CPEN522: Assignment4

Jiahui Tang 77383255
Srikanthan Thivaharan 41497900

2019/04/02

1 Automated Test Generation

1.1 Working with Monkey

For this task, we tested the open source Android application: MovieGuide. After we downloaded it from github, we requested themoviedb.org API key and built the application in Android Studio. Figure 1 shows the screenshot of running the application in a virtual device (Nexus 5X API 27).

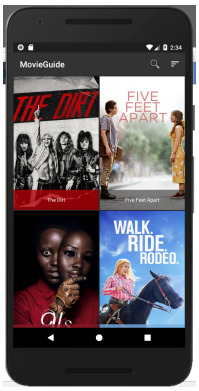


Figure 1: Running MovieGuide on an emulator.

Then we tried to use Monkey to automatically generate manual tests on the application. Since we can set the options of running Monkey, in our case, we chose to use the commandline below to do the experiment: ***adb shell monkey -p com.esoxjem.movieguide -s 500 –ignore-crashes –ignore-timeouts –monitor-native-crashes -v -v -v 100000***. We specified 100000 events for this Monkey run, so as shown in Figure 2, this run took more than 5 minutes (469.3 seconds).

```
:Sending Trackball (ACTION_MOVE): 0:(-2.0,3.0)
Events injected: 100000
:Sending rotation degree=0, persist=false
:Dropped: keys=62 pointers=25 trackballs=0 flips=0 rotations=0
## Network stats: elapsed time=469301ms (0ms mobile, 0ms wifi, 469301ms not connected)
// Monkey finished
```

Figure 2: The time Monkey took to execute 100000 events.

1.1.1 Coverage of Monkey

Since Monkey is random testing, we do not know exactly the code coverage of every Monkey run. Luckily, we found a way to get the code coverage of Android application which is using **Jacoco (Updated**

from Emma)[1][2] to get the coverage.ec file. To do this, we need to first instrument the source code to listen the start and finish of MovieGuide application. We added three files called FinishListener.java, InstrumentedActivity.java and JacocoInstrumentation.java, so the app structure became to Figure 3. Besides, we also need to modify AndroidManifest.xml and build.gradle to enable test coverage. All the changes can be found in the appendixA to E.

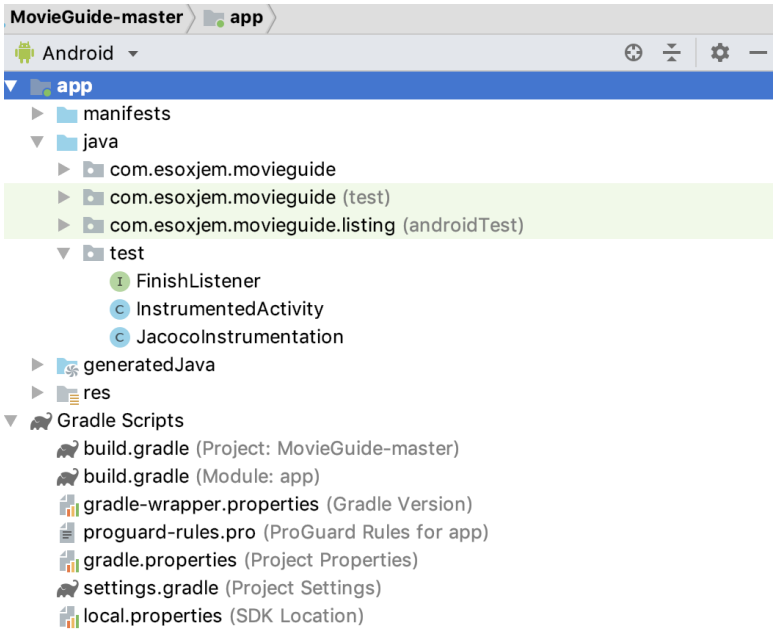


Figure 3: MovieGuide project structure after adding three instrumentation files.

After we rebuilt and installed the application into the emulator, we need to use commandline ***adb shell am instrument com.esoxjem.movieguide/test.JacocoInstrumentation*** to start the application. Then we tried to run Monkey by using the same commandline which generated 100000 events. According to the paper[3], the coverage of Monkey converges after about five minutes running. In our case, we have executed Monkey for more than 5 minutes, so the coverage rate should be likely correct.

When we finished the Monkey test, then we dumped the coverage.ec file from *emulator/data/data/-com.esoxjem.movieguide/files/coverage.ec*. Then we ran ***gradle jacocoTestReport***, we can get the HTML coverage report shown as Figure 4. The coverage report can be found in app/build/reports folder. As we can see from the report, the code coverage is about 53%. If we exclude the unrelated class like test and io.realm, the coverage rate will be higher. Still, the result is as expected according to the paper (The code coverage rate converges at about 50% on average). The code coverage report can be found on our github repo[4].

[1]https://www.jianshu.com/p/9990512ea689
[2]https://blog.csdn.net/itfootball/article/details/45618609
[3]Choudhary SR, Gorla A, Orso A. Automated test input generation for android: Are we there yet?(e). In2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE) 2015 Nov 9 (pp. 429-440). IEEE.
[4]https://github.com/UBC-TestingCourse/group8/tree/master/MovieGuide-master/jacocoreports/jacoco/jacocoTestReport

app

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
io.realm		19%		8%	188	213	483	602	31	56	1	5
com.esoxjem.movieguide.listing		72%		39%	60	130	87	314	20	84	0	15
com.esoxjem.movieguide		69%		29%	54	118	76	247	35	99	2	18
test		34%		12%	24	32	51	84	7	15	0	2
com.esoxjem.movieguide.favorites		65%		41%	25	48	32	90	15	37	0	6
com.esoxjem.movieguide.details		90%		45%	44	100	17	255	6	58	0	10
com.esoxjem.movieguide.network		64%		24%	26	42	11	57	5	21	1	5
com.esoxjem.movieguide.listing.sorting		89%		47%	28	85	3	160	3	54	0	11
com.esoxjem.movieguide.util		59%		50%	9	18	12	30	5	13	1	3
Total	3,398 of 7,290	53%	529 of 693	24%	458	786	772	1,839	127	437	5	75

Figure 4: Jacoco Report for code coverage.

1.1.2 Can Monkey find bugs?

Because Monkey test is random stress test, the original aim is not to find the bugs of the applications, but we can still find if there are some errors during the execution from the log file. More ideally, we can generate bug report for each Monkey test. In our case, we took advantage of chkBugReport.jar[5] to get the system bug report.

Beside, to make the test more automatic, we wrote a simple Python script to help us to execute Monkey, get the bug report step by step. The Python code can be found at Appendix F.

Additionally, we also changed the source code so that there is a bug in the application. We changed ***onFavoriteClick()*** method as shown in Figure 5. When the user clicks Favorite button, then an NullPointerException error will be thrown.

```
    @Override
    public void onFavoriteClick(Movie movie) {
        if (isViewAttached()) {
            boolean isFavorite = favoritesInteractor.isFavorite(movie.getId());
            if (isFavorite) {
                favoritesInteractor.unFavorite(movie.getId());
                view.showUnFavorited();
            } else {
                favoritesInteractor.setFavorite(movie);
                view.showFavorited();
                throw new NullPointerException("test code for CPEN522");
            }
        }
    }
}
```

Figure 5: Change the source code to add a bug inside the app.

