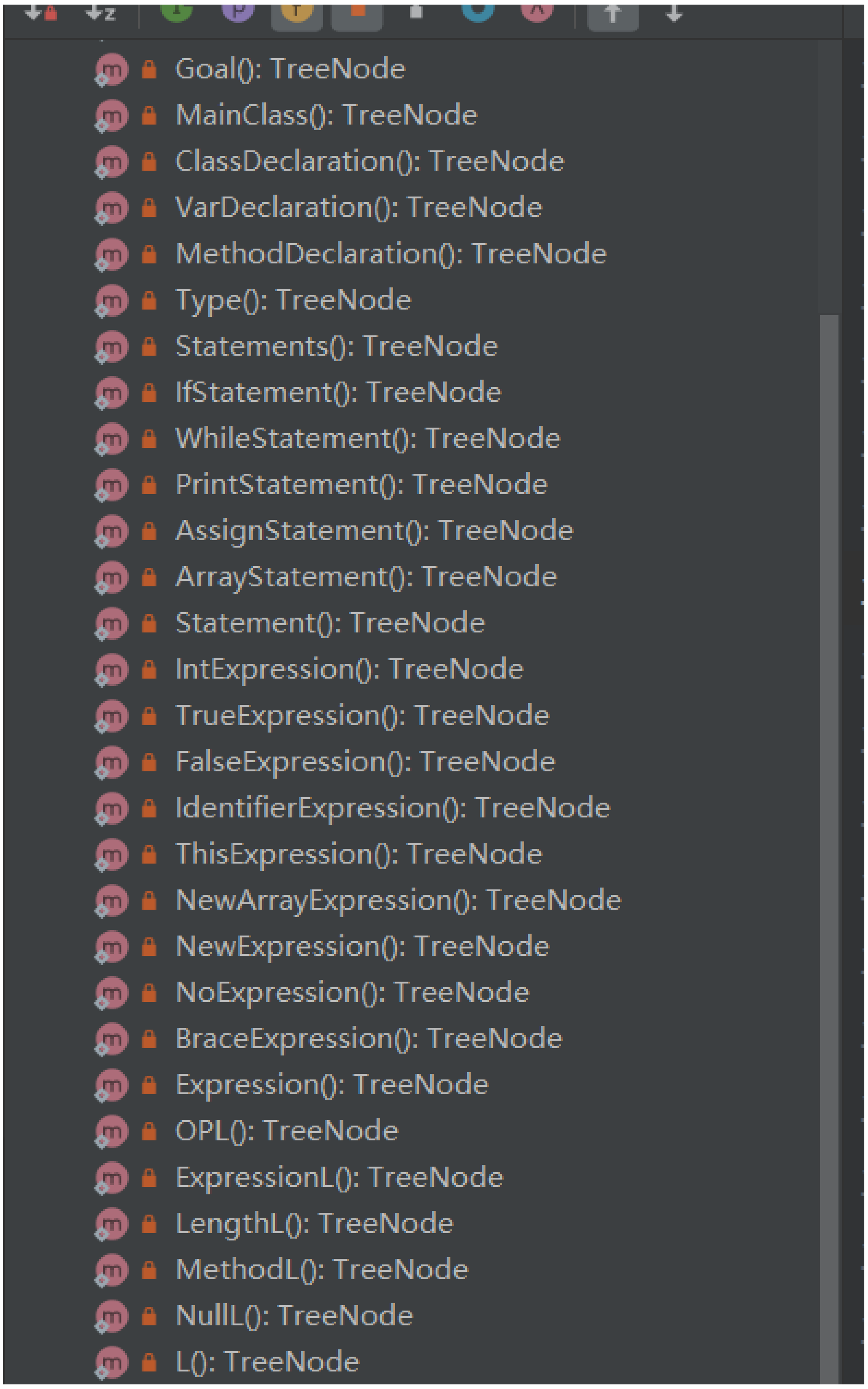
注意事项
算法
一部分测试结果
代码结构

## 注意事项

1. 在词法分析中未分析的浮点数和1flow1这两中类型在语法分析中处理完成
2. 语法分析时，和IDEA中的提示方式类似，只能提示出第一个语法分析中出现的错误，所以每次查看的时候只看syntaxErrorOut.txt中的第一行，之后更改后，要重新运行词法分析程序，之后在运行语法分析程序。
3. 测试结果在测试文件中，分别显示初始文件测试结果，文件夹：根据错误提示更改过后的正确的测试文件中时改正所有错误后的正确的分析结果，有正确的语法树。
4. 在语法树的输出中，不同的缩进代表树的层词，数的节点只显示了identifier和integerLiteral这两种类型，和最后的EOF。

## 算法

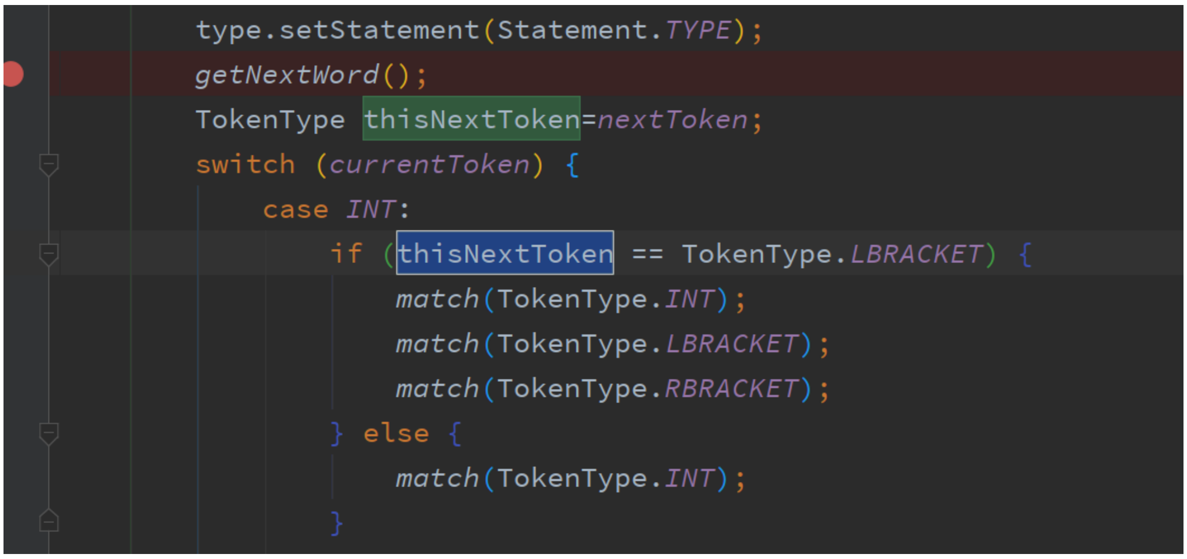
利用了递归下降的算法，一下是各个函数



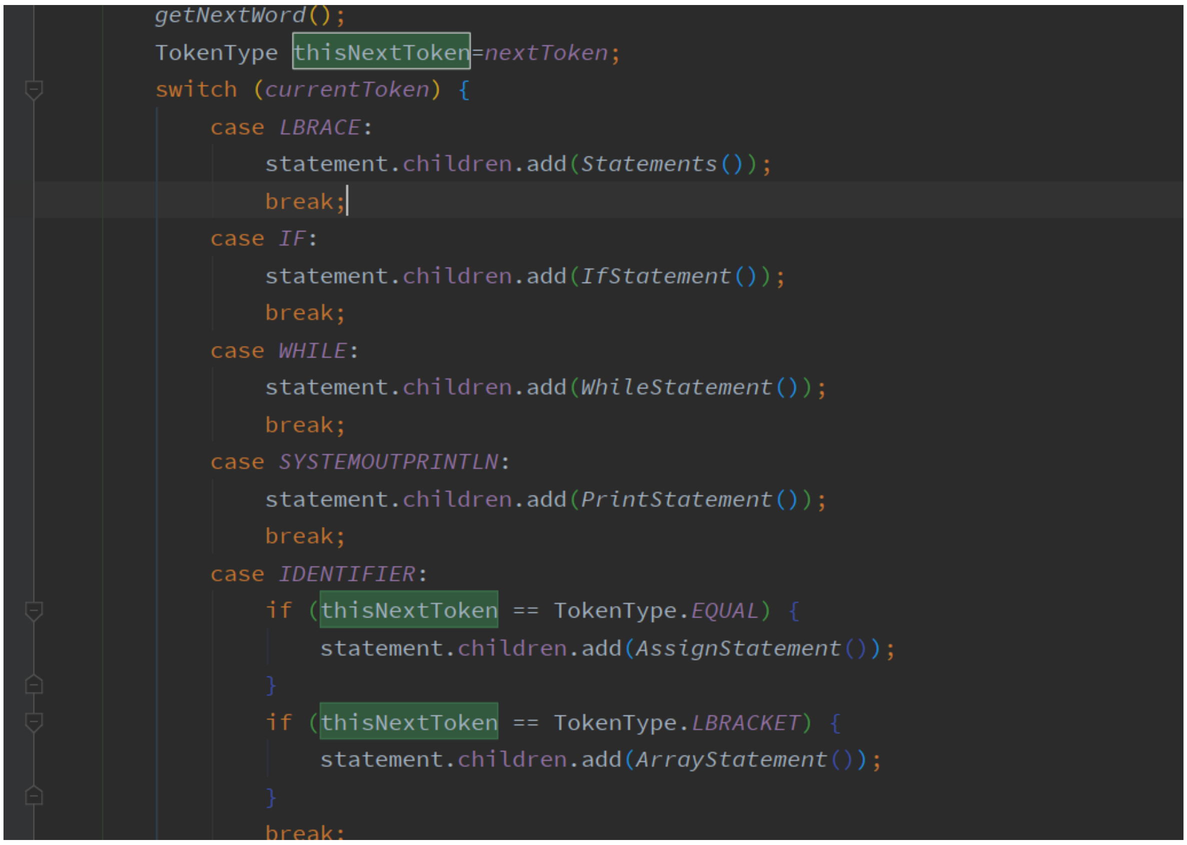
```
Goal(): TreeNode
MainClass(): TreeNode
ClassDeclaration(): TreeNode
VarDeclaration(): TreeNode
MethodDeclaration(): TreeNode
Type(): TreeNode
Statements(): TreeNode
IfStatement(): TreeNode
WhileStatement(): TreeNode
PrintStatement(): TreeNode
AssignStatement(): TreeNode
ArrayStatement(): TreeNode
Statement(): TreeNode
IntExpression(): TreeNode
TrueExpression(): TreeNode
FalseExpression(): TreeNode
IdentifierExpression(): TreeNode
ThisExpression(): TreeNode
NewArrayExpression(): TreeNode
NewExpression(): TreeNode
NoExpression(): TreeNode
BraceExpression(): TreeNode
Expression(): TreeNode
OPL(): TreeNode
ExpressionL(): TreeNode
LengthL(): TreeNode
MethodL(): TreeNode
NullL(): TreeNode
L(): TreeNode
```

在一些局部还利用了LL（1）的思想

例如：



```java
        type.setStatement(Statement.TYPE);
        getNextWord();
        TokenType thisNextToken=nextToken;
        switch (currentToken) {
            case INT:
                if (thisNextToken == TokenType.LBRACKET) {
                    match(TokenType.INT);
                    match(TokenType.LBRACKET);
                    match(TokenType.RBRACKET);
                } else {
                    match(TokenType.INT);
                }
```

