**collection framework**

collection is used to store group of object in a single unit.

What is the difference between Array and collection ?

Array is a a fixed size.

Collection is a growable and writeable in nature.

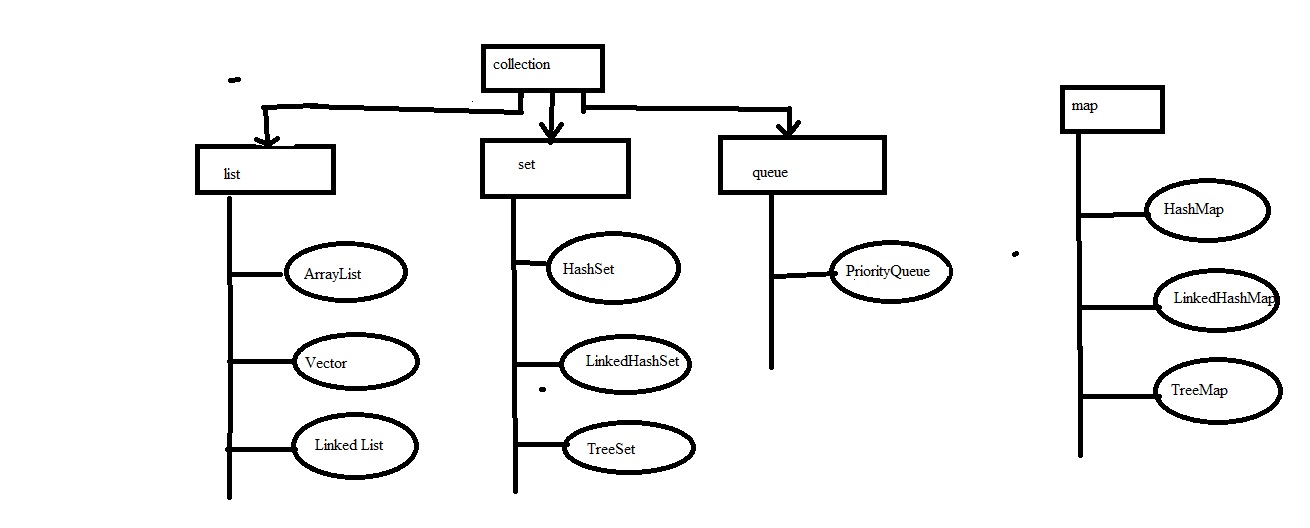
collection - interface

Collections - class

A **Collection** in Java is a group or container that holds multiple objects. It provides a way to store, organize, and access these objects in different ways. Collections in Java can be of various types, such as:

* **List**: An ordered collection (like an array) where elements can be accessed by their index. Allow duplicates.
* **Set**: A collection that doesn't allow duplicate elements.
* **Queue**: A collection used to store elements in a first-in-first-out (FIFO) manner.
* **Map**: A collection that stores data in key-value pairs.

Java provides classes and interfaces under the java.util package for these collections.



* **List**: An ordered collection (like an array) where elements can be accessed by their index. Allow duplicates.

**ArrayList** :

It is a growable array. It gives you fast iteration and fast random access.

It is an ordered collection but not sorted.

It gives you fast iteration but cannot do lot of insertion and deletion.

**Vector** :

Vector is same as ArrayList, but vector methods are synchronized for thread safety.

**LinkedList**

A LinkedList is ordered by index position like ArrayList, except that the element are doubly-linked to one another. The element or object can be add in front and also back.

LinkedList may iterate more slowely than an ArrayList , but it is a good choice when you need fast insertion and deletion.

**Set :**

A collection that doesn't allow duplicate elements.

**HashSet**

A HashSet is an unsorted , unordered Set.

It doesn’t allow duplicates

Use this class when you don’t care about the iteration order.

**LinkedHashSet**

A LinkedHashSet is an ordered version of HashSet that maintains a doubly-linked list

across all the elements.

Use this class instead of HashSet when you care about the iteration order.

**TreeSet**

The TreeSet is guarantee that the elements will be in ascending order.

**Queue :**

A collection used to store elements in a first-in-first-out (FIFO) manner.

**PriorityQueue**

The purpose of PriorityQueue is to create a “priority-in, priority out” queue as opposed to a typical FIFO queue.

**Map** :

* A collection that stores data in key-value pairs. You map a unique key to a specific value.

**HashMap**

It gives you an unsorted, unordered Map.

You can use this class if you don’t care about the order.

HashMap allows one null key and multiple null values in a collection.

**HashTable**

It is a synchronized counterpart to HashMap. The key methods of the class are synchronized.

Hashtable doesn’t let you have anything that’s null.

