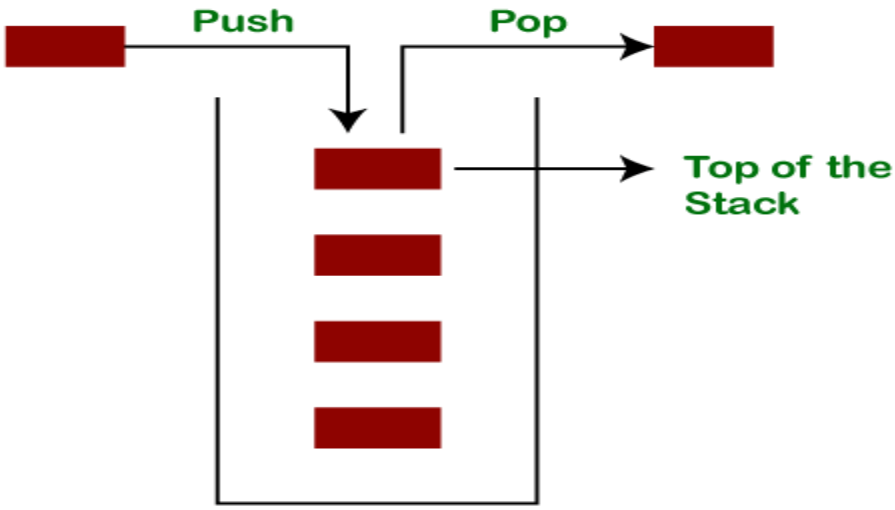
ADVANCE PROGRAMMING (JAVA LANGUAGE) PART 2

By
Dr. Amal Sufuih Ajrash

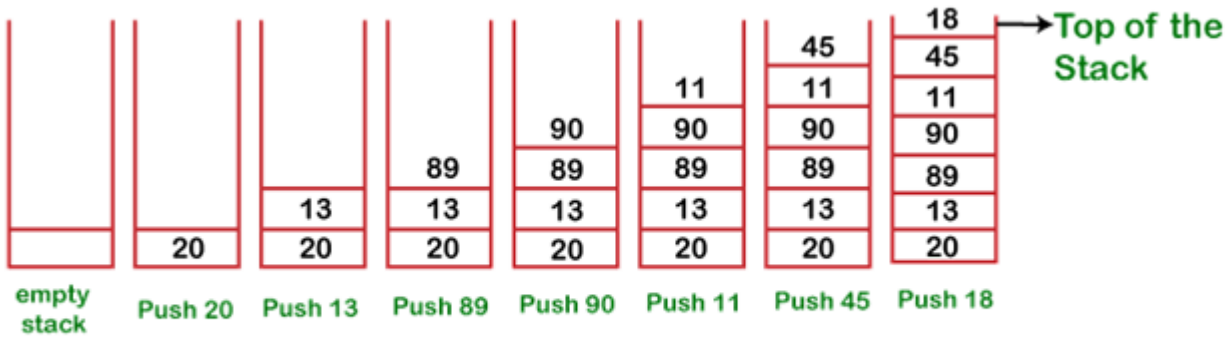
Java Stack

The **stack** is a linear data structure that is used to store the collection of objects. It is based on **Last-In-First-Out** (LIFO). **Stack class** provides different operations such as push, pop, search, etc.

The stack data structure has the two most important operations that are **push** and **pop**. The push operation inserts an element into the stack and pop operation removes an element from the top of the stack. Let's see how they work on stack.

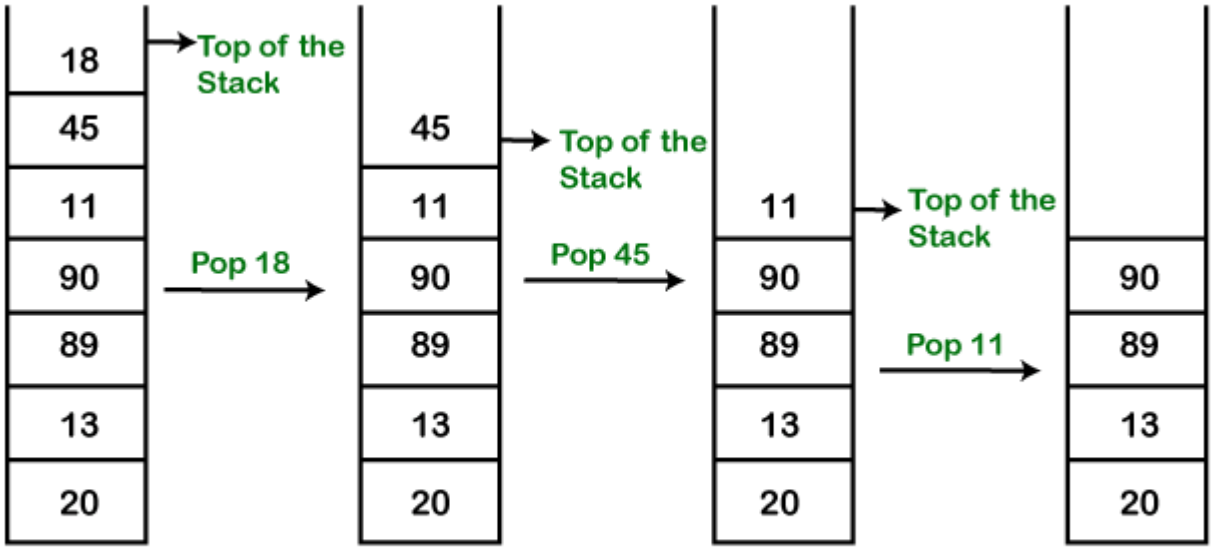


Let's push 20, 13, 89, 90, 11, 45, 18, respectively into the stack.



