Lab 6: Understanding Static Methods with Test Suites

Instructor: David Chiu Due: Beginning of Next Lab

**Important:** In this lab, you will be writing loops. You will inevitably write some infinite loops that will cause your BlueJ environment to freeze. When this happens, hold down the [ctrl] key and press [d].

**Instructions**

- Download the code provided for this lab from the course page, file named `MysteryMethods.java`.

- Not including the `main` method (which is provided, but empty), you should see several mystery methods below `main()`. In practice, is no hard-and-fast rule on whether the static methods should appear above or below `main()`. In this part of the lab, you will be digging in and determining each of the mystery method’s functionality.

- I’ll walk you through the first one. Let’s hone in on `method0()`. Remember, the basic step to understanding a method is to interpret its signature (or header). Always be ready to answer these questions:
  - What does this method require as input? A single parameter, which is a double
  - What are these inputs named inside the method? `num`
  - What does this method promise to return to the caller? An `int`

- We don’t yet know what this method does, but this header communicates an enormous amount of information on how to call it, and what type of value to expect in return.

- Now let’s figure out what this method does by calling it a few times with various inputs. Type the following “dummy code” (known as test cases) inside `main`:

```java
/** Method 0 test */
System.out.println("*** Method 0 Dummy Test Suite ***");

//dummy 1
System.out.println("Dummy Test 1: " + method0(55));

//dummy 2
int mysteryVar;
mysteryVar = method0(3.14);
System.out.println("Dummy Test 2: " + mysteryVar);

//dummy 3
double dVar1 = 3.14, dVar2 = 2.18;
System.out.println("Dummy Test 3: " + method0(dVar1) * method0(dVar2));

//dummy 4
mysteryVar = 5 - method0(method0(1.25) * 5.9);
System.out.println("Dummy Test 4: " + mysteryVar);
```
• Compile and run this code. If you typed everything out correctly, you should see the following output on the Terminal:

```
*** Method 0 Dummy Test Suite ***
Dummy Test 1: 55
Dummy Test 2: 3
Dummy Test 3: 6
Dummy Test 4: 0
```

• Let’s dig into each of the test cases:
  – Dummy 1: We input 55, and 55 is returned to us. A good starting point, but not very interesting.
  – Dummy 2: We input 3.14, and store the returned value in `int mysteryVar` before printing it out. The print statement tells us that 3 was returned, which appears to be the whole number portion of the input. This observation also holds for Dummy 1.
  – Dummy 3: Two things to observe here. First, notice that the inputs into the method can also be variables, and second, the methods can be used just like any other value (in this case, in an expression that multiplies them together). Keep note that our assumption that the method returns the whole number portion still holds true.
  – Dummy 4: Finally, observe that the method calls can be nested in this case. The inner method will be called first, returning an integer value of 1. That is multiplied with 5.9 to return 5.9. This value is a `double`, which is what `method0()` expects as input. The outer `method0()` is now called on 5.9, returning an integer value of 5. Finally, `mysteryVar` holds 0 since it takes the difference from 5.

• Of course, we can’t exhaustively run every test case possible. There’s an infinite amount of cases to check, which is impossible for even the world’s fastest computer to compute — remember Alan Turing’s *Halting Problem*?

• However, we can guess with reasonably high confidence that `method0()` takes any real number (double) and returns the whole number portion (int).

• To confirm, let’s now take a closer look at `method0()`’s body, which contains single line of code:

```
public static int method0(double num)
{
    return (int) Math.abs(num);
}
```

• We know from our data type lectures that the syntax:

```
(datatype) expression
```

will cast the expression into the given data type, and that when you cast a double into an int, only the integer portion of the double is kept. Therefore, if this method simply returns the casted value, we can now confirm our hunch that this method returns the whole number portion of any given real number.

