Collection Framework

Prepared By : Ronak R Patel

Introduction of Array

- An array is an indexed collection of fixed no of homogeneous data elements. (or)
- An array represents a group of elements of same data type.
- The main advantage of array is we can represent huge no of elements by using a single variable. So that readability of the code will be improved.

Limitations of Object[] array:

- Arrays are fixed in size that is once we created an array there is **no** chance of **increasing (or) decreasing** the **size** based on our requirement hence to use arrays concept compulsory we **should know the size in advance** which may not possible always.
- Arrays can hold only **homogeneous** data elements.

Example:

```
Student[] s=new Student[10000];
s[0]=new Student();//valid
s[1]=new Customer();//invalid(compile
time error)
```

```
Object[] o=new Object[10000];
o[0]=new Student();
o[1]=new Customer();
```

Solution of the Limitation

- Collections are growable in nature based on our requirement we can increase (or) decrease the size hence memory point of view collections concept is recommended to use.
- Collections can hold both homogeneous and heterogeneous objects.
- Every collection class is implemented based on some standard data structure hence for every requirement ready-made method support is available being a programmer we can use these methods directly without writing the functionality on our own.

Differences between Arrays and Collections

Arrays	Collections
1) Arrays are **fixed** in size.	1) Collections are **growable** in nature.
2) Memory point of view arrays are **not** recommended to use.	2) Memory point of view collections are **highly recommended to use.**
3) **Performance** point of view arrays are **recommended to use.**	3) **Performance** point of view collections are **not recommended to use.**
4) Arrays can **hold** only **homogeneous data type elements.**	4) Collections can **hold** both **homogeneous** and **heterogeneous** elements.
5) There is **no underlying data structure** for arrays and hence there is no readymade method support.	5) Every collection class is implemented based on **some standard data structure** and hence readymade method support is available.
6) Arrays can hold both **primitives and object types.**	6) Collections can hold only **objects but not primitives.**

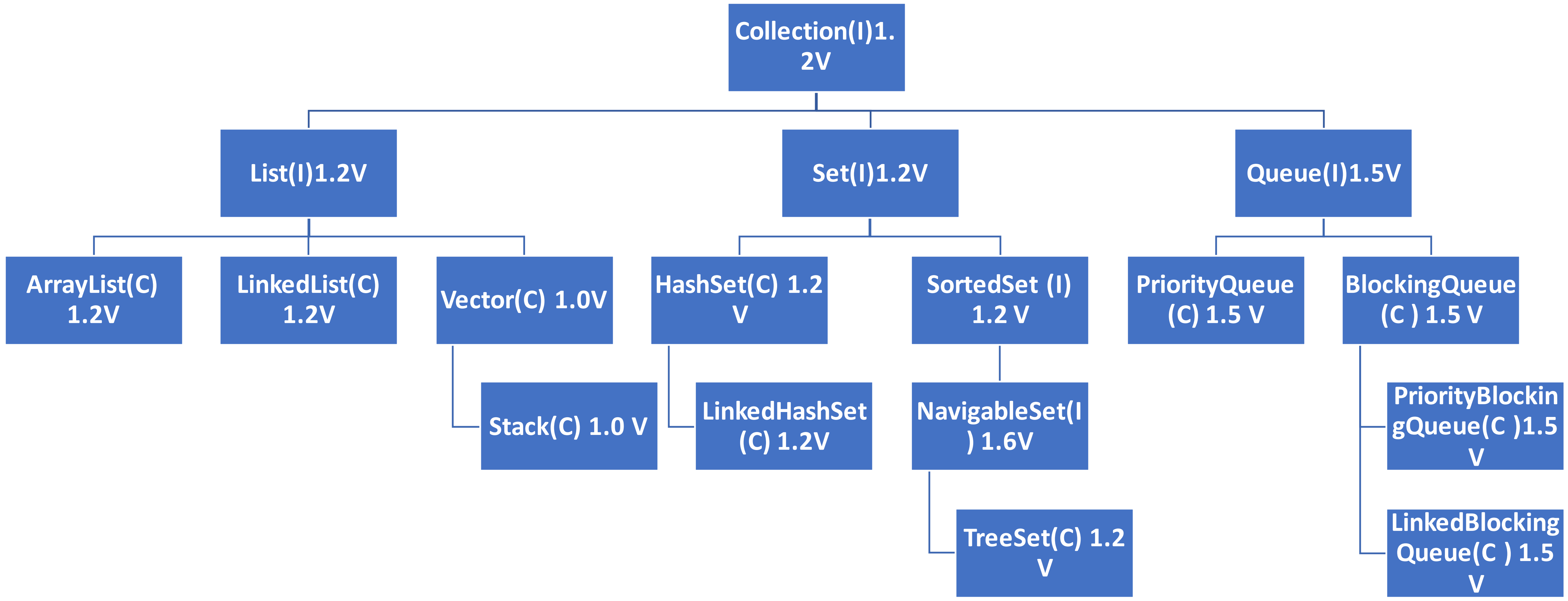
Collection:

- If we want to represent a group of objects as single entity then we should go for collections.

