Ashlyn Goh 1002840 Cl02

50.003 Problem Set 7

Cohort Exercise 1:
See MonkeyTestISTD.java

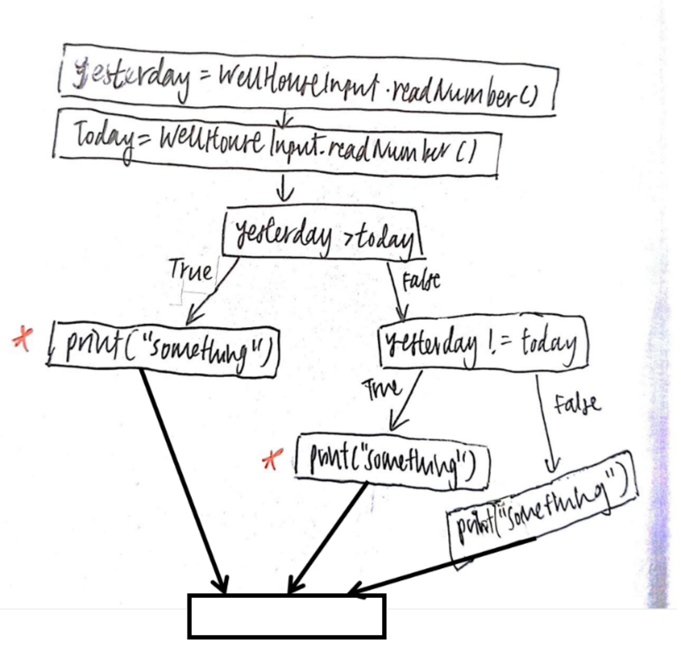
Cohort Exercise 2:
See LoginBotWithInvalidValidUser.java

Cohort Exercise 3:
See HeaderNameFinder.java

Cohort Exercise 4:
See calculator-grammar-fuzzing.py

(Although not stated in the rules in the slides, if a factor is in the form -Integer, I wrapped it brackets. For example, (-4). This is so that the output resembles conventional input to a calculator)

Cohort Exercise 5:
The points of insertions of instrumental code are represented with an asterisk (*).



Cohort Exercise 6:
See FitnessCalc.java

The fitness function is modified as shown in the .java file. The main difference is in the getFitness(Individual individual) method where we compare the genes of the individual based on their positions. (Compare the *ith* gene with *size-ith* gene)

The selection/crossover/mutation operator were not modified.

Ashlyn Goh 1002840 Cl02

Cohort Exercise 7:
See Easier.java and Harder.java

The results are as follows:

Class Name	Example.java	Harder.java	Easier.java
Conditional Branch Coverage	93%	40%	100%
Average Test Suite Coverage	98%	71%	100%
Time Taken (seconds)	38.537	41.763	13.363

As seen above, the Harder class took a longer time to test while the Easier class took a much shorter time to test. In the Harder class, there were a lot of "==" conditions which made it harder to test because the probability of coming up with a test that satisfies these strict conditions is lower. On the other hand, in the easier class, the conditions were mostly "<" and passing some conditions will automatically allow you to pass the subsequent conditions. The conditions were not strict and thus the probability of coming up with a test that satisfies the conditions were much higher, leading to a much shorter time taken. In the Example class, the conditions were mostly ">" then "<", meaning that the inputs `x, y, z` have to be in a range in order to satisfy the conditions. Hence, these conditions were stricter than the Easier class but more relaxed than the Harder class. Unsurprisingly, the time taken to test the Example class falls in between the Harder and Easier classes.

Homework:
See generalised_fuzzer.py

Cohort Exercise 8:
See BrokenLinkFinderNoSmell.java.

Cohort Exercise 9:
See AccountNoSmell.java

Cohort Exercise 10:
See ShootTheAccount.java for the first part.
See ShootTheAccountPlus.java for the second part.

Cohort Exercise 11:
No, if we put in non-ASCII characters in between `<script>`, it would pass the pattern check, but after that when we replace all non-ASCII charcters to "", we would get back the `<script>`.

For example, we can pass in the String s like the following:
String s = "<scri¥pt>";
The fixed code is in XSSFixed.java.

Ashlyn Goh   1002840   Cl02

Cohort Exercise 12:
See Exercise4Fixed.java

Observed problem:
`System.out.println(cal2.after(cal1));` // should return true but it returns false

Documentation of Calendar class's after() method:

```
public boolean after(Object when) {
  return when instanceof Calendar && compareTo((Calendar) when) > 0;
}
```

In the original code, the two calendars are compared using the subclass's `after()` method which invokes the superclass's `after()` method after the `if` condition (at the else statement). However, the superclass's `after()` method would call the `compareTo()` method which is delegated to the subclass's `compareTo()` method. As a result, the subclass's `after()` method would call it's overridden `compareTo()` method. The overridden `compareTo()` method would return 0 and thus the superclass's `after()` method would return false when we expect it to return true.

In the Exercise4Fixed.java, we have a forwarder class (`ForwardingCalendar`) and its methods redirects to methods of `CalendarImplementation` class, which is a class that extends Calendar. The `CompositeCalendar` is a wrapper class that provides the same overridden methods found in the `CalendarSubclass`.

When we call the overriden `after()` method in `CompositeCalendar`, we use the `CalendarImplementation` class's `compareTo()` method. Using `super.after(when)` forwards to `ForwardingCalendar`, which invokes `CalendarImplementation`'s `after()`. Hence, `java.util.Calendar.after()` invokes `CalendarImplementation.compareTo()` method. As a result, we won't get the same problem encountered in Exercise4.java.

Homework:
Extract the Klee.zip.

See loopfreeprogram.c for the loop-free C program. The computed models by Z3 are in the same folder named "model_00*" while the SMT2 files are named "test00000*.smt2.