# ASSIGNMENT 1: FABRIC BREAKUP & GROWABLE DEQUES

**Goal:** The goal of this assignment is to get some practice with stack and queue implementation. One of the goals is also to explore the use of these data structures in the context of an application. In Part (a) of the assignment you have to implement the growable array version of the deque data structure. In Part (b) of the assignment you have to solve a puzzle.

# ASSIGNMENT 1(A): GROWABLE DEQUES

**Problem Statement:** Your first task is to implement a generic ArrayDeque class. As is to be expected, ArrayDeque must implement the Deque ADT we provided in class. The exact interface will be as follows:

```
public interface DequeInterface {

   public void insertFirst(Object o);

   public void insertLast(Object o);

   public Object removeFirst() throws EmptyDequeException;

   public Object removeLast() throws EmptyDequeException;

   public Object first() throws EmptyDequeException;

   public Object last() throws EmptyDequeException;

   public int size();

   public boolean isEmpty();

   public String toString();

}
```

Your implementation must be done in terms of an array that grows by doubling as needed. Your initial array must have a length of one slot only!

Your implementation must support all Deque operations except insertion in (worst-case) constant time; insertion can take longer every now and then (when you need to grow the array), but overall all insertion operations must be constant amortized time as discussed in lecture.

You should provide a toString method in addition to the methods required by the Deque interface. A new Deque into which 1, 2, and 3 were inserted using insertLast() should return "[1, 2, 3]" while an empty Deque should print as "[]".

Ensure that the version of your code you hand in does not produce any extraneous debugging output anymore!

## What is being provided?

Your assignment packet contains two folders with the starter code to each of the assignment parts. Part_A of the packet contains four java files –

1. DequeInterface.java: File defining the interface. (You should not modify this file)
2. EmptyDequeException.java: File defines a custom exception (You should not modify this file)
3. ArrayDeque.java: Here you have to implement all the functions in ArrayDeque class.

4. ArrayDequeTester.java: File with the main function that uses all the functions that you will implement in ArrayDeque class. If you run this code without any changes, it raises *"java.lang.UnsupportedOperationException: Not implemented yet"* (You should not modify this file)

## How to run?

Assignment packet Part_A contains a Makefile with the necessary commands to run the program. If you are unable to run the makefile, just copy-paste the commands in the makefile all: section sequentially in your shell.

# ASSIGNMENT 1(B): FABRIC BREAKUP

**Problem Statement:** You just had a breakup and your ex has dumped all your shirts in a big heap in the center of your tiny hostel room. And man! These are way too many shirts. No wonder (s)he was fed up helping you choose the best among them for every date :).

Anyways, whenever you find time, you start arranging the shirts from the big heap one by one. We will call the arranged shirts (that are no longer in the heap) as good shirts. Since you have limited space in your hostel room, you arrange the good shirts in a big pile (tower) on a side.

But, of course, every now and then you also want to party with friends to get over your ex. And at any time you go out, you wish to wear your most favorite shirt from among the good shirts in the pile. Unfortunately, you are clumsy! So, in trying to take out that chosen shirt, all shirts on top of that shirt get toppled and get mixed in the big heap again. Your goal is to programmatically find how many shirts get toppled every time you party with friends.

**Input:** The input to your software is a sequence of operations. The first line is the total number of operations N. And every successive line is operation #i with i ranging from 1 to N. Each operation is action id 1 or 2. Id 1 means a move from heap to pile. Id 2 means party with friends. For move action there is one additional number, a score which represents how much you like that shirt. It is an integer between 0 and 10000, where 10000 means max liking and 0 means hate it! As an example suppose you perform 8 operations: move shirt with score 400, move shirt with score 500, party, move shirt with score 300, party, party, move shirt with score 20, party. The input will be:

8

1 1 400

2 1 500

3 2

4 1 300

5 2

6 2

7 1 20

8 2

**Output:** The output is, for each operation #, which is a party operation, return the number of shirts that get toppled back into the heap. In case there are multiple shirts with the same best score in the pile, you always pick that one which topples the least number of shirts. Also, if you try to party without any shirts in the pile you should output -1 to indicate error. In the example above, the output will be:

3 0

5 1

6 -1

8 0

That is, when you party first, you pick a shirt with score 500 and, since it's at top, no other shirt gets toppled. Second time, you topple one shirt and pick the shirt with score 400 to wear. You are out of shirts the third time you try to party the third time, and so on.

