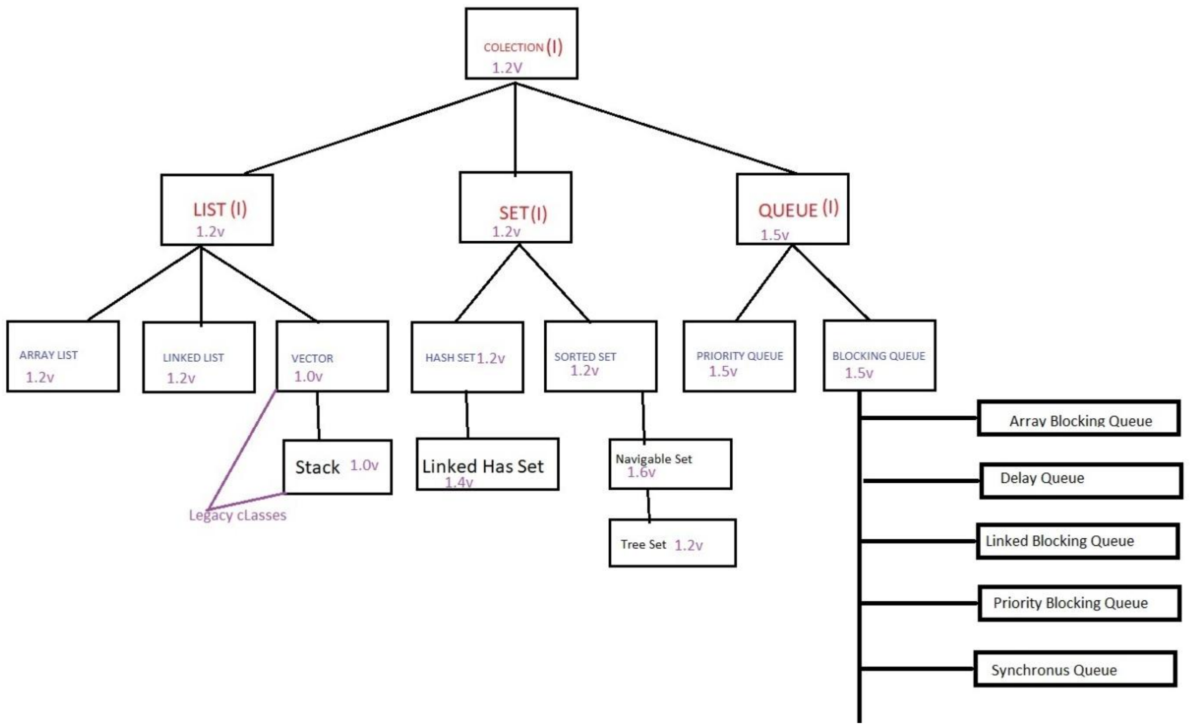
COLLECTIONS

1. Collection Interface



- Collection is an Interface which is used to represent a group of individual objects into a single entity.
- We can use collections to hold and transfer objects from one place to another place, to provide support for this requirement every collection already implements Serializable and Clonable interface.
- Every Collection class is already implemented as serializable and Clonable Interfaces.

- It is root interface for followed interfaces which are :
  i) List Interface
  ii) Set Interface
  iii) Queue Interface

- Collections concept is related to Containers concept in C++
- **Methods in Collection Interface:**

boolean **add**(Object o)	ads new object
boolean **addAll**(Collection c)	add new group of objects
boolean **remove**(Object o)	remove object
boolean **removeAll**(Collection c)	remove collection group
boolean **retainAll**(Collection c)	remove all groups except mentioned collection group
void **clear**()	clear all objects
boolean **isEmpty**()	check whether group is empty or not
int **size()**	size of object
boolean **contains(**object o)	Check whether object is available or not
boolean **containsAll** **(**Collection c)	Check whether collection group is available or not
object[] **toArray()**	convert collection group in to array
Iterator **Iterator()**	To Get the Objects in to one by one

2. Collections framework

- It defines several classes and interfaces which can be used to represent a group of individual objects in to single entity.
- It is related to STL(standard template Library) in C++

3. Collections

- Collections is a utility class present in java.util.package to define several utility methods like sorting, searching for collection object.
- EX: I have a Arraylist and I want to sort an arraylist. In collections class we have **Collections.sort(arraylist)** method is there. We can sort arraylist by using that.

