CSE 490E1 Assignment 3: Project Proposal
Optional Checker Framework

Justin Kotalik (jkotalik), Jasper Hugunin (jasperh), Han Zhang (gzezhang)

Movitation:

Let’s say you are a new java developer that has just created their first program in eclipse. You run your code and get a NullPointerException (NPE) when trying to access a field of a string. You are not sure how to fix this problem, so you go on StackOverflow and ask why you are getting the exception? A community member responds and says that you should be using Java’s new Optional Class to check instead. Now your NullPointerException is gone.

However, this is a misuse of the Optional Class. If a user says x.get() on an optional object x, instead of a NPE being thrown, we get a NoSuchElementException. This doesn’t fix the defect or improve the code, rather redirects the problem to another exception. On top of that, there are now a new set of problems that need to be handled introducing a new class. Michael Ernst has a whole article dedicated to showing the issues with Optional, shown in [1].

The problem we are trying to solve is to warn users of the potential of a NoSuchElementException being raised when using the .get() method. This, in tandem, will help promote correct use of the Optional class. This is a useful problem to solve because it allows for users to fix any known exceptions that could be thrown. To solve this problem, either a tool or applet could be created that would check the correctness of using Optional.

Multiple prevalent figures have noted the common issues with using the Optional class. These are primarily in dev and blog posts, and Michael Ernst’s post in the bibliography [1] provides a strong breakdown.

Approach:

We will directly be using the Checker Framework to create a tool to check uses of the Optional class. Specifically, we will create an Optional Checker which, like the Nullness Checker of the Checker Framework, uses a type system to provide guarantees that no NoSuchElementException will be thrown by calling .get() on an empty Optional. We will use the Checker Framework to handle interfacing with javac for parsing and warning output, as well as flow analysis etc. The general architecture of the Checker Framework can be seen in figure 1. On top of this, we will add appropriate unit and integration tests to validate our checker.

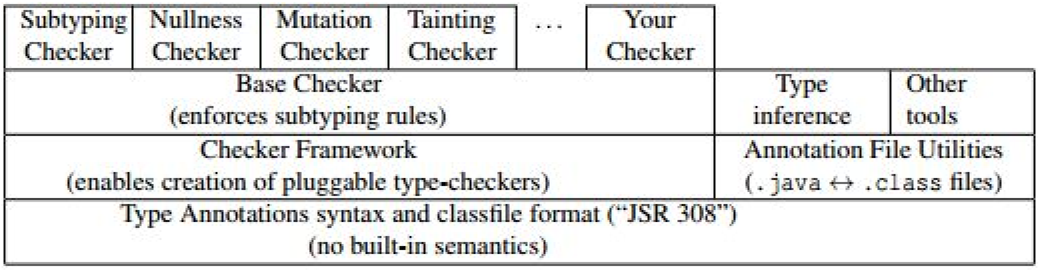
Subtyping Checker
Nullness Checker
Mutation Checker
Tainting Checker
. . .
Your Checker
Base Checker (enforces subtyping rules)
Type inference
Other tools
Checker Framework (enables creation of pluggable type-checkers)
Annotation File Utilities (.java ↔ .class files)
Type Annotations syntax and classfile format ("JSR 308") (no built-in semantics)

Figure 1: Checker Framework wireframe [2] section 29.1

Challenges and Risks:

Implementing an Optional Checker will require understanding the Nullness Checker (about 2300 lines of Java code) well enough to replicate most of its analysis and translate that to checking uses of Optional. The checker framework manual says that "a bad choice [to copy and modify] is the Nullness Checker, which is more sophisticated than anything you want to start out building" [2]. This indicates that understanding and modifying the Nullness Checker to check uses of Optional.get() will be difficult. We anticipate that understanding how the Nullness Checker works will be the most difficult part of this project.

Fortunately, our teacher Michael Ernst is one of the developers of the Checker Framework. By asking for his help when we are confused by the Nullness Checker, we can avoid being blocked and continue to make progress in translating the Nullness Checker to the Optional Checker. Also, we can look at other checkers, like the Subtyping Checker, to understand the architecture more easily.

Bibliography:

[1] M. Ernst, "Nothing is better than the optional type," 2015. [Online]. Available: http://homes.cs.washington.edu/~mernst/advice/nothing-is-better-than-optional.html. Accessed: Jan. 12, 2017.
[2] M. Ernst, "The Checker Framework Manual: Custom pluggable types for Java" in *Checker Framework*, 2017. [Online]. Available: http://types.cs.washington.edu/checker-framework/current/checker-framework-manual.pdf. Accessed: Jan. 12, 2017.