Then we ran our script to get the system bug report. Figure 6 shows the result of bug report. There were 11 errors occurred during the monkey execution. We looked deeper into those errors, and we found that only the first six errors are directly related to MovieGuide application. All those six errors are actually because of the bug we introduced before. Figure 7 indicates that the errors are fatal exceptions (AndroidRuntime: java.lang.NullPointerException: test code for CPEN522). The source code and bug report can also be found on our github repo[6].

[5] https://github.com/sonyxperiadev/ChkBugReport/wiki/Quick-instructions
[6] https://github.com/UBC-TestingCourse/group8/tree/master/MovieGuide-master/runMonkey

Figure 6: System bug report.

Figure 7: Fatal exception we introduced.

1.1.3 Summary

To sum up, Monkey is very easy to use since we do not need to download any package or tool. Monkey is integrated into Android studio by using command lines. Monkey can be used to improve the application stability, it can detect some unknowable situations and problems that can not be detected by normal clicks. However, based on our experience, we still think that there are some limitations of the tools: first, from our code coverage report, it is easy to find that the overall coverage is not sufficient, also normally the user need to run Monkey multiple times to achieve maximum coverage. Second, different from DynoDroid, Monkey does not support manual provided inputs, so some bugs or problems of the input fields of the application are hard to be found. Third, the outputs of Monkey is hard to understand for normal users. In our test, we applied some third part tools to help us to generate the report. If we do not use those tools, we need to analyze the log file manually, which will take more time than we did right now. Hence, it would be better if Monkey is able to generate a detailed report.

1.2 Working with Randoop

1.2.1 Test Generation using Randoop

Randoop generates unit tests using feedback-directed random test generation. The tool is able to generates random tests for a given set of Java classes during a limited time interval, which is preset by the tester. The test engine uses Java's ability to introspect about each class's type structure and generates random test sequences made up from constructors and methods published in each test class's public interface.

In this task, we used Randoop to generate random tests for Japacman. Before doing that, we should download the relevant Randoop jar files: randoop-all-4.1.1.jar and junit-4.13-beta-2.jar (for test case minimization). To use it, we can either set the environment variables ($(RANDOOP_JAR) and $(RANDOOP_PATH)) or use the absolute paths of the jars. In our case, we used the former way. We used the command line below to generate the test cases:

java -classpath $RANDOOP_PATH/target/classes/:$RANDOOP_JAR randoop.main.Main gentests –classlist=myclasses1.txt –time-limit=30 –flaky-test-behavior=DISCARD.

As we can see here, we specified 30 seconds to generate tests and discarded the flaky test behavior. Additionally, in the myclasses1.txt file, we added all the classes of org.jpacman.framework.model. Figure 8 shows the generation process: we finally got 132 Error-revealing test cases and 1281 Regression test cases in 30 seconds. The average method execution time for normal termination is 0.000347 second.

```
⚙ tangjiahui@spphins-2 ~/eclipse-workspace/group8/assignment1/jpacman-framework  master java -classpath $RANDOOP_PATH/target/class
es/:$RANDOOP_JAR randoop.main.Main gentests --classlist=myclasses1.txt --time-limit=30 --flaky-test-behavior=DISCARD
Ignoring interface org.jpacman.framework.model.IBoardInspector specified via --classlist or --testclass; provide classes, not interfaces.
Ignoring interface org.jpacman.framework.model.IGameInteractor specified via --classlist or --testclass; provide classes, not interfaces.
Ignoring interface org.jpacman.framework.model.IPointInspector specified via --classlist or --testclass; provide classes, not interfaces.
PUBLIC MEMBERS=87
Explorer = randoop.generation.ForwardGenerator(allSequences.size()=0,observers.size()=0,subsumed_sequences.size()=0,runtimePrimitivesSeen.
size()=38)

Progress update: steps=1, test inputs generated=0, failing inputs=0      (Mon Apr 01 21:15:25 PDT 2019)
Progress update: steps=1000, test inputs generated=820, failing inputs=39
Progress update: steps=2000, test inputs generated=1740, failing inputs=81
Progress update: steps=3000, test inputs generated=2644, failing inputs=124
Progress update: steps=3124, test inputs generated=2761, failing inputs=132      (Mon Apr 01 21:15:55 PDT 2019)
Normal method executions: 491401
Exceptional method executions: 478

Average method execution time (normal termination):      0.000347
Average method execution time (exceptional termination): 0.118

Error-revealing test output:
Error-revealing test count: 132
Writing JUnit tests...

Created file /Users/tangjiahui/eclipse-workspace/group8/assignment1/jpacman-framework/ErrorTest0.java
Created file /Users/tangjiahui/eclipse-workspace/group8/assignment1/jpacman-framework/ErrorTest.java

Regression test output:
Regression test count: 1281
Writing JUnit tests...


Created file /Users/tangjiahui/eclipse-workspace/group8/assignment1/jpacman-framework/RegressionTest0.java
Created file /Users/tangjiahui/eclipse-workspace/group8/assignment1/jpacman-framework/RegressionTest1.java
Created file /Users/tangjiahui/eclipse-workspace/group8/assignment1/jpacman-framework/RegressionTest2.java
Created file /Users/tangjiahui/eclipse-workspace/group8/assignment1/jpacman-framework/RegressionTest.java

Invalid tests generated: 0
```

Figure 8: Test case generation process of Randoop.

There is also another function of Randoop called test minimization which helps narrow down the error-revealing test cases by doing some pre-checks. We tried to use it on our results:

java -classpath $RANDOOP_PATH/target/classes/:$RANDOOP_JAR randoop.main.Main minimize –suitepath=ErrorTest0.java –suiteclasspath=$RANDOOP_PATH/target/classes –minimizetimeout=60000

As shown in Figure 9, original error-revealing file length was 3432 lines and the minimized file length became only 915 lines.

```
129/132 tests minimized, Minimized method: test129
130/132 tests minimized, Minimized method: test130
131/132 tests minimized, Minimized method: test131
132/132 tests minimized, Minimized method: test132
Minimizing complete.
Original file length: 3432 lines.
Minimized file length: 915 lines.
```

Figure 9: Test case minization process of Randoop.

Since the test cases files are too larger, so we decided not to put them in the report. However, all the files can be found on our github repo[7].

1.2.2 Code coverage of Randoop

After we obtained the random test cases, we want to integrate it into the jpacman framework and get the code coverage report of them. We copied the test cases as four test classes of the framework: RegressionTest0, RegressionTest1, RegressionTest2 and ErrorTestMinimized. Then we coveraged it as Junit test, as shown in Figure 10, all the regression tests are passed while all the error-revealing tests (53+79=132) are failed.

