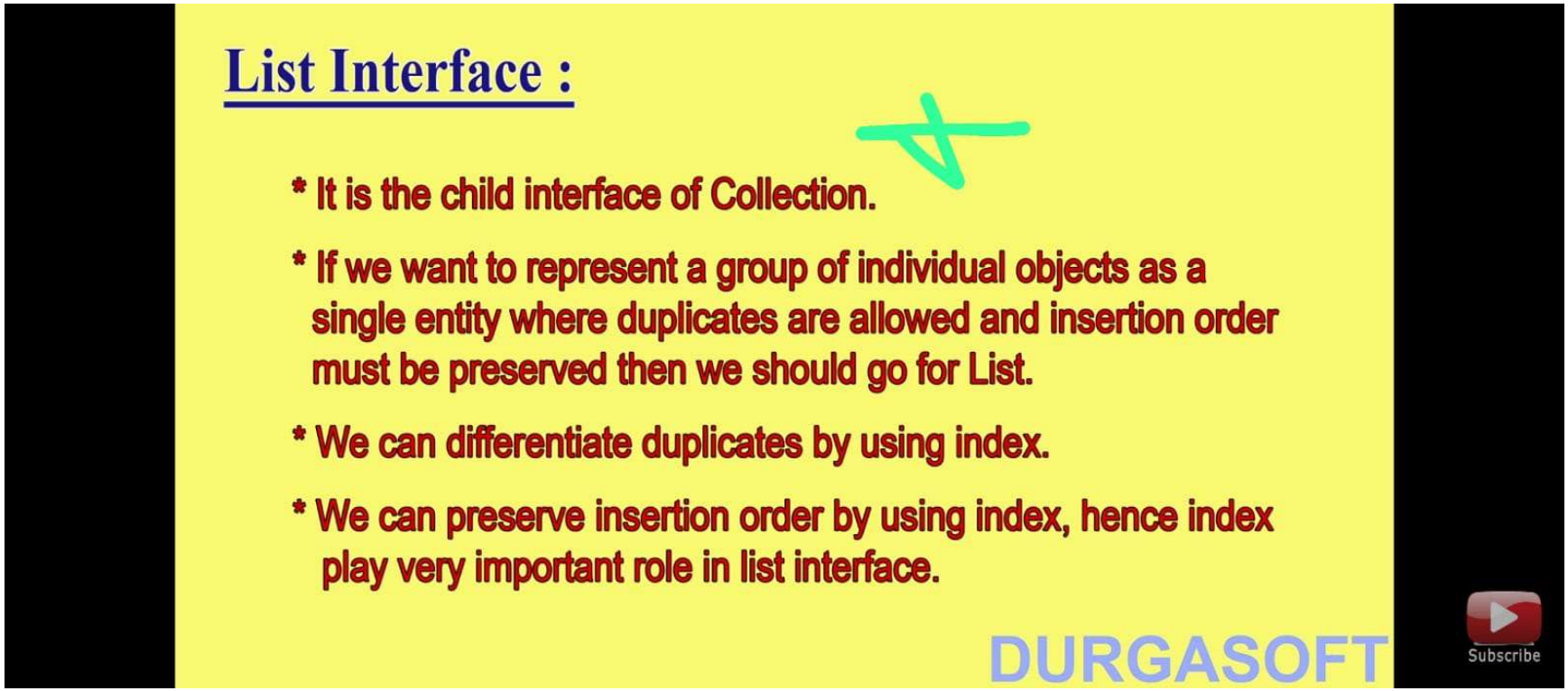
CFList_ArrayList3

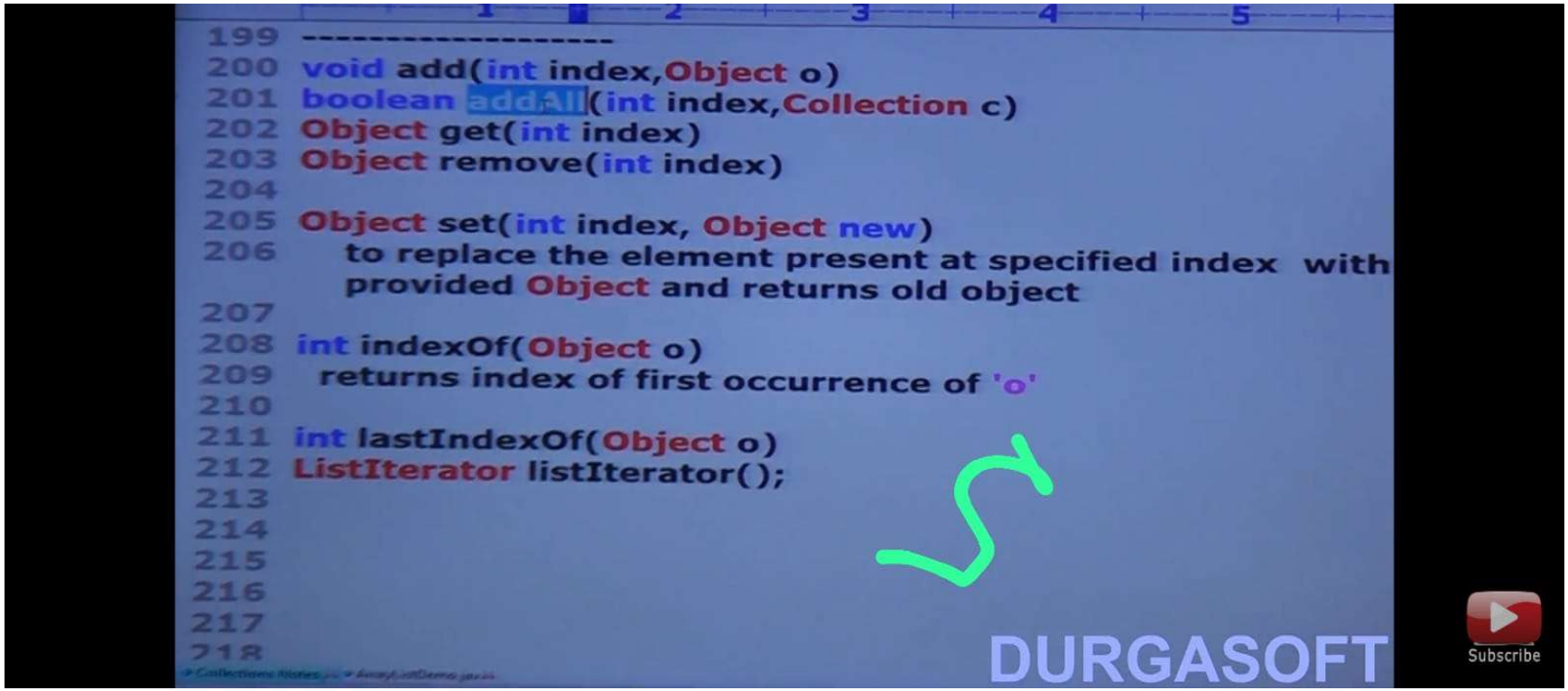
```
175  boolean add(Object o)
176  boolean addAll(Collection c)
177  boolean remove(Object o)
178  boolean removeAll(Collection c)
179  boolean retainAll(Collection c)
180        To remove all objects except those
181           present in c
182  void clear()
183  boolean contains(Object o)
184  boolean containsAll(Collection c)
185  boolean isEmpty()
186  int size();
187  Object[] toArray();
188  Iterator iterator()
189
190
191
192
193
```

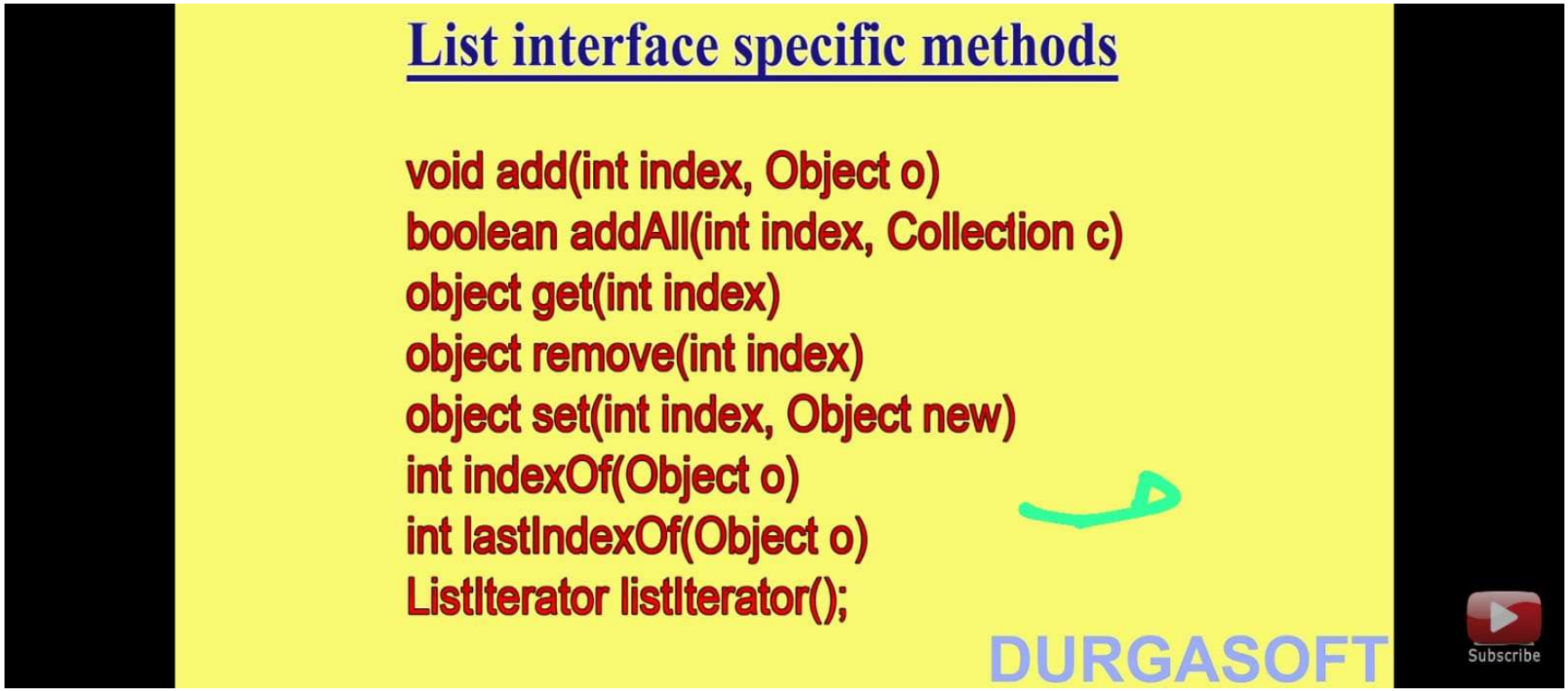
Important methods of Collection Interface

