

**List :** it extends collection framework and

* **it store elements in indexed approach**
* **We can add duplicate elements in the list.**
* **LIST MAINTAIN THE INSERTION ORDER**

Type of list:

1. **ArrayList**
2. **LinkedList**
3. **Vector**

**ARRAYLIST:**

ArrayList is the implementation of List Interface where the elements can be dynamically added or removed from the list. Also, the **size of the list is increased dynamically** if the elements are added more than the initial size.

ArrayList<Object> list2 = **new** ArrayList<Object>(); // it can store any type of objects

list2.add("johan");

list2.add(1);

list2.add(2.2222);

list2.add('A');

list2.add(10.1);

System.***out***.println("Iterating using for loop");

**for** (**int** i = 0; i < list2.size(); i++) {

System.***out***.println(list2.get(i));

}

System.***out***.println(“Size is ” +list2.size());

**Output:**

1

2.2222

A

10.1

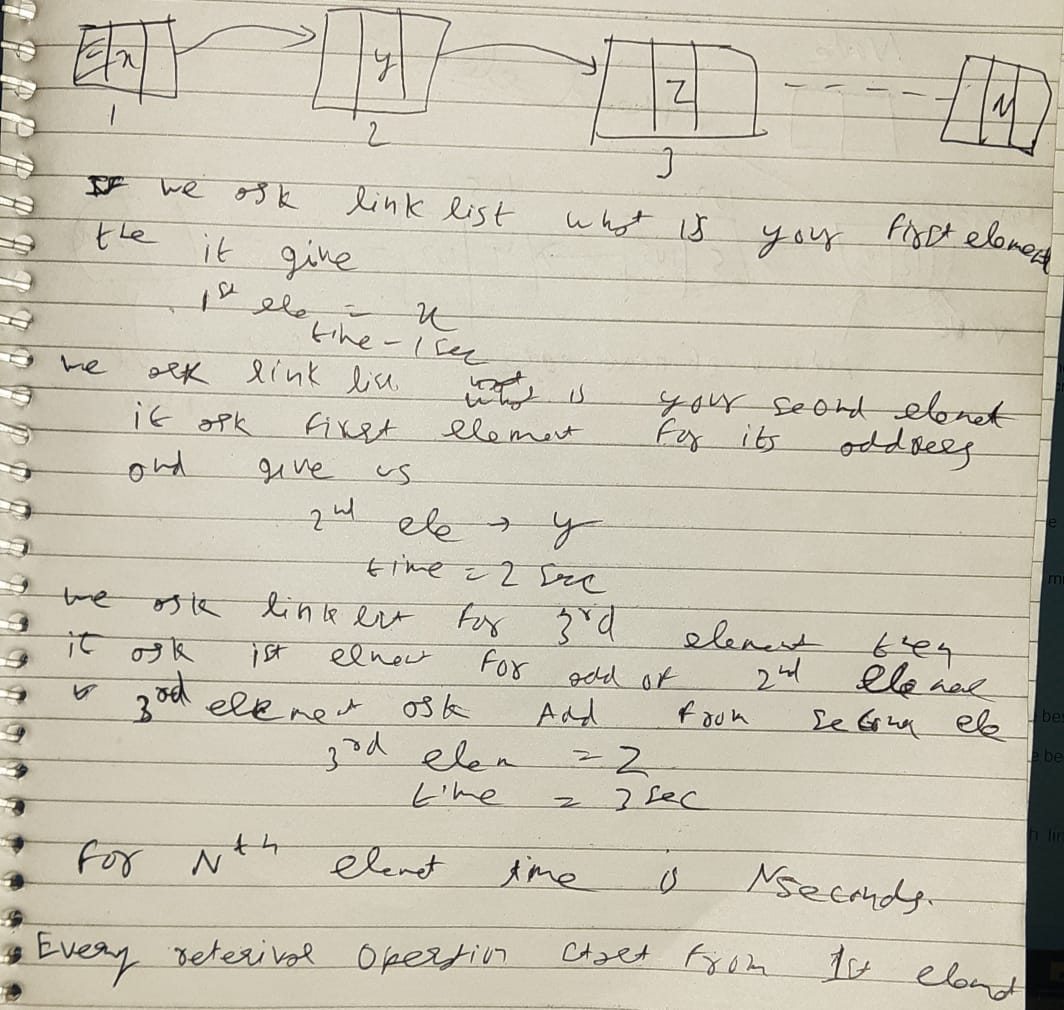
Size is: 4

Where arraylist is **best choice is retrival operations**, it do it fast because it work with index numbers

Where arraylist is worst choice is when we want to insert and delete in middle because we required many shift(move) operations/ it start from 0th index

**LINKEDLIST:**

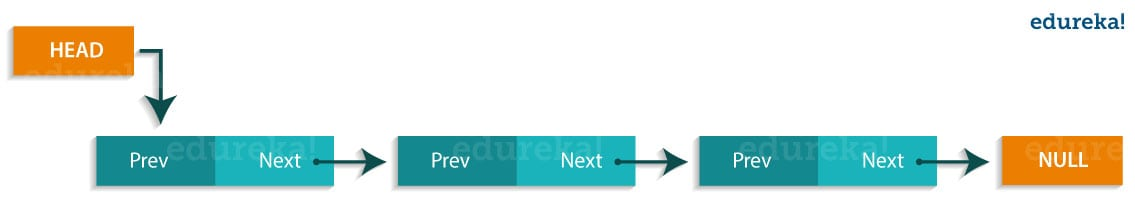
* If we want to insertion or deletion from middle, then link list is the best choice
* If our operation is retrieval/searching operation then linked list is worst choice because
* Linked List is a sequence of links which contains items. Each link contains a connection to another link.



