Lecture 13 - Monday, February 27

Announcements

- Updated semester calendar
- ProgTest1: Guide & PracticeTest
- Makeup Lecture for WrittenTest1
    + Expected to complete by: March 20

Lecture

Stack ADT vs. Queue ADT

Abstract Data Types (ADTs)

Data Structures

trees, binary trees, balanced BT, BST.

---

stacks vs queues

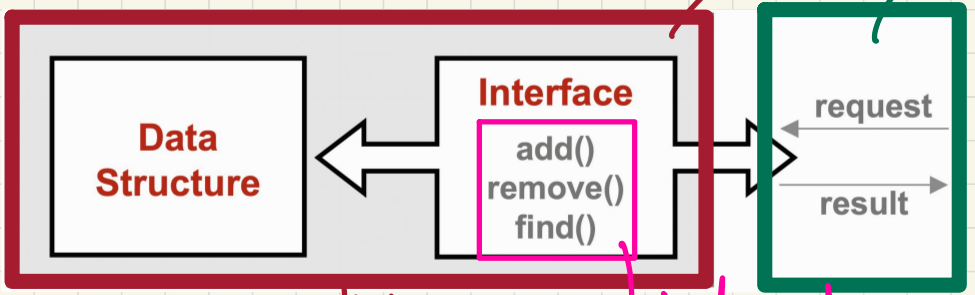
---

arrays vs. SLLs vs DLLs.

Abstract Data Types (ADTs)

supplier (callee)

client (caller)



1. input types
2. output type
3. description about going from inputs to output

headers of methods

supplier

```
class Microwave {
  private boolean on;
  private boolean locked;
  void power() {on = true;}
  void lock() {locked = true;}
  void heat(Object stuff) {
    /* Assume: on && locked */
    /* stuff not explosive. */
  } }
```

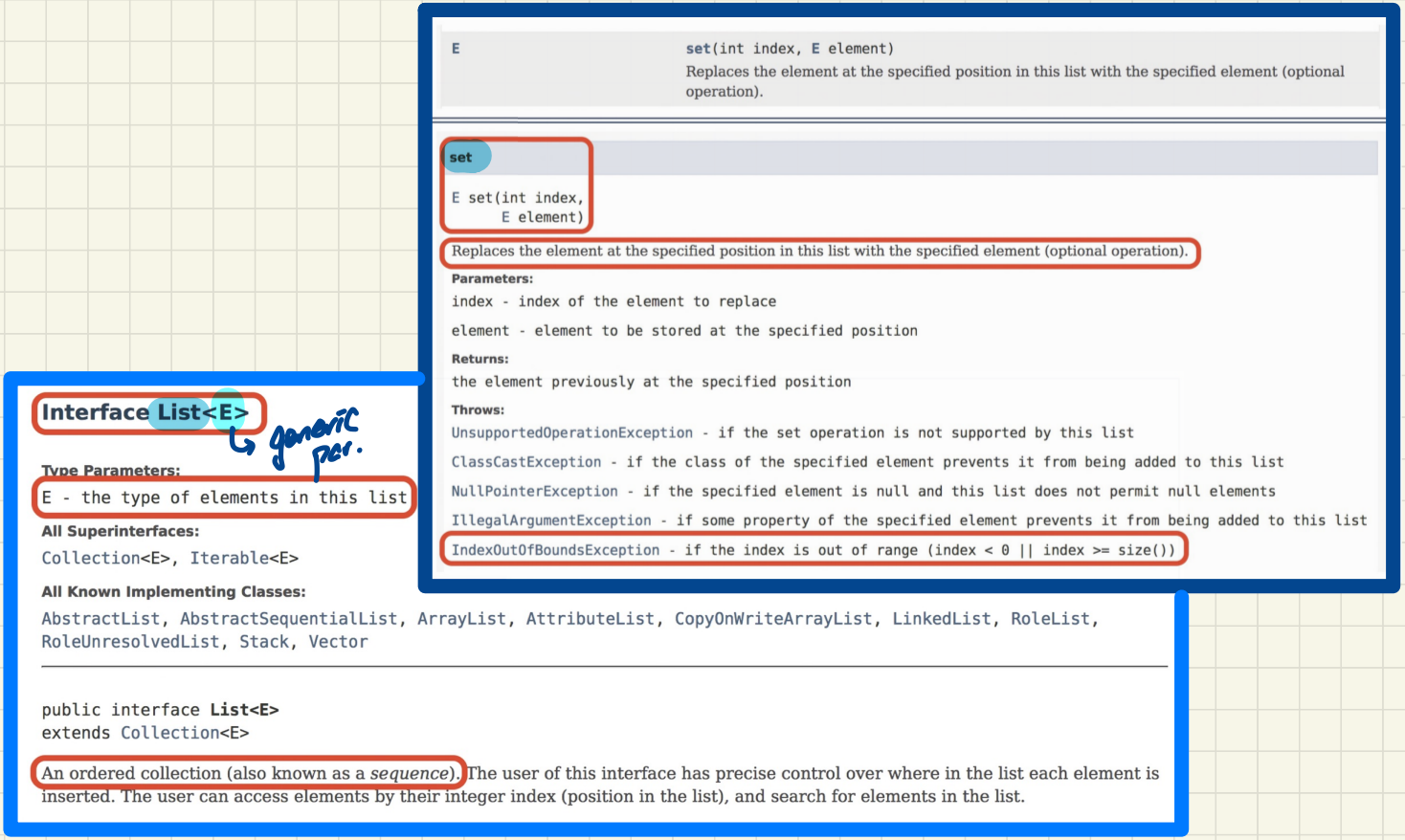
client

```
class MicrowaveUser {
  public static void main(...) {
    Microwave m = new Microwave();
    Object obj = ???;
    m.power(); m.lock();
    m.heat(obj);
  } }
```