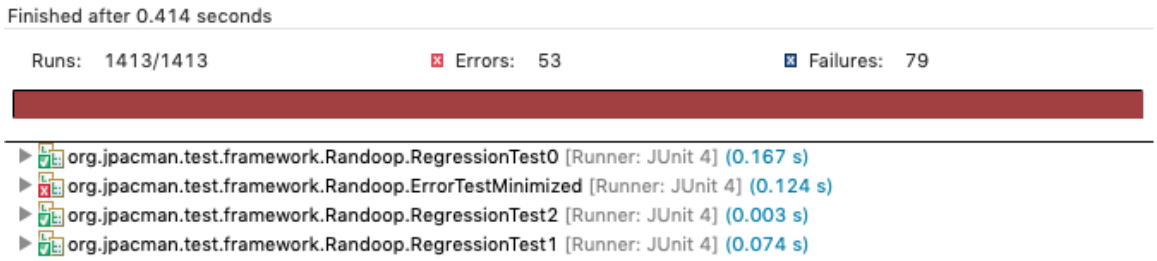


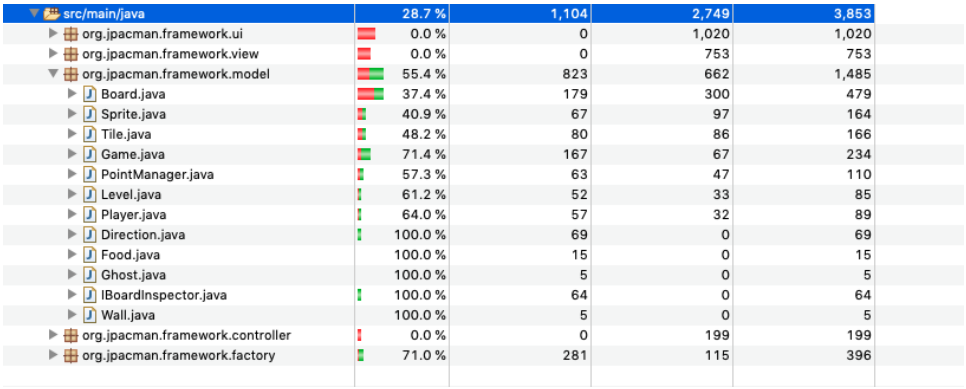
Figure 10: Junit result of Randoop generated test cases.

Figure 11 and Figure 12 indicate the code coverage result of the random generated test cases with and without assertion checking enabled respectively. As we can see here, since our focus was the model classes, the code coverage reached 72.7% with -ea option and only 55.4% without using -ea option.

src/main/java	35.9 %	1,383	2,470	3,853
org.jpacman.framework.ui	0.0 %	0	1,020	1,020
org.jpacman.framework.view	0.0 %	0	753	753
org.jpacman.framework.model	72.7 %	1,080	405	1,485
Board.java	58.5 %	280	199	479
Game.java	76.1 %	178	56	234
Sprite.java	71.3 %	117	47	164
Tile.java	77.7 %	129	37	166
Level.java	69.4 %	59	26	85
PointManager.java	78.2 %	86	24	110
Player.java	82.0 %	73	16	89
Direction.java	100.0 %	69	0	69
Food.java	100.0 %	15	0	15
Ghost.java	100.0 %	5	0	5
IBoardInspector.java	100.0 %	64	0	64
Wall.java	100.0 %	5	0	5
org.jpacman.framework.controller	0.0 %	0	199	199
org.jpacman.framework.factory	76.5 %	303	93	396

Figure 11: Code coverage result of Randoop generated test cases.

[7] https://github.com/UBC-TestingCourse/group8/tree/master/assignment1/jpacman-framework/randoop_output/model



src/main/java	28.7 %	1,104	2,749	3,853
org.jpacman.framework.ui	0.0 %	0	1,020	1,020
org.jpacman.framework.view	0.0 %	0	753	753
org.jpacman.framework.model	55.4 %	823	662	1,485
Board.java	37.4 %	179	300	479
Sprite.java	40.9 %	67	97	164
Tile.java	48.2 %	80	86	166
Game.java	71.4 %	167	67	234
PointManager.java	57.3 %	63	47	110
Level.java	61.2 %	52	33	85
Player.java	64.0 %	57	32	89
Direction.java	100.0 %	69	0	69
Food.java	100.0 %	15	0	15
Ghost.java	100.0 %	5	0	5
IBoardInspector.java	100.0 %	64	0	64
Wall.java	100.0 %	5	0	5
org.jpacman.framework.controller	0.0 %	0	199	199
org.jpacman.framework.factory	71.0 %	281	115	396

Figure 12: Code coverage result of Randoop generated test cases without assertion checking enabled.

Then we looked deeper into those less covered classes, such as Board.java. We found that the test cases generated by Randoop cannot go deep into if statements, as shown in Figure 13. In this case, the other branch of if (s == null) was not covered. We also checked some references, and found that this is a common drawback of Randoop.

```
@Override
public SpriteType spriteTypeAt(int x, int y) {
    assert withinBorders(x, y) : "PRE: " + onBoardMessage(x, y);
    Sprite s = spriteAt(x, y);
    SpriteType result;
    if (s == null) {
        result = SpriteType.EMPTY;
    } else {
        result = s.getSpriteType();
    }
    return result;
}
```

Figure 13: Example: less covered Board.java.

1.2.3 ErrorTest in Randoop

For this section, we want to analyze error-revealing test cases in detail. We summarized the minimized error-revealing test cases into two categories as shown in Figure 14.

ErrorTests				
Category	*Count*	*Class*	*Method*	*Description*
Errors	53	Game	died()	java.lang.NullPointerException
		Game	moveGhost()	java.lang.NullPointerException
Failures	79	Sprite	deoccupy()	java.lang.AssertionError: PRE: Must occupy a cell already.
		Game	movePlayer()	java.lang.AssertionError: Board can't be null when moving.
		Game	moveGhost()	java.lang.AssertionError: PRE1: start tile should not be null.

Figure 14: Minimized Error-revealing test cases generated by Randoop.

Since we enabled assertion checks, so we do not need to care about the failures which are likely because of AssertionErrors. For the errors, we then looked it in details and found that most of them are actually because of potential defects of code. As shown in Figure 15 and Figure 16, died() and moveGhost() methods failed because of NullPointerException.

```
@Test
public void test003() throws Throwable {
    Game game0 = new Game();
    game0.died();
}
```

Figure 15: ErrorTest Example1: died().

```
@Test
public void test004() throws Throwable {
    Game game0 = new Game();
    Ghost ghost2 = null;
    Direction direction6 = null;
    game0.moveGhost(ghost2, direction6);
}
```

Figure 16: ErrorTest Example2: moveGhost().

To solve those potential errors, we changed the source code as depicted in Figure 17 and Figure 18. We added additional checks for getPlayer(), theGhost and dir to make sure that they cannot be null.

```
@Override
public boolean died() {
    return (getPlayer() != null && !getPlayer().isAlive());
}
```

Figure 17: ErrorTest Example1: code changes for died().

```
@Override
public void moveGhost(Ghost theGhost, Direction dir) {
    if(theGhost != null && dir != null) {
        Tile target = theBoard.tileAtDirection(theGhost.getTile(), dir);
        if (tileCanBeOccupied(target)) {
            Sprite currentContent = target.topSprite();
            if (currentContent instanceof Player) {
                ((Player) currentContent).die();
            }
            theGhost.deoccupy();
            theGhost.occupy(target);
            notifyViewers();
        }
    }
}
```

Figure 18: ErrorTest Example2: code changes for moveGhost().