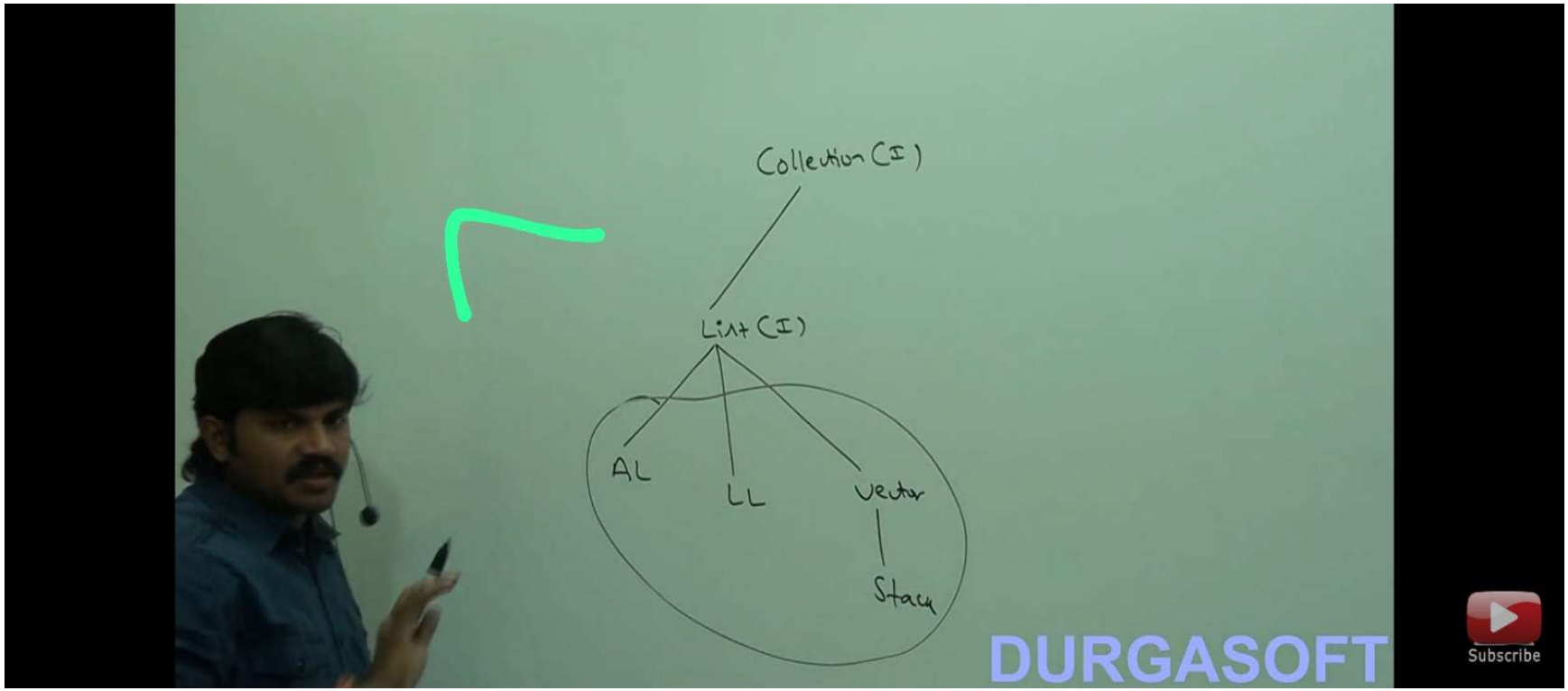
boolean add(Object o)
boolean addAll(Collection c)
boolean remove(Object o)
boolean removeAll(Collection c)
boolean retainAll(Collection c)
void clear()

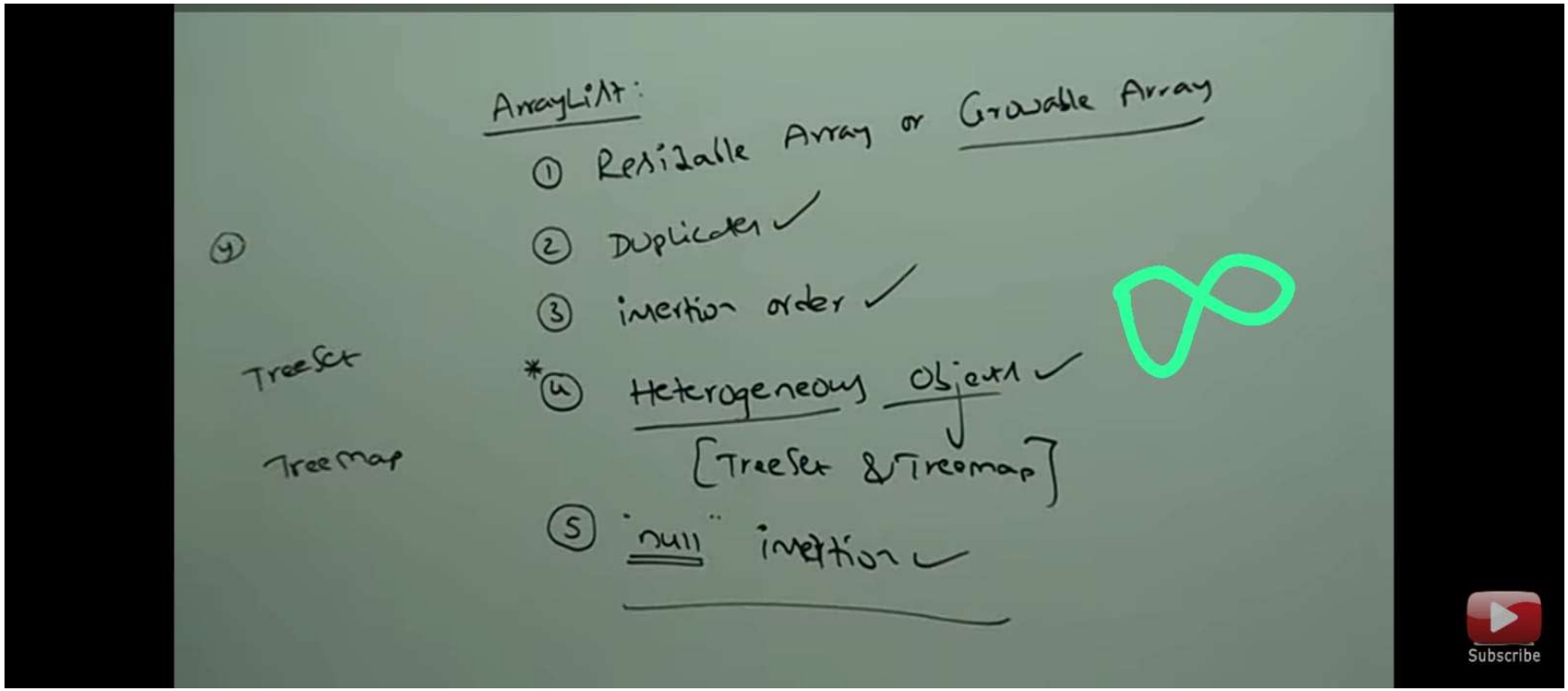
boolean contains(Object o)
boolean containsAll(Collection c)
boolean isEmpty()
int size()
object[] toArray()
Iterator iterator()

