Specifications of the ADT Stack

- Organizes entries according to order in which the have been added
- Additions are made to one end → The Top
- The item most recently added is always on the top

Based on:
Data Structures and Abstractions with JAVA

From DS and Abstract Carrano & Savitch 2003. From Goodrich, Tamassia, 2004
Specifications of the ADT Stack

Specification of a stack of objects:

```java
public interface StackInterface {
    public void push(Object newEntry);
    /** 1) Task: Adds a new entry to the top of the stack. 
    * @param newEntry is of type object */
    public Object pop();
    /** 2) Task: Removes and returns the top of the stack.
    * @return either the object at the top of the stack or
    *        null if the stack was empty */
    public Object peek();
    /** 3) Task: Retrieves the top of the stack. Top()
    * @return either the object at the top of the stack or
    *        null if the stack is empty */
    public boolean isEmpty();
    /** 4) Task: Determines whether the stack is empty.
    * @return true if the stack is empty */
    public void clear();
    /** 5) Task: Removes all entries from the stack */
} // end StackInterface
```

Applications of Stacks

• Direct applications
  – Page-visited history in a Web browser
  – Undo sequence in a text editor
  – Chain of method calls in the Java Virtual Machine

• Indirect applications
  – Auxiliary data structure for algorithms
public static void countDown(int n)
{
    System.out.println(n);
    if (n > 1)
        countDown(n - 1);
}

The stack of activation records during the execution of a call to \texttt{countDown(3)}...
The Java Virtual Machine (JVM) keeps track of the chain of active methods with a stack. When a method is called, the JVM pushes on the stack a frame containing:
- Local variables and return values
- Program counter, keeping track of the statement being executed
When a method ends, its frame (Activation Record) is popped from the stack and control is passed to the method on top of the stack.

From Goodrich, Tamassia, 2004
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

Example 2

The program stack at 3 points in time: (a) when `main` begins execution; (b) when `methodA` begins execution, (c) when `methodB` begins execution.

From DS and Abstr. Carrano & Savitch
Using a Stack to Process Algebraic Expressions

**Infix expressions**
- Binary operators appear *between* operands
  
  \[ a + b \]

**Prefix expressions**
- Binary operators appear *before* operands
  
  \[ + a b \]

**Postfix expressions**
- Binary operators appear *after* operands
  - Easier to process (no need for parentheses nor precedence)
  
  \[ a b + \]

Checking for Balanced (), [], {} or HTML Tags

The contents of a stack during the scan of an expression that contains the balanced delimiters \{ [ ( ) ] \}

From DS and Abstr. Carrano & Savitch
Checking for Balanced ( ), [ ], { }

The contents of a stack during the scan of an expression that contains the unbalanced delimiters { [ ( ) ] }

Accepting Languages such as:
- \( L = \{ a^n b^n : n > 1 \} \)
- \( L = w \in \{0,1\}^* \)

Later in Theory of Computation

Using [from DS and Abstr. Carrano & Savitch]

Pushdown Automata
Evaluation of algebraic Expression

\[ 20 - 2 \times 2^3 = 20 - 2 \times 2^3 \quad \text{has higher precedence} \]
\[ = 20 - 16 \quad \text{has higher precedence} \]
\[ = 4 \quad \text{has less precedence} \]

Same Precedence: Left to Right *, / or +, – :
- \[ 8 - 4 + 2 = (8 - 4) + 2 = 4+2 = 6 \]
- and not:
\[ 8 - 4 + 2 \neq 8 - (4 + 2) = 2 \]
- Right to Left
\[ 2^2^3 = 2^2^3 = 2^8 \neq (2^2)^3 = 4^3 \]
- Parentheses in an expression override the normal Operator Precedence

Infix-to-Postfix Algorithm

<table>
<thead>
<tr>
<th>Symbol in Infix</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operand</td>
<td>Append to end of output expression</td>
</tr>
<tr>
<td>Operator ^</td>
<td>Push ^ onto stack (Precedence R to L) as indication for Exponentiation</td>
</tr>
<tr>
<td>Operator +,*, /</td>
<td>Pop operators from stack, append to output expression until stack is empty or top has lower precedence than new operator. Then push new operator onto stack (Precedence L to R)</td>
</tr>
</tbody>
</table>

- Open parenthesis: Push ( onto stack
- Close parenthesis: Pop operators from stack, append to output expression until we pop an open parenthesis. Discard both parentheses.

