SoftEng 701
Advanced Software Engineering Development Methods
Assignment #2 (20% of final grade)
Due: 11:59pm, 15 April 2016

Kelly Blincoe, k.blincoe@auckland.ac.nz

Summary

You have learnt the tools needed to parse and compile a real programming language, such as Java. In this project, you will implement *semantic analysis* and incorporate a new programming feature into Java. This will require you to build a source-to-source compiler. You are required to select one of the following new features and incorporate it into Java.

- ***yield Keyword.*** Allow a yield keyword in a method definition to allow code to be injected into an existing method.
- ***Map Literals.*** Allow a new Java expression for initialising maps that requires fewer statements and infers types.
- ***Tuple Return Types.*** Allow a Java method to return up to two values which can be of different types.

You are also required to submit a report that justifies why your selected new feature should be included in the Java language. This justification must be supported by relevant literature. The report must also discuss the design and implementation of your symbol table.

Details

You will be provided with java_1_5.jj which specifies the Java grammar for the entire Java language (v1.5). You must modify this .jj file so code which includes your new feature does not throw any ParseExceptions. You will take a *.javax file as input and output a *.java file which can be compiled using a standard Java compiler. Therefore, you will be creating a source-to-source compiler. You must incorporate semantic analysis for your new feature as well as general Java code. You should cover simple semantic cases such as:

- Scope
- Types
- Forward references
- Member access

However, your compiler does not need to be complete, you can do some hard coding. You should hard code the required primitive types, classes and libraries in your global scope since your compiler will not have access to these. For example, if your input calls the System.out.println method, you can hard code the System class, the out variable and the println method. You should cover commonly used classes (e.g. String, Integer, etc.) and any other classes needed for your feature.

You will be given two sample input files to help you test your semantic analysis, but your submission will be marked against many other input files - you must therefore be confident in your semantic analysis when you submit it! You are free to discuss the assignment with other students and to exchange sample input files for testing your semantic analysis. However, each student must implement the semantic analysis and new feature individually. All submitted work will be checked against software plagiarism tools, with heavy penalties in place.

Feature Choices

yield keyword

This Java extension will implement a yield keyword which will allow you to inject code into an existing method, something that is currently unsupported in Java. With the yield keyword, you don't have to write a new method if you want a method to work in a slightly different way, instead you can reuse the existing method, but give it a different yield block. For example, the following code specifies a yeild block when calling the sampleMethod(). Within the sampleMethod(), the yield keyword designates where the code contained within the yeild block should be injected.

```
public static void main(String[] args){
    sampleMethod(){
        //The yield block
        for(int i = 0; i < 20; i++) {
            System.out.println(i);
        }
    };
}

public static void sampleMethod() {
    System.out.println("Execution started");
    yield;
    System.out.println("Execution ended");
}
```

Map Literals.

To initialise a Map, current Java syntax requires two steps. First, you must initialise a Map instance with type arguments. Next, you invoke methods on that instance to add entries. For example:

```
Map<String, String> states = new HashMap<>();
states.put("AL", "Alabama");
states.put("AK", "Alaska");
states.put("AZ", "Arizona");
...
states.put("WY", "Wyoming");
```

This java extension will allow a new initialiser expression which handles both instantiation and declaration of map entries. For example:

```
Map states = new HashMap {
    "AL" : "Alabama",
    "AK" : "Alaska",
    "AZ" : "Arizona",
    ...
    "WY" : "Wyoming"
};
```

Note that type arguments are not specified in the new expression, and must be inferred from the types of the map entries in the literal expression.

Tuple Return Types

Java currently only allows one value to be returned from a method. One way to return multiple values from a Java method is to utilize some type of Collection object which holds the values you wish to return. For example:

```
public static void main(String[] args) {
    ArrayList<Integer> squareCube;
    squareCube = calcSquareCube(new Integer(3));
    System.out.println("Square is: " + squareCube.get(0));
    System.out.println("Cube is: " + squareCube.get(1));
}

public static ArrayList<Integer> calcSquareCube(Integer value) {
    ArrayList<Integer> squareCube = new ArrayList<Integer>();
    Integer square = value * value;
    Integer cube = square * value;
    squareCube.add(square);
    squareCube.add(cube);
    return squareCube;
}
```

Problems arise when you want to return objects of different types or primitive types which can not be directly added to Collections. This java extension will allow a method to return up to two values. For example:

```
public static void main(String[] args) {
    int square, cube;
    square, cube = calcSquareCube(3);
    System.out.println("Square is: " + square);
    System.out.println("Cube is: " + cube);
}

public static int, int calcSquareCube(int value) {
    int square = value * value;
    int cube = square * value;
    return square, cube;
}
```

Note that the return values can be of different types, and primitive types should be allowed.

