The State Universtiy of New York, Korea
Computer Science

CSE 114 Handout 6: PS 4 April 3, 2019

This problem set is due at **11:55pm on Wednesday, April 10, 2019, KST**. Don't go by the due date that you see on Blackboard because it is in EST. Go by the one given in this handout.

- Please read carefully and follow the directions exactly for each problem. Files and classes should be named exactly as directed in the problem (including capitalization!) as this will help with grading.
- You should create your programs using emacs.
- Your programs should be formatted in a way thats readable. In other words, indent appropriately, use informative names for variables, etc. If you are uncertain about what a readable style is, see the examples from class and textbook as a starting point for a reasonable coding style.
- This problem set assumes you have installed Java and emacs on your computer. Please see the course web for installation instructions.
- You must use the command-line interface to run your programs. That is, you must use the `javac` and `java` commands to do this problem set. Do not use Eclipse yet.
- **Be sure to include** a comment at the top of each file submitted that gives **your name** and **email address**.

Submit the following files on **Blackboard**. Multiple submissions are allowed before the due date. Please do **not** submit `.class` files or any that I did not ask for.

```
Arr.java
Rev.java
Rs.java
```

What Java Features to Use

For this assignment, you are not allowed to use more advanced features than what we have covered in Lectures 1 through 10 (April 3 lecture) [also Chapters 1 through 7 of our textbook].

Partial vs. Complete Solutions

Please read what I said in PS 1 on this issue.

Naming Conventions In Java And Programming Style In General

Refer to the ones given in PS 1.

Problem 1 (80 points)

Create a class named `Arr` in a file named `Arr.java` and introduce the following methods:

1. (6 pts) Define a method, named `randomArray` with two formal parameters: one integer indicating the size of the array to be created and the other indicating the upper-limit for the range of random numbers to be generated. If the second number is 101, then it would mean that the random numbers will be in the range of 0 and 100 inclusive. The function returns the array created as its return value. Read the entire problem including the hints at the end of this problem before you try to write your code.

2. (1 pt) In your `main` call `randomArray` with two actual arguments: 100 as the size and 21 as the upper-limit, and store the returned array into a local variable of your choice in `main`.

3. (2 pt) Define a method, named `print` with one formal parameter of type array of integers, which outputs the elements of the given array to the standard output device, i.e., the computer screen.

4. (1 pt) In your `main` call `print` with the local variable that you chose in step 2 above to see if the elements in the array are printed okay. You will want to use this print function whenever you want to see the elements of an array containing integers. Feel free to include a call to this function wherever useful, but comment out all the calls to this function in the final version that you hand in except the ones I specifically asked for.

5. (7 pts) Define a method, named `arrSum` with one formal parameter of type array of integers, which returns the sum of the elements in the array.

6. (2 pts) In your `main` print the average of the numbers in the array that you obtained in step 2 above. To compute the average, you must use `arrSum` that you defined earlier. Print the average to the standard output device, i.e., screen. When you generate output to the screen, add some annotation so that the output will be meaningful.

7. (6 pts) Define a method named `contains` with two formal parameters: one an array of integers and the other an integer. The method returns `true` if the second argument is contained in the first array argument; returns `false` otherwise.

8. (2 pts) In your `main` call `contains` with the array obtained in step 2 above and 12 as the second argument, and print the result to the standard output device.

9. (7 pts) Define a method named `contains2` with two formal parameters: one an array of integers and the other an integer. The method returns the index of the array where the *first* occurrence of the second argument is found if found. If the second argument is not contained in the first array argument, it returns -1.

10. (2 pts) In your `main` call `contains2` with the array obtained in step 2 above and 12 as the second argument, and print the result to the standard output device.

11. (10 pts) Define a method named `countMultiplesOf` with two parameters: one an array of integers and the other an integer. This method returns the count of the integers in the array that are multiples of the second parameter. For example, 8 is a multiple of 2, but not a multiple of 3, and zero is a multiple of any number. I suggest that you also define and use an auxiliary function that tests if a number is a multiple of another and returns a boolean value.

12. (2 pts) In your `main` call `countMultiplesOf` with the array obtained in step 2 above and 7 as the second argument, and print the result to the standard output device.

13. (20 pts) Define a method named `buildHistogram` with one parameter: an array of integers, say `scores`. This function builds a histogram and returns it. The histogram will also be an array itself, say it is named `hist`. This is how you build `hist`: First, find the largest element in `scores`, add one to it, and use the result as the size of `hist`. Now that `hist` is created we need to fill the array with some elements as follows: If `scores` contains 5 occurrences of 6, then 5 will be stored as the value in the array index 6 of `hist`. That is, `hist[6]` will hold 5. If you do that for each index of `hist`, then you have finished filling the array `hist`. I suggest that you add an auxiliary function that finds the largest element given an array.

14. (2 pts) In your `main` call ~~`buildHistogram`~~ with the array obtained in step 2 above. Store the histogram that is returned into a local variable of your choice in `main`.

