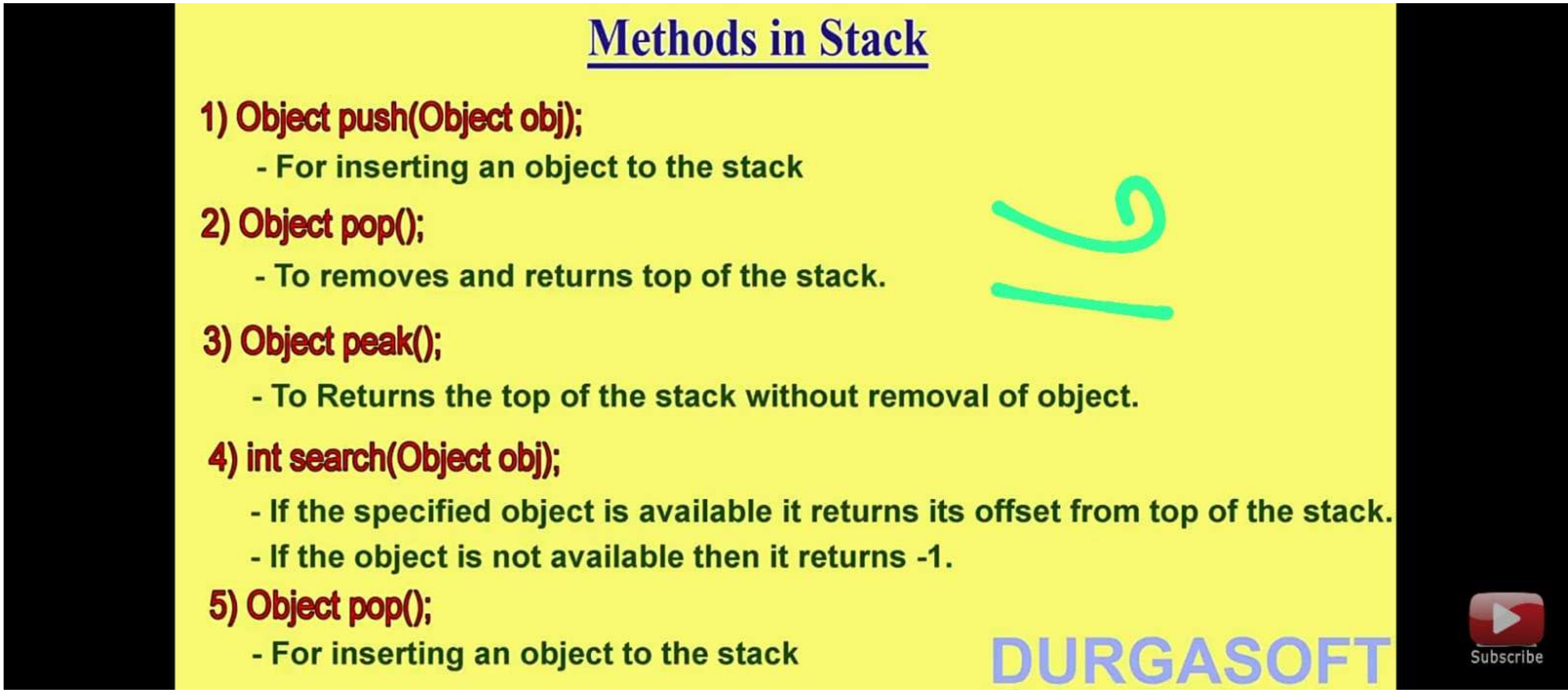
4CFLinkedList_Vector_Stack



Methods in Stack

1) Object push(Object obj);
   - For inserting an object to the stack

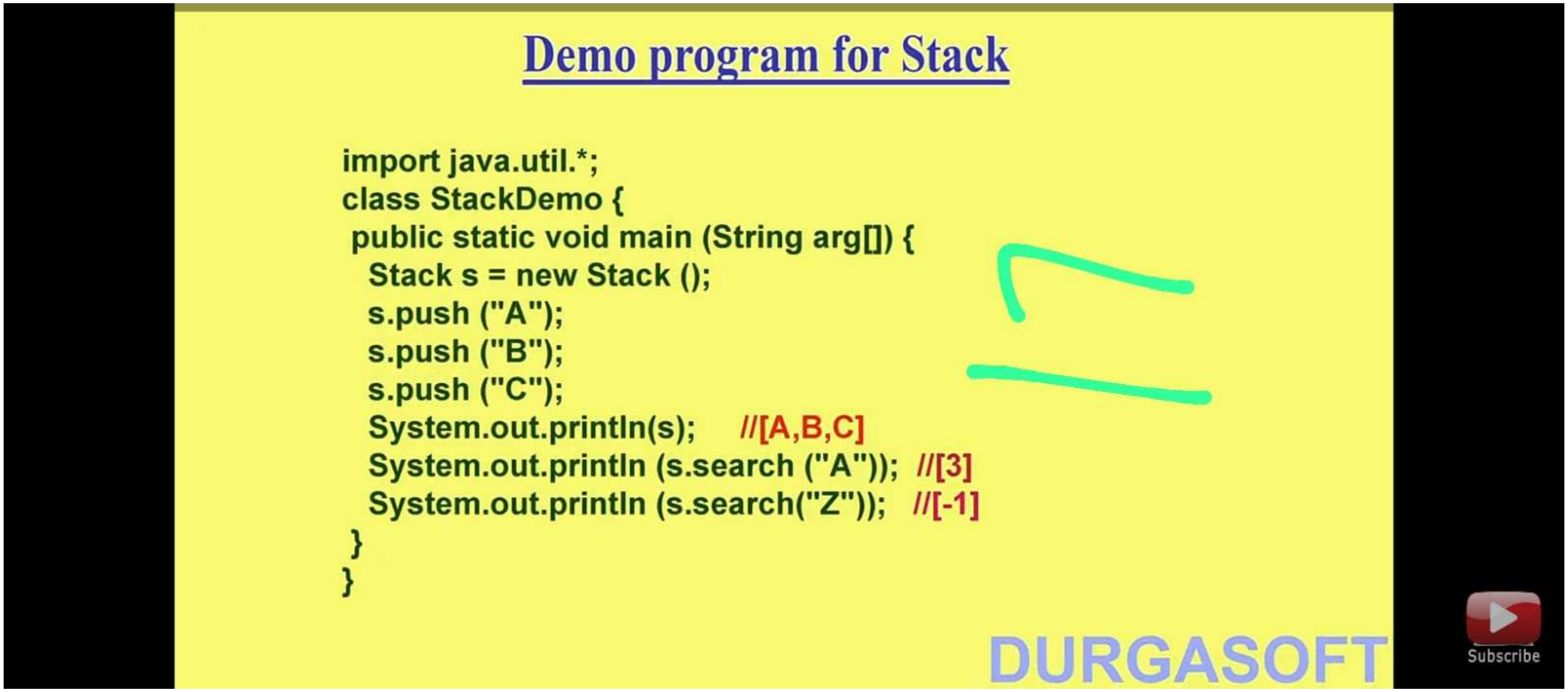
2) Object pop();
   - To removes and returns top of the stack.

3) Object peak();
   - To Returns the top of the stack without removal of object.

4) int search(Object obj);
   - If the specified object is available it returns its offset from top of the stack.
   - If the object is not available then it returns -1.

