**Collection Framework – Part\_16**

* **1.6 version enhancements in Collection framework:**

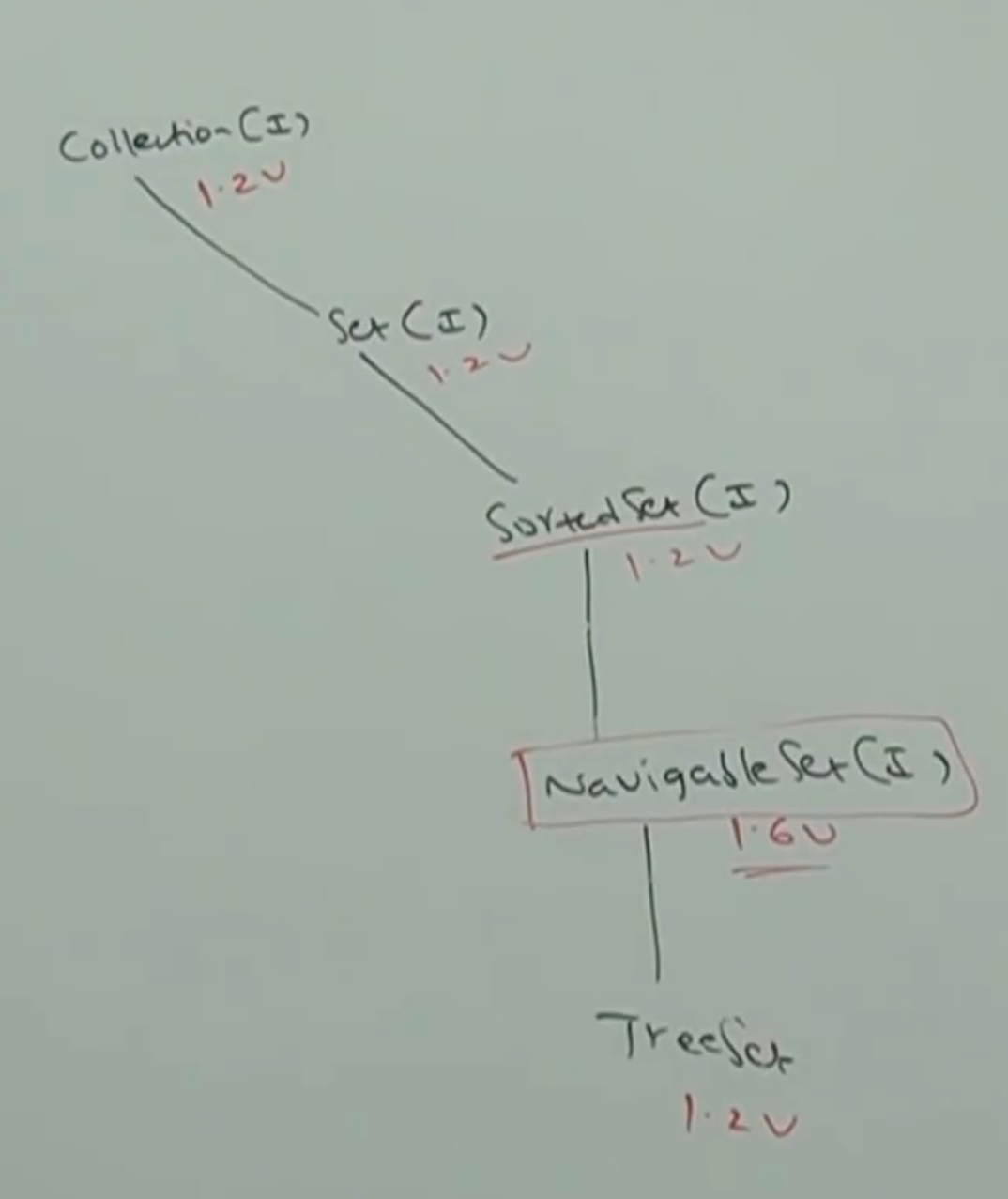
As part of 1.6 version the following two concepts introduced in Collection framework.

