

CMPSCI 187 (Fall 2018) Lab 03: Stacks and Parameter Passing

The goal of this lab is to: understand parameter passing, stacks and exceptions.

- Go to **File -> Make a Copy** to make an editable copy of this Google Doc for your account.
- When you are done, go to **File -> Download As -> PDF Document**. You must select PDF, as Gradescope does not accept other formats.
- **Submit your PDF in [Gradescope](#) under Lab**. Note: submit to Lab, not to Project!
- There is NO starter code for this lab -- you should work on the exercises without using any Java IDE or compiler. This is the same during exams.

Section A: Parameter passing

Defined below is a class **Account**, which stores balance in a person's bank account. Carefully read the three **swap** functions defined in this class and answer the questions that follow.

```
public class Account{
    public int bal; // this store the balance amount in the account
    public Account(int initialBalance)
    {
        bal = initialBalance;
    }
    public static void swap_1(int num1, int num2)
    {
        int temp = num1;
        num1 = num2;
        num2 = temp;
    }
    public static void swap_2(Account acc1, Account acc2)
    {
        int temp = acc1.bal;
        acc1.bal = acc2.bal;
        acc2.bal = temp;
    }
    public static void swap_3(Account acc1, Account acc2)
    {
        Account temp = acc1;
        acc1 = acc2;
        acc2 = temp;
    }
}
```

Now, different students tried to use different **swap_***() methods to swap the balances in their friends' bank accounts. Predict the output of each of the following code snippets [1 pts each]

(a)

```
Account tom = new Account(100);
Account jim = new Account(2000);
System.out.println("1.Tom has $" + tom.bal + "Jim has $" + jim.bal);
swap_1(tom.bal, jim.bal);
System.out.println("2.Tom has $" + tom.bal + "Jim has $" + jim.bal);
```

```
1.Tom has $100Jim has $2000
2.Tom has $100Jim has $2000
```

(b)

```
Account tom = new Account(100);
Account jim = new Account(2000);
System.out.println("1.Tom has $" + tom.bal + "Jim has $" + jim.bal);
swap_2(tom, jim);
System.out.println("2.Tom has $" + tom.bal + "Jim has $" + jim.bal);
```

```
1.Tom has $100Jim has $2000
2.Tom has $2000Jim has $100
```

(c)

```
Account tom = new Account(100);
Account jim = new Account(2000);
System.out.println("1.Tom has $" + tom.bal + "Jim has $" + jim.bal);
swap_3(tom, jim);
System.out.println("2.Tom has $" + tom.bal + "Jim has $" + jim.bal);
```

```
1.Tom has $100Jim has $2000
2.Tom has $100Jim has $2000
```

Section B: Fun with stacks [2 pts each question]

Consider a generic **Stack** class implemented using linked list with the following methods:

- **public boolean isEmpty()**, which returns true *if and only if* the stack is empty;
- **public T pop() throws StackUnderflowException**, which removes and returns the top element of the stack (if the stack is empty, it throws **StackUnderflowException**);
- **public T peek() throws StackUnderflowException**, which returns the top element of the stack (but does not remove it; it throws exception if stack is empty);
- **public void push(T element)**, which pushes an element onto the stack

Predict the output of each of the following code snippets

```
(a) Stack<Integer> s = new Stack<Integer>();
    try{
        s.push(5);
        s.push(10);
        System.out.println(s.peek());
        System.out.println(s.pop());
        s.push(20);
        System.out.println(s.pop());
    }
    catch(Exception e) { System.out.print("An exception was thrown"); }
```

```
10
10
20
```

```
(b) Stack<Integer> s = new Stack<Integer>();
    try{
        s.push(5);
        s.push(50);
        System.out.println(s.pop());
        System.out.println(s.pop());
        System.out.println(s.peek());
        s.push(30);
    }
    catch(Exception e) { System.out.println("An exception was thrown"); }
```

```
50
5
An exception was thrown
```

```
(c) Stack<Integer> s = new Stack<Integer>();
    try{
        s.push(5);
        System.out.println(s.pop());
        System.out.println(s.isEmpty());
        s.push(10);
        s.push(43);
        s.push(s.pop());
        System.out.println(s.peek());
    } catch(Exception e) {
        System.out.println("An exception was thrown");
    }
```

```
5
true
43
10
```

```
(d) Stack<Integer> s = new Stack<Integer>();
    try {
        s.push(5);
        s.push(10);
        while(!s.isEmpty())
        {
            System.out.println(s.peek());
        }
    }
    catch(Exception e) { System.out.println("An exception was thrown"); }
```

```
10
10
...
(infinite loop)
```

Section C: Implementing Stack using Linked List

In this section, you will implement a generic Stack class implemented using linked list. Assume the linked list node class is already defined as below:

```
public class LLNode<T> {  
    public LLNode<T> link;  
    public T info;  
    public LLNode(T in) { info = in; link = null; }  
}
```

Note that both class variables are public so any outside class can access them directly. Also assume that class StackUnderflowException has been defined that inherits Java's Exception class. Your task is to implement four methods in the generic class LinkedListStack<T>.

```
public class LinkedListStack<T> {  
    private LLNode<T> head; // head of linked list, also stack top pointer  
    public LinkedListStack() { head = null; } // constructor  
  
    public boolean isEmpty() { // [1 pts]  
        // TODO: return true if stack is empty, false otherwise  
        // NO MORE THAN 1 LINE OF CODE!  
        return head == null;  
    }  
  
    public void push(T element) { // [2 pts]  
        // TODO: push an element to the stack  
        // NO MORE THAN 3 LINES of CODE!  
        LLNode<T> node = new LLNode<T>(element);  
        node.link = head;  
        head = node;  
    }  
  
    public T peek() throws StackUnderflowException { // [2 pts]  
        // TODO: return the top element of the stack (but do NOT  
        // remove it). NO MORE THAN 4 LINES of CODE!  
        if (isEmpty())  
            throw new StackUnderflowException();  
        return head.info;  
    }  
}
```

```
public T pop() throws StackUnderflowException { // [3 pts]
    // TODO: remove and return the top element of the stack
    // It throws StackUnderflowException if stack is empty
    // NO MORE THAN 6 LINES of CODE!
    T result = peek(); //peek() throws StackUnderflowException
    head = head.link;
    return result;
}
}
```