CS179E Phase 1: Type-Checking

by Brandon Yi

Requirements and Specifications

Phase 1 of this project is required to type check a MiniJava program. Given a MiniJava program, the type-checker must print out `Program type checked successfully` if there is no type error. However, if there is a type error, the program must print out `Type error`. The program takes the MiniJava program in from standard input.

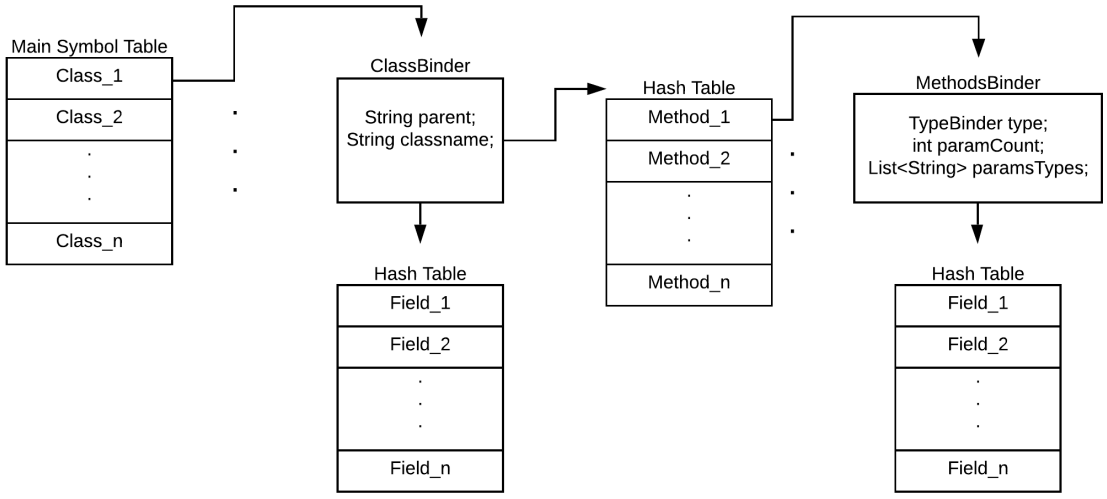
Phase 1 uses JTB and JavaCC to create a syntax tree and visitor classes. The generated visitor classes allow easy traversal of the aforementioned syntax tree. My program uses modified versions of the generated `DepthFirstVisitor` and `GJDepthFirst` visitors.

Design

Overview

My implementation does two passes over the source code: the first pass constructs the symbol table and the second pass uses the symbol table to type check the expressions and statements. The details of these structures will be described below.

Symbol Table Design



The symbol table is multi-layered. Each level of the symbol table uses a hash table to store `Binder`s which are the data structures that contain information about the identifier stored in the hash table. Each `Binder` class contains information unique to the identifier's type and a `Symbol` which stores the name of the identifier. The `Symbol` class is used to determine where in the hash table the `Binder` is stored.

The first level stores an extended class of `Binder` called `ClassBinder`; The `ClassBinder` contains the name of parent of the identifiers's class (i.e. the classname after `extends`), the identifier's classname and two hash tables. The first hash table is used to store the fields of the class. Each of the identifiers stored in the first hash table have a `Binder` for their respective type (e.g. `IntBinder`, `BoolBinder`, `ArrayBinder` or `ClassBinder`). The second hash table is used to store `MethodBinder`s.

The hash table of `MethodBinder`s contains all of the information about each method in the class's definition. The `MethodBinder` class contains a `TypeBinder` which indicates the return type of the current method. It also stores information about the amount and types of input parameters the method accepts. Finally, the `MethodBinder` stores `Binder`s for each field in the method where each `Binder` is a `Binder` of the identifiers respective type.

First Pass: SymbolTableConstructor

The class `SymbolTableConstructor` constructs the symbol table. It keeps track of the scope of the program by storing two `Binder`s: `currClass` and `currMethod`. This allows the `SymbolTableConstructor` to correctly identify which class's and/or method's symbol table it should search for an identifier in. For example, this is how a variable declaration is handled:

```
public void visit(VarDeclaration n) {
    n.f0.accept(this);
    n.f1.accept(this);
    n.f2.accept(this);

    if (currMethod == null) {
      if (currClass.myItems.alreadyExists(Symbol.symbol(idName(n.f1))))
        RegTypeError();

      if (n.f0.f0.choice instanceof IntegerType)
        currClass.myItems.put(Symbol.symbol(idName(n.f1)), new IntBinder());
      if (n.f0.f0.choice instanceof BooleanType)
        currClass.myItems.put(Symbol.symbol(idName(n.f1)), new BoolBinder());
      if (n.f0.f0.choice instanceof ArrayType)
        currClass.myItems.put(Symbol.symbol(idName(n.f1)), new ArrayBinder());
      if (n.f0.f0.choice instanceof Identifier)
        currClass.myItems.put(Symbol.symbol(idName(n.f1)), new
ClassBinder(((Identifier) n.f0.f0.choice).f0.toString()));
    } else {
```

```
      if (currMethod.myItems.alreadyExists(Symbol.symbol(idName(n.f1))))
        RegTypeError();

      if (n.f0.f0.choice instanceof IntegerType)
        currMethod.myItems.put(Symbol.symbol(idName(n.f1)), new IntBinder());
      if (n.f0.f0.choice instanceof BooleanType)
        currMethod.myItems.put(Symbol.symbol(idName(n.f1)), new BoolBinder());
      if (n.f0.f0.choice instanceof ArrayType)
        currMethod.myItems.put(Symbol.symbol(idName(n.f1)), new
ArrayBinder());
      if (n.f0.f0.choice instanceof Identifier)
        currMethod.myItems.put(Symbol.symbol(idName(n.f1)), new
ClassBinder(((Identifier) n.f0.f0.choice).f0.toString()));
    }
  }
```

Second Pass: Check Visitor

Testing and Verification