Stack Class in Java

Difficulty Level : Easy   ●   Last Updated : 24 Mar, 2023

Read	Discuss	Courses	Practice	Video

Java **Collection framework** provides a Stack class that models and implements a **Stack data structure**. The class is based on the basic principle of last-in-first-out. In addition to the basic push and pop operations, the class provides three more functions of empty, search, and peek. The class can also be said to extend Vector and treats the class as a stack with the five mentioned functions. The class can also be referred to as the subclass of Vector.

The below diagram shows the **hierarchy of the Stack class**:

The class supports one *default constructor* **Stack()** which is used to *create an empty stack*.

Declaration:

Login

Register

and deserialized.

- **Cloneable:** This is an interface in Java which needs to be implemented by a class to allow its objects to be cloned.
- **Iterable<E>:** This interface represents a collection of objects which is iterable – meaning which can be iterated.
- **Collection<E>:** A Collection represents a group of objects known as its elements. The Collection interface is used to pass around collections of objects where maximum generality is desired.
- **List<E>:** The List interface provides a way to store the ordered collection. It is a child interface of Collection.
- **RandomAccess:** This is a marker interface used by List implementations to indicate that they support fast (generally constant time) random access.

How to Create a Stack?

In order to create a stack, we must import **java.util.stack** package and use the Stack() constructor of this class. The below example creates an empty Stack.

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

Here E is the type of Object.

Example:

Java

```java
// Java code for stack implementation

import java.io.*;
import java.util.*;

class Test
{
    // Pushing element on the top of the stack
    static void stack_push(Stack<Integer> stack)
    {
```

Start Your Coding Journey Now!

```java
    // Popping element from the top of the stack
    static void stack_pop(Stack<Integer> stack)
    {
        System.out.println("Pop Operation:");

        for(int i = 0; i < 5; i++)
        {
            Integer y = (Integer) stack.pop();
            System.out.println(y);
        }
    }

    // Displaying element on the top of the stack
    static void stack_peek(Stack<Integer> stack)
    {
        Integer element = (Integer) stack.peek();
        System.out.println("Element on stack top: " + element);
    }

    // Searching element in the stack
    static void stack_search(Stack<Integer> stack, int element)
    {
        Integer pos = (Integer) stack.search(element);

        if(pos == -1)
            System.out.println("Element not found");
        else
            System.out.println("Element is found at position: " + pos);
    }


    public static void main (String[] args)
    {
        Stack<Integer> stack = new Stack<Integer>();

        stack_push(stack);
        stack_pop(stack);
        stack_push(stack);
        stack_peek(stack);
        stack_search(stack, 2);
        stack_search(stack, 6);
    }
}
```

Output:

Start Your Coding Journey Now!

```
3
2
1
0
Element on stack top: 4
Element is found at position: 3
Element not found
```

Performing various operations on Stack class

1. Adding Elements: In order to add an element to the stack, we can use the *push() method*. This **push()** operation place the element at the top of the stack.

Java

```java
// Java program to add the
// elements in the stack
import java.io.*;
import java.util.*;

class StackDemo {

    // Main Method
    public static void main(String[] args)
    {

        // Default initialization of Stack
        Stack stack1 = new Stack();

        // Initialization of Stack
        // using Generics
        Stack<String> stack2 = new Stack<String>();

        // pushing the elements
        stack1.push("4");
        stack1.push("All");
        stack1.push("Geeks");

        stack2.push("Geeks");
        stack2.push("For");
        stack2.push("Geeks");
```

Start Your Coding Journey Now!

```
}
```

Output:

```
[4, All, Geeks]
[Geeks, For, Geeks]
```

2. Accessing the Element: To retrieve or fetch the first element of the Stack or the element present at the top of the Stack, we can use **peek()** method. The element retrieved does not get deleted or removed from the Stack.

Java

```java
// Java program to demonstrate the accessing
// of the elements from the stack
import java.util.*;
import java.io.*;

public class StackDemo {

    // Main Method
    public static void main(String args[])
    {
        // Creating an empty Stack
        Stack<String> stack = new Stack<String>();

        // Use push() to add elements into the Stack
        stack.push("Welcome");
        stack.push("To");
        stack.push("Geeks");
        stack.push("For");
        stack.push("Geeks");

        // Displaying the Stack
        System.out.println("Initial Stack: " + stack);

        // Fetching the element at the head of the Stack
        System.out.println("The element at the top of the"
                        + " stack is: " + stack.peek());

        // Displaying the Stack after the Operation
        System.out.println("Final Stack: " + stack);
```

Start Your Coding Journey Now!

```
Initial Stack: [Welcome, To, Geeks, For, Geeks]
The element at the top of the stack is: Geeks
Final Stack: [Welcome, To, Geeks, For, Geeks]
```

3. Removing Elements: To pop an element from the stack, we can use the **pop()** method. The element is popped from the top of the stack and is removed from the same.

