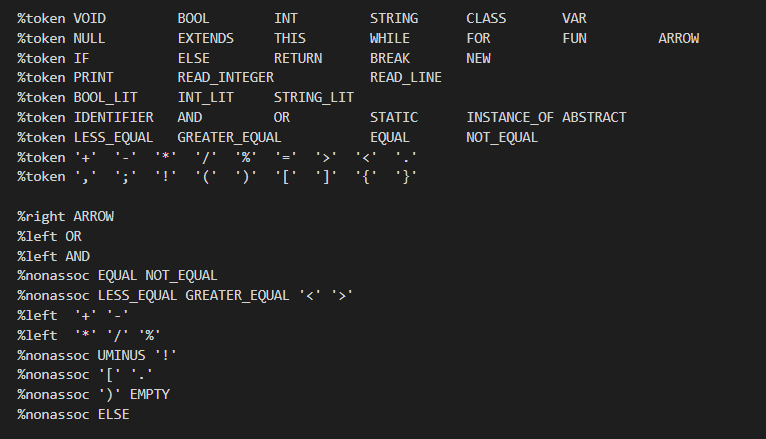
**PA1-A实验报告**

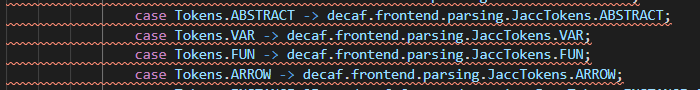
**实验前导：注册关键字和运算符，涉及五个文件——**

三次实验涉及到关键字ABSTRACT（抽象类）、VAR（局部变量）、FUN（Lambda表达式）、ARROW（=>运算符）

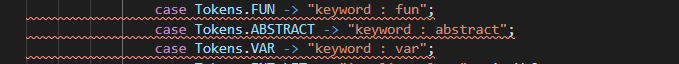
src/main/jack/Decaf.jacc:将关键字添加到token中



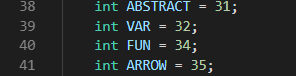
src/main/java/frontend/parsing/JaccParser.java:



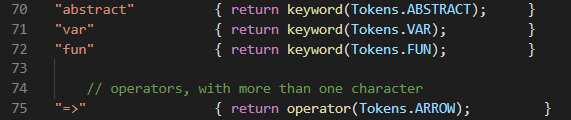
src/main/java/frontend/parsing/SemValue. java:

src/main/java/frontend/parsing/Tokens. java:（注意需要避过单字符的ASCII码）



src/main/jflex/Decaf.jflex:



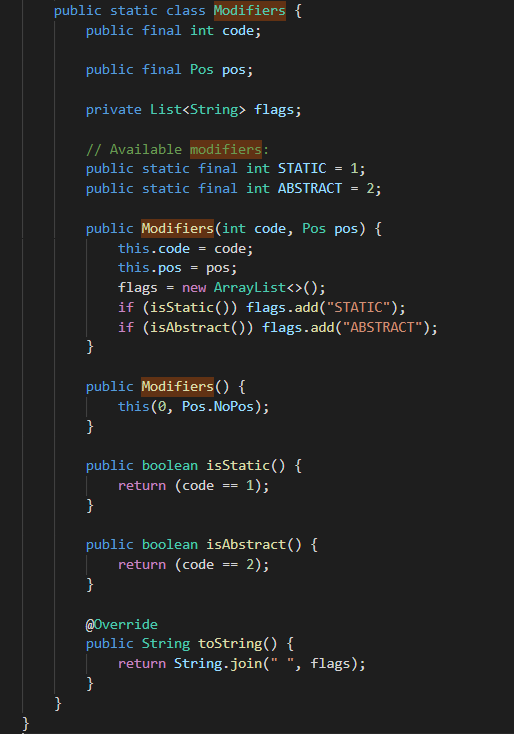
**任务1：抽象类**

1. 任务要求与指导

根据语法规范中的classDef和methodDef的改动，可知需要对Tree.java中的classDe类和methodDef类进行改动。按照上述过程完成关键字ABSTRACT的注册。核心思想是抽象类和静态类的关键字用法相似，只要在对static进行定义的地方进行重复操作即可。



1. 修改Modifiers类加入ABSTRACT关键字，仿照static，设置静态常数ABSTRACT并令其为2，同时增加isAbstract函数并修改构造函数。具体代码如图：



