CITS5501 Software Testing and Quality Assurance
Semester 1, 2022
Week 7 exercise – solutions

Please refer to the CITS5501 website and the LMS for details of the due date and submission procedure.

Test plan for the `isHex()` method

Usually when writing integers, we represent them in base 10, but it can be convenient sometimes to write them in other bases, such as base 8 (octal) or base 16 (hexadecimal).

In software engineering, when an integer is represented in some base other than 10, the representation will usually have a prefix added to it to show what base it is in – for instance, "`0x`" for hexadecimal, or "`0o`" for octal.

A method `isHex()` that your team is testing has the following signature and Javadoc documentation:

```
/** Determine whether a String <code>s</code> represents an integer
 * in hexadecimal notation.
 *
 * To represent an integer in hexadecimal notation, a string must
 * satisfy all the following conditions:
 *
 * - It must start with the letters "0x"
 * - All the characters after the initial two must be either
 *   digits (i.e. in the range '0'-'9') or lowercase letters
 *   from 'a' to 'f'
 * - The string must have no leading zeroes - that is, the first
 *   character after the "0x" must be either a digit from '1' to
 *   '9', or a letter from 'a' to 'f'.
 *
 * @param s A string to be tested
 *
 * @return Returns true if <code>s</code> represents an integer in
 *    hexadecimal notation, and false if it does not.
 */
public static boolean isHex(String s);
```

A colleague of yours is devising a test plan for the `isHex` method by applying the Input Space Partitioning technique. They have identified that in this scenario, `isHex` is the only

relevant function, and that it has no parameters other than the `String s` (and you may assume these decisions of theirs are correct).

They have proposed several characteristics to be used in partitioning the input space:

Test plan excerpt – ISP characteristics for `isHex(String s)`

a. Is the `String s` equal to `null`?
   - Divides the input space into 2 partitions: situations where `s` is `null`, and those where it is not.

The following characteristics sub-partition the second partition of characteristic (a) (i.e., the situation where `s` is not `null`).

b. Does the `String s` start with an `0`, with an `0x`, or with some other sequence of characters?
   - Produces 3 partitions: Strings starting with `0`, Strings starting with `0x`, and other Strings.

c. Does `s` contain any characters besides the digits `'0'` through `'9'` and the letters `'a'` to `'f'`?
   - Produces 2 partitions: Strings which contain only these characters, and those which do not.

d. Before the first non-zero hex digit (i.e. character in the range `'1'`–`'9'` or `'a'`–`'f'`), does the digit '0' appear zero times, or once, or twice or more?
   - Produces 3 partitions: Strings where 0 appears zero times, Strings where it appears once, and String where it appears twice or more.

Answer the following questions about your colleague's proposed characteristics:

1. Has your colleague made a good choice of characteristics? Justify your answer. If you would modify or drop any of your colleague's characteristics, state which of these you would do, and why. (Max. 500 words; 10 marks)
2. Are there additional characteristics you recommend be used? (Assume that any changes you have recommended for question (1) have now been made.) If there are, you may suggest up to two, explaining your reasoning. If there are not, explain why not. (Max. 500 words; 5 marks)

Sample proficient answer:

Q. 1 – choice of characteristics

No, the characteristics suggested are a fairly poor choice.

Characteristic (a) is a poor choice of characteristic. It is normally assumed for Java methods that their parameters must not be `null` unless the method documentation explicitly says they can be. Writing tests with `null` as a test value will not tell us anything interesting about the method under test – they will just tell us that the Java runtime does indeed throw `NullPointerException`s when trying to operate on `null` values. The characteristic should be dropped.

Characteristic (b) is a poorly designed characteristic, because it does not result in a

partitioning. The set of "Strings that begin with `0x`" is a subset of the set of "Strings that begin with `0`"; therefore, the so-called partitions overlap and fail the test of being "pairwise disjoint". I would modify the characteristic. Firstly, I would specify that it only applies to Strings with two or more letters in them (i.e. it's a sub-characteristic). Secondly, I would state the characteristic as being "Are the first two letters of the string `0x`, or something else?", which gives rise to 2 partitions. The characteristic now gives rise to a proper partitioning.

c. Does `s` contain any characters besides the digits `0` through `9` and the letters `'a'` to `'f'`?
   - Produces 2 partitions: Strings which contain only these characters, and those which do not.

Characteristic (c) is a poor characteristic, because valid hex Strings *should* contain characters besides the digits `0` through `9` and the letters `'a'` to `'f'` – namely, they should contain an `'x'` as the second character. Using this characteristic tells us nothing useful about whether the `isHex` method works. A better characteristic would be if we (i) firstly, specified this is a sub-characteristic of Strings with at least two characters, whether the first two letters are "`0x`", and (ii) secondly, phrased the characteristic as "*After the first two characters*, does `s` contain any characters besides the digits `0` through `9` and the letters `'a'` to `'f'`?" The rephrased characteristic now provides a more useful partitioning of `s`.

