**1. Problem Discussion :**

As mentioned in the question, you are required to complete the code of our LLToStackAdapter class.As data members, you've a linked list available in the class. Here is the list of functions that you are supposed to complete push -> Should accept new data in LIFO manner. pop -> Should remove and return data in LIFO manner. If not available, print "Stack underflow" and return -1. top -> Should return data in LIFO manner. If not available, print "Stack underflow" and return -1. size -> Should return the number of elements available in the Stack. Input and Output is managed for you. Note -> The intention is to use linked list functions to achieve the purpose of a stack. All the functions should work in constant time. It is recommended that you go and check out our question video of this problem to understand WHAT needs to be done in a better way.

**2. Approach :**

We will look at each function of our Stack Adapter individually. As we know, it is also given in the question that Stack adds and removes elements in LIFO manner, i.e. Last In First Out. Which means, the end at which element will be added lastly will be the first to get accessed. Element is removed and added from one end only. To make a linked list work like a stack, we need to make use of functions, such that addition and removal occurs at one end only. We have two options to achieve this: we can perform these operations like addition and removal either at the beginning or at the end. If we use the linkedlist which is in java then it does not matter which way we choose to do it because addition and removal at end as well as beginning are O(1) i.e. take constant time to perform their action. But if you use the linked list which we implemented then we chose to operate the starting of the linked list.

Why?

We do so because the functions; addFirst, getFirst and removeFirst are of O(1) i.e. take constant time to perform their action. Whereas the function removeLast which we implemented takes O(n) time, therefore we discard the idea of operating at the end of the linked list. Let's look at what needs to be done in each function of Stack using the functions addFirst, removeFirst and getFirst.

Push

Time Complexity: O(1) Space Complexity: O(1)

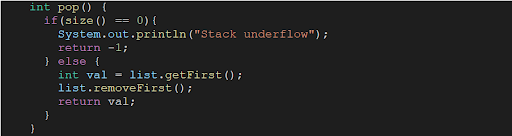
Whenever an element is pushed inside the stack it means an element is getting added in the list. So we simply call the addFirst function of list and further pass the val as an argument which will be passed in push .

****

Pop

Time Complexity: O(1) Space Complexity: O(1)

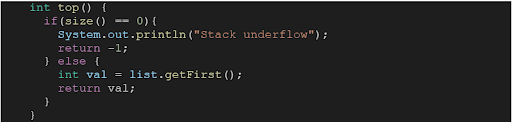
Whenever an element is pupped it means an element is removed in the list. So we simply call the removeFirst function inside this function.

****

Top

Time Complexity: O(1) Space Complexity: O(1)

Whenever the function top is called it means that we need to return the value of top most (which is the first) element of stack. For this we will call getFirst() function, which will return us the value of top of stack.

****

Size

Time Complexity: O(1) Space Complexity: O(1)

Whenever the function size is called it means that we need to return the size of the stack. For this, it is quite obvious that we call the size() function of the list, which will return us the size of the stack.

****

ConsoleJava

import java.io.\*;

import java.util.\*;

public class Main {

public static class LLToStackAdapter {

LinkedList< Integer> list;

public LLToStackAdapter() {

list = new LinkedList< >();

}

int size() {

return list.size();

}

void push(int val) {

list.addFirst(val);

}

int pop() {

if (size() == 0) {

System.out.println("Stack underflow");

return -1;

} else {

int val = list.getFirst();

list.removeFirst();

return val;

}

}

int top() {

if (size() == 0) {

System.out.println("Stack underflow");

return -1;

} else {

int val = list.getFirst();

return val;

}

}

}

public static void main(String[] args) throws Exception {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

LLToStackAdapter st = new LLToStackAdapter();

String str = br.readLine();

while (str.equals("quit") == false) {

if (str.startsWith("push")) {

int val = Integer.parseInt(str.split(" ")[1]);

st.push(val);

} else if (str.startsWith("pop")) {

int val = st.pop();

if (val != -1) {

System.out.println(val);

}

} else if (str.startsWith("top")) {

int val = st.top();

if (val != -1) {

System.out.println(val);

}

} else if (str.startsWith("size")) {

System.out.println(st.size());

}

str = br.readLine();

}

}

}

That was easy. Wasn't it? Our desire to make you learn will remain unsatisfactory if you still have doubts. We strongly recommend you to watch our video lecture on Get Value for clearing any type of doubts. Suggestions and feedback are always welcomed. You can contact us via our website. All the best for a bright future! Never dodge the bullet. Happy Coding!