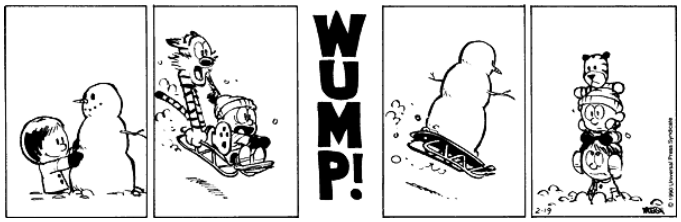
Lecture 12: Stacks



CS 62
Pomona College
February 17, 2014

Beth Trushkowsky

Today

- Thanks for having me!
- Announcements
  - This week's lab: Twin towers
  - Assignment: Compression, due Feb. 23
- Objectives
  - Stacks – JS Ch. 10.1
  - Queues (maybe) – JS Ch. 10.2

Stacks

- Linear data structure
  - Last week: linked lists
- Analogy: stack of plates

- Operations
  - LIFO: "last in, first out"
  - Push(), Pop()
- Limit access to data
  - Facilitates certain algorithms
  - Less easily corruptible w/o random access

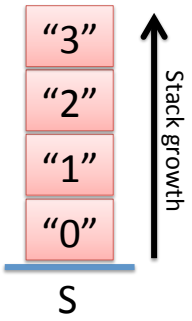
Stack: class definition

```
public class Stack<E> {
   // Test if stack is empty
   boolean empty()

   // Looks at object at top of stack without
   // removing it from the stack
   E peek()

   //Removes and returns object at the top of the stack
   E pop()

   //Pushes an item onto the top of the stack
   E push(E item)
}
```

Using a stack

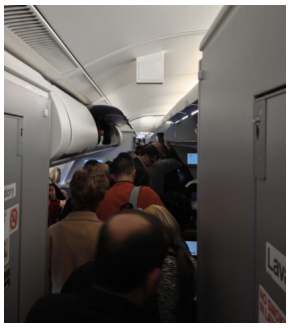
```
Stack<String> s = new Stack<String>();

for (int i = 0; i < 4; ++i) {
   s.push("" + i);
}

while (s.size() > 0) {
   System.out.print(s.pop());
}
```

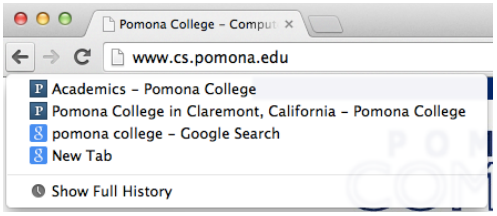
3 2 1 0



Stacks in the world

Examples of stacks in real life?





Stack applications

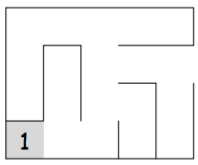
Runtime stack

```
/**
* Recursive function to compute factorial
*/
public int fact(int n) {
   if (n<=1) return 1;
   else return n * fact(n-1)
}
```

Stack applications

- Backtracking: maze example
  - Recursive solution
  - Can make non-recursive using stack!
- Recursive solution pseudocode:

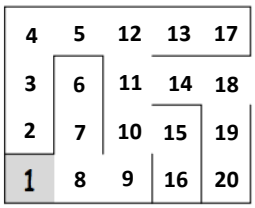
```
method findExit(start) {
    if (isExit(start)) FOUND
    else if (!alreadySeen(start)) {
        markSeen(start)
        for each spot x adjacent to start
            findExit(x)
    }
}
```



Stack applications

- Non-recursive solution pseudocode:

```
method findExit(start) {
   S = new Stack()
   S.push(start)
   while (!S.isEmpty()) {
      start = S.pop()
      if (isExit(start)) FOUND
      else if (!alreadySeen(start)) {
         markSeen(start)
         for each spot x adjacent to start
            S.push(x)
   }
}
```



Stack applications

- Evaluating *postfix* expressions
  - E.g., 52 5 7 + 4 * - → 4
- How can we use a stack to evaluate?
  - Read tokens left to right

Exercise!

- Transform an infix expression to its postfix equivalent
  - Tokens: integers and the binary operators +, -, *, /
  - Example
    Infix: 10 + 2 * 8 /4 - 3
    Postfix: 10 2 8 * 4 / + 3 -
- Write pseudocode or English for the algorithm
- Hints
  - Try first only considering the operators +,-
  - Think about priority of operators

Stack implementations

	ArrayList	SinglyLinkedList
Head location?	Highest index	Head of list
Push()	O(1), worst=O(n)	O(1)
Pop()	O(1)	O(1)

- Space differences?
  - What about multiple stacks?
- java.util.Stack uses *Vector* – don't use!

Stack implementations: Arraylist

```
public class StackArrayList<E> extends AbstractStack<E>
                               implements Stack<E> {
    // The ArrayList containing the stack data.
    protected ArrayList <E> data;

    public StackArrayList()
    {
        data = new ArrayList<E>();
    }
    public void push(E item)
    {
        data.add(item);
    }
    public E pop() {
        return data.remove(size()-1);
    }
    public E peek() {
        // raise an exception if stack is already empty
        return data.get(size()-1);
    }
}
```