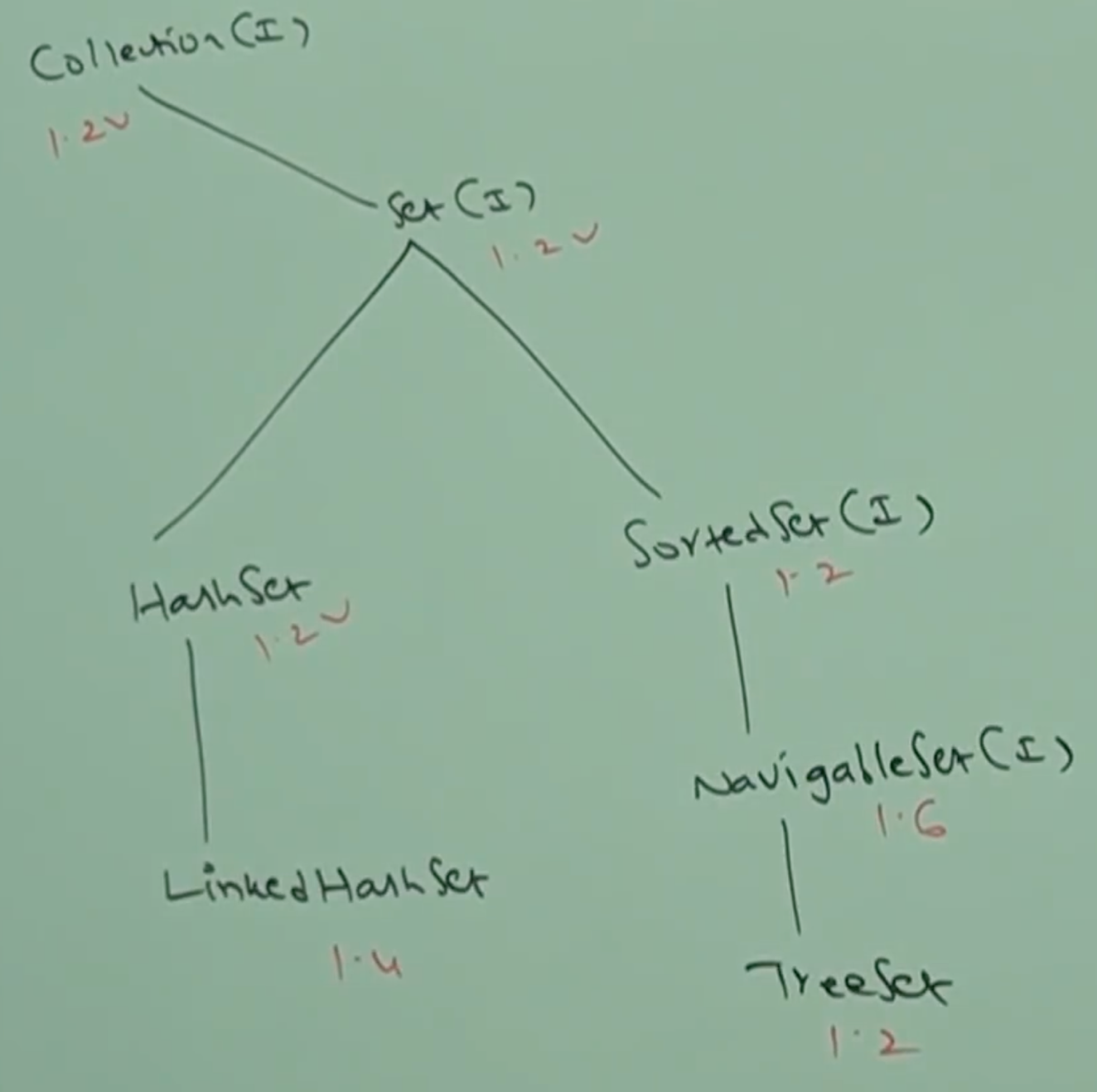
**Collection Framework – Part\_07**

* **Set(I):**



Set is child interface of Collection

If we want to represent a group of individual objects as a single

entity where duplicates are not allowed and insertion order not

preserved.

Set interface doesn’t contain any new method and we have to use

only Collection interface methods.

* **HashSet:**

1. The underlying data structure is Hashtable.
2. Duplicate objects are not allowed.
3. Insertion order is not preserved and it is based on Hashcode of objects.
4. null insertion is possible (only once, because duplicates not allowed).
5. Heterogeneous objects are allowed.
6. Implements Serializable, Cloneable but not RandomAccess interface.
7. HashSet is the best choice if our frequent operation is search.

Note:

In HashSet duplicates are not allowed, if we are trying to insert duplicates then we won’t get any compile time or runtime errors and add() simply returns false.

HashSet hs = new HashSet();

System.out.println(hs.add(“A”)); //true

System.out.println(hs.add(“A”)); //false

**Constructors:**

HashSet h = new HashSet();

Creates an empty HashSet object with default initial capacity of 16 and default fill ratio = 0.75

HashSet h = new HashSet(int initialCapacity);

Creates an empty HashSet object with specified initial

capacity and default fill ratio.

HashSet h = new HashSet(int initialCapacity, float fillRatio);

HashSet h = new HashSet(Collection c);

Creates an equivalent HashSet for the given Collection. This constructor meant for interconversion between Collection objects.

Fill Ration or Load Factor:

After filling how much ratio a new HashSet object will be

created, this ratio is called Fill Ratio or Load Factor.

For example:

Fill Ratio 0.75 means after filling 75% ratio a new HashSet object will be created automatically.

Example:

import java.util.\*;

class HashSetDemo{

public static void main(String[] args){

HashSet h = new HashSet();

h.add(“B”);

h.add(“C”);

h.add(“D”);

h.add(“Z”);

h.add(null);

h.add(10);

System.out.println(h.add(“Z”)); // false

System.out.println(h); // [

}

}

**Output:** [null, D, B, C, 10, Z]

* **LinkedHashSet:**

1. It is the child class of HashSet.
2. It is exactly same as HashSet (including constructors and methods) except the following differences.

|  |  |
| --- | --- |
| HashSet | LinkedHashSet |
| The underlying data structure is Hashtable. | Underlying data structure is a combination of LinkedList and Hashtable. |
| Insertion order is not preserved | Insertion order is preserved. |
| Introduced in 1.2V | Introduced in 1.4V |

1. In the above program if we replace HashSet with LinkedHashSet then the output is

[B, C, D, Z, null, 10]

That is insertion order is preserved.

Note:

In general, we can use LinkedHashSet to develop cache based applications where duplicates are not allowed and the insertion order preserved.

* **SortedSet:**

1. SortedSet is the child interface of Set.
2. If we want to represent a group of individual objects according to some sorting order without duplicates, then we should go for SortedSet.
3. SortedSet interface defines the following specific methods.

Object first();

Returns first element of the SortedSet

Object last();

Returns last element of the SortedSet.

SortedSet headSet(Object object);

Returns SortedSet whose elements are less than obj.

SortedSet tailSet(Object obj);

Returns SortedSet whose elements are >= obj1 and < obj2

Comparator comparator();

Returns Comparator object that describes underlying sorting technique. If we are using default natural sorting order then we will get null.

Note:

The default natural sorting order for numbers: Ascending Order

The default natural sorting order for String: Alphabetical order.

1. Example:

Input: [100, 101, 104, 106, 110, 115, 120]

first() 🡺 100

last() 🡺 120

headSet(106) 🡺 [100, 101, 104]

tailSet(106) 🡺 [106, 110, 115, 120]

subSet(101, 115) 🡺 [101, 104, 106, 110]

comparator()