JAVA Collections Framework

1. ArrayList

- **Dynamic size**
- *Syntax* – **ArrayList<dataType> variableName = new ArrayList<>();**
- To **add element at the end of the list** use – **variableName.add()**
- To **add element at a specific index of the list** use – **variableName.add(index, newElement)**
- We can **add an entire list at once** –

  newList.add(50);
  newList.add(60);
  oldList.add(newList);
  Now the oldList contains all its own elements and the elements of oldList as well.
- To **get** an element from index – **list.get(index);**
- To **remove** an element from index – **list.remove(index);**
- To **remove** an element – **list.remove(Integer.valueOf(element));**
- To **remove all elements** from list – **list.clear();**
- **Time complexity of adding and removing elements in O(n)** as after every addition or removal elements shift one place.
- To **replace an element at a given index** – **list.set(index, element);**
- To **check if any element is present** or not – **list.contains(element);**
- To **traverse a arraylist** –
  - For loop
  - ForEach loop
  - Iterator –

    -
      ```
      Iterator<dataType> it = list.iterator();
      while(it.hasNext()) {
              it.next();
      }
      ```

2. Stack

- **LIFO**
- *Syntax* – **Stack<dataType> stackName = new Stack<>();**
- To **push element** in stack – **stack.push(element);**
- To **peek at the topmost element** – **stack.peek();**
- To **pop the topmost element** – **stack.pop()**

JAVA Collections Framework

3. Queue

- **FIFO**
- There are **two ends rear and front.**
- **Elements are added from the rear side** while **elements are removed from the front side.**
- Syntax - **Queue<dataType> queue = new LinkedList<>();**
- To **add an element - queue.offer(element);**
  offer returns true or false based on if the operation was successful or not.
- To **remove element - queue.poll();**
  poll returns null if the queue is empty.
- To **peek** the element which will be popped next - **queue.peek();**
  peek returns null if the queue is empty.

4. LinkedList

- Same as ArrayList. Uses all the same functions.
- Syntax - **List<dataType> linkedList = new LinkedList<>();**

5. PriorityQueue

- A **queue that gives priority to elements.**
- **Most use cases - minHeap, maxHeap.**
- Syntax - **Queue<dataType> pq = new PriorityQueue<>();**
  By default it becomes minHeap.
- To **add an element - pq.offer(element);**
- To **remove element - pq.poll(element);**
  By default the smallest element will be popped.
- To **peek** the element - **pq.peek();**
- To convert default minHeap priorityQueue to maxHeap -
  - **Queue<dataType> pq = new PriorityQueue<>(Comparator.reverseOrder());**

JAVA Collections Framework

6. ArrayDeque

- The **ArrayDeque in Java** provides a way to apply **resizable-array in addition to the implementation of the Deque interface.** It is also known as **Array Double Ended Queue or Array Deck.**
- Syntax - **ArrayDeque<dataType> adq = new ArrayDeque<>();**
- To add an element - **offer(element); / offerLast(element);**
- To **add an element** to the **front** - **adq.offerFirst(element);**
- To **peek** we have three functions - **adq.peek(); / adq.peekFirst(); / adq.peekLast();**
- To **poll** we have three functions - **adq.poll(); / adq.pollFirst(); / adq.pollLast();**

7. Set Interface

- Duplicate Elements not allowed.

8. HashSet

- Syntax - **Set<dataType> set = new HashSet<>();**
- Not ordered.
- To **add** - **set.add();**
- To **remove - set.remove(element);**
- To **check if a element is in the hashSet - set.contains(element);**
- To check if **set is empty - set.isEmpty();**

9. LinkedHashSet

- **Same as Hashset but ordered.**

10. TreeSet

- **Same as Hashset but ordered and sorted.**

JAVA Collections Framework

11. HashMap

- Syntax - **Map<dataType1, dataType2> hashMap = new HashMap<>();**
- To **add - hashMap.put(key, value);**
- Updates **value pair** if **put** is used with an already **existing key.**
- Traversing HashMap -
    - **for(Map.Entry<dataType1, dataType2> e: hashMap.entrySet()) {**

      ```
          sysout(e.getKey());
          sysout(e.getValue());
      }
      ```
- To **check if a key is in hashMap - hashMap.containsKey(key);**
- To **check if a value is in hashMap - hashMap.containsValue(value);**

12. TreeMap

- Same as **HashMap but ordered.**