**LinkedHashMap**

LinkedHashMap collection maintains insertion order.

Although it will be somewhat slower than HashMap for adding and removing elements, you can expect faster iteration with a LinkedHashMap.

**TreeMap**

is a sorted map.

**Generic concept is used to provide type safetly.**

**Mostly Generic concept affect in the collection framework only.**

**Generic concept allows only Wrapper class.**

**ArrayList**

1)

**import** java.util.ArrayList;

**public** **class** ArrayListDemo {

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

ArrayList<Integer> al=**new** ArrayList<Integer>();

al.add(25);

al.add(22);

al.add(11);

al.add(10);

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

}

}

2)

**import** java.util.ArrayList;

**public** **class** ArrayListDemo1 {

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

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

System.***out***.println("Initial size : "+al.size());

al.add("C");

al.add("A");

al.add("B");

al.add(1,"D");

System.***out***.println("size of al after additions: "+al.size());

System.***out***.println("contents of al : "+al);

al.remove("B");

al.remove(2);

System.***out***.println("size of al after deletions : "+al.size());

System.***out***.println("contents of al: "+al);

}

}

3)

**import** java.util.ArrayList;

**import** java.util.Iterator;

**public** **class** ArrayListDemo3 {

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

// **TODO** Auto-generated method stub

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

al.add("Ram");

al.add("vasu");

al.add("param");

al.add("balaji");

al.add("venkat");

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

Iterator<String> ii=al.iterator();

**while**(ii.hasNext()) {

System.***out***.println(ii.next());

}

}

}

Vector Example program

**import** java.util.Enumeration;

**import** java.util.Vector;

**public** **class** VectorDemo {

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

// **TODO** Auto-generated method stub

Vector<Integer> v=**new** Vector<Integer>(3,2);

System.***out***.println("Initial Size: "+v.size()+" and capacity: "+v.capacity());

v.addElement(1);

v.addElement(2);

v.addElement(3);

v.addElement(4);

System.***out***.println("capacity after four addition: "+v.capacity());

v.addElement(5);

System.***out***.println("current capacity : "+v.capacity());

v.addElement(6);

System.***out***.println("first element : "+v.firstElement());

System.***out***.println("last element : "+v.lastElement());

**if**(v.contains(5))

System.***out***.println("vector contains 5");

Enumeration vEnum=v.elements();

System.***out***.println("\n Elements in vector :");

**while**(vEnum.hasMoreElements())

System.***out***.print(vEnum.nextElement()+" ");

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

}

}

LinkedList Program

**import** java.util.LinkedList;

**public** **class** LinkedListDemo {

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

// **TODO** Auto-generated method stub

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

ll.add("C");

ll.add("D");

ll.add("B");

ll.addLast("E");

ll.addFirst("A");

ll.add(1,"F");

System.***out***.println("Original contents of ll : "+ll);

ll.remove("F");

ll.remove(2);

System.***out***.println("contents of ll after deletion : "+ll);

ll.removeFirst();

ll.removeLast();

System.***out***.println("ll after deleting first and last : "+ll);

String val=ll.get(1);

ll.set(1, val+ "Changed");

System.***out***.println("ll after change : "+ll);

}

}

Set Example :

**import** java.util.Set;

**import** java.util.TreeSet;

**public** **class** SetDemo {

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

// **TODO** Auto-generated method stub

// Set<String> hs=new HashSet<String>();

// Set<String> hs=new LinkedHashSet<String>();

Set<String> hs=**new** TreeSet<String>();

hs.add("C");

hs.add("D");

hs.add("A");

hs.add("B");

hs.add("D");

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

}

}

Map Interface Example :

**import** java.util.HashMap;

**import** java.util.LinkedHashMap;

**import** java.util.Map;

**import** java.util.Set;

**import** java.util.TreeMap;

**public** **class** HashMapDemo {

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

// **TODO** Auto-generated method stub

// Map<String,Double> hm=new HashMap<String,Double>();

// Map<String,Double> hm=new LinkedHashMap<String,Double>();

Map<String,Double> hm=**new** TreeMap<String,Double>();

hm.put("Raj",1000.50);

hm.put("John",2000.25);

hm.put("Jabeer", 3000.75);

hm.put("Robert", 4000.50);

Set<Map.Entry<String,Double>> st=hm.entrySet();

**for**(Map.Entry<String,Double> me : st) {

System.***out***.print(me.getKey()+" : ");

System.***out***.println(me.getValue());

}

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

**double** balance = hm.get("Raj");

System.***out***.println("Raj balance : "+balance);

hm.put("Raj", balance + 1000);

System.***out***.println("Raj new balance : "+hm.get("Raj"));

}

}