Push operation

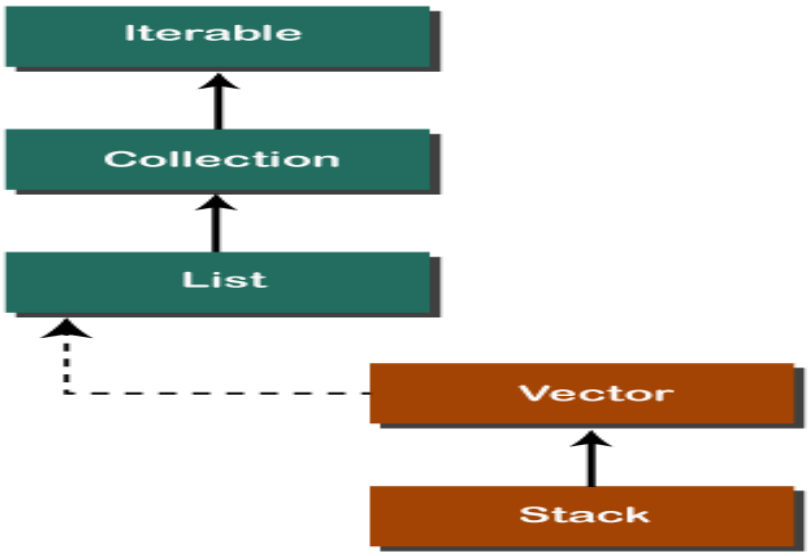
Let's remove (pop) 18, 45, and 11 from the stack.



Pop operation

Java Stack Class

In Java, **Stack** is a class that falls under the Collection framework that extends the **Vector** class. It also implements interfaces **List, Collection, Iterable, Cloneable, Serializable.** It represents the LIFO stack of objects. Before using the Stack class, we must import the java.util package. The stack class arranged in the Collections framework hierarchy, as shown below.



Stack Class Constructor

The Stack class contains only the **default constructor** that creates an empty stack.

1. **public** Stack()

Creating a Stack

If we want to create a stack, first, import the java.util package and create an object of the Stack class.

Stack stk = **new** Stack();

Or

Stack<type> stk = **new** Stack<>();

Where type denotes the type of stack like Integer, String, etc.

Methods of the Stack Class

We can perform push, pop, peek and search operation on the stack. The Java Stack class provides mainly five methods to perform these operations.

Method	Modifier and Type	Method Description
empty()	boolean	The method checks the stack is empty or not.
push(E item)	E	The method pushes (insert) an element onto the top of the stack.
pop()	E	The method removes an element from the top of the stack and returns the same element as the value of that function.
peek()	E	The method looks at the top element of the stack without removing it.
search(Objecto)	int	The method searches the specified object and returns the position of the object.

Stack Class empty() Method

The **empty()** method of the Stack class check the stack is empty or not. If the stack is empty, it returns true, else returns false.

Syntax

1. **public boolean** empty()

Returns: The method returns true if the stack is empty, else returns false.

In the following example, we have created an instance of the Stack class. After that, we have invoked the empty() method two times. The first time it returns **true** because we have not pushed any element into the stack. After that, we have pushed elements into the stack. Again we have invoked the empty() method that returns **false** because the stack is not empty.

StackEmptyMethodExample.java

```
1. import java.util.Stack;
2. public class StackEmptyMethodExample
3. {
4. public static void main(String[] args)
5. {
6. //creating an instance of Stack class
7. Stack<Integer> stk= new Stack<>();
8. // checking stack is empty or not
9. boolean result = stk.empty();
10.System.out.println("Is the stack empty? " + result);
11.// pushing elements into stack
12.stk.push(78);
13.stk.push(113);
14.stk.push(90);
15.stk.push(120);
16.//prints elements of the stack
17.System.out.println("Elements in Stack: " + stk);
18.result = stk.empty();
19.System.out.println("Is the stack empty? " + result);
20.}
21.}
```

Output:

```
Is the stack empty? true
Elements in Stack: [78, 113, 90, 120]
Is the stack empty? false
```

Stack Class push() Method

The method inserts an item onto the top of the stack. It passes a parameter **item** to be pushed into the stack.

Syntax

1. **public** E push(E item)

Parameter: An item to be pushed onto the top of the stack.

Returns: The method returns the argument that we have passed as a parameter.

Stack Class pop() Method

The method removes an object at the top of the stack and returns the same object. It throws **EmptyStackException** if the stack is empty.

