Assignment 3 – Software Analysis

Extended Java Typechecking

Claudio Maggioni

1 Project selection

The assignment description requires to find a project with more than 1000 lines of code making significant use of arrays or strings.

Given these requirements, I decide to analyze the Apache Commons Text project in the GitHub repository **apache/commons-text**.

1.1 The Apache Commons Lang Project

The Apache Commons family of libraries is an Apache Software Foundation[1] sponsored collection of Java libraries designed to complement the standard libraries of Java. The Apache Commons Text project focuses on text manipulation, encoding and decoding of *String*s and *CharSequence*-implementing classes in general.

All the source and test classes are contained within in the package *org.apache.commons.text* or in a sub-package of that package. For the sake of brevity, this prefix is omitted from now on when mentioning file paths and classes in the project.

I choose to analyze the project at the *git* commit `78fac0f157f74feb804140613e4ffec449070990` as it is the latest commit on the *master* branch at the time of writing.

To verify that the project satisfies the 1000 lines of code requirement, I run the *cloc* tool. Results are shown in table 1. Given the project has more than 29,000 lines of Java code, this requirement is satisfied.

Language	Files	Blank	Comment	Code
Java	194	5642	18704	26589
XML	16	205	425	1370
Text	6	194	0	667
Maven	1	23	24	536
YAML	6	39	110	160
Markdown	4	40	106	109
Velocity Template Language	1	21	31	87
CSV	1	0	0	5
Properties	2	2	28	5
Bourne Shell	1	0	2	2
Total	232	6166	19430	29530

Table 1: Output of the *cloc* tool for the Apache Commons Text project at tag *78fac0f1* (before refactoring is carried out).

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

2 Running the CheckerFramework Type Checker

The relevant source code to analyze has been copied to the directory *sources* in the assignment repository

usi-si-teaching/msde/2022-2023/software-analysis/maggioni/assignment-3

on *gitlab.com*. The Maven build specification for the project has been modified to run the CheckerFramework extended type checker (version 3.33.0) as an annotation processor to be ran on top of the Java compiler. Both source code and test code is checked with the tool for violations, which are reported with compilation warnings. To run the type checker simply run:

```
mvn clean compile
```

in a suitable environment (i.e. with JDK 1.8 or greater and Maven installed). To additionally run the Apache Commons Text test suite and enable `assert` assertions (later useful for CheckerFramework `@AssumeAssertion(index)` assertions) simply run:

```
env MAVEN_OPTS="-ea" mvn clean test
```

The state of the assignment repository when the type checker was first ran successfully is pinned by the *git* tag *before-refactor*. A copy of the CheckerFramework relevant portion of the compilation output at that tag is stored in the file *before-refactor.txt*.

No CheckerFramework checkers other than the index checker is used in this analysis as the code in the project mainly manipulates strings and arrays and a significant number of warnings are generated even by using this checker only..

3 Refactoring

Warning type	Before refactoring	After refactoring
argument	254	241
array.access.unsafe.high	130	117
array.access.unsafe.high.constant	31	28
array.access.unsafe.high.range	22	22
array.access.unsafe.low	59	58
array.length.negative	3	3
cast.unsafe	2	2
override.return	12	12
Total	513	483

Table 2: Number of CheckerFramework Type Checker warnings by category before and after refactoring.

Table 2 provides a summary on the extent of the refactoring performed in response to index checker warnings across the Apache Commons Text project. In total, 513 warnings are found before refactoring, with 30 of them later being extinguished by introducing annotations and assertions in the code in the following classes:

- AlphabetConverter
- StringSubstitutor
- similarity.LongestCommonSubsequence
- translate.AggregateTranslator
- translate.CharSequenceTranslator
- translate.CodePointTranslator
- translate.CsvTranslators
- translate.JavaUnicodeEscaper
- translate.SinglePassTranslator
- translate.UnicodeEscaper

Listing 1 Method *toMillisInt(Duration)* of class *time.DurationUtils* in Apache Commons Lang 3.12.0.

```
139 public static int toMillisInt(final Duration duration) {
140     Objects.requireNonNull(duration, "duration");
141     // intValue() does not do a narrowing conversion here
142     return LONG_TO_INT_RANGE.fit(Long.valueOf(duration.toMillis())).intValue();
143 }
```

4 Conclusions

Did using the checker help you find any bugs or other questionable design and implementation choices?

¿ no bugs found, couple of design choices

How complex was it to apply the checker, and what benefits did you gain in return?

¿ not so complex, lots of false positives

Compare the checker's trade-off between complexity of usage and analysis power to that of other software analysis techniques you're familiar with (in particular, those used in previous assignments).