

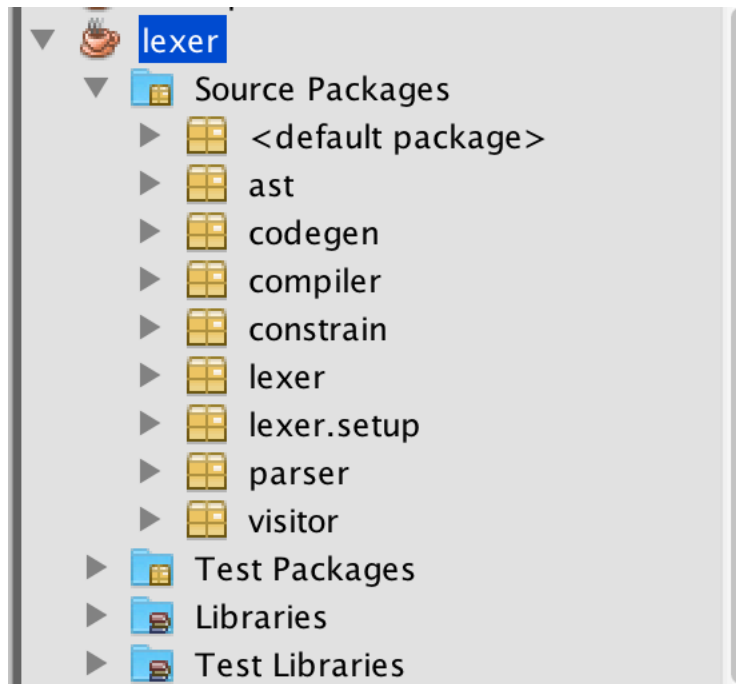
Project 3: Paser text printing
CSC 413 Spring 2016 Professor Yoon
Developer's Guide
March 12, 2016
Qihong Kuang

Table of Contents

1. Introduction	2
2. Class diagram	4
3. Compile information.....	5
3.1. Test Result.....	5
4. Implementation of Parser.....	13
4.1. Introduction of paser.java.....	13
4.2. rProgram()	13
4.3. rBlock()	13
4.4. rDecl().....	13
4.5. rEhead()	13
4.6. rAssign()	13
4.7. rFactor()	14
4.8. rFF()	14
5. Implementation of Visitor	14
6. Conclusion	15

1. Introduction































This document is a guide to explain how I handle the parser. In this parser project, we mainly take advantage of the characteristics of recursion. We experienced the beauty of recursion



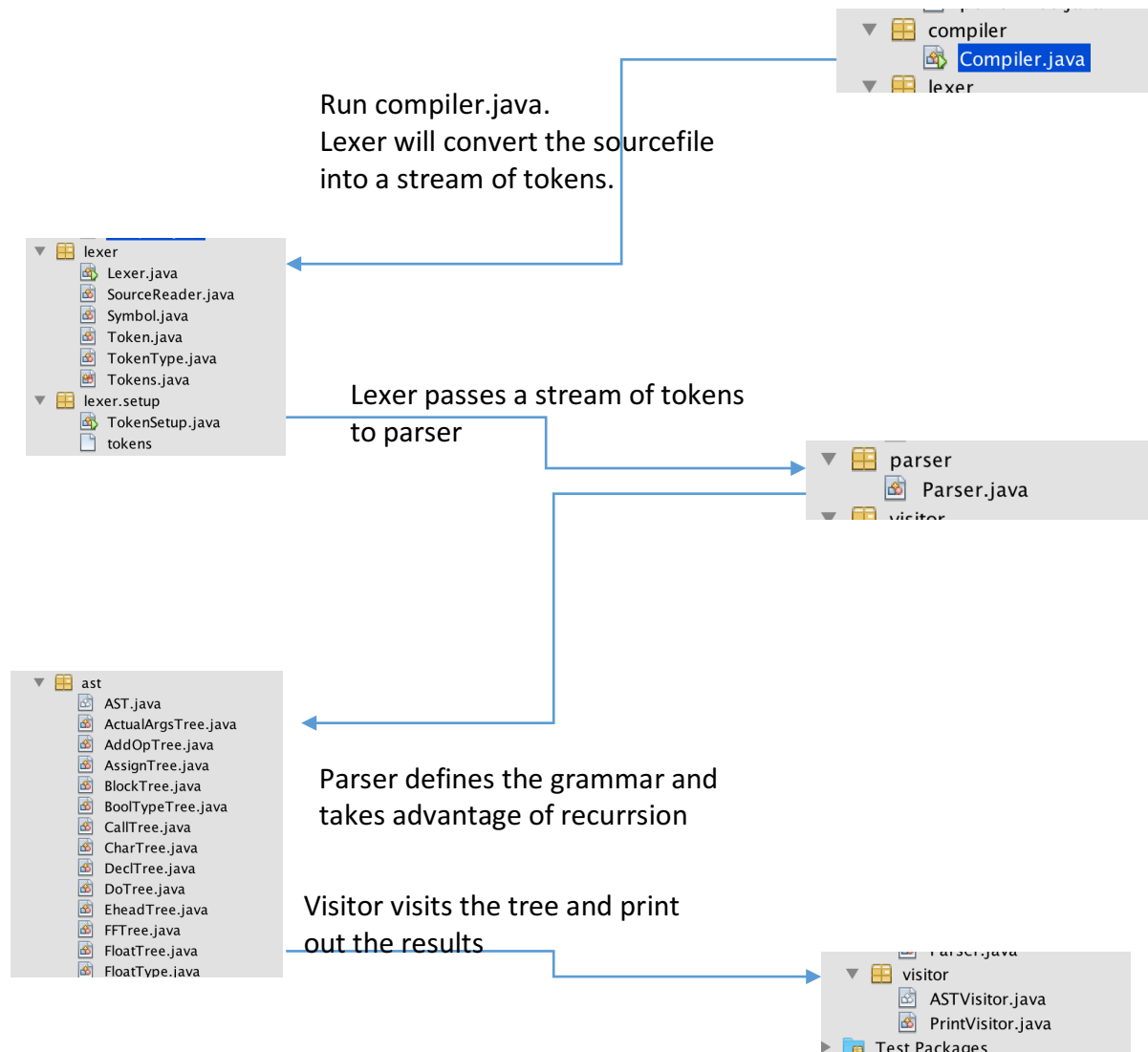
In project 3, I need to use the implementation of project2. That is to say, I may need to add more tokens like semicolon, power, for into project 2.

