**ArrayList:**

class ArrayList extends AbstractList implements List

the collection classes stores only objects but we are passing primitives these primitives are automatically converts into objects is called autoboxing.

1) Introduced in 1.2 version.

2) ArrayList supports dynamic array that can be grow as needed.it can dynamically increase and

decrease the size.

3) Duplicate objects are allowed.

4) Null insertion is possible.

5) Heterogeneous objects are allowed.

6) The under laying data structure is growable array.

7) Insertion order is preserved.

Ex:-

import java.util.\*;

class ArrayListDemo

{

public static void main(String[] args)

{

//creation of ArrayList

ArrayList al=new ArrayList();

System.out.println("initial size of the arraylist:"+al.size());

//adding elements to the ArrayList

al.add("a");

al.add("A");

al.add("a");

al.add(null);

al.add(10);

al.add(1,"ratan");

//print the ArrayList elements

System.out.println(al);

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

//remove the elements of ArrayList

al.remove("a");

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

System.out.println(al);

}

}

Ex:- ArrayList with generics

import java.util.\*;

class ArrayListDemo

{

public static void main(String[] args)

{

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

al.add(10);

al.add(20);

al.add(30);

al.add(40);

System.out.println(al);

Integer[] a=new Integer[al.size()];

al.toArray(a);

int sum=0;

for (Integer a1:a)

{

sum=sum+a1;

System.out.println(a1);

}

System.out.println(sum);

}

}

**LinkedList**

Class LinkedList extends AbstractSequentialList implements List,Deque,Queue

1) Introduced in 1.2 v

2) Duplicate objects are allowed

3) Null insertion is possible

4) Heterogeneous objects are allowed

5) The under laying data structure is double linked list.

6) Insertion order is preserved.

Ex:-LinkedList with generics.

import java.util.\*;

class Test

{

public static void main(String[] args)

{

LinkedList<String> l=new LinkedList<String>();

l.add("a");

l.add("ratan");

l.add("anu");

l.add("aaa");

System.out.println(l);

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

}

}

Ex:-

import java.util.\*;

class Demo

{

public static void main(String[] args)

{

LinkedList ll=new LinkedList();

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

//add the elements to the LinkedList

ll.add("a");

ll.add(10);

ll.add(10.6);

ll.addFirst("ratan");

ll.addLast("anu");

System.out.println("original content :"+ll);

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

//remove elements from LinkedList

ll.remove(10.6);

ll.remove(0);

System.out.println("after deletion content :"+ll);

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

//remove first and last elements

ll.removeFirst();

ll.removeLast();

System.out.println("ll after deletion of first and last :"+ll);

//get and set a value

int a=(Integer)ll.get(0);

ll.set(0,a+"ratan");

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

}

}

**Stack:- (legacy class introduced in 1.0 version)**

**1) It is a child class of vector**

**2) Introduce in 1.0 v legacy class**

**3) It is designed for LIFO(last in fist order )**

**Ex:-**

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

**class Test**

**{**

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

**{**

**Stack s=new Stack();**

**s.push("A");**

**s.push(10);**

**s.push("aaa");**

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

**s.pop();**

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

**System.out.println(s.search("A"));**

**}**

**}**

Cursors:-

The main purpose of the constructors is to retrieve the data from the collection objects.

There are three types of cursors present in the java language.

1. Enumaration

2. Iterator

3. ListIteator

**Enumeration:-**

**1. It is used for only legacy classes (Vector,Stack)**

**2. Based on above reason the enumeration cursor is not a universal cursor**

**3. By using this cursor it is possible to read the data only it not possible to update the data and not possible to delete the data.**

**4. By using elements method we are getting enumeration object.**

Ex:-

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

**class Test**

**{**

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

**{**

**Vector v=new Vector();**

**for (int i=0;i<10 ;i++ )**

**{**

**v.addElement(i);**

**}**

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

**Enumeration e=v.elements();**

**while (e.hasMoreElements())**

**{**

**Integer i=(Integer)e.nextElement();**

**if (i%2==0)**

**{**

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

**}**

**}**

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

**}**

**}**

**Iterator:-**

1. it is universal cursor we can apply any type of collection class.