5) Object pop();
   - For inserting an object to the stack

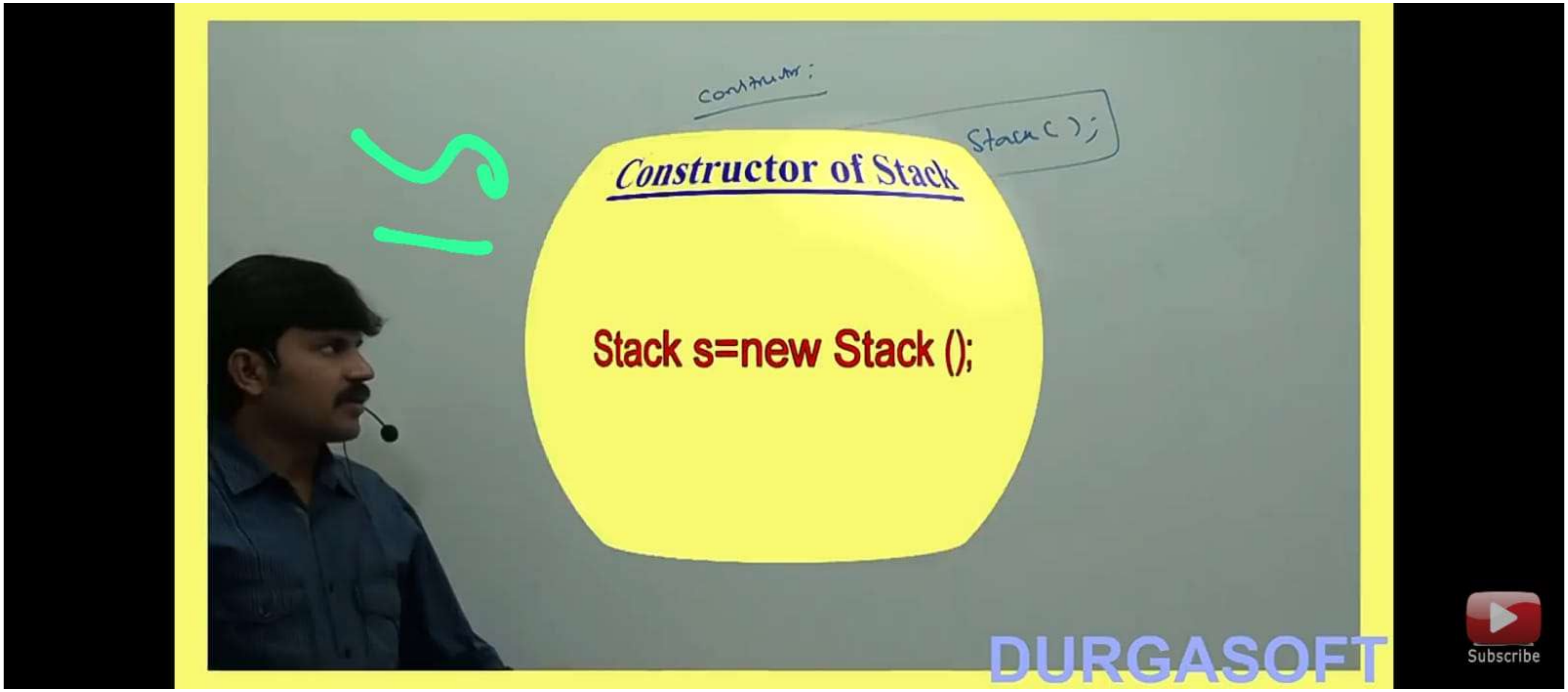
16



Demo program for Stack

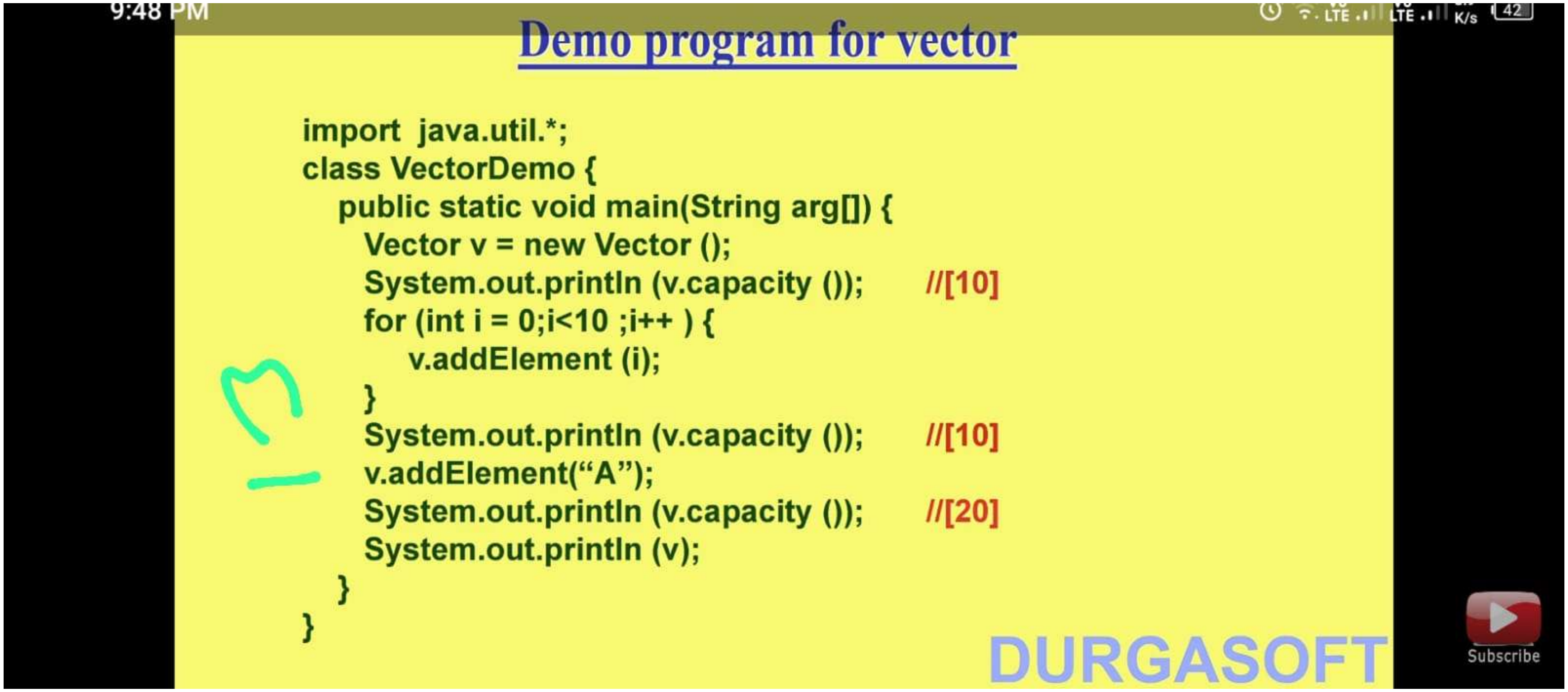
```
import java.util.*;
class StackDemo {
 public static void main (String arg[]) {
  Stack s = new Stack ();
  s.push ("A");
  s.push ("B");
  s.push ("C");
  System.out.println(s);    //[A,B,C]
  System.out.println (s.search ("A")); //[3]
  System.out.println (s.search("Z"));  //[-1]
 }
}
```

17



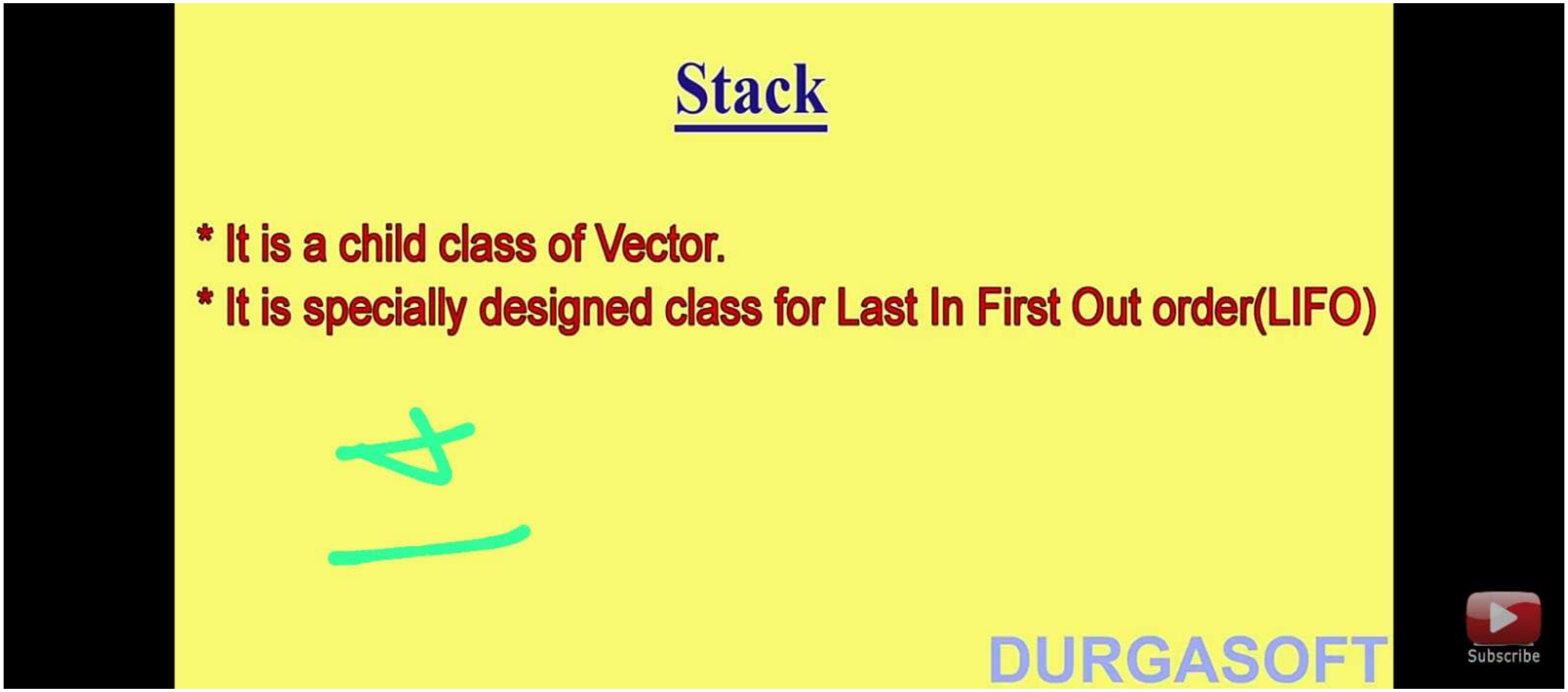
Constructor of Stack

```
Stack s=new Stack ();
```



