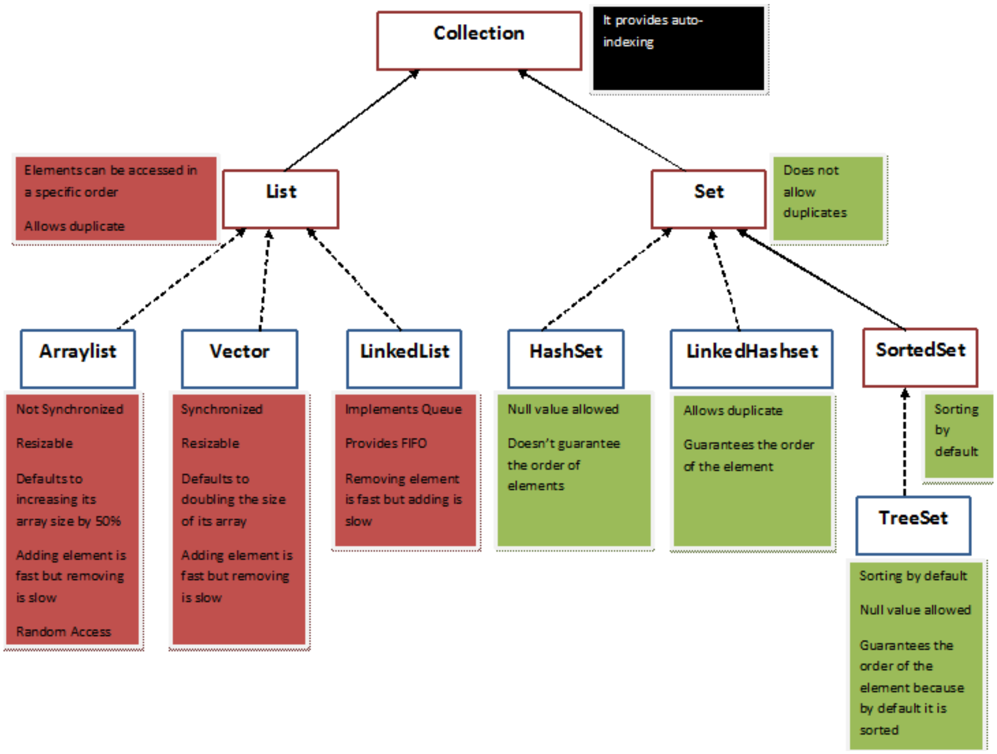
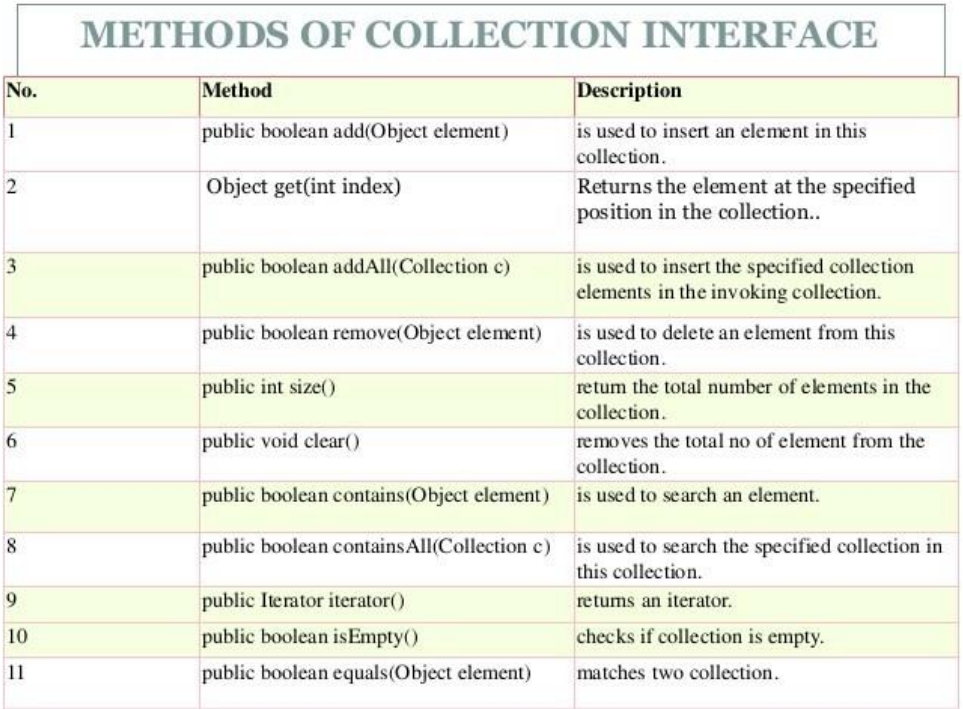
COLLECTIONS:

- A collection is a single object managing a group of objects known as its elements.
- Collection – A group of objects called elements.
- Implementation determine and whether duplicates are permitted.

