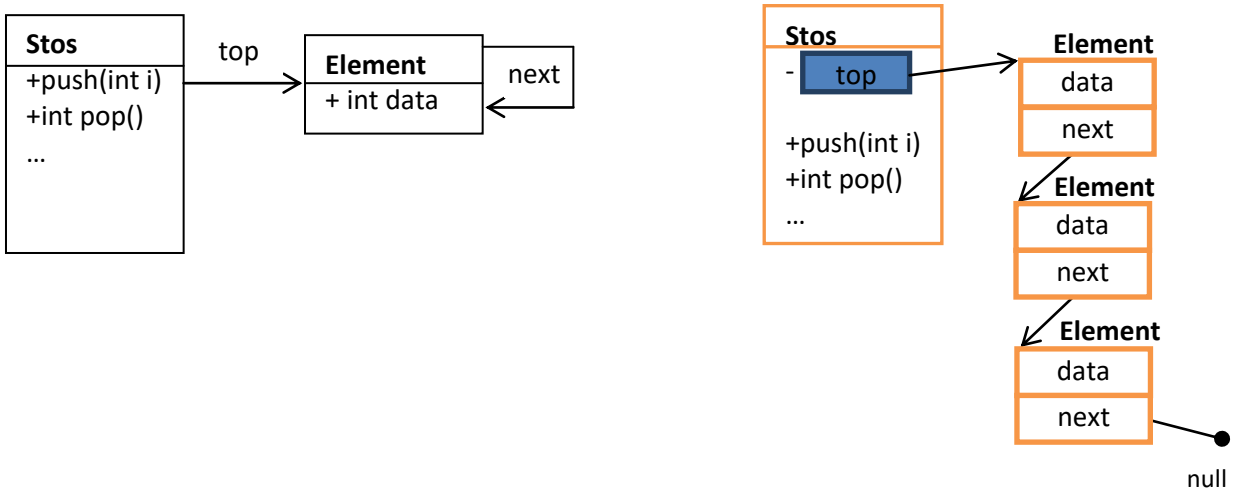
Algorithms and Complexity

Laboratory 3: Stacks. Simple algorithms that use stacks.

1. **Stack concept.**
   A stack is one of the types of data structures. The subsequent elements (data) are placed at the top of the stack. Only the element at the top of the stack can be removed. A good example of how the stack works is a stack of books on your desk. We can add another book by placing it on the top, after which the newly added book becomes the top of the stack. If we want to take a book from such a stack, we can only take a book that lies on top. If we want to reach for another book, we have to take the books off the top one by one.

2. **Stack operations.**
   push - adding an item to the stack
   pop - download the item from the stack (and return its value)
   isEmpty - check if the stack is empty
   Often, stack implementations provide additional operations, such as
   peek - check the value of the element at the top of the stack without removing it

3. **Class diagram and visualisation of example stack elements (connected by references)**



4. **Interface IStack and the class Element – building blocks for the stack implementation.**

```
public interface IStos {
    public void push(int i);  // adds an element
    public int pop();         // removes the top element (if exists)
    public int peek();        // returns the top value, but doesn't change anything
    public boolean isEmpty(); // checks if the stack is empty
    public void print();      // prints the stack contents on the console
    public void clear();      // removes all the elements of the stack
}

public class Element{           // the class representing a stack element
    public int data;            // the data attribute (public, for simplicity)
    public Element next;        // reference to the next stack element
}
```

5. **Tasks.**

1. Create the class Stack that implements the IStos interface as a linked structure using objects of the class Element. Write a program testing the implemented methods (eg inserting several elements on the stack, writing out the contents of the stack, removing the element from the top, writing out the contents of the stack, etc.).

Tips:

- in the Stack class, create a private field (attribute) of type Element (eg named top);
- check if the stack is empty before you remove the item from the stack;
- writing a stack (print) can not change the stack content

[4p]

Remarks: The digram above shows how to use the Stack and Element instances. Note that a Stack object stores only one reference to an Element object (the top of the stack), not all items on the stack. In particular, when the stack is empty, the condition *top == null* holds. Stack elements (like all Java objects) are allocated in the memory after calling the constructor (eg Element x = new Element ()). Each instance of the Element type stores an integer (the *data* field), and in the next field, a reference to the next element (the one that is just below on our stack), or a null value if the element is at the bottom of the stack (if it is the last one).

1. Write a program exploiting a stack that checks if the parentheses are correctly nested in the given expression. For example, the expression *"(() ((()))"* is correct, while *") ("* and *"(()"* are incorrect. **[2p]**
Tip: Place some item on the stack when you encounter '(', and remove an item when you encounter ')'

2. Write a program exploiting a stack that checks if the given word is a palindrome. **[2p]**
Tip: Use the stack to reverse the word.

3. Write a program that reads arithmetic expressions in postfix notation (Reverse Polish Notation) and calculates their value using the stack. **[2p]**
Tip: When you read a number, place it on the stack. When you read an operator symbol, remove two numbers from the stack, compute the result, and place the result on the stack.
Example expression in ONP: 5 2 + 7 * corresponds to (5 + 2) * 7. Result 49.

Homework:

1. Write a program that reads an arithmetic expression from file (concern at least + - * / () operators) and converts it to postfix notation using stack.

2. Implement the Stack class as a generic class so that you can store any type of object on the stack.

3. Solve tasks 2-4 using the java.util.Stack class or the java.util.Deque implementation (e.g. LinkedList).

Next week:

Lists. Queues.