Java	C++
Collection	Containers
Collection framework	STL(Standard Template Library)

Collection:

- If we want to represent a group of "**individual objects**" as a **single entity** then we should go for collection.
- In general we can consider **collection** as **root interface** of **entire collection framework.**
- Collection interface defines the **most common methods** which can be applicable for **any collection object**.
- There is **no concrete class** which **implements Collection** interface directly.


Collection(I)1.2V
List(I)1.2V
Set(I)1.2V
Queue(I)1.5V
ArrayList(C) 1.2V
LinkedList(C) 1.2V
Vector(C) 1.0V
HashSet(C) 1.2 V
SortedSet (I) 1.2 V
PriorityQueue (C) 1.5 V
BlockingQueue (C) 1.5 V
Stack(C) 1.0 V
LinkedHashSet (C) 1.2V
NavigableSet(I) 1.6V
PriorityBlockingQueue(C)1.5 V
TreeSet(C) 1.2 V
LinkedBlocking Queue(C) 1.5 V

Collection Interface: (1.2 V)

- If we want to represent a group of individual objects as a single entity then we should go for Collection interface. This interface defines the most common general methods which can be applicable for any Collection object.

- There is no concrete class which implements Collection interface directly.

List interface: (1.2 V)

- It is the child interface of Collection.
- If we want to represent a group of individual objects as a single entity where **duplicates are allow and insertion order is preserved**. Then we should go for List.
- We can differentiate duplicate objects and we can maintain insertion order by means of index hence "index play very important role in List".

ArrayList Class : (1.2 V)

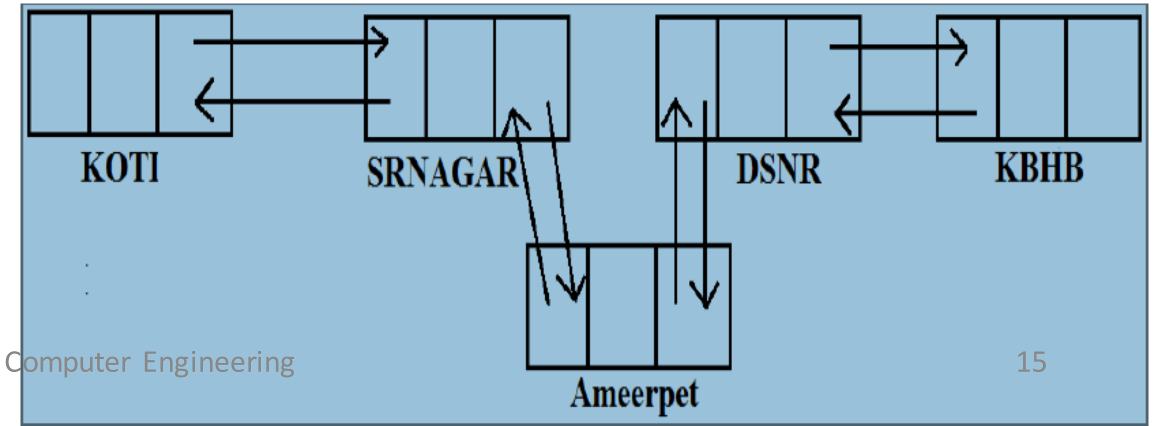
- The underlying data structure is a **resizable array** (or) **growable array.**
- Duplicate objects are allowed.
- Insertion order preserved.
- Heterogeneous objects are allowed.
- Null insertion is possible.
- initial capacity "10“
  - *New capacity=(current capacity*3/2)+1*
- Most Suitable for Searching Operation
- For **Insertion** and **Deletion** Operation ArrayList is the **Worst Choice**.

Constructors for ArrayList:

- ArrayList a=new ArrayList();
- ArrayList a=new ArrayList(int initialcapacity);
  - Creates an empty ArrayList object with the specified initial capacity.
- ArrayList a=new ArrayList(collection c);
  - Creates an equivalent ArrayList object for the given Collection that is this constructor meant for inter conversation between collection objects. That is to dance between collection objects.

