2. Stack and Queue

Part 1: Stack

Stack

Objectives

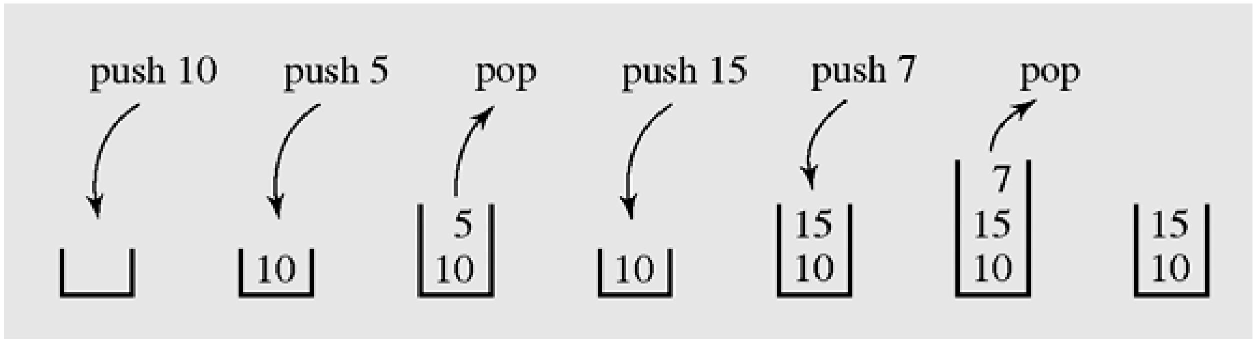
- Stacks
- Array-based stack
- Stack implemented by a singly linked list
- Stack class in java.util

What is a stack?

- A **stack** is a linear data structure that can be accessed only at one of its ends for storing and retrieving data
- A stack is a Last In, First Out (LIFO) data structure
- Anything added to the stack goes on the “top” of the stack
- Anything removed from the stack is taken from the “top” of the stack
- Things are removed in the reverse order from that in which they were inserted

Operations on a stack

- The following operations are needed to properly manage a stack:
  - *clear()* — Clear the stack
  - *isEmpty()* — Check to see if the stack is empty
  - *push(el)* — Put the element *el* on the top of the stack
  - *pop()* — Take the topmost element from the stack
  - *top()* — Return the topmost element in the stack without removing it



Operations on a stack

Stack Exceptions

- Attempting the execution of an operation of ADT may sometimes cause an error condition, called an exception
- Exceptions are said to be “thrown” by an operation that cannot be executed
- In the Stack ADT, operations pop and top cannot be performed if the stack is empty
- Attempting the execution of pop or top on an empty stack throws an StackEmptyException.

Applications of Stacks

- Stacks are used for:
  - Any sort of nesting (such as parentheses)
  - Evaluating arithmetic expressions (and other sorts of expression)
  - Implementing function or method calls
  - Keeping track of previous choices (as in backtracking)
  - Keeping track of choices yet to be made (as in creating a maze)
  - Undo sequence in a text editor.
  - Auxiliary data structure for algorithms
  - Component of other data structures

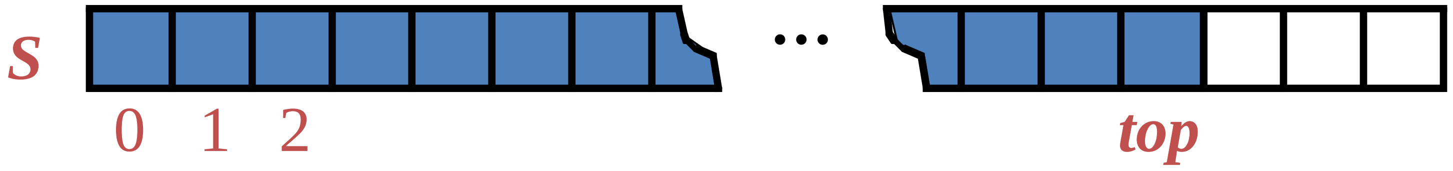
Stack in computer memory

- How does a stack in memory actually work?
  - Each time a method is called, an activation record (AR) is allocated for it. This record usually contains the following information:
    - Parameters and local variables used in the called method.
    - A dynamic link, which is a pointer to the caller's activation record.
    - Return address to resume control by the caller, the address of the caller's instruction immediately following the call.
    - Return value for a method not declared as void. Because the size of the activation record may vary from one call to another, the returned value is placed right above the activation record of the caller.
  - Each new activation record is placed on the top of the run-time stack
  - When a method terminates, its activation record is removed from the top of the run-time stack
  - Thus, the first AR placed onto the stack is the last one removed

