Yuri Kayser da Rosa 12201035
Alexandre Pereira Back 18100844

Para iniciar a implementação do analisador sintático, começamos copiando todo o código descrito na seção 4.2 do livro “Como construir um compilador usando ferramentas java” disponível no Moodle da disciplina. Durante este processo também inserimos o suporte aos tipos de dado Boolean, Float e Char nas partes solicitadas pelo professor na questão 3.A.

Com o código completo, fizemos a leitura completa das seções 2.3,2.4,2.4 e 4.2 para melhor compreender o que estava sendo feito. Depois disso, partimos para a implementação dos requisitos descritos na questão 3.C do trabalho

Para conseguir que variáveis fossem atribuídas na lista de argumentos dos métodos, alteramos a expressão regular do estado paramlist() para que, após a declaração do tipo e do nome da variável fosse possível inserir o token “=” seguido de uma alocexpression() ou expression().

```
void paramlist():
{

}
{
   [
        (< INT>|<STRING>|<IDENT>|<FLOAT>|<CHAR>|<BOOLEAN>) < IDENT > (<
        RBRACKET > < LBRACKET >)* (<ASSIGN> (alocexpression() | expression()))?
        (< COMMA > (<INT>|<STRING>|<IDENT>|<FLOAT>|<CHAR>|<BOOLEAN>) < IDENT
        > (< RBRACKET > <LBRACKET >)* (<ASSIGN> (alocexpression() | expression()))?)*
   ]
}
```

Depois, partimos para implementação do item D referente a criar a possibilidade de criar métodos e métodos construtores em qualquer ordem na classe. Para isso, nos inspiramos na classe statement() que gera diversos tipos de terminais uma única vez, ou que gera statlist(), um estado que basicamente gera um novo statement() seguido da presença ou não do próprio statlist() gerando assim uma recursão à direita que permite que novos statements sejam inseridos. Sendo assim, alteramos o estado classbody() e a alteramos para gerar a produção

```
< LBRACE >
        [classbodylist()]
  <RBRACE >
```

O estado classbodylist() ficou da seguinte maneira:

```
void classbodylist():
{
}
{
        classbodystatement()[classbodylist()]
}
```

O estado classbodystatement() contem as produções possíveis para o ClassBody():

```
void classbodystatement():
{
}
{
        classlist()
        |
        LOOKAHEAD(3)vardecl() < SEMICOLON >
        |
        LOOKAHEAD(3) atribstat() < SEMICOLON >
        |
        constructdec()
        |
        methoddecl()
}
```

Feito isto, partimos para a implementação do ultimo requisito ainda não cumprido, descrito na questão 3.B que envolve o uso dos operadores AND OR XOR NOT. Para isso, alteramos o estado Expression() e criamos mais níveis de estados para comportar os novos operadores e continuar trabalhando com os implementados anteriormente.
Criamos um estado chamado logicexpression() e o adicionamos como uma produção de expression(). Adicionamos tambem o operador AND, que pode ou não ser utilizado, e é seguido de uma ou mais produções de logicexpression().

```
void expression():
{
}
{
        logicexpression()(<AND > logicexpression())*
}
```

Uma logic expression, produz um avlexpression() que pode ou não ser precedida do operador Not, seguida ou não dos operadores XOR ou OR que podem tambem ser seguidos de um NOT e são seguidos por mais uma avlexpression():

```
void logicexpression() :
{
}
{
  [<NOT>] avlexpression()  (( <XOR> | <OR>) [<NOT>] avlexpression())*
}
```

Na avlexpression() apenas transferimos o antigo conteúdo do estado expression() para que pudéssemos utilizar os operadores lógicos.

```
void avlexpression():
{
}
{
        numexpr() [(< LT > | < GT > | < LE > | < GE > | < EQ > | < NEQ > ) numexpr()]
}
```

Dessa forma, não há nenhuma modificação no funcionamento das operações aritméticas e de avaliação antes suportadas, e além disso adicionamos o suporte aos operadores lógicos.

A implementação deste trabalho não foi fácil, tivemos que percorrer uma curva de aprendizado um tanto grande principalmente para nos ajustarmos à sintaxe do JavaCC e o suporte aos operadores lógicos exigiu algumas horas de planejamento e tentativas, mas cremos que conseguimos cumprir a maior parte dos objetivos propostos.

Tabela de testes

Nome Programa	Resultado
programaaceito1.fun	Programa analisado com sucesso
programaaceito2.fun	Programa analisado com sucesso
programaaceito3.fun	Programa analisado com sucesso
programaaceito4.fun	Programa analisado com sucesso
programanaoaceito1.fun	Encountered " ")" ") "" at line 15, column 27. Was expecting one of: "+" ... "-" ... "not" ... "(" ... <int_constant> ... <float_constant> ... <char_constant> ... <string_constant> ... "null" ... <boolean_constant> ... <IDENT> ...
programanaoaceito2.fun	Encountered " "(" "("" at line 17, column 24. Was expecting one of: ">" ... "<" ... "==" ... "<=" ... ">=" ... "!=" ... "+" ... "-" ... "*" ... "/" ... "%" ... "and" ... "or" ... "xor" ... "[" ... ";" ... "." ...
programanaoaceito3.fun	Encountered " ")" ") "" at line 18, column 13.

	Was expecting one of: "+" ... "-" ... "not" ... "(" ... <int_constant> ... <float_constant> ... <char_constant> ... <string_constant> ... "null" ... <boolean_constant> ... <IDENT> ...
programanaoaceito4.fun	Encountered " "=" "= "" at line 19, column 18. Was expecting one of: ">" ... "<" ... "==" ... "<=" ... ">=" ... "!=" ... "+" ... "-" ... "*" ... "/" ... "%" ... "and" ... "or" ... "xor" ... "[" ... ";" ... "." ...
programanaoaceito5.fun	Encountered " "float" "float "" at line 7, column 9. Was expecting one of: "=" ... "[" ... "[" ...
programanaoaceito6.fun	Encountered "<EOF>" at line 30, column 1. Was expecting one of: "class" ... "constructor" ... "int" ... "float" ... "string" ...

	"char" ... "boolean" ... "}" ... <IDENT> ...
programanaoaceito7.fun	Encountered " <int_constant> "3 "" at line 8, column 21. Was expecting one of: ">" ... "<" ... "==" ... <=" ... ">=" ... "!=" ... "+" ... "-" ... "*" ... "/" ... "%" ... "and" ... "or" ... "xor" ... ";" ...
programanaoaceito8.fun	Encountered " "/" "/ "" at line 5, column 1. Was expecting one of: "class" ... "constructor" ... "int" ... "float" ... "string" ... "char" ... "boolean" ... "}" ... <IDENT> ...
programanaoaceito9.fun	Encountered " <int_constant> "420 "" at line 8, column 25. Was expecting one of: ">" ... "<" ... "==" ... <=" ... ">=" ... "!=" ... "+" ... "-" ...

	"*" ... "/" ... "%" ... "and" ... "or" ... "xor" ... ";" ...
programanaoaceito10.fun	Encountered " "}" "} "" at line 13, column 1. Was expecting one of: ">" ... "<" ... "==" ... "<=" ... ">=" ... "!=" ... "+" ... "-" ... "*" ... "/" ... "%" ... "and" ... "or" ... "xor" ... "[" ... ";" ... "." ...