LinkedList:

- The underlying data structure is **double LinkedList**
- If our frequent operation is **insertion (or) deletion** in the middle then LinkedList is the **best choice**.
- If our frequent operation is **retrieval operation** then LinkedList is **worst choice.**
- **Duplicate** objects are **allowed**.
- **Insertion** order is **preserved**.
- **Heterogeneous** objects are allowed.
- **Null** insertion is **possible**.
- Implements **Serializable** and **Cloneable** interfaces but not **RandomAccess**.



Vector Class: (1.0 V) -> Legacy Class

- The underlying data structure is resizable array (or) growable array.
- Duplicate objects are allowed.
- Insertion order is preserved.
- Heterogeneous objects are allowed.
- Null insertion is possible.
- Implements Serializable, Cloneable and RandomAccess interfaces.
- default initial capacity 10.
  - newcapacity=currentcapacity*2
- **Every method present in Vector is synchronized and hence Vector is Thread safe.**

Constructors:

1. Vector v=new Vector();
   - Creates an empty Vector object with default initial capacity 10.
   - Once Vector reaches its maximum capacity then a new Vector object will be
   - created with double capacity. That is "newcapacity=currentcapacity*2".
2. Vector v=new Vector(int initialcapacity);
3. Vector v=new Vector(int initialcapacity, int incrementalcapacity);
4. Vector v=new Vector(Collection c);

Difference between ArrayList and Vector

ArrayList	Vector
No method is synchronized	Every method is synchronized
At a time multiple Threads are allow to operate on ArrayList object and hence ArrayList object is not Thread safe.	At a time only one Thread is allow to operate on Vector object and hence Vector object is Thread safe.
Relatively performance is high because Threads are not required to wait.	Relatively performance is low because Threads are required to wait.
It is non legacy and introduced in 1.2v	It is legacy and introduced in 1.0v

Stack class: 1.0 V -> legacy Class

- It is the child class of Vector.
- Whenever last in first out(LIFO) order required then we should go for Stack.

Cursors

- want to get objects one by one from the collection then we should go for cursor.
  1. **Enumeration**
  2. **Iterator**
  3. **ListIterator**

Enumeration:

- We can use Enumeration to get objects one by one from the legacy collection objects.
- We can create Enumeration object by using elements() method.

  Enumeration e=v.elements();
- Enumeration interface defines the following two methods
  - public boolean hasMoreElements();
  - public Object nextElement();

Limitations of Enumeration:

- We can apply Enumeration concept only for legacy classes and it is not a universal cursor.
- By using Enumeration we can get only read access and we can't perform remove operations.
- To overcome these limitations sun people introduced Iterator concept in 1.2v.

Iterator:

- We can use Iterator to get objects one by one from any collection object.
- We can apply Iterator concept for any collection object and it is a universal cursor.
- While iterating the objects by Iterator we can perform both read and remove operations.
- We can get Iterator object by using iterator() method of Collection interface.

  Iterator itr=c.iterator();

- Iterator interface defines the following 3 methods.
  - public boolean hasNext();
  - public object next();
  - public void remove();

Limitations of Iterator:

- Both enumeration and Iterator are single direction cursors only. That is we can always move only forward direction and we can't move to the backward direction.
- While iterating by Iterator we can perform only read and remove operations and we can't perform replacement and addition of new objects.

ListIterator:

- ListIterator is the child interface of Iterator.
- By using listIterator we can move either to the forward direction (or) to the backward direction that is it is a bi-directional cursor.
- While iterating by listIterator we can perform replacement and addition of new objects in addition to read and remove operations