**Code:** Your code must compile and run on Java 8. Your code will be run with a parameter inputfile, which will have the input, and your code should output one line per party action as per output format. Your code should also have a stack implementation (which should be the ONLY manner in which your code for the above problem should use a stack. Do NOT use any JAVA libraries which already implement any form of set or sequence). The stack interface should be:

```
public interface Stack {
    public void push (Object O);
    public Object pop() throws EmptyDequeException;
    public boolean isEmpty();
    public Object top() throws EmptyDequeException;
    public int size();
}
```

Your implementation of the stack interface MUST use the growable deques implementation stated in part (A) of the assignment. Do NOT use any other data-structure like arrays or any other JAVA library to implement the stack interface.

**Desiderata:** Each operation in the input must be processed in constant time.

## What is being provided?

Part_B of your assignment packet contains five java files –

1. StackInterface.java: File defining the interface. (You should not modify this file)
2. EmptyStackException.java: File defines a custom exception (You should not modify this file)
3. Stack.java: Here you have to implement all the functions in Stack class.
4. StackTester.java: File with the main function that tests all the stack functions you will implement. If you run this code without any changes, it raises *"java.lang.UnsupportedOperationException: Not implemented yet"* (You should not modify this file)
5. FabricBreakup.java: File with the main function to implement the program to solve this puzzle.

Your assignment packet also contains a formatChecker.jar, sampleInputFile.txt, and sampleOutputFile.txt. Running formatChecker.jar with arguments inputfile.txt and output.txt double checks whether your code satisfies the input-output requirements of the assignment. Note that it does not check the correctness of your code but just confirms the input / output format. Usage -

java -jar formatChecker.jar sampleInputFile.txt sampleOutputFile.txt

## How to run?

Assignment packet Part_B contains a Makefile with the necessary commands to run the program. If you are unable to run the makefile, just copy-paste the commands in the makefile all: section sequentially in your shell.

## Assignment Folder Structure

Assignment packet follows a certain folder structure, so here we provide some explanation for that. Within the resources directory you should see 3 folders: Part_A, Part_B and classes. Once you run makefile present in any of the part folders, all .class files are created in the third folder called classes. This is done, so that Part_B can access your implementation of ArrayDeque to implement your stack. The -d and -cp flags in javac and java commands are used for that purpose. You should read more about them for a better understanding.

## What to submit?

1. You need to submit a zip file <your-entry-number>.zip which contains your code, structured as mentioned in the subsequent points.
2. Your submission upon extraction, should yield a folder named <your-entry-number>.
3. This folder should contain 2 folders named "part_A" and "part_B", and a "writeup.txt" file.
4. The folder "part_A" contains the .java files corresponding to your implementation of "Growable Deques".
5. The folder "part_B" contains the .java files corresponding to your implementation of "Fabric Breakup & Stack".
6. The "writeup.txt" should have a line that lists names of all students you discussed/collaborated with (see guidelines on collaboration vs. cheating on the course home page). If you never discussed your assignment with anybody, write "None". After this you are welcome to write any comments about your submission, though this is not necessary.

7. Eg. If your entry number is 2016CS50393, then you need to submit a zip file 2016CS50393.zip. This zip file on extraction gives a directory structure as follows:

```
2016CS50393
 |
 -----part_A
 |    |
 |    -- .java files for the Growable Deques problem
 |
 -----part_B
 |    |
 |    -- .java files for Fabric Breakup & Stack problem
 |
 ----- writeup.txt
```

8. Please compress your directory structure into a .zip file only.
9. Please submit the .java files only.
10. You will be penalized for any submissions that do not conform to the above requirements.

## Evaluation Criteria

The assignment is worth 6 points. Your code will be autograded against a series of tests. 4 points are for the autograding and 2 points are for demo performance with the TA.

## What is allowed? What is not?

1. This is an individual assignment.
2. Your code must be your own. You are not to take guidance from any general purpose code or problem specific code meant to solve these or related problems.
3. You are not allowed to use built-in (or anyone else's) implementations of stacks, queues, and other basic data structures. A key aspect of the course is to have you learn how to implement these data structures.
4. You should develop your algorithm using your own efforts. You should not Google search for direct solutions to this assignment. However, you are welcome to Google search for generic Java-related syntax.
5. You must not discuss this assignment with anyone outside the class. **Make sure you mention the names in your write-up in case you discuss with anyone from within the class.** Please read academic integrity guidelines on the course home page and follow them carefully.
6. Your submitted code will be automatically evaluated against another set of benchmark problems. You get a significant penalty if your output is not automatically parsable and does not follow input-guidelines.
7. We will run plagiarism detection software. Anyone found guilty will be awarded a suitable penalty as per IIT rules.