Collections API:


Collection
It provides auto-indexing
List
Elements can be accessed in a specific order
Allows duplicate
Set
Does not allow duplicates
Arraylist
Not Synchronized
Resizable
Defaults to increasing its array size by 50%
Adding element is fast but removing is slow
Random Access
Vector
Synchronized
Resizable
Defaults to doubling the size of its array
Adding element is fast but removing is slow
LinkedList
Implements Queue
Provides FIFO
Removing element is fast but adding is slow
HashSet
Null value allowed
Doesn't guarantee the order of elements
LinkedHashset
Allows duplicate
Guarantees the order of the element
SortedSet
Sorting by default
TreeSet
Sorting by default
Null value allowed
Guarantees the order of the element because by default it is sorted

Collections-Methods:



METHODS OF COLLECTION INTERFACE

No.	Method	Description
1	public boolean add(Object element)	is used to insert an element in this collection.
2	Object get(int index)	Returns the element at the specified position in the collection..
3	public boolean addAll(Collection c)	is used to insert the specified collection elements in the invoking collection.
4	public boolean remove(Object element)	is used to delete an element from this collection.
5	public int size()	return the total number of elements in the collection.
6	public void clear()	removes the total no of element from the collection.
7	public boolean contains(Object element)	is used to search an element.
8	public boolean containsAll(Collection c)	is used to search the specified collection in this collection.
9	public Iterator iterator()	returns an iterator.
10	public boolean isEmpty()	checks if collection is empty.
11	public boolean equals(Object element)	matches two collection.

Iterator Interface:

- Iterator interface provide the facility of iterating the elements in a forward direction only.
- The iterable interface is the root interface for the collection classes.The collection interface extends the iterable interface and therefore all the subclasses of collection interface also implement the iterable interface.

Methods Of Iterator Interface :

- public boolean hasNext() - it returns true if the iterator has more elements
- public object next() - it returns the element and moves the cursor pointer to the next element
- Public void remove()- it removes the last elements returned by the iterator.

Collection Interface:

- The collection interface is the interface which is implemented by all the classes in the collection framework.It declares the methods that methods that every collection will have.
- Some of the methods of collection interface are boolean add(Object.obj), Boolean addAll(collection c), void clear(),etc which are implemented by all the subclasses of collection interface.

List Interface:

- List interface is the child interface of collection interface.It inhibits a list type data structure in which e can store the ordered collection of objects.
- It can have duplicates.
- List interface is implemented by the classes ArrayList,LinkedList,vectorandStack.

Syntax:

```
List<data-type>list1 = new ArrayList();

List<data-type>list1 = new LinkedList();

        List<data-type>list1 = new vector();

        List<data-type>list1 = new stack();
```

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-synchronised,

- The elements stored in the arraylist can be randomly accessed

EXAMPLE:

```
import java.util.*;

class JavaCollection1{

        public static void main(String args[]){

                ArrayList<String> list=new ArrayList<String>();//Creating arraylist

                list.add("max");//Adding object in arraylist

                list.add("vijay");

                list.add("max");

                list.add("Ajith");

                //Traversing list through Iterator

                Iterator itr=list.iterator();

                while(itr.hasNext()){

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

                }

        }

}
```

OUTPUT:

```
max

vijay

max

Ajith
```

LinkedList:

- LinkedList implements the Collection interface.
- It uses a doubly linked list internally to store the elements. It can store the duplicate elements.
- It maintains the insertion order and is not synchronized. In LinkedList, the manipulation is fast because no shifting is required.

EXAMPLE:

```
import java.util.*;

        public class JavaCollection2{

            public static void main(String args[]){

                LinkedList<String> al=new LinkedList<String>();

                al.add("queen");

                al.add("king");

                al.add("queen");

                al.add("bishop");

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

                while(itr.hasNext()){

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

                }

             }

        }
```

Output:

```
queen
King
```

queen

bishop

SET:

- An unordered collection;no duplicates are permitted.
- 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 duplicate items.we can store atmost one null value in set.
- Set is implemented byHashSet,LinkedHashSet and TreeSet.

INSTANTIATING SET:

```
Set<data-type> s1 = new HashSet<data-type>();
```

HashSet :

- HashSet class implements Set interface. It represents the collection that uses a hash table for storage.
- Hashing is used to store the elements in the Hashset. It contains unique element.

EXAMPLE:

```
import java.util.*;

        public class TestJavaCollection7{

                public static void main(String args[]){

                //Creating HashSet and adding elements

                        HashSet<String> set=new HashSet<String>();
                        set.add("Ravi");

                        set.add("Vijay");

                        set.add("Ravi");

                        set.add("Ajay");

                        //Traversing elements

                        Iterator<String> itr=set.iterator();

                        while(itr.hasNext()){

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

                        }

                }

        }
```

Output:

```
Vijay

Ravi

Ravi
```

COMPARABLE AND COMPARATOR INTERFACE:

COMPARABLE:

- Comparable is an interface defining a stragety of comparing an object with other objects of the same type. This is called the class's "natural ordering"

COMPARATOR:

- The comparator operator interface defines a compare(arg1.arg2)method with two arguments that represent compared objects and works similarly to the comparable.compareTo()method.

    - CREATING COMPARATORS:
        - To create a comparator we have to implement the comparator interface.

COMPARABLE vs COMPARATOR:

- **The comparable interface is good choice when used for defining the default ordering.**
    - **Why use a comparator if we already have comparable?**
        - Sometimes,we can't modify the source code of the class whose objects we want to sort,thus making the use of comparable impossible
        - Using comparators allows us to avoid adding additional code to our domain classes.
        - We can define multiple different comparison strageties which isn't possible when using comparable

EXAMPLE:

```
// A Java program to demonstrate use of Comparable
import java.io.*;
import java.util.*;
// A class 'Movie' that implements Comparable
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; }
}

// 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));

Collections.sort(list);
System.out.println("Movies after sorting : ");
for (Movie movie: list)
{
System.out.println(movie.getName() + " " +
movie.getRating() + " " +
movie.getYear());
}
}
}
```