Before I modify the parser.java file, I need to add more trees into the AST package.

Below is part of the trees in ast package:

	AddOpTree.java
	AssignTree.java
	AST.java
	BlockTree.java
	BoolTypeTree.java
	CallTree.java
	CharTree.java
	charTypeTree.java
	DeclTree.java
	DoTree.java
	EheadTree.java
	FFTree.java
	FloatTree.java
	FloatType.java
	FormalsTree.java
	ForTree.java
	FunctionDeclTree.java
	IdTree.java
	IfTree.java
	IntTree.java
	IntTypeTree.java
	MultOpTree.java
	NegTree.java
	NotTree.java
	powerTree.java
	ProgramTree.java
	RelOpTree.java
	ReturnTree.java
	SciNTree.java
	StringTree.java

2. Class diagram



3. Compile information.

Developer already added the extra tokens into the token file and run TokenSetup.java. There is no need the users compile the TokenSetup.java again.

Below is the compile information:

Use cd command navigate to the directory of Parser source codes. Be sure all the test files and java files are in the same directory. Then, type:

java -jar parser.java (follow by one argument which is the test file)

3.1. Test Result

Below are a series of test cases print out result:

```
a. program {  
    float f  
    float quotient ( float numerator, float denominator) {  
        boolean condition  
        condition = !!denominator == 0  
        if (condition) then { return 1.2e3}  
        else {return numerator / denominator}  
    }  
    do { do { f = quotient( 2.0, 1.01) } while !q } while 1.2  
}
```

```

      s      left 0 right 0
-----AST-----
1: Program
2:   Block
5:     Decl
3:       FloatType
4:       Id: f
8:     FunctionDecl
6:       FloatType
7:       Id: quotient
9:       Formals
12:        Decl
10:          FloatType
11:          Id: numerator
15:        Decl
13:          FloatType
14:          Id: denominator
16:      Block
19:        Decl
17:          BoolType
18:          Id: condition
20:        Assign
21:          Id: condition
22:          invert
23:            invert
25:            RelOp: ==
24:            Id: denominator
26:            Int: 0
27:        If
28:          Id: condition
29:          Block
30:            Return
31:              scientificN: 1.2e3
32:          Block
33:            Return
35:              MultOp: /
34:              Id: numerator
36:              Id: denominator
37:      do
38:        Block
39:          do
40:            Block
41:              Assign
42:                Id: f
44:                Call
43:                  Id: quotient
45:                  float: 2.0
46:                  float: 1.01
47:              invert
48:                Id: q
49:              float: 1.2

```

b.

```
program {  
  float print(int a, int b) {  
    return print(a, b)  
  }  
  if args then { return println(Hello) } else { return c}  
  argw = println(everybody)  
  do {a = 3.4} while println(i + 1 + .1 + args)  
}
```


-----AST-----

```
1: Program
2:   Block
5:     FunctionDecl
3:       FloatType
4:       Id: print
6:       Formals
9:       Decl
7:         IntType
8:         Id: a
12:      Decl
10:      IntType
11:      Id: b
13:      Block
14:        Return
16:        Call
15:          Id: print
17:          Id: a
18:          Id: b
19:      If
20:        Id: args
21:        Block
22:          Return
24:          Call
23:            Id: println
25:            Id: Hello
26:        Block
27:          Return
28:            Id: c
29:      Assign
30:        Id: argw
32:        Call
31:          Id: println
33:          Id: everybody
34:      do
35:        Block
36:          Assign
37:            Id: a
38:            float: 3.4
40:        Call
39:          Id: println
46:          AddOp: +
44:          AddOp: +
42:          AddOp: +
41:            Id: i
43:            Int: 1
45:            float: .1
47:          Id: args
```

