COLLECTIONS

Collection API framework are set of classes that implement various data structures like lists,sets,maps and vectors.Basically

1.we can add objects to the collection

2.we can remove objects from the collection

3.find out if the particular object is there in collection

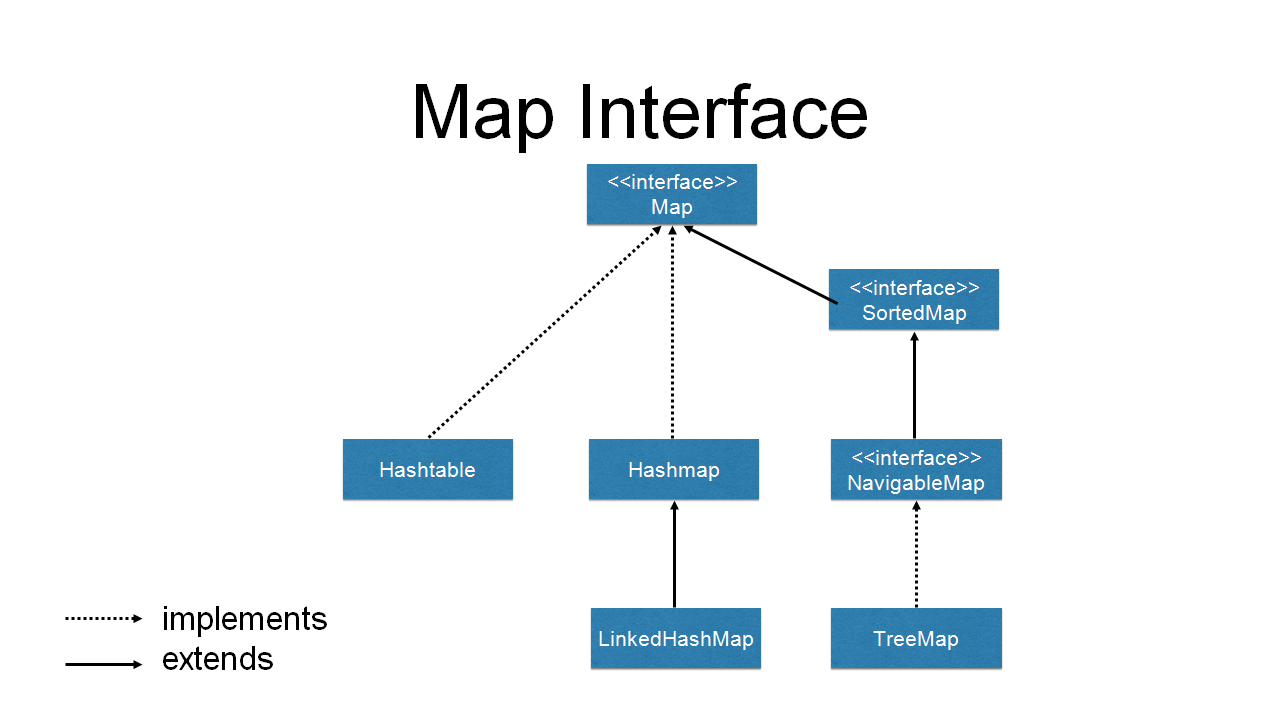
4.retrieve objects from the collections

5.iterate through the collection

Note:

In general collection is an object that contains other object and object present in the collection are called element

https://www.javatpoint.com/images/collection-hierarchy.png



Note:

1.All the Collections are under util packages.

2.Collections class prints the object and if the objects are Printing values then corresponding classes of the object , toString() method is over ridden

Ex:if we add 10, it is wrapped into integer class Object and when the object is printed, the value is printed instead of the address since toString() method of the Interger wrapper class is over ridden

3.All the collection classes has to take Object as argument

Note:

