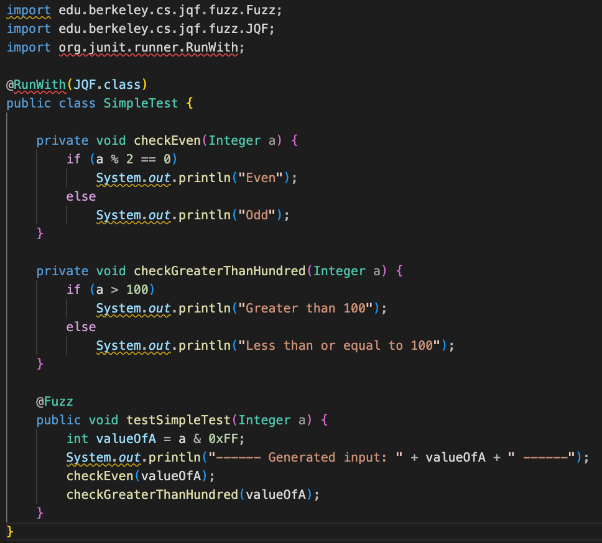
**2 Implementation Documentation**

2.1 Detailed, step-by-step description of your implementation

Step 1. Download and set up JQF.

Substep 1.1. Command: git clone https://github.com/rohanpadhye/JQF.gitSubstep 1.2. Check install dependencies: JDK: run ‘java -version’ and ‘javac -version’Maven: run ‘mvn -v’

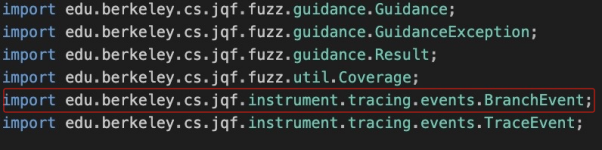
Substep 1.3. Create PUT(vim SimpleTest.java),press “i", and insert the code as in figure 1.



**Figure 1.**

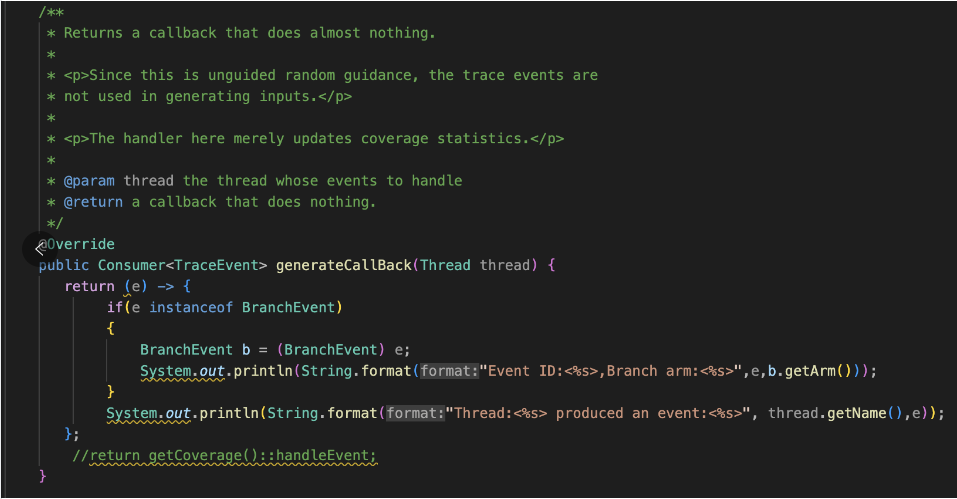
Step 2. Fuzz the PUT using JQF. Substep 2.1: Compile the PUT (SimpleTest.java) with essential dependencies: javac -cp.Command: javac -cp .:$(./scripts/classpath.sh) SimpleTest.javaSubstep 2.2: Run jqf-random on the PUT bytecode for 10 iterations.Command: ./bin/jqf-random SimpleTest testSimpleTest 10

Step 3. Print out coveragesSubstep 3.1 Modify JQF source code to print branch decisions.”import from” line added as in figure 2.: import edu.berkeley.cs.jqf.instrument.tracing.events.BranchEvent;



**Figure 2.**

Source Code modified as in figure 3. Code path: JQF/fuzz/src/main/java/edu/berkeley/cs/jqf/fuzz/random/NoGuidance.java



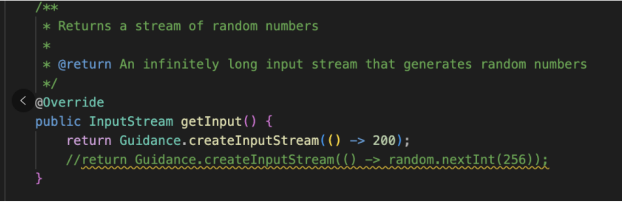
**Figure 3.**

Substep 3.2 Build JQF again.

