CSE115 Lab 5 Fall 2016

DUE DATES:
Monday recitations: 9:00 PM on 10/16
Wednesday recitations: 9:00 PM on 10/18
Thursday recitations: 9:00 PM on 10/19
Friday recitations: 9:00 PM on 10/20
Saturday recitations: 9:00 PM on 10/21

Ready!

This lab consists of a series of debugging exercises. Think of them as little mysteries or puzzles that you need to solve. The lab is again structured into several parts (each in its own package), and each one is a little more challenging than the previous ones, so do them in order.

Packages part1 through part4 have:

> **Driver.java**: Use this to run your code. This class contains a main method that instantiates the Lab5 class.
>
> **Lab5.java**: This class contains code that, when run, tells you what the code should be giving as an answer as well as what is actually does give. Use this to help you track down what the problem(s) might be, and how to fix them.
>
> **Zoe.java**, **Mal.java**, and **Wash.java**: All parts have Zoe, parts 2 through 4 have Mal, and parts 3 and 4 have Wash. These contain quotes from characters with these names, from the Joss Whedon series *Firefly* (described as a space-western).

In each part you may only makes changes to the Zoe.java, Mal.java and/or Wash.java files to fix the bugs. Do NOT change anything in Driver.java. Make changes to the Lab5 constructor as indicated. When you submit to Web-CAT the Driver.java and Lab5.java files are excluded.

Packages part5 and part6 have:

> **Driver.java**: Use this to run your code. This class contains a main method that instantiates the Lab5 class.
>
> **Lab5.java**: This class contains code that, when run, tells you what the code should be giving as an answer as well as what is actually does give. Use this to help you track down what the problem(s) might be, and how to fix them.
>
> **Words.java** and **Vocabulary.java**: The Words class is defined for you. Do not change it. The Vocabulary class has some some methods stubbed out.

In each of these parts you may only makes changes to the Vocabulary class to produce desired behavior. Do NOT change anything in any other classes. When you submit to Web-CAT the ONLY class that will be submitted is Vocabulary.java.

WORKING FROM YOUR OWN MACHINE

If you wish to work on labs from your own machine, please see the 'working on your own machine' instructions posted on the lab 5 page of the course website. YOU MUST follow the software

installation instructions to get the software components necessary for you to be able to work on your own machine AND be able to submit your work to Web-CAT.

If you decide to work from your own machine you should bring it with you to recitation so that you can start and finish your work there.

NOTE: It is NOT acceptable to skip recitation just because you are working from your own machine. Students who have attended recitation have priority in office hours. Both TA and instructor office hours are supplements to recitation and lecture, not a replacement for either.

The java.lang package
The java.lang package has a special status in the Java language: everything in this package is automatically imported. This means you can use members of java.lang without fully qualifying their names.

The java.lang.String class
In this lab we will be using java.lang.String objects quite a bit. Since this class is defined in the special package java.lang we do not need to full qualify its name, and refer to it simply as String.

String objects represent text. The String class is unique among the Java classes in that it allows us to express String objects directly as sequences of characters enclosed in double quotation marks. What does this mean? Well, suppose we have declared an instance variable in this way:

```
private String _quote1;
```

then we can initialize that variable with an assignment that looks like this:

```
_quote1 = "We live in a spaceship, dear.";
```

Even though we do not see the familiar *new + constructor* class instantiation expression, _quote1 holds a reference to a String object representing the character sequence *We live in a spaceship, dear*.

You can combine existing Strings to form new Strings using concatenation. String concatenation is performed using the operator '+'. For example, the expression:

```
"Zoe: " + _quote1
```

has as its value the String:

```
"Zoe: We live in a spaceship, dear."
```

Note that a new String object is created – neither of the original String objects is changed.

The java.lang.System.out.println method
Another useful class in the java.lang package is called System. It has a public static variable named 'out' which holds a reference to a java.io.PrintStream object. The PrintStream class defines

a method named 'println', which can take a String as an argument. The println method prints the String to "standard output". In Eclipse you see this output in the Console view. Your TA will show you how to open this view if you don't already have it visible. If you run your program outside of Eclipse standard output is displayed in the terminal window from which your program was started.

For example, when this line of code is executed:

```
System.out.println("What I got was: ");
```

the character sequence *What I got was:* is printed to standard output.

Comments
Sometimes it is useful to embed text into a .java file that is meant only for human readers, not for the compiler. Any such text is referred to as a comment. Java provides two ways to embed comments in a source code file.

The first is a line comment. A line comment begins with two forward slashes, and ends at the end of the line. The compiler will ignore everything from the two slashes to the end of the line (including the slashes). For example, in this code,

```
public Lab5() {  // Firefly is hilarious
```

the compiler would treat this as if it had been written as follows:

```
public Lab5() {
```

We can also use this to selectively include or exclude lines of code on different runs of our program. For example, this code:

```
public Lab5() {
    checkZoeFirstQuote();
//  checkZoeSecondQuote();
}
```

makes a call only to the checkZoeFirstQuote() method, and not to the checkZoeSecondQuote() method. In contrast,

```
public Lab5() {
//  checkZoeFirstQuote();
    checkZoeSecondQuote();
}
```

makes a call only to the checkZoeSecondQuote() method, and not to the checkZoeFirstQuote() method.

You will find this useful to independently verify different parts of the code in parts 1 through 6.

Set!

1. Log in
2. Start Eclipse
3. Switch to the CVS Repository Exploring perspective
4. Check out the CSE115-Lab5 project from the Labs repository
5. Switch to the Java perspective

Go!

