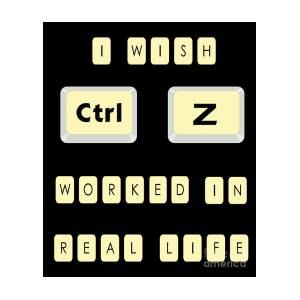
**Lesson 11 – Stacks I**

**Reading: Chapter 7, Sections 1-3 of the text.**

**Learning Objectives:**

* Describe the abstract-level functionality of an ADT Stack.
* List the Stack ADT operations.
* Define a stack interface in Java.
* Implement an array-based stack.

**The Abstract Data Type Stack:**

* Describe the concept of a stack ADT:
  + The term “stack” is intended to conjure up visions of things encountered in daily life, such as a stack of dishes in the school cafeteria, a stack of books on your desk, or a stack of assignments waiting for you to work on them.
  + To a computer scientist, a stack means something very particular. A stack has the property that the last item placed on the stack will be the first item removed. This property is commonly referred to as **last-in, first- out**, or simply **LIFO**.
* The ADT Stack Operations:
  + Create an empty stack.
  + Determine whether a stack is empty.
  + Add a new item to the stack (**push**).
  + Remove from the stack the item that was added most recently (**pop**).
  + Remove all the items from the stack.
  + Retrieve from the stack the item that was added most recently (**peek**).
* Pseudocode Specification for the Stack ADT:

// StackItemType is the type of the items stored in the stack.

+createStack()

// Creates an empty stack

+isEmpty() : Boolean

// Determines whether a stack is empty

+push(in newItem : StackItemType) throws StackException

// Adds newItem to the top of the stack. Throws

// StackException if the insertion is not successful.

+pop() : StackItemType throws StackException

// Retrieves and then removes the top of the stack (the

// item that was added most recently). Throws

// StackException if the deletion is not successful.

+popAll()

// Removes all items from the stack.

+peek() : StackItemType throws StackException

// Retrieves the top of the stack. That is, peek

// retrieves the item that was added most recently.

// Retrieval does not change the stack. Throws

// StackException if the retrieval is not successful.

Defining an interface for the ADT Stack

* First, define StackException:

public class StackException extends RuntimeException {

    public StackException(String s) {

        super(s);

    } // end constructor

}

* Now, write the interface:

public interface Stackable<E> {

    public boolean isEmpty();

    // Determines whether the stack is empty.

    public void popAll();

    // Removes all the items from the stack.

    public E peek() throws StackException;

    // Retrieves the top of a stack.

    public void push(E newItem) throws StackException;

    // Adds an item to the top of a stack.

    public E pop() throws StackException;

    // Removes the top of a stack.

}

**Talk through the operations from a conceptual point of view.**

**An array-based implementation of the ADT Stack:**

* The StackArrayBased class:

public class StackArrayBased<E> implements Stackable<E> {

    private final int MAX\_STACK = 50;

    private Object[] items; // cannot use generic arrays in Java

    private int top; // index of the top

    public StackArrayBased() {

        this.items = new Object[MAX\_STACK];

        this.top = -1;

    }

    @Override

    public boolean isEmpty() {

        return top < 0;

    }

    // added due to restrictive nature of implementation

    public boolean isFull() {

        return top == (MAX\_STACK - 1);

    }

    @Override

    public void popAll() {

        this.items = new Object[MAX\_STACK]; // garbage collect references

        top = -1;

    }

    @Override

    public void push(E newItem) throws StackException {

        if (isFull())

            throw new StackException("Stack maximum is at maximum capacity.");

        else {

            top++;

            this.items[top] = newItem;

        }

    }

    @Override

    public E pop() throws StackException {

        if (isEmpty())

            throw new StackException("Stack is empty.");

        else {

            E item = (E) this.items[top];

            top--;

            return item;

        }

    }

    @Override

    public E peek() throws StackException {

        if (isEmpty())

            throw new StackException("Stack is empty.");

        else {

            return (E) this.items[top];

        }

    }

    @Override

    public String toString() {

        String s = "[";

        for (int i = top; i >= 0; i--) {

            s = s + this.items[i].toString();

            if (i != 0)

                s = s + ", ";

        }

        s = s + "]";

        return s;

    }

}

* A main method that uses it:

public static void main(String[] args) {

        StackArrayBased<Integer> stack = new StackArrayBased<Integer>();

        // add numbers 1 through 15

        for (int i = 1; i <= 15; i++) {

            Integer num = i;

            stack.push(num);

        }

        System.out.println("Original Stack: ");

        System.out.println(stack);

        System.out.println();

        // remove top 5

        for (int i = 1; i<= 5; i++)

            System.out.println( stack.pop());

        System.out.println();

        System.out.println("Stack after popping: ");

        System.out.println(stack);

        stack.popAll();

        System.out.println();

        System.out.println("Stack after pop all:");

        System.out.println(stack);

    }

}