Collections in Java

Collection are the special types of classes in java .They are used to perform data structure operation in java.

**Collections in java** is a framework that provides an architecture to store and manipulate the group of objects.

All the operations that you perform on a data such as searching, sorting, insertion, manipulation, deletion etc. can be performed by Java Collections

#### What is framework in java

* provides readymade architecture.
* represents set of classes and interface.

Interface :-

List

Map

Set

Queue

Classes in collection framework

AbstractList

ArrayList

LinkedList

PriorityQueue

TressSet

HashSet

HashMap

Hashtable

TreeMap

--

Etc.

hierarchy of collection framework

# Generic VS Non Generic

# From JDK 1.5 collection Framework is generic. All collection classes present in java.util package.

# Generic :- It means type Safe

# Non Generic :- It means Type Unsafe.

# Example of Generic:-

ArrayList<String> list=**new** ArrayList<String>();

  list.add("Ravi");

# list.add("Vijay");

# list.add(10); // error

# Example of Non Generic:-

ArrayList list=**new** ArrayList();

  list.add("Ravi");

# list.add("Vijay");

# list.add(10);

# list.add(10.5);

# Java List Interface

List Interface is the subinterface of Collection.It contains methods to insert and delete elements in index basis.It is a factory of ListIterator interface.

List allow duplicate value.

Java ArrayList class

Java ArrayList class uses a dynamic array for storing the elements. It inherits AbstractList class and implements List interface.

The important points about Java ArrayList class are:

* Java ArrayList class can contain duplicate elements.
* Java ArrayList class maintains insertion order.
* Java ArrayList class is non synchronized.
* Java ArrayList allows random access because array works at the index basis.
* In Java ArrayList class, manipulation is slow because a lot of shifting needs to be occurred if any element is removed from the array list.

Method of ArrayList

add()

add(indexno,value)

get(indexno)

size()  
iterator()

forEach()

set()

import java.util.\*;

class T

{

public static void main(String k[])

{

ArrayList<Integer> obj = new ArrayList<Integer>();

obj.add(10);

obj.add(20);

obj.add(70);

obj.add(10);

obj.add(30);

obj.add(10);

obj.add(100);

obj.add(50);

for(Integer b:obj)

{

System.out.print(b+" ");

}

}

}

Innd way

---------

import java.util.\*;

class T

{

public static void main(String k[])

{

ArrayList<Integer> obj = new ArrayList<Integer>();

obj.add(10);

obj.add(20);

obj.add(70);

obj.add(10);

obj.add(30);

obj.add(10);

obj.add(100);

obj.add(50);

for(int i=0;i<obj.size();i++)

{

System.out.print(obj.get(i)+" ");

}

}

}

-------------

iird

------

import java.util.\*;

class T

{

public static void main(String k[])

{

ArrayList<Integer> obj = new ArrayList<Integer>();

obj.add(10);

obj.add(20);

obj.add(70);

obj.add(10);

obj.add(30);

obj.add(10);

obj.add(100);

obj.add(50);

Iterator it=obj.iterator();

while(it.hasNext())

{

System.out.print(it.next()+" ");

}

}

}

----

ivth

----

import java.util.\*;

class T

{

public static void main(String k[])

{

ArrayList<Integer> obj = new ArrayList<Integer>();

obj.add(10);

obj.add(20);

obj.add(70);

obj.add(10);

obj.add(30);

obj.add(10);

obj.add(100);

obj.add(50);

obj.forEach((y)->{System.out.print(y+" ");});

}

}

Vth

import java.util.\*;

class T

{

public static void main(String k[])

{

ArrayList<Integer> obj = new ArrayList<Integer>();

obj.add(10);

obj.add(20);

obj.add(70);

obj.add(10);

obj.add(30);

obj.add(10);

obj.add(0, 100);

obj.add(50);

obj.set(1,500);

obj.forEach((y)->{System.out.print(y+" ");});

}

}

Java LinkedList class

Java LinkedList class uses doubly linked list to store the elements. It provides a linked-list data structure. It inherits the AbstractList class and implements List and Deque interfaces.

The important points about Java LinkedList are:

* Java LinkedList class can contain duplicate elements.
* Java LinkedList class maintains insertion order.
* Java LinkedList class is non synchronized.
* In Java LinkedList class, manipulation is fast because no shifting needs to be occurred.
* Java LinkedList class can be used as list, stack or queue.