4. Array vs. Collections

- Arrays are fixed in size and Collections have growable nature that means based on our requirement size will be increased and decreased.
- In memory point of view arrays are not recommended to use collections are highly recommended.
- In Performance point of view Arrays are recommended.
- Arrays are stored only Homogeneous values that mean only one data type   can be allowed. But Collections Can Store Both Homogeneous and Heterogeneous values.
- Arrays doesn`t follow any data structures that’s why arrays don’t have any predefined methods. But Collections are implemented based on some standard data structure so collections have predefined methods to perform some operations like searching, sorting.
- Arrays can hold Primitive values and Objects But Collections can hold only Objects.

5. List Interface

- If we want to represent a group of objects in to a single entity along with that allow duplicate values and maintain insertion order then we should go for List
- It uses the index for differentiate the values and maintain the insertion order.
- It is child interface of Collection Interface. So that we can use here all the methods which are available in collection interface.
- Along with that List interface have some other methods those are:

i. void **add**(int index, Object o);
ii. boolean **addAll**(int index, Collection c)
iii. Object **get**(int index)
iv. Object **remove**(int index)
v. Object **set**(int index, Object o)
vi. Int **indexOf**(Object o)
vii. int **lastIndexOf**(Object o)
viii. Listiterator **listiterator**();

- **Properties**:
  i. It allows the Duplicates.
  ii. Insertion Order is preserved
- **Classes in List interface**:
  i. ArrayList
  ii. Linked List
  iii. Vector
  iv. Stack

6. Array List

- Array List is a class which is implements in List Interface.
- Array List is implemented based on the growable array data structure.
- It allows the Duplicate Values.
- Insertion Order Is Preserved.
- It allows Heterogeneous values (Except Tree map and Treeset remaining all collections are allow the Heterogeneous values. Because Tree map and Tree set are implemented on some a sorting order. So that it doesn't allow Heterogeneous values).
- It allows the null values.
- It implements by Serializable, Clonable and Random access Interface.
- Arraylist is highly recommended for Retrieval operations.
- Arraylist is not suitable for insertion and removal of objects in the middle.
- By using arraylist we can retrieve the element at anywhere with the same speed because it implanted by Random Access interface.

CONSTRCTORS:

i. **Arraylist al = new Arraylist();** It creates an Empty array list object with default capacity of **10**. If arraylist maximum capacity reaches then it creates a new bigger array list and shifts all the values in to new arraylist. The Formula for new arraylist capacity is

New Capacity = (Current Capacity * 3/2) + 1;

ii. **Arraylist al = new Arraylist(int capacity);**
iii. **Arraylist al = new Arraylist(Collection c);** by interconnecting the array list in to any other collection.

7. Array List vs. Vector

- Array List is **a non-synchronized** but vector is **Synchronized**.
- ArrayList Is not a Thread safe because it allows multiple threads at a time, but Vector is Thread safe because it allows only one thread at a time.
- ArrayList Performance is high but vector performance is low compared to ArrayList.
- ArrayList comes in 1.2 v
- Vector comes in 1.0 v so that vector is called as Legacy class.

8. How to Synchronize Array List

- By default Array List is **a non-synchronized we can Synchronized** by using **SynchronizedList (List l)** method which is available in Collections class.

  Public static List **SynchronizedList (List l);**

  Non-synchronized arraylist
  ArrayList al = new ArrayList ();

  Synchronized arraylist
  List l = Collections.synchronizedList (al) ;

- In similar way we can get the **synchronized** versions of **Set** and **Map** also.

  Public static Set **SynchronizedSet (Map m);**

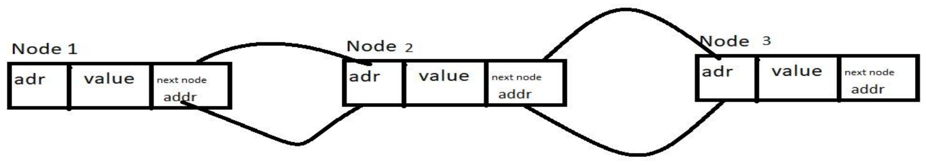
  Public static Map **SynchronizedMap (Set s);**

9. Linked List

- Linked List is a class which is implements in List Interface.
- Linked List is implemented based on the doubly linked list data structure.
- It allows the Duplicate Values.
- Insertion Order Is Preserved.
- It allows Heterogeneous values (Except Tree map and Treeset remaining all collections are allow the Heterogeneous values. Because Tree map and Tree set are implemented on some a sorting order. So that it doesn't allow Heterogeneous values).
- It allows the null values.
- It implements by Serializable and Clonable Interface But not Random access Interface.
- Linked List is highly recommended for insertion and removal of objects in the middle.
- Linked List is not suitable for Retrieval operations.

