RPN Parser

Lars Obist, Yuliya Litvin,
Niels Wennesheimer, Tabea Schuster

Solving a Mathematical Issue

usually: **operand** operator **operand**

Solving a Mathematical Issue

RPN: **operand operand** operator

RPN =
Reverse Polish Notation

mathematical notation:
uses *postfix notation* to portray expressions

operand operand operator

The Task

Converting infix expression into RPN notation by implementing a corresponding expression converter

The Language

```
Expressions         ::=    (AddExpression ”;”)* <EOF>

AddExpression       ::=    MulExpression(“+” MulExpression)*
                     |     MulExpression(“-” MulExpression)*

MulExpression       ::=    PrimaryExpression(“*” PrimaryExpression)*
                     |     PrimaryExpression(“/” PrimaryExpression)*

PrimaryExpression   ::=    <Number>
                     |     “-” PrimaryExpression
                     |     “(“ PrimaryExpression ”)”
```

How is the Reverse Polish Notation built?

Basic Example

3 + 4

operand operator operand

left operand right operand operator

3 4 +

Complex Example

How do we get from

5 + ((1 + 2) * 4) - 3

to

5 1 2 + 4 * + 3 -

5 + ((1 + 2) * 4) - 3

5 + ((1 2 +) * 4) - 3

5 + ((1 2 +) * 4) - 3

5 + (1 2 + 4 *) - 3

5 + (1 2 + 4 *) - 3

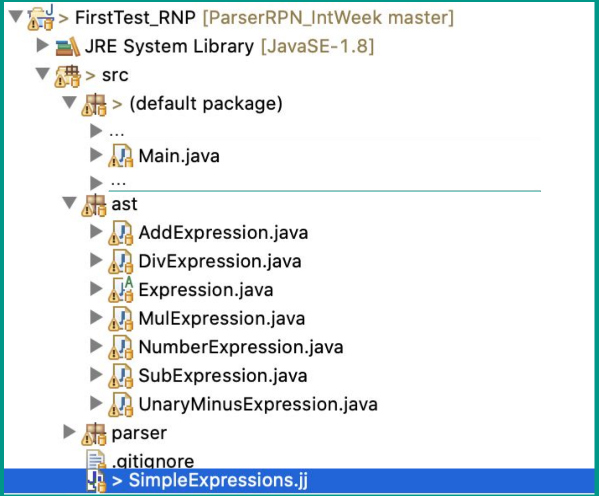