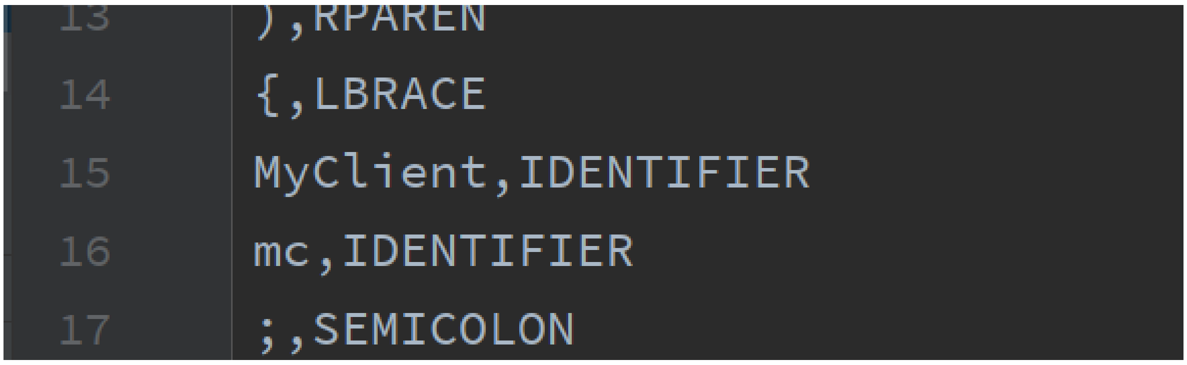


```java
        getNextWord();
        TokenType thisNextToken=nextToken;
        switch (currentToken) {
            case LBRACE:
                statement.children.add(Statements());
                break;
            case IF:
                statement.children.add(IfStatement());
                break;
            case WHILE:
                statement.children.add(WhileStatement());
                break;
            case SYSTEMOUTPRINTLN:
                statement.children.add(PrintStatement());
                break;
            case IDENTIFIER:
                if (thisNextToken == TokenType.EQUAL) {
                    statement.children.add(AssignStatement());
                }
                if (thisNextToken == TokenType.LBRACKET) {
                    statement.children.add(ArrayStatement());
                }
                break;
```

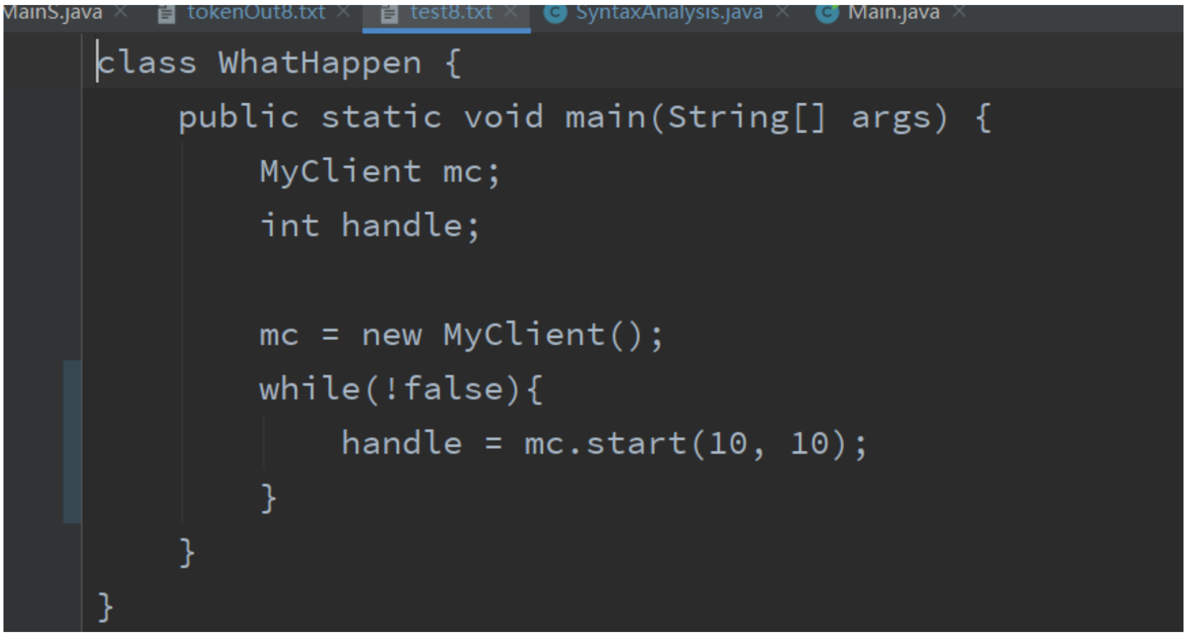
**一部分测试结果**

text8中：

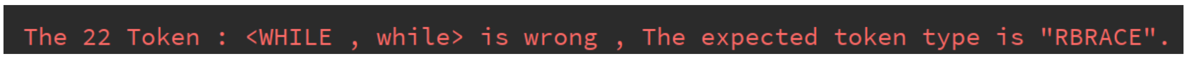
```
The 15 Token : <IDENTIFIER , MyClient> is wrong , The expected token type is "RBRACE".
```



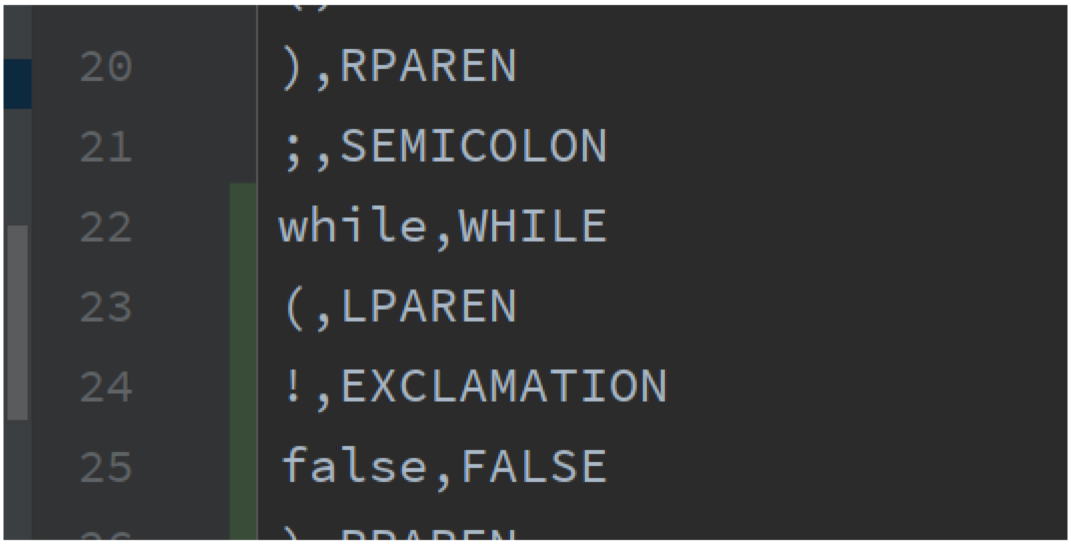
```
13    ),RPAREN
14    {,LBRACE
15    MyClient,IDENTIFIER
16    mc,IDENTIFIER
17    ;,SEMICOLON
```



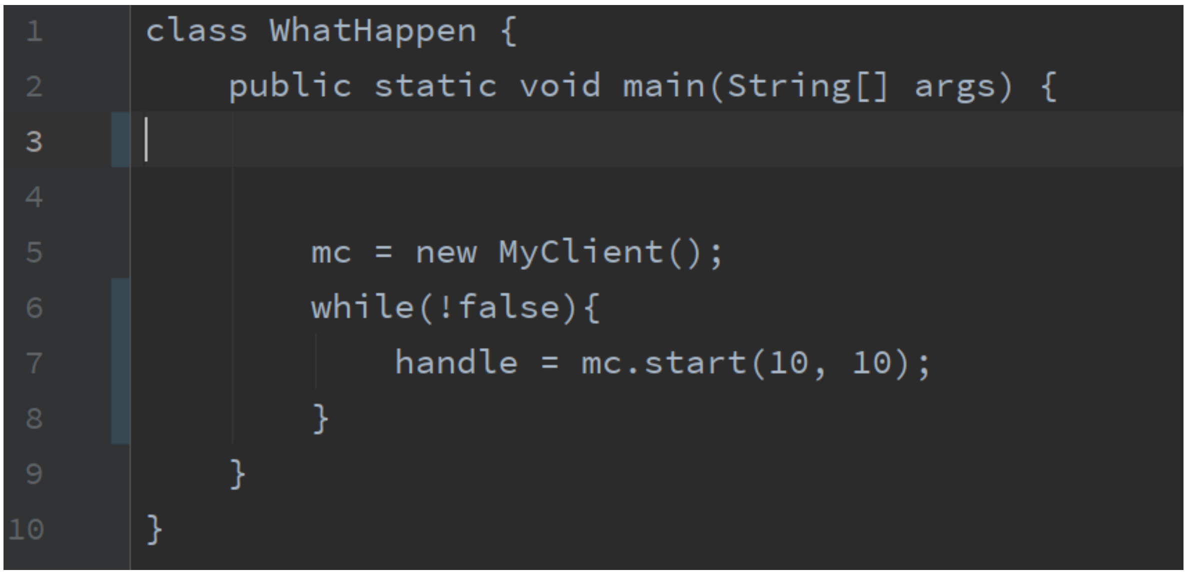
```
class WhatHappen {
    public static void main(String[] args) {
        MyClient mc;
        int handle;

        mc = new MyClient();
        while(!false){
            handle = mc.start(10, 10);
        }
    }
}
```



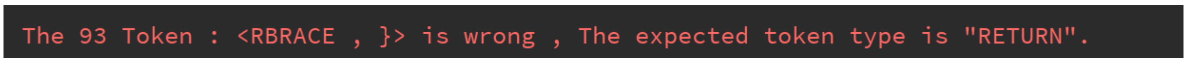
```
The 22 Token : <WHILE , while> is wrong , The expected token type is "RBRACE".
```



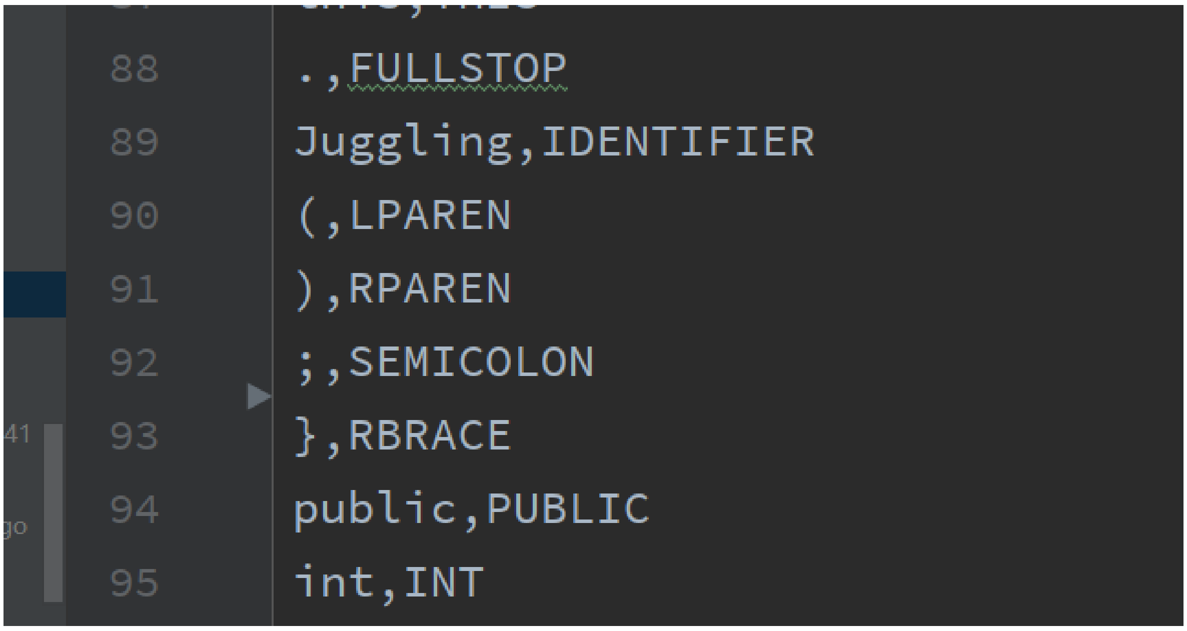
```
20  ),RPAREN
21  ;,SEMICOLON
22  while,WHILE
23  (,LPAREN
24  !,EXCLAMATION
25  false,FALSE
```



