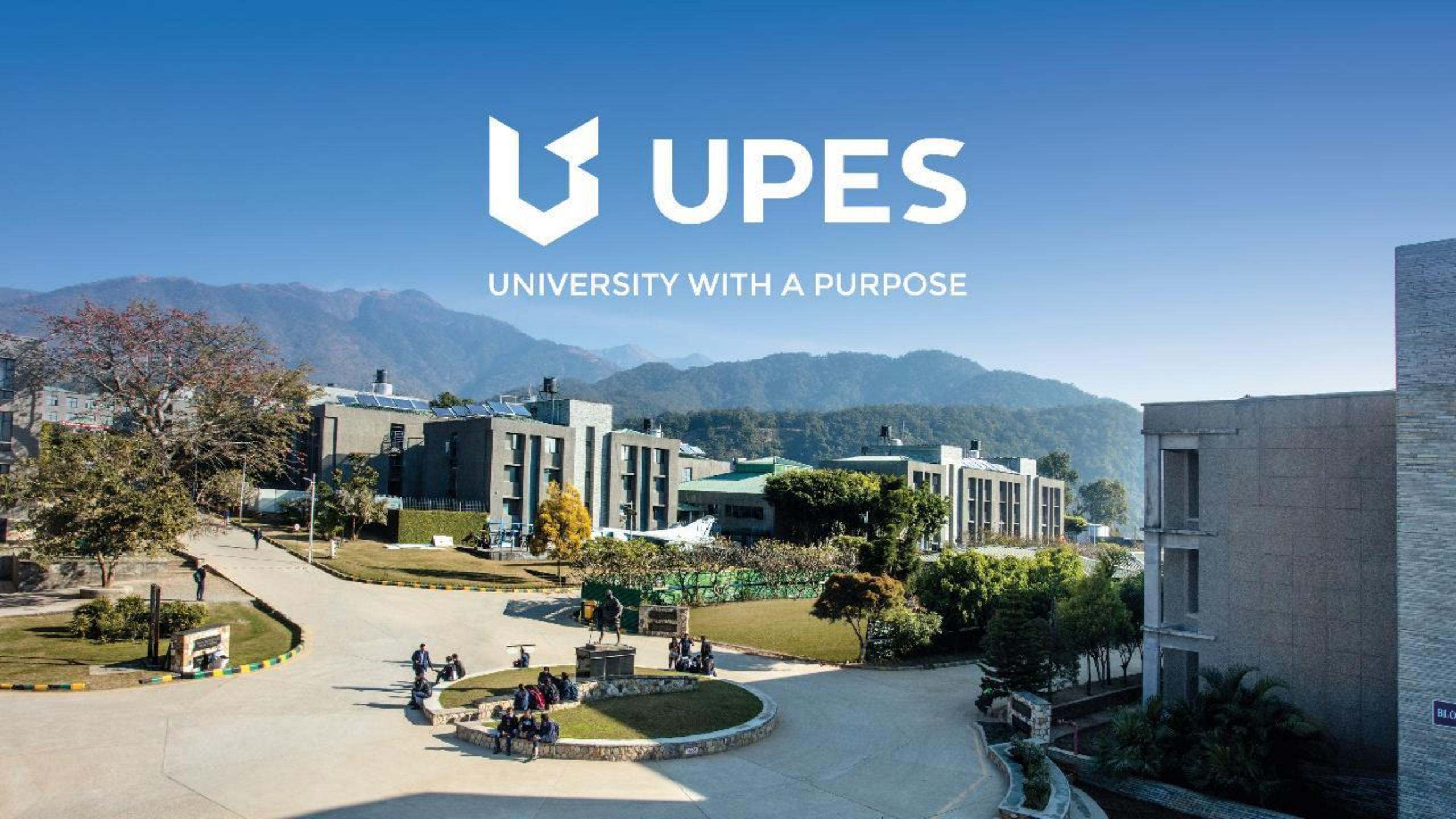

UPES
UNIVERSITY WITH A PURPOSE

Collection Framework

Collection Framework

- A **collection** — sometimes called a container is simply an object that groups multiple elements into a single unit.
- Collection framework provides a well designed set of Interfaces and classes for storing and manipulating groups of data as a single unit, a Collection.

Collection Framework

What is a Collections Framework?