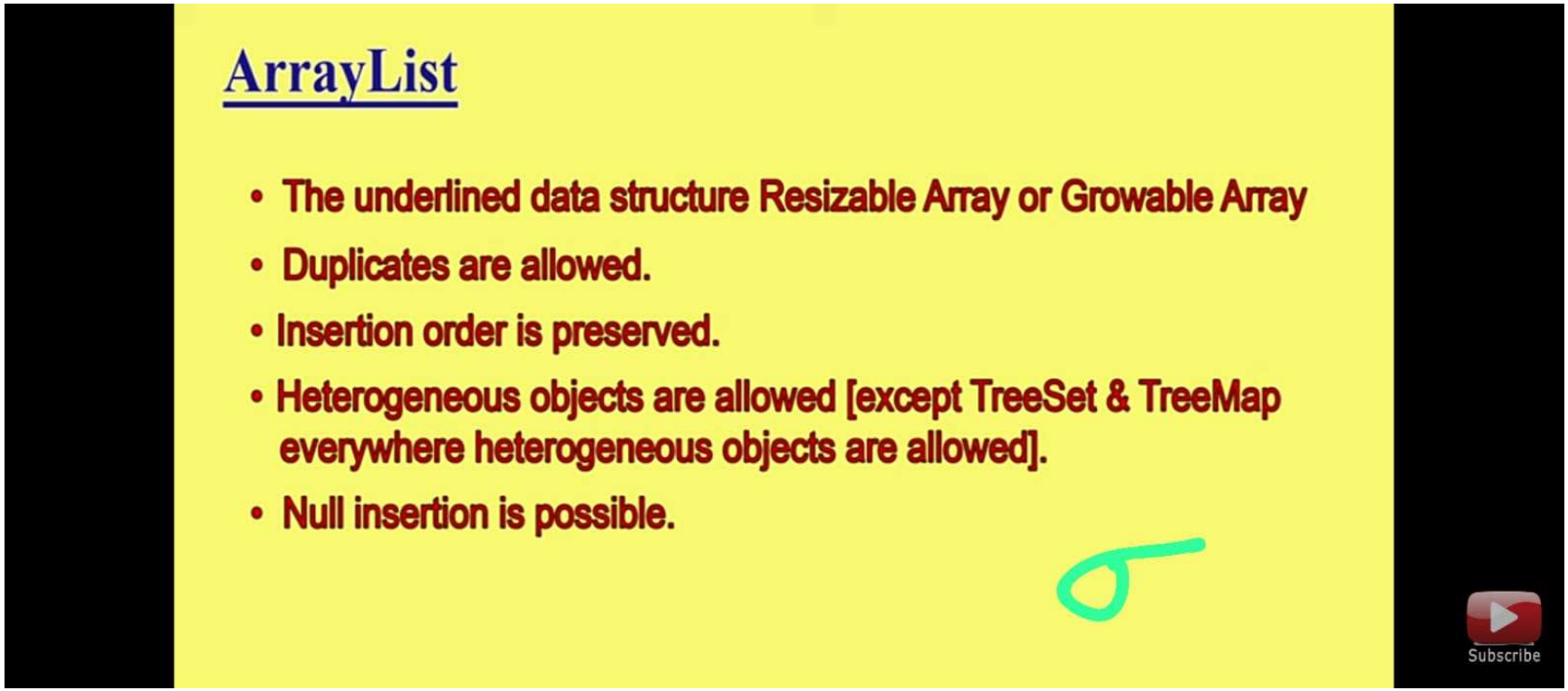

List Interface :
* It is the child interface of Collection.
* If we want to represent a group of individual objects as a single entity where duplicates are allowed and insertion order must be preserved then we should go for List.
* We can differentiate duplicates by using index.
* We can preserve insertion order by using index, hence index play very important role in list interface.
DURGASOFT
Subscribe

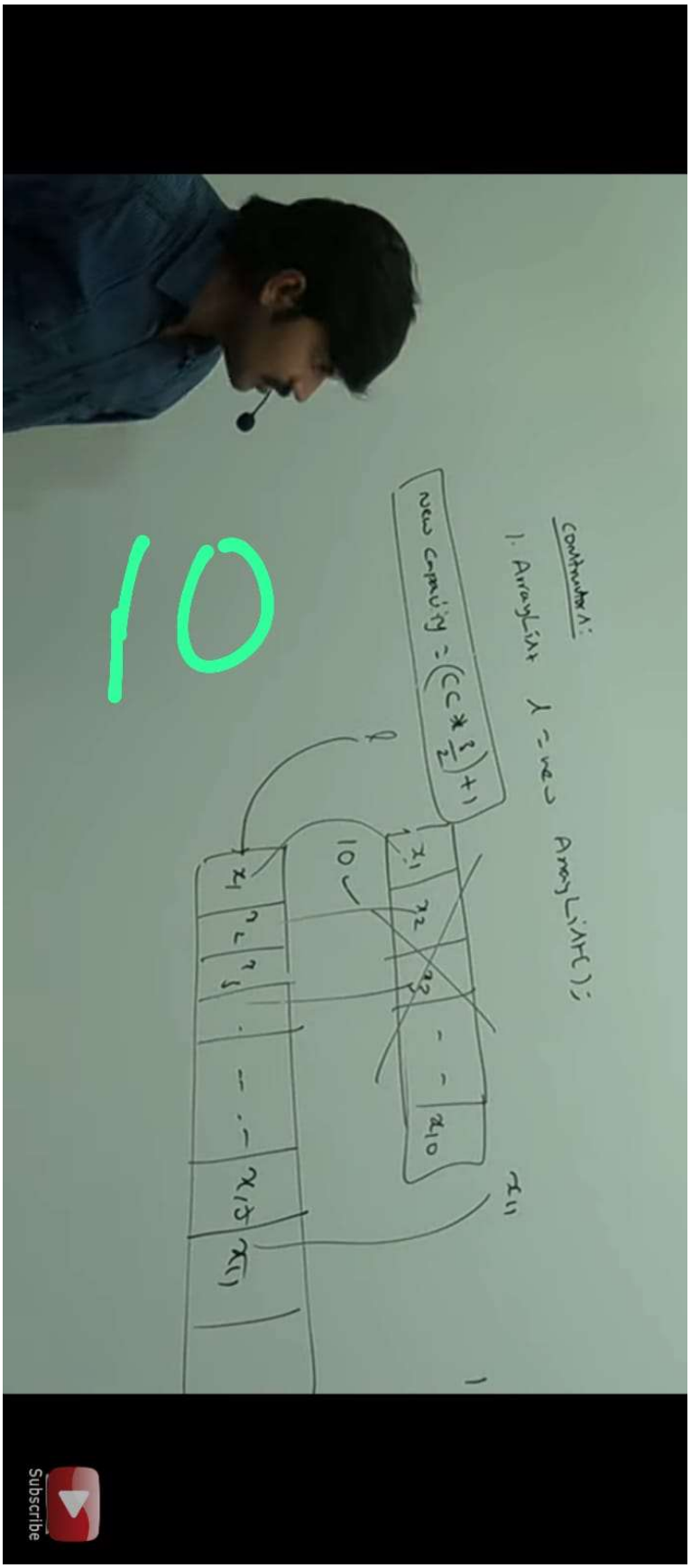

1 2 3 4 5
199 ------------------
200 void add(int index,Object o)
201 boolean addAll(int index,Collection c)
202 Object get(int index)
203 Object remove(int index)
204
205 Object set(int index, Object new)
206 to replace the element present at specified index with provided Object and returns old object
207
208 int indexOf(Object o)
209 returns index of first occurrence of 'o'
210
211 int lastIndexOf(Object o)
212 ListIterator listIterator();
213
214
215
216
217
218
DURGASOFT
Subscribe

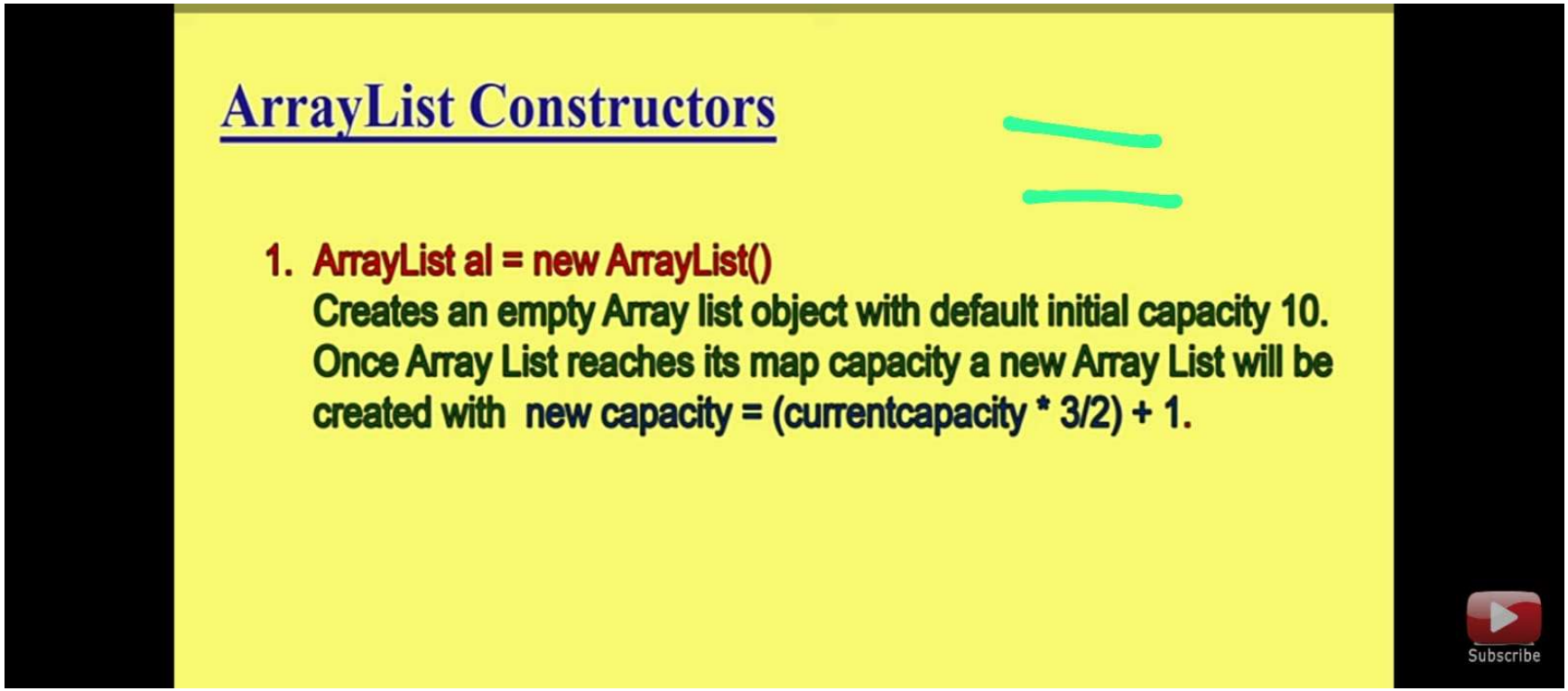

List interface specific methods
void add(int index, Object o)
boolean addAll(int index, Collection c)
object get(int index)
object remove(int index)
object set(int index, Object new)
int indexOf(Object o)
int lastIndexOf(Object o)
ListIterator listIterator();
DURGASOFT
Subscribe

