A.I.S.S.M.S
INSTITUTE OF INFORMATION TECHNOLOGY
Kennedy Road, Near R.T.O., Pune - 411 001.

✱ Collections

Arrays:-
limitation → (1) only fixed size
(2) Homogenous in nature (same type of elements)
(3) No inline implimentation for DS (i.e not method
• • we need to add explicit method code)

But in collection ⇒
1] we can increase & decarase the size
2] Homogenous or Hetrogenous elements
3] Inline support for DS, methods like (contains())

Array	Collection
1] fixed in size	1] variable in size
2] wrt memory ✗	2] wrt memory ✓ (if 1000 array spce & only 2 element)
3] wrt performa ✓	3] wrt performance ✗
4] Homogenous ✗	4] Homogenous & Hetrogenous. only Object Integer[]
5] Primitives ✓ int[] objects ✓ Integer[]	

Collection :-
⇒ It is group of object of single Entity.
(i.e group of books)

Collection framework :-
⇒ to represent the grp of object as a single Entity needs the several classes & interfaces, ;

Java	C++
collection	container
collection	STL (Standard Template Library)

⇒ 9 key interfaces of collection framework:-

1] Collection
2] List
3] Set
4] Map
5] NavigableSet
6] SortedMap
7] SortedSet
8] NavigableMap
9] Queue

1] Collection (Interface)

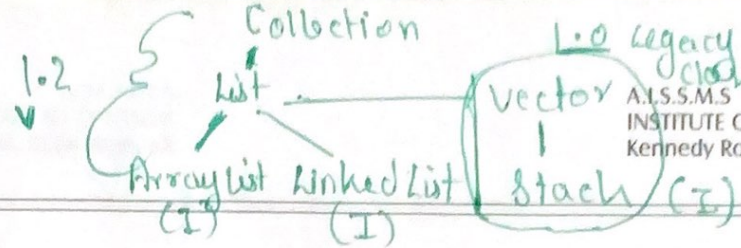
⇒ if we want Represent a group of Individual obj as a single Entity, go for collection.
→ Collection defines the most common methods which are applicable for any collection obj.
⇒ in general it is considered as root interface of collection framework.
→ there is no concreate class which implements collection interface directly.

Collection	Collections (i
1] Interface	1] Class
2] to represent grp of obj as single entity	2] is an utility class that is present in util package, to define several utility methods for collection obj (Sorting, Searching)

(i.e to sort we use Collections.sort() — class, method)

2] List (I)

→ child of Collection, if want represent group of individual obj as a single Entity, where duplicate is allowed and insertion order is preserved.



3] Set (I)

⇒ If we dont want duplicates, & no insertion order is required.

Collection
↓
Set (1.2) (I) → HashSet (1.2v) (I)
↓
LinkedHashSet (1.4v) (I)
↓
1.2 SortedSet (I) → NavigableSet (I) → TreeSet (I)
1.6 1.2

4] ⇒ for sorted order, we go for SortedSet.

5] ⇒ if we want last, next element then go for NavigableSet (I)

List	Set
1] duplicates	1] No duplicate
2] Insertion order preserved	2] Insertion order not preserved.
3]	3]

6] Queue ⇒

⇒ when we want prior to processing, the grp of indivisual objects

⇒ their is also Priority queue.

Ex. before sending mail, all mail id have to store in DS, in wich order we added mail id's in same order only mail should be delivered.

⇒ Above all collection is meant for representing a group of indivisual objects.

⇒ for key-value pairs then we should use Map

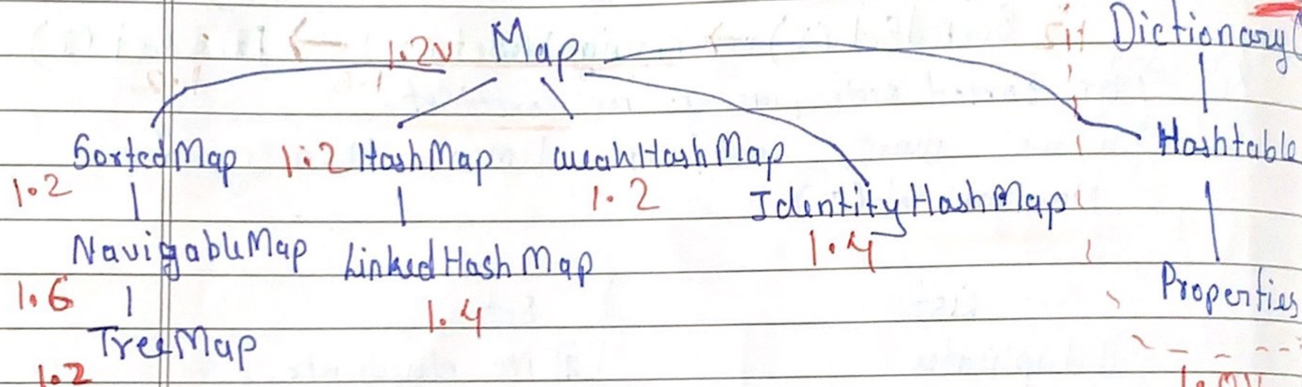
7] ⇒ MAP

★ MAP is not child Interface of collection.

