Stacks

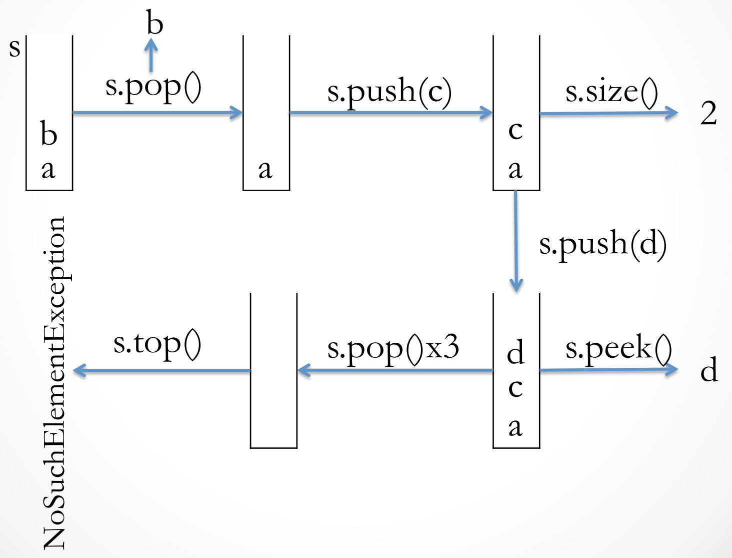
Based on the notes from David Fernandez-Baca and Steve Kautz

Bryn Mawr College
CS206 Intro to Data Structures

Stacks

- A **stack** is an access-restricted list. You may manipulate only the item at the top of the stack:
  - **push** a new item onto the top of the stack
    - **void push(E item):** Adds an element to the top of stack.
  - **pop** the top item off the stack
    - **E pop():** Removes and returns the top element of the stack. Throws NoSuchElementException if the stack is empty
  - examine (**peek** at) the top item of the stack
    - **E peek():** Returns the top element of the stack without removing it. Throws NoSuchElementException if the stack is empty
  - **boolean isEmpty():** Return true if the stack is empty, false otherwise
  - **int size():** Returns the number of elements in the stack.

Stack Example



Java Implementation

It is easy to implement a stack as a Java List:

Stack Method	List Method
push()	add()
peek()	get(size()-1)
pop()	remove(size()-1)
isEmpty()	isEmpty()
size()	size()

Java Implementation (cont.)

- Java provides different implementations of the List interface.
  - ArrayList implements it as a resizable array, so all the stack methods run in O(1) time.
    (To be precise, add() runs in O(1) amortized time.)
  - LinkedList implements List using doubly-linked lists. In this case, the time complexities of all stack operations is O(1) again.

Deque

In fact, Java has a legacy Stack class that implements all the required methods. However, Oracle recommends using the more modern Deque (for "doubly-ended queue) interface instead, as it provides "a more complete and consistent set of LIFO stack operations".

Stack Method	Deque Method
push()	addFirst()
pop()	removeFirst()
peek()	peekFirst()

Deque has many other methods. We will revisit this interface when we study queues.

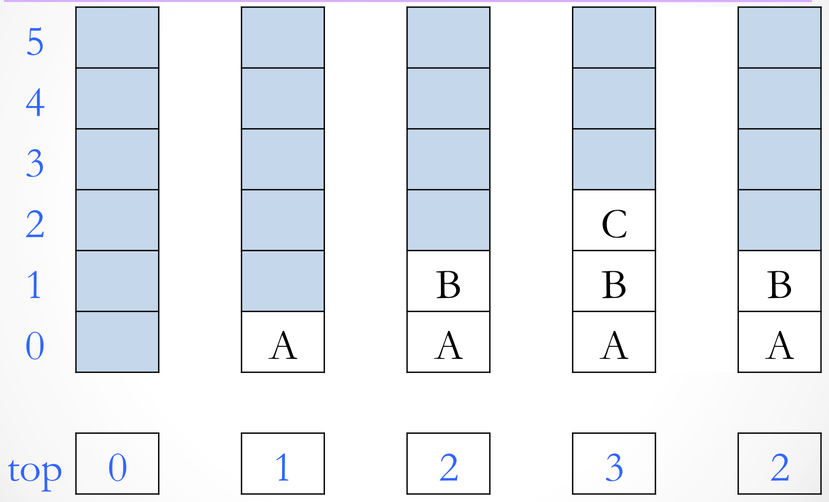
Two of the implementations of Deque are ArrayDeque and LinkedList.

