BSM 261: Object Oriented Programming
LAB 6: Loops, Strings, and StringBuilder
Also an introduction to testing

November 10/11, 2022

Date Performed:	November 10/11, 2022
Instructor:	ORHAN ÖZGÜNER
Teaching Assistant:	YUNUS EMRE AVCI

1 Class Material, Lab&Quiz Scores

You can access to the class material from the following link:
https://drive.google.com/drive/folders/1yo4pvaTngQ1L2TkHuWPCzbfapLDyvwUt?usp=sharing

You can check your lab attendance and quiz scores from the following link:
https://docs.google.com/spreadsheets/d/1xKQEqMSGlj3RBo9kLh97UzbTO1fyCvaT/edit?usp=sharing&ouid=104851561367296854968&rtpof=true&sd=true

2 Getting Started

Before you perform the lab please check the following information:
• Make sure that the lab computer you are using has DrJava downloaded into the desktop. Here is the jar file to check:

drjava-beta-20190813-220051.jar

• If you can not locate DrJava, please download it from the following link and place it to your desktop.

https://sourceforge.net/projects/drjava/files/latest/download

• If you have the DrJava jar file in your desktop, you are ready to perform your lab. In order to use jar files in linux, you need to run it from terminal. Run the following commend from the terminal to start DrJava

```
java -jar  /Desktop/drjava-beta-20190813-220051.jar
```

3 Lab Rules

The purpose of the lab is to get you used to using DrJava, to start experimenting with Java, and to digest the topics you learn over the week.
Here are the lab rules:

- The teaching assistant will make a short recap of the week. Then you will perform the lab in the recitation hour.
- Your grade will be determined by your participation and finishing the assigned work during the recitation. When you get stuck on a question, please ask for help.
- Once you finish all of the tasks, show your work to the TA to receive full credit.
- **If you did not join the lab, or joined more than 10 minutes late or did not show your work to the Teaching Assistant, or did not finish all of the questions in class time you will receive 0.**

4 Lab Instructions and Questions

In this lab, you will practice writing loops, and you will manipulate the String and StringBuilder classes. You will also start writing a testing class that will verify your code.

4.1 A first class and method

Create a class called Lab6 and add the following method to the Lab6:

Method 1: Create a static int method called countSpaces that takes a single String parameter named s. Write the method so that it returns the number of spaces in the String s.
Hint: the only methods from the API you need are the **charAt** and **length** methods of String.
Compile and test your method.

4.2 Writing a testing class

Up to now, you have used the interactions pane for testing your code. This is nice but you have to keep entering the tests. Instead, we can create a testing class so that you can run all your tests with a single button click.

Create a new class called Lab6Tester. You need to import two things at the top:

```
import org.junit.Test;
import static org.junit.Assert.assertEquals;

 public class Lab6Tester {
 }
```

The first imports the **JUnit** Test class. That is a class that contains code to let us create automatic tests for our program.
The second import is different. Notice the static. That means we are actually importing the static method **assertEquals** from the **Assert** class. This will let us write **assertEquals** instead of **Assert.assertEquals** in our code.
(You could also just do import org.junit.Assert, but then you would need to use Assert.assertEquals.)

Now, create an instance method **testCountSpaces** inside the **Lab6Tester** class that takes no input, and the method needs the **@Test** annotation:

```
public class Lab6Tester{
    @Test
    public void testCountSpaces(){
    }
}
```

We can now place a test inside the method. To create a test we use the **Assert.assertEquals** method. Since we did a static import, we can just write **assertEquals.** The method takes two inputs. The first is the expected value that our method should return, and the second is the actual method call.
For example, the string "**abcde**" has no spaces so we expect **Lab6.countSpace("abcde")** to return 0. We code that as:

```
@Test
public void testCountSpaces() {
    assertEquals(0, Lab6.countSpaces("abcde"));
}
```

Now compile the file. When it compiles and if there are no errors, the *Test* button at the top should become active. If you click the test button you should see All tests completed successfully and a green Test Progress bar.

