Homework 1 (Programming): Simple Static Analysis

17-355/17-665/17-819O: Program Analysis
Claire Le Goues and Jonathan Aldrich
clegoues@cs.cmu.edu, aldrich@cs.cmu.edu

Due: Thursday, January 25, 2018 (11:59 PM) 100 points total

Assignment Objectives:

- Set up the Soot analysis infrastructure in your environment and successfully compile and run an analysis
- Understand intermediate representations in the Soot tool
- Implement simple analyses based on visiting different parts of the intermediate representation

Handin Instructions Place all your homework files in your private GitHub repository in a folder called `hw1`. The directory should contain the same content as the download, but with your analyses implemented. Running `ant` in the `hw1` directory should automatically compile and test the analyses, including the new analysis that you write from scratch. After the deadline, we will clone your repository and run the `ant` script. You can test that you did everything correctly by running the following in a temporary directory:

```
git clone https://github.com/CMU-program-analysis/<YOUR-ANDREW-ID >
cd <YOUR-ANDREW-ID>/hw1
ant
```

1 Setup and Tool Information

Clone your GitHub repository and create a new folder called `hw1`. Download hw1.zip from the course website and unzip it in the `hw1` directory.

Ensure you have installed the Java Development Kit version 7 or 8. Also ensure that you have ant installed and running. Note that Andrew Unix machines, `unix.andrew.cmu.edu`, have both the JDK and `ant` already installed for you.

To make sure everything is running, run `ant build` in the `hw1` directory. The project should compile successfully and report:
`BUILD SUCCESSFUL.`

Next, type `ant`, which will both build the project and run the tests. This time, after compilation succeeds, you will get a `BUILD FAILED` message indicating that the tests failed. That is expected, since in doing the assignment you will implement the analyses so that the tests pass.

After compiling with `ant`, you can also run Soot manually. to produce Jimple output for a class such as `edu.cmu.se355.hw1.Shifty`, go to the `hw1` directory and run:

```
java -cp lib/soot-trunk.jar soot.Main -cp "build:lib/rt.jar" -f J
    edu.cmu.se355.hw1.Shifty
```

To run the example PrintAnalysis on the same class, use:

```
java -cp "build:lib/soot-trunk.jar"
    edu.cmu.se355.hw1.PrintAnalysisMain -cp
"build:lib/rt.jar" -p jap.printanalysis on
    edu.cmu.se355.hw1.Shifty
```

The main analysis files for bad shift analysis and unread field analysis also specify unit tests, using the JUnit framework. The unit tests are run by the ant build script (`build.xml`), but you can also run them manually. The command for the bad shift analysis unit test is:

```
java -cp "build:lib/soot-trunk.jar:lib/junit-4.11.jar:
lib/hamcrest-core-1.3.jar" org.junit.runner.JUnitCore
edu.cmu.se355.hw1.ShiftAnalysisMain
```

The above command lines have been tested on Linux, but they will be different if you are using Windows: you will need to replace / with and replace : with ;

Useful documentation for Soot include the Soot Survivor's Guide as well as the API Javadocs.

2 Analysis Implementation

In this assignment, you will implement three simple static analyses. The first two are specified in the assignment: bad shift analysis and unread

field analysis. We have provided starter code and test cases for these two analyses, along with some hints to get you started. You will design and implement a third analysis from scratch.

We recommend you use the Soot analysis tool, as described in the first recitation section. However, you may choose to use some other program analysis engine and even some other language for implementing and testing your analysis; if you want to do so, contact the instructor to discuss whether any aspects of the assignment need to be adapted.

Question 1, Shift Analysis, *(25 points).*

> Complete the code in `ShiftAnalysis.java` in order to identify and report warnings about bit shift operations that shift by a constant that is greater than 31 or less than 0. You can report an error using the `Utils.reportWarning` *(element,message)*, where *message* is one of the enumerated error message constants in the class `ErrorMessage`, and *element* is the Soot representation of the thing causing the error (e.g., a `SootField` or a `Stmt`).

Question 2, Unread Field Analysis, *(25 points).*

> Complete the code in `UnreadAnalysis.java` in order to identify and report fields that are declared but never read. You should report fields that are written but not read from.
>
> We have provided a test case, which you can run with the `ant` command; passing the test case is an indication that you are on the right path, though earning credit for the assignment requires implementing your own analysis in a general way so that it will also work correctly with other test programs.
>
> Note: in some cases the `javac` compiler may replace reads of a variable that acts like constant with the constant itself. It's OK if your analysis warns about these situations. This does not occur in the test case we provide, though.

Question 3, Describe Your Own Analysis, *(10 points).*

Secify another simple analysis, similar to those above, that finds some kind of bug. You may choose what kind of bug on which to focus; the list of bugs covered by FindBugs may give you some ideas:

http://findbugs.sourceforge.net/bugDescriptions.html

It's OK if the bug you discuss is relatively simple; we have not yet discussed program analysis techniques that are capable of finding deep bugs.

Include a `README.md` file in your `hw1` folder describing precisely under what conditions your analysis should report a bug.

Question 4, Implement Your Own Analysis, *(40 points).*

Implement the analysis you specified above. Test it on a sample program that has at least 3 different instances of the bug, as well as 2 instances of code that is similar to code that would trigger the bug, but which is correct. You will need to add a new enumerated constant to the `ErrorMessage` class. Write a JUnit test similar to the test cases provided for ShiftAnalysis and UnreadAnalysis that specifies what warnings should be generated for the sample program you provide. Extend `build.xml` so that your test case is automatically run when `ant` is invoked; this should be as simple as adding another `<fileset>` tag under `<batchtest>`.

Note: We reserve the right to run your analysis code on examples other than the test code we give you, looking both for accuracy of the analysis and its robustness (i.e., it should not throw unexpected exceptions when run on a larger codebase). Of course, any problems that are due to external libraries such as Soot are not your problem, as long as you are not misusing the library.