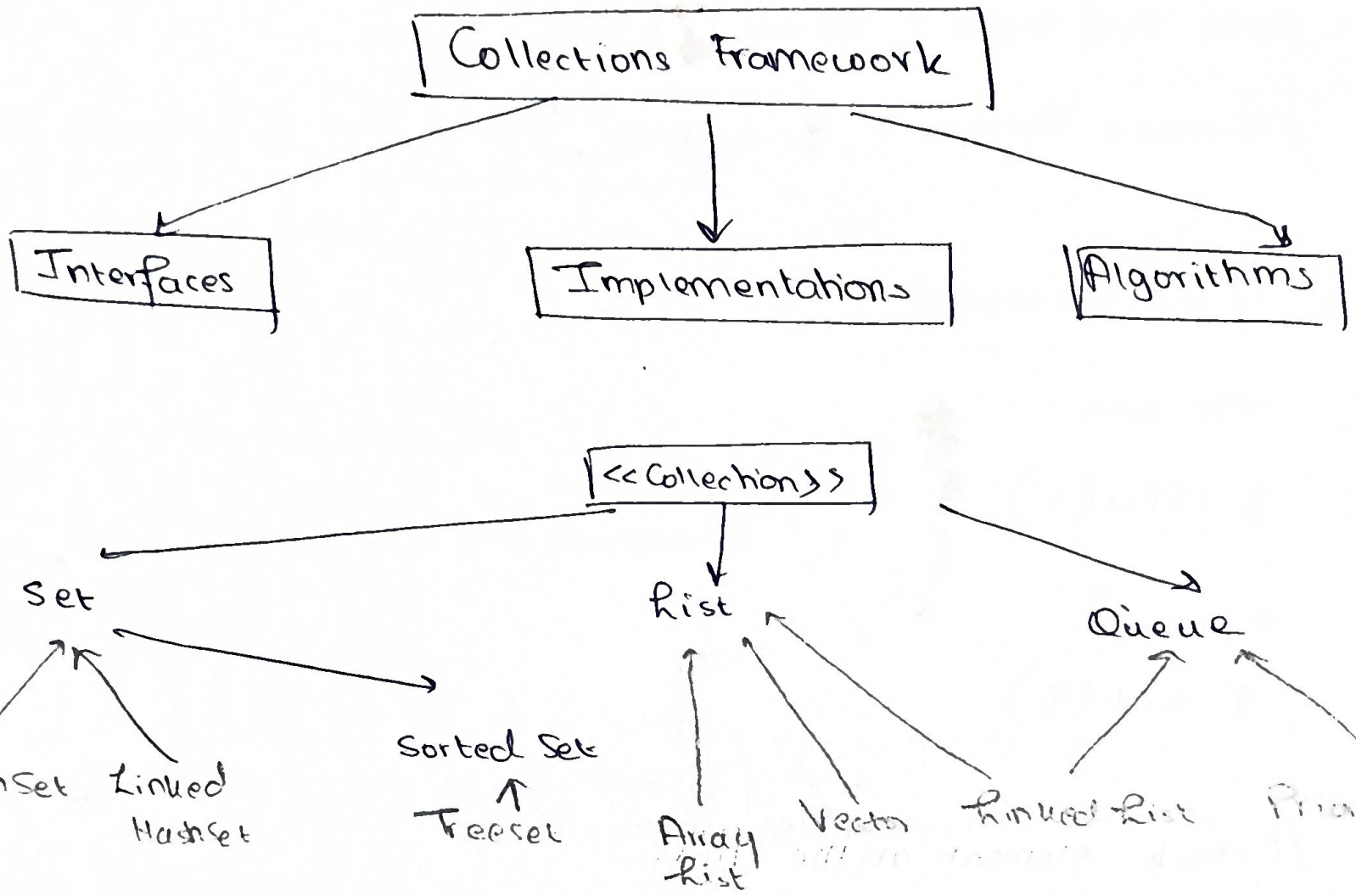
Collections

Collections: a framework that provides an architecture to store and manipulate groups of objects.



Iterator Interface: standardized way of accessing the elements within a collection, one by one.

Collection Interface Functions

1. add (item)
2. addAll (another list)
2. clear ()
2. contains (item)
2. containsAll (another list)
   isEmpty ()
   remove (obj)
   removeAll (another list)
   retainAll (another list)
   size()

List Interface Functions

get (index) — use during iteration

indexOf

lastIndexOf

remove (index)

set (ind, obj)

subList (start, end)

add (ind, element);

Queue Operations

PROGRAM 1

```
import java.util.*;
class queue{
    public static void main (String[] args){
        Queue<Integer> q = new LinkedList<Integer>();

        //add elements
```

(i) w/o error

```
q.offer(1)
```

(ii) w/ errors

```
q.add(2)

// check element at the front
```

(i) w/o errors

```
s.o.p(q.peek())
```

(ii) w/ errors

```
s.o.p(q.element())

// remove elements
```

(i) w/o errors

```
s.o.p (q.poll());
```

(ii) w/ errors

```
s.o.p(q.remove());
```

```
// Sorting an array
Collections.sort(arr);

// Reversing an array
Collections.reverse(arr);
```

* Iterating through the array list

(i) `for (String i : arr)`
`s.o.p(i)`

(ii)
```
for (int i=0; i<size; i++){
    s.o.p(arr.get(i));
}
```

* User defined classes in ArrayLists

```
class Student {
    int rno;
    String name;
    Student(int rno, String name) {
        this.rno = rno;
        this.name = name;
    }
    public String toString() {
        return "Rno: " + rno + " Name: " + name;
    }
}
```

// in main

`ArrayList<Student> s = new ArrayList<Student>();`

```
s.add( new Student (1,"Pooja"));
s.add( new Student (3, "Rita"));
```

* User defined sorting using ArrayList Objects

(1) Using Comparable

```
class Student implements Comparable<Student> {
    overrido public String toString()

    overrido compareTo

    public int compareTo ( Student s) {
        if (name.compareTo(s.name) >0) {
            return 1;
        }
        else if (name.compareTo(s.name) <0){
            return -1;
        }
        else {
            if (rno.compareTo(s.rno) >0) {
                return 1;
            }
            else {
                return -1;
            }
    }
```

use Collections.sort(obj)

(2) using Comparator

```
class Student implements Comparator<Student> {

    override toString

    overide compare

public int compare (Student s1, Student s2) {
        if (s1.name.compareTo(s2.name) > 0 {
                return 1;
        else if (s1.name.compareTo(s2.name) < 0 }
                return -1;
        }
        else {
                if (s1.rno > s2.rno) {
                        return 1;
                }
                else {
                        return -1;
        }               }
```

use : `Comparator<Student> c = new Student();`

`Collections.sort(obj,c);`

Linked List

operation	w/o error	w/ error
add elements to beginning	offerFirst( )	addFirst( )
add elements to end	offerLast( )	addLast( )
see first element	peekFirst( )	getFirst( )
see last element	peekLast( )	getLast( )
remove first	pollFirst( )	removeFirst( )
remove last	pollLast( )	removeLast( )

Iterator Interface

For traversal.

```
Iterator < String > itr = ll.iterator( );

while (itr.hasNext( )) {

    String element = itr.next( );
    s.o.p (element) ;
}
```

or

```
ListIterator < String > itr = l.listIterator( );

while ( itr.hasNext( )) {

    String element = itr.next( );
    itr.set (element + "***");
}
```

[A, B, C]

⇓

[A***, B***, C***]