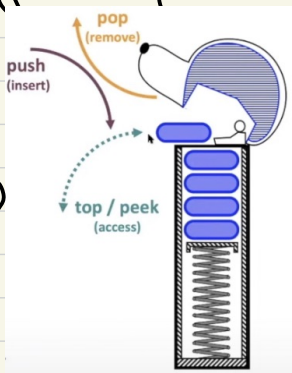
Video 1

Stacks: array and list implementations

stack – pile of elements, can only access/modify the top

LIFO (Last-in, First-out) principle



Operations:

- push – insert element at the top
- pop – remove element at the top
- top/peek – access top element (no remove)

Abstract Data Structure (ADT)

- specifies signatures of operations
- may contain constants, default / static methods

Stack ADT

Update methods

void push(E *e*)	adds element *e* to the top of the stack
E pop( )	removes and returns element at the top of the stack

Accessor methods

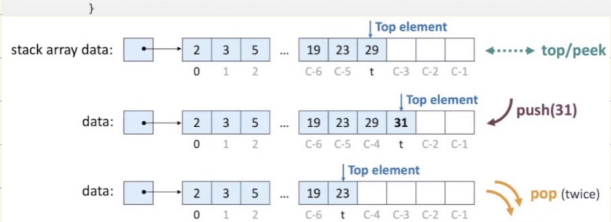
E top( )	returns the top element on the stack, without removing it
int size( )	returns the number of elements in the stack

Generic parameterized type (allows specification of element type at declaration)

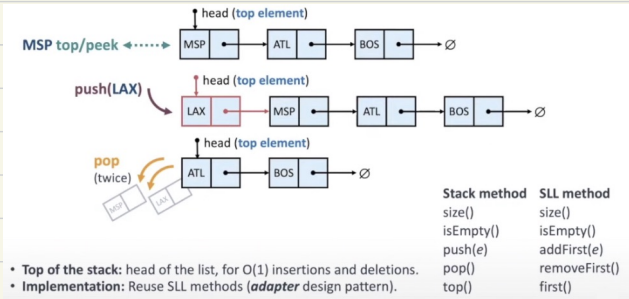
```
public interface Stack<E> {
  /**
   * Returns the number of elements in the stack.
   * @return number of elements in the stack
   */
  int size();

  /**
   * Tests whether the stack is empty.
   * @return <code>true</code> if the stack is empty, <code>false</code> otherwise
   */
  boolean isEmpty();

  /**
   * Inserts an element at the top of the stack.
   * @param e the element to insert
   */
  void push(E e);
  ...
}
```

Javadoc-style comments.
Keyword examples:
@return describes output values
@param describes parameters

Methods in an **interface** are implicitly **public**.



array capacity C
top element: at t
stack size: t+1
insertion/deletion: O(1)



space: n
top element: head
grows efficiently compared to the fixed size array

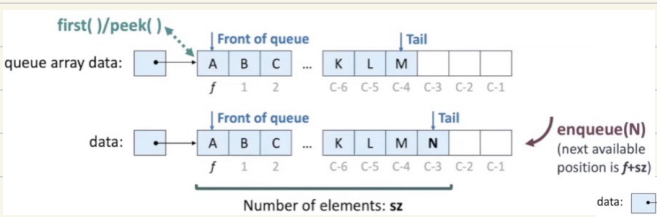
Video 2.

Queues: array and list implementations

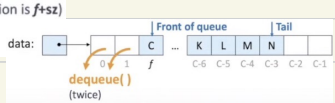
FIFO (First-in, First-out) principle

Queue ADT

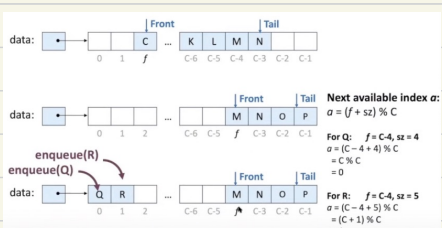
		java.util.Queue methods: throws Exception	java.util.Queue methods: returns special value
Update methods			
void enqueue(E *e*)	adds element *e* to the tail of the queue	add(*e*)	offer(*e*)
E dequeue( )	removes & returns the head element (or null)	remove( )	poll( )
Accessor methods			
E first( )	returns the head element, without removing it	element( )	peek( )
int size( )	returns the number of elements in the queue	size( )	
boolean isEmpty( )	returns true if the queue is empty, false otherwise	isEmpty( )	



array capacity C
front of queue: f
queue size: sz



queue
↳ 'wrap around' the array



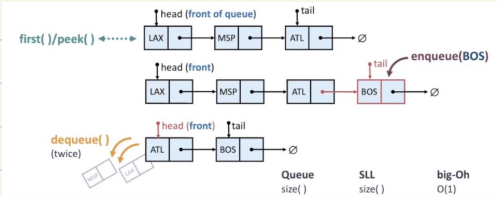
Time complexity

Method	big-Oh	Reasoning
size	O(1)	stored in variable **sz**, constant access
isEmpty	O(1)	relies on size **sz**, same reasoning
first	O(1)	constant access via stored index *f*
enqueue	O(1)	insertion at available index **(*f* + sz) % C**
dequeue	O(1)	deletion at index *f*, *update* **f = (*f* + 1) % C**

No shifting of elements in enqueue/dequeue!

Space complexity

Fields	big-Oh	Reasoning
array **data**	O(C)	fixed capacity **C**, independent of queue size **sz**
f	O(1)	primitive type **int**
sz	O(1)	primitive type **int**
	Overall **O(C)**	



Queue	SLL	big-Oh
size( )	size( )	O(1)
isEmpty( )	isEmpty( )	O(1)
enqueue(*e*)	addLast(*e*)	O(1)
dequeue( )	removeFirst( )	O(1)
first( )	first( )	O(1)

- **Implementation:** Reuse SLL (***adapter*** design pattern).
- **Front:** head, since tail deletion is not efficient.
- **Trade-off:** Grows efficiently. More space/time than array.

Video 3 — Deques: array and list implementations

Double-Ended Queue (Deque) ADT (deque is pronounced 'deck')

```
public interface Deque<E>
```

Update methods		Accessor methods	
addFirst(*e*)	insert element *e* at the front	first( )	returns first element without removing
addLast(*e*)	insert element *e* at the tail	last( )	returns last element without removing
removeFirst( )	remove and return first element	size( )	returns number of elements in the deque
removeLast( )	remove and return last element	isEmpty( )	true if deque is empty, false otherwise

Array implementation
Circular array (as in queue)
Space O(**C**) where **C** is array capacity

List implementation
Doubly-linked list
Space O(**n**) where **n** is number of elements

Deque ADT	Time	java.util.Deque (java.util.ArrayDeque / java.util.LinkedList) throws exceptions	returns special value
size( )	O(1)	size( )	
isEmpty( )	O(1)	isEmpy( )	
first( )	O(1)	getFirst( )	peekFirst( )
last( )	O(1)	getLast( )	peekLast( )
addFirst( )	O(1)	addFirst( )	offerFirst( )
addLast( )	O(1)	addLast( )	offerLast( )
removeFirst( )	O(1)	removeFirst( )	pollFirst( )
removeLast( )	O(1)	removeLast( )	pollLast( )

circular array:
add first
$f = (f - 1 + C) \% C$
List → Doubly-linked