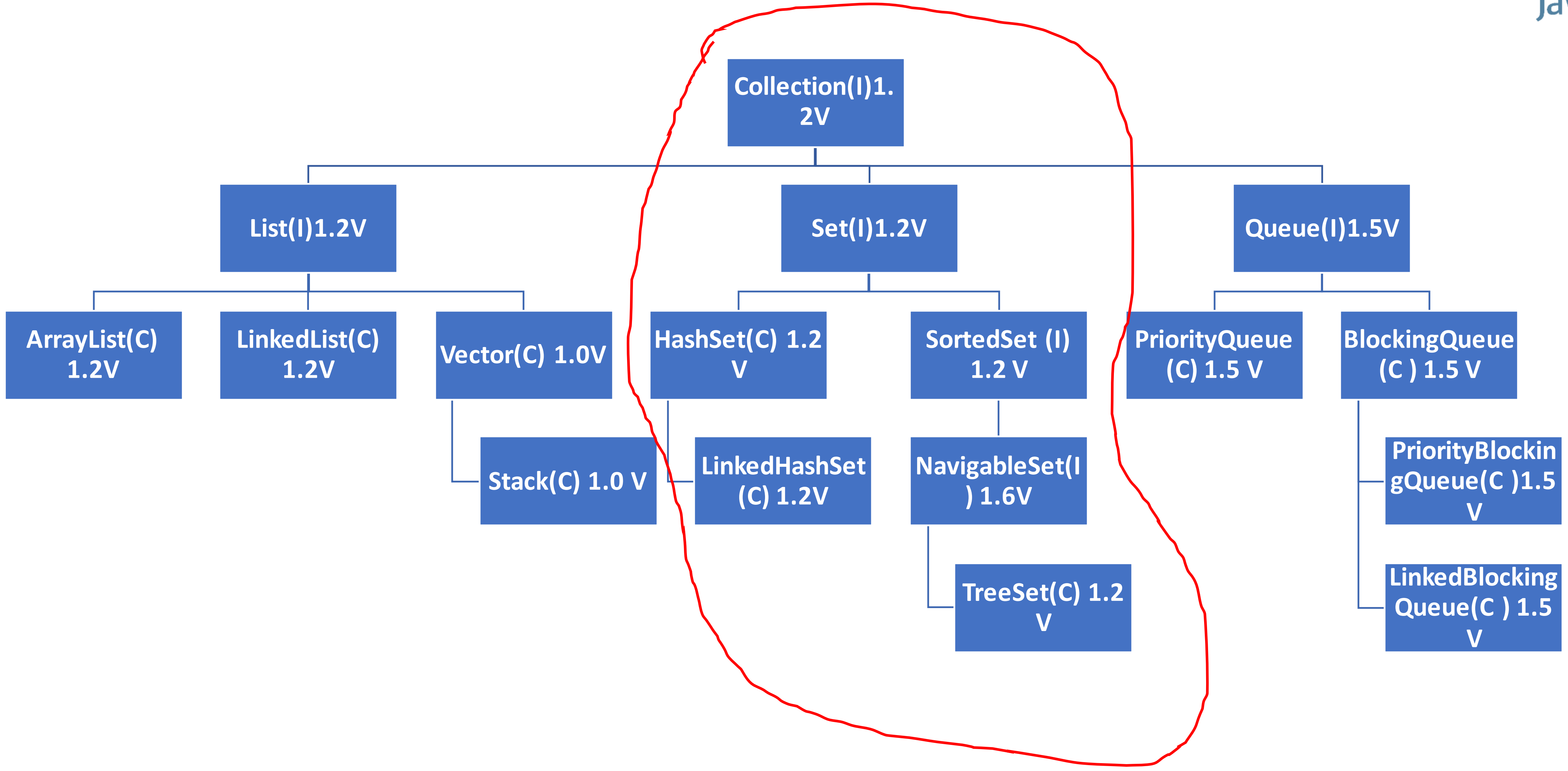
- By using listIterator method we can create listIterator object.
  - ListIterator itr=l.listIterator();
  - (l is any List object)

ListIterator interface : 9 methods.

1. public boolean hasNext();
2. public Object next(); forward
3. public int nextIndex();
4. public boolean hasPrevious();
5. public Object previous(); backward
6. public int previousIndex();
7. public void remove();
8. public void set(Object new);
9. public void add(Object new);

Compression of Enumeration Iterator and ListIterator ?

Property	Enumeration	Iterator	ListIterator
1) Is it legacy ?	Yes	No	No
2) It is applicable for ?	Only legacy classes.	Applicable for any collection object.	Applicable for only list objects.
3) Moment?	Single direction cursor(forward)	Single direction cursor(forward)	Bi-directional.
4) How to get it?	By using elements() method.	By using iterator()method.	By using listIterator() method.
5) Accessibility?	Only read.	Both read and remove.	Read/remove/replace/add.
6) Methods	hasMoreElement() nextElement()	hasNext() next() remove()	9 Methods


Collection(I)1.2V
List(I)1.2V
Set(I)1.2V
Queue(I)1.5V
ArrayList(C) 1.2V
LinkedList(C) 1.2V
Vector(C) 1.0V
HashSet(C) 1.2 V
SortedSet (I) 1.2 V
PriorityQueue (C) 1.5 V
BlockingQueue (C) 1.5 V
Stack(C) 1.0 V
LinkedHashSet (C) 1.2V
NavigableSet(I) 1.6V
PriorityBlockingQueue(C)1.5 V
TreeSet(C) 1.2 V
LinkedBlocking Queue(C) 1.5 V

Set interface:

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

HashSet:

- The underlying data structure is Hashtable.
- Insertion order is not preserved and it is based on hash code of the objects.
- Duplicate objects are not allowed.
- If we are trying to insert duplicate objects we won't get compile time error and runtime error add() method simply returns false.
- Heterogeneous objects are allowed.
- Null insertion is possible.(only once)
- Implements Serializable and Cloneable interfaces but not RandomAccess.
- HashSet is best suitable, if our frequent operation is "Search".
- default initial capacity 16
  - default fillratio 0.75(fill ratio is also known as load factor).