CONSTRCTORS:

i. **LinkedList ll = new LinkedList();**
ii. **LinkedList ll = new LinkedList (Collection c);** by interconnecting the array list in to any other collection.



First node Contains the address, value and next node address
Second Node Contains previous node address value and this node address......

- **Methods in Linked List:**

  Usually we are using the Linked to implement stacks and Queues. To provide support for this requirement it defines some specific methods which are applicable for only Liked list class.

void **addFirst**()	ads new object at First
void **addLast**()	ads new object at last
Object getFirst()	Get First Object
Object getLast()	Get Last Object
Object removeFirst()	Remove First Object
Object removeLast()	Remove last Object

10.ArrayList vs. LinkedList?

- ArrayList is suitable for Retrieval operations because it implements based on Random Access Interface. So that we can get the elements at same speed in any position.
- Linked List is suitable for Insertion and Deletion operations in the middle.
- Array List follows growable array data structure
- Linked List follows the doubly linked list data structure.

11.Vector

- Vector is a class which is implements in List Interface.
- Vector is implemented based on the growable array data structure.
- It allows the Duplicate Values.
- Insertion Order Is Preserved.
- It allows Heterogeneous values (Except Tree map and Treeset remaining all collections are allow the Heterogeneous values. Because Tree map and Tree set are implemented on some a sorting order. So that it doesn't allow Heterogeneous values).
- It allows the null values.
- It implements by Serializable, Clonable and Random access Interface.
- Vector is highly recommended for Retrieval operations.
- Most of the methods present in vector are synchronized hence it is thread safe.

<u>CONSTRCTORS:</u>

i. **Vector v= new Vector ();** It creates an Empty vector object with default capacity of **10**. If vector maximum capacity reaches then it creates a new bigger vector. The Formula for new vector capacity is

New Capacity = (2 * Current Capacity) ;

ii. **Vector v = new Vector (int capacity);**
iii. **Vector v = new Vector (int initial capacity, int incremental capacity);**

iv. **Vector v = new Vector (Collection c);** by interconnecting the vector in to any other collection.

- **Methods in Vector:**

void **addElement**(Object o)	ads new object
void **removeElement**(Object o)	Remove object
Object removeElementAt(int index)	Remove element at particular position
removeAllElemets()	Clear all Objects
Object elementAt(int index)	Get Object at particular position
Object firstElement()	Get First Object
Object lastElement()	Get Last Object
Int size()	Get Vector Size
Int capacity()	Get vector capacity
Enumeration elements()	Get Elements one by one

12.Stack

- Stack is a child class of Vector class.
- Stack is implemented for Last in First out Order (LIFO).

CONSTRCTORS:

Stack s= new Stack ();

Object **push()**	To insert object in to the stack
Object pop()	To **remove** and **returns Top** of the Stack
Object peak()	To R**eturns Top** of the Stack without removal
Int search()	If object is available it returns **offset** value from the **top of stack** otherwise return **-1**
Empty()	Returns true if stack is empty.

13.Cursors

- If we want to retrieve objects from collection one by one then we should go for cursors.
    i. Enumeration(I)
    ii. Iteration (I)
    iii. List Iteration (I)

14.Enumeration Interface

- If we want retrieve objects from vector one by one then we should go for Enumeration interface.
- This is a single directional cursor that means we can traverse only forward direction only.
- We can create Enumeration object by using elements () method which is available in vector class.
- This is applicable for only vector class. So that it is not a universal cursor.
- We can do read operations only we can`t do any write operations.

CREATION:

Enumeration e= vectorobjet.elements();

METHODS:

Public boolean hasMoreElements();	Returns true or false
Object nextElement()	Returns next object in group

15.Iterator Interface

- If we want retrieve objects from any collection group one by one then we should go for Iterator interface.
- This is a single directional cursor that means we can traverse only forward direction only.
- This is applicable for all collection groups. So that it is a universal cursor.
- We can do remove and read operations only.

CREATION:

Iterator I = c.iterator(); → c means any collection group

METHODS:

Public boolean hasNext();	Returns true or false
Object next ()	Returns next object in group
Public void remove ()	remove object in group

LIMITATIONS:

i. we can`t go backward direction

ii. we can`t do update add operations

16. List Iterator Interface

- If we want retrieve objects from any list group one by one then we should go for List Iterator interface.
- This is a Bi-Directional cursor that means we can traverse both forward and backward direction.
- This is applicable for all list interfaces only. So that it is not a universal cursor.
- We can do add, remove, update and read operations.

