CSE114 Fall 2023 : Assignment 4

Due: Oct 5, 2023 at 11:59 PM [KST]

Read This Right Away

For the due date of this assignment, don't go by the due date that you see on Brightspace unless you have set your timezone to KST. By default, Brightspace shows times in EST/EDT in the US.

Directions

- At the top of every file you submit, include the following information in a comment
  - Your first and last name
  - Your Stony Brook email address
- Please read carefully and follow the directions exactly for each problem.
- Your files, Java classes, and Java methods must be named as stated in the directions (including capitalization). Work that does not meet the specifications may not be graded.
- Your source code is expected to compile and run without errors. Source code that does not compile will have a heavy deduction (i.e. at least 50% off).
- You should create your program using a text editor (e.g. emacs, vim, atom, notepad++, etc).
- Your programs should be formatted in a way that is 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.

What Java Features to Use

For this assignment, you are *not* allowed to use more advanced features in Java than what we have studied ***at the time I post the assignment to Brightspace***. If you have a question about features you may use, please ask!

What to Submit

Combine all your .java files from the problems below into a single zip file named as indicated in the **Submission Instructions** section at the end of this assignment. Upload this file to **Brightspace** as described in that section.

Multiple submissions are allowed before the due date. Only the last submission will be graded.

Please do **not** submit `.class` files or any that I did not ask for.

Partial vs. Complete Solutions

Your programs should compile and run without errors, both compile-time and run-time errors! Please do not submit programs that do *not* even compile! Its better to submit a partial solution that compiles and runs as opposed to an almost complete solution that does not even compile. A program that does not compile even if it is very close to being perfect would only receive less than 50% maximum of the possible score for the program! This policy applies not only to this problem set but also to all the other ones in the remainder of the semester. I will not repeat this in each problem set though. So, please remember this.

Naming Conventions In Java And Programming Style In General

Please use these conventions as you establish your *programming style.* Programming professionals use these, too.

- **Names:** Choose informative names,
  - e.g. `hourlyRate` rather than `hr` for a variable name.
  - `CheckingAccount` rather than `CA` for a Java class name.
- **Class names:** We start a class name with an uppercase letter as I have in my examples: `Hello` not `hello` or `HELLO`. For multi-word names you should capitalize the first letter of each word, This is called 'Pascal' case.
  - e.g. `UndergraduateStudent`, `GraduateStudent`, `CheckingAccount`
- **Variable names:** We start a variable name with a lowercase letter. This is called 'Camel' case.
  - e.g. `count`, `hourlyRate`, averageMonthlyCost
- **Method names:** Naming convention for method names is the same as that for variable names.
- **Use of white space:** Adopt a good indentation style using white spaces and be consistent throughout to make your program more readable. Most text editors (like emacs and vim) should do the indentation automatically for you. If this is not the case, see me and I can help you configure your setup.

I will not repeat these directions in each assignment, but you should follow this style throughout the semester.

Problem 1 (15 points)

Create a java file named **EliminateDuplicates.java** containing the class **EliminateDuplicates.**

I will discuss the code in the **EliminateDuplicates** class shortly. For your code in that class to be easy to write, you will need some 'utility' methods. You will place these methods into a source file named **ArrUtils.java** and it will contain a single class names **ArrUtils**.

The helper methods you will write should at least include:

randomArray – This generates an array filled with random integer values.

printIntArray – Prints all the elements in an integer array

printIntArrayRange – Prints the elements of an integer array from a given first element to a given last element.

contains – A method that returns true or false. It returns true if the provided array contains a given value. False otherwise.

locateFirst – Returns the index of the first occurrence of an integer value in an array.

I will give prototypes for each of these methods here. Note that you may add additional helper methods if there is a good reason for them.

public static int [] randomArray(int rangeMax, int count)

This will return a new integer array containing count elements with values in the range 0 through rangeMax-1.

public static void printIntArray(String head, int[] ary)

This method first prints the string in head. Then starting on a new line, it prints all the elements in ary with space separator characters.

public static boolean contains(int[] theArray, int validElements, int theValue)

This method examines each element in the argument theArray up to the index that matches validElements. If validElements is -1, it will search the entire array. It returns true if theValue occurs as one of the elements in the array. It returns false otherwise.

public static void printIntArrayRange(String head, int[] ary, int firstElement, int lastElement)

This method first prints the string in head. Then starting on a new line, it prints all the elements in ary from the element at index firstElement up to (but not including) the element at index lastElement. It places a space between each integer value as a separator. If the array has no elements between the indicated indices, have the method print '**Nothing to print!**'.

public static int locateFirst(int[] ary, int val)

This returns the index of the first integer in the array matching val. If val does not occur in the array, it reutrns -1.

Write a method called eleminateDuplicates. The prototype (method header) is:

public static int[] eliminateDuplicates(int[] ary)

This method takes an integer array as an argument. It returns an integer array that contains the unique elements of the input array (so without any duplicate values) followed by a 'marker' with the value -1 followed by the list of duplicated values from the original array.