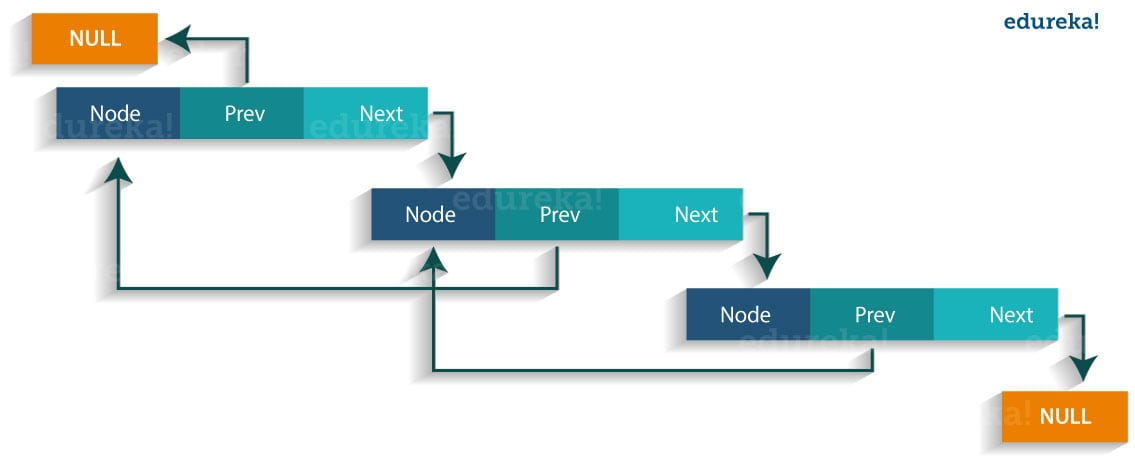
two types of Linked list to store the elements:

* Singly Linked List
* Doubly Linked List

**Singly Linked List**: In a singly linked list each node in this list stores the data of the node and a pointer or reference to the next node in the list. Refer to the below image to get a better understanding of single Linked list.

****

**Doubly Linked List**: In a doubly Linked list, it has two references, one to the next node and another to previous node. You can refer to the below image to get a better understanding of doubly linked list.

****

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

**LinkedList<String> al=new LinkedList<String>();// creating linked list**

**al.add("Rachit"); // adding elements**

**al.add("Rahul");**

**al.add("Rajat");**

**Iterator<String> itr = al.iterator();**

**while(itr.hasNext()){**

**System.out.println(itr.next());**

**}**

**}**

**}**

The output of the above program would be:

Rachit

Rahul

Rajat

**ArrayList vs LinkedList :**

|  |  |
| --- | --- |
| **ArrayList** | **LinkedList** |
| 1) ArrayList internally uses a **dynamic array** to store the elements. | LinkedList internally uses a **doubly**  **linked list** to store the elements. |
| 2) Manipulation with ArrayList is **slow** because it internally uses an array. If any element is removed from the array, all the bits are shifted in memory. | Manipulation with LinkedList is **faster** than  ArrayList because it uses a doubly linked  list, so no bit shifting is required in memory. |
| 3) An ArrayList class can **act as a list** only because it implements List only. | LinkedList class can **act as a list and queue**  both because it implements List and Deque  interfaces. |
| 4) ArrayList is **better for storing and accessing** data.  5) Arraylist is of one type  6) it is used for retrival operation | LinkedList is **better for manipulating** data.  5) Linkedlist is of two types.  6) it is used for insertion and deletion in between. |

**Vectors:**

* Vectors are similar to arraylist, where the elements of the vector object can be accessed via an index.
* the vector is not limited to a specific size, it can shrink or grow automatically whenever required. It is similar to ArrayList.

But the **difference is** vector increase its size by 100% whereas **Arraylist** increase its size by 50% it conclude vectors waste too much memory as compare to arraylist.

**public static void main(String aargs[]) {**

**Vector v = new Vector();**

**System.*out*.println(v.capacity());**

**}**

**OUTPUT:**

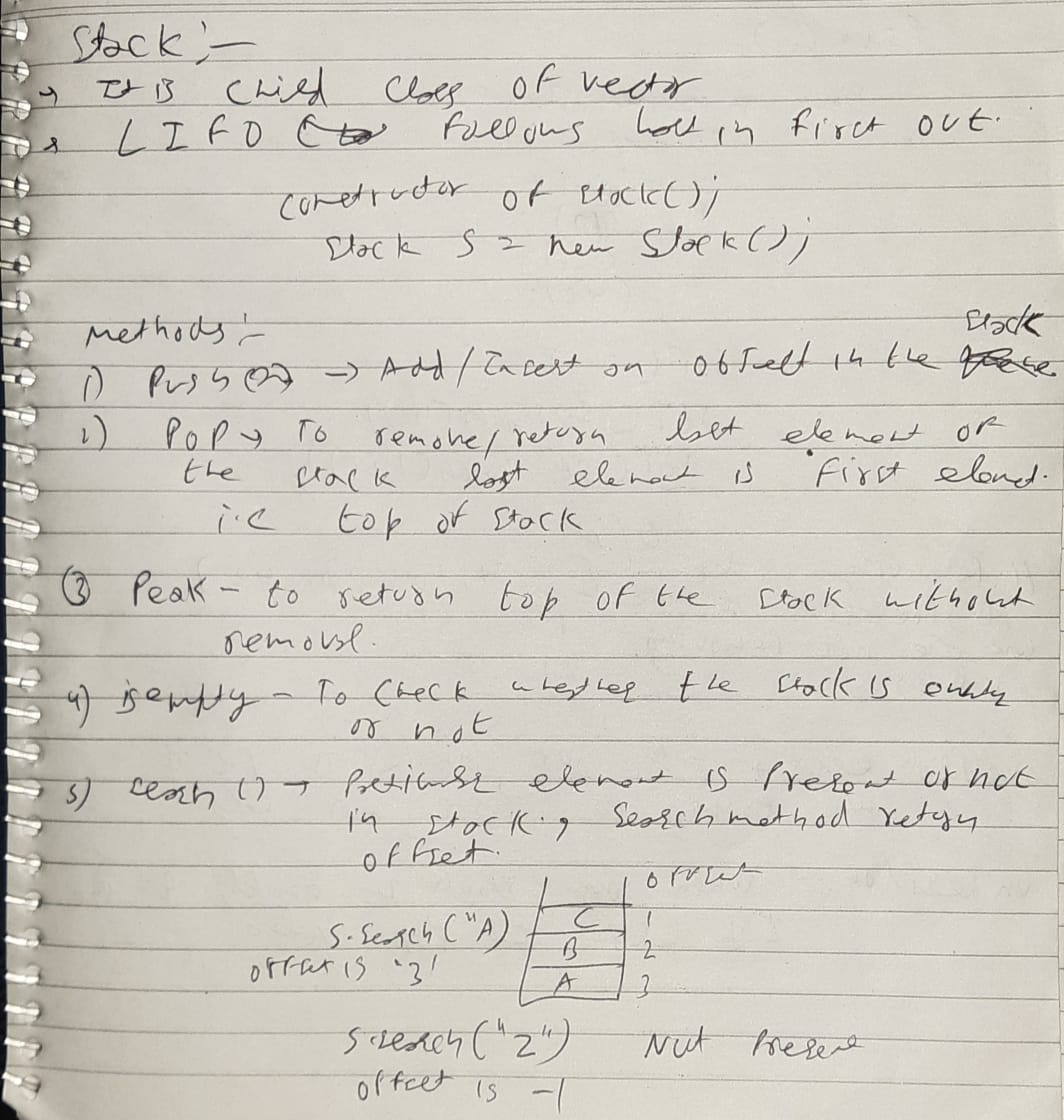
10 //even for 1 extra element it allocate a space of 10

