COSC 222: Data Structures
Lab 2 – Stacks & Queues (20 marks)

Question 1: Lakes of Canada [15 marks]
Write a program that implements Stacks to store information about the lakes of Canada. You need to complete the following java files, which can be tested by running the **StackTest.java** file.

Part A [1 mark]:
First, create a **Lake.java** file. This class should have:

- Instance variables `name` (String), `province` (String), `area` (double) and `altitude` (double) of a lake
- A constructor that accepts those four values, in the order listed above
- Get/set methods for all four variables
- A `toString()` method to return a string summarizing all the information stored about the lake (e.g., *`name, province, area, altitude`*)

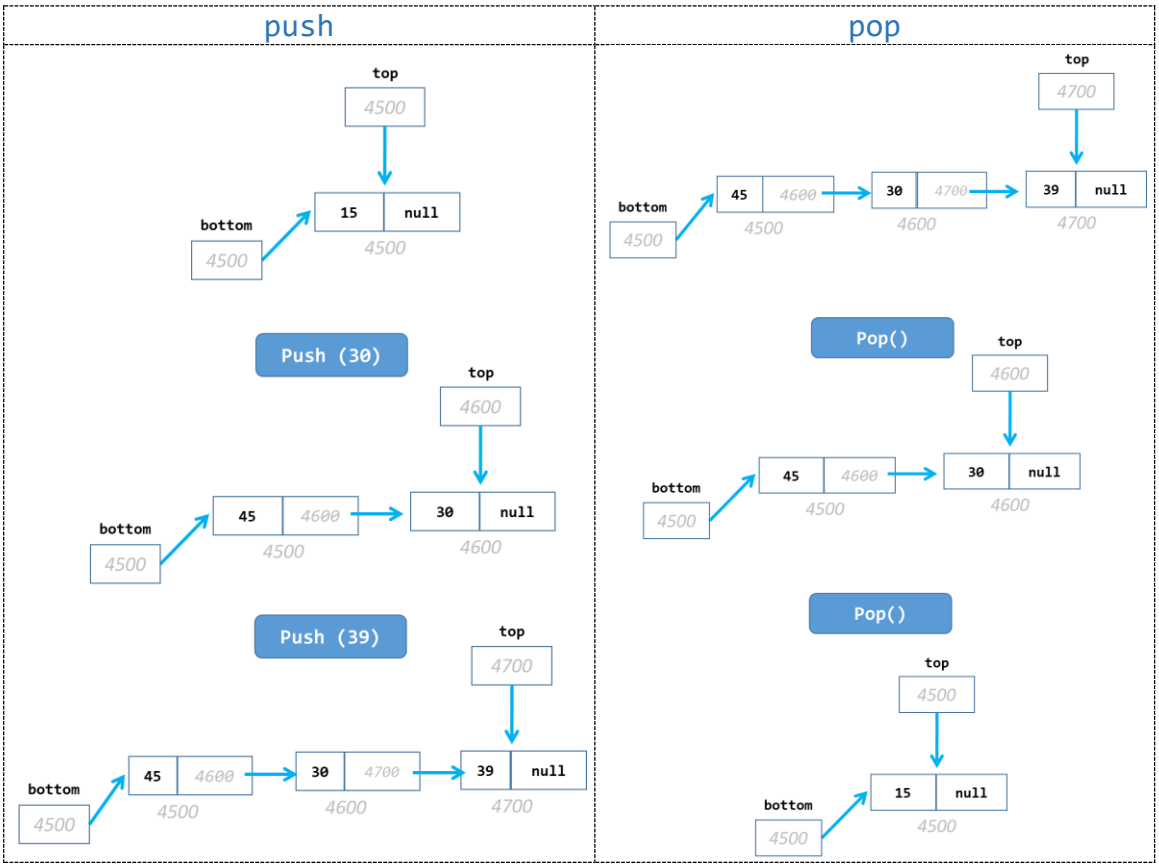
Part B [5 marks]: Array Implementation of Stack
Now complete the file **ArrayStack.java**. This class will use an array to represent a stack. Implement the interface provided: **StackADT.java** (Stack Abstract Data Type).

