Summary previous lecture - Stacks

- Stack is an **abstract data structure** of type LIFO (Last In First Out)
- Allowing access to the last item inserted. If removed, you get access to the next-to-last item inserted, and so on..... Complexity $O(1)$.  
- It is a simple useful tool in programming

List of basic operations using a Stack

- push(item);
- pop();
- peek()
- isEmpty()
- isFull()
- size()
public Stack(int maxSize) {
    this.maxSize = maxSize; // set array size
    array = new Object[maxSize]; // create array
    top = -1; // no item yet
}

public boolean isEmpty() {
    // instructions
}

public boolean isFull() {
    // instructions
}

public int size() {
    // instructions
}

public Object peek() {
    // instructions
}

public Object pop() {
    // instructions
}

public void push(Object item) {
    // instructions
}
Stacks - Simple Application Examples

- Generic stack tool

```java
class StackApp1
{
    public static void main(String[] args)
    {
        Stack mystack = new Stack(5);
        for(int i=1; i<=4; i++) mystack.push(2*i);
        System.out.print(mystack.pop());
        System.out.print(mystack.size());
        System.out.print(mystack.pop());
        System.out.print(mystack.peek());
    }
}
```

Guess?

- Reversing a word: Push D, Push O, Push G, Pop, Pop, Pop → GOD
- Delimiter matching: a[b(c[d]e)f] matching?
Stacks- Parsing Arithmetic Expressions - Overview

- Parsing or analyzing arithmetic expressions is an important application of stacks - Examples:
  - 2+3, 2*(3+4), ((2+4)*7), 3*(9-5)+(8-2)*2... How would you do it?
- It is difficult to implement an algorithm that can evaluate arbitrary infix expressions directly. We often proceed in two steps:
  - 1- Transform the expression from infix to postfix notation called Reverse Polish Notation or RPN
  - 2- Evaluate the RPN using a Stack
- Automation of Step 1 is rather complex (see Textbook p149-166), or the Shunting-Yard algorithm using a combination of stack & queue.
- We assume here that step 1 is already performed (by us humans) and we focus on step 2
- Some older powerful calculators adopted this RPN model!
Process the RPN using a stack (step 2)

- Evaluate 2+3
  - Traditional calculator (infix): 2+3=
  - Postfix expression is: 23+
  - RPN calculator 2 ↓ 3 ↓ +

- Evaluate 3*(9-5)+(8-2)*2
  - Traditional calculator 3*(9-5)+(8-2)*2= … will do the parsing for you
  - Postfix expression: 395- * 82 - 2 * +
  - RPN calculator 3 ↓ 9 ↓ 5 ↓ - X 8 ↓ 2 ↓ - 2 ↓ X ↓ +
Process the RPN using a stack (step 2)

- Evaluate 2+3

RPN calculator 2 ↓ 3 ↓ +

2 3 5

3←Pop
2←Pop
5=2+3
Push(5)

- Evaluate 3*(9-5)+(8-2)*2

3 ↓ 9 ↓ 5 ↓ - × 8 ↓ 2 ↓ - 2 ↓ × +

3 3 3 3 12 12 12 12 12 12 12 24
Comments on RPN transformations (step-1)

- From Infix to Postfix
  - A+B-C  \rightarrow  AB+C-
  - A*B/C  \rightarrow  AB*C/
  - A+B*C  \rightarrow  ABC*+
  - A*B+C  \rightarrow  AB*C+
  - A*(B+C)  \rightarrow  ABC+*
  - A*B+C*D  \rightarrow  AB*CD*+
  - (A+B)*(C-D)  \rightarrow  AB+CD-*
  - ((A+B))*(C-D)  \rightarrow  AB+CD-*
  - A+B*(C-D/(E+F))  \rightarrow  ABCDEF+/-*+

Remarks on the automated generation of the postfix:
- The postfix notation is a queue (FIFO)
- A stack is used for handling the delimiters
Stack class – additional methods

**public void** add(){
    if (!isEmpty()){
        Object i1=pop();
        Object i2=pop();
        push(i1+i2);
    }
}

**public void** subtract(){
    if (!isEmpty()){
        Object i1=pop();
        Object i2=pop();
        push(i2-i1);
    }
}

**public void** multiply(){
    if (!isEmpty()){
        Object i1=pop();
        Object i2=pop();
        push(i2*i1); /* if allowed */
    }
}

class StackApp4{
    public static void main(String[] args)
    {
        Stack mystack = new Stack(5);
        for(int i=5; i>=1; i--) mystack.push(i*10);
        mystack.add();
        mystack.subtract();
        mystack.multiply();
        mystack.add();
        System.out.print(mystack.peek());
    }
}

**Remark:** In Java, you may want to use an extended class to the class Stack

Guess?

Result is 50
Using the class Stack.java (with interface methods: pop, push, isEmpty, add, multiply, subtract, divide), we can implement a simple RPN calculator.

a- Complete the following code that solves: “(4+2)*8-3” in postfix notation:

```java
class StackApp {
    public static void main(String[] args) {
        Stack mystack = new Stack(5);
        // to complete... solve: (4+2)*8-3
    }
}
```

b- Let assume you use the instruction System.out.println(mystack.peek()); after each new instructions in you code above. Write the successive outputs below:
class StackCalcApp
{
    public static void main(String[] args)
    {
        Stack mystack = new Stack(100); //100 max capacity
        EasyIn easy = new EasyIn();
        int n=easy.readInt();
        for (int i=n;i>0;i--) mystack.push(i);
        boolean opadd=true;
        while (mystack.size()>1)
        {
            if (opadd) {
                mystack.push((int) mystack.pop() + (int) mystack.pop());
                opadd=false;
            }
            else {
                mystack.push((int) mystack.pop() * (int) mystack.pop());
                opadd=true;
            }
        }
        System.out.println("Result is "+mystack.peek());
    }
}