AR is also called Stack frame

Array-based Stack - 1

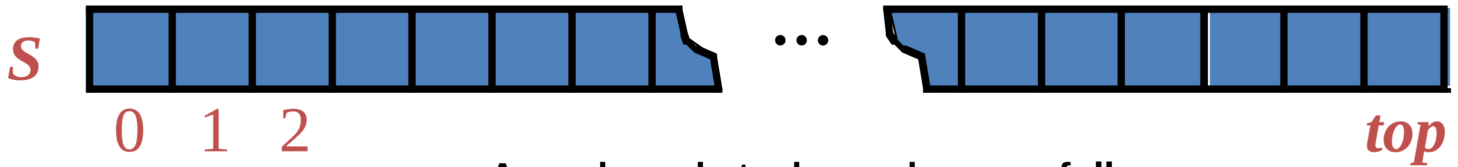
- A simple way of implementing the Stack ADT (abstract data type) uses an array
- We add elements from left to right
- A variable top keeps track of the index of the top element



Array-based stack

Array-based Stack - 2

- The array storing the stack elements may become full
- A push operation will then throw a FullStackException
  - Limitation of the array-based implementation
  - Not intrinsic to the Stack ADT



Array-based stack may become full

Array implementation of a stack

```
class ArrayStack
  {protected  Object [] a; int top, max;
   public ArrayStack()
     { this(50);
     }
   public ArrayStack(int max1)
     { max = max1;
       a =  new Object[max];
       top = -1;
     }
   protected  boolean grow()
     { int max1 = max + max/2;
       Object [] a1 = new Object[max1];
       if(a1 == null) return(false);
       for(int i =0; i<=top; i++) a1[i] = a[i];
       a = a1;
       return(true);
     }
   public boolean isEmpty()
     { return(top==-1);}
```

```
   public boolean isEmpty()
     { return(top==-1);}
   public boolean isFull()
     { return(top==max-1);}
   public void clear()
     { top=-1;}
   public void push(Object x)
     { if(isFull() && !grow()) return;
        a[++top] = x;
     }
   Object top() throws EmptyStackException
     { if(isEmpty()) throw new EmptyStackException();
        return(a[top]);
     }
   public Object pop() throws EmptyStackException
     { if(isEmpty()) throw new EmptyStackException();
        Object x = a[top];
        top--;
        return(x);
     }
```

Linked implementation of a stack

```
class Node
  { public Object info;
    public Node  next;
    public Node(Object x, Node p)
      { info=x; next=p; }
    public Node(Object x)
      { this(x,null); }
  };
```

```
class LinkedStack
  { protected Node head;

    public LinkedStack()
      { head = null; }

    public boolean isEmpty()
      { return(head==null);}

    public void push(Object x)
      { head = new Node(x,head);
      }
```

```
Object top() throws EmptyStackException
     { if(isEmpty()) throw new EmptyStackException();
       return(head.info);
     }

   public Object pop() throws EmptyStackException
     { if(isEmpty()) throw new EmptyStackException();
       Object x = head.info;
       head=head.next;
        return(x);
     }
```

DREAM OF INNOVATION

Implementing a stack using ArrayList & LinkedList classes in Java

```
import java.util.*;
class MyStack
 {ArrayList h;
  MyStack() {h = new ArrayList();}
  boolean isEmpty()
    {return(h.isEmpty());}
  void push(Object x)
    {h.add(x);
    }
  Object pop()
    {if(isEmpty()) return(null);
     return(h.remove(h.size()-1));
    }
 }
```

```
import java.util.*;
class MyStack
 {LinkedList h;
  MyStack() {h = new LinkedList();}
  boolean isEmpty()
    {return(h.isEmpty());}
  void push(Object x)
    {h.add(x);
    }
  Object pop()
    {if(isEmpty()) return(null);
     return(h.removeLast());
    }
 }
```

Convert decimal integer number to binary number using a stack

```
public class Main
 {public static void decToBin(int k)
    {MyStack s = new MyStack();
     System.out.print(k + " in binary system is: ");
     while(k>0)
      {s.push(new Integer(k%2));
       k = k/2;
      }
     while(!s.isEmpty())
      System.out.print(s.pop());
     System.out.println();
    }
  public static void main(String [] args)
   {decToBin(11);
    System.out.println();
   }
 }
```

Validate expression using stack - 1

We consider arithmetic expressions that may contain various pairs of grouping symbols, such as

- Parentheses: “(” and “)”
- Braces: “{” and “}”
- Brackets: “[” and “]”

Each opening symbol must match its corresponding closing symbol. For example, a left bracket, “[,” must match a corresponding right bracket, “],” as in the following expression

$[(5+x)-(y+z)]$.

The following examples further illustrate this concept:

- Correct: ( )(( )){([( )])}
- Correct: ((( )(( )){([( )])}))
- Incorrect:)(( )){([( )])}
- Incorrect: ({[ ])}
- Incorrect: (

We leave the precise definition of a matching group of symbols to Exercise R-6.6.

