

CITS5501 Software Testing and Quality Assurance

Semester 1, 2022

Workshop 1 – Testing introduction – solutions

0. Accessing required software

We will be using the Java language for the bulk of the workshops, so you should make use of a Java IDE (Integrated Development Environment).

If you are using a university computer, you should be able to access the [BlueJ](#) IDE.

If you are using a laptop or home computer, you may install and use [BlueJ](#), or you may install another Java IDE if you prefer.

Some freely available options are:

- Netbeans, downloadable from <https://netbeans.org>
- Eclipse IDE for Java Developers, downloadable from <https://www.eclipse.org/downloads/packages/release/2020-12/r/eclipse-ide-java-developers>
- IntelliJ IDEA Community, downloadable from <https://www.jetbrains.com/idea/download/>

As a first step for today’s workshop, ensure you can install and/or access at least one of these.

We will give instructions for BlueJ version 5.0, but it should be straightforward to adapt these to other IDEs.

1. JUnit tests

Download and compile workshop code

Download the `workshop-01-code.zip` file from <https://cits5501.github.io/resources/#lab-workshops>, and unzip it somewhere on your computer.

Open the code as a “project” in your IDE. For BlueJ, this is done by selecting “Project” / “Open Non BlueJ”, and selecting the directory containing the Workshop 1 code.

Ensure you can compile the project – in BlueJ, by selecting “Tools” / “Compile”.

If using an IDE other than Java: you may need to instruct the IDE to add the “JUnit 5” libraries to the project; typically, viewing the project *properties* in your IDE will reveal some way of doing this.

Take a look at the `Calculator` class, in `Calculator.java` – this class has trivial functionality, but is useful as an example of a *class under test*.

Take a look at the `CalculatorSimpleTest` class, in `CalculatorSimpleTest.java`. This class defines a number of *JUnit tests* for our `Calculator` class.

Test classes can be called anything, but by convention, *unit tests* (which are written to test a single class) usually start with the same name as the *class under test*, followed by a description of the test (or just the word “Test”).

Run the JUnit tests

Run the tests in the `CalculatorSimpleTest` class.

In BlueJ, this is done by right-clicking on the class (after compiling) and selecting “Test All”.

You should see that some tests “pass” (with green ticks) and some “fail” (with red crosses) – see if you can work out what the failing `testSubtract` test is telling you about what the problem is.

Inspect the JUnit tests

Look at the parts of the `CalculatorSimpleTest` test class, using the JUnit User Guide (<https://junit.org/junit5/docs/current/user-guide/>) as a reference.

- Test classes can be called anything.
- Test cases are written in methods annotated `@Test`
- For each test, the methods annotated `@BeforeEach` and `@AfterEach` are run before the test and after the test, respectively.
These methods can be used to create and destroy test *fixtures* – in Java, fixtures are normally a set of objects in a known state. (The state *can* include things outside the Java program, however – databases, files on a remote system, anything we like. But for unit tests, the fixtures will only be Java objects.)
- The `testSubtractThrowsException()` test is intended to discover whether the `Calculator.subtract()` method throws an *exception* in circumstances where it should.

The code

```
47     Throwable exception = assertThrows(  
48         ArithmeticException.class,  
49         () -> c.subtract()  
50     );
```

calls the `assertThrows` method, which is used to assert that when its second parameter (a bit of executable Java code, called a *lambda expression*) is run, it throws the exception specified by its first parameter.

We will look at these more later.

- Note that the first few test methods take no arguments, but the test `addZeroHasNoEffect` is what JUnit calls a parameterized test – unlike other test methods, it does take arguments.

We will look at these more later; but JUnit’s parameterized tests are designed to make it easy to run what are called *data-driven tests* (see Wikipedia on [Data-driven testing](#)), as well as a subset of data-driven testing called *property-based testing* (see the explanation given by the [Hypothesis](#) Python-based library for doing this sort of testing).

Consider the following question: if you want to get all the tests passing, how do you determine what each method is supposed to *do*, and when it is correct? (After all, someone writing the test could have made a mistake in the test code.)

Sample solutions:

To find out what each method is supposed to do, the only reliable option is to read its documentation. If there is no documentation for the method, then you can never really know what the developer’s intention was.

2. API documentation

Look at the `Calculator.java` class from the workshop 1 code.

Can you identify

- a. A Javadoc comment, which documents the API?
- b. A Java comment which is *not* Javadoc?

Use your IDE to run the `javadoc` tool, which generates API documentation from source code.

- In BlueJ, select “Tools” / “Project Documentation”, then look for a directory called “`doc`” which should be created within the source code directory.
- BlueJ should automatically open the generated documentation in your browser.

View the generated documentation in your browser.

Identify one class member marked `private`, and make it `public` and write a Javadoc comment for it. Re-run `javadoc` – what changes do you see in the generated documentation?

3. Fix the code

See if you can fix the code in the `Calculator` class so that all the tests pass.

For the `subtract` method – aside from other changes you might need to make, you might want code something like the following:

```
1   if (/* some condition goes here */) {  
2       throw new ArithmeticException("can't return a negative result");  
3   }
```

Try creating your own new tests. In BlueJ, if you right click on a class, there should be an option to create a test class. Use the existing tests as an example – can you think of other tests we might add?

Sample solutions:

Fixing errors:

The code for `subtract` has the numbers in the wrong order.

