**Java Collection:**

It provides an architecture to store and manipulate the group of objects

List (ArrayList, LinkedList, vector)

Queue (Deque, ArrayDeque)

Set (HashSet, LinkedHashSet , SortedSet…TreeSet)

Map (Hashtable, LinkedHashMap, HashMap, TreeMap)

ArrayList – uses a dynamic array for storing the elements. It has no size limit. Can add or remove elements. Can have duplicate elements .It implements List Interface so we can use all methods of the List interface.

Java collection framework was non-generic before JDK 1.5 since 1.5 it is generic

Old non-generic syntax

ArrayList list =new ArrayList()

Generic syntax

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

Example:

**package** com.lokesh;

**import** java.util.\*;

**public** **class** HelloWorld2 {

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

ArrayList<String> list1=**new** ArrayList<String>();

list1.add("ramu");

list1.add("raju");

list1.add("ravi");

System.***out***.println(list1);

//using for each loop

**for**(String list2:list1) {

System.***out***.println(list2);

}

//using iterator

Iterator itr=list1.iterator(); //getting the iterator

**while**(itr.hasNext()) { //checking it iterator has elements

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

}

ArrayList<Integer> list3 =**new** ArrayList<Integer>();

list3.add(22);

list3.add(24);

list3.add(29);

System.***out***.println(list3);

}

}

get and set in ArrayList – get() method returns the element while set() method sets the element

Collection.sort(list..name) sorts the list

Example:

**package** com.lokesh;

**import** java.util.ArrayList;

**import** java.util.Collections;

**import** java.util.Iterator;

**public** **class** HelloWorld2 {

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

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

list.add("ramu");

list.add("raju");

list.add("ravi");

System.***out***.println(list.get(2)); //returns raju

list.set(1, "raju2"); //changes raju to raju2

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

//sorts the list

Collections.*sort*(list);

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

}

}

Ways to iterate the elements of the collection in Java

By Iterator interface

By for-each loop

By ListIterator interface

By for loop

By for-each method

By forEachremaining() method

Example:

**package** com.lokesh;

**import** java.util.ArrayList;

**import** java.util.Collections;

**import** java.util.Iterator;

**import** java.util.ListIterator;

**public** **class** HelloWorld2 {

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

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

list.add("ramu");

list.add("raju");

list.add("ravi");

list.add("ajay");

//Iterator interface

Iterator itr=list.iterator();

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

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

}

//for-each loop

**for**(String list2:list) {

System.***out***.println(list2);

}

//ListIterator interface

ListIterator<String> list3=list.listIterator();

**while**(list3.hasNext()){

String str =list3.next();

System.***out***.println(list3);

}

//for loop

**for**(**int** i = 0;i<list.size();i++) {

System.***out***.println(list.get(i));

}

//for-each method

list.forEach(a->{

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

});

//for-each remaining

Iterator<String> itr2=list.iterator();

itr2.forEachRemaining(b->{

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

});

}

}

User-defined class objects in ArrayList

Example:

**package** com.lokesh;

**public** **class** HelloWorld1 {

**int** empid;

String empname;

**int** empage;

HelloWorld1(**int** empid,String empname,**int** empage) {

**this**.empid=empid;

**this**.empname=empname;

**this**.empage=empage;

}

}

**package** com.lokesh;

**import** java.util.ArrayList;

**import** java.util.Collections;

**import** java.util.Iterator;

**import** java.util.ListIterator;

**public** **class** HelloWorld2 {

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

HelloWorld1 hw1=**new** HelloWorld1(1, "ramu", 29);

HelloWorld1 hw2=**new** HelloWorld1(1, "raju", 25);

HelloWorld1 hw3=**new** HelloWorld1(1, "ravi", 33);

ArrayList<HelloWorld1> list=**new** ArrayList<HelloWorld1>();

list.add(hw1);

list.add(hw2);

list.add(hw3);

Iterator itr=list.iterator();

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

HelloWorld1 hw4=(HelloWorld1)itr.next();

System.***out***.println(hw4.empid+" "+hw4.empname+" "+hw4.empage);

}

}

}

Example: Below is an example to remove elements ad to check if the list is empty

**package** com.lokesh;

**import** java.util.\*;

**public** **class** HelloWorld2 {

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

ArrayList<String> list1=**new** ArrayList<String>();

list1.add("ramu");

list1.add("raju");

list1.add("ravi");

System.***out***.println(list1);

list1.remove(1); //removes raju

System.***out***.println(list1);

ArrayList<Integer> list2 =**new** ArrayList<Integer>();

list2.add(22);

list2.add(24);

list2.add(29);

