Language and Traslator - Code Generator Report

Francesca Daniele
Carmelo Gugliotta
Marco Leto

A.Y. 2023/24

Our Code Generator implementation respect and satisfy all the "project statement" requirments. Our approch to build the ByteCode of the code consist in the AST analysis produced from the Parser and checked by the Semantic Analysis.

1 ScopesTable

First of all we implemented the class `ScopesTable.java`, this data structur is used to manage the scopes in the compiler. Each table is formed by a `Map<String, ScopeEntry>`, where `ScopeEntry` memorize information about a variable (String) used in the scope where the table has been defined. Every entry has the following structure:

```
class ScopeEntry {
    Integer index; //position of the variable into the stack
    Type type;     //Type of the variable
    ScopeEntry(int index, Type type){
        this.index=index;
        this.type=type;
    }
}
```

The constructor of the class create a link the new table to a `previousScope`, thats usefull to interrate multiple scope to use more external variable in a internal scope.

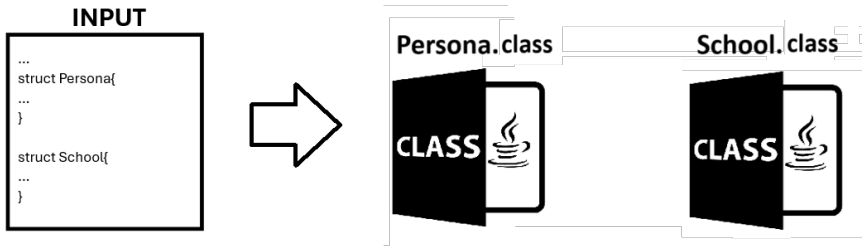
2 Visitor Code Generator

The class `CodeGenerationVisito.java` utilize an instance of the class `ClassWriter`, part of the ASM Java's library that allows to create the .class file that will serve to generate the final result of the intial input. In the class different visit methods are defined. Each one take a node of the AST created by the Parser:

```
        while (it.hasNext()) {
            ASTNode node= it.next();
            node.accept(codeGenerationVisitor, table, mainMethodWriter);
        }
```

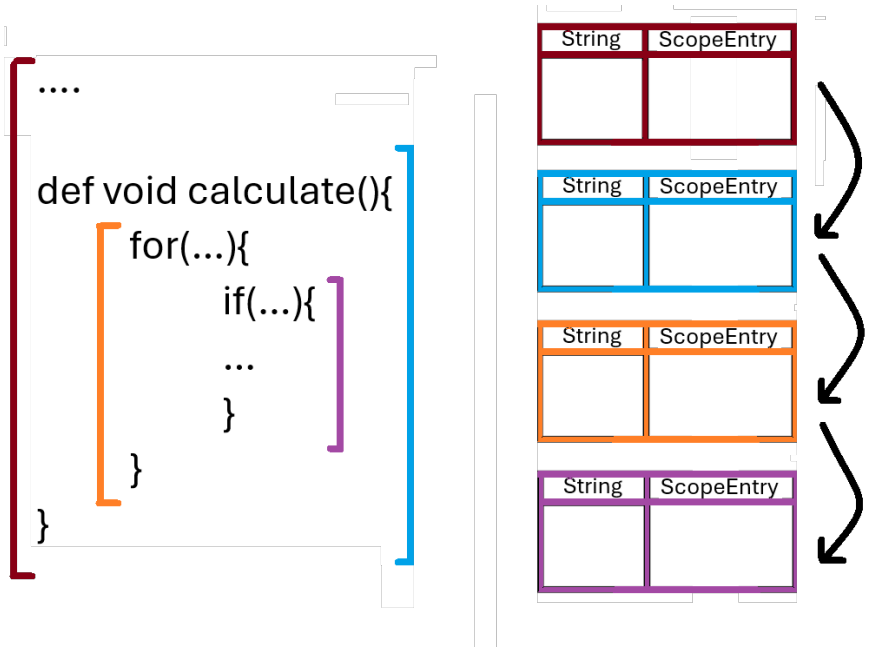
Every visit method need to be discussed separately:

- Constant $->$ Add the constant to the `curr_scope` with (-1) as index, this because constant are globally accessible. On the extracted right part called the `.accept` method of the `EvaluateVisitor` that has the goal to calculate the value of the expression.
- Struct $->$ When a Struct is visited, the `curr_scope` is not used, but for each one, a new Java class is created, this contains all the parameters, and the constructor to instantiate new objects of the class.