2. By using this it is possible to read the data and remove the data.

3. We can use iterator() method to get the iterator object.

import java.util.\*;

class Test

{

public static void main(String[] args)

{

Vector v=new Vector();

for (int i=0;i<10 ;i++ )

{

v.addElement(i);

}

System.out.println(v);

Iterator itr=v.iterator();

while (itr.hasNext())

{

Integer i=(Integer)itr.next();

if (i%2==0)

{

System.out.println(i);

}

else

itr.remove();

}

System.out.println(v);

}

}

**ListIterator:-**

1. It is applicable for only list type of objects.

2. By using this it is possible to read the data upate the data and delete data also.

3. By using listIterator() method we are getting LIstIterator object

Ex:-

import java.util.\*;

class Test

{

public static void main(String[] args)

{

Vector v=new Vector();

for (int i=0;i<10 ;i++ )

{

v.addElement(i);

}

System.out.println(v);

ListIterator litr=v.listIterator();

while (litr.hasNext())

{

Integer i=(Integer)litr.next();

if (i==0)

{

litr.add("veeru");

}

if (i==5)

{

litr.set("sambha");

}

if (i==9)

{

litr.remove();

}

}

System.out.println(v);

}

}

HashSet:-

1. Introduced in 1.2 v

2. Duplicate objects are not allowed if we are trying to insert duplicate values then we won’t get

any compilation errors an won’t get any Execution errors simply add method return false.

3. Null insertion is possible

4. Heterogeneous objects are allowed

5. The under laying data structure is hashTable.

6. Insertion order is not preserved.

import java.util.\*;

class Test

{

public static void main(String[] args)

{

HashSet h=new HashSet();

h.add("a");

h.add("a");

h.add("aaaa");

h.add(10);

h.add(null);

System.out.println(h);

}

}

LinkedHashSet:-

1. Introduced in 1.4 v

2. Duplicate objects are not allowed if we are trying to insert duplicate values then we wont get

any compilation errors an won’t get any Execution errors simply add method return false.

3. Null insertion is possible

4. Heterogeneous objects are allowed

5. The under laying data structure is linkedList & hashTable.

6. Insertion order is preserved.

7. It is a child class of HashSet.

import java.util.\*;

class Test

{

public static void main(String[] args)

{

LinkedHashSet h=new LinkedHashSet();

h.add("a");

h.add("a");

h.add("aaaa");

h.add(10);

h.add(null);

System.out.println(h);

}

}

TreeSet:-

1. The underlying data Structure is BalencedTree.

2. Insertion order is not preserved it is based some sorting order.

3. Hetrogeneous data is not allowed.

4. Duplicate objects are not allowed

5. Null insertion is possible only once.

import java.util.\*;

class Test

{

public static void main(String[] args)

{

TreeSet t=new TreeSet();

t.add(50);

t.add(20);

t.add(40);

t.add(10);

t.add(30);

System.out.println(t);

SortedSet s1=t.headSet(50);

System.out.println(s1);//[10,20,30,40]

SortedSet s2=t.tailSet(30);

System.out.println(s2);//[30,40,50]

SortedSet s3=t.subSet(20,50);

System.out.println(s3);//[20,30,40]

}

}

**Map:-**

**1. Map is a child interface of collection.**

**2. Up to know we are working with single object and single value where as in the map collections we are working with two objects and two elements.**