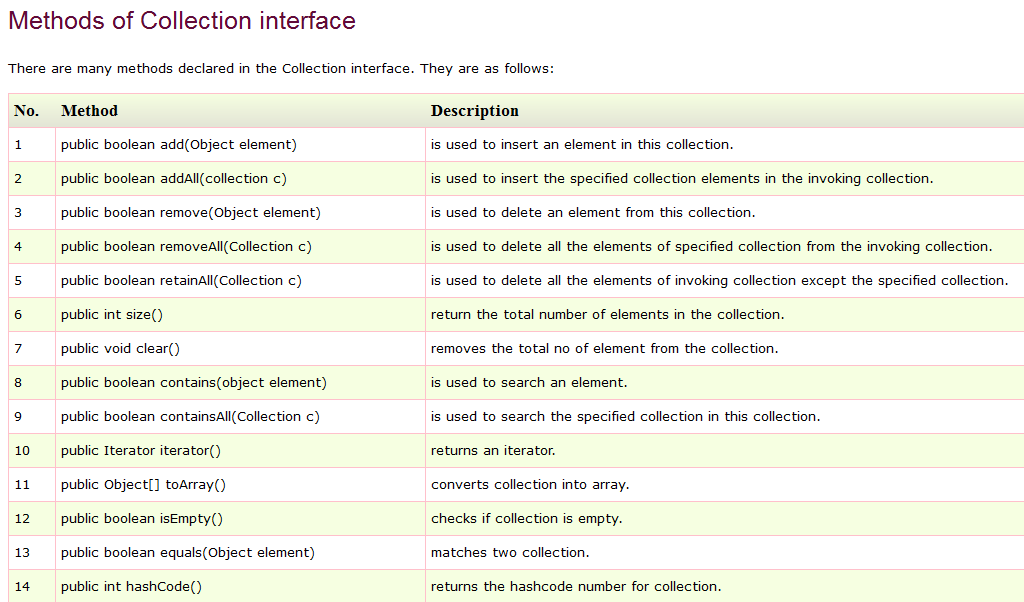
collection represent any collection(data structure) in which Objects are stored and can be iterated

Collection which is actually the java.util.Collection Interface from which set,list and Queue are extended.

Collections is java.util.Collections class which holds set of static utility method for use with Collections

Important Abstract classes and interfaces in the CollectionsFramework

|  |  |
| --- | --- |
| Iterable | A class implementing this interface can be used for iterating with enahnced loop |
| Collection | Base interface in the collection heirarchy |
| List | Interface for containers that store a sequence of elements.We can store duplicate elements in a List |
| Set,SortedSet,NavigableSet,Queue,  Deque | Interfaces for container classes which don't allow duplicate elements.  SortedSet have elements in sorted order  NavigableSet allows searching the set for the closest match  Queue is the base interface for container classes that holds elements.classes implementing Queue can be LIFO as in stack data structure) of FIFO as in queue data structure  Deque, we can insert or remove elements from both ends |
| Map,SortedMap,NavigableMap | Interfaces for containers that map keys to values  In SortedMap, the keys are in a sorted order  NavigableMap allows you to search and return the closest match  for given criteria  Note: Map heirarchy does not extend Collection interface |
| Iterator,ListIterator | If a container class implements Iterator, we can traverse the container elements in forward direction  If a class implements ListIterator we traverse in both directions |



LISTS:

Lists are used for Storing sequence of elements and it allows duplicates and maintains same order.

ArrayLists:

ArrayLists is an ordered(indexed) collection, not sorted and auto-growable array. Its internally implemented as resizeable array.

This is one of most widely used concrete class . Its fast to search, but slow to insert or delete and it allows duplicate.

Declare ArrayList

ArrayList list=new ArrayList();

New array list is created, In order to add the elements into this array list we use add() method and after adding the elements if we want to know the size then we use size() method.

package com.qsp.collections;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.ListIterator;

interface Fruit{

}

class Orange implements Fruit

{

}

public class ArrayListDemo {

public static void main(String[] args) {

ArrayList list=new ArrayList();

list.add(10);

list.add("hello");

list.add(true);

list.add(new Orange());

Orange o1=new Orange();

list.add(o1);

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

System.out.println("----enhanced for loop");

for(Object o:list)

{

System.out.println(o);

}

System.out.println("--with using simple for loop--");

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

{

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

}

O/P:

5

----enhanced for loop

10

hello

true

com.qsp.collections.Orange@2a139a55

com.qsp.collections.Orange@15db9742

--with using simple for loop--

10

hello

true

com.qsp.collections.Orange@2a139a55

com.qsp.collections.Orange@15db9742

Note:

We used enhanced for loop to view the elements and we also used a method called get() method to access the elements in the arraylist.