How your code will be tested

To ensure that your code can be marked easily, you have been provided with a number of helper files:

- `tests/`: Inside this folder, there are two `*.javax` files. These files are correct with respect to the scanner and parser, but have various semantic problems. After implementing semantic analysis, compiling these files should throw appropriate errors.
- `japa/`: Inside this package folder is the standard JavaCC 1.5 parser and AST. To allow for your language extension, you will be modifying the `java_1_5.jj` file (so it parses). You will then be adding more visitors and AST nodes as necessary.
- `se701.StudentSample.javax`: This is a "Java extension" file which includes code that utilzes the new feature you have implemented. Three versions of this file have been provided. You should select the file that corresponds to the feature you have selected and rename it to StudentSample.javax. Using this file as input, your source-to-source compiler should generate the equivalent `StudentSample.java` file, which can then be compiled and run using a standard Java compiler/interpreter. Notice this file is part of the `se701` package, and it is contained within the `src` folder since the generated Java file should be compilable (without any issues) and should be able to be executed. The marker will use your compiler and the provided StudentSample.javax file to create StudentSample.java and then compile and run the generated `StudentSample.java` file. You should not modify the StudentSample.javax file. The marker will also run your compiler on derivitatives of StudentSample.javax to test your semantic analysis for your new feature.

- `se701.A2MainRunner.java`: The markers will use a file exactly like this to run your compiler against the files inside the `tests/` folder. For your purposes, the only thing you may safely change is the for-loop boundary specifying what files to test against. Do not modify anything else, otherwise you are setting yourself up for problems if it deviates from the original file.
- `se701.A2Compiler.java`: This is the interface for the markers to access your compiler. You should complete the `compile(File)` method, however do not modify its signature! Inside this `compile` method, you create a new instance of your parser, new instances of your respective visitors, perform the visit, then save the result into a `*.java` file in the same directory as the input file. You are provided with the helper methods `getAsJavaFile()` and `writeToFile()`.
- `se701.A2SemanticsException.java`: This is the runtime exception that your visitor(s) must throw whenever they have detected some semantics problem. The message you throw should be descriptive enough to accurately say what the error is, what line number it occurred in, etc. Notice that the `A2MainRunner.java` file catches this exception. You should not need to change this file.

Make sure you can get your code compiled and running using the *original* `A2MainRunner.java` file given. The marker will regenerate your parser from the `java_1_5.jj` file, so make sure there are no errors in your JavaCC file. If the marker cannot get your code running, you will most likely get no marks!

Report

You must also write a report detailing why your feature is important. How does it help programmers write better code? For example, motivation in terms of the complexity of having to manually do it the old-fashioned way. *More credit will be given for having demonstrated a sound research investigation to justify your argument that it promotes "good software" aspects*; **you must consult some literature regarding this.** Ultimately, your report will be making a claim that you have developed an innovative programmer construct that addresses a problematic issue that programmers face. You need to convince the reader that the problem is something significant, important and useful for programmers. If there are any restrictions on how your feature can be used, you must state them.

Your report should also give an overview of what you did to implement the feature (what nodes in your AST did you have to work with? etc.) and what semantic analysis is included specific to your new feature. You must describe the design of your symbol table and visitors implemented. **The report should be a maximum of 3 pages long in total**, using double-column format and **in PDF format.** This means, do not waste space explaining the background of parsing to beginners, but get straight into the design aspects of your feature, the motivation and actual code transformation.

Marking scheme

This assignment is worth 20% of your final grade. Marks will be allocated as follows:

- Report 40%: concise, well formatted and professional, justifies feature, motivation based on sound research, discusses important design/implementation aspects, both in regards to your feature and symbol table; maximum 3 pages. **Must be in PDF format.** The report being marked will be that submitted in the Canvas Turnitin Assignment.
- Implementation 60%: clean code, easy to read, etc. A portion of this will also include the depth of your semantic analysis, how well your compiler picks up incorrect semantic details on sample Java code. Examples of semantic analysis will include referring to symbols out of scope, using variables and methods not being declared, incompatible values being assigned. For all these situations, types and methods must be declared in the same file (i.e. you don't need to worry about Java non-primitive types, but should be able to understand primitive types).

Submission

You will submit in two locations.

On the Assignment Drop Box, by visiting `https://adb.auckland.ac.nz`, submit the following, in a single ZIP archive file:

- A **signed and dated declaration** stating that you worked on the assignment independently, and that it is your own work. Include your name, ID number, the date, the course and assignment number. All submitted reports will be checked using `www.turnitin.com`, and all code will be checked against other submissions (including those from previous years).
- Your modified `java_1_5.jj`
- Your modified `se701.A2Compiler.java`
- The *entire* Java source needed to run your project, regardless of whether modifications were made to the original files or not.
- Your report.

  IMPORTANT: If you do a resubmission on Assignment Drop Box, make sure you submit EVERYTHING again.

In addition, you must also submit your report using the Assignment 2 Turnitin Assignment in Canvas. The report submitted in Canvas will be read and marked. **If your report is scanned, it will be rejected during the submission** (i.e. don't print off your report, then scan it in again. It needs to be a "searchable PDF document" where you can search for words). More information can be found here:

`https://www.auckland.ac.nz/en/about/learning-and-teaching/policies-guidelines-and-procedures/academic-integrity-info-for-students/about-academic-integrity/turnitin-faqs.html`

Turnitin will compare your submitted report against a continuously increasing collection of:

- Over 120 million previously submitted students assignments
- Over 12 billion web sites (including those archived)
- Over 25 million journal articles
- Books from leading publishers

Late submissions

Late submissions incur the following penalties. The penalty will only apply for the particular late component (e.g. if only the report is late, then you are penalised only for that component – and not on the code).

- 15% penalty for 1 to 24 hours late
- 30% penalty for 24 to 48 hours late
- 100% penalty for over 48 hours late (dropbox automatically closes)