Stacks

- References:
  - Text book : Chapter 5 and Chapter 6

1. Overview

- Definition : A stack is an ordered list in which insertions and deletions are made at one end called "top". It is also known as Last-In-First-Out (LIFO) list.

- Example Applications :

  Stack of books
  Page-visited history in IE (use ‘back’ button)
  Undo sequence in text editor
  Program function stack

- ADT Stack Operations (interface)

```
public interface StackInterface<T>
{
    /** Task: Adds a new entry to the top of the stack.
     *  @param newEntry  an object to be added to the stack */
    public void push(T newEntry);

    /** Task: Removes and returns the stack調 top entry.
     *  @return either the object at the top of the stack or, if the
     *          stack is empty before the operation, null */
    public T pop();

    /** Task: Retrieves the stack’s top entry.
     *  @return either the object at the top of the stack or null if
     *          the stack is empty */
    public T peek();
```

```
    /** Task: Detects whether the stack is empty.
     *  @return true if the stack is empty */
    public boolean isEmpty();

    /** Task: Removes all entries from the stack */
    public void clear();
} // end StackInterface
```

- Example usage of stack

```
StackInterface<String> myStack = new LinkedStack<String>();
myStack.push("Jim");
myStack.push("Jess");
myStack.push("Jill");
myStack.push("Jane");
myStack.push("Joe");

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

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

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

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

2. Applications

- Application : Checking for balanced parentheses, blackest and braces

  a{b[c(d+e)/2-f]+1} → {[()]} // skip all non-delimeter chars

General Strategy:

Scan expression from left to right, use a stack to hold open delimiters. For each close delimiter, pop 1st open delimiter from stack and match it close delimiter

Example: { [( ) ( )] () } // balance!
[( ]) // not balance

// not balance! Special case, stack not empty at the end
{ [( ) ( )] ()
// not balance! Special case, stack empty before last }
{ [( ) ( )] () }}

Java Program:

```
public class BalanceChecker
{
    /** Task: Decides whether the parentheses, brackets, and braces
     *       in a string occur in left/right pairs.
     *  @param expression  a string to be checked
     *  @return true if the delimiters are paired correctly */
    public static boolean checkBalance(String expression)
    {
        StackInterface<Character> openDelimiterStack =
                                    new LinkedStack<Character>();

        int characterCount = expression.length();
        boolean isBalanced = true;
        int index = 0;
        char nextCharacter = ' ';
```

```
    for (; isBalanced && (index < characterCount); 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);
                } // end if
                break;

            default: break;
        } // end switch
    } // end for

    if (!openDelimiterStack.isEmpty())
       isBalanced = false;

    return isBalanced;
} // end checkBalance
```

```
    /** Task: Detects whether two delimiters are a pair of
     *        parentheses, brackets, or braces.
     *  @param open   a character
     *  @param close  a character
     *  @return true if open/close form a pair of parentheses, brackets,
     *               or braces */
    private static boolean isPaired(char open, char close)
    {
        return (open == '(' && close == ')') ||
               (open == '[' && close == ']') ||
               (open == '{' && close == '}');
    } // end isPaired
} // end BalanceChecker


// sample usage:

String expression = "a {b [c (d + e)/2 - f] + 1}";
boolean isBalanced = BalanceChecker.checkBalance(expression);
if (isBalanced)
        System.out.println(expression + " is balanced");
else
        System.out.println(expression + " is not balanced");
```

- Application: The program stack

  - When a method is called
    Runtime environment creates activation record
    Shows method's state during execution

  - Activation record pushed onto the program stack (Java stack)
    Top of stack belongs to currently executing method
    Next method down is the one that called current method

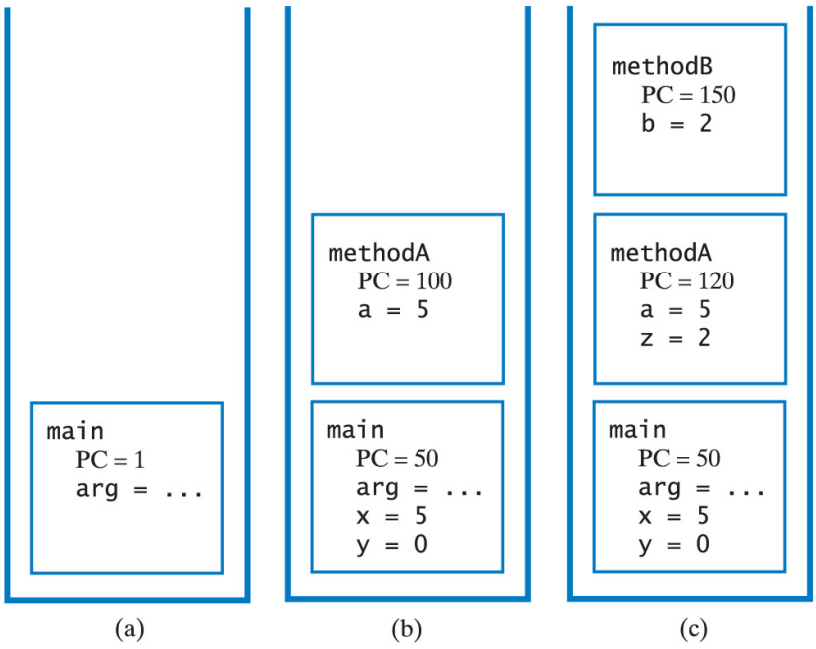
    The following example shows: main() → methodA() → methodB()
    Each activation record: local variables & pc (current instruction addr)

```
1     public static
      void main(string[] arg)
      {
         . . .
         int x = 5;
50       int y = methodA(x);
         . . .
      } // end main

100   public static
      int methodA(int a)
      {
         . . .
         int z = 2;
120      methodB(z);
         . . .
         return z;
      } // end methodA

150   public static
      void methodB(int b)
      {
         . . .
      } // end methodB
```

Program



Program stack at three points in time (PC is the program counter)

- Java class library: java.util.Stack

  http://download.oracle.com/javase/6/docs/api/java/util/Stack.html

  public class Stack<E> extends Vector<E>

  extend from class vector → inherits all methods from vector

  additional methods:

	Stack() Create an empty Stack.
boolean	empty() Tests if this stack is empty.
E	peek() Looks at the object at the top of this stack without removing it from the stack.
E	pop() Removes the object at the top of this stack and returns that object as the value of this function.
E	push(E item) Pushes an item onto the top of this stack.
int	search(Object o) Returns the position (starting with 1) where an object is on this stack.

Example usage:

```
$ cat StackDemo.java

import java.util.*;
public class StackDemo{

  public static void main(String[] args) {

    Stack<Object> stack=new Stack<Object>();
    stack.push(new Integer(10));
    stack.push("hello world");
```

```
    System.out.println("============================\n");
    System.out.println("The contents of Stack is: " + stack);
    System.out.println("The size of a Stack is: " + stack.size());
    System.out.println("Check if stack empty: " + stack.empty());
    System.out.println("Peek top of a Stack: " + stack.peek());
    System.out.println("Search 10 from Stack: " + stack.search(new Integer(10)));

    System.out.println("============================\n");
    System.out.println("The item poped out is: " + stack.pop());
    System.out.println("The item poped out is: " + stack.pop());
    System.out.println("============================\n");

    System.out.println("The contents of Stack is: " + stack);
    System.out.println("The size of a Stack is: " + stack.size());
    System.out.println("Check if stack empty: " + stack.empty());
    System.out.println("Search 10 from Stack: " + stack.search(new Integer(10)));
    System.out.println("============================\n");
  }
}
```

```
$ java StackDemo
============================

The contents of Stack is: [10, hello world]
The size of a Stack is: 2
Check if stack empty: false
Peek top of a Stack: hello world
Search 10 from Stack: 2
============================

The item poped out is: hello world
The item poped out is: 10
============================

The contents of Stack is: []
The size of a Stack is: 0
Check if stack empty: true
Search 10 from Stack: -1
============================
```

3. Stack Implementations

It is a special case for general lists. Implementations are simple.

A Linked Implementation

Use a linked list, where the first node should reference the stack's top.