⇒ key has to be unique, value can be duplicate.

⇒ both key & value are objects only.

key	value
101	Mayur
102	Sakshan
103	Chiku

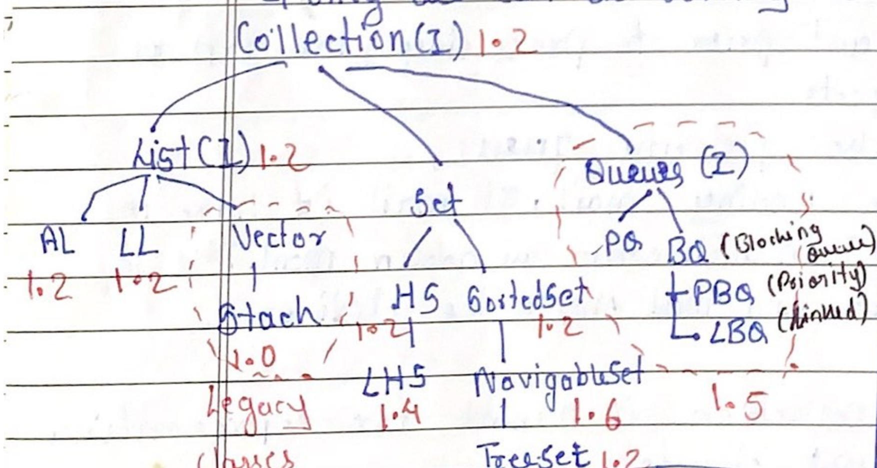


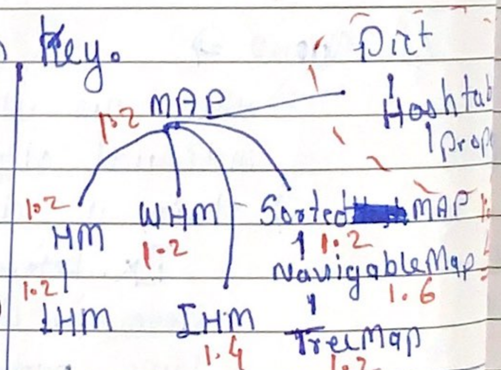
8] Sorted Map :-

⇒ child Interface of Map

⇒ if we want to represent the key in sorting order, then use SortedMap.

⇒ only we can do sorting on key.





Cursors 1.0 (1.0)
1] Enumeration (I)
2] Iterator (I)
3] List Iterator (I)

utility classes
① collection
② Arrays

Sorting ⇒ 1] comparable (I) - Default sorting
2] comparator (I) - Customized sorting

Collection (3)

A.I.S.S.M.S
INSTITUTE OF INFORMATION TECHNOLOGY
Kennedy Road, Near R.T.O., Pune - 411 001.

Collection (I)

Methods :-

boolean add(object o) => to add single object
boolean addAll(Collection c) => to add all object in part
boolean remove(object o) => to remove single obj
boolean removeAll(Collection c) => to remove all obj's
boolean retainAll(Collection c) => to remove all except in c
void clear() => to remove all obj in collection
boolean contains(object o) => if a obj contains
boolean containsAll(Collection c) => group of obj contains or not
boolean isEmpty() => if not return size
int size() => size of Collection
object[] toArray(); => to convert to Array
Iterator iterator() => to parse the collection.
(returns iterator obj) (traverse)

* there is no concrete class which implements Collection (I) interface directly.

* List() => duplicates allowed, & insertation order is preserved
-> Insertion order is preserved w.r.t index, and differentiate duplicate obj using Index.
-> idex plays imp part in list.