Syntax

1. **public** E pop()

Returns: It returns an object that is at the top of the stack.

Let's implement the stack in a Java program and perform push and pop operations.

StackPushPopExample.java

```
1. import java.util.*;
2. public class StackPushPopExample
3. {
4. public static void main(String args[])
5. {
6. //creating an object of Stack class
7. Stack <Integer> stk = new Stack<>();
8. System.out.println("stack: " + stk);
```

```
9.  //pushing elements into the stack
10.pushelmnt(stk, 20);
11.pushelmnt(stk, 13);
12.pushelmnt(stk, 89);
13.pushelmnt(stk, 90);
14.pushelmnt(stk, 11);
15.pushelmnt(stk, 45);
16.pushelmnt(stk, 18);
17.//popping elements from the stack
18.popelmnt(stk);
19.popelmnt(stk);
20.//throws exception if the stack is empty
21.try
22.{
23.popelmnt(stk);
24.}
25.catch (EmptyStackException e)
26.{
27.System.out.println("empty stack");
28.}
29.}
30.//performing push operation
31.static void pushelmnt(Stack stk, int x)
32.{
33.//invoking push() method
34.stk.push(new Integer(x));
35.System.out.println("push -> " + x);
36.//prints modified stack
37.System.out.println("stack: " + stk);
38.}
39.//performing pop operation
40.static void popelmnt(Stack stk)
41.{
42.System.out.print("pop -> ");
43.//invoking pop() method
44.Integer x = (Integer) stk.pop();
```

```
45.System.out.println(x);
46.//prints modified stack
47.System.out.println("stack: " + stk);
48.}
49.}
```

Output:

```
stack: []
push -> 20
stack: [20]
push -> 13
stack: [20, 13]
push -> 89
stack: [20, 13, 89]
push -> 90
stack: [20, 13, 89, 90]
push -> 11
stack: [20, 13, 89, 90, 11]
push -> 45
stack: [20, 13, 89, 90, 11, 45]
push -> 18
stack: [20, 13, 89, 90, 11, 45, 18]
pop -> 18
stack: [20, 13, 89, 90, 11, 45]
pop -> 45
stack: [20, 13, 89, 90, 11]
pop -> 11
stack: [20, 13, 89, 90]
```

Stack Class peek() Method

It looks at the element that is at the top in the stack. It also throws **EmptyStackException** if the stack is empty.

Syntax

1. **public** E peek()

Returns: It returns the top elements of the stack.

Let's see an example of the peek() method.

StackPeekMethodExample.java

```
1. import java.util.Stack;
2. public class StackPeekMethodExample
3. {
4. public static void main(String[] args)
5. {
6. Stack<String> stk= new Stack<>();
7. // pushing elements into Stack
8. stk.push("Apple");
9. stk.push("Grapes");
10.stk.push("Mango");
11.stk.push("Orange");
12.System.out.println("Stack: " + stk);
13.// Access element from the top of the stack
14.String fruits = stk.peek();
15.//prints stack
16.System.out.println("Element at top: " + fruits);
17.}
18.}
```

Output:

```
Stack: [Apple, Grapes, Mango, Orange]
Element at the top of the stack: Orange
```

Stack Class search() Method

The method searches the object in the stack from the top. It parses a parameter that we want to search for. It returns the 1-based location of the object in the stack. This top most object of the stack is considered at distance 1.

Suppose, o is an object in the stack that we want to search for. The method returns the distance from the top of the stack of the occurrence nearest the top of the stack. It uses **equals()** method to search an object in the stack.

Syntax

```
1. public int search(Object o)
```

Parameter: o is the desired object to be searched.

Returns: It returns the object location from the top of the stack. If it returns -1, it means that the object is not on the stack.

Let's see an example of the search() method.

StackSearchMethodExample.java

```
1. import java.util.Stack;
2. public class StackSearchMethodExample
3. {
4. public static void main(String[] args)
5. {
6. Stack<String> stk= new Stack<>();
7. //pushing elements into Stack
8. stk.push("Mac Book");
9. stk.push("HP");
10.stk.push("DELL");
11.stk.push("Asus");
12.System.out.println("Stack: " + stk);
13.// Search an element
14.int location = stk.search("HP");
15.System.out.println("Location of Dell: " + location);
16.}
17.}
```

Java Stack Operations

Size of the Stack

We can also find the size of the stack using the size() method of the Vector class. It returns the total number of elements (size of the stack) in the stack.

Syntax

```
1. public int size()
```

StackSizeExample.java

```
1. import java.util.Stack;
2. public class StackSizeExample
3. {
4. public static void main (String[] args)
5. {
6. Stack stk = new Stack();
7. stk.push(22);
8. stk.push(33);
9. stk.push(44);
10.stk.push(55);
11.stk.push(66);
12.// Checks the Stack is empty or not
13.boolean rslt=stk.empty();
14.System.out.println("Is the stack empty or not? " +rslt);
15.// Find the size of the Stack
16.int x=stk.size();
17.System.out.println("The stack size is: "+x);
18.}
19.}
```

Output:

```
Is the stack empty or not? false
The stack size is: 5
```