• **Not so fast!** We see that the method-body’s expression is further calling the `Math.abs()` method to return the absolute value of `num`. Our original hunch was wrong, and our test-suite failed us, D’oh! I should have been more careful and created a few cases that inputs negative values:
A value of 4 printed informs me that, indeed, this method not only returns the whole number, but also returns the absolute value of that number.

- **Scope:** But what is the relationship of `num`, inside the method, to the rest of the code in `main()`? Try printing out `num` inside `main()`. What can you conclude? Remove this line afterwards.

- **Scope:** What if you changed the value of `num` inside the method? Try assigning `num` to -1 before the return statement. Then after `dummy 3` is called, try printout the values of `dVar1` and `dVar2`. They were used to call `method0()`, and if changes are preserved, then these variables should hold -1 inside `main()`. Undo these changes afterwards.

- **Scope:** Similarly, are any values from `main()` visible within `method0()`? Add a line to print `mysterVar` inside `method0()` right above the return statement. What can you conclude? Remove this line afterwards.

- **Return Statement:** I mentioned in class that the `return` statement immediately hands the control-flow back to the method’s caller. Confirm this is true. Add a line of code to print out “Does control-flow reach here?” after the `return` statement. What can you conclude? Remove this line.

- **Documentation:** Now that we know what this method actually does, let’s give it a proper name and documentation so that other programmers down-the-road won’t be confused.

```java
/**
 * This method applies absolute value to the given real number, and returns the whole number portion.
 * @param num A given real number
 * @return An integer representing the whole number of the given value
 */
public static int getWholeNumberAbs(double num)
{
    return (int) Math.abs(num);
}
```

This specific way of documentation is known as Javadocs. Once you’ve made these changes, from your source code editor, click on the selection menu on the top right corner and switch from “Source Code” to “Documentation.” You should see an entry for `getWholeNumberAbs()` with the description we commented.

- In Javadocs-style commenting, you always lead off with a high-level description of the method. This is followed by a list of its parameters, denoted with `@param paramName Description of parameter`. For methods with multiple parameters, you would have multiple `@param` lines. The list of params is followed with `@return Description of returned value`.

**Your Task: Finish the Rest!**

- Using the above steps, your goal is to determine the service provided by the remaining methods. For each method, interpret its signature, create a reasonable test-suite to determine its function, change the method’s name to something more appropriate, and provide the Javadocs-style comments.
Your test-suites should go completely within `main()`. Label each test-suite with comments so that the sections are clear.

Keep this in mind when creating test-suites: Your goal is two-fold: (1) come up with multiple cases that are a representative sample, and (2) come up with cases that might elicit interesting or informative results. For instance, because I left out a test for negative values, I almost misinterpreted what `method0()` actually does, had I not been more careful. Trivial values like 0 (when inputting numbers) and empty-string `""` (when inputting Strings) should normally be included in a test-suite.

Another thing to try in your test-cases is to call other methods! For instance, now that we know what `getWholeNumberAbs()` does, we can try it in conjunction with `method1()`. For instance, what would happen if:

```java
System.out.println("Dummy X: " + method1(getWholeNumberAbs(-9.6182), 3));
```

• Note: Method 7 is curious... it returns nothing and seems to call itself! But if it calls itself, would it ever stop? See if you can trace the call on paper to understand what the method does. For the trace, use the following:

```java
int [] A = {10, 20, 30, 40};
method7(A, 0, A.length-1);
```

Recall that unlike scalar inputs, changes made to an array input are preserved after the method ends!

• Switch partners between every method for which you complete!

• You will be graded on the quality of your test cases, and the quality of your Javadoc comments.

Part III: Submitting Your Lab

After you have completed the lab, please do the following to submit your work.

□ Zip up all necessary files.

□ Rename the zip file to `LastName1_LastName2_Lab6.zip`, where the LastNames are the respective last names of the authors.

□ Submit it on Moodle. You may submit as often as you'd like before the deadline. I will grade the most recent copy.