Lab 5 : WordAnalyzer class

Introduction – READ THE INSTRUCTIONS FIRST!

The WordAnalyzer class was written to perform a few different analyses of words:
  - identify the first repeated (adjacent) character in a word
  - identify the first multiply occurring (not necessarily adjacent) character in a word
  - count the number of groups of repeated (adjacent) characters

However, the version of the class given to you has some problems. It is your job to fix it so that it meets all the specifications in the comments of the source code and the test programs (unit tests).

In addition, you must keep an interactive Lab Notes log to show us along with demonstrating your WordAnalyzer class works properly to get checked off at the end of the lab. These are mostly relatively minor 1 or 2 line programming fixes; do not redesign the program.

This lab was inspired by a lab by Cay Horstmann.

Program Specifications

Make a new Lab5-WordAnalyzer directory in your cs15u home directory and copy all the files from ~/../public/Lab5-WordAnalyzer/ to your new Lab5-WordAnalyzer directory you just made in your home directory.

There are three unit test programs to test each of the three public methods in class WordAnalyzer. But even the unit test programs can use some work.

Follow these lab instructions:

WordAnalyzerTester1

Take a look at WordAnalyzerTester1.java

This program is intended to unit test the method firstRepeatedCharacter() in class WordAnalyzer.

Notice the logging statements. In general, logging is a much more acceptable software engineering practice than using println statements.

Compile and run the first test program: WordAnalyzerTester1.java

$ javac WordAnalyzerTester1.java
$ java WordAnalyzerTester1

Yikes!

Well, we do have some logging statements in WordAnalyzerTester1.java. Logging is turned off right now, so let's go ahead and turn them on by commenting out the line

    log.setLevel( Level.OFF );

so logging will now be on. Do this and recompile and run WordAnalyzerTester1.

That did not help much. Let's add some instrumentation to class WordAnalyzer.
As discussed in class, assertions should not be used to check preconditions for public methods/ctors, but can be used to check postconditions for public or private methods and check class invariants. When a WordAnalyzer object gets created, we want to make sure the String reference `word` is not null, otherwise there is no word to analyze. This can be considered a class invariant - something that must be true about each instance of an object of some class type, such as checking that the state of an object is valid.

Since we may want to check this at multiple locations in a class, it may be better to write a private method to do this. Something like:

```java
private boolean hasValidState()
{
    return this.word != null;
}
```

So let's add this class invariant assertion check to the end of the WordAnalyzer ctor:

```java
public WordAnalyzer(String word)
{
    word = word;
    assert hasValidState() : "ctor failed - not valid state.";
}
```

Now recompile and run WordAnalyzerTester1:

```
$ javac WordAnalyzerTester1.java
$ java WordAnalyzerTester1
```

Before going any further, why could we just compile the test source file / main driver after making a change to WordAnalyzer.java? Don't we need to compile WordAnalyzer.java? _________________________________

Try this:

```
$ ls -lt
$ touch WordAnalyzer.java
$ javac WordAnalyzerTester1.java
$ ls -lt
```

We talked about this in class.

Now let's run the program now that we added an assert statement.

```
$ java WordAnalyzerTester1
```

Hmmm … that did not help much. Oops, we forgot that assertions are turned off by default. We need to turn them on when we run our program:

```
$ java -ea WordAnalyzerTester1
or
$ java -enableassertions WordAnalyzerTester1
```

The exception stack trace is a little different, pointing to the assertion in the ctor WordAnalyzer.<init>
Fix the code in the ctor to really assign the formal parameter to the instance variable. Recompile and run WordAnalyzerTester1 again (java -ea WordAnalyzerTester1).

Looks like the first couple of tests worked, but then we hit another runtime exception.

This is a very common error that can probably be fixed without adding logging info.

What general kind of programming error is this called? ____________________________________________
(Not the kind of exception, the common programming error you are fixing).

Fix this error and recompile/test until all the initial tests pass.

Now edit WordAnalyzerTester1.java and add logging and test runs for the cases listed in each of the TODO comments.

For the case of passing in a null reference to the WordAnalyzer ctor, we first need to add a precondition check in the ctor to make sure the formal parameter is not null because the object is not in a valid state if word is null.

Add a check at the top of the WordAnalyzer ctor to throw an IllegalArgumentException with the message "word == null" if the incoming parameter is null. Again, we do not want to use an assertion to check arguments in public methods/ctors. Java defines an appropriate exception for just this kind of precondition check.

