Relatório Árvore Sintático X+++

Equipe

14104255 - Bruno Aurélio Rôzza de Moura Campos
14101370 - Fabiano Pereira de Oliveira
14101383 - Laís Ferrigo Perazzolo
14101398 - Thary Correia

Papeis no Desenvolvimento

Houve 1 encontro com **todos** os membros participando do desenvolvimento da terceira parte do trabalho.

Alterações que foram realizadas sobre o projeto sugerido nos capítulos 6 e 7 de Delamaro (2004)

- construção da árvore sintática
- impressão da árvore sintática

syntacticTree/FloatConstNode.java

- Arquivo criado

```
package syntacticTree;

import parser.*;


public class FloatConstNode extends ExpreNode {
    public FloatConstNode(Token t) {
        super(t);
    }
}
```

syntacticTree/ByteConstNode.java

- Arquivo criado

```
package syntacticTree;

import parser.*;


public class ByteConstNode extends ExpreNode {
    public ByteConstNode(Token t) {
        super(t);
```

```
    }
}
```

syntacticTree/LongConstNode.java

- Arquivo criado

```
package syntacticTree;

import parser.*;


public class LongConstNode extends ExpreNode {
    public LongConstNode(Token t) {
        super(t);
    }
}
```

syntacticTree/ShortConstNode.java

- Arquivo criado

```
package syntacticTree;

import parser.*;


public class ShortConstNode extends ExpreNode {
    public ShortConstNode(Token t) {
        super(t);
    }
}
```

syntacticTree/PrintTree.java

- Alterado o arquivo
- Adicionado novos métodos para os tipos:
    - Byte
    - Short
    - Long
    - Float

```
    // PARTE 03
    // ----------------------- Constante BYTE ----------------------------
    public void numberByteConstNode(ByteConstNode x) {
```

```
        if (x == null) {
            return;
        }

        x.number = kk++;
    }

    public void printByteConstNode(ByteConstNode x) {
        if (x == null) {
            return;
        }

        System.out.println();
        System.out.print(x.number + ": ByteConstNode ===> " +
                x.position.image);
    }

    // PARTE 03
    // ------------------------ Constante SHORT ---------------------------
-
    public void numberShortConstNode(ShortConstNode x) {
        if (x == null) {
            return;
        }

        x.number = kk++;
    }

    public void printShortConstNode(ShortConstNode x) {
        if (x == null) {
            return;
        }

        System.out.println();
        System.out.print(x.number + ": ShortConstNode ===> " +
                x.position.image);
    }

    // PARTE 03
    // ------------------------ Constante LONG ----------------------------
    public void numberLongConstNode(LongConstNode x) {
        if (x == null) {
            return;
        }

        x.number = kk++;
    }

    public void printLongConstNode(LongConstNode x) {
        if (x == null) {
            return;
        }

        System.out.println();
```

```
        System.out.print(x.number + ": LongConstNode ===> " +
                x.position.image);
    }

    // PARTE 03
    // ----------------------- Constante FLOAT ---------------------------
-
    public void numberFloatConstNode(FloatConstNode x) {
        if (x == null) {
            return;
        }

        x.number = kk++;
    }

    public void printFloatConstNode(FloatConstNode x) {
        if (x == null) {
            return;
        }

        System.out.println();
        System.out.print(x.number + ": FloatConstNode ===> " +
                x.position.image);
    }
```

- Na parte `expressão em geral`
- Adicionado condições para a impressão da árvore com novos tipos de variáveis.

```
    public void printExpreNode(ExpreNode x) {
    ...
        // PARTE 03
        } else if (x instanceof ByteConstNode) {
            printByteConstNode((ByteConstNode) x);
        } else if (x instanceof ShortConstNode) {
            printShortConstNode((ShortConstNode) x);
        } else if (x instanceof LongConstNode) {
            printLongConstNode((LongConstNode) x);
        } else if (x instanceof FloatConstNode) {
            printFloatConstNode((FloatConstNode) x);
```

```
    public void numberExpreNode(ExpreNode x) {
    ...
        // PARTE 03
        } else if (x instanceof ByteConstNode) {
            numberByteConstNode((ByteConstNode) x);
        } else if (x instanceof ShortConstNode) {
            numberShortConstNode((ShortConstNode) x);
        } else if (x instanceof LongConstNode) {
            numberLongConstNode((LongConstNode) x);
```

```
        } else if (x instanceof FloatConstNode) {
            numberFloatConstNode((FloatConstNode) x);
```

langX+++.jj

- Arquivo baseado no livro *Como Construir um Compilador: utilizando ferramenta Java*, cap 07
- Foi removido as funções de tipagem e acesso e inserido os tokens dentro das funções.

```
// adicionado tokens de: tipagem de variável, tipo de acesso, atribuição e
o token final
VarDeclNode vardecl(RecoverySet g) throws ParseEOFException :
{
    Token t1 = null, t2;
    int k = 0;
    ListNode l = null;
}
{
    try {
            [<FINAL>]   // variavel pode ser ou não FINAL
            [<PUBLIC> | <PRIVATE> | <PROTECTED>]
            ( t1 = <INT>
            | t1 = <STRING>
            | t1 = <BYTE>
            | t1 = <SHORT>
            | t1 = <LONG>
            | t1 = <FLOAT>
            | t1 = <IDENT> )
            t2 = <IDENT> ( [<ASSIGN> factor()] )
            { l = new ListNode(new VarNode(t2, k)); }
            (<COMMA> { k = 0; }  t2 = <IDENT> ( <LBRACKET> <RBRACKET> {
k++; })*
            { l.add(new VarNode(t2, k)); }
    )*
    { return new VarDeclNode(t1, l); }
    } catch (ParseException e) {
    consumeUntil(g, e, "vardecl");
    return new VarDeclNode(t1, l);
    }
}
```

