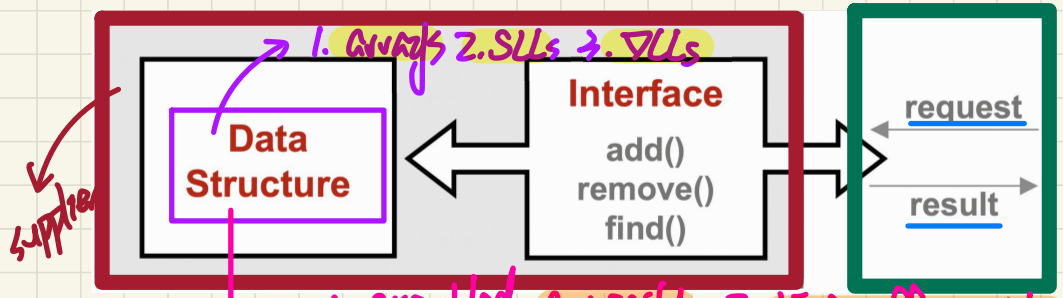
Abstract Data Types (ADTs)



client relies on the public interface of ADT()
1. input types
2. output types
3. what to be expected on IO related.

```
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. */
  } }
```

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

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

Java API ≈ Abstract Data Types

∴ NT is subject to ambiguities & contradictions -

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>

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

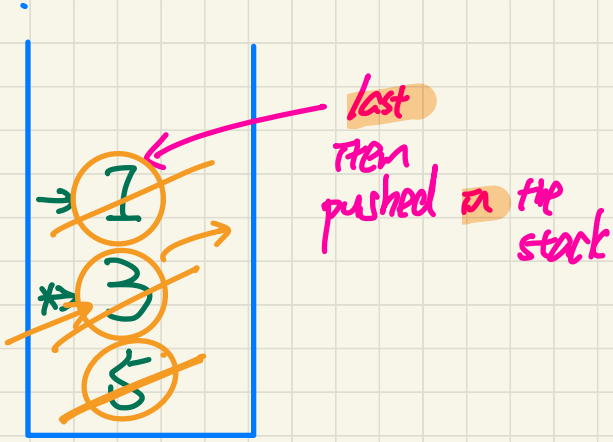
---

```
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.

Stack ADT: Illustration

	isEmpty	size	top
new stack	T	0	n.a.
push(5)	F	1	5
push(3)	F	2	3
push(1)	F	3	1
pop rtn. 1	F	2	3
pop rtn. 3	F	1	5
pop rtn. 5	T	0	n.a.



last pushed element

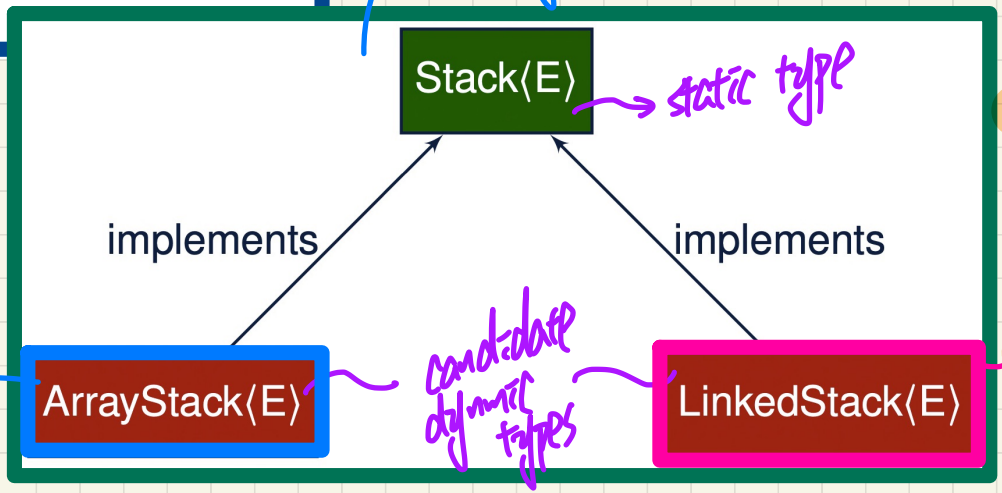
2nd last pushed element

(LIFO)

The order in which items are popped off the stack is the reverse of how these items were pushed.

Implementing the Stack ADT in Java: Architecture

```
public interface Stack<E> {
  public int size();
  public boolean isEmpty();
  public E top();
  public void push(E e);
  public E 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); }                    O(1)
  public boolean isEmpty() { return (t == -1); }           O(1)

