SWE_261P_ Project_Part_6
Team Members:
· Yuxin Huang: yuxinh20@uci.edu
· Changhao Liu: liuc50@uci.edu
· Ruokun Xu: ruokunx@uci.edu
Team Name: OffersAreHere
Github link: https://github.com/yx-hh/commons-io

Introduction to Static Analysis:

Before discussing static analysis tools, we should first learn more about what static analysis is and its purposes. Static analysis, as its name implies, is to seek for potential bugs in software programs without running the program. Its goal is to identify potential issues in the program in the early stage.

There are several ways to conduct static analysis and one of them is through code review, which is to invite other developers to examine the codes. Usually, developers will utilize their past experience and intelligence to provide some feedback and may find some flaws. Such practice allows the development team to find and fix bugs in early stages. There are a few good practices for code reviews (Jones, pg 3):

- Review small chunks of the program at a time(Jones, pg 3)
- Add comments and feedback(Jones, pg 3)
- Review the code individually(Jones, pg 3)
- Create some checklist for essential actionable items(Jones, pg 3)

Besides code review by developers, another way of static analysis is to utilize some automatic static analysis tools, such as Checkstyle or SportBug etc. We will discuss the usage of those tools in the next section. However, some tools may display some warnings that may not be actual problems (Jones, pg 6), so we should be critical about the result and take actions accordingly.

Benefits of Static Analysis:

There are some benefits brought by static analysis. Here is a list of benefits that we concluded (Hamilton, 2022):

- Detect potential issues early
- Make good use of every developer's work experience and intelligence to locate defects in design or implementation
- Reduce time and cost for testing
- Facilitates developers' teamwork in the project

- Makes the codebase maintainable, as review checks whether the new codes are consistent with the coding style

Two Different Static Analyzers' usage:

Checkstyle - Static Analyzer

It checks whether Java code is consistent with a specific coding standard in different modes such as Sun Checks Mode or Google Checks Mode..

1. Sun Checks Mode

We first installed the Checkstyle plugin, and then clicked Analyze -> Inspect Code. Results are as Figure