From DS and Abstr. Carrano & Savitch
### Transforming Infix to Postfix

**Example:**

<table>
<thead>
<tr>
<th>Next Character</th>
<th>Postfix</th>
<th>Operator Stack (bottom to top)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>a</td>
<td>+</td>
</tr>
<tr>
<td>b</td>
<td>ab</td>
<td>+</td>
</tr>
<tr>
<td>*</td>
<td>ab</td>
<td>+ *</td>
</tr>
<tr>
<td>c</td>
<td>abc</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>abc*</td>
<td></td>
</tr>
<tr>
<td></td>
<td>abc*+</td>
<td></td>
</tr>
</tbody>
</table>

Converting the infix expression $a + b * c$ to postfix form

### Transforming Infix to Postfix

**Example:**

<table>
<thead>
<tr>
<th>Next Character</th>
<th>Postfix</th>
<th>Operator Stack (bottom to top)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>−</td>
<td>a</td>
<td>−</td>
</tr>
<tr>
<td>b</td>
<td>ab</td>
<td>−</td>
</tr>
<tr>
<td>+</td>
<td>ab−</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>ab−</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>ab−c</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>ab−c+</td>
<td></td>
</tr>
</tbody>
</table>

Converting infix expression to postfix form: $a - b + c$

From DS and Abstr. Carrano & Savitch
Transforming Infix to Postfix

Example:

<table>
<thead>
<tr>
<th>Next Character</th>
<th>Postfix</th>
<th>Operator Stack (bottom to top)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>^</td>
<td>a</td>
<td>^</td>
</tr>
<tr>
<td>b</td>
<td>ab</td>
<td>^</td>
</tr>
<tr>
<td>^</td>
<td>ab</td>
<td>^ ^</td>
</tr>
<tr>
<td>c</td>
<td>abc</td>
<td>^ ^</td>
</tr>
<tr>
<td>^ ^</td>
<td>abc</td>
<td>^ ^</td>
</tr>
</tbody>
</table>

Steps to convert the infix expression $a / b * (c + (d - e))$ to postfix form.

Converting infix expression to postfix form: $a ^ b ^ c$
Evaluating Postfix Expression

- Evaluating a postfix expression requires no rule of operator precedence (the order of its operators and operands determines the order of operation)
- Postfix expression contain no parentheses, which would simplify the Evaluation process

- When scanning the Postfix Expression, save operands in the Stack (i.e. push them in the Stack) until finding the operators that apply to them.
- Then pop two values from the Stack, evaluate them and push result in the Stack
- Repeat this process till finishing reaching the end of the Postfix expression

Example:

The stack during the evaluation of the postfix expression $a \ b /$ when $a$ is 2 and $b$ is 4. 4 is the most recently saved in the stack.
Transforming Infix to Postfix

Example:

The stack during the evaluation of the postfix expression

\[ \text{a b + c /} \]

which corresponds to the infix Expression

\[ \frac{(a+b)}{c} \].

Let \( a = 2 \), \( b = 4 \) and \( c = 3 \), i.e. \( \frac{2+4}{3} = 2 \).

Java Class Library: The Class `Stack`

- The Package `java.util` contains the class `Stack`

  ```java
  public Object push(Object item);
  public Object pop();
  public Object peek();
  public boolean empty();
  public int search(Object desiredItem);
  public Iterator iterator();
  ```

- An Iterator is a program component that enables the user to traverse a collection of Data such as a List, Stack …

Project 1: Implement the algorithm evaluating Postfix expressions within a class Converting infix Expression in Postfix Form.

Project 2: write a Java Program that is capable of Matching tags in an HTML document (select any HTML Document That contains Arabic Texts)

Hint: import. Util.StringTokenizer; Use our Stack implementation or Java class Stack
Stack Implementations

A Linked-Based Implementation

Using a chain of linked nodes to implement a stack

The first node should reference the stack's top

A chain of linked nodes that implements a stack.