5 1 2 + 4 * + - 3

5 1 2 + 4 * + - 3

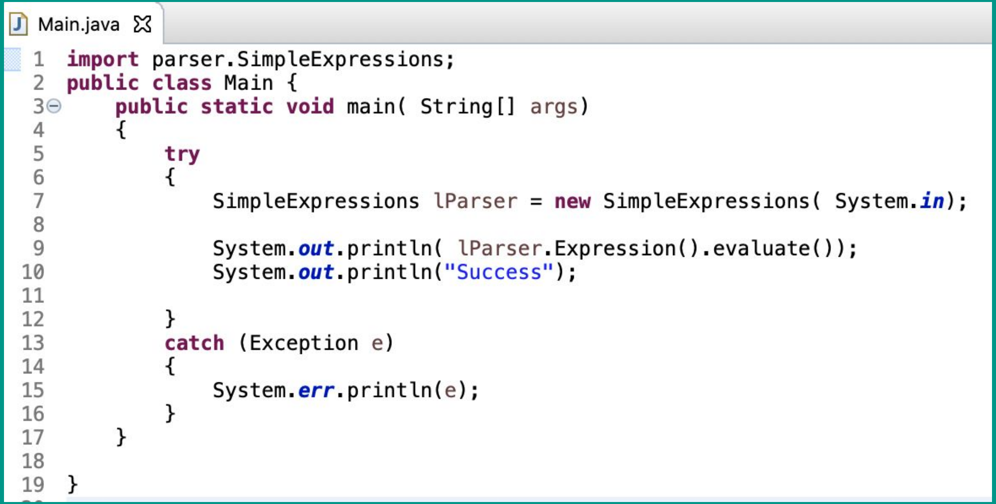
5 1 2 + 4 * + 3 -

The Code

General Structure



```
▼ > FirstTest_RNP [ParserRPN_IntWeek master]
  ▶ JRE System Library [JavaSE-1.8]
  ▼ > src
    ▼ > (default package)
      ▶ ...
      ▶ Main.java
      ▶ ...
    ▼ ast
      ▶ AddExpression.java
      ▶ DivExpression.java
      ▶ Expression.java
      ▶ MulExpression.java
      ▶ NumberExpression.java
      ▶ SubExpression.java
      ▶ UnaryMinusExpression.java
    ▶ parser
      .gitignore
      > SimpleExpressions.jj
```



Main.java

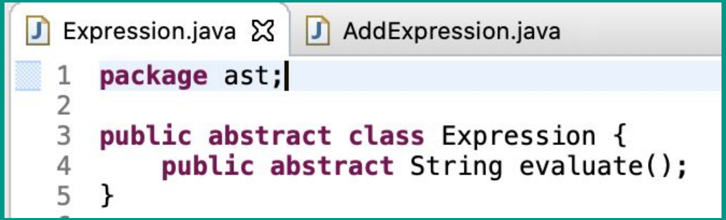
```java
import parser.SimpleExpressions;
public class Main {
    public static void main( String[] args)
    {
        try
        {
            SimpleExpressions lParser = new SimpleExpressions( System.in);

            System.out.println( lParser.Expression().evaluate());
            System.out.println("Success");

        }
        catch (Exception e)
        {
            System.err.println(e);
        }
    }

}
```

- used the pattern from the lectures, not the visitor pattern
- SimpleExpressions: Parser
- ast: contains expression from the Syntax-Tree

- Main: instantiates and calls the parsers evaluate methode

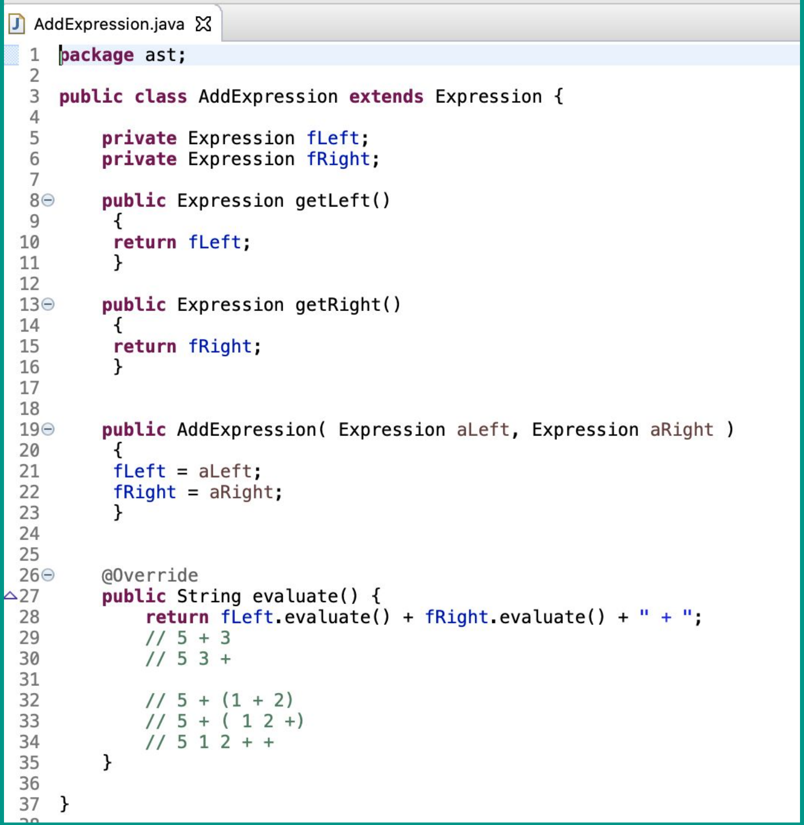
Expression-Classes

1.)



