Full Stack Development/Software Workshop 2

Formative Exercises: Testing Java programs

Exercise 1

A *palindrome* is a word or phrase that reads the same both forwards and backwards (see `https://en.wikipedia.org/wiki/Palindrome`).

a) Consider a method `public static boolean isPalindrome(String s)` that checks whether the given string is a palindrome. Case is ignored, as are non-letter characters such as spaces or punctuation, so e.g. `"O no"` counts as a palindrome. Decide a set of test cases for this method, making sure to include the boundary cases and to think about the purpose of each test.

b) Implement the tests in JUnit.

c) The following implementation is known to be buggy. What is the outcome of running the buggy method on your test cases? How can the bugs be fixed?

```
1  public static boolean isPalindrome(String s) {
2      s = s.toLowerCase(); // ignore case
3      int fromStart = 0;
4      int fromEnd = s.length()-1;
5      while (++fromStart < --fromEnd) {
6          // skip non-letter characters
7          if (s.charAt(fromStart) < 'a' || s.charAt(fromStart) > 'Z')
8              fromStart++;
9          if (s.charAt(fromEnd) < 'a' || s.charAt(fromEnd) > 'Z')
10             fromEnd--;
11         // different characters --> not a palindrome
12         if (s.charAt(fromStart) != s.charAt(fromEnd))
13             return false;
14     }
15     return true;
16 }
```

Exercise 2

a) Consider a method

`public static ArrayList<Integer> removeDuplicates(ArrayList<Integer> a)`

that takes a sorted list of integers `a`, which may contain multiple copies of each value, and returns a sorted list containing exactly one copy of each value that appears in `a`. Decide a set of test cases for this method, making sure to include the boundary cases and to think about the purpose of each test.

b) Implement the tests in JUnit.

Hint: you can turn an `ArrayList<Integer> arrList` into an array of type `Integer[]` using `arrList .toArray(new Integer[0])` (where the argument is needed because `toArray()` with no argument returns an array of type `Object[]`), and then write tests using `assertArrayEquals`.

c) The following implementation is known to be buggy. What is the outcome of running the buggy method on your test cases? How can the bugs be fixed?

```java
public static ArrayList<Integer> removeDuplicates(ArrayList<Integer> a) {
    int i=0;
    ArrayList<Integer> result = new ArrayList<Integer>();
    while (i<a.size()) {
        if (a.get(i) == a.get(i+1))
            i++;
        result.add(a.get(i));
        i++;
    }
    return result;
}
```

Exercise 3

The *Caesar shift* is a substitution cipher that cyclically shifts every letter in the alphabet by a fixed number of spaces. For example, with a shift of 4: A becomes E, B becomes F, ..., Y becomes C, and Z becomes D. (See also `https://en.wikipedia.org/wiki/Caesar_cipher`.)

a) Consider a method `public static String caesarShift(String plaintext, int shift)` where `plaintext` is an arbitrary string and `shift` is a number between `0` and `25`. The method should Caesar-shift all letter characters (both upper and lower case) and leave non-letter characters in the string unchanged. Decide a set of test cases for this method, making sure to include the boundary cases and to think about the purpose of each test.

b) Implement the tests in JUnit.

c) The following implementation is known to be buggy. What is the outcome of running the buggy method on your test cases? How can the bugs be fixed?

```java
public static String caesarShift(String plaintext, int shift) {
    char currentChar;
    StringBuilder ciphertext = new StringBuilder(plaintext.length());
    for (int i=0; i<plaintext.length(); i+=shift) {
        currentChar = plaintext.charAt(i);
        if (currentChar > 'a' && currentChar < 'z' || currentChar > 'A' &&
            ↪ currentChar < 'Z') {
            currentChar += shift;
            ciphertext.append(currentChar);
        }
    }
    return ciphertext.toString();
}
```

Optional extra practice

Go back to code you have written for any of last term's exercises and think about whether the code would benefit from testing and whether it is written so that it is easily testable.

- If not, could you change this?
- If yes, come up with a set of test cases, and implement them in JUnit. Does your code work correctly?