CSC3301 ASSIGNMENT 1: UNDERSTANDING PROGRAM (SOURCE CODE) STRUCTURE

To understand source code structure by analyzing Abstract Syntax Trees

Group Members

Nebert Hampoko	**Luwi Kakoma**
Patrick Sakala	**Chomba A Masase**
Aaron Mbuzi	**Chansa Mwita**

Contents

Abstract Syntax Tree

What is an Abstract Syntax Tree ?

An abstract syntax tree (AST) is a language-agnostic, hierarchical representation of the elements comprising the source code of a computer program. With the use of an abstract syntax tree, it is possible to reproduce code with the same functionality of the original source code. This makes it possible to transform source code of one programming language to another programming language.

Tokens

What are Tokens ?

We have sets of rules that are used to describe a magnitude of code. Code can be broken down into simpler parts known as Lexems.

Lexems cannot be defined by these rules on a lower scope so we use lexical specification/ Lexical Analysis to describe these smaller components by looking at them individually.

These Lexems are commonly refered to as Tokens. They can be grouped as follows:

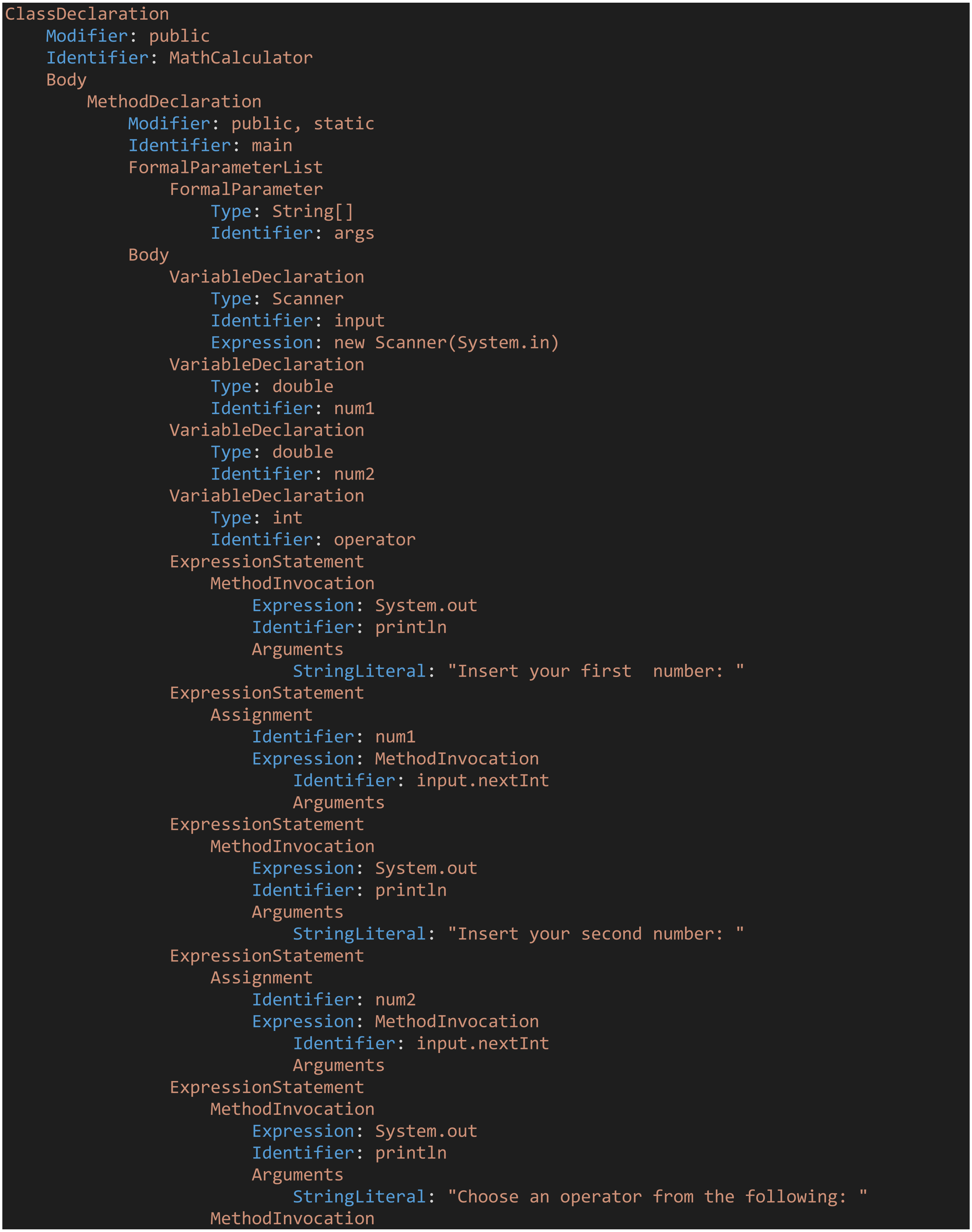
- Literals
- Identifiers
- Keywords
- Operators

The Abstract Syntax Tree is a graph of these tokens, specifically designed to preserve the context of each element and its attributes.