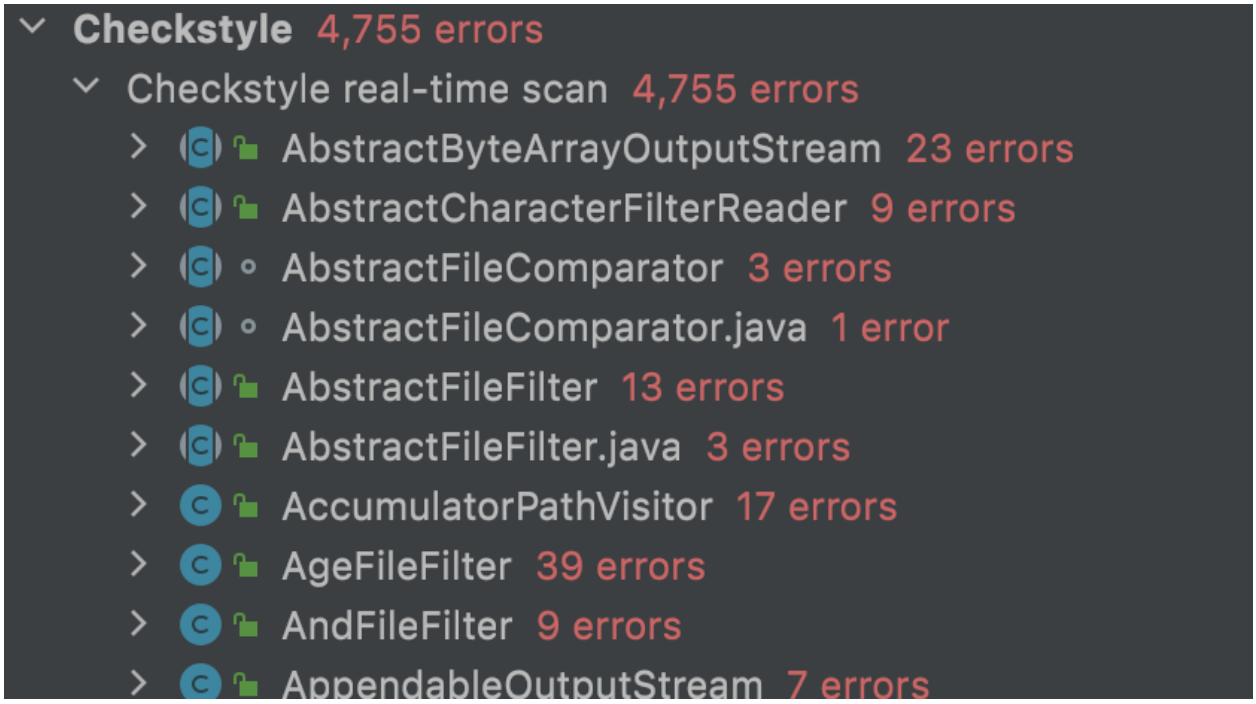

Figure 1, Sun Checks result, by Ruokun Xu

- New line: '+' should be on a new line when nothing follows '+'. It is not a problem because the position of '+' will not affect results when executed. See Figure 1-1 below.

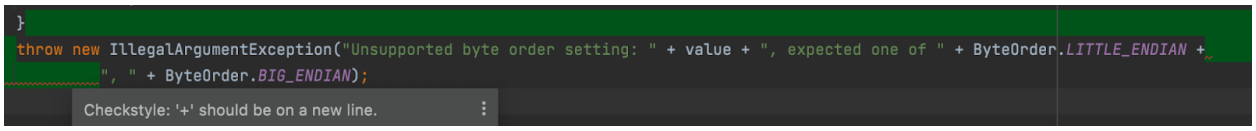

Figure 1-1, new line, by Ruokun Xu

- Long line: Line is longer than 80 characters. It is not a problem because the length of each line will not affect results when executed. See Figure 1-2 below.

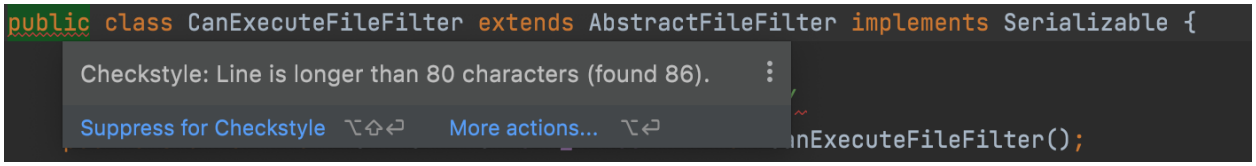

Figure 1-2, long line, by Ruokun Xu

- Declaration: Cass should be declared as final. It is not a problem because the declaration of final will not affect results when executed. See Figure 1-3 below.

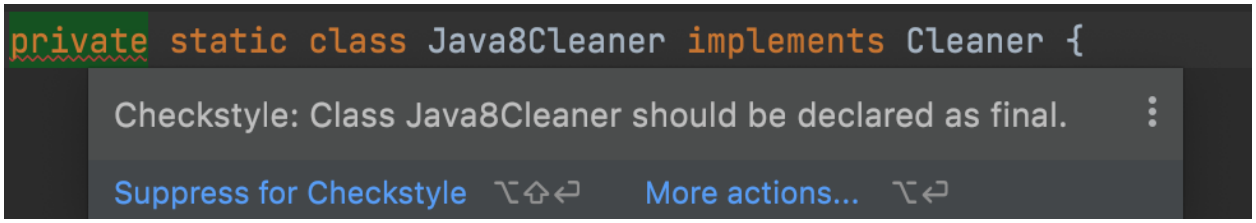

Figure 1-3, declaration, by Ruokun Xu

2. Google Checks Mode

Click Analyze -> Inspect Code. Results are as Figure 2.