The code for `subtract` also needs to throw an exception when the result would be negative.

e.g.

```
1   if (num2 > num1) {  
2       throw new ArithmeticException("can't return a negative result");  
3   }
```

4. Concepts review questions

Answer the following questions to test your understanding of concepts introduced in the lectures and prescribed reading.

For each of the following scenarios, explain whether you think a *failure*, a *fault* or an *erroneous state* (or none of these, or more than one) has occurred, and explain why. If it is a failure – is it non-conformance with a functional or a non-functional requirement?

- The social media site “Witter” allows users to specify that their email and date of birth should not be displayed publicly. But after a system update, that information is now visible for all users.
- The ride-sharing app Habari runs on a user’s mobile phone, and communicates with Habari’s servers to find nearby divers and arrange a ride. However, the communications are not encrypted, meaning a tech-savvy user could manipulate the system and obtain free rides.
- Your colleague Mila is writing a method which should return the arithmetic mean of numbers in a list:

```
1 double total = 0;
2 for (double num : number_list) { total += num ; }
3 return total / number_list.length
```

However, when the list is of length 0, this code returns the result “INFINITY”.

Possible solutions and discussion:

To come to a conclusion for these questions you will need to make *reasonable assumptions* and state what they are. Your answer can vary, depending on what assumptions you make.

The answers below are one possibility; but yours may be different.

- a. We are told that users can specify that their email and date of birth should not be displayed publicly. So if those are displayed publicly, then that’s a failure: the system is not behaving in accordance with requirements.

We don’t know in this case what the exact requirement is, but most likely, it is something like:

“If the user specifies that their email and address and date of birth should not be displayed publicly on their profile, then those details should not appear on the user’s public profile.”

So the failure is non-conformance with a functional requirement.

Presumably, there’s also a problem somewhere in the code for the Witter site that causes this failure, but we aren’t given details: this problem constitutes a *defect*. And once execution hits that point in the code, the system will go into an *erroneous state*.

[This scenario is loosely based on a security failure that occurred to Google’s social networking site, “Google+”, in 2018. See the article by Newman (2018) reporting on this [here](#); there is also a [Wikipedia page](#) on the incident.]

- b. In this scenario, we aren’t told exactly what the requirements for the Habari system are; but presumably one of them is that communications between the mobile app and the Habari company servers should be secure. (Even if this is not an explicit requirement, it’s a reasonable expectation of drivers, users, and Habari management. Drivers and users presumably don’t want details of transactions to be visible to third parties, and management presumably would like people to pay for rides rather than getting them for free – else the company will go out of business.)

If the requirement is something like “All communications between the mobile app and the Habari servers should be made securely” then this would be a *non-functional* requirement. (We’ll discuss later in the unit whether that requirement is precise enough.) In this context, “functional” means “treating the system as if it were a *function*, a black box that takes in inputs and spits out outputs”; and “non-functional” means “the manner in which some system function executes”, rather “not functioning”.

For this unit, you aren’t expected to know the details of how secure computer

communications work, but if you do, you are welcome to discuss what you can infer about the defect in the source code which causes the above failure. (In the scenario given, we probably could point to specific spots in the code where things should have been done differently – spots in the code where the programmers developing the Android app and the Habari servers should have used appropriate techniques to ensure that communications were secure. These would be *defects*.)

[This scenario is loosely based on one described by Shubham Paramhans (2015), who identified flaws in the security for the “Ola” ride-sharing app, as described in a [Medium.com](#) article, “[Fooing the Startup of the Year](#)”.]

- c. When Mila’s method returns the result “INFINITY”, we’d need to know what method’s specifications where in order to know if this was a failure or not (i.e., we’d need to look at the method’s documentation).

As a matter of mathematics, the arithmetic mean for n numbers a_0 through a_n is defined as $\frac{1}{n} \sum_{i=1}^n a_i$, or $\frac{a_1 + a_2 + \dots + a_n}{n}$ (Weisstein 2005).

So when $n = 0$, the result is undefined, because dividing by zero is not defined. If we were writing a specification for this method, what are some reasonable things we could require the method to do in this case?

Here are a few reasonable options:

- i. The method should return the special floating-point value “INFINITY”. This is Java’s default behaviour when floating-point numbers are divided by zero. The advantage of this approach is it requires no extra work; but “INFINITY” perhap isn’t the best choice, since it doesn’t mean exactly the same thing as “undefined”.
- ii. The method should return the special floating-point value “Nan” (“Not A Number”). This will require a little extra code, but reflects the correct mathematical answer slightly better than option (i) does.
- iii. The method should throw an exception (e.g. something like Java’s `ArithmeticException` class). Throwing an exception is usually what maps best to a mathematically undefined result; but the caller having to handle the exception may not be very convenient.

We’d need to know more about the business requirements here to know which is the best option.

In any case: if the behaviour of Mila’s method does match its requirements, then there hasn’t been a failure; but if it doesn’t match them, then there has.

References

- Newman, Lily Hay. 2018. “Google+ Exposed Data of 52.5 Million Users and Will Shut Down in April.” *Wired*, December 10, 2018. <https://www.wired.com/story/google-plus-bug-52-million-users-data-exposed/>.
- Paramhans, Shubham. 2015. “Fooing the Startup of the Year: Hacking into Ola Wallet.” *Medium Remember What That Was. I Was Monitoring My Phone Traffic from*. <https://medium.com/@CodeTheDevil/busting-ola-wallet-1ceea6174b1f>.

Weisstein, Eric W. 2005. “Arithmetic Mean.” MathWorld. March 4, 2005. <https://mathworld.wolfram.com/ArithmeticMean.html>.