**Collections in java**

i)***Array list-*** It uses a dynamic array to store the duplicate element of different data types. The ArrayList class maintains the insertion order and is non-synchronized. The elements stored in the ArrayList class can be randomly accessed.

**package** com;

**import** java.util.Iterator;

**public** **class** ArrayList {

**private** **static** **final** String ***String*** = **null**;

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

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

java.util.ArrayList<String>list= **new** java.util.ArrayList<String>();

list.add("keerthy");

list.add("kavya");

list.add("heba");

list.add("neethu");

list.add("santhosh");

list.add("mary");

list.add("tom");

list.add("jerry");

Iterator<java.lang.String> itr = list.iterator();

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

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

}

}

ii)*Linked list-* LinkedList implements the Collection interface. It uses a doubly linked list internally to store the elements. It can store the duplicate elements.

**package** com;

**import** java.util.Iterator;

**public** **class** LinkedList {

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

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

java.util.LinkedList<String>Ll= **new** java.util.LinkedList<String>();

Ll.add("hello");

Ll.add("hi");

Ll.add("bye");

Ll.add("okay");

Iterator<java.lang.String> itr=Ll.iterator();

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

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

}

}

}

iii) *Vector-* Vector uses a dynamic array to store the data elements. It is synchronized and contains many methods that are not the part of Collection framework.

**package** com;

**import** java.util.Iterator;

**public** **class** Vector {

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

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

java.util.Vector<String>v= **new** java.util.Vector<String>();

v.add("keerthy");

v.add("1");

v.add("2");

v.add("keerthy");

Iterator<java.lang.String> itr=v.iterator();

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

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

}

}

}

iv) ***Stack-***  The stack is the subclass of Vector. It implements the last-in-first-out data structure, i.e., Stack. The stack contains all of the methods of Vector class

**package** com;

**import** java.util.Iterator;

**public** **class** Stack {

**private** **static** **final** String ***String*** = **null**;

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

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

java.util.Stack<String>stack= **new** java.util.Stack<String>();

stack.push("Keerthy");

stack.push("is");

stack.push("a good");

stack.push("girl");

//stack.pop();

Iterator<java.lang.String> itr=stack.iterator();

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

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

}

}

}

v) ***Priority queue-***  The Priority Queue class implements the Queue interface. It holds the elements or objects which are to be processed by their priorities. Priority Queue doesn't allow null values to be stored in the queue.

**package** com;

**import** java.util.Iterator;

**import** java.util.PriorityQueue;

**public** **class** PriorityQue {

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

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

PriorityQueue<Object>q= **new** PriorityQueue<Object>();

q.add(10);

q.add(20);

q.add(30);

q.add(40);

Iterator<Object>itr=q.iterator();

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

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

}

}

}

vi) ***Array deque-***  ArrayDeque class implements the Deque interface. It facilitates us to use the Deque. Unlike queue, we can add or delete the elements from both the ends. ArrayDeque is faster than ArrayList and Stack and has no capacity restrictions.

**package** com;

**import** java.util.Iterator;

**public** **class** ArrayDeque {

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

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

java.util.ArrayDeque<String>ad= **new** java.util.ArrayDeque<String>();

ad.add("my name");

ad.add("is");

ad.add("keerthy");

Iterator<java.lang.String> itr=ad.iterator();

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

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

}

}

}

vii)***Tree set-*** TreeSet class implements the Set interface that uses a tree for storage. Like HashSet, TreeSet also contains unique elements. The elements in TreeSet stored in ascending order.

**package** com;

**import** java.util.Iterator;

**public** **class** TreeSet {

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

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

java.util.TreeSet<String>t1= **new** java.util.TreeSet<String>();

t1.add("I am") ;

t1.add("Keerthy") ;

t1.add("S") ;

Iterator<java.lang.String> itr=t1.iterator();

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

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

}

}

}

viii) ***Hashset-*** HashSet class implements Set Interface. It represents the collection that uses a hash table for storage. Hashing is used to store the elements in the HashSet. It contains unique items.

**package** com;

**import** java.util.Iterator;

**public** **class** HashSet {

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

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

java.util.HashSet<String>set= **new** java.util.HashSet();

set.add("One");

set.add("Two");

set.add("Three");

set.add("Four");

set.add("Five");

Iterator<String> i=set.iterator();

**while**(i.hasNext())

{

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

}

}

}

ix) ***Linked hash set-***  represents the LinkedList implementation of Set Interface. It extends the HashSet class and implements Set interface. Like HashSet, It also contains unique elements. It maintains the insertion order and permits null elements.

**ackage** com;

**import** java.util.Iterator;

**public** **class** LinkedHashset {

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

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

java.util.LinkedHashSet<String>lh= **new** java.util.LinkedHashSet<String>();

lh.add("hi");

lh.add("hello");

lh.add("how are you");

lh.add("thankyou");

Iterator<java.lang.String> itr=lh.iterator();

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

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

}

}

}