15. (10 pts) Define a method named `printHistogram` with one parameter of type array of integers. Call this method from the `main` with the histogram obtained in one of the previous steps and let this method print the histogram in the following format:

```
i: <n>: <n of *'s for the index i of the array. n is the element
         of the histogram array in the index location i>
```

So, if the array holds the following values assuming the size of the histogram array is 8:

```
+---+---+---+---+---+---+----+---+
| 5 | 1 | 4 | 6 | 0 | 2 | 10 | 6 |
+---+---+---+---+---+---+----+---+
  0   1   2   3   4   5   6    7
```

Then, the histogram printed would look like this:

```
0: 5: *****
1: 1: *
2: 4: ****
3: 6: ******
4: 0:
5: 2: **
6: 10: **********
7: 6: ******
```

Note that the distribution in this histogram will show you how good the random number generator is, that is, how random are those numbers in the array.

Hints: Try to solve this problem *incrementally*. That is, solve it one step at a time and make sure the current step works correctly *before* you move on to the next step. Using random numbers are convenient if you don't want to generate arrays manually. However, testing your program with random numbers generated on the fly is very hard because you don't know what numbers you will be getting. So, while you are developing your program, I suggest that you use a small array with known elements that you include in the array yourself. After you finish debugging your program, switch it to a random number array with the specified size before you hand it in. Of course, you should test your final version to your satisfaction before you actually hand it in. You will most likely want to write a function that prints the elements of an array and use it as a debugging tool as you develop your program.

Hand in your revised `Arr.java`.

Problem 2 (30 points)

Create a class named `Rev` in a file named `Rev.java` and write the following methods:

- `public static void reverse (int[] A)`: This method reverses the numbers in `A` in place, i.e., *without using any additional array.* For example, if it receives an array containing 1, 2, 3, 4 with the elements in that order as actual argument, it reorders the values in the array to hold 4, 3, 2, 1.
- `public static void print (int[] A)`: This method prints the values in `A` to the standard output device, i.e., screen, in the order they appear in the array.

So, if we use a main method as follows:

```
public static void main (String[] args) {
    int[] arr = {1, 2, 3, 4, 5, 6, 7};
    print(arr);
    reverse(arr);
    print(arr);
}
```

it would generate the following output to the standard output device:

```
1 2 3 4 5 6 7
7 6 5 4 3 2 1
```

Include the main I gave you above in your program and add to the main at least one more array and calls to print and reverse much like I did in the one I gave you above. Hand in `Rev.java`.

Problem 3 (50 points)

Implement a class named `Rs` in a file named `Rs.java`. (`Rs` stands for `RandomSquare` and I am using this short name so that it will be easier to type when your grader grades your program later.)

In this problem you will be creating a 2-dimensional square array (called `board`) of element type `char` of size n where n is provided by the user from the keyboard. Your program is to fill the `board` with a marker, for example a dot ('.'). The user will provide another number which indicates what percentage of the cells on the board is to be filled in. So, if the size read in was 10 and the percentage read in was 50, then you will be filling in 50% of those 100 cells on the board. Your program will continue to fill in the cells until you reach the percentage value. How do you decide which cell to fill each time you fill another cell? Well, you will randomly select a cell each time – by selecting the indices of the 2-dimensional array, twice for each cell: once for the first index and once more for the second index. So, you will be generating a pair of random numbers to identify a cell to fill on the `board`. Note that it is possible to generate a pair that happens to identify a cell that has already been filled in. You are not allowed to fill in the same cell multiple times. Well, you can fill the same cell multiple times, but it should be counted as one in terms of reaching the percentage count.

As you design your program you must satisfy the following requirements:

1. Your `main` should read in two numbers: one for the size and another for the percentage value. It should also create the 2-D array, call a function that initializes the `board` with an appropriate initial value, call a function that fills the `board` by selecting cells randomly, and finally call a function that prints the `board` to the screen when the number of cells filled in reaches the percentage value. Your print function would just print the values in the `board` without worrying about printing the grid. That is, the printed result will be a mix of blank spaces and instances of the marker that you chose.
2. Your program must define and use a function that tests if a given cell is filled in or not. This would be a very simple function, but that is okay. I just want you to gain some experience of passing a 2-D array as an actual argument to a function and decomposing a complex problem into smaller pieces that make up the whole solution.

3. Your program must also define and use a function that fills in a specified cell of the board. This one will also be a simple function, but another chance for more experience! Later on, this kind of approach in designing a program will be very important when your program gets more complex.

4. Even if you know what non-local variables (static fields) are, you are *NOT* allowed to use non-local variables *AT ALL* in this problem. Use only local variables (including parameters) in this problem. Non-local variables make sense in some context as we will see when we write a Tic Tac Toe program later, but it is not desirable in this problem.

Would you expect to see any noticeable pattern in your output? Try a rather large size for the board and various percentage values and see what pattern, if any, you see. Hand in `Rs.java`.