System.***out***.println(list2);

list2.remove(2); //removes 29

System.***out***.println(list2);

}

}

LinkedList – uses a double linked list(i.e. we can add or remove from both sides) to store the elements. Can contain duplicate elements. Maintains insertion order. Not synchronized, manipulation is fast because no shifting needs to occur, used as List, stack and queue

Example:

**package** com.lokesh;

**import** java.util.\*;

**public** **class** HelloWorld2 {

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

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

list.add("ramu");

list.add("raju");

list.add("ravi");

list.add("ajay");

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

//adding values at first and last

list.addFirst("lion");

list.addLast("tiger");

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

//removing values from first and last

list.removeFirst();

list.removeLast();

list.remove(2);

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

LinkedList<Integer> list2=**new** LinkedList<Integer>();

list2.add(23);

list2.add(27);

list2.add(35);

System.***out***.println(list2);

list2.remove(0); //removes 23

System.***out***.println(list2);

}

}

Example : Using user defined class objects

**package** com.lokesh;

**public** **class** HelloWorld1 {

**int** empid;

String empname;

**int** empage;

HelloWorld1(**int** empid,String empname,**int** empage) {

**this**.empid=empid;

**this**.empname=empname;

**this**.empage=empage;

}

}

**package** com.lokesh;

**import** java.util.\*;

**public** **class** HelloWorld2 {

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

HelloWorld1 hw1 =**new** HelloWorld1(1, "ramu", 22);

HelloWorld1 hw2 =**new** HelloWorld1(2, "raju", 26);

HelloWorld1 hw3 =**new** HelloWorld1(3, "ravi", 33);

LinkedList<HelloWorld1> list=**new** LinkedList<HelloWorld1>();

list.add(hw1);

list.addFirst(hw3);

list.add(hw2);

**for**(HelloWorld1 hw4:list) {

System.***out***.println(hw4.empid+" "+hw4.empname+" "+hw4.empage);

}

}

}

List – provides the facility to maintain the ordered collection. It contains index-based methods to insert, update, delete, and search the elements. Can have duplicates, can store null elements in list

List is same as ArrayList just syntax changes remaining same

Example

**package** com.lokesh;

**package** com.lokesh;

**import** java.util.\*;

**public** **class** HelloWorld2 {

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

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

list.add("ramu");

list.add("raju");

list.add(**null**);

list.add("ravi");

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

//seeting a value

list.set(2, "lion"); //changes null to lion

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

//getting values

System.***out***.println(list.get(3)); //gets ravi

//sorting

Collections.*sort*(list);

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

//converting array to List

String s1[] = {"elephant","tiger","bear"};

List<String> list2 =**new** ArrayList<String>();

**for**(String s2:s1) {

list2.add(s2);

}

System.***out***.println(list2);

List<Integer> list3=**new** ArrayList<Integer>();

list3.add(43);

list3.add(23);

list3.add(18);

list3.add(56);

System.***out***.println(list3);

//setting values

list3.set(1, 28); // changes 23 to 28

//getting values

System.***out***.println(list.get(0)); //returns 43

System.***out***.println(list3);

//sorting

Collections.*sort*(list3);

System.***out***.println(list3);

}

}

Example: using user defined objects

**package** com.lokesh;

**public** **class** HelloWorld1 {

**int** empid;

String empname;

**int** empage;

HelloWorld1(**int** empid,String empname,**int** empage) {

**this**.empid=empid;

**this**.empname=empname;

**this**.empage=empage;

}

}

**package** com.lokesh;

**import** java.util.\*;

**public** **class** HelloWorld2 {

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

HelloWorld1 hw1 =**new** HelloWorld1(1, "ramu", 22);

HelloWorld1 hw2 =**new** HelloWorld1(2, "raju", 26);

HelloWorld1 hw3 =**new** HelloWorld1(3, "ravi", 33);

List<HelloWorld1> list=**new** LinkedList<HelloWorld1>();

list.add(hw1);

list.add(hw3);

list.add(hw2);

**for**(HelloWorld1 hw4:list) {

System.***out***.println(hw4.empid+" "+hw4.empname+" "+hw4.empage);

}

}

}

HashSet – stores the elements by using a mechanism called hashing, contains unique elements only, allows null values, is not synchronized, doesn’t maintain the insertion order, elements are inserted on basis of their hashcode, best approach for search operations

Difference between List and Set is List contains duplicate elements whereas Set contains unique elements only.

