*CSE 102*

**Stack and Queue**

**Stack:**

* Conceptually it is like a “stack of books”, which you can add or remove “books” but only from the top. It is a LIFO (last in first out) structure.
* Main operations associated with a stack are adding a new element on top of it (aka push), removing an element from the top (aka pop), peeking the element on the top, and checking the size.
* In Java Collections we have a Stack class which relies on (extends) Vector class we have learned before.

**Queue:**

* Conceptually it resembles a real-life queue where people leave it from front (after being served let’s say) and new people join from the behind. It is a FIFO (first in first out) structure.
* Main operations associated with a queue are adding a new element to queue (aka enqueue), remove an element from the queue (aka dequeue), peeking the front element of the queue and checking the size.
* We’ll use LinkedList class of Java for our queue needs.

**Exercises**:

1. What is wrong with the following program?

**public** **static** **void** main(String[] args) {

Stack<String> stack = **new** Stack<>();

stack.push("item");

stack.pop("item");

}

1. A stack must be created with an initial capacity.
2. Identifier names cannot be the same as their type.
3. Pop method does not take any parameter.
4. Push method needs one more parameter.
5. What is wrong with the following program?

**public** **static** **void** main(String[] args) {

Stack<Character> st = **new** Stack<>();

st.push(**null**);

st.pop();

st.pop();

}

1. You cannot push null values to a character stack.
2. You must provide a parameter for pop method, because you must specify what you want to remove.
3. Push method needs to know position information, one more argument is needed.
4. You cannot pop an element from an empty stack.
5. What does the following program do?

**public** **static** **void** main(String[] args) {

Scanner sc = **new** Scanner(System.***in***);

Stack<Character> st = **new** Stack<>();

**char** c;

**while**((c = sc.next().charAt(0)) != 'q')

st.push(c);

**while**(!st.empty())

System.***out***.println(st.pop());

}

1. Stores characters until a ‘q’ is entered and prints stored characters to the screen in reverse order.
2. Stores characters until a ‘q’ is entered and prints the second last (right before ‘q’) character to the screen.
3. Stores characters until a ‘q’ is entered and prints stored characters to the screen in the order they were given.
4. Waits for user to enter a ‘q’ character and begins storing the characters thereafter.
5. Do you think the following program will terminate? Assume that your computer has infinite memory.

**public** **static** **void** main(String[] args) {

LinkedList<Double> queue = **new** LinkedList<>();

**while**(**true**)

**if**(Math.*random*() < ??)

queue.removeFirst();

**else**

queue.addLast(Math.*random*());

}

1. Yes
2. No, since the condition of while loop is always true.
3. Depends on the value that replaces the question marks. If it is larger than 0.5, the program will halt eventually almost for sure.
4. Depends on the value that replaces the question marks. If it is less than 0.5, the program will halt eventually almost for sure.
5. The following program is supposed to implement a simple calculator. What should be written in place of the comment?

**public** **static** **void** main(String[] args) {

Scanner sc = **new** Scanner(System.***in***);

Stack<Integer> st = **new** Stack<>();

**while**(**true**) {

String token = sc.next();

**switch**(token) {

**case** "+":

st.push(st.pop() + st.pop());

**break**;

**case** "=":

System.***out***.println(st.peek());

**break**;

**default**:

// here

}

}

}

1. **break**;
2. st.push(Integer.*valueOf*(token));
3. st.push(token);
4. st.pop();
5. Write a program which provides a command-line interface to a stack of strings. Your program should push a string to a stack when user enters “push *string*”. Likewise it should pop a string from the stack and prints it to the screen when user enters “pop”. The program should warn the user when she tries to pop from an empty stack. Lastly, when user enters “quit”, the program should report the number of items remaining in the stack and terminate.
6. Implement the same program for a queue structure, use addLast and removeFirst methods of LinkedList class.
7. Suppose you have two stacks of strings, one of them is empty and the other is not. You want to transfer the content from one to other without changing the order. Write a function which takes these stacks as arguments and performs this task. (You can assume that the first argument will always be the non-empty stack.)

**ANSWERS**

1. C
2. D
3. A
4. C
5. B
6. **public** **static** **void** main(String[] args) {

Scanner sc = **new** Scanner(System.***in***);

Stack<String> st = **new** Stack<>();

String in;

**while**(!(in = sc.next()).equals("quit"))

**if**(in.equals("push"))

st.push(sc.next());

**else** **if**(in.equals("pop"))

**if**(st.empty())

System.***out***.println("stack is empty!");

**else**

System.***out***.println(st.pop());

System.***out***.println(st.size() + " elements left.");

}

1. **public** **static** **void** main(String[] args) {

Scanner sc = **new** Scanner(System.***in***);

LinkedList<String> q = **new** LinkedList<>();

String in;

**while**(!(in = sc.next()).equals("quit"))

**if**(in.equals("enqueue"))

q.addLast(sc.next());

**else** **if**(in.equals("dequeue"))

**if**(q.size() == 0)

System.***out***.println("queue is empty!");

**else**

System.***out***.println(q.removeFirst());

System.***out***.println(q.size() + " elements left.");

}

1. **static** **void** stackTransfer(

Stack<String> s1,

Stack<String> s2)

{

Stack<String> temp = **new** Stack<>();

**while**(!s1.empty())

temp.push(s1.pop());

**while**(!temp.empty())

s2.push(temp.pop());

}