List:
=====

--It is the child interface of Collection interface and declares the behavour of a collection to
preserve the sequence of an element.

--elements can inserted and accessed by thier position using zero based index.

--here insertion order will be preserved and duplicate elements are allowed.

--in addition to the Collection inteface methods ,List interface defines some of its own methods also.

public Object get(int index);

public Object set(int index, Object newObject); it will return the overriden object

public Object remove(int index); // will return the removed object.

public int indexOf(Object obj);

etc..

*****Note: Collection f/w does not support primitives, it only supports objects.**

add(Object obj);

ArrayList:
========

--it is the implementation of the List interface.

--it is basically a dynamic array (it dynamically increase and decrease in size).

--ArrayList class is the best choice if our frequent operation is retrieval based on index.

--duplicate are allowed.

--null insertion is possible (multiple time also)

example:

```
ArrayList al=new ArrayList();

System.out.println(al);//[]
```

Note: all the collection classes has overriden the toString() method internally. so they will print the elements inside that collection in [] square bracket.

--all the collection classes are like a container or bag which holds multiple objects.

--in the above statement we have created an empty AL object with the default initial capacity 10.

--once AL reaches to its max capacity then a new AL object will be created in the memory automatically with the new capacity using following formula:

newCapacity = (currentCapacity * 3/2) +1;

```
ArrayList al=new ArrayList(1000); // AL created with the initial capacity 1000;
```

Autoboxing and Autounboxing:
==========

this concept comes in java 1.5 version

boxing: converting primitives into the object(box) it is known as boxing and reverse is called unboxing.

--8 primitive datatype.

--for each primitive data types we have corresponding wrapper classes are there.

--int --- java.lang.Integer
--byte --- java.lang.Byte
--char ---> java.lang.Charecter
--boolean --> java.lang.Boolean

--
--

--before java 1.5 inorder to add the primitives in the collection we need to wrap that primitives to their corresponding wrapper class object.

```
int i=10;

Integer i1= Integer.valueOf(i); //boxing

int x=  i1.intValue(); //unboxing
```

from java 1.5 onwords we have a concept called autoboxing and autounboxing

```
int i =10;

Integer i1 = i;  //autoboxing

int x=i1; // autounboxing
```

example:

```
ArrayList al=new ArrayList();



al.add("delhi");
al.add("mumbai");
al.add("chennai");
al.add("kolkata");
al.add(new A());
al.add(new Student(10, "Amit", 780));
al.add(null);
al.add(null);
al.add("delhi");
al.add(10);  // Integer
al.add(true); //Boolean
al.add(10.55); //Double
```

```
                System.out.println(al);
                System.out.println(al.size());



//              Object obj= al.get(1);
//              String city= (String)obj;

                int x= (Integer)al.get(9);


                System.out.println(x);
```

---in the above application our ArrayList object is not a type safe Collection object.

--if our collection is not type safe collection then we can add any type of object at any position inside our collection.

--here while getting the elements from the type unsafe collection every time we need to downcast the element, which is not feasiable. there might be a change of ClassCastException.

--so in realtime, our collection should be type safe collection.
--type safe collection means making our collection homogenious.

benifit of type safe collection:
----

1. if we try to add any other type of element then compiler will stop at compile time.

2.we will get rid of downcasting problem.

ArrayList<Object> al=new ArrayList<>();

--taking the type of Object is simmillar to creating type-unsafe collection.

example

```
package com.masai;

import java.util.ArrayList;

public class Demo {

	public static void main(String[] args) {

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



		al.add("delhi");
		al.add("mumbai");
		al.add("chennai");
		al.add("kolkata");
		al.add("delhi");
		al.add(10);
		al.add(null);



		String s= al.get(2);

		System.out.println(s.toUpperCase());


	}

}
```

--the above type safe collection concept is called Genrics concept.

--Generics concept also introduced in java 1.5 version

```
//ArrayList class sudo code before generics

class ArrayList implements List{

public boolean add(Object obj){
//adding the object obj to the AL.
}
```

```
public Object get(int index){
it will return the obj to the specified index
}

//remaining methods.


}


//ArrayList class sudo code after generics

class ArrayList<T> implements List{

public boolean add(T t){
//adding the object t to the AL.
}

public T get(int index){
it will return the obj to the specified index
}

//remaining methods.
}
```

example:

```
ArrayList<String> al=new ArrayList<>(); // it is the List of String object


ArrayList<Integer> al=new ArrayList<>(); // it is the List of Integer object

ArrayList<Student> al=new ArrayList<>(); // it is the List of Student object
```

Demo.java:

```
package com.masai;

import java.util.ArrayList;
import java.util.Scanner;
```

```
public class Demo {

	public static void main(String[] args) {


		Scanner sc= new Scanner(System.in);

		ArrayList<Student> students = new ArrayList<>();

		int count = 1;

		while(true) {


			System.out.println("Enter details of Student "+(count++));

			System.out.println("Enter Roll");
			int roll= sc.nextInt();

			System.out.println("Enter Name");
			String name= sc.next();

			System.out.println("Enter Marks");
			int marks= sc.nextInt();

			Student student = new Student(roll, name, marks);

			students.add(student);

			System.out.println("Student object added sucessfully...");

			System.out.println("Want more(y/n) ?");
			String choice=  sc.next();

			if(choice.equalsIgnoreCase("n"))
				break;

		}


		for(Student student: students) {
```

```
                System.out.println("Roll is :"+student.getRoll());
                System.out.println("Name is :"+student.getName());
                System.out.println("Marks is :"+student.getMarks());

                System.out.println("================================");
            }

        }

}
```