A collections framework is a unified architecture for representing and manipulating collections. All collections frameworks contain the following:

1. **Interfaces:** Interfaces allow collections to be manipulated independently of the details of their representation.
2. **Implementations:** These are the concrete implementations of the collection interfaces. In essence, they are reusable data structures.
3. **Algorithms:** These are the methods that perform useful computations, such as searching and sorting, on objects that implement collection interfaces.

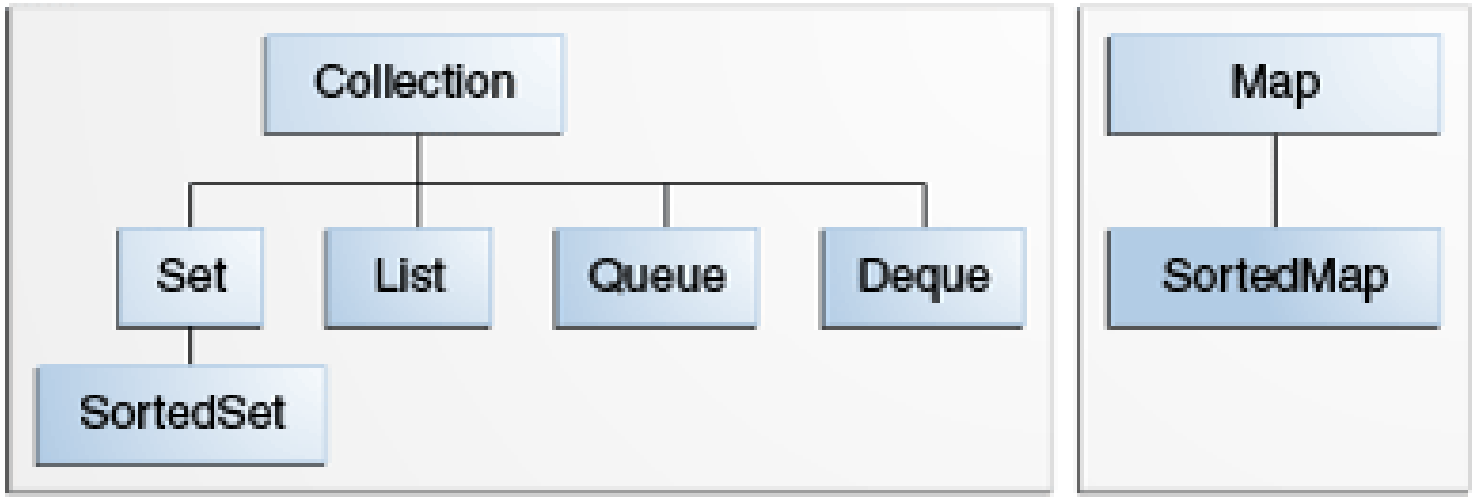
Collection Framework

Benefits of the Java Collection Framework:

1. **Reduces programming effort:** By providing useful data structures and algorithms, the Collection Framework frees you to concentrate on the important parts of your program.
2. **Increases program speed and quality:** The various implementations of each interface are interchangeable, so programs can be easily tuned by switching collection implementations.
3. **Allows interoperability among unrelated APIs:** The collection interfaces are the means by which APIs pass collections back and forth.
4. **Fosters software reuse:** New data structures that conform to the standard collection interfaces are by nature reusable. The same goes for new algorithms that operate on objects that implement these interfaces.

Interfaces

The core collection interfaces encapsulate different types of collections, which allow collections to be manipulated independently of the details of their representation.



Interfaces

Collection:

•the root of the collection hierarchy. A collection represents a group of objects known as its *elements*.

•Some types of collections allow duplicate elements, and others do not.

•Some are ordered and others are unordered.

•The Java platform doesn't provide any direct implementations of this interface but provides implementations of more specific subinterfaces, such as Set and List.

Set:

• A collection that cannot contain duplicate elements.

•This interface models the mathematical set abstraction and is used to represent sets, such as the cards comprising a poker hand, the courses making up a student's schedule, or the processes running on a machine.

Interfaces

List:

• An ordered collection (sometimes called a *sequence*).

•Lists can contain duplicate elements.

•The user of a List generally has precise control over where in the list each element is inserted and can access elements by their integer index (position).

