1. Java Collection Framework
2. Collection Interface
3. Interfaces that extends Collection Interface :
4. Map Interface
5. Comparable
6. Comparators

Advanced DSA : Contest 3

↓

Fri 24 Nov

9 - 10:30 PM

↓

1.5 hrs → 3 Qns

Syllabus

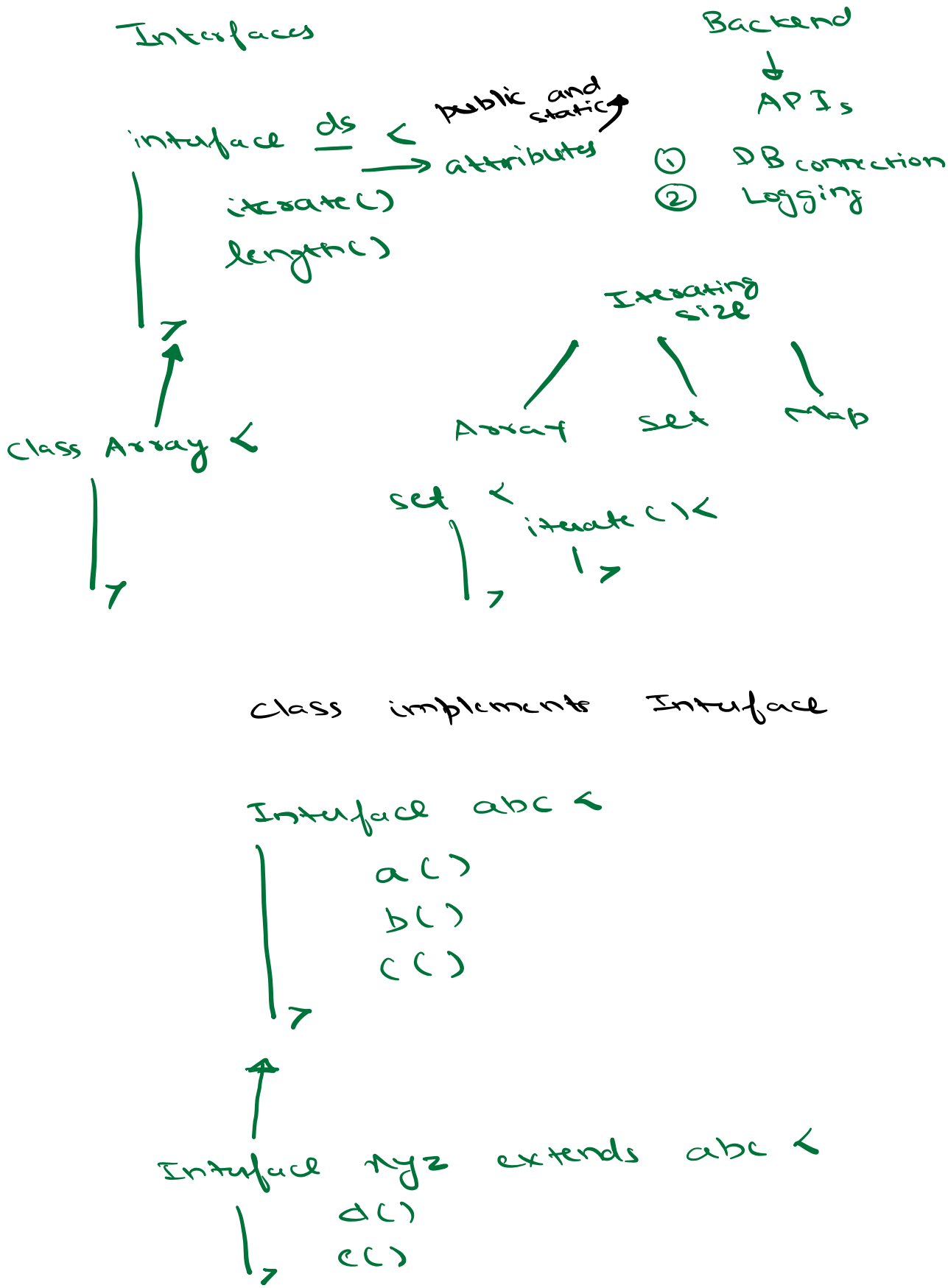
↓

Sorting, Searching and 2 pointers

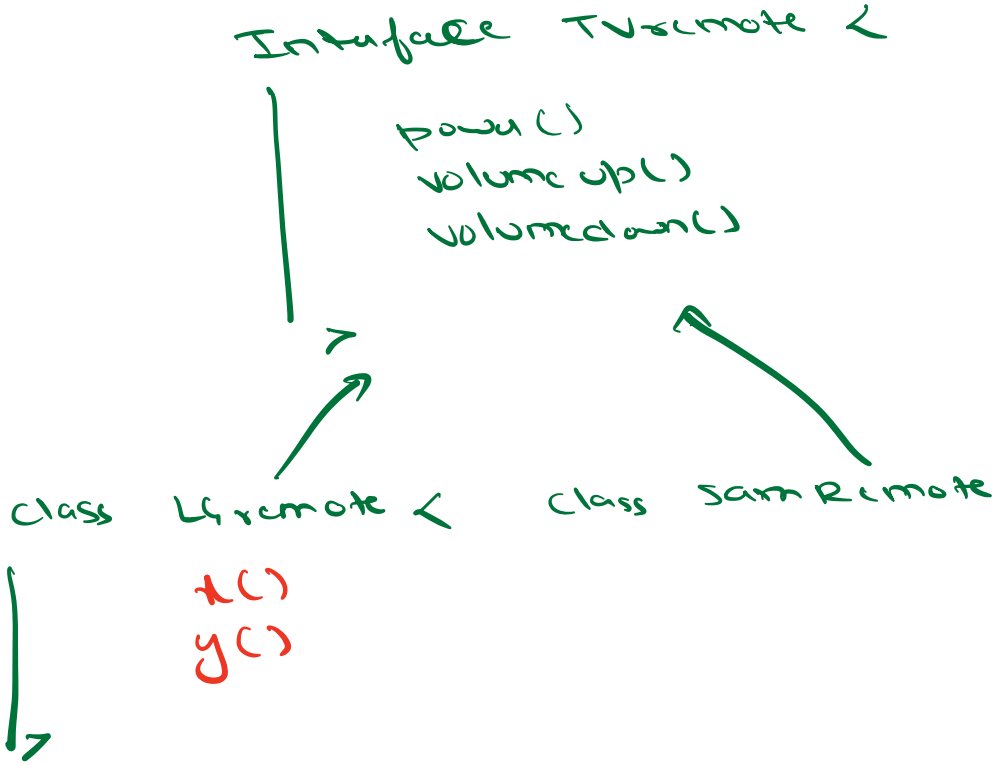
# Java Collection Framework

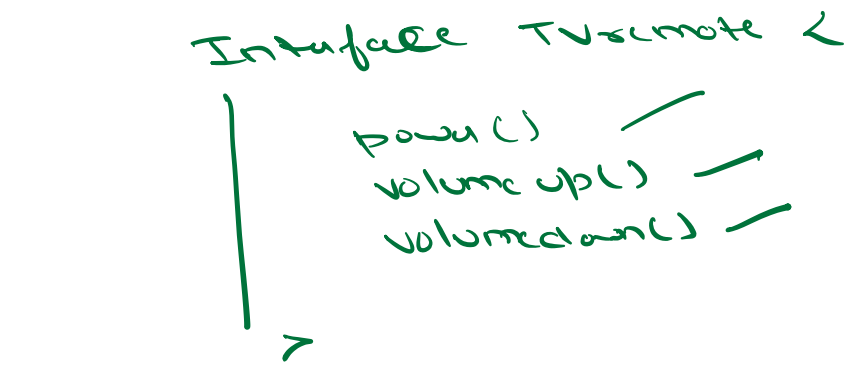
Collection - group of individual objects that are represented as a single unit