Method of LinkedList class

add(value)

add(indexno,value)

addFirst(value)

addLast()

removeFirst()

removeLast()

remove(value)

getFirst()

getLast()

**import** java.util.\*;

**public** **class** T{

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

  LinkedList<String> al=**new** LinkedList<String>();

  al.add("Ravi");

  al.add("Vijay");

  al.add("Ravi");

  al.add("Ajay");

  Iterator<String> itr=al.iterator();

**while**(itr.hasNext()){

   System.out.println(itr.next());

  }

 }

}

* Difference between ArrayList and LinkedList
* ArrayList and LinkedList both implements List interface and maintains insertion order. Both are non synchronized classes.
* But there are many differences between ArrayList and LinkedList classes that are given below.

|  |  |
| --- | --- |
| **ArrayList** | **LinkedList** |
| 1) ArrayList internally uses **dynamic array** to store the elements. | LinkedList internally uses **doubly linked list** to store the elements. |
| 2) Manipulation with ArrayList is **slow** because it internally uses array. If any element is removed from the array, all the bits are shifted in memory. | Manipulation with LinkedList is **faster** than ArrayList because it uses doubly linked list so no bit shifting is required in memory. |
| 3) ArrayList class can **act as a list** only because it implements List only. | LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces. |
| 4) ArrayList is **better for storing and accessing** data. | LinkedList is **better for manipulating** data. |

Set interface

Duplicate values are not allowed.

Java HashSet class

Java HashSet class is used to create a collection that uses a hash table for storage. It inherits the AbstractSet class and implements Set interface.

The important points about Java HashSet class are:

* HashSet stores the elements by using a mechanism called **hashing.**
* HashSet contains unique elements only.

## Difference between List and Set

List can contain duplicate elements whereas Set contains unique elements only.

**import** java.util.\*;

**class** T{

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

   HashSet<String> al=**new** HashSet<String>();

  al.add("Ravi");

  al.add("Vijay");

  al.add("Ravi");

  al.add("Ajay");

  Iterator<String> itr=al.iterator();

**while**(itr.hasNext()){

   System.out.println(itr.next());

  }

 }

}

To remove duplicate value of an array

import java.util.\*;

class T{

public static void main(String args[]){

int a[]={1,2,3,4,5,5,6,6,1,2,3,4,5,8,9};

HashSet<Integer> al=new HashSet<Integer>();

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

{

al.add(a[i]);

}

Iterator<Integer> itr=al.iterator();

while(itr.hasNext()){

System.out.println(itr.next());

}

}

}

Java TreeSet class

Java TreeSet class implements the Set interface that uses a tree for storage. It inherits AbstractSet class and implements NavigableSet interface. The objects of TreeSet class are stored in ascending order.

The important points about Java TreeSet class are:

* Contains unique elements only like HashSet.
* Access and retrieval times are quiet fast.
* Maintains ascending order

**import** java.util.\*;

**class** T{

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

  TreeSet<String> al=**new** TreeSet<String>();

  al.add("Ravi");

  al.add("Vijay");

  al.add("Ravi");

  al.add("Ajay");

  Iterator<String> itr=al.iterator();

**while**(itr.hasNext()){

   System.out.println(itr.next());

  }

 }

}

Stack class

It is used to implement LIFO(Last in First out type of list).

Method of Stack class

push():- To insert Elemnt

pop():- To delete Element

peek():- Return last (peek) element.

Example

import java.util.\*;

class T

{

public static void main(String k[])

{

Stack obj = new Stack();

obj.push(5);

obj.push(2);

obj.push(10);

System.out.println(obj);

obj.pop();

System.out.println(obj);

System.out.println(obj.peek());

}

}

Difference between ArrayList and Vector

ArrayList and Vector both implements List interface and maintains insertion order.

But there are many differences between ArrayList and Vector classes that are given below.

|  |  |
| --- | --- |
| **ArrayList** | **Vector** |
| 1) ArrayList is **not synchronized**. | Vector is **synchronized**. |
| 2) ArrayList **increments 50%** of current array size if number of element exceeds from its capacity. | Vector **increments 100%** means doubles the array size if total number of element exceeds than its capacity. |
| 3) ArrayList is **not a legacy** class, it is introduced in JDK 1.2. | Vector is a **legacy** class. |
| 4) ArrayList is **fast** because it is non-synchronized. | Vector is **slow** because it is synchronized i.e. in multithreading environment, it will hold the other threads in runnable or non-runnable state until current thread releases the lock of object. |
| 5) ArrayList uses **Iterator** interface to traverse the elements. | Vector uses **Enumeration** interface to traverse the elements. But it can use Iterator also. |

### Example of Java Vector

1. **import** java.util.\*;
2. **class** TestVector1{
3. **public** **static** **void** main(String args[]){
4. Vector<String> v=**new** Vector<String>();    v.addElement("umesh");
5. v.addElement("irfan");
6. v.addElement("kumar");
8. Enumeration e=v.elements();
9. **while**(e.hasMoreElements()){
10. System.out.println(e.nextElement());
11. }
12. }

}

Java Map Interface

A map contains values on the basis of key i.e. key and value pair. Each key and value pair is known as an entry. Map contains only unique keys.

Map is useful if you have to search, update or delete elements on the basis of key.

## Map.Entry Interface

Entry is the sub interface of Map. So we will be accessed it by Map.Entry name. It provides methods to get key and value.

