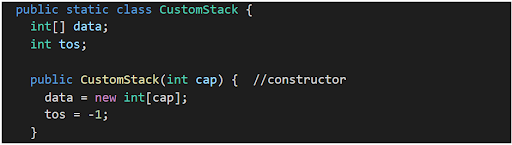
**1. Problem Discussion :**

In the question it is mentioned that you are required to complete the code of our CustomStack class. The class should mimic the behaviour of the java.util.Stack class and implement LIFO semantics. A list of functions that you are supposed to complete is given as below:

1• push -> Should accept new data if there is space available in the underlying array or print "Stack overflow" otherwise. 2• pop -> Should remove and return last data if available or print "Stack underflow" otherwise and return -1. 3• top -> Should return last data if available or print "Stack underflow" otherwise and return -1. 4• size -> Should return the number of elements available in the stack. 5• display -> Should print the elements of stack in LIFO manner (space-separated) ending with a line-break.

**2. Approach :**

So, our main focus will be to complete these above stated functions. For that we first take a look at the signatures which are already present on the code editor of this question on our website.

****

Look at the constructor carefully. Whatever value is passed as "cap" is now the size of the array. Yes, array, we use arrays to build our stack. And tos (top of stack; it points toward the topmost index of array at which value is placed) is initialized to -1. Because the array "data" is empty in the starting.

Push

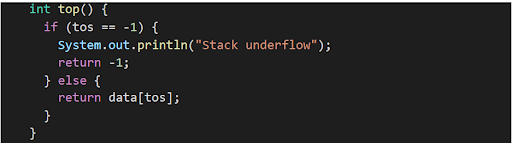
So, talking of our first function, "push", whenever an element is pushed it means that some value is added at the top of the stack (array). In that case tos is incremented by 1. And value is placed at index tos of data array. But what if the stack is already full which means that tos is equal to ( data.length - 1) it means that no more elements can be stored in the data array. In this case our job is to print, "Stack Overflow".

****

Time Complexity : O(1) It takes constant time to decrement tos and return val. SPACE COMPLEXITY : O(1) Since no extra space is used therefore space complexity remains constant.

Top

Third function is, "top", whenever this function is called then we simply need to return the value of the topmost element, i.e. the value stored at tos is collected and returned. But what if the stack is already empty which means that tos is equal to (- 1) therefore no element is present in the array. In this case our job is to print, "Stack Underflow" and return -1.

****

Time Complexity : O(1) It takes constant time to return the data[tos]. SPACE COMPLEXITY : O(1) Since no extra space is used therefore space complexity remains constant.

Size

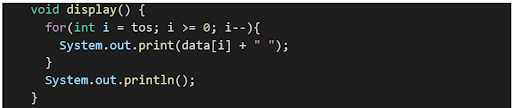
Fourth function is "size". Whenever function size is called then we need to return the size of the used stack not the length of array (cap). It is easy to relate size with tos. If tos is -1 then the size of the stack is 0. If tos is 0 then size is 1. So we can say that at any moment if we add 1 to tos then we get the size of the array. Therefore we simply return tos + 1, whenever the size of the function is called.

****

Time Complexity : O(1) It takes constant time to return tos+1. SPACE COMPLEXITY : O(1) Since no extra space is used therefore space complexity remains constant.

Display

Last function is "display". Whenever display is called, we need to print our stack. For that we run a for loop from tos to 0 on array data and print data[i].

****

Time Complexity : O(n) A "for" loop has been used to print the entire stack which takes O(n) time, n being the number of elements in the stack. SPACE COMPLEXITY : O(1) Since no extra space is used therefore space complexity remains constant.

**3. Code:**

ConsoleJava

import java.io.\*;

import java.util.\*;

public class Main {

public static class CustomStack {

int[] data;

int tos;

public CustomStack(int cap) {

data = new int[cap];

tos = -1;

}

int size() {

return tos + 1;

}

void display() {

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

System.out.print(data[i] + " ");

}

System.out.println();

}

void push(int val) {

if (tos == data.length - 1) {

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

} else {

tos++;

data[tos] = val;

}

}

int pop() {

if (tos == -1) {

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

return -1;

} else {

int val = data[tos];

tos--;

return val;

}

}

int top() {

if (tos == -1) {

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

return -1;

} else {

return data[tos];

}

}

}

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

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

int n = Integer.parseInt(br.readLine());

CustomStack st = new CustomStack(n);

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());

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

st.display();

}

str = br.readLine();

}

}

}