Assignment 1 – Software Analysis

Static Analysis with Infer

Claudio Maggioni

1 Project selection

Given that this assignment draws parallels with the class of Software Design and Modelling of last semester, specifically regarding static analyzers, I choose to analyze the same project I analyzed in the past with PMD and SonarQube using Infer[1] to make for an interesting comparison between static analysis paradigms.

The project I analyze is therefore **apache/commons-lang**.

1.1 The Apache Commons Lang Project

The Apache Commons family of libraries is an Apache Software Foundation[2] sponsored collection of Java libraries designed to complement the standard libraries of Java. The Apache Commons Lang project focuses on classes that would have fitted in the *java.lang* package if they were included with Java.

All the source and test classes are contained within in the package *org.apache.commons.lang3* 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 version 3.12.0 of the library (i.e. the code under the *git* tag *rel/commons-lang-3.12.0*) because it is the same version analyzed during the SDM class.

To verify that the project satisfies the 5000 lines of code requirement, I run the *cloc* tool. Results are shown in table 1. Given the project has more than 86,000 lines of Java code, this requirement is satisfied.

Language	Files	Blank	Comment	Code
Java	409	15,790	60,363	86,056
HTML	22	1,015	100	13,028
Text	30	1,858	0	12,415
XML	38	434	539	4,819
Maven	1	31	37	940
JavaScript	5	21	78	698
Markdown	3	38	0	202
CSS	4	36	66	140
Velocity Template Language	1	23	31	90
Groovy	1	12	22	81
YAML	3	12	42	55
Bourne Shell	1	0	2	2
Total	518	19,270	61,280	118,526

Table 1: Output of the *cloc* tool for the Apache Commons Lang project at tag *rel/commons-lang-3.12.0* (before fixes are applied).

[1] `https://fbinfer.com/`
[2] `https://apache.org/`

2 Running the Infer tool

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

usi-si-teaching/msde/2022-2023/software-analysis/maggioni/assignment-2

on *gitlab.com*. The script *docker-infer.sh* can be executed to automatically run the Infer tool using default options through the course tools docker image *bugcounting/satools:y23*.

The script selects Java 17 to compile the project as this is required to not make the Animal Sniffer Maven build plugin fail with the message "This feature requires ASM7".

The script executes Infer in Maven capture mode executing the *compile* and *test* targets while disabling the Apache RAT software license checker (which fails for this release). Since unit tests are executed, running the script before and after the warning guided refactoring ensures the fixes I introduce do not introduce regressions.

The analysis outputs are located in *before/infer-out/report.txt*.

3 Results

Table 2 shows the results of the analysis performed by Infer providing comments on true and false positives and the actions taken for each result.

File	**Line**	**Kind**	**True Pos.**	**Reason why flagged expression is a false positive**
AnnotationUtils.java	72	Null	Yes	–
reflect/MethodUtils.java	486	Null	Yes	
reflect/FieldUtils.java	126	Null	Yes	
concurrent/MultiBackgroundInitializer.java	160	Thread Safety	Yes	
builder/ToStringBuilder.java	223	Null	No	Infer flags the value `null` when used as a nullable method argument
builder/ReflectionToStringBuilder.java	131	Null	No	
time/DurationUtils.java	142	Null	No	The method which may return a null value returns a non-null value if its parameter is non-null, and a non-null parameter is given
CharSetUtils.java	181	Null	No	According to *java.lang* documentation, the method always returns a non-null value
reflect/FieldUtils.java	341	Null	No	A utility method is used to guard the dereference reported with an exception throw
reflect/FieldUtils.java	385	Null	No	
reflect/FieldUtils.java	599	Null	No	
reflect/FieldUtils.java	644	Null	No	
reflect/MethodUtils.java	987	Null	No	The method which may return a null value returns a non-null value if its parameter is non-null, and a non-null parameter is always given according to the *java.lang* documentation for the inner nested method

Table 2: Results of the Infer static analysis tool execution with default options. *True Pos.* denotes whether a result is a true positive, while *Kind* denotes with *Null* and *Thread Safety* respectively null dereference warnings and thread safety violations.

In total Infer reports 13 warnings, 12 of which are null dereference warnings and 1 is a thread safety violation. Of all warnings, 4 are true positives and 9 are false positives, resulting in a false positive ratio of over 69%. These values are summarized in table 3.

Total number of warnings:	13
Null dereference warnings:	12
Thread safety violations:	1
True positives:	4
False positives:	9

Table 3: Quantitative results of the static analysis performed by Infer.

3.1 False Positives

As can be deduced from table 2, Infer especially struggles at determining Vthe null safety of nested method calls or non-trivial data flow paths. This is sometimes caused by insufficient knowledge of the nullability contracts of the Java standard library (e.g. *java.lang* package).

In other cases, the flagged expression is guarded by an utility method throwing an exception if its value is indeed `null`. One of such guards is the `static Validate.notNull(Object, String, Object...)` method, which checks if its first argument is null and throws a `NullPointerException` if so.

Additionally, Infer seems to struggle with boxed primitives and not understanding that their value is always non-null by construction. As an example I provide the warning reported in the file *time/DurationUtils.java*:

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 }
```

Here Infer reports that the first argument of `LONG_TO_INT_RANGE.fit(Long)` may be null. However, the return value of `Long.valueOf(long)` is always non-null since the method simply boxes its `long` argument.

Finally, some warnings are caused by Infer flagging the use of the `null` keyword as a method argument for methods that would accept a nullable argument in that position without causing null dereferences. This warning could point to a potential ambiguity is selecting the right method to call at compile time given the argument types (i.e. static dispatching), as `null` is a valid expression of all object types. For example, in the given class:

```
class C {
  m(String s) {}
  m(Object o) {}
}
```

A call to `C.m(null)` would be ambiguous as `null` is both a `String` and an `Object`, thus a cast to either type woule be required to make the code compile. However, the warnings reported by Infer do not present such ambiguity as in those cases overloaded methods have different numbers of parameters. Additionally, even introducing explicit casts for all `null` arguments does not extinguish the warning. Therefore, I can not find a conclusive explaination on the nature of the false positive, however I can

attest to these instances being indeed false positives by having manually verified that the methods in question indeed can accept a null value without causing null dereferences.

3.2 True Positives

In this section I now cover the warnings that are true positives and thus are causes for refactoring.