→ in client MU, use the service "heat" from supplier Micro

	benefits	obligations
CLIENT	obtain a service	follow instructions
SUPPLIER	assume instructions followed	provide a service

Java API ≈ Abstract Data Types



E	set(int index, E element) Replaces the element at the specified position in this list with the specified element (optional operation).

set

```
E set(int index,
      E element)
```

Replaces the element at the specified position in this list with the specified element (optional operation).

Parameters:

index - index of the element to replace

element - element to be stored at the specified position

Returns:

the element previously at the specified position

Throws:

UnsupportedOperationException - if the set operation is not supported by this list

ClassCastException - if the class of the specified element prevents it from being added to this list

NullPointerException - if the specified element is null and this list does not permit null elements

IllegalArgumentException - if some property of the specified element prevents it from being added to this list

IndexOutOfBoundsException - if the index is out of range (index < 0 || index >= size())

Interface List<E> ↳ generic par.

Type Parameters:

E - the type of elements in this list

All Superinterfaces:

Collection<E>, Iterable<E>

All Known Implementing Classes:

AbstractList, AbstractSequentialList, ArrayList, AttributeList, CopyOnWriteArrayList, LinkedList, RoleList, RoleUnresolvedList, Stack, Vector

---

```
public interface List<E>
extends Collection<E>
```

An ordered collection (also known as a *sequence*). The user of this interface has precise control over where in the list each element is inserted. The user can access elements by their integer index (position in the list), and search for elements in the list.