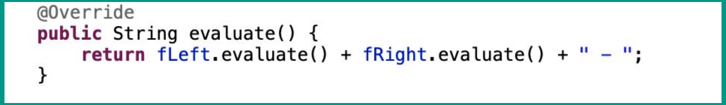
```java
package ast;

public abstract class Expression {
      public abstract String evaluate();
}
```

2.)



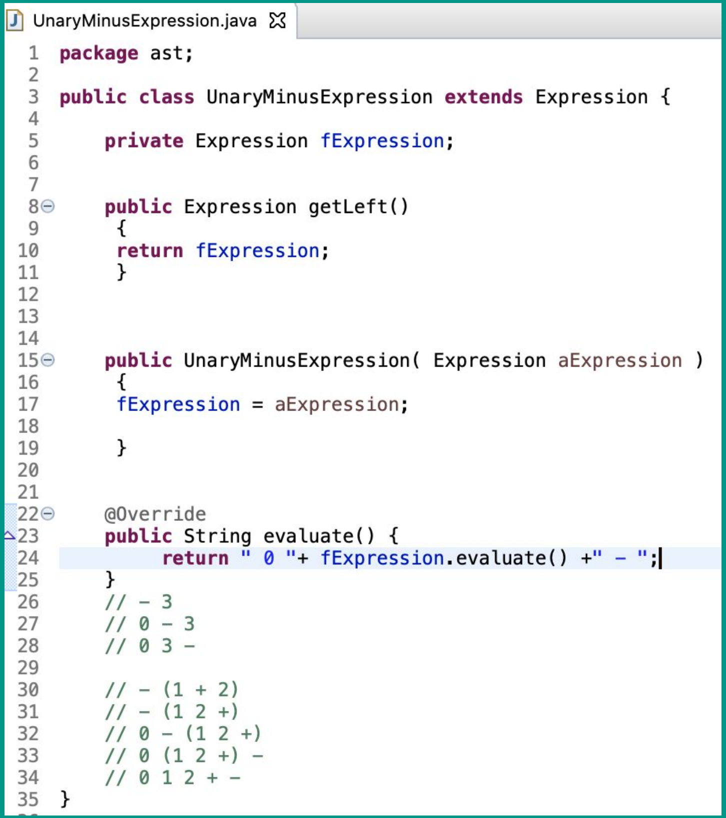
```java
package ast;

public class AddExpression extends Expression {

    private Expression fLeft;
    private Expression fRight;

    public Expression getLeft()
     {
     return fLeft;
     }

    public Expression getRight()
     {
     return fRight;
     }


    public AddExpression( Expression aLeft, Expression aRight )
     {
     fLeft = aLeft;
     fRight = aRight;
     }


    @Override
    public String evaluate() {
        return fLeft.evaluate() + fRight.evaluate() + " + ";
        // 5 + 3
        // 5 3 +

        // 5 + (1 + 2)
        // 5 + ( 1 2 +)
        // 5 1 2 + +
    }

}
```

