<http://java2learn.com/>

[Java2Learn](http://java2learn.com/)

changes your future

#linky { display:block; position:absolute; width:1100px; height:60px; z-index:1; } #branding { height:60px; } #site-title, #site-description, #sheader { position:relative; z-index:2 }

[Skip to content](l%20)

* [Home](http://java2learn.com/)
* [CoreJava](http://java2learn.com/?page_id=42)
* [Servlets](http://java2learn.com/?page_id=138)
* [Jsp](http://java2learn.com/?page_id=136)
* [Sample Page](http://java2learn.com/?page_id=2)
* [About us](http://java2learn.com/?cat=12)
* [Login](http://java2learn.com/wp-admin)

[Java2Learn » HYPERLINK "http://java2learn.com/?page\_id=42"CoreJava »](http://java2learn.com) Collections & Generics

* Top of Form 1



Bottom of Form 1

* **Site Resources**
* [CoreJava](http://java2learn.com/?page_id=42)
* [Assertions](http://java2learn.com/?page_id=154)
* [Collections & Generics](http://java2learn.com/?page_id=164)
* [Declarations & Access modifiers](http://java2learn.com/?page_id=146)
* [Exception Handling](http://java2learn.com/?page_id=151)
* [File i/o & Serialization](http://java2learn.com/?page_id=166)
* [Flow Control](http://java2learn.com/?page_id=148)
* [Garbage Collection](http://java2learn.com/?page_id=168)
* [Inner classes](http://java2learn.com/?page_id=158)
* [Internationalization](http://java2learn.com/?page_id=170)
* [java.lang package](http://java2learn.com/?page_id=162)
* [Language Fundamentals](http://java2learn.com/?page_id=49)
* [array](http://java2learn.com/?page_id=407)
* [data types](http://java2learn.com/?page_id=267)
* [identifier](http://java2learn.com/?page_id=96)
* [Literals](http://java2learn.com/?page_id=341)
* [reserved words](http://java2learn.com/?page_id=209)
* [Oops](http://java2learn.com/?page_id=156)
* [operators and assignments](http://java2learn.com/?page_id=142)
* [Threads & Concurrency](http://java2learn.com/?page_id=160)
* [Jsp](http://java2learn.com/?page_id=136)
* [Sample Page](http://java2learn.com/?page_id=2)
* [Servlets](http://java2learn.com/?page_id=138)

**Collections & Generics**

Arrays are an indexed collection of fixed no.of homogenieous data elements .The main advantage of arrays is we can represent any no.of values by using single name.

ex :- int[] x=new int[600];

byusingxvariablewecanrepresent600valuesbymeans of index we can different values

**The main limitations of object arrays are:**

1) Arrays are fixed in size i.e based on our requirement.we cannot increase or we can’t decrease the size .hence memory arrays shows worst performance

2)Arrays can hold only homogenious data elements

Student[] s=new Student[600];

s[0]=new student();

s[1]=new Customer();

But we can resolve this problem by using Object arrays

Object[] a=newObject[600];

a[0]=new Student();

a[1]=new Customer();

3) There is no underlying data structure for arrays hence for every requirment we have to code manually and automatic method support is not available

We can resolve the above problems by using the Collections.

1) Collections are growable in nature i.e based on our requirment we can increase/decrease the size.Hence w.r.t memory collections shows execellent performance.

2)Collections can hold both heterogeneious and homogenious objects.

3)Collections are built based on some dataStructures hence for every requirment automatic method support is available.

**Comparision b/w Arrays and Collections**

|  |  |
| --- | --- |
| **Arrays** | Collections |
| 1)Arrays are fixed in size | collections are not fixed in size |
| 2)w.r.t memory arrays are not recommanded to use | w.r.t memory collections are recommanded to use |
| 3)performance is high | performance is low |
| 4)Automatic method support is not available | Automatic method support is available |
| 5)Arrays concept is applicable for both primitives and objects | Collections concept is applicable only for objects but not primtives |

Collection Frame Work :-

It defines a group of interfaces and Classes Which can be used for representing a group of objects as String entity

C++ STL Container

java CollectionFrameWork Collection

**9)** **key interfaces of collection frameWork :-**

1) Collection(I):- It is the root interface of entire collection frameWork this can be used for representing a group of “indvidual” Objects as a single entity

This interface defines the most common general methods which can be applicable on any collection object

There is no concrete class which implements Collection interface directly

**Note :-difference b/w Collection and Collections**

Collection is an interface which can be used for representing a group of individual objects as a single entity

Where as a Collections is an utility class to provide several utility methods(like Sorting ,Searching etc…) to Collection Object

Both Collection and COllections are available in java.util.package only

**2)List :-**

1) It is the child interface of collection

2)If you want represent a group of individual objects as a single entity where insertion order is preserved and duplicate objects are allowed .Then we should go for list.