Example:

**package** com.lokesh;

**import** java.util.\*;

**public** **class** HelloWorld2 {

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

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

set.add("ramu");

set.add("raju");

set.add("ramu"); //doesn't get stored since duplicate

set.add("ravi");

Iterator itr=set.iterator();

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

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

}

set.remove("ramu");

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

HashSet<Integer> set2=**new** HashSet<Integer>();

set2.add(63);

set2.add(87);

set2.add(28);

set2.add(63); // doesn't get stored since duplicate

System.***out***.println(set2);

set2.remove(28);

System.***out***.println(set2);

}

}

Example: Below is an example of HashSet from another collection

**package** com.lokesh;

**import** java.util.\*;

**public** **class** HelloWorld2 {

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

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

list.add("ramu");

list.add("raju");

HashSet<String> set=**new** HashSet<String>(list);

set.add("ravi");

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

}

}

Example: Below is an example for user-defined objects

**package** com.lokesh;

**public** **class** HelloWorld1 {

**int** empid;

String empname;

**int** empage;

HelloWorld1(**int** empid,String empname,**int** empage) {

**this**.empid=empid;

**this**.empname=empname;

**this**.empage=empage;

}

}

**package** com.lokesh;

**import** java.util.\*;

**public** **class** HelloWorld2 {

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

HelloWorld1 hw1 =**new** HelloWorld1(1, "ramu", 22);

HelloWorld1 hw2 =**new** HelloWorld1(2, "raju", 27);

HelloWorld1 hw3 =**new** HelloWorld1(3, "ravi", 33);

HelloWorld1 hw4 =**new** HelloWorld1(1, "ravi", 33);

HashSet<HelloWorld1> set=**new** HashSet<HelloWorld1>();

set.add(hw2);

set.add(hw3);

set.add(hw1);

**for**(HelloWorld1 hw5:set) {

System.***out***.println(hw5.empage+" "+hw5.empname+" "+hw5.empname);

}

}

}

LinkedHashSet – contains unique elements only like HashSet, provides all optional set operations and permits null elements, is non-synchronized, maintain insertion order

Example:

**package** com.lokesh;

**import** java.util.\*;

**public** **class** HelloWorld2 {

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

LinkedHashSet<String> lset =**new** LinkedHashSet<String>();

lset.add("ramu");

lset.add("raju");

lset.add("ravi");

Iterator itr=lset.iterator();

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

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

}

lset.remove("raju");

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

LinkedHashSet<Integer> lset2 =**new** LinkedHashSet<Integer>();

lset2.add(56);

lset2.add(67);

lset2.add(43);

System.***out***.println(lset2);

lset2.remove(67);

System.***out***.println(lset2);

}

}

Example: Below is an example for user-defined objects

**package** com.lokesh;

**public** **class** HelloWorld1 {

**int** empid;

String empname;

**int** empage;

HelloWorld1(**int** empid,String empname,**int** empage) {

**this**.empid=empid;

**this**.empname=empname;

**this**.empage=empage;

}

}

**package** com.lokesh;

**import** java.util.\*;

**public** **class** HelloWorld2 {

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

HelloWorld1 hw1 =**new** HelloWorld1(1, "ramu", 22);

HelloWorld1 hw2 =**new** HelloWorld1(2, "raju", 27);

HelloWorld1 hw3 =**new** HelloWorld1(3, "ravi", 33);

LinkedHashSet<HelloWorld1> set=**new** LinkedHashSet<HelloWorld1>();

set.add(hw2);

set.add(hw1);

set.add(hw3);

**for**(HelloWorld1 hw4:set) {

System.***out***.println(hw4.empid+" "+hw4.empname+" "+hw4.empage);

}

}

}

Treeset – implements the Set interface that uses tree as storage. Contains only unique elements, access and retrieval times are fast, doesn’t allow null elements, is non-synchronized, maintain ascending order,

Example:

**package** com.lokesh;

**import** java.util.\*;

**public** **class** HelloWorld2 {

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

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

tset.add("ramu");

tset.add("raju");

tset.add("ravi");

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

tset.remove("raju");

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

TreeSet<Integer> tset2 =**new** TreeSet<Integer>();

tset2.add(56);

tset2.add(23);

tset2.add(47);

tset2.add(82);

System.***out***.println(tset2);

Iterator<Integer> itr =tset2.iterator();

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

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

}

//get highest value

System.***out***.println(tset2.pollLast());

//get lowest value

System.***out***.println(tset2.pollFirst());

}

}

User defined objects will same like other as above

Queue and PriorityQueue –

Java Queue is used to keep the elements in First in first out (FIFO) manner, provides support for all the methods of Collection interface including insertion, deletion…etc . follows ascending order