3.) SubExpression etc. are similar



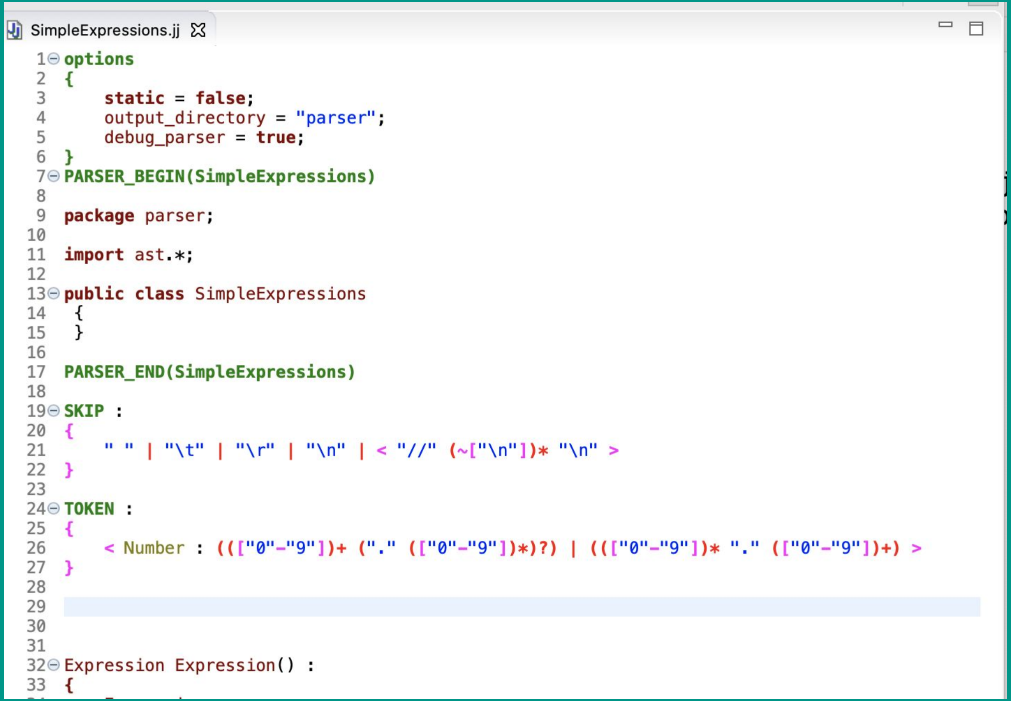
```java
@Override
public String evaluate() {
    return fLeft.evaluate() + fRight.evaluate() + " - ";
}
```

Expression-Classes



```java
package ast;

public class UnaryMinusExpression extends Expression {

    private Expression fExpression;


    public Expression getLeft()
     {
     return fExpression;
     }



    public UnaryMinusExpression( Expression aExpression )
     {
     fExpression = aExpression;

     }


    @Override
    public String evaluate() {
          return " 0 "+ fExpression.evaluate() +" - ";
    }
    // - 3
    // 0 - 3
    // 0 3 -

    // - (1 + 2)
    // - (1 2 +)
    // 0 - (1 2 +)
    // 0 (1 2 +) -
    // 0 1 2 + -
}
```

