CITS5501 Software Testing and Quality Assurance
Semester 1, 2020
Week 7 Workshop – Property-based testing

Reading

It is strongly suggested you complete the recommended readings for weeks 1-6 *before* attempting this lab/workshop.

Property-based testing

Many of the tests we've seen in lectures (and see in code in industry) rely on developers or testers coming up with *example*-based tests. That is, for a software component to be tests, we come up with some set of test values, and some expected results, and check that the actual results are the same as the expected results.

Problems with example-based testing of this sort include:

- Even when we apply techniques such as Input Space Partitioning, we still may miss some edge cases.
- Our tests may not very thoroughly cover the domain of the component.

Property-based testing, on the other hand, will typically generate hundreds of tests and run them quickly. It can often reveal edge cases that other tests have missed. Property-based testing checks that some function, method or component under test abides by a property.

It is suggested you take a quick look at the explanation of property-based testing given for the Hypothesis testing library from Python, here: https://hypothesis.works/articles/intro/.

We will use a property-based testing library for Java called QuickTheories. The API for the library is available here: https://javadoc.io/doc/org.quicktheories/quicktheories/latest/index.html

You will need to add the `.jar` file for QuickTheories to your BlueJ project – after creating a project, go to "Preferences" / "Libraries" / "User Libraries from config" / "Add File", and add the `quicktheories-0.26.jar` file to your BlueJ project.

Property-based testing libraries are much more common in languages such as JavaScript, Python, Clojure and Haskell than they are in Java; one author of such a library discusses why this might be so here: https://blog.johanneslink.net/2020/01/28/why-is-pbt-not-popular-in-java/.

One technique we will look at later is *fuzzing*, which can be seen as a sort of property-based testing in which the invariant you test is that "The program should not crash". Read the following brief description of fuzzing: https://hypothesis.works/articles/getting-started-with-hypothesis/.

Test cases

Look at the tests in the files called `PropTest01Ints.java`, `PropTest02IntSum.java`, and so on, and compile and run these in turn.

The first three give simple examples of how the QuickTheories library can be used. Before running each one, see if you can guess what the results might be. Try suggesting and writing some other simple property-based tests.

For the last test class, `PropTest04SpaceCollapse.java`:

- Examine and run the test.
- See if you can suggest and implement *other* properties the `collapseSpaces` method should abide by.

Discuss with a partner what properties you have come up with – have they come up with anything different?

Sample solutions

Other possible properties:

- If a string contains *no* spaces, collapseSpaces should always return the same result
- If a string contains *all* spaces, collapseSpaces should always return the empty string