ARBITRARY PRECISION ARITHMETIC

CS24BTECH11026

May 2, 2025

Introduction

This project is about implementing an **arbitrary precision arithmetic** library in Java. The main goal is to implement accurate arithmetic operations on extremely large integers and floating point numbers without encountering overflow or round off errors which occurs generally. In programming, data types such as int or double have fixed sizes, which restrict the range and precision of numerical computations. To overcome these limitations, this project represents numbers as strings and implements arithmetic operations manually, allowing for complete control over precision and scale.

The library supports basic **operations addition, subtraction, multiplication, and division** for both integer and floating-point types. All operations are implemented in such a way that they preserve the full information of the numbers involved, avoiding any loss of precision due to rounding. For floating-point numbers, up to 30 digits of decimal precision are supported. and different ways to run this library by using python script and Ant make file.

Project Structure

```
SDF_PROJECT/
      Java_files/
              arbitraryarithmetic/AFloat.java
                                  AInteger.java
      MyInfArith.java
      arbitraryarithmetic/
                          aarithmetic.jar
      build/
           arbitraryarithmetic/AInteger.class
                               AFloat.class
                               MyInfArith.class
      build.xml
      PythonScript.py
```

Integer Class(AInteger.java):

AInteger class implements arbitrary precision for integers and implements methods for addition,subtraction,mutliplication and division even for large integers.

Constructors:

- Default constructor AInteger() that initializes the instance with value 0.
- Constructor AInteger(String s) that initializes the instance by the number whose string representation is given by 's'. Eg: AInteger("- 34534536454");
- Copy constructor that creates an instance of AInteger

Methods:

Functions used to implement AInteger Class

- **public AInteger sub(AInteger s):**
  Subtracts the given `AInteger` from the current instance and returns a new `AInteger` representing the result. Internally uses the `Sub` method on string values.
- **public AInteger mul(AInteger s):**
  Multiplies the current instance with the given `AInteger` and returns a new `AInteger` representing the product. Uses the `Multiply` method on string representations.
- **public AInteger div(AInteger s):**
  Divides the current instance by the given `AInteger` and returns a new `AInteger` representing the quotient. Relies on the `Division` method for computation.
- **public int compare(String s1, String s2):**
  Compares two non-negative integer strings `s1` and `s2`. Returns `1` if `s1 > s2`, `-1` if `s1 < s2`, and `0` if both are equal. It removes leading zeros before comparison.
- **public String RemoveStartZero(String s):**
  Removes leading zeros from the given string representation of a number. If the entire string is made of zeros, it returns `"0"`.
- **public static AInteger parse(String s):**
  A static method that takes a string `s` and returns a new `AInteger` instance initialized with that string.

- **public AInteger(AInteger other):**
  Copy constructor that creates a new `AInteger` with the same value as another `AInteger` instance.
- **public AInteger(String s):**
  Constructor that initializes an `AInteger` object with a string value representing a large integer.
- **public AInteger():**
  Default constructor that initializes the value to `"0"`.

UML Table(Integer class

AInteger
- value : String
+ AInteger() + AInteger(value : String) + AInteger(other : AInteger) + parse(value : String) : AInteger + add(other : AInteger) : AInteger + sub(other : AInteger) : AInteger + mul(other : AInteger) : AInteger + div(other : AInteger) : AInteger + Add(a : String, b : String) : String + Sub(a : String, b : String) : String + Multiply(a : String, b : String) : String + Division(a : String, b : String) : String + compare(a : String, b : String) : int + RemoveStartZero(s : String) : String

Float Class(AFloat.java):

AFloat class implements arbitrary precision for floating point numbers and implements methods for addition,subtraction,mutliplication and division even for large numbers with correct precision rounded off to 30 decimals after decimal point.

- **public AFloat add(AFloat s)**
  Adds the given `AFloat` to the current instance and returns a new `AFloat` representing the result. Internally uses the `Add` method on string values.
- **public AFloat sub(AFloat s)**
  Subtracts the given `AFloat` from the current instance and returns a new

`AFloat` representing the result. Internally uses the `Sub` method on string values.

- **public AFloat mul(AFloat s)**
  Multiplies the current instance with the given `AFloat` and returns a new `AFloat` representing the product. Uses the `Multiply` method on string representations.

- **public AFloat div(AFloat s)**
  Divides the current instance by the given `AFloat` and returns a new `AFloat` representing the quotient. Relies on the `Division` method for computation.

- **public String Add(String Str1, String Str2)**
  Adds two string representations of numbers and returns the result as a string.

- **public String RemoveStartZero(String s)**
  Removes leading zeros from the given string representation of a number. If the entire string is made of zeros, it returns `"0"`.

- **public int compare(String Str1, String Str2)**
  Compares two string representations of numbers numerically. Returns `1` if `Str1 > Str2`, `-1` if `Str1 < Str2`, and `0` if both are equal.

- **public String Multiply(String Str1, String Str2)**
  Multiplies two string representations of numbers and returns the result as a string.

- **public String MultiplyF(String Str1, String Str2)**
  Multiplies two floating-point string representations and returns the result as a string.

- **public String Division(String Str1, String Str2)**
  Divides two string representations of numbers and returns the quotient.

- **public String RemoveEndZero(String Str)**
  Removes trailing zeros from a string representation of a number with a decimal point.

