Data Structures and Algorithms

Stacks and Queues

Amilcar Aponte

amilcar@cct.ie

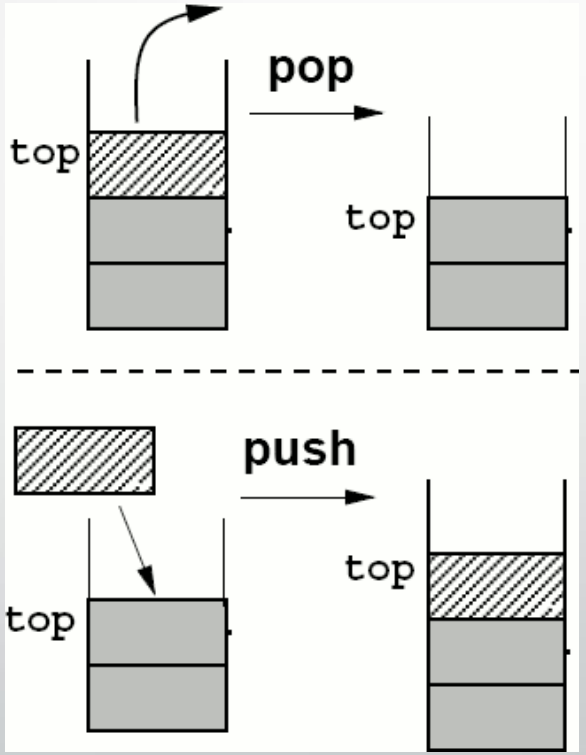
Today's Plan

- LIFO (Stacks)
- FIFO (Queues)

Stacks

- A stack is a classic collection used to help to solve many types of problems
- A stack is a linear collection whose elements are added in a last in, first out (LIFO) manner
- This means that the last element to be put on a stack is the first one to be removed
- Think of a stack of books, where you add and remove from the top, but can't reach into the middle

Stacks



Stack Methods

- Some collections use particular terms for their operations. But this are some classic ones

Operation	Description
Push	Adds an entry to the top of the stack
Pop	Removes an entry from the top of the stack
Peek	Looks at the entry at the top of the stack without removing it
isEmpty	Returns a boolean indicating if the stack is empty
size	Returns the number of items in the stack

Example

Method	Return Value	Stack Contents
push(5)	–	(5)
push(3)	–	(5, 3)
size( )	2	(5, 3)
pop( )	3	(5)
isEmpty( )	false	(5)
pop( )	5	( )
isEmpty( )	true	( )
pop( )	null	( )
push(7)	–	(7)
push(9)	–	(7, 9)
top( )	9	(7, 9)
push(4)	–	(7, 9, 4)
size( )	3	(7, 9, 4)
pop( )	4	(7, 9)
push(6)	–	(7, 9, 6)
push(8)	–	(7, 9, 6, 8)
pop( )	8	(7, 9, 6)

(bottom → top)

http://www.cs.armstrong.edu/liang/animation/animation.html

Typical applications for Stacks

- Direct applications
  - Undo sequence in a text editor
  - Chain of method calls in the Java Virtual Machine
- Indirect applications
  - Auxiliary data structure for algorithms
  - Component of other data structures

Stacks in Java

- Of course Java has a Stack built-in that we can use
  - `Stack<String> myStack = new Stack<>();`
- Let's give it a go

Stacks in Java

- You'll notice that the object has some other methods that are available for you
- But the most important ones are there

Let’s write our own

- What we want to do at this point is to go a bit further and write our own stack
- The whole point is that we get an idea of how this structure works in the background

Array-Based Stack

- Give it a go
- Let's create a class called "MyOwnStack" that implements the interface that I'm going to give you
- This class will have an underlying array where the data is saved.

- Reminder: Interface is the set of methods that your class must have. You cannot modify the interface, and you have to write some code in all of the methods that I give you.
- Reminder 2: Generics are classes/interfaces written for any type of objects

