Relatório de atividade

Alunos: Bruno Ribeiro da Silva (12200992), Djéssica Schell Crocetta (12203762), Luan Felipe Sievers (12204515)

Atribuições

Os alunos Bruno e Luan foram responsáveis pela implementação das extensões da liguagem X+++ para atenderem os requisitos da atividade. A aluna Djéssica foi responsável por parte do relatório. Abaixo uma lista de contribuições ao repositório do trabalho:

- 980e36bf - Completed work log. Minor fix in README
- fc3c86d - (HEAD -> master, origin/master, origin/HEAD) Update relatorio .md - incompleto (25 hours ago) Djéssica S.C
- 4dec0e2 - Added tests and fixed code to deliver (7 days ago) Luan Felipe Sievers
- 76bc112 - Updated log package(7 days ago) Luan Felipe Sievers
- ceeb3d6 - Updated task description (8 days ago) Bruno Ribeiro da Silva
- 4c359e7 - Added coments and constants bytes, short, long and float (11 days ago) Luan Felipe Sievers
- 882e73d - Added some constants. Implemented float (12 days ago) Bruno Ribeiro da Silva
- 82a8f10 - Formatted parser code. Moved examples to proper folders (12 days ago) Bruno Ribeiro da Silva
- 6085ef0 - Updated README file. Added X+++ code. Minor fixes. (13 days ago) Bruno Ribeiro da Silva
- 12d1711 - Added Examples (3 weeks ago) Luan Felipe Sievers
- 5c4dc3d - POM enhanced to automatize some tasks. Instructions added to read me file (3 weeks ago) Bruno Ribeiro da Silva
- 7bca20d - add delamaro book and example code (3 weeks ago) Bruno Ribeiro da Silva
- 5cea126 - Initial commit (3 weeks ago) Bruno Ribeiro da Silva

Especificação léxica da linguagem X+++

Para o desenvolvimento do analisador léxico dessa atividade foi utilizado o código fonte do capítulo 3 do livro "Como Construir um Compilador Utilizando Ferramentas Java". As extensões solicitadas para essa atividade foram:

- Operadores lógicos AND, OR, XOR e NOT;
- Novos tipos de variáveis e literais: BYTE, SHORT, LONG e FLOAT, além dos já existentes;
- Qualificadores de identificadores: FINAL, PUBLIC, PRIVATE e PROTECTED, como usado em Java

O Java Compiler Compiler (JavaCC) foi utilizado através de um plugin para o automizador de compilação Maven. Conforme foram realizadas as alterações no código fonte, foram utilizados para controle os exemplos do livro para verificar se as alterações feitas estavam mantendo a consistência da linguagem, isto é, não estavam sendo detectados erros onde não haviam erros ou tokens inválidos.

Operadores lógicos

Os operadores lógicos AND, OR, XOR e NOT foram definidos conforme o trecho de código abaixo:

```
< AND: "&&" > |
< OR: "||" > |
< XOR: "^" > |
< NOT: "!" >
```

Eles foram definidos no mesmo grupo de tokens de operadores. Adicionamos esses operadores no exemplo de código da linguagem X+++ e utilizamos o analisador léxico gerado. O resultado da análise do código fonte X+++ estão disponíveis na pasta log, junto desse relatório.

O código X+++ para validação desses tokens:

```
void checkOperadoresLogicos() {
    if (1 && 1) {
        print "Operador lógico AND - OK";
    }
    if (1 || 0) {
        print "Operador lógico OR - OK";
    }
    if (1 ^ 0) {
        print "Operador lógico XOR - OK";
    }
    if (!0) {
        print "Operador lógico NOT - OK";
    }
}
```

Variáveis e literais

Diferentemente da implementação dos operadores lógicos, para a implementação das variáveis foi necessário o emprego de expressão regular que identifique os padrões de números float. Não foi necessário criar expressões regulares para os tipos short, byte e long porque esses já faziam parte da definição da linguagem do exemplo do livro.

A identificação dessas variáveis e literais foram adicionados aos tokens de palavras reservadas:

```
< BYTE: "byte" > |
< SHORT: "short" > |
< LONG: "long" > |
< FLOAT: "float" >
```

E a expressão regular para float:

```
< float_constant: (
                     <int_constant> "."
                    ( <int_constant> ( ("e" | "E")? ("-")? <int_constant>
)? )? |
```

```
                    (<int_constant>)? "." <int_constant>
                    (("e" | "E")? ("-")? <int_constant>)? ) ("F" | "f")?
                    >
```

Essa expressão regular permite que sejam identificados números de ponto conforme os padrões abaixo:

- 5.3876e4;
- 7.321E-3;
- 3.2E+4;
- 0.5e-6;
- 0.45;
- 6.e10;

Utilizamos o código abaixo para validação das variáveis e literais implementadas:

```
void checkNovasVariaveis() {
    byte testeByte = 1;
    short testeShort = 1;
    long testeShort = 1;
    float testeFloat1 = 5.3876e4;
    float testeFloat2 = 7.321E-3;
    float testeFloat3 = 3.2E+4;
    float testeFloat4 = 0.5e-6;
    float testeFloat5 = 0.45;
    float testeFloat6 = 6.e10;
}
```

Qualificadores e identificadores

Na etapa de definição do analisador léxico, para adicionar os qualificadores foi necessário apenas reservar as palavras dos mesmos:

```
< FINAL: "final" > |
< PUBLIC: "public" > |
< PRIVATE: "private" > |
< PROTECTED: "protected" >
```

O teste desses qualificadores foi feito adicionando-os aos métodos já existentes. Por exemplo:

```
protected int insert(data k) {
    int x;

    x = k.compara(key);
    if (x < 0) {
        if (left != null)
            return left.insert(k);
```

```
            left = new bintree(k);
            return 1;
        }
        if (x > 0) {
            if (right != null)
                return right.insert(k);
            right = new bintree(k);
            return 1;
        }
        return 0;
    }
```