COMP251 - Assignment1
Deadline: Oct 18, 2021

Goal:

In this assignment you will design your first ADT and implement it. You need to respect the ***abstraction*** principle in your implementation.

Problem:

In this assignment you will build a “Double Ended Queue” using an array (array based implementation).
The ***double ended queue*** is an abstract data type which is an extended form of the regular queue. Here is a definition from JAVA API:

> *A linear collection that supports element insertion and removal at both ends (front and rear). The name deque is short for "double ended queue" and is usually pronounced "deck".*

Specifically, any ***deque*** implementation must have exactly the following operations:

- Add an object (of type `AnyType`) to the front of the deque:
  `public void addFirst(AnyType item)`
- Add an object (of type `AnyType`) to the rear of the deque:
  `public void addLast(AnyType item)`
- Remove and return the item at the front of the deque. If no such item exists, throw an exception
  `public AnyType removeFirst()`
- Remove and return the item at the rear of the deque. If no such item exists, throw an exception:
  `public AnyType removeLast()`
- Check if the deque is empty (return true if yes):
  `public boolean isEmpty()`

- Return the number of items in the deque:
  `public int size()`
- Clear the deque:
  `public void clear()`
- Return the item at the front of the deque. If no such item exists, throw an exception:
  `public AnyType getFirst()`
- Return the item at the rear of the deque. If no such item exists, throw an exception:
  `public AnyType getLast()`
- Return the item at the given index where 0 is the front, 1 is the next item, and so forth. If no such index exists, throw an exception, (Note: you should not alter the deque):
  `public AnyType get(int index)`

Your class should accept any generic type. You may have a look at a lecture note about generic types I posted on blackboard.

Your operations are subject to the following rules:

- All four operations on **`add`** and **`remove`** must not involve any looping or recursion. Each operation must take “constant time”, except during expanding the array.
- **`size`**, **`get`**, **`getFirst`** and **`getLast`** operations must take “constant time”.

Array Based Deque:

You will implement Deque ADT using array as the core data structure. The class you should implement is called `ArrayBasedDeque` (like `ArrayBasedList`).

In addition, your implementation should include:

- Create an empty deque (constructor of the class):
  `public ArrayBasedDeque()`
- Return a string representation of this collection (in order, from the front to the rear):
  `public String toString()`

You may add any private helper classes or methods in `ArrayBasedDeque` if you find it necessary.

You should treat your array as a **circular array** for this exercise. In other words, if your front index is at position zero, and you `addFirst`, the front index should loop back around to the end of the array (so the new front item in the deque will be the last item in the underlying array). See the lecture notes on array based implementation of queue on blackboard (**Week 4, ADTQueue**).

Dynamic Structure

ArrayBasedDeque should be a dynamic structure (i.e it can grow if needed). See the lecture notes on array based implementation of queue on blackboard for tips on expanding the array.

Test

Testing is an important part of code writing in industry and academia. It is an essential skill that can prevent monetary loss and hazardous bugs in industry, or in your case, losing points.
A sample test class is given to you for this assignment. You can add more lines (for testing your code).

Requirements:

Make sure your final submission passes all the following requierments to get the full mark:

- Your ADT is generic.
- Use array as the core data structure (You are **not allowed** to use linked list)
- Your ADT is a dynamic structure (it will be expanded whenever needed, use circular array)
- All 12 methods listed in the previous section are implemented, and the signature of the methods are as listed. Two files called `deque.java` and `ArrayBasedDeque.java` are given with this instruction which include the signature of all methods. (Note: you can add more methods to class `ArrayBasedDeque.java` if you need)
- Time complexity of the methods should follow Table 1.

- Your code should pass all tests given in `test.java` (Note: you can add more tests, but **do not** change or remove the given tests methods.)

Table 1

Method	Time Complexity
`get()`	O(1)
`getFirst()`	O(1)
`getLast()`	O(1)
`removeFirst()`	O(1)
`addFirst()`	O(1)
`removeLast()`	O(1)
`addLast()`	O(1)
`clear()`	O(1)

Delivery

- Please record a short video in which you explain your work. Run through your code and show how it works. The design of your ADT and all the methods (just a brief description, algorithms and data structure you use). If you could not finish all parts of the assignment, mention them in your video.
- **Important**: post your video on youtube and set the **publish time to be three days after the deadline** of the corresponding assignment.
- Submit: the **source code** (.java files), and a **readme** file which you should put the link to your video on youtube.
- **Important**: I'll randomly choose some students for **oral presentation**, and they will be notified by email. If you get selected, I'll schedule a time slot and **you should present your assignment to me and answer some questions.**

Important Remarks[1]

It's very important that you make sure your program compiles without any problem (you may get 0 if your program didn't compile)

For this project, you must work alone!

By alone rule: All code that you submit should be written by you alone, except for small snippets that solve tiny subproblems (of course you are allowed to use the source codes we discussed in class or I post on blackboard).

Do Not Possess or Share Code: Before you've submitted your final work for a project, you should never be in possession of solution code that you did not write. You will be equally culpable if you distribute such code to other students or future students of COMP251 (within reason). DO NOT GIVE ANYONE YOUR CODE – EVEN IF THEY ARE DESPERATELY ASKING. DO NOT POST SOLUTIONS TO PROJECTS ONLINE (on GitHub or anywhere else)! If you're not sure what you're doing is OK, please ask.

Permitted:

- Discussion of approaches for solving a problem.
- Giving away or receiving significant conceptual ideas towards a problem solution. Such help should be cited as comments in your code. For the sake of other's learning experience, we ask that you try not to give away anything juicy, and instead try to lead people to such solutions.
- Discussion of specific syntax issues and bugs in your code.
- Using small snippets of code that you find online for solving tiny problems (e.g. googling "uppercase string java" may lead you to some sample code that you copy and paste into your solution). Such usages should be cited as comments in your hw, lab, and especially project code!

Absolutely Forbidden:

- Possessing another student's project code in any form before a final deadline, be it electronic or on paper. This includes the situation where you're trying to help someone debug. Distributing such code is equally forbidden.
- Possessing project solution code that you did not write yourself (from online (e.g. GitHub), staff solution code found somewhere on a server it should not have

[1] The description of rules about code submission are from UBerkeley.

been, etc.) before a final deadline. Distributing such code is equally forbidden.

Submission

Create a directory called `src` and copy all your source files and any readme file in it, zip it and submit it to the blackboard.