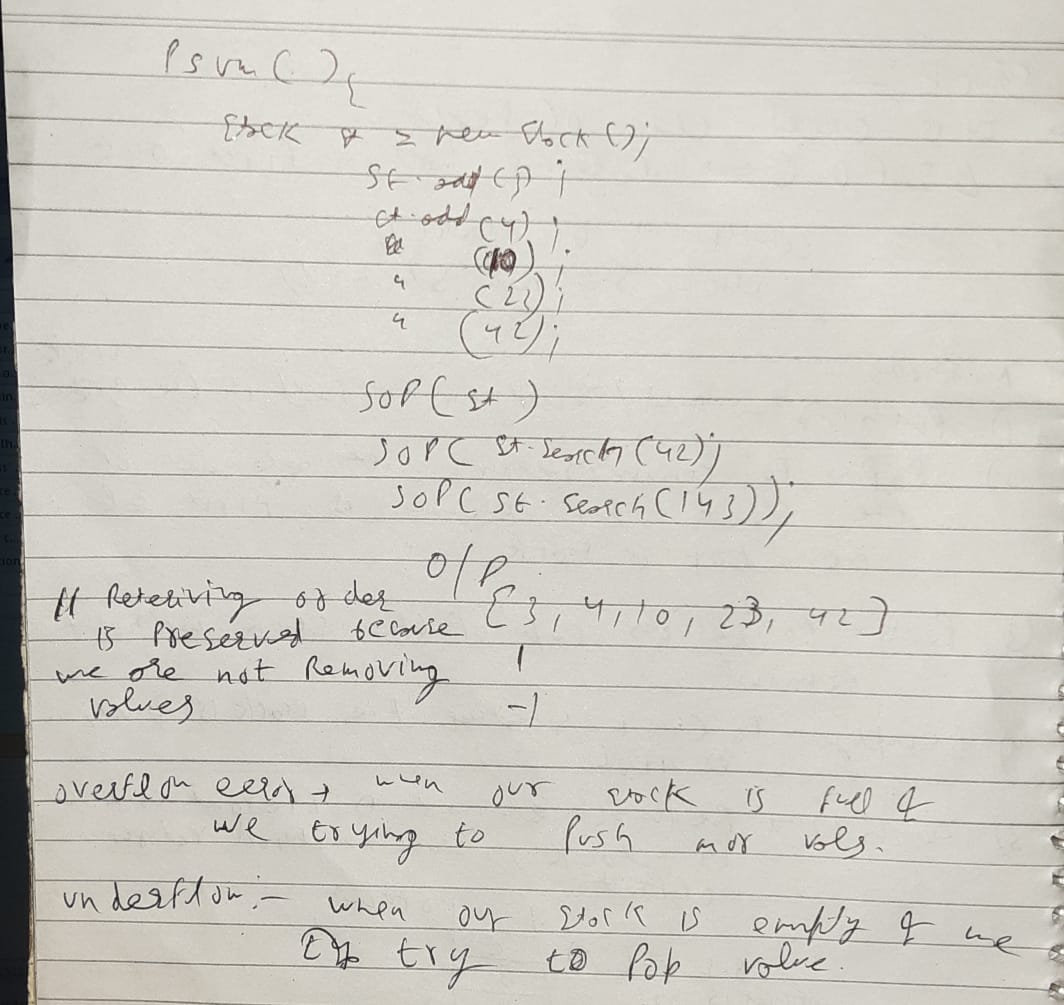
**But** vectors have few differences :

* **Vector is synchronized**, i.e. thread safe version of arraylist, which implies vectors are slow.
* vector increase its size by 100% whereas Arraylist increase its size by 50%.

**What is Stack:**

It is child class of vector.