- Adicionado tokens de: tipagem de variável e tipo de acesso.

```
// Trabalho - parte 03
// Add tokens de acesso e tipagem
MethodDeclNode methoddecl(RecoverySet g) throws ParseEOFException : {
    Token t1 = null,
          t2 = null;
    int k = 0;
    MethodBodyNode m = null;
```

```
}
{
    try {
        [ <PUBLIC> | <PRIVATE> | <PROTECTED> ]
        ( t1 = <INT>
        | t1 = <STRING>
        | t1 = <BYTE>
        | t1 = <SHORT>
        | t1 = <LONG>
        | t1 = <FLOAT>
        | t1 = <IDENT> )
        (<LBRACKET> <RBRACKET> { k++; } )*
        t2 = <IDENT> m = methodbody(g)
        { return new MethodDeclNode(t1, k, t2, m); }
    }
    catch (ParseException e) {
        consumeUntil(g, e, "methoddecl");
        return new MethodDeclNode(t1, k, t2, m);
    }
}
```

- Adicionado tokens de: tipagem de variável, tipo de acesso e atribuição.

```
// Trabalho - parte 03
// Add métodos de acesso e tipagem
ListNode paramlist(RecoverySet g) throws ParseEOFException : {
    ListNode p = null, q = null;
    int k = 0;
    Token t1 = null;
    Token t2 = null;
}
{
    try {
        [
            [ <PUBLIC> | <PRIVATE> | <PROTECTED> ]
            ( t1 = <INT>
            | t1 = <STRING>
            | t1 = <BYTE>
            | t1 = <SHORT>
            | t1 = <LONG>
            | t1 = <FLOAT>
            | t1 = <IDENT> ) t2 = <IDENT>
            (<LBRACKET> <RBRACKET> { k++; } )*
            {
                q = new ListNode(new VarNode(t2, k));
                p = new ListNode(new VarDeclNode(t1, q));
            }
            (<COMMA> {k = 0;}   ( t1 = <INT>
                                | t1 = <STRING>
                                | t1 = <BYTE>
                                | t1 = <SHORT>
                                | t1 = <LONG>
```

```
                                | t1 = <FLOAT>
                                | t1 = <IDENT> )
            t2= <IDENT> (<LBRACKET> <RBRACKET> {k ++;} )* {
                q = new ListNode(new VarNode(t2, k));
                p.add(new VarDeclNode(t1, q));
            }
        )*
       ] { return p;}
    } catch (ParseException e) {
        consumeUntil(g, e, "paramlist");
        return null;
    }
}
```

- Adicionado tokens de: tipagem de variável.

```
//parte 03
ExpreNode alocexpression(RecoverySet g) throws ParseEOFException : {
    ExpreNode e1 = null,
              e2 = null;
    ListNode l = null;
    Token t1, t2;

    RecoverySet f1 = new RecoverySet(RPAREN).union(g),
                f2 = new RecoverySet(RBRACKET).union(g);
}
{
    t1 = <NEW>
    (LOOKAHEAD(2) t2 = <IDENT> <LPAREN> l = arglist(f1) <RPAREN>
         { e1 = new NewObjectNode(t1, t2, l); }
        |   ( t2 = <INT>
            | t2 = <STRING>
            | t2 = <BYTE>
            | t2 = <SHORT>
            | t2 = <LONG>
            | t2 = <FLOAT>
            | t2 = <IDENT> )

      (<LBRACKET> e2 = expression(f2) <RBRACKET>
      {
        if ( l == null )
            l = new ListNode(e2);
        else
                    l.add(e2);
      }
      )+
          { e1 = new NewArrayNode(t1, t2, l); }
    )
    { return e1; }
}
```

- Adicionados novos tipos que foram criados no package syntacticTree

```
// Adicionados novos tipos, criados em syntacticTree
ExpreNode factor() throws ParseEOFException : {
    ExpreNode e = null;
    Token t;
}
{
    (
        t = <int_constant> { e = new IntConstNode(t); }
        | t = <string_constant> { e = new StringConstNode(t); }
        | t = <null_constant> { e = new NullConstNode(t); }
        | t = <long_constant> { e = new LongConstNode(t); }
        | t = <short_constant> { e = new ShortConstNode(t); }
        | t = <float_constant> { e = new FloatConstNode(t); }
        | e = lvalue(null)
        | <LPAREN> e = expression(null) <RPAREN>
    )
    { return e; }
}
```

Comandos utilizados

```
sudo apt install javacc
```

- Geração do parser

```
javacc parser/langX+++.jj
```

- Geração dos arquivos `.class`

```
javac parser/langX.java
```

- Testes

```
java parser.langX testes_e_logs/teste_com_erro_classbody.x
java parser.langX testes_e_logs/teste_expressoes_logicas.x
```

- Debug Analisador Sintático

```
java parser.langX -debug_AS testes_e_logs/debugAS.x
```

- Árvore Sintática

```
java parser.langX -print_tree testes_e_logs/teste_expressoes_logicas.x
java parser.langX -print_tree testes_e_logs/teste_com_erro_classbody.x
java parser.langX -print_tree testes_e_logs/bintree.x
```

Notas

- O arquivo `langX+++.jj` foi identado com 4 espaços,
- Todo o trabalho foi versionado usando a ferramenta git
- Encoding dos arquivos: US-ASCII