Then we reran Junit test, as we can see from Figure 19, all the errors (53 cases) are solved.

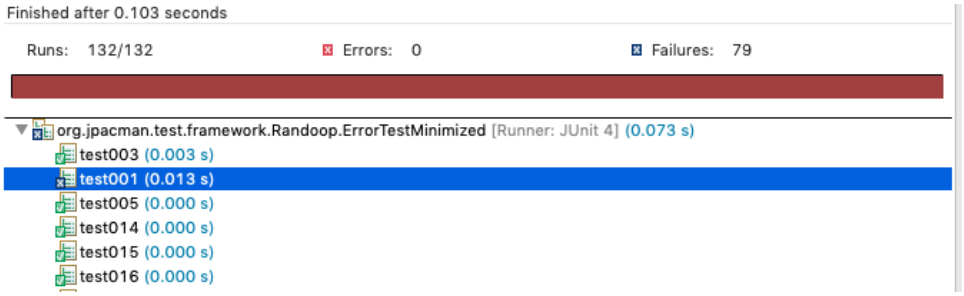


Figure 19: Error-revealing test cases after we changed the code

However, if we re-ran all the test cases generated by Randoop, including RegressionTest, then there will some other failures (79 + 7 =86), as shown in Figure 20. This is because we changed the code and some regression test cases need to be updated (Figure 21).

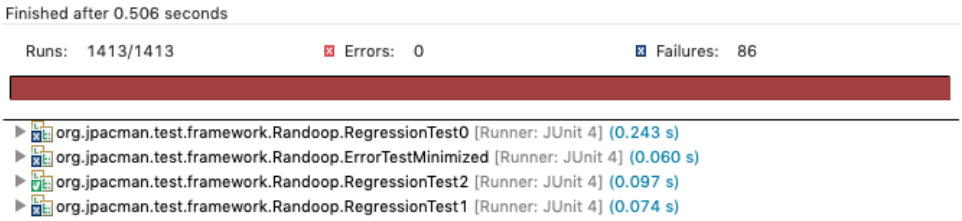


Figure 20: All the random test cases after we changed the code

```
@Test
public void test385() throws Throwable {
    if (debug)
        System.out.format("%n%s%n", "RegressionTest1.test385");
    org.jpacman.framework.model.Game game0 = new org.jpacman.framework.model.Game();
    org.jpacman.framework.model.Board board3 = new org.jpacman.framework.model.Board(62, 10);
    game0.setBoard(board3);
    java.util.Observer observer5 = null;
    game0.deleteObserver(observer5);
    org.jpacman.framework.model.Board board7 = game0.getBoard();
    org.jpacman.framework.model.PointManager pointManager8 = game0.getPointManager();
    try {
        boolean boolean9 = game0.died();
        org.junit.Assert.fail("Expected exception of type java.lang.NullPointerException; message: null");
    } catch (java.lang.NullPointerException e) {
    }
    org.junit.Assert.assertNotNull(board7);
    org.junit.Assert.assertNotNull(pointManager8);
}
```

Figure 21: Regression Test code.

1.2.4 Summary

In summary, Randoop is a useful tool which takes advantage of the feedback-directed aspect to help generate random tests for a given set of Java classes. The user can use it through commandline and also maven plugin. However, we did not find the available maven plugin during our experiment.

Besides, based on our experience, we think that there are still some limitations for this tool: first, as we can see from the regression test results, most of them are more then 10000 lines and each test case contains opaque generated names like game0, game1 and game2, making any deep understanding of the test programs impossible. Second, from the code coverage, we can find that the coverage rate is not that high as our expected (only around 75% for model). This is because Randoop cannot handle well with all the branch of if statement, especially if there is multiple if statements. This could be the future work for the tool developer. Third, Since the regression test cases are too larger, it is of high cost to maintain the test code. Lastly, it is possible that the flaky tests have negative effects on the result, but we are not so sure about this.

Still and all, Randoop is useful for finding bugs in the program also creating regression tests for the program.

2 Static Analysis

For this part, we need to create our own bugchecker using static analysis frameworks. In our case, we chose to use ErrorProne, based on those reasons:

1. It is a state-of-art static analysis tool for JAVA built on top of the javac JAVA compiler.
2. The custom bugchecker is using JAVA language which is the language we are familiar with.
3. Error-Prone keeps updating. A lot of developers make contributions to the bugcheckers.
4. The static analysis is based on JAVA APIs like com.sun.source.tree.*, which are well-documented.

2.1 Working with ErrorProne

After we downloaded the source code of Error-Prone, We first added our custom bug checker directly into ***/core/src/main*** folder and created a test class in ***/core/src/test***.

Our custom bug checker is called ***IfStatementChecker.java***. The test class is ***IfStatementCheckerTest.java***. Our test cases are ***IfStatementPositiveCases.java*** and ***IfStatementNegativeCases.java***.

Then we tried to "maven install" Error-Prone, as shown in Figure 22 and Figure 23, our bug checker passed all the test cases successfully.

```
[INFO] Reactor Summary:
[INFO]
[INFO] Error Prone parent POM ............................. SUCCESS [  1.346 s]
[INFO] @BugPattern annotation ............................. SUCCESS [  1.849 s]
[INFO] error-prone annotations ............................ SUCCESS [  0.371 s]
[INFO] error-prone check api .............................. SUCCESS [  7.785 s]
[INFO] error-prone test helpers ........................... SUCCESS [  8.682 s]
[INFO] error-prone type annotations ....................... SUCCESS [  0.202 s]
[INFO] error-prone library ................................ SUCCESS [08:36 min]
[INFO] JSR-269 annotation processor for @BugPattern annotation SUCCESS [  0.569 s]
[INFO] Documentation tool for generating Error Prone bugpattern documentation SUCCESS [  1.285 s]
[INFO] Refaster rule compiler ............................. SUCCESS [  0.912 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 08:59 min
[INFO] Finished at: 2019-04-08T22:53:14-07:00
[INFO] Final Memory: 76M/733M
[INFO] ------------------------------------------------------------------------
```

Figure 22: Error-Prone build Result.

```
Tests run: 8, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.670 sec - in com.google.errorprone.bugpatterns.HashtableContainsTest
Running com.google.errorprone.bugpatterns.HidingFieldTest
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.395 sec - in com.google.errorprone.bugpatterns.HidingFieldTest
Running com.google.errorprone.bugpatterns.IdentityBinaryExpressionTest
Tests run: 6, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.661 sec - in com.google.errorprone.bugpatterns.IdentityBinaryExpressionTest
Running com.google.errorprone.bugpatterns.IfStatementCheckerTest
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.329 sec - in com.google.errorprone.bugpatterns.IfStatementCheckerTest
Running com.google.errorprone.bugpatterns.ImmutableModificationTest
Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.374 sec - in com.google.errorprone.bugpatterns.ImmutableModificationTest
Running com.google.errorprone.bugpatterns.IncomparableTest
Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.376 sec - in com.google.errorprone.bugpatterns.IncomparableTest
```

Figure 23: Error-Prone build Result - IfStatementCheckerTest all passed

Moreover, we wanted to test our bug checker on some examples, so we also used a plugin to help us to evaulate our bug checker. We copied our code into ***/examples/plugin/maven*** folder of Error-Prone. The result will be shown in Usage Instructions section.