Part 1

Run the Driver in the part1 package. You should see the following output in the Console view:

```
Calling getQuote() the first time on a Zoe object should produce:
    Zoe: We live in a spaceship, dear.
What I got was:
    We live in a spaceship, dear.
```

If needed, edit the code in Zoe.java so that the output matches what is expected when the Driver is run. Remember, you may not make any changes to Driver.java, nor to Lab5.java, to fix this bug. There may be more than one way to solve the problem.

Once you have fixed this problem, edit the constructor of part1.Lab5 so that the checkZoeSecondQuote() method is called rather than the checkZoeFirstQuote() method. If needed, edit the code in Zoe.java so that the output matches what is expected when the Driver is run.

BEFORE CONTINUING WITH PART 2, ENSURE THAT BOTH OF THE 'check' METHOD CALLS PRODUCE THE CORRECT VALUE. Uncomment both methods calls in the constructor, run the Driver, and make sure that both produce correct answers.

Part 2

Before you start part 2, make sure you close ALL your currently open editor windows. Otherwise you run the risk of editing a file from part1 rather than a file from part2.

Run part2.Driver. Notice that no output is produced. This is because all of the lines in the part2.Lab5() constructor are commented out. Comment in one method at a time, editing the part2.Zoe and part2.Mal classes to make the output correct.

BEFORE CONTINUING WITH PART 3, ENSURE THAT ALL SIX OF THE 'check' METHOD CALLS PRODUCE THE CORRECT VALUE. Uncomment all six of the method calls, run the Driver, and make sure that all six produce correct answers.

Part 3

Before you start part 3, make sure you close ALL your currently open editor windows. Otherwise you run the risk of editing a file from an earlier part rather than a file from part3.

Run part3.Driver. Notice that no output is produced. This is because the body of the part3.Lab5 constructor is empty. In this part there are twelve methods that you need to check. Write each of the twelve method calls. Comment in one method at a time, editing the part3.Zoe, part3.Mal and part3.Wash classes to make the output correct.

BEFORE CONTINUING WITH PART 4, ENSURE THAT ALL TEN OF THE 'check' METHOD CALLS PRODUCE THE CORRECT VALUE. Uncomment all twelve of the method calls, run the Driver, and make sure that all of them produce correct answers.

Part 4

Before you start part 4, make sure you close ALL your currently open editor windows. Otherwise you run the risk of editing a file from an earlier part rather than a file from part4.

Run part4.Driver. This time output IS produced. There are already method calls written into the constructor. This time things are a little more challenging. You might have noticed that in the earlier part each of the check methods could be run independently of the others because new Zoe, Mal and Wash objects were created in these methods and the getQuote() method was called as many times as needed to get the relevant value.

In part 4, however, the calls are dependent on one another. You cannot check the correct functioning of checkSecondQuote without checkFirstQuote having been called first. Likewise you cannot verify that checkThirdQuote is working correctly without both checkFirstQuote and checkSecondQuote having been called, in that order. Edit the part4.Zoe, part4.Mal and part4.Wash classes to make the output correct.

BEFORE SUBMITTING, ENSURE THAT ALL OUTPUT OF THE 'check' METHOD CALL IN ALL THE PARTS PRODUCE THE CORRECT VALUE.

Part 5

Before you start part 5, make sure you close ALL your currently open editor windows. Otherwise you run the risk of editing a file from an earlier part rather than a file from part5.

Notice that there are compiler errors in part5.Lab5. You will notice that the code in methods check_1, check_2, check_3, and check_4 makes calls to methods that **should** be defined in Vocabulary. This gives you a hint of something that you need to add to the Vocabulary class.

Keep in mind that you may edit ONLY the Vocabulary class, as that's the only class from part 5 that will be submitted to Web-CAT. The Vocabulary class must make use of the part5.Words class.

HINT: there should be five composition relationships between Vocabulary and Words.

BEFORE SUBMITTING, ENSURE THAT ALL OUTPUT OF THE 'check' METHOD CALL IN ALL THE PARTS PRODUCE THE CORRECT VALUE.

Part 6

Before you start part 6, make sure you close ALL your currently open editor windows. Otherwise you run the risk of editing a file from an earlier part rather than a file from part6.

Notice that there are compiler errors in part6.Lab5. You will notice that the code in methods check_1, check_2, check_3, check_4, check_5, check_6 and check_7 makes calls to methods that **should** be defined in Vocabulary. This gives you a hint of something that you need to add to the Vocabulary class.

Keep in mind that you may edit ONLY the Vocabulary class, as that's the only class from part 6 that will be submitted to Web-CAT. The Vocabulary class must make use of the part6.Words class.

HINT: there should be seven composition relationships between Vocabulary and Words.

BEFORE SUBMITTING, ENSURE THAT ALL OUTPUT OF THE 'check' METHOD CALL IN ALL THE PARTS PRODUCE THE CORRECT VALUE.

Submitting your project to Web-CAT

Make sure you submit your work on time; due dates are listed at the beginning of this lab description. This lab will be automatically graded by Web-CAT. You may submit as many times as you wish. Your last submission is the one that counts (so consider carefully whether you want to make any late submissions, as the late penalty is 20 points per day or portion thereof late).

Pay attention to the output produced by Web-CAT. If your submission scores 100 (without any early submission bonus you might be entitled to) you're all set. If your submission does NOT score 100, fix the problem and resubmit. Prior to the submissions deadline we expect that you will continue working until your submission scores 100.
