Alia Paddock (akp42) and Lemei Zhang (lemeiz)
Project 1: Pitch
28 March 2018

NullPointerExceptions are `Optional`

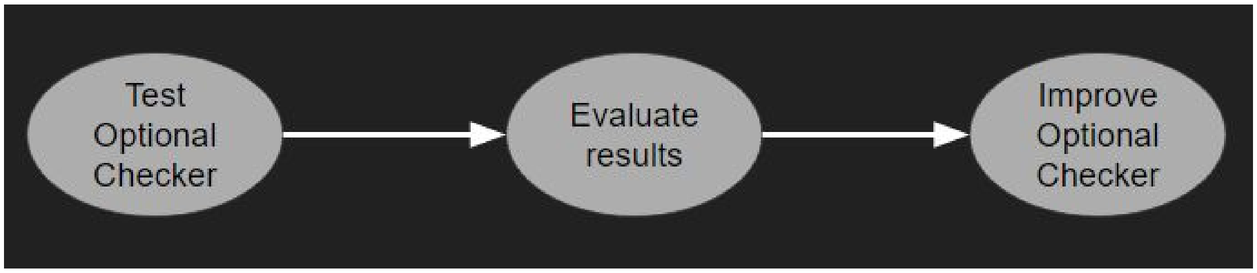
Vision

As Java programmers know, NullPointerExceptions are among the most common exceptions thrown in Java programs, and they're not very fun. Java 8 gave us the Optional class, which is intended to reduce (or, optimistically, eliminate) the bugs that cause NullPointerExceptions. But Optional, while arguably convenient in situations where it is used correctly, can be abused. Thus, with this project the aim is to evaluate and extend the Optional Checker, which is a part of Java's Checker Framework. We can test the effectiveness of the Optional Checker as it is, and then make improvements in areas where it can be more precise and robust.

There are those who are skeptical of the relative usefulness of Optional. Admittedly, it does have some downsides. The problems Optional aims to solve are still possible with its syntax. Using Optional can also introduce clutter, due to its syntax, and reduce efficiency, due to it introducing overheads. So, understandably, people often choose not to use Optional. However, it does offer some advantages. Its methods, such as orElse() and map(), reduce clutter and are built-in, so the user can rely on them. Also, more broadly, sometimes Optional is used just to emphasize that there is a possibility of getting a null value. This makes people remember to perform the necessary null checks. Where there is motivation to use Optional, there is motivation to have a good Optional Checker in the Java Checker Framework. Especially because using Optional can introduce its own new kinds of errors. There is a Nullness Checker in the Java Checker Framework, and its logic is similar to that of the Optional Checker, but it isn't quite the tool for the job when the problem is Optional-specific.

If the current Optional Checker is evaluated and extended, Java programmers can feel more confident in using Optional in their code. The whole purpose behind creating Optional in the first place was to reduce NullPointerExceptions, so having confidence that one is using Optional correctly gives one confidence that they are cognizant of null checking issues that may arise. Since incorrect null checking is such a common bug, this would be good news for Java programmers. Having a demonstrably precise and robust Optional Checker opens up the use of Optional to those who were hesitant to use it before.

Proposed Approach



To start, it would be prudent to run tests using the Optional Checker to get an idea about how it responds to different usages of Optional. This would involve cataloguing common mistakes with using Optional and running the Optional Checker on snippets of code that commit them. This shows us qualitatively what kind of error messages appear and how informative they are. It would also show us if the current Optional Checker fails to catch misuses of Optional that it should be catching. With this, we would have a good idea about how well the Optional Checker works and where it could use refinement or extension.

According to the Checker Framework manual, the current Optional Checker addresses these problems:

- Never use Optional.get() unless you can prove that the optional is present
- Prefer alternative APIs over Optional.isPresent() and Optional.get()
- Don't use Optional for the purpose of chaining methods from it to get a value
- Avoid using Optional in fields, method parameters, and collections
- Don't use Optional to wrap any collection type (List, Set, Map)

This gives us what the previous approach has already addressed. There is one problem that it does not address, however, which is that if an Optional chain has a nested Optional chain or an Optional intermediate result, it's probably too complex. This gives us one concrete way we could improve the Optional Checker.

Our approach could differ from the previous work done if we focus on the aforementioned unaddressed problem, and also the larger question of why some people don't like using Optional at all. In what ways can we make it worth a programmer's while to use Optional through improving the Optional Checker? The problems that have already been addressed all have to do with catching specific problematic uses of Optional. What if the Optional Checker could suggest places where Optional *could* be used, like where the methods orElse() or map() would work nicely? Can we find other ways in which using Optional would be a good choice or a bad choice?

Our approach is limited in the fact that most of the mentioned main problems with using Optional are already addressed by the Optional Checker, so we would have to get creative with finding problems that we could tackle in the improvement phase. It's also easier to find problematic uses of Optional than it is to try to find places where Optional is a good choice, where people would be willing to use it.

Risks

The biggest risk to this project, as we see it, is not coming up with good enough code samples for the evaluation phase. Because the direction of the improvement phase relies on the results of the evaluation phase, the success of the project relies on getting good evaluation data. We can minimize the risk of getting unreliable or otherwise misrepresentative results by having many code samples to test, and perhaps several different programmers to perform the tests so we get variety of perspectives. The more results we get in the evaluation phase, the more confident we can be about where we should go from there in order to improve the tool.