Extented Java Typechecking with Checker Framework

Software Analysis - Assignment 2

Luca Di Bello

April 22, 2024

Contents

1 Introduction

This assignment aims to extend the Java type system with the *Checker Framework*[1]. The *Checker Framework* is a powerful tool that integrates with the Java compiler to detect bugs and verify their absence at compile time. This is done by pluggable type-checkers, which via explicit annotations in the code, are able to check for a wide range of errors, such as null pointer dereferences, type casts, and array bounds. It includes over 20 type-checkers, which can be used to verify a wide range of properties. Some of the most useful type-checkers include:

- **Nullness Checker**: Prevent any `NullPointerException` by ensuring that variables are not null when dereferenced.
- **Index Checker**: Prevents array index out-of-bounds errors by ensuring that array accesses are always within bounds.
- **Regex Checker**: Prevents runtime exceptions due to invalid regular expressions by checking the syntax of regular expressions at compile time.

Note: a full list of available checkers can be found on the Checker Framework manual in the `Introduction` section

[1] `https://checkerframework.org/`

To showcase the effectiveness of this tool, the Checker Framework's index checker has been integrated into an existing codebase to prevent any out-of-bounds access to arrays. The project is a legacy Java library named `RxRelay`[2], a small library that aims to extend the capabilities of a famous Java library called `RxJava` by providing a set of `Relay` classes that act as both an `Observable` and a `Consumer`. The library is used to relay events from one component to another, and it is widely used in Android applications. [1]

The library is composed of a single package, `com.jakewharton.rxrelay3`, and it contains a total of 5 Java files

- `Relay.java`: An interface that extends both RxJava's `Observable` and `Consumer` interfaces. This interface is implemented by all the relay classes to provide a general API for relaying events.
- `AppendOnlyLinkedList.java`: An unconventional unbounded linked list implementation that is used by several relay classes to store events. Rather than using a traditional Node-based linked list, this implementation uses a single array, which is then expanded when it reaches its capacity. This is done using a clever trick that allows the array to be expanded without the need to copy the elements from the old array to the new one.
- `PublishRelay.java`: A concrete implementation of the `Relay` interface that relays events to all the subscribers that are currently subscribed to it.
- `ReplayRelay.java`: Another `Relay` implementation that relays events to all the subscribers that are currently subscribed to it, but it also caches a certain number of events and replays them to new subscribers when they subscribe.
- `BehaviorRelay.java`: A `PublishRelay` implementation that relays only the most recent event to new subscribers. This is done by caching the most recent event and replaying it to new subscribers when they subscribe.

[2] `https://github.com/JakeWharton/RxRelay`

2 Project setup

The project uses Java 8 and uses *Maven* as a build system. The *Checker Framework* is integrated into the project using the `checker-qual` dependency, which provides the necessary annotations to use the checkers. The `maven-compiler-plugin` has been configured to use the *Checker Framework* as annotation processor and to Google's `Error Prone` as a compiler plugin to provide additional static analysis checks.

Additional compiler flags have been added to the `maven-compiler-plugin` configuration to enable additional features of the framework. The configuration is as follows:

- `-Xmaxerrs 10000`: Set the maximum number of errors to display before stopping the compilation process.
- `-Xmaxwarns 10000`: Similar to `-Xmaxerrs`, but for warnings.
- `-Awarns`: Show *Checker Framework* errors as warnings instead of errors to allow the compilation process to continue even if errors are found.
- `-AresolveReflection`: Enable the reflection resolver, which is used to resolve reflection calls at compile time. This is useful to infer the type of reflection calls such as `Array.newInstance` or `Class.forName`.
- `-ArequirePrefixInWarningSuppressions`: Require the `SuppressWarnings` annotation to have a prefix that matches the checker name. This is useful to prevent accidental suppression of warnings from other checkers. For example, to suppress a warning from the `Index Checker`, the annotation should formatted as `SuppressWarnings("index:<specific_error_to_suppress>")`.
- `-AassumeAssertionsAreEnabled`: This flag enables the framework to infer additional information about the assertions made in the code. This is useful to help the framework understand complex control flows.

A small `Makefile` along with some scripts have been provided to simplify the testing and compilation of the project. The `Makefile` contains the following targets:

- `setup`: Setup the project by setting the right Java version and installing the necessary dependencies using *Maven*.
- `build`: Build the project using *Maven*.
- `compile`: Compile the project using *Maven*.
- `test`: Run unit tests and mock tests using *Maven*.

3 Integration of the Index Checker: Challenges and Solutions

The *Index Checker* has been successfully integrated into the project by adding the necessary annotations to the codebase to ensure that all array accesses are within bounds. At first, the checker was able to find several errors, which were then fixed one by one using the provided set of annotations.

As certain Java idioms used in the project were not supported out-of-the-box by the checker as they would require reflection to be resolved, for this reason the `-AresolveReflection` flag was enabled to allow the checker to resolve reflection calls at compile time. Without this flag, the checker would not be able to infer the type of reflection calls such as `Array.newInstance` or `Class.forName`, used several times in the project specifically in the `AppendOnlyLinkedList` class to expand the array when it reaches its capacity.

Unfortunately, even after enabling the reflection resolver, the checker was unable to resolve certain reflection calls, specifically the ones related to creating new arrays using `Array.newInstance`. For example, the following code snippet was not correctly inferred by the checker:

```
if (array.length < s) {
    // create a new array of size s
    array = (t[]) array.newinstance(array.getclass().getcomponenttype(),
        s);
}
```

Listing 1: Reflection call to create a new array

To fix this issue, the checker was instructed to leverage assertions to provide additional information about the code. This was done by enabling the `-AassumeAssertionsAreEnabled` flag. Along with this, as explicitly advised in the documentation of the checker, each assertion is accompanied by a comment that explains what checkers it is intended to help, with a brief description of why the assertion was needed. The code before has been modified as follows:

```
if (array.length < s) {
    // create a new array of size s
    array = (t[]) array.newinstance(array.getclass().getcomponenttype(),
        s);

    // As the CheckerFramework is not able to infer that the array is of
        length s, we need to make an assertion
    assert array.length == s: "@AssumeAssertion(index): The array is
        exactly of length s, as we have just created it";
}
```

Listing 2: Using assertions to help the Index Checker infer the array length

4 Conclusions

References

[1] J. Wharton. Rxrelay - readme.md. `https://github.com/JakeWharton/RxRelay/blob/master/README.md`. Last accessed: 19.04.2024.