Recompile and run (java -ea WordAnalyzerTester1). You should get something like the following with the last test:

```
Jul 18, 2012 9:31:22 AM WordAnalyzerTester1 main
INFO: Testing null - expect: IllegalArgumentException
Exception in thread "main" java.lang.IllegalArgumentException: word == null
    at WordAnalyzer.<init>(WordAnalyzer.java:15)
    at WordAnalyzerTester1.test(WordAnalyzerTester1.java:52)
    at WordAnalyzerTester1.main(WordAnalyzerTester1.java:45)
```

We do not want our test program to error out with this exception since we are explicitly testing for this exception. Change the test program to catch this exception and log it with the exception message like:

```
Jul 18, 2012 9:35:26 AM WordAnalyzerTester1 main
INFO: Testing null - expect: IllegalArgumentException
Jul 18, 2012 9:35:26 AM WordAnalyzerTester1 main
INFO: Got IllegalArgumentException: word == null
```

You should be able to put this test case anywhere in your series of tests and have your test program complete all the tests and terminate normally (no exit or uncaught exception).

**WordAnalyzerTester2**

Take a look at WordAnalyzerTester2.java

This program is intended to unit test the method firstMultipleCharacter() in class WordAnalyzer.

Compile and run WordAnalyzerTester2
The output is not nearly as helpful as that from WordAnalyzerTester1. If you have time, add logging to WordAnalyzerTester2 similar to that of WordAnalyzerTester1.

But more importantly, the output is wrong!

Let's use logging to find the problem in **WordAnalyzer.java**. Specifically in the `find()` method. Add the framework for logging to **WordAnalyzer.java** (not the tester file).

```java
import java.util.logging.*;

private Logger log = Logger.getLogger("WordAnalyzer");
```

And at the top of the ctor:

```java
// log.setLevel(Level.OFF);
```

so we can easily turn logging off in this class by uncommenting this line.

In the method `find()`, add the following lines in the appropriate places

```java
log.info("Entering find with c = '" + c + "', pos = " + pos);
log.info("Leaving find with i = " + i);
log.info("Leaving find with return value -1");
```

Compile and run WordAnalyzerTester2.

Use the logging messages to help you figure out why `firstMultipleCharacter()/find()` is not working properly to find the next index of the current char. Fix the bug, recompile and run. Make sure all test cases run properly.

Once all the tests are running correctly, disable logging by uncommenting the line

```java
log.setLevel(Level.OFF);
```

in WordAnalyzer.java. Recompile and run. Double-check all unit test results.

---

**WordAnalyzerTester3**

Take a look at **WordAnalyzerTester3.java**

This program is intended to unit test the method `countRepeatedCharacters()` in class **WordAnalyzer**.

Compile and run WordAnalyzerTester3

Oops, the `countRepeatedCharacters()` method does the right thing for the first two test cases, but it fails on the word "aabbcdaaaabb". It should report 4 repetitions, but it only reports 3. Time to debug!

If you decide you need to print some state of any of the variables in this method, use logging!
The countRepeatedCharacters() method looks for character sequences of the form $xyy$, that is, a character followed by the same character and preceded by a different one. That is the start of a group of repeated chars. Note that there are two conditions. The condition

```java
if (word.charAt(i) == word.charAt(i + 1)) // found a repetition
```

tests for $yy$, that is, a character that is followed by another one just like it. But if we have a sequence $yyyy$, we only want to count it once. That is why we want to make sure that the preceding character is different:

```java
if (word.charAt(i - 1) != word.charAt(i)) // it's the start
```

This logic works almost perfectly: it finds three group starts: $aa$ $bcda$ $aaa$b

Why doesn't the method find the start of the first ($aa$) group?

Look at the result of the test case for the word "aa" also.

Why can't you simply fix the problem by letting $i$ start at 0 in the for loop?

Fix the bug. You may need to add code vs. changing a few char characters. Sometimes dealing with a special case is needed.

Be sure to keep a running log (Lab Notes) of each observed bug, hypothesize what is wrong, how you fixed it, and how you tested the fix. Be sure to include erroneous hypotheses and non-fixes. Learning from mistakes is as important (if not more important).

Get one of the CSE 15L staff to check you off for the lab. You will need to go through your log describing each bug interaction and demo your fixed version with all three unit tests.