Methods :-

1] void add(int index, object o) // at the index add the obj
2] boolean addAll(int index, Collection c) // from index add the obj
3] Object get(int index) // the index of object present location
4] Object remove(int index) // remove the obj
5] object set(int index, object new) // replace the obj with new obj
6] int indexOf(object o) // 8] ListIterator listIterator();
7] int lastIndexOf(object o)

ArrayList

1) Resizable or growable Array
2) Duplicates Allowed
3) Insertion order preserved
4) Hetrogeneous objects allowed (except Treeset)
5) Null Insertion

Constructor :-

(1) ArrayList l=new ArrayList(); //create new obj with capacity 10. default
=> once max capacity is reached,
new capacity = (current cap x 3/2) + 1;

(2) ArrayList l=new ArrayList(int intialcapacity);

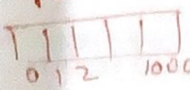
(3) ArrayList l=new ArrayList(Collection c);
=> Create an equivalent arraylist obj for given
=> let it be LinkedSet, Vector, Set,

Imp
=> Collection by default implements Serializable, Cloneable
=> RandomAccess Interface is also (Arraylist, Vector) only
to access any random elements
=> RandomAccess present in util package and does not
any methods, it is marker interface, and ability is
by JVM.
=> Best for retrival, but worst for add, modification
remove.

=> Comparator (I) - Customized

A.I.S.S.M.S
INSTITUTE OF INFORMATION TECHNOLOGY
Kennedy Road, Near R.T.O., Pune - 411 001.

For insertion and deletion of random we



→ for one insert several shift is required

ArrayList	Vector
1] Non-Synchronized	1] Synchronized
2] Not Thread-safe	2] Threadsafe
3] Performance High	3] Low performance
4] 1.2 v	4] 1.0 Legacy

Quest →

Arraylist is non synchronized, but we can Achive it by

AL l=new AL();

= List l1 = Collections.synchronizedList(l);

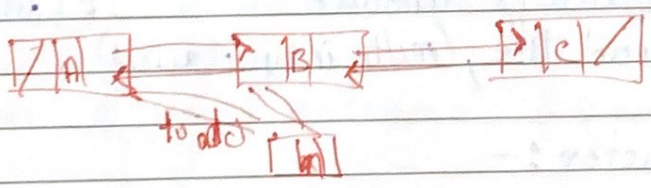
↳ Synchronized. ↳ Non-Synchronized

Method ⇒ Public static List synchronizedList(List); collections class

* we can also get synchronized for Set & Map for SynSet, SynMap

* for insertation and deletion linked List is best

becaus e.



⇒ But for retrival it is worst.

Empty obj

Constructor :- ① LinkedList l=new LinkedList(); ↗

② LinkedList l=new LinkedList(Collection c);

creates an c obj

Methods :- to develop Stack & queue (to provide support it has methods) LinkedList is used.

1) void addFirst(object o)
2) void addLast(object o)
3) object getFirst()
4) Object getLast()
5) Object removeFirst()
6) Object removeLast()

A.I.S.S.M.S
INSTITUTE OF INFORMATION TECHNOLOGY
Kennedy Road, Near R.T.O., Pune - 411 001

Arraylist	LinkedList
1] Best for Retrival	1] insertion/deletion
2] Relant for insertion/deletion (Shift operation is performed)	2] worst for retrival
3] element stored in consecutive memory Loc, & retrival is easy	3] in Linked List the elem is not stored in consecutiv memory Location, So retriva is Hard.

Vector :-

1] Resizable/Growable Array
2] Insertition order preserved
3] Duplicates allowed
4] Null insertion possib
5] Hetrogeneous obj
6] Implements Seriz, Cloneable, Rand Acc
7] Thread safe, (meth: is synchronized)

id
C 2
B 1
A 0

⇒ Constructor :-

1] Vector v=new Vector(); //Empty vector obj
Default capacity is 10, and after max it doubles
2] Vector v=new Vector(int intialcaptiy); //Size
3] Vector v=new Vector(int intial, int incremental);
after max, we want to add some Sepecified elem
3] Vector v=new Vector(Collection c);

Methods

addElement(Object o)	int size()
remove Element(Object o)	int capacity()
remove ElementAt(int index)	Enumeration elements()
removeAllElements()	
Object elementAt(int index)	
Object firstElement()	
Object lastElement()	

} Vector

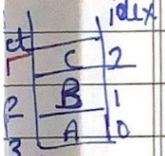
Create

public Object public

A.I.S.S.M.S
INSTITUTE OF INFORMATION TECHNOLOGY
Kennedy Road, Near R.T.O., Pune - 411 001.