Collection(I)

List(I)

Array

**Note :-** Vector and Stack classes are reengineered in 1.2 version to fit in to new collection framework by implementing List interface.

**3) Set(I):-**

1) It is the child interface of collection

2) if duplicate objects are not allowed and insertions order i s not required to preserve then we should go for set interface.

HashSet and LinkedHashSet provides implementations for set Interface .

**4)SortedSet(I):-**if you want represent a group of unique objects according to some sorting order then we should go for SortedSet interface .

It is the child interface of set.

**5)NavigableSet(I):-**It is the child of SortedSet which defines several methods for Navigation.

TreeSet is the implementation class of Navigable Set interface.

dgsp

**6)Qurue(I):-**If you want represent a group of individual objects prior to processing then should go for queue interface it is the child interface of collection introduced in 1.5vQueues for typically follow FIFO order but we can change it based on our requirment

dgsp

Note:-From 1.5 version onwards LinkedList also implements Queue interface.

**7)Map(I):-If you wannt represent a group of objects as key value pairs then we should go for Map interface.**

Map interface is not child interface of collection.BothKey and value are objects.

duplicate keys are not allowed but values can be duplicated.

**8)SortedMap(I):-** if you want represent all entries according to some sorting order of keys then we should go for SortedMap interface

**9)NavigableMap(I):**-It is the child interface of sortedMap defines several methods for navigation

TreeMap is the implemention class of NavigableMap

dgsp

Lgdgsp

**Collection Interface:-**It defines the common general methods which can be applient fror any collection object

1)bollean add(Object o)

2)bollean addAll(Collection o)

3)bollean remove(Object o)

4)bollean removeAll(Object o)

5)bollean retainAll(Collection o)

remove all objects except those present in c

6)void clear()

7)boolean contains(Object o)

8)boolean containsAll(Collection c)

9)boolean isEmpty()

10)int size()

11)Object[] toArray()

12)Iterator iterator()

**List Interface:-**

It is child interface of collection can be used to represent a group of indiviual objetcs where

1)insertion order is preserved

2)duplicate objects are allowed

we can differentiate duplicate objects and we can preserve insertion order by using indrex.

1)bollean add(Object o)

2)bollean add(int index,Object o)

3)bollean addAll(Collection c)

4)bollean addAll(int index,Collection c)

5)bollean remove(int index)

6)Object get(int index)

7)object set(int index,Object new)

8)int indexOf(Object o)

9)int lastIndexOf(Object o)

10)ListIterator ListIterator();

**ArrayList:-**

1)underlying dataStructure is resizable array or growable array

2)Duplicate objects are allowed

3)Insertion order is preserved

4)Heterogenious Objects are allowed

5)null insertion is also possible

**Constructors:-**

1)ArrayList l=new ArrayList()

Creates an empty ArrayList object with default initial capacity 10.once an ArrayList reaches it’s maximum capacity then a new ArrayList object will be created with

new Capacit=(currentCapacity\*3/2)+1

2)ArrayList l=new ArrayList(int initialCapacity)

Creates an empty ArrayList object with the specified initial capacity

3) ArrayList l=new ArrayList(Collection c)

this is for interconversion b/w collection objects.

import java.util.\*;

public classnDemo{

public static void main(String[] a) {

ArrayList l=new ArrayList();

l.add(“A”);

l.add(new Integer(10));

l.add(null);

l.add(“B”);

System.out.println(l);

l.remove(1);

l.set(2,Z);

System.out.println(l);

}

}

6) Vector and ArrayList class implements RandomAccess interface ,hence we can access any element with the same speed.Because of this ArrayList is best choice.if our frequent operation is retrival.

7)RandomAccess interface present in java.lang.package and it doesn’t contain any method it is a manner interface.

8)ArrayList is worst choice if our frequent operation is insertion or deletion in the middle.(because internally it requires several shift operations).

9)Usually th e collection objects can be used for dataStorage and transportation purposes.To support this requirement every collection implemented class is already implemented Serializable and Cloneable interfaces.

Ex:-consider the declarstions

ArrayList l1=new ArrayList();

LinkedList l2=new LinkedList();

which of the following returns true?

1)l1 instanceOf RandomAccess–>true

2)l2 instanceOf RandomAccess–>false

3)l1 instanceOf Serializable–>true

4)l2 instanceOf Cloneable–>true

**LinkedList:-**1) The Underlying data structure is double linkedList

2) Insertion Order preserved and duplicate objects are allowed

3)heteorogenious objects are allowed

4)null insertion is possible

5)Best choice if freguent operation insertion (or) decleration in the middle,worest choice if frequent operation is retrival

6)implements serializable and cloneable interfaces but not RandomAccess

7)LinkedList is mostly commonly used for implementing”Stacks&Queues”.