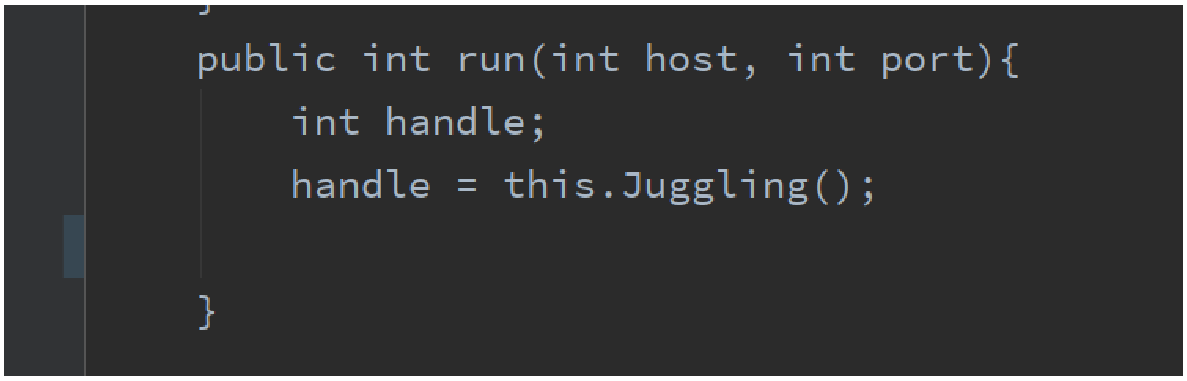
```
class WhatHappen {
    public static void main(String[] args) {


        mc = new MyClient();
        while(!false){
            handle = mc.start(10, 10);
        }
    }
}
```



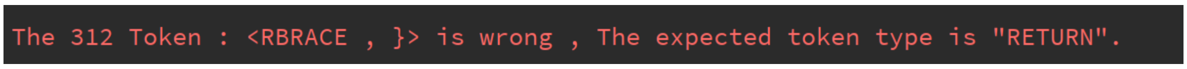
```
The 93 Token : <RBRACE , }> is wrong , The expected token type is "RETURN".
```



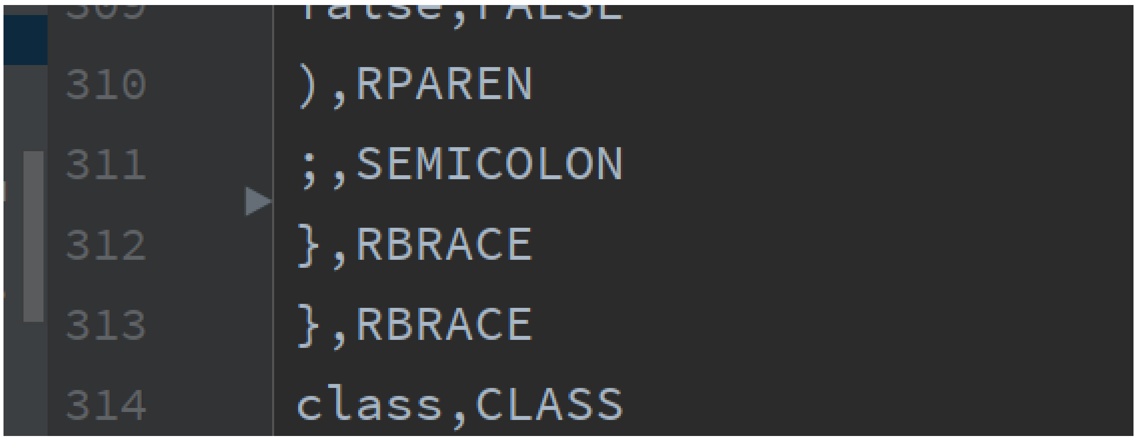
```
88  .,FULLSTOP
89  Juggling,IDENTIFIER
90  (,LPAREN
91  ),RPAREN
92  ;,SEMICOLON
93  },RBRACE
94  public,PUBLIC
95  int,INT
```



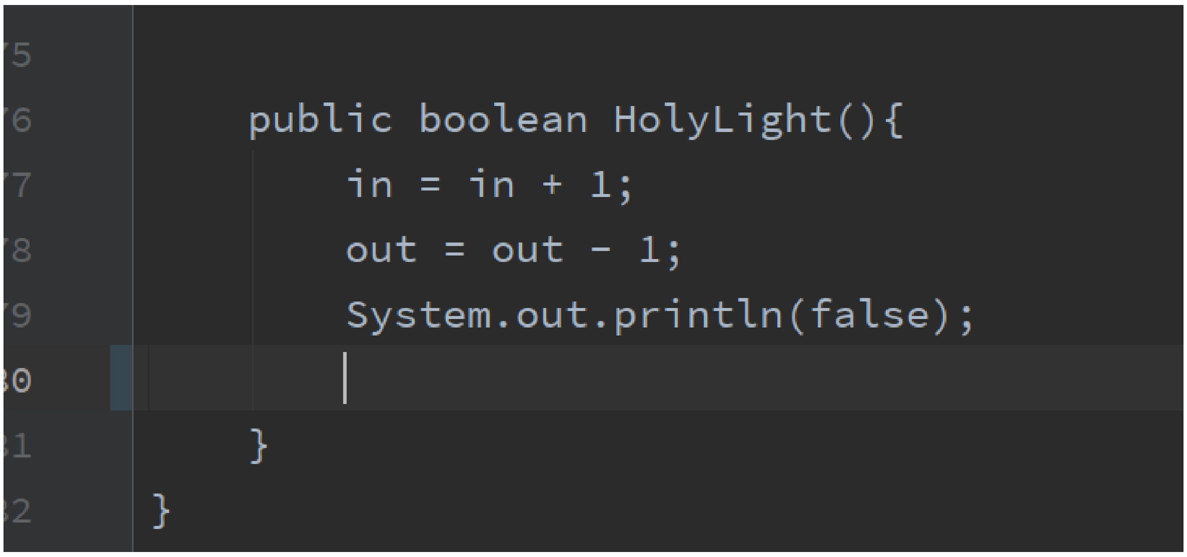
```
    public int run(int host, int port){
        int handle;
        handle = this.Juggling();

    }
```



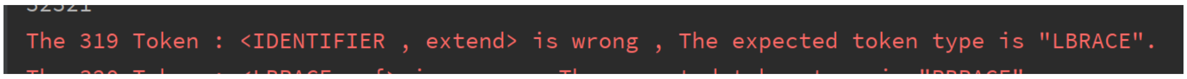
```
The 312 Token : <RBRACE , }> is wrong , The expected token type is "RETURN".
```



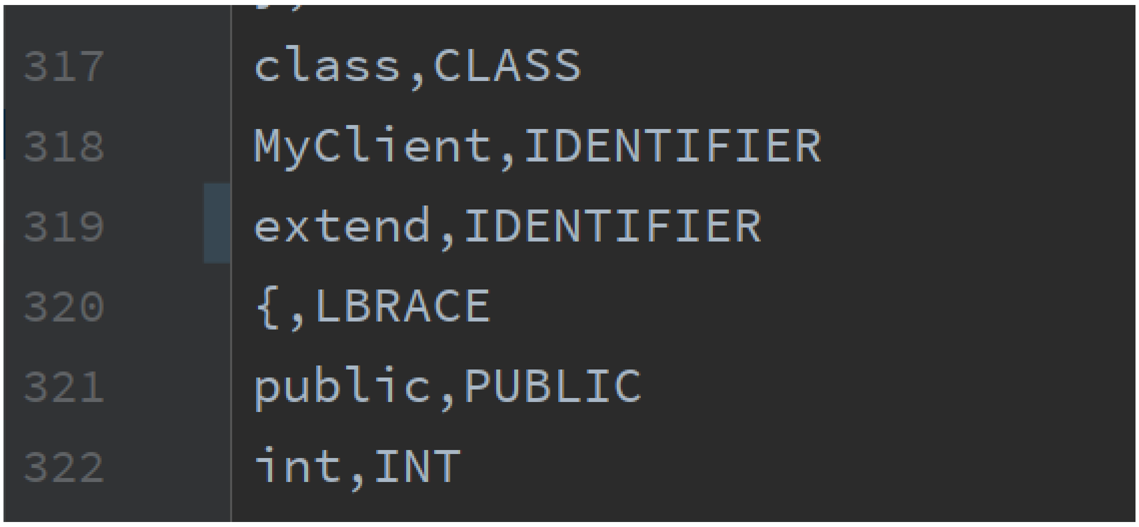
```
310  ),RPAREN
311  ;,SEMICOLON
312  },RBRACE
313  },RBRACE
314  class,CLASS
```



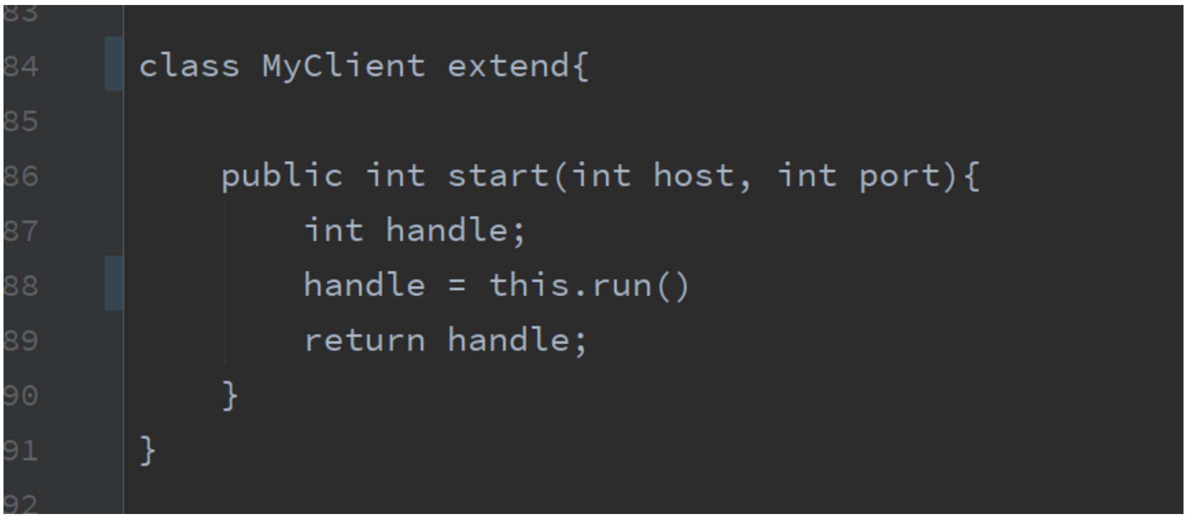
```
    public boolean HolyLight(){
        in = in + 1;
        out = out - 1;
        System.out.println(false);
        
    }
}
```



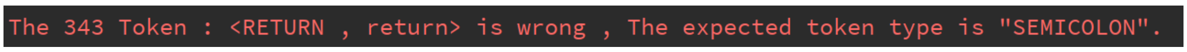
```
The 319 Token : <IDENTIFIER , extend> is wrong , The expected token type is "LBRACE".
```



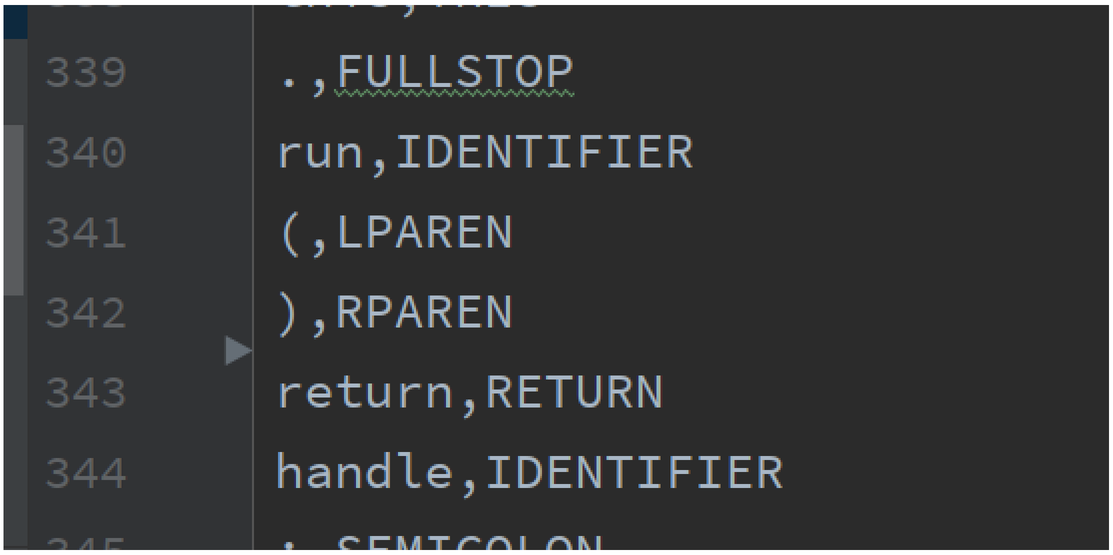
```
317  class,CLASS
318  MyClient,IDENTIFIER
319  extend,IDENTIFIER
320  {,LBRACE
321  public,PUBLIC
322  int,INT
```