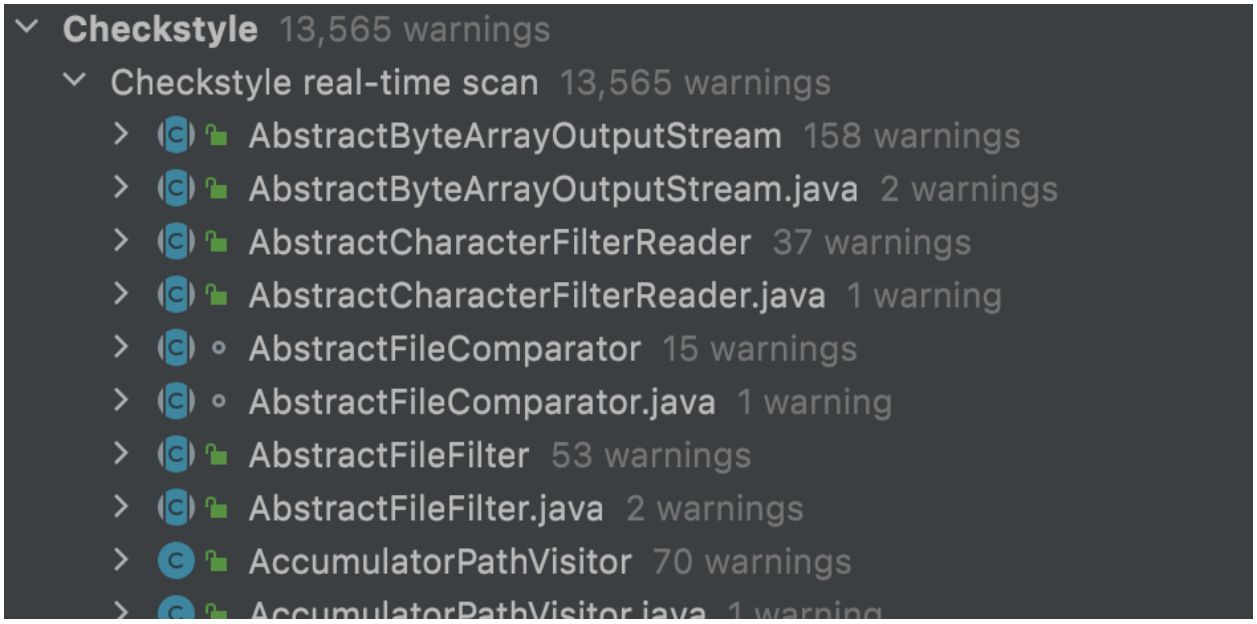

Figure 2, Google Checks result, by Ruokun Xu

- Incorrect indentation: the expected indentation is 2 tabs, and we got an error warning because there are 4 tabs. It is not a problem because the number of tabs will not make an error when executed. See Figure 2-1 below.

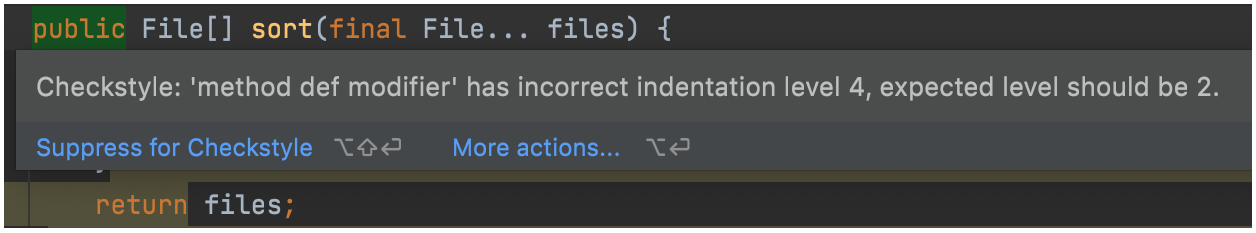

Figure 2-1, indentation, by Ruokun Xu

- Package position: ‘package’ should be isolated. It is not a problem because the position of the package line will not make errors when executed. See Figure 2-2 below.