- The default capacity of the stack should be set to 3. If the total number of items is greater than the current stack size, a larger stack (2 times larger than the current one) should be created, and the contents of the old stack should be moved to the new stack. Make sure that the **order is not changed** when moving the old stack's contents over (1 mark).
- Note that your **ArrayStack** class should contain no reference to the `Lake` object. Rather it should refer only to an abstract object `T`, in the same way that the **StackADT** interface does.
- You have to complete the following methods:
    - `push()`: Adds the given elements to the top of the stack (1 mark)
    - `pop()`: Removes and returns the top element in the stack (1 mark)
    - `peek()`: Returns the top element in the stack *without removing it* (0.5 mark)
    - `size()`: Returns the number of elements in the stack (0.5 mark)
    - `isEmpty()`: Returns true if this stack is empty; false otherwise (0.5 mark)
- If the `pop() or peek()` method is called on an empty stack, the program should print a warning message (e.g. "Stack is empty") – (0.5 mark)

Part C [9 marks]: Linked List Implementation of Stack
Now you will complete a second class (**LinkedListStack**) which will function similarly to the **ArrayStack** class created in Part B, except it will use a linked list instead of an array to implement a stack. Implement the **StackADT.java** interface again.

- You should first review the provided **Node.java** class. Notice that the data type stored in the node is an abstract object `T`.

- Now create a **LinkedListStack.java** class. You should keep two special nodes to hold the information about the `top` and `bottom` nodes of the stack (see the image below, note that instead of `Lake` type data, we used numbers for our convenience. Also, assume that the grey-shaded (i.e., *4500, 4600, 4700*) numbers are addresses. When showing output (i.e. `toString()` method), we want to show the output starting from the bottom to the top.



- The methods of this class:
  - `push()`: Adds the given elements to the top of the stack (2 marks)
  - `pop()`: Removes and returns the top element in the stack (3 marks)
  - `peek()`: Returns the top element in the stack *without removing it* (1 mark)
  - `size()`: Returns the number of elements in the stack (1 mark)
  - `isEmpty()`: Returns true if this stack is empty; false otherwise (1 mark)
  - `toString()`: Show the output starting from the `bottom` to the `top` (1 mark)

  Use similar techniques to the **ArrayStack.java** class of Part B. However, a linked list should be used to implement the operations instead of an array.

If you complete all of the above classes correctly, you should see the following output when running the **StackTest.java** file.

Sample Output:

```
*** Lakes with Array Stack ***
Stack is empty
Stack size: 1
The stack contents:
Okanagan Lake, British Columbia, 351.0 sq km, 342.21 meters

Stack size: 1
The stack contents:
Lake Erie, Ontario, 103700.9 sq km, 173.7 meters

*** Lakes with Linked List Stack ***
Stack is empty
Stack size: 1
The stack contents:
Okanagan Lake, British Columbia, 351.0 sq km, 342.21 meters

Stack size: 1
The stack contents:
Lake Erie, Ontario, 103700.9 sq km, 173.7 meters
```

Question 2 (Simple Stack): Card Sorting [5 marks]

Write a program to create a deck of cards, and after it is shuffled, sort the cards by suit. Start with the file **CardSort.java**. For this question, you will use the default methods provided with **java.util.Stack**.

Part A [2 marks]:

Complete the method `makeDeck()` so that it returns a `Stack<String>` containing a full deck of 52 unique cards, one for each combination of suit and rank. Use the final String arrays provided (`RANKS` and `SUITS`) to save yourself some typing. This is similar to the `CardShuffle.makeDeck()` method of Lab 1 but remember that you should call different methods (e.g., `.push()`) for Stacks than Lists.

Part B [3 marks]:

Complete the method `sortSuits()`. This method takes one argument: a deck of cards (`Stack<String> deck`). It should return an ArrayList of Stacks, each stack representing all the cards of one suit (i.e., if you have four suits - Spades, Hearts, Clubs, Diamonds, it should return ArrayLists of Spades, Hearts, Clubs, and Diamonds). The returned ArrayList should be in the same order as the `SUITS` array. Use a loop to look at each card, determine its suit (methods such as `peek()/pop() and String.contains()` may be helpful), and move the card from the deck to the stack associated with its suit.

If you complete these methods properly, you should see a randomly shuffled output like this.