Command: javac -cp .:$(./scripts/classpath.sh) SimpleTest.java

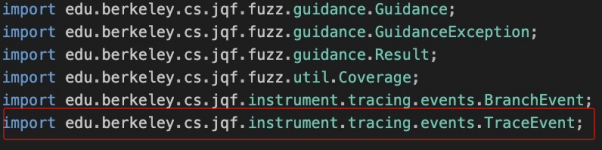
Substep 3.3: Run jqf-random on the PUT bytecode for 10 iterations.Command: ./bin/jqf-random SimpleTest testSimpleTest 10

Step 4. Change test inputs and print coverages.Substep 4.1 Modify JQF source code to generate integer 200 for every iteration.Source Code modified as in figure 4. Code path: Source Code: JQF/SimpleTest.java



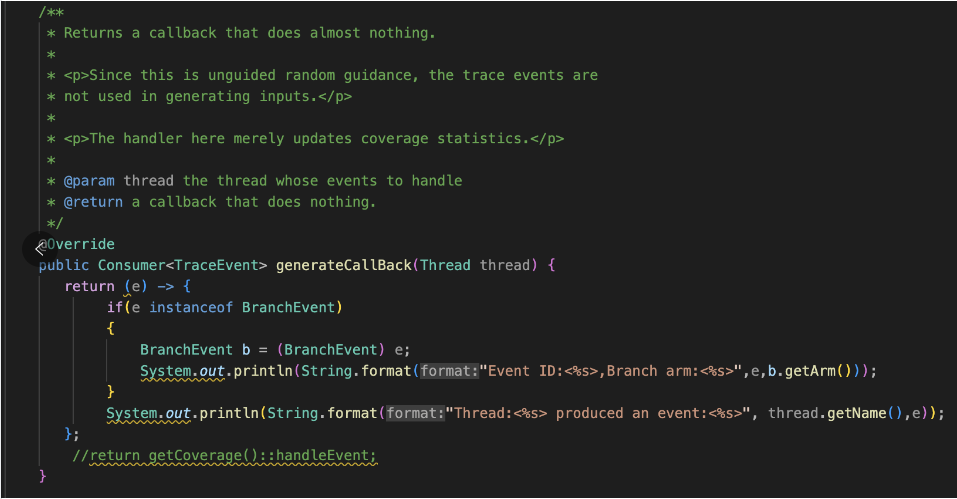
**Figure 4.**

Substep 4.2 Modify JQF source code to print trace events.”import from” line added as in figure 5: import edu.berkeley.cs.jqf.instrument.tracing.events.TraceEvent;



**Figure 5.**

Source Code modified as in figure 6. Code path: JQF/fuzz/src/main/java/edu/berkeley/cs/jqf/fuzz/random/NoGuidance.java



**Figure 6.**

Substep 4.3 Build JQF again.

Command: javac -cp .:$(./scripts/classpath.sh) SimpleTest.java

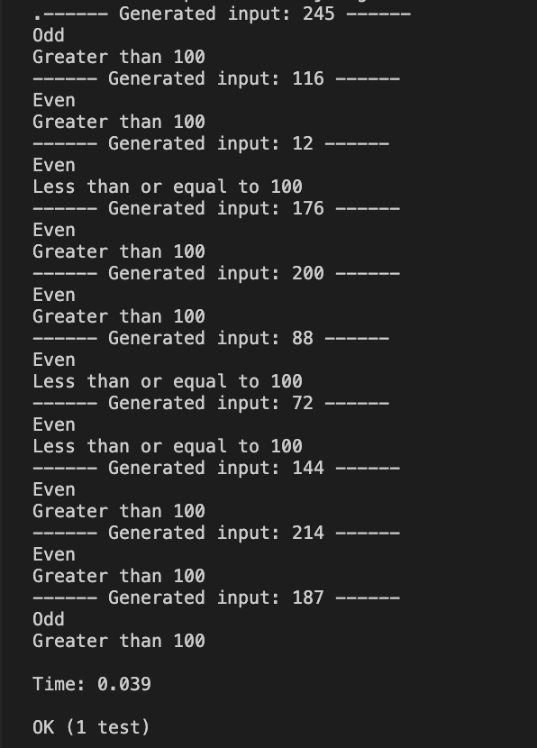
Substep 4.4 Run jqf-random on the PUT bytecode for 10 iterations.Command: ./bin/jqf-random SimpleTest testSimpleTest 10

**3 Result Documentation**

3.1 Describe your obtained results, e.g. JQF outputs, comparisons of different prompts

Step 2 Fuzz the PUT using JQF.

