Stacks
Chapter 5

Supplementary Material:
1. Javadoc for java.util.Stack<E> (docs.oracle.com/javase/6/docs/api/java/util/Stack.html)
2. Stack trace in Java: http://java.sun.com/developer/technicalArticles/Programming/Stacktrace/

Copyright ©2012 by Pearson Education, Inc. All rights reserved
Motivation

• Have you used a stack yet?
  ▪ Program Stack

• A popular data-structure
  ▪ Supports Last In First Out (LIFO)

• History: en.wikipedia.org/wiki/Stack_%28abstract_data_type%29
  ▪ Patented in 1950s

• Used to implement run-time for programming languages
  ▪ Track calls to methods, functions, sub-routines
  ▪ Sometimes implemented in hardware

• Connection to Csci 1901 (Scheme)
  ▪ Stack can implement recursion
  ▪ Recursive operations easily implemented by LIFO Stack!
    • Reverse
    • Tower of Hanoi
    • …
Contents

• Specifications of the ADT Stack
• Using a Stack to Process Algebraic Expressions
  ▪ Checking for Balanced Delimiters in an Infix Algebraic Expression
  ▪ Transforming an Infix Expression to a Postfix Expression
  ▪ Evaluating Postfix Expressions
  ▪ Evaluating Infix Expressions
• The Program Stack
• Java Class Library: The Class Stack
Objectives

• Describe operations of ADT stack
• Use stack to solve problems
  ▪ Check matched parenthesis
  ▪ convert infix expression to postfix expression
  ▪ evaluate postfix expression
  ▪ evaluate infix expression
• Use a stack in a program
• Describe how Java run-time environment uses stack
  ▪ to track execution of methods
Specifications of a Stack

• Organizes entries according to order added
• All additions added to one end of stack
  ▪ Added to “top”
  ▪ Called a “push”
• Access to stack restricted
  ▪ Access only top entry
  ▪ Remove called a “pop”
Figure 5-1 Some familiar stacks
### Abstract Data Type Stack

#### Date
- A collection of objects in reverse chronological order and having the same data type

#### Operations

<table>
<thead>
<tr>
<th>Pseudocode</th>
<th>UML</th>
<th>Description</th>
</tr>
</thead>
</table>
| `push(newEntry)` | `+push(newEntry: T): void` | Task: Adds a new entry to the top of the stack.  
Input: `newEntry` is the new entry.  
Output: None. |
| `pop()` | `+pop(): T` | Task: Removes and returns the stack’s top entry.  
Input: None.  
Output: Returns either the stack’s top entry or, if the stack is empty before the operation, `null`. |

**ADT Stack**
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>peek()</td>
<td>Retrieves the stack’s top entry without changing the stack in any way.</td>
</tr>
<tr>
<td>isEmpty()</td>
<td>Detects whether the stack is empty.</td>
</tr>
<tr>
<td>clear()</td>
<td>Removes all entries from the stack.</td>
</tr>
</tbody>
</table>

**Arguments:**
- `peek()`: Takes no arguments.
- `isEmpty()`: Takes no arguments.
- `clear()`: Takes no arguments.

**Return Values:**
- `peek()`: Returns the stack’s top entry or `null` if the stack is empty.
- `isEmpty()`: Returns `true` if the stack is empty.
- `clear()`: Does not return a value.
Specify Generic Class **Stack** (listing 5-1, pp. 116)

```java
public interface StackInterface<T> {
    public void push(T newEntry); // Adds a new entry to the top of this stack.
    public T pop(); // Remove & return top entry if non-empty
    public T peek(); // Retrieves this stack's top entry if non-empty
    public boolean isEmpty(); // Return true if this stack is empty
    public void clear(); // Removes all entries from this stack
}
```

**Use Class Stack**

```java
StackInterface<String> stringStack = new OurStack<String>();
stringStack.push("Jim");
stringStack.push("Jess");
stringStack.push("Jill");
stringStack.push("Jane");
stringStack.push("Joe");

String top = stringStack.peek(); // returns "Joe"
System.out.println(top + " is at the top of the stack.");

String top = stringStack.pop(); // removes and returns "Joe"
System.out.println(top + " is removed from the stack.");

String top = stringStack.peek(); // returns "Jane"
System.out.println(top + " is at the top of the stack.");

String top = stringStack.pop(); // removes and returns "Jane"
System.out.println(top + " is removed from the stack.");
```

// Q? What is OurStack?
// Assume class OurStack implements interface StackInterface
Figure 5-2. A stack of strings after
(a) push adds Jim;
(b) push adds Jess;
(c) push adds Jill;
(d) push adds Jane;
(e) push adds Joe;
(f) pop retrieves and removes Joe;
(g) pop retrieves and removes Jane
Exercises