1. 修改classDef函数和methodDef函数。

（1）对前者，由表达式可知，只需添加关键字ABSTRACT即可。将是否是ABSTRACT作为参数传入classDef的构造函数中，并由此确定Modifiers的内容。

1. /\*\*
2. \* Class definition.
3. \* <pre>
4. \*     'class' id {'extends' parent}? '{' fields '}'
5. \* </pre>
6. \*/
7. public static class ClassDef extends TreeNode {
8. // Tree elements
9. public Modifiers modifiers;
10. public final Id id;
11. public Optional<Id> parent;
12. public final List<Field> fields;
13. // For convenience
14. public final String name;
15. public ClassDef(Id id, Optional<Id> parent, List<Field> fields, Pos pos, Boolean isAbstract) {
16. super(Kind.CLASS\_DEF, "ClassDef", pos);
17. this.id = id;
18. this.parent = parent;
19. this.fields = fields;
20. this.name = id.name;
21. if(isAbstract)
22. this.modifiers = new Modifiers(Modifiers.ABSTRACT, pos);
23. else
24. this.modifiers = new Modifiers();
25. }
26. public boolean isAbstract() {
27. return modifiers.isAbstract();
28. }
29. public boolean hasParent() {
30. return parent.isPresent();
31. }
32. public List<MethodDef> methods() {
33. var methods = new ArrayList<MethodDef>();
34. for (var field : fields) {
35. if (field instanceof MethodDef) {
36. methods.add((MethodDef) field);
37. }
38. }
39. return methods;
40. }
41. @Override
42. public Object treeElementAt(int index) {
43. return switch (index) {
44. case 0 -> modifiers;
45. case 1 -> id;
46. case 2 -> parent;
47. case 3 -> fields;
48. default -> throw new IndexOutOfBoundsException(index);
49. };
50. }
51. @Override
52. public int treeArity() {
53. return 4;
54. }
55. @Override
56. public <C> void accept(Visitor<C> v, C ctx) {
57. v.visitClassDef(this, ctx);
58. }
59. }

（2）对于后者，本质上仍是模仿static的操作，观察到isStatic作为参数传入构造函数中，于是也设置布尔变量isAbstract来指示。此处将body设为optionnal<block>的目的是方便在输出的时候完成抽象方法的特殊性（没有函数体）。

/\*\*

     \* Member method definition.

     \* <pre>

     \*     'static'? returnType id '(' type1 id1 ',' type2 id2 ',' ... ')' '{' body '}'

     \* </pre>

     \* Decaf has static methods but NO static variables, strange!

     \*/

    public static class MethodDef extends Field {

        // Tree elements

        public Modifiers modifiers;

        public Id id;

        public TypeLit returnType;

        public List<LocalVarDef> params;

        public Optional<Block> body;

        // For convenience

        public String name;

        public MethodDef(boolean isStatic, boolean isAbstract, Id id, TypeLit returnType, List<LocalVarDef> params, Optional<Block> body, Pos pos) {

            super(Kind.METHOD\_DEF, "MethodDef", pos);

            //this.modifiers = isStatic ? new Modifiers(Modifiers.STATIC, pos) : new Modifiers();

            this.id = id;

            this.returnType = returnType;

            this.params = params;

            this.body = body;

            this.name = id.name;

            if(isStatic)

                this.modifiers = new Modifiers(Modifiers.STATIC, pos);

            else if(isAbstract)

                this.modifiers = new Modifiers(Modifiers.ABSTRACT, pos);

            else

                this.modifiers = new Modifiers();

        }

        public boolean isStatic() {

            return modifiers.isStatic();

        }

        public boolean isAbstract() {

            return modifiers.isAbstract();

        }

        @Override

        public Object treeElementAt(int index) {

                return switch (index) {

                    case 0 -> modifiers;

                    case 1 -> id;

                    case 2 -> returnType;

                    case 3 -> params;

                    case 4 -> body;

                    default -> throw new IndexOutOfBoundsException(index);

                };

        }

        @Override

        public int treeArity() {

            return 5;