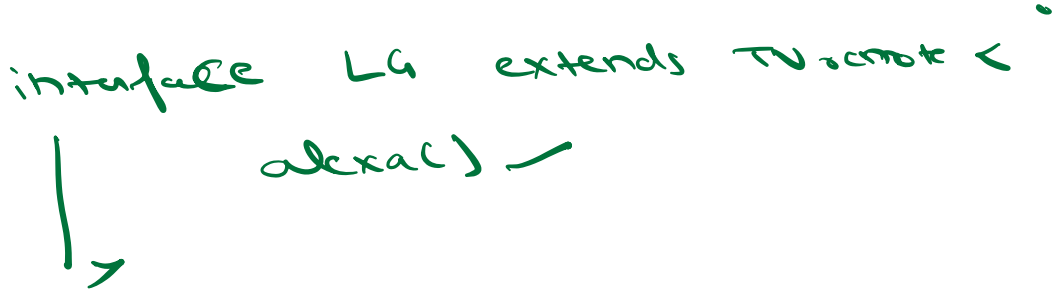
Framework - set of classes and interfaces which provide a ready made architecture

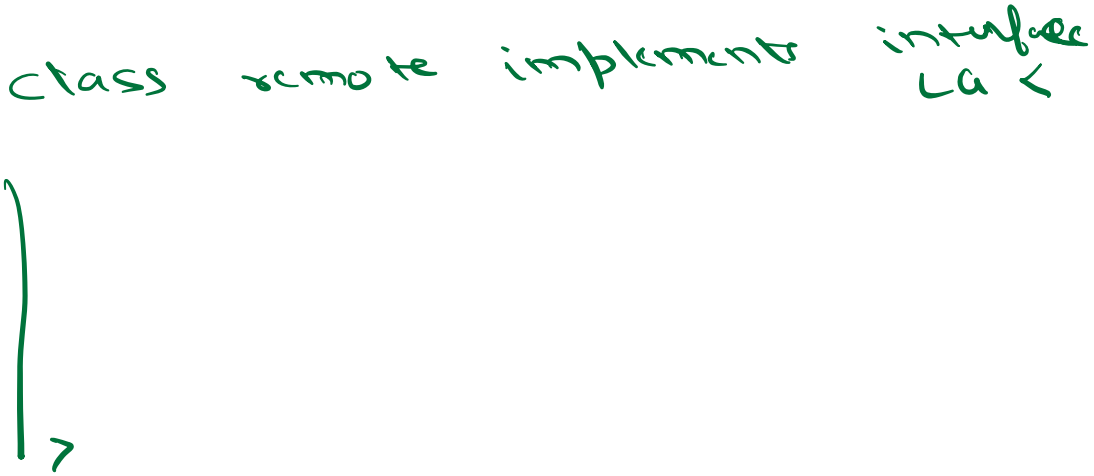

Interfaces
Backend
APIs
interface ds
public and static
attributes
① DB connection
② Logging
iterate()
length()
Iterating size
Array
Set
Map
Class Array
set
iterate()
class implements Interface
Interface abc
a()
b()
c()
Interface xyz extends abc
d()
e()

