menu
- Articles

featured articles
- QuickSort Tutorial
- Linked Lists Tutorial

A tutorial on Stacks in Java

What is a Stack?

If you've ever put something on top of something else of the same type (e.g. sheets of paper), you have created a stack! In terms of programming, stacks mean just that.

Since Stacks are practically identical to Queues (with one major difference), this article builds on the amazing Queues tutorial. I advise you to read it first if you feel a bit confused.

Basic operations of Stacks

The following is a list of operations that can generally be performed on a stack. This of course will depend on your implementation.

```
Method names for my implementation are in blue.
Descriptions are in green.
Examples are in red.


push(object) - Adds the object to the top of the stack.
               (Putting another sheet of paper on top of the pile)

pop()        - Removes the object from the top of the stack and returns
               it to the caller.
               (Taking a piece of paper from the top of the pile)

peek()       - Returns the object at the top of the stack without
               removing it.
               (Only reading the sheet of paper at the top of the pile)
```

Example of using Stacks

```
The current state of the stack is highlighted in green.
Note: The top of the stack is at the right of the green boxes.
The operation being performed is in blue.
Comments are in green.

We begin with an empty stack.
peek(); would return null.

push(A);
A
A is placed on top of the empty stack.

push(B);
AB
B is placed on top. A is now under B, not on top.

peek();
AB
peek() does not change the stack itself, it only returns
the reference to the object on top, which is B.

pop();
A
B is removed from the stack and returned to the caller.
```

```
B is removed from the stack and returned to the caller.

pop();
A is removed from the stack and returned to the caller.
The stack is now empty.
```

Java example of a Stack class

My implementation of the stack uses linked lists. For consistency, I've chosen to use the linked list code from my Linked Lists Tutorial. **Copy the code from there into a file and name it LinkedList.java.**

Below is my implementation of the Stack class. If you've looked at the code in my Queues tutorial, you would notice that the only difference is that 1s are changed to list.size(). Instead of returning the first element in the list, we return the last element in the list. In my implementation, the first element is the bottom and the last element is the top of the stack.

```
public class Stack {
        private LinkedList list;

        // Stack constructor
        public Stack()
        {
                // Create a new LinkedList.
                list = new LinkedList();
        }

        public boolean isEmpty()
        // Post: Returns true if the stack is empty. Otherwise, false.
        {
                return (list.size() == 0);
        }

        public void push(Object item)
        // Post: An item is added to the back of the stack.
        {
                // Append the item to the end of our linked list.
                list.add(item);
        }

        public Object pop()
        // Pre: this.isEmpty() == false
        // Post: The item at the front of the stack is returned and
        //         deleted from the stack. Returns null if precondition
        //         not met.
        {
                // Store a reference to the item at the front of the stack
                //   so that it does not get garbage collected when we
                //   remove it from the list.
                // Note: list.get(...) returns null if item not found at
                //   specified index. See postcondition.
                Object item = list.get(list.size());
                // Remove the item from the list.
                // My implementation of the linked list is based on the
                //   J2SE API reference. In both, elements start at 1,
                //   unlike arrays which start at 0.
```

Written by Pavel.