Demo program for vector

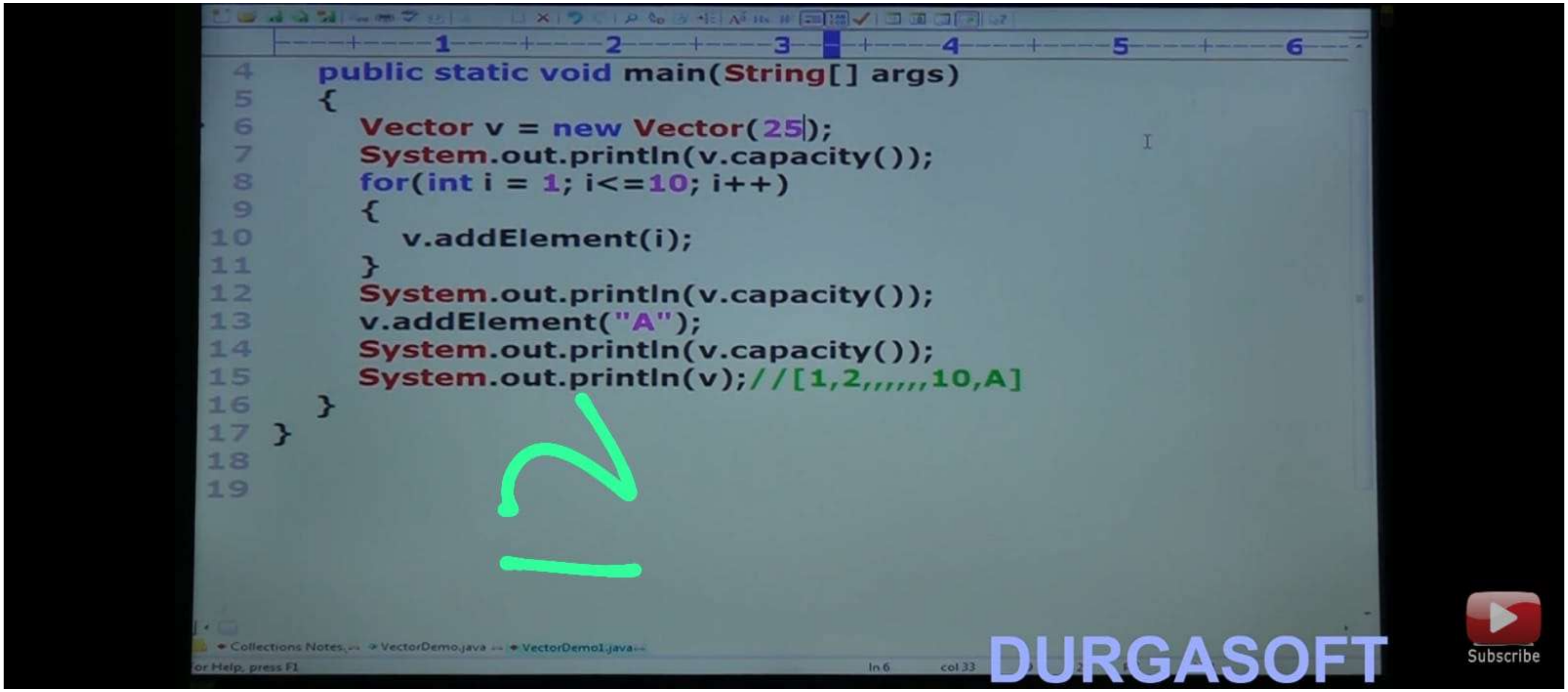
```
import  java.util.*;
class VectorDemo {
   public static void main(String arg[]) {
     Vector v = new Vector ();
     System.out.println (v.capacity ());       //[10]
     for (int i = 0;i<10 ;i++ ) {
         v.addElement (i);
     }
     System.out.println (v.capacity ());       //[10]
     v.addElement("A");
     System.out.println (v.capacity ());       //[20]
     System.out.println (v);
   }
}
```



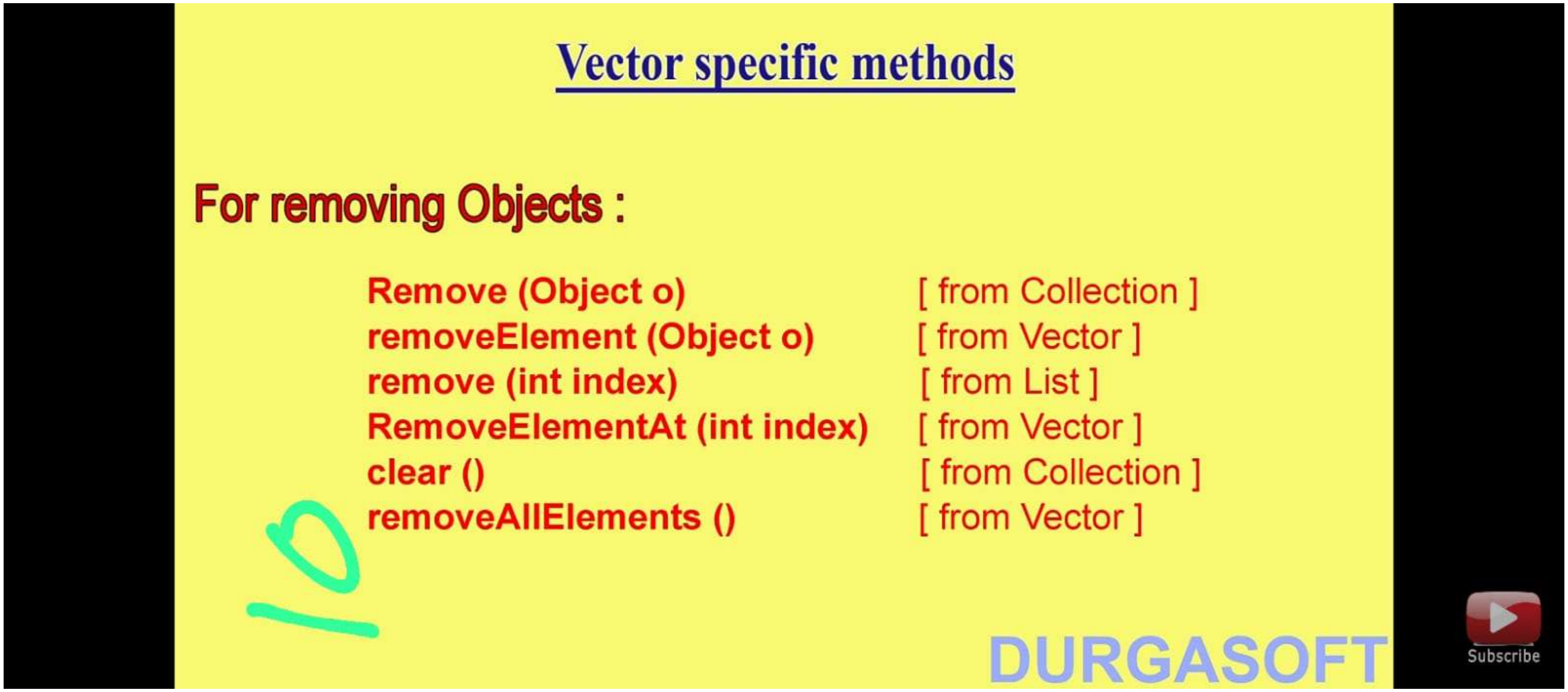
Stack

- * It is a child class of Vector.
- * It is specially designed class for Last In First Out order(LIFO)