Array-Based Stack

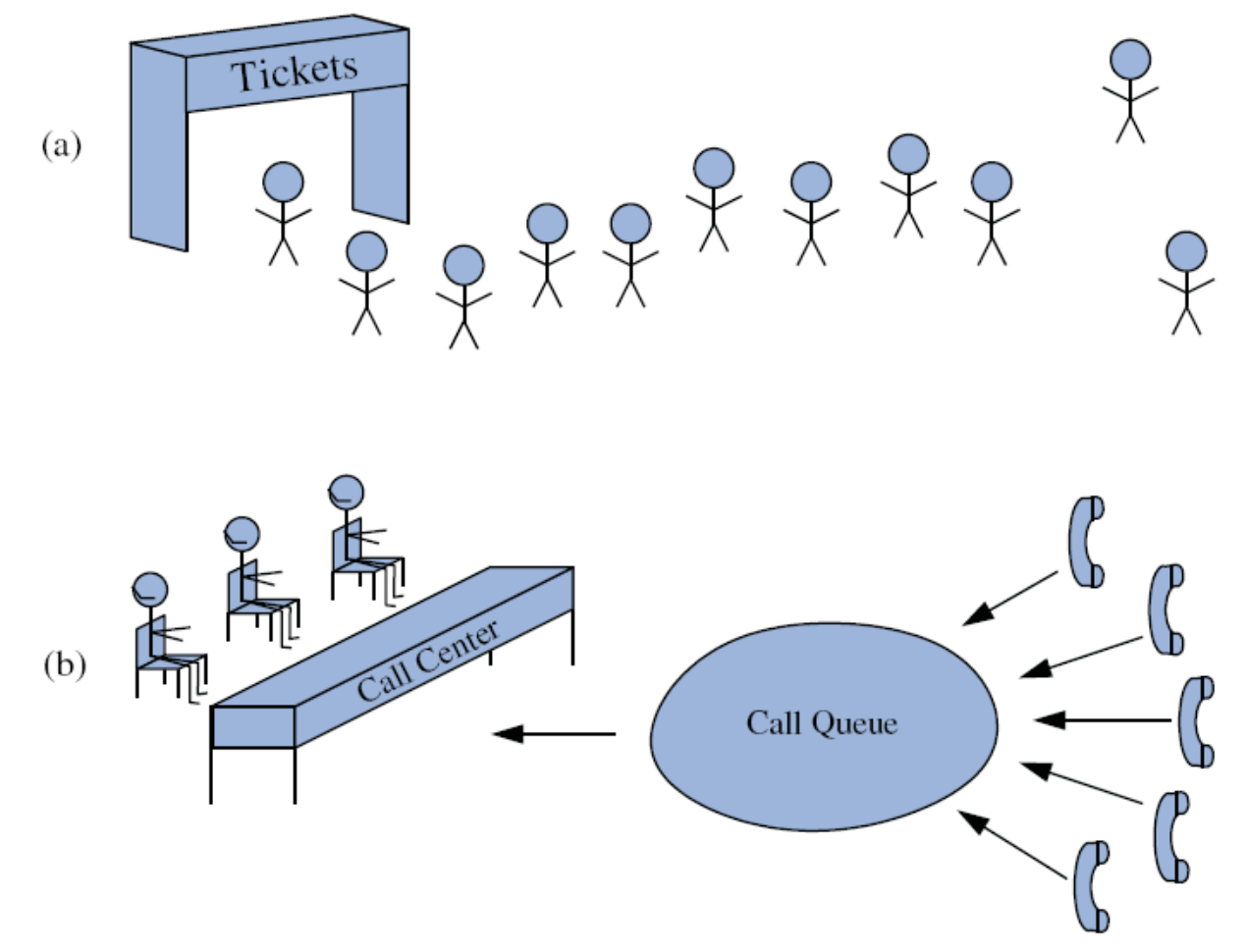
Array-Based Stack – Some Key Points

- We need to keep track of
  - Number of elements in the stack
  - Index of the top element
  - Conversions of types
  - Size of the underlying array
- You could also return an exception instead of a null object if the array is empty

Queues

- The Queue ADT stores arbitrary objects
- Insertions and deletions follow the first-in first-out scheme
- Insertions are at the rear of the queue and removals are at the front of the queue

Queues



Queue Methods

- Some collections use particular terms for their operations. But this are some classic ones

Operation	Description
Enqueue	Adds an entry to the back of the queue
Dequeue	Removes an entry from the front of the queue
First	Looks at the entry at the front of the queue without removing it
Last	Looks at the entry at the back of the queue without removing it
size	Returns the number of items in the queue
isEmpty	Returns a boolean indicating if the queue is empty

Example

Operation	***Return Value***	***Queue*** *(back → Front)*
enqueue(5)	–	(5)
enqueue(3)	–	(3, 5)
dequeue()	*5*	(3)
enqueue(7)	–	(7, 3)
dequeue()	*3*	(7)
first()	*7*	(7)
dequeue()	*7*	()
dequeue()	*null*	()
isEmpty()	*true*	()
enqueue(9)	–	(9)
enqueue(7)	–	(7, 9)
size()	*2*	(7, 9)
enqueue(3)	–	(3, 7, 9)
enqueue(5)	–	(5, 3, 7, 9)
dequeue()	*9*	(5, 3, 7)

http://www.cs.armstrong.edu/liang/animation/web/Queue.html

Typical Applications for Queues

- Direct applications
  - Waiting lists, bureaucracy
  - Access to shared resources (e.g., printer)
  - Multiprogramming
- Indirect applications
  - Auxiliary data structure for algorithms
  - Component of other data structures