```
public class LinkedStack < T > implements StackInterface < T >
{
   private Node topNode; // references the first node in the chain

   public LinkedStack ()
   {
      topNode = null;
   } // end default constructor

   public void push (T newEntry)
   {
      Node newNode = new Node (newEntry, topNode);
      topNode = newNode;
   } // end push

   public T peek ()
   {   // return null if it is empty
      T top = null;
      if (topNode != null)
         top = topNode.getData ();
      return top;
   } // end peek

   public T pop ()
   {  // return null if it is empty
      T top = peek ();
      if (topNode != null)
         topNode = topNode.getNextNode ();
      return top;
```

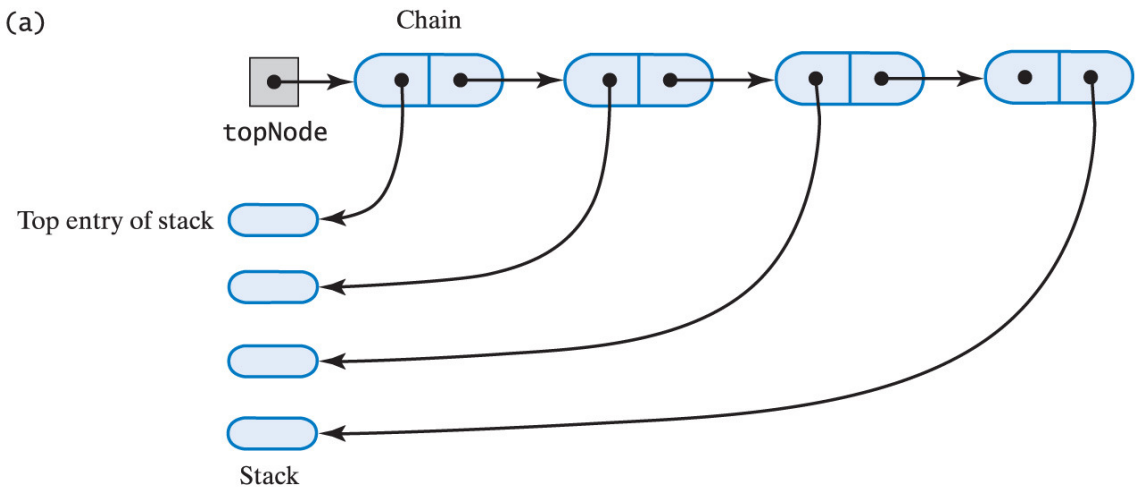
```
    } // end pop

    public boolean isEmpty ()
    {
       return topNode == null;
    } // end isEmpty

    public void clear ()
    {
       topNode = null;
    } // end clear

    private class Node
    {
       private T data; // entry in stack
       private Node next; // link to next node
       // Constructors and the methods getData, setData,
       // getNextNode, andsetNextNode are here.
    } // end Node
} // end LinkedStack
```

Figure illustrate linked list nodes. Each node holds a stack object :



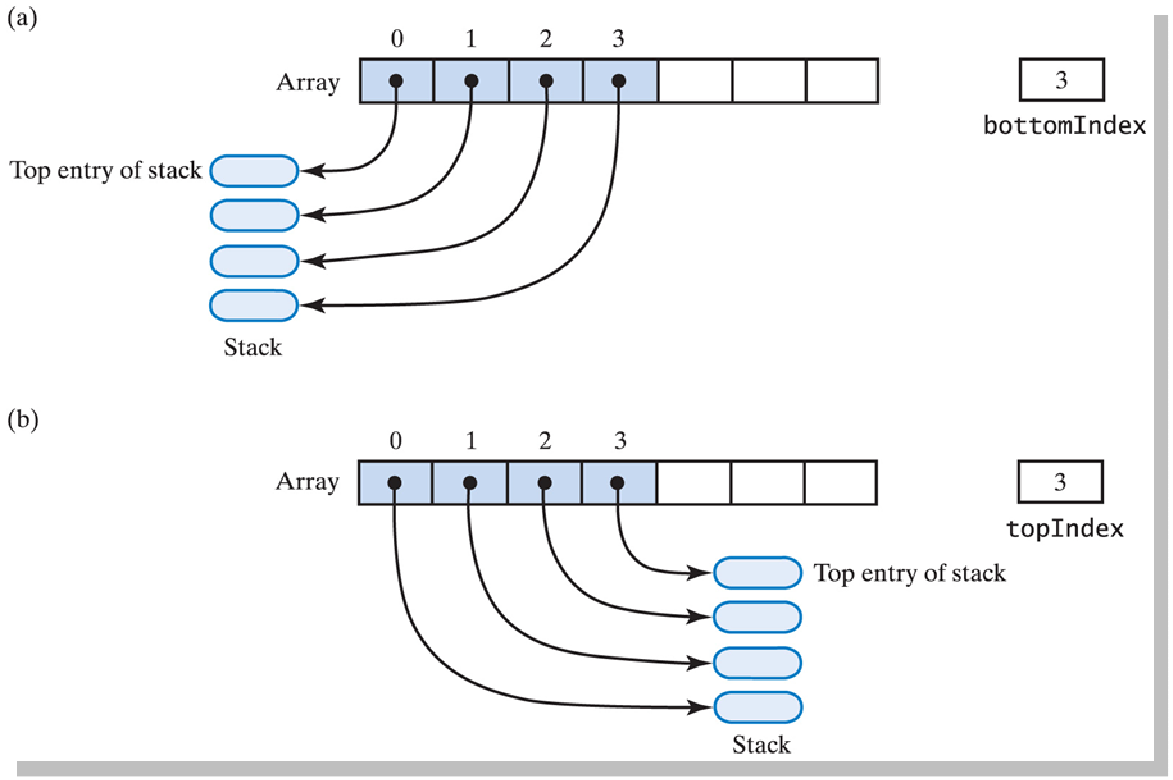
An Array-Based Implementation

- When using an array to implement a stack
  The array's first element should represent the bottom of the stack
  The last occupied location in the array represents the stack's top

- This avoids shifting of elements of the array if it were done the other way around

  Use 1st entry of an array as stack top, see (a) below.
  push() and pop() require to move stack objects

  Use last entry of an array as stack top, see (b) below.
  push() and pop() do not require to move stack objects



