28-08-2024

28 August 2024 06:01

Collection

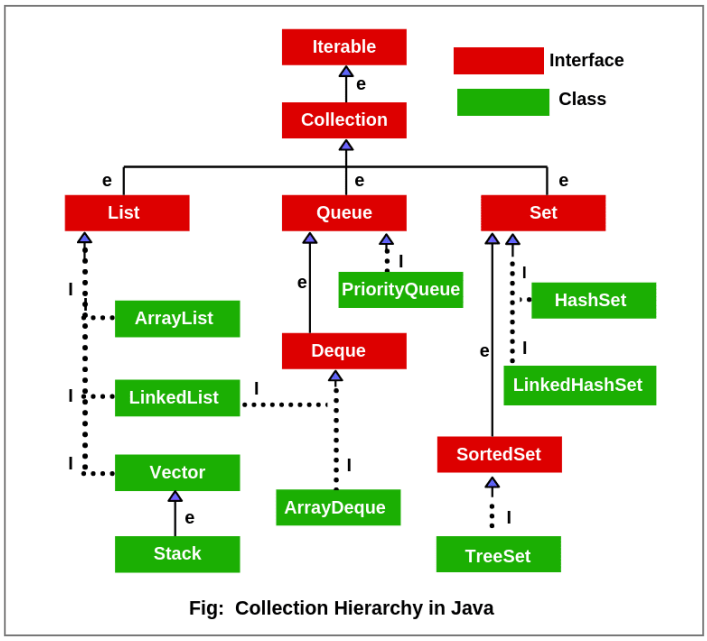


Fig: Collection Hierarchy in Java

ArrayList

How to create ArrayList

```
ArrayList<Integer> arr = new ArrayList<>();
List<Integer> arr1 = new ArrayList<>();
arr.add(10);
arr.add(20);
arr.add(30);
arr.get(1); // 20
. . . . .
. . . . . .
```

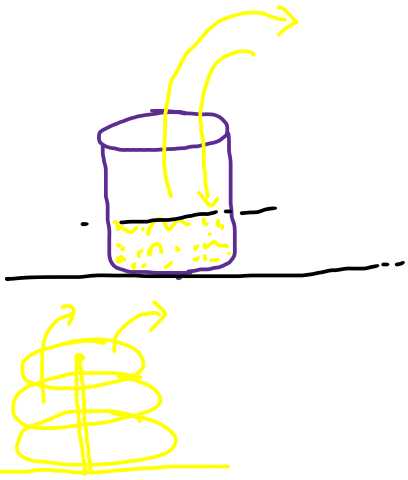
```
ArrayList<Integer> list = new ArrayList<>();

// add values
list.add(1);   // index - 0
list.add(2);   // index - 1
list.add(3);   // index - 2
list.add(4);   // index - 3
list.add(4);   // index - 4
list.add(4);   // index - 5


System.out.println(list.get(1));

// Set Element
list.set(1, 20);
System.out.println(list.get(1));

// True/False
System.out.println(list.contains(2));

// Last Element
System.out.println(list.lastIndexOf(4));

// index Of
System.out.println(list.indexOf(3));

// Remove
System.out.println(list.remove(5));


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

LinkedList

List<Integer> list = new LinkedList<>();

Same work as like ArrayList

Stack



Last in First out

[LIFO]

Push ⇒ add

Pop ⇒ remove

Peek ⇒ First ele

isEmpty ⇒ Stack is empty or not.

```
Stack<Integer> stack = new Stack<>();

int i = 0;

// insert operation
while(i < 5) {
   stack.push(i);
   i++;
}

System.out.println(stack);

while(!stack.isEmpty()) {
   System.out.println(stack.pop());
}

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

Stack<Integer> st = new Stack<>();

Priority Queue (Heap Sort)

Min Heap

Asc

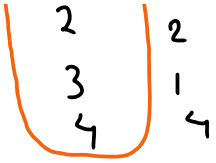
Max Heap

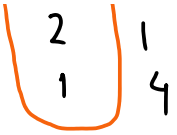
DSCE

1	3
2	2
3	1

4	3
3	2
2	1
1	4

```
PriorityQueue<Integer> minHeap = new PriorityQueue<>
();  // min Heap
      PriorityQueue<Integer> maxHeap = new
PriorityQueue<>(Collections.reverseOrder());  // min
Heap

      // minHeap
      minHeap.add(11);
      minHeap.add(2);
      minHeap.add(31);
      minHeap.add(4);

      // maxHeap
      maxHeap.add(5);
      maxHeap.add(6);
      maxHeap.add(7);
      maxHeap.add(8);


      System.out.println("Min Heap Element : " +
```






```
    maxHeap.add(8);


    System.out.println("Min Heap Element : " +
minHeap);
    System.out.println("Max Heap Element : " +
maxHeap);

    System.out.println("--- Min Heap Elements ---");
    while(!minHeap.isEmpty()) {
      System.out.println("Min Heap Element : " +
minHeap.remove());
    }

    System.out.println("--- Max Heap Elements ---");
    while(!maxHeap.isEmpty()) {
      System.out.println("Max Heap Element : " +
maxHeap.remove());
    }
```

Set

→ Contains only unique elements

→ [1,2,2,3,3,4,5,6] I/P

→ [1,2,3,4,5,6] O/P

```
Set<Integer> set = new HashSet<>();

set.add(1);
set.add(1);
set.add(1);
set.add(1);
set.add(1);
set.add(2);
set.add(3);
set.add(4);
set.add(5);

System.out.println(set);
System.out.println(set.contains(2));

System.out.println(set.remove(4));
System.out.println(set);
```

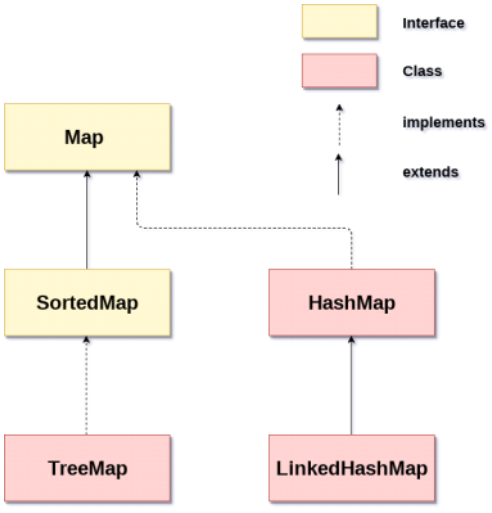
HashSet ⇒ unorder Set.

↳ [1,2,3,4] = [3,2,1,4]

LinkedHashSet ⇒ insert order ⇒ [1,2,3,4] ⇒ [1,2,3,4]

TreeSet ⇒ Sorted Set ⇒ [4,3,2,1] → [1,2,3,4]

Hash map



Map

⟨key, value⟩

Unique

Duplicate

Map<Integer, Integer> map = new HashMap<>();

```
int[] arr =
{1,1,1,3,4,56,6,5,4,23,2,5,5,5,3,32,2,4,5,45,6,6,4,75,
3};
Map<Integer, Integer> map = new HashMap<>
();

for(int x : arr) {
  map.put(x, map.getOrDefault(x, 0) + 1);
}

System.out.println(map);
for(Map.Entry<Integer, Integer> entry :
map.entrySet()) {
   System.out.println(entry.getKey() + " : " +
entry.getValue());
}
```