Note:

We use contains method to check whether the certain elements is their in the arraylist are not

package com.qsp.collections;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.ListIterator;

interface Fruit{

}

class Orange implements Fruit

{

}

public class ArrayListDemo {

public static void main(String[] args) {

ArrayList list=new ArrayList();

list.add(10);

list.add("hello");

list.add(true);

list.add(new Orange());

Orange o1=new Orange();

list.add(o1);

System.out.println(list.contains(true));

System.out.println(list.contains("hello"));

System.out.println(list.contains(10));

System.out.println(list.contains(15));

System.out.println("----for specific object-----");

String s=(String)list.get(1);

System.out.println(s.length());

Note:

Here, for a specific object if we want to know the length, then we are converting it to string and later we are finding the length

O/P:

true

true

true

false

----for specific object-----

5

Note:

If we want arraylist to take elements of single type then we need to declare it as

ArrayList<String> lst1=new ArrayList<String>();

Here arraylist takes elements of only String type

ArrayList<integer> lst1=new ArrayList<String>();// works only in JDK 1.7

ArrayList<integer> lst1=new ArrayList<integer>();//this arraylist takes only element of integer

ArrayList<Orange> lst1=new ArrayList<Orange>();//this arraylist takes only element of type orange

ArrayList<fruit> lst1=new ArrayList<fruit>();//this arraylist takes only element of type fruit

//lst1.add(new Orange());objects of fruit interface,objects of class implementing fruit interface

Program:

package com.qsp.collections;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.ListIterator;

interface Fruit{

}

class Orange implements Fruit

{

}

public class ArrayListDemo {

public static void main(String[] args) {

ArrayList<String> lst1=new ArrayList<String>();

lst1.add("harsha");

lst1.add("vimal");

lst1.add("madhan");

for(String o:lst1)

{

System.out.println(o);

}

System.out.println("-------without using enhanced for loop");

for(int i=0;i<lst1.size();i++)

{

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

}

O/P:

harsha

vimal

madhan

-------without using enhanced for loop

harsha

vimal

madhan

Note:

Converting the arraylist to array, we use toArray method

package com.qsp.collections;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.ListIterator;

interface Fruit{

}

class Orange implements Fruit

{

}

public class ArrayListDemo {

public static void main(String[] args) {

ArrayList list=new ArrayList();

list.add(10);

list.add("hello");

list.add(true);

list.add(new Orange());

Orange o1=new Orange();

list.add(o1);

System.out.println("converting arraylist to object array ");

Object[] obj=list.toArray();

for(Object o:obj)

{

System.out.println(o);

}

}

}

O/P

converting arraylist to object array

10

hello

true

com.qsp.collections.Orange@2a139a55

com.qsp.collections.Orange@15db9742

Note:

Using Iterator class we can go through the elements,Using Iterator we can only go through objects from top to bottom and Using ListIterator we can go through objects from top to bottom and bottom to top

package com.qsp.collections;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.ListIterator;

interface Fruit{

}

class Orange implements Fruit

{

}

public class ArrayListDemo {

public static void main(String[] args) {

ArrayList lst1=new ArrayList ();

lst1.add("harsha");

lst1.add("vimal");

lst1.add("madhan");

System.out.println("using iterator");

Iterator itr=lst1.iterator();

while(itr.hasNext())

{

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

}

ListIterator litr=lst1.listIterator();

while(litr.hasNext())

{

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

}

System.out.println("going reverse");

while(litr.hasPrevious())

{

System.out.println(litr.previous());

}

}

}

O/P:

using iterator

harsha

vimal

madhan

harsha

vimal

madhan

going reverse

madhan

vimal

harsha

NOTE:

Addall()- we should have two lists, suppose Ex: list1 and list2

So list1.addAll(list2)—it adds list2 elements in list1

Removeall();we should have two lists, suppose Ex: list1 and list2

So list1.removeAll(list2)—it removes list2 elements in list1

Retainall(); we should have two lists, suppose Ex:list1 and list2

So list1.retainall(list2)-it will remove the elements of list1 and retains elements of list2