c.

```
program { float i
do {
float factorial(int n) { return 1.0 }
do { do { } while 7.5} while 5.34E-3 <= 1
i = i + 1
} while true
}
```

```

-----AST-----
1:  Program
2:    Block
5:      Decl
3:        FloatType
4:        Id: i
6:      do
7:        Block
10:       FunctionDecl
8:         FloatType
9:         Id: factorial
11:        Formals
14:        Decl
12:          IntType
13:          Id: n
15:        Block
16:        Return
17:        float: 1.0
18:      do
19:        Block
20:        do
21:          Block
22:          float: 7.5
24:        RelOp: <=
23:        scientificN: 5.34E-3
25:        Int: 1
26:      Assign
27:        Id: i
29:        AddOp: +
28:        Id: i
30:        Int: 1
31:      Id: true
```

d.

```
program {  
  int a  
  a = 1 + 2 + 3 + 4  
  a = 1 ^ 2 ^ 3 ^ 4  
}
```

```
-----AST-----  
1:  Program  
2:    Block  
5:      Decl  
3:        IntType  
4:        Id: a  
6:        Assign  
7:        Id: a  
13:       AddOp: +  
11:         AddOp: +  
9:           AddOp: +  
8:             Int: 1  
10:            Int: 2  
12:            Int: 3  
14:            Int: 4  
15:        Assign  
16:        Id: a  
18:        power: ^  
17:          Int: 1  
20:        power: ^  
19:          Int: 2  
22:        power: ^  
21:          Int: 3  
23:          Int: 4
```

e.

```
program {  
  int j  
  j = 0  
  for (int i; i < 3; i = i+1) {  
    j = j + i  
  }  
}
```

-----TOKENS-----

Source file: test2.x.txt
user.dir: /Users/qihongkuang

```
READLINE: 1    program {  
               program      left: 0 right: 6  
               {            left: 8 right: 8
```

READLINE: 2

```
READLINE: 3    int j  
               int          left: 0 right: 2  
               j            left: 4 right: 4
```

READLINE: 4

```
READLINE: 5    j = 0  
               j            left: 0 right: 0  
               =            left: 2 right: 2  
               0            left: 4 right: 4
```

READLINE: 6

```
READLINE: 7    for (int i; i < 3; i = i+1) {  
               for          left: 0 right: 2  
               (            left: 4 right: 4  
               int          left: 5 right: 7
```

Expected: Identifier

*****exception*****parser.SyntaxError

Qihong-MacBook-Pro:~ qihongkuang\$

In this test case, for should followed by ASSIGN. Int i belongs to declaration not to ASSIGN so it will throw a syntax error.

f.

```
program {  
  int j  
  j = 0  
  for ( ; i < 3; i = i+1) {  
    j = j + i  
  }  
}
```

```
-----AST-----  
1:  Program  
2:    Block  
5:      Decl  
3:        IntType  
4:          Id: j  
6:      Assign  
7:        Id: j  
8:        Int: 0  
9:      for  
10:      Ehead  
12:        RelOp: <  
11:          Id: i  
13:          Int: 3  
14:      Assign  
15:        Id: i  
17:        AddOp: +  
16:          Id: i  
18:          Int: 1  
19:      Block  
20:        Assign  
21:          Id: j  
23:          AddOp: +  
22:            Id: j  
24:            Id: i
```

4. Implementation of Parser

4.1. Introduction of parser.java

This java file is used to define the grammar and recurs to the related tree.

4.2. rProgram()

Program tree should be the parent tree. According to the grammar, the program should expect token “program”, otherwise it will throw syntax error. Program tree should have one child which is the block tree.

4.3. rBlock()

In Block tree, it should expect the left brace first. And then, Block tree can have two different types of tree. One is declaration tree and the other is statement tree. Block tree can have zero or more declaration tree and have zero or more statement tree.

4.4. rDecl()

The grammar can define a variable or a function. So I need to declare the function or the variable in declaration tree.

4.5. rEhead()

EHEAD is a new rule to project 3 which is used to define the grammar for the For-loop.

4.6. rAssign()

Assign is also a new rule. The grammar is Name ‘=’ E. We can notice that Ehead rule is ‘(‘(ASSIGN)? ‘;’ (E)? ‘;’ (ASSIGN) ‘)’’. Assign tree is the child of Ehead tree.

4.7. rFactor()

rFF() is the child of rFactor(). This is because we also need to handle the power. Power is right-associated and has higher priority. Factor can generate rFF() or generate the power on the right side of rFF().

4.8. rFF()

In rFF(), I need to add new tree like scientific notation tree, float tree, String tree, character tree, negative tree and not tree. All of these tree should be regarded as the terminal tree so that the program can stop recursion. In rFF(), we nearly reach the terminals.

5. Implementation of Visitor

In Visitor class, I make use of the abstract class. I extend the sub visit tree class to AST class. So In visitor, I need to add some more visit sub tree.

6. Conclusion

1. After revising, parser can now follow the given grammar.
2. The revised parser passed all the test cases.
3. More tokens are added to the token file such as semicolon, power.
4. This project is only can print the result in text format. In next step, I will extend the program so that it can print the result tree in graph.