14



```java
    public static void main(String[] args)
    {
       Vector v = new Vector(25);
       System.out.println(v.capacity());
       for(int i = 1; i<=10; i++)
       {
          v.addElement(i);
       }
       System.out.println(v.capacity());
       v.addElement("A");
       System.out.println(v.capacity());
       System.out.println(v);//[1,2,,,,,,10,A]
    }
}
```

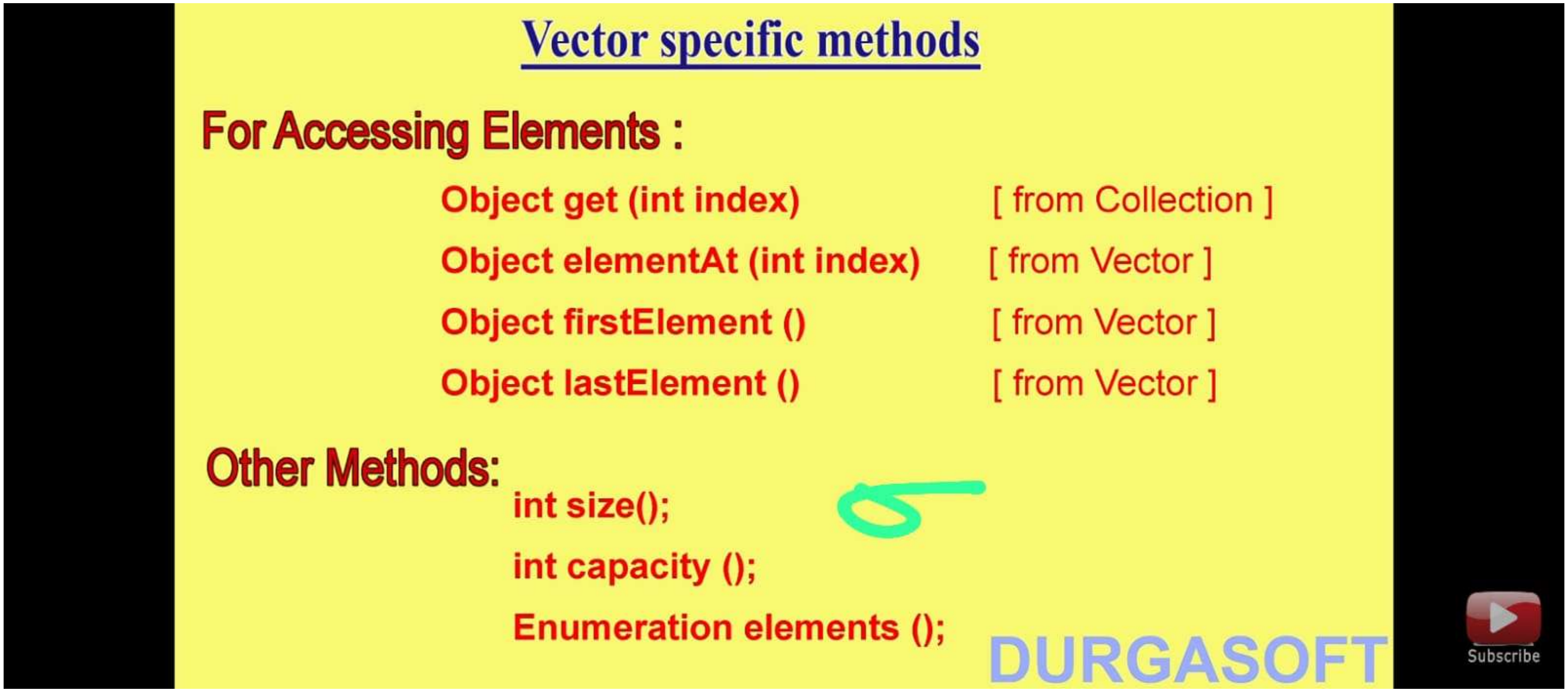


Vector specific methods

For removing Objects :

Remove (Object o)	[from Collection]
removeElement (Object o)	[from Vector]
remove (int index)	[from List]
RemoveElementAt (int index)	[from Vector]
clear ()	[from Collection]
removeAllElements ()	[from Vector]

10



Vector specific methods

For Accessing Elements :

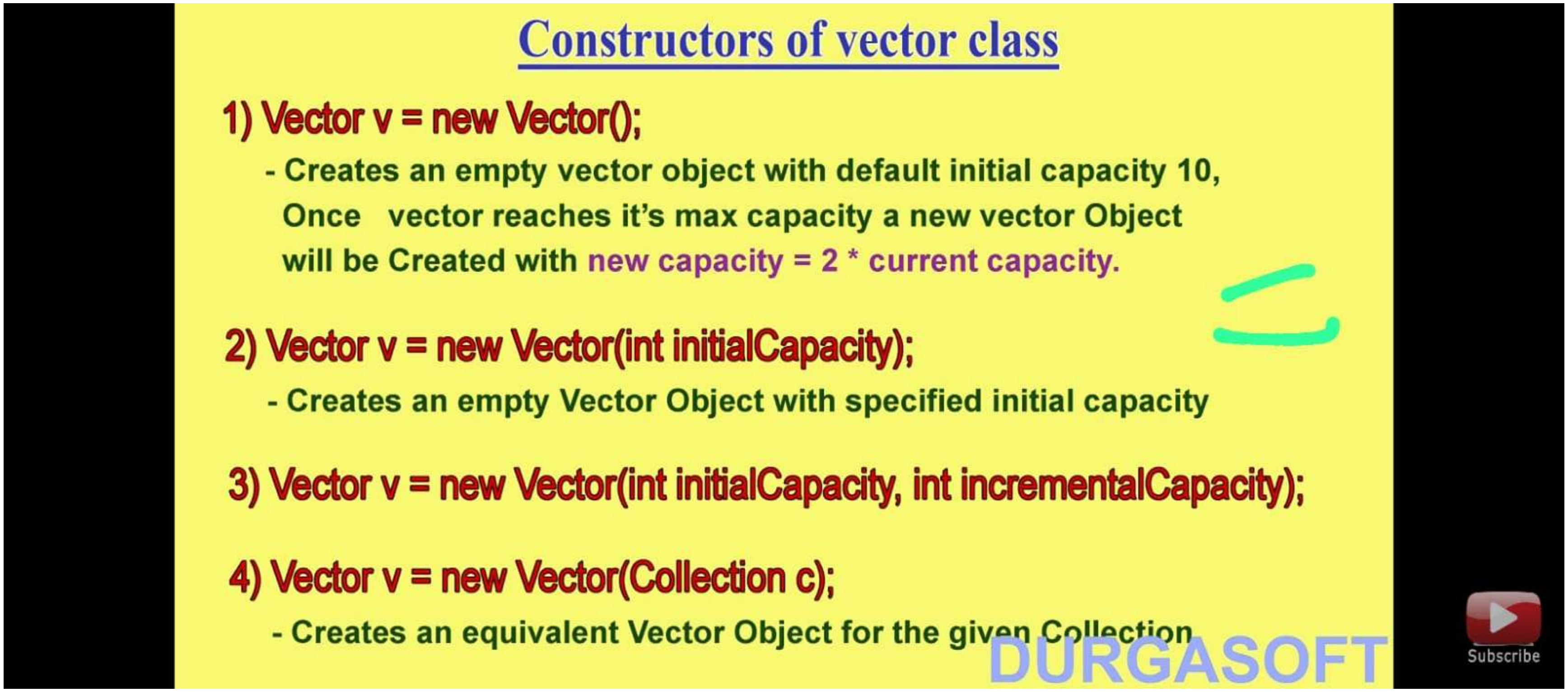
Object get (int index)	[from Collection]
Object elementAt (int index)	[from Vector]
Object firstElement ()	[from Vector]
Object lastElement ()	[from Vector]

Other Methods:

int size();

int capacity ();

Enumeration elements ();



Constructors of vector class

1) Vector v = new Vector();

- Creates an empty vector object with default initial capacity 10, Once vector reaches it's max capacity a new vector Object will be Created with new capacity = 2 * current capacity.