  public E top() {                                         O(1)
    if (isEmpty()) { /* Precondition Violated */ }
    else { return data[t]; }
  }
  public void push(E e) {                                  O(1)
    if (size() == MAX_CAPACITY) { /* Precondition Violated */ }
    else { t ++; data[t] = e; }
  }
  public E pop() {                                         O(1)
    E result;
    if (isEmpty()) { /* Precondition Violated */ }
    else { result = data[t]; data[t] = null; t --; }
    return result;
  }
}
```

limitation: fixed size

ArrayStack<String>

↳ instantiates E for Stack:

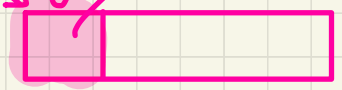
Stack<String>

(E[]) Object[____]

↳ what you have to write in Java.

efficient top. of stack

goal: treat this first item as the top.



Implementing the Stack ADT using a SLL

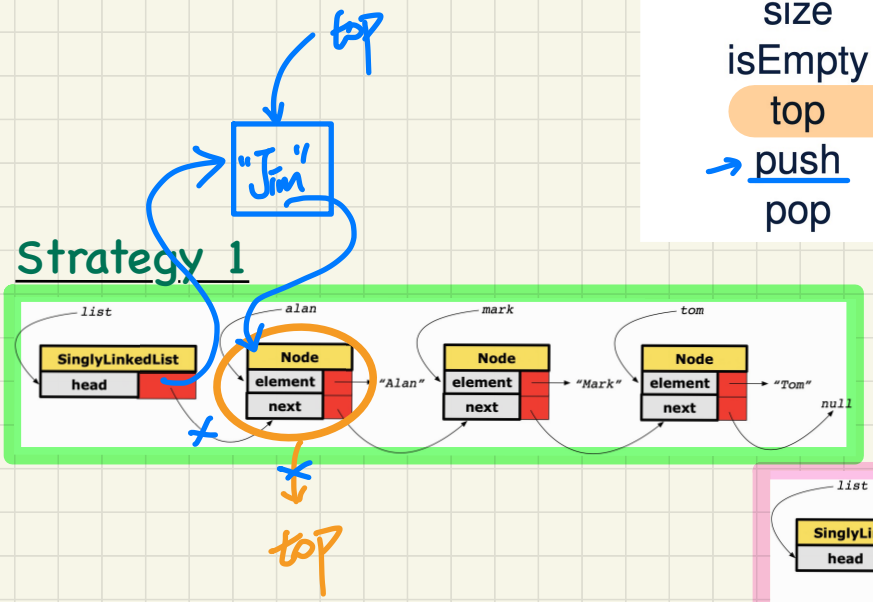
```
public class LinkedStack<E> implements Stack<E> {
  private SinglyLinkedList<E> list;
  ...
}
```

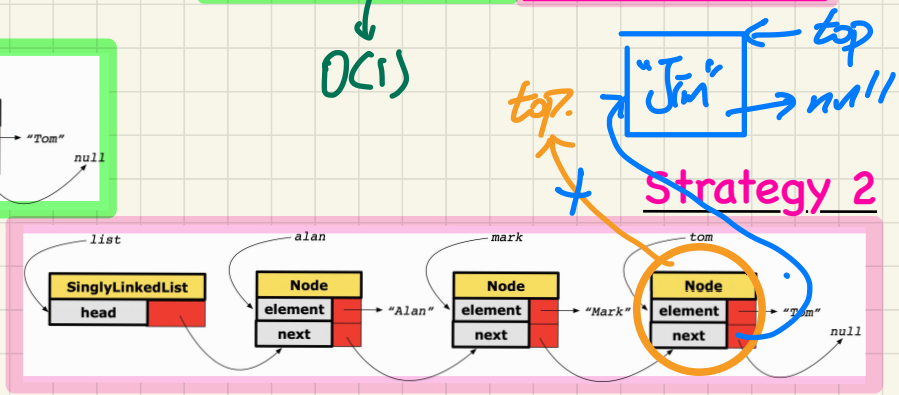
Stack Method	Singly-Linked List Method	
	Strategy 1	Strategy 2
size	list.size	
isEmpty	list.isEmpty	
top	list.first	list.last
push	list.addFirst	list.addLast
pop	list.removeFirst	list.removeLast

Improved to O(1) if a DLL is used.

O(1)

O(1)

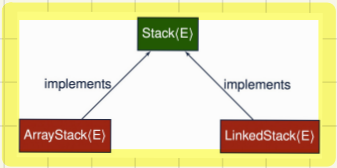




Stack ADT: Testing Alternative Implementations

Stack<S> s = new Stack<>();

✗ interface can't be a DT.




```
public class ArrayStack<E> implements Stack<E> {
  private final int MAX_CAPACITY = 1000;
  private E[] data;
  private 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;
  }
}
```

static type

```
@Test
public void testPolymorphicStacks() {
  Stack<String> s = new ArrayStack<>();
  s.push("Alan"); /* dynamic binding */
  s.push("Mark"); /* dynamic binding */
  s.push("Tom"); /* dynamic binding */
  assertTrue(s.size() == 3 && !s.isEmpty());
  assertEquals("Tom", s.top());

  s = new LinkedStack<>();
  s.push("Alan"); /* dynamic binding */
  s.push("Mark"); /* dynamic binding */
  s.push("Tom"); /* dynamic binding */
  assertTrue(s.size() == 3 && !s.isEmpty());
  assertEquals("Tom", s.top());
}
```

dynamic type

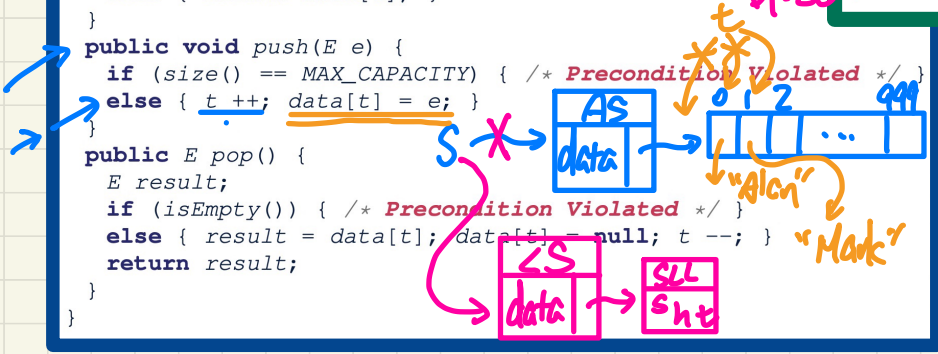
DT: AS

DT: LS

version in AS.

DT changes

version in LS. class of Stack?



Is the DT of s a descendant of:		*	**
s	instanceof Stack	T	T
s	instanceof ArrayStack	T	F
s	instanceof LinkedStack	F	T