CREATION:

ListIterator I = l.iterator(); → l means any list object

METHODS:

Public boolean hasNext();	Returns true or false
Object next ()	Returns next object in group
Public void remove ()	remove object in group
Public int nextIndex();	Returns next object index value
Public int previousIndex();	Returns previous object index value
Object previous ()	Returns previous object in the group
Public void set(object new);	Update with new object
Public void add(object new);	Add new object

17. SET Interface

- Set is the child interface for collection interface
- If we want to represent a group of individual objects as a single entity, where duplicates are not allowed and insertion order is not preserved then we should go for set.

  i. **Hashset(1.2v)**
     a. **Linked Hashset(1.4v)**

  ii. **Sorted set(I)1.2v**
     a. **Navigable Set(I)1.6v**
        i. **Tree Set(1.2v)**

18. Hash Set

- Hash Set follows Hashtable data structure.
- Hashset is most suitable for search operations.
- It doesn`t allow duplicate values. If we try to add duplicate add() method will return false then object is not add to group.
- Insertion order is not preserved. All objects are stored based on the hash-code of objects.
- Null insertion is possible.
- Heterogeneous values are added.
- By default it satisfies the serializable and Clonable interface.

Constructors:

HashSet h = new HashSet () ;

→It creates an empty HashSet object with default initial capacity 16 and default fill ratio (or) load factor is 0.75.

HashSet h = new HashSet (int initial capacity) ;

→It creates an empty HashSet object with specified initial capacity and default fill ratio (or) load factor is 0.75.

HashSet h = new HashSet (int initial capacity, float load factor) ;

→It creates an empty HashSet object with specified initial capacity and specified fill ratio (or) load factor.

HashSet h = new HashSet (Collection c); by interconnecting the HashSet in to any other collection.

Load factor or Fill Ratio:

After loading the how much of factor, a new HashSet object will be created, that factor is called as Load factor or Fill Ratio.

19. Linked Hash Set

- Linked Hash Set follows Hashtable and Linked List.
- It is child class of HashSet
- Introduced in 1.4 version
- If we want to represent group of objects in to single entity, where insertion order is preserved and duplicates not allowed then we should go **for Linked HashSet.**
- Duplicates not allowed.
- Insertion order is preserved.
- Linked Hashset is the best choice for cache based applications. Because it requires insertion order and must be not allowed.

20. Sorted Set

- It is the child interface of Set interface.
- If we want to represent a group of individual objects according to some sorting order and duplicates are not allowed then we should go for Sorted Set.

Note:

i. Default natural sorting order for numbers is ascending order and strings is alphabetical order

METHODS: These methods are applicable for Sorted set and it` implemented classes.

first();	Returns first object
last()	Returns last object in group
headSet(Object o)	returns objects which are greater than specified object in group
tailSet(Object o)	returns objects which are less than specified object in group
SubSet(Object o , Object oo)	returns objects which are between specified objects in group
comparator()	null

21. Tree Set

- Tree Set is a class which is implements in Set Interface.
- Tree Set is implemented based on the Balanced Tree data structure.
- Duplicate Values Not allowed.
- Insertion Order Is Not Preserved but objects will be inserted based on some sorting order.
- It doesn`t allows Heterogeneous values. If we try to add at run time it gives class cast exception.
- It implements by Serializable, Clonable and Random access Interface.
- It allows the null value but only once.
- For Empty Tree Set First element null is possible to insert. But after inserting null if we try to add any value it gives null pointer exception at run time. Because the value will compared to null value.
- For Non-empty Tree Set if try to add null value it returns null pointer exception at run time. So we can add null for empty Tree Set at first element only, we can`t add another element after inserting the null in Tree Set.
- If we are depends on default sorting order, the objects should be homogeneous and comparable otherwise we will get run time exception says class cast exception.
- An object said to be **Comparable** when the corresponding class implements java.lang.Comparable interface only. Ex: **String** and all **wrapper** classes.
- **String Builder** is example for Not a Comparable Class.

