Set (I)

- It is sub interface of collection interface
- Does not allow duplicate values
- Insertion order not preserved
- Allow only 1 null value insertion
- Does not deal with array index operation
- It deal with hash table operation

1) Hash set

- It is implementation of set interface
- Only 1 null insertion is allowed
- Do not allow duplicate values
- Order of insertion => random
- Heterogenous
- **Storage type => hash table (key, values)**
- Data structure => hash table
- Searching easy so hashSet is best choice
- Default initial capacity 16
- Fill ration (load factor) 0.75
- It grows double when condition arrived (from 16 to 32)

Program:

```
package CollectiionProg;

import java.util.HashSet;

public class HashSetProg {

                public static void main(String[] args) {

                 HashSet a = new HashSet();                    //IC=16, LF=.75=75%
                 HashSet b = new HashSet(1000);            //IC=1000, LF=.75=75%
                 HashSet c = new HashSet(100,90);  //IC=100, LF=.90=90%
                 HashSet d = new HashSet(a);

                 a.add(10);
                 a.add(12);
                 a.add(20);
                 a.add(9);
                 a.add(1);
                 a.add(2);
                 a.add(5);
                 System.out.println(a);
                 a.add("Harry");
                 a.add("Json");
                 a.add("Army");
                 System.out.println(a);

                 a.add(20);
                 System.out.println(a);
```

```
        a.add(null);
        System.out.println(a);
        a.add(null);
        System.out.println(a);
    }
}
```

We used hash table to remove duplicate values and if order of insertion is not maintained
111%10 = 1 2344%10 = 4

Hashset

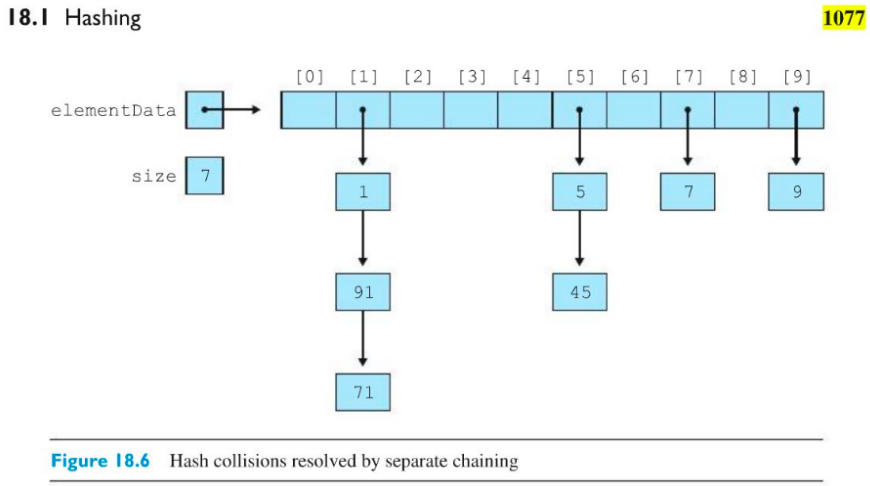

Figure 18.6 Hash collisions resolved by separate chaining

2) Linked Hash set
- It is implementation class of set interface
- Only one null insertion allowed
- Don't allow duplicate values
- Order of insertion is maintained
- Storage type => hash table
- default capacity 16
- Data structure => Hybrid

We use linked has set cause duplicate are not allowed and insertion order needs to be maintained

Program:

```
package CollectiionProg;

import java.util.LinkedHashSet;

public class LinkedHashSetProg {

        public static void main(String[] args) {
```

```
                        LinkedHashSet a = new LinkedHashSet();
                        LinkedHashSet b = new LinkedHashSet(900);
                        LinkedHashSet c = new LinkedHashSet(900, 99);
                        LinkedHashSet d = new LinkedHashSet(a);

                        a.add(10);
                        a.add(20);
                        a.add(2);
                        a.add(5);
                        a.add(11);
                        a.add(3);
                        System.out.println(a);
                        a.add("ZZZZZ");
                        a.add("AAAAAA");
                        a.add("BBBB");
                        a.add("A");
                        System.out.println(a);
                        a.add(10);
                        System.out.println(a);
                       }
}
```

3) Tree set

- It is implementation class of sorted set interface
- Don't allow duplicate values
- Order of insertion is ascending
- Storage type => hash table
- No default capacity
- Data structure => balanced tree
- No heterogenous

Program:

```
package CollectiionProg;

import java.util.TreeSet;

public class TreeSetProg {

                        public static void main(String[] args) {

                        TreeSet a = new TreeSet();
                        TreeSet b = new TreeSet();

//                      a.add(null);  //Not accepting :- NullPointerException
                        a.add("Z"); //100
                        a.add("E");
                        a.add("B");
                        a.add("Y");
//                      a.add(100);
```

```
                    System.out.println(a);

                    b.add(100);
                    b.add(90);
                    b.add(50);
                    b.add(30);
                    b.add(1);
                    System.out.println(b);
                    b.add("C");
                    System.out.println(b);
                   }
}
```

Note :- If we try to add null then it cannot compare with existing other values so it gives null pointer exception
It accept any one type of value (string or number) cause both cannot compare.

Cursor
Cursor is an interface and it is used to retrieve data from collection object, **one by one**
Types of cursor

1) Enumeration
- Applicable only for legacy class hence it is not universal cursor
- We can only perform read operation
- We can only move in forward direction
- It contain two method

Enumeration e = vectorObject.elements();
- hasMoreElements()
- nextElement()

2) Iterator
- We can apply iterator cursor for any collection object hence it is **universal** cursor
- We can perform read and remove operation
- We can move in forward direction hence it is single directional cursor

  Iterator i = v1.iterator()
- Methods
  - hasNext()
  - next()
  - remove()

3) List iterator
- It is only applicable for List iterator type implementation class
- We can perform read, remove, addition of new object and replacement operation
- We can move forward and backward direction hence it is bidirectional cursor
- Only applicable for list implementation classes so it is **not universal** cursor

Method:

hasNext()	hasPrevious()
next()	previous()
nextIndex()	nextPrevious()
remove()	
set(object)	
add(object()	

Program:

Enumeration	**Iterator**	**ListIterator**
Applicable for Legacy class	Any Collection	Only List classes
Only forward	Only forward	Forward and backward
Only read	Read and remove	Read, write, remove, replace
We can use elements() method	Iterator()	listIterator
hasMoreElement() nextElement()	hasNext() next() remove()	hasNext() hasPrevious() next() previous() nextIndex() nextPrevious() remove() set(object). add(object()

	Hash Set	**Linked Hash Set**	**Tree Set**
Implementation class	Set	Set	Set
Null insertion	Only 1	Only 1	Not allowed
Duplicate values	Not allowed	Not allowed	Not allowed
Order of insertion	Random	Maintained	Ascending
Data structure	Hash table	Hybrid (Hash table + Linked list)	Balanced tree
Storage type	Hash table	Hash table	Hash table
Default capacity	16	16	No
		Cash based app	