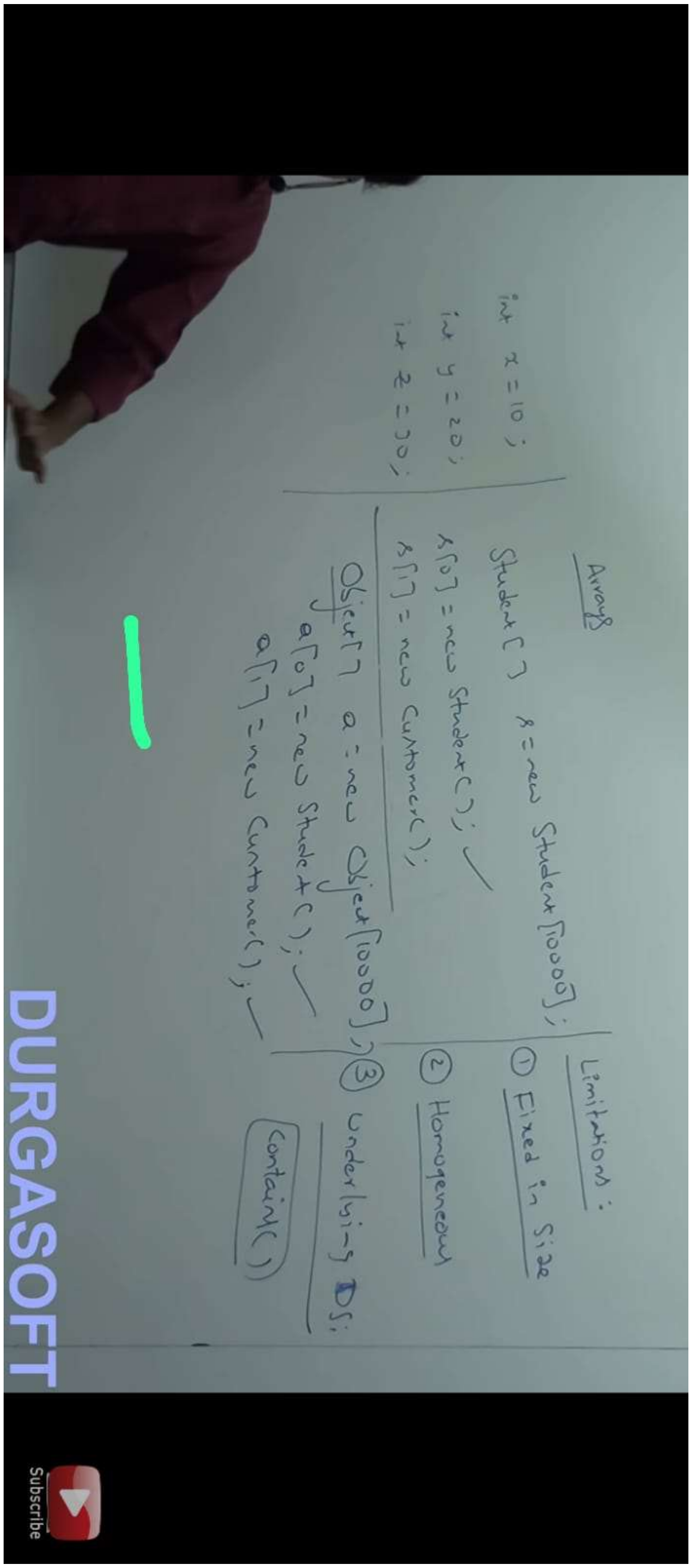
# Collection Framework Introduction



```
int x = 10;
int y = 20;
int z = 30;
```

Arrays

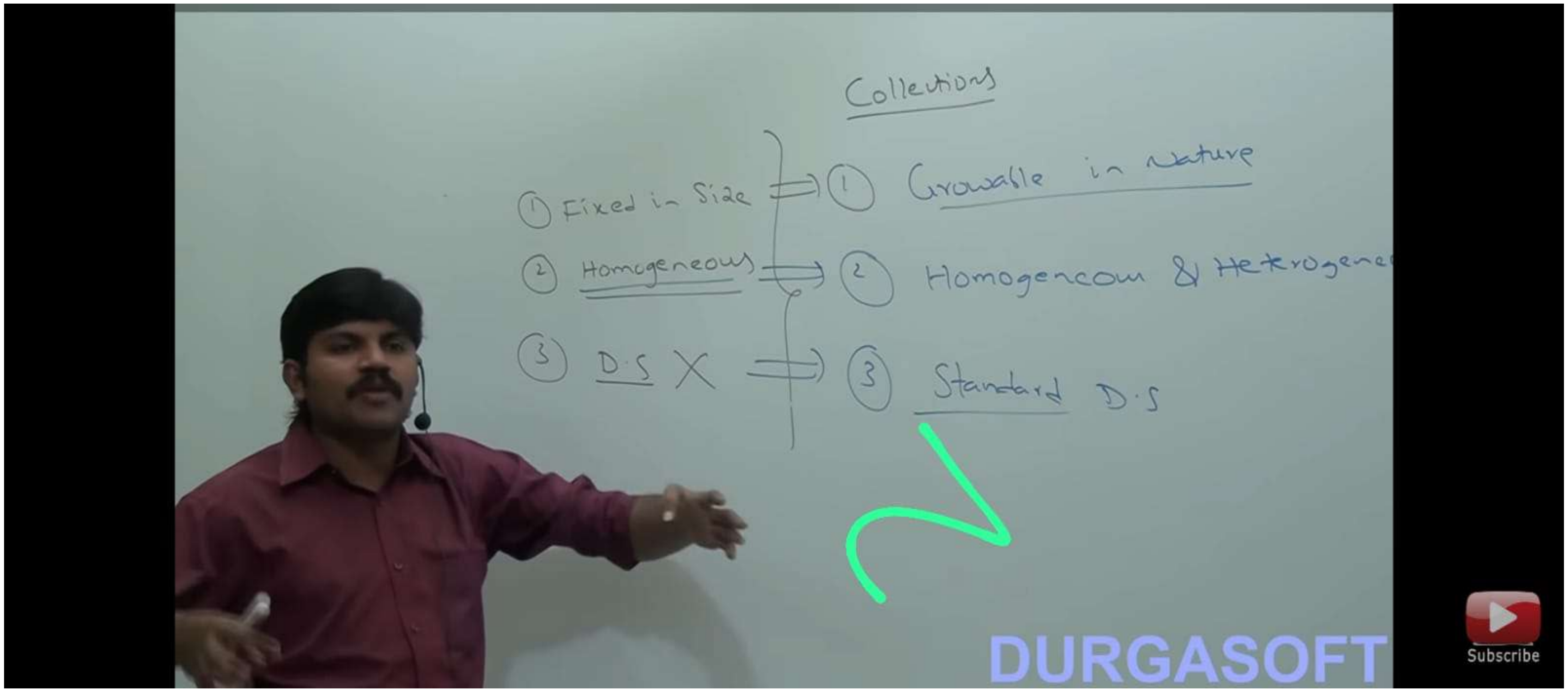
```
Student[] s = new Student[10000];
s[0] = new Student(); ✓
s[1] = new Customer();
```

```
Object[] a = new Object[10000];
a[0] = new Student(); ✓
a[1] = new Customer(); ✓
```

Limitations :