Priority Queue is same like Java queue like FIFO and insertion, deletion but sometimes elements of the queue are needed to be processed according to the priority , that’s where PriorityQueue comes into action

Example:

**package** com.lokesh;

**import** java.util.\*;

**public** **class** HelloWorld2 {

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

PriorityQueue<String> pq1 =**new** PriorityQueue<String>();

pq1.add("ramu");

pq1.add("raju");

pq1.add("ravi");

System.***out***.println(pq1);

//removing elements

pq1.remove(); //removes raju because in ascending order raju is first

System.***out***.println(pq1);

pq1.poll(); //removes ravi since it is first now

System.***out***.println(pq1);

PriorityQueue<Integer> pq2 =**new** PriorityQueue<Integer>();

pq2.add(47);

pq2.add(23);

pq2.add(36);

pq2.add(84);

System.***out***.println(pq2);

Iterator<Integer> itr=pq2.iterator();

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

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

}

//removing elements

pq2.remove(); //removes 23 because in ascending order 23 is forst

System.***out***.println(pq2);

pq2.poll(); //removes 36 since it is forst now

System.***out***.println(pq2);

}

}

Example: Below is an example for user defined objects. The object class should implement Comparable

**package** com.lokesh;

**public** **class** HelloWorld1 **implements** Comparable<HelloWorld1> {

**int** empid;

String empname;

**int** empage;

HelloWorld1(**int** empid,String empname,**int** empage) {

**this**.empid=empid;

**this**.empname=empname;

**this**.empage=empage;

}

@Override

**public** **int** compareTo(HelloWorld1 o) {

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

**return** 0;

}

}

**package** com.lokesh;

**import** java.util.\*;

**public** **class** HelloWorld2 {

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

HelloWorld1 hw1 =**new** HelloWorld1(1, "ramu", 23);

HelloWorld1 hw2 =**new** HelloWorld1(3, "raju", 43);

HelloWorld1 hw3 =**new** HelloWorld1(2, "ravi", 33);

PriorityQueue<HelloWorld1> pq1 =**new** PriorityQueue<HelloWorld1>();

pq1.add(hw1);

pq1.add(hw2);

pq1.add(hw3);

**for**(HelloWorld1 hw4:pq1) {

System.***out***.println(hw4.empid+" "+hw4.empname+" "+hw4.empage);

}

pq1.remove(); //removes id - 1 "ramu" 23 since it will be first

**for**(HelloWorld1 hw4:pq1) {

System.***out***.println(hw4.empid+" "+hw4.empname+" "+hw4.empage);

}

}

}

Deque and ArrayDeque –

Java Deque supports the addition and removal of elements from both ends. Therefore Deque can be used as stack or queue. Stack supports LIFO operation and queue supports FIFO operation. Deque supports both

ArrayDeque – we know it is not possible to create an interface in java so we need a class that implements the Deque interface, and that class is ArrayDeque.

ArrayDeque is faster than LinkedList and Stack

Example:

**package** com.lokesh;

**import** java.util.\*;

**public** **class** HelloWorld2 {

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

ArrayDeque<String> ad1 =**new** ArrayDeque<String>();

ad1.add("ramu");

ad1.add("raju");

ad1.add("ravi");

System.***out***.println(ad1);

//adding values

ad1.offer("rahul"); //adds last

System.***out***.println(ad1);

ad1.offerFirst("kranthi"); //adds first

ad1.offerLast("pradip"); //adds last

System.***out***.println(ad1);

//removing values

ad1.remove();

System.***out***.println(ad1);

ad1.poll(); //removes first

ad1.pollFirst(); //removes first poll and pollFirst are same

ad1.pollLast(); //removes last element

System.***out***.println(ad1);

**for**(String s1 :ad1) {

System.***out***.println(s1);

}

}

}

Example: Below is an example for user defined objects

**package** com.lokesh;

**public** **class** HelloWorld1 {

**int** empid;

String empname;

**int** empage;

HelloWorld1(**int** empid,String empname,**int** empage) {

**this**.empid=empid;

**this**.empname=empname;

**this**.empage=empage;

}

}

**package** com.lokesh;

**import** java.util.\*;

**public** **class** HelloWorld2 {

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

HelloWorld1 hw1 =**new** HelloWorld1(1, "ramu", 23);

HelloWorld1 hw2 =**new** HelloWorld1(2, "raju", 33);

HelloWorld1 hw3 =**new** HelloWorld1(3, "ravi", 43);

ArrayDeque<HelloWorld1> ad1 =**new** ArrayDeque<HelloWorld1>();

ad1.add(hw2);

ad1.add(hw3);

ad1.add(hw1);

