Data Structure

Recitation V
Topic

- Array based Implementation Stack
- Linked based Implementation Stack
Array based implementations

- Array based implementations: Bounded Stack
- Size of array is static: Stacks with an upper bound on the number of elements
  - push() can raise an overflow exception.
  - A predicate is also supplied so that clients can detect full stacks before causing an exception(isFull()).
Interface

```java
package ch03.Stacks;

public interface StackInterface<T> {
    void pop() throws StackUnderflowException;
    T top() throws StackUnderflowException;
    boolean isEmpty();
}

package ch03.Stacks;

public interface BoundedStackInterface<T> extends StackInterface<T> {
    void push(T element)
        throws StackOverflowException;
    boolean isFull();
}
```
package ch03.stacks;
public class ArrayStack<T> {
    implements BoundedStackInterface<T> {
    private final int DEFSIZE = 100;
    private T[] stack;
    private int topIndex = -1;
    public ArrayStack(int size) {
        stack = (T[]) new Object[size];
    }
    public ArrayStack() { this(DEFSIZE); }
}
ArrayStack Class

```java
public boolean isEmpty() { return topIndex == -1; }
public boolean isFull() { return topIndex == stack.length - 1; }
public void push(T element) {
    if (isFull())
        throw new StackOverflowException("helpful msg");
    else
        stack[++topIndex] = element;
}

public void pop() {
    if (isEmpty())
        throw new StackUnderflowException("helpful msg");
    else
        stack[topIndex--] = null;
}

public T top() {
    if (isEmpty())
        throw new StackUnderflowException("helpful msg");
    else
        return stack[topIndex];
}
```
Homework 28A

- A. Remove the final attribute from the DEF_CAP instance

- Final key word: In the Java programming language, the final keyword is used in several different contexts to define an entity which cannot later be changed.

- Final variable: A final variable can only be initialized once, either via an initializer or an assignment statement.
  - If the variable is a reference, this means that the variable cannot be re-bound to reference another object. But the object that it references is still mutable, if it was originally mutable.
Homework 28C

- C Change the value assigned to DEF_CAP to -10
- java.lang.NegativeArraySizeException
Homework 28D

D In the first constructor change the statement to
stack = (T[]) new Object[100]
Homework 28F

- F Reverse the order of the two statements in the if clause of the push method
- Stack[++topIndex] = element & Stack[topIndex++] = element
Homework 30C

- C. Set bottom equal to the bottom element in myStack, leaving myStack unchanged.
  - Declare a additional stack B
  - Keep doing pop until if myStack is empty
  - When doing pop from myStack, push those elements popped from myStack to Stack B (Order is reversed)
  - After finding the last element in myStack, keep doing pop from stack B until its empty and push those elements back to myStack (Order is reversed back)
Linked Based Implementation

- Linked based implementations: Unbounded Stack
- Size of Linked list is dynamic: the linked structure cannot be full or overflow
- LLNode Class and LinkedStack Class
Linked Based Implementation

```java
package support;
public class LLNode<T> {
    private LLNode<T> link = null;
    private T info;
    public LLNode(T info) { this.info = info; }
    public T getInfo() { return info; }
    public LLNode<T> getLink() { return link; }
    public void setInfo(T info) { this.info = info; }
    public void setLink(LLNode<T> link) { this.link = link; }
}
```

Self referential class, A link is the same thing as a reference
Linked Base Implementation

- Only hold one reference to the linked list of objects that represents the stack.

- In this case, we maintain a reference to the node representing the most recent element pushed onto the stack (top).

- An particular node hold a null reference in its link attribute to indicate the bottom of the stack.
package ch03.stacks;
import support.LLNode;
public class LinkedStack<T> implements UnboundedStackInterface<T> {
    private LLNode<T> top = null;
    public LinkedStack(){
    }
}
Push

- Create a new node and linking it to the current chain of object
- When preforming the push operation we must allocate space for each new node dynamically
- Order is critical!

```java
public void push (T info) {
    LLNode<T> node = new LLNode<T>(info);
    node.setLink(top);
    top = node;
}
```
Pop

- Deleting the first node of a linked list
- Resetting stack’s top variable to reference the node that represents the next element
- After resetting, we are no longer have access to the pervious top element(labeled as garbage)
- Pop from empty stack

```java
public void pop() throws StackUnderflowException {
    if (isEmpty())
        throw new StackUnderflowException("helpful msg");
    else
        top = top.getLink();
}
```
Other operations

```java
public T top() {
    if (isEmpty())
        throw new StackUnderflowException("helpful msg");
    else
        return top.getInfo();
}

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

- Differences are:
  - Linked-based stacks are unbounded
  - A linked implementation requires space for the links.
  - The array-based implementation is a little simpler and pushes should be faster since they don't involve new.
An interesting question

- How to find the minimal element in a linked stack? Stack remain the same after this operation.

- Probably pop every element in the stack do compare, keep track of the minimal element, push them to another stack. After the original stack is empty we will get the minimal element, then pop the element from that additional stack and push them back to keep the original one be the same.
An interesting question

- How to find the minimal element in a stack in constant time? Write a new method in LinkedStack, return the minimal element in that stack.

- Hint: You need to modified the LLNode class and push method in LinkedStack class a little bit.