ArrayLists

Mason Vail
Boise State University Computer Science

Interface - Implementation Separation

A list interface defines the operations and expected behavior of a list, but does not specify how a list implementation will manage list elements internally.

The user of a list does not need to know anything about the implementation to interact with a list through its interface. However, there is no single, “best” implementation of a list and the internal management will affect the efficiency of interface methods and memory use.

What if list elements are stored in an array?

Advantages:

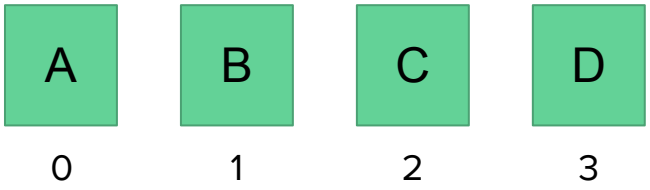
- Linear organization - easy to visualize
- O(1) lookup of elements by index if list element indexes correspond to array indexes

Disadvantages:

- Arrays do not automatically grow or shrink - array length and list size rarely match
- Elements must be shifted to keep them in correct index positions when adding to or removing from the list - O(n) operations other than at the rear of the list

Interface vs Internal Perspective

User's abstract, interface perspective



Programmer's internal, implementation perspective

T[ ]

0	1	2	3	4	5
A	B	C	D	-	-

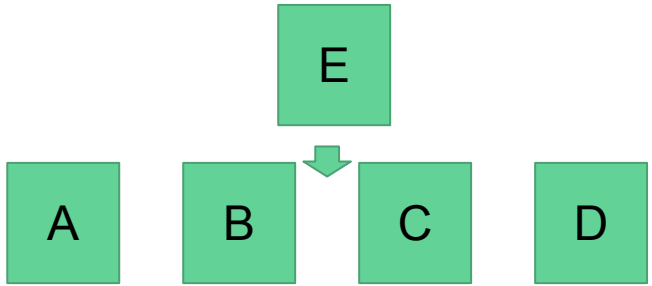
rear

Code - Getting Started

- Implement list interface
- Declare instance variables for array and rear
- Initialize variables in constructors
  - Default initial capacity
  - Specified initial capacity
- Identify utility methods to implement first (e.g. toString(), size(), isEmpty(), indexOf())