Queues in Java

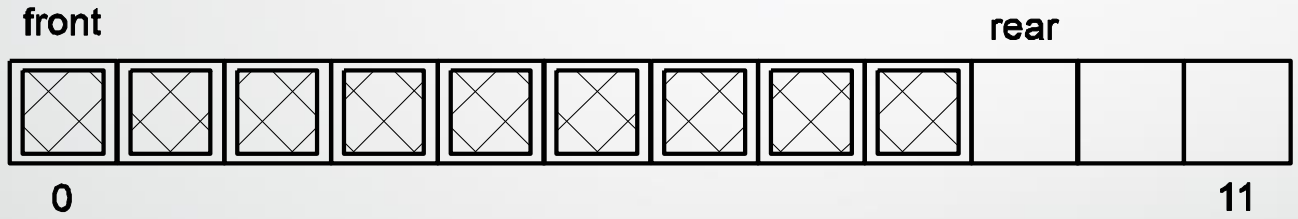
- Of course Java has a Queue built-in that we can use
  - `Queue<Integer> q = new LinkedList<>();`
- Let's give it a go

Let’s write our own

- Same as before, let’s try to understand how this works in the background
- In this case, we have different approaches. So let’s see some of them and their pros/cons

Array-Based Queue

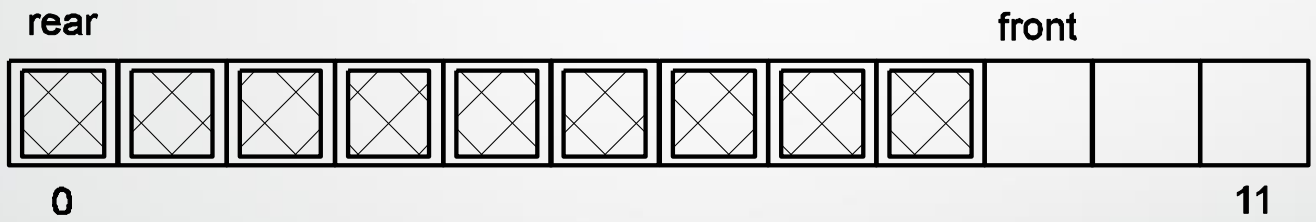
- We can consider the first index of the array to be the front of the queue



- The positive: Easy to add more elements to the queue
- The negative: When removing elements, we need to manually shift the remaining elements to the front of the array.

Array-Based Queue

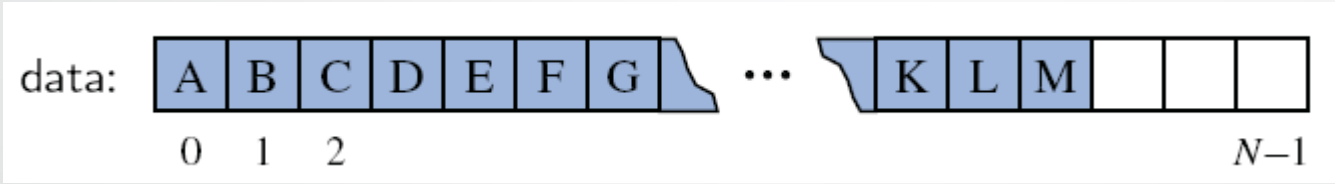
- We can consider the first index of the array to be the back of the queue



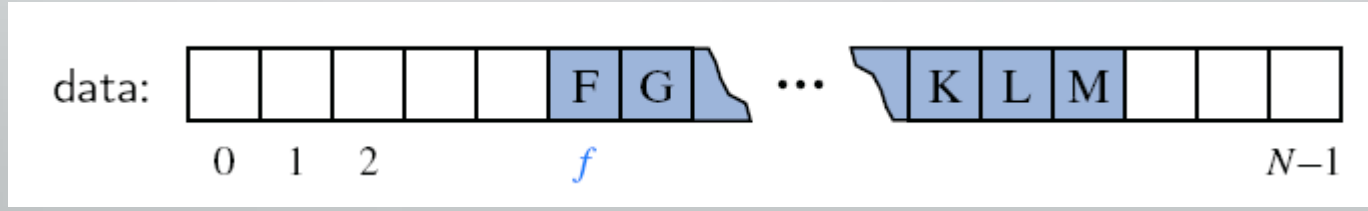
- The positive: Easy to add remove elements
- The negative: When adding elements, we need to manually shift the previous elements to the back of the array.

Array-Based Queue

- Using an array to store elements of a queue, such that the first element inserted, "A", is at cell 0, the second element inserted, "B", at cell 1, and so on.



- We will replace a dequeued element in the array with a **null reference**, and maintain an explicit variable ***f*** to represent the index of the element that is currently at the front of the queue.



Array-Based Queue

- The positive: We don’t need to rearrange any elements when adding/removing elements in the queue
- The negative: I have a problem when it comes to the end of the array. There is no more space to add elements to it.
- The negative: Potentially, we could end up with a massive array that has many null elements at the front.

Array-Based Queue

- Give it a go
- Let's create a class called "MyOwnQueue" that implements the interface that I'm going to give you
- This class will have an underlying array where the data is saved.

Array-Based Stack – Some Key Points

- We need to keep track of
  - Number of elements in the queue
  - Index of the front element
  - Index of the rear elelement
  - Conversions of types
  - Size of the underlying array
  - If my rear or front index is the last index of the array
- You could also return an exception instead of a null object if the array is empty

That’s all folks

- Any questions?