Write a test method that can be used to test your eliminateDuplicates method. This test method can be called from main() to run several different test cases. The test method can simply be called with the 'max range' and 'count' values you will pass to randomArray. The testEliminateDuplicates() method does not return a value (so it returns void) and it should do the following:

1. Call randomArray() with the provided parameters for max range and count.
2. Call eliminateDuplicates() which will give a returned array that contains both the filtered elements and the list of duplicate values (separated by a -1).
3. Check for the -1 separator element in the returned array.
    a. If there is no -1 separator in the returned array, then just print the full array. This would happen because there are no duplicates or possibly if the array is empty. If this array has at least 1 element (but no duplicates), have the method print '**No Dups:**' before printing the filtered array.

b. If there is a separator, use **printIntArrayRange()** twice to print the filtered array first and then the duplicates (remember, you can use the String argument to label each of these outputs for you.

Here is a few example calls to **testEliminateDuplicates()** as well as sample output. Remember, the output will always differ as the numbers in the array are random.

Example code calling testEliminateDuplicates()

```
System.out.println("Test 1: range 25, count 50");
testEliminateDuplicates(25, 50);

System.out.println("Test 2: range 5, count 0");
testEliminateDuplicates(5, 0);

System.out.println("Test 3: range 100, count 10");
testEliminateDuplicates(100, 10);
```

Example output

```
Test 1: range 25, count 50
Removed Dups:
3, 4, 13, 2, 9, 24, 5, 11, 20, 8, 16, 6, 7, 21, 14, 15, 18, 1, 19, 0, 12, 10, 22, 23
Duplicates:
2, 2, 11, 9, 3, 11, 9, 2, 18, 4, 3, 21, 9, 19, 0, 15, 9, 24, 11, 6, 19, 8, 9, 23, 7
Test 2: range 5, count 0
No Dups:
Test 3: range 100, count 10
Removed Dups:
60, 46, 32, 36, 89, 81, 16, 13, 4, 80
Duplicates:
Nothing to print!
```

Note on the example tests I chose above.

Notice carefully how I picked 3 test cases that would really exercise the code. One is fairly normal (25 is the range so there will definitely be duplicates!) The second case, creates a 0 length array to make sure I don't crash on an empty array (this can happen a lot in novice code). The third case tries to create a condition where there are no duplicates using a large range and short count.

This is how you should test all of your code going forward. You can try it with example test cases I give, but also, try to exercise some inputs that would really push your code to the limit and test the thoroughness of its logic.

Turn in both **ArrUtils.java** and **EliminateDuplicates.java**.

Problem 2 (10 points)

Create a Java file named **InterleaveArrays.java**

Part 1)

Write a method `public static int[] interleaveArrays(int[] array1, int[] array2)` that takes in two arrays of integers. Create and return a new array by interleaving the array together. This is a

simple 1 by 1 interleaving (take first element of first array then first element of second array, etc) You may assume that both arrays will have the same number of elements, but that the number of elements in the arrays could vary from one method call to the next.

Include in your program the test cases listed below and add at least 2 additional tests for interleaveArrays().

Some example test cases:

interleaveArrays(new int[]{1,3,5}, new int[]{2,4,6})
= [1, 2, 3, 4, 5, 6]

interleaveArrays(new int[]{10,20,30,40}, new int[]{2,4,6,8})
= [10, 2, 20, 4, 30, 6, 40, 8]

Part 2)

For this problem, create a `method public static int[] mergeArrays(int[] array1, int[] array2)` that will merge two sorted arrays so that the resulting array is also sorted. The method can assume each array has at least 1 element but cannot make assumptions about the size of either input array including whether or not the arrays differ in size.

The returned array should have a size that is exactly the combined lengths of the two input arrays.

Include in your program the test cases listed below and add at least 2 additional test cases for mergeArrays().

Some example test cases:

mergeArrays(new int[]{2,5,15,20,25,40}, new int[]{10,20,30,40}= [2, 5, 10, 15, 20, 20, 25, 30, 40, 40]

mergeArrays(new int[]{1,11,31,45,77,78,79,101}, new int[]{2,5,12,80,103,120}) = [1, 2, 5, 11, 12, 31, 45, 77, 78, 79, 80, 101, 103, 120]

Turn in InterleaveArrays.java

Submission Instructions

Please follow this procedure for submission:

1. Place the deliverable source files (`ArrUtils.java, EliminateDuplicates.java,` and `InterleaveArrays.java`) into a folder by themselves. The folder's name should be CSE114_PS4_<yourname>_<yourid>. So if your name is Alice Kim and your id is 12345678, the folder should be named 'CSE114_PS4_AliceKim_12345678.

2. Compress the folder and submit the zip file.
    a. On windows, do this by pressing the right-mouse button while hovering over the folder. Select 'Send to -> Compressed (zipped) folder'. The compressed folder will have the same name with a .zip extension. You will upload that file to the Brightspace dropbox for **Assignment4**
    b. On mac, move the mouse over the folder then right-click (or for single button mouse, use Control-click) and select **Compress**. There should now be a file with the same name and a .zip extension. You will upload that file to the Brightspace dropbox for **Assignment4**.

3. Navigate to the course brightspace site. Click **Assignments** in the left column menu. Click **Assignment4** in the content area. Under **ASSIGNMENT SUBMISSION**, click **Browse My Computer** next to **Attach Files.** Find the zip file and click it to upload it to the web site.