Worksheet 6 - Iteration and Arrays

Updated: 13th April, 2016

- Implement and understand the 3 loop types.
- Understand how to identify possible loops in a given problem
- Use arrays in conjunction with loops
- Understand arrays and their relation to Strings

1. Multiple Shapes

Again your first task is to modify the ***pseudocode*** of your ShapeCalculator. This time several modifications should be added:

- Alter your selection menu so it now loops until the exit condition is met.
- Add input validation to the menu selection. Any input that isn't one of your four options should prompt the user to re-enter a value until it is valid.
- Add input validation to all the measurement inputs. If any measurement is zero or negative then the user should be prompted to re-enter the value.

Note: Things to consider.

- Which loop type is most appropriate in each situation? FOR, WHILE, DO-WHILE?
- How will you structure your boolean expression?
- How do you represent your chosen structure in pseudo code?
- Menu loop: How does the data type you used in the selection affect your loop implementation?
- Now that your submodules are getting complex you should start making use of the assertion part of your documentation.

Once your pseudocode has been modified and checked by your tutor you can adjust your java code to match. Remember to *indent* your code and *update your documentation*!

Remember to test your new algorithm!

2. Leibniz Approximation of PI

By this point you have used Math.PI at least once, so lets investigate it. There are several formulas for calculating π, one of the common ones is Leibniz.

$$\frac{\pi}{4} \approx \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1}$$

If we expand this summation to say 2 terms we get the following.

$$\frac{\pi}{4} \approx \frac{(-1)^0}{2*0+1} + \frac{(-1)^1}{2*1+1}$$

What about 4 terms?

$$\frac{\pi}{4} \approx \frac{(-1)^0}{2*0+1} + \frac{(-1)^1}{2*1+1} + \frac{(-1)^2}{2*2+1} + \frac{(-1)^3}{2*3+1}$$

Now 6 terms?

$$\frac{\pi}{4} \approx \frac{(-1)^0}{2*0+1} + \frac{(-1)^1}{2*1+1} + \frac{(-1)^2}{2*2+1} + \frac{(-1)^3}{2*3+1} + \frac{(-1)^4}{2*4+1} + \frac{(-1)^5}{2*5+1}$$

Are you getting the point yet? Could you use some kind of loop to implement this pattern?

Note: Note that the formula calculates $\frac{\pi}{4}$, ie: one quarter of π. At some point you will need to account for this.

Your algorithm should prompt the user to input an Integer indicating the precision (ie: number of terms to be calculated), your algorithm should then calculate and output the approximate value of pi. It is a good idea to also output the value of the constant Math.PI to compare the difference.

Design your algorithm in pseudocode and **hand test** it. This means you will need to manually calculate some test data! Remember to ask for help if you get stuck.

You may then implement the algorithm in java.

In your algorithm did you use the Math.pow method? Lets have a look at what that is actually doing?

$n =$	Expanded Equation	Result
0	-	**1**
1	-1	**-1**
2	$-1*-1$	**1**
3	$-1*-1*-1$	**-1**
4	$-1*-1*-1*-1$	**1**
5	$-1*-1*-1*-1*-1$	**-1**
6	$-1*-1*-1*-1*-1*-1$	**1**
7	$-1*-1*-1*-1*-1*-1*-1$	**-1**

Each iteration is simply changing the sign. Have a look back to the expanded formula and substitute this new information. Now can you modify your algorithm so you do not have to use the Math.pow method? If you get stuck on this remember to **ask for help**.

3. Not Quite String Matching

Given what you know of Strings you must be curious how a search function works. So lets have a go at making a scaled version that counts the occurrences of a single character.

To go about this task your algorithm should preform the following steps. Have a look at the table of expected results if you get confused.

- Prompt the user to input a string.
- Copy each character of the string to an array — **read the note**.
- Prompt the user for a character to search for.
- Count how many times the character occurs in the array.
- Replace each occurrence of the character with the count.
- Output the final version of the array. OUTPUT array **is not sufficient!**

Note: In pseudocode you may treat a String the same way you would an array. The java equivalent of LENGTH OF myString is `myString.length()`

Why would (CONVERT TO Character 65) give you `A`?
To ensure that you actually get a number in your output you should initialise your variable like so `count = '0'`.
Now (CONVERT TO Character count) will give you 0. Do you understand why?

By replacing each occurrence of the character you can test if your algorithm is actually working. Here is a small set of test data you can use to verify your algorithm.

String	Char	Expected Output
Hello this is the looping prac for oopd.	o	Hell6 this is the l66ping prac f6r 66pd.
My name is Inigo Montoya prepare to die.	M	2y name is Inigo 2ontoya prepare to die.
never ends never ends never ends never	n	7ever e7ds 7ever e7ds 7ever e7ds 7ever
never ends never ends never ends never	e	n11v11r 11nds n11v11r 11nds n11v11r 11nds n11v11r

Once you have designed your pseudocode algorithm you may implement it in Java. Make sure you consider which submodules can be tested using JUnit and which need to be manually tested.

Have you tested your algorithm using the provided data? What was your result for the last test case?

End of Worksheet