Programming Assignment #1

75 points

Due Date/Time

Your program is due uploaded to edoras at 11:59 p.m. Monday, February 18th. An exact printed copy of your program will be turned in at the beginning of class, Tuesday, February 19th 0800.

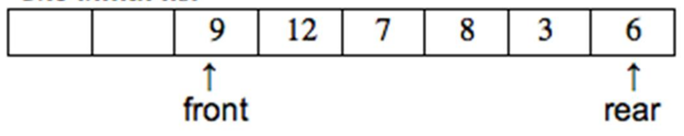
The Assignment

For this assignment, you will create an implementation of the ArrayListADT interface (below). A 'list' is a sequence of values, which may include duplicates. The ordering of the items in the list is not specified but does matter and is in fact problem dependent. No insertion ever occurs at an arbitrary location. When an item is removed from the list, the ordering of the remaining elements in the list is unchanged. There may not be any empty or unused cells between the front and rear of the list.

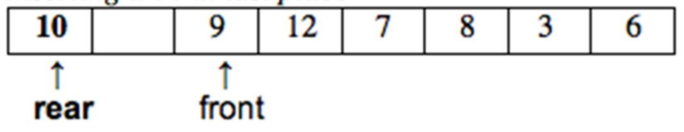
Your implementation of the ArrayListADT offers a meaningful improvement over basic arrays for list-based operations. Inserting or removing an element at index [0] is a time-consuming operation for arrays. If you wish to insert an element at index [0] and the array is not empty, then you must shift all the existing elements out of the way before the insertion can occur. Similarly, if you remove the element at index [0] from a non-empty array, you must shift elements down to 'fill in the hole'. Your implementation must be able to insert or remove elements from either end of the array in constant time, with no shifting of elements necessary. A circular array strategy offers this capability. *"Circular"* is an abstraction; the underlying array doesn't form a circle but rather will be a standard linear array.

A solution to this problem can be found if you abandon the notion that the first element in the list must be at index [0]. Instead, you maintain a class level variables that hold the index of the 'front' and 'rear' of the list. The front and rear indices move independently, allowing insertion and deletion to happen without shifting anything.

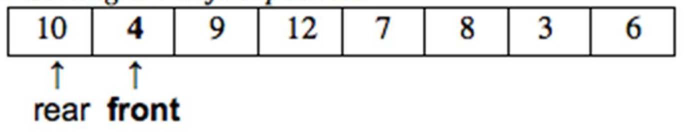
The initial list



Inserting a 10 in last place



Inserting a 4 in first position



This list (in order) is: 4, 9, 12, 7, 8, 3, 6, 10. The 'front' and 'rear' indices simply wrap around when they run off the end.

We want to segregate our data structures and separate them from any application programs. Accordingly, you must place all data structures in a package named `data_structures`.
Your `ArrayLinearList` class must implement the `LinearListADT` interface. Your project will consist of exactly the following two files, both of which **must** be in a `data_structures/` package:

- `LinearListADT.java` The linear list interface (provided below)
- `ArrayLinearList.java` Your implementation of the interface

Both of the above files must go in package `data_structures`. Any driver/tester programs will go in the level above the `data_structures` subdirectory. *Sample tester programs will be provided.*

IMPORTANT: The package organization is of critical importance. If your project fails to compile during the grading process due to package errors, your grade for this project will be zero.

Submitting Your Assignment

You must use the `handin` directory at the top level of your class account. For submission of your projects, the source code files for all projects this semester will be placed within this `handin` directory tree. Each project must go in a separate directory within `handin`. Since this is programming assignment #1, your source code (just `ArrayLinearList.java`) must go in `handin/prog1`. Please review the Program Submission Guidelines page carefully. You are responsible for following the directions given on this page.

Before the due date, copy your `ArrayLinearList.java` source code file into your `handin/prog1` subdirectory. Do not put the `LinearListADT.java` interface in `handin/prog1`. I will use my copy to compile your program.

IMPORTANT: Do not create any data_structures folder anywhere within `handin/;`
your `ArrayLinearList.java` **file goes directly in the** `handin/prog1` **folder**.

You will submit a hardcopy printout of the `ArrayLinearList.java` file you have written at the beginning of class on the due date.

IMPORTANT: Once you have submitted the assignment, you may NOT resubmit. Under no circumstances may you submit multiple hard copy of programming assignments. ONE submission only. There will be a *substantial* grade penalty if I find that you have submitted more than one printout of your source code.