Constructors:

1. HashSet h=new HashSet();

Creates an empty HashSet object with default initial capacity 16 and default fill ratio 0.75(fill ratio is also known as load factor).

2. HashSet h=new HashSet(int initialcapacity);

Creates an empty HashSet object with the specified initial capacity and default fill ratio 0.75.

3. HashSet h=new HashSet(int initialcapacity,float fillratio);

4. HashSet h=new HashSet(Collection c);

LinkedHashSet:

- It is the child class of HashSet.
- LinkedHashSet is exactly same as HashSet except the following differences.

HashSet	**LinkedHashSet**
The underlying data structure is Hashtable.	The underlying data structure is a combination of LinkedList and Hashtable.
Insertion order is not preserved.	Insertion order is preserved.
Introduced in 1.2 v.	Introduced in 1.4v.

SortedSet:

- It is child interface of Set.
- If we want to represent a group of "unique objects" where duplicates are not allowed and all objects must be inserting according to some sorting order then we should go for SortedSet interface.
- That sorting order can be either default natural sorting (or) customized sorting order.

SortedSet interface define the following 6 specific methods.

1. Object first();
2. Object last();
3. SortedSet headSet(Object obj);

    Returns the SortedSet whose elements are <obj.

4. SortedSet tailSet(Object obj);

    It returns the SortedSet whose elements are >=obj.

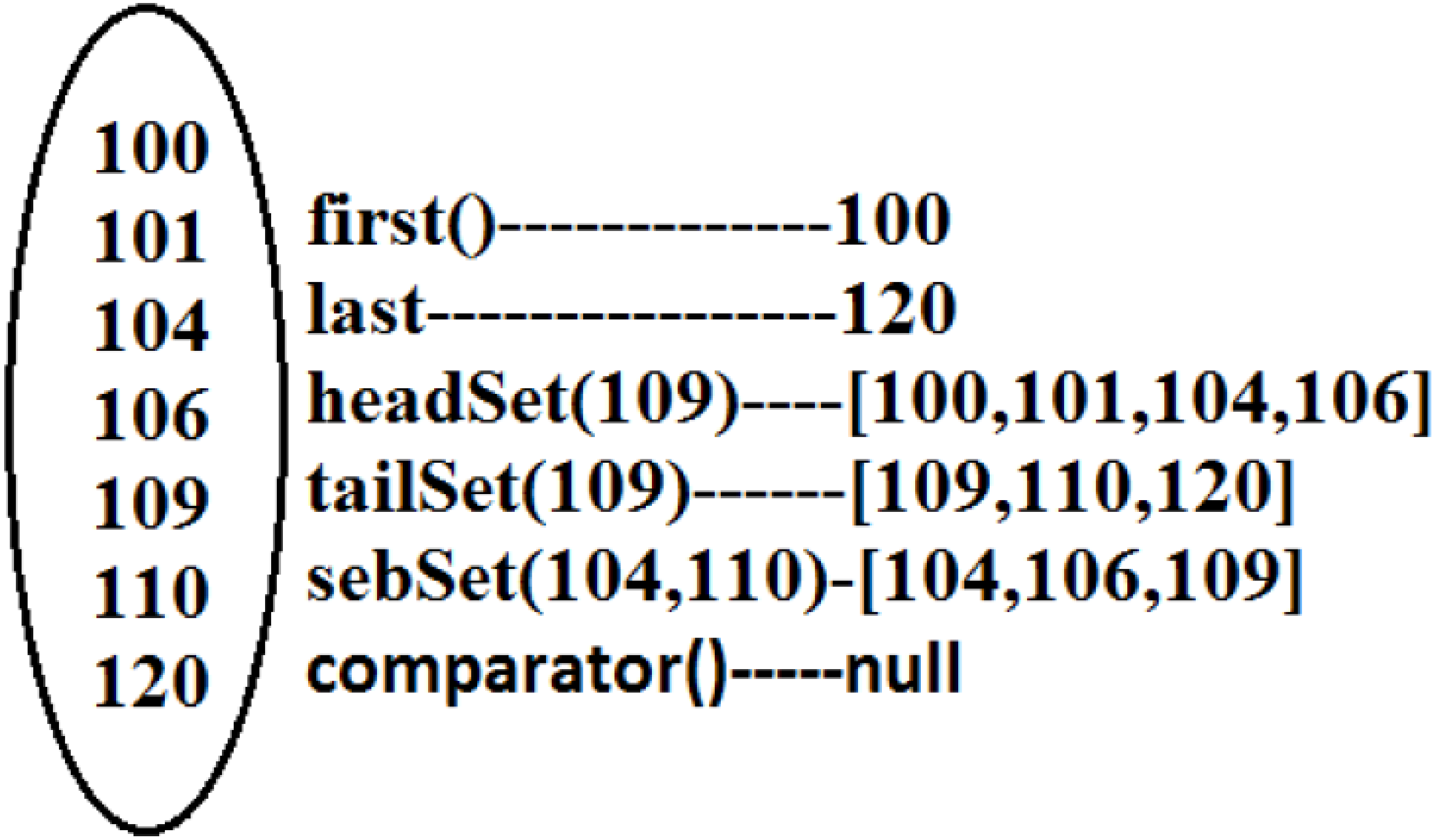
5. SortedSet subset(Object o1,Object o2);

