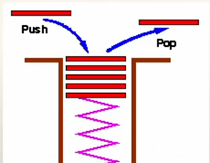
# Stacks



Stacks can be visualized as a Candy Dispenser.

A stack is a collection of items , all of the same type, where items are

added to and removed from the top of the stack.

Two Functions used to manipulate the stack

Push – Adding a item from **top** of the stack.

Pop – remove an item from the **top** of the stack.

Since we can only manipulate the top of the stack, either we can add or remove only on the top of the stack, stack access storage is called LIFO Stack (Last in First Out)

A stack is also an abstract data type (ADT), meaning the definition is the same in any language, python, java or C languages

<http://www.youtube.com/watch?v=ux0aQF2mbUk>



<http://en.wikipedia.org/wiki/Stack_(data_structure)>

Read the above resource and answer the following questions:

1. What is a Stack?
2. How is a Stack different from ArrayList and Array?
3. What are the common operations that can be performed on a Stack?

<http://download.oracle.com/javase/7/docs/api/java/util/Stack.html>

Briefly go through the above Java API for Stack and also refer to video in youtube.

The table below shows some methods that are commonly associated with the stack data structure. Find out what these methods are used for.

**Num Method Description**

1 push(item) Pushes an item onto the top of this stack

2 pop() Removes the object at the top of this stack and returns that object as the value of this function.

3 peek() Looks at the object at the top of this stack without removing it from the stack.

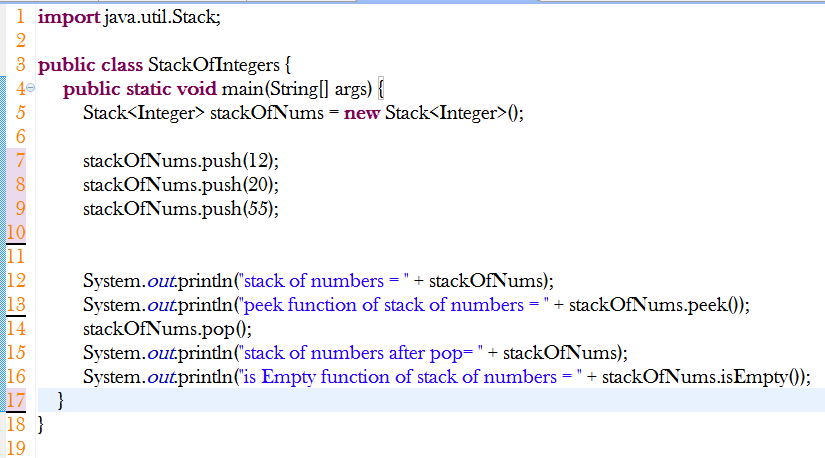
4 size() Returns the number of components in this vector.

5 empty() checking empty string

6 clear() removes all of the elements from this list

# Experimenting Stack class

Try this stack of Integers and analyze the code.



1. Write down the output of the above code segment.

|  |
| --- |
| Stack of numbers [12,20,55] |

1. Write down the size of the stack at line number 12.

|  |
| --- |
| 3 |

1. At line number 12, write down the elements in a stack

|  |
| --- |
| 12,30,55 |

1. Which method helps to remove the elements from the stack.

|  |
| --- |
| Pop() |

1. What is the element at the top and bottom of the stack after the code segment is executed.

|  |
| --- |
|  |

1. Add an another element (33) into the stack stackOfNums.

|  |
| --- |
|  |

1. Add one line of code that empty the stack and clear all the elements in the stack

|  |
| --- |
|  |

1. Write a code to implement the stack of numbers using arraylists.

|  |
| --- |
|  |

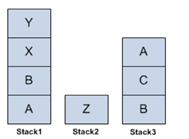
1. Comparison of arraylists methods, stack and array

|  |  |  |  |
| --- | --- | --- | --- |
| **Description** | **Array** | **ArrayLists** | **Stack** |
| Declare a variable that holds a list of colors of a rainbow | String[]rainbow = new String[7]; | ArrayList< String >rainbow = new ArrayList<String>() | Stack<String>rainbow = new Stack<String>(); |
| Add the colors into the list | rainbow[0] = “red”  rainbow[1] = “orange” | rainbow.add(“red”);  rainbow.add(“orange”); | rainbow.push(“red”);  rainbow.push(orange”); |
| Remove the color red from the list | rainbow[0] = null | rainbow.remove(0) | rainbow.pop()  rainbow.pop()  because there are 2 stack, the first pop will shift the data to the second stack. The second pop will shift the data out since there is no more stack. |
| Display the list of colors | for(int i = 0;i<rainbow.length;i++){system.out.printIn(rainbow[i]);  } | system.out.printIn(rainbow); | system.out.printIn(rainbow); |
| Size of the list | rainbow.length | rainbow.size() | rainbow.size() |
| Get the Top / bottom item in the list | bottom item  rainbow[0]  top item  rainbow[rainbow.length-1] | bottom item  rainbow.get(0);  top item  rainbow.get(rainbow.size()-1) | top item  rainbow.peek()  bottom item  rainbow.peek()  rainbow.get(0); |
|  |  |  |  |

1. Write down a code to add your team members names into a stack and name your stack as **myTeam**

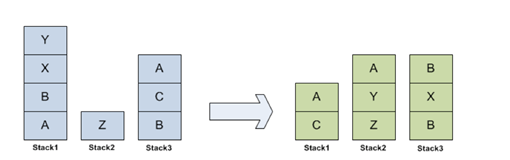
|  |
| --- |
| Stack<Integer>myTeam = **new** Stack<String>();  MyTeam.push(“Xavier”);  MyTeam.push(“Moe”);  MyTeam.push(“ChangLun”);  MyTeam.push(“Lester”); |

