Collection

- The collection framework is nothing but the API which contains a number of interfaces and classes. The important interfaces in this API are :

1. Collection
2. List
3. Queue
4. Set

Meaning of word “collection”

- In java the word ‘collection’ is overloaded as follows:
- collection : (with small ‘c’) it is basically refer to any data structure, which stores the object and iterate over.
- Collection: (with capital ‘C’) it is basically refer to java.util.Collection interface.
- Collections: (with capital ‘C’ and s at the end) it is basically refers to the java.util.Collections class which contains static utility methods.

Classes of collection API

- The important Classes of collection API are:

Maps	Sets	Lists	Queues	Utilities
HashMap	HashSet	ArrayList	PriorityQueue	Collections
Hashtable	LinkedHashSet	Vector		Arrays
TreeMap	TreeSet	LinkedList		
LinkedHashMap				

ArrayList

- ArrayList is nothing but the class which is used to store the object dynamically.
- In this we don’t need to mention the length of the ArrayList, instead of this, it’s length increases as new objects are added and shrinks when objects are removed from it.
- ArrayList never stores the ‘primitive values’.
- All values stored in the ArrayList are object, if we try to add primitive value, it will be implicitly changes to the objects by “autoboxing”

Wrapping, Unwrapping and Auto boxing of primitives

- Autoboxing:

Autoboxing is basically a technique in which compiler automatically wrap and unwrap the primitives in order to store them in “ARRAYLIST”.

- Wrapping and Unwrapping a integer variable:
- **public class Test {**

```
public static void main(String[] args)
{
int i = 45;
Integer wrapper = new Integer(i);
System.out.println(wrapper);
int unwrapped = wrapper.intValue();
System.out.println(unwrapped);
}}
```

Wrapper classes

- Wrapper classes are those classes, which are used to ‘wrap’ primitive variables in such a way, that they behave like an object. These classes are:

1. Integer – for wrapping int value.
2. Character – for wrapping char value.
3. Boolean – for wrapping boolean value.
4. Long – for wrapping long value.
5. Float – for wrapping float value.
6. Double – for wrapping double value

Creating a ArrayList

- An ArrayList is created just like an array, without mentioning the length of the array :

```
ArrayList myList = new ArrayList();
```

Methods of ArrayList

- The important methods of ArrayList are:

1. myList.add() // for adding a object to the arrayList.
2. myList.remove() // for removing the object from the arrayList
3. myList.contains() // returns true if the object is present in the list otherwise false
4. myList.get() // retrieve the object from the index given in the argument

Method of ArrayList continues

5. myList.isEmpty() // returns true if the list is empty otherwise false.
6. myList.size() // returns the actual size of the arrayList.
7. myList.indexOf() // it will show the index of the object passed in the arguments.

Parameterization of ArrayList

- After Java 5, parameterization of ArrayList is possible, that is we are allowed to restrict the ArrayList to store only ‘same’ type of data.

ArrayList<Integer> myList = new ArrayList<Integer>() // it will store only the integer values

List

- Lists are usually used to keep things/objects in some kind of order.
- Lists allow you to manually override the ordering of elements by adding or removing elements via the element's index.
- Before Java 5, and the enhanced for loop, the most common way to examine a List "element by element" was by the use of an Iterator.

Iterator

- An Iterator is an object that's associated with a specific collection. It lets you loop through the collection step by step. The two Iterator methods you need to understand for are:

■ **boolean hasNext() Returns true if there is at least one more element in** the collection being traversed.

Invoking hasNext() does NOT move you to the next element of the collection.

■ **Object next() This method returns the next object in the collection,**

AND moves you forward to the element after the element just returned.

Iterater example

- **public static void main(String[] args)**
- {

- ArrayList<Integer> a = **new ArrayList<Integer>();**

- a.add(23);
- a.add(45);
- a.add(56);
- a.add(67);
- a.add(89);

- Iterator i = a.iterator();

- **while(i.hasNext())**
- {
- Object element = i.next();
- System.*out.println(element);*
- }

- }