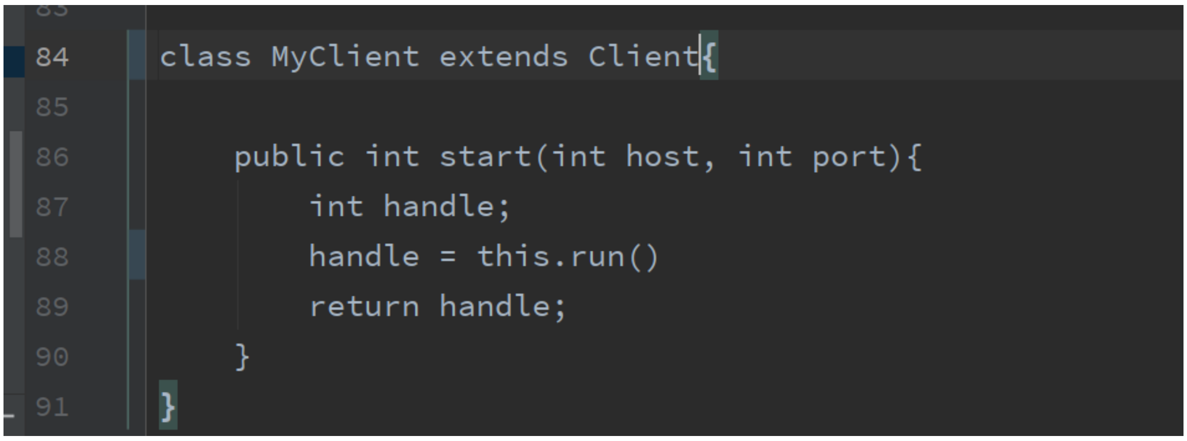
```
class MyClient extend{

    public int start(int host, int port){
        int handle;
        handle = this.run()
        return handle;
    }
}
```



```
The 343 Token : <RETURN , return> is wrong , The expected token type is "SEMICOLON".
```

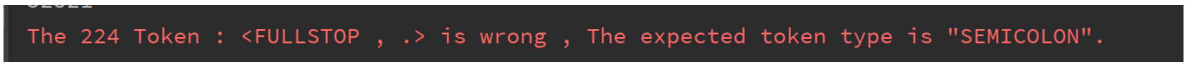


```
.,FULLSTOP
run,IDENTIFIER
(,LPAREN
),RPAREN
return,RETURN
handle,IDENTIFIER
```

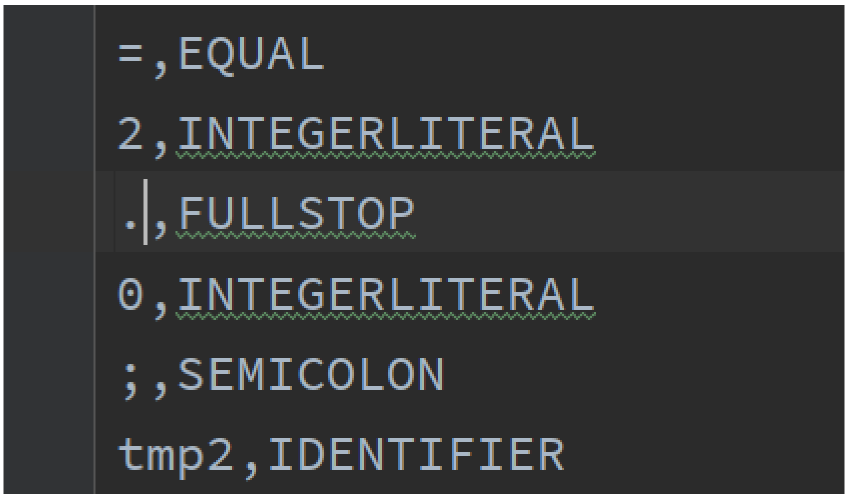


```
class MyClient extends Client{

    public int start(int host, int port){
        int handle;
        handle = this.run()
        return handle;
    }
}
```

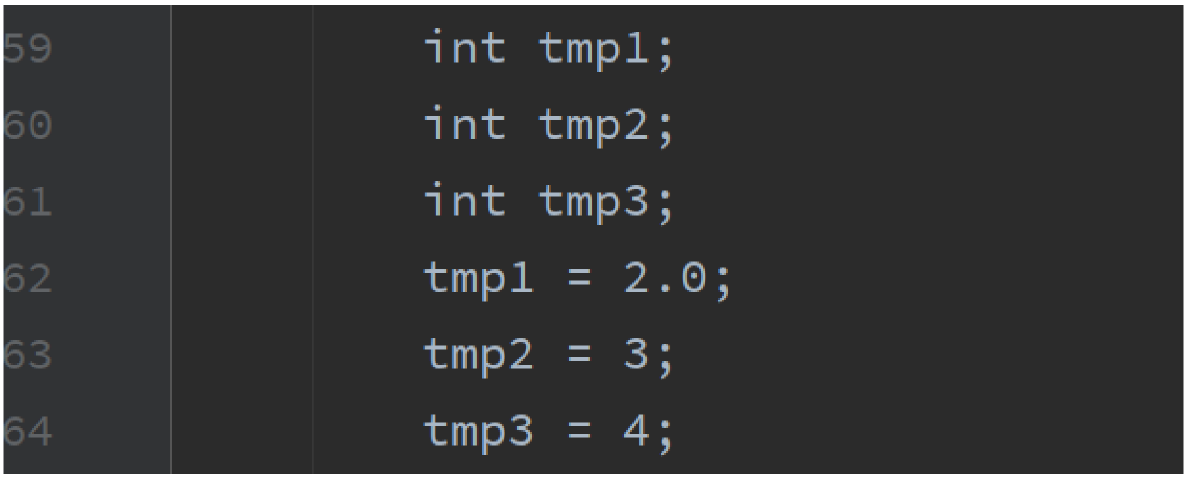
text12:



```
The 224 Token : <FULLSTOP , .> is wrong , The expected token type is "SEMICOLON".
```

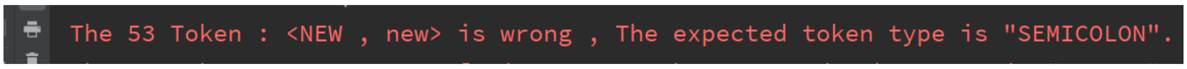


```
=,EQUAL
2,INTEGERLITERAL
.,FULLSTOP
0,INTEGERLITERAL
;,SEMICOLON
tmp2,IDENTIFIER
```

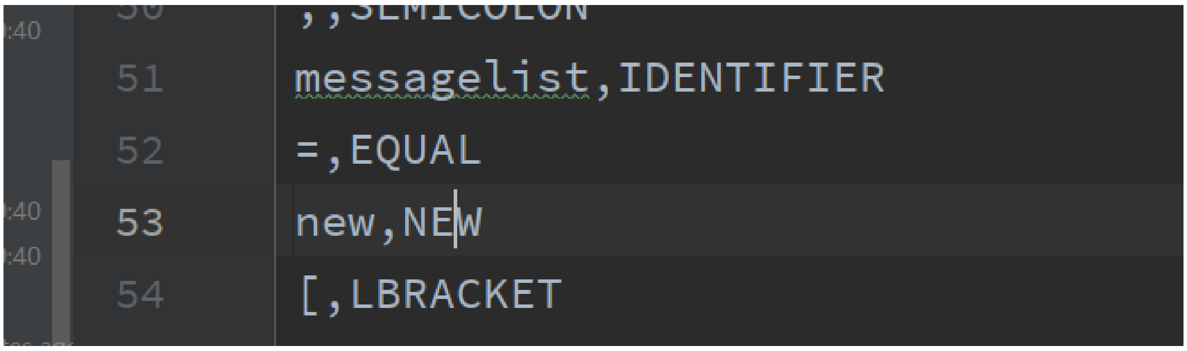


```
    int tmp1;
    int tmp2;
    int tmp3;
    tmp1 = 2.0;
    tmp2 = 3;
    tmp3 = 4;
```

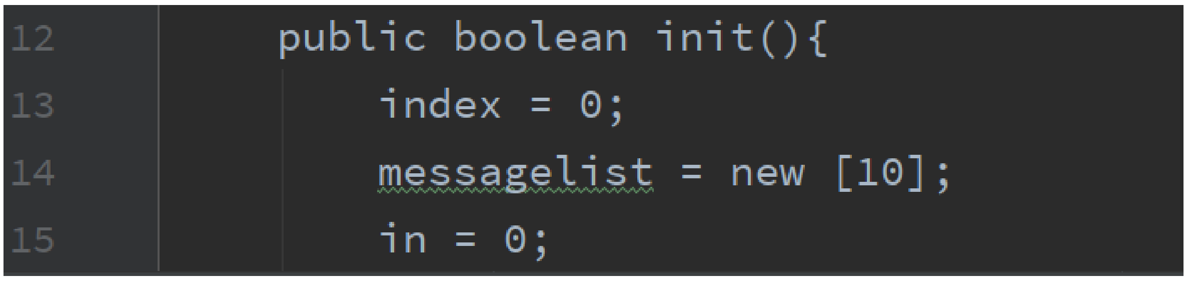
text13:



```
The 53 Token : <NEW , new> is wrong , The expected token type is "SEMICOLON".
```

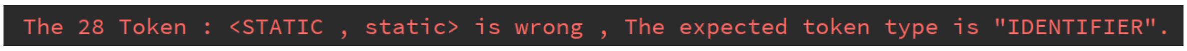


```
50  ;,SEMICOLON
51  messagelist,IDENTIFIER
52  =,EQUAL
53  new,NEW
54  [,LBRACKET
```

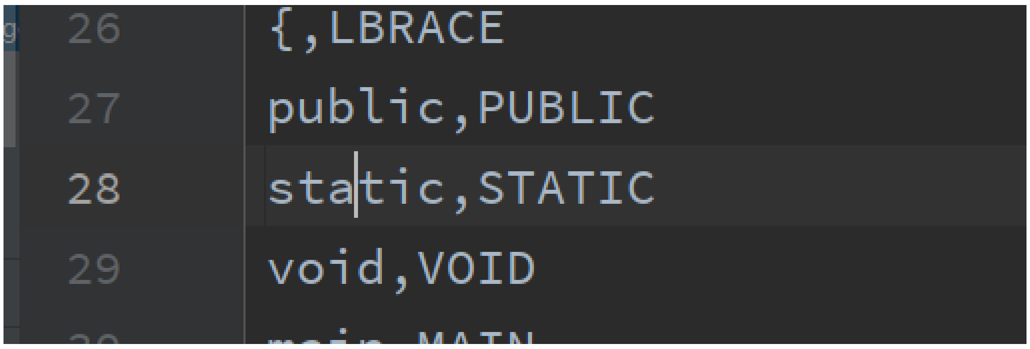


```
public boolean init(){
    index = 0;
    messagelist = new [10];
    in = 0;
```

