The **Collection in Java** is a framework that provides an architecture to store and manipulate the group of objects.

Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.

Java Collection means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).

Iterator interface

Iterator interface provides the facility of iterating the elements in a forward direction only.

public boolean hasNext()	It returns true if the iterator has more elements otherwise it returns false.
public Object next()	It returns the element and moves the cursor pointer to the next element.

ArrayList

The ArrayList class implements the List interface. It uses a dynamic array to store the duplicate element of different data types. The ArrayList class maintains the insertion order and is non-synchronized

```
ArrayList<String> list=new ArrayList<String>();//Creating arraylist
list.add("Ravi");//Adding object in arraylist
list.add("Vijay");
```

LinkedList

LinkedList implements the Collection interface. It uses a doubly linked list internally to store the elements. It can store the duplicate elements.

Set Interface

Set Interface in Java is present in java.util package. It extends the Collection interface. It represents the unordered set of elements which doesn't allow us to store the duplicate items.

HashSet

HashSet class implements Set Interface. It represents the collection that uses a hash table for storage

```
HashSet<String> set=new HashSet<String>();
set.add("Ravi");
set.add("Vijay");
Iterator<String> itr=set.iterator();
```

```
while(itr.hasNext()){
System.out.println(itr.next());
}
```

TreeSet

Java TreeSet class implements the Set interface that uses a tree for storage. Like HashSet, TreeSet also contains unique elements. However, the access and retrieval time of TreeSet is quite fast

```
TreeSet<String> set=new TreeSet<String>();
set.add("Ravi");
set.add("Vijay");

Iterator<String> itr=set.iterator();
while(itr.hasNext()){
System.out.println(itr.next());
}
```

Java Map Interface

A map contains values on the basis of key, i.e. key and value pair. Each key and value pair is known as an entry. A Map contains unique keys.

A Map is useful if you have to search, update or delete elements on the basis of a key.

```
Map map=new HashMap();
//Adding elements to map
map.put(1,"Amit");
map.put(5,"Rahul");
map.put(2,"Jai");
    //Traversing Map
Set set=map.entrySet();//Converting to Set so that we can traverse
Iterator itr=set.iterator();
while(itr.hasNext()){
  //Converting to Map.Entry so that we can get key and value separately
  Map.Entry entry=(Map.Entry)itr.next();
  System.out.println(entry.getKey()+" "+entry.getValue());
}
```

Other way to iterate map

```
Map<String,String> gfg = new HashMap<String,String>();

        // enter name/url pair
        gfg.put("GFG", "geeksforgeeks.org");
        gfg.put("Practice", "practice.geeksforgeeks.org");
        gfg.put("Code", "code.geeksforgeeks.org");
        // looping over keys
        for (String name : gfg.keySet())
        {
            // search  for value
            String url = gfg.get(name);
            System.out.println("Key = " + name + ", Value = " + url);
        }
```

Comparable

A comparable object is capable of comparing itself with another object. The class itself must implements the `java.lang.Comparable` interface in order to be able to compare its instances.

Comparator

A comparator object is capable of comparing two different objects. The class is not comparing its instances, but some other class's instances. This comparator class must implement the `java.util.Comparator` interface.

```
class Movie implements Comparable<Movie>
{
    private double rating;
    private String name;
    private int year;

    // Used to sort movies by year
    public int compareTo(Movie m)
    {
        return this.year - m.year;
    }

    // Constructor
    public Movie(String nm, double rt, int yr)
    {
        this.name = nm;
        this.rating = rt;
        this.year = yr;
    }

    // Getter methods for accessing private data
    public double getRating() { return rating; }
    public String getName()   {  return name; }
    public int getYear()      {  return year;  }
}

// Class to compare Movies by ratings
class RatingCompare implements Comparator<Movie>
{
    public int compare(Movie m1, Movie m2)
    {
```

```
        if (m1.getRating() < m2.getRating()) return -1;
        if (m1.getRating() > m2.getRating()) return 1;
        else return 0;
    }
}

// Class to compare Movies by name
class NameCompare implements Comparator<Movie>
{
    public int compare(Movie m1, Movie m2)
    {
        return m1.getName().compareTo(m2.getName());
    }
}

// Driver class
class Main
{
    public static void main(String[] args)
    {
        ArrayList<Movie> list = new ArrayList<Movie>();
        list.add(new Movie("Force Awakens", 8.3, 2015));
        list.add(new Movie("Star Wars", 8.7, 1977));
        list.add(new Movie("Empire Strikes Back", 8.8, 1980));
        list.add(new Movie("Return of the Jedi", 8.4, 1983));

        // Sort by rating : (1) Create an object of ratingCompare
        //                  (2) Call Collections.sort
        //                  (3) Print Sorted list
        System.out.println("Sorted by rating");
        RatingCompare ratingCompare = new RatingCompare();
        Collections.sort(list, ratingCompare);
        for (Movie movie: list)
            System.out.println(movie.getRating() + " " +
                               movie.getName() + " " +
                               movie.getYear());


        // Call overloaded sort method with RatingCompare
        // (Same three steps as above)
        System.out.println("\nSorted by name");
        NameCompare nameCompare = new NameCompare();
        Collections.sort(list, nameCompare);
        for (Movie movie: list)
            System.out.println(movie.getName() + " " +
                               movie.getRating() + " " +
                               movie.getYear());

        // Uses Comparable to sort by year
        System.out.println("\nSorted by year");
        Collections.sort(list);
        for (Movie movie: list)
            System.out.println(movie.getYear() + " " +
                               movie.getRating() + " " +
                               movie.getName()+" ");
    }
}
```