The Parser



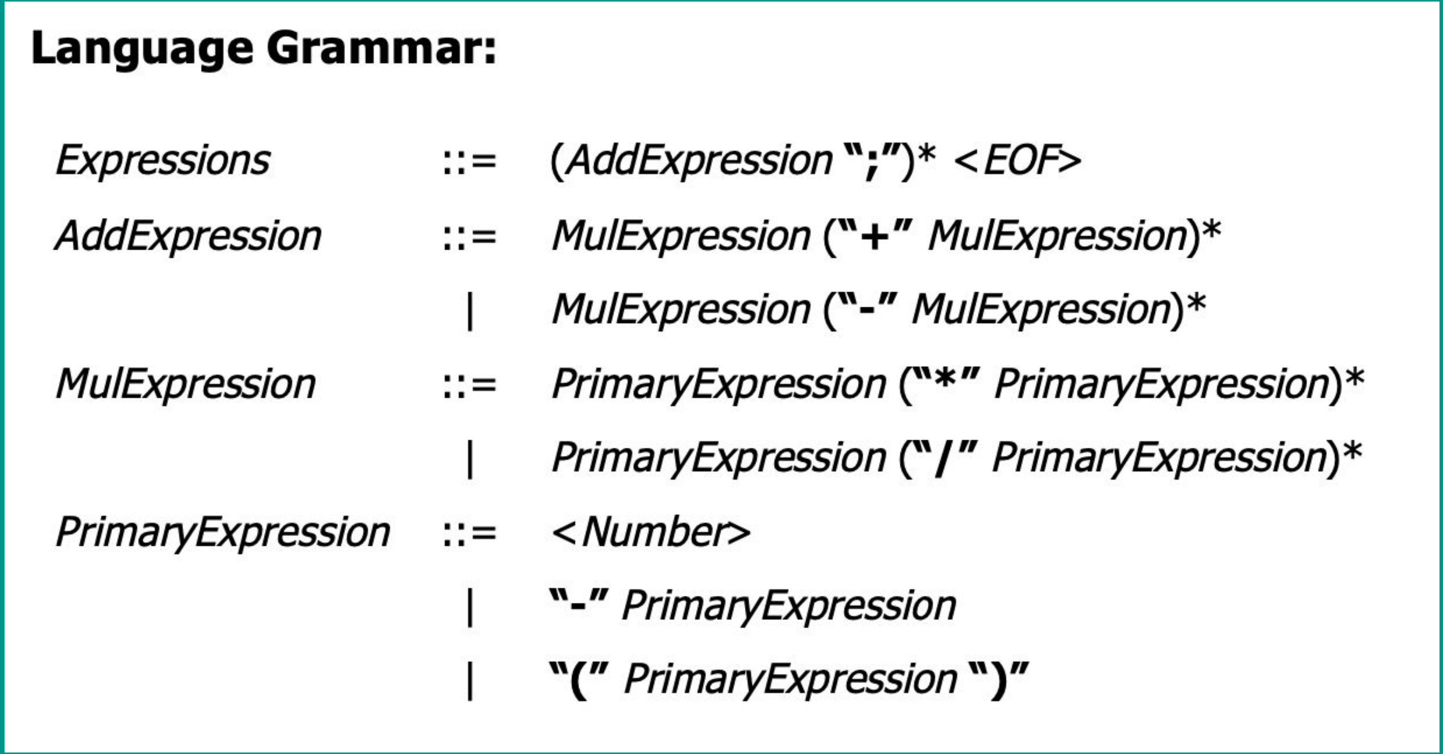
SimpleExpressions.jj

```
options
{
    static = false;
    output_directory = "parser";
    debug_parser = true;
}
PARSER_BEGIN(SimpleExpressions)

package parser;

import ast.*;

public class SimpleExpressions
 {
 }

PARSER_END(SimpleExpressions)

SKIP :
{
    " " | "\t" | "\r" | "\n" | < "//" (~["\n"])* "\n" >
}

TOKEN :
{
    < Number : ((["0"-"9"])+ ("." (["0"-"9"])*)?) | ((["0"-"9"])* "." (["0"-"9"])+) >
}




Expression Expression() :
{
```