Characteristic (d) is partly okay, except that it should apply only to characters after the first two. *Every* valid hex string should in fact have the digit `'0'` appearing as the first character, so that's not a useful partitioning. Similar to characteristic (c), we should make this a sub-partitioning of Strings with at least 2 characters and a correct prefix.

Sample proficient answer:

Q. 2 – additional characteristics

I would recommend having a characteristic "Does the string contain at least two letters?". Or, alternatively, incorporating that into characteristic (b) – i.e. make (b) be the characteristic "Does the String have at least two letters, of which the first two are `'0'` and `'x'`?" That's a valid partitioning (it divides the input space into two), and then characteristics (c) and (d) can be made sub-partitions of the "yes" case for characteristic (b).

Other than that, the proposed characteristics seem to cover everything that is necessary. The method documentation sets out three criteria which distinguish valid from invalid hex strings, and we have used all three criteria in our test plan.

General comments

You may have noticed that the specification for the `isHex` method has some peculiarities, compared with how you personally might have designed a method intended to distinguish valid hexadecimal representations of integers from invalid

ones.

For instance, the rule against leading zeroes means that zero, itself, is not representable! `0x0` should not be accepted as valid.

But when it comes to writing tests for the method, the documentation you are given is all you are permitted to work off. (Else, what exactly are you testing? Some imaginary, other method that you have invented.)

Common mistakes

Below are listed some common mistakes made when answering questions of this sort.

Failure to remove inappropriate characteristics

- **`null`-ness**. As slide 18 of lecture 4b (https://cits5501.github.io/lectures/lect04b--isp.pdf) points out, unless the documentation for a method explicitly states that parameters can be `null`, it's a precondition of Java methods that `null` parameters may not be passed (and it's undefined what happens if a `null` *is* passed). Therefore, `null`-ness is an inappropriate characteristic to use.

Incorrect understanding of what a method specification is

When writing test plans (or critiquing them, as in this case), it's important that you test *exactly the method that is specified* – not some other imaginary method you've invented. The purpose of testing is to check whether the described expectations of behaviour are met – no more, no less.

So behaviour *not* mentioned in the method description is irrelevant, and discussing irrelevant factors is penalized. Such behaviour or factors include:

- **Maximum size of an `int` in Java.** The documentation for `isHex` doesn't say anything about whether the String is, in fact, going to be parsed into a Java `int`, or whether it is going to be stored in an `int`, or even used in a Java data structure or program at all. So introducing characteristics based on the size of an `int` is incorrect.

  It is quite possible to store arbitrarily large integers in Java. The `java.math.BigInteger` class is available for this; it can store an integer as large as your computer's entire available memory if need be. If the checked String *was* intended to later be passed as an integer, this could be done using the `BigInteger` class; it's incorrect to assume the size of an `int` is relevant here.

Not reading the question

The question asks you one thing: are your colleague's characteristics good ones? Answering other questions indicates a failure to read or understand the question, and will usually be penalized on the grounds that irrelevant factors are discussed.

In particular, the question does *not* ask about:

- **Test coverage.** You are not asked to assess test coverage here, so any mention of coverage criteria (including "base choice" criteria) is irrelevant. Make sure you read the question carefully and do only what is asked.
- **Selecting test values and creating test cases**. In the questions given, you are not asked to create any test cases, nor to select test values – so any mention of those topics is irrelevant. Make sure you read the question carefully and do only what is asked.
- **Implementing the tests as Java code**. You are not asked to actually implement any tests. Any Java code included will likely be ignored when marking.

Repetition of irrelevant definitions

Answers that begin with a recitation of principles or definitions from the slides or textbook tend not to receive many marks. (For instance, an answer that begins: "Input Space Partitioning is a technique for modelling software components as functions and deriving tests from the model. This is done by dividing the input space into partitions using characteristics. Partitions must: (a) be pairwise distinct, and (b) ... etc".)

For one thing, you can assume that your markers are familiar with definitions and don't need to be reminded of them. (Know your audience.)

For another, this doesn't show an ability to justify *your particular recommendations*, by appealing to particular guidelines or factors.

Lastly, the question doesn't *ask* you to provide a list of definitions; make sure you read the question carefully and do only what is asked.

Poor or poorly justified reasons

- **Appeals to authority.** When justifying an answer, *never* do so by appealing to the fact that "the lecture slides say so". (Or, "the lecturer says so", or "the textbook says so".)

  In the lectures, I take care to explain *why* it's a good idea to take one approach over another, in terms of how it will contribute to software quality. Those are the reasons why you should favour one approach over another; not "because the lecture slides say so". (If you believe you've found a counterexample to this, please do let me know.)

Poor conciseness, clarity, and legibility

It takes little effort to make your answers clear and legible, and only a little more to keep them concise.

- *Do* make sure you clearly separate your answer into paragraphs. This makes it easier for a marker to read and understand.

- *Do* add headings or sub-headings if necessary, to show how your answer is structured and what topics you have addressed.
- *Do* check your spelling. Poor spelling and grammar will result in fewer marks for clarity and legibility.