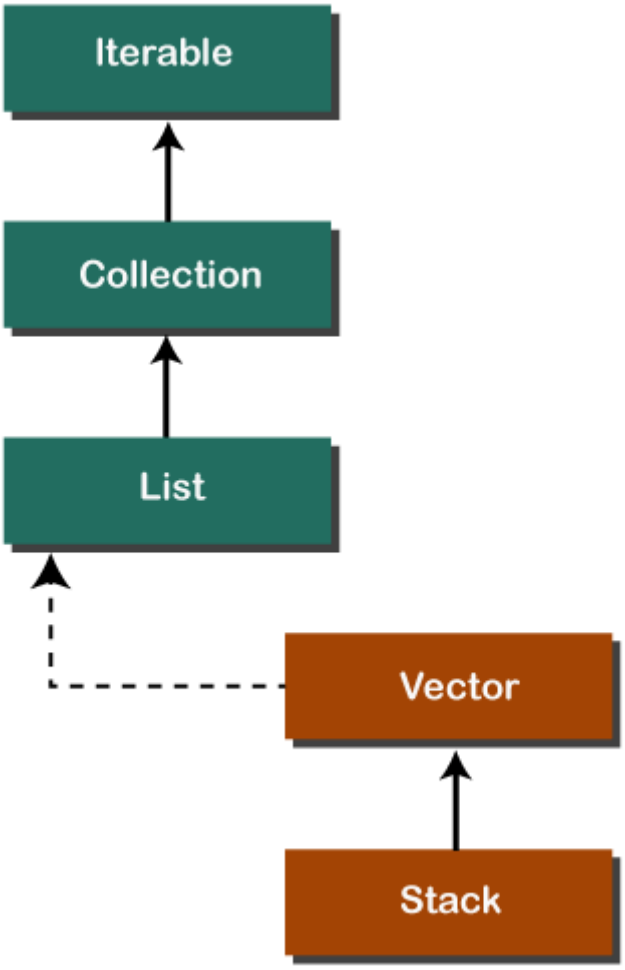
Java Stack Class

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.

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. Along with this, it also provides all the methods of the Java Vector class.

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(Object o)	int	The method searches the specified object and returns the position of the object.

1) Stack Class empty() Method

public boolean empty()

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

```
import java.util.Stack;

class StackEmptyMethodExample

{

public static void main(String[] args)

{

//creating an instance of Stack class

Stack<Integer> stk= new Stack<>();

// checking stack is empty or not

boolean result = stk.empty();

System.out.println("Is the stack empty? " + result);

// pushing elements into stack

stk.push(78);

stk.push(113);

stk.push(90);

stk.push(120);

//prints elements of the stack

System.out.println("Elements in Stack: " + stk);

result = stk.empty();
```

```
System.out.println("Is the stack empty? " + result);
}
}
```

Output:

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

2. Stack Class push() Method

The method inserts an item onto the top of the stack. It works the same as the method addElement(item) method of the Vector class. It passes a parameter item to be pushed into the stack.

Syntax

```
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.

3. 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

```
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

```
import java.util.*;
```

```
class StackPushPopExample
{
	public static void main(String args[])
	{
		//creating an object of Stack class
		Stack <Integer> stk = new Stack<>();
		System.out.println("stack: " + stk);
		//pushing elements into the stack
		pushelmnt(stk, 20);
		pushelmnt(stk, 13);
		pushelmnt(stk, 89);
		pushelmnt(stk, 90);
		pushelmnt(stk, 11);
		pushelmnt(stk, 45);
		pushelmnt(stk, 18);
		//popping elements from the stack
		popelmnt(stk);
		popelmnt(stk);
		//throws exception if the stack is empty
		try
		{
			popelmnt(stk);
		}
		catch (EmptyStackException e)
		{
			System.out.println("empty stack");
```

```
            }
        }
        //performing push operation
        static void pushelmnt(Stack stk, int x)
        {
            //invoking push() method
            stk.push(x);
            System.out.println("push -> " + x);
            //prints modified stack
            System.out.println("stack: " + stk);
        }
        //performing pop operation
        static void popelmnt(Stack stk)
        {
            System.out.print("pop -> ");
            //invoking pop() method
            Integer x = (Integer) stk.pop();
            System.out.println(x);
            //prints modified stack
            System.out.println("stack: " + stk);
        }
}
```

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]

4. 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

public E peek()

Returns: It returns the top elements of the stack.