To suuport this requirement LinkedList class contains the following specific methods

1) void addFirst(object o)

2)void addLast(object o)

3)Object removeFirst()

4)Object removeLast()

5)Object getFirst()

6)Object removeLast()

**Construtors** :-1) LinkedList l=new LinkedList();

creates an empty LinkedList object

2)LinkedList l=new LinkedList(collection c)

interconversion between collection objects

ex:-

import java.utill.\*;

public class LinkedListDemo {

public void main(String a[]) {

LinkedList l=new LinkedList();

l.add(“durga”);

l.add(new Integer(10));

l.add(null);

l.add(“durga”);

l.set(0,”software”);

l.add(0,”venky”);

l.removeLast();

l.addFirst(“ccc”);——>[ccc,venky,software,1o,null]

System.out.println(l);

}

}

**vector :-**

1)The Underlying dataStructure is resizable array or growable array.

2)Duplicate objects are allowed,insertion order is preserved.

3)Heteogenious objects are allowed.

4)null insertion is possible

5)vector class implements Serializable,cloneable and RandomAccess,interfaces.

6)Vector is best suitable if our frequent operationmis retrival and worst choice .if freqent operation is insertion or deletion in the middle.

**Difference between vector and ArrayList**

**veactor ArrayList**

1) all methods are synchronized 1)no method is synchronized

2)vector object is thread safe 2)ArrayList objects is not threadsafe by default

3)performance is low 3) performance is high

4)introdused in 1.0v and it is legacy class 4)introdused in 1.2 and it is non legacy class

**Note:-**ArrayList object is by default non-synchronized but by using collections class method we can get synchronized version of ArrayList.

public static List sychronized(List l)

ex:-ArrayList l= new ArrayList();

List l1=Collections.syncronizedList(l);

Here li is synchronizedList and l is non-syncronizedList.

Similarly we can get synchronized versions of set and Map Object by using the following Collection class methods.

1)public static set synchronizedSet(Set s1)

2)Public static Map SynchronizedMap(Map m1)

**Important methods of vector:-**

1) For adding objects

add(Object 0)————>Collection

add(int index,Object obj)——->List

addElement(Object obj)———–>Vector

2)For removing objects

remove(Object obj)————>Collection

remove(int index)———>List

removeElement(Object obj)———>Vector

removeElementAt(int index)——–>Vector

removeAllElements()———>Vector

clear()———>Collection

3)for accesssing objects

get(int index)——–>List

elementAt(int index)——–>Vector

firstElement()

lastElement()

4) Other methods

int size()

int capacity()——–>vector specific

enumeration elements();

**Constructors:-**

1)Vector v =new Vector();

Creates an empty vector object with default initila capacity is 10.

when ever vector reaches it’s max capacity a new vector object will be created with double capacity

2)Vector v=new Vector(int initialCapacity)

3)Vector v new Vector(int initialCapacity,int incrementalCapacity)

4)Vector v =new Vector(collection c)

ex:-import java.util.\*;

public class VectorDemo{

public static void main(String a[]) {

Vector v=new Vector();

System.out.println(v.Capacity());———>**10**

for(int i=1;i<=10;i++)

{

v.addElement(i);

}

System.out.println(v.capacity());———->**10**

v.addElement(“A”);

System.out.println(v.capacity());———>**20**

System.out.println(v);——–>[1,2,3,...................10,A]

}

}

**Stack class:-** it is the child class of vector and contain only one Constructor.

Stack s=new Stack();

**Methods:-**

1)Object push(Object o)

for inserting an object to the stack

2)Object po()

removes and returns the top of stack

3)Object peek();

returns the top of the stack without removal.

4)boolean empty()

5)int search(Object o)

if the specified object is present .it returns it’s offset from the top of the stack other wise it returns -1

**ex:-**import java.util.\*;

public class StdckDemo {

public static void main(String a[]) {

Stack s= new Stack();

s.push(“A”);

s.push(“B”);

s.push(“c”);

System.out.println(s);———->**[A,B,C]**

System.out.println(s.search(“A”));———->**3**

System.out.println(s.search(“Z”)———> **-1**

}

}

**Leave a Reply** [**Cancel reply**](/?page_id=164)

Top of Form 2

Your email address will not be published. Required fields are marked \*

Name \*

Email \*

Website

Comment

You may use these HTML tags and attributes: <a href="" title=""> <abbr title=""> <acronym title=""> <b> <blockquote cite=""> <cite> <code> <del datetime=""> <em> <i> <q cite=""> <strike> <strong>



Bottom of Form 2

[Java2Learn](http://java2learn.com/) | Mantra Theme by [Cryout Creations](http://www.cryoutcreations.com) | Powered by [WordPress.](http://wordpress.org/)

[WordPress Automation Suite brought to you by Travis Weston](http://travisweston.com)