All of our code can be found both in Appendix G to L and github repo[8][9].

2.1.1 MyBugChecker: Pattern And WorkFlow

Here we would like to describe our bug checker: IfStatementChecker. The basic idea is to detect the usage of if(true) and if(false). Even though in real life, those kind of code is not that common, in our assumption, we think it is possible to have those patterns.

Initially, our motivation is to detect a if statement is always true or false, but it turned out we need to "solver" the condition just like constraint solver of symbolic execution, which is hard for us to realize, so we simplified our idea into testing "if(true)" and "if(false)".

Below are our ***basic rules*** for our bug checker:

1. If a if statement is "if(true)", then the "then" statement shall not be null or empty;
2. If a if statement is "if(false)", then the "Else" statement shall not be null or empty;

By empty statement, according to existing definition of EmptyStatement in Error-Prone, it means statement like ***;***.

Hence, based on our idea, we created our own bug checker which is called IfStatmentChecker. Here is the workflow of our bugchecker:

1. If the kind of a statement is ***IF***, which means it is a ***IfTree***, then go step 2, otherwise visit next statement;
2. Get the condition of the IfTree, if the kind of the condition is ***PARENTHESIZED***, then go step 3, otherwise return and visit next statements;
3. Get the expression of the result from step 2, if it is ***BOOLEAN_LITERAL*** type, then go step 4, otherwise return and visit the next statement;
4. Get the value of the result from step 3, if it is ***"true"***, use ***Rule 1*** to detect if it is a match; if it is ***"false"***, use ***Rule 2*** to detect if it is a match.

If there is no match, our bug checker will report nothing. On the other hand, if a match is found, our bug checker will report the corresponding "if" statement.

2.1.2 MyBugChecker: Positive Examples

As we talked before, we provided some examples for our bug checker. For the positive example, which means the incorrect usage of "if(true)" or "if(false)", we provided seven cases:

1. "if true" without a "then" statement;
2. "if true" with an empty "then" statement and an "Else" statement;
3. "if true" with an empty "then" statement and an empty "else" statement;
4. "if true" with an empty "then" statement and an "Else" statement without using brackets;
5. "if false" with no "Then" and "Else" statements;
6. "if false" with a "then" statement and An empty "Else" statement;
7. "if false" with empty "then" and "else" statements.

[8]https://github.com/UBC-TestingCourse/group8/tree/master/MyBugChecker
[9]https://github.com/UBC-TestingCourse/group8/tree/master/MyBugChecker/plugin

2.1.3 MyBugChecker: Negative Examples

Similarly, for negative examples, we provided five cases:

1. "if true" with both "then" and "else" statements;
2. "if true" with a "then" statement;
3. "if true" with a "then" statement and an empty "else";
4. "if false" with both "then" and "else" statements;
5. "if false" with an empty "then" and an "Else" statement.

Figure 24 presents part of the positive and negative examples. The other examples can be found in our source code.

```
//  Negative Example 4:
//"if false" with both "then" and "else" statements
  public static void negativeExample4() {
      if(false) {
          return;
      } else {
          return;
      }
  }

//  Negative Example 5:
//  "if false" with an empty "then" and an "Else" statement
  public static void negativeExample5() {
      if(false);
      else {
          return;
      }
  }

//  Positive Example 1:
//  "if true" without a "then" statement
  public static void positiveExample1() {
      if(true);
  }

//  Positive Example 2:
//  "if true" with an empty "then" statement
//  and an "Else" statement
  public static void positiveExample2() {
      if(true);
       else {
          return;
      }
  }
```

Figure 24: Part of Positive Examples and negative Examples

2.1.4 MyBugChecker: Usage Instructions

To use our bug checker, there are two options:

Since we added our bug checker into the source code of Error-Prone, so the user should be able to install it and generate a new JAR, then use commandline to invoke javac with error prone argument. The official website of Error Prone[10] provides more information about how to run it in commandline.

More easily, the alternative is to use it as a plugin. We already copied our code into the plugin folder and Error Prone has already set all the maven configurations, so we can simply run "maven install" then the code in ***Hello.java*** will be checked using error-prone.

[10]https://errorprone.info/docs/installation

Figure 25: Result of running our bug checker as plugin.

As shown in Figure 25, our bug checker is able to detect all 7 positive cases in the code. For the other 5 negative cases, they are not reported.

We also uploaded MyBug.md, MyAnalyzer.md and README.md into github[11].

Appendix A Task1.1.1: Monkey coverage: FinishListener.java

```
package test;

public interface FinishListener {
void onActivityFinished();
void dumpIntermediateCoverage(String filePath);
}
```

Appendix B Task1.1.1: Monkey coverage: InstrumentedActivity.java

```
package test;
import com.esoxjem.movieguide.listing.MoviesListingActivity;

public class InstrumentedActivity extends MoviesListingActivity {
public FinishListener finishListener ;
public void setFinishListener(FinishListener finishListener){
this.finishListener = finishListener;
}

@Override
public void onDestroy() {
if (this.finishListener !=null){
finishListener.onActivityFinished();
}
super.onDestroy();
}

}
```

