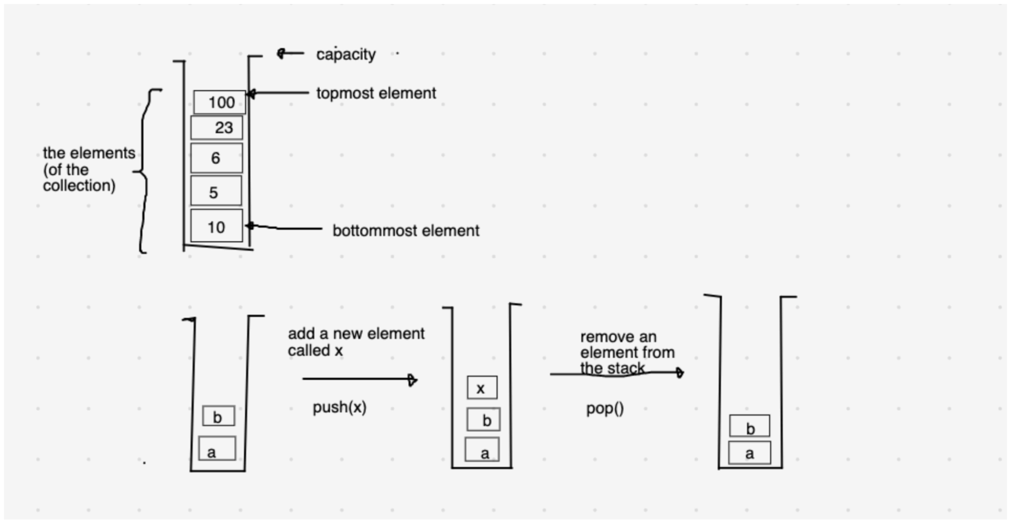
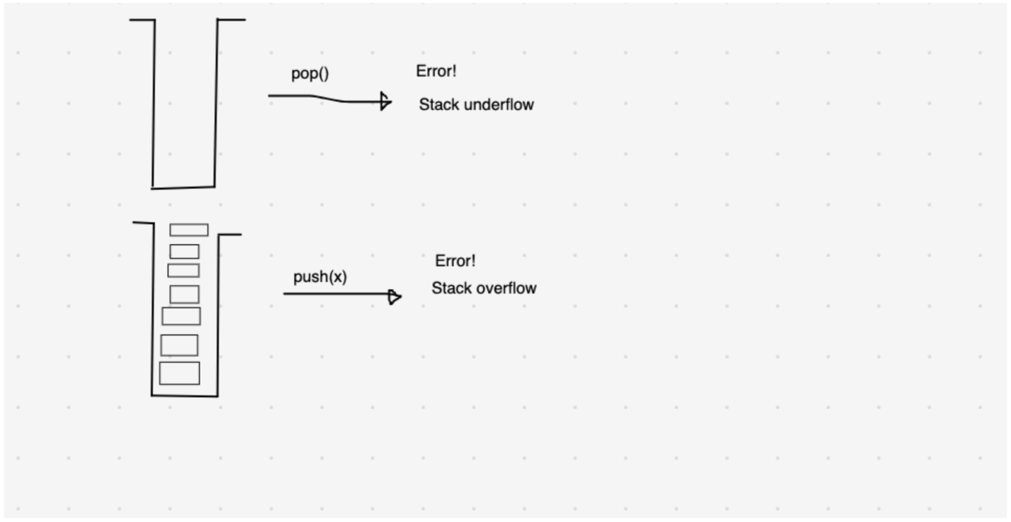
Stack

Stack values

- Collection of data, we'll call 'elements'
- Elements are accessed in an order: LIFO (last-in, first-out)
- Two operations: push and pop


capacity
topmost element
the elements (of the collection)
100
23
6
5
10
bottommost element
b
a
add a new element called x
push(x)
x
b
a
remove an element from the stack
pop()
b
a

- Possible issues with push and pop



Stack API

- API = Application Programming Interface
- Operation by operation define some features: description, signature, preconditions, return.
- Preconditions: condition met in order to perform the operation.

We will use `int` for now!

push(x)

Description	Add an element to the top of the stack.
Signature	void push(int x) *
Preconditions	stack must exist **, stack not full.
Returns	None. *

- can also return `boolean` (ex: true/false if error), `String` (Ex: error message).