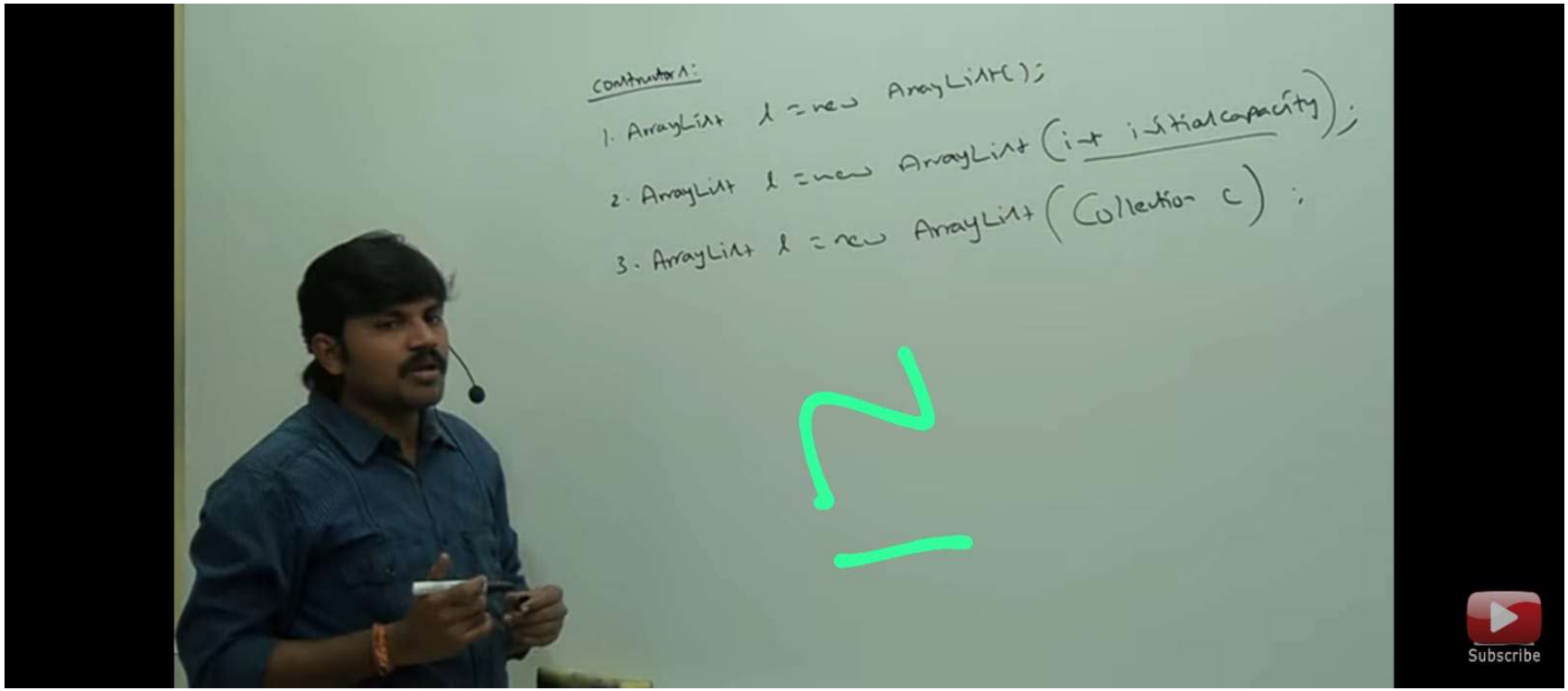

Collection (I)
List (I)
AL
LL
Vector
Stack
DURGASOFT
Subscribe

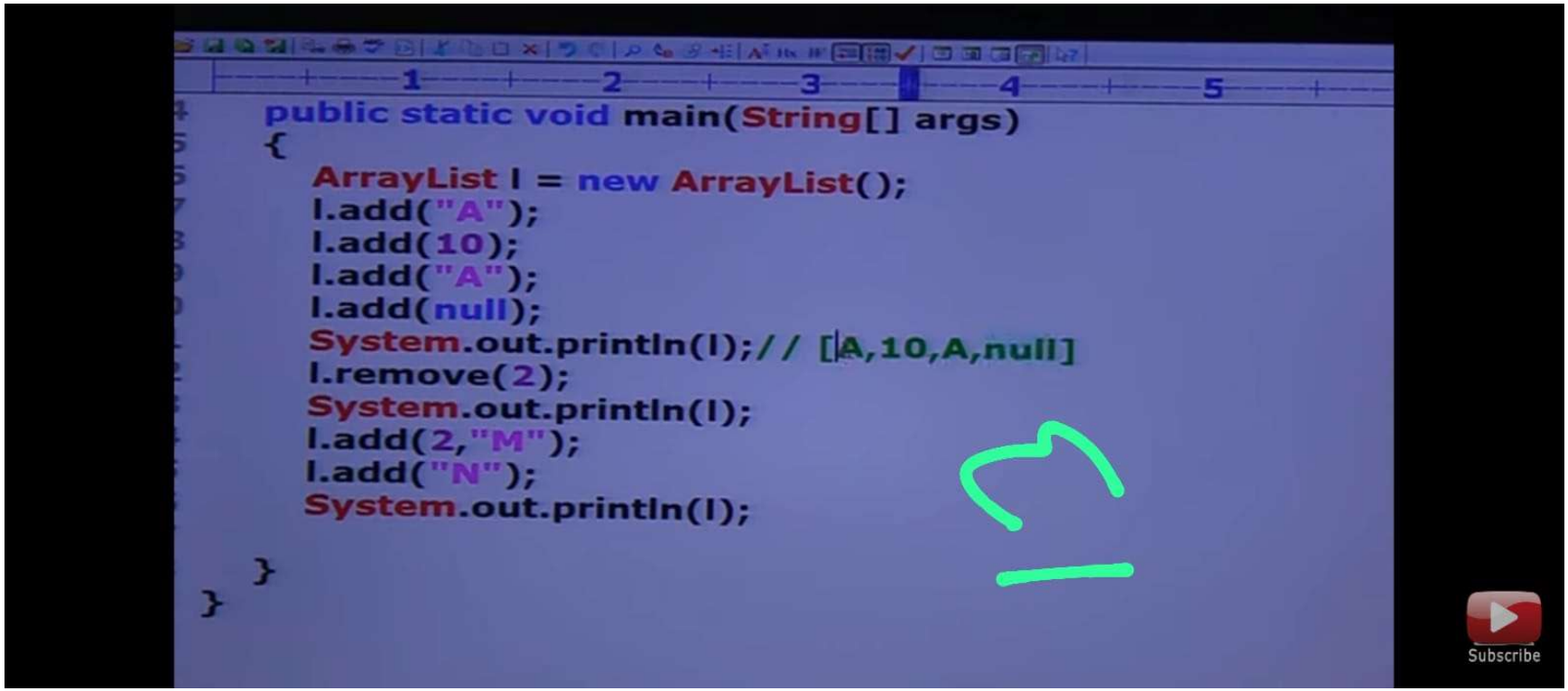

ArrayList:
1 Resizable Array or Growable Array
2 Duplicates ✓
3 insertion order ✓
4 Heterogeneous Objects ✓
[TreeSet & TreeMap]
5 "null" insertion ✓
TreeSet
TreeMap
Subscribe


ArrayList
• The underlined data structure Resizable Array or Growable Array
• Duplicates are allowed.
• Insertion order is preserved.
• Heterogeneous objects are allowed [except TreeSet & TreeMap everywhere heterogeneous objects are allowed].
• Null insertion is possible.
Subscribe


Constructor:
1. ArrayList x = new ArrayList();
New capacity = (CC * 3/2) + 1
10
Subscribe


ArrayList Constructors
1. ArrayList al = new ArrayList()
Creates an empty Array list object with default initial capacity 10.
Once Array List reaches its map capacity a new Array List will be created with new capacity = (currentcapacity * 3/2) + 1.
Subscribe