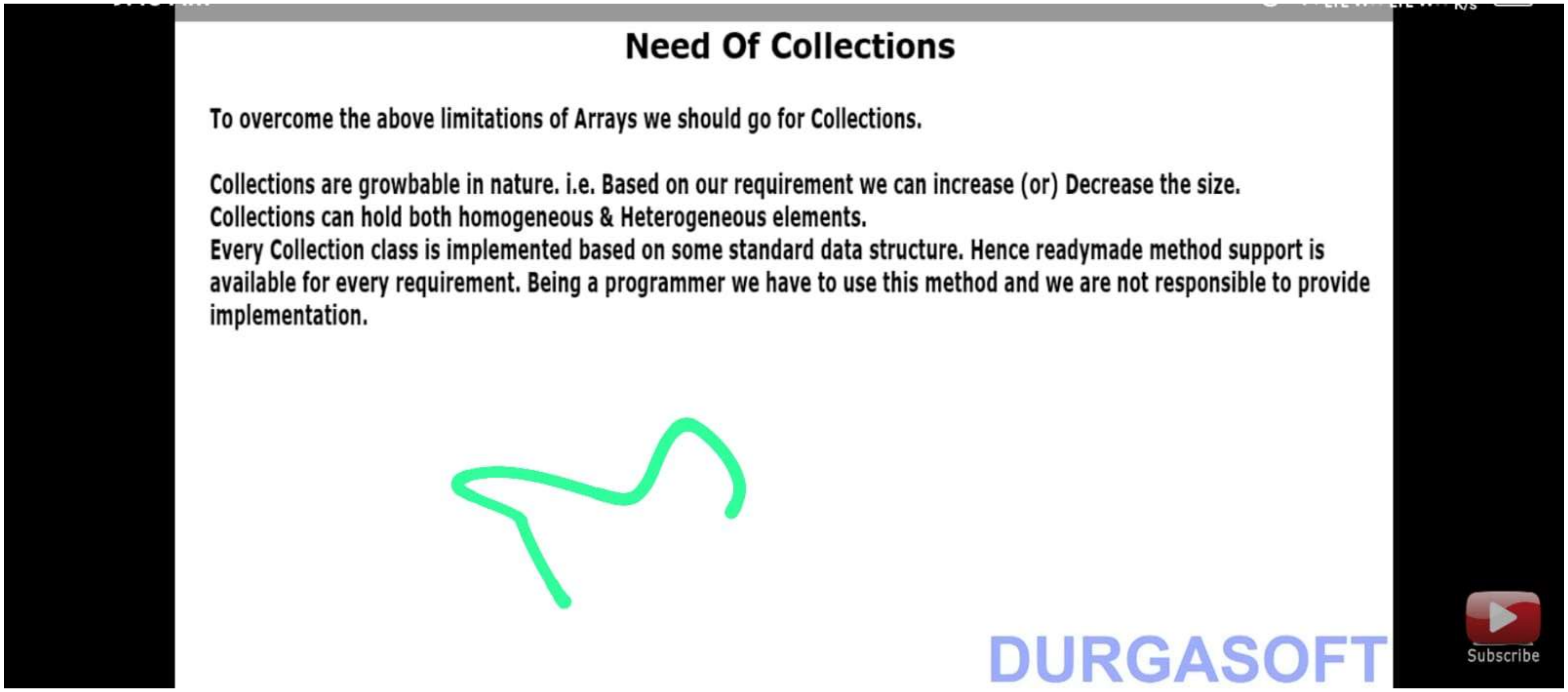
1. Fixed in Size
2. Homogeneous
3. underlying DS;

contains()



Collections

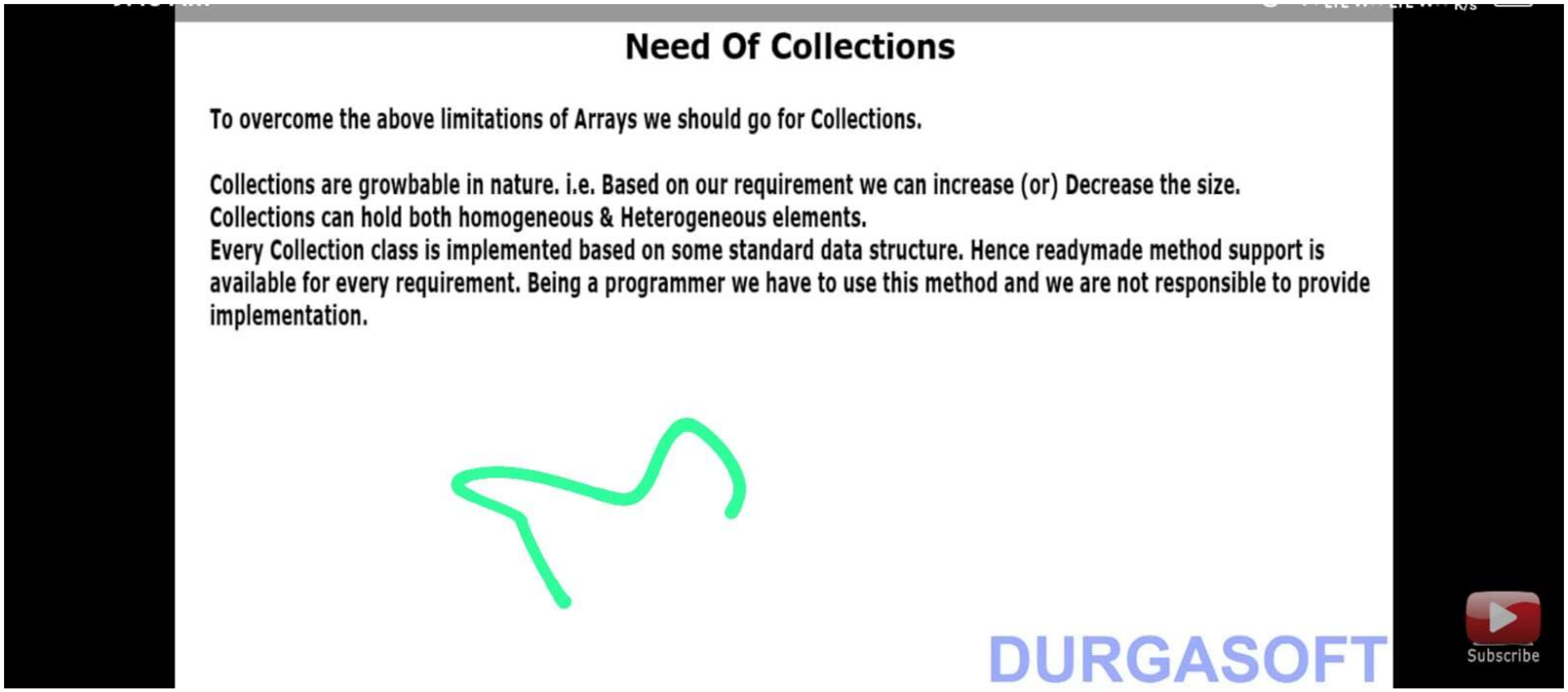
| | |
|---|---|
| (1) Fixed in Size ⇒ | (1) Growable in Nature |
| (2) Homogeneous ⇒ | (2) Homogeneous & Hetrogene |
| (3) D.S X ⇒ | (3) Standard D.S |



## Need Of Collections

To overcome the above limitations of Arrays we should go for Collections.