★ Stack :- child of Vector
=> LIFO
Constructor => Stack s=new Stack();

Methods =>
1] Object push(Object o)
to insert an object into the stack
2] Object pop()
=> To remove the return top of stack
3] Object peak()
to return top of Stack without removal
4] boolean empty()
returns true if the stack is empty.
5] int search(Object o)
returns offset if the element is available otherwise
returns -1,



★ Cursors :-
1] Enumeration
2] Iterator
3] ListIterator.

1] Enumeration => using Legacy Collection obj to get obj one by one

Create Obj => Enumeration e=v.elements();

it has two methods.
public boolean e.hasMoreElements();
Object e.nextElement();
public

Limitation;
1] We can Apply only to Legacy classes Stack, Vector,
2] We have only read access can't perform remove operation

★ Iterator :- (Universal Access to All collection)
2] we can do read / remove operation.

⇒ we can create obj using iterator method of collection interface.

⇒ Public Iterator iterator()
Iterator itr = c.iterator();
↓ Any collection obj

★ Methods() :-
1) Public boolean hasNext()
2] Public object next()
3] Public void remove()

★ Limitations :- 1) only move only in forward direction
2] only Read / Remove operation, & can't perform Replace & Add
⇒ to overcome above limitations go for List iterator

★ ListIterator :- (only for List objects / not universal)
1] we can move to frwd / backward direction, bidirectional
2] can perform Read / Remove / Replace / Add operations

Iterator
↓
ListIterator
child ⇒ Public ListIterator listIterator()
All mtds avail ListIterator itr = l.listIterator();
↓ Any list object

Forward movement	Backward movement	Extra Operations
1) Public boolean hasNext()	1) Public boolean hasPrevious	1) Public void remove()
2] Public Object next()	2] Public Object previous()	2] Public void add(Obje
3] Public int nextIndex()	3] Public int previousIndex()	3) Public void set(Obje

We get the Implimented class objects for demo, which are Anoymous class object

A.I.S.S.M.S
INSTITUTE OF INFORMATION TECHNOLOGY
Kennedy Road, Near R.T.O., Pune - 411 001.

iperty	Enumeration	Iterator	ListIterator
can Apply	only for Legacy classes	For Any Collection obj	only for ListObject
s legacy	yes (1.0v)	No (1.2v)	No (1.2v)
ovement	Single direction (only forward)	Single Dire (only forward)	BI-Directional (Fwd) bcs
owed operatio	only Read	Read / Remove	Read / Replace / Remove / add
ue to get	By using elements() of vector class	By using iterator() of Collection (I)	By using listIterator() of List (I)
ethods	2 mthd hasMoreElement() nextElement()	3 methods hasNext() next() remove()	9 methods

★ Set :-
⇒ Child of Collection (I), where duplicates are not allowed & insertion order not allowed.
⇒ All Collections methods are only used in Set, (9 methods)

1) HashSet :-
1) UnderLine Data Structure is Hashtable
2) Duplicate not allowed
3) Insertion order not preserved
4) Based on hashCode of Object
5) Null (only once allowed to insert)
6) Heterogeneous object allowed
7) Impelement Serizable, Clonable Interface (but no RandomAccess)
8) Search (where search is frequent best to go)

In hashset duplicate not Allowed if we are trying to insert, then no error (CE, RE), it returns false

① HashSet h=new HashSet();
Sop (h.add("A")); // true
Sop (h.add("A")); // false

Default capacity (16), fill ratio => 0.75
=> New object will be created, after completing 75%.

② HashSet h=new HashSet(int intialCapacity);
③ HashSet h=new HashSet(int intial, float fillratio);
④ HashSet h=new HashSet(Collection c);

LinkedHashSet:- (LinkedList + HashTable)
it does not allow the duplicate but preserves the insertation order.

it is more often used in cache memory Applications

SortedSet:-
-> if we want to represent the group of objects in some order, but duplicates not allowed.

=> methods:- [100, 101, 104, 106, 108, 110, 115, 120]

1] first(): - 100
2] last(): - 120
3] headSet(106): - (106<) 100, 101, 104
4] tailSet(106): - (106>=) 106, 108, 110, 115, 120
5] subSet(101, 115): - return element >= obj and element < obj 2
6] comparator(): - default sorting order (Ascending for number, Alphabetical for String) and

we should typecast the StringBuffer obj to String() obj if wrapper class use String s=(String)obj;

