Stacks

What is a stack?, standard stack operations, algorithm using stacks, implementing stacks as linked lists, implementing stacks with composition

What is a stack?

A **stack** is a special kind of list with a restricted set of operations.

- The last element added to a stack is the first to be removed.
- The only stack element you can access without changing the stack is the **top** element, which is the most recently added one.

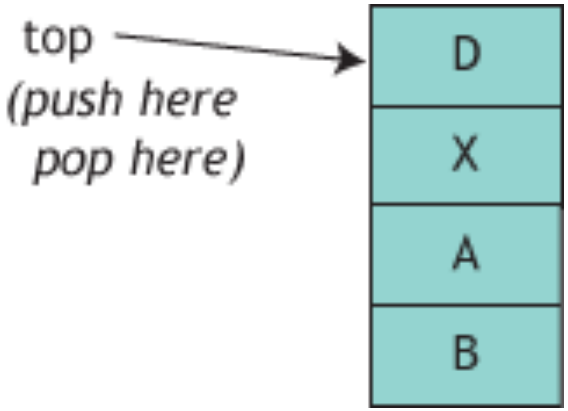
Stacks are called last-in-first-out lists, or **LIFO** lists.

Stacks are meant to be simple and efficient. All pure stack operations should have complexity O(1).

Stacks are like…

- Stacks of cafeteria trays
- Stacks of plywood
- Runtime stack (method calls)
- Browser history stack

You always get the one off the top. You always put new ones on the top.



Standard stack operations

Standard stack operations are:

- **push:** A stack can add new elements only at the top (the front of the list). To push an element on the stack means to add it to the top of the stack.
- **pop:** A stack can remove only its top element. To pop the stack means to remove that top element and return it as the value of the operation.
- **isEmpty:** A stack can tell whether it is empty.
- **peek**: A stack can tell what its top element is without removing it (popping the stack).

Example exercise

A postfix expression has the operators after the operands rather than between:

2 + 3	becomes	2 3 +
1 + 2 * 3 + 4	becomes	1 2 3 * + 4 +

Evaluation algorithm uses a stack:

- When you read an operand (number), push it on the stack.
- When you read an operator, pop the stack twice and apply the operator to the two values. Push the result back on the stack.
- When there's nothing left to read, the stack should have exactly one element, which is the result.

Implementing stacks via linked lists

This implementation is standalone (does not require other classes). Declaration and data:

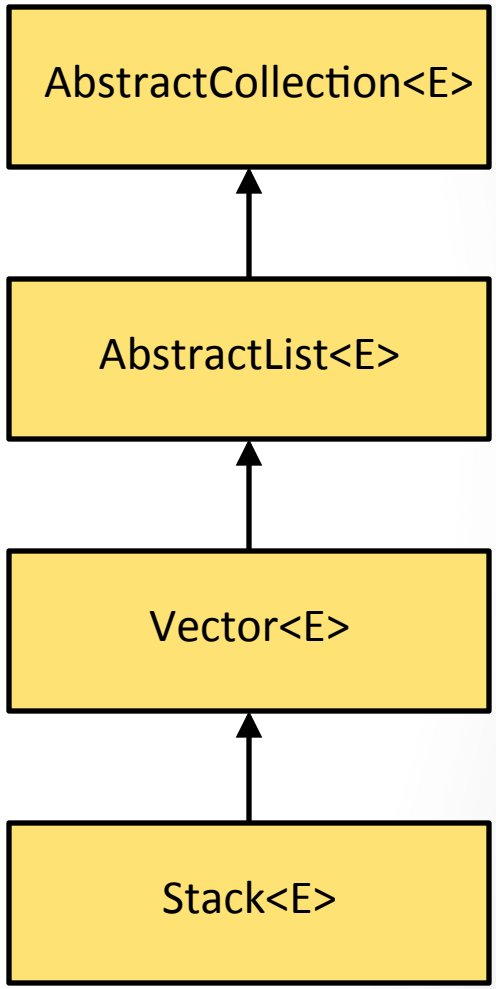
```
public class Stack<E> {
   private Node top;

   private static class Node {
      public E data;
      public Node next;

      public Node(E data, Node next) {
          this.data = data;
          this.next = next;
       }
   }
```

Stack via linked lists

```
public Stack() { top = null; }
public boolean isEmpty() { return top == null; }
public void push(E s) { top = new Node(s, top); }
public E pop() {
    if (top == null)
        return null;  // or throw NoSuchElementException
    E value = top.data;
    top = top.next;
    return value;
}
public String peek() {
    if (top == null)
        return null; // or throw NoSuchElementException
    return top.data;
}
```

Stack<E> from the Java API

- java.util.Stack is built into the Java API. Its inheritancy is shown on the right.
- It has many operations that aren't true stack operations, including 50 from Vector<E>.
- Java.util.Stack defines the standard stack operations (push, pop, etc) plus a non-standard search operation.
- Question: Can we use Stack<E> to create a true stack?



Composition for reuse

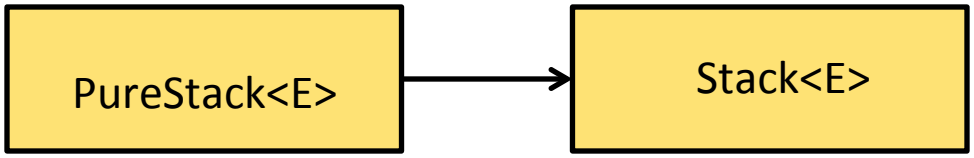
- Composition is a powerful form of code reuse. (Not as "brittle" as inheritance.)
- Composition means defining a class that has a member of a different class type.
- Composition can **delegate** the work for a new class to an existing class.
- Composition can cloak the operations of an existing class when it is to be used in a particular restrictive fashion. That means you can use existing code for exactly the behaviors that you want to provide.

You can use composition to create a pure stack type from Stack<E>.

Pure stack via composition

PureStack has a Stack<E> instance variable, stack. All the work of PureStack is delegated to stack.

```
public class PureStack<E> {
   private Stack<E> stack;

   public PureStack(){ stack = new Stack<E>(); }
   public boolean isEmpty() { return stack.empty(); }
   public void push(E s)    { stack.push(s); }
   public E pop()           { return stack.pop(); }
   public E peek()          { return stack.peek(); }
}
```



You could modify this class to include more methods, though it would no longer be a pure stack.