CS-1331

Homework 7

Due Friday, October 26th, 8PM.

ArrayWrapper.java

In this homework, you'll be wrapping an array in a class called ArrayWrapper. This class will allow us to perform some complex functions on our arrays. For example:

```
ArrayWrapper a = new ArrayWrapper(0, 1, 2, 3, 4, 5);
ArrayWrapper b = new ArrayWrapper(6, 7, 8, 9, 10);
a.concatenate(b);
System.out.println(a.toString()); // prints [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
System.out.println("AVG: " + a.mean()); // prints AVG: 5.0

b.insert(1, 45);
System.out.println(b.toString()); // prints [6, 45, 7, 8, 9, 10]
```

Core Requirements

Some further background on arrays: The positions in an array are called *indices*. The primitives or objects that reside at each index are known as *elements*.

These are the required methods for your `ArrayWrapper` class. You are **forbidden** from using any methods in the standard library that handle these tasks for you. If you aren't sure whether something is allowed, be sure to ask your TA.

- A constructor that takes in a *variable number of integers*, as shown above. This list of numbers should then be stored in a private variable of type `int[]`, and will be know as your backing array. This is the array that your ArrayWrapper will wrap. When we refer to "the array" in the rest of the guide, we will be referencing this one.
- A `toString()` method that takes in no parameters and returns a string of the array formatted like the following:
  - `[9, 4, 2, 1, 4]`
  - `If there are no elements: []`
- `int indexOf(int i)`
  - Return the index of the first occurrence of `i` in the array. If there is no occurrence, return `-1`.
- `int size()`
  - Return the number of elements in the array.
- `void insert(int index, int element)`

    - Insert element at the given index. If the index is beyond the array's size, then consider the insert to be the same as adding to the end of the array. If the index is below zero, then consider the insertion point to be at index zero. Remember, you will have to expand the array to fit the extra element.
- `void concatenate(ArrayWrapper)`
    - This method takes as a parameter another `ArrayWrapper`, and adds every element in that `ArrayWrapper` to the end of the backing array. You can see an example of this in the above sample code. This method doesn't return anything, but actually alters the internal array.
- `void addToEnd(int … n)`
    - This method will add some number of integers to the back of the array.
- `int removeIndex(int index)`
    - Remove the element at the given index, and return it. The cases where the index specified is out of bounds should be handled the same as it is in `insert(int, int)`. Remember, you will have to shrink the array after you remove the element. You don't have to worry about the case where the array has no elements in it.
- `int removeElement(int element)`
    - Remove the first occurence of the element from the array and return it. You don't have to worry about instances where the element is not in the array.
- `void clear()`
    - Removes all the elements in the array.
- `int get(int index)`
    - Get the element at the given index. You don't have to worry about instances where the index is out-of-bounds.
- `double mean()`
    - Calculate and return the average of all the elements. You don't need to worry about the case where the array is empty.
- `int[] getArray()`
    - Return the backing array.

AWTest.java Requirements

In a separate class, `AWTest.java`, you should test every single method in `ArrayWrapper.java`, to be sure they work. Your testing suite should use the `getArray()` method to look at the underlying array after you perform an operation, to be sure that it altered the array as intended.

Turn-in Procedure

Turn in the following files on T-Square. When you're ready, double-check that you have *submitted* and not just saved as draft.

- `ArrayWrapper.java`
- `AWTest.java`
- `Any other essential classes`

All .java files should have a descriptive javadoc comment.

Don't forget your collaboration statement. You should include a statement with every homework you submit, even if you worked alone.

Verify the Success of Your HW Turn-In

Practice "safe submission"! Verify that your HW files were truly submitted correctly, the upload was successful, and that the files compile and run. It is solely your responsibility to turn in your homework and practice this safe submission safeguard.

1. After uploading the files to T-Square you should receive an email from T-Square listing the names of the files that were uploaded and received. If you do not get the confirmation email almost immediately, something is wrong with your HW submission and/or your email. Even receiving the email does not guarantee that you turned in exactly what you intended.
2. After submitting the files to T-Square, return to the Assignment menu option and this homework. It should show the submitted files.
3. Download copies of your submitted files from the T-Square Assignment page placing them in a new folder.
4. Recompile and test those exact files.
5. This helps guard against a few things.
    a. It helps insure that you turn in the correct files.
    b. It helps you realize if you omit a file or files.**
    (If you do discover that you omitted a file, submit all of your files again, not just the missing one.)
    c. Helps find last minute causes of files not compiling and/or running.

**Note: Missing files will not be given any credit, and non-compiling homework solutions will receive few to zero points. Also recall that late homework (past the grace period of 2 am) will not be accepted regardless of excuse. Treat the due date with respect. The real due date and time is 8 pm Friday. Do not wait until the last minute!

Extra Credit:

You may only get a maximum of ten extra credit points.

`(+5): void shift(int a)`

Shift every element in the array to a new location that is `a` indices greater than its current index. Elements will wrap around if their new location would be out of bounds. You should support negative `a`'s as well.

(+5): `void sort()`

Sort the entire array into increasing numerical order. You must implement either QuickSort or MergeSort to do this.

(+5): `apply(Function)`

Implement a method that lets you apply an arbitrary function to your array, and then come up with at least 4 functions and test them in your AWTest.java. Anonymous inner classes are a good way to implement the functions themselves, but you may make them private inner classes as well.

All functions follow the interface (which you should include with your submission, if you choose to do this extra credit):

```
public interface Function {
     public int function(int i);
}
```

You will need to make a method that takes in a `Function`, and then for every element in the array, submit it to the function and set the value to what the function returns. For example, you can make a map function that returns (`i*i`). So, if I map that function to the ArrayWrapper containing `[0, 1, 2, 3, 4]`, it will then contain `[0, 1, 4, 9, 16]` after the `apply` method is called on the `ArrayWrapper`.