Collections

unit-4

- use, comparator, Iterator, Algo, for each loop, Collection view of map.

- Collections is a framework that p/v implementation of Data Structures, algos
- It is based on generics.
- It also p/v several algos like sort, search, max, min that can work on any data type.

Adv. of Collections :-

1) Consistent API
   - All collections (DS) use same fun like add(), remove etc.

2) Reduces effort
   - No need to implement code for various DS.

3) Reusability
   - Same DS can be used for diff. data types

4) Inc speed & quality.
   - Internally, the implementation of DS is highly optimised. So they r fast.

eg. of a collection :-

Creating an ArrayList.

```
import java.util.*;
CJ{ PSVM(){

ArrayList<String> al = new ArrayList<String>();
  al.add("abc");
  al.add("xyz");
  al.add("www");
  al.add("dej");
  ---------
  al.remove(1);   → index - xyz is removed
  al.remove("abc");   → el - abc is ...
  Sopln(al);
  ---------
//using Iterator to print arr:
Iterator<String> itr = al.iterator();
  while(itr.hasNext()){
      String s = itr.next();
      Sopln(s);
  }
}
```

```
//Creating a hashset.
HashSet<Integer> hs = new HashSet<Integer>();
hs.add(new Integer(5));
hs.add(10);
hs.add(new Integer(10));
-----------.
hs.remove(2);
          ↳ index
hs.remove(new Integer(2));
-----------
ListIterator litr = hs.listIterator(hs.size()-1);
while(litr.hasPrevious()){
    String s = litr.previous();
    sopln(s);
    litr.set(s+"+");
}
}
//list iterator can move in reverse dir..
```

Q) What is an iterator? Types? ***

A) · Iterator is similar to a ptr. It is used to access the elements of a Collection.

· 2 types — iterator() → can move in forward dir only.

— ListIterator() → can move in both forward & rev dir.
→ can modify elements.

Q) What r algos?

A • algos r commonly used fns in Collections
• 2 diff classes p/v algos - Collections & Arrays.

eg Collections.sort(al); → arraylist obj
Collections.shuffle(al);
Collections.binarySearch(al);
Collections.max(al);
" .min(al);

• Arrays

• It p/v some fns to manipulate arrays.

int ar[] = {10, 1, 2, 3, 5};
Arrays.binarySearch(ar, 10);
int ar2[] = Arrays.copyOf(ar, len);
Arrays.fill(ar.value);
Arrays.sort(ar);
Arrays.sort(ar, start, end);

Collections classes :- ***

1) ArrayList
- p/v arrays, wch r growable i.e size can be inc or dec.
- random access allowed.
- el r stored in insertion order.

2) LinkedList
- p/v implementation of linked list.
- random access allowed.
- el r stored in insertion order.

3) ArrayDequeue
- p/v implementation of Queue, Dequeue, ~~Arrays~~, Stack
- Queue - add/remove from start/end only
- Dequeue - " " " both " "
- Stack - add/remove from top.
- elements r stored in insertion order.

4) PriorityQueue
- p/v imp of a Queue, in wch el r stored based on priority.
- by def. el r stored in inc order.
- to reverse order, comparator can be used.

5) ~~Set~~ HashSet

- used to store unique elements.
- ~~this~~ hash table is used, so searching is fast.
- used when unique el r needed & fast search is needed.
- el stored in undefined order.

6) Linked HashSet

- used to store ~~el~~ ~~is~~ unique el & fast searching
- el r stored in insertion order as linked list used.

7) TreeSet

- ~~used~~ unique el, fast search
- el r stored in sorted order.

8) HashMap

- used to store key, value pair
- unique keys r allowed.
- fast search.
- el stored in undefined order.

9) LinkedHashMap

- key, value pair
- unique keys
- fast search
- el r stored in insertion order.

10) TreeMap

- ~~Unique~~ el, • Key, value pair
- ~~fast~~ • unique keys
  • fast search
  • el r stored in sorted order.

Comparator ***

- By default, PriorityQueue, TreeMap & TreeSet store el in inc order.
- To reverse the order comparator should be used.
- Create a custom comparator class, override the compare fun & pass comparator obj to the Constructor.

```
class mycomp implements Comparator<String> {
    //override compare fun.
    public int compare(String a, String b) {
        if(a.compareTo(b) > 0) {
            return -1;
        }
        else
            return +1;
    }
}
```

// Create treemap obj & pass the comparator obj to Constructor

```
TreeMap<String, Double> tm = new TreeMap<String, Double>
                                  (new mycomp());
```

pass comp. obj

```
tm.put("Hddd", 57);
tm.put("btt", 98);
tm.put("ddn", 36);
```

// Collection view of map :-

- map can't be accessed via iterator.
- obtain a collection view of map - convert it to a set.

```
-Set<Map.Entry<String, Double>>
        set = tm.entrySet();
```

→ convert to set

```
for(Map.Entry<String, Double> x : set){
    Sop ln(x.getKey());
    Sop ln(x.getValue());
}
```

// for each style to of loop

For each style of loop :-

```
for (int i=0; i<10; i++) {
        sopln(ar[i]);
}
```

} normal loop.

```
for (int x : ar) {
        sopln(x)
}
```

} for each style.
- No initialization needed
- No step condition "
- No inc needed.
- Can be used only for read not for write