[11] https://github.com/UBC-TestingCourse/group8/tree/master/MyBugChecker

Appendix C Task1.1.1: Monkey coverage: JacocoInstrumentation.java

```
package test;

import android.app.Activity;
import android.app.Instrumentation;
import android.content.Intent;
import android.os.Bundle;
import android.os.Looper;
import android.util.Log;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;


public class JacocoInstrumentation extends Instrumentation implements
FinishListener {
public static String TAG = "JacocoInstrumentation:";
private static String DEFAULT_COVERAGE_FILE_PATH =
   "/mnt/sdcard/coverage.ec";
private final Bundle mResults = new Bundle();
private Intent mIntent;
private static final boolean LOGD = true;
private boolean mCoverage = true;
private String mCoverageFilePath;

public JacocoInstrumentation() {

}

@Override
public void onCreate(Bundle arguments) {
Log.d(TAG, "onCreate(" + arguments + ")");
super.onCreate(arguments);
DEFAULT_COVERAGE_FILE_PATH =
   getContext().getFilesDir().getPath().toString() + "/coverage.ec";

File file = new File(DEFAULT_COVERAGE_FILE_PATH);
if (file.isFile() && file.exists()){
if (file.delete()){
System.out.println("file del successs");
}else {
System.out.println("file del fail !");
}
}
if (!file.exists()) {
try {
file.createNewFile();
} catch (IOException e) {
Log.d(TAG, "error : " + e);
e.printStackTrace();
}
}
```

```
if (arguments != null) {
mCoverageFilePath = arguments.getString("coverageFile");
}

mIntent = new Intent(getTargetContext(), InstrumentedActivity.class);
mIntent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
start();
}

@Override
public void onStart() {
System.out.println("onStart def");
if (LOGD)
Log.d(TAG, "onStart()");
super.onStart();

Looper.prepare();
InstrumentedActivity activity = (InstrumentedActivity)
    startActivitySync(mIntent);
activity.setFinishListener(this);
}

private boolean getBooleanArgument(Bundle arguments, String tag) {
String tagString = arguments.getString(tag);
return tagString != null && Boolean.parseBoolean(tagString);
}

private void generateCoverageReport() {
OutputStream out = null;
try {
out = new FileOutputStream(getCoverageFilePath(), false);
Object agent = Class.forName("org.jacoco.agent.rt.RT")
.getMethod("getAgent")
.invoke(null);
out.write((byte[]) agent.getClass().getMethod("getExecutionData",
    boolean.class)
.invoke(agent, false));
} catch (Exception e) {
Log.d(TAG, e.toString(), e);
e.printStackTrace();
} finally {
if (out != null) {
try {
out.close();
} catch (IOException e) {
e.printStackTrace();
}
}
}
}

private String getCoverageFilePath() {
if (mCoverageFilePath == null) {
return DEFAULT_COVERAGE_FILE_PATH;
} else {
return mCoverageFilePath;
```

```
}
}

private boolean setCoverageFilePath(String filePath){
if(filePath != null && filePath.length() > 0) {
mCoverageFilePath = filePath;
return true;
}
return false;
}

private void reportEmmaError(Exception e) {
reportEmmaError("", e);
}

private void reportEmmaError(String hint, Exception e) {
String msg = "Failed to generate emma coverage. " + hint;
Log.e(TAG, msg, e);
mResults.putString(Instrumentation.REPORT_KEY_STREAMRESULT, "\nError: "
+ msg);
}

@Override
public void onActivityFinished() {
if (LOGD)
Log.d(TAG, "onActivityFinished()");
if (mCoverage) {
System.out.println("onActivityFinished mCoverage true");
generateCoverageReport();
}
finish(Activity.RESULT_OK, mResults);
}

@Override
public void dumpIntermediateCoverage(String filePath){
// TODO Auto-generated method stub
if(LOGD){
Log.d(TAG,"Intermidate Dump Called with file name :"+ filePath);
}
if(mCoverage){
if(!setCoverageFilePath(filePath)){
if(LOGD){
Log.d(TAG,"Unable to set the given file path:"+filePath+" as dump
    target.");
}
}
generateCoverageReport();
setCoverageFilePath(DEFAULT_COVERAGE_FILE_PATH);
}
}
}
```

Appendix D Task1.1.1: Monkey coverage: build.gradle (app)

```
apply plugin: 'jacoco'
```

```
jacoco {
toolVersion = "0.7.4+"
}

android {
buildTypes {
debug {
testCoverageEnabled = true
}
release {
minifyEnabled false
proguardFiles getDefaultProguardFile('proguard-android.txt'),
    'proguard-rules.pro'
}
}
}

def coverageSourceDirs = [
'../app/src/main/java'
]
task jacocoTestReport(type: JacocoReport) {
group = "Reporting"
description = "Generate Jacoco coverage reports after running tests."
reports {
xml.enabled = true
html.enabled = true
}
classDirectories = fileTree(
dir: './build/intermediates/classes/debug',
excludes: ['**/R*.class',
'**/*$InjectAdapter.class',
'**/*$ModuleAdapter.class',
'**/*$ViewInjector*.class'
])
sourceDirectories = files(coverageSourceDirs)
executionData =
    files("$buildDir/outputs/code-coverage/connected/coverage.ec")

doFirst {
new File("$buildDir/intermediates/classes/").eachFileRecurse { file ->
if (file.name.contains('$$')) {
file.renameTo(file.path.replace('$$', '$'))
}
}
}
}

def getProperty(String filename, String propName) {
def propsFile = rootProject.file(filename)
if (propsFile.exists()) {
def props = new Properties()
props.load(new FileInputStream(propsFile))
if (props[propName] != null) {
return props[propName]
} else {
print("No such property " + propName + " in file " + filename);
```

```
}
} else {
print(filename + " does not exist!")
}
}
```

Appendix E Task1.1.1: Monkey coverage: AndroidManifest.xml

```
 <?xml version="1.0" encoding="utf-8"?>
<manifest
package="com.esoxjem.movieguide"
xmlns:android="http://schemas.android.com/apk/res/android">

<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.USE_CREDENTIALS" />
<uses-permission android:name="android.permission.GET_ACCOUNTS" />
<uses-permission android:name="android.permission.READ_PROFILE" />
<uses-permission android:name="android.permission.READ_CONTACTS" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>

<application
android:name=".BaseApplication"
android:allowBackup="true"
android:icon="@mipmap/ic_launcher"
android:label="@string/app_name"
android:theme="@style/AppTheme">
<activity
android:name=".listing.MoviesListingActivity"
android:launchMode="singleTask">
<intent-filter>
<action android:name="android.intent.action.MAIN"/>

<category android:name="android.intent.category.LAUNCHER"/>
</intent-filter>
</activity>
<activity
android:name=".details.MovieDetailsActivity"
android:parentActivityName=".listing.MoviesListingActivity"
android:theme="@style/AppTheme">
<meta-data
android:name="android.support.PARENT_ACTIVITY"
android:value=".listing.MoviesListingActivity"/>
</activity>
<activity android:label="InstrumentationActivity"
    android:name="test.InstrumentedActivity" />
</application>

<instrumentation
android:handleProfiling="true"
android:label="CoverageInstrumentation"
android:name="test.JacocoInstrumentation"
android:targetPackage="com.esoxjem.movieguide"/>

</manifest>
```

Appendix F Task1.1.2: Monkey Bug: runmonkey.py

```
from subprocess import call, PIPE, Popen, check_output
import time, os, sys
import glob
import shutil

monkeyclickcount = 1000

WORKSPACE = os.path.abspath(".")

def getWorkConfig():
    f = open("./.config", "r")
    config = {"monkeyclickcount": monkeyclickcount}
    try:
        while True:
            line = f.readline()
            if line:
                line = line.strip()
                linesplit = line.split("ï Ž")
                if linesplit.__sizeof__() > 1:
                    if linesplit[0] == 'phone':
                        config['phone'] = linesplit[1]
                    elif linesplit[0] == 'monkeyclickcount':
                        config["monkeyclickcount"] = linesplit[1]
            else:
                break
    finally:
        f.close()
        print("config : %s" % config)
        return config

def killTestApp():
    call(['adb', '-s', workConfig.get('phone'), 'shell', 'am',
       'force-stop', 'com.esoxjem.movieguide'])

def fullmonkey(workconfig):
    killTestApp()
    options = "-p com.esoxjem.movieguide " \
              "-s 500 --ignore-crashes --ignore-timeouts
                 --monitor-native-crashes " \
              "-v -v -v"
    monkey_output = 'monkey_log.txt'
    with open(monkey_output, 'wb') as f:
        call(['adb', '-s', workconfig.get("phone"), 'shell', 'monkey',
           options, workconfig.get("monkeyclickcount")], stdout=f,
           stderr=f)


workConfig = getWorkConfig()

print("execute monkey")
fullmonkey(workConfig)

print("create bugreport file")
```

```
bugreport = 'bugreport.txt'
with open(bugreport, 'wb') as f:
    call(['adb', 'bugreport'], stdout=f, stderr=f)

print("create bugreport file, done")

zip_file = glob.glob('./bugreport*.zip')

call(['java', '-jar', 'chkbugreport-0.4-185.jar', zip_file[0]])

for file in zip_file:
    os.remove(file)

print("Completion of the current round of testing")
input("Enter key to close")
```

