CCPS506 Lab 1 – Pharo & Smalltalk Basics

Preamble

This lab is intended to introduce you to the Pharo environment and get you started in Smalltalk. Pharo is intimidating at first, particularly for those who are accustomed to compiling and executing simple source files. Luckily Pharo has a handy tutorial built in! Once you've worked through that, you'll be solving a few problems that would be trivial in a more familiar language but might prove more vexing in Smalltalk.

Lab Description

First thing's first, you can download Pharo for yourself at the following link:

https://pharo.org/

When you install and run the Pharo Launcher, you'll have to download a Pharo image. At the time of this writing, the latest version is 8.0. I choose 32-bit because 64-bit versions have given me trouble in the past (you won't need 64-bit for anything we'll do in this course). Once the image is downloaded, select and launch it. Take a few minutes to play around in the environment. Click around a bit, explore the context menus, and open the *Playground*, *Transcript*, and *System Browser* windows. Each of these three can be found under *Tools* when left-clicking.

Playground is an interactive window where you can write and execute Smalltalk code, Transcript is where the output will be displayed, and System Browser is where you can browse the class hierarchy and create your own classes. We saw examples of all this in class, check out the lecture slides if you need a reference.

0) Time for fun! In the Playground window, type **`ProfStef go.`** This statement sends the **`go`** message to the class object **`ProfStef`**. Highlight it, right-click, and select *do-it*. Alternatively, CTRL-D is the keyboard shortcut for *do-it*. Please don't just skip this tutorial. It's very good and will teach you a lot. At around page 19 you'll get into stuff we haven't covered in class. Feel free to skip the rest of the tutorial or keep going.

1) Next, create a new class category through the System Browser called CCPS506, and a new class called Lab1. Check the lecture slides for a reference on how to do this. Your Lab1 class will implement a rudimentary circular buffer. It must have the following methods and instance variables:

- A method called **`bufferInit:`** that accepts an array of literals as an argument.
- An instance variable called **`arr`** that is used to store the array object passed with the **`bufferInit:`** message.
- A method called **`getElem:`** that accepts an integer as an argument. This integer is to be used as a *circular index*. Out-of-bounds indexes should correctly wrap around. This includes negative values! Remember that Smalltalk is 1-indexed. An argument of 1 should correctly return the first element, and an argument of 0 should return the *last* element.

All of this can be done with some clever arithmetic. You don't need any branching or control structures (which we haven't learned yet).

Submission

Submitting Smalltalk code is a bit tedious, since we can't easily extract a single source file containing all the code of a given class. For this lab, you will submit two things:

i) The code for your **`getElem:`** method. This can be pasted into a simple text file.
ii) A screenshot of your Playground and Transcript windows that demonstrates the correct output for a handful of test cases. Be sure to choose tests that demonstrate that your getElem: method indexes in the correct circular fashion. For example:

```
buf := Lab1 new.
buf bufferInit: #($A $B $C).
buf getElem: 0     "Should return $C"
buf getElem: 4     "Should return $A"
buf getElem: -2    "Should return $A"
buf getElem: 11    "Should return $B"
```

The screenshot and source code can be submitted as two separate files, or you can paste both the code and screenshot into a word document or PDF or similar.

Labs are to be submitted ***individually***! Submit your answers on D2L.