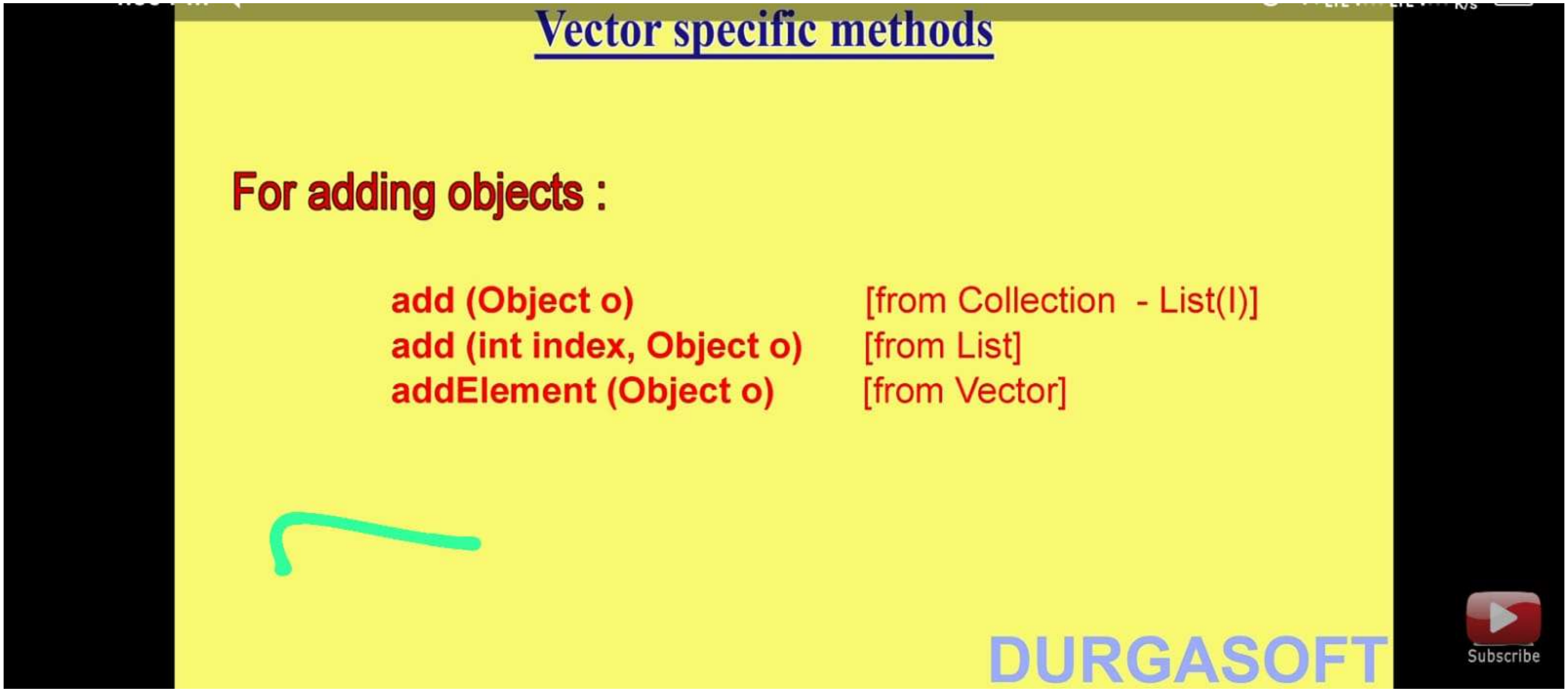
2) Vector v = new Vector(int initialCapacity);

- Creates an empty Vector Object with specified initial capacity

3) Vector v = new Vector(int initialCapacity, int incrementalCapacity);

4) Vector v = new Vector(Collection c);

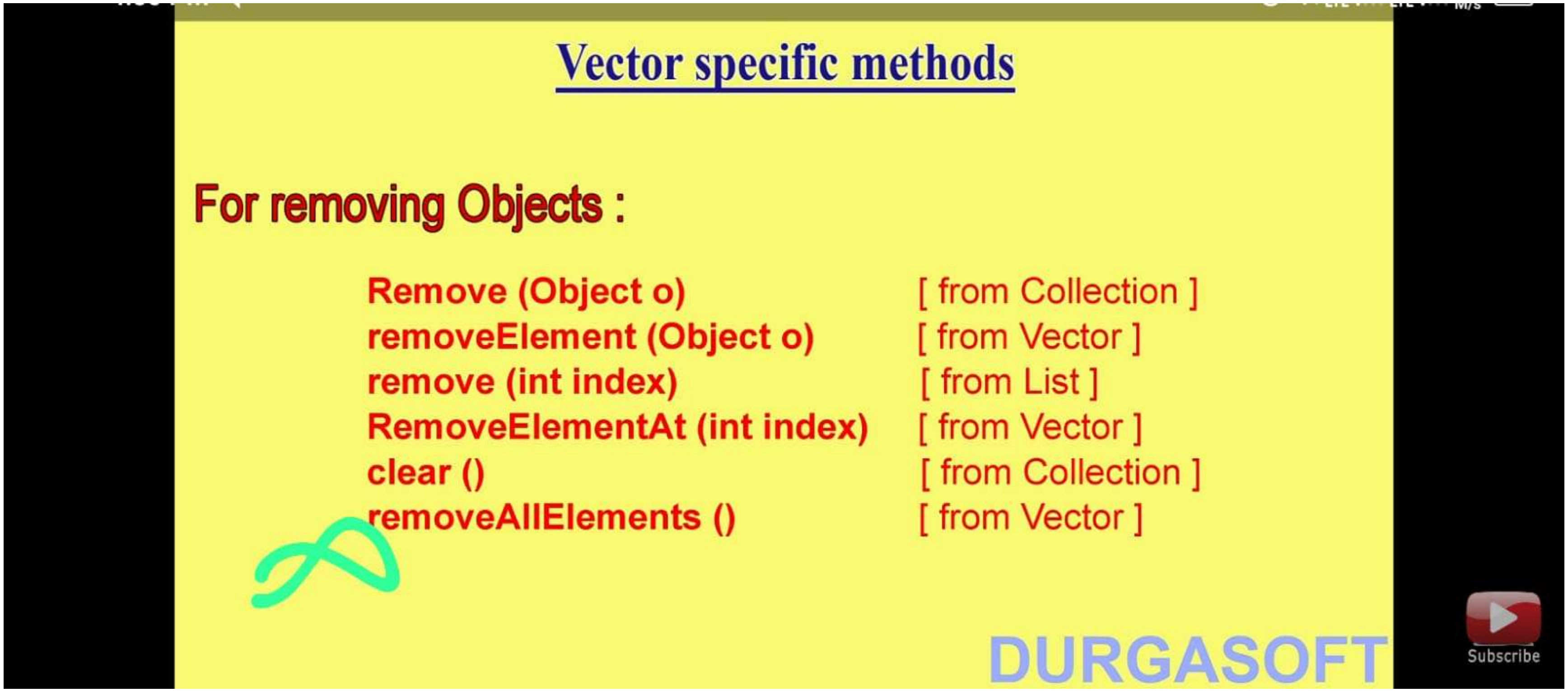
- Creates an equivalent Vector Object for the given Collection



Vector specific methods

For adding objects :