**3. The main purpose of the collection is to compare the key value pairs and to perform necessary operation.**

**4. The key and value pairs we can call it as map Entry.**

**5. Both keys and values are objects only.**

**6. In entire collection keys can’t be duplicated but values can be duplicate.**

**HashMap**

**1. It used to hold key value pairs**

**2. Underlying data Structure is HashTable and HashSet**

**3. Duplicate keys are not allowed but values can be duplicated.**

**4. Insertion order is not preserved.**

**5. Null is allowed for key (only once)and allows for values any number of times.**

**6. Every method is non-synchronized so multiple Threads are operate at a time hence permanence is high.**

import java.util.\*;

class Test

{

public static void main(String[] args)

{

HashMap h=new HashMap();

h.put("sambha",100);

h.put("veeru",100);

h.put("durga",100);

System.out.println(h);

Set s=h.keySet();

System.out.println(s);

Collection c=h.values();

System.out.println(c);

Set s1=h.entrySet();

System.out.println(s1);

Iterator itr=s1.iterator();

while (itr.hasNext())

{

Map.Entry m1=(Map.Entry)itr.next();

System.out.println(m1.getKey()+"------"+m1.getValue());

if (m1.getKey().equals("sambha"))

{

m1.setValue("gayan TeamLead");

}

}

System.out.println(s1);

}

}

**TreeMap**

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.

|  |  |
| --- | --- |
| Method | Description |
| boolean containsKey(Object key) | It is used to return true if this map contains a mapping for the specified key. |
| boolean containsValue(Object value) | It is used to return true if this map maps one or more keys to the specified value. |
| Object firstKey() | It is used to return the first (lowest) key currently in this sorted map. |
| Object get(Object key) | It is used to return the value to which this map maps the specified key. |
| Object lastKey() | It is used to return the last (highest) key currently in this sorted map. |
| Object remove(Object key) | It is used to remove the mapping for this key from this TreeMap if present. |
| void putAll(Map map) | It is used to copy all of the mappings from the specified map to this map. |
| Set entrySet() | It is used to return a set view of the mappings contained in this map. |
| int size() | It is used to return the number of key-value mappings in this map. |
| Collection values() | It is used to return a collection view of the values contained in this map. |

**import** java.util.\*;

**class** TestCollection15{

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

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

hm.put(100,"Amit");

1. hm.put(102,"Ravi");
2. hm.put(101,"Vijay");
3. hm.put(103,"Rahul");
4. **for**(Map.Entry m:hm.entrySet()){
5. System.out.println(m.getKey()+" "+m.getValue());
6. }
7. }
8. }

### Java TreeMap Example: Book

**import** java.util.\*;

1. **class** Book {
2. **int** id;
3. String name,author,publisher;
4. **int** quantity;
5. **public** Book(**int** id, String name, String author, String publisher, **int** quantity) {
6. **this**.id = id;
7. **this**.name = name;
8. **this**.author = author;
9. **this**.publisher = publisher;
10. **this**.quantity = quantity;
11. }
12. }
13. **public** **class** MapExample {
14. **public** **static** **void** main(String[] args) {
15. //Creating map of Books
16. Map<Integer,Book> map=**new** TreeMap<Integer,Book>();
17. //Creating Books
18. Book b1=**new** Book(101,"Let us C","Yashwant Kanetkar","BPB",8);
19. Book b2=**new** Book(102,"Data Communications & Networking","Forouzan","Mc Graw Hill",4);
20. Book b3=**new** Book(103,"Operating System","Galvin","Wiley",6);
21. //Adding Books to map
22. map.put(2,b2);
23. map.put(1,b1);
24. map.put(3,b3);
26. //Traversing map
27. **for**(Map.Entry<Integer, Book> entry:map.entrySet()){
28. **int** key=entry.getKey();
29. Book b=entry.getValue();
30. System.out.println(key+" Details:");
31. System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);
32. }
33. }
34. }

**HashTable**

1. It is a legacy class introduced in the 1.0 version.

2. Every method is synchronized hence only one thread is allow to access.

3. The performance of the application is low.

4. Null insertion is not possible if we are trying to insert null values we are getting

NullPointerException.