        }

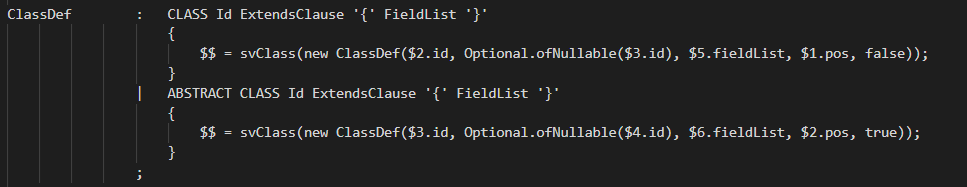
        public <C> void accept(Visitor<C> v, C ctx) {

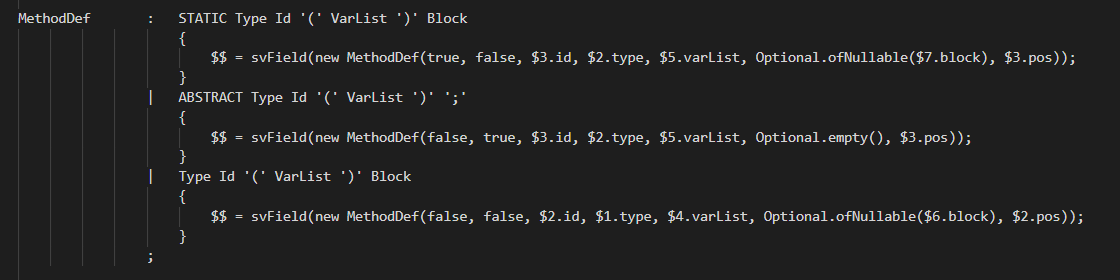
            v.visitMethodDef(this, ctx);

        }

    }

（3）上述过程体现在Decaf.jacc中为（需要注意的是$k的k需要随着输入格式的变化而变化）：





4.心得与困难：

（1）一开始没注意ABSTRACT应用在classDef和methodDef中的微小不同，在methodDef中抽象函数没有函数体应以’,’结尾，仍然下意识地写成了block，造成了错误。

（2）开始理解optional类在输出格式中的应用。

**任务2：局部类型推断**

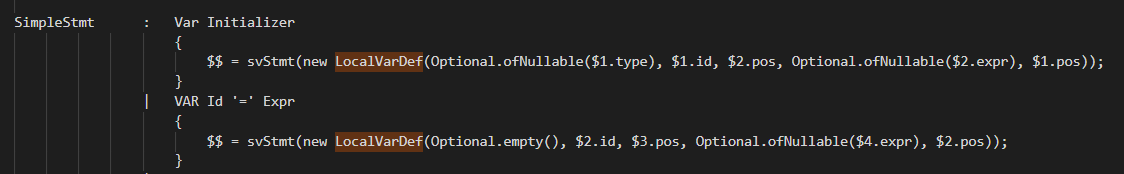
1. 任务要求与指导：

根据语法规范，可以知道需要对simplestmt进行改动。仍然需要先行注册关键字VAR。根据输出要求可知此处需要用到Optional类来使得第一个输出为<none>。



1. 修改LocalVarDef类的两个构造方法，使第一个变量变为Optional<TypeLit>。
2. public static class LocalVarDef extends Stmt {
3. // Tree elements
4. public Optional<TypeLit> typeLit;
5. public Id id;
6. public Pos assignPos;
7. public Optional<Expr> initVal;
8. // For convenience
9. public String name;
10. public LocalVarDef(Optional<TypeLit> typeLit, Id id, Pos assignPos, Optional<Expr> initVal, Pos pos) {
11. // pos = id.pos, assignPos = position of the '='
12. // TODO: looks not very consistent, maybe we shall always report error simply at `pos`, not `assignPos`?
13. super(Kind.LOCAL\_VAR\_DEF, "LocalVarDef", pos);
14. this.typeLit = typeLit;
15. this.id = id;
16. this.assignPos = assignPos;
17. this.initVal = initVal;
18. this.name = id.name;
19. }
20. public LocalVarDef(Optional<TypeLit> typeLit, Id id, Pos pos) {
21. this(typeLit, id, Pos.NoPos, Optional.empty(), pos);
22. }
23. @Override
24. public Object treeElementAt(int index) {
25. return switch (index) {
26. case 0 -> typeLit;
27. case 1 -> id;
28. case 2 -> initVal;
29. default -> throw new IndexOutOfBoundsException(index);
30. };
31. }
32. @Override
33. public int treeArity() {
34. return 3;
35. }
36. @Override
37. public <C> void accept(Visitor<C> v, C ctx) {
38. v.visitLocalVarDef(this, ctx);
39. }
40. }