Sample Output:

```
New deck: 52 cards
[Ace of Spades, Ace of Hearts, Ace of Clubs, Ace of Diamonds, 2 of Spades, 2 of
Hearts, 2 of Clubs, 2 of Diamonds, 3 of Spades, 3 of Hearts, 3 of Clubs, 3 of
Diamonds, 4 of Spades, 4 of Hearts, 4 of Clubs, 4 of Diamonds, 5 of Spades, 5 of
Hearts, 5 of Clubs, 5 of Diamonds, 6 of Spades, 6 of Hearts, 6 of Clubs, 6 of
Diamonds, 7 of Spades, 7 of Hearts, 7 of Clubs, 7 of Diamonds, 8 of Spades, 8 of
Hearts, 8 of Clubs, 8 of Diamonds, 9 of Spades, 9 of Hearts, 9 of Clubs, 9 of
Diamonds, 10 of Spades, 10 of Hearts, 10 of Clubs, 10 of Diamonds, Jack of Spades,
Jack of Hearts, Jack of Clubs, Jack of Diamonds, Queen of Spades, Queen of Hearts,
Queen of Clubs, Queen of Diamonds, King of Spades, King of Hearts, King of Clubs,
King of Diamonds]

Shuffled deck: 52 cards
[8 of Hearts, 8 of Diamonds, 5 of Clubs, 5 of Spades, 9 of Hearts, 2 of Hearts, 7
of Hearts, 4 of Clubs, 9 of Clubs, 6 of Hearts, Queen of Hearts, Queen of Clubs, 2
of Spades, King of Spades, 4 of Hearts, Queen of Spades, 10 of Spades, Ace of
Diamonds, 5 of Hearts, Ace of Spades, 4 of Diamonds, 6 of Spades, 3 of Spades, 5 of
Diamonds, 10 of Diamonds, 4 of Spades, King of Hearts, 8 of Spades, Jack of Hearts,
6 of Diamonds, Jack of Diamonds, Ace of Hearts, 10 of Hearts, 7 of Spades, 7 of
Diamonds, 3 of Hearts, Jack of Spades, Queen of Diamonds, 8 of Clubs, King of
Diamonds, 2 of Clubs, Ace of Clubs, 9 of Spades, 3 of Diamonds, King of Clubs, 10
of Clubs, 7 of Clubs, 9 of Diamonds, Jack of Clubs, 2 of Diamonds, 6 of Clubs, 3 of
Clubs]

Spades: 13 cards
[9 of Spades, Jack of Spades, 7 of Spades, 8 of Spades, 4 of Spades, 3 of Spades, 6
of Spades, Ace of Spades, 10 of Spades, Queen of Spades, King of Spades, 2 of
Spades, 5 of Spades]

Hearts: 13 cards
[3 of Hearts, 10 of Hearts, Ace of Hearts, Jack of Hearts, King of Hearts, 5 of
Hearts, 4 of Hearts, Queen of Hearts, 6 of Hearts, 7 of Hearts, 2 of Hearts, 9 of
Hearts, 8 of Hearts]

Clubs: 13 cards
[3 of Clubs, 6 of Clubs, Jack of Clubs, 7 of Clubs, 10 of Clubs, King of Clubs, Ace
of Clubs, 2 of Clubs, 8 of Clubs, Queen of Clubs, 9 of Clubs, 4 of Clubs, 5 of
Clubs]

Diamonds: 13 cards
[2 of Diamonds, 9 of Diamonds, 3 of Diamonds, King of Diamonds, Queen of Diamonds,
7 of Diamonds, Jack of Diamonds, 6 of Diamonds, 10 of Diamonds, 5 of Diamonds, 4 of
Diamonds, Ace of Diamonds, 8 of Diamonds]

Deck is empty? true (0 cards)
```

Submission Instructions:

- Create a folder called "Lab2_<*student_ID*>" where <*student_ID*> is your student number/ID (e.g. **Lab2_12345678**). Only include the mentioned files in the folder. **DO NOT** include any other files (e.g., .class, .java~ or any other files)
    - For Question 1, include your **Lake.java**, **ArrayStack.java**, **Node.java**, and **LinkedListStack.java** files.
    - For Question 2, include your **CardSort.java** file.
- Make a **zip file** of the folder and upload it to Canvas.
- To be eligible to earn full marks, your Java programs **must compile and run** upon download, without requiring any modifications.
- These assignments are your chance to learn the material for the exams. **Code your assignments independently.**