SDF Project

Jagadeesh Merugumala

May 2025

1 Introduction

This project develops an arbitrary-precision arithmetic system in Java to perform calculations on large integers and high-precision floating-point numbers, surpassing the limitations of standard Java data types such as `int`, `long`, and `double`. Two core classes were implemented:

- `Ainteger`: Handles arithmetic operations (addition, subtraction, multiplication, division) on integers of arbitrary size using string-based digit manipulation.
- `Afloat`: Extends `Ainteger` to support floating-point arithmetic with precision up to 30 decimal places, managing decimal alignment, normalization, and signs.

These classes provide a modular, precise, and reliable framework for applications in scientific, financial, or mathematical domains requiring high-precision computations without overflow or rounding errors.

2 Arithmetic Operations in `Ainteger`

The `Ainteger` class enables arithmetic operations on arbitrarily large integers by manipulating digits as strings, overcoming the size constraints of standard integer types.

Addition

The addition operation processes two integers digit by digit from right to left, handling carries. For operands with opposite signs, the operation is redirected to subtraction with adjusted signs. When signs are the same, digits are added with carry propagation, and the result is reversed, normalized, and assigned the appropriate sign.

Key points:

- Opposite signs trigger subtraction with sign adjustment.

- Same-sign addition processes digits with carry handling.
- Result is normalized and signed based on operands.

Subtraction

Subtraction compares operand magnitudes using the `compareTo()` method. For different signs, it converts to addition. For same signs, the `abssub()` helper function performs digit-by-digit subtraction, managing borrows. The result is normalized, and its sign is set according to the operands.

Key points:

- Different signs convert subtraction to addition.
- Same-sign subtraction uses digit-by-digit borrow handling.
- Result is normalized with correct sign.

Multiplication

Multiplication employs the schoolbook algorithm, multiplying each digit of one operand by each digit of the other to generate partial products. These are padded with zeros based on position and summed using the `add()` method. The final result is normalized, with its sign determined by the operand signs.

Key points:

- Schoolbook method computes partial products.
- Products are zero-padded and summed via `add()`.
- Result is normalized with appropriate sign.

Division

Division uses long division, processing the dividend digit by digit and building the quotient by repeatedly subtracting the divisor from the running remainder. Division by zero throws an `ArithmeticException`. The quotient is returned as a new `Ainteger` object.

Key points:

- Long division builds quotient digit by digit.
- Remainder is updated via repeated subtraction.
- Throws `ArithmeticException` for zero divisor.

3 Arithmetic Operations in `Afloat`

The `Afloat` class supports arbitrary-precision floating-point arithmetic, leveraging `Ainteger` for integer-level operations. It ensures precision up to 30 decimal places by managing decimal alignment, normalization, and signs.

Addition

Addition aligns the decimal points of both operands by padding the shorter operand with zeros. The decimals are removed, and the operation is performed using `Ainteger.add()`. The result is converted back to a floating-point number, normalized, and rounded to 30 decimal places.

Key points:

- Aligns decimals by padding with zeros.
- Uses `Ainteger.add()` for integer addition.
- Normalizes and rounds to 30 decimal places.

Subtraction

Subtraction aligns decimal points by padding with zeros, removes decimals, and uses `Ainteger.sub()` for the operation. The result is normalized and adjusted to maintain 30 decimal places of precision.

Key points:

- Aligns decimals with zero padding.
- Performs integer subtraction via `Ainteger.sub()`.
- Normalizes to 30 decimal places.

Multiplication

Multiplication calculates the total decimal places in both operands, removes decimals, and performs the operation using `Ainteger.mul()`. The decimal point is repositioned in the result based on the operand decimal counts, and the result is normalized to 30 decimal places.

Key points:

- Tracks decimal places and removes decimals.
- Uses `Ainteger.mul()` for integer multiplication.
- Adjusts decimal point and normalizes to 30 decimal places.

Division

Division scales the numerator by 10^{30} to preserve precision, removes decimals, and uses `Ainteger.div()`. The result is adjusted by placing the decimal point correctly and normalized to 30 decimal places.

Key points:

- Scales numerator by 10^{30} for precision.
- Performs integer division via `Ainteger.div()`.
- Normalizes and rounds to 30 decimal places.

4 MyInfArith Class

The `MyInfArith` class is a command-line utility for performing arbitrary-precision arithmetic using `Ainteger` or `Afloat`. It processes four arguments: `type` (`int` or `float`), `operation` (`add`, `sub`, `mul`, `div`), and two operands.

Functionality

For `int`, it creates `Ainteger` objects from the operands and executes the specified operation. For `float`, it uses `Afloat` objects. The result is printed to the console. Error handling catches invalid inputs, division by zero, or other exceptions, displaying appropriate error messages.

Usage

```
java MyInfArith <type> <operation> <operand1> <operand2>
```

Example: `java MyInfArith int add 12345 6789` performs integer addition and outputs the result.

5 Class Diagram

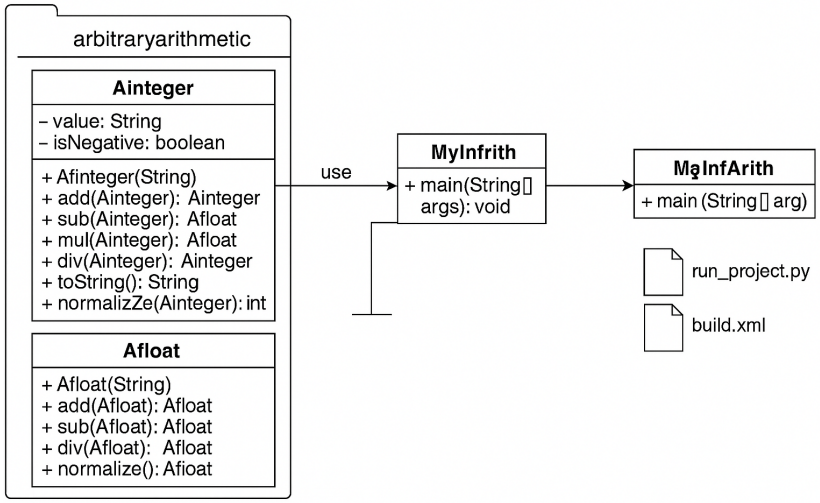

Figure 1: UML Diagram of Ainteger, Afloat, and MyInfArith Classes

6 Python Script: `run_project.py`

The `run_project.py` script automates the compilation and execution of the Java project, simplifying the testing process.

Functionality:

- Verifies that four command-line arguments are provided.
- Compiles `Afloat.java`, `Ainteger.java`, and `MyInfArith.java` to the `out/` directory using `javac`.
- Executes `MyInfArith` with the provided arguments using the `java` command.

Usage:

```
python run_project.py <type> <operation> <operand1> <operand2>
```

Example: `python run`$_{p}$*roject.pyintadd*123456789*compilesandrunstheadditionoperation.*

Error Handling: If arguments are missing, usage instructions are printed. Compilation or runtime errors are displayed in the terminal for debugging.

7 Build Automation with Apache Ant

An Apache Ant build.xml script automates the build process with three targets: compile, jar, and run.

Key points:

- compile: Creates out/ directory and compiles Java files to it.
- jar: Packages compiled classes into arithmetic.jar.
- run: Executes MyInfArith from the JAR, passing four command-line arguments (arg0 to arg3).

The project, named AArithmetic, defaults to the jar target, streamlining compilation, packaging, and execution.