3.修改Decaf.jacc中的表达式。



4.困难与心得：

（1）对于所有有LocalVarDef的表达式，都需要将第一个参数改为Optional.pfNullable或者Optional.empty，否则会出现编译不通过的问题。

**任务3：First-class Functions**

*1°函数类型*

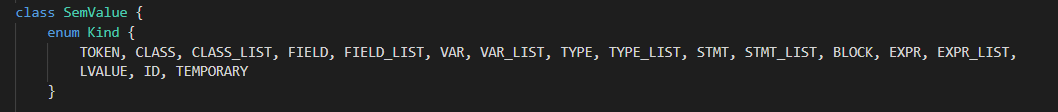
1. 任务要求和指导：

需要实现两种新的表达式，分别是类似于构造函数的type = type(type…) 和 迭代形式的 typeList = (type, type, …)。由于含有正则表达式，参照varList等类的写法，添加了typeList类以及对应的支持。

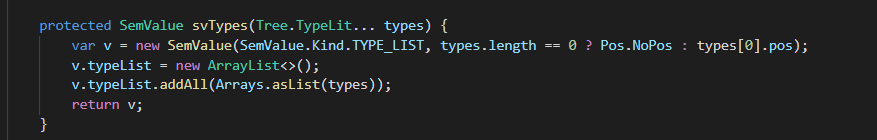


1. 实验步骤

在SemValue.Kind中注册:



在AbstractParser.java中加入定义svTypes：

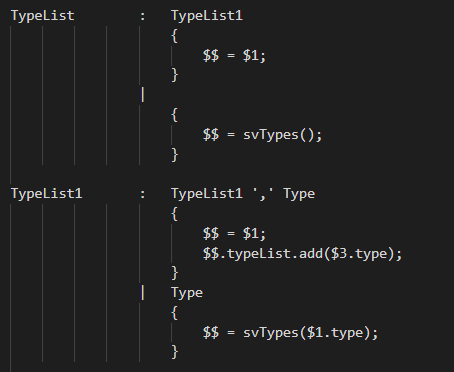


接着修改Decaf.jacc。根据表达式的格式，可以用上下文无关文法表达如下：

S -> (S) | A

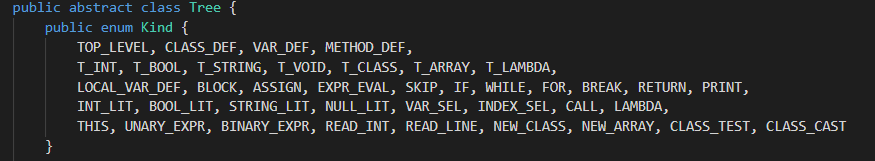
A -> type | type ‘,’ A

用表达式进行表达如图：

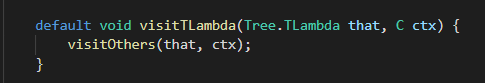


至此完成了typeList = (type, type, …)的表达式。

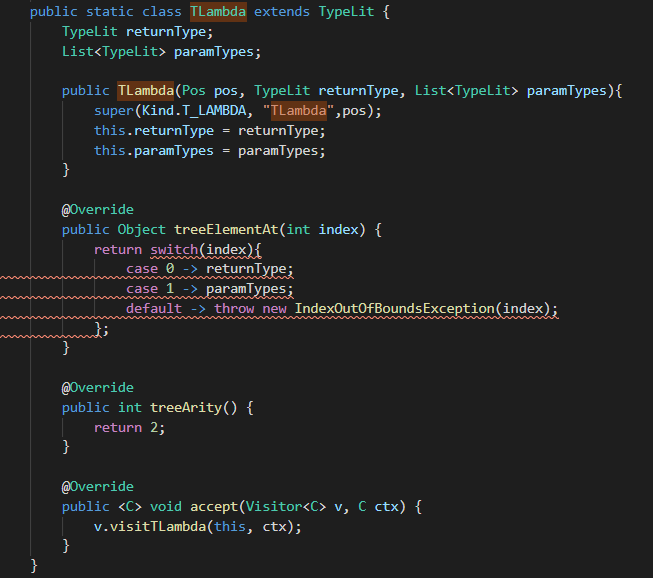
type = type ( type… )的完成需要用到Lambda表达式，需要新建一个TLambda类。所有新建的类需要现在Tree.java中进行注册。同样是在Tree.Kind中进行注册。