Program:

package com.qsp.collections;

import java.util.ArrayList;

public class ArrayListDemo2 {

public static void main(String[] args) {

ArrayList list1=new ArrayList();

list1.add(10);

list1.add("hello");

list1.add(true);

list1.add(new Orange());

Orange o1=new Orange();

list1.add(o1);

ArrayList list2=new ArrayList();

list2.add("java");

list2.add(100);

list2.add(2, true);

System.out.println(list2);

list1.addAll(list2);

System.out.println(list1);

System.out.println("----removing1----");

list1.remove(5);

System.out.println(list1);

System.out.println("------removing2-----");

list1.remove(Integer.valueOf(10));

System.out.println(list1);

System.out.println("------remove3-----");

list1.remove(o1);

System.out.println(list1);

//

//

System.out.println("---remove all method-----");

list1.removeAll(list2);

System.out.println(list1);

System.out.println("----adding againg elements of list 2---");

list1.addAll(list2);

System.out.println(list1);

list1.retainAll(list2);

System.out.println("------when retaining------s");

System.out.println(list1);

list1.clear();

System.out.println("when cleared everything in the list");

System.out.println(list1);

}

}

O/p:

[java, 100, true]

[10, hello, true, com.qsp.collections.Orange@2a139a55, com.qsp.collections.Orange@15db9742, java, 100, true]

----removing1----

[10, hello, true, com.qsp.collections.Orange@2a139a55, com.qsp.collections.Orange@15db9742, 100, true]

------removing2-----

[hello, true, com.qsp.collections.Orange@2a139a55, com.qsp.collections.Orange@15db9742, 100, true]

------remove3-----

[hello, true, com.qsp.collections.Orange@2a139a55, 100, true]

---remove all method-----

[hello, com.qsp.collections.Orange@2a139a55]

----adding againg elements of list 2---

[hello, com.qsp.collections.Orange@2a139a55, java, 100, true]

------when retaining------s

[java, 100, true]

when cleared everything in the list

[]

Note:

There are two remove methods

remove(int index), which remove the value present at the specified index

public boolean remove([Object](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html) o)

using this remove method if we want to remove a value we can use it like

list1.remove(Integer.valueOf(10));

if we want to remove object we can use reference variable

list1.remove(o1);

Methods in Iterator Interface

|  |  |
| --- | --- |
| Method | Description |
| 1.boolean hasNext()  2.E next()  3.void remove() | Checks if the iterator has more elements to traverse  Moves iterator to the next element and returns that element  Removes last visited element from underlying element class.  Note: next() should have been called before calling remove() else it will throw IllegalStateException |

Methods in ListIterator Interface

ListIterator extends Iterator interface

|  |  |
| --- | --- |
| Method | Description |
| boolean hasPrevious()  E previous()  int nextIndex()  int previousIndex()  void set(Element)  void add(Element) | Checks if the iterator has more elements to traverse in reverse elements  Moves the iterator to next element and returns that element in reverse  Returns the index of the next element in the iteration in forward direction  Returns the index of the next element in the iteration in forward direction  Sets the last element visited (using next or previous ) it replaces the existing element  Adds element into the list at the current iteration position |

Program

package com.qsp.collections;

import java.util.ArrayList;

class Apple{

int wt;

Apple(int wt)

{

this.wt=wt;

}

public boolean equals(Object o)

{

this.wt=((Apple)o).wt;

return true;

}

public int hashcode()

{

return 7\*5\*11;

}

}

public class ArrayListDemo3 {

public static void main(String[] args) {

ArrayList list =new ArrayList();

list.add(new Apple(100));

list.add(new Apple(120));

list.add(new Apple(150));

System.out.println(list);

Apple a1=new Apple(120);

list.add(a1);

list.remove(new Apple(100));//when you remove like this it would remove the object

System.out.println("--------------After removing-------------");

System.out.println(list);

list.remove(a1);//when you remove using reference variable then the object is deleted

System.out.println("----------------after removing---------");

System.out.println(list);

}

}

O/P:

[com.qsp.collections.Apple@2a139a55, com.qsp.collections.Apple@15db9742, com.qsp.collections.Apple@6d06d69c]

--------------After removing-------------

[com.qsp.collections.Apple@15db9742, com.qsp.collections.Apple@6d06d69c, com.qsp.collections.Apple@7852e922]