NavigableSet(I)

NavigableMap(I)

* **NavigableSet(I):**

It is the child interface of SortedSet and it defines several methods for navigation purposes.



* **NavigableSet methods:**

NavigableSet defines the following methods.

floor(e)

It returns highest element which is <=e;

lower(e)

It returns highest element which is <e

ceiling(e)

It returns lowest which is >= e

higher(e)

It returns lowest element which is > 2

pollFirst()

Remove and return first element.

pollLast()

Remove and return last element.

descendingSet()

It returns NavigableSet in reverse order.

* **Example:**

import java.util.\*;

class NavigableSetDemo{

public static void main(String[] args){

TreeSet<Integer> t = new TreeSet<Integer>();

t.add(1000);

t.add(2000);

t.add(3000)

t.add(4000);

t.add(5000);

System.out.println(t);[1000,2000,3000,4000,5000]

System.out.println(t.ceiling(2000)); // 2000

System.out.println(t.higher(2000)); //3000

System.out.println(t.floor(3000)); // 3000

System.out.println(t.lower(3000)); // 2000

System.out.println(t.pollFirst()); // 1000

System.out.println(t.pollLast()); // 5000

System.out.println(t.descendingSet()); // [4000,3000,2000]

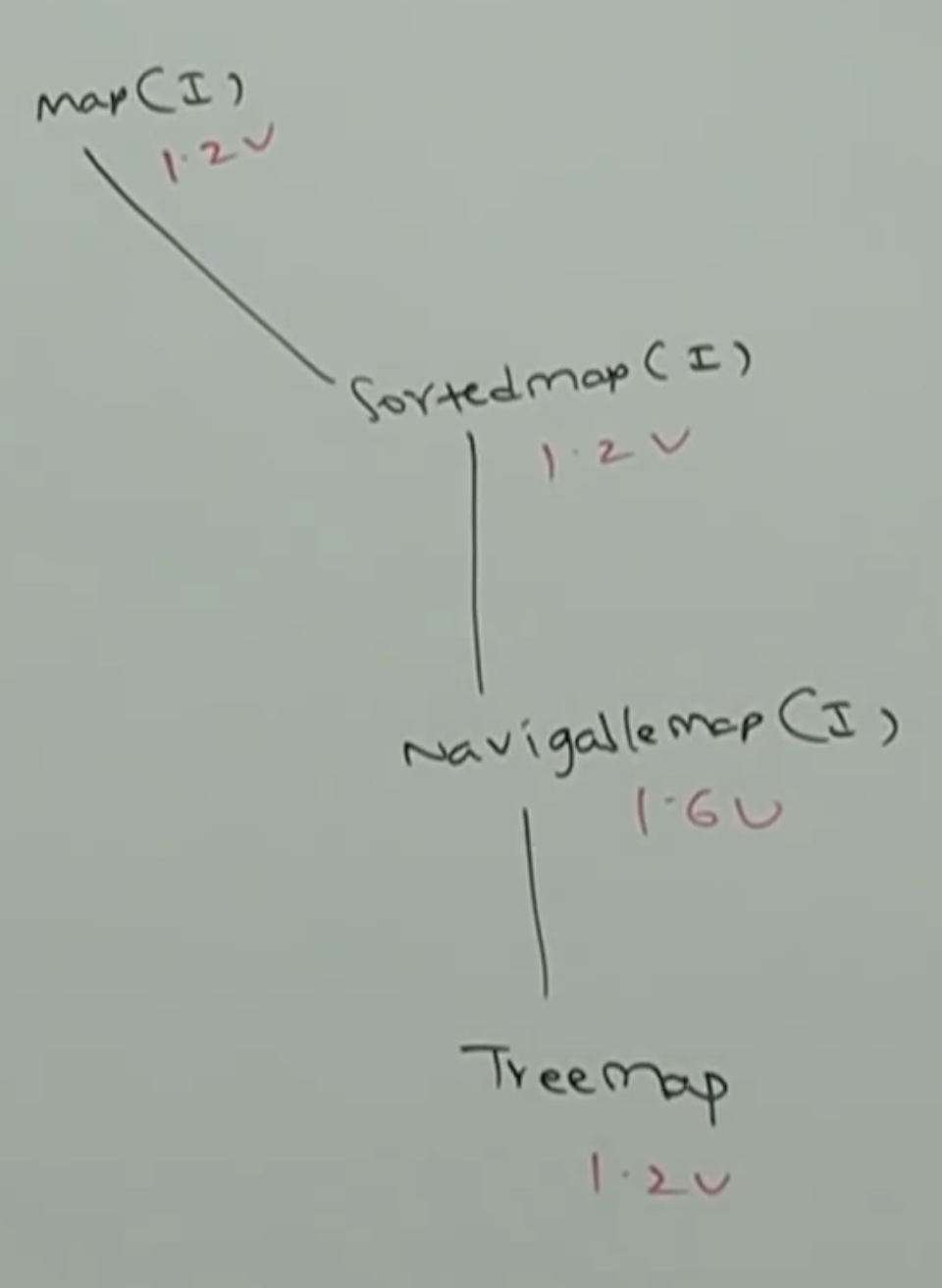
System.out.println(t); // [2000,3000,4000]