EX: **TreeSet T = new TreeSet (); t.add ("A"); t.add ("B"); t.add ("a");**

TreeSet T = new TreeSet (); t.add (null); t.add ("B"); t.add ("a"); ❌

TreeSet T = new TreeSet (); t.add (null);

TreeSet T = new TreeSet (); t.add ("A"); t.add (null); t.add ("a"); ❌

CONSTRCTORS:

i. **TreeSet t = new TreeSet ();** It creates an Empty Tree Set object with default natural Sorting Order. Default Natural Sorting Order for numbers is ascending order and Strings are Alphabetical Order.
ii. **TreeSet t = new TreeSet (Comparator c);** It creates an Empty Tree Set object with customized Sorting Order.
iii. **TreeSet t = new TreeSet (Collection c);** For Inter converting to any collection Object.
iv. **TreeSet t = new TreeSet (Sorted Set S)**

22.Comparable vs. Comparator

- Comparable is an Interface which is available in java.lang package. Comparator is an interface which is available in java.util package.
- Comparable have default natural sorting order. Comparator has Customized Sorting Order.
- Comparable interface have only on method that is **compareTo();**
- Comparator interface have two methods **compare ()** and **equals ().** But when we implement comparator interface no need to provide implementation for **equals ()** method because it already available in Object class. Object class is super class for all classes in java.
- All wrapper classes and String Class implements Comparable interface.
- The only implemented classes for Comparator are **Collator** and **Rule Based Collator.**
- Compare() and compareTo() → returns **+ve** if object 1 comes **before** object 2, returns **-ve** if object 1 comes **after** object 2, returns **0** if object 1 and object 2 **equal**.

23.Navigable Set

- Navigable Set is a child interface of Sorted Set.
- In 1.6 version it was released
- It defines several methods for navigation pupose.

METHODS: These methods are applicable for Sorted set and it` implemented classes.

floor(e);	Returns higher elements which is <=e
lower(e)	Returns highest elements which is < e
ceiling(e)	Returns lowest elements which is >=e
higher(e)	Returns lowest elements which is > e
pollFirst()	Remove and Returns first element
pollLast()	Remove and Returns last element
descendingSet()	Returns Navigable Set in reverse order

24.Queue

- Queue is a child interface of Collection.
- If we want to represent a group of elements in to single entity prior to processing that means before processing then we should go for Queue.
- It follows **FIFO** order. But we can create our own priority by using priority queue
- From 1.5 versions onwards LinkedList also implements Queue interface.
- Ex: we have to send mails for 1000 members. In this case we have to store all mail ids for 1000 members we can store in some data structure before sending.

METHODS:

int size();	Returns number of elements in the queue
Boolean offer();	To add object into the queue
Object peek()	Returns head element of the queue. If the queue is empty it returns null
Object element()	Returns head element of the queue. If the queue is empty it raises run time exception says No Such Element Exception.
Object poll()	Remove and Returns head element of the queue. If the queue is empty it returns null
Object remove()	Remove and Returns head element of the queue. If the queue is empty it raises run time exception says No Such Element Exception.

25. Priority Queue

- Priority Queue is a class which is used for prior to processing objects according to some priority.
- Default sorting order and customized sorting order both are possible.
- Insertion order is not preserved but elements can be stored according some sorting order.
- Heterogeneous values are not allowed.
- If we are depending on Default natural sorting order then elements must be homogeneous and comparable otherwise it gives run time exception says class cast exception.
- If we are depends our own custom sorting order then elements no need to homogeneous and comparable.
- Null insertion is not possible.

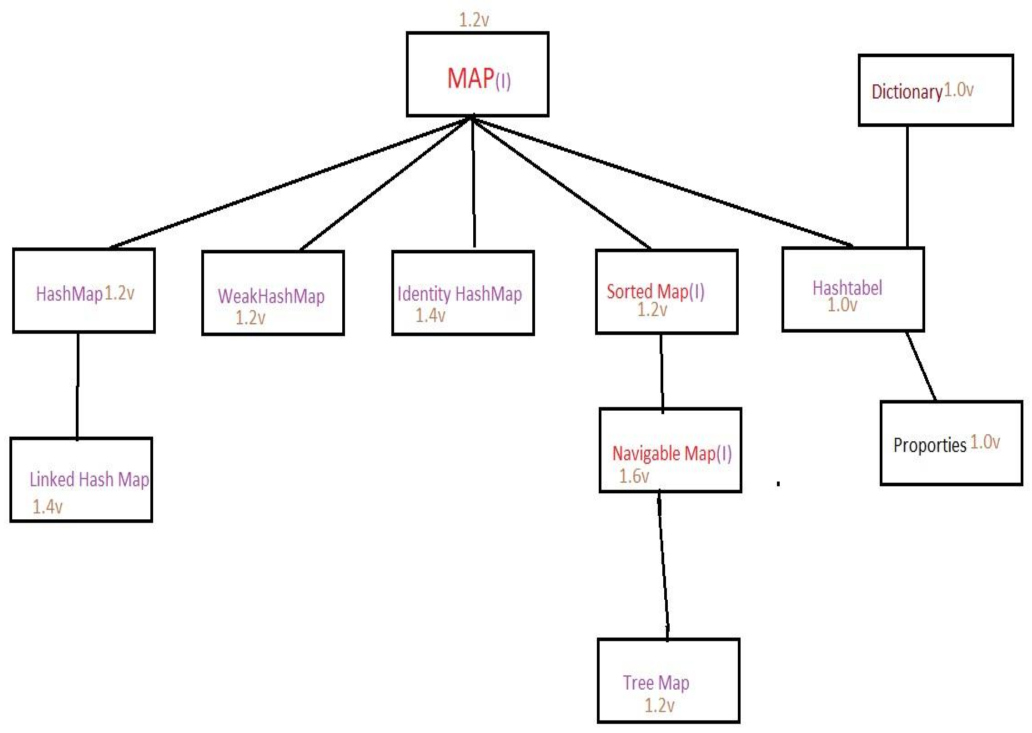
CONSTRCTORS:

i. **PriorityQueue PQ = new PriorityQueue ();** It creates an Empty Priority Queue object with default natural Sorting Order and initial capacity of **11**.
ii. **PriorityQueue PQ = new PriorityQueue (int initial capacity);** It creates an Empty Priority Queue object with initial capacity.
iii. **PriorityQueue PQ = new PriorityQueue (Collection c);** For Inter converting to any collection Object.
iv. **PriorityQueue PQ = new PriorityQueue (Sorted Set S)**

26.Blocking Queue

- Blocking Queue is a child interface of Queue.
-

27.Map Interface



- If we want to represent a group of objects as key-value pair then we should go for map.
- Both key and values must be objects only.
- Duplicate keys are not allowed but duplicate values are allowed.
- Each key-value pair is called as Entry. So Map is called as collection of Entry Objects.
- Entry is an interface available within Map interface.
- Without existing map object there is no chance for existing Entry object.

METHODS:

Object put(key, value);	To add one key-value pair to the map. If the key is already present then old value will be replaced with new value and returns old value otherwise it returns null.
Void putAll (Map m)	Add all entries in old map in to new map
Object get(object o)	It returns key name if not exists returns null.
Object remove(object o)	It removes key if not exists returns null.
Boolean containsKey (object key)	Return true if key is available otherwise return false
Boolean containsValue (object value)	Return true if value is available otherwise return false
Boolean isEmpty()	Return true if map is empty.
Int size();	Return size of the map
Void clear();	Clear the map object
Set keyset();	Return all keys as set object
Collection values	Return all values as collection object
Set entrySet();	Return all key-values as set object
Object getKey();	Return key from Entry in map object
Object getValue();	Return value from Entry in map object
Object setValue(Object o);	Update value in Entry

28.Entry Interface

- Entry is an interface which is available in Map interface
- In Map objects will be stored in key-value format. Each key-value is called as Entry.
- Without existing map object there is no chance for existing Entry object.
- Entry specific methods are applicable for only Entry Object.

  EX:

```
Interface Map {
        Interface Entry {

                Object getKey ();
                Object getValue ();
                Object setValue ();

        }
}
```

29.HashMap

- HashMap is class which is available in Map interface.
- It follows the Hashtable data structure.
- Insertion order is not preserved. But objects are inserted according some sorting order.
- Duplicate keys not allowed but duplicate values are allowed.
- Null insertion is possible for values but as a key null allows only once.
- HashMap is already implements serializable and Clonable interface.
- It is the best choice for search operations.

Constructors:

HashMap h = new HashMap () ;

→It creates an empty HashMap object with default initial capacity 16 and default fill ratio (or) load factor is 0.75.

HashMap h = new HashMap (int initial capacity) ;

→It creates an empty HashMap object with specified initial capacity and default fill ratio (or) load factor is 0.75.

HashMap h = new HashMap (int initial capacity, float load factor) ;

→It creates an empty HashMap object with specified initial capacity and specified fill ratio (or) load factor.

HashMap h = new HashMap (Map m); by interconnecting the HashMap in to any other map object.

EXample:

```
Class c {
        Set s = map. EntrySet (); →returns entry object
        Iterator itr = c.iterator ();
        While {
                map.Entry m= (map. Entry )itr.Next();
                sop (m.getKey());→prints key
                Sop (m.getValue());→prints value
                If (m.getKey().equals("A") ){
                        m.setValue("a");
                }
        }
}
```

30.HashMap vs. Hashtable

- Every Method Present in HashMap is Not Synchronized but Hashtable is synchronized.
- HashMap allows multiple threads at a time so it is not thread safe and performance is high because of thread waiting time is low .
- Hashtable doesn`t allows multiple threads at a time so it is thread safe and performance is low because of thread waiting time is high .
- Null is allowed for key and value in HashMap.
- Null is not allowed in Hashtable.
- HashMap introduced in 1.2 version
- Hashtable introduced in 1.0 version. So it is legacy class.

