COMP 2150 – Spring 2014
Homework 6: Array Lists
(Posted Mar. 31 – Due Apr. 7 by 12:40 pm)

Remember that as stated on the syllabus, this assignment should be an individual effort (contact me or visit the Computer Science Learning Center, www.cs.memphis.edu/cslc, if you need help).

Submission: Please zip your source code into a single file (you can zip the entire project folder if you're using BlueJ) and upload it to the proper folder in the eCourseware dropbox at https://elearn.memphis.edu. The dropbox will cut off all submissions after the indicated deadline, so don't wait until the very last minute to submit your work!

Grading: This assignment will be graded by the TA Vincent Nkawu (venkawu@memphis.edu). If you have questions or concerns about your grade, please contact him first. I'll be happy to look over your assignment myself if he's not able to resolve the situation to your satisfaction. Also remember that as stated on the syllabus, all submissions MUST compile and run to receive credit. The TA does not have time to find and correct your syntax errors!

Coding Style: Be sure to follow the good coding practices that were discussed in COMP 1900:

- Use descriptive variable and method names. Avoid using single-character names unless it's very obvious what the variable's being used for (like a loop counter).
- Follow standard Java programming conventions for **variableAndMethodNames**, **ClassNames**, **CONSTANT_NAMES**.
- Consistently indent your code.
- Use comments judiciously. Every source code file should include a comment block at the top that lists your name and the assignment number. Every non-trivial method should have a comment preceding it that specifies what the method does, what its parameters are, and what it returns. For simple methods like accessors or mutators this isn't necessary, but it doesn't hurt to put in a simple comment like `// accessor methods`. I also expect to see comments throughout your code explaining what actions are being taken!

Add the following methods to the **ArrayListGeneric<E>** class that we wrote during lecture (code posted on Mar. 31). You may call the existing methods in **ArrayListGeneric<E>** if you want, but do not use anything from the built-in **java.util.ArrayList** class!

1. **(2 pts)** Write a new method named **clear()** that removes all elements from the calling list, resetting its capacity to some default value.

2. **(5 pts)** Write a new method named **add(int index, E newItem)** that adds the specified **newItem** to the specified **index** in the calling list, shifting all subsequent elements up by one index. Be sure to reallocate the data array if necessary. This method should throw an **IndexOutOfBoundsException** if an invalid index is supplied.

3. **(5 pts)** Write a new method named **slice(int beginIndex, int endIndex)** that returns a new **ArrayListGeneric<E>** object containing the elements of the calling list between **beginIndex** (inclusive) and **endIndex** (exclusive). The calling list should not be modified. This method should throw an **IndexOutOfBoundsException** if an invalid index is supplied, or if **beginIndex** is not at least 1 less than **endIndex**.

   Example: Let **list1** be an **ArrayListGeneric<Integer>** object containing the elements {1, 2, 3, 3, 6, 2, 2, 3, 1, 4}. Then calling **list1.slice(4, 7)** should return a new **ArrayListGeneric<Integer>** object containing the values at indices 4, 5, and 6. The returned list would contain the elements {6, 2, 2}.

4. **(4 pts)** Write a new method named **addAll(ArrayListGeneric<E> anotherList)** that adds all of the elements in **anotherList** to the back of the calling list. Be sure to reallocate the data array if necessary. **anotherList** should not be modified.

5. **(5 pts)** The remove method that we wrote in class never alters the capacity of the **ArrayListGeneric<E>** object. Once the capacity is increased by the **add** method, it remains that way until increasing again. Write a new method named **removeAndTrim(int index)** that removes and returns the element at the specified **index**. In addition, it should perform a "trim" operation on the data array: if the removal results in a new size that is less than half of the capacity, the capacity should be halved.

   Example: Suppose you have an **ArrayListGeneric<E>** object with a size of 5 and a capacity of 8. Calling **removeAndTrim** once would make the size 4. Because 4 is not less than half of 8, **removeAndTrim** would take no further action. However, if you called **removeAndTrim** again, the new size would be 3. Because 3 is less than half of 8, the method would also reduce the capacity to 4.

6. **(4 pts)** Write a new method named **lastIndexOf(E someData)** that returns the highest index in the calling list that contains **someData**, or -1 if the list does not contain that data. Assume that the data stored in the list is from a class that has an **equals** method defined.

   Example: Let **list1** be an **ArrayListGeneric<Integer>** object containing the elements {1, 2, 3, 3, 6, 2, 2, 3, 1, 4}. Then calling **list1.lastIndexOf(1)** should return 8.

As usual, be sure to test your methods thoroughly!