### Methods of Map.Entry interface

|  |  |
| --- | --- |
| **Method** | **Description** |
| Object getKey() | It is used to obtain key. |
| Object getValue() | It is used to obtain value. |

Java HashMap class

Java HashMap class implements the map interface by using a hashtable. It inherits AbstractMap class and implements Map interface.

The important points about Java HashMap class are:

* A HashMap contains values based on the key.
* It contains only unique elements.
* It may have one null key and multiple null values.
* It maintains no order.

1. **import** java.util.\*;
2. **class** M{
3. **public** **static** **void** main(String args[]){
4. HashMap<Integer,String> map=**new** HashMap<Integer,String>();
5. map.put(100,"Amit");
6. map.put(101,"Vijay");
7. map.put(102,"Rahul");
8. **for**(Map.Entry m:map.entrySet()){
9. System.out.println(m.getKey()+" "+m.getValue());
10. }
11. }
12. }

Java TreeMap class

Java TreeMap class implements the Map interface by using a tree. It provides an efficient means of storing key/value pairs in sorted order.

The important points about Java TreeMap class are:

* A TreeMap contains values based on the key. It implements the NavigableMap interface and extends AbstractMap class.
* It contains only unique elements.
* It cannot have null key but can have multiple null values.
* It is same as HashMap instead maintains ascending order.

**import** java.util.\*;

* **class** T{
* **public** **static** **void** main(String args[]){
* TreeMap<Integer,String> hm=**new** TreeMap<Integer,String>();
* hm.put(100,"Amit");
* hm.put(102,"Ravi");
* hm.put(101,"Vijay");
* hm.put(103,"Rahul");
* **for**(Map.Entry m:hm.entrySet()){
* System.out.println(m.getKey()+" "+m.getValue());
* }
* }
* }

Java Hashtable class

Java Hashtable class implements a hashtable, which maps keys to values. It inherits Dictionary class and implements the Map interface.

The important points about Java Hashtable class are:

* A Hashtable is an array of list. Each list is known as a bucket. The position of bucket is identified by calling the hashcode() method. A Hashtable contains values based on the key.
* It contains only unique elements.
* It may have not have any null key or value.
* It is synchronized.
* **import** java.util.\*;
* **public** **class** H {
* **public** **static** **void** main(String args[]) {
* Hashtable<Integer, String> map = **new** Hashtable<Integer, String>();
* map.put(102,"Let us C");
* map.put(103, "Operating System");
* map.put(101, "Data Communication and Networking");
* System.out.println("Values before remove: "+ map);
* map.remove(102);
* System.out.println("Values after remove: "+ map);
* }
* }

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**next →**](https://www.javatpoint.com/java-enumset)[**← prev**](https://www.javatpoint.com/java-hashtable)  Difference between HashMap and Hashtable  HashMap and Hashtable both are used to store data in key and value form. Both are using hashing technique to store unique keys.  But there are many differences between HashMap and Hashtable classes that are given below.   |  |  | | --- | --- | | **HashMap** | **Hashtable** | | 1) HashMap is **non synchronized**. It is not-thread safe and can't be shared between many threads without proper synchronization code. | Hashtable is **synchronized**. It is thread-safe and can be shared with many threads. | | 2) HashMap **allows one null key and multiple null values**. | Hashtable **doesn't allow any null key or value**. | | 3) HashMap is a **new class introduced in JDK 1.2**. | Hashtable is a **legacy class**. | | 4) HashMap is **fast**. | Hashtable is **slow**. | | 5) HashMap is **traversed by Iterator**. | Hashtable is **traversed by Enumerator and Iterator**. | |  |  | | 6) HashMap inherits **AbstractMap** class. | Hashtable inherits **Dictionary** class. | |

# Java Queue Interface

Java Queue interface orders the element in FIFO(First In First Out) manner. In FIFO, first element is removed first and last element is removed at last.

## PriorityQueue class

The PriorityQueue class provides the facility of using queue. But it does not orders the elements in FIFO manner. It inherits AbstractQueue class.

1. **import** java.util.\*;
2. **class** T{
3. **public** **static** **void** main(String args[]){
4. PriorityQueue<String> queue=**new** PriorityQueue<String>();
5. queue.add("Amit");
6. queue.add("Vijay");
7. queue.add("Karan");
8. queue.add("Jai");
9. queue.add("Rahul");
10. System.out.println("head:"+queue.element());
11. System.out.println("head:"+queue.peek());
12. System.out.println("iterating the queue elements:");
13. Iterator itr=queue.iterator();
14. **while**(itr.hasNext()){
15. System.out.println(itr.next());
16. }
17. queue.remove();
18. queue.poll();
19. System.out.println("after removing two elements:");
20. Iterator<String> itr2=queue.iterator();
21. **while**(itr2.hasNext()){
22. System.out.println(itr2.next());
23. }
24. }
25. }