Appendix G Task 2: Error Prone: IfStatementChecker.java

```
/*
 * Group 8's custom bugchecker for CPEN522
 */

package com.google.errorprone.bugpatterns;

import static com.google.errorprone.matchers.Matchers.nextStatement;
import static com.google.errorprone.matchers.Matchers.parentNode;
import static com.sun.source.tree.Tree.Kind.IF;
import static com.sun.source.tree.Tree.Kind.PARENTHESIZED;
import static com.sun.source.tree.Tree.Kind.BOOLEAN_LITERAL;


import static com.google.errorprone.BugPattern.SeverityLevel.ERROR;

import com.google.auto.service.AutoService;
import com.google.errorprone.BugPattern;
import com.google.errorprone.BugPattern.ProvidesFix;
import com.google.errorprone.VisitorState;
import
   com.google.errorprone.bugpatterns.BugChecker.EmptyStatementTreeMatcher;
import com.google.errorprone.fixes.SuggestedFix;
import com.google.errorprone.matchers.Description;
import com.google.errorprone.matchers.Matchers;
import com.sun.source.tree.EmptyStatementTree;
import com.sun.source.tree.IfTree;
import com.sun.source.tree.ParenthesizedTree;
import com.sun.source.tree.LiteralTree;
import com.sun.source.tree.StatementTree;
import com.sun.source.tree.Tree;
import com.google.errorprone.bugpatterns.BugChecker;

@BugPattern(
            name = "IfStatementChecker",
            altNames = {"ifstatement"},
            summary = "If statement is always true or false",
```

```
            severity = ERROR)

public class IfStatementChecker extends BugChecker implements
    EmptyStatementTreeMatcher {
  /**
   * This is a custom bugchecker wrote for CPEN522
   * The basic idea is to detect the usage of if(true)
   * and if(false).
   *
   * If the if statement is "if(true)", then
   * the "then statement" should not be null or empty;
   *
   * if the if statement is "if(false)", then
   * the "Else statement" should not be null or empty;
   *
   */
  @Override
  public Description matchEmptyStatement(EmptyStatementTree tree,
      VisitorState state) {
    boolean matches = false;
    Tree parent = state.getPath().getParentPath().getLeaf();
    if (parent.getKind() == IF) {
      IfTree parentAsIf = (IfTree) parent;
      if (parentAsIf.getCondition().getKind() == PARENTHESIZED){
                  ParenthesizedTree test = (ParenthesizedTree)
                      (parentAsIf.getCondition());
                  if(test.getExpression().getKind() == BOOLEAN_LITERAL) {
                        LiteralTree test_boolean = (LiteralTree)
                           (test.getExpression());
                        if(test_boolean.getValue().toString() == "true") {
                                matches =
                                                ((parentAsIf.getElseStatement()
                                                   != null) &&
                                                   !(parentAsIf.getElseStatement()
                                                   instanceof
                                                   EmptyStatementTree)) ||
//
    ((parentAsIf.getThenStatement() == null) ||
                                                                (parentAsIf.getThenStatement()
                                                                   instanceof
                                                                   EmptyStatementTree);
                        } else if (test_boolean.getValue().toString() ==
                            "false") {
                                matches =
//
    ((parentAsIf.getThenStatement() != null) ||
    !(parentAsIf.getThenStatement() instanceof EmptyStatementTree)) ||
                                                (parentAsIf.getElseStatement()
                                                   == null) ||
                                                   (parentAsIf.getElseStatement()
                                                   instanceof
                                                   EmptyStatementTree);
                        }
                  }
      }
     }
```

```
    if (!matches) {
      return Description.NO_MATCH;
    }
    assert (state.getPath().getParentPath().getLeaf().getKind() == IF);
    IfTree ifParent = (IfTree) state.getPath().getParentPath().getLeaf();
    return describeMatch(ifParent);
  }
}
```

Appendix H Task 2: Error Prone: IfStatementCheckerTest.java

```
/*
 * CPEN522 - group 8
 */

package com.google.errorprone.bugpatterns;

import com.google.errorprone.CompilationTestHelper;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.junit.runners.JUnit4;

@RunWith(JUnit4.class)
public class IfStatementCheckerTest {
  CompilationTestHelper compilationHelper;

  // since we did not provide a SuggestedFixes, so it is expected to be
     errors
  @Test (expected = AssertionError.class)
  public void testIfStatementPositiveCases() {
        compilationHelper =
           CompilationTestHelper.newInstance(IfStatementChecker.class,
           getClass());
    compilationHelper.addSourceFile("IfStatementPositiveCases.java").doTest();
  }

  @Test
  public void testIfStatementNegativeCases() {
        compilationHelper =
           CompilationTestHelper.newInstance(IfStatementChecker.class,
           getClass());
    compilationHelper.addSourceFile("IfStatementNegativeCases.java").doTest();
  }
}
```

Appendix I Task 2: Error Prone: IfStatementPositiveCases.java

```
/*
 * CPEN522 - group 8
 */

package com.google.errorprone.bugpatterns.testdata;

import com.google.common.base.Objects;
```

```
public class IfStatementPositiveCases {
//  Positive Example 1:
//  "if true" without a "then" statement
  public static void positiveExample1() {
          if(true);
  }

//  Positive Example 2:
//  "if true" with an empty "then" statement
//  and an "Else" statement
  public static void positiveExample2() {
          if(true);
           else {
                  return;
          }
  }

//  Positive Example 3:
//  "if true" with an empty "then" statement
//  and an empty "else" statement
  public static void positiveExample3() {
          if(true);
           else ;
  }

//  Positive Example 4:
//  "if true" with an empty "then" statement
//  and an "Else" statement without using brackets
  public static void positiveExample4() {
          if(true) ;
          else return;
  }

//  Positive Example 5:
//  "if false" with no "Then" and "Else" statements
  public static void positiveExample5() {
          if(false);
  }

//  Positive Example 6:
//  "if false" with a "then" statement and
//  An empty "Else" statement
  public static void positiveExample6() {
          if(false) {
                  return;
          } else ;
  }

//  Positive Example 7:
//  "if false" with empty "then" and "else" statments
  public static void positiveExample7() {
          if(false) ;
          else ;
  }
}
```

