JAVA NOTES

DATA STRUCTURES AND ALGORITHMS

Terry Marris July 2001

7 STACK INTERFACE

7.1 LEARNING OUTCOMES

By the end of this lesson the student should be able to

- describe the characteristic features of a stack
- explain the stack interface
- show how a stack may be used to solve simple problems

7.2 PRE-REQUISITES

The student should be comfortable with using arrays (Part B §2) and be familiar with the concept of an interface (Part A §18).

7.3 INTRODUCTION

There are times, when using a word processor, you would like to undo the last command. Look at Microsoft Word Edit Undo command for example.

Each command we issue is placed one on top of the other on a *stack*.

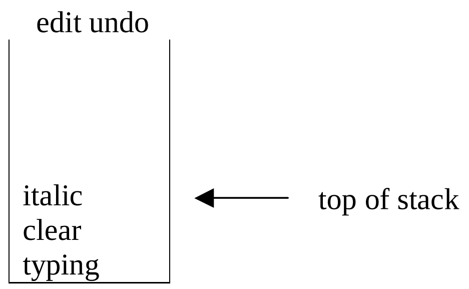


Figure 7.1 *A Stack of Edit Commands*

The last command issued is placed at the top of the stack. The last command issued is *italic* and is the first to be undone. If you want to undo the edit *typing* command you would first have to undo the *italic* and *clear* commands.

The essence of a stack is that items are added and removed one-by-one from the same end of a line.

We say that a stack is a linear data structure.

Stacks are used for writing compilers (program language translators) and for managing operating system interrupts.

7.4 THE STACK INTERFACE

push(), *pop()* and *peek()* are standard terms used when referring to stacks. The *push()* operation puts an item on a stack. The *pop()* operation removes an item from a stack. The *peek()* operation returns (a reference to) the item at the top of the stack; the contents of the stack remain unchanged. A stack interface is shown below.

```
/* Stack.java
   Terry Marris  19 July 2001
*/

public interface Stack {
  /* defines the standard stack operations push(Object),
     pop(), peek() and isEmpty().
  */


  public boolean isEmpty();
  /*  Returns true if this stack contains no objects.
      Returns false if this stack contains objects.
  */


  public String push(Object obj);
  /* If there is space on this stack, adds the given object
     to the top of this stack and returns "success".
  */


  public String pop();
  /*  If this stack is not empty, removes the item at
      the top of this stack and returns "success".
      If this stack is empty, returns "failure - stack empty"
      and nothing is removed.
  */


  public Object peek();
  /* Returns the object at the top of this stack if
     this stack is not empty,.
     If this stack is empty, returns null.
     In both cases the contents of this stack remain
     unchanged.
  */
}
```

7.5 REVIEW

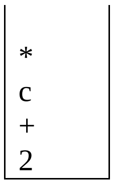
7.6 FURTHER READING

LEWIS J & LOFTUS W *Java software Solutions* pp 507

In the next lesson we see how to implement a stack using an array.

7.7 EXERCISES

1 Given a stack in the state shown below



draw the stack after the following operations have been performed

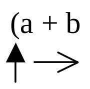
```
stack.pop();
stack.pop();
stack.push(new Character('&'));
stack.pop();
stack.push(new Character('#'));
stack.push(new character('#'));
stack.pop();
```

2 Specify a method *size()* that returns the number of elements currently on the stack.

3 By drawing a sequence of diagrams, explain how you could use a stack to determine if an arithmetic string expression is balanced with respect to its brackets. The following expressions are unbalanced:

(a + b	missing closing bracket,)
a + b)	missing opening bracket, (
(a + b) / c - d)	missing bracket

You could go through the string character by character



If you find an opening bracket, put it on the stack.
If you find a closing bracket, remove its matching closing bracket from the stack (What if the stack is empty?)
If you find any other character, ignore it.
If you reach the end of the string and the stack is empty, the brackets are balanced.
Hence write an implementation for *boolean isBalanced(String arithmeticExpression).*