Substep 2.2 in project description: Run jqf-random on the PUT bytecode for 10 iterations as in figure 7.

.

**Figure 7.**

From the figure, we have:

①The generated input with info of odd/even and comparison with 100.

②Execution time.

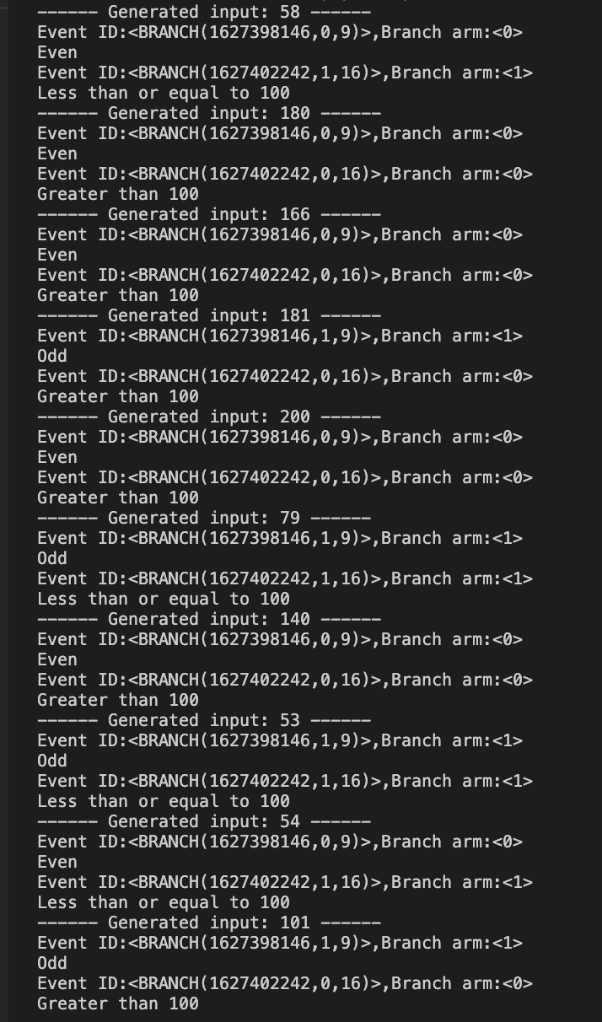
③Program process status.

Step 3: Run jqf-random on the PUT bytecode for 10 iterations and print branch decisions.In this step, Substep 3.1 doesn’t produce any outcome. To better show the result of the modified code, we put the related result we get from Substep 3.3.

Substep 3.1 Modify the source code of JQF to print out the branch decision for every event in

the format of Event ID: <event-id>, branch arm: <branch-arm> as in figure 8.

 **Figure 8.**

Substep 3.3 Run jqf-random on the PUT bytecode for 10 iterations and print out the branch decisions as in figure 9 and figure 10 . 