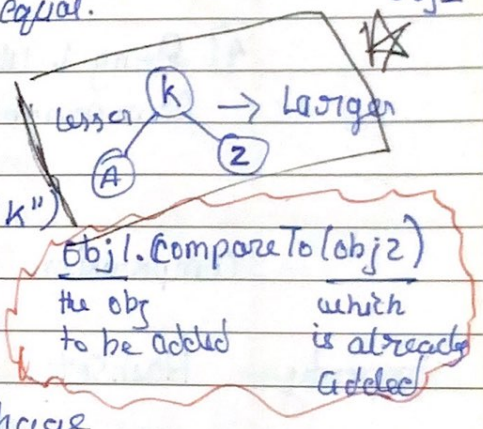
TreeSet:- We cannot add null from 1.7v, & only homogeneous elements are Allowed, we give. (new StringBuffer()); // class cast Exe
does not implement Comparable interface

A.I.S.S.M.S
INSTITUTE OF INFORMATION TECHNOLOGY
Kennedy Road, Near R.T.O., Pune - 411 001.

⇒ Comparable we for default natural sorting order
⇒ Comparator meant for customizable sorting order

★ if default sorting ordere we dont wantf, then we can use Comparator

★ Comparable (I) :- it is present in Java.lang. pacahage
→ and it contains only 1 method, compareTo()
- public int compareTo();
  - -ve if obj1 comes before obj2
  - +ve if obj1 comes after obj2 // obj2 come before obj1
  - 0 if obj1 & obj2 are equal.



⇒ TreeSet T=new TreeSet();
t.add("K");
t.add("Z"); +ve ("Z").compareTo("K")
t.add("A"); -ve
t.add("A"); 0

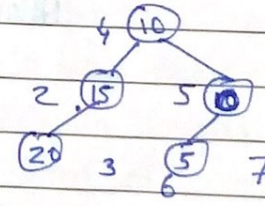
' Should come after K // come before K // Not added

★ Comparator (I) :- In Java.util Pachage
⇒ Has 2 methods 1] compare()
2] equals()

1) public int compare(Object obj1, Object obj2)
→ -ve if obj1 has to come before obj2
→ +ve if obj1 has to come after obj2
→ 0 if both are equal

2) Public boolean equals(object obj)
→ whenever we are implimenting comparator interface we should provide implementation for compare method but not equals(), because it availble for our class through Object class.

t.add(10); ✓
t.add(0); ⇒ +ve compare(0,10)
t.add(15); ⇒ (15,10)
t.add(5) ⇒ (5,10) (5,0) root
t.add(20) ⇒ (20,10) (20,15)



Print order is
1 2 3 4 5 5 7, left, root, right

★ for possible comparets at last page methods are written

★ Difference between Comparable & Comparator

Comparable	Comparator
1) Default Natural Sorting order	1] Customize Sorting
2] Java.lang	2] Java.util package
3] One method (i) compareTo()	3] 2 mthds :- compare(); equals();
4] String & Wrapper implement Comparable	4] the only implemented classes are Collator, RuleBasedCollator

Comparision for Set implemented classes (Gui classes)

Property	HashSet	LinkedHashSet	TreeSet
Underline DS	Hashtable	LL + Hashtable	Balanced Tree
Duplicate Objects	Not Allowed	No	No
Insertion Order	Not preserved	preserved	Not preserved
Sorting Order	N.A	N.A	Applicable
Hetrogenous	Allowed	Allowed	Not Allowed
Null Accepta	Allowed	Allowed	For empty trees as 1st element only till 1.6v aft 1.7 it is not Allo

Map → Hashmap → LinkedHashmap; IdentityHashmap; WeakHashMap; SortedMap → NavigableMap → TreeMap

Collection → Hastable → Dictionary → Properties

★ Map (I)

① Map is not child Interface of Collection

② If we want to Represent the Objects as (key, value) pair

[101 | Mayur] [102 | chiku] [103 | ABC]

→ Both are objects only, key cannot be duplicate, but values can be duplicate, each key, value is called entry

→ Map is considered as 'Entry' Object

★ Methods :-

1) Object put(Object key, Object Value)

m.put(101, "Mayur"); ⇒ return null if ~~already present~~

if key is already present then old value will be replaced with new value, & returns old value.

m.put(101, "may"); ⇒ return old value ~~durga~~ Mayur is replaced with may.