    Returns the SortedSet whose elements are >=o1 but <o2.

6. Comparator comparator();

    o Returns the Comparator object that describes underlying sorting technique.

    o If we are following default natural sorting order then this method returns null.



100
101
104
106
109
110
120

first()-------------100
last-----------------120
headSet(109)----[100,101,104,106]
tailSet(109)------[109,110,120]
sebSet(104,110)-[104,106,109]
comparator()-----null

TreeSet:

1. The underlying data structure is balanced tree.

2. Duplicate objects are not allowed.

3. Insertion order is not preserved and it is based on some sorting order of objects.

4. Heterogeneous objects are not allowed if we are trying to insert heterogeneous objects then we will get ClassCastException.

5. Null insertion is possible(only once).

Constructors

1. TreeSet t=new TreeSet();

Creates an empty TreeSet object where all elements will be inserted according to

default natural sorting order.

2. TreeSet t=new TreeSet(Comparator c);

Creates an empty TreeSet object where all objects will be inserted according to customized sorting order specified by Comparator object.

3. TreeSet t=new TreeSet(SortedSet s);

4. TreeSet t=new TreeSet(Collection c);

Comparable interface: (DNSO)

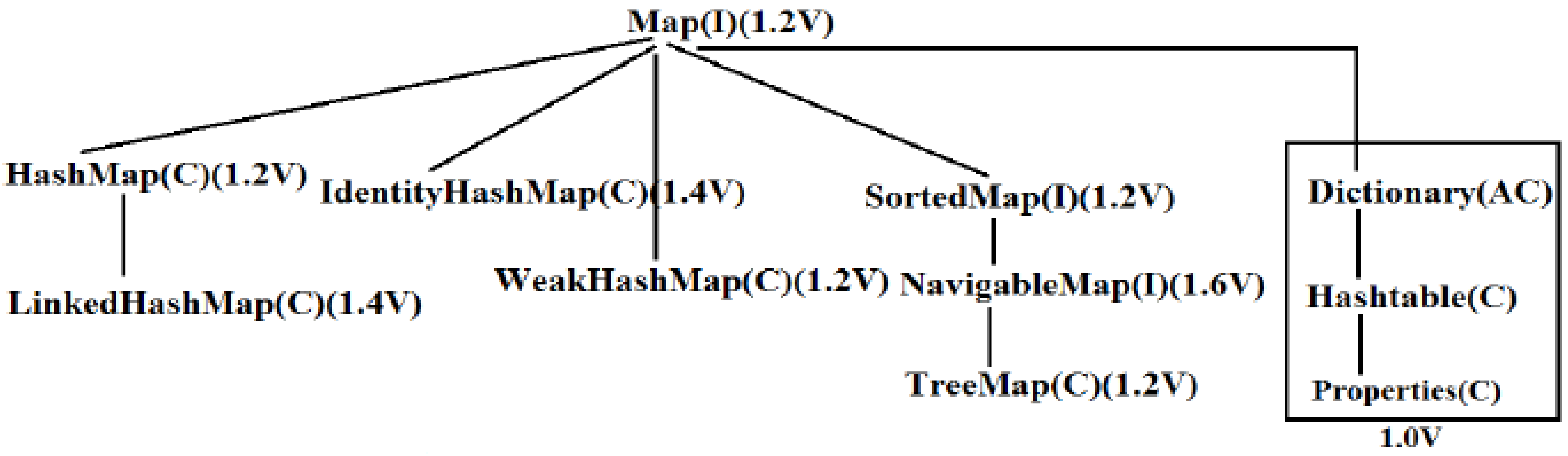
- Comparable interface present in java.lang package and contains only one method compareTo() method.
  - public int compareTo(Object obj);
- Example:
  - obj1.compareTo(obj2);
    - ➢ return –ve if and only if obj1 has to come before obj2
    - ➢ return +ve if and only if obj1 has to after before obj2
    - ➢ return 0 (Zero) if and only if obj1 & obj2 are the same.