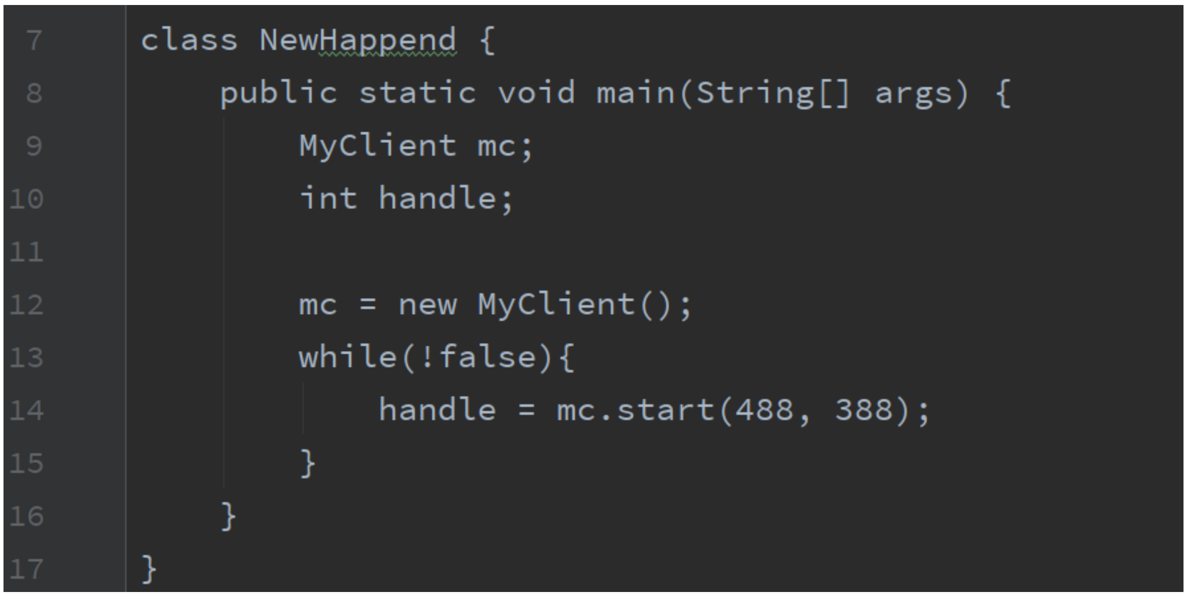
text15:



```
The 28 Token : <STATIC , static> is wrong , The expected token type is "IDENTIFIER".
```

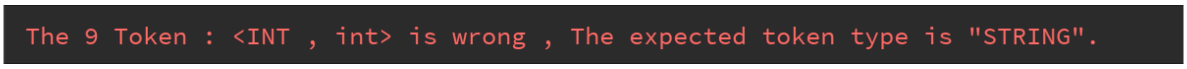


```
26  {,LBRACE
27  public,PUBLIC
28  static,STATIC
29  void,VOID
30  main,MAIN
```

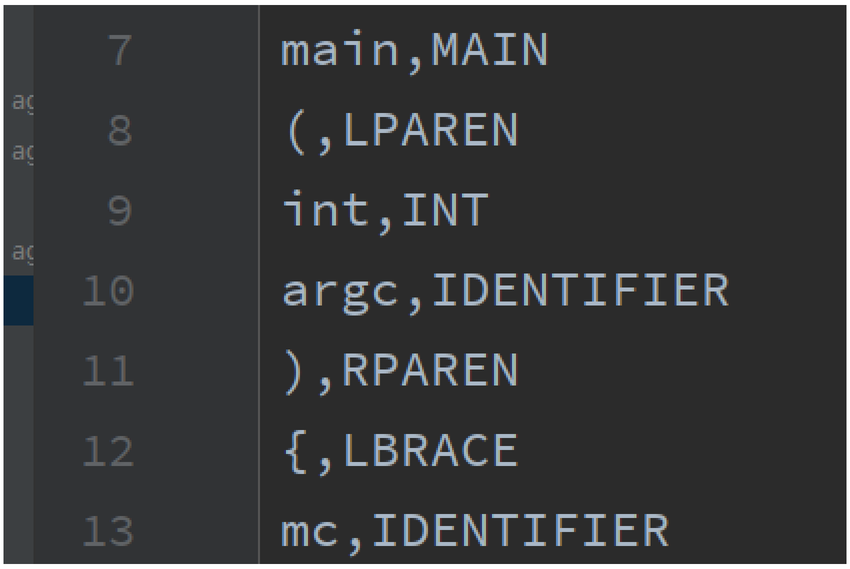


```
class NewHappend {
    public static void main(String[] args) {
        MyClient mc;
        int handle;

        mc = new MyClient();
        while(!false){
            handle = mc.start(488, 388);
        }
    }
}
```

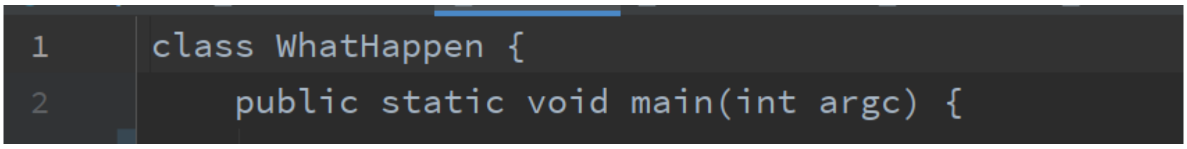
text16:



```
The 9 Token : <INT , int> is wrong , The expected token type is "STRING".
```

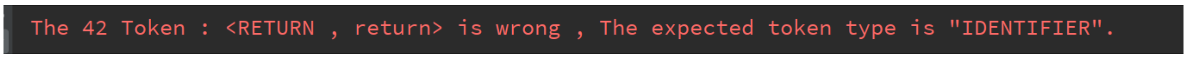


```
7    main,MAIN
8    (,LPAREN
9    int,INT
10   argc,IDENTIFIER
11   ),RPAREN
12   {,LBRACE
13   mc,IDENTIFIER
```

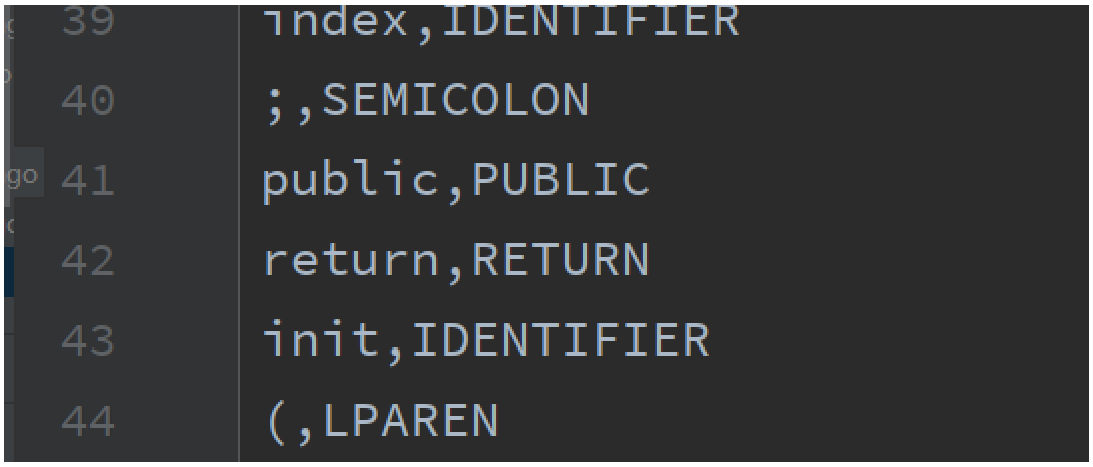


```
class WhatHappen {
    public static void main(int argc) {
```

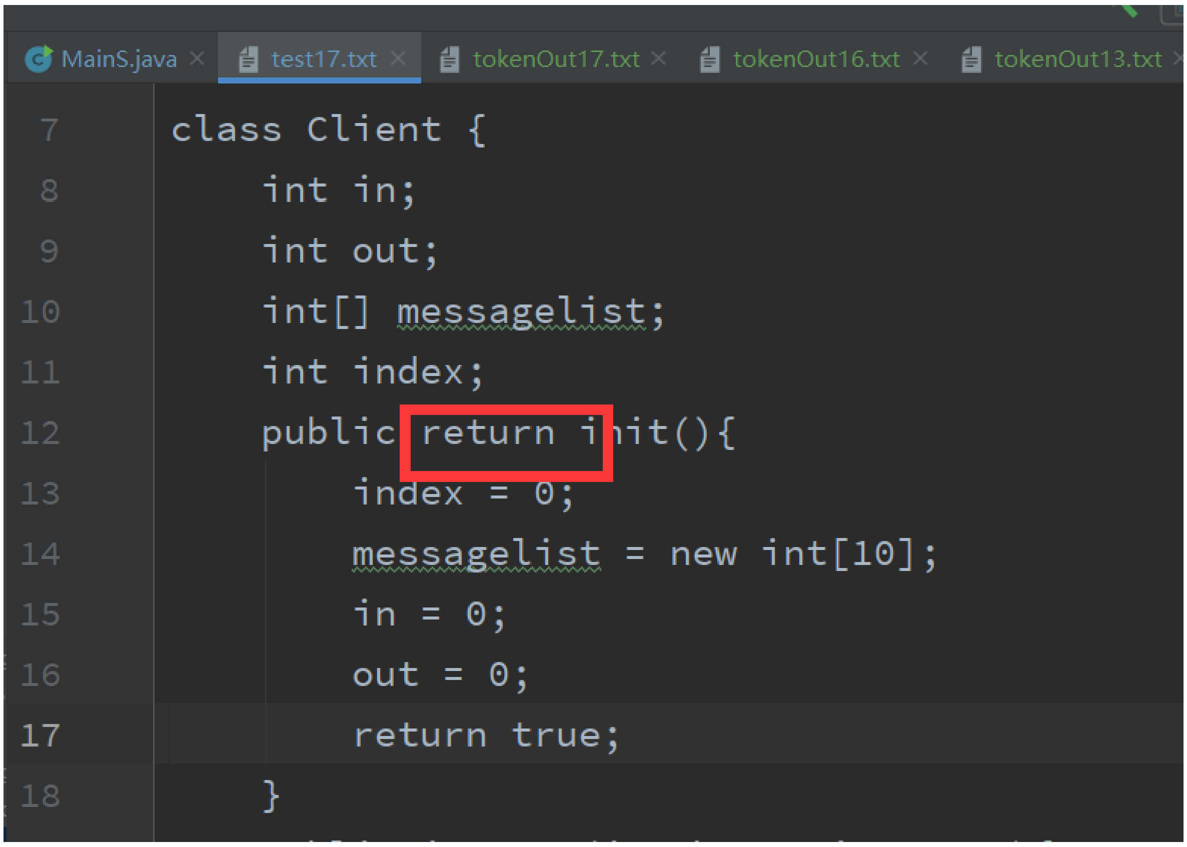
17



```
The 42 Token : <RETURN , return> is wrong , The expected token type is "IDENTIFIER".
```

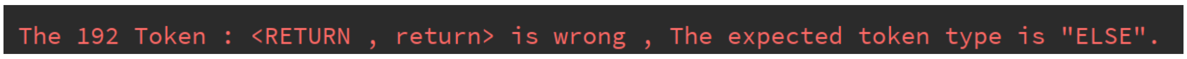


```
39   index,IDENTIFIER
40   ;,SEMICOLON
41   public,PUBLIC
42   return,RETURN
43   init,IDENTIFIER
44   (,LPAREN
```

