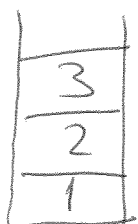
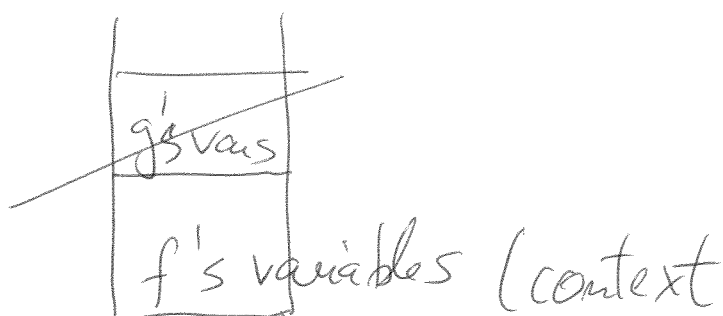


Stacks

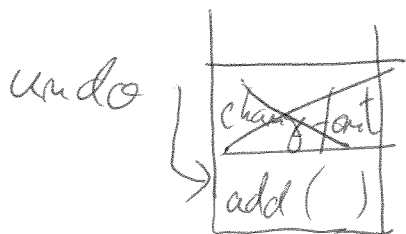
put things on a stack $\rightarrow$ on top
 take things off $\rightarrow$ from the top

1) Execution stack (in Java, other languages)

```
f() {
    g()
}
```



2) "Undo" feature $\rightarrow$ fixed size
 (arrays)



3) Compilers $\rightarrow$ "balanced" parentheses (similar
 balanced operations)

$(1 + (3 - 5) * 6$

Stack $(\Rightarrow$ put it on
 $) \Rightarrow$ take off

)(
 (Lists)

At end stack should be empty

stack Abstract Data Type (ADT)

set of Methods that any Stack has to provide

Add/remove only from the top

(even if inside eg. array which can be indexed anywhere)

$O(1)$ operations! (or "roughly" $O(1)$)

- boolean isEmpty()
- (maybe) int size()
- void push (Object o) or void push (~~AT~~ o)
- Object pop() or T ~~pop~~ pop()
- Object top() → look at content on top of stack but not remove it (like in pop)
- (maybe) isFull() → return true if no more space

```
public abstract class Stack {
```

```
    public abstract boolean isEmpty();
```

```
    public abstract void push (Object o);
```

```
    public Object pop();
```

```
    public Object top();
```

```
}
```

```
Stack<T>
```

```
Object → T
```

main () { "Same implementation" (21-3)
Stack<Integer> s = new Stack<Integer> ();
s.push (new Integer (5));
Integer i = s.pop ();
↑
methods of Integer apply

Object implementation:

Stack s = new Stack ();

s.push (new Integer (5))

Integer i = (Integer) s.pop ();

s.push (new BankAccount (- - -))

pop → remember what is coming out!

Stack<Object> s

Stacks using Lists

Stack only accesses through top $\rightarrow$ 1st element.

```
public class StackList implements extends Stack {
    List l;
```

```
    public void StackList() {
        l = new List(); // Single-linked List
    }
```

```
    public boolean isEmpty() {
        return l.isEmpty();
    }
```

$O(1)!$

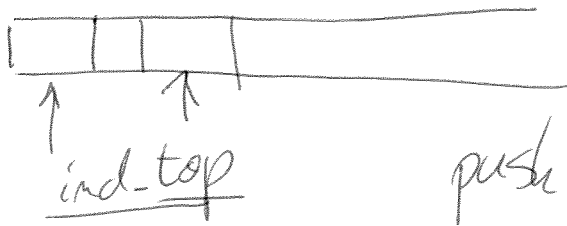
```
    public void push(Object o) {
        l.insertFirst(o);
    }
```

```
    public Object pop() { // maybe Exception handling
        return l.removeFirst();
    }
```

```
}
```

// restriction on types of operations permitted
no "new" functionality

```
}
```

Stacks using arrayspush: $\text{ind_top}++$ pop: $\text{ind_top}--$  $\text{ind_top} = -1 \Rightarrow$ stack is empty

Unlike List $\Rightarrow$ need to allocate some memory
 Editors $\rightarrow$ size is fixed to some amt (Options)
 Other applications $\rightarrow$ resize allowed

 $n \rightarrow 2n$ resize $\Rightarrow$ copy over old content ($O(n)$ operation)"Amortized" $O(1)$ operations $O(1)$ operations almost all the time $O(n)$ resize rarely (no more than $\approx \frac{1}{n}$)

L21-6

```
public class StackArray extends implements Stack {
```

```
    Object[] c;
```

```
    int ind_top; // references the topmost element  
                // -1 if stack is empty;
```

```
    public StackArray (int size) {  
        c = new Object[size];  
        ind_top = -1;  
    }
```

```
    public boolean isEmpty() {  
        return (ind_top == -1); // ind_top < 0  
    }
```

```
    public void push (Object o) {  
        ind_top++; // move to next available location  
        c[ind_top] = o;  
    } // short version
```

longer version 1:

```
    if (ind_top < c.length)  
        c[ind_top] = o;
```

```
    else // throw Exception
```

StackException → helps tell where the error is

Longer version 2: `resize`
 if (`ind_top < c.length`)
 `c[ind_top] = 0;`
 else `resize()`;

private void `resize()` {

`Object[] tmp = new Object(2 * c.length);`

 for (`int i = 0; i < c.length; i++`)

`tmp[i] = c[i];`

~~`c = tmp;`~~

3 // $O(n)$ but hopefully not usually called.

