Project 2

Due Date: Friday 23 September 2022 by 11:59 PM

General Guidelines.

The instructions given below describe the required methods in each class. You may need to write additional methods or classes that will not be directly tested but may be necessary to complete the assignment.

In general, you are not allowed to import or use any additional classes in your code without explicit permission from your instructor!

Note: It is okay to use the Math class if you want.

Note on academic dishonesty: Please note that it is considered academic dishonesty to read anyone else's solution code, whether it is another student's code, code from a textbook, or something you found online. You MUST do your own work! It is also considered academic dishonesty to share your code with another student. Anyone who is found to have violated this policy will be subject to consequences according to the syllabus and university policy.

Project Overview

I've found that students who code in Java often use the ArrayList data structure as it is such a flexible structure with so many operations. Although there is nothing wrong with using an ArrayList in general, in this class we are very careful when and how we are allowed to use it. That is because some operations are relatively efficient while some are not, and to be a good programmer, you should really have a good understanding of how data structures are implemented so that you can make good, informed decisions.

An ArrayList is called an ArrayList because it is a list built with an array. That means that it is built as a growable structure from a fixed-size structure (an array).

To better understand how this affects efficiency and how some operations can be implemented in an efficient way, you are going to write your own ArrList data structure. Similar to Project 1, you must use the Array class as your implementation will be tested for efficiency based on how many array accesses you use.

Required Methods for ArrList.java

Method Signature	Description
`ArrList()`	constructor: default size of Array should be 10
`ArrList(int cap)`	constructor: starting Array size is <cap>
`void addLast(int num)`	adds a new element to the end of the list//should be O(1) except when resizing is necessary
`void addFirst(int num)`	adds a new element to the front of the list//should be O(1) except when resizing is necessary
`int get(int i)`	returns the element at index i (of the ArrList)//should be O(1)
`int indexOf(int num)`	return the ArrList index of the first occurrence of <num> or -1 if <num> is not in the list//should be O(N)
`boolean contains(int num)`	return true if the list contains <num> and false otherwise//should be O(N)
`isEmpty()`	return true if the list is empty and false otherwise//should be O(1)
`int lastIndexOf(int num)`	return the ArrList index of the last occurrence of <num> or -1 if <num> is not in the list//should be O(N)
`int removeFirst() throws EmptyListException`	remove and return the first element in the list; throw the exception if the list is empty//should be O(1) unless resizing is necessary

`int removeLast() throws EmptyListException`	`remove and return the last element in the list; throw the exception if the list is empty//should be O(1) unless resizing is necessary`
`int removeByIndex(int i) throws EmptyListException`	`remove and return the element at index i and close the gap; throw the exception if the list is empty//can be O(N) due to closing the gap but should be O(1) if there is no gap to close`
`boolean removeByValue(int num) throws EmptyListException`	`remove the first occurrence of element <num> from the list if it exists; close the gap; return true if the element was removed and false otherwise; throw the exception if the list is empty//should be O(N)`
`int set(int index, int num)`	`set the value at <index> to <num>//should be O(1)`
`int size()`	`return the number of elements in the list//should be O(1)`

Guidelines:

- Implement your solution in a class called *ArrList.java*
- You are not allowed to use any additional data structures–only the underlying Array (but this will need to be resized); (single variables are fine)
- Familiarize yourself with the Array class if you haven't already. There are many methods provided already, including a resizing method.
- Some of the operations above are difficult (if not impossible) to do very efficiently (i.e. inserting or removing from the middle of the list). However, adding/removing from the front or the back of the list can be done with reasonable efficiency if you follow these guidelines:
    - Resize your underlying array by multiplying it by some factor (not by adding a constant amount)--for example, multiply it by 2 when it gets full or cut it in half when it drops below ¼ full. This ensures that you aren't wasting too much space while still providing a reasonably efficient runtime

    (mathematical justification to come in Unit 3). Also, resize when you are trying to add an element to a full Array–that means you should check to see if the Array needs to be resized at the beginning of any "add" method.

- When removing an element, you should remove the element first, then check if the Array is less than ¼ full. In that case, resize to half the capacity *unless that would put the capacity below 10.*
- Some of these methods are required to "throw" an exception. The Exception code is provided for you, but if you are unfamiliar with exceptions in Java, you may want to look up how to use exceptions. Some of you are going to wonder if the test code is incorrect in how it is handling exceptions. It is not. It is handling the exceptions with try-catch blocks because it is expecting your method to "throw" the exception in the appropriate place.
- Use a wrap-around method for adding/removing from the front and back of the list.
    - This will require you to keep track of where the *front* and *back* elements of the list are.
    - This will also mean that the index of the ArrList may not be the same as the index for the underlying Array. All indexes in the ArrList API are ArrList indexes. You will need to go back and forth between those in your implementation.
    - Consider the example below and think about where the *front* and *back* of the list are in each case.

`addLast(5)`

5						

`addLast(6)`

5	6					

`addLast(7)`

5	6	7				

`addFirst(8)`

5	6	7				8

`addFirst(9)`

5	6	7			9	8

`addLast(10)`

5	6	7	10		9	8

`removeFirst()`

5	6	7	10			8

`removeLast()`

5	6	7			9	8

Testing & Submission Procedure.

This time, test code is provided for you, but keep in mind that the test code may not be exhaustive and catch all possible errors. The test code also compares your access counts to the expected ones, so you should get a good idea of whether or not your efficiency is okay. Again, you should not rely solely on the test cases as your code will also be checked manually. Make sure your code runs with the test code *as it is*. Since you won't be submitting the test code file, your submission cannot depend on any changes you make to that file. The same is true for the Array.java file. You can change it for yourself for testing purposes, but we will use the original one when testing, so make sure your code works with that.

Your code must compile and run on lectura, and it is up to you to check this before submitting.

To test your code on lectura:

- Transfer all the files (including test files) to lectura.
- Run *javac *.java* to compile all the java files.
- Run *java <filename>* to run a specific java file.

After you are confident that your code compiles and runs properly on lectura, you can submit the required files using the following **turnin** command. Please do not submit any other files than the ones that are listed.

```
turnin cs345p2 ArrList.java
```

Upon successful submission, you will see this message:

```
Turning in:
      ArrList.java -- ok
All done.
```

Note: Only properly submitted projects are graded! No projects will be accepted by email or in any other way except as described in this handout.

Grading.

Auto-grading

Part	**Total Points**	**Details**
ArrList.java	60	12 tests total 5 points each (3 for correct results + 2 for access counts)

Your score will be determined by the tests we do on your code minus any deductions that are applied according to the following list of possible deductions.

- Bad Coding Style (up to 10 points)
- Not following directions (up to 100% of the points depending on the situation)
- Late Submission (10 points per day)
- Inefficient code–up to 10 points (may be separate from the access count tests in case your code is doing something unnecessarily efficient but still passing the auto-graded tests.)

If your code does not compile/run on lectura, the TA may give you an automatic 0. If you find that this is due to a small error that is easily fixed, you may fix the code and submit a regrade request to the TA, but projects like this will still get a deduction of at least 15 points.

For regrades, you should contact the TAs directly (email all of them), and you should do so within 72 hours of the grade being released on D2L.