Stack Implementation by Stack<E> class.

Stack<E> is a class that implements the stack data structure. Stack<E> extends Vector<E>. In turn Vector<E> extends [AbstractList<E>](http://data-structure-learning.blogspot.com/2015/05/java-collections-part-11-abstractlist.html) and implements [List<E>](http://data-structure-learning.blogspot.com/2015/05/java-collections-part-5list-interface.html).

Note: If you call **pop()** or **peek()** on **empty stack** then **EmptyStackException** is thrown.

We create new instance of Stack<E> class in constructor.

StackX class is used to demonstrate the working of Stack<E>.

**Constructor of StackX** class.

/\*\*

\* Constructor for StackX class

\* \*/

**public** StackX() {

// Create new instance of Stack<String>

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

}

push(String s)

pushes element of type String in the stack. This method is used as wrapper for Stack<> class push.

/\*\*

\* Push String s in stack.

\* \*/

**public** String push(String s) {

**return** stack.push(s);

}

pop()

pops element from the stack. This method is used as wrapper for Stack<> class pop.

/\*\*

\* returns and removes the top

\* element from Stack.

\* \*/

**public** String pop() {

**if** (!stack.isEmpty()) {

**return** stack.pop();

}

**throw** **new** EmptyStackException();

}

peek()

returns top element from the stack. This method is used as wrapper for Stack<> class peek.

/\*\*

\* returns top element from Stack

\* without removing the element.

\* \*/

**public** String peek() {

**if** (!stack.isEmpty()) {

**return** stack.peek();

}

**throw** **new** EmptyStackException();

}

exists(String s)

search for element in stack from top. This method is used as wrapper for Stack<> class search.

/\*\*

\* searches for given element in Stack.

\* uses lastIndexOf(object o) of Vector class.

\* \*/

**public** **boolean** exists(String str) {

**return** stack.search(str) >= 0;

}

isEmpty()

returns boolean. true if stack is empty else false.

/\*\*

\* Checks if stack is empty or not.

\* returns boolean.

\* \*/

**public** **boolean** isEmpty() {

**return** stack.isEmpty();

}

toString()

return String version of entire stack.

/\*\*

\* returns String version of stack.

\* If insertions is as Element 1, Element 2, Element 3.

\* return string "Element 1, Element 2, Element 3"

\* \*/

**public** String toString() {

StringBuilder sb = **new** StringBuilder();

**for** (String str : stack) {

sb.append(str).append(" ");

}

**return** sb.toString();

}

Below is the code for entire class.

**package** stackimpl;

**import** java.util.EmptyStackException;

**import** java.util.Stack;

/\*\*

\* This class demonstrates the working of Stack<E>

\* class of Java Collection Framework.

\*

\* \*/

**public** **class** StackX {

// Declare object of Stack<Integer>

Stack<String> stack;

/\*\*

\* Constructor for StackX class

\* \*/

**public** StackX() {

// Create new instance of Stack<String>

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

}

/\*\*

\* Push String s in stack.

\* \*/

**public** String push(String s) {

**return** stack.push(s);

}

/\*\*

\* returns and removes the top

\* element from Stack.

\* \*/

**public** String pop() {

**if** (!stack.isEmpty()) {

**return** stack.pop();

}

**throw** **new** EmptyStackException();

}

/\*\*

\* Checks if stack is empty or not.

\* returns boolean.

\* \*/

**public** **boolean** isEmpty() {

**return** stack.isEmpty();

}

/\*\*

\* returns top element from Stack

\* without removing the element.

\* \*/

**public** String peek() {

**if** (!stack.isEmpty()) {

**return** stack.peek();

}

**throw** **new** EmptyStackException();

}

/\*\*

\* searches for given element in Stack.

\* uses lastIndexOf(object o) of Vector class.

\* \*/

**public** **boolean** exists(String str) {

**return** stack.search(str) >= 0;

}

**public** **static** **void** main(String[] args) {

StackX s=**new** StackX();

s.push("Element 1");

s.push("Element 2");

s.push("Element 3");

System.***out***.println(s.stack+" stack size is "+s.size());

s.pop();

System.***out***.println("After pop operation "+s.stack+" stack size is "+s.size());

s.pop();

System.***out***.println("After pop operation "+s.stack+" stack size is "+s.size());

}

}

Output:

[Element 1, Element 2, Element 3] stack size is 3

After pop operation [Element 1, Element 2] stack size is 2

After pop operation [Element 1] stack size is 1