Homework 2

Posted January 29, Due February 5

50 points

1. Setting up: Eclipse, Git, Soot, and Starter Code

I assume you are familiar with Eclipse, Git, and Submitty.
First, clone your `hw02` Git repository in some local directory. I'll call this `repo_dir`:

```
git clone https://submitty.cs.rpi.edu/git/s24/csci4450/hw02/YourRCSID
```

You might need an authentication token to clone:
https://submitty.org/student/account/authentication_tokens.
Next, create your Eclipse project using the following steps:

(1) Go to File → New → Java Project.
(2) In Project Name, type "class_analysis" (or if you want to name your project differently, go ahead), unselect "Use Default Location" then in Location browse to the target directory where your local Git repo lives (this will be `repo_dir/YourRCSID`), then click **Finish**. At this point in **Package Explorer** view you should see a project named "class_analysis" with subdirectory `src`. In `src` you should see packages `analysis` and `analysis.RTA`. These packages should remain under the build path. However, subdirectory `programs` should not be under the build path. Select all `src/programs/pN` → right click → Build Path → Remove/Exclude. Packages `analysis.*` contain the program analysis code and `programs` contains the Java programs that we will analyze.
(3) At this point, packages `analysis` and `analysis.RTA` show errors because they are missing Soot libraries. Download the Soot jar:
`https://www.cs.rpi.edu/~milanova/csci4450/soot-develop.jar`. Then add it to the build path for your class analysis project: right click on project `class-analysis` → Build Path → Add External Archives... then select `soot-develop.jar`.
(4) Lastly, include JUnit to get rid of the JUnit errors: Right-click on project class-analysis, then Build Path → Add Libraries... → JUnit (JUnit 4) → Finish. At this point there should be no compilation errors.
(5) Soot requires a **Java 7** `rt.jar` file. Download the `rt.jar`:
`https://www.cs.rpi.edu/~milanova/csci4450/rt.jar`, then set the `RT_HOME` global variable in `RTATests.java` to point to your local `rt.jar`.
(6) We are now ready to run the starter code. If you are running with JRE System Library 1.7 or 1.8 select `RTATests.java` → Run As → JUnit Test and it should work. Unfortunately, Soot appears to be exposing a backward compatibility error and if you are running Eclipse with a recent version of Java, Soot throws a `NullPointerException` as it attempts to access a system property that no longer exists.

    You need to change your JRE to 1.8 (1.7 also works). To do so, download and install a Java 8 archive from the Oracle website (e.g., I downloaded `jdk-8u202-macosx-x64.dmg`) or from the OpenJDK website. Do Build Path → Add Libraries... → JRE System Library → Next. Select Alternate JREs then Installed JREs... then **Search...** and Eclipse should find the newly installed JRE on your machine. Select the 1.8 JRE and Apply. Don't forget to change the Compiler as well: Build Path → Java Complier.

IMPORTANT: Make sure that your directory structure is correct or compilation on Submitty will fail. You need project `class-analysis` with subdirectory `src`. `src` has subdirectories `analysis` and `programs`. In Project Explorer you should see packages `analysis` and `analysis.RTA` under the build path, and directory `programs` with toy Java programs. Directory `programs` shouldn't be under the build path.

All homework is autograded on Submitty. When you are ready to submit, go to the repository and push your changes. For example, in HW2 you will be modifying `RTAAnalysis.java` and `ChaUtils.java`. Push those files, then go to the Submitty gradable and click Grade My Repository. For those of you new to Submitty, don't forget to push your changes to the repo before Grade My Repository!

2. Overview of Class Analysis Framework

Starter code in package `analysis` builds a Java class analysis framework on top of Soot. Code in `Analysis.java` abstracts away Jimple into 8 kinds of statements relevant to class analysis:

(1) Assignment: x = y
(2) Field read: x = y.f
(3) Field write: x.f = y
(4) Array read: x = y[]
(5) Array write: x[] = y
(6) Object allocation: x = new A
(7) Direct call: either static invoke x = sm(args) or special invoke x = y.m(args)
(8) Virtual call: x = y.m(args)

Since we are interested in class analysis, i.e., flow of values of reference type, we ignore all statements and expressions on primitive types. All exposed variables (essentially all, there is one exception) are of reference type.

`Analysis.java` includes a worklist-like algorithm for solving constraints (constraints is just another name for transfer functions). Your task is to encode constraints as needed for a specific analysis then fire up the algorithm to compute the fixpoint solution. In the first assignment, HW2, you will be implementing Rapid Type Analysis (RTA), which requires the simplest constraints (i.e., transfer functions) and dataflow information (i.e., lattice). In HW3 you will implement XTA, a more precise and slightly more complex analysis. One can code a specific analysis by instantiating the designated hook methods. For example, for HW2 you will instantiate certain hook methods in `RTAAnalysis.java`. **We will elaborate on the framework, starter code, and analysis code in class. Come prepared with questions on Thursday!**

Folder `programs` contains several toy Java programs. Run the analysis on each of these and carefully examine the Jimples created by Soot. The autograder tests your analysis on these programs.

3. HW2

Your first task is to code Rapid Type Analysis (RTA), which we covered in class. Place your code in package `analysis.RTA`. Starter code for this package is already there in your repo. Study the code, as well as the rest of the framework. Places where you will be adding your code are marked as `TODO: YOUR CODE HERE`. (There are other TODOs scattered throughout the code; these are reminders for me to fix, eventually.)

Once analysis is done, print analysis results on the console. Your `analysis.RTA.showResults` should print all RTA-reachable methods, by full name in alphabetical order, followed by all instantiated classes, also in alphabetical order. For example, the expected output from one of the toy programs looks like this:

```
Reachable methods:
=== <A: int add(A)>
=== <A: void <init>()>
=== <A: void m()>
=== <A: void main(java.lang.String[])>
=== <A: void sm()>
=== <B: void <init>()>


Instantiated classes:
=== A
=== B
```

When you are done, push into your repository and click Submit in Submitty. I will be running your `AnalysisRTA` with a slightly different driver to compare your output with the expected output.