac MyInfArith.java

Input: java MyInfArith float add 84486723.420039 70974199.843732
Output: 155460923.263771

Input: java MyInfArith float mul 6400251.9377695 2326541.6827934
Output: 14890452913599.9717457253213

Input: java MyInfArith int sub 311651167400659980649551275857757745242300346381144446453884008
Output: -54628730626339781337950941125431

6 Limitations

- The division logic uses repeated subtraction, which can be slow for large inputs and when dividing by small numbers.

- It can perform four operations Addition,subtraction,Multiplication.Division. Other than this it cannot perform any other operations.

7 Verification

Using Java

- **Integer Mutiplication**

  ```
  Input:  java MyInfArith int mul 1434416316044592994268069731232223017167694823904478474013730519
  Output:  330162008905899217578310782382075660760972861550182008086155118
  ```

- **Integer division**

  ```
  Input:  java MyInfArith int div 879272636528306057983395052167721149383525361708964745499835 8
  Output:  17804979
  ```

- **Float Division**

  ```
  Input:  java MyInfArith float div 879272636528306057983395052167721 1.0
  4938352536170896474549983 58
  Output:  17804979.0914699893029611595200878785 33
  ```

Using ant makefile

command: ant run -Darg1=< *type* > -Darg2=< *operation* > -Darg3=< *operand*1 > -Darg4=< *operand*2 >

- Input: ant run -Darg1=float -Darg2=sub -Darg3=840196454.51725 -Darg4=127609490.81442
  Output: 3.22160363453775211 1008551505615
- Input: ant run -Darg1=int -Darg2=div -Darg3=236500782249129494973109332402 50
  -Darg4=429397832624671137983863844014 98
  Output: 665898614873800632956973176417 48

Using python script

command: `python3 execute_java.py <type> <operation> <operand1> <operand2>`

- ```
  Input:  python3 execute_java.py float add 84486723.420039 70974199.843732
  Output:  155460923.263771
  ```

- ```
  Input:  python3 execute_java.py float mul 6400251.9377695 2326541.6827934
  Output:  14890452913599.9717457253213
  ```

Using JAR file

- ```
  java -cp arbitraryarithmetic/aarithmetic.jar:javasrc MyInfArith
  float mul 6400251.9377695 2326541.6827934
  Output:  14890452913599.9717457253213
  ```

8 Key Learnings:

- Java oops
- Ant makefile
- Python scripting
- Git commands
- Docker
  ```