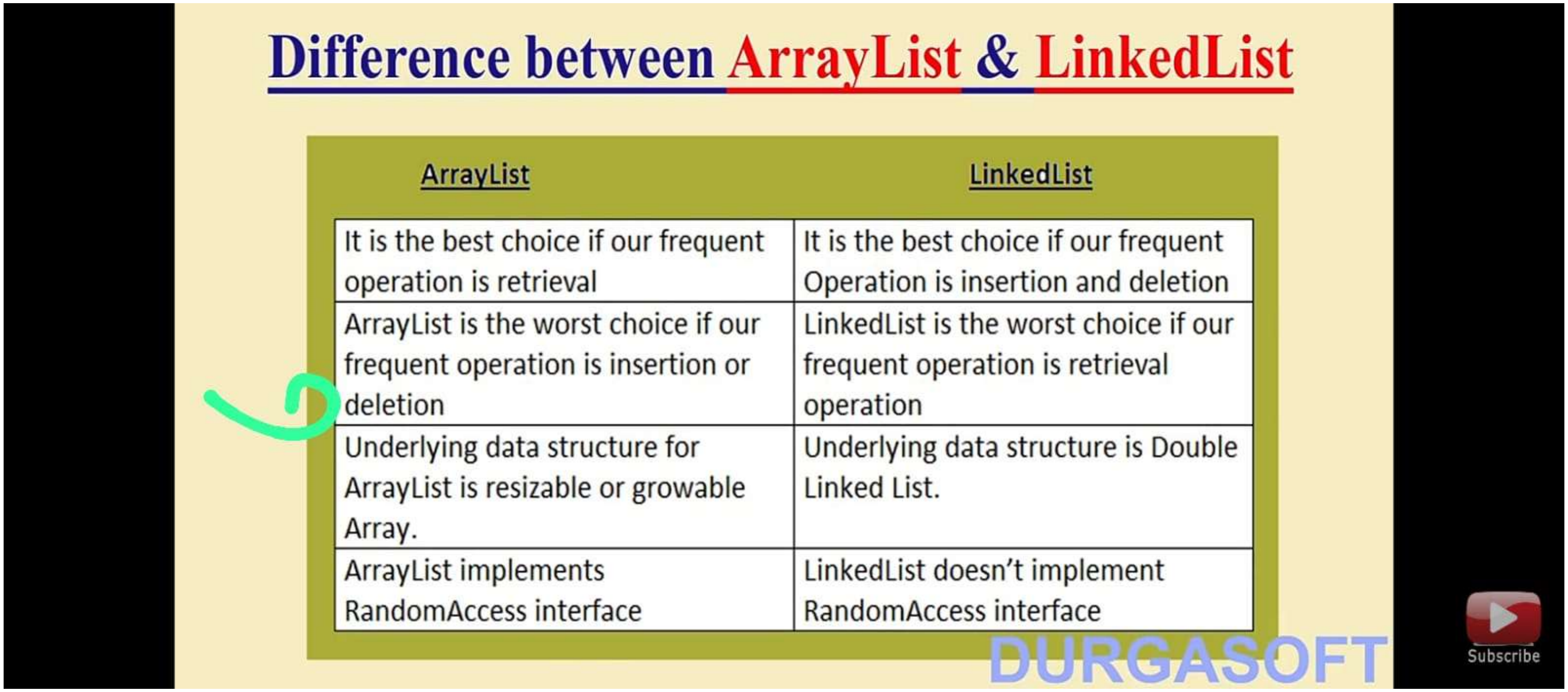
add (Object o)	[from Collection - List(I)]
add (int index, Object o)	[from List]
addElement (Object o)	[from Vector]



Vector specific methods

For removing Objects :

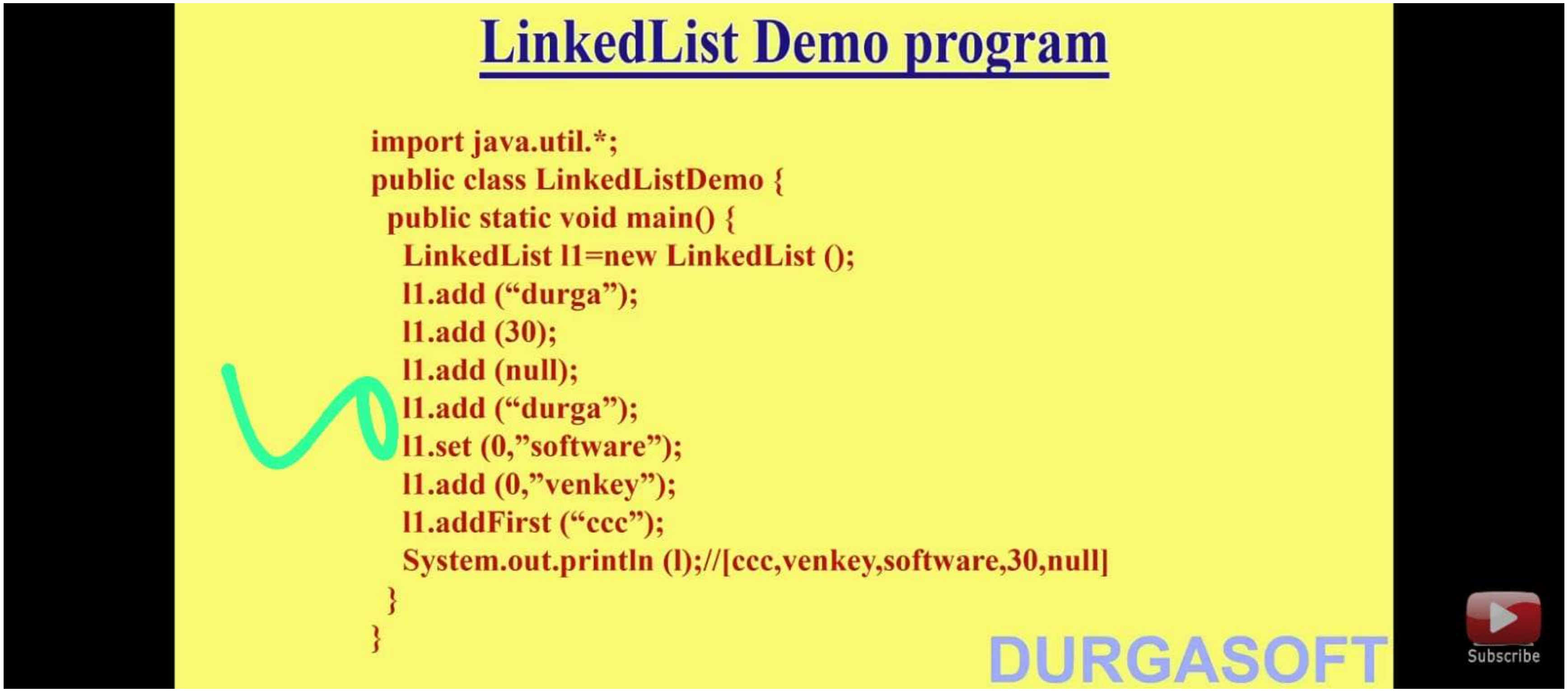
Remove (Object o)	[from Collection]
removeElement (Object o)	[from Vector]
remove (int index)	[from List]
RemoveElementAt (int index)	[from Vector]
clear ()	[from Collection]
removeAllElements ()	[from Vector]



6

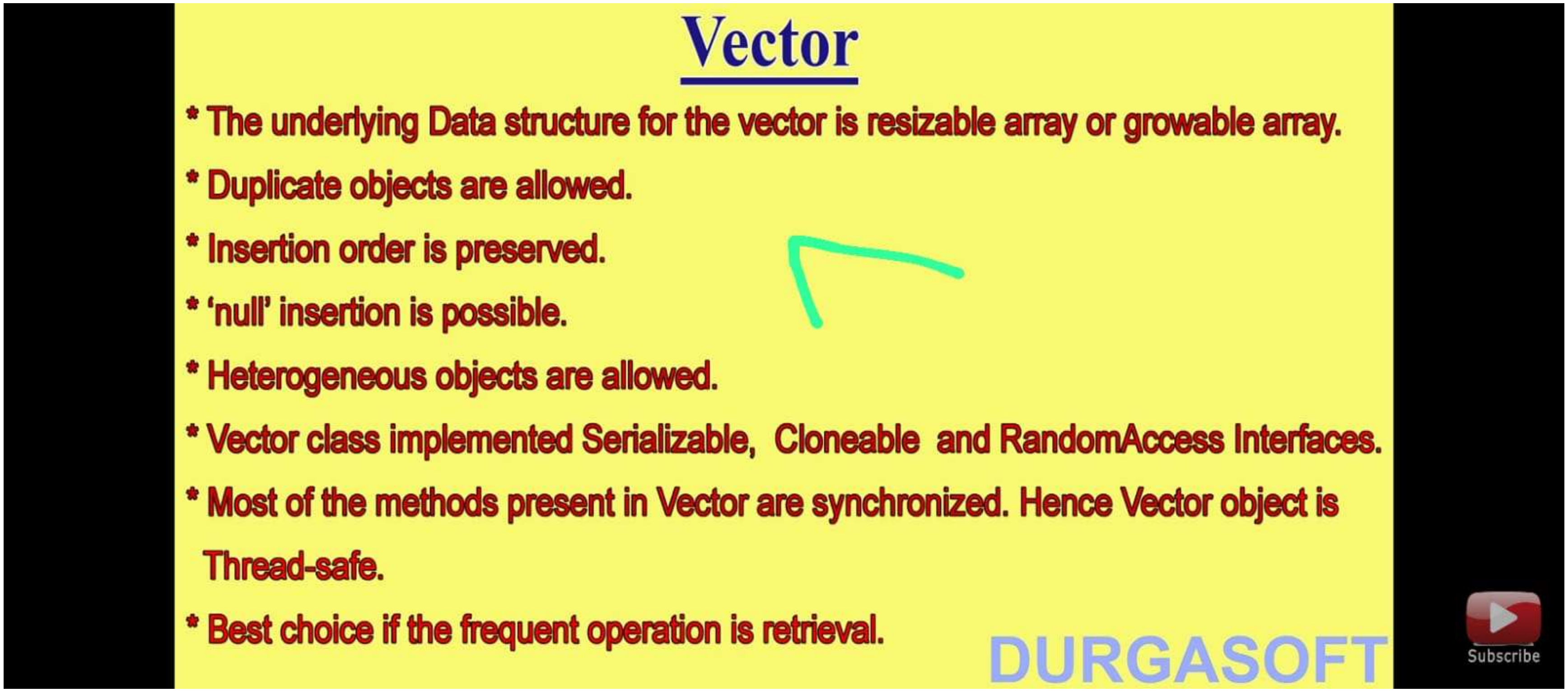
Difference between ArrayList & LinkedList