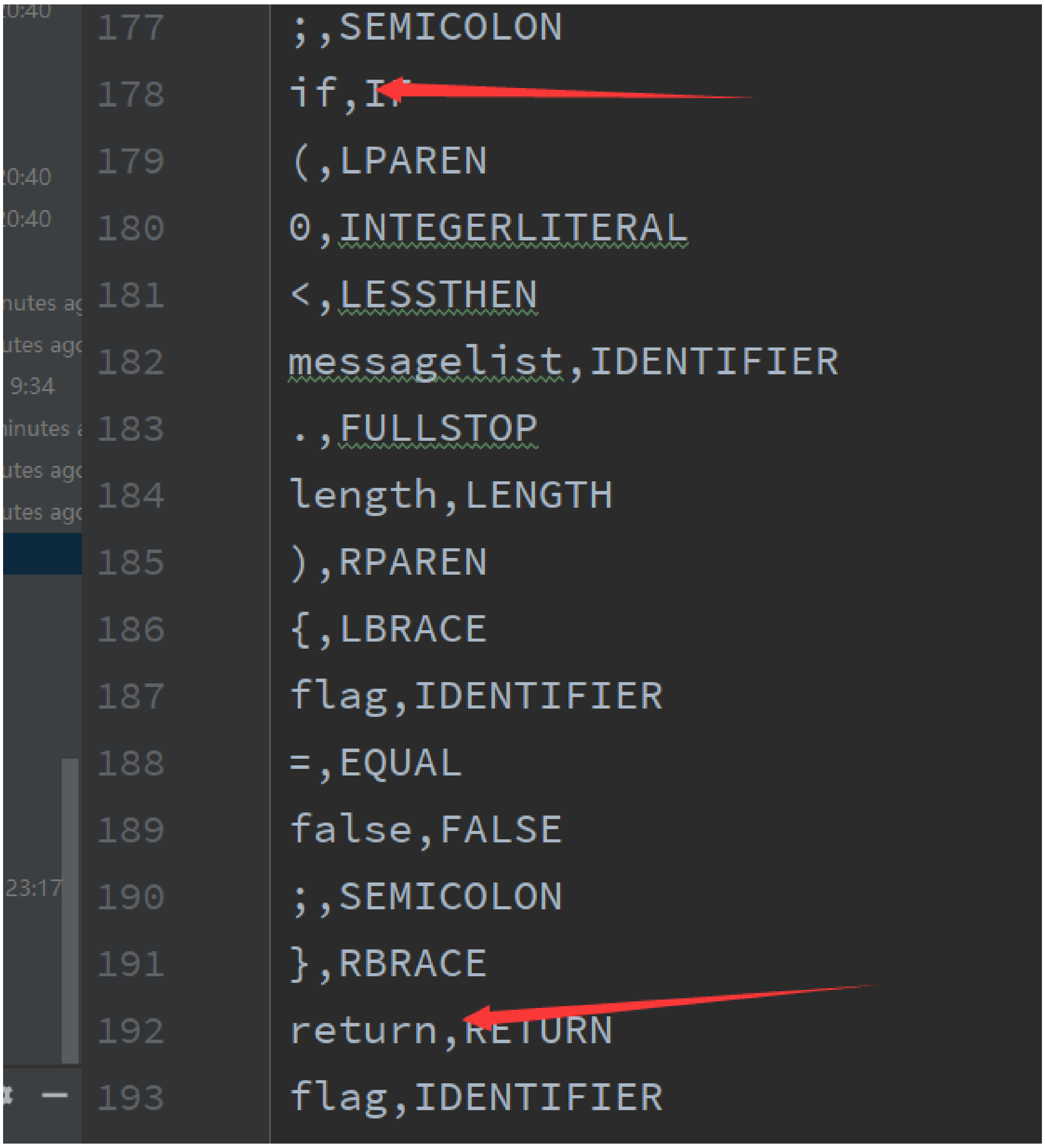


```
class Client {
    int in;
    int out;
    int[] messagelist;
    int index;
    public return init(){
        index = 0;
        messagelist = new int[10];
        in = 0;
        out = 0;
        return true;
    }
```

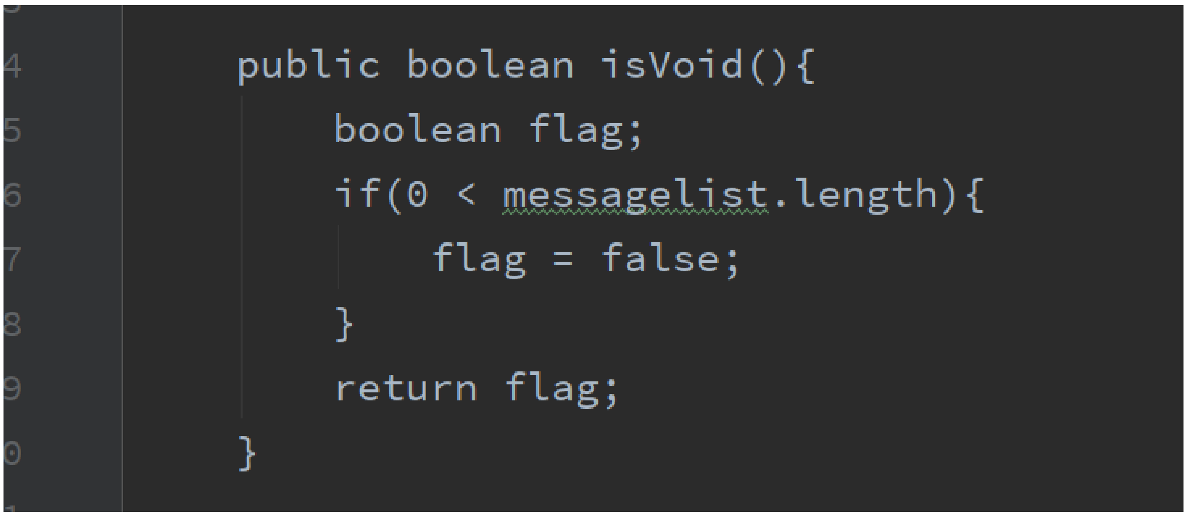
text18中：



```
The 192 Token : <RETURN , return> is wrong , The expected token type is "ELSE".
```

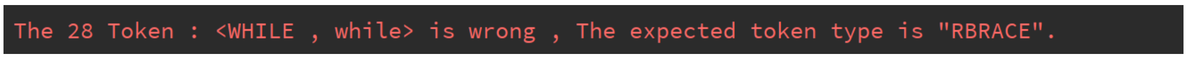


```
;,SEMICOLON
if,IF
(,LPAREN
0,INTEGERLITERAL
<,LESSTHEN
messagelist,IDENTIFIER
.,FULLSTOP
length,LENGTH
),RPAREN
{,LBRACE
flag,IDENTIFIER
=,EQUAL
false,FALSE
;,SEMICOLON
},RBRACE
return,RETURN
flag,IDENTIFIER
```

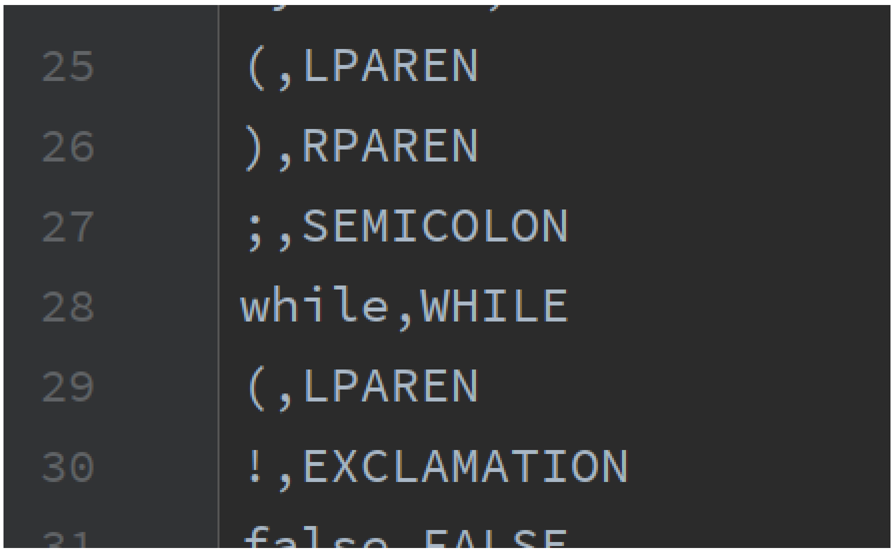


```
    public boolean isVoid(){
        boolean flag;
        if(0 < messagelist.length){
            flag = false;
        }
        return flag;
    }
```

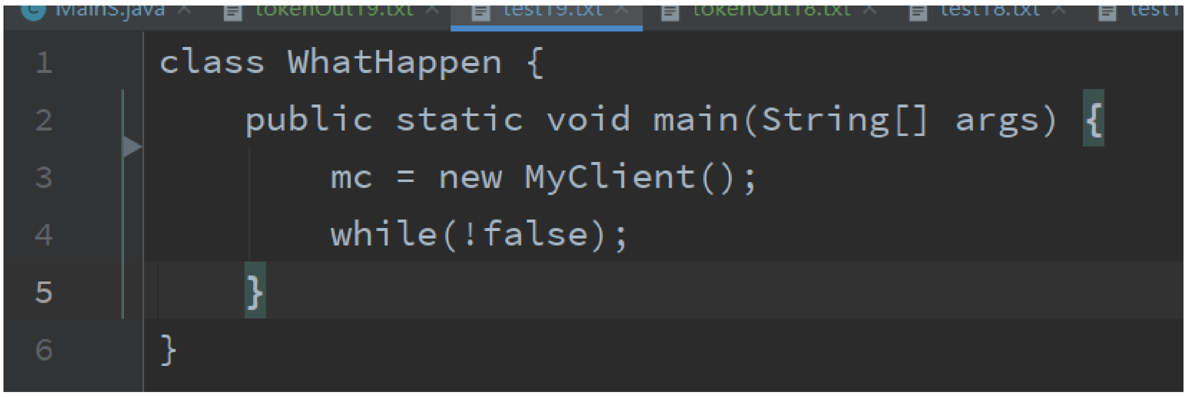
19



```
The 28 Token : <WHILE , while> is wrong , The expected token type is "RBRACE".
```

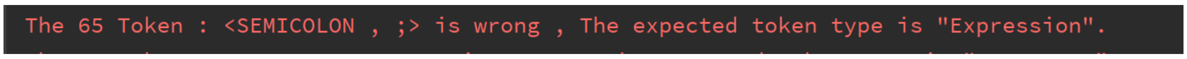


```
(,LPAREN
),RPAREN
;,SEMICOLON
while,WHILE
(,LPAREN
!,EXCLAMATION
```

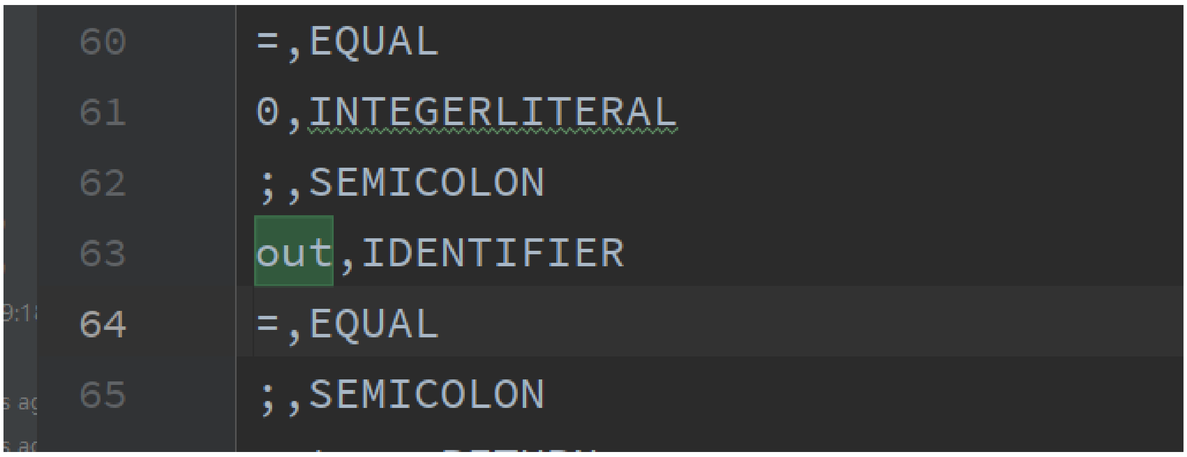


```
class WhatHappen {
    public static void main(String[] args) {
        mc = new MyClient();
        while(!false);
    }
}
```