----------------after removing---------

[com.qsp.collections.Apple@6d06d69c, [com.qsp.collections.Apple@7852e922](mailto:com.qsp.collections.Apple@7852e922)]

Note:

When you use list.remove(new Apple(100)); to remove an object it will remove it

If you want to remove an object then we need to assign It to a reference variable and then we need to remove it

list.remove(a1)

Program:

package com.qsp.collections;

import java.util.ArrayList;

class Chocolate

{

void giveAroma(){

System.out.println("------give chocolate aroma------");

}

}

class Halwa

{

void getAroma()

{

System.out.println("--------get halwa aroma----------");

}

}

public class ArrayListDemo4 {

public static void main(String[] args) {

ArrayList list=new ArrayList();

list.add(new Chocolate());

list.add(new Halwa());

list.add(new Chocolate());

list.add(new Halwa());

list.add(new Halwa());

list.add(new Chocolate());

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

for(Object o:list)

{

if(o instanceof Chocolate)

{

((Chocolate)o).giveAroma();

}

else if(o instanceof Halwa)

{

((Halwa)o).getAroma();

}

System.out.println(o);

}

}

}

O/P:

6

------give chocolate aroma------

com.qsp.collections.Chocolate@2a139a55

--------get halwa aroma----------

com.qsp.collections.Halwa@15db9742

------give chocolate aroma------

com.qsp.collections.Chocolate@6d06d69c

--------get halwa aroma----------

com.qsp.collections.Halwa@7852e922

--------get halwa aroma----------

com.qsp.collections.Halwa@4e25154f

------give chocolate aroma------

com.qsp.collections.Chocolate@70dea4e

Program:

package com.qsp.collections;

import java.util.ArrayList;

import java.util.Collections;

public class ArrayList5 {

public static void main(String[] args) {

ArrayList list=new ArrayList();

list.add(100);

list.add(150);

list.add((int)12.5);

list.add(10);

System.out.println("before sorting");

System.out.println(list);

Collections.sort(list);

System.out.println("After sorting");

System.out.println(list);

list.clear();

list.add("harsha");

list.add("girish");

list.add("bhanu");

System.out.println("before shuffle");

System.out.println(list);

Collections.shuffle(list);

System.out.println("After shuffle");

System.out.println(list);

}

}

O/P:

before sorting

[100, 150, 12, 10]

After sorting

[10, 12, 100, 150]

before shuffle

[harsha, girish, bhanu]

After shuffle

[bhanu, harsha, girish]

Note:

Sort() method is present in the Collections class, we need to import it and later we need to use it

Collections.sort(list);

When you are using sort funtion then all the elements of collections should be of same type or else it will give class cast exception

Same is the case with shuffle, its used to shuffle the strings

Collections.shuffle(list);

LinkedList:

|  |
| --- |
| Collection(I) |

|  |  |  |
| --- | --- | --- |
| List(I) |  | Queue(I) |

|  |
| --- |
| DeQueue(I) |

|  |
| --- |
| LinkedList(C) |

Linked List is a combination of both list and queue

A Linked list is ordered by index position, like Arraylist,except that the elements are doubly linked to one another , Inserting and deleting is very fast in Linked list. But, accessing elements is slow as we have to traverse

When you want to add or remove elements frequently, then you can use Linked list

INTERVIEW QUESTIONS

Q.What is the difference between Arraylist and Linked list and Vector?

NOTE:

Linked List implements both interfaces List and Dequeue(It should act as both list as well as Queue)

Program:

package com.qsp.collections;

import java.util.LinkedList;

import java.util.ListIterator;

public class LinkedListDemo {

public static void main(String[] args) {

LinkedList list=new LinkedList();

list.add(100);

list.add("java");

list.add(12.45);

list.add(true);

System.out.println(list);

System.out.println("--using get method--");

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

System.out.println("----gets first item------");

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

System.out.println(list.get(list.size()-1));

System.out.println("---gets last item-----");

System.out.println(list.get(1));

System.out.println("--adding items----");

list.addFirst("welcome to java world");

list.addLast("Hope you enjoyed");

System.out.println(list);

System.out.println("print all the elements");

ListIterator litr=list.listIterator();

System.out.println("using iterator");

System.out.println("-------");