Class implement xyz

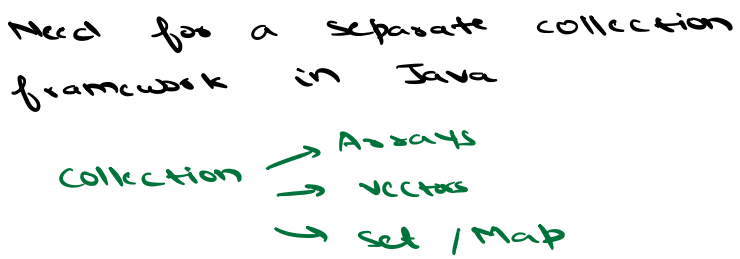


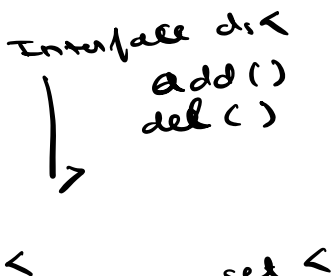





**The Java Collections Framework (JCF)** is a set of classes and interfaces that implement commonly reusable collection data structures like **List, Set, Queue, Map, etc**. The JCF is organized into interfaces and implementations of those interfaces. The interfaces define the functionality of the collection data structures, and the implementations provide concrete implementations of those interfaces.



```
// Java program to demonstrate
// why collection framework was needed
import java.io.*;
import java.util.*;

class CollectionDemo {

    public static void main(String[] args)
    {
        // Creating instances of the array,
        // vector and hashtable
        int arr[] = new int[] { 1, 2, 3, 4 };
        Vector<Integer> v = new Vector();
        Hashtable<Integer, String> h = new Hashtable();

        // Adding the elements into the
        // vector
        v.addElement(1);
        v.addElement(2);

        // Adding the element into the
        // hashtable
        h.put(1, "geeks");
        h.put(2, "4geeks");

        // Array instance creation requires [],
        // while Vector and hastable require ()
        // Vector element insertion requires addElement(),
        // but hashtable element insertion requires put()

        // Accessing the first element of the
        // array, vector and hashtable
        System.out.println(arr[0]);
        System.out.println(v.elementAt(0));
        System.out.println(h.get(1));

        // Array elements are accessed using [],
        // vector elements using elementAt()
        // and hashtable elements using get()
    }
}
```



class List
|→ add()

|→ add()

## Advantages of Java collection Framework

↓

STL in C++

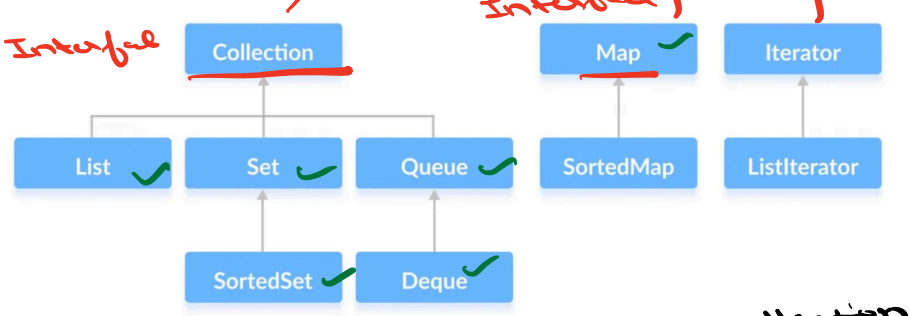
Collection in Python

1. Consistent API
2. Reduce programming effort
3. Increases program speed and quality

Collections <
|
=
|→

Package → java.util

Iterable interface <
|→ iterator()




List
/ \
Static Array
[] List

Collection
/
Interface List

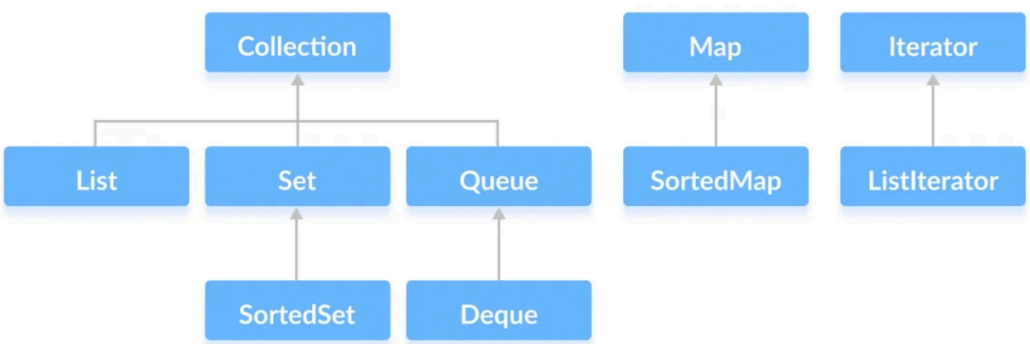
# Methods of Collection Interface

1. add()
2. size()
3. remove()
4. iterator()
5. addall()
6. removeall()
7. clear()

hs → 1, 2, 3, 4

ar → 1, 2, 3, 4, 5, 6

ar.removeall(hs)