** ”must exist” is true of all operations, so we can omit it.

```
Stack s;// = new Stack();
s.push(123);  // NullPointerexception
```

pop()

Description	Remove the element at the top of the stack
Signature	int pop() *
Preconditions	stack not empty.
Returns	The element removed from the top.

- can also return `void`, but then we need another operation to examine the top of the stack.

size

Description	Determine the numbers of elements in the stack
Signature	int size()
Preconditions	none
Returns	the number of elements

is full

Description	Determine if the stack is full
Signature	boolean isFull()
Preconditions	none
Returns	true if the stack is full, false otherwise

is empty

Description	Determine if the stack is empty
Signature	boolean isEmpty()
Preconditions	none
Returns	true if the stack is empty, false otherwise

Exercise: paired brackets

Code a method `hasPairedBrackets` that uses a stack to determine if brackets (), {}, [] and <> are properly paired: each opened bracket is closed but only after subsequent opened brackets are closed…

Ex:

Good:	(), ([]), <[]()>, ((())), etc…
Bad:	(, (], {[}], «>, <», ((((((((((
Good* :	(abc), (a[b]c), etc…

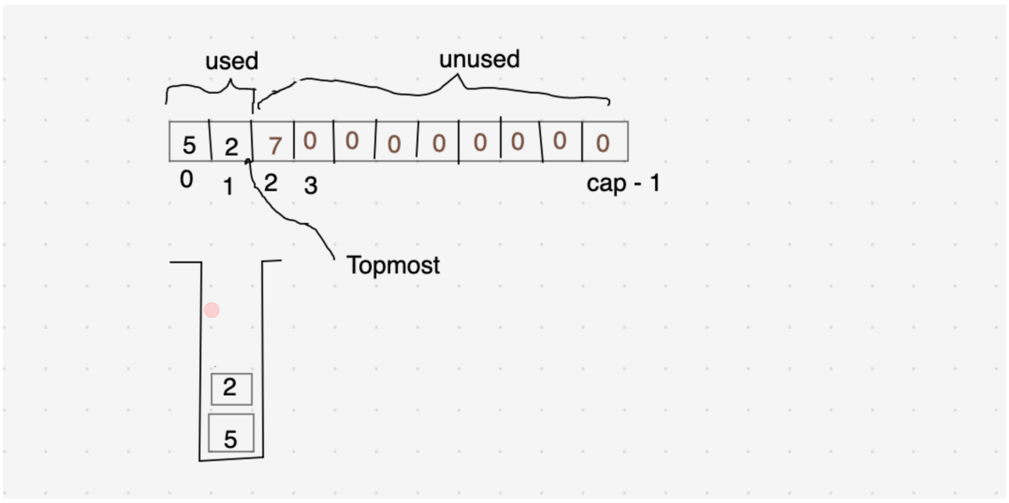
- If we ignore non-brackets characters.

Sample Solution

```
public static boolean hasPairedBrackets(String input) {

    // stack is storing currently unclosed brackets as we scan the input
    IntStack stack = new IntStack(input.length());

    for(char c : input.toCharArray()) {
        switch (c) {
            // for open brackets push the corresponding closing bracket
            case '(':
                stack.push(')');
                break;
            case '[':
                stack.push(']');
                break;
            case '{':
                stack.push('}');
                break;
            case '<':
                stack.push('>');
                break;

            // for closing brackets, check the stack to see if they correspond to currentl
            case ')':
            case '}':
            case ']':
            case '>':
                // no more brackets on the stack
                if (stack.isEmpty())
                    return false;

                // cast is needed since int's are bigger than chars
                char bracket = (char) stack.pop();

                // no match means unbalanced
                if (bracket ≠ c)
                    return false;
                break;
            default:
                // non-bracket character... do nothing
        }
    }
    // if the stack isn't empty then there is no match
    return stack.isEmpty();
}
```

Exercise: implement the stack API using an array

- Idea: store the stack in the bottom portion of the array, in indices [0, size-1].



- Implement as a Java class.

Sample Solution

```
public class IntStack {

   private int[] elements;
   private int top;

   public IntStack(int capacity) {
      elements = new int[capacity];  // primitive arrays default to 0
      top = 0;
   }

   public void push(int x) {
      if(isFull())
          throw new StackOverflowException();
      elements[top++] = x;
   }

   public int pop() {
      if(isEmpty())
          throw new StackUnderflowException();
      return element[--top];
   }

   public int size() {
      return top;
   }

   public boolean isEmpty() {
      return top == 0;
   }

   public boolean isFull() {
      return top == element.length;
   }


}
```