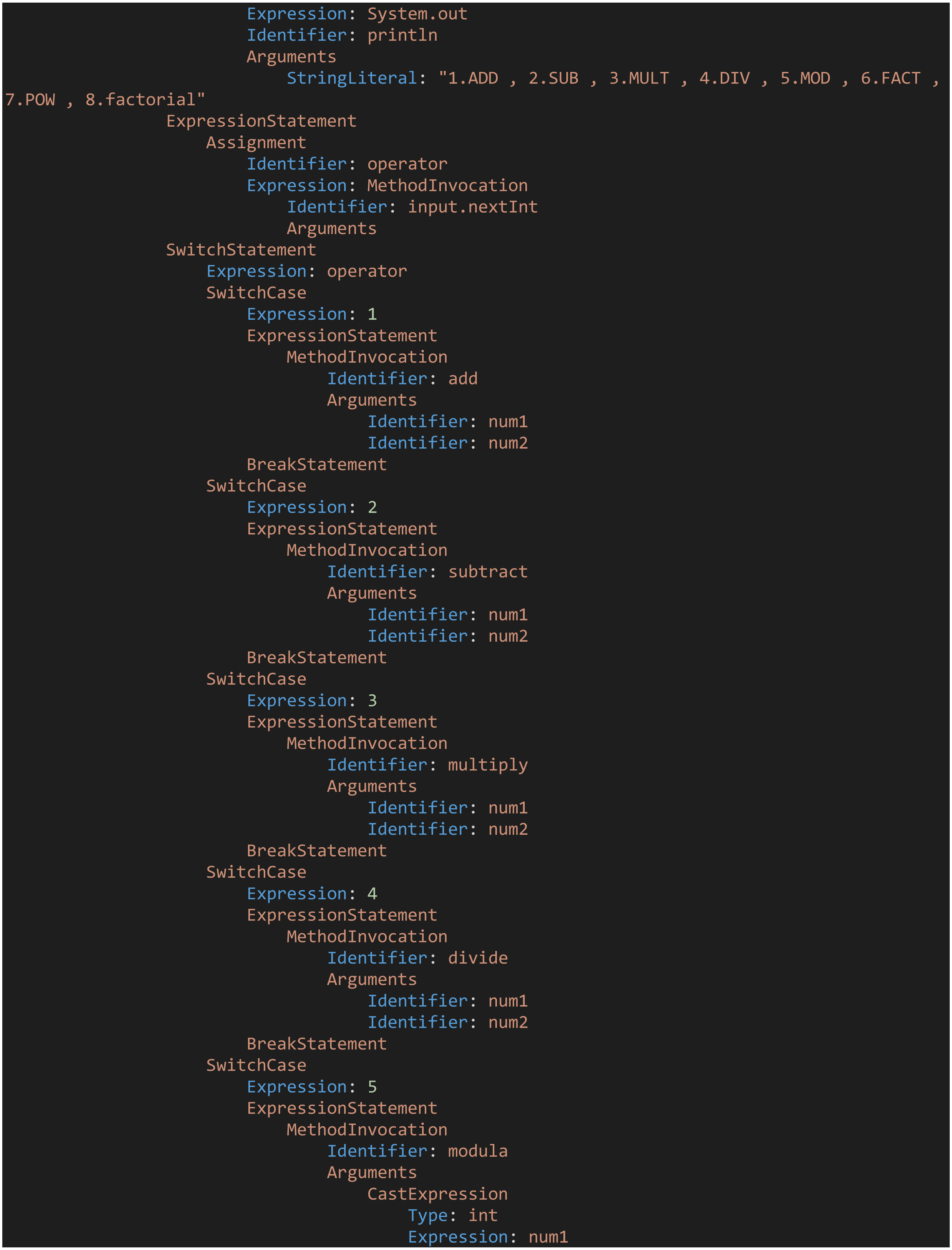
Objective

In the given Assignment, we were tasked to create a simple calculator and parse it to produce an abstract syntax tree and to identify its tokens.