List → stores data in a list format
grows linearly
Duplicate data

Collection
↓
List

List Interface:

ordered collection of objects
duplicate values are stored

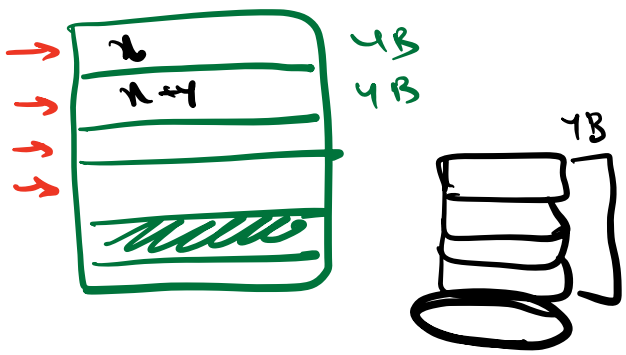
insertional, positional access

Classes implement List Interface

1. ArrayList — Resizable array implementation of List Interface
2. Vector — Synchronized resizable array
3. Stack — LIFO order

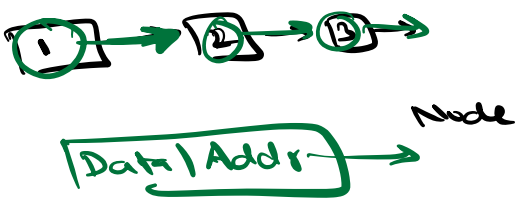
4. Linked List

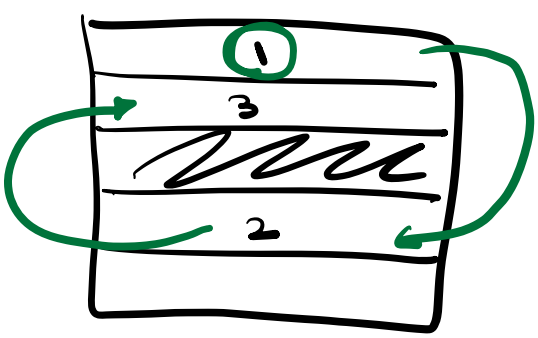
int ar[4]