****

**public** **static** **void** main(String aargs[]) {

Stack st = **new** Stack();

st.push(12);

st.add(3);

st.add(34);

st.add(45);

st.add(23);

st.add(10);

st.add(23);

st.pop(); // remove the element from the top of the stack

System.***out***.println(st.empty()); // return true if stack is empty

System.***out***.println(st.peek()); // Looks at the object at the top of this stack without removing itfrom the

// stack

System.***out***.println(st);

System.***out***.println("size of stack is " + st.size());

System.***out***.println(st.search(10));

System.***out***.println(st.search(123));

}

**OUTPUT:**

false

10

[12, 3, 34, 45, 23, 10]

size of stack is 6

1

## -1

## ****Queue :****

Queue in Java follows a FIFO approach i.e. it orders the elements in First in First out manner.

**Each basic method exists in two forms: one throws an exception if the operation fails, the other returns a special value**.

* **QUEUE NOT ALLOW INSERTION OF NULL ELEMENTS**  although some implementations, such as LinkedList , do not prohibit insertion of **null**
* **QUEUE ALLOW DUPLICATE VALUES**
* The Queue is used to insert elements at the end of the queue and removes from the beginning of the queue. It follows FIFO concept.

**Types of Queue:**

Priority Queue

Dequeue

ArrayBlockingQueue

**Priority Queue:**

* A Priority Queue is used when the objects are supposed to be processed based on the priority,  **priority queue** are ordered according to their natural ordering or by a Comparator provided at queue construction time.

**Comparator interface:** It help us to give priority at construction time(suppose we made two classes Student and children we can define priority of both according to our need)

* **PriorityQueue** doesn't allow **null values** to be **stored** in the **queue**.
* The queue retrieval operations poll, remove, peek, and element access the element at the head of the queue.
* In priority queue polling/removal of element take place on its priority basis

**public** **static** **void** main(String args[])

{

// Creating empty priority queue

PriorityQueue<Integer> pQueue = **new** PriorityQueue<Integer>();

//Queue<Integer> pQueue = new LinkedList<>();

// Adding items to the pQueue using add()

pQueue.add(2);

pQueue.add(10);

pQueue.add(3);

pQueue.add(9);

pQueue.add(2);

pQueue.add(14);

pQueue.add(12);

System.***out***.println(pQueue);

System.***out***.println(pQueue.poll());

System.***out***.println(pQueue.poll());

System.***out***.println(pQueue.poll());

System.***out***.println(pQueue.poll());

// Printing the top element of PriorityQueue

System.***out***.println(pQueue.peek());

// Printing the top element and removing it

// from the PriorityQueue container

System.***out***.println(pQueue.poll());

// Printing the top element again

System.***out***.println(pQueue.peek());//it shows the element jo next bahar nikalne vaala hai

}

**OUTPUT:**

[2, 2, 3, 10, 9, 14, 12] //element in pri queue is present in any order

2 // but at the time of removal it is ordered

2

3

9

10

10

12

**We can use Priority queue in that case suppose we have 10000 integers and we want to fetch top 5 smallest integers so, in that case we simply poll the 5 elements and we got top 5 smallest elements**

**ArrayBlockingQueue:**

* **BlockingQueue** doesn't accept null **values and throw** NullPointerException **if you try to store null value**
* **Size of the Queue is fixed**. Once created, the capacity cannot be changed.
* Attempts to **put an element into a full queue** will result in the **operation blocking.**
* Similarly attempts to take an **element from an empty queue** will also be **blocked**.
* This queue orders elements **FIFO (first-in-first-out)**

**public** **static** **void** main(String[] args)

{

**int** capacity = 15;

ArrayBlockingQueue<Integer> abq = **new** ArrayBlockingQueue<Integer>(capacity);

// add numbers

abq.add(2);

abq.add(10);

abq.add(3);

abq.add(9);

abq.add(2);

abq.add(14);

abq.add(12);

// print queue

System.***out***.println("ArrayBlockingQueue:"+ abq);

System.***out***.println(abq.poll());

System.***out***.println(abq.peek());

System.***out***.println("Remaining capacity is : "+abq.remainingCapacity());

}

**OUTPUT:**

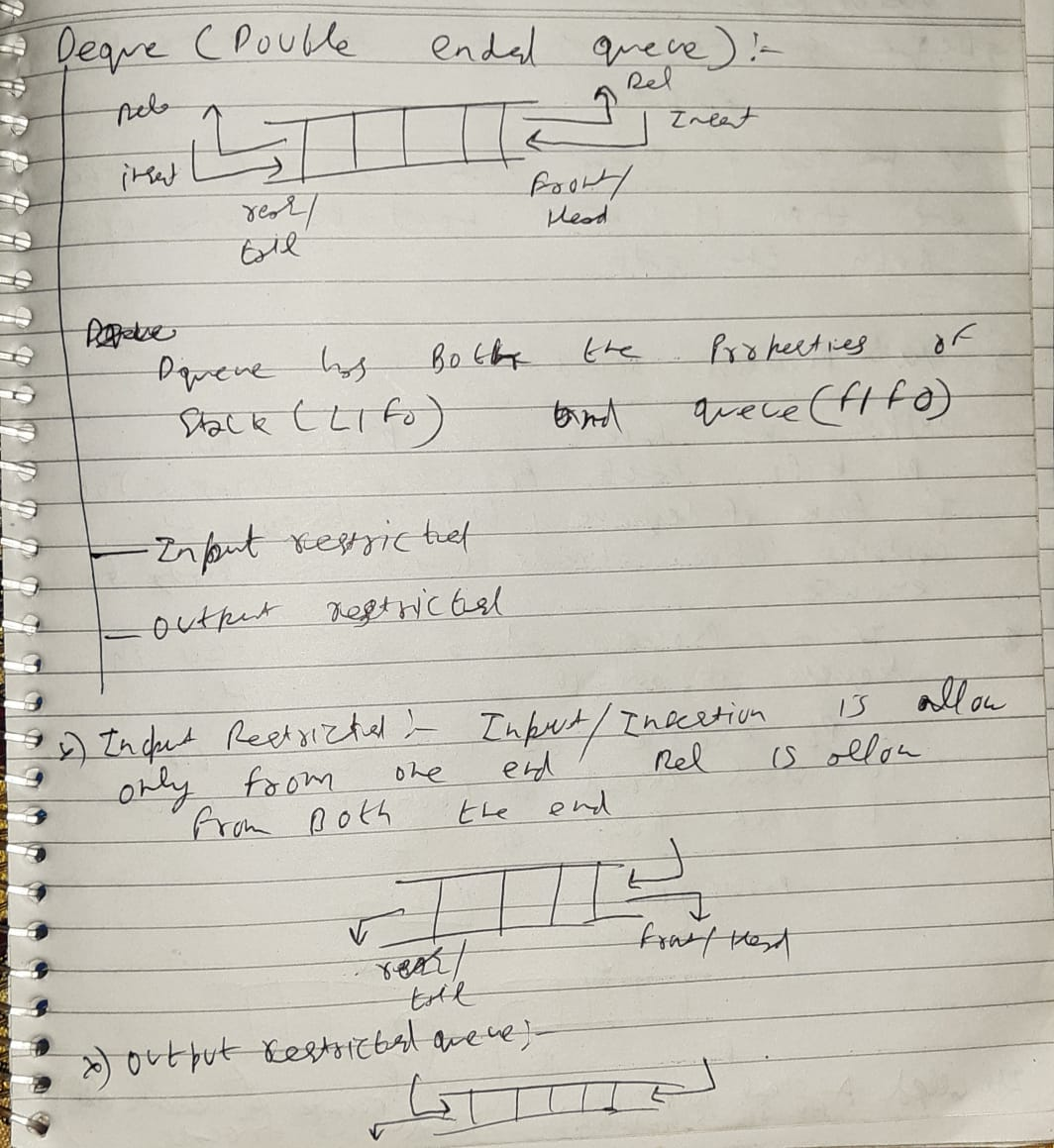
ArrayBlockingQueue:[2, 10, 3, 9, 2, 14, 12]

