Stacks

- Put things on a stack → on top
- Take things off → from the top

1) Execution stack (in Java, other languages)
   ```
   f()
   {
     g();
   }
   ```

2) "Undo" feature → fixed size (arrays)
   ```
   undo
   ```

3) Compilers → "balanced" parentheses (similar balanced operations)
   - Stack ( =) put it on
   - ) =) take off

At end stack should be empty

Example:

```
(1 + (3 - 5) × 6)
```

(1)
Stack Abstract Data Type (ADT) set of Methods that any Stack has to provide
Add/remove only from the top
(even if inside e.g. 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 (T o)
  • Object pop () or T 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 ()
};
main ( ) {  
	"See implementation"

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 -> 1st element.

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

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

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

    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 arrays

\[
\text{ind\_top}
\]

push: \text{ind\_top}++

pop: \text{ind\_top}--

\[\text{ind\_top} = -1 \Rightarrow \text{stack is empty}\]

Unlike list => need to allocate some memory. Edits => size is fixed to move out (Options)

Other applications => resize allowed

\[m \rightarrow 2m\]

\[\text{resize} \Rightarrow \text{copy over old content} (O(m) \text{ operation})\]

\"Amortized\" 0(1) operations

0(1) operations almost all the time

0(m) resize rarely \(\text{no more than } \approx \frac{1}{m}\)
public class StackAway extends Stack {
    Object[] c;
    int ind_top; // references the topmost element
    // -1 if stack is empty

    public StackAway (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:
    if (ind_top < c.length)
        c[ind_top] = o;
    else // throw Exception
        StackException -> helps tell where the error is
Long version 2: 

```java
if (ind_top < c.length)
    c[ind_top] = 0;
else resize();
```

```java
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 \( \theta(n) \) but hopefully not usually called.