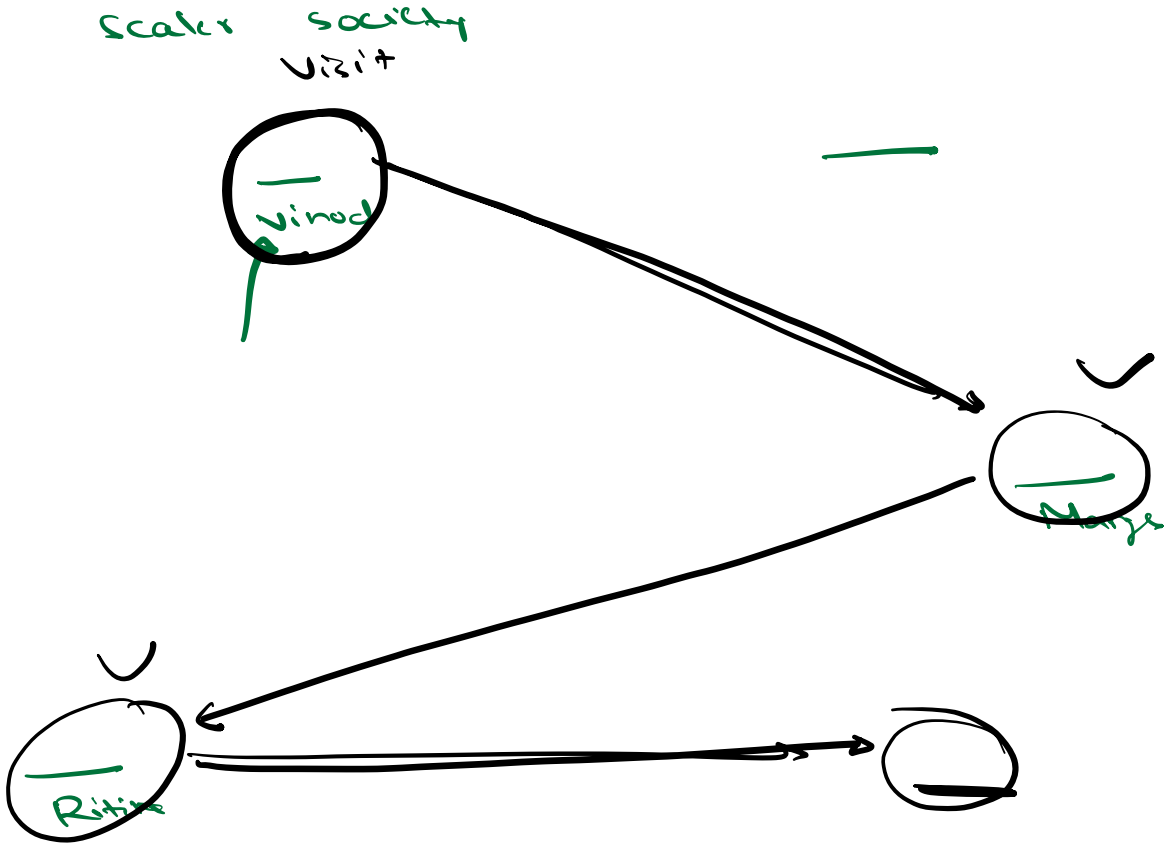



int ar[2]

LL





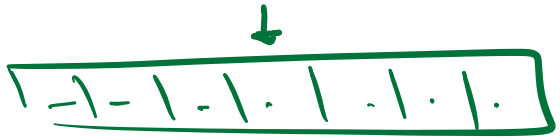

## ArrayList

↓

Class works with dynamic list

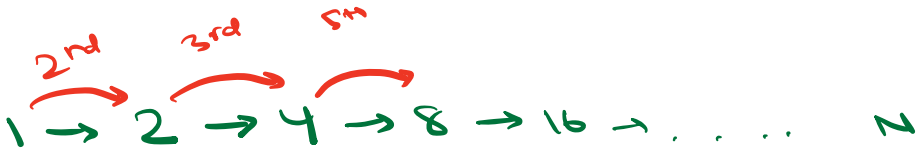
1. Backing array → default size of an underlying array (10)
2. Resizing
3. Dynamic sizing
4. Access by index → O(1)
5. Insertion/deletion → move multiple elements O(N)

allowed = 10
size = 10



11

capacity = 20
size = 11




Total no. of operations = $1 + 2 + 4 + \ldots N$

$= 2N - 1$

$\simeq 2N$

N insertions → 2N operation

1 insertion → 2 operation

Amortized TC

1 insertion → O(1)