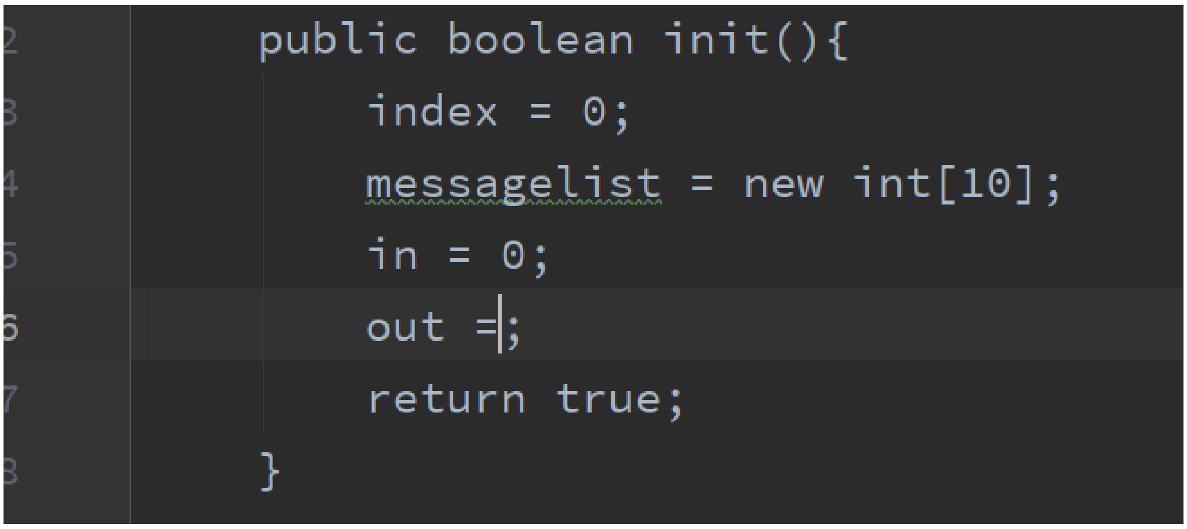
text20:



```
The 65 Token : <SEMICOLON , ;> is wrong , The expected token type is "Expression".
```

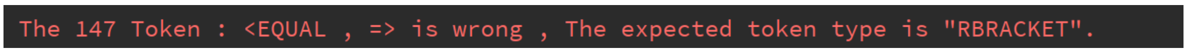


```
=,EQUAL
0,INTEGERLITERAL
;,SEMICOLON
out,IDENTIFIER
=,EQUAL
;,SEMICOLON
```

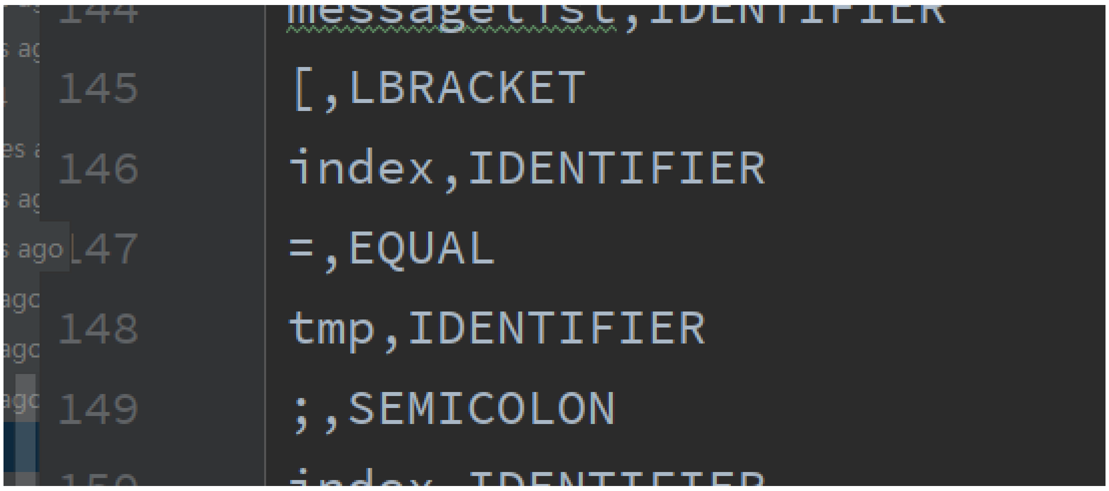


```
public boolean init(){
    index = 0;
    messagelist = new int[10];
    in = 0;
    out =;
    return true;
}
```

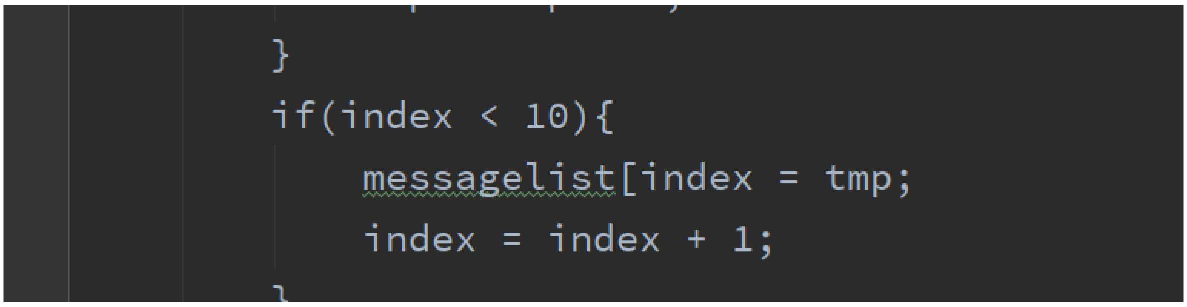
text21:



```
The 147 Token : <EQUAL , => is wrong , The expected token type is "RBRACKET".
```

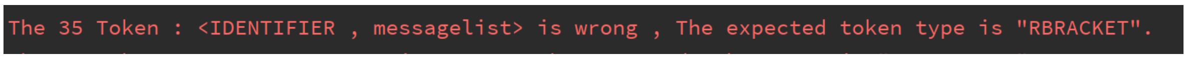


```
144  messagelist,IDENTIFIER
145  [,LBRACKET
146  index,IDENTIFIER
147  =,EQUAL
148  tmp,IDENTIFIER
149  ;,SEMICOLON
150  index,IDENTIFIER
```

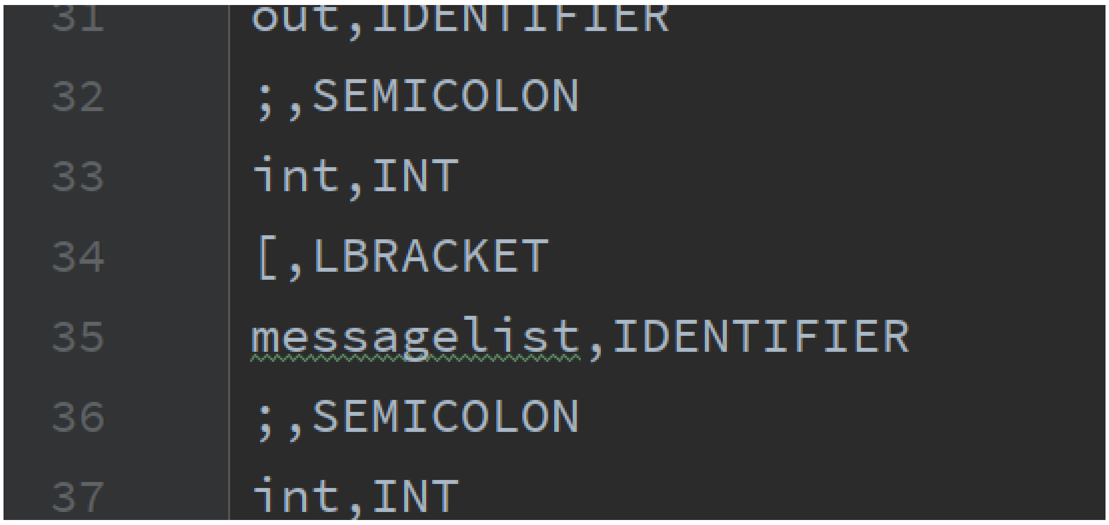


```
    }
    if(index < 10){
        messagelist[index = tmp;
        index = index + 1;
    }
```

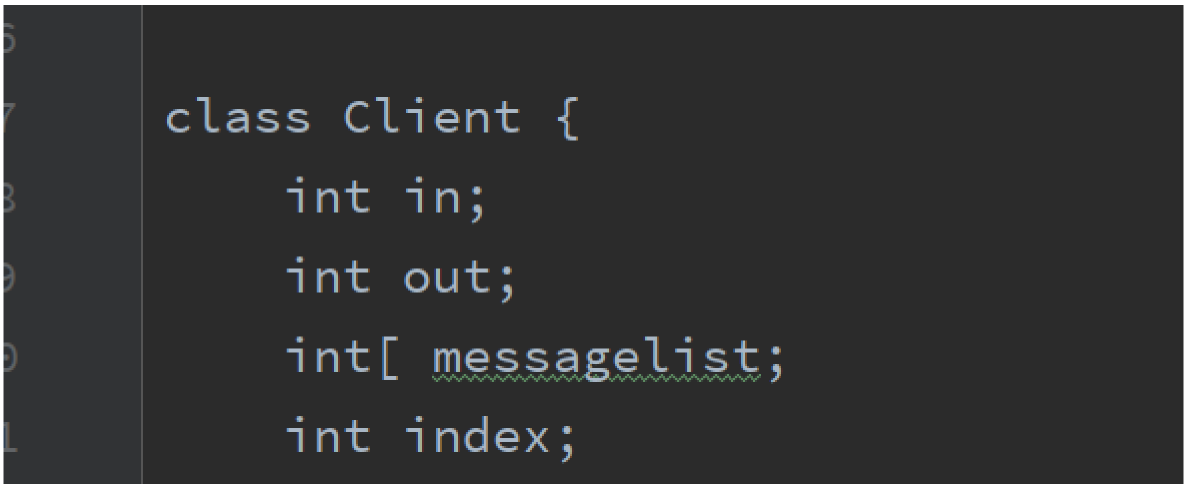
text22:



```
The 35 Token : <IDENTIFIER , messagelist> is wrong , The expected token type is "RBRACKET".
```

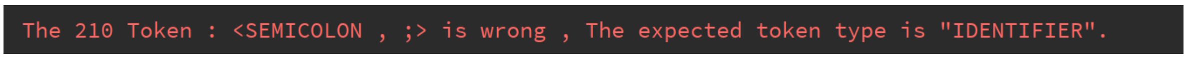


```
31  out,IDENTIFIER
32  ;,SEMICOLON
33  int,INT
34  [,LBRACKET
35  messagelist,IDENTIFIER
36  ;,SEMICOLON
37  int,INT
```

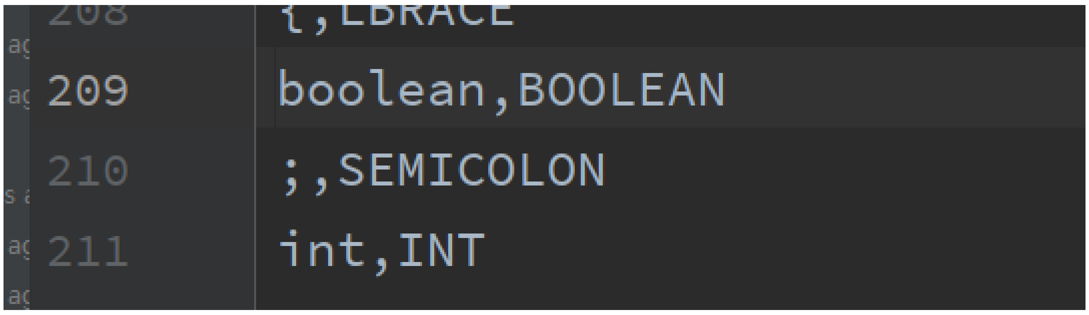


```
class Client {
    int in;
    int out;
    int[ messagelist;
    int index;
```

