


CSIS 3103

Stacks



The *Stack* ADT

A **stack** is a collection ADT where elements are maintained in the order they were inserted and accessed using the last-in-first-out (LIFO) protocol.

- Only the top element is accessible

Operations

1. *Peek*: Return the top element of a non-empty stack.
2. *Pop*: Delete and return the top element of a non-empty stack.
3. *Push*: Add a given element to the top of a stack.
4. *Empty*: Is the stack is empty?

```

<<interface>>
Stack
+ E peek()
+ E pop()
+ E push(E)
+ boolean empty()
                    
```

Specification of the Stack Abstract Data Type

Methods	Behavior
boolean empty()	Returns true if the stack is empty; otherwise, returns false .
E peek()	Returns the object at the top of the stack without removing it.
E pop()	Returns the object at the top of the stack and removes it.
E push(E obj)	Pushes an item onto the top of the stack and returns the item pushed.

A Stack of Strings

Jonathan
Dustin
Robin
Debbie
Rich

(a)

Dustin
Robin
Debbie
Rich

(b)

Philip
Dustin
Robin
Debbie
Rich

(c)

- "Rich" is the oldest element on the stack and "Jonathan" is the youngest
`String last = names.peek();` stores a reference to "Jonathan" in last
- `String temp = names.pop();` removes "Jonathan" and stores a reference to it in temp
- `names.push("Philip");` pushes "Philip" onto the stack

Applications of Stacks

- Call frame stack for method calls
- **Back** button on a Web browser
- Keyboard input buffer
- Reversing the order of a collection
- Checking for balanced parenthesis
- Evaluating arithmetic expressions
- Palindrome finder

Stack Application - Palindromes

Data Fields	Attributes
private String inputString	The input string.
private Stack<Character> charStack	The stack where characters are stored.
Methods	Behavior
public PalindromeFinder(String str)	Initializes a new PalindromeFinder object, storing a reference to the parameter str in inputString and pushing each character onto the stack.
private void fillStack()	Fills the stack with the characters in inputString.
private String buildReverse()	Returns the string formed by popping each character from the stack and joining the characters. Empties the stack.
public boolean isPalindrome()	Returns true if inputString and the string built by buildReverse have the same contents, except for case. Otherwise, returns false .

Testing Palindrome Finder

Inputs:

- the empty string (considered a palindrome)
- a single character (always a palindrome)
- multiple characters in a word
- multiple words
- different cases
- even-length strings
- odd-length strings

Balanced Parentheses

(a + b * (c / (d - e))) + (d / e)

- The problem is further complicated if braces or brackets are also used
- The solution is to use stacks!

Balanced Parentheses

Method	Behavior
public static boolean isBalanced(String expression)	Returns true if expression is balanced with respect to parentheses and false if it is not.
private static boolean isOpen(char ch)	Returns true if ch is an opening parenthesis.
private static boolean isClose(char ch)	Returns true if ch is a closing parenthesis.

Balanced Parentheses

Algorithm for method isBalanced

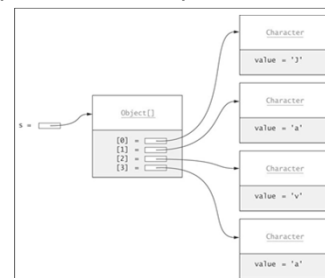
1. Create an empty stack of characters.
2. Assume that the expression is balanced (balanced is true).
3. Set index to 0.
4. **while** balanced is true and index < the expression's length
5. Get the next character in the data string.
6. **if** the next character is an opening parenthesis
7. Push it onto the stack.
8. **else if** the next character is a closing parenthesis
9. Pop the top of the stack.
10. **if** stack was empty or its top does not match the closing parenthesis
11. Set balanced to false.
12. Increment index.
13. Return true if balanced is true and the stack is empty.

Testing

- Several levels of nested parentheses
- Nested parentheses where corresponding parentheses are not of the same type
- Improperly nested parenthesis
- Unbalanced parentheses
- PITFALL: attempting to pop an empty stack will throw an EmptyStackException. Guard against this by either testing for an empty stack or catching the exception

Array Implementation of Stack

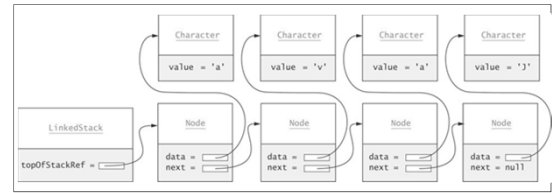
- Allocate an array
- Keep track of the top of the stack



Class Invariant for ArrayStack

-1 ≤ topOfStack < theData.length
and
 theData[0..topOfStack] contains
 the elements of the stack
and
 topOfStack ≥ 0 ⇒
 theData[topOfStack] is
 the most recently added item

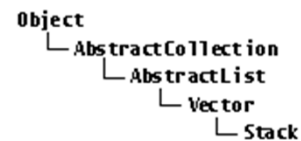
Single Linked List Implementation of Stack



Class Invariant for LinkedStack

topOfStackRef == null
or
 topOfStackRef references the
 most recently inserted node in
 a single-linked list

The java.util.Stack Class



Bad Idea: Extending a Vector is a poor choice for the stack implementation as all Vector methods are accessible

Another Stack Implementation

- Easiest implementation would be to use an ArrayList component for storing data
- All insertions and deletions are constant time regardless of the type of implementation
 - All insertions and deletions occur at one end by delegating methods