**for**(HelloWorld1 hw4:ad1) {

System.***out***.println(hw4.empid+" "+hw4.empname+" "+hw4.empage);

}

ad1.pollLast(); //removes ---1 "raju" 23 since in above it is added last and is last in queue

**for**(HelloWorld1 hw4:ad1) {

System.***out***.println(hw4.empid+" "+hw4.empname+" "+hw4.empage);

}

}

}

Map Interface – map contains values on basis of key and value pair. Each key and value pair is known as entry. Map contains unique keys

There are two interfaces for implementing Map : Map and SortedMap and three classes: HashMap, LinkedHashMap and TreeMap

Map doen’t allow duplicate values, but you can have duplicate values. HashMap and LinkedHashMap allows null keys and values but TreeMap doesn’t allow any null key or value

Example: Below is an example of old way non-generic

**package** com.lokesh;

**import** java.util.\*;

**import** java.util.Map.Entry;

**public** **class** HelloWorld2 {

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

Map m1 =**new** HashMap();

m1.put(1, "ramu");

m1.put(3, "ravi");

m1.put(2, "raju");

System.***out***.println(m1);

//Traversing Map

Set s1 =m1.entrySet(); //converting to Set so that we can traverse

Iterator itr =s1.iterator();

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

//converting to Map.Entry so that we can get key value separately

Map.Entry<Integer, String> m2=(Entry) itr.next();

System.***out***.println(m2.getKey()+" "+m2.getValue());

}

}

}

Example: Below is an example of new way i.e Generic

**package** com.lokesh;

**import** java.util.\*;

**import** java.util.Map.Entry;

**public** **class** HelloWorld2 {

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

Map<Integer, String> m1 =**new** HashMap<Integer, String>();

m1.put(1, "ramu");

m1.put(3, "ravi");

m1.put(2, "raju");

System.***out***.println(m1);

**for**(Map.Entry<Integer, String> m2 : m1.entrySet()) {

System.***out***.println(m2.getKey()+" "+m2.getValue());

}

}

}

HashMap – allows us to store key and value pair where keys should be unique, if you try to insert duplicate key it will replace the element of the corresponding key, it is not synchronized, allows us to store null elements as well but there should be only one null key , can have multiple null values

Difference between HashSet and HashMap is HashSet contains only values while HasMap contains keys and values

Example:

**package** com.lokesh;

**import** java.util.\*;

**import** java.util.Map.Entry;

**public** **class** HelloWorld2 {

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

HashMap<Integer, String> m1 =**new** HashMap<Integer, String>();

m1.put(1, "ramu");

m1.put(3, "ravi");

m1.put(2, "raju");

m1.put(4, **null**);

m1.put(**null**, "mahesh");

System.***out***.println(m1);

**for**(Map.Entry<Integer, String> m2 : m1.entrySet()) {

System.***out***.println(m2.getKey()+" "+m2.getValue());

}

//adding elements

m1.put(5, "rakesh");

System.***out***.println(m1);

//remove elements

m1.remove(2); //removes key of 2

System.***out***.println(m1);

//replace elements

m1.put(3, "rahul"); //replaces value ravi of key 3 to rahul

m1.put(**null**, **null**); //replaces value of mahesh of key null to null

System.***out***.println(m1);

}

}

Example: Below is an example of user defined method objects

**package** com.lokesh;

**public** **class** HelloWorld1 {

**int** empid;

String empname;

**int** empage;

HelloWorld1(**int** empid,String empname,**int** empage) {

**this**.empid=empid;

**this**.empname=empname;

**this**.empage=empage;

}

}

**package** com.lokesh;

**import** java.util.\*;

**import** java.util.Map.Entry;

**public** **class** HelloWorld2 {

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

HelloWorld1 hw1 =**new** HelloWorld1(1, "ramu", 23);

HelloWorld1 hw2 =**new** HelloWorld1(2, "raju", 33);

HelloWorld1 hw3 =**new** HelloWorld1(3, "ravi", 43);

HashMap<Integer, HelloWorld1> m1 =**new** HashMap<Integer, HelloWorld1>();

m1.put(1, hw3);

m1.put(3, hw2);

m1.put(2, hw1);

//traversing Map

**for**(Map.Entry<Integer, HelloWorld1> m2 : m1.entrySet()) {

**int** key =m2.getKey();

HelloWorld1 hw4 =m2.getValue();

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

System.***out***.println(hw4.empid+" "+hw4.empname+" "+hw4.empage);

}

}

}

LinkedHashMap – contains values based on key i.e. key value pair, contains unique elements, may have one null key and multiple null values, is not synchronized, maintains insertion order