Direct Implementation

The Java implementations of stacks are fine for many applications, but they do come loaded with unnecessary features; e.g., indexOf() and listIterator(). In what follows, we avoid these excess features, and use a more "lightweight" implementation.

```
public interface PureStack<E> {
    void push(E item);
    E pop();
    E peek();
    boolean isEmpty();
    int size();
}
```

Implementing Stack - Array

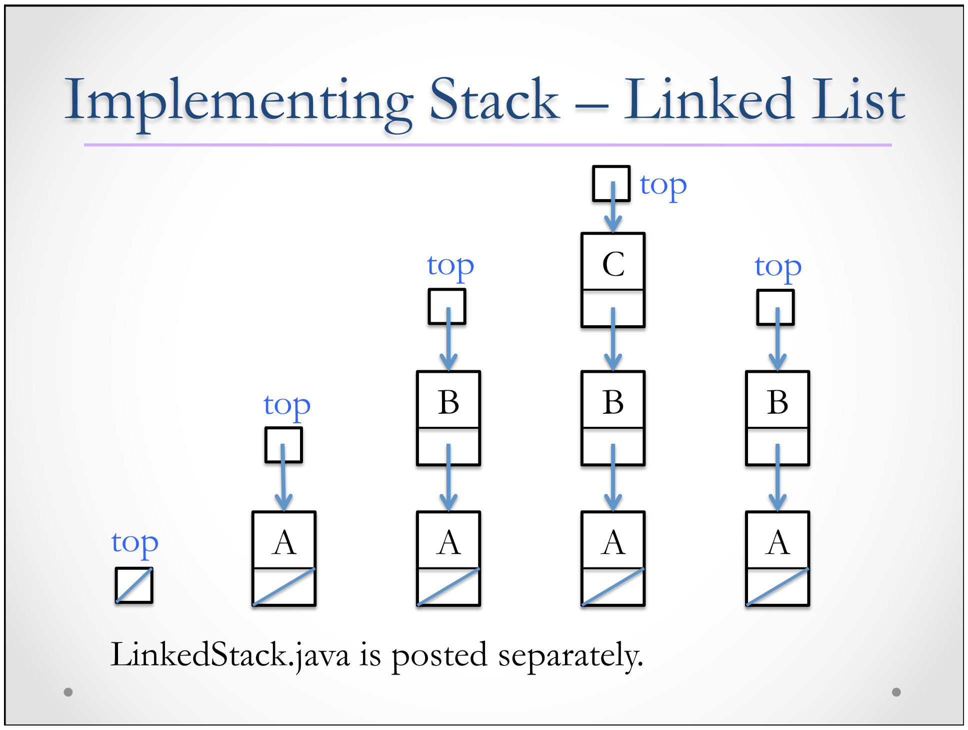


Implementing Stack - Array

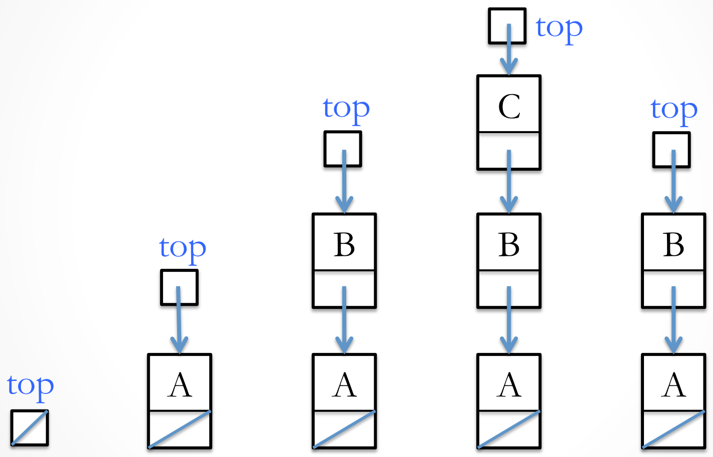
- We need a data array, and an index top into data. Entries data[0], … , data[top-1] contain the elements of the stack. A sequence of pushes and pops, starting from an empty stack.
- When there is no more space in the data array for another push, just double the size of the array.
- All operations take O(1) time (amortized, in the case of push).
- ArrayBasedStack.java is posted separately.

Implementing Stack – Linked List

- Singly-linked lists work well for stacks, since we only need access to the top.
- The idea is simple: just use a sequence of linked nodes, with a pointer top to the first node, which is viewed as the top of the stack.
- All operations take O(1) time.



Implementing Stack – Linked List



LinkedStack.java is posted separately.