System.out.println("---------from top to bottom-----------");

while(litr.hasNext())

{

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

}

System.out.println("---------from bottom to top------------");

while(litr.hasPrevious())

{

System.out.println(litr.previous());

}

}

}

O/P:

[100, java, 12.45, true]

--using get method--

100

----gets first item------

true

true

---gets last item-----

java

--adding items----

[welcome to java world, 100, java, 12.45, true, Hope you enjoyed]

print all the elements

using iterator

-------

---------from top to bottom-----------

welcome to java world

100

java

12.45

true

Hope you enjoyed

---------from bottom to top------------

Hope you enjoyed

true

12.45

java

100

welcome to java world

NOTE:

getFirst()

suppose we have added certain elements in the list and we want to get the first element then we can use this method,this method returns the first element of the list.

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

getLast()

suppose we have added certain elements in the list and we want to get the last element then we can use this method,this method returns the last element of the list.

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

addFirst(Object)

suppose we have added certain elements in the list and again we want to add an element to the top of the element i.e, as an first element then we use this method

list.addFirst("welcome to java world");

addLast(Object)

suppose we have added certain elements in the list and again we want to add an element to the bottom of the element i.e, as an last element then we use this method

list.addLast("Hope you enjoyed");

IMPORTANT:

In order to traverse through(go through) the items or objects we can use simple for loop, enhanced for loop, toArray(),Iterator(),descenting Iterator, peak, poll.

Difference Between Arraylist and Linked list

|  |  |
| --- | --- |
| ArrayList | LinkedList |
| 1.Arraylist internally uses dynamic array to store the elements | 1.Linkedlist internally uses doubly linked list to store the elements |
| 2.Manipulation with Arraylist is slow because it internally uses array. If any elements is removed from the array, all bits are shifed in memory | 2. Manipulation with Linkedlist is faster than array list because it internally uses doubly linked list, so bit shift is not required in memory |
| 3.Arraylist class can act only as a list ,because it implements list only | 3.Linked list class acts as list and queue both because it implements list and DeQueue interfaces |
| 4.Arraylist is better for storing and accessing the elements | 4.LinkedList is better for manipulating data. |

Priority Queue:

Priority Queue class Extends Abstractqueue which implements Queue interface but it does not order the elements in fifo manner.

Internally autosort

Random display

Dissimilar data throws class cast Exception

Null is not allowed in this collection

The Iterator() of this priority queue does not guarentee for traversal of the queue elements in any particular order.

Program

package com.qsp.collections;

import java.util.Iterator;

import java.util.PriorityQueue;

import java.util.Queue;

public class PriorityQueueDemo {

public static void main(String[] args) {

PriorityQueue pq=new PriorityQueue();

pq.add("Sharukh");

pq.add("Rajnikanth");

pq.add("Prabhas");

pq.add("Hrithik");

System.out.println(pq);

System.out.println("-----using element() and peek() methods------");

System.out.println(pq.element());

System.out.println("- element() gets the first element but does not remove it");

System.out.println(pq);

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

System.out.println("- peek() gets the first element but does not remove it");

System.out.println(pq);

System.out.println("------printing elements -----");

Iterator itr=pq.iterator();

while(itr.hasNext())

{

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

}

System.out.println("---using the poll and remove methods---");

System.out.println(pq.poll());

System.out.println("- poll() gets the first element and also removes it");

System.out.println(pq);

System.out.println(pq.remove());

System.out.println("- remove() gets the first element and also removes it");

Queue<Integer> q =new PriorityQueue<>();//here we are upcasting

int arr[]={10,50,25,100,152,149};

for(int a:arr)

{

q.add(a);

}

System.out.println(q);

for(int a:arr)

{

System.out.println(q.poll());//here poll() is removing everyelement from the list

}

/\*alternate way

\* int size=q.size()

\* for(int i=0;i<size;i++)

\* {

\* System.out.println(pq.poll());

\* }

\*/

/\*we can also use like this, however the list is empty now it wont print anything

\* for(Object o:q)

{

System.out.println(o);

}\*/

System.out.println("after using poll method the list became empty");

System.out.println(q);//[]