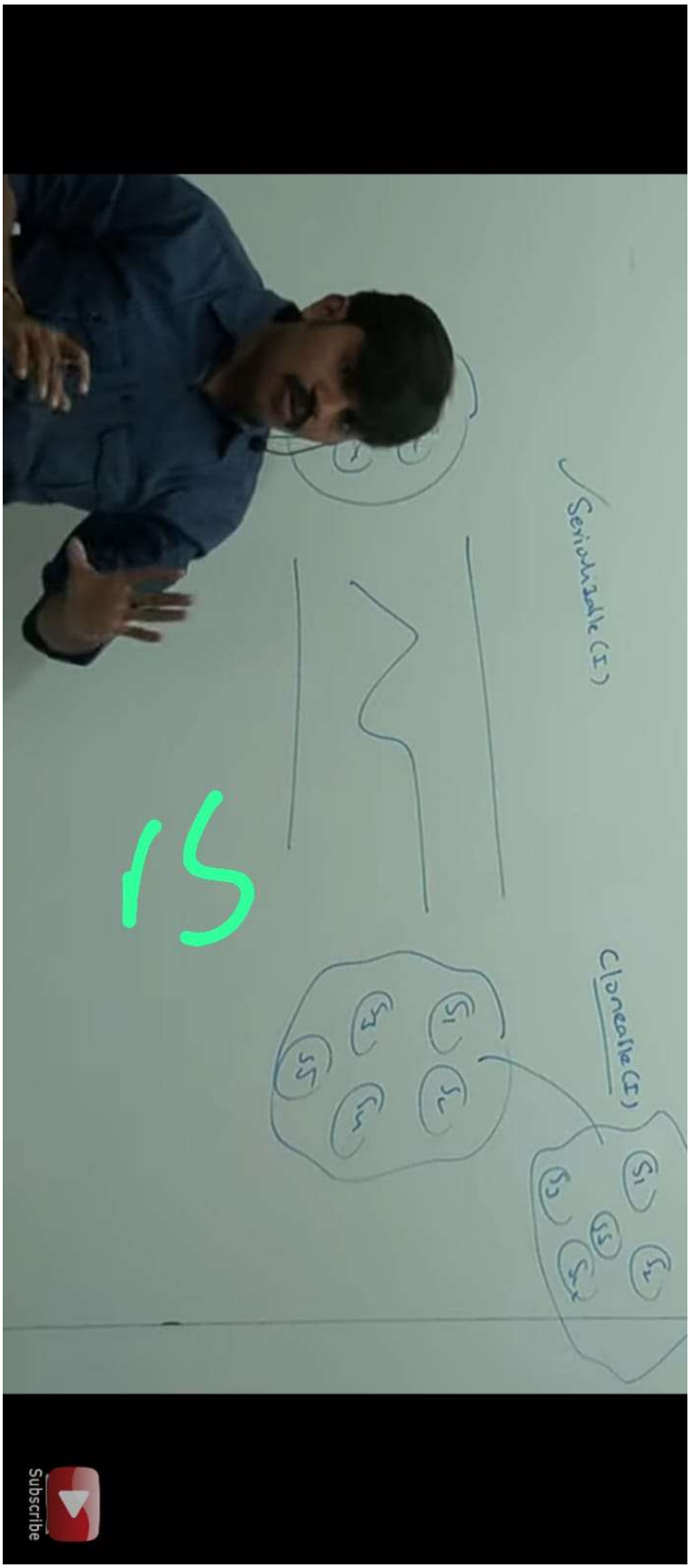

Constructors:
1. ArrayList l = new ArrayList();
2. ArrayList l = new ArrayList(int initialCapacity);
3. ArrayList l = new ArrayList(Collection c);
Subscribe

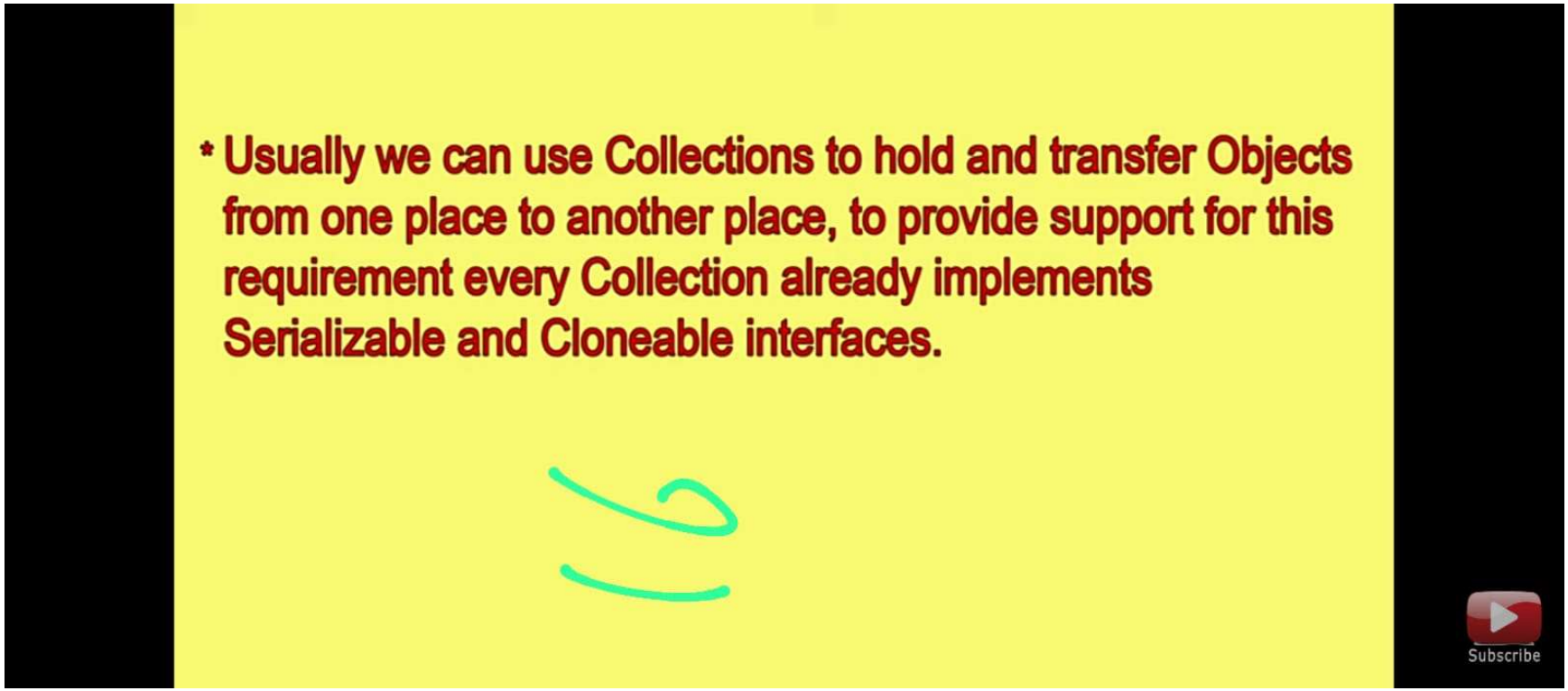

public static void main(String[] args)
{
ArrayList l = new ArrayList();
l.add("A");
l.add(10);
l.add("A");
l.add(null);
System.out.println(l);// [A,10,A,null]
l.remove(2);
System.out.println(l);
l.add(2,"M");
l.add("N");
System.out.println(l);
}
}
Subscribe

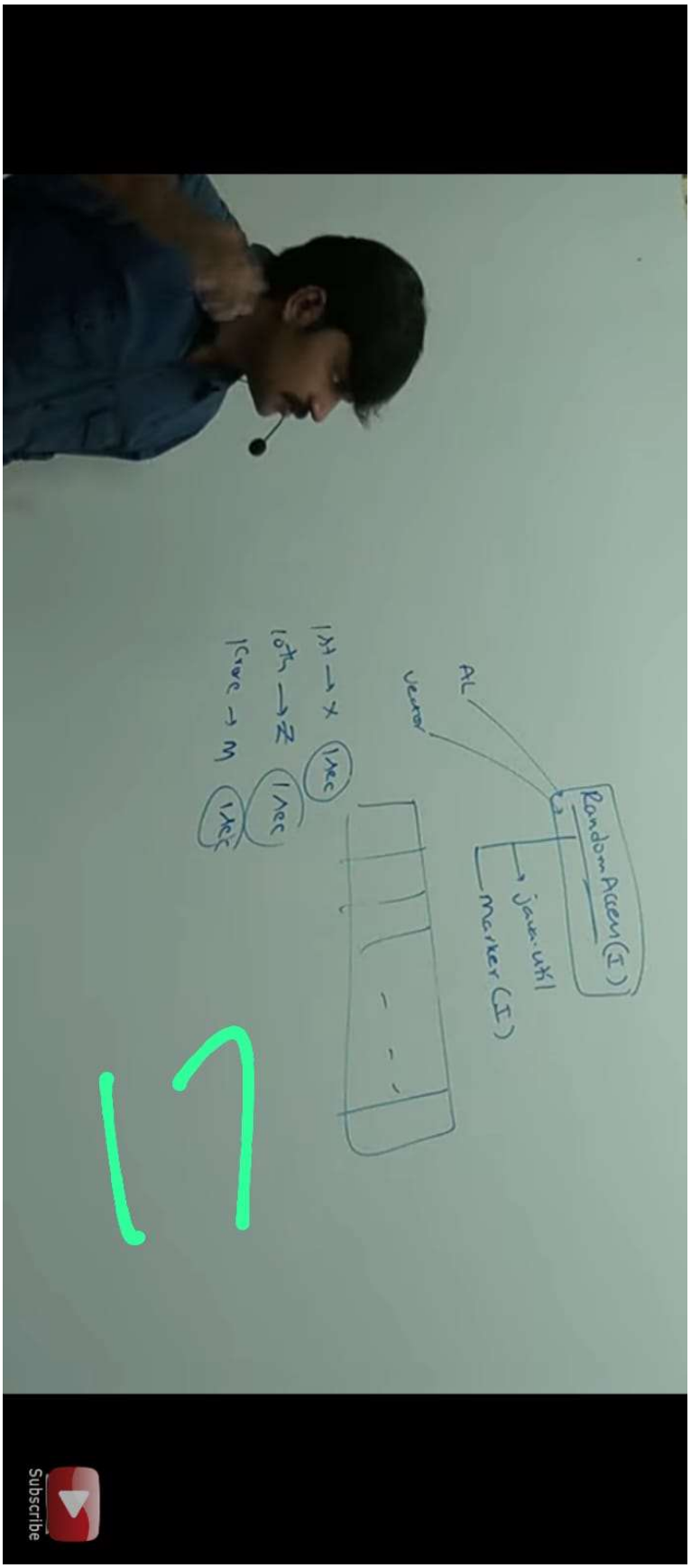
```
import java.util.*;
class ArrayListDemo
{
    public static void main(String[] args)
    {
        ArrayList l = new ArrayList();
        l.add("A");
        l.add(10);
        l.add("A");
        l.add(null);
        System.out.println(l);// [A,10,A,null]
        l.remove(2);
        System.out.println(l);// [A,10,null]
        l.add(2,"M");
        l.add("N");
        System.out.println(l);//[A,10,M,null,N]
    }
}
```

