CSSSKL142 - Lab 4

Looping

University of Washington Bothell

Summary

This lab deals with building individual and nested loops, evaluating loop behavior, and debugging. You are welcome to structure the code as you find best, that is, feel free to put it all in a single file or to split each part into separate files. Parts 1 and 2 are the lab exercise, you just completed!

Part 3

(Single Loops) A loop that contains no inner (or nested) loop represents a single Repetition Control Structure, and is related to branching in that a boolean expression will govern the behavior of the loop. ***Loop 0*** above is an example of a singular loop, as are the following problems. Once we've practiced with a few individual loops, we'll experiment further with **nested loops** (loops inside loops, just like the loop 2 above).

Starting with one grain of rice, double the number of grains of rice you receive each day. If a king paid you for **64 days**, how many grains of rice would you have? Write a short, method with the name countGrains that calculates and prints the number of grains you earn each day and the running sum of all the grains to date.
Note: Do you observe anything unusual in your output when you run your code? Explain, in comments.

```
Day 1 and you got 1 grain(s) of rice for a total of 1 grain(s).
Day 2 and you got 2 grain(s) of rice for a total of 3 grain(s).
Day 3 and you got 4 grain(s) of rice for a total of 7 grain(s).
...
Day 30 and you got X grain(s) of rice for a total of Y grain(s).
Day 31 and you got X grain(s) of rice for a total of Y grain(s).
Day 32 and you got X grain(s) of rice for a total of Y grain(s).
...
```

Part 4

(More Single Loop Exercises)

(a) Write a method powerOfTwo that takes an integer argument and prints to the console what power of 2 the given number is. If the number is not a power of 2, then simply print a message that says so. For example,

```
powerOfTwo(64); //should print: "64 is 2 to the power of 6"
powerOfTwo(73); //should print: "73 is not a power of 2"
```

Note: you are not allowed to use any library functions!

(b) Write a method numBackward that reverses the digits of a given integer (that is, an integer passed in as an argument). Use a single loop and modular arithmetic to accomplish this goal. Play around with ideas on paper first. After you reverse the digits, compare the number you got with the original argument and determine if the original argument is a **palindrome**. Your method should print a message to the screen that states the reversed number and whether or not it is a palindrome. For example,

```
numBackward(123456789) //should print: "backward: 987654321, not palindrome!"
numBackward(123454321) //should print: "backward: 123454321, palindrome!"
```

Part 5

(Nested Loops) In this section, we'll produce more complex output by wrapping one loop inside another. This so-called nesting produces more complex results at the cost of increasing program complexity, and in the worst case can make some programs intractable. We'll limit our degree of nesting to *k* == 2 here to get started. Build a short, standalone program with the class name **StraightLine** that produces a straight line, just as in loop 0 in section 2 above. Use a loop variable called size to terminate the loop. The output should look like:

```
***                         //size == 3
*****                       //size == 5
```

Now, wrap this loop that produces a line inside another loop, as demonstrated in loop 1 in section 2. Make the outer loop also terminate using the "**size**" variable. The only thing to add to the outer loop is a System.out.println(); to move the cursor to the next line. When you're done, the output should look like the square below, with the same number of rows and columns.

```
***                           //size == 3
***
***

*****                         //size == 5
*****
*****
*****
*****
```

Part 6

Write a program with the class name **BoxMaker** that asks the user for an integer x (using Scanner), and then builds a box of x asterisks. For example, if x == 4, then your output would be:

```
****                    // x == 4
*  *                    // inner body " ": x - numLids ("*" on each side) == x - 2
*  *
****
```

Part 7

(More Nested Loop Exercises) Write methods that print the following patterns to the console such that their size depends on a passed-in integer argument.

(a) This first shape is a triangular pattern with sides of 6 asterisks. The call that produces this pattern is `shape7a (6);`. Your code should work for any positive integer.

```
* * * * * *
. * * * * *
. . * * * *
. . . * * *
. . . . * *
. . . . . *
```

(b) This second shape is an "X" pattern with arms of 3 asterisks each. The call that produces this pattern is `shape7b (3);`.Your code should work for any positive integer.

```
* . . . . . *
. * . . . * .
. . * . * . .
. . . * . . .
. . * . * . .
. * . . . * .
* . . . . . *
```

Part 8

(Debugging) Debugging your code is a valuable time-saving and problem-solving technique, and is easy to do in both BlueJ and Eclipse. Learn how to set a breakpoint (usually in the "gutter" of the code window, a column to the left) in your software and how to execute your code in "debug mode" so that your software pauses execution at each breakpoint. Using this technique, one can pause execution and inspect the values of variables. Debug any of the previous code you have written in this lab, and determine intermediate values of loop variables as they execute.

Don't forget to submit (parts 3 through 7) on Canvas.