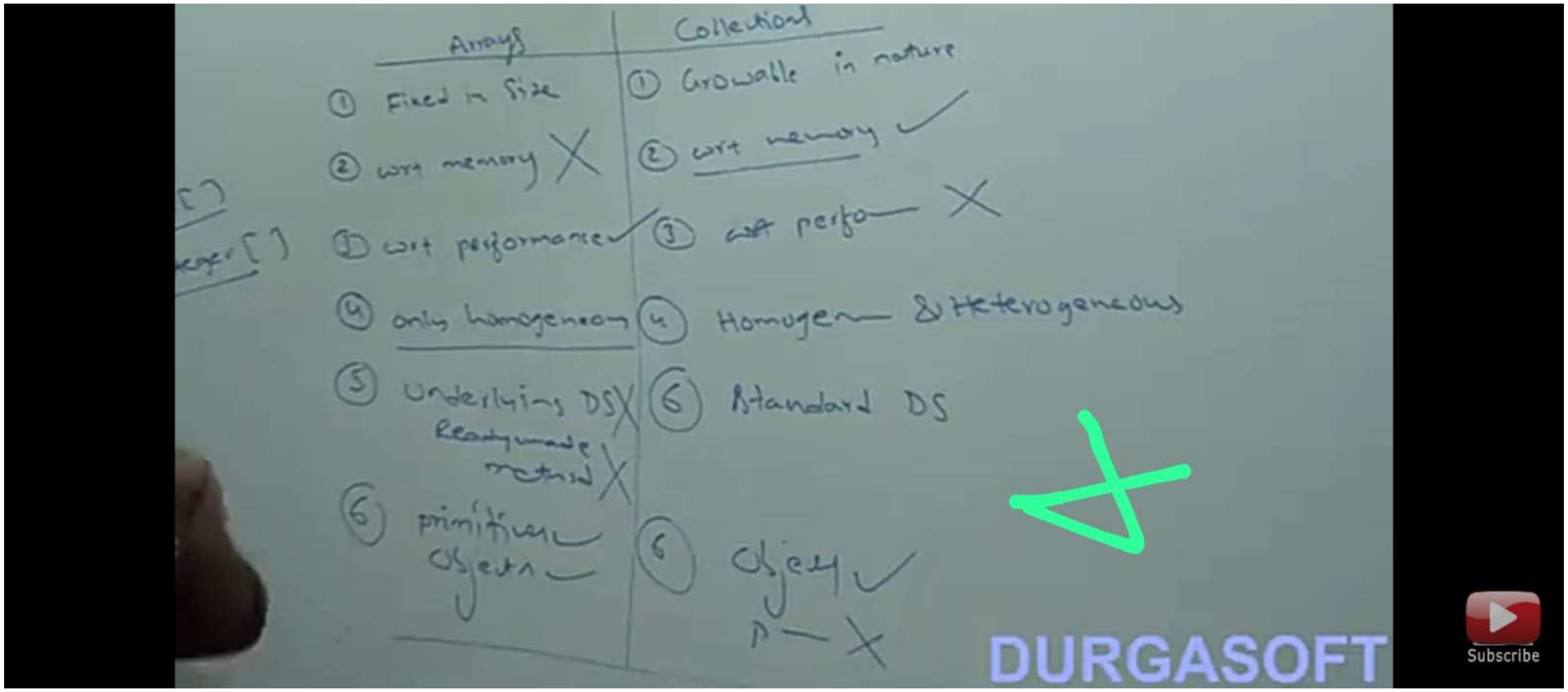
Collections are growbable in nature. i.e. Based on our requirement we can increase (or) Decrease the size.
Collections can hold both homogeneous & Heterogeneous elements.
Every Collection class is implemented based on some standard data structure. Hence readymade method support is available for every requirement. Being a programmer we have to use this method and we are not responsible to provide implementation.



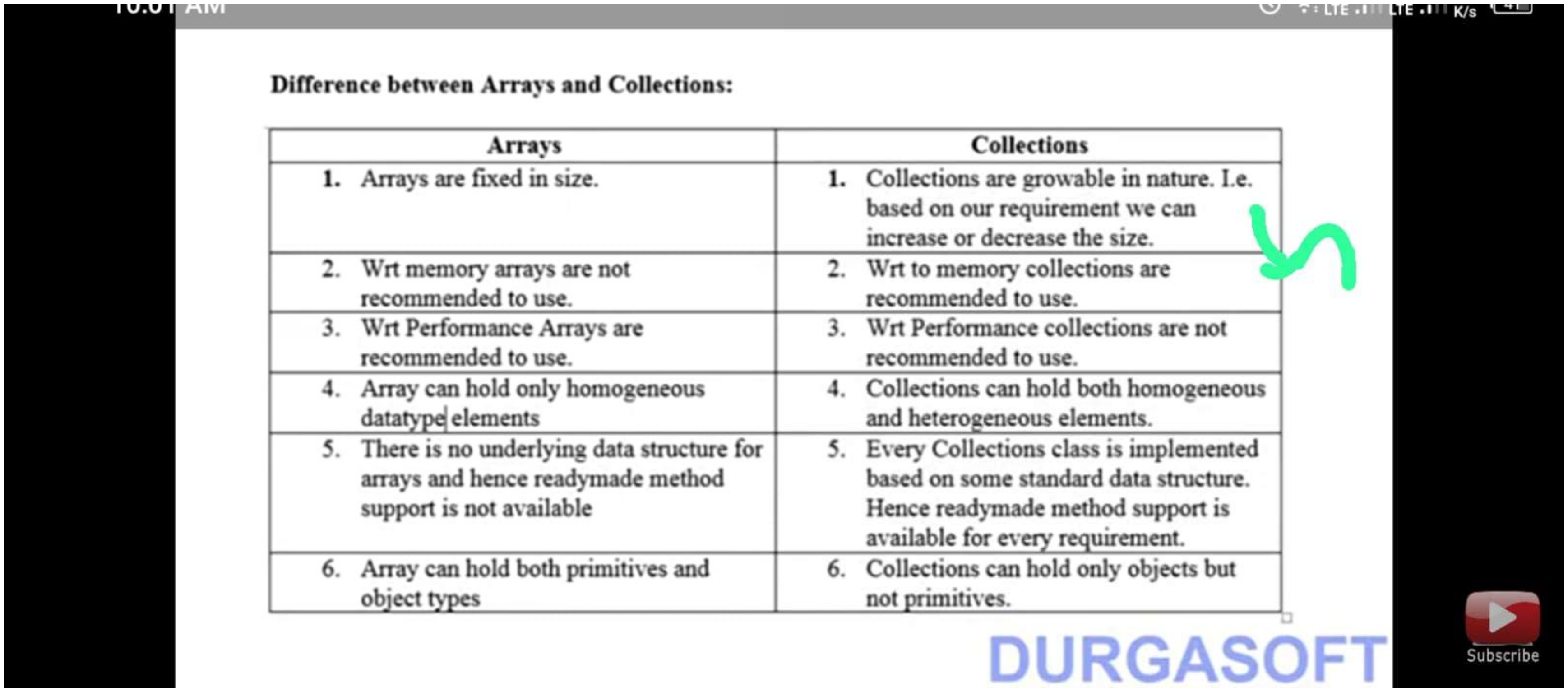
## Need Of Collections

To overcome the above limitations of Arrays we should go for Collections.

Collections are growbable in nature. i.e. Based on our requirement we can increase (or) Decrease the size.
Collections can hold both homogeneous & Heterogeneous elements.
Every Collection class is implemented based on some standard data structure. Hence readymade method support is available for every requirement. Being a programmer we have to use this method and we are not responsible to provide implementation.