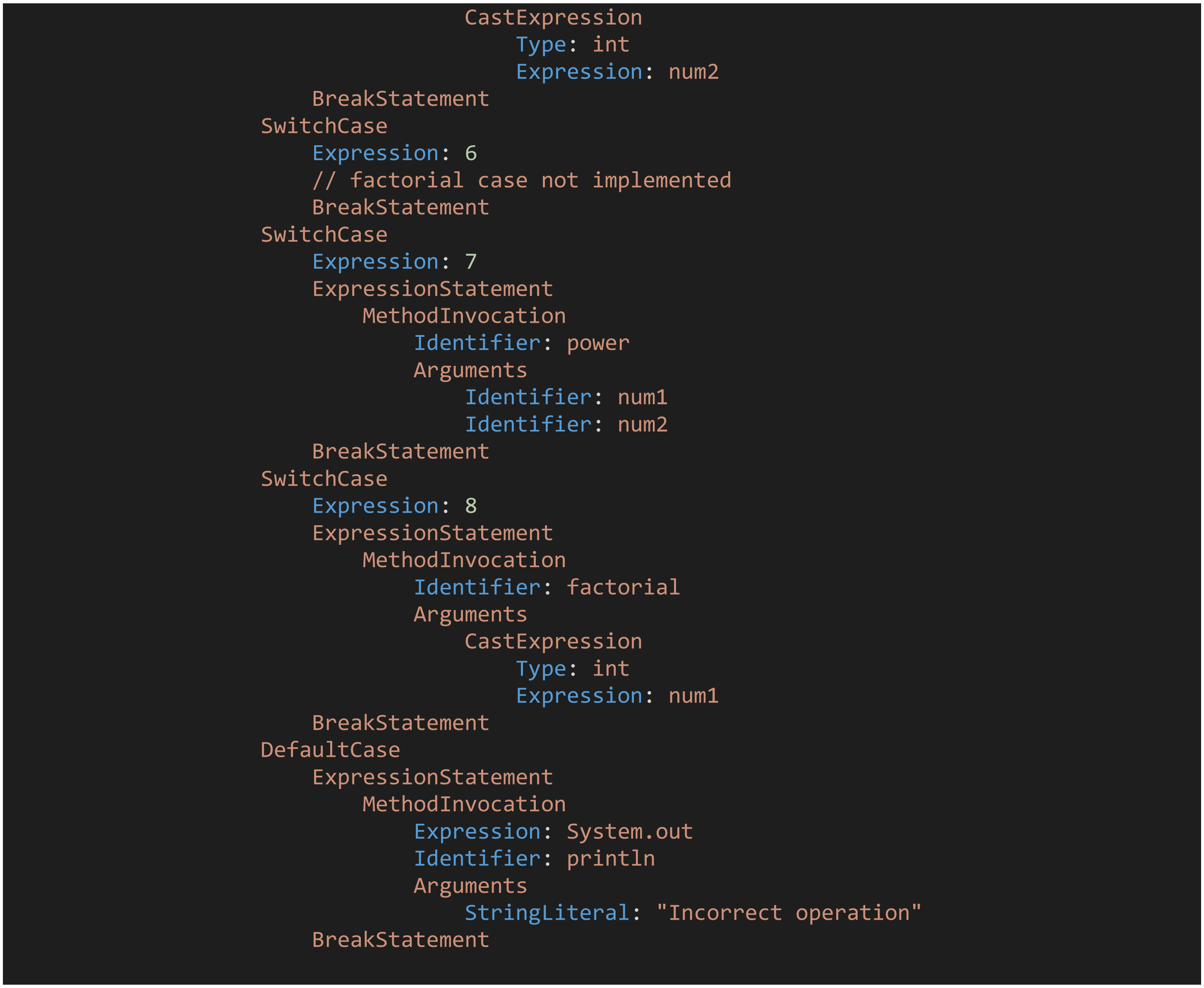
Below is our Abstract Syntax tree created in a text format.



```
ClassDeclaration
    Modifier: public
    Identifier: MathCalculator
    Body
        MethodDeclaration
            Modifier: public, static
            Identifier: main
            FormalParameterList
                FormalParameter
                    Type: String[]
                    Identifier: args
            Body
                VariableDeclaration
                    Type: Scanner
                    Identifier: input
                    Expression: new Scanner(System.in)
                VariableDeclaration
                    Type: double
                    Identifier: num1
                VariableDeclaration
                    Type: double
                    Identifier: num2
                VariableDeclaration
                    Type: int
                    Identifier: operator
                ExpressionStatement
                    MethodInvocation
                        Expression: System.out
                        Identifier: println
                        Arguments
                            StringLiteral: "Insert your first  number: "
                ExpressionStatement
                    Assignment
                        Identifier: num1
                        Expression: MethodInvocation
                            Identifier: input.nextInt
                            Arguments
                ExpressionStatement
                    MethodInvocation
                        Expression: System.out
                        Identifier: println
                        Arguments
                            StringLiteral: "Insert your second number: "
                ExpressionStatement
                    Assignment
                        Identifier: num2
                        Expression: MethodInvocation
                            Identifier: input.nextInt
                            Arguments
                ExpressionStatement
                    MethodInvocation
                        Expression: System.out
                        Identifier: println
                        Arguments
                            StringLiteral: "Choose an operator from the following: "
                    MethodInvocation
```



```
                        Expression: System.out
                        Identifier: println
                        Arguments
                            StringLiteral: "1.ADD , 2.SUB , 3.MULT , 4.DIV , 5.MOD , 6.FACT ,
7.POW , 8.factorial"
                ExpressionStatement
                    Assignment
                        Identifier: operator
                        Expression: MethodInvocation
                            Identifier: input.nextInt
                            Arguments
                SwitchStatement
                    Expression: operator
                    SwitchCase
                        Expression: 1
                        ExpressionStatement
                            MethodInvocation
                                Identifier: add
                                Arguments
                                    Identifier: num1
                                    Identifier: num2
                        BreakStatement
                    SwitchCase
                        Expression: 2
                        ExpressionStatement
                            MethodInvocation
                                Identifier: subtract
                                Arguments
                                    Identifier: num1
                                    Identifier: num2
                        BreakStatement
                    SwitchCase
                        Expression: 3
                        ExpressionStatement
                            MethodInvocation