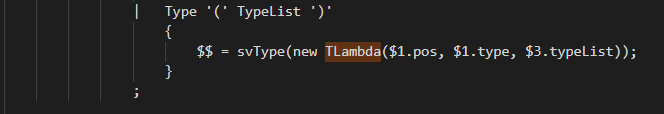
然后需要在Visitor.java中添加支持：



接着添加类的定义：



最后添加表达式：



至此完成了第一个小特性。

1. 困难与心得：

（1）与同学沟通后才意识到新类的建立需要进行注册和支持操作。

*2°Lambda表达式*

1. 任务要求与指导：

需要实现block lambda和expression lambda两种类型的Lambda表达式，需要用到’=>’操作符。提示中已经指出该操作符应该优先度最低，因而在添加关键字的时候需要在前方加上%right并且置于所有操作符之前。

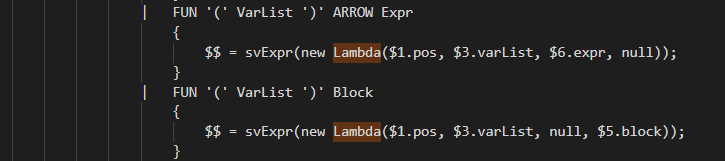


1. 实验步骤

实现Lambda类（省去了注册和支持的代码）：



然后添加表达式：



1. 心得与困难：

（1）更加彻底的认识到添加新功能的常规操作：添加关键字-注册新类并支持-实现类定义-添加表达式

*3°函数调用：*

1. 任务要求与指导：

由语法规范可知需要更改Call类的定义。

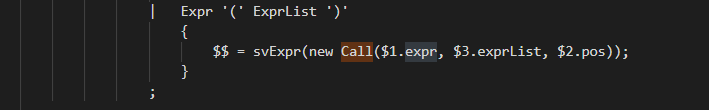


1. 实验步骤：

修改Call定义，去除不需要的参数。



修改表达式。



1. 困难与心得：

（1）与同学商议之后决定删除不必要的参数，并在此基础上直接重写了整个Call类。在之后的修改表达式的过程中，又出现了需要更改全局所有调用了Call的构造函数的表达式的问题。

**任务4：习题**

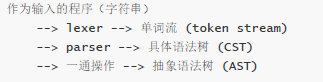
1. AST 结点间是有继承关系的。若结点 A 继承了 B ，那么语法上会不会 A 和 B 有什么关系？限用 100 字符内一句话说明。

答：A的构造函数需要调用B的构造函数，继承关系表示A是特化后的B，A是B的一部分。

1. 原有框架是如何解决空悬 else (dangling-else) 问题的？限用 100 字符内说明。

答：由ElseClause的表达式可以看出来，它只会对else+stmt和无else的情况进行推导，有效避免了只有else而无实现的问题。

3. PA1-A 在概念上，如下图所示：



输入程序 lex 完得到一个终结符序列，然后构建出具体语法树，最后从具体语法树构建抽象语法树。这个概念模型与框架的实现有什么区别？我们的具体语法树在哪里？限用 120 字符内说明。

答：区别在于本框架中CST和AST同时构造，其中编译时产生错误时显示的yysv数组即为CST，而构造该数组需要产生AST上的节点，二者互相依赖最终同时产生。

**任务5：总结**

完成PA的过程让我意识到我对词法分析和文法分析的理解还不太透彻，在完成任务的过程中和同学不断讨论，加深了我对这两个过程的理解。jacc文件中表达式的构造让我觉得非常有意思，也是我第一次在代码中看到自动机形式的表达式，让我眼前一亮。实现过程中碰到的最大困难就是IDE自动补全时容易出现大小写错乱和少字被忽略的情况，导致gradle build的过程不断报错，不过在一次次解决的过程中也让我对整个结构的理解更为充分，总的来说感觉收益颇丰。