**Figure 9.**

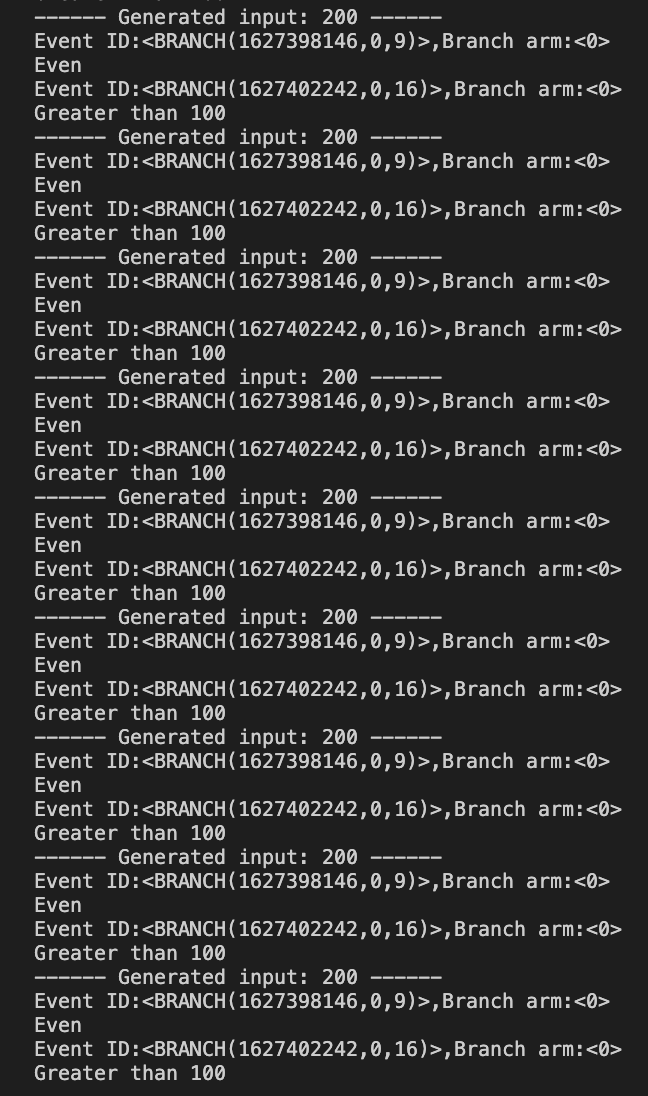


**Figure 10.**

Step 4: Run jqf-random on the PUT bytecode for 10 iterations and print branch decisions.

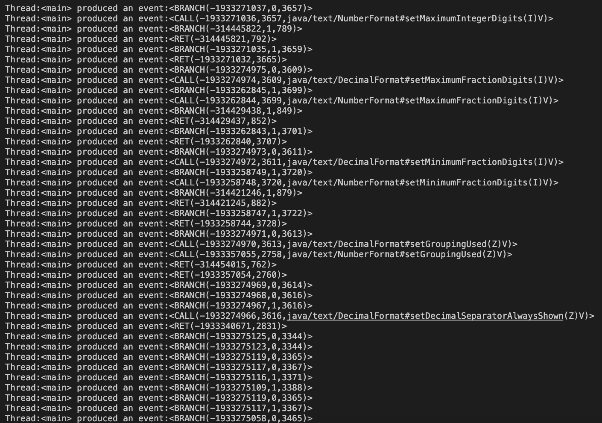
In this step, Substep 4.1 and Substep 4.2 don’t produce any outcome. To better show the result of the modified code, we put the related result we get from Substep 4.4.

Substep 4.1 Modify JQF source code to generate integer 200 for every iteration as in figure 11.



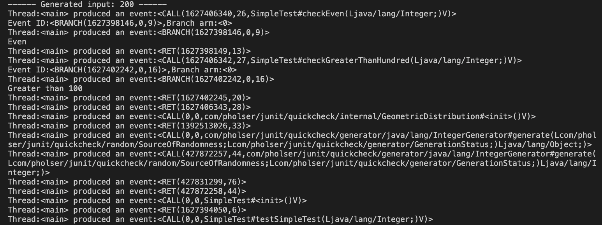
**Figure 11.**

Substep 4.2 Modify JQF source code to print trace events as in figure 12.

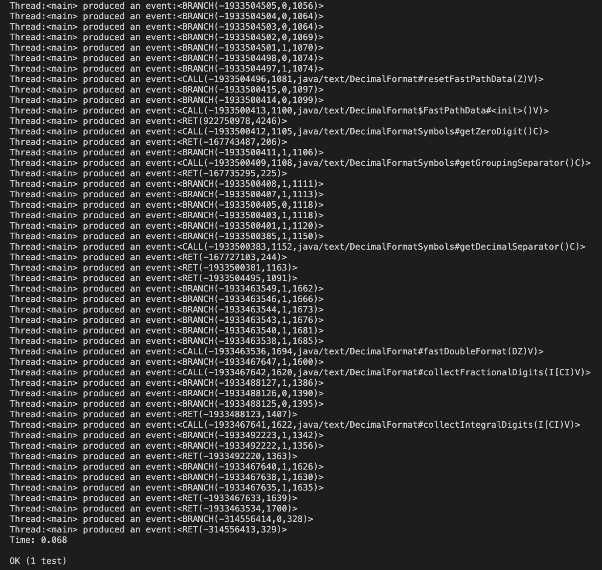


**Figure 12.**

Substep 4.4 Run jqf-random on the PUT bytecode for 10 iterations and print out the branc decisions as in figure 13 and figure 14.



**Figure 13.**



**Figure 14.**

**5 Describe potential optimizations your team can envision**

Potential optimizations for the described project option involving JQF and classical software testing could include:1)Parallel Fuzzing:①Explore parallel fuzzing strategies to improve test coverage.②Utilize multiple instances of JQF running in parallel to speed up the fuzzing process.2)Customized Guidance Strategies:①Experiment with different guidance strategies provided by JQF.②Develop or customize guidance strategies tailored to the characteristics of the program under test for more effective coverage.