20MCA132 OBJECT ORIENTED PROGRAMMING LAB

Notes(CO4)

SUBMITTED BY

VIVIN V. ABRAHAM
R MCA-2020-S2
ROLL NO : 42

SUBMITTED TO ,

Sis Elsin

STACK

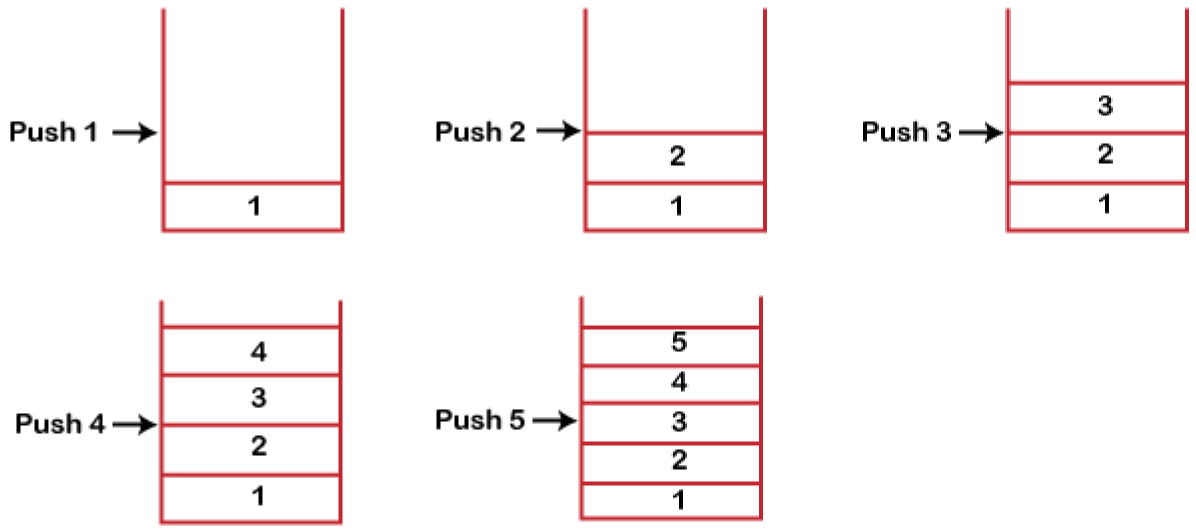
Stack is a linear data structure which follows a particular order in which the operations are performed. The order may be LIFO(Last In First Out)

- It is called as stack because it behaves like a real-world stack, piles of books, etc.
- A Stack is an abstract data type with a pre-defined capacity, which means that it can store the elements of a limited size.
- It is a data structure that follows some order to insert and delete the elements, and that order can be LIFO or FILO.

Working of Stack

Stack works on the LIFO pattern. As we can observe in the below figure there are five memory blocks in the stack; therefore, the size of the stack is 5.

Suppose we want to store the elements in a stack and let's assume that stack is empty. We have taken the stack of size 5 as shown below in which we are pushing the elements one by one until the stack becomes full.



Since our stack is full as the size of the stack is 5. In the above cases, we can observe that it goes from the top to the bottom when we were entering the new element in the stack. The stack gets filled up from the bottom to the top.

When we perform the delete operation on the stack, there is only one way for entry and exit as the other end is closed. It follows the LIFO pattern, which means that the value entered first will be removed last. In the above case, the value 5 is entered first, so it will be removed only after the deletion of all the other elements.

Standard Stack Operations

- **push():** When we insert an element in a stack then the operation is known as a push. If the stack is full then the overflow condition occurs.
- **pop():** When we delete an element from the stack, the operation is known as a pop. If the stack is empty means that no element exists in the stack, this state is known as an underflow state.
- **isEmpty():** It determines whether the stack is empty or not.
- **isFull():** It determines whether the stack is full or not.'
- **peek():** It returns the element at the given position.
- **count():** It returns the total number of elements available in a stack.
- **change():** It changes the element at the given position.
- **display():** It prints all the elements available in the stack.

<u>BUBBLE SORTING</u>

Bubble sort is a simple sorting algorithm. This sorting algorithm is comparison-based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order. This algorithm is not suitable for large data sets as its average and worst case complexity are of O(n^2) where **n** is the number of items.

