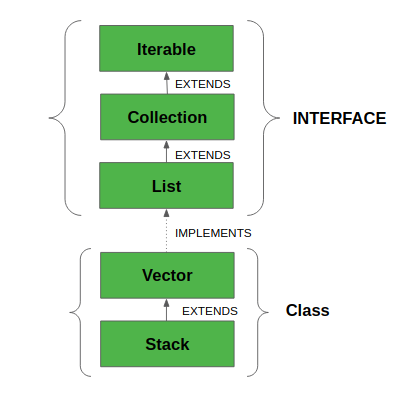
**Stack Class in Java**

* Java [Collection framework](https://www.geeksforgeeks.org/collections-in-java-2/) provides a Stack class that models and implements a [**Stack data structure**](https://www.geeksforgeeks.org/stack-data-structure/).
* basic principle of last-in-first-out.
* basic push and pop operations
* 3 more functions: empty, search, and peek.
* The class can also extends Vector
* The class can also be referred to as the subclass of Vector.

**The hierarchy of the Stack class**:



* To Create a Stack:

Stack stk = **new** Stack();

Or

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

|  |  |  |
| --- | --- | --- |
| **Method** | **Modifier and Type** | **Method Description** |
| [empty()](https://www.javatpoint.com/java-stack#empty) | boolean | The method checks the stack is empty or not. |
| [push(E item)](https://www.javatpoint.com/java-stack#push) | E | The method pushes (insert) an element onto the top of the stack. |
| [pop()](https://www.javatpoint.com/java-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()](https://www.javatpoint.com/java-stack#peek) | E | The method looks at the top element of the stack without removing it. |
| [search(Object o)](https://www.javatpoint.com/java-stack#search) | int | The method searches the specified object and returns the position of the object. |

### **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**

Iterator iterator = stk.iterator();

**while**(iterator.hasNext())

{

Object values = iterator.next();

System.out.println(values);

}

### **Using the forEach() Method**

stk.forEach(n -> { System.out.println(n);  });

### **Using listIterator() Method**

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);

}

**Stack Operations - Example:**

**package** StackOperations;

**import** java.util.Iterator;

**import** java.util.Stack;

**public** **class** StackExample

{

**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);

//peek element

System.***out***.println("Top element in stack: "+ stk.peek());

//Pop element

System.***out***.println("Pop top element:"+stk.pop());

//Printing all elements from the stack

System.***out***.println("Stack elements:");

Iterator iterator = stk.iterator();

**while**(iterator.hasNext())

{

Object values = iterator.next();

System.***out***.println(values);

}

result = stk.empty();

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

}

}

**Infix to Postfix Conversion:**

public class InfixToPostfix

{

private static boolean isOperator(char c)

{

return c == '+' || c == '-' || c == '\*' || c == '/' || c == '^'

|| c == '(' || c == ')';

}

private static boolean isLowerPrecedence(char op1, char op2)

{

switch (op1)

{

case '+':

case '-':

return !(op2 == '+' || op2 == '-');

case '\*':

case '/':

return op2 == '^' || op2 == '(';

case '^':

return op2 == '(';

case '(':

return true;

default:

return false;

}

}

public static String convertToPostfix(String infix)

{

Stack<Character> stack = new Stack<Character>();

StringBuffer postfix = new StringBuffer(infix.length());

char c;

for (int i = 0; i < infix.length(); i++)

{

c = infix.charAt(i);

if (!isOperator(c))

{

postfix.append(c);

}

else

{

if (c == ')')

{

while (!stack.isEmpty() && stack.peek() != '(')

{

postfix.append(stack.pop());

}

if (!stack.isEmpty())

{

stack.pop();

}

}

else

{

if (!stack.isEmpty() && !isLowerPrecedence(c, stack.peek()))

{

stack.push(c);

}

else

{

while (!stack.isEmpty() && isLowerPrecedence(c, stack.peek()))

{

Character pop = stack.pop();

if (pop != '(')

{

postfix.append(pop);

}

}

}

stack.push(c);

}

}

}

return postfix.toString();

}

public static void main(String[] args)

{

System.out.println(convertToPostfix("A\*B-(C+D)+E"));

}

}