1. Show the contents of stack S after each stack operation:
   // Assume OurStack class implements StackInterface
   StackInterface<String> S = new OurStack<String>();
   S.push("A") ; S.push("B"); S.push("C"); S.pop(); S.pop();
   S.push("D") ; S.push("E"); S.push("F"); S.pop();
   S.push("G") ; S.pop(); S.pop(); S.pop();

2. Show the contents of stack S after each stack operation:
   // Assume OurStack class implements StackInterface
   StackInterface<String> S = new OurStack<String>();
   S.push("A") ; S.push("B"); S.push("C"); S.push("D");
   S.pop(); S.pop(); S.pop(); S.pop();

3. Which collections behave like stacks (LIFO)?
   (a.) Sheets in a printer paper-tray  (b.) people standing in a check-out line
   (c.) Trays in a cafeteria    (d.) Discs on a rod in “Tower of Hanoi”
   (See Demo at http://en.wikipedia.org/wiki/Tower_of_Hanoi)

4. Would it make sense to call a stack
   (a.) a last in last out structure? (b.) a first in last out structure ?
   (c.) a first in first out structure
Class `java.util.Stack<T>` in Java library

- [http://docs.oracle.com/javase/6/docs/api/java/util/Stack.html](http://docs.oracle.com/javase/6/docs/api/java/util/Stack.html)
- Implements many interfaces, e.g., `Collection<E>`, `List<E>`, ...
- Differences from our `StackInterface` are highlighted
  - `Stack() ; // constructor - creates an empty stack`
  - `public T push(T item);`
  - `public T pop();`
  - `public T peek();`
  - `public boolean empty();`
  - `public boolean empty();`
  - `int search(Object o)`
  - Inherited methods

// Recall textbook definition
public interface `StackInterface < T >` {
    public void push (T newEntry); // Adds a new entry to the top of this stack.
    public T pop (); // Remove & return top entry if non-empty
    public T peek (); // Retrieves this stacks top entry if non-empty
    public boolean isEmpty (); // Return true if this stack is empty
    public void clear (); // Removes all entries from this stack
}
FIGURE 5-13 The program stack at three points in time: (a) when `main` begins execution; (PC is the program counter)
FIGURE 5-13 The program stack at three points in time: 
(b) when methodA begins execution; (PC is the program counter)
FIGURE 5-13 The program stack at three points in time: 
(c) when `methodB` begins execution; (PC is the program counter)
Using a Stack to Process Algebraic Expressions

- Use of parentheses - must be balanced
  - Positive Examples:
    - $A \{b \frac{c(d + e)}{2} - f \} + 1$
    - $\{ [ ( ) ] \}$
  - Negative Examples:
    - $\{ [1 * (2 + 3) 4 ] + 5 \}$
    - $\{ [ ( ) ] \}$

- Use stacks to evaluate parentheses usage
  - Scan expression
  - Push symbols
  - Pop symbols

- See java code in listing 5.2 (pp. 122-123)

- Test the code with
  - $\{ [ () ] \}$
  - $[ () ]$
  - $\{ [ ] ] \}$
Figure 5-3 The contents of a stack during the scan of an expression that contains the balanced delimiters \{ *[ ( ) ] ] *\}
public class BalanceChecker {

    public static boolean checkBalance (String expression) {
        StackInterface<Character> openDelimiterStack = new OurStack<Character>();
        boolean isBalanced = true;
        int index = 0;
        char nextCharacter = ' ';
        for (; isBalanced && (index < expression.length()); index++) {
            nextCharacter = expression.charAt(index);
            switch (nextCharacter) {
                case '(':
                case '[':
                case '{':
                    openDelimiterStack.push(nextCharacter);
                    break;
                case ')':
                case ']':
                case '}':
                    if (openDelimiterStack.isEmpty()) isBalanced = false;
                    else {
                        char openDelimiter = openDelimiterStack.pop();
                        isBalanced = isPaired(openDelimiter, nextCharacter);
                    }
                    break;
                default:
                    break;
            }
        }
        if (!openDelimiterStack.isEmpty()) isBalanced = false;
        return isBalanced;
    }

    private static boolean isPaired (char open, char close) {
        return (open == '(' && close == ')') ||
               (open == '[' && close == ']') ||
               (open == '{' && close == '}');
    }

    } // end BalanceChecker class
Figure 5-4 The contents of a stack during the scan of an expression that contains the unbalanced delimiters \{ [ ( ] ) \}
Figure 5-5 The contents of a stack during the scan of an expression that contains the unbalanced delimiters [ ( ) ] }
Figure 5-6 The contents of a stack during the scan of an expression that contains the unbalanced delimiters { [ ( ) ]}
End

Chapter 5