Example:

**package** com.lokesh;

**import** java.util.\*;

**import** java.util.Map.Entry;

**public** **class** HelloWorld2 {

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

LinkedHashMap<Integer, String> m1 =**new** LinkedHashMap<Integer, String>();

m1.put(1, "ramu");

m1.put(3, "raju");

m1.put(2, "ravi");

System.***out***.println(m1);

**for**(Map.Entry<Integer, String> m2 : m1.entrySet()) {

System.***out***.println(m2.getKey()+" "+m2.getValue());

}

//adding elements

m1.put(4, "pradip");

System.***out***.println(m1);

//removing elements

m1.remove(3); //removes the key of 3 i.e. 3 and raju

System.***out***.println(m1);

//replacing elements

m1.put(2, "rahul"); //replaces of key 2

m1.replace(1, "mahesh");

System.***out***.println(m1);

}

}

Example: Below is an example of user defined objects

**package** com.lokesh;

**public** **class** HelloWorld1 {

**int** empid;

String empname;

**int** empage;

HelloWorld1(**int** empid,String empname,**int** empage) {

**this**.empid=empid;

**this**.empname=empname;

**this**.empage=empage;

}

}

**package** com.lokesh;

**import** java.util.\*;

**import** java.util.Map.Entry;

**public** **class** HelloWorld2 {

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

HelloWorld1 hw1 =**new** HelloWorld1(1, "ramu", 23);

HelloWorld1 hw2 =**new** HelloWorld1(2, "raju", 33);

HelloWorld1 hw3 =**new** HelloWorld1(3, "ravi", 43);

LinkedHashMap<Integer, HelloWorld1> m1 =**new** LinkedHashMap<Integer, HelloWorld1>();

m1.put(3, hw1);

m1.put(2, hw3);

m1.put(1, hw2);

**for**(Map.Entry<Integer, HelloWorld1> m2 : m1.entrySet()) {

**int** key =m2.getKey();

HelloWorld1 hw4 =m2.getValue();

System.***out***.println(hw4.empid+" "+hw4.empname+" "+hw4.empage);

}

}

}

TreeMap – contains values based on key i.e. key value pair, contains unique elements, cannot contain null key but can contain multiple null values, not synchronized, maintains ascending order

Example:

**package** com.lokesh;

**import** java.util.\*;

**import** java.util.Map.Entry;

**public** **class** HelloWorld2 {

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

TreeMap<Integer, String> t1 =**new** TreeMap<Integer, String>();

t1.put(3, "ramu");

t1.put(1, "raju");

t1.put(2, "ravi");

System.***out***.println(t1);

**for**(Map.Entry<Integer, String> t2 : t1.entrySet()) {

System.***out***.println(t2.getKey()+" "+t2.getValue());

}

//adding elements

t1.put(4, "pradip");

System.***out***.println(t1);

//removing elements

t1.remove(3); //removes key 3 i.e of "3 ramu"

System.***out***.println(t1);

//replacing elements

t1.put(2, "mahesh"); //replaces key 2 of ravi to mahesh

t1.replace(4, "rahul"); //changes key 4 of pradip to rahul

System.***out***.println(t1);

}

}

Example: below is an example for user defined objects

**package** com.lokesh;

**public** **class** HelloWorld1 {

**int** empid;

String empname;

**int** empage;

HelloWorld1(**int** empid,String empname,**int** empage) {

**this**.empid=empid;

**this**.empname=empname;

**this**.empage=empage;

}

}

**package** com.lokesh;

**import** java.util.\*;

**import** java.util.Map.Entry;

**public** **class** HelloWorld2 {

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

HelloWorld1 hw1 =**new** HelloWorld1(1, "ramu", 23);

HelloWorld1 hw2 =**new** HelloWorld1(2, "raju", 33);

HelloWorld1 hw3 =**new** HelloWorld1(3, "ramu", 43);

TreeMap<Integer, HelloWorld1> t1 =**new** TreeMap<Integer, HelloWorld1>();

t1.put(3, hw1);

t1.put(1, hw3);

t1.put(2, hw2);

**for**(Map.Entry<Integer, HelloWorld1> t2 : t1.entrySet()) {

**int** key =t2.getKey();

HelloWorld1 hw4 =t2.getValue();

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

System.***out***.println(hw4.empid+" "+hw4.empname+" "+hw4.empage);

}

}

}

HashTable – is a array of list, each list is known as bucket. The position of the bucket is identified by calling the hashcode() method, contains values based on key i.e. key value pair, contains unique elements, doesn’t allow null key or value, synchronized

Example:

**package** com.lokesh;

**import** java.util.\*;

**import** java.util.Map.Entry;

**public** **class** HelloWorld2 {

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

Hashtable<Integer, String> h1 =**new** Hashtable<Integer, String>();

h1.put(3, "ramu");

h1.put(1, "raju");

h1.put(2, "ravi");

**for**(Map.Entry<Integer, String> h2 : h1.entrySet()) {

System.***out***.println(h2.getKey()+" "+h2.getValue());

}

//adding elements

h1.put(4, "pradip");

System.***out***.println(h1);

//removing elements

h1.remove(3); //removes key 3 i.e "3 ramu"

System.***out***.println(h1);

//replacing elements

h1.put(2, "mahesh"); //replaces key 2 of ravi to mahesh

h1.replace(1, "rahul"); //replaces key 1 of raju to rahul

System.***out***.println(h1);

//in below it will check that if it is found

//it will provide that value otherwise the second parameter below

//like for example in below we gave "not there"

System.***out***.println(h1.getOrDefault(2, "not there"));

System.***out***.println(h1.getOrDefault(6, "not there"));

//the below adds elements if that key doesn't exist

h1.putIfAbsent(2, "ramesh"); //since key already present it doesn't add

h1.putIfAbsent(5, "ramesh"); //since key not present it will add

System.***out***.println(h1);

}

}

Example: Below is an example of user defined objects

**package** com.lokesh;

**public** **class** HelloWorld1 {

**int** empid;

String empname;

**int** empage;

HelloWorld1(**int** empid,String empname,**int** empage) {

**this**.empid=empid;

**this**.empname=empname;

**this**.empage=empage;

}

}

**package** com.lokesh;

**import** java.util.\*;

**import** java.util.Map.Entry;

**public** **class** HelloWorld2 {

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

HelloWorld1 hw1 =**new** HelloWorld1(1, "ramu", 23);

HelloWorld1 hw2 =**new** HelloWorld1(1, "raju", 33);

HelloWorld1 hw3 =**new** HelloWorld1(1, "ravi", 43);

Hashtable<Integer, HelloWorld1> h1 =**new** Hashtable<Integer, HelloWorld1>();

h1.put(3, hw2);

h1.put(1, hw3);

h1.put(2, hw1);

**for**(Map.Entry<Integer, HelloWorld1> h2 :h1.entrySet()) {

**int** key =h2.getKey();

HelloWorld1 hw4 =h2.getValue();

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

System.***out***.println(hw4.empid+" "+hw4.empname+" "+hw4.empage);

}

}

}

Before we see about Enum collection we will know what is Enum

Java Enums – is a data type which contains fixed set of constants

Enum can have fields, constructors and method, may implement many interfaces but cannot extend any class because it internally extends Enum class

Example:

**package** com.lokesh;

**import** java.util.\*;

**import** java.util.Map.Entry;

**public** **class** HelloWorld2 {

**public** **enum** season { ***WINTER***, ***SPRING***, ***SUMMER***, ***RAINY***}

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

//to get one from above values

season s1=season.***WINTER***;

System.***out***.println(s1);

//to get all

**for**(season s2:season.*values*())

System.***out***.println(s2);

}

}

EnumSet – is a specialized Set implementation for use with enum types

Example:

**package** com.lokesh;

**public** **enum** HelloWorld1 {***WINTER***,***SPRING***,***SUMMER***,***RAINY***}

**package** com.lokesh;

**import** java.util.\*;

**import** java.util.Map.Entry;

**public** **class** HelloWorld2 {

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

//to get certain values

Set<HelloWorld1> e1 =EnumSet.*of*(HelloWorld1.***WINTER***, HelloWorld1.***RAINY***);

Iterator<HelloWorld1> itr =e1.iterator();

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

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

}

//to get all values

Set<HelloWorld1> e2 =EnumSet.*allOf*(HelloWorld1.**class**);

System.***out***.println(e2);

//not to get any values

Set<HelloWorld1> e3 =EnumSet.*noneOf*(HelloWorld1.**class**);

System.***out***.println(e3);

}

}

EnumMap – is a specialized Map for Enum keys

**package** com.lokesh;

**import** java.util.\*;

**import** java.util.Map.Entry;

**public** **class** HelloWorld2 {

**public** **enum** days{

***MONDAY***,***TUESDAY***,***WEDNESDAY***,***THURSDAY***,***FRIDAY***,***SATURDAY***,***SUNDAY***

}

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

//to get certain values

EnumMap<days, String> e1 =**new** EnumMap<days, String>(days.**class**);

e1.put(days.***MONDAY***, "1");

e1.put(days.***TUESDAY***, "2");

e1.put(days.***WEDNESDAY***, "3");