2

10

Remaining capacity is : 9

**Deque interface:**

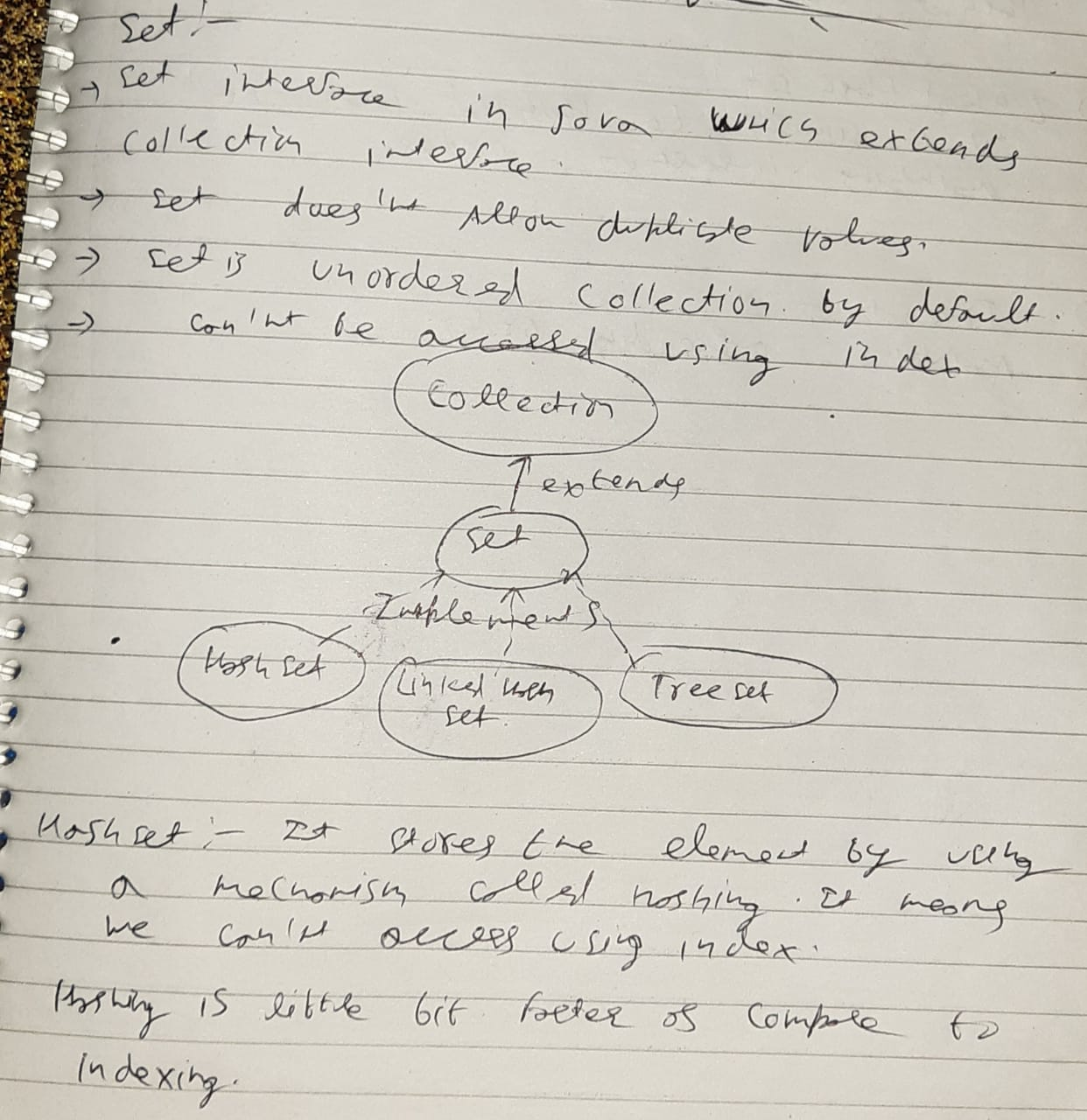
****

* Array **deques prohibit** the use of **Null** elements and do not accept any such elements.
* Deque is **not thread-safe** or non- synchronization

**Set:**

**Not Null allow in tree set** (It give null pointer exception when we add null ), HashSet and LinkedHash Set allow one null value ignore the other null values if we add moe than one null.