Lecture

Stack ADT vs. Queue ADT

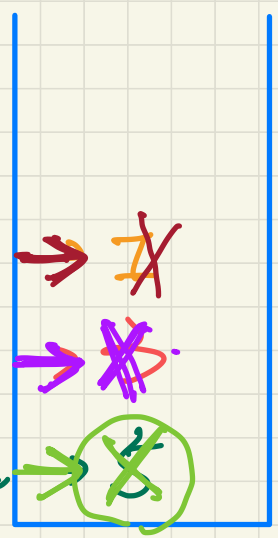
Stack ADT -
Last In First Out (LIFO)
Implementations in Java

Stack ADT: Illustration

	isEmpty	size	top
new stack	T	0	
push(5)	F	1	5
push(3)	F	2	3
push(1)	F	3	1
pop	F	~~3~~ 2	1
pop	F	~~2~~ 1	3
pop	F	~~1~~ 0	5
POP	T	0	

Error : precond. not satisfies

order in which elements added: 5, 3, 1



LIFO.

order in which elements removed: 1, 3, 5

Error : precond. of pop is violated

Implementing the Stack ADT in Java: Architecture

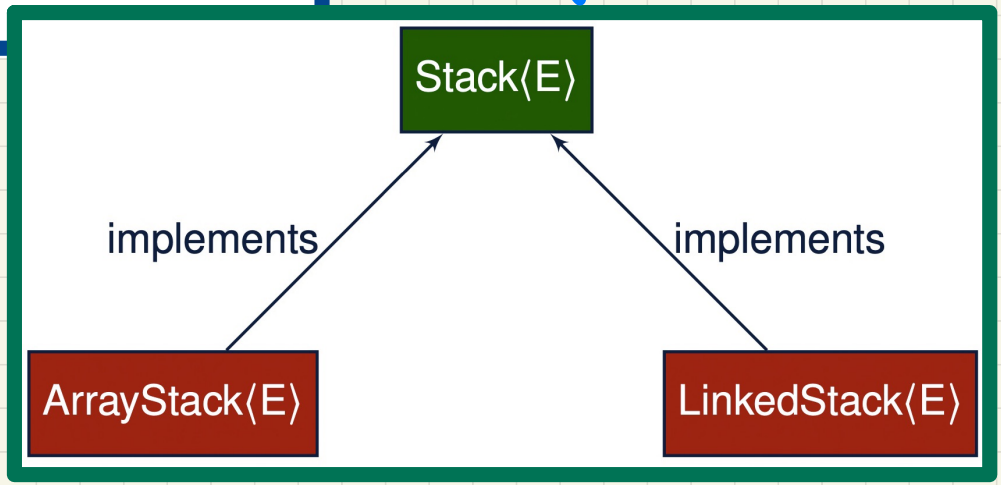
≈ ADT

```
public interface Stack<E> {
  public int size();
  public boolean isEmpty();
  public E top();
  public void push(E e);
  public E pop();
}
```

generic parameter

wrapper class

```
Stack<String> s1 = ...
Stack<Integer> s2 = ...
```




```
s1.push("A");
s2.push(23);
String v1 = s1.pop();
Integer v2 = s2.pop();
```

Implementing the Stack ADT using an Array

```
public class ArrayStack<E> implements Stack<E> {
  private final int MAX_CAPACITY = 1000;
  private E[] data;
  private int t; /* index of top */
  public ArrayStack() {
    data = (E[]) new Object[MAX_CAPACITY];
    t = -1;
  }

  public int size() { return (t + 1); }
  public boolean isEmpty() { return (t == -1); }

  public E top() {
    if (isEmpty()) { /* Precondition Violated */ }
    else { return data[t]; }
  }
  public void push(E e) {
    if (size() == MAX_CAPACITY) { /* Precondition Violated */ }
    else { t ++; data[t] = e; }
  }
  public E pop() {
    E result;
    if (isEmpty()) { /* Precondition Violated */ }
    else { result = data[t]; data[t] = null; t --; }
    return result;
  }
}
```

exception to push if stack already full.