**for**(Map.Entry<days, String> e2:e1.entrySet()) {

System.***out***.println(e2.getKey()+" "+e2.getValue());

}

}

}

Java Vector – is like dynamic array which can grow or shrink its size. Unlike array we can store n-number of elements in it as there is no size limit, It implements List interface, It is recommended to use Vector in thread-safe implementation only, If you don’t need to use thread-safe implementation you should use ArrayList, Vector contains many legacy methods that are not part of collections framework

Example:

**package** com.lokesh;

**import** java.util.\*;

**public** **class** HelloWorld2 {

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

Vector<String> v1 =**new** Vector<String>();

v1.add("ramu");

v1.add("raju");

v1.add("ravi");

System.***out***.println(v1);

System.***out***.println("size is "+v1.size());

System.***out***.println("default capacity is "+v1.capacity());

//you can add using addElement as well

v1.addElement("pradip");

v1.addElement("rahul");

//check if raju is present in v1

**if**(v1.contains("raju")) {

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

}**else**{

System.***out***.println("not present");

}

Vector<Integer> v2 =**new** Vector<Integer>();

v2.add(23);

v2.add(87);

v2.add(48);

System.***out***.println(v2);

}

}

Java Stack – is a linear data structure used to store collection of objects, it is LIFO, Stack provides operation like push (inserts the element), pop (removes the element), search etc..

Example:

**package** com.lokesh;

**import** java.util.\*;

**public** **class** HelloWorld2 {

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

Stack<String> s1 =**new** Stack<String>();

s1.push("ramu");

s1.push("raju");

s1.push("ravi");

System.***out***.println(s1);

Iterator<String> itr =s1.iterator();

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

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

}

//as per LIFO the last element enetered becomes first in stack

//since ravi is enetered last it will be first in stack

//peek method returns first element in stack

String t1 =s1.peek();

System.***out***.println(t1);

//removing elements

// (removes from first since it is LIFO the last ravi becomes first in stack )

s1.pop(); //removes ravi since it is first as per LIFO

System.***out***.println(s1);

Stack<Integer> s2 =**new** Stack<Integer>();

s2.push(56);

s2.push(87);

s2.push(23);

s2.push(41);

System.***out***.println(s2);

//removes first element in stack as per LIFO i.e. 41

s2.pop();

System.***out***.println(s2);

}

}

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [List](http://www.javamadesoeasy.com/2015/04/list-hierarchy-in-java-detailed.html) | Duplicate elements | insertion order | Sorted by natural order | [synchronized](http://www.javamadesoeasy.com/2015/03/synchronization-blocks-and-methods.html) | null elements |
| [ArrayList](http://www.javamadesoeasy.com/2015/04/arraylist-in-java.html) | Yes | Yes |  |  | Yes |
| [LinkedList](http://www.javamadesoeasy.com/2015/04/linkedlist-in-java.html) | Yes | Yes |  |  | Yes |
| [Vector](http://www.javamadesoeasy.com/2015/04/arraylist-vs-vector-similarity-and.html) | Yes | Yes |  | Yes | Yes |
|  |  |  |  |  |  |
| [Set](http://www.javamadesoeasy.com/2015/04/set-hierarchy-in-java-detailed-hashset.html) | Duplicate elements | insertion order | Sorted by natural order | synchronized | null elements |
| [HashSet](http://www.javamadesoeasy.com/2015/04/hashset-in-java.html) |  |  |  |  | Yes |
| [LinkedHashSet](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html) |  | Yes |  |  | Yes |
| [TreeSet](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html) |  |  | Yes |  | No |
|  |  |  |  |  |  |
| [Map](http://www.javamadesoeasy.com/2015/04/map-hierarchy-in-java-detailed-hashmap.html) | Duplicate Keys | insertion order of keys | Sorted by natural order of keys | synchronized | null keys or null values |
| [HashMap](http://www.javamadesoeasy.com/2015/04/hashmap-in-java.html) |  |  |  |  | one null key and many null values |
| [Hashtable](http://www.javamadesoeasy.com/2015/04/hashmap-and-hashtable-similarity-and.html) |  |  |  | Yes | No |
| [LinkedHashMap](http://www.javamadesoeasy.com/2015/04/hashmap-vs-hashtable-vs-linkedhashmap.html) |  | Yes |  |  | one null key and many null values |
| [TreeMap](http://www.javamadesoeasy.com/2015/04/hashmap-vs-hashtable-vs-linkedhashmap.html) |  |  | Yes |  | Null key not allowed,  Allow many null values |