The LinearListADT interface:

```
package data_structures;

import java.util.Iterator;
import java.util.NoSuchElementException;


public interface LinearListADT<E> extends Iterable<E> {

    public static final int DEFAULT_MAX_CAPACITY = 100;

    /* Outputs "Front: indexFront Rear: indexRear"
     */
    public void ends();

    /* Adds the Object obj to the beginning of list and returns true if the list is not
    *  full.
    *  returns false and aborts the insertion if the list is full.
    */
    public boolean addFirst(E obj);

    /* Adds the Object obj to the end of list and returns true if the list is not full.
    *  returns false and aborts the insertion if the list is full.
    */
    public boolean addLast(E obj);

    /* Removes and returns the parameter object obj in first position in list if the list
    *  is not empty, null if the list is empty.
    */
    public E removeFirst();

    /* Removes and returns the parameter object obj in last position in list if the list
    *  is not empty, null if the list is empty.
    */
    public E removeLast();

    /* Removes and returns the parameter object obj from the list if the list contains
    *  it, null otherwise. The ordering of the list is preserved.  The list may contain
    *  duplicate elements.  This method removes and returns the first matching element
    *  found when traversing the list from first position.
    *  Note that you may have to shift elements to fill in the slot where the deleted
    *  element was located.
    */
    public E remove(E obj);

    /* Returns the first element in the list, null if the list is empty.
    *  The list is not modified.
    */
    public E peekFirst();

    /* Returns the last element in the list, null if the list is empty.
    *  The list is not modified.
    */
    public E peekLast();

    /* Returns true if the parameter object obj is in the list, false otherwise.
    *  The list is not modified.
    */
    public boolean contains(E obj);

    /* Returns the element matching obj if it is in the list, null otherwise.
    *  In the case of duplicates, this method returns the element closest to front.
    *  The list is not modified.
    */
    public E find(E obj);

    /* The list is returned to an empty state.
    */
    public void clear();

    /* Returns true if the list is empty, otherwise false
    */
    public boolean isEmpty();

    /* Returns true if the list is full, otherwise false
    */
    public boolean isFull();

    /* Returns the number of Objects currently in the list.
    */
    public int size();

    /* Returns an Iterator of the values in the list, presented in
    *  the same order as the underlying order of the list. (front first, rear last)
    */
    public Iterator<E> iterator();

}
```

Additional Details

- The ordering of the elements in the list is user defined. Thus, you must never alter the ordering of items in the list internally. For instance, a call to the method `remove(E obj)` will leave a 'hole' in the array if it is in the middle of the list. It would be simple to just copy the last element to the location of the removed item and decrease the size. But this would alter the order of elements. This is forbidden.
- Your `ArrayLinearList` class will have two constructors, one that takes a `maxCapacity` parameter, and one with no arguments that creates an array of `DEFAULT_MAX_CAPACITY`.
- Since this is an array based implementation, there is a maximum capacity which cannot be exceeded. If an insertion into a full array is attempted, then the insertion method returns false and aborts the insertion. An exception is **NOT** thrown in such cases.
- The `ArrayLinearList.java` file specified in this assignment **must** have the exact public names and signatures as given in the interface.
- You must not make any changes to the `LinearListADT` interface; I will use my copy to compile and run your program.
- You will submit only your `ArrayLinearList.java` file; do not submit a copy of the interface provided, nor any drivers/testers that you have written.
- You may import only classes needed for the Iterators. You may use any class in `java.lang` (the default package). You may **not** use any data structure or class in `java.util` other than those specified. You will need `java.util.Iterator`, and `java.util.NoSuchElementException`.
- Every class file must begin with your name and edoras class account number.
- Each method should be as efficient as possible. For example, your `size()` method should not loop through the array and count the elements.

- Your project must compile and run on edoras to receive any credit for the assignment. For grading, your file will be copied from your `handin/prog1` folder to my account. The project layout will be recreated, and then compiled and run. Watch your package structure! Any project that fails to compile will receive a zero.

  Here is an example of the directory structure for this program

  ```
  [cssc0000@edoras ~]$ ls handin
  prog1
  [cssc0000@edoras ~]$ ls handin/prog1
  ArrayLinearList.java
  ```

Early Programs:

Early programs will be accepted with a bonus of 5% per day for up to two days before the due date. The submission date will be determined by the timestamp of your file on edoras. Your printout is due at the next class session after your program file is placed in `handin`. For example, this program is due Monday at essentially midnight. If your program timestamp is any time Saturday, February 16th, your will receive a 15 point bonus, and if your program timestamp is any time Sunday, February 17th, your will receive a 7.5 point bonus.

IMPORTANT: Late programs are NOT accepted. Late programs will receive a score of zero. Failure to turn in a hardcopy of your program the class session after the 11:59 p.m. due date is considered LATE and your program will receive a score of zero.

The goal here is not to be needlessly punitive. Rather, it is to enforce the simple reality that all projects will have due dates, both in school and in industry. Many professors accept late work with a penalty—which I did in the past. Students simply started late (and often *too* late) on their projects. Best practice is to turn in your work on time, every time.

Cheating Policy:

There is a **zero-tolerance policy on cheating** in this course. You are expected to complete all programming assignments on your own. Collaboration with other students in the course is not permitted. You may discuss ideas or solutions in general terms with other students, but you must not exchange code. Nor may you copy code from former CS310 students, nor the Internet. You must be able to produce code on your own--otherwise no one will ever hire you. Remember that you can get help from the TAs (GMCS 429) or from me. This is not cheating but is in fact encouraged.

I will examine your code carefully. Anyone caught cheating on a programming assignment or on an exam will receive an "F" in the course, and a referral to Judicial Procedures.

Checking that everything works on edoras

Use a sandbox directory, like this:

~/sandbox/prog1/

```
Driver.java
```

~/sandbox/prog1/data_structures

```
LinearListADT.java

ArrayLinearList.java
```

Compile from ~/sandbox/prog1 directory:

```
[cssc0000@edoras prog1]$ javac Driver.java

[cssc0000@edoras prog1]$ java Driver
```