- parser skeleton
- skip whitespace
- recognise <Number> as token

- the expressions developed from the grammar

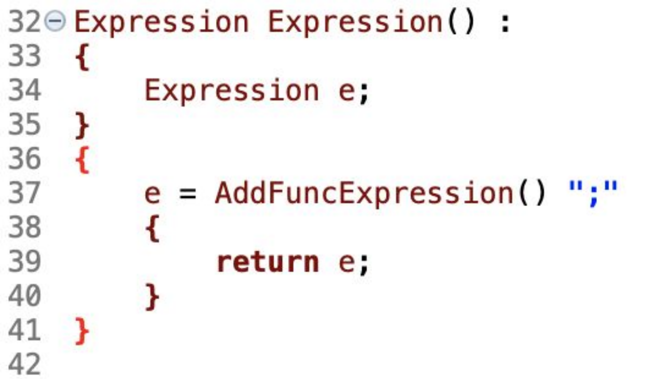
The Grammar



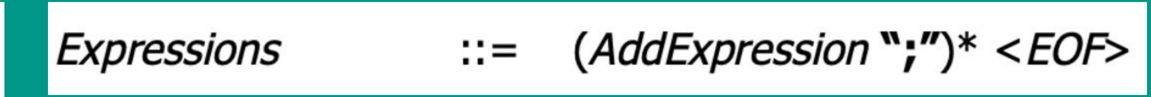
Language Grammar:

```
Expressions          ::=   (AddExpression “;”)* <EOF>
AddExpression        ::=   MulExpression (“+” MulExpression)*
                      |    MulExpression (“-” MulExpression)*
MulExpression        ::=   PrimaryExpression (“*” PrimaryExpression)*
                      |    PrimaryExpression (“/” PrimaryExpression)*
PrimaryExpression    ::=   <Number>
                      |    “-” PrimaryExpression
                      |    “(” PrimaryExpression “)”
```

The Parser

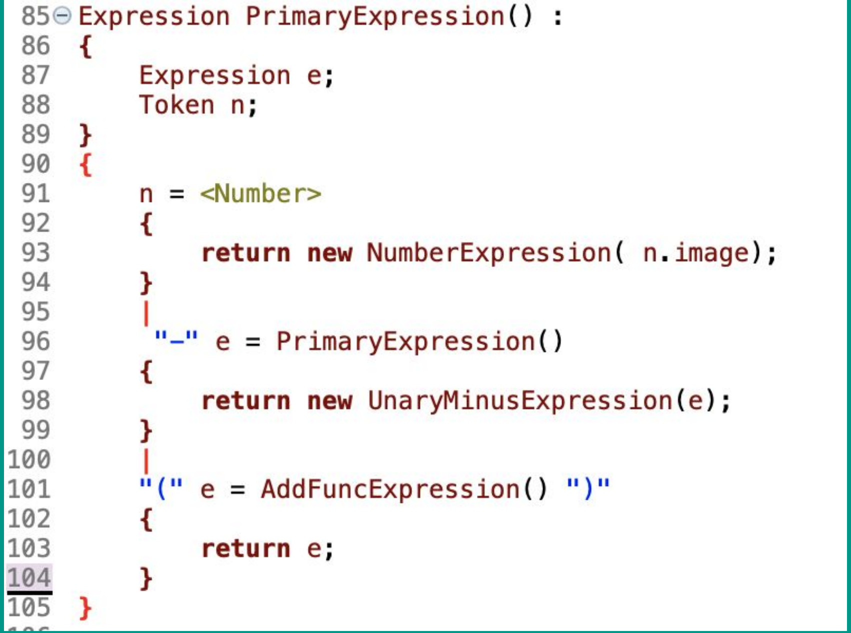


```
Expression Expression() :
{
    Expression e;
}
{
    e = AddFuncExpression() ";"
    {
        return e;
    }
}
```