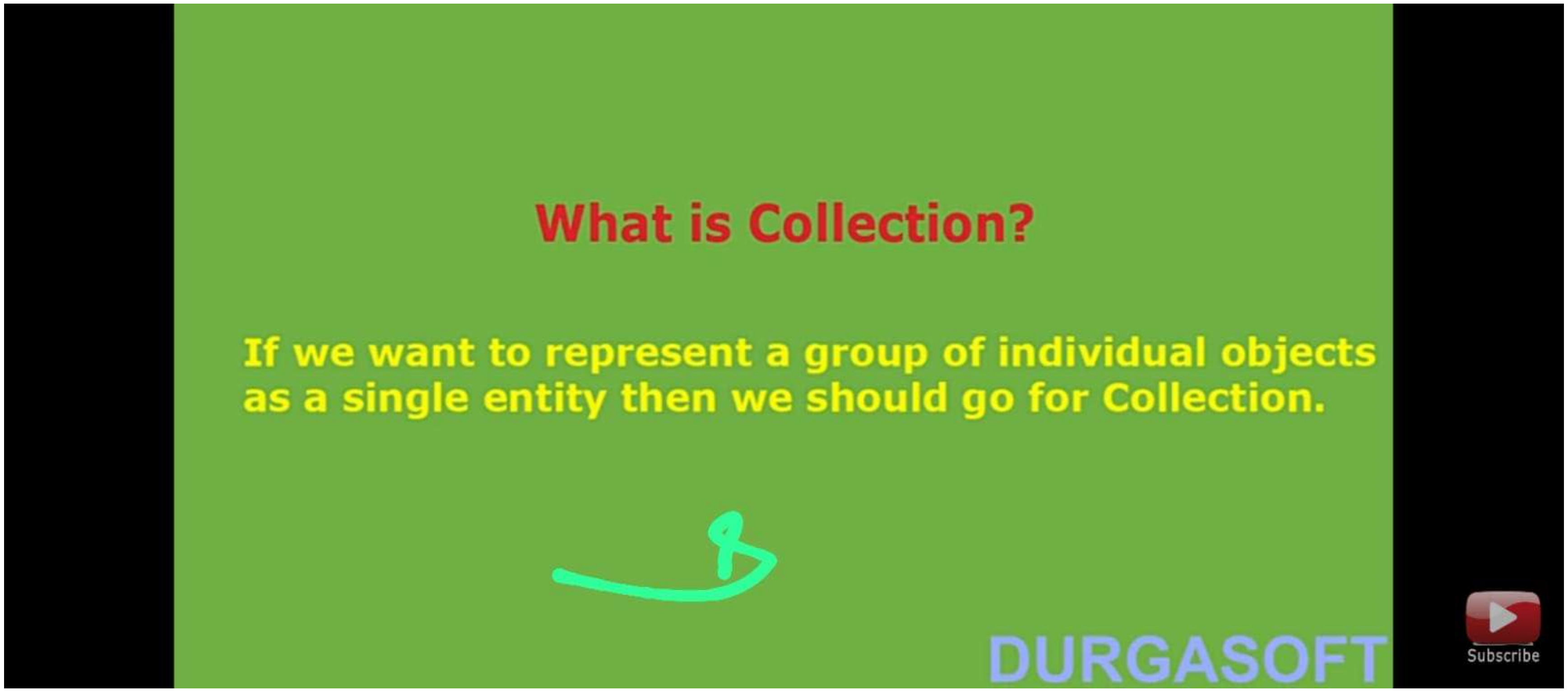


| Arrays | Collections |
|---|---|
| ① Fixed in size | ① Growable in nature |
| ② wrt memory ✗ | ② wrt memory ✓ |
| ③ wrt performance ✓ | ③ wrt perfo— ✗ |
| ④ only homogeneous | ④ Homogen— & Heterogeneous |
| ⑤ Underlying DS ✗ Readymade method ✗ | ⑤ Standard DS |
| ⑥ primitives ✓ objects ✓ | ⑥ objects ✓ p— ✗ |



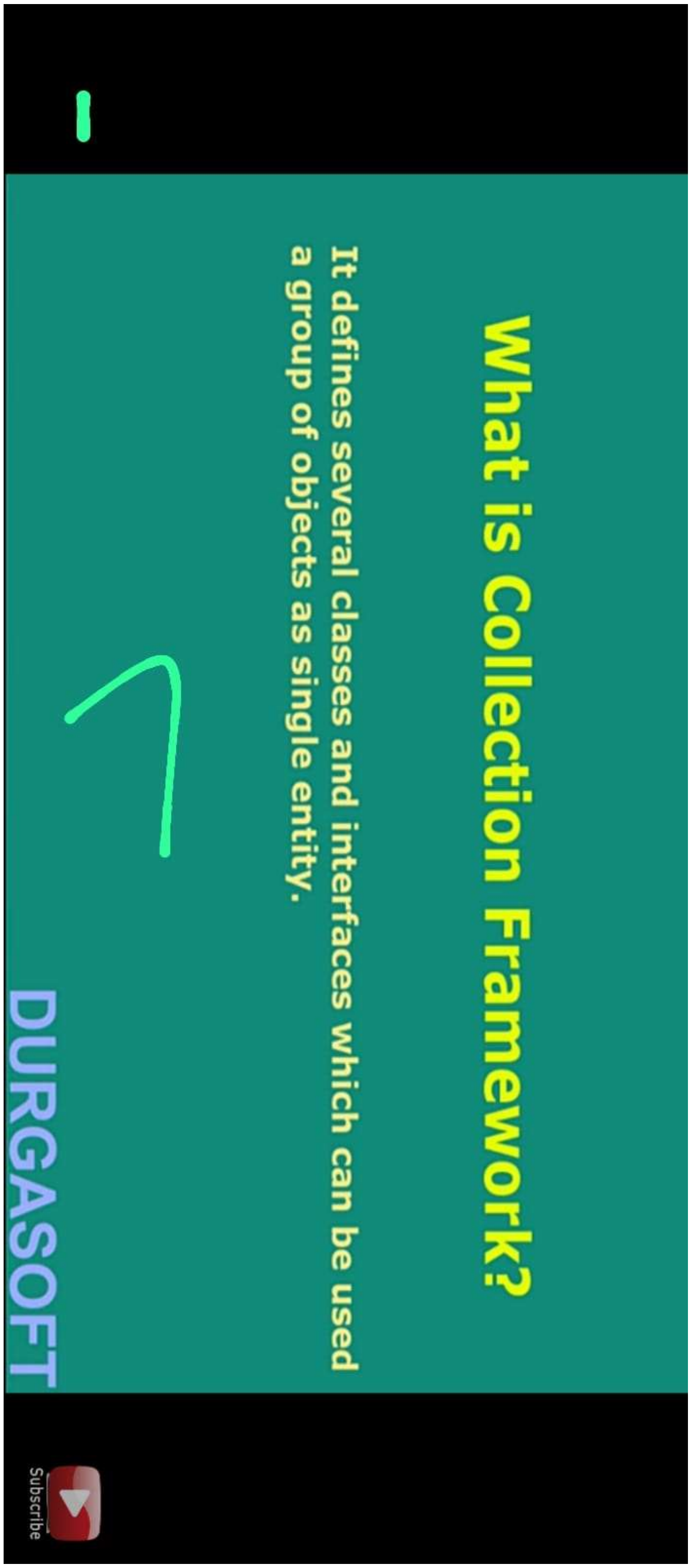
**Difference between Arrays and Collections:**

| Arrays | Collections |
|---|---|
| 1. Arrays are fixed in size. | 1. Collections are growable in nature. I.e. based on our requirement we can increase or decrease the size. |
| 2. Wrt memory arrays are not recommended to use. | 2. Wrt to memory collections are recommended to use. |
| 3. Wrt Performance Arrays are recommended to use. | 3. Wrt Performance collections are not recommended to use. |
| 4. Array can hold only homogeneous datatype elements | 4. Collections can hold both homogeneous and heterogeneous elements. |
| 5. There is no underlying data structure for arrays and hence readymade method support is not available | 5. Every Collections class is implemented based on some standard data structure. Hence readymade method support is available for every requirement. |
| 6. Array can hold both primitives and object types | 6. Collections can hold only objects but not primitives. |