31.Linked HashMap

- Linked HashMap is just like a HashMap with an additional feature maintaining an order of elements inserted into it.
- HashMap provided the advantage of quick insertion, search and deletion but it never maintained the track and order of insertion which the LinkedHashMap provides where the elements can be accessed in their insertion order.
- It implements the Map interface and extends HashMap class.

Constructors:

LinkedHashMap h = new LinkedHashMap () ;

→It creates an empty LinkedHashMap object with default initial capacity 16 and default fill ratio (or) load factor is 0.75.

LinkedHashMap h = new LinkedHashMap (int initial capacity) ;

→It creates an empty LinkedHashMap object with specified initial capacity and default fill ratio (or) load factor is 0.75.

LinkedHashMap h = new Linked HashMap (int initial capacity, float load factor) ;

→It creates an empty LinkedHashMap object with specified initial capacity and specified fill ratio (or) load factor.

LinkedHashMap h = new LinkedHashMap (Map m); by interconnecting the LinkedHashMap in to any other map object.

32. equals() vs. ==

- equals () method is used to compare content.
- == Operator is used to compare reference or address of elements.

EX:

```
Integer I1 = new Integer (10); // here I1 =10
Integer I2 = new Integer (10); // here I2 =10
SOP (I1==I2); // address will compared so returns false
SOP (I1.equals (I2)); // values will compared so returns true
```

33.Identity HashMap

- Identity HashMap is just like HashMap but small difference is there.
- In the case of Normal HashMap JVM will use .equals () method to compare keys. So that here content of key will be compared. So there is no chance for duplicate keys in Normal HashMap.
- In the case of Identity HashMap JVM will use == operator to compare keys. Here it will compare address of keys. So there is a chance for duplicate keys in Identity HashMap.

  EX:

```
Integer I1 = new Integer (10); // here I1 =10
Integer I2 = new Integer (10); // here I2 =10
SOP (I1==I2); // address will compared so returns false
SOP (I1.equals (I2)); // values will compared so returns true
HashMap HM = new HashMap ();
IdentityHashMap IM = new IdentityHashMap ();
HM.put(I1);
HM.put(I2);
IM.put(I1);
IM.put(I2);
SOP (HM); // {"10":"kalyan"} – duplicates not allowed
SOP (IM); // {"10":"pawan","10":"Pawan"} –duplicate allowed
```

NOTE: I1 and I2 are duplicate keys in Normal HashMap because it checks like I1.equals (I2) so the last element only added to map object.

I1 and I2 are not duplicate keys in Identity HashMap because it checks like I1 == I2 so the both element are added to map object.

34.Weak HashMap

- Weak HashMap is just like HashMap but small difference is there.
- In the case of Normal HashMap Even Object don`t have any reference and associated with map object it is not eligible for garbage collector. Here HashMap dominates garbage collector.
- In the case of weak HashMap Object don`t have any reference and even associated with map object it is eligible for garbage collector. Here garbage collector dominates weak HashMap

EX:

```
WeakHashMap WHM = new WeakHashMap ();
HashMap HM = new HashMap ();
Temp t =new Temp ();
WHM.put(t, "surya");
HM.put(t, "surya");
SOP (HM); // {temp = surya}
SOP (WHM); // {temp = surya}
t=null;
system.gc();
Thread.sleep(5000);
SOP (HM); // temp = surya } – temp is not removed
SOP (WHM); // { }
```

35.Sorted Map

- It is the child interface of Map interface.
- If we want to represent a group of individual objects according to some sorting order then we should go for Sorted Map.
- Its implemented class is Tree Map.

Note:

METHODS: These methods are applicable for Sorted Map and it` implemented classes.

firstKey();	Returns the first (lowest) key currently in this Map
lastKey()	Returns the last (highest) key currently in this Map.
headMap(K toKey)	Returns a view of the portion of this Map whose keys are strictly less than toKey.
tailMap(K fromKey)	Returns a view of the portion of this Map whose keys are greater than or equal to fromKey.
SubMap(fromkey k , toKey tk)	Returns a view of the portion of this Map whose keys range from fromKey, inclusive, to toKey, exclusive.
comparator()	Returns the Comparator used to order the keys in this Map, or null if this Map uses the natural ordering of its keys.
values()	Returns a Collection view of the values contained in this map
keySet()	Returns a Set view of the keys contained in this map
entrySet()	Returns a Set view of the mappings contained in this map