Validate expression using stack - 2

An Algorithm for Matching Delimiters:
An important task when processing arithmetic expressions is to make sure their delimiting symbols match up correctly. We can use a stack to perform this task with a single left-to-right scan of the original string.
Each time we encounter an opening symbol, we push that symbol onto the
stack, and each time we encounter a closing symbol, we pop a symbol from the stack (assuming it is not empty) and check that these two symbols form a valid pair. If we reach the end of the expression and the stack is empty, then the original expression was properly matched. Otherwise, there must be an opening delimiter on the stack without a matching symbol. If the length of the original expression is *n*, the algorithm will make at most *n* calls to push and *n* calls to pop.

Matching Parentheses and HTML Tags

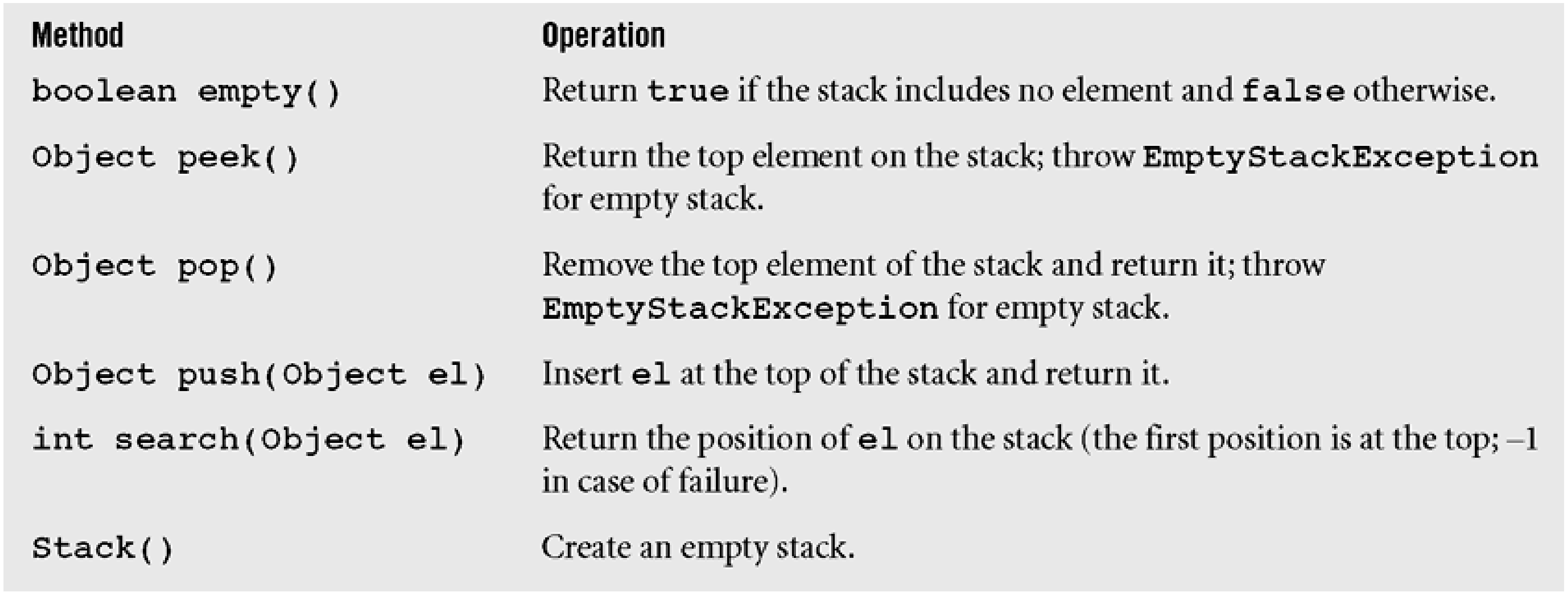
Another application of matching delimiters is in the validation of markup languages such as HTML or XML. HTML is the standard format for hyperlinked documents on the Internet and XML is an extensible markup language used for a variety of structured data sets.

In an HTML document, portions of text are delimited by *HTML tags*. A simple opening HTML tag has the form “<name>” and the corresponding closing tag has the form “</name>”. For example, the <body> tag has the matching </body> tag at the close of that document.

Ideally, an HTML document should have matching tags, although most browsers tolerate a certain number of mismatching tags. We can use stack to check whether an HTML document is valid or not.

Stack class in Java

A Stack class implemented in the java.util package is an extension of class Vector to which one constructor and five methods are added.



Method	Operation
`boolean empty()`	Return `true` if the stack includes no element and `false` otherwise.
`Object peek()`	Return the top element on the stack; throw `EmptyStackException` for empty stack.
`Object pop()`	Remove the top element of the stack and return it; throw `EmptyStackException` for empty stack.
`Object push(Object el)`	Insert `el` at the top of the stack and return it.
`int search(Object el)`	Return the position of `el` on the stack (the first position is at the top; –1 in case of failure).
`Stack()`	Create an empty stack.

Stack class in Java

Summary

- A stack is a linear data structure that can be accessed at only one of its ends for storing and retrieving data.
- A stack is called an LIFO structure: last in/first out.

Reading at home

Text book: Data Structures and Algorithms in Java.

- 6 Stacks, Queues, and Deques 225
- 6.1 Stacks - 226