System.out.println(q.poll());//null

System.out.println(q.size());//0

System.out.println(q.element());//NoSuchElement Exception Occurs

System.out.println(q.peek());//Null

System.out.println(q.remove());// NoSuchElement Exception occurs

}

}

O/P:

[Hrithik, Prabhas, Rajnikanth, Sharukh]

-----using element() and peek() methods------

Hrithik

- element() gets the first element but does not remove it

[Hrithik, Prabhas, Rajnikanth, Sharukh]

Hrithik

- peek() gets the first element but does not remove it

[Hrithik, Prabhas, Rajnikanth, Sharukh]

------printing elements -----

Hrithik

Prabhas

Rajnikanth

Sharukh

---using the poll and remove methods---

Hrithik

- poll() gets the first element and also removes it

[Prabhas, Sharukh, Rajnikanth]

Prabhas

- remove() gets the first element and also removes it

[10, 50, 25, 100, 152, 149]

10

25

50

100

149

152

after using poll method the list became empty

[]

null

0

Exception in thread "main" java.util.NoSuchElementException

at java.util.AbstractQueue.remove(Unknown Source)

at com.qsp.collections.PriorityQueueDemo.main(PriorityQueueDemo.java:72)

Peek() returns the highest priority element in the queue without removing it

Poll() returns the highest priority element from the queue and removes it

Difference between poll() and remove()? Difference between peek() and element()?

if no element is present then Peek/poll returns null

and remove/element methods generate NoSuchElement Exception

Sets

Duplicates data is not allowed .

For String unique data, use set.Its is an Unsorted,Unordered.

Important Concrete Classes are Hashset and Treeset

Hashset:

It uses the Hashcode of an Object being inserted.this class can be used when we want collection with no duplicates and order is not important when we iterate.

Program:

package com.qsp.collections;

import java.util.HashSet;

import java.util.Iterator;

import java.util.Set;

public class HashsetDemo {

public static void main(String[] args) {

HashSet hs=new HashSet();

hs.add(100);

hs.add("hello");

hs.add(true);

hs.add(12.45);

System.out.println(hs);

System.out.println("---we can gothrough the objects through enhanced for loop as shown below-------");

for(Object o:hs)

{

System.out.println(o);

}

System.out.println("-------or we can use iterator as below--------");

Iterator itr =hs.iterator();

while(itr.hasNext()){

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

}

System.out.println("-----using set------");

Set<Integer> hs1=new HashSet();

hs1.add(10);

hs1.add(50);

hs1.add(150);

hs1.add(1500);

System.out.println(hs1);

HashSet<Integer> hs2=new HashSet<Integer>();

hs2.add(60);

hs2.add(70);

hs2.add(80);

hs2.add(100);

System.out.println("-----using addAll() method-----");

System.out.println(hs1.addAll(hs2));

System.out.println("----printing the list---");

System.out.println(hs1);

System.out.println("------using retainAll() method--");

System.out.println(hs1.retainAll(hs2));

System.out.println("----printing the list------------");

System.out.println(hs1);

}

}

O/p:

[100, hello, 12.45, true]

---we can gothrough the objects through enhanced for loop as shown below-------

100

hello

12.45

true

-------or we can use iterator as below--------

100

hello

12.45

true

-----using set------

[50, 150, 10, 1500]

-----using addAll() method-----

true

----printing the list---

[80, 50, 100, 150, 70, 10, 1500, 60]

------using retainAll() method--

true

----printing the list------------

[80, 100, 70, 60]

NOTE:

In HashSet, No dupicates are allowed, But we can add elements of different type.

Here.Add method, addAll method, retainAll() method returns boolean.

TreeSet:

In Treeset,No duplicates are allowed, and also we cannot add elements of different type(gives ClassCastException),its will sort the elements internally(auto sorted)

Program

package com.qsp.collections;

import java.util.Iterator;

import java.util.TreeSet;

public class TreeSetDemo {

public static void main(String[] args) {

TreeSet ts=new TreeSet();

ts.add(12.5);

ts.add(15.3);

ts.add(13.35);

ts.add(20.50);

System.out.println(ts);

System.out.println("-----using enhanced for loop-----");

for(Object o:ts)

{

System.out.println(o);

}

System.out.println("---------using iterator--------");

Iterator itr=ts.iterator();

while(itr.hasNext()){

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

}

}

}

O/P:

[12.5, 13.35, 15.3, 20.5]

-----using enhanced for loop-----

12.5

13.35

15.3

20.5

---------using iterator--------

12.5

13.35

15.3

20.5

HashMap:

HashMap is used for maintaining key and value mapping, often denoted as HashMap<Key, Value> or HashMap<K, V>. HashMap implements Map interface. It is not an ordered collection which means it does not return the keys and values in the same order in which they have been inserted into the HashMap. It neither does any kind of sorting to the stored keys and Values.