Example:

First Pass:

(**5 1** 4 2 8) –> (**1 5** 4 2 8), Here, algorithm compares the first two elements, and swaps since 5 > 1.
(1 **5 4** 2 8) –> (1 **4 5** 2 8), Swap since 5 > 4
(1 4 **5 2** 8) –> (1 4 **2 5** 8), Swap since 5 > 2
(1 4 2 **5 8**) –> (1 4 2 **5 8**), Now, since these elements are already in order (8 > 5), algorithm does not swap them.

Second Pass:

(**1 4** 2 5 8) –> (**1 4** 2 5 8)
(1 **4 2** 5 8) –> (1 **2 4** 5 8), Swap since 4 > 2
(1 2 **4 5** 8) –> (1 2 **4 5** 8)
(1 2 4 **5 8**) –> (1 2 4 **5 8**)
Now, the array is already sorted, but our algorithm does not know if it is completed. The algorithm needs one **whole** pass without **any** swap to know it is sorted.

Third Pass:

(**1 2** 4 5 8) –> (**1 2** 4 5 8)
(1 **2 4** 5 8) –> (1 **2 4** 5 8)
(1 2 **4 5** 8) –> (1 2 **4 5** 8)
(1 2 4 **5 8**) –> (1 2 4 **5 8**)

Worst and Average Case Time Complexity: O(n*n). Worst case occurs when array is reverse sorted.
Best Case Time Complexity: O(n). Best case occurs when array is already sorted.

ArrayList

Arraylist class implements List interface and it is based on an Array data structure. It is widely used because of the functionality and flexibility it offers. Most of the developers **choose Arraylist over Array** as it's a very good alternative of traditional java arrays. ArrayList is a resizable-array implementation of the List interface. It implements all optional list operations, and permits all elements, including null.

The limitation with array is that it has a **fixed length** so if it is full you cannot add any more elements to it, likewise if there are number of elements gets removed from it the memory consumption would be the same as it doesn't shrink.

On the other **ArrayList can dynamically grow and shrink** after addition and removal of elements (See the images below). Apart from these benefits ArrayList class enables us to use predefined methods of it which makes our task easy. Let's see the diagrams to understand the addition and removal of elements from ArrayList and then we will see the programs.

Methods of ArrayList class

In the above example we have used methods such as add() and remove(). However there are number of methods available which can be used directly using object of ArrayList class. Let's discuss few **important methods of ArrayList class**.

1) add(Object o): This method adds an object o to the arraylist.

```
obj.add("hello");
```

This statement would add a string hello in the arraylist at last position.

2) add(int index, Object o): It adds the object o to the array list at the given index.

```
obj.add(2, "bye");
```

It will add the string bye to the 2nd index (3rd position as the array list starts with index 0) of array list.

3) remove(Object o): Removes the object o from the ArrayList.

```
obj.remove("Chaitanya");
```

This statement will remove the string "Chaitanya" from the ArrayList.

4) remove(int index): Removes element from a given index.

```
obj.remove(3);
```

It would remove the element of index 3 (4th element of the list – List starts with o).

5) set(int index, Object o): Used for updating an element. It replaces the element present at the specified index with the object o.

```
obj.set(2, "Tom");
```

It would replace the 3rd element (index =2 is 3rd element) with the value Tom.

6) int indexOf(Object o): Gives the index of the object o. If the element is not found in the list then this method returns the value -1.

```
int pos = obj.indexOf("Tom");
```

This would give the index (position) of the string Tom in the list.

7) Object get(int index): It returns the object of list which is present at the specified index.

```
String str= obj.get(2);
```

Function get would return the string stored at 3rd position (index 2) and would be assigned to the string “str”. We have stored the returned value in string variable because in our example we have defined the ArrayList is of String type. If you are having integer array list then the returned value should be stored in an integer variable.

8) int size(): It gives the size of the ArrayList – Number of elements of the list.

```
int numberofitems = obj.size();
```

9) boolean contains(Object o): It checks whether the given object o is present in the array list if its there then it returns true else it returns false.

```
obj.contains("Steve");
```

It would return true if the string “Steve” is present in the list else we would get false.

10) clear(): It is used for removing all the elements of the array list in one go. The below code will remove all the elements of ArrayList whose object is obj.

```
obj.clear();
```