CITS5501 Software Testing and Quality Assurance
Semester 1, 2022
Week 3 workshop – ISP – solutions

1. Binary search

Consider the Javadoc documentation and signature for the following Java method, which searches inside an array of `char`s for a particular value.

(Adapted from the Android version of the Java standard library.)

```
/**
 * Performs a binary search for @code value in the ascending sorted
 * array @code array, in the range specified by fromIndex (inclusive)
 * and toIndex (exclusive).  Searching in an unsorted array has an
 * undefined result. It's also undefined which element is found if there
 * are multiple occurrences of the same element.
 *
 * @param array the sorted array to search.
 * @param startIndex the inclusive start index.
 * @param endIndex the exclusive start index.
 * @param value the element to find.
 * @return the non-negative index of the element, or a negative index
 *         which is <code>-index - 1</code> where the element would be
 *         inserted.
 * @throws IllegalArgumentException if <code>startIndex > endIndex</code>
 * @throws ArrayIndexOutOfBoundsException if
 *         <code>startIndex < 0 || endIndex > array.length</code>
 * @since 1.6
 */
public static int binarySearch(char[] array, int startIndex, int endIndex,
    ↪ char value)
```

Discuss how you would go about creating tests using Input Space Partitioning.

a. What steps are involved?
b. What is the input domain?
c. what characteristics and partitions would you use?

Sample solutions:

a. Steps involved are:

1. Identify functions
2. Identify parameters
3. Model the input domain (partitioning it using characteristics)
4. Choose partitions, and values within them
5. Refine these into tests
6. Review

b. The input domain consists of:

- the set of all possible *arrays* of `char`s (for all practical purposes, we can consider this domain as being infinite in size)
- the set of all possible `int`s (2 parameters of this sort)
- the set of all possible `char`s (there are 256 of them)

c. Characteristics and partitions:

Some characteristics we could use for our `int`s are:

i. Are they less than, greater than, or equal to zero?
ii. Do one (or both) of them represent positions at the start of the array (i.e. are they zero)?
iii. Do one (or both) of them represent positions at the end of the array (i.e. are they `array.length - 1`)?
iv. Is the first parameter greater than, equal to, or less than the second?
(this gives us 3 partitions)

Some comments on these:

Characteristic (i) is okay, but doesn't show much thought has been put into what we're testing for here, and what the method does. It's something a machine might come up with (interface-based) rather than a person thinking about the intended behaviour of the method.

Characteristic (iii) is a characteristic of *two* parameters combined.

For characteristic (iv) - all of these partitions *are* sensible, and are worth testing for. When the first parameter is greater than the second, then the method documentation says an `IllegalArgumentException` should be thrown – so we should test for that and make sure that it is.

(You might ask: why should we write a test just to make sure some exception is thrown? The answer is that this is part of the "contract" of the method, and *is* important. In practice, software developers *do* want to know what exceptions a method can throw, and do rely on methods throwing what they say they will.)

Some characteristics we could use for our array are:

- Is it of size zero, one, or is it larger than one?
(Note that if it is of zero, our int parameters will necessarily be outside it, and exceptions will be thrown.)

Note that a characteristic like "is the array in ascending sorted order?" is *not* a useful characteristic here. The array *must* be in ascending sorted order; that's a precondition of calling the method, and if it's not true, the behaviour is undefined, so what "expected behaviour" could we possibly test for?

Some characteristics we could use for our char are:

- Is it equal to 0?
- Is it equal to 255?

(These are boundary values for the `char` type.)

2. Stack class

Suppose we have a Stack class that stores `int`s, with the following method signatures:

- `public IntStack ();`
- `public void push (int i);`
- `public int pop (); /** Throws an exception if empty */`
- `public boolean isEmpty()`

Assume the object state consists of an `int` array.

a. If we wanted to model `push` as a function, what sort of function would we use? How about `pop`?
b. Identify all the parameters for the `pop` method, and suggest some characteristics that can be used to partition the input space.