}

}

* **NavigableMap:**

NavigableMap is the child interface of SortedMap. It defines several methods for navigation purposes.

****

NavigableMap defines the following methods:

floorKey(e)

lowerKey(e)

ceilingKey(e)

higherKey(e)

pollFirstEntry(e)

pollLastEntry(e)

descendingMap()

* **Example:**

import java.util.\*;

class NavigableMapDemo{

public static void main(String[] args){

TreeSet<String, String> t = new TreeMap<String, String>();

t.put(“b”, “banana”);

t.put(“c”, “cat”);

t.put(“a”, “apple”);

t.put(“d”, “dog”);

t.put(“g”, “gun”);

System.out.println(t); // {a=apple,b=banana,c=cat,d=dog, g=gun}

System.out.println(t.ceilingKey(“c”)); // c

System.out.println(t.higherKey(“e”)); //g

System.out.println(t.floorKey(“e”)); // d

System.out.println(t.lowerKey(“e”)); // d

System.out.println(t.pollFirstEntry()); // a=apple

System.out.println(t.pollLastEntry()); // g=gun

System.out.println(t.descendingMap()); //{d=dog, c=cat, b=banana}

System.out.println(t); //{b=banana,c=cat,d=dog}

}

}

* **Collections:**

Collections class defines several utility methods for Collection objects like sorting, searching, reversing etc.

**Sorting elements of list:**

Collections class defines the following two sort() methods

public static void sort(List l)

To sort based on Default Natural Sorting Order.

In this case List compulsory contains homogeneous and comparable objects, otherwise we will get runtime exception saying: ClassCastException

List should not contain null, otherwise we will get runtime exception saying: NullPointerException

public static void sort(List l, Comparator c);

To sort based on customized sorting order.

* **Demo program for sorting elements of List according to default natural sorting order:**

import java.util.\*;

class CollectionSortDemo{

public static void main(String[] args){

ArrayList l = new ArrayList();

l.add(“Z”);

l.add(“A”);

l.add(“K”);

l.add(“N”);

//l.add(new Integer(10)); // CCE

//l.add(null); // NPE

System.out.println(“Before sorting:”+l);//[Z,A,K,N]

Collections.sort(l);

System.out.println(“After sorting:”+l);//[A,K,N,Z]

}

}

Output:

Before sorting: [Z, A, K, N]

After sorting: [A,K,N,Z]

* **Demo program to sort elements of list according to customized sorting:**

import java.util.\*;

class CollectionSortDemo{

public static void main(String[] args){

ArrayList l = new ArrayList();

l.add(“Z”);

l.add(“A”);

l.add(“K”);

l.add(“L”);

System.out.println(“Before sorting:”+l) //[Z,A,K,L]

Collections.sort(l, new MyComparator());

System.out.println(l); //[Z,L,K,A]

}

}

class MyComparator implement Comparator{

public int compare(Object ob1, Object obj2){

String s1 = (String) obj1;

String s2 = obj2.toString();

return s2.compareTo(s1);

}

}

Output:

Before sorting: [Z,A,K,L]

After sorting: [Z,L,K,A]