- **public String Sub(String Str1, String Str2)**
  Subtracts two string representations of numbers and returns the result as a string.

- **public String AddF(String Str1, String Str2)**
  Adds two floating-point string representations and returns the result as a string.

- **public String SubF(String Str1, String Str2)**
  Subtracts two floating-point string representations and returns the result as a string.

- **public String DivisionF(String Str1, String Str2)**
  Divides two floating-point string representations and returns the result as a string.
- **public String RoundOffTo30(String Str)**
  Rounds a floating-point number represented as a string to 30 decimal places.
- **public static AFloat parse(String s)**
  A static method that takes a string `s` and returns a new `AFloat` instance initialized with that string.
- **public AFloat(AFloat other)**
  Copy constructor that creates a new `AFloat` with the same value as another `AFloat` instance.
- **public AFloat(String s)**
  Constructor that initializes an `AFloat` object with a string value representing a floating-point number.
- **public AFloat()**
  Default constructor that initializes the value to `"0.0"`.

UML Table(Float class):

AFloat
- value : String
+ AFloat() + AFloat(value : String) + AFloat(other : AFloat) + parse(value : String) : AFloat + add(other : AFloat) : AFloat + sub(other : AFloat) : AFloat + mul(other : AFloat) : AFloat + div(other : AFloat) : AFloat + AddF(a : String, b : String) : String + SubF(a : String, b : String) : String + MultiplyF(a : String, b : String) : String + DivisionF(a : String, b : String) : String + RoundOffTo30(s : String) : String + RemoveStartZero(s : String) : String + RemoveEndZero(s : String) : String + compare(a : String, b : String) : int

MyInfArith(Main file):

MYInfArith is the main file to perform arithmetic operations .It takes input through command line arguements with supporting four operations Addition,Subtraction,Multiplication and Division for both Integer and floating point numbers

Input Format:

Format: java MyInfArith $<$ *type* $><$ *operation* $><$ *operand*1 $><$ *operand*2 $>$

Arguements

- $<$ *type* $>$: Specifies the data type for arithmetic.
  - `"int"` – use `AInteger` class for operations on large integers.
  - `"float"` – use `AFloat` class for operations on large floating-point numbers.
- $<$ *operation* $>$: The arithmetic operation to perform.
  - `"add"` – addition
  - `"sub"` – subtraction
  - `"mul"` – multiplication
  - `"div"` – division
- $<$ *operand*1 $>$: The first number (as a string). Must be a valid integer or floating-point number depending on the type.
- $<$ *operand*2 $>$: The second number (as a string). Must be a valid integer or floating-point number depending on the type.

Based on the type given in arguements two objects(instances) are created using third and fourth arguements respectively, and performs the operation specified in the second arguement,prints the result as a string on screen if the number of digits after the decimal point exceeds 30,it will print upto 30 decimal points

Examples

- Integer addition:

```
java MyInfArith int add 123456789123456789 987654321987654321
```

- Floating-point multiplication:

```
java MyInfArith float mul 12345.6789 987.654321
```

Limitations of Library:

- For exytremely large numbers the multiplication and division operations takes time to give result because of manual digit by digit operation in methods
- It Supports only for four operations Addition,Subtraction,Division,Multiplication,does not support for higher operations like suqare,max,min functions
- Does not support for inputing scientific notation of large numbers and cannot handle other strings containg alphabets and different characters

Verification:

Using Java:

command: java MyInfArith $< type >< operation >< operand1 >< operand2 >$

- Ex: Input: java MyInfArith int add 2365007822491294949731093324025042939783262467113798386384401498
  Output: 665898614873800632956973176417 48
- Ex: Input: java MyInfArith int div 879272636528306057983395052167721149383525361708964745499835 8
  Output: 17804979
- Ex: Input: java MyInfArith float div 2.88832837283283 0.00000
  Output: Exception in thread "main" java.lang.ArithmeticException: Division by Zero Error
- Ex: Input: java MyInfArith float div 879272636528306057983395052167721149383525361708964745499835 8.0
  Output: 17804979.09146998930296115952008787853 3

Using Python Script:

command: python3 PythonScript.py $< type >< operation >< operand1 >$ $< operand2 >$ (ubuntu)

- Ex: Input: python3 PythonScript.py float sub 840196454.51725 712586963.70283
  Output: 127609490.81442
- Ex: Input: python3 PythonScript.py float mul 6400251.9377695 2326541.6827934
  Output: 14890452913599.9717457253213

Using Ant:

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

- Ex: Input: ant run -Darg1=float -Darg2=div -Darg3=244727.15202 -Darg4=75964.3891
  Output: 3.2216036345377521110855150561 5
- Ex: Input: ant run -Darg1=int -Darg2=div -Darg3=22 -Darg4=125
  Output: 0

Using Jar file:

command: java -cp $Java_files$: $arbitraryarithmetic/aarithmetics.jar$ MyInfArith < *type* > < *operation* > < *operand*1 > < *operand*2 >

- Ex: Input: java -cp $Java_files$: $arbitraryarithmetic/aarithmetic.jar$ MyInfArith int sub 3116511674006599806495512758577 57745 242300346381144446453884008
  Output: -546287306263397813379509411254 31

Key Learnings:

- Java oops,creating packages
- Ant Make File
- Jar File
- Git commands and pushing into github
- Running Python script using os module
- Docker