Queue:

•A collection used to hold multiple elements prior to processing.

•Besides basic Collection operations, a Queue provides additional insertion, extraction, and inspection operations.

•Queues typically, but do not necessarily, order elements in a FIFO manner.

•Among the exceptions are priority queues, which order elements according to a supplied comparator or the elements' natural ordering.

•Whatever the ordering used, the head of the queue is the element that would be removed by a call to remove.

Interfaces

Deque:

- A collection used to hold multiple elements prior to processing.
- Besides basic Collection operations, a Deque provides additional insertion, extraction, and inspection operations.
- Deques can be used both as FIFO and LIFO.
- In a deque all new elements can be inserted, retrieved and removed at both ends.

Map:

- An object that maps keys to values.
- A Map cannot contain duplicate keys; each key can map to at most one value.

SortedSet:

- A Set that maintains its elements in ascending order.
- Several additional operations are provided to take advantage of the ordering.

SortedMap:

- A Map that maintains its mappings in ascending key order. This is the Map analog of SortedSet. Sorted maps are used for naturally ordered collections of key/value pairs, such as dictionaries and telephone directories.

Collection Interface

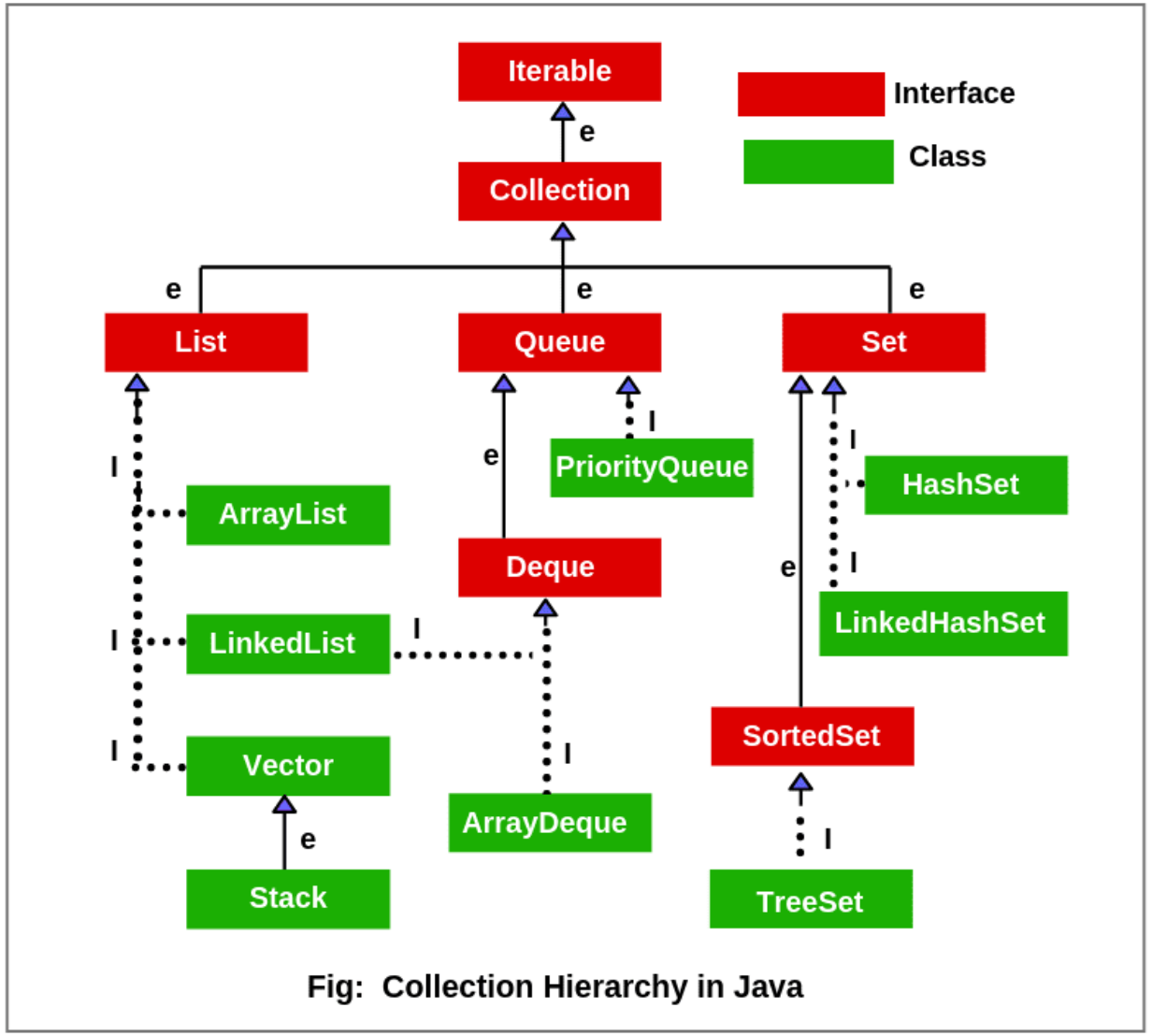


Fig: Collection Hierarchy in Java

Collection Interface

- The basic interface of the collection framework is the Collection interface which is the root interface of all collections in the API (Application programming interface) and placed at the top of the collection hierarchy.

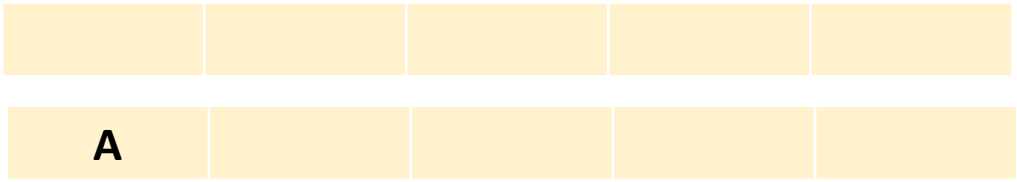
- Collection interface extends the Iterable interface. The Iterable interface has only one method called `iterator()`. The function of the iterator method is to return the iterator object. Using this iterator object, we can iterate over the elements of the collection.