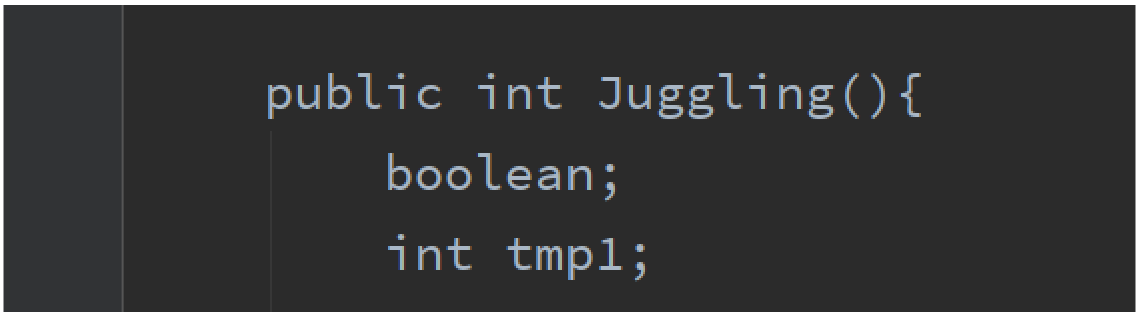
text23：



```
The 210 Token : <SEMICOLON , ;> is wrong , The expected token type is "IDENTIFIER".
```

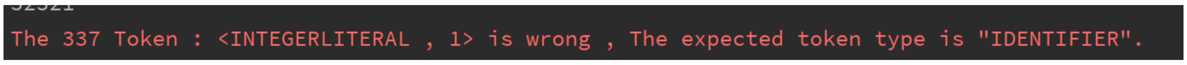


```
208    {,LBRACE
209    boolean,BOOLEAN
210    ;,SEMICOLON
211    int,INT
```

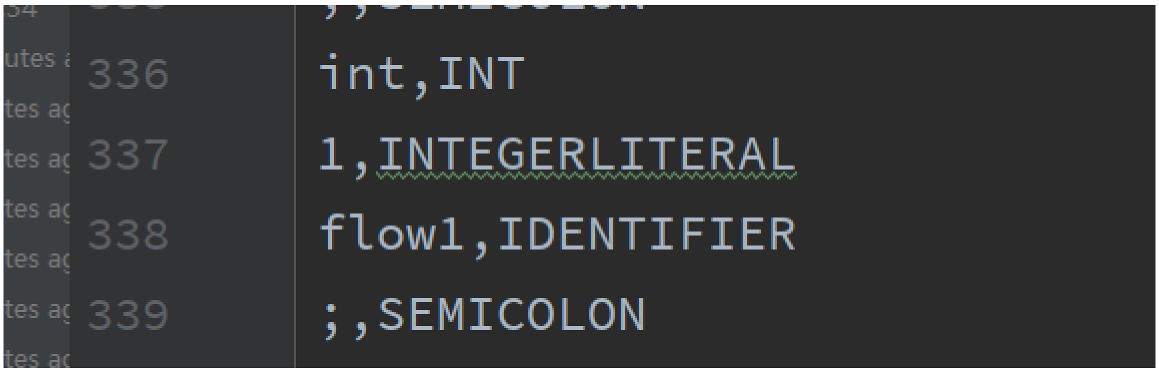


```
    public int Juggling(){
        boolean;
        int tmp1;
```

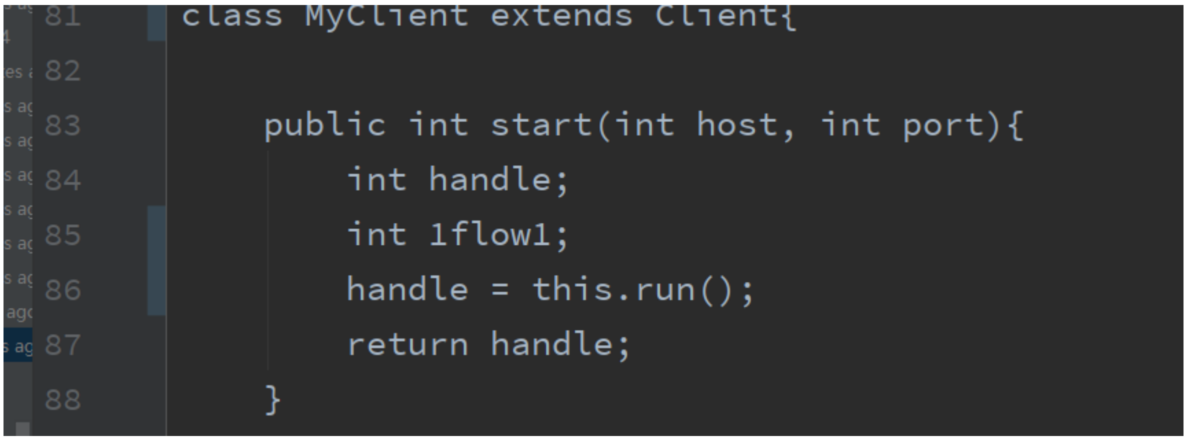
再text23中自己添加的：



```
The 337 Token : <INTEGERLITERAL , 1> is wrong , The expected token type is "IDENTIFIER".
```

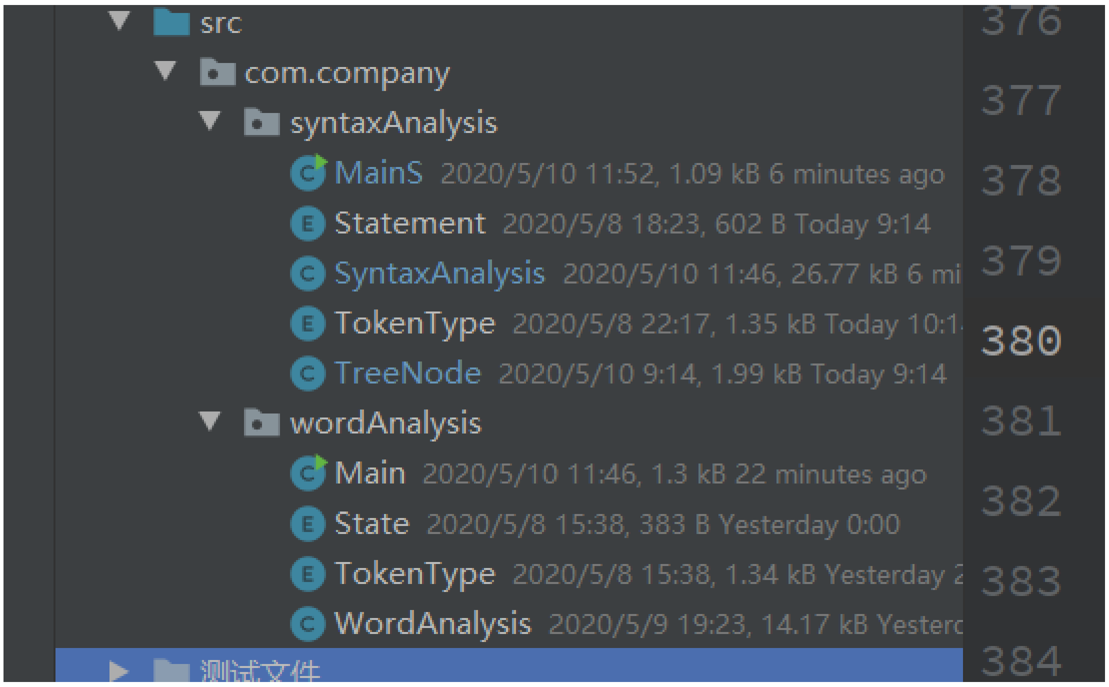


```
       ;,SEMICOLON
336    int,INT
337    1,INTEGERLITERAL
338    flow1,IDENTIFIER
339    ;,SEMICOLON
```



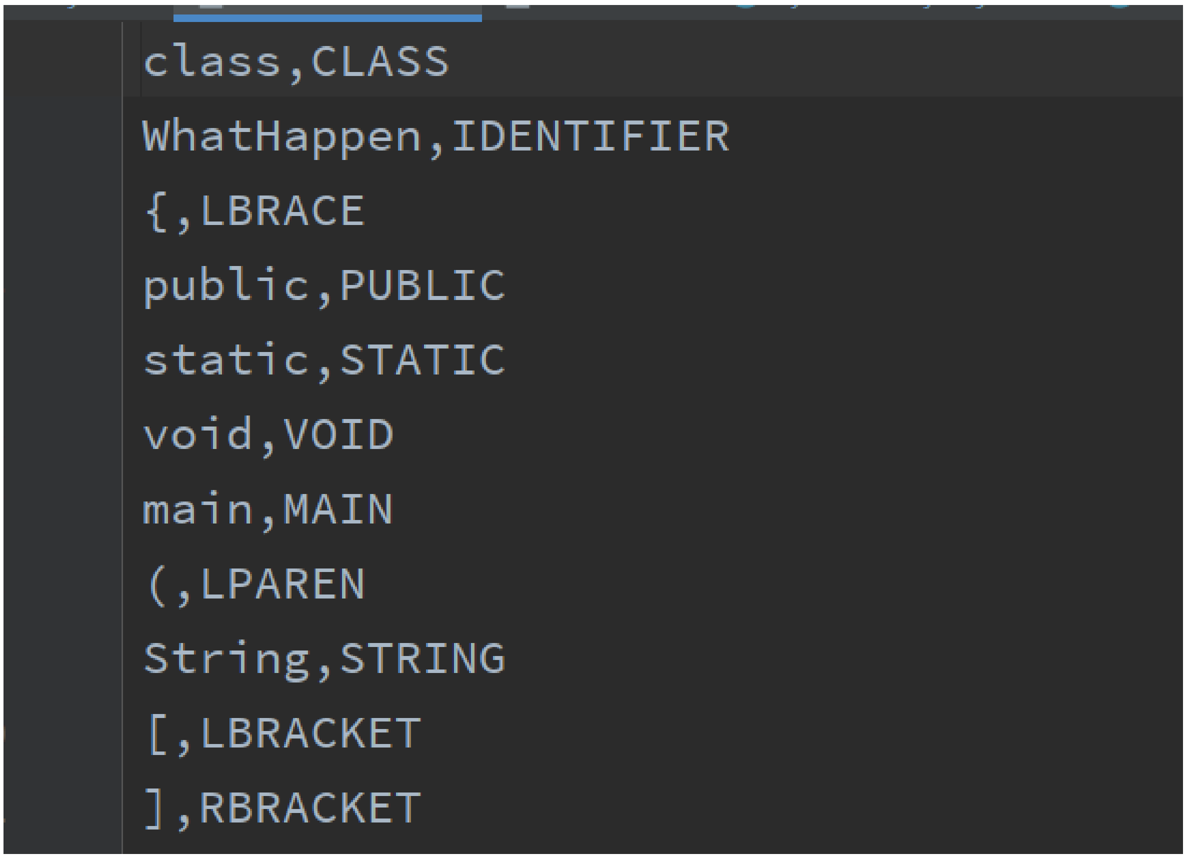
```
class MyClient extends Client{

    public int start(int host, int port){
        int handle;
        int 1flow1;
        handle = this.run();
        return handle;
    }
```

**代码结构**



```
▼ src
  ▼ com.company
    ▼ syntaxAnalysis
        MainS  2020/5/10 11:52, 1.09 kB 6 minutes ago
        Statement  2020/5/8 18:23, 602 B Today 9:14
        SyntaxAnalysis  2020/5/10 11:46, 26.77 kB 6 mi
        TokenType  2020/5/8 22:17, 1.35 kB Today 10:1
        TreeNode  2020/5/10 9:14, 1.99 kB Today 9:14
    ▼ wordAnalysis
        Main  2020/5/10 11:46, 1.3 kB 22 minutes ago
        State  2020/5/8 15:38, 383 B Yesterday 0:00
        TokenType  2020/5/8 15:38, 1.34 kB Yesterday 2
        WordAnalysis  2020/5/9 19:23, 14.17 kB Yesterc
  ▶ 测试文件
```

在第一次实验的基础上对词法分析的结果进行了一些修改，输出的样子为



```
class,CLASS
WhatHappen,IDENTIFIER
{,LBRACE
public,PUBLIC
static,STATIC
void,VOID
main,MAIN
(,LPAREN
String,STRING
[,LBRACKET
],RBRACKET
```

syntaxAnalysis

Statement是所有语法树的根节点的状态，

SyntaxAnalysis：语法分析类

TokenType 是词的类型

TreeNode是语法树的结构