**Duplicate are not allow in set ,** It simply ignore the other duplicates we will not get an exception

****

Types of set:

Hash Set

Linked HashSet

Tree Set

Hash Set :

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

HashSet<String> hset = **new** HashSet<String>();

hset.add("jack");

hset.add("sam");

hset.add("john");

hset.add("jimmy");

hset.add("nick");

hset.add("drew");

hset.add("drew");

hset.add("drew");

hset.add("drew");

**for** (String abs : hset) {

System.***out***.println(abs);

}

# }

# OUTPUT:

nick

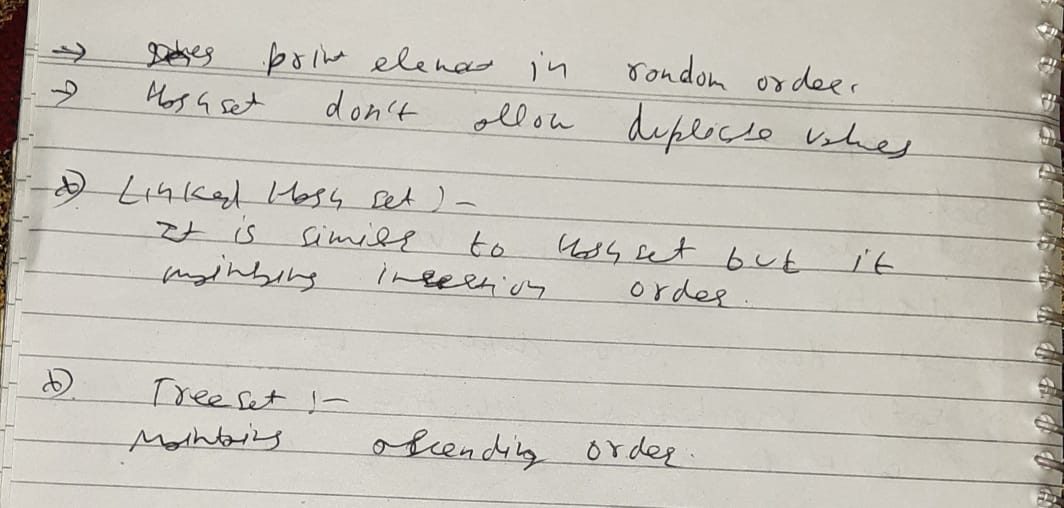
john

jimmy

jack

sam

drew //return duplicate only once

****

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

LinkedHashSet<String> lhs = **new** LinkedHashSet<String>();

lhs.add("test");

lhs.add("rest");

lhs.add("api");

lhs.add("selemium");

lhs.add("appium");

**for**(String abc : lhs) {

System.***out***.println(abc);

}

# }

# OUTPUT:

test

rest

api

selemium

appium

**Tree Set:**

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

TreeSet<Integer> teset = **new** TreeSet<Integer>();

teset.add(4);

teset.add(56);

teset.add(2912);

teset.add(93);

teset.add(34);

teset.add(53);

teset.add(34);

**for**(Integer abc : teset) {

System.***out***.println(abc);

}

System.***out***.println("@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@");

Iterator it = teset.descendingIterator();

**while**(it.hasNext()) {

System.***out***.println(it.next());

}

# }

# OUTPUT:

4

34

53

56

93

2912

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

2912

93

56

53

34

4

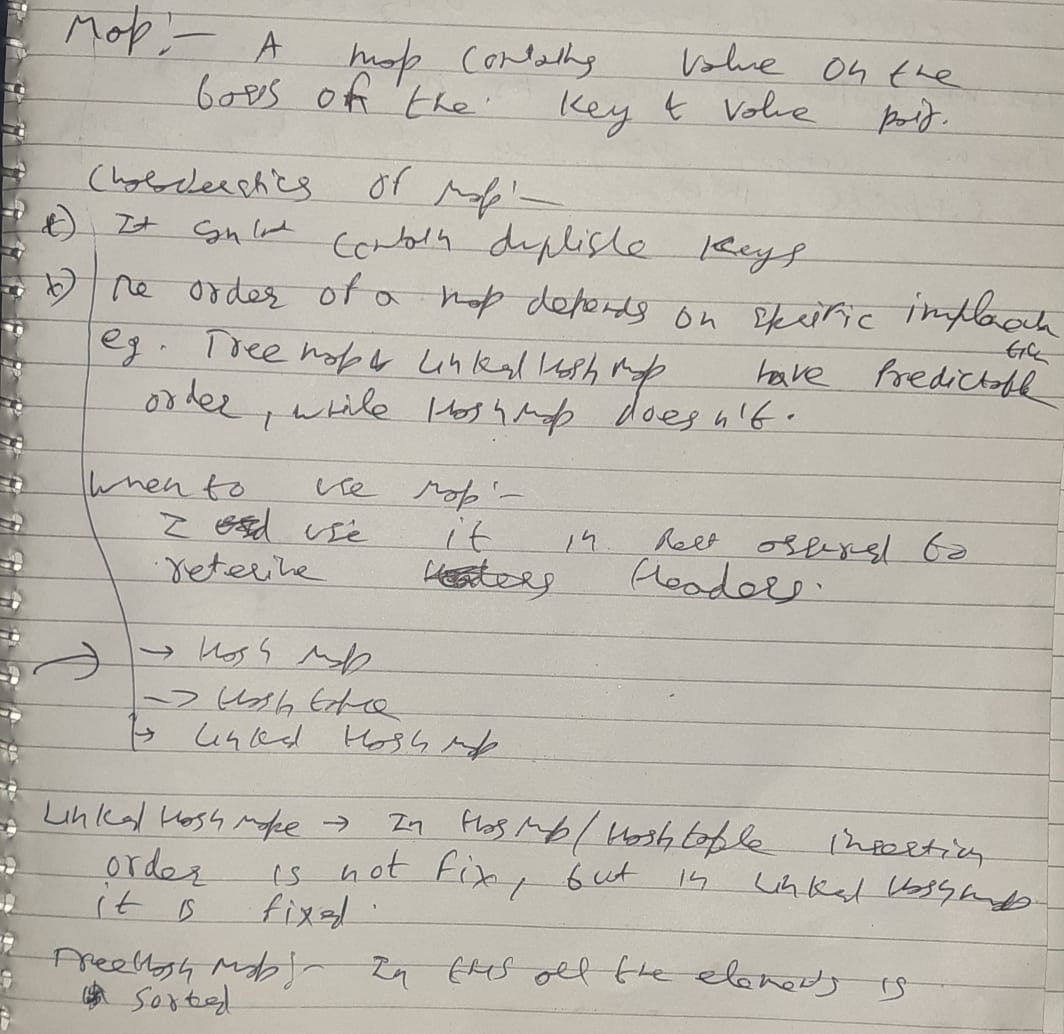
**Difference between HashSet and TreeSet.**