- Global Variable $->$ are treated in the same way of the constant, added with (-1) index and evaluated by the other visitor implementation.
- Struct $->$ the struct are the complex node to manage, as seen in the previous parts. For each procedure, a new `.class` file is created and a new `ScopeTable` is linked to the previous.

  Is important to remember that structs contains a `Block`, that admit the definition of the loops and the definition of the variable. For this reason in this parts a chain reaction of scope tables connected to each other occurs.



Every visit method, contains some instruction of the ClassWriter that consent to generate the bytecode of the program, that is written on an output file at the end of the iteration on the AST.

3 Evaluate Visitor

The class `EvaluateVisitor.java` is used to evaluate the expression during compilation, the main methods are:

- `visit(BinaryExpression binaryExpression,...)` that consente to evaluate a BinaryExpression. It is the base unit of the expressions, that could be formed by other binaryExpression. For this reason is evaluated with a series of a recursivly calls:

```
public void visit(BinaryExpression, ScopesTable, MethodVisitor ){

    binaryExpression.getLeft().accept(this,curr_scope,mw);
    binaryExpression.getRight().accept(this,curr_scope,mw);
    //Do the operation (IADD,FADD,ISUB,FSUB...)
    ...
}
```

- `visit(ArrayAccess arrayAccess,...)` that evaluate the array access, and implements a particular approch, with a boolean condition, to don't get in error for calls like:

```
Person p= Person(1,"string",{1,2,3})
int a = p.array[z];
```

In this situation, trying to access to an array value of an object instantiation, the operation refer to the scope table of the Object, but z is in contained in program scope tables.

- `visit(StructAccess structAccess,...)`, at the first call of the method, the left part that contain the instantiation variable of struct type is loaded, while the right part is a field that could be a baseType, a struct type (having a recursvly call on the method) or an array with the conseguently call of the visit(arrayAccess).
- `visit(FunctionCall functionCall,...)`

4 Main

During this phase has been necessary redefine the idea about what part of the program was the executable part. For this reason, in the class that is created during the code generator phase, a `public void main(String[] args)` method as been defined. It contain:

- a call to a `main` method, if it is has been defined in the input, in this way the main is well evaluated
- is empty if there is not any `main` method, obtaining an empty output

5 Extra Features

- Is possible to define Global Variable and procedure in flexible order
- Is possible instanciate array in two different ways:
  - `int[] a= int[3]`, that create an array of size 3
  - `int[] a= {1,2,3}` that create an array with 1,2,3 inside
- Is possible to define Object arrays, two different ways:
  - `Person[] p={Person(...),Person(...),...}`
  - `Person[] p={p1,p2,p2}`, with variable references
- Is possible to do operation between Integers and Float. The operation `3.0/2` return a float.
- Is possible to have Struct with other struct, an array, or an array of struct as a field
- Is possible to have complex expression `peoples[getOne()].favoritePlaces[getIndex(x)-1].x`
  - peoples is an array of Person
  - getOne() is a function that return an integer
  - favoritePlaces is a field array of the struct Person that store Point
  - x is a field of point
  - the x into getIndex(x) is a variable
- Is possible to instanciate all loops configuration, with Expression in the conditions

- A function `len` has been defined, and return the length of arrays of BaseType and StructType
- Is possible to modify a value of an array, with a specific index: `person[3]=Person( )`
- The Struct constructor admint all the type of expression `Person(3+4,{1,2,3,4}, 4==3 && false)`
- The case `float[] array={1.2,3,4,5,6}` is managed, converting all integer in flaoat.

6 Things not done

- In the Code Generation we didn't manage the possibility of change value of fields of an instanciate Struct. In all the other parts we implemented it, but not in this part becouse we notice this feature late.