2] void putAll(Map m)

→ A group of (key, values) to be added from another Map

3] Object get(Object key)

4] object ~~get~~ remove(object key)

5] boolean containsKey(Object key)

6] boolean containsValue(Object ~~key~~ value)

7) boolean isEmpty()

8] int size()

9] void clear() ⇒ to remove all key, value pair

key	value
1	A ← entry
2	B
3	C

★ Collection Views of Map :-

1] Set keySet() → only keys, and set do not allow duplicate so set is type

2] collection values() → only values

3] Set entrySet() → Set of entries key - value, key - value

Entry

★ Map is a group of key value pair & each key value pair is called, an entry
Hence, Map is considered as collection of entry object
→ without existing map object there is no chance of existing Entry object,
→ So Entry Interface is inner ~~class~~ Interface of Interface Map

```
Interface Entry
{
    Object getKey()
    Object getValue()      } only Applicable on Entry object
    Object setValue(Object newO)
}
```

★ HashMap :-
1] underline data structure is Hash table
2] hash code of keys
3] Duplicate key not allowed
4] Duplicate values allowed
5] Heterogeneous objects allowed
6] null — key only once — but null values any times
7] Serializable & clonable & but not Not Random Access
8] It best to use for, if frequent operation is search

★ Constructor → 1) HashMap m=new HashMap(); default capacity = 16, fillration = 0.75
2) HashMap m=new HashMap(intial capacity);
3) HashMap m=new HashMap(Map m); (int intial, float fillrati

A.I.S.S.M.S
INSTITUTE OF INFORMATION TECHNOLOGY
Kennedy Road, Near R.T.O., Pune - 411 001.

HashMap	Hashtable
1) Non Synchronized	1) Synchronized
2) HM → t1, t2	2) Ht → t1
3) Performance is High	3) Performance is slow
4) Null key, value	4) No null key / value
5) 1.2 V	5) 1.0 V

To get the Synchronized HashMap, HashMap m=new HashMap()
Collection C=Collections.synchronizedMap(m);
Synchronized — Non Synchronized

⇒ if we take LinkedHashMap m=new LinkedHashMap();
//then insertion order is preserved as it is LL+Ht datastruct
to develop cache based Application

⇒ ★ In HashMap Jvm uses ⇒ equals() = for comparision of content
★ In Identity HashMap uses ⇒ (==) for comparision (Address comparision)
★ WeakHashMap is same as HashMap → in HashMap if there is a reference, it is not eligible for gc, if it is associated with HashMap, it dominates gc,
→ but WeakHashMap, if object does not contain any reference it is eligible for gc, (gc calls the finalize() to perform cleanup Activity)

★ SortedMap :- Child Interface of Map if we want to represent the values, key-values according to sorting order of keys.
(keys only)
→ 6 methods 1st entry key, last entry key
1) Object FirstKey(), Object LastKey()
2) SortedMap headMap(Object key) < key
3) SortedMap tailMap(Object key) >= key
4) SortedMap subMap(object key1, Obj key2) key1 >=, < key2
5) Comparator comparator() ⇒ Null, as NO

Comparable → compareTo();
Comparator → compare();
equals();

☆ **TreeMap() :-** 1] Data Structure is Red-BLACK tree
2] Insertion order not preserved, sorting order of keys
3] Duplicate key not Allowed, Values allowed
4] Hetrogenous, ⇒ if we use, comparable then Homogenous key
or if we define our own order, then Hetrogenous allowed
or RE: ClassCastException.
5] Till 1.6 we can add Null for empty TreeMap
but from 1.7 null Insertation was allowed.
→ for non-empty we cannot add null we will ge RE: NPE (null Pointer)

Constructors :- 1] TreeMap t=new TreeMap(); - Def NSO
2] TreeMap t=new TreeMap(Comparator c); → customize SO
3] TreeMap t=new TreeMap(SortedMap m);
4] TreeMap t=new TreeMap(Map m);

☆ **HashTable :-** (Printed)
1] Underline DataStruct is Hashtable, (it will get from our hascode() value)
2] Insertion order not preserved, as Hashcode wise Sort.
3] Duplicate key not allowed, values allowed
4] Hetrogenous allowed for key & values.
5] Null not allowed for Key & values. (RE: NPE)
6] Implements serializable & cloneable / not Random Access
7] Thread Safe
8] Best for Search operation
☆ Constructors :-
1) Hashtable h=new Hashtable(); (Default capacity=11, fill ratio=0.75)
2) Hashtable h=new Hashtable(int intialcap);
3] Hashtable h=new Hashtable(int intial, float fillratio);
4] Hashtable h=new Hashtable (Map m)

☆ The element are stored in top-bottom order & Righ to left if there are multiple element at particular loction in bucket

11
5 | 5→A, 16→B
0

Properties ☆ Constructor ☆ Java Properties file; Map file — store() → abc.properties ☆

Properties :-

=> There is no need to recompile if we change the properties file, the change is directly reflected

=> If there is any Property that Holds the username & password, has to change every often, then never use it in the Source file because, it involves the following step:-