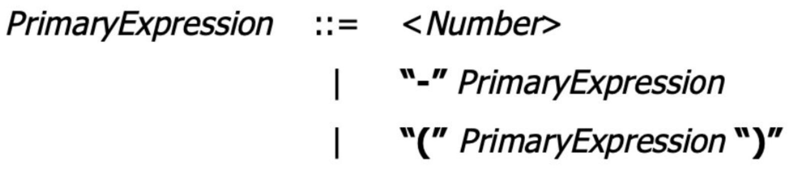


Expressions ::= (*AddExpression* “;”)* <*EOF*>

The Parser

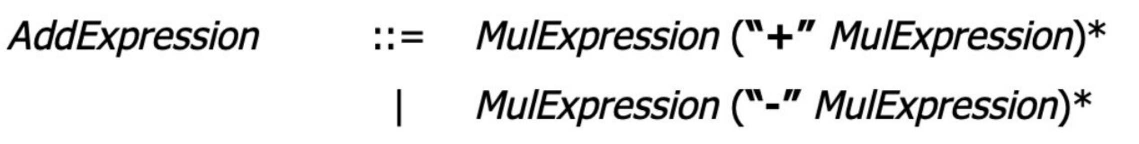


```
Expression PrimaryExpression() :
{
    Expression e;
    Token n;
}
{
    n = <Number>
    {
        return new NumberExpression( n.image);
    }
    |
     "-" e = PrimaryExpression()
    {
        return new UnaryMinusExpression(e);
    }
    |
    "(" e = AddFuncExpression() ")"
    {
        return e;
    }
}
```

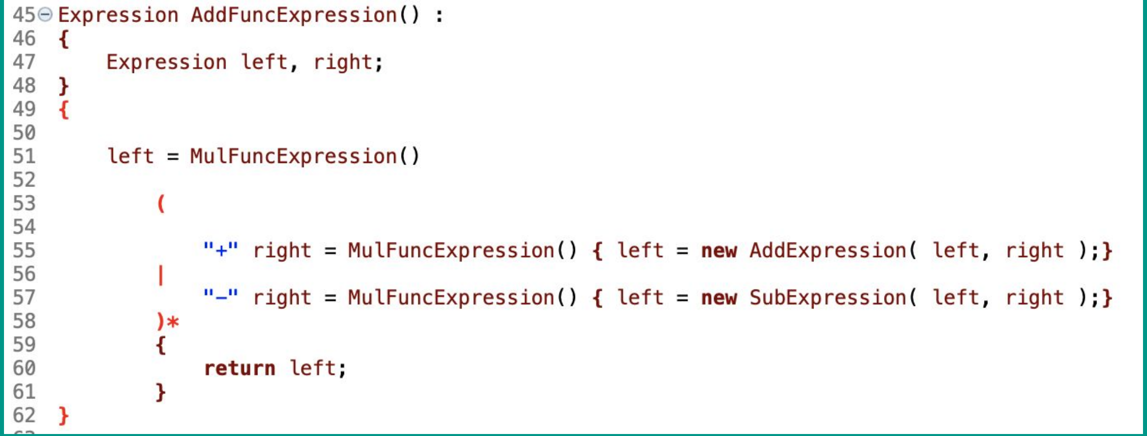


```
PrimaryExpression   ::=   <Number>
                      |   "-" PrimaryExpression
                      |   "(" PrimaryExpression ")"
```

The Parser

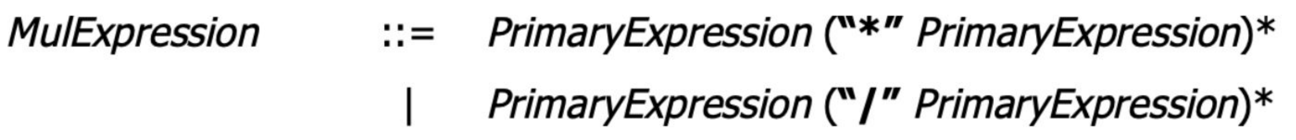


```
AddExpression    ::=   MulExpression (“+” MulExpression)*
                   |   MulExpression (“-” MulExpression)*
```

