编译原理PA1-A

实验过程

abstract关键字

从词法分析的角度，这是属于新关键字，需要修改jflex文件将其加入。注意到返回值需要用`Tokens.java`中的常量，声明名为ABSTRACT的新int常量。

为了这一关键字在语法分析中使用，yacc文件的`%token`域也要加入新的`ABSTRACT`关键字。为了建立yacc文件和jflex文件中对应符号的关系，需要在`JaccParser.java`文件中添加一个`Token.ABSTRACT`到`JaccToken.ABSTRACT`的映射。

修改文法。新的关键字可以用在`class`上，也可以用在`method`上，向`ClassDef`/`MethodDef`添加一条带有ABSTRACT关键字的文法规则，对应的代码需要返回一个被标记为抽象的类/方法。因为原来的框架没有考虑abstract，需要自行扩展`ClassDef`与`MethodDef`类：添加新属性，修改构造函数，修改`treeElementAt`等方法。这过程中甚至因为修改了构造函数的定义，导致部分原代码无法继续工作，我对其进行了修补（其实我本可以利用函数重载避免这一困扰）。

var关键字

这一特性与上述abstract在词法上大同小异，不再赘述。

在文法上，这一关键字只能用于赋值语句，因此只需要扩展`SimpleStmt`即可。问题在于，需要向传递一个类型为空的`LocalVarDef`，然而在旧有代码中这样做是未经定义的。根据输出要求，我将`treeElementAt`中的返回值改为了`Optional.ofNullable(typeLit)`，构造时直接向typeLit属性传入null。类似的一处：`MethodDef`中抽象函数的body也为空。这无疑是对源代码改动最小的方案，但是null的存在埋下了程序崩溃的隐患。不过，仍然通过了给出的PA1测试点。

Lambda表达式

词法上，这一特性要求增加`=>`与`fun`两个关键字，不过修改过程仍然与之前类似。

语法上涉及三处修改：新类型的增加，Lambda函数体定义，函数调用格式修改。

- 新类型的增加需要引入新的文法非终结符：`TypeList`；需要引入新的`TypeLit`的派生类`TLambda`；需要SemValue支持储存一个类型列表。
- Lambda函数体分为两类：一类是带表达式的，一类是带函数体的。除了文法上要分情况讨论，还需要对于两种情况分别构造Expr的子类。在我的实现中，`LambdaExpr`与`LambdaBlock`均继承自`Lambda`再继承自`Expr`以实现功能。
- 由于函数成为了一等公民，应该支持把表达式作为函数名进行调用的文法，这样以前的函数调用文法就需要做出变动，不再是总是需要Receiver和Id，而是只需要一个Expr就可以接括号了。同理，现在一对(Receiver, Id)就可以成为表达式。二者分别用`Call`类和`VarSel`描述，前者为了兼容以前的代码需要进行修改，构造函数改为接收Expr。

问题思考

Q1

> 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？

语法上未必有关系。继承只能描述节点的种类关系，比如一个`Call`对象继承于`Expr`表示它是一个表达式。节点用指针的形式描述AST，只有当`A`和`B`有直接或间接的指针路径相连才会产生语法关系，`A`和`B`具体是`Expr`的哪个派生类是不重要的。

Q2

> 原有框架是如何解决空悬 else (dangling-else) 问题的？

将else声明为最高优先级的运算符（不可结合），这样`ElseClause`会优先得到解析，从而和`if`进行匹配。这样实现下，空悬else总是和最内层的if进行匹配。

Q3

> PA1-A 在概念上，如下图所示：
>
> ```
> 作为输入的程序（字符串）
> --> lexer --> 单词流 (token stream)
> --> parser --> 具体语法树 (CST)
> --> 一通操作 --> 抽象语法树 (AST)
> ```
>
> 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？

区别：

1. 框架不是先生成单词流，而是每次通过lexer从字符串中取出一个Token。
2. Parser不经显式生成CST直接构造AST。

具体语法树是在jacc进行语法分析时顺便“构造”的，例如有文法`A:B C D`，那么解析出的符号`A`则拥有指向解析出来`B`、`C`、`D`的指针，形成一棵CST。不过，我们的代码中构造和返回的都是AST对象与节点。

总结

本次实验我熟悉了词法、语法框架，成功地加入了新的语法特性。因为有大量的代码可以参照仿写，因此似乎并没有什么太大的挑战。我想最大的问题与挑战应该在后面，如何读懂框架，之后如何精妙地扩写代码让它正确优雅地工作。