36.Tree Map

- It is the child class of Sorted Map Interface.
- It follows Red-Black Tree data structure.
- Insertion Order is not preserved. But all objects are stored according to some sorting order of keys.
- Duplicate keys not allowed but duplicate values are allowed.
- If we are depending on the default natural sorting order keys should be homogeneous and comparable.
- If we are depending on the customized sorting order keys need not be homogeneous and comparable.
- For Non empty TreeMap null is not accepted.
- For Empty TreeMap first element will be added null. But after null if we try to add any elements it raises run time exception null pointer exception. Because if we compare null with any element it returns null pointer exception.
- Null is accepted as value.
- From 1.7 versions on wards null is not accepted as key if it is first key also.

<u>Constructors:</u>

```
 TreeMap t = new TreeMap ();
TreeMap h = new TreeMap (Comparator c);
TreeMap h = new TreeMap (SortedMaps m);
TreeMap h = new TreeMap (Map m);
```

37.Navigable Map

- Navigable Map is a child interface of Sorted Map.
- In 1.6 version it was released
- It defines several methods for navigation purpose.

METHODS:

floorKey(e);	Returns higher elements which is <=e
lowerKey(e)	Returns highest elements which is < e
ceilingKey(e)	Returns lowest elements which is >=e
higherKey(e)	Returns lowest elements which is > e
pollFirstEntry()	Remove and Returns first element
pollLastEntry()	Remove and Returns last element
descendingMap()	Returns Navigable Map in reverse order

38.Hashtable

- In java Hashtable is class which is implemented based on Hashtable Data Structure.
- Insertion order is not preserved. Elements should be stored according to hash code of keys.
- Duplicate keys are not allowed but duplicate values are allowed.
- Heterogeneous objects are allowed for both key and value.
- Null is not allowed.
- Already implements serializable and Clonable interface.
- Every method present in Hashtable is synchronized. So it is Thread safe.
- Hashtable is suitable for search operations.

Constructors:

Hashtable h = new Hashtable ();

→It creates an empty Hashtable object with default initial capacity 11 and default fill ratio (or) load factor is 0.75.

Hashtable h = new Hashtable (int initial capacity) ;

→It creates an empty Hashtable object with specified initial capacity and default fill ratio (or) load factor is 0.75.

Hashtable h = new Hashtable (int initial capacity, float load factor);

→It creates an empty Hashtable object with specified initial capacity and specified fill ratio (or) load factor.

Hashtable h = new Hashtable (Map m); by interconnecting the Hashtable in to any other map object.

Rules For Hash Code:

- If we store object in Hashtable it generates its hash code for that key and stored on that bucket. For example my key is k and value is v. Now it generates Hash code for key k for example hash code for k is 5 then it scored in the 5th bucket like {5 = v}.
- If corresponding Hash Code is exceeded in Hashtable then it generates new Hash code for that by using formula **Hash Code % Capacity.**
- If object is already in bucket and new object also same hash code then new object also stored in same bucket at the time of retrieving the entries from right to left it returns.
- Objects will be return from top to bottom.

39.Properties

- In any programming we are not recommended to write hard code which are used frequently like password, username and mail id like that. If we write hard code for that it creates lot of problems.
- We can overcome this problem by using properties file such type of variables are configured in to the properties file.
- From the properties file we have to read the java program and we will use those properties.
- Java properties object will read properties from file and hold the values. We are using those properties by getting from that object only.
- In Normal map key and values are any type. But in properties key and value should be string only.

Constructors:

```
Properties p = new Properties ();
```

METHODS:

String getProporty(String pName);	To get the property name
void setProporty(String pName, String pValue);	To set the property
Enumeration proportnames ();	Returns all properties.
Void load(Input Stream is)	To load properties from properties file in to java property object.
void store(output Stream os)	To store properties from java properties object in to property file.

40.List vs. Set vs. Map

- List allows the duplicate values but set doesn`t allow duplicate values and HashMap is key value format it allows duplicate values but we must maintain unique key.
- List follows insertion order that means in which order we store the elements that order is followed but set doesn’t follow insertion order.
- List allows multiple null values and set allows only one null value. Map allows the multiple null values but only one key as null because key is unique in Map.
- List implements : Array List, Linked List and Vector
- Set implements : Hash Set, Linked Hash Set and Tree Set(tree Set also implemented by Sorted Set)
- Map implements: Hash Map, Linked Hash Map, Hash Table and Tree Map.