RO
Filter → 90

Jan → 30
Feb → 30
Mar → 30

1 bottle = ₹ 10

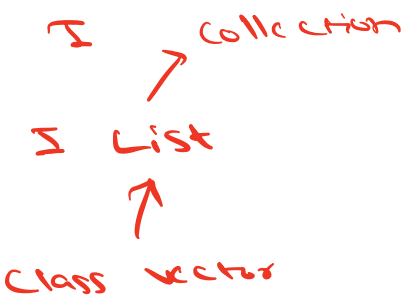
Jan → ₹ 50
Feb → 30
Mar → 100

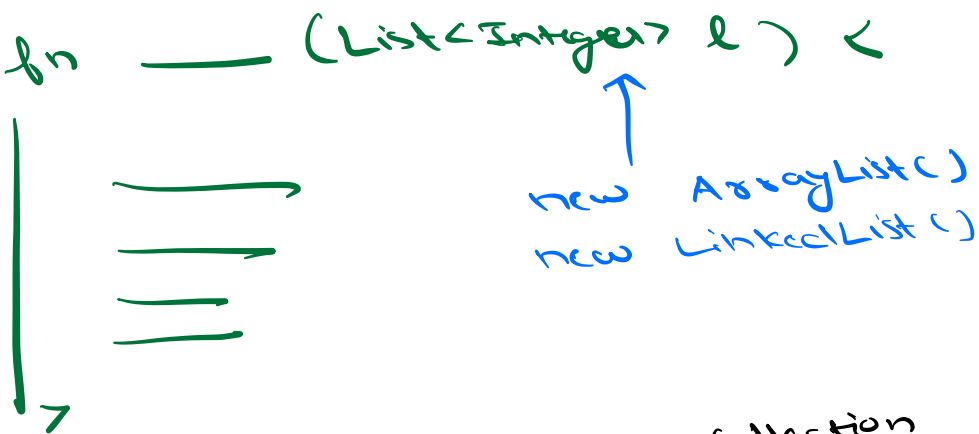
10 : 37

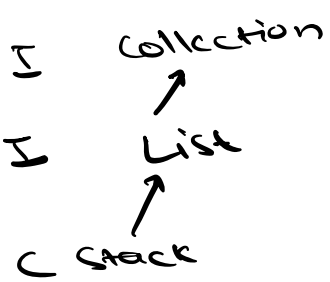
Vector → Resizable array (synchronized)

```java
// Java program to demonstrate the
// creation of list object using the
// Vector class

import java.io.*;
import java.util.*;

class ListObjectUsingVector {
    public static void main(String[] args)
    {
        // Size of the vector
        int n = 5;

        // Declaring the List with initial size n
        List<Integer> v = new Vector<Integer>(n);

        // Appending the new elements
        // at the end of the list
        for (int i = 1; i <= n; i++)
            v.add(i);

        // Printing elements
        System.out.println(v);

        // Remove element at index 3
        v.remove(3);

        // Displaying the list after deletion
        System.out.println(v);

        // Printing elements one by one
        for (int i = 0; i < v.size(); i++)
            System.out.print(v.get(i) + " ");
    }
}
```

List<Integer> v = new ArrayList<Integer>();



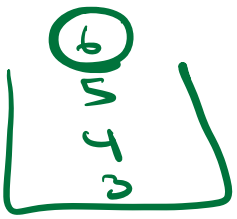
Stack  Last In First out

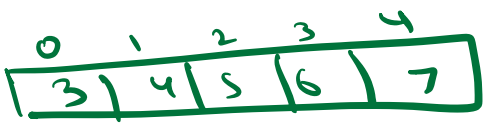
add → push
del → pop

List<Integer> s = new Stack<Integer>()

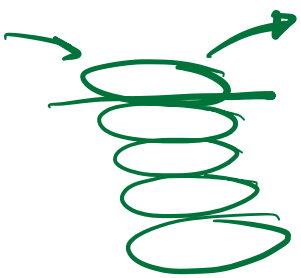
s.add(3), 4, 5, 6

s.remove()

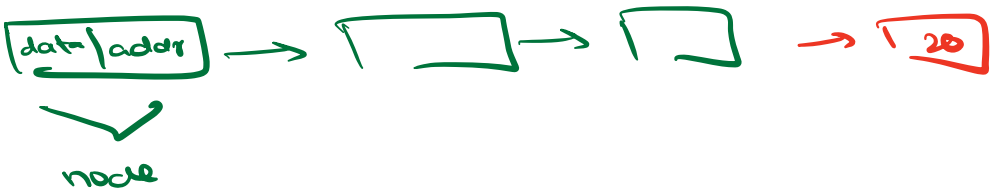



s.get(3)



Linked List



Set → represent unordered set of elements
no duplicates allowed

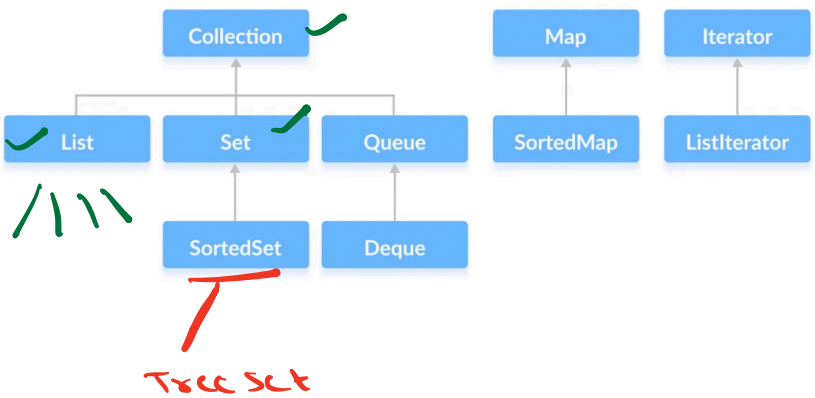

```java
import java.util.*;

// Main class
class HashSetClass {

    // Main driver method
    public static void main(String[] args)
    {
        // Creating an object of Set and
        // declaring object of type String
        Set<String> hs = new HashSet<String>();

        // Adding elements to above object
        // using add() method
        hs.add("B");
        hs.add("B");
        hs.add("C");
        hs.add("A");

        // Printing the elements inside the Set object
        System.out.println(hs);   → A, C, B
    }
}
```