ArrayList	LinkedList
It is the best choice if our frequent operation is retrieval	It is the best choice if our frequent Operation is insertion and deletion
ArrayList is the worst choice if our frequent operation is insertion or deletion	LinkedList is the worst choice if our frequent operation is retrieval operation
Underlying data structure for ArrayList is resizable or growable Array.	Underlying data structure is Double Linked List.
ArrayList implements RandomAccess interface	LinkedList doesn't implement RandomAccess interface



LinkedList Demo program

```
import java.util.*;
public class LinkedListDemo {
  public static void main() {
    LinkedList l1=new LinkedList ();
    l1.add (“durga”);
    l1.add (30);
    l1.add (null);
    l1.add (“durga”);
    l1.set (0,”software”);
    l1.add (0,”venkey”);
    l1.addFirst (“ccc”);
    System.out.println (l);//[ccc,venkey,software,30,null]
  }
}
```



Vector

* The underlying Data structure for the vector is resizable array or growable array.

* Duplicate objects are allowed.

* Insertion order is preserved.

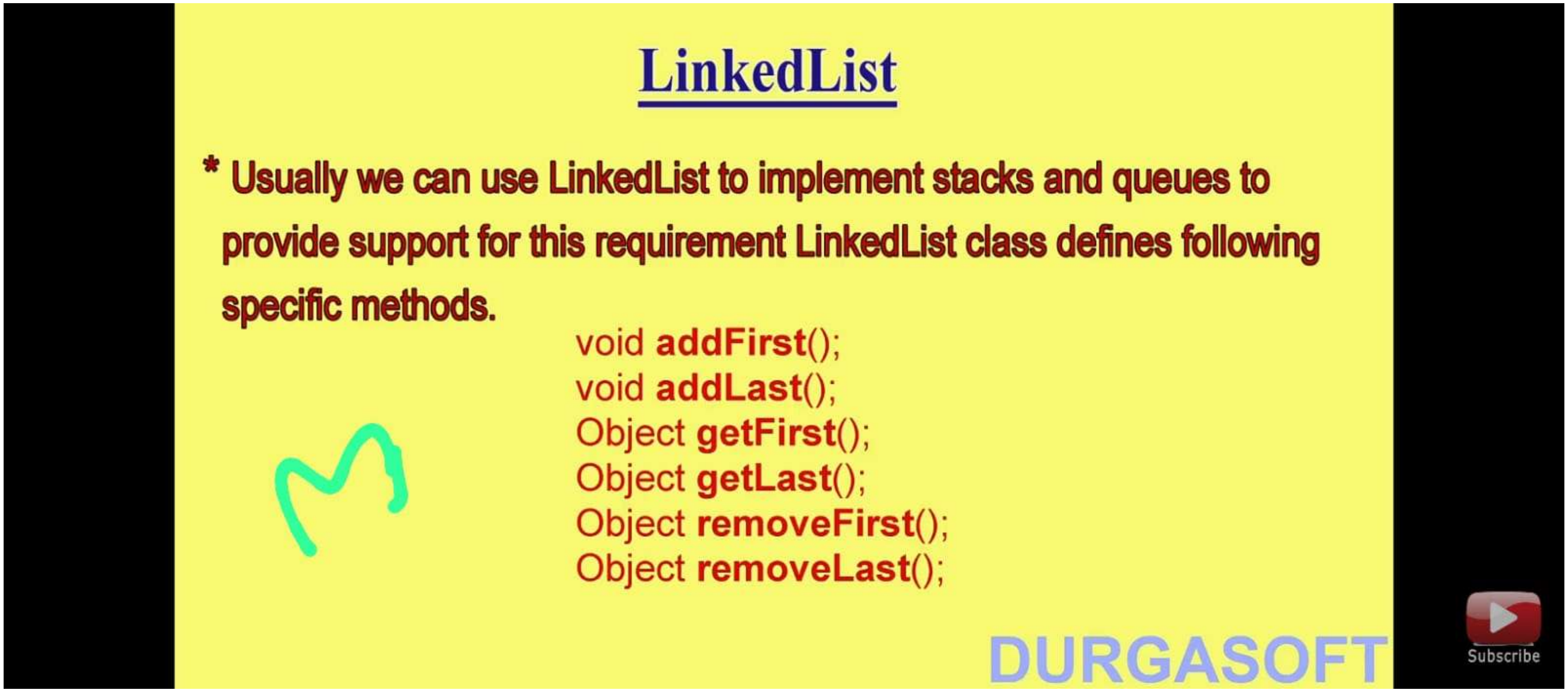
* ‘null’ insertion is possible.

* Heterogeneous objects are allowed.

* Vector class implemented Serializable, Cloneable and RandomAccess Interfaces.

* Most of the methods present in Vector are synchronized. Hence Vector object is Thread-safe.

* Best choice if the frequent operation is retrieval.

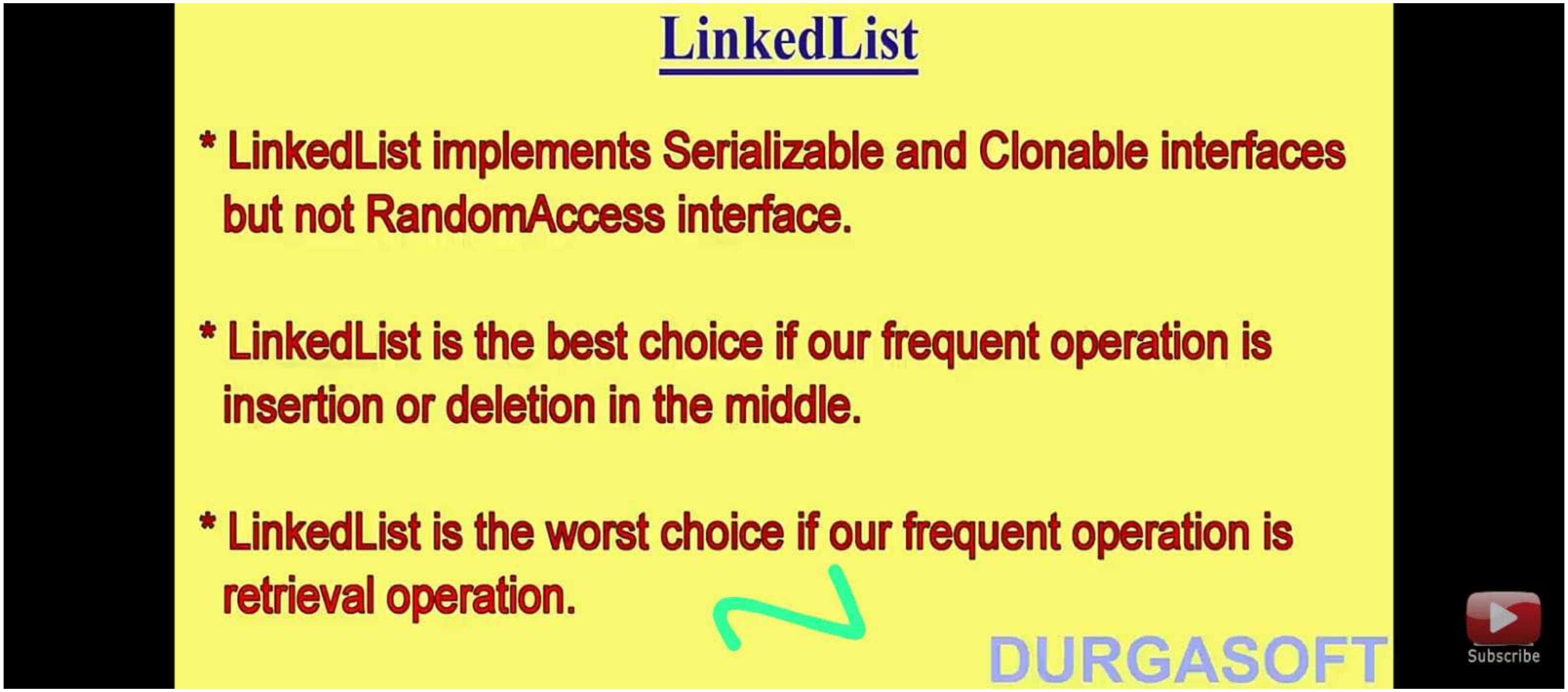


LinkedList

- * Usually we can use LinkedList to implement stacks and queues to provide support for this requirement LinkedList class defines following specific methods.

```
void addFirst();
void addLast();
Object getFirst();
Object getLast();
Object removeFirst();
Object removeLast();
```