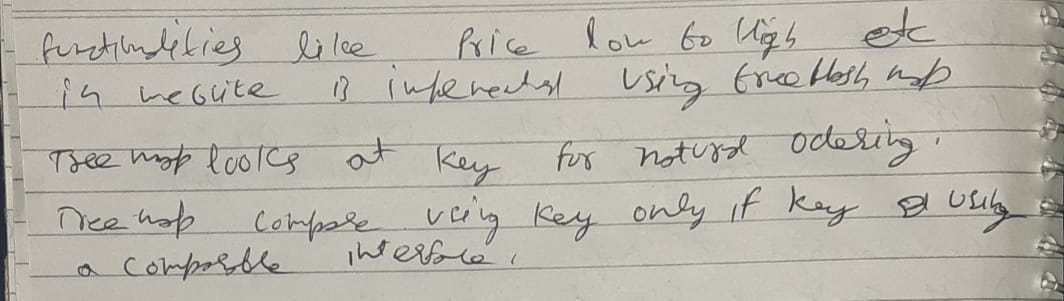
**Answer:** **Difference between HashSet and TreeSet can be seen below:**

| **HashSet** | **TreeSet** |
| --- | --- |
| Inserted elements are in random order | Maintains the elements in the sorted order |
| Can able to store null objects  hset.add(**null**); | Couldn’t store null objects  teset.add(**null**); |
| Performance is fast | Performance is slow(it take time to sort the elements) |

**MAP:**

**DUPLICATE ELEMENTS NOT ALLOWED**

****

****

**Difference between HashMap and HashTable.**

| **HashMap** | **HashTable** |
| --- | --- |
| Methods are not synchronized i.e Not  thread safe | Key methods are synchronized i.e  thread safe |
| Introduce in JDK 1.2 | Introduce since java is introduce |
| Performance is high than HashTable | Performance is slow |
| Allows one null key and multiple null values  hmap.put(**null**, **null**);  hmap.put(75, **null**);  hmap.put(98, **null**); | Doesn’t allow anything that is null  ht.put(**null**, **null**);Exception in thread "main" java.lang.NullPointerException  at java.util.Hashtable.put(Unknown Source)  at collection.hashtable.main(hashtable.java:19) |

**Difference between ArrayList and HashSet in Java?**

| **ArrayList** | **HashSet** |
| --- | --- |
| ArrayList implements List interface | HashSet implements Set interface |
| ArrayList allows duplicates | HashSet doesn’t allow duplicates |
| ArrayList is an ordered collection and maintains insertion order of elements | HashSet is an unordered collection and doesn’t maintain insertion order |
| ArrayList is backed by an Array | HashSet is backed by an HashMap instance |
| ArrayList is an index based | HashSet is object based |
| To put an object into array we need to specify the index.  name[1] = “book” | No index required. Store element using hashing technique  name.add(“book”) |

**What is meant by Ordered and Sorted in collections?**

**Answer:**

**Ordered:**It means the values that are stored in a collection is based on the values that are added to the collection. So we can iterate the values from the collection in a specific order.

**Sorted:**Sorting mechanisms can be applied internally or externally so that the group of objects sorted in a particular collection is based on the properties of the objects.

### What is the difference between Iterator and ListIterator?

|  |  |  |
| --- | --- | --- |
| **No.** | **Iterator** | **ListIterator** |
| 1) | The Iterator traverses the elements in the forward direction only. | ListIterator traverses the elements in backward and  forward directions both. |
| 2) | The Iterator can be used in List, Set, and Queue. | ListIterator can be used in List only. |

### What is the difference between List and Set?

The List and Set both extend the collection interface. However, there are some differences between the both which are listed below.

* The List can contain duplicate elements whereas Set includes unique items.
* The List is an ordered collection which maintains the insertion order whereas Set is an unordered collection which does not preserve the insertion order.
* The List interface contains a single legacy class which is Vector class whereas Set interface does not have any legacy class.
* The List interface can allow n number of null values whereas Set interface only allows a single null value.

**Legacy class in java?**

The classes which introduce in java 1st version is called legacy class.

For ex: Vector

**Homogenous objects and Heterogeneous objects?**

Homogenous objects: Same type objects

Heterogeneous Objects: Different type objects

In array list we can store homogenous and heterogeneous object(using Object class)

### What is the difference between HashSet and TreeSet?

The HashSet and TreeSet, both classes, implement Set interface. The differences between the both are listed below.