LinkedHashSet → HashSet + maintaing insertion order

lh.add("India")
lh.add("Australia")
lh.add("SA")

print(lh) → India, Australia, SA

Sorted Set Interface → no duplicates + maintain natural order




Asc order → int

A/B/C → Strings

SortedSet (Person)

Set

HashSet

LinkedHashSet

TreeSet

TreeSet → class which stored unique elements and maintains natural order

↓

Balanced Binary Search Tree (BBST)

↓

Red - Black Tree

```
Set<String> s = new TreeSet<String>()
s.add("India")
s.add("New Zealand")
s.add("Australia")
print(s)
```

Australia
India
New Zealand

Map Interface

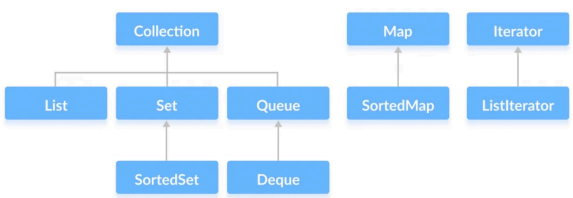
java.util

Class
HashMap
Key, value

Class
LinkedHashMap
Insertion order of key-value is maintained

TreeMap
keys are sorted based on natural order

Dinesh, 4
Kalu, 2
Apurva, 10




Apurva, 10
Dinesh, 4
Kalu, 2

Queue interface - used to hold elements in FIFO


Class
LinkedList

PriorityQueue

adding at end
deletion from start
O(1)

→ queue which orders elements based on priority

→ priority will be natural order or

Can add custom priority also

→ Duplicates are allowed

poll / top

```
pq.add("Geeks")
pq.add("For")
pq.add("Geeks")
```

print(pq) → For, Geeks, Geeks

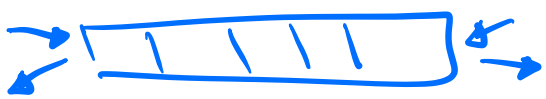
pq.poll() → for

pq.remove("Geeks")

Implemented by heap data structure

---

Deque



Flexible queue

LIFO, FIFO or anything

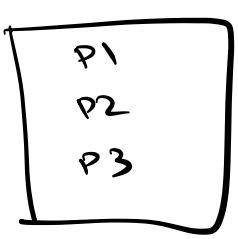
---

Comparable

→ Java Interface which allows us to define natural order of class

→ Natural order → sort elements of class, store class obj in DS like treeset

Treeset




→ int compareTo(T other)

-ve 0 +ve

```java
import java.util.*;

class Person implements Comparable<Person> {

    String name;
    int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
     }

    @Override
     public int compareTo(Person other) {
         return Integer.compare(this.age, other.age);
    }

}
```

Person p1 = new Person ("Amrit", 35)

Person p2 = new Person ("Zimmi", 25)

p3 ("Aburai", 30)

this → p2.compareTo(p1) ← other

p2.age, p1.age

25 35

Set<Person> s = new TreeSet<Person>()

s.add(p1)
s.add(p2)
s.add(p3)

print(s) → p2, p3, p1

# Comparator Interface

It is an interface which defines custom order for a class

Comparator

class → fn which compares 2 objects

```
class Person {

    String name;
    int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
}

class AgeComparator implements Comparator<Person> {

    @Override
    public int compare(Person person1, Person person2) {
        return Integer.compare(person1.age, person2.age);
    }
}

public class ComparatorExample {

    public static void main(String[] args) {

        List<Person> people = new ArrayList<>();

        people.add(new Person("Alice", 28));
        people.add(new Person("Bob", 22));
        people.add(new Person("Charlie", 25));

        // Sort the list using the AgeComparator
        Collections.sort(people, new AgeComparator());

        // Iterate the List of people and check if it is now sorted on the basis of age or not.
        for (Person person : people) {
            System.out.println(person.name + " - " + person.age);
        }
    }
}
```

Comparable
↓
compareTo
1 arg

Bob, 22
Charlie, 25
Alice, 28