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

StackPeekMethodExample.java

```
import java.util.Stack;
class StackPeekMethodExample
{
	public static void main(String[] args)
	{
		Stack<String> stk= new Stack<>();
		//stk.peek();
		// pushing elements into Stack
		stk.push("Apple");
		stk.push("Grapes");
		stk.push("Mango");
		stk.push("Orange");
		System.out.println("Stack: " + stk);
		// Access element from the top of the stack
		String fruits = stk.peek();
		System.out.println("Element at top: " + fruits);
	}
}
```

Output:

Stack: [Apple, Grapes, Mango, Orange]

Element at the top of the stack: Orange

5. 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. Thes topmost 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

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

```
import java.util.Stack;

class StackSearchMethodExample
{
	public static void main(String[] args)
	{
		Stack<String> stk= new Stack<>();
		//pushing elements into Stack
		stk.push("Mac Book");
		stk.push("HP");
		stk.push("DELL");
		stk.push("Asus");
		System.out.println("Stack: " + stk);
		// Search an element
		int location = stk.search("DELL");
		System.out.println("Location of Dell: " + location);
	}
}
```

- **Java Stack Operations**

1. 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

public int size()

Let's see an example of the size() method of the Vector class.

StackSizeExample.java

```
import java.util.Stack;
public class StackSizeExample
{
public static void main (String[] args)
{
Stack stk = new Stack();
stk.push(22);
stk.push(33);
stk.push(44);
stk.push(55);
stk.push(66);
// Checks the Stack is empty or not
boolean rslt=stk.empty();
System.out.println("Is the stack empty or not? " +rslt);
// Find the size of the Stack
int x=stk.size();
System.out.println("The stack size is: "+x);
}
```

}

Output:

Is the stack empty or not? false

The stack size is: 5

2. Iterate Elements

Iterate means to fetch the elements of the stack. We can fetch elements of the stack using three different methods are as follows:

Using iterator() Method

Using forEach() Method

Using listIterator() Method

Using the iterator() Method

It is the method of the Iterator interface. It returns an iterator over the elements in the stack. Before using the iterator() method import the java.util.Iterator package.

Syntax

Iterator<T> iterator()

Let's perform an iteration over the stack.

StackIterationExample1.java

```
import java.util.Iterator;

import java.util.Stack;

class StackIterationExample1

{

public static void main (String[] args)

{

//creating an object of Stack class
```

```
Stack stk = new Stack();
//pushing elements into stack
stk.push("BMW");
stk.push("Audi");
stk.push("Ferrari");
stk.push("Bugatti");
stk.push("Jaguar");
//iteration over the stack
Iterator iterator = stk.iterator();
while(iterator.hasNext())
{
Object values = iterator.next();
System.out.println(values);
}
}
}
```

Output:

BMW

Audi

Ferrari

Bugatti

Jaguar

Using the forEach() Method

Java provides a forEach() method to iterate over the elements. The method is defined in the Iterable and Stream interface.

Syntax

default void forEach(Consumer<super T>action)

Let's iterate over the stack using the forEach() method.

StackIterationExample2.java

```
import java.util.*;
class StackIterationExample2
{
public static void main (String[] args)
{
//creating an instance of Stack class
Stack <Integer> stk = new Stack<>();
//pushing elements into stack
stk.push(119);
stk.push(203);
stk.push(988);
System.out.println("Iteration over the stack using forEach() Method:");
//invoking forEach() method for iteration over the stack
stk.forEach(n ->  {System.out.println(n);});
}
}
```

Output:

```
Iteration over the stack using forEach() Method:
119
203
988
```

Using listIterator() Method

This method returns a list iterator over the elements in the mentioned list (in sequence), starting at the specified position in the list. It iterates the stack from top to bottom.

Syntax

ListIterator listIterator(int index)

Parameter: The method parses a parameter named index.

Returns: This method returns a list iterator over the elements, in sequence.

Exception: It throws IndexOutOfBoundsException if the index is out of range.

Let's iterate over the stack using the listIterator() method.

StackIterationExample3.java

```
import java.util.Iterator;
import java.util.ListIterator;
import java.util.Stack;
class StackIterationExample3
{
public static void main (String[] args)
{
Stack <Integer> stk = new Stack<>();
stk.push(119);
stk.push(203);
stk.push(988);
ListIterator<Integer> ListIterator = stk.listIterator(stk.size());
System.out.println("Iteration over the Stack from top to bottom:");
while (ListIterator.hasPrevious())
{
Integer avg = ListIterator.previous();
```

```
System.out.println(avg);
}
}
}
```

Output:

```
Iteration over the Stack from top to bottom:
988
203
119
```