* HashSet maintains no order whereas TreeSet maintains ascending order.
* One null is allowed in HashSet, in TreeSet it throw null pointer exception.

### What is the difference between Set and Map?

The differences between the Set and Map are given below.

* Set contains values only whereas Map contains key and values both.
* Set holds a single number of null value whereas Map can include a single null key with n number of null values.

### What is the difference between HashSet and HashMap?

The differences between the HashSet and HashMap are listed below.

* HashSet contains only values whereas HashMap includes the entry (key, value).
* HashSet implements Set interface whereas HashMap implements the Map interface
* HashSet cannot have any duplicate value whereas HashMap can contain duplicate values with unique keys.
* HashSet contains the only single number of null value whereas HashMap can hold a single null key with n number of null values.

### What is the difference between HashMap and TreeMap?

The differences between the HashMap and TreeMap are given below.

* HashMap maintains no order, but TreeMap maintains ascending order.
* HashMap may contain a null key with multiple null values whereas TreeMap cannot hold a null key but can have multiple null values.

### What is the difference between HashMap and Hashtable?

|  |  |  |
| --- | --- | --- |
| **No.** | **HashMap** | **Hashtable** |
| 1) | HashMap is not synchronized. | Hashtable is synchronized. |
| 2) | HashMap can contain one null key and multiple null values. | Hashtable cannot contain any null key or null value. |

### What is the advantage of Properties file?

it makes the application easy to manage. It is used to store information which is to be changed frequently.

Properties prop = **new** Properties();

FileInputStream fis = **new** FileInputStream(System.getProperty("user.dir") + "/src/main/java/com" + "/qa/configuration/config.properties");

prop.load(fis);

### [What is the difference between Array and ArrayList?](https://www.javatpoint.com/array-vs-arraylist-in-java)

|  |  |
| --- | --- |
| **Array** | **ArrayList** |
| 1 | The Array is of fixed size, means we cannot resize the array as per need. | ArrayList is not of the fixed size we can change the  size dynamically. |
| 2 | Arrays are of the static type. | ArrayList is of dynamic size. |
| 3 | Arrays can store primitive data types as well as objects. | ArrayList cannot store the primitive data types it can  only store the objects. |

# Differences between Collection and Collections in Java?

* The **Collection**is an interface whereas **Collections**is a **utility**class in Java.
* The **Set, List,** and **Queue**are some of the subinterfaces of **Collection**interface, a **Map**interface is also part of the **Collections**Framework, but it doesn't inherit **Collection**interface.
* The important methods of **Collection**interface are **add(), remove(), size(), clear()**etc and **Collections**class contains only **static**methods like **sort(), min(), max(), fill(), copy(), reverse()** etc.

 public static void main(String args[]) {

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

      // Adding elements to the ArrayList

      list.add(5);

      list.add(20);

      list.add(35);

      list.add(50);

      list.add(65);

Collections.min() method to display minimum value

      System.out.println("Minimum value: " + Collections.min(list));

Collections.max() method to display maximum value

      System.out.println("Maximum value: " + Collections.max(list));

   }

## Output

Minimum value: 5

Maximum value: 65

**Iterator vs Iterable:**

**Iterator** helps us to iterate over a group or collection of objects

It has two main methods:

hasNext(): return true if we have next object

next(): help to move next object in a collection

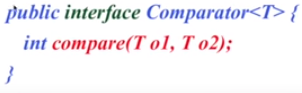
**Iterable:**

It is a interface which is extended by collection interface.

**Comparator vs Comparable[use for sorting]**

They both are interface

* **Comparator** has a method which name has **compare**



* **Comparable** has a method which name has **compareTo**



### [How to reverse ArrayList?](https://www.javatpoint.com/how-to-reverse-arraylist-in-java)

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

     List list = **new** ArrayList<>();

     list.add(10);

     list.add(50);

     list.add(30);

     Iterator i = list.iterator();

     System.out.println("printing the list....");

**while**(i.hasNext())

     {

         System.out.println(i.next());

     }

     Iterator i2 = list.iterator();

     Collections.reverse(list);

     System.out.println("printing list in reverse order....");

**while**(i2.hasNext())

     {

         System.out.println(i2.next());

     }

    }

### [How to sort ArrayList in descending order?](https://www.javatpoint.com/how-to-sort-java-arraylist-in-descending-order)

Comparator cmp = Collections.reverseOrder();

    Collections.sort(list,cmp);

     System.out.println("printing list in descending order....");

     Iterator i2 = list.iterator();

**while**(i2.hasNext())

     {

         System.out.println(i2.next());  }

### [How to remove duplicates from ArrayList?](https://www.javatpoint.com/how-to-remove-duplicates-from-arraylist-in-java)

There are two ways to remove duplicates from the ArrayList.

* **Using HashSet:** By using HashSet we can remove the duplicate element from the ArrayList, but it will not then preserve the insertion order.
* **Using LinkedHashSet:** We can also maintain the insertion order by using LinkedHashSet instead of HashSet.

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

List<String> l = **new** ArrayList<String>();

l.add("Mango");

l.add("Banana");

l.add("Mango");

l.add("Apple");

System.***out***.println(l);

Set<String> s = **new** LinkedHashSet<String>(l);

System.***out***.println(s);

}

### [What is the difference between the length of an Array and size of ArrayList?](https://www.javatpoint.com/difference-between-length-of-array-and-size-of-arraylist-in-java)

The length of an array can be obtained using the property of length whereas ArrayList does not support length property, but we can use size() method to get the number of objects in the list.

**Finding the length of the array**

Int [] array = **new** **int**[4];

System.out.println("The size of the array is " + array.length);

**Finding the size of the ArrayList**

ArrayList<String> list=**new** ArrayList<String>();

list.add("ankit");

list.add("nippun");

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

**Generics in java?**

It is use to achieve type safety

List<String> list = new ArrayList<String>();

We make it generic by using <String> now it accepts only integer values