```
public class ArrayStack < T > implements StackInterface < T >
{
      private T [] stack; // array of stack entries
      private int topIndex; // index of top entry
      private static final int DEFAULT_INITIAL_CAPACITY = 50;

      public ArrayStack ()
      {
         this (DEFAULT_INITIAL_CAPACITY);
      } // end default constructor


      public ArrayStack (int initialCapacity)
      {
        // the cast is safe because the new array contains null entries
        @SuppressWarnings("unchecked")
        T[] tempStack = (T[])new Object[initialCapacity];
        stack = tempStack;
        topIndex = -1;
      } // end constructor


      public void push (T newEntry)
      {
         topIndex++;
         if (topIndex >= stack.length) // if array is full,
            doubleArray (); // expand array
         stack [topIndex] = newEntry;
      } // end push


      public T peek ()
      {
         T top = null;
         if (!isEmpty ())
            top = stack [topIndex];
         return top;
      } // end peek
```

```
    public T pop ()
    {
       T top = null;
       if (!isEmpty ())
       {
          top = stack [topIndex];
          stack [topIndex] = null;
          topIndex--;
       } // end if
       return top;
    } // end pop


    public boolean isEmpty ()
    {
       return topIndex < 0;
    } // end isEmpty
} // end ArrayStack
```

A Vector-Based Implementation

- Java class library: Vector. It uses dynamic array expansion
- http://download.oracle.com/javase/6/docs/api/java/util/Vector.html

```
public Vector()
public Vector(int initialCapacity)
public boolean add(T newEntry)
public T remove(int index)
public void clear()
public T lastElement()
public boolean isEmpty()
public int size()
```

- As in array:
  the vector's first element should represent the bottom of the stack
  The last occupied location in the vector represents the stack's top

- Create a vector (instead of an array). Use vector methods to implement stack operations

```
import java.util.Vector;
public class VectorStack < T > implements StackInterface < T >
{
   private Vector < T > stack; // last element is the top entry in stack
   public VectorStack ()
   {
      stack = new Vector < T > (); // vector doubles in size if necessary
   } // end default constructor

   public VectorStack (int initialCapacity)
   {
      stack = new Vector < T > (initialCapacity);
   } // end constructor
```

```
    public void push (T newEntry)
    {
       stack.add (newEntry);
    } // end push

    public T peek ()
    {
       T top = null;
       if (!isEmpty ())
          top = stack.lastElement ();
       return top;
    } // end peek

    public T pop ()
    {
       T top = null;
       if (!isEmpty ())
       {
          top = stack.lastElement ();
          stack.remove (stack.size () - 1);
       } // end if
       return top;
    } // end pop

    public boolean isEmpty ()
    {
       return stack.isEmpty ();
    } // end isEmpty

    public void clear ()
    {
       stack.clear ();
    } // end clear

} // end VectorStack
```