The figure below shows three stacks containing letters. Write code to create the above three stacks using the Stack class



1. Write code to create the above three stacks, Stack1, Stack2 and Stack3 using the Stack class

|  |
| --- |
| **import** java.util.Stack;  **public** **class** StackOfThree {  **public** **static** **void** main(String[] args) {  Stack<Character> Stack1 = **new** Stack<Character>();  Stack<Character> Stack2 = **new** Stack<Character>();  Stack<Character> Stack3 = **new** Stack<Character>();    // add characters for Stack 1  Stack1.push('A'); // single quote for single character, double quote for string  Stack1.push('B');  Stack1.push('X');  Stack1.push('Y');    //add characters for Stack 2  Stack2.push('Z');    //add characters for Stack 3  Stack3.push('B');  Stack3.push('C');  Stack3.push('A');  }  } |



1. Using the appropriate stack operations, write code to move the letters among the three stacks to produce the stacks shown on the right in the figure above.

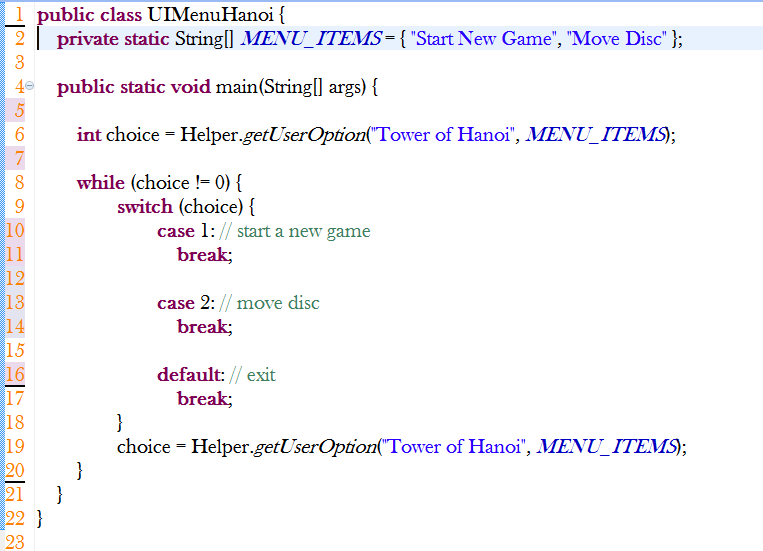
|  |  |  |
| --- | --- | --- |
| Num | Transfer | Java Code |
| 1 | Move Y from Stack1 to Stack2 | stack2.push(stack1.pop()); |
| 2 | Move A from Stack3 to Stack2 | Stack2.push(stack3.pop()); |
| 3 | Move C from Stack3 to Stack2 | Stack2.push(stack3.pop()); |
| 4 | Move X from Stack1 to Stack3 | Stack3.push(stack1.pop()); |
| 5 | Move B from Stack1 to Stack3 | Stack3.push(stack1.pop()); |
| 6 | Move A from Stack1 to Stack3 | Stack3.push(stack1.pop()); |
| 7 | Move C from Stack2 to Stack1 | Stack1.push(stack2.pop()); |
| 8 | Move A from Stack3 to Stack1 | Stack1.push(stack3.pop()); |

Complete the codes get the end result by using the above table to help you

|  |
| --- |
|  |

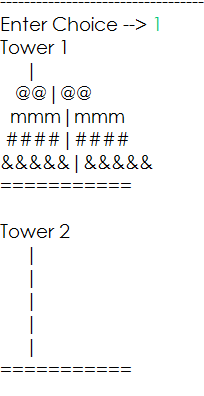
# Implementing the Stack class

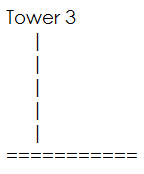
Create a project in Eclipse. Create a UI class, which displays the menu



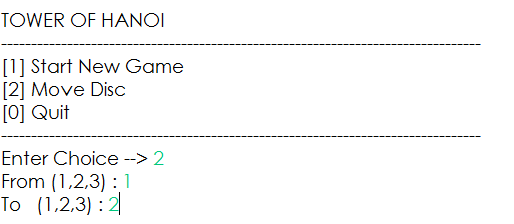
Modelling the Problem

* When user enters 1 to start the game, the three towers must be displayed as shown below. The tower 1 will have 4 disks as shown and the other two towers will be empty.
* The disc is a string and each tower is a stack of strings. When a game is started, the stack of towers must be initialized.





* When user enters 2 to move the disc, the user must be given a choice to choose the tower from which the disc has to be moved and then again a choice of the tower to which the disc has to be moved to.



View this to move the disc accordingly. Number of moves required is 15 moves. The minimum number of moves required to solve a Tower of Hanoi puzzle is - 1, where n is the number of disks.

https://www.youtube.com/watch?v=z6lBOAzjvhQ

# Completing the Problem

Every **Tower class** is a Stack of Disc.

You will need 3 Towers.

Map the following tower operations to stack operations:

**Tower Stack Comments**

Removing a Disc pop() Same as normal Stack

Placing a Disc push Only Disc smaller than the top Disc can be placed

So you have to handle this when you are placing Disc onto a Tower.

Implement the game

<http://www.youtube.com/watch?v=bIgjzlumfsQ> - good one

<http://www.youtube.com/watch?v=F1F2imiOJfk>

<http://www.youtube.com/watch?v=5_6nsViVM00>

<http://www.youtube.com/watch?v=CvVOpAfTFUM>

*End of Worksheet*