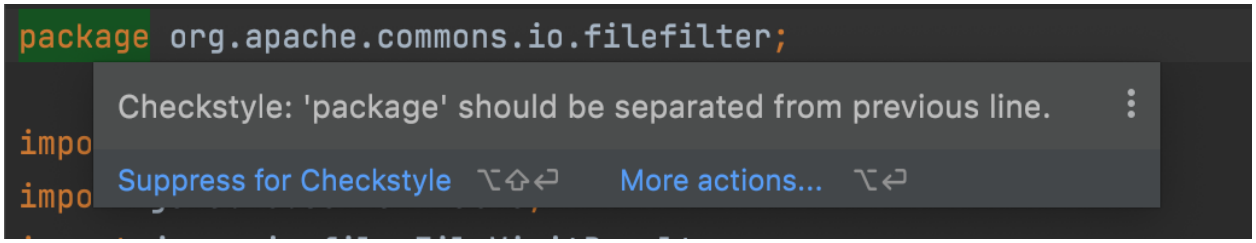

Figure 2-2, line position, by Ruokun Xu

- White space: white space should be added. It is not a problem because missing whitespace will not make an error when executed. See Figure 2-3 below.

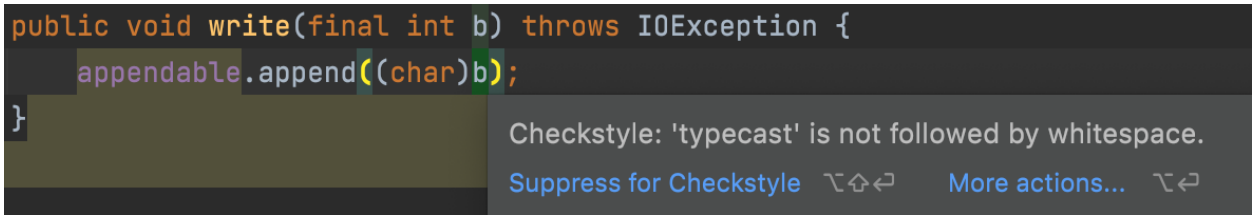

Figure 2-3, whitespace, by Ruokun Xu

Infer - Static Analyzer

Infer is a static analysis tool that examines your Java or C/C++/Objective-C code and identifies some potential issues. Infer allows developers to locate and fix bugs before shipping them to production. Also, it prevents potential crashes or poor performance in the program (*Infer Static Analyzer*, 2022).

We ran Infer our Commons-IO project to check some potential issues, including null pointer exceptions, resource leaks, annotation reachability, missing lock guards, and concurrency race conditions (*Infer Static Analyzer*, 2022).

Install Infer in MacOS system

Infer only supports Intel Chip. We can execute **brew install infer** command in terminal to install this software.

Run Infer for Maven Project
We can run **infer -- mvn package -Drat.skip=true -PMaster** to analyze the master branch

Report by Infer:
Figure 3 presents the **report.txt** under the folder of **infer-out**

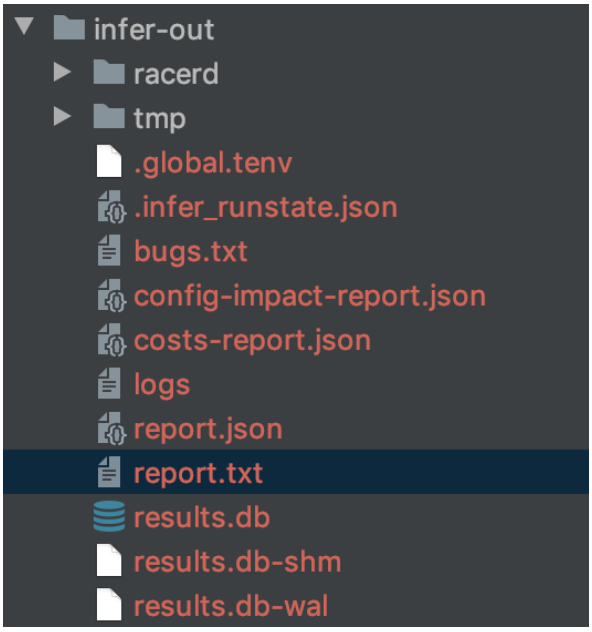

Figure 3, report location, by Yuxin Huang

There are a total of 100 issues and warnings found in common-io projects, including 56 Thread Safety violations, 4 Resource Leak warnings, and 3 Null Dereference. Figure 4 illustrates nume of issues and issue types reported by Infer. Figure 5 is a detailed example of an issue that Inter found: It displays the serial number of the issue, the issue type and detailed code line with the problem in order.

```
Found 100 issues
                        Issue Type(ISSUED_TYPE_ID): #
  Thread Safety Violation(THREAD_SAFETY_VIOLATION): 56
                     Resource Leak(RESOURCE_LEAK): 41
                 Null Dereference(NULL_DEREFERENCE): 3
```