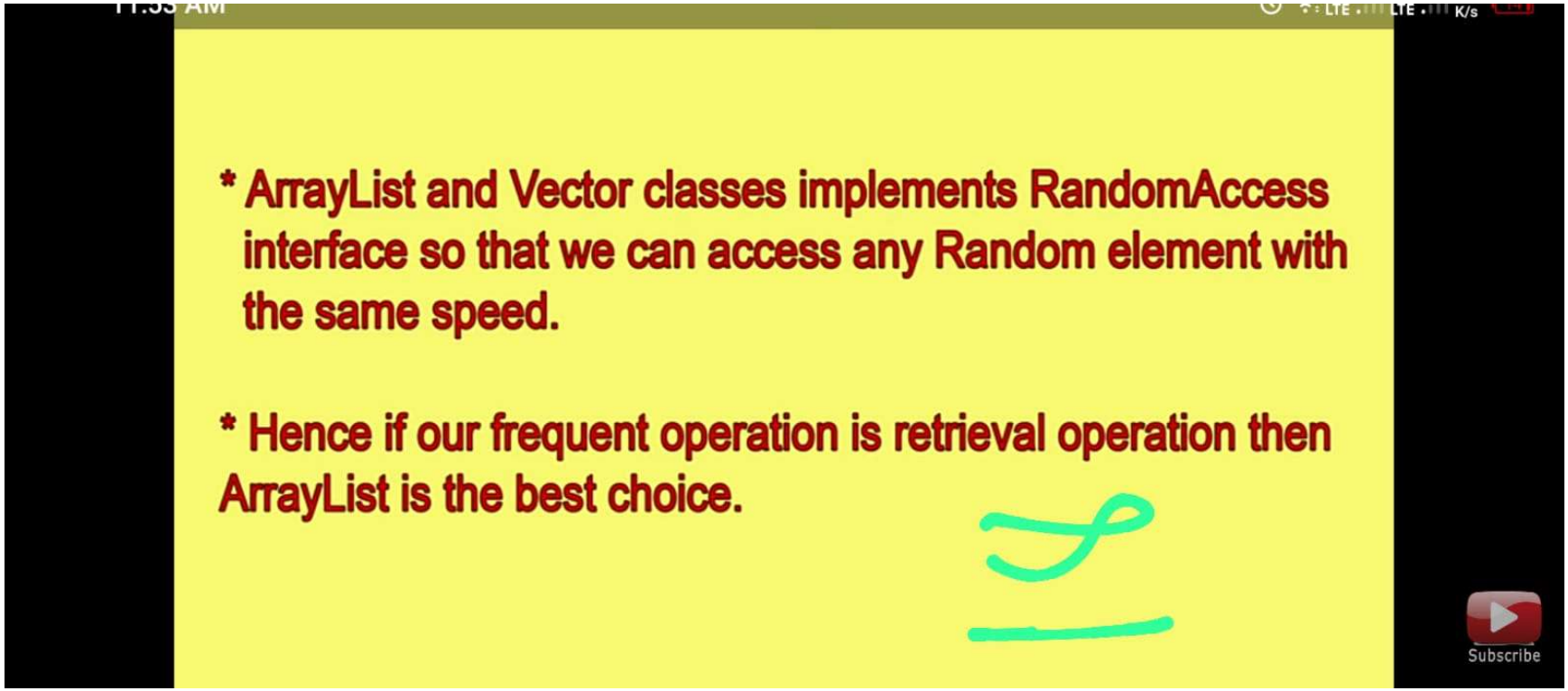
14

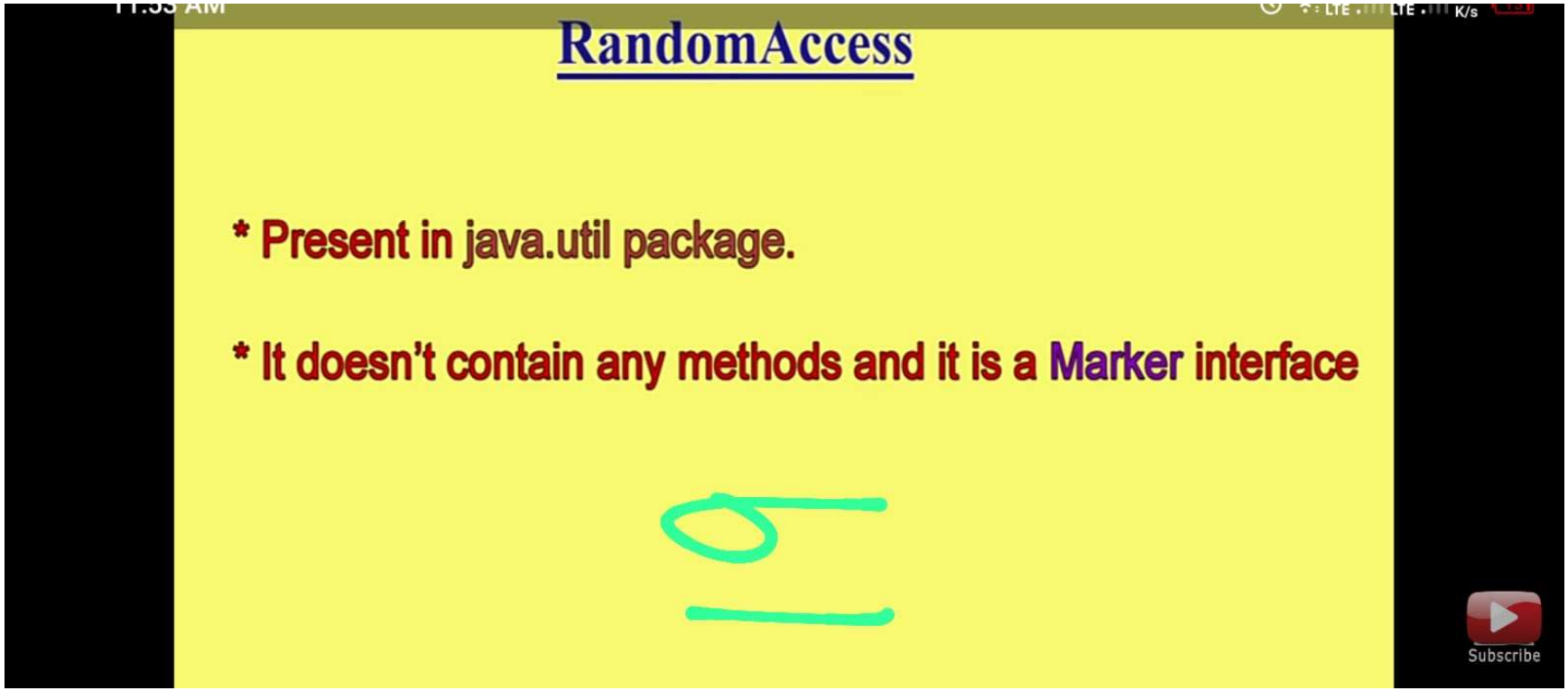
Subscribe

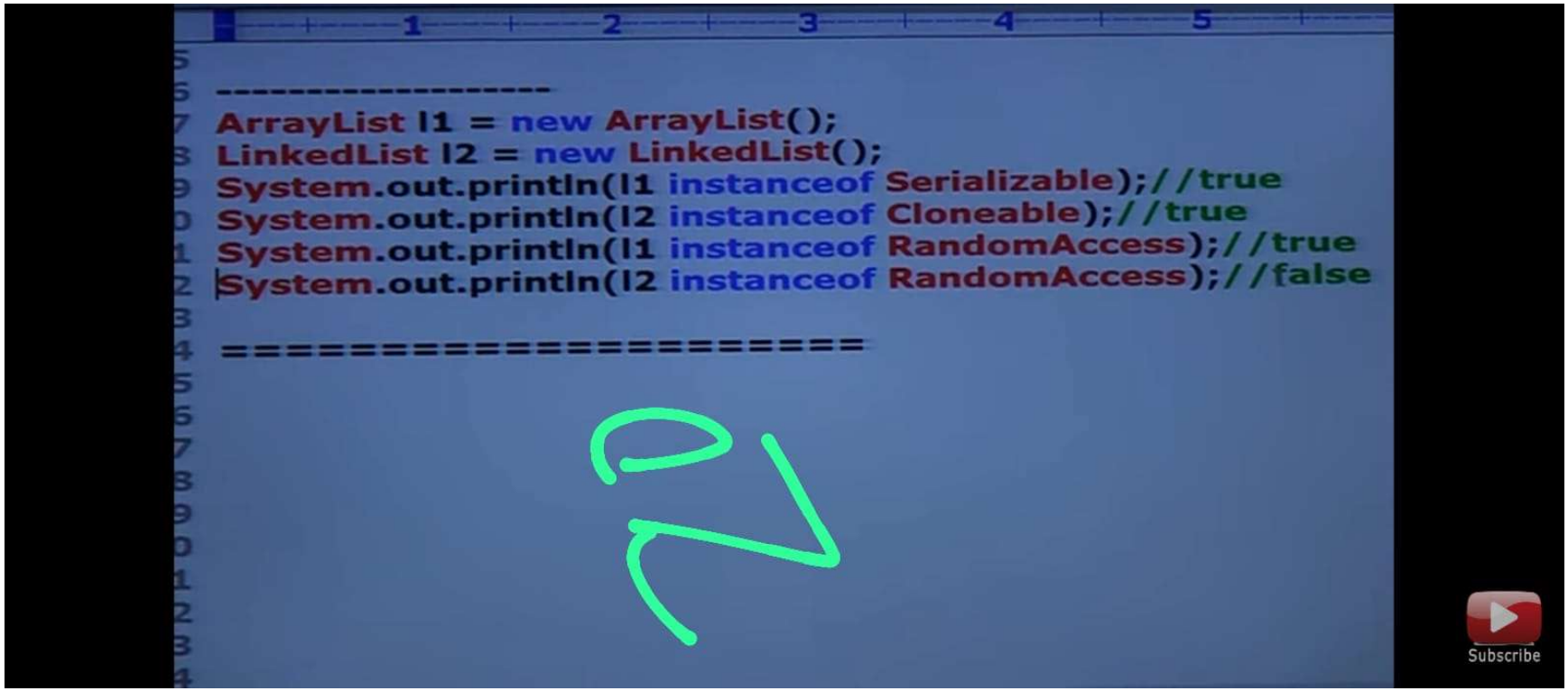

15
Serializable (I)
Cloneable (I)
Subscribe


* Usually we can use Collections to hold and transfer Objects from one place to another place, to provide support for this requirement every Collection already implements Serializable and Cloneable interfaces.
Subscribe

