测试

MainClass

测试语句如下

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

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

测试结果：

```
>>>Syntax error Invalid statement at line 4

>>>Syntax error Invalid Token at line 5

>>>Syntax error Invalid statement at line 5

```

说明：

mainclass中语法定义中为一个statement

line 4中`MyClient mc;`为一个declaration语句，与statement不相符，抛出Invalid statement错误

line 5中进入statement语句时，当token为*int*时没有捕捉到合法的statement关键字，所以抛出了Invalid Token错误，继续读入下一个token handle后发现这是一个identifier，在statement的First集中，进入identifier_Statement节点，但是下一个Token为SEMICOLON不符合语法规定，故抛出Invalid statement错误。

在接下来的语句中测试了赋值语句以及while_Statement，由于下面的语句均符合语法，语法分析器未抛出错误。

ClassDeclaration

测试语句如下

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

```
    }
}

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

测试结果：

```
>>>Syntax error unexpected Token at line 9
```

说明：

在这个测试文件中，我再ClassDeclaration设置了错误，即：line 9中extends后应当跟随一个identifier，在语句中缺少了该项，经过语法分析器检查出了该错误。

VarDeclaration 与 MethodDeclaration

测试语句如下：

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

class MyClient{
    public start(int host, int port){
        int handle;
        boolean test
        boolean test/
        boolean test+
        return handle;
    }
}
```

测试结果：

```

>>>Syntax error unexpected token: at line 10
(
>>>Syntax error unexpected token: at line 13
boolean
>>>Syntax error get illegal token at line 13

>>>Syntax error get unexpected symbol at line 14

```

说明：

第十行处，methodDeclaration中应该有一个Type的定义，这里没有Type而直接出现了(，所以抛出了unexpected token的信息。在第12行缺少一个分号，故到达下一个token boolean是出现unexpected token错误。在13行结束，应当为一个分号这里出现了一个不被支持的/，所以出现了illegal token错误。第14行出现了一个错误符号，得到了unexpected symbol的错误。

Statement与Expression

测试语句如下：

```
class WhatHappen {
    public static void main(String[] args) {
        while(true){
            System.out.println(123);
        }
        test = 1;
    }
}
```

测试结果：

```

>>>Syntax error unexpected token: at line 6
test
>>>Syntax error unexpected token: at line 6
test
>>>Syntax error unexpected token: at line 6
test
>>>Syntax error unexpected token: at line 6
=
>>>Syntax error unexpected VarType at line 6

>>>Syntax error unexpected token: at line 6
1
>>>Syntax error unexpected token: at line 8
}
>>>Syntax error unexpected token: at line 8
}
>>>Syntax error unexpected token: at line 8
}
```

说明：

语法规定，连续的statement需要使用{}包裹，否则在mainclass中只能出现一个statement。在这里去掉了{}包裹，test = 1为非法语句，根据正确的识别应当进入classdeclaration当中。根据classdeclaration语法判断出现了如测试结果当中所示的错误信息。当我们使用正确的{}包裹后，语句如下：

```
class WhatHappen {
    public static void main(String[] args) {
        {
            while(true){
                System.out.println(123);
            }
            test = 1;
        }
    }
}
```

注意第3行与第8行的{}，再包裹语句后符合语法要求，语法分析器未报错。

下面测试expression

语句如下：

```
class WhatHappen {
    public static void main(String[] args) {
            while(true){
                System.out.println(123);
                System.out.println(new mc().length);
                System.out.println(new ());
            }
    }
}
```

测试结果

```

>>>Syntax error Unexpected token at line 6

>>>Syntax error unexpected token: at line 6
)
>>>Syntax error Invalid Token at line 6
```

在这里我是用了System.out.peintln中的expression语句来测试相关语法。如上所示，在第六行当中的new后应当跟随一个identifier，在这里缺少了，所以语法分析器根据new identifier ()的检查规则在这里抛出了三个错误，分别对应identifier （ ）的三个位置。

完全正确的语法与抽象语法树

接下来我使用一段正确的语句打印其抽象语法树：

```
class WhatHappen {
    public static void main(String[] args) {
            while(true){
                System.out.println(a);
            }
    }
}
class MyClient extends Client{
    public int start(int host, int port){
        {
            handle = this.run();
```

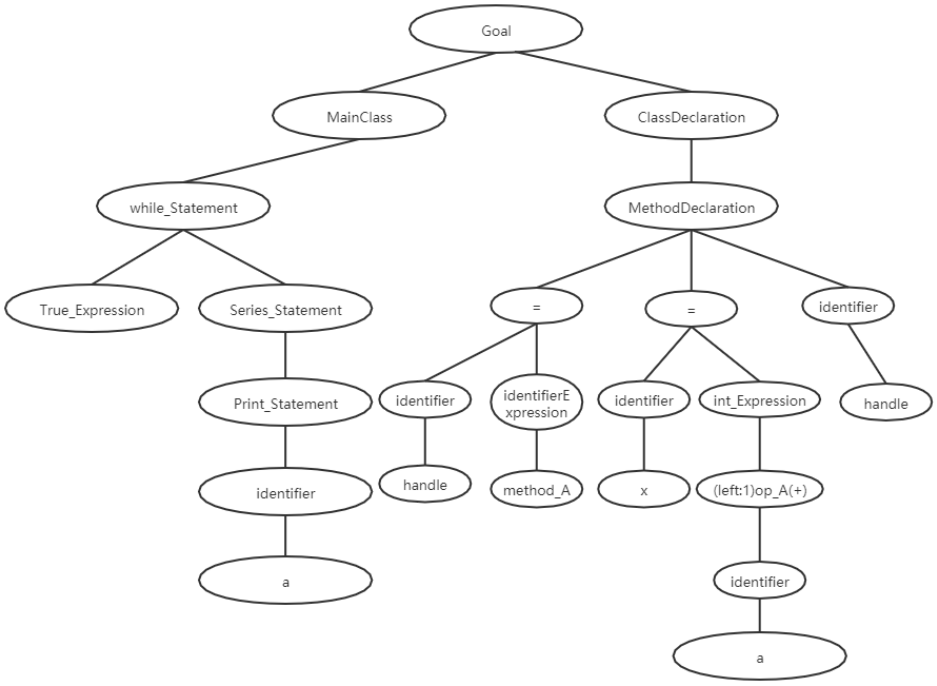
```
            x = 1 + a ;
        }
        return handle;
    }
}
```

AST:

```
第0层
 Type: Goal
第1层
 Type: MainClass
第2层
 Type: While_Statement
第3层
 Type: True_Expression
第3层
 Type: Series_Statement
第4层
 Type: Print_Statement
第5层
 Type: Identifier_Expression
第1层
 Type: ClassDeclaration
第2层
 Type: MethodDeclaration
第3层
 Type: VarAssign_Statement Target val: handle
第4层
 Type: Identifier_Expression
第5层
 Type:method_A
第3层
 Type: VarAssign_Statement Target val: x
第4层
 Type: Int_Expression
第5层
 Type:op_A opreator: + left: 1
第6层
 Type: Identifier_Expression
第3层
 Type: Identifier_Expression

```

根据程序输出的AST可以得到以下抽象语法树：


Goal
MainClass
ClassDeclaration
while_Statement
MethodDeclaration
True_Expression
Series_Statement
=
=
identifier
Print_Statement
identifier
identifierExpression
identifier
int_Expression
handle
identifier
handle
method_A
x
(left:1)op_A(+)
identifier
a
a
第0层
第1层
第2层
第3层
第4层
第5层