Java

```java
// Java program to demonstrate the removing
// of the elements from the stack
import java.util.*;
import java.io.*;

public class StackDemo {
    public static void main(String args[])
    {
        // Creating an empty Stack
        Stack<Integer> stack = new Stack<Integer>();

        // Use add() method to add elements
        stack.push(10);
        stack.push(15);
        stack.push(30);
        stack.push(20);
        stack.push(5);

        // Displaying the Stack
        System.out.println("Initial Stack: " + stack);

        // Removing elements using pop() method
        System.out.println("Popped element: "
                        + stack.pop());
        System.out.println("Popped element: "
                        + stack.pop());

        // Displaying the Stack after pop operation
        System.out.println("Stack after pop operation "
                        + stack);
    }
}
```

Start Your Coding Journey Now!

```
Popped element: 20
Stack after pop operation [10, 15, 30]
```

Methods in Stack Class

METHOD	DESCRIPTION
empty()	It returns true if nothing is on the top of the stack. Else, returns false.
peek()	Returns the element on the top of the stack, but does not remove it.
pop()	Removes and returns the top element of the stack. An 'EmptyStackException' An exception is thrown if we call pop() when the invoking stack is empty.
push(Object element)	Pushes an element on the top of the stack.
search(Object element)	It determines whether an object exists in the stack. If the element is found, It returns the position of the element from the top of the stack. Else, it returns -1.

Methods inherited from class java.util.Vector

METHOD	DESCRIPTION

Start Your Coding Journey Now!

add(int index, Object obj)	Inserts the specified element at the specified position in this Vector.
addAll(Collection c)	Appends all of the elements in the specified Collection to the end of this Vector, in the order that they are returned by the specified Collection's Iterator.
addAll(int index, Collection c)	Inserts all the elements in the specified Collection into this Vector at the specified position.
addElement(Object o)	Adds the specified component to the end of this vector, increasing its size by one.
capacity()	Returns the current capacity of this vector.
clear()	Removes all the elements from this Vector.
clone()	Returns a clone of this vector.
contains(Object o)	Returns true if this vector contains the specified element.
containsAll(Collection c)	Returns true if this Vector contains all the elements in the specified Collection.
copyInto(Object []array)	Copies the components of this vector into the specified array.
elementAt(int index)	Returns the component at the specified index.
elements()	Returns an enumeration of the components of this vector.

	at least the number of components specified by the minimum capacity argument.
equals()	Compares the specified Object with this Vector for equality.
firstElement()	Returns the first component (the item at index 0) of this vector.
get(int index)	Returns the element at the specified position in this Vector.
hashCode()	Returns the hash code value for this Vector.
indexOf(Object o)	Returns the index of the first occurrence of the specified element in this vector, or -1 if this vector does not contain the element.
indexOf(Object o, int index)	Returns the index of the first occurrence of the specified element in this vector, searching forwards from the index, or returns -1 if the element is not found.
insertElementAt(Object o, int index)	Inserts the specified object as a component in this vector at the specified index.
isEmpty()	Tests if this vector has no components.
iterator()	Returns an iterator over the elements in this list in proper sequence.
lastElement()	Returns the last component of the vector.
lastIndexOf(Object o)	Returns the index of the last occurrence of the specified element in this vector, or -1

Start Your Coding Journey Now!

lastIndexOf(Object o, int index)	Returns the index of the last occurrence of the specified element in this vector, searching backward from the index, or returns -1 if the element is not found.
listIterator()	Returns a list iterator over the elements in this list (in proper sequence).
listIterator(int index)	Returns a list iterator over the elements in this list (in proper sequence), starting at the specified position in the list.
remove(int index)	Removes the element at the specified position in this Vector.
remove(Object o)	Removes the first occurrence of the specified element in this Vector If the Vector does not contain the element, it is unchanged.
removeAll(Collection c)	Removes from this Vector all of its elements that are contained in the specified Collection.
removeAllElements()	Removes all components from this vector and sets its size to zero.
removeElement(Object o)	Removes the first (lowest-indexed) occurrence of the argument from this vector.
removeElementAt(int index)	Deletes the component at the specified index.
removeRange(int fromIndex, int toIndex)	Removes from this list all the elements whose index is between fromIndex, inclusive, and toIndex, exclusive.

Start Your Coding Journey Now!

retainAll(Collection c)	Retains only the elements in this Vector that are contained in the specified Collection.
set(int index, Object o)	Replaces the element at the specified position in this Vector with the specified element.
setElementAt(Object o, int index)	Sets the component at the specified index of this vector to be the specified object.
setSize(int newSize)	Sets the size of this vector.
size()	Returns the number of components in this vector.
subList(int fromIndex, int toIndex)	Returns a view of the portion of this List between fromIndex, inclusive, and toIndex, exclusive.
toArray()	Returns an array containing all of the elements in this Vector in the correct order.
toArray(Object []array)	Returns an array containing all of the elements in this Vector in the correct order; the runtime type of the returned array is that of the specified array.
toString()	Returns a string representation of this Vector, containing the String representation of each element.
trimToSize()	Trims the capacity of this vector to be the vector's current size.