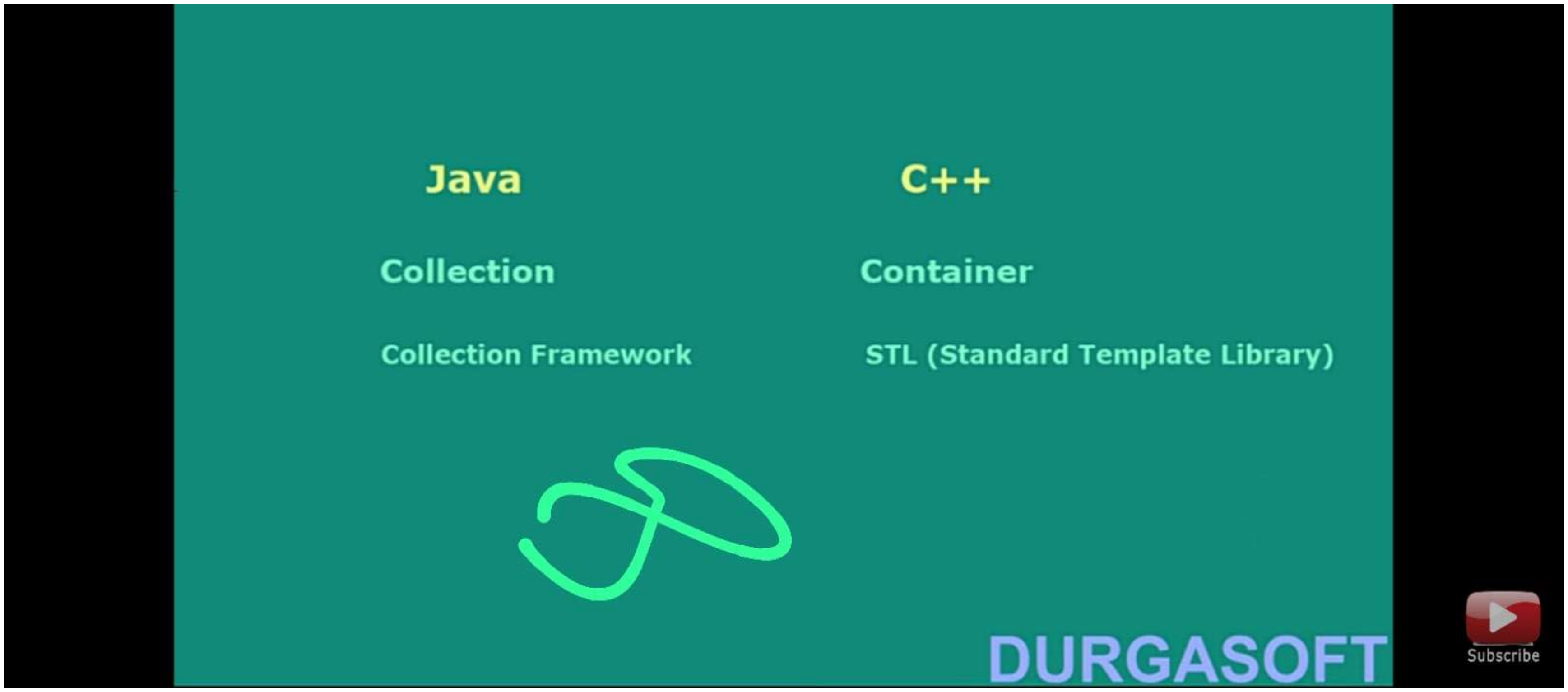
## What is Collection?

If we want to represent a group of individual objects as a single entity then we should go for Collection.

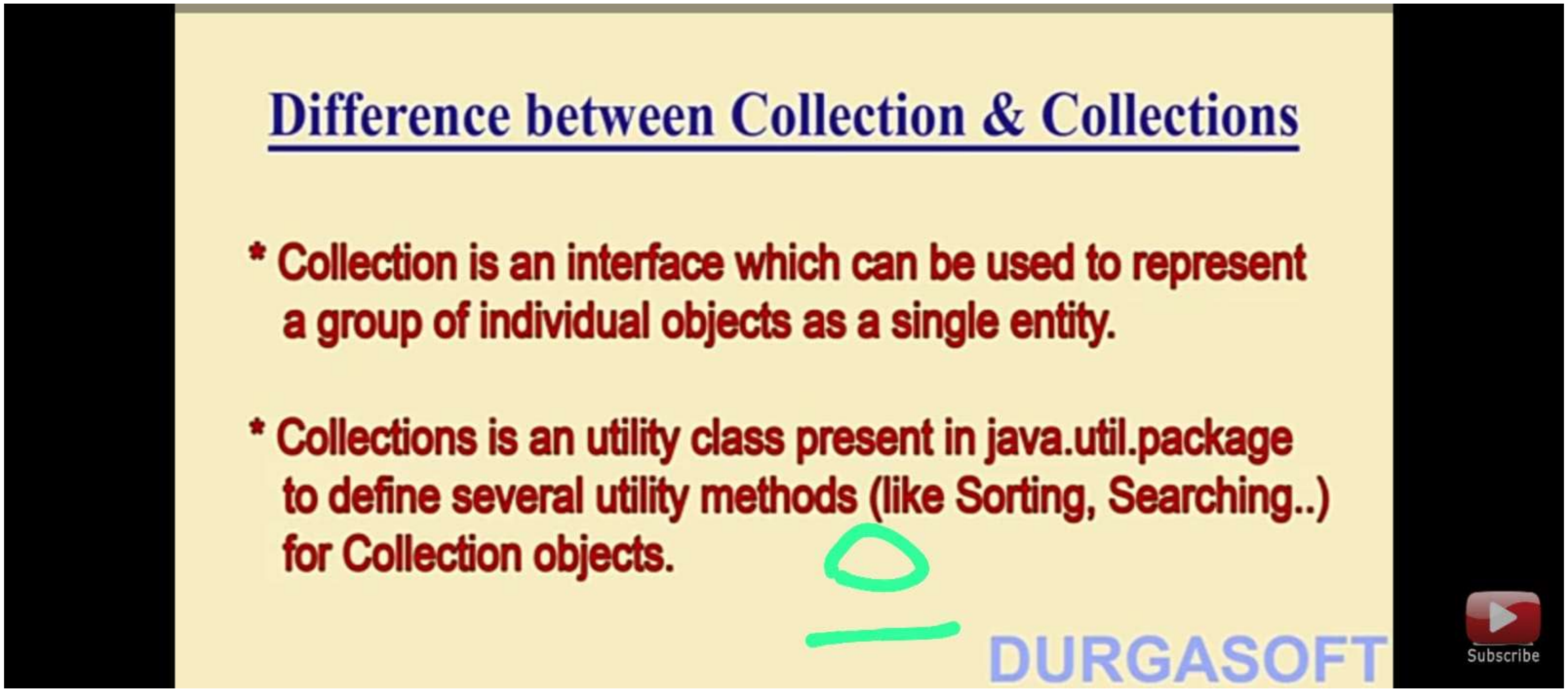


## What is Collection Framework?

It defines several classes and interfaces which can be used a group of objects as single entity.



| Java | C++ |
|---|---|
| Collection | Container |
| Collection Framework | STL (Standard Template Library) |