Sample solutions:

a. We would model the `push` Java method as a mathematical function that maps from "the old state" and an integer, to a new state:

$$push : ([\mathbb{Z}], \mathbb{Z}) \to [\mathbb{Z}]$$

(We use $[\mathbb{Z}]$ here to indicate an array of integers.)

Basically, the function is treating the method as if were something more like: `[int] push( int oldState[], int i)`, and returned a new state.

For `pop`, we would model it as $pop : [\mathbb{Z}] \to (\mathbb{Z}, [\mathbb{Z}])$ – a function that takes in the state of the stack, and returns a result and a new state. That is, it's as if the method instead of having signature `public int pop ()` had instead a signature something like `Pair<int, [int]> pop()`.

(See https://docs.oracle.com/javase/9/docs/api/javafx/util/Pair.html for the `Pair` type.)

b. Pop has effectively one parameter, the object state.

Some possible characteristics:

- Does the array have zero elements in it, or one, or more?
  (This partitions the domain into 3.)

It doesn't make sense to have "nullness" as a characteristic. If the array were null, the object would be in an invalid state, and what "expected behaviour" could we possibly test for?

3. Requirements

Consider the following, each of which is supposed to be a system requirement. Discuss with a partner – do you think it would be straightforward to write tests for them? If not, why not?

a. The flight booking system should be easy for travel agents to use.
b. The `int String.indexOf(char ch)` method should return a -1 if `ch` does not appear in the receiver string, or the index at which it appears, if it does.
c. Internet-aware Toast-O-Matic toasters should have a mean time between failure of 6 months.

System requirements:

a. "Easiness of use" requirement

This would be difficult to test.

- "Easy to use" is not a very *precise* requirement. It is the opposite of precise – it is *vague* or *fuzzy*; it is difficult to pinpoint exactly which systems satisfy it and which don't.

  A better requirement might be something like:

  > "Travel agents shall be able to use all the system functions after successful completion of a training course designed by the software provider. After this training, the average number of errors made by experienced users shall not exceed two per hour of system use."

  (This is adapted from *Pressman*.)

- A test like this requires human users, and would often not be done until the *acceptance testing* phase. Prior to that, the software provider might try to come up with a quicker and cheaper test to act as a *proxy* for the acceptance test – they might test it on non-technical staff in their own organisation, for instance.

  (The staff in their own organisation are acting as a sort of *test double* or *mock*, here – we will see more about these later. When we write unit tests, a "mock" is some sort of object that "stands in for" a real object that for some reason is difficult or inappropriate to use.)

⚠ If you are not clear about what makes a good requirement, you might want to review the chapter from the *Pressman* textbook on "Understanding Requirements" (in the 9th edition) or "Requirements Engineering" (in earlier editions).
There is also a quick summary (taken from documentation for an IBM requirements

management product) available here.

b. indexOf method

This seems straightforward to test, but leaves some behaviour *unspecified*.

It doesn't specify what happens if the character appears in the string multiple times – it says "the index at which it appears", implying there is only one. It would be better to specify "the *first* index at which it appears".

Once that is done, the method could still be made *more* precise – compare the actual Javadoc for the Java `String.indexOf` method. That documentation clarifies that `ch` represents a Unicode code point, and explains what happens when `ch` falls in various ranges.

Once corrected, the requirement is straightforward to write tests for.

c. "mean time between failure" requirement

For many purposes, this is probably precise enough (though one might want to add "under normal operating conditions", as opposed to "when operated in the open in a desert environment frequently subject to sandstorms").

As it stands, it is not easy to test, however, until after the toasters have been sold and are in normal operational use.

As with requirement (a), the manufacturer might make use of some sort of proxy test to assess the resilience of toasters. (Consider testing of car safety, for instance: do manufacturers "test" the safety of cars by simply selling them, and seeing what accidents occur? No – they do things like simulating wear and tear, and the effects of collisions.)

The manufacturer could perform studies to estimate what the effects of months or years of use on a toaster would look like, and find ways of achieving the same wear in a testing lab.