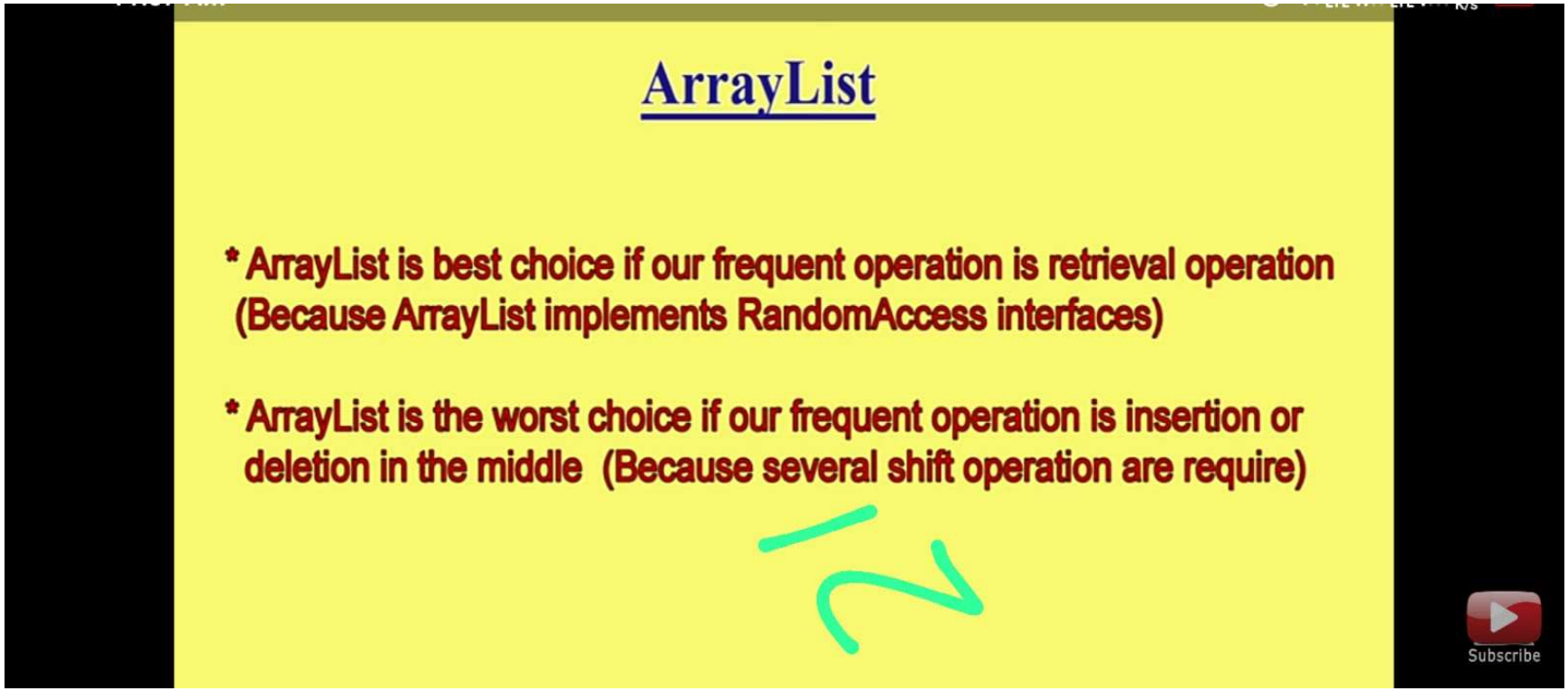

RandomAccess(I)
java.util
Marker (I)
AL
Vector
1st → X
10th → Z
15th → M
1Sec
1Sec
1Sec
17
Subscribe

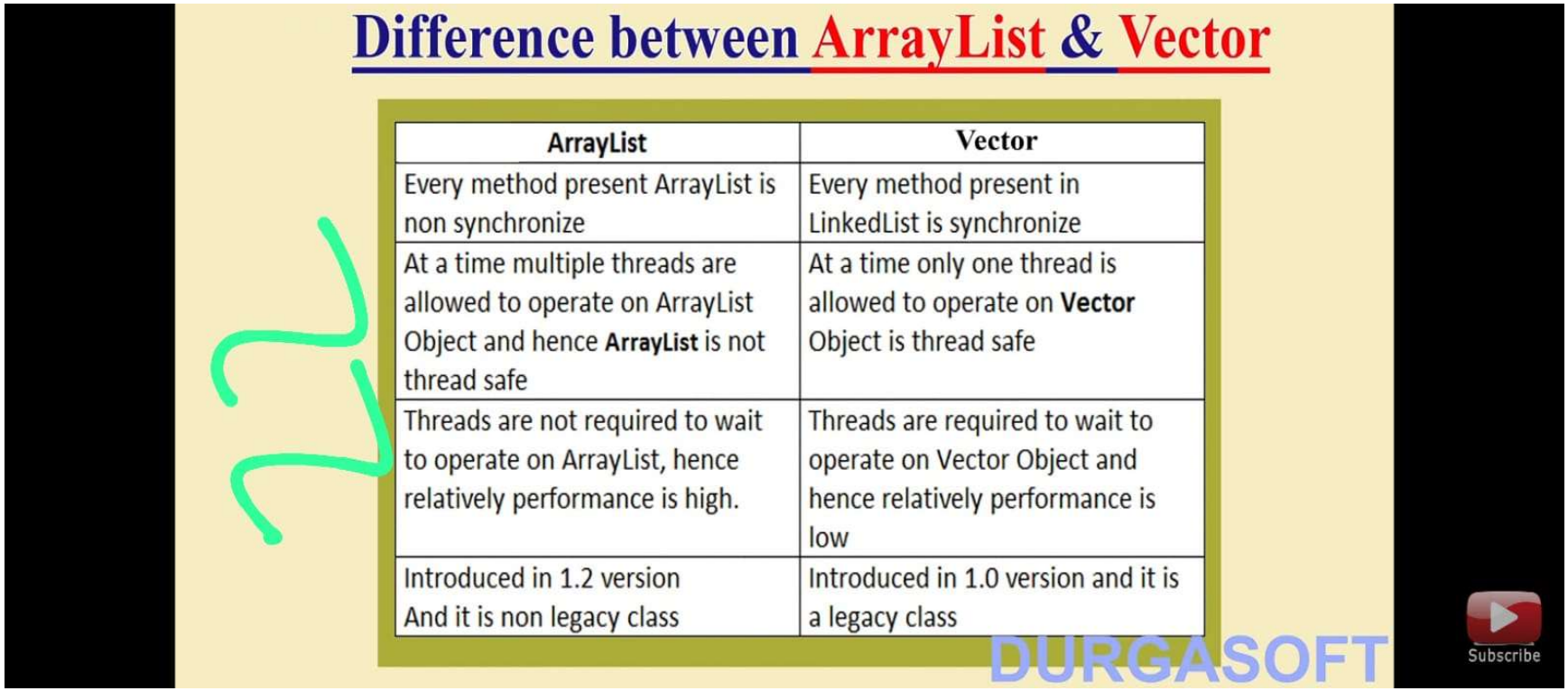

* ArrayList and Vector classes implements RandomAccess interface so that we can access any Random element with the same speed.
* Hence if our frequent operation is retrieval operation then ArrayList is the best choice.
Subscribe

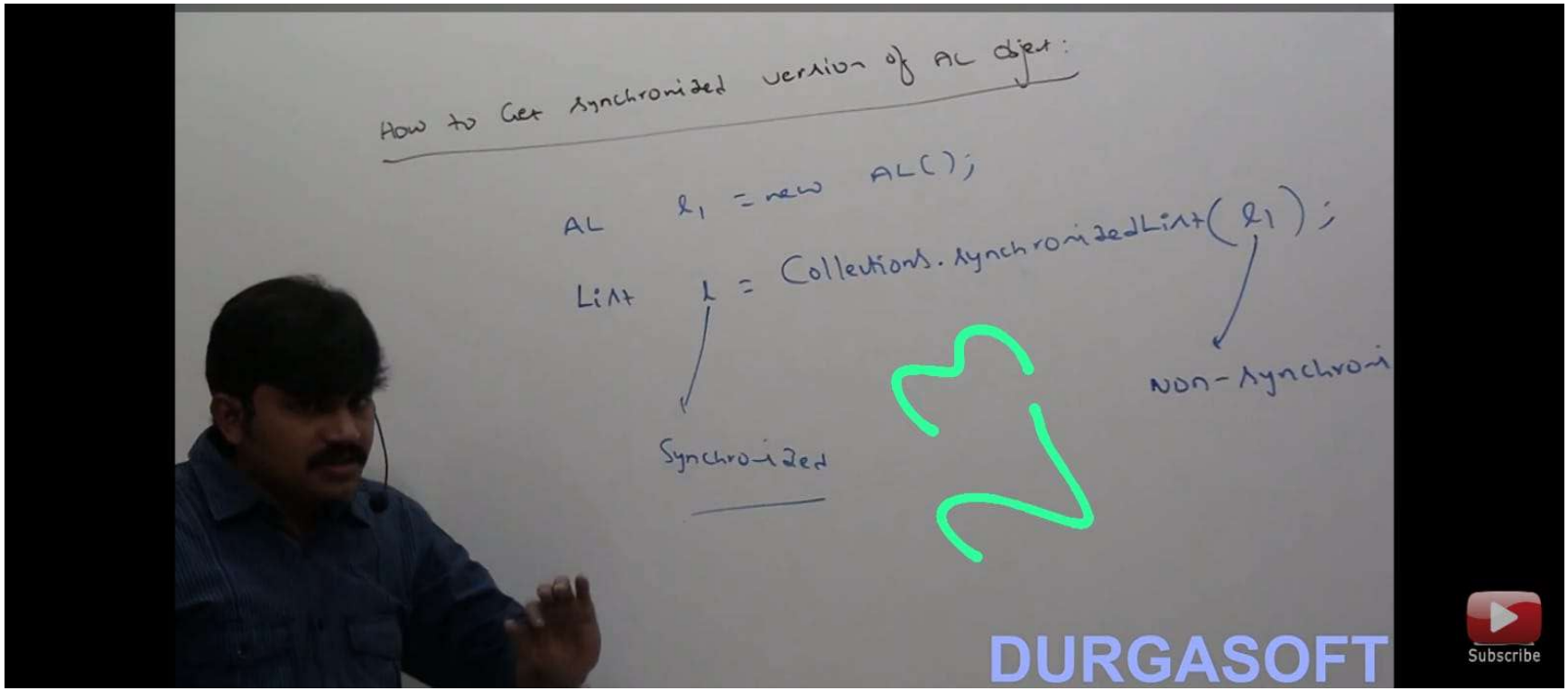

RandomAccess
* Present in java.util package.
* It doesn't contain any methods and it is a Marker interface
Subscribe