TreeSet by default **uses** the **compareTo()** method for the comparison of two objects, and based on that, the **Binary Tree** will generate.

Comparator interface:

- Comparator interface present in java.util package this interface defines the following 2 methods.

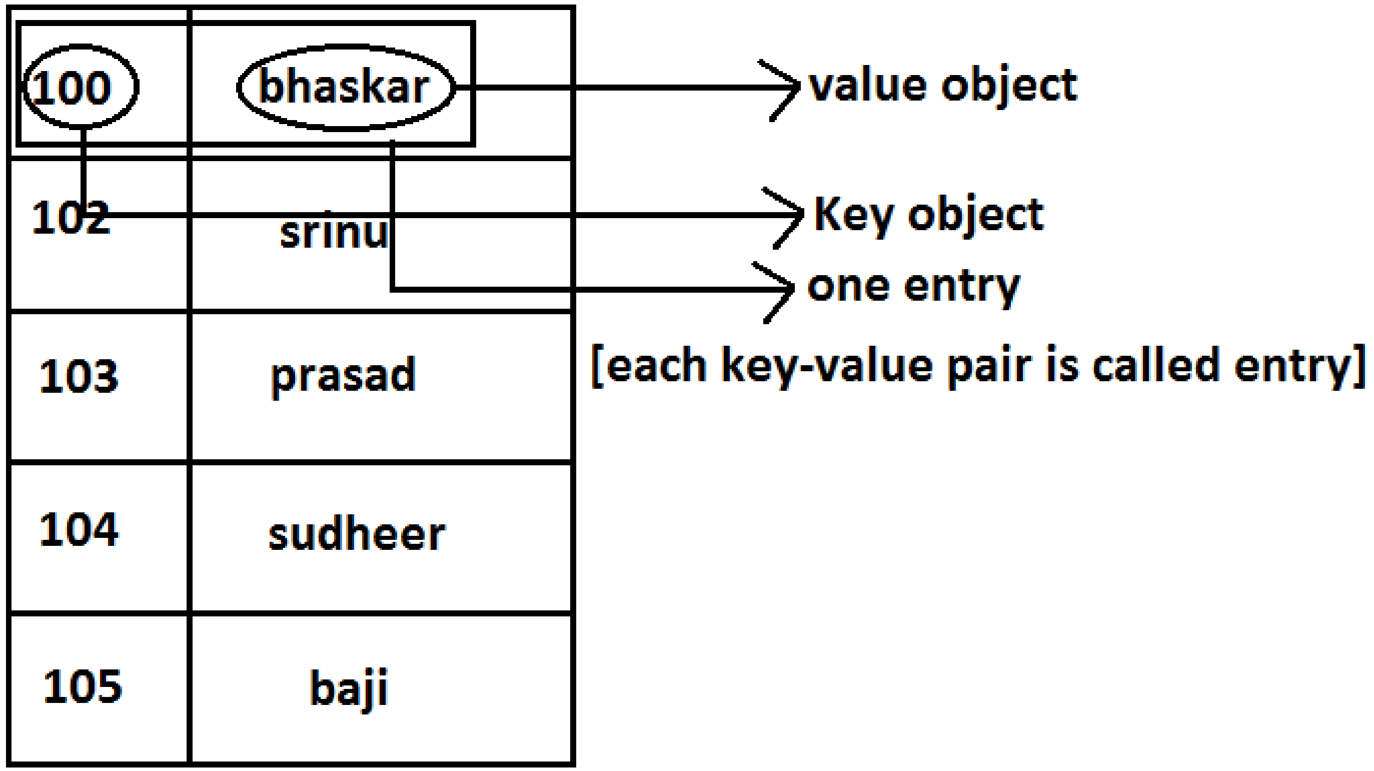
1. ***public int compare(Object obj1,Object Obj2);***
   - ➢ return –ve if and only if obj1 has to come before obj2
   - ➢ return +ve if and only if obj1 has to after before obj2
   - ➢ return 0 (Zero) if and only if obj1 & obj2 are the same.
2. ***public boolean equals(Object obj);***
   - Whenever we are implementing Comparator interface we have to provide implementation only for compare() method.
   - Implementing equals() method is optional because it is already available from Object class through inheritance.