Figure 4, issue summary, by Yuxin Huang

```
#0
src/main/java/org/apache/commons/io/StreamIterator.java:49: error: Resource Leak
  resource of type `org.apache.commons.io.StreamIterator` acquired by call to `new()` at line 49 is not released after line 49.
  47.       @SuppressWarnings("resource") // Caller MUST close or iterate to the end.
  48.       public static <T> Iterator<T> iterator(final Stream<T> stream) {
  49. >         return new StreamIterator<>(stream).iterator;
  50.       }
  51.
```

Figure 5, issue style, by Yuxin Huang

Issue example

Thread Safety Violation Example

Figure 6 is an example of Thread Safety Violation found by Infer. It points out that a non-private method writes `this.pos` field out of synchronization.In figure 7, we can see that pos is a private field of the BoundedInputStream class. For the threads inside of this class, `pos` field is a public variable. If two threads operate on this `pos` at the same time, it will have a Thread Safety Violation issue.

```
#89
src/main/java/org/apache/commons/io/input/BoundedInputStream.java:220: warning: Thread Safety Violation
  Unprotected write. Non-private method `BoundedInputStream.skip(...)` writes to field `this.pos` outside of synchronization.
 Reporting because another access to the same memory occurs on a background thread, although this access may not.
  218.           final long toSkip = max>=0 ? Math.min(n, max-pos) : n;
  219.           final long skippedBytes = in.skip(toSkip);
  220. >         pos+=skippedBytes;
  221.           return skippedBytes;
  222.       }
```

Figure 6, Thread Safety Violation Issue, by Yuxin Huang

```
/** the number of bytes already returned */
private long pos;
```

Figure 7, pos field location, by Yuxin Huang

Resource Leak Example

Figure 8 is an example of Resource Leak reported by Infer. It points out resources obtained by sw are not released: It may cause resource leak. This is an actual issue because sw is an object of CharArrayWriter, which will use IO resources if not closed after using.

```
#43
src/main/java/org/apache/commons/io/IOUtils.java:2770: error: Resource Leak
  resource of type `java.io.CharArrayWriter` acquired to `sw` by call to `new()` at line 2768 is not released after line 2770.
**Note**: potential exception at line 2769
  2768.           final CharArrayWriter sw = new CharArrayWriter();
  2769.           copy(reader, sw);
  2770. >         return sw.toCharArray();
  2771.       }
  2772.
```

Figure 8, Resource Leak Example Issue, by Yuxin Huang

Null Dereference Example

Figure 9 presents an example of the Null Dereference identified by Infer. It points out that countDirectory (directory) can be null and will cause Null Dereference issue. This issue will happen when the countDirectory method returns a null value and raises a NullPointerException.

```
#39
src/main/java/org/apache/commons/io/file/PathUtils.java:1459: error: Null Dereference
  object returned by `countDirectory(directory)` could be null and is dereferenced at line 1459.
  1457.         */
  1458.        public static long sizeOfDirectory(final Path directory) throws IOException {
  1459. >          return countDirectory(directory).getByteCounter().getLong();
  1460.        }
  1461.
```

Figure 9, Null Dereference Example Issue, by Yuxin Huang

Contrast Tools

The tools, Checkstyle and Infer are fundamentally different in their purposes

- Checkstyle aims to check code format and with different standards, including length of line, declaration, and indentation.
- Infer aims to check null pointer exceptions, resource leaks, annotation reachability, missing lock guards, and concurrency race conditions

Reference

Jones. (2022a). ***Code Review, and Automated Static Analysis Tools*** **[Slides]. Canvas. https://canvas.eee.uci.edu/courses/43617/files/folder/Lecture%20Slides?preview=17870946**

Hamilton. (2022). ***What is Static Testing? What is a Testing Review?*** **Guru99. https://www.guru99.com/testing-review.html**

Infer Static Analyzer**. (2022). Infer. https://fbinfer.com/**