Lecture 13: Stacks

CS 62

Fall 2018

Alexandra Papoutsaki & William Devanny

Reading about Collection Classes

- Oracle’s Java Tutorials
  - Trail: Collections
  - https://docs.oracle.com/javase/tutorial/collections/

Stack ADT

Linear data structure that stores arbitrary objects

Objects are inserted and removed following
the LIFO principle (Last-In First-Out) from the same enc

Similar to lists, there is a sequential nature to the data
Unlike lists, can only add and remove most recent item

Metaphor of cafeteria plate dispenser.

- Want a plate? *Pop* the top plate
- Add a plate? *Push* it to make it the new top plate

Stack Interface

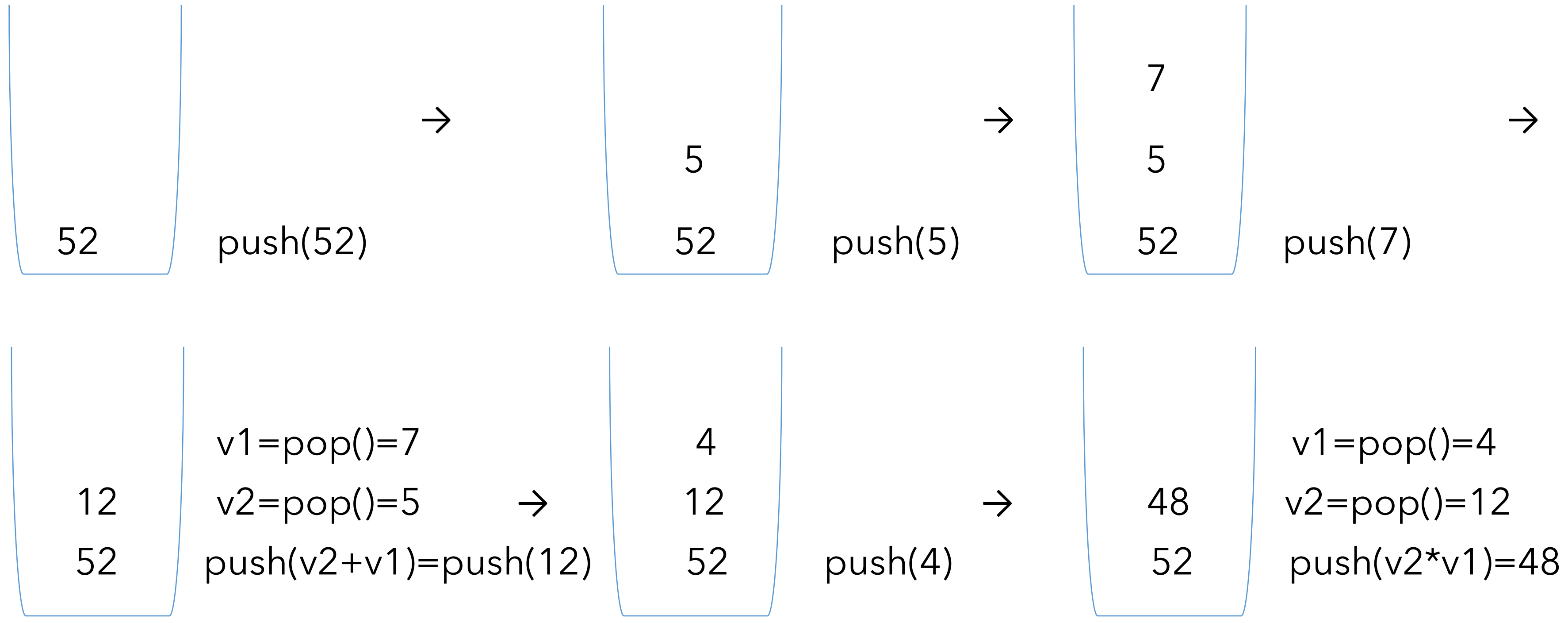
```
public interface Stack<E> extends Linear<E> {
  //same as add(E item)
  public void push(E item); //add item to top of stack
  //same as remove()
  public E pop(); //remove item from top of stack
  //same as get()
  public E peek(); //return reference to top of stack
  public boolean empty();
  public int size();
}
```

Stack Applications

- Run-time stack:
  - See `sum` demo
- Backtracking
  - Solving Maze demo
- Tools to parse programs
- Undo Command
- Browser History

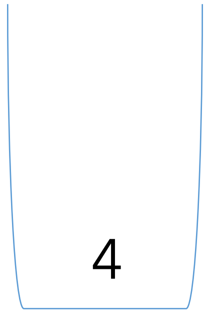
Evaluating expression in postfix form

Example: (52 - ((5 + 7) * 4) $\Rightarrow$ 52 5 7 + 4 * -



Evaluating expression in postfix form cont.

Push as long as you see operands.
value1 = pop(). value2 = pop(). push(value2 operator value1).



v1=pop()=48

v2=pop()=52 $\rightarrow$ peek()=4

push(v2-v1)=4

Implementing Stacks with Linked Lists

Where should the top go?
The **head** represents the top of the stack
To push an item `addFirst()`
To pop an item `removeFirst()`

Look at `LinkedStack` in `structure5`

Singly linked or doubly linked?

Runtime of the different operations:

- `push()`: $O(1)$
- `pop()`: $O(1)$
- `peek()`: $O(1)$
- `empty()`: $O(1)$

Implementing Stacks with ArrayLists

Where should the top go?

Use the END of the list at the top of the stack

To push an item `add()`

To pop an item `get(list.size()-1)` to return it and `remove(list.size()-1)`

Look at `ArrayListStack` in `structure5`

Runtime of the different operations:

- `push()`: $O(1)$
- `pop()`: $O(1)$
- `peek()`: $O(1)$
- `empty()`: $O(1)$

Which one is better?

- ArrayList is "amortized" $O(1)$ run-time, however, any individual push operation could be $O(n)$
- Memory trade-off is less clear
  - ArrayList could have lots of "open" memory
  - LinkedList has an extra reference for each data item

- `java.util.Stack` based on Vector - don't use!
  - `ArrayDeque` is better choice (*more details later)*