1) Recompile
2) Rebuild (ear, jar, war) } Imp
3) Redeploy
4) Restart Server

* It best to use the another file properties so that we can use these properties from the properties files in Source file, so only we need following steps:-

1) Redploy => the code

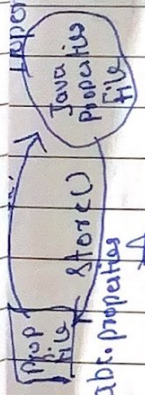
Properties

* In Normal HashMap => the key & value can be any type & but in properties => the key & value should be String

Constructor => Properties P = new Properties();

* Method():-

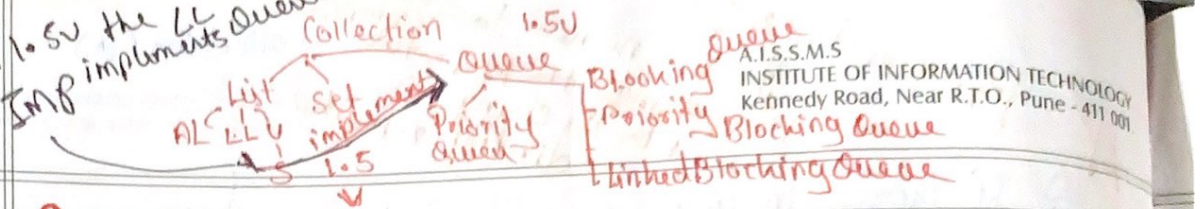
1) String setProperty(String pname, String pvalue)
   -> to set a new Property (if already present, it replace & return old value
2) String getProperty(String pname)
   -> to get associated with specified property, if not it returns (null).
3) Enumeration propertyNames() => all Property name in properties obj



* Void load(InputStream is) :
  to Load Properties from Properties file into Java Propobj

Void store(OutputStream os, String comment)
: to Store Properties from Java.properties object into properties file

from 1.5v the LL implements Queue



Queue(I):-

→ if we want to represent the group of individual obj prior to processing,
Ex. Before sending SMS, store the mobile msg in some DS, in order we added the Msg should be delivered in that order for this queue is Best choice.

→ usually queue follows (FIFO) but we can use Priority based on our requirement, and maintain our own priority order.

IMP → From 1.5v LinkedList class also implements Queue Interface

→ LL based implementation of Queue follows (FIFO)

Methods:-

1] boolean offer(Object o)
→ to add element, object into the queue

2] Object peek()
→ to return head element of queue, if queue is empty it returns null.

3] Object element()
→ to return head element of queue, If queue is empty then it ~~returns~~ raises:- RE. NoSuchElementException

4] object poll()
→ to remove and return head element of queue, if queue is empty it returns null

5] Object remove()
→ to remove and return head element of the queue if queue is empty then this method raises the RE:
NoSuchElementException

A.I.S.S.M.S
INSTITUTE OF INFORMATION TECHNOLOGY
Kennedy Road, Near R.T.O., Pune - 411 001.

(Some OS not Support the priority) same in multithreading

★ Priority Queue :-

1] if want to represent grp of individual obj prior to processing, according to some priority

2] The priority can be DNSO, defined by customize sorting order, by comparator

3] Insertion order not preserved based on priority

4] Duplicates obj not allowed

5] if we are depending on DSO, Obj should be Homogeneous & comparable or else

RE: ClassCastException

6] If we use customizable order then Heterogeneous obj allowed

7] Null not allowed as the first element also.

★ Constructors :-

1] PriorityQueue p=new PriorityQueue();
Create empty priority que, Default cap: 11, inserted according to DNSO

2] PriorityQueue q=new PriorityQueue(int initialcapacity);

3] PriorityQueue q=new PriorityQueue(int initialcap, Comparator c);

4] PriorityQueue q=new PriorityQueue(SortedSet s);

5] PriorityQueue q=new PriorityQueue(Collection c);

★ NavigableSet :- Child of SortedSet, method of navigation purposes
In 1.6V → 2 concepts → NavigableSet, NavigableMap

1] floor(e) → it returns highest element which is <=e (element)

2] lower(e) → it returns highest element which is <e (last element before the set)

3] ceiling(e) → it returns lowest element which is >=e

4] higher(e) → it returns lowest element which is >e

before either element → (10,00)
last element before either → (10,00)
After element → (10,00)

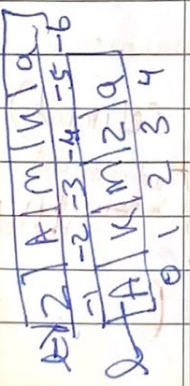
6] PollFirst(e) → remove & return first element
6] PollLast(e) → remove & return last element
7] descendingSet(e) → it returns NavigableSet in reverseOrder

★ Navigable Map (I) :-
1) floorKey(e)
2) LowerKey(e)
3) ceilingKey(e)
4) higherKey(e)
5) pollFirstEntry()
6] pollLastEntry()
7] descendingMap()

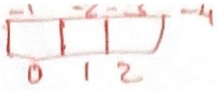
★ Collections → This utility class provides methods for Searching Sorting, reverse, etc.