- List, Queue, and Set have three components which extend the collection interface. A map is not inherited by the collection interface.

Collection Interface

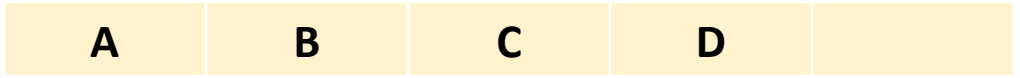
boolean add (Object o): Add an object to the collection

```
ArrayList l1 = new ArrayList (5);
l1.add (A) ;
```



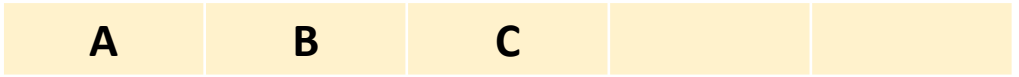
boolean addAll (Collection c): Add all the objects of specified collection

```
l2.addAll (l1);
Given that: l1= B,C,D
```



boolean remove (Object o): Removes a single instance of element from this collection

```
l1.remove (D);
```



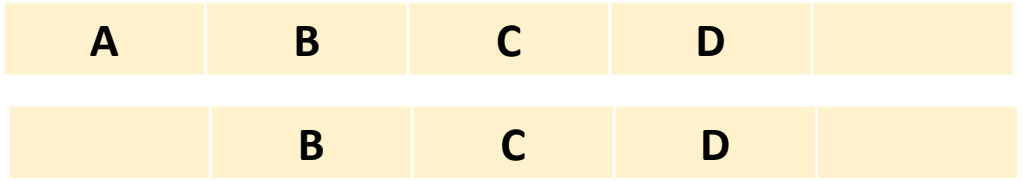
boolean removeAll (Collection c): Removes all the collections elements that are also contained in the specified collection


```
l1.removeAll (l2);
```

Collection Interface

boolean retainAll (Collection c): Retains only the elements in this collection that are contained in the specified collection.

```
ArrayList l1 = new ArrayList (5);
l1.retainAll (l2) ;
```



int size ( ): Returns the number of elements in this collection

boolean contains (Object o): Returns true if this collection contains the specified object.

```
l1.contains (A): false
```

void clear ( ): Removes all the elements from this collection.

```
l1.clear ();
```


boolean isEmpty ( ): Returns true if this collection contains no elements.

```
l1.isEmpty (): true
```

List Interface

- This interface represents a collection of elements whose elements are arranged in sequentially ordered.
- List maintains an order of elements means the order is retained in which we add elements, and the same sequence we will get while retrieving elements.
- We can insert elements into the list at any location. The list allows storing duplicate elements in Java.
- `ArrayList, vector, and LinkedList` are three concrete subclasses which implement list interface.

