**CS 273 Laboratory 6: Arrays (Part 1)**

This laboratory will give you practice creating, using, and manipulating arrays.

**Preliminaries**

Load, compile and run ArrayMain.java. It first prompts the user for the size of an array to create (any positive number will work). Once you give it a number, it creates an array of ints, numberArray, of the given size in the variable numElements, and initializes each cell in the array to a random value between -1000 and 1000, inclusive.

Presently, the program prints nothing because everything is commented out and dummied up. Take the time to examine the Java code and see how it is creating this array. For the first part of this lab, you will be adding code to examine the numbers in the array.

**Laboratory**

**Part 1: Fun with one-dimensional arrays**

Modify ArrayMain.java so that performs each of the following tasks. After you complete each task, demonstrate your program to your lab instructor or assistant using array sizes of 5 and 31.

**Checkpoint 1 (10 points):** Modify the code so that it prints out the elements of the array all on one line, separated by a single space. Then use a loop to print a row of 20 dash (‘-‘) characters on the next line.

**Checkpoint 2 (10 points):** Modify the code so that it computes the average of the values in the array. The average must be as **precise as possible** (i.e. reporting 20 as the average isn’t good enough when 20.543 is actually the average). Uncomment the line that prints this value to the terminal window and edit it so that it works with your code.

**Checkpoint 3 (10 points):** Use the debugger to examine an array by setting a breakpoint somewhere in the code you added. You can add a breakpoint by clicking in the left margin where the line numbers appear. The code must be compiled before you can set a breakpoint. You can set breakpoints only on executable code. In other words, you cannot set a breakpoint on a comment line or a blank line.

After you have set the breakpoint, run the program. When the breakpoint is hit, the debugger window should open. Double-click on the array’s name in the debugger window. A window will pop up that displays the array’s contents.

**Show your instructor/assistant that you can examine the array’s contents.** This skill set may be useful to you later in this lab, in future labs, or in your CS 203 homework assignments. You can remove a breakpoint by clicking on it. You do not need to recompile when you add or remove breakpoints since the code is not changing.

**Checkpoint 4 (10 points):** Modify the code so that it finds and returns the minimum element in the array. Do not sort the array first. Uncomment the line that prints this value to the terminal window and edit it so that it works with your code.

**Checkpoint 5 (10 points):** Modify the code so that it computes the percentage of elements that are divisible by two. The percentage must be as **precise as possible** (i.e. reporting 20 as the average when 20.543 is the percentage isn’t enough). Uncomment the line that prints this value to the terminal window and edit it so that it works with your code.

**Checkpoint 6 (10 points):** Modify the code so that it computes the percentage of the elements whose value lies between -300 and 300 **inclusive**. Uncomment the lines that print these values to the terminal window and edit them so that they work with your code.

**Checkpoint 7 (10 points):** Modify the code so that it sorts the array's elements in **descending** order using **selection sort**. Print the sorted elements to the terminal window, all on one line, separated by a single space. Then print a line of 20 dashes.

**Checkpoint 8 (10 points):** Print out the sorted elements from checkpoint 7 again, but this time print them across multiple lines with up to 10 elements on each line. (The last line may have fewer elements.) After completing this, you should have the sorted elements printed once all on one line followed by the sorted elements printed again across multiple lines, up to 10 on one line.

**Part 2: Fun with two-dimensional arrays**

In your next lab you will implement the logic for capturing pieces in the game of "Go." If you are not familiar with the game of "Go," don't worry about that right now. This week, your goal is merely to become familiar with the game board and with the program that you will be modifying. Get started by performing these steps:

* Create a folder named lab7 in your cs273 folder. Go to the course website and access the starter code for Lab 7. Unzip the archive into the lab7 folder you just created.
* Compile and run GoFrame.java.
* At the present time, the project causes a frame to be displayed that contains four buttons and a Go board that by default is 19x19. The only button that is implemented is the "Spiral" button. Pressing this will cause two spirals made out of Go pieces to be drawn on the board. No other functionality is currently implemented.
* Review the code.

### Using the board Array

This program contains a variable named board. This is a two-dimensional, rectangular array of integers that represents our board in the game of Go. (Note: the boardvariable does not appear in your GoFrame.java file, nor should it.) Write the declaration for it in the space below, just so you know what it looks like. Do \*\*NOT\*\* add this to your GoFrame.java file.

The declaration for board looks like this: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

An element of the array typically contains one of the values WHITE, BLACK or EMPTY, depending on whether the square is occupied (and if so, with what color) or not. These names are defined as symbolic constants. See the definitions for these constants at the top of the GoFrame class. The actual values are integer constants such as 0, 1 and 2, but *it is good programming practice to refer to them by their symbolic names*. So, you should use the constants WHITE, BLACK or EMPTY when writing code for this assignment.

Whenever one of the squares changes in the array, it is reflected on the display the next time that the board is repainted. Your job is to modify/extend GoFrame.java so that it performs each of the tasks described below.

### Clear the board

Modify clearBoard so that it clears the entire board of pieces. (The clearBoard method is automatically called when the Clear button is pressed.) Do not hardcode the dimensions of the board to make this happen.

**Checkpoint 9 (10 points):** **Demonstrate to your instructor or lab assistant that all stones are removed from the board when the Clear button is pressed.**

### Add/change single stones

The method pressedOnSpace is called whenever the user presses the mouse-button at (or near) an intersection on the Go board. The intent is that when the user presses the mouse button, the stone at that location changes. Its parameters are the row and column coordinates representing the location on the Go board where the user pressed the mouse button. The row and col coordinates start from 0, with the top-left intersection being (0,0). Thus, col == 1 and row == 4 would indicate the 2nd intersection across and the 5th one down.

Your task is to modify pressedOnSpace to change board whenever the mouse is pressed at a legal intersection:

* if the stone is white, replace it with a black stone
* if the stone is black, remove it (i.e., make it empty)
* if the space is empty, place a white stone there

Unfortunately, pressedOnSpace is also called when the user clicks somewhere off the board itself (i.e., something that would be an intersection if the board had been extended). You will therefore need to do bounds-checking (to make sure you're not accessing the array out of bounds) before you perform the modification. If some kind of exception occurs, like array-out-of-bounds, a big black **X** will be displayed on the board.

Do not hardcode the dimensions of the board to make this work.

**Checkpoint 10 (10 points):** Show your code and demonstrate to your instructor or lab assistant that you can click near an intersection and the stone will change as specified above. Demonstrate that clicking near the edge does not display a big black X.