Adding an element - Interface perspective

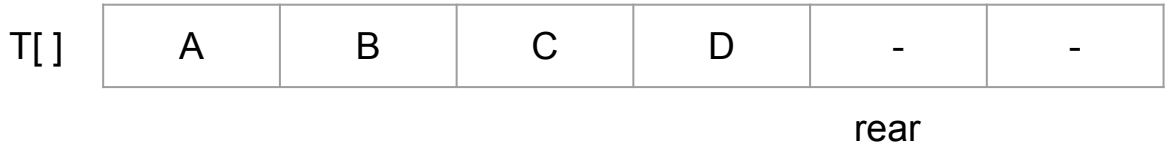
Add E after B



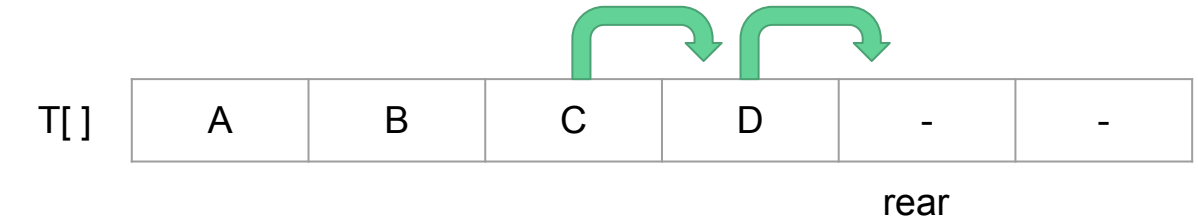
Done

Adding an element - Internal perspective

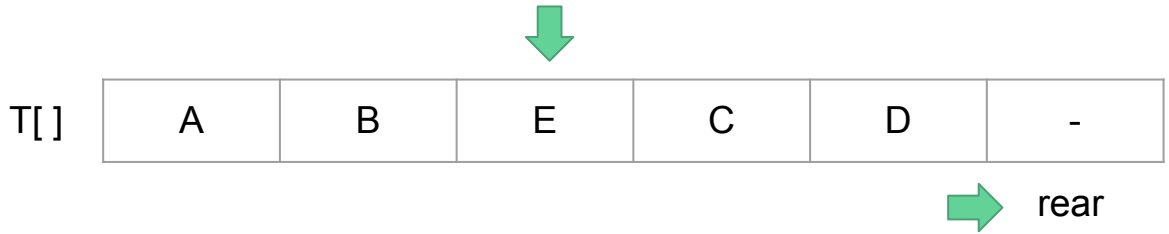
1. Check for adequate array capacity. Expand the array if necessary.



2. Shift elements to make room for E between B and C.



3. Insert E at its intended position. Update rear.

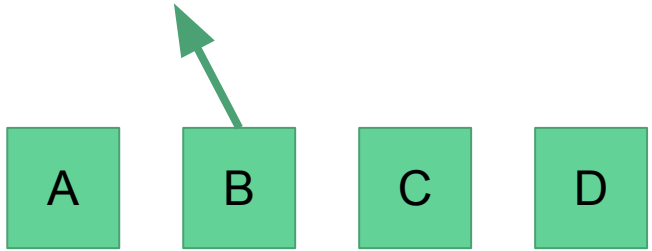


Code - addAfter(newElement, targetElement)

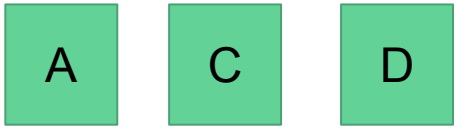
- Locate target element
  - Throw NoSuchElementException if not found
- Expand capacity if necessary
- Shift all elements after target back to make room for the new element
- Insert new element
- Update rear

Removing an element - Interface perspective

Remove B

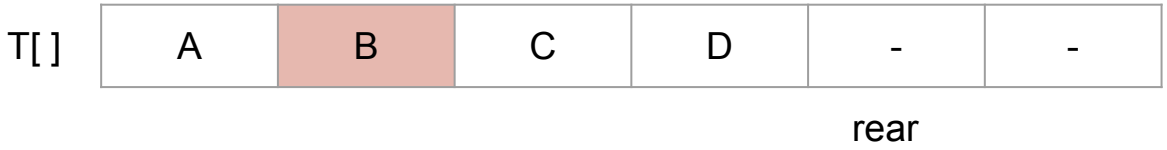


Done

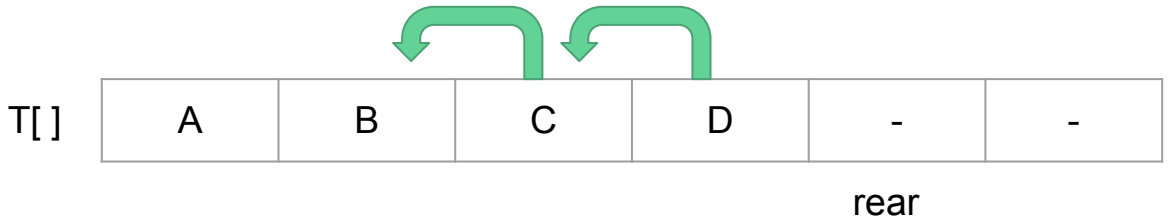


Removing an element - Internal perspective

1. Locate and remember the value being removed for later return.



2. Shift elements forward to overwrite B and leave no gaps.



3. Clear duplicate last value.
Decrement rear.
Return removed element.

Code - remove(element)

- Locate element and store the value for later return
  - Throw NoSuchElementException if not found
- Shift all following elements forward to overwrite removed element and leave no gaps
- Clear duplicate last element
- Decrement rear
- Return removed element

ArrayLists

Mason Vail
Boise State University Computer Science