import java.util.\*;

class Test

{

public static void main(String[] args)

{

Hashtable h=new Hashtable();

h.put("hyd",100);

h.put("bang",200);

h.put("pune",300);

System.out.println(h);

System.out.println(h.contains(300));//true

System.out.println(h.containsValue(500));//false

Collection c=h.values();

System.out.println(c);

Set c1=h.keySet();

System.out.println(c1);

}

}

**LinkedHashMap**

1. It used to hold key value pairs

2. Underlying data Structure is HashTable & LinkedList.

3. Duplicate keys are not allowed but values can be duplicated.

4. Insertion order is preserved

import java.util.\*;

class Test

{

public static void main(String[] args)

{

LinkedHashMap h=new LinkedHashMap();

h.put("sambha",100);

h.put("veeru",100);

h.put("durga",100);

System.out.println(h);

Set s=h.keySet();

System.out.println(s);

Collection c=h.values();

System.out.println(c);

Set s1=h.entrySet();

System.out.println(s1);

Iterator itr=s1.iterator();

while (itr.hasNext())

{

Map.Entry m1=(Map.Entry)itr.next();

System.out.println(m1.getKey()+"------"+m1.getValue());

if (m1.getKey().equals("sambha"))

{

m1.setValue("gayan TeamLead");

}

}

System.out.println(s1);

}

}

**Comparable Interface**

Java Comparable interface is used to order the objects of user-defined class.This interface is found in java.lang package and contains only one method named compareTo(Object). It provide single sorting sequence only i.e. you can sort the elements on based on single data member only. For example it may be rollno, name, age or anything else.

c**lass** Student **implements** Comparable<Student>

{

**int** rollno;

String name;

**int** age;

Student(**int** rollno,String name,**int** age){

**this**.rollno=rollno;

**this**.name=name;

**this**.age=age;

}

**public** **int** compareTo(Student st){

**if**(age==st.age)

**return** 0;

**else** **if**(age>st.age)

**return** 1;

**else**

**return** -1;

}

}

**Queue**

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.

|  |  |
| --- | --- |
| Method | Description |
| boolean add(object) | It is used to insert the specified element into this queue and return true upon success. |
| boolean offer(object) | It is used to insert the specified element into this queue. |
| Object remove() | It is used to retrieves and removes the head of this queue. |
| Object poll() | It is used to retrieves and removes the head of this queue, or returns null if this queue is empty. |
| Object element() | It is used to retrieves, but does not remove, the head of this queue. |
| Object peek() | It is used to retrieves, but does not remove, the head of this queue, or returns null if this queue is empty. |

**Import** java.util.\*;

**class** Book **implements** Comparable<Book>{

**int** id;

String name,author,publisher;

**int** quantity;

**public** Book(**int** id, String name, String author, String publisher, **int** quantity) {

**this**.id =id;

**this**.name = name;

**this**.author = author;

**this**.publisher = publisher;

**this**.quantity = quantity;

}

**public** **int** compareTo(Book b) {

**if**(id>b.id){

**return** 1;

}**else** **if**(id<b.id){

**return** -1;

}**else**{

**return** 0;

}

}

}

**public** **class** LinkedListExample {

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

Queue<Book> queue=**new** PriorityQueue<Book>();

//Creating Books

Book b1=**new** Book(121,"Let us C","Yashwant Kanetkar","BPB",8);

Book b2=**new** Book(233,"Operating System","Galvin","Wiley",6);

Book b3=**new** Book(101,"Data Communications & Networking","Forouzan","Mc Graw Hill",4);

//Adding Books to the queue

queue.add(b1);

queue.add(b2);

queue.add(b3);

System.out.println("Traversing the queue elements:");

//Traversing queue elements

**for**(Book b:queue){

System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);

}

queue.remove();

System.out.println("After removing one book record:");

**for**(Book b:queue){

System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);

}

}

}