Methods of List Interface

void add (int index, Object o): inserts the specified element at the specified position in this list. (elements may shift)

```
ArrayList l1 = new ArrayList (5);
l1.add (A);  //Method of Collection interface
l1.add (B);  //Method of Collection interface
l1.add (0,C);  //Method of List interface
l1.add (1,D);  //Method of List interface
```

A				
A	B			
C	A	B		
C	D	A	B	

object set (int index, Object o): Replaces the element at the specified position in this list with the specified element.

```
l1.set (1,F);  //Method of List interface
```

C	F	A	B	

object get (int index): Returns the element at the specified position in this list

```
l1.get (2): A
```

Methods of List Interface

- int indexOf (Object o): Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.
- int lastIndexOf (Object o): Returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element.
- remove (int index): Removes the element at the specified position in the list.
- subList (int fromIndex, int toIndex): Returns a view of the portion of this list between the specified fromIndex, inclusive, and toIndex, exclusive.
- There is no concrete class which implements Collection interface directly.

ArrayList

- ArrayList is defined using dynamic arrays. Therefore it is resizable.
- Duplicates are allowed.
- Order of elements are preserved.
- Null insertion is possible.
- Heterogeneous objects are allowed.

ArrayList

```
ArrayList a = new ArrayList ();
```

- Creates an empty ArrayList objects with default initial capacity “10” if ArrayList reaches its maximum capacity then a new ArrayList object will be created with

```
New capacity = ( current capacity * 3/2)+1
```

```
ArrayList a = new ArrayList (int initialcapacity);
```

Exp:

```
ArrayList l1 = new ArrayList (5);
```



```
ArrayList l2 = new ArrayList ();
```


New capacity = 10*3/2+1 = 16

ArrayList

```
ArrayList a = new ArrayList (Collection c);
```

- Creates an equivalent ArrayList object for the given Collection

Exp:

```
ArrayList l3 = new ArrayList (l2);
```

Examples

How do you find the number of elements present in an ArrayList?

```
import java.util.ArrayList;
public class ArrayList1
{
    public static void main(String[] args)
    {
        ArrayList list = new ArrayList();

        list.add(1.1);

        list.add(2.2);

        list.add(3.3);

        list.add(4.4);

        list.add(5.5);

        System.out.println(list);

        System.out.println("Size Of ArrayList = "+list.size());
    }
}
```

Examples

How do you find out whether the given ArrayList is empty or not?

```
public class MainClass
{
    public static void main(String[] args)
    {
        ArrayList list = new ArrayList();

        System.out.println(list.isEmpty());     //Output : true
    }
}
```

Examples

How do you check whether the given element is present in an ArrayList or not?

```
public class MainClass
{
    public static void main(String[] args)
    {
        ArrayList<Double> list = new ArrayList<Double>();

        list.add(1.1);

        list.add(11.11);

        list.add(111.111);

        list.add(1111.1111);

        //Checking whether list conatins '111.1111'

        System.out.println(list.contains(111.1111));    //Output : false
    }
}
```

Examples

How do you get the position of a particular element in an ArrayList?

```
public class MainClass
{
    public static void main(String[] args)
    {
        ArrayList<String> list = new ArrayList<String>();
         list.add("JAVA");
         list.add("J2EE");
         list.add("JSP");
         list.add("JAVA");
         list.add("SERVLETS");
         list.add("JAVA");
         list.add("STRUTS");
         System.out.println(list);     //Output : [JAVA, J2EE, JSP, JAVA, SERVLETS, JAVA, STRUTS]
         //Getting the index of first occurrence of "JAVA"
         System.out.println(list.indexOf("JAVA"));     //Output : 0
         //Getting the index of last occurrence of "JAVA"
         System.out.println(list.lastIndexOf("JAVA"));    //Output : 5
    }
}
```

References

http://java.sun.com/docs/books/tutorial/collections/interfaces/collection.html.

THANK YOU