Program:

package com.qsp.collections;

import java.util.Collection;

import java.util.HashMap;

import java.util.Iterator;

import java.util.Set;

public class HashMapDemo {

public static void main(String[] args) {

HashMap hm=new HashMap();

hm.put("ind","INDIA");

hm.put("nz","NEWZEALAND");

hm.put("chn", "CHINA");

hm.put("usa", "UNITED STATES OF AMERICA");

System.out.println(hm);

System.out.println("--THE SIZE--");

System.out.println(hm.size());

System.out.println(hm.containsKey("ind"));

if(hm.containsKey("usa"))

{

System.out.println(hm.get("ind"));

}

System.out.println(hm.containsKey("chn"));

System.out.println(hm.containsValue("INDIA"));

System.out.println("---PRINT ALL THE KEYSETS----");

System.out.println(hm.keySet());

System.out.println("-----PRINT ALL THE VALUES--------");

System.out.println(hm.values());

System.out.println("--- printing all the keys using enhanced for loop----");

for(Object o:hm.keySet())

{

System.out.println(o);

}

System.out.println("--- printing all the keys using iterator----");

System.out.println(hm.get("ind"));

Set key=hm.keySet();

Iterator itr=key.iterator();

String s;

while(itr.hasNext())

{

System.out.println(s=(String)itr.next());

System.out.println(hm.get(s));

}

System.out.println("--another way---");

for(Object o:key)

{

System.out.println(o);

}

System.out.println("--printing the values using enhanced for loop--");

for(Object o:hm.values()){

System.out.println(o);

}

System.out.println("differnent method ");

Collection cls=hm.values();

Iterator itr1=cls.iterator();

while(itr1.hasNext())

{

System.out.println(itr1.next());

}

}

}

O/P:

{usa=UNITED STATES OF AMERICA, chn=CHINA, nz=NEWZEALAND, ind=INDIA}

--THE SIZE--

4

true

INDIA

true

true

---PRINT ALL THE KEYSETS----

[usa, chn, nz, ind]

-----PRINT ALL THE VALUES--------

[UNITED STATES OF AMERICA, CHINA, NEWZEALAND, INDIA]

--- printing all the keys using enhanced for loop----

usa

chn

nz

ind

--- printing all the keys using iterator----

INDIA

usa

UNITED STATES OF AMERICA

chn

CHINA

nz

NEWZEALAND

ind

INDIA

--another way---

usa

chn

nz

ind

--printing the values using enhanced for loop--

UNITED STATES OF AMERICA

CHINA

NEWZEALAND

INDIA

differnent method

UNITED STATES OF AMERICA

CHINA

NEWZEALAND

INDIA

TreeMap Demo:

The main difference between them is that HashMap is an unordered collection while TreeMap is sorted in the ascending order of its keys.

package com.qsp.collections;

import java.util.TreeMap;

public class TreeMapDemo {

public static void main(String[] args) {

TreeMap tm=new TreeMap();

tm.put("ind","INDIA");

tm.put("nz","NEWZEALAND");

tm.put("chn", "CHINA");

tm.put("usa", "UNITED STATES OF AMERICA");

System.out.println(tm);

System.out.println(tm.descendingMap());

System.out.println(tm.tailMap("nz"));

}

}

O/P

{chn=CHINA, ind=INDIA, nz=NEWZEALAND, usa=UNITED STATES OF AMERICA}

{usa=UNITED STATES OF AMERICA, nz=NEWZEALAND, ind=INDIA, chn=CHINA}

{nz=NEWZEALAND, usa=UNITED STATES OF AMERICA}

Note:

Descending map will sort in desending order of the keys, whereas Tailmap will display the all the keys and its values from the key given in the paranthesis.