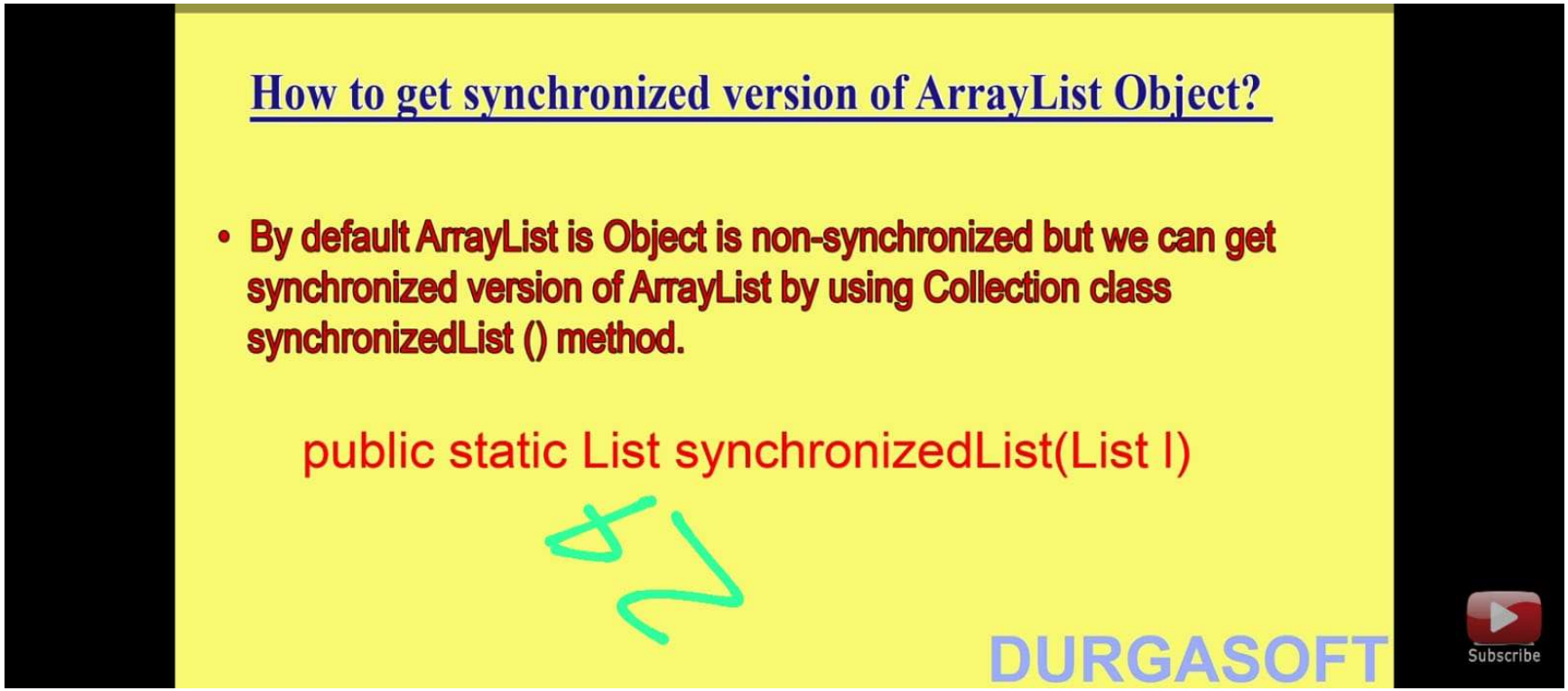

1 2 3 4 5

ArrayList l1 = new ArrayList();
LinkedList l2 = new LinkedList();
System.out.println(l1 instanceof Serializable);//true
System.out.println(l2 instanceof Cloneable);//true
System.out.println(l1 instanceof RandomAccess);//true
System.out.println(l2 instanceof RandomAccess);//false
==========
22
Subscribe


ArrayList
* ArrayList is best choice if our frequent operation is retrieval operation (Because ArrayList implements RandomAccess interfaces)
* ArrayList is the worst choice if our frequent operation is insertion or deletion in the middle (Because several shift operation are require)
Subscribe


Difference between ArrayList & Vector
ArrayList
Vector
Every method present ArrayList is non synchronize
Every method present in LinkedList is synchronize
At a time multiple threads are allowed to operate on ArrayList Object and hence ArrayList is not thread safe
At a time only one thread is allowed to operate on Vector Object is thread safe
Threads are not required to wait to operate on ArrayList, hence relatively performance is high.
Threads are required to wait to operate on Vector Object and hence relatively performance is low
Introduced in 1.2 version And it is non legacy class
Introduced in 1.0 version and it is a legacy class
DURGASOFT
Subscribe


How to Get Synchronized version of AL object:
AL l1 = new AL();
List l = Collections.synchronizedList(l1);
Synchronized
Non-Synchroni
DURGASOFT
Subscribe


How to get synchronized version of ArrayList Object?
By default ArrayList is Object is non-synchronized but we can get synchronized version of ArrayList by using Collection class synchronizedList () method.
public static List synchronizedList(List l)
DURGASOFT
Subscribe

How to get synchronized version of ArrayList Object?

public static List synchronizedList(List l)

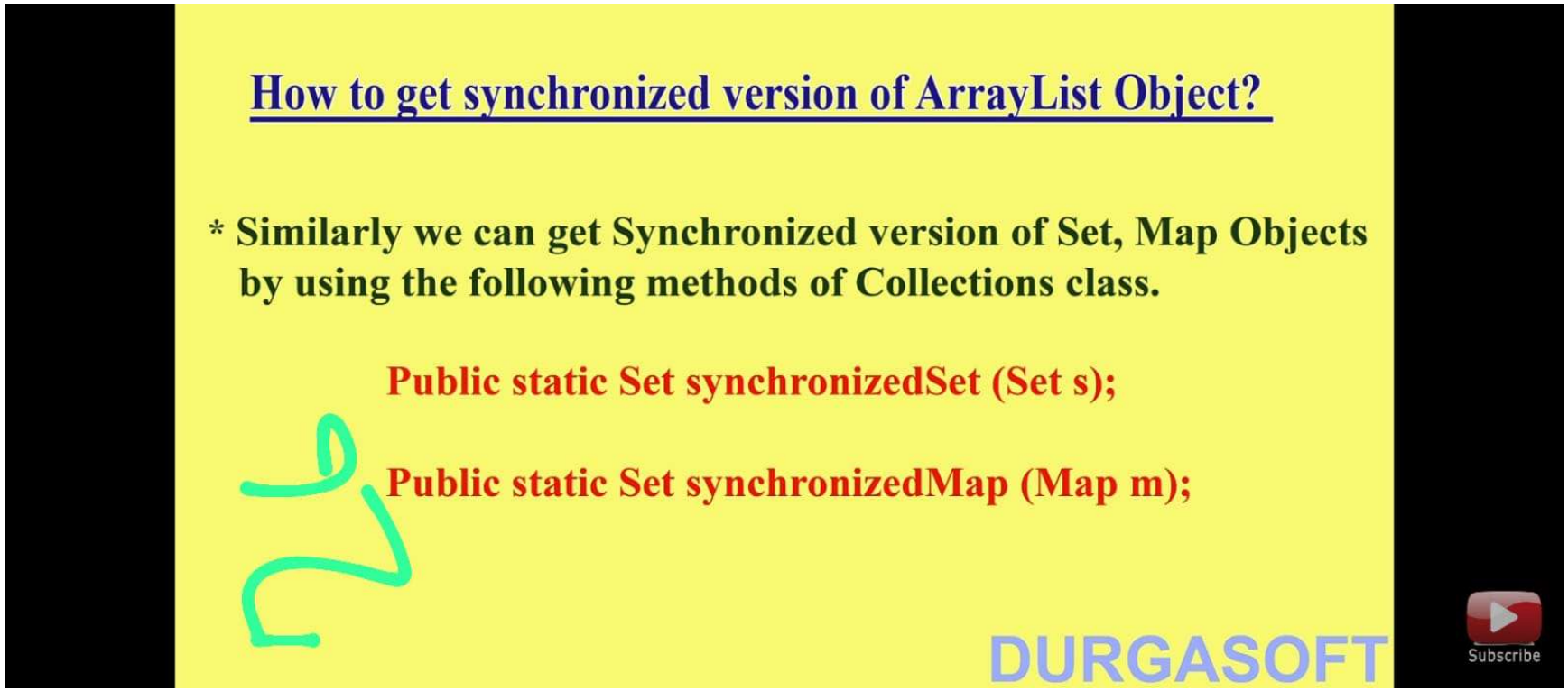
Non-Synchronized

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

Synchronized

```
List l=Collections.synchronizedList (l1);
```

DURGASOFT


How to get synchronized version of ArrayList Object?
* Similarly we can get Synchronized version of Set, Map Objects by using the following methods of Collections class.
Public static Set synchronizedSet (Set s);
Public static Set synchronizedMap (Map m);
DURGASOFT
Subscribe