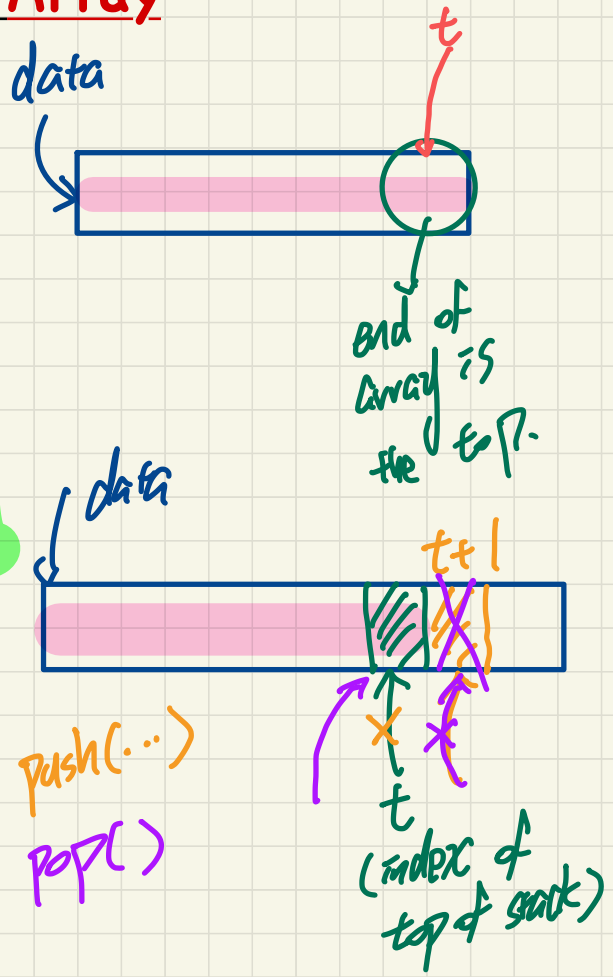
empty stack → no top.

- no loops
- no method calls

↳ O(1) all ops

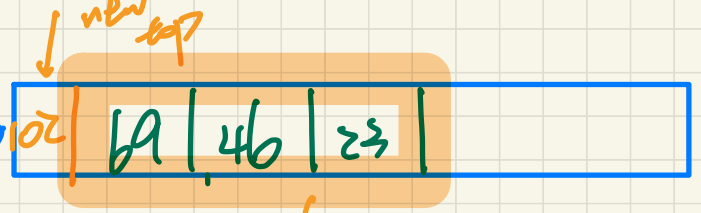
to relax this constraint ⇒ dynamic array

② amortized RT of push: ① doubling strategy O(1)



alternative imp. of stack using array

beginning of array : top of stack.



data

push(23)

push(46)

push(69)

push(102).

to make room for the new top, shift all items to right by 1 pos.

$\Rightarrow O(n)$

Implementing the Stack ADT using a SLL

Exercise
- DLL (first is top)
- DLL (last is top)

```
public class LinkedStack<E> implements Stack<E> {
  private SinglyLinkedList<E> list;
  ...
}
```

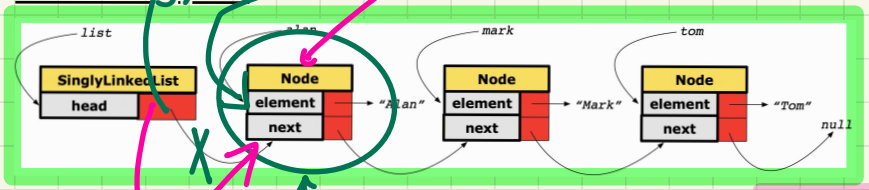
where's the top? first? last?

Stack Method	Singly-Linked List Method	
	Strategy 1	Strategy 2
→ size	list.size O(1)	
→ isEmpty	list.isEmpty O(1)	
top	O(1) list.first	list.last O(1)
push	O(1) list.addFirst	list.addLast O(1)
pop	list.removeFirst O(1)	list.removeLast

head

tail

Strategy 1



get 2nd last node (O(1))

O(1)

top

Strategy 2