Stacks

All data structures we have seen so far are kinds of collections (Array, ArrayList). The elements of a collection are not necessarily supposed to be ordered (from the user's perspective), but all the implementations we have seen are ordered somehow, although the ordering may not reflect anything useful (e.g. is the order of the links in a web page really important?). Sometimes order is less important. Sometimes other things are more important in a collection.

Consider a web browser with Back and Forward buttons. How can the browser implement this? It must save the previous URLs in order to be able to jump back. What kind of collection should it use to store the URLs? Should it be ordered? [yes] What other properties are necessary?

(Every forward navigation puts a link on the stack; every time you go back, you remove the item from the stack.)

A (pure) stack is a data structure with the following set of allowable operations:

- create an empty stack
- check whether the stack is empty
- find the size of the stack (number of items in it)
- push (add something to the top of the stack)
- pop (remove the top item from the stack),
- peek or top (look at the top of the stack without removing it).

- LIFO discipline: last in, first out.
- Called stacks because of similarity to real-world stack in which adding and removing things in the middle is hard.

Uses for Stacks

- processing nested structure, like matching parentheses or palindromes or HTML tags.
- in daily activities, people are often in the middle of something when something else comes along that must be dealt with first, e.g., building a house, go to work on computer to
prepare a lecture, talk to visitor, answer the phonecall from David Greene, pipe starts dripping acid, fire alarm or play game, write report, give lecture, answer question, emergency (pipe starts dripping acid)
  o run-time stack. The RT stack keeps track of which methods have invoked which other methods (it is a stack of frames or activation records) and so it behaves like a stack in that stack frames are pushed on and popped off. But note that elements of the RT stack can be accessed by methods other than push and pop.
  o reversing the cars in a train.

Questions:
  o Should we be allowed to put null onto a stack of objects?
  o Can we add the same object twice to the stack?
  o Can a stack become full?
  o Which methods can be implemented as convenience methods? [peek and isEmpty]
  o What should happen if we pop or peek an empty stack?
    We could do nothing or we could throw an Exception.

• Exceptions
Exceptions are a built-in class that is used for exiting methods when an error occurs. E.g., NullPointerException. If an error occurs in a method so that it can't continue, it can "throw" an exception to the method that called it. Think of it as the second method throwing back to the caller the excuse for why it can't do what was expected of it.

FullStackException and EmptyStackException are just subclasses of Exception.

(https://docs.oracle.com/javase/tutorial/essential/exceptions/definition.html)

• Implementations of Stacks
  o built-in Stack class in java.util package
  o Linked list—nice because adding to the beginning and removing from the beginning are easy
  o Array—use an index for the top of the Stack (next empty one or last filled one?) Copy when full.
Implementation of Stacks using Arrays

Demo

Limitations of array-based stack

- Maximum size must be defined beforehand and cannot be changes
- Pushing a new element on a full stack causes an exception

Node-Based Stack

- A node is a data element
- Each pointing to the next node by means of a pointer/reference/link
- Each node is composed of data and a reference (in other words, a link) to the next node in the sequence. Null is the end.
- The principal benefit of a linked list over a conventional array is that the list elements can easily be inserted or removed without reallocation or reorganization of the entire structure because the data items need not be stored contiguously in memory
  - But They use more memory than arrays because of the storage used by their pointers.
- Insertions and deletions can be made anywhere in List

```
12 -| 99 -| 37 -| None
|
```

Inserting a new element

```
12 -| 99 -| 37 -| None
|
```

```
12 -| 99 -| newNode 37 -| None
|
```

```
12 -| 99 -| None
|
```

```
```

```
```
Deleting an element

Implementation

For the list implementation, we need a ListNode that has an Object as cargo:

```java
public class ListNode {

    Object data; // one or more data members are object of that class
    ListNode next; // pointer

    ListNode(Object d, ListNode n) { // constructor
        this.data = d;
        this.next = n;
    }
}
```

The instance variable here is called topNode and it just point to the element on the top of the stack:

```java
public class NodeStack implements Stack {

    private Node topNode; // top of stack

    public Stack() { // constructor
        topNode = null;
    }
}
```

The constructor creates an empty stack by initializing the instance variable to null. So naturally we can test for an empty stack.
public boolean isEmpty()
{
    return (topNode == null);
}

Now, for putting things on and taking things off the list
push is the same as add:

    public void push(Object obj)
    {
        topNode = new Node (obj, topNode);
    }

pop is the same as removeFirstNode, except that we have to play some games with a temporary variable:

    public Object pop()
    {
        if (topNode == null) throw new EmptyStackException();
        Object ret = topNode.data;
        topNode = topNode.next;
        return ret;
    }

peek is a no-brainer.

    public Object peek()
    {
        if (topNode == null) throw new EmptyStackException();
        return topNode.item;
    }

Note:
• Quiz – Friday as usual
• Reading Assignment: Linked Lists

Bonus or In-class Exercise
Your job: Examine the code below and write down the output
import java.util.*;

class StackEx {
    static void showpush(StackEx st, int a) {
        st.push(new Integer(a));
        System.out.println("push(" + a + ")");
        System.out.println("stack: " + st);
    }
    static void showpop(StackEx st) {
        System.out.print("pop -> ");
        Integer a = (Integer) st.pop();
        System.out.println(a);
        System.out.println("stack: " + st);
    }
    public static void main(String args[]) {
        StackEx st = new StackEx();
        System.out.println("stack: " + st);
        showpush(st, 42);
        showpush(st, 66);
        showpush(st, 99);
        showpop(st);
        showpop(st);
        showpop(st);
        try {
            showpop(st);
        } catch (EmptyStackException e) {
            System.out.println("empty stack");
        }
    }
}

stack: [ ]
push(42)
stack: [42]
push(66)
stack: [42, 66]
push(99)
stack: [42, 66, 99]
pop -> 99
stack: [42, 66]
pop -> 66
stack: [42]
pop -> 42
stack: [ ]
pop -> empty stack