3

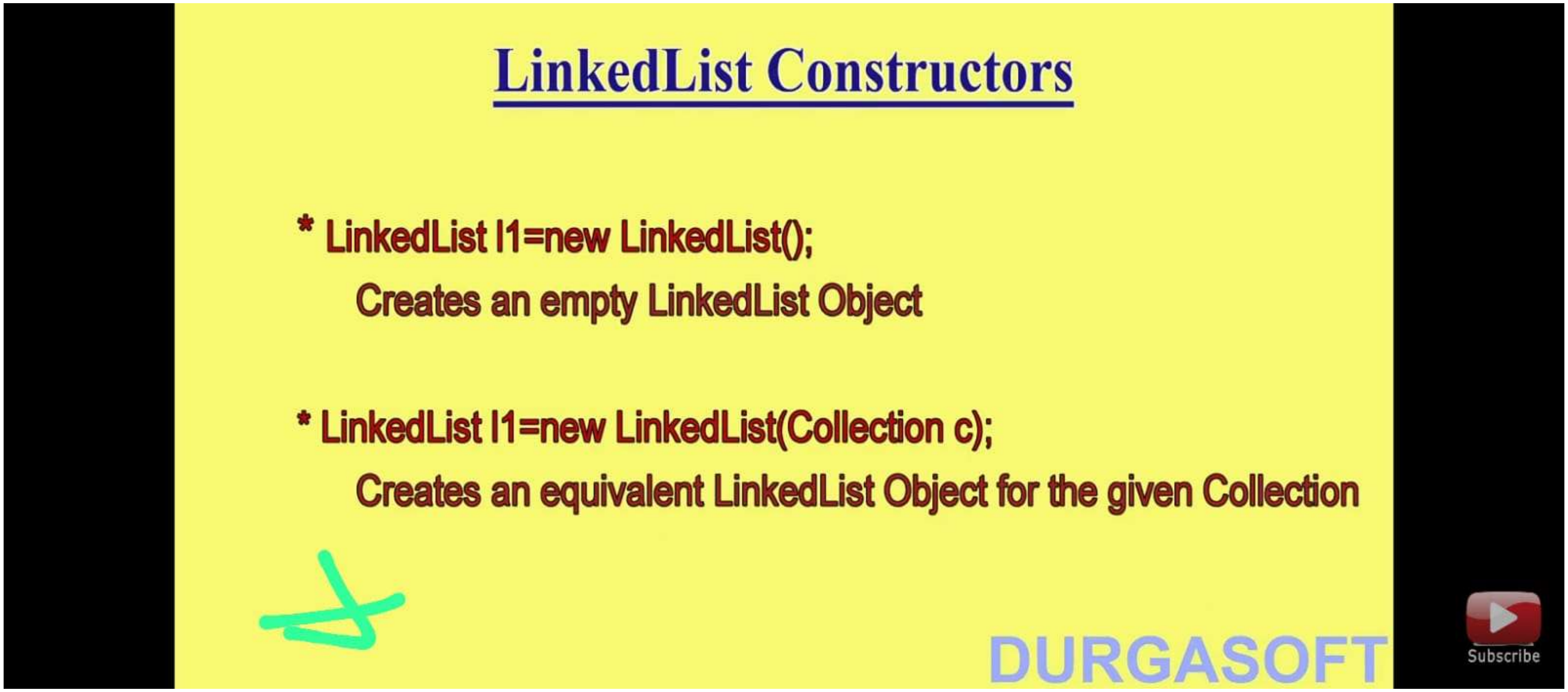


LinkedList

* LinkedList implements Serializable and Clonable interfaces but not RandomAccess interface.

* LinkedList is the best choice if our frequent operation is insertion or deletion in the middle.

* LinkedList is the worst choice if our frequent operation is retrieval operation.



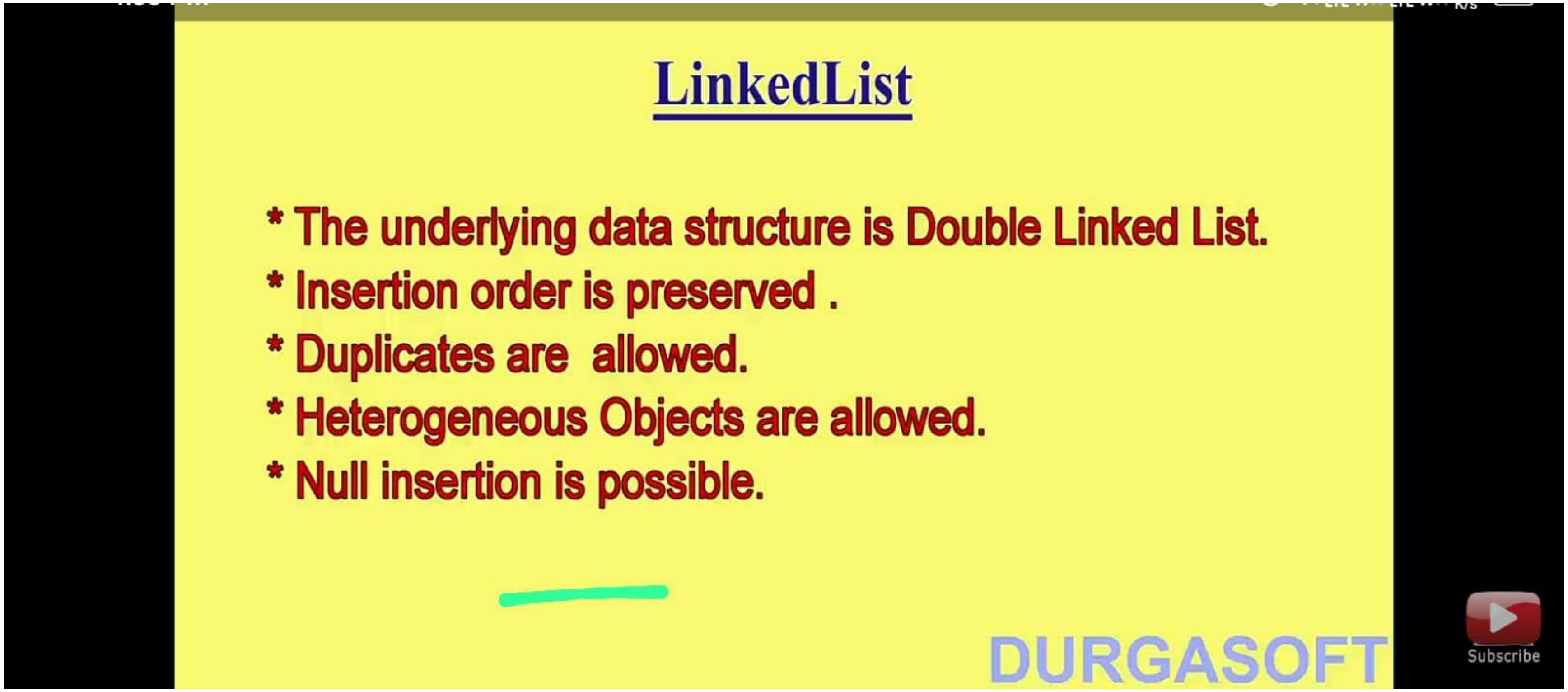
LinkedList Constructors

* LinkedList l1=new LinkedList();

Creates an empty LinkedList Object

* LinkedList l1=new LinkedList(Collection c);

Creates an equivalent LinkedList Object for the given Collection



LinkedList

* The underlying data structure is Double Linked List.
* Insertion order is preserved .
* Duplicates are allowed.
* Heterogeneous Objects are allowed.
* Null insertion is possible.