Now, let's create a test that gives an error. "**a b c d e**" has 4 spaces, but let's give the wrong expected value:

```
@Test
public void testCountSpaces() {
    assertEquals(0, Lab6.countSpaces("abcde"));
    assertEquals(5, Lab6.countSpaces("a b c d e"));
}
```

Now compile the file. When it compiles and if there are no errors, the *Test* button at the top should become active. If you click the test button you should see a red Test Progress bar and the message `Failure: java.lang.AssertionError: expected <5> but was:<4>`. This shows that the countSpaces method returned 4, but the test was expecting it to return 5.

Fix the test so that it expects 4. Write a few more tests. Here is a guideline:
Test 0, 1, many. Test first, middle, last. That means that you should have a test with 0 spaces, a test with 1 space, and a test with many spaces. You should also make sure that some test has a space as the first character of the string, some test has a space as the last character, and some test has a space in the middle.

4.3 Writing more methods and loops

Method 2: Write a static String method **removeSpaces** in the **Lab6** class that takes a single String parameter **s** and returns out a new String with all spaces removed. Recall that you must use StringBuilder or StringBuffer when building a String.
Hint: Use the **charAt** method of String and the **append** method of StringBuilder.

Write a test method **testRemoveSpaces** in the **Lab6Tester** class. (Remember to put the @**Test** annotation above the method header.) Write a bunch of test cases using **assertEquals**, except now the expected value should be a **String** instead of an **int**. Remember to use the **Test 0, 1, many; Test first, middle, last** guideline, but that means you should use similar test strings to the ones you did for **testCountSpaces**.
I recommend writing all the test cases before you fix any errors in your method. That way you can just keep hitting the Test button and fixing errors until all tests are passed.

Compile, test, and fix your code.

A note on the assertEquals method: You may notice that when some test fails, it is not always easy to see which specific test failed. Here are two ways to design your tests to help you.

First, the assertEquals method is overloaded so that it can also have a String input that is a message printed if the test fails. The message should be the first input. For example:

```
@Test
public void testCountSpaces() {
    assertEquals(0, Lab6.countSpaces("abcde"));
    assertEquals(5, Lab6.countSpaces("a b c d e"));
}
```

can be changed to:

```
@Test
public void testCountSpaces() {
    assertEquals("The test with no spaces fails", 0, Lab6.countSpaces("abcde"));
    assertEquals("The test of five letters fails", 5, Lab6.countSpaces("a b c d e"));
}
```

Now you will have a custom message printed when a specific test fails.
Second, you get only one error message per testing method. If more than one **assertEquals** test fails, only the first one that fails prints a message. If you want to see all the tests that fail, then you should put each **assertEquals** call in its own method inside the Lab6Tester class.

Method 3: Write a static String method **everyNthChar** that takes two parameters, a **String s** and an **int n**. The method should output a string that contains **each nth character, separated by spaces.**

For example:
everyNthChar("abcdefghijklmn", 2) should output"**a c e g i k m**" and
everyNthChar("abcdefghijklmn", 3) should output "**a d g j m**"

You should use only a single loop.

Write a **testEveryNthChar** method in the tester class. The **Test 0, 1, many** guideline means you should have a test with the input string is an **empty string, where it contains a single character, and where it is a long string**. You should have the input int be 1 as well as a large value. The Test first, middle, last guideline means you should have a test where the last character must be output, and you should have a test where the last character should not be output.

Demonstrate your class to your lab instructor to receive full credit
LAb is over at this point but I wanted to add a challenge question for you to work on yourself:

Method 4: Challenge Write a static String method **allDigits** that takes a single int parameter i and outputs a string containing the digits of i one at a time, on a single line, in reverse order, separated by commas.

For example, given **5641**, the method will output "**1, 4, 6, 5**"
A hint is to use % and to isolate the digits. Try using a while loop.

Write a testAllDigits method in your tester class. The Test 0, 1, many guideline means you should have tests with input 0, a single digit number non-zero number, and a many digit number. The Test first, middle, last guideline does not really apply to this problem because all digits are output.