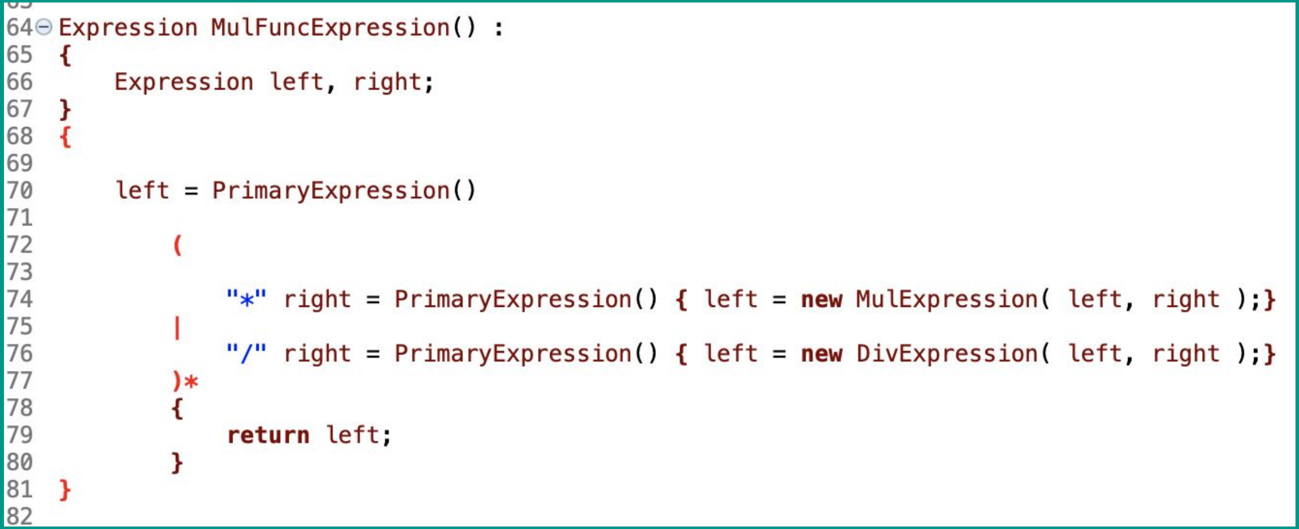


```
Expression AddFuncExpression() :
{
    Expression left, right;
}
{

    left = MulFuncExpression()

        (

            "+" right = MulFuncExpression() { left = new AddExpression( left, right );}
        |
            "-" right = MulFuncExpression() { left = new SubExpression( left, right );}
        )*
        {
            return left;
        }
}
```

The Parser

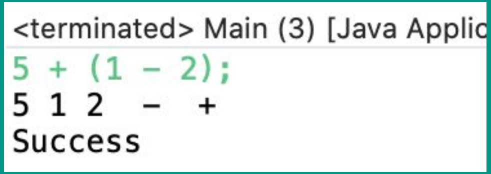


MulExpression ::= *PrimaryExpression* ("*" *PrimaryExpression*)*
| *PrimaryExpression* ("/" *PrimaryExpression*)*

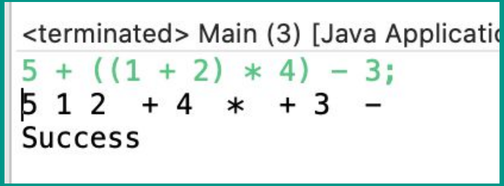


```
Expression MulFuncExpression() :
{
    Expression left, right;
}
{

    left = PrimaryExpression()

        (

            "*" right = PrimaryExpression() { left = new MulExpression( left, right );}
        |
            "/" right = PrimaryExpression() { left = new DivExpression( left, right );}
        )*
        {
            return left;
        }
}

```

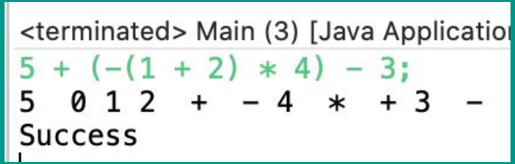
The Result



```
<terminated> Main (3) [Java Applic
5 + (1 - 2);
5 1 2  -  +
Success
```



```
<terminated> Main (3) [Java Applicati
5 + ((1 + 2) * 4) - 3;
5 1 2  + 4  *  + 3  -
Success
```



```
<terminated> Main (3) [Java Applicatio
5 + (-(1 + 2) * 4) - 3;
5  0 1 2  +  - 4  *  + 3  -
Success
```

The End

Our code at GitHub: https://github.com/TabSchu/ParserRNP