1) Sorting :- Collections class defines two Sort methods
1] public static void sort(List l) → DNSO (Homogen, and Not NN NP)
2] public static void sort(List l, Comparator c); Class cast Exc
⇒

★ Searching :-
⇒ it follows binary Search methods
1) public static int binarySearch(List l, Object target);
⇒ if list is sort according to DNSO
2] Public static int binarySearch(List l, Object key, Comparator
⇒ if there is customized Sorting order
conclusion ⇒ 1) the Search method internally use binary Search Algorithm successful Search return index, unsuccessful Search returns insertion point (insertion point is a loca where we can place element in the Sorted list,)
⇒ Before calling binary Search method compulsory list shou be Sorted, or we will get unexpected result
⇒ if list is Sorted according to comparator at Search we have pass compa object or unpredictable result



A.I.S.S.M.S
INSTITUTE OF INFORMATION TECHNOLOGY
Kennedy Road, Near R.T.O., Pune - 411 001.



for n elements

Success search range = 0 to 2 → 0 to n-1
unsuccess rang = -4 to -1 → -(n+1) to -1
Total range = -4 to 2 → -(n+1) to n-1

★ Reversing elements of list :-
1) Public static void reverse(List l)
⇒ it reverse the insertion order
[15, 10, 20, 0, 5] ⇒ [5, 0, 20, 10, 15]

★ reverse() vs reverseOrder()
⇒ We use reverse to reverse the order of insertion of list but, we use reverseOrder() to get reverse comparator
Comparator c1 = Collections.reverseOrder(comparator c)
Descending order. Ascending order

★ Arrays :- it is a utility class to define several utility method

① Public static void sort(primitive[] p) (int arr, char c)
To sort according to DNSO

② Public static void sort(Object[] o) ⇒ on object array we can customize sorting order
To sort according to DNSO

③ Public static void sort(Object[] o, comparator c);
customized sorting order

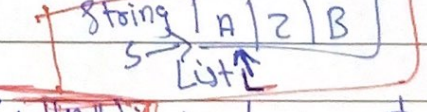
★ Searching (same as collection class)
1) Public static int binarySearch(Primitive[] p, Primitive tgt)
2) -||- -||- -||- -||-(Object[] a, object tgt);
3) -||- -||- -||- -||-(Object[] a, object tgt, comparator c);

return index if found or else insertion point

★ Conversion of Array to list
1) Public static List asList(Object[] a)

List points to same array obj

String[] s = {"A", "Z", "B"};
List l = Arrays.asList(s);
we can perform l[0] = "M"; l.set(1, "A"); but cannot perform remove(), add(), that changes the size ★



```
Public int compare(Object obj1, Object obj2)
{   Integer I1= (Integer) obj1;
    Integer I2= (Integer) obj2;
(1) return I1.compareTo(I2); // Default natural sorting
(2) return -I1.compareTo(I2); //Descending sorting order
(3) return I2.compareTo(I1); //Descending order
(4) return -I2.compareTo(I1); //Ascending order
(5) return +1; //Insertion order
(6) return -1; //Reverse of Insertion order
(7) return 0; //only 1st el added others are
                treated duplicate
```

O/p:- [10, 0, 15, 5, 20, 20]

1] 0, 5, 10, 15, 20
2] 20, 15, 10, 5, 0
3] 20, 15, 10, 5, 0
4] 0, 5, 10, 15, 20
5] 10, 0, 15, 5, 20,
6] 20, 5, 15, 0, 10
7] 10

For String we typecast to :-
String s1= (String) obj1;
For String Buffer we have to call : (toString());
String s1= obj.toString();

(1) In Array collections if we change 1st element reflects in Array & vice versa, pointing to same obj
(2) we cannot add, remove operation on it that change size
(3) cannot add Hetrogenous obj in it or get ArrayStore Exception
l.set(1, New Integer(10)); → UnsupportedOperationException