                                Identifier: multiply
                                Arguments
                                    Identifier: num1
                                    Identifier: num2
                        BreakStatement
                    SwitchCase
                        Expression: 4
                        ExpressionStatement
                            MethodInvocation
                                Identifier: divide
                                Arguments
                                    Identifier: num1
                                    Identifier: num2
                        BreakStatement
                    SwitchCase
                        Expression: 5
                        ExpressionStatement
                            MethodInvocation
                                Identifier: modula
                                Arguments
                                    CastExpression
                                        Type: int
                                        Expression: num1
```



```
                                    CastExpression
                                        Type: int
                                        Expression: num2
                        BreakStatement
                    SwitchCase
                        Expression: 6
                        // factorial case not implemented
                        BreakStatement
                    SwitchCase
                        Expression: 7
                        ExpressionStatement
                            MethodInvocation
                                Identifier: power
                                Arguments
                                    Identifier: num1
                                    Identifier: num2
                        BreakStatement
                    SwitchCase
                        Expression: 8
                        ExpressionStatement
                            MethodInvocation
                                Identifier: factorial
                                Arguments
                                    CastExpression
                                        Type: int
                                        Expression: num1
                        BreakStatement
                    DefaultCase
                        ExpressionStatement
                            MethodInvocation
                                Expression: System.out
                                Identifier: println
                                Arguments
                                    StringLiteral: "Incorrect operation"
                        BreakStatement
```

In our assignment with the math calculator we had created we found the the total number of tokens are **131**.

Types of tokens and how many there are of each type are as follows.

- Punctuation: 57
- Literals: 19
- Identifiers: 13
- Keywords: 19
- Operators: 23

However, you will notice that our Abstract syntax tree may not allow us to identify all the tokens and the number of tokens of each type. This is because we had used Prompt Engineering in conjunction with Artificial Intelligence to produce it.

We did not end there as we decided to also use the Legacy Java parser to produce a much bigger tree. We are not adding that tree directly into this report because it is too large. It contains approximately 750 lines. Instead we shall upload it to our GitHub Repository here MathCalculator Parsed Code, under the name ast.yml.

With the abstract syntax tree made by the java parser on our GitHub repository we are now able to see in detail all our tokens.

References

[1] Caupolican Diaz, gara news, Kyra Thompso, Steven Swiniarski ,” Abstract Syntax Tree”, https://www.codecademy.com/resources/docs/general/abstract-syntax-tree