From DS and Abstr. Carrano & Savitch, 2003
### A Linked Implementation

- **Data field and constructor**

```java
public class LinkedStackL implements StackInterface{
    private Node topNode; // references first node in chain
    public LinkedStackL()
    {
        topNode = null;
    } // end default constructor
    ... 
}
```

- **We need also the class Node**

```java
private class Node
{
    private Object data; // entry in stack, data portion
    private Node next; // link to next node
    private Node (Object dataPortion);
    {
        data = dataPortion;
        next = null;
    }
    private Node (Object dataPortion, Node nextNode) // Private or Public
    {
        data = dataPortion;
        next = nextNode;
    }
} // end Node
```

---

**Pushing: using one end; i.e. the topNode**

(a) A new node that references the top of the stack

(b) The new node is now at the top of the stack.
A Linked Implementation

- Adding to top: push

```java
public void push (Object NewEntry) {
    Node newNode = new Node(newEntry, topNode);
    topNode = newNode;
}
```

Deleting using the topNode: public Object pop()

The stack after the first node in the chain is deleted.

From DS and Abstr. Carrano & Savitch, 2003
A Linked Implementation

Deleting using the topNode: public Object pop()

```java
public Object pop()
{
    Object top = null;
    if (topNode != null)
    {
        top = topNode.data;
        topNode = topNode.next;
    }
    return top;
}
```

Peek: Retrieving top Entry: public Object peek()

```java
public Object peek()
{
    Object top = null;
    if (topNode != null)
    {
        top = topNode.data;
    }
    return top;
}
```

- public void clear() & public boolean isEmpty()

```java
public void clear()
{
    topNode = null;
}

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

An array that implements a stack. Its **first location references**
- the top of the stack (Problems: adding or removing then **shifting** all Entries up)
- the bottom of the stack.

Problems of Array Based Implementation are in general:
- **unused Memory**
- if full Double Array; i.e. expanding the Array means even more unused Memory
An Array-Based Implementation

• Data fields and constructors

```java
public class ArrStack implements StackInterface {
    private Object[] stack; // array of stack entries
    private int topIndex; // index of top entry
    private static final int DEFAULT_SIZE = 40;

    public ArrStack() {
        stack = new Object[DEFAULT_SIZE];
        topIndex = -1;
    }

    public ArrStack(int maxSize) {
        stack = new Object[maxSize];
        topIndex = -1;
    }

    ... // additional methods
}
```

**Indication for Empty**, the initial value for the top:
Initial value is -1

```java
public void push(Object NewEntry) {
    topIndex++;
    if (topIndex >= stack.length) // if the stack is full
        doubleArray(); // expand the Array
    stack[topIndex] = newEntry;
}
```

topIndex is after the first pushing = 0
public Object pop()
{
    Object top = null;
    if (!isEmpty())
    {
        top = stack[topIndex];
        stack[topIndex] = null;
        topIndex --;
    }
    return top;
}
An Array-Based Implementation

```java
public Object peek()
{
    Object top = null;
    if (!isEmpty())
        top = stack[topIndex];
    return top;
}

public boolean isEmpty()
{
    return topIndex < 0;
}

public void clear()
for (; topIndex > -1; topIndex --)
    stack[topIndex] = null;
}
```
### Data fields and constructors

```java
import java.util.Vector;
public class VectorStack implements StackInterface {
    public VectorStack()
        { stack = new Vector(); // vector doubles in size if necessary
        }
    public VectorStack(int maxSize)
        { stack = new Vector(maxSize);
        }
    ...
}
```

See the next Exercise in the next page

### Exercises:

Implement the stack in three different ways:

a) A Linked-Based implementation
b) An Array-Based implementation
c) Vector-Based implementation

Hint: Use the Vector methods

1) `public void addElement(Object newElement)`
2) `public Object lastElement()`
3) `public void removeElementAt(int index)`
4) `public boolean isEmpty()`
5) `public void removeAllElement()`