Prioritize use of Deque over Stack -:

The Stack class in Java is a legacy class and inherits from **Vector in Java**. It is a thread-safe

Start Your Coding Journey Now!

Java

```java
// A Java Program to show implementation
// of Stack using ArrayDeque

import java.util.*;

class GFG {
    public static void main (String[] args) {
        Deque<Character> stack = new ArrayDeque<Character>();
        stack.push('A');
        stack.push('B');
        System.out.println(stack.peek());
        System.out.println(stack.pop());
    }
}
```

Output:

```
B
B
```

One more reason to use Deque over Stack is Deque has the ability to use streams convert to list with keeping LIFO concept applied while Stack does not.

Java

```java
import java.util.*;
import java.util.stream.Collectors;

class GFG {
    public static void main (String[] args) {

        Stack<Integer> stack = new Stack<>();
        Deque<Integer> deque = new ArrayDeque<>();

        stack.push(1);//1 is the top
        deque.push(1);//1 is the top
        stack.push(2);//2 is the top
        deque.push(2);//2 is the top
```

Start Your Coding Journey Now!

```java
        System.out.println();

    List<Integer> list2 = deque.stream().collect(Collectors.toList());//[2,1]
        System.out.println("Using Deque -");
        for(int i = 0; i < list2.size(); i++){
            System.out.print(list2.get(i) + " " );
        }
        System.out.println();

    }
}
```

Output

```
Using Stack -
1 2
Using Deque -
2 1
```

Related Articles

1.   Sort a stack using a temporary stack

2.   Infix to Postfix using different Precedence Values for In-Stack and Out-Stack

3.   Find maximum in stack in O(1) without using additional stack

4.   Reversing a Stack with the help of another empty Stack

5.   Stack Permutations (Check if an array is stack permutation of other)

6.   What are the negative aspects of Java Stack class inheriting from Vector?

7.   Java.lang.Class class in Java | Set 2

8.   Java.lang.Class class in Java | Set 1

Start Your Coding Journey Now!

10.   Run-Time Stack Mechanism in Java [Use the updated images]

<div style="border:1px solid #f5c6cb; display:inline-block; padding:10px;">

Like   177
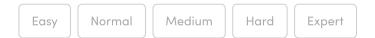
</div>

Previous   Next

Article Contributed By :

⊛ **GeeksforGeeks**

Vote for difficulty

Current difficulty : <u>Easy</u>

| Easy | Normal | Medium | Hard | Expert |

Improved By :   Chinmoy Lenka,   rohitm17,   Ganeshchowdharysadanala,   arorakashish0911,   vikas20noi,   vishalkumarsahu04

Article Tags :   Java - util package,   Java-Collections,   Java-Stack,   DSA,   Java,   Stack

Practice Tags :   Java,   Java-Collections,   Stack

Improve Article   Report Issue

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our <u>Cookie Policy</u> & <u>Privacy Policy</u>

Start Your Coding Journey Now!

Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

Company

About Us

Careers

In Media

Contact Us

Privacy Policy

Copyright Policy

Advertise with us

Languages

Python

Java

C++

GoLang

SQL

R Language

Android Tutorial

Data Structures

Array Data Structure

String Data Structure

Linked List Data Structure

Stack Data Structure

Queue Data Structure

Tree

Graph

Algorithms

Sorting

Searching

Greedy

Dynamic Programming

Pattern Searching

Recursion

Backtracking

Web Development

HTML

CSS

JavaScript

Bootstrap

ReactJS

AngularJS

NodeJS

Write & Earn

Write an Article

Improve an Article

Pick Topics to Write

Write Interview Experience

Internships

Video Internship

Start Your Coding Journey Now!

Database Management System   Machine Learning Tutorial

Software Engineering   Maths For Machine Learning

Digital Logic Design   Pandas Tutorial

Computer Graphics   NumPy Tutorial

Engineering Maths   OpenCV Python Tutorial

Interview Corner   ## Python

Company Preparation   Python Tutorial

Preparation for SDE   Python Programming Examples

Company Interview Corner   Django Tutorial

Experienced Interview   Python Projects

Internship Interview   Python Tkinter

Competitive Programming   OpenCV Python Tutorial

School [Class 6-12]   ## UPSC/SSC/BANKING

CBSE Notes for Class 8   SSC CGL Syllabus

CBSE Notes for Class 9   SBI PO Syllabus

CBSE Notes for Class 10   IBPS PO Syllabus

CBSE Notes for Class 11   UPSC Ethics Notes

CBSE Notes for Class 12   UPSC Economics Notes

English Grammar   UPSC History Notes