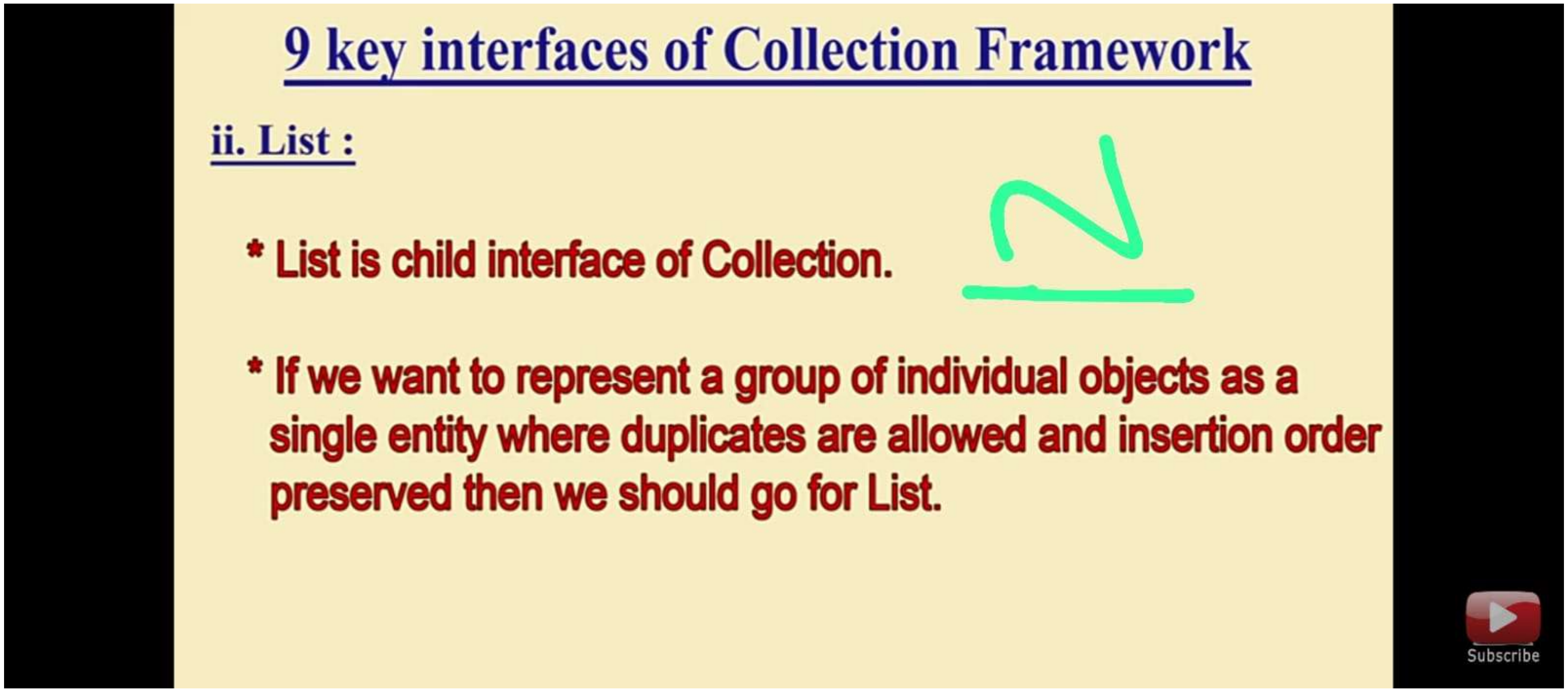


## Difference between Collection & Collections

* Collection is an interface which can be used to represent a group of individual objects as a single entity.

* Collections is an utility class present in java.util.package to define several utility methods (like Sorting, Searching..) for Collection objects.

10



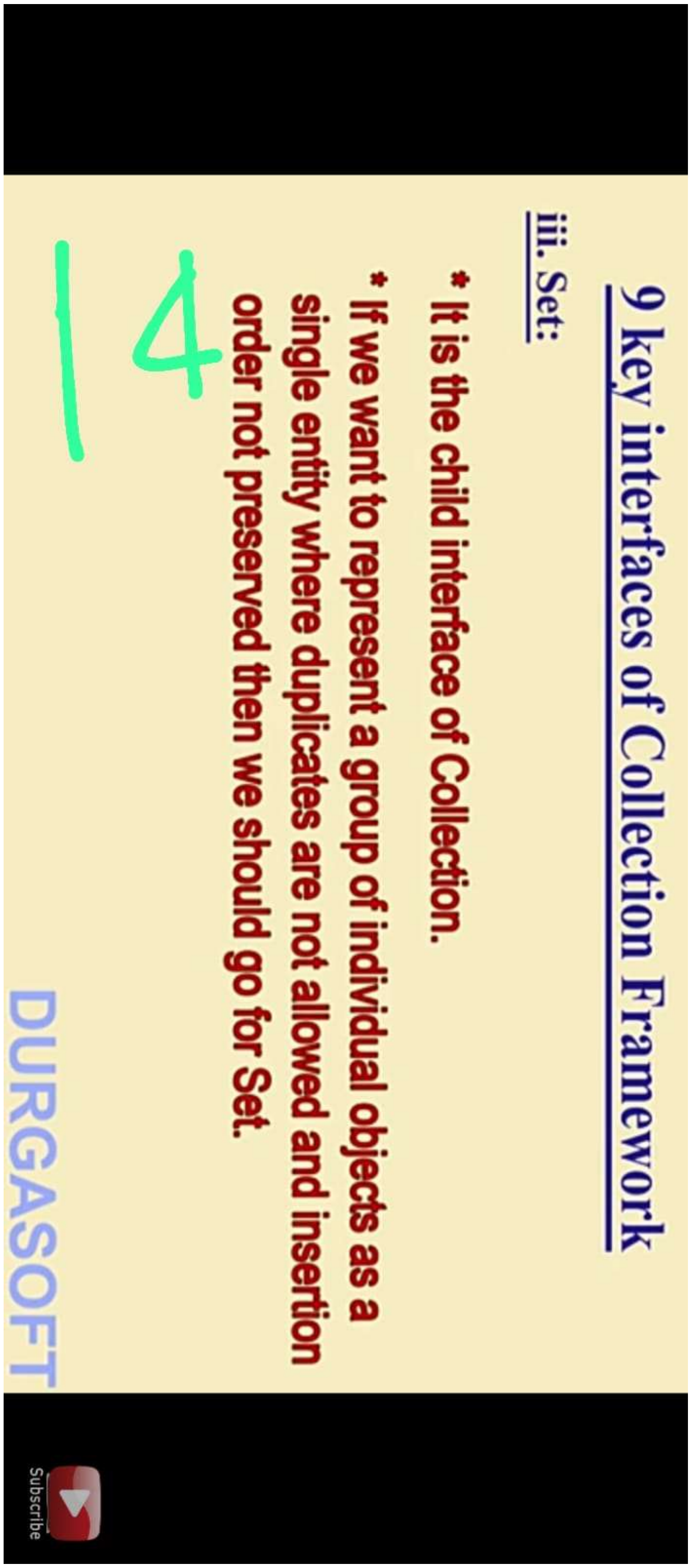
## 9 key interfaces of Collection Framework

**ii. List :**

* List is child interface of Collection.

12

* If we want to represent a group of individual objects as a single entity where duplicates are allowed and insertion order preserved then we should go for List.



## 9 key interfaces of Collection Framework

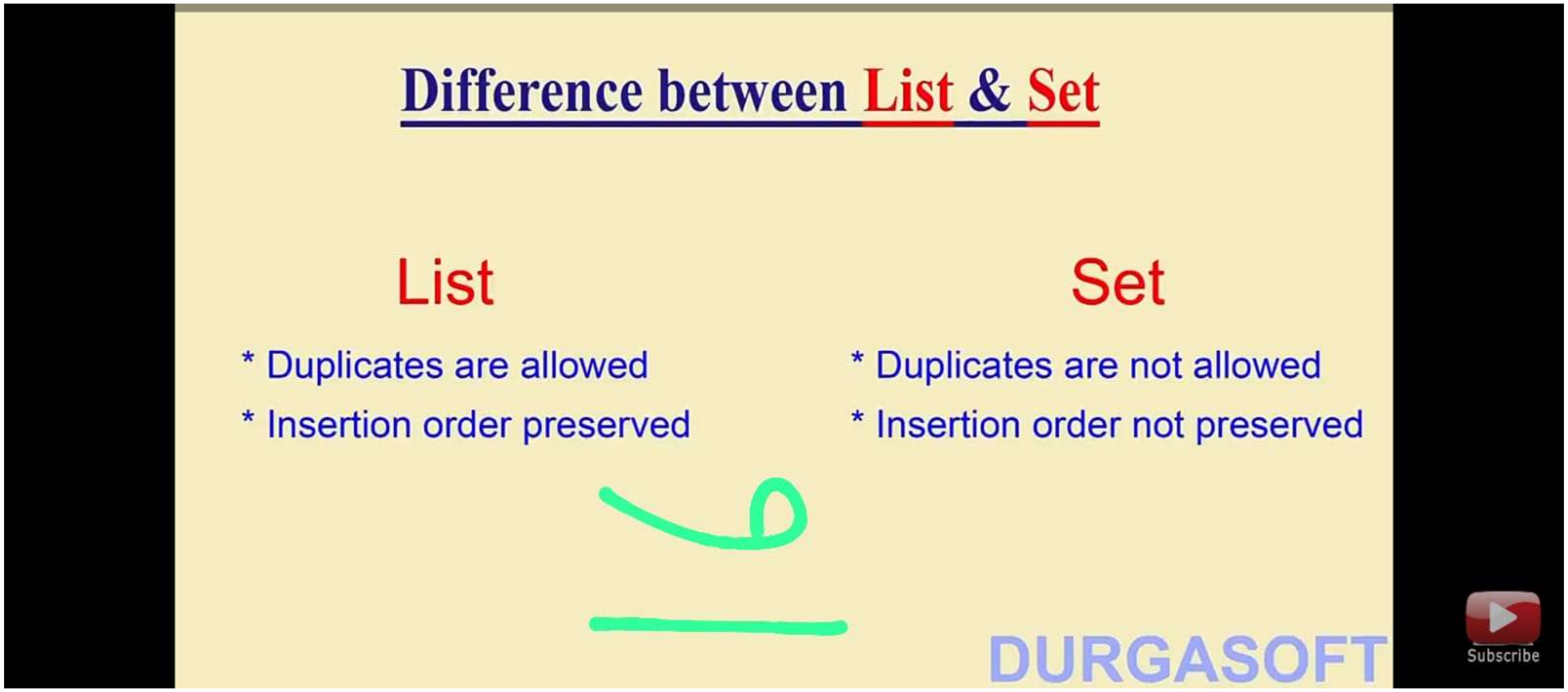
iii. Set:

14

* It is the child interface of Collection.
* If we want to represent a group of individual objects as a single entity where duplicates are not allowed and insertion order not preserved then we should go for Set.

DURGASOFT

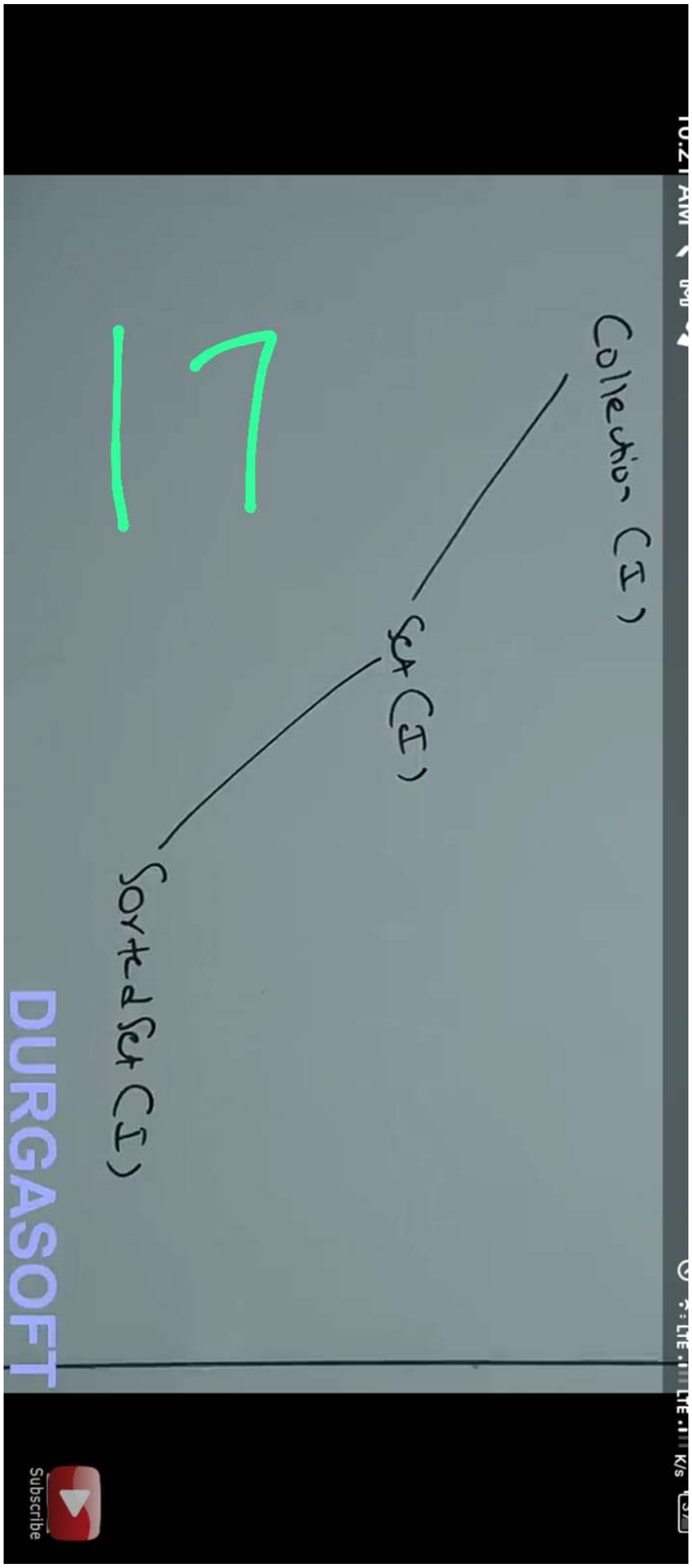
Subscribe

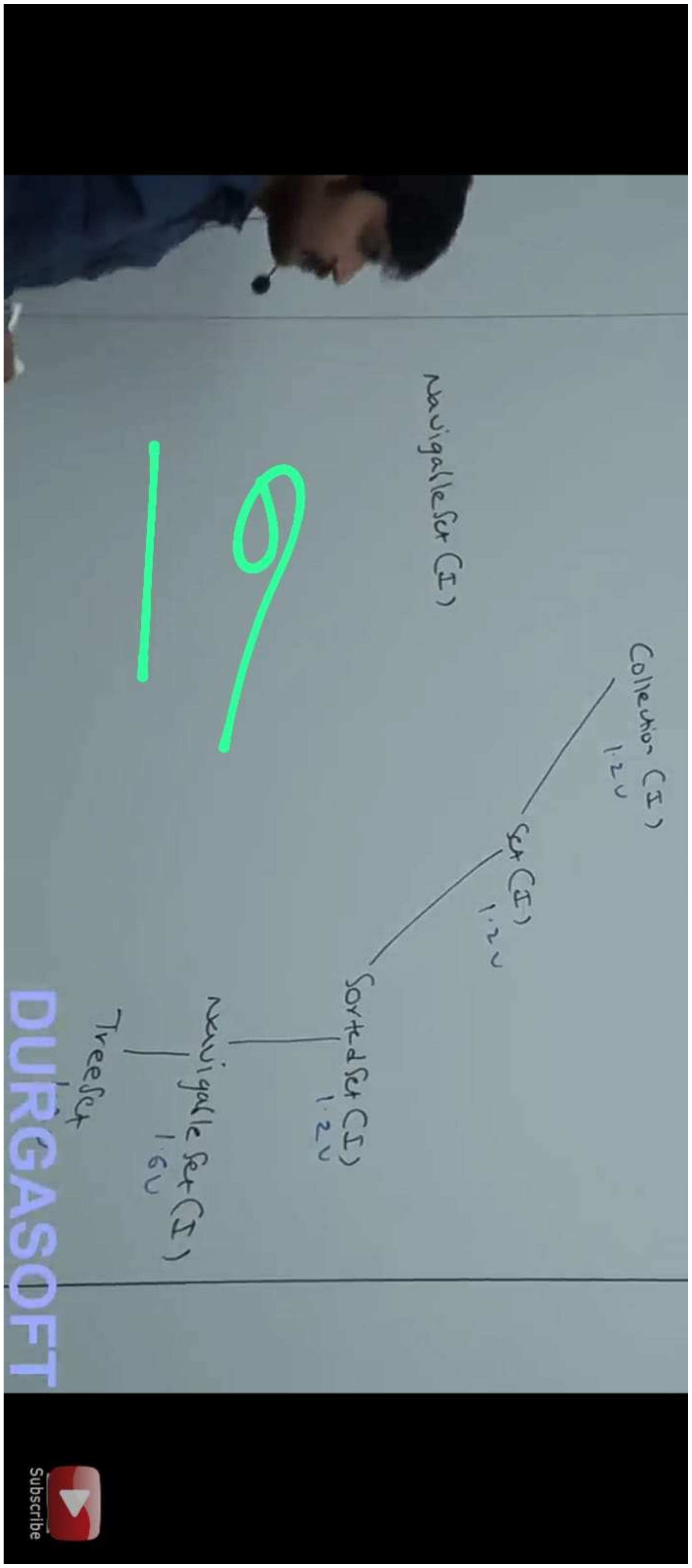


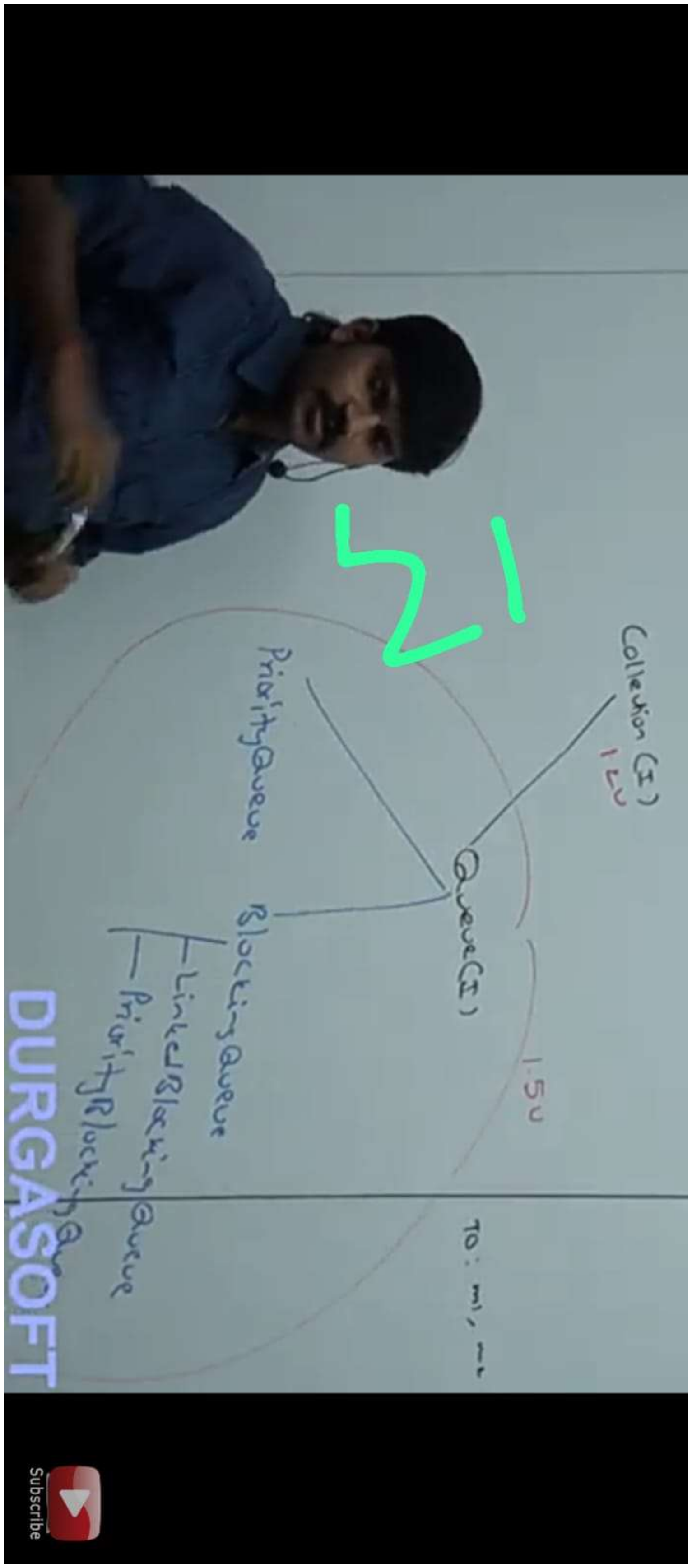
## Difference between List & Set

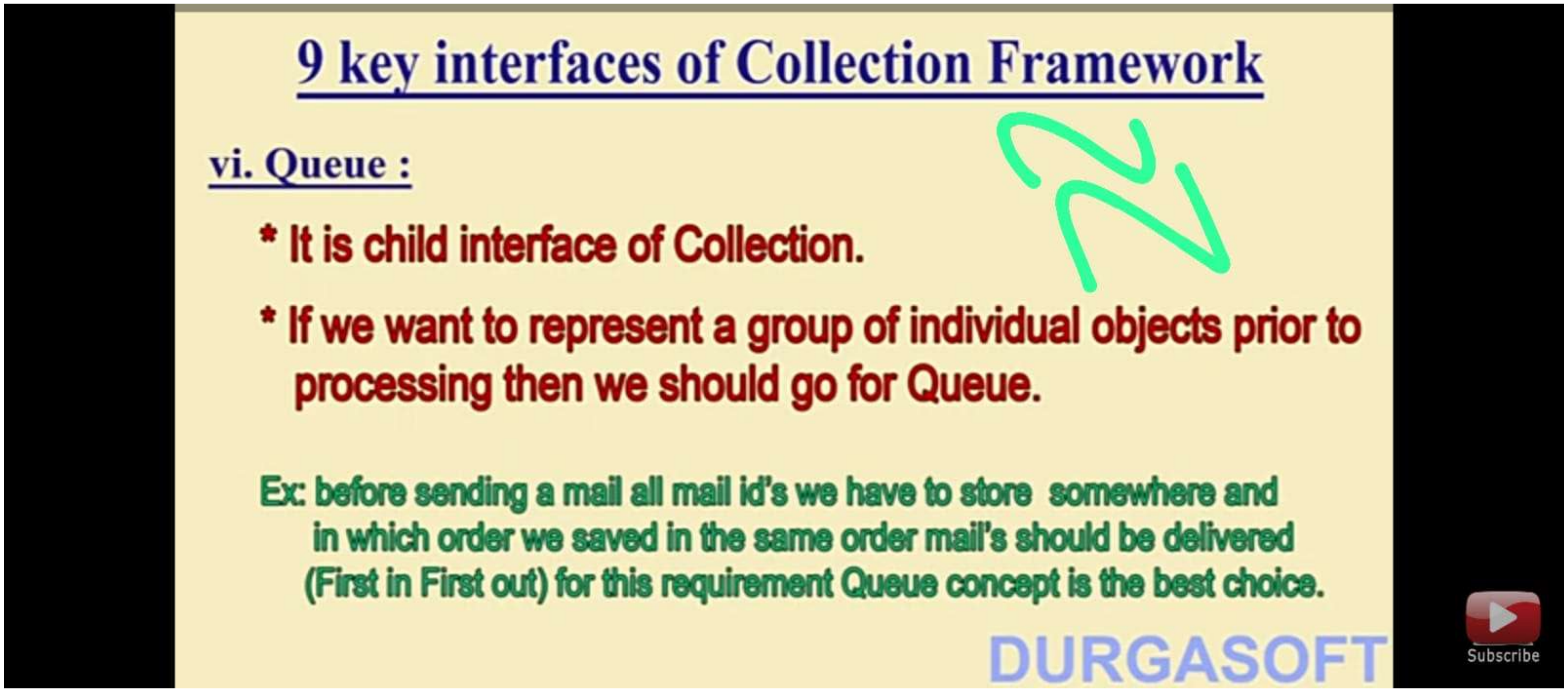
| List | Set |
|---|---|
| * Duplicates are allowed | * Duplicates are not allowed |
| * Insertion order preserved | * Insertion order not preserved |

16


17
Collection (I)
Set (I)
SortedSet (I)
DURGASOFT


Collection (I)
1.2v
Set (I)
1.2v
SortedSet (I)
1.2v
NavigableSet (I)
1.6v
TreeSet
NavigableSet (I)
19
DURGASOFT
Subscribe


21
Collection (I)
1.2V
Queue(I)
1.5V
PriorityQueue
BlockingQueue
LinkedBlockingQueue
PriorityBlockingQueue
TO: m1, m2
DURGASOFT
Subscribe