Map(I)(1.2V)
HashMap(C)(1.2V)
IdentityHashMap(C)(1.4V)
SortedMap(I)(1.2V)
Dictionary(AC)
LinkedHashMap(C)(1.4V)
WeakHashMap(C)(1.2V)
NavigableMap(I)(1.6V)
Hashtable(C)
TreeMap(C)(1.2V)
Properties(C)
1.0V

Map:

- If we want to represent a **group of objects** as **"key-value"** pair then we should go for the Map interface.
- Both key and value are objects only.
- Duplicate keys are not allowed but values can be duplicated
- Each key-value pair is called "one entry".
- Map interface is not child interface of Collection and hence we can't apply Collection interface methods here



Entry interface:

- Each key-value pair is called one entry.
- Entry interface is define in Map interface only.

HashMap: (Bottom to Top, Left to Right)

1. The underlying data structure is Hashtable.
2. Duplicate keys are not allowed but values can be duplicated.
3. Insertion order is not preserved and it is based on hash code of the keys.
4. Heterogeneous objects are allowed for both key and value.
5. Null is allowed for keys(only once) and for values(any number of times).
6. It is best suitable for Search operations.
7. Default capacity is 16
    - Load factor is 0.75

Constructor

1. HashMap m=new HashMap();

    Creates an empty HashMap object with default initial capacity 16 and default fill ratio "0.75".

2. HashMap m=new HashMap(int initialcapacity);
3. HashMap m =new HashMap(int initialcapacity, float fillratio);
4. HashMap m=new HashMap(Map m);

Steps to Iterative

```
Set s = h.entrySet();
Iterator i = s.iterator();
while(i.hasNext())
{
        Map.Entry m = (Map.Entry) i.next();
        System.out.println(m.getKey()+"  "+m.getValue());
}
```

LinkedHashMap:

- The underlying data structure is a combination of Hashtable+ LinkedList.
- Insertion order is preserved.
- Introduced in 1.4v

SortedMap:

- It is the child interface of Map.
- If we want to represent a group of key-value pairs according to some sorting order of keys then we should go for SortedMap.
- Sorting is possible only based on the keys but not based on values.
- SortedMap interface defines the following 6 specific methods.

TreeMap:

1. The underlying data structure is RED-BLACK Tree.
2. Duplicate keys are not allowed but values can be duplicated.
3. Insertion order is not preserved and all entries will be inserted according to some sorting order of keys.
4. If we are depending on default natural sorting order keys should be homogeneous and Comparable otherwise we will get ClassCastException.
5. If we are defining our own sorting order by Comparator then keys can be heterogeneous and non Comparable.
6. There are no restrictions on values they can be heterogeneous and non Comparable.
7. For the empty TreeMap as first entry null key is allowed but after inserting that entry if we are trying to insert any other entry we will get NullPointerException.
8. For the non empty TreeMap if we are trying to insert an entry with null key we will get NullPointerException.
9. There are no restrictions for null values.

Constructor

- Hashtable h=new Hashtable();
  - Creates an empty Hashtable object with default initialcapacity 11 and default fill ratio 0.75.
- Hashtable h=new Hashtable(int initialcapacity);
- Hashtable h=new Hashtable(int initialcapacity,float fillratio);
- Hashtable h=new Hashtable (Map m);

Constructor

1. TreeMap t=new TreeMap();
   - For default natural sorting order.
2. TreeMap t=new TreeMap(Comparator c);
   - For customized sorting order.
3. TreeMap t=new TreeMap(SortedMap m);
4. TreeMap t=new TreeMap(Map m);

Hashtable:

- The underlying data structure is Hashtable.
- Insertion order is not preserved and it is based on hash code of the keys.
- Heterogeneous objects are allowed for both keys and values.
- Null key (or) null value is not allowed otherwise we will get NullPointerException.
- Duplicate keys are allowed but values can be duplicated.
- Every method present inside Hashtable is syncronized and hence Hashtable objet is Thread-safe.
- default initialcapacity 11
  - Fill ratio 0.75.

Generics in Java

- Generics means **parameterized types**.
- The idea is to allow type (Integer, String, … etc., and user-defined types) to be a parameter to methods, classes, and interfaces.