Appendix J Task 2: Error Prone: IfStatementNegativeCases.java

```
/*
 * CPEN 522 - group 8
 */

package com.google.errorprone.bugpatterns.testdata;

import com.google.common.base.Objects;

public class IfStatementNegativeCases {
//  Negative Example 1:
//  "if true" with both "then" and "else" statments
  public static void negativeExample1() {
          if(true) {
                  return;
          } else {
                  return;
          }
  }

//  Negative Example 2:
//  "if true" with a "then" statement
  public static void negativeExample2() {
          if(true) {
                  return;
          }
  }

//  Negative Example 3:
//  "if true" with a "then" statement and an empty "else"
  public static void negativeExample3() {
          if(true) {
                  return;
          } else ;
  }

//  Negative Example 4:
//  "if false" with both "then" and "else" statements
  public static void negativeExample4() {
          if(false) {
                  return;
          } else {
                  return;
          }
  }

//  Negative Example 5:
//  "if false" with an empty "then" and an "Else" statement
  public static void negativeExample5() {
          if(false);
          else {
                  return;
          }
  }
```

```
}
```

Appendix K Task 2: Error Prone: MyCustomCheck.java in plugin folder

```
/*
 * Group 8's custom bugchecker for CPEN522
 */

package com.google.errorprone.sample;

import static com.google.errorprone.matchers.Matchers.nextStatement;
import static com.google.errorprone.matchers.Matchers.parentNode;
import static com.sun.source.tree.Tree.Kind.IF;
import static com.sun.source.tree.Tree.Kind.PARENTHESIZED;
import static com.sun.source.tree.Tree.Kind.BOOLEAN_LITERAL;


import static com.google.errorprone.BugPattern.SeverityLevel.ERROR;

import com.google.auto.service.AutoService;
import com.google.errorprone.BugPattern;
import com.google.errorprone.BugPattern.ProvidesFix;
import com.google.errorprone.VisitorState;
import
    com.google.errorprone.bugpatterns.BugChecker.EmptyStatementTreeMatcher;
import com.google.errorprone.fixes.SuggestedFix;
import com.google.errorprone.matchers.Description;
import com.google.errorprone.matchers.Matchers;
import com.sun.source.tree.EmptyStatementTree;
import com.sun.source.tree.IfTree;
import com.sun.source.tree.ParenthesizedTree;
import com.sun.source.tree.LiteralTree;
import com.sun.source.tree.StatementTree;
import com.sun.source.tree.Tree;
import com.google.errorprone.bugpatterns.BugChecker;

@AutoService(BugChecker.class)
@BugPattern(
            name = "IfStatementChecker",
            altNames = {"ifstatement"},
            summary = "If statement is always true or false",
            severity = ERROR)

public class MyCustomCheck extends BugChecker implements
    EmptyStatementTreeMatcher {
  /**
   * This is a custom bugchecker wrote for CPEN522
   * The basic idea is to detect the usage of if(true)
   * and if(false).
   *
   * If the if statement is "if(true)", then
   * the "then statement" should not be null or empty;
   *
   * if the if statement is "if(false)", then
```

```
   * the "Else statement" should not be null or empty;
   *
   */
  @Override
  public Description matchEmptyStatement(EmptyStatementTree tree,
      VisitorState state) {
    boolean matches = false;
    Tree parent = state.getPath().getParentPath().getLeaf();
    if (parent.getKind() == IF) {
      IfTree parentAsIf = (IfTree) parent;
      if (parentAsIf.getCondition().getKind() == PARENTHESIZED){
                  ParenthesizedTree test = (ParenthesizedTree)
                      (parentAsIf.getCondition());
                  if(test.getExpression().getKind() == BOOLEAN_LITERAL) {
                        LiteralTree test_boolean = (LiteralTree)
                           (test.getExpression());
                        if(test_boolean.getValue().toString() == "true") {
                              matches =
                                                    ((parentAsIf.getElseStatement()
                                                       != null) &&
                                                       !(parentAsIf.getElseStatement()
                                                       instanceof
                                                       EmptyStatementTree)) ||
//
    ((parentAsIf.getThenStatement() == null) ||
                                                                  (parentAsIf.getThenStatement()
                                                                     instanceof
                                                                     EmptyStatementTree);
                        } else if (test_boolean.getValue().toString() ==
                            "false") {
                                  matches =
//
    ((parentAsIf.getThenStatement() != null) ||
    !(parentAsIf.getThenStatement() instanceof EmptyStatementTree)) ||
                                                 (parentAsIf.getElseStatement()
                                                    == null) ||
                                                    (parentAsIf.getElseStatement()
                                                    instanceof
                                                    EmptyStatementTree);
                        }
                  }
      }
    }
    if (!matches) {
      return Description.NO_MATCH;
    }
    assert (state.getPath().getParentPath().getLeaf().getKind() == IF);
    IfTree ifParent = (IfTree) state.getPath().getParentPath().getLeaf();
    return describeMatch(ifParent);
  }
}
```

Appendix L Task 2: Error Prone: Hello.java in plugin folder

```
/*
```

```java
 * Negative and Positive Examples for group 8's custom bugchecker
 */

package com.google.errorprone.sample;

import java.util.Arrays;

public class Hello {
  public static void main() {
          // run all negative examples which are the correct
          // usage of if(true) or if(false)
          negativeExample1();
          negativeExample2();
          negativeExample3();
          negativeExample4();
          negativeExample5();

          // run all positive examples which are the incorrect
          // usage of if(true) or if(false)
          positiveExample1();
          positiveExample2();
          positiveExample3();
          positiveExample4();
          positiveExample5();
          positiveExample6();
          positiveExample7();
  }

//  Negative Example 1:
//  "if true" with both "then" and "else" statments
  public static void negativeExample1() {
          if(true) {
                  return;
          } else {
                  return;
          }
  }

//  Negative Example 2:
//  "if true" with a "then" statement
  public static void negativeExample2() {
          if(true) {
                  return;
          }
  }

//  Negative Example 3:
//  "if true" with a "then" statement and an empty "else"
  public static void negativeExample3() {
          if(true) {
                  return;
          } else ;
  }

//  Negative Example 4:
//  "if false" with both "then" and "else" statements
```

```
  public static void negativeExample4() {
          if(false) {
                  return;
          } else {
                  return;
          }
  }

//  Negative Example 5:
//  "if false" with an empty "then" and an "Else" statement
  public static void negativeExample5() {
          if(false);
          else {
                  return;
          }
  }

//  Positive Example 1:
//  "if true" without a "then" statement
  public static void positiveExample1() {
          if(true);
  }

//  Positive Example 2:
//  "if true" with an empty "then" statement
//  and an "Else" statement
  public static void positiveExample2() {
          if(true);
           else {
                  return;
          }
  }

//  Positive Example 3:
//  "if true" with an empty "then" statement
//  and an empty "else" statement
  public static void positiveExample3() {
          if(true);
           else ;
  }

//  Positive Example 4:
//  "if true" with an empty "then" statement
//  and an "Else" statement without using brackets
  public static void positiveExample4() {
          if(true) ;
          else return;
  }

//  Positive Example 5:
//  "if false" with no "Then" and "Else" statements
  public static void positiveExample5() {
          if(false);
  }

//  Positive Example 6:
```

```
//  "if false" with a "then" statement and
//  An empty "Else" statement
  public static void positiveExample6() {
          if(false) {
                  return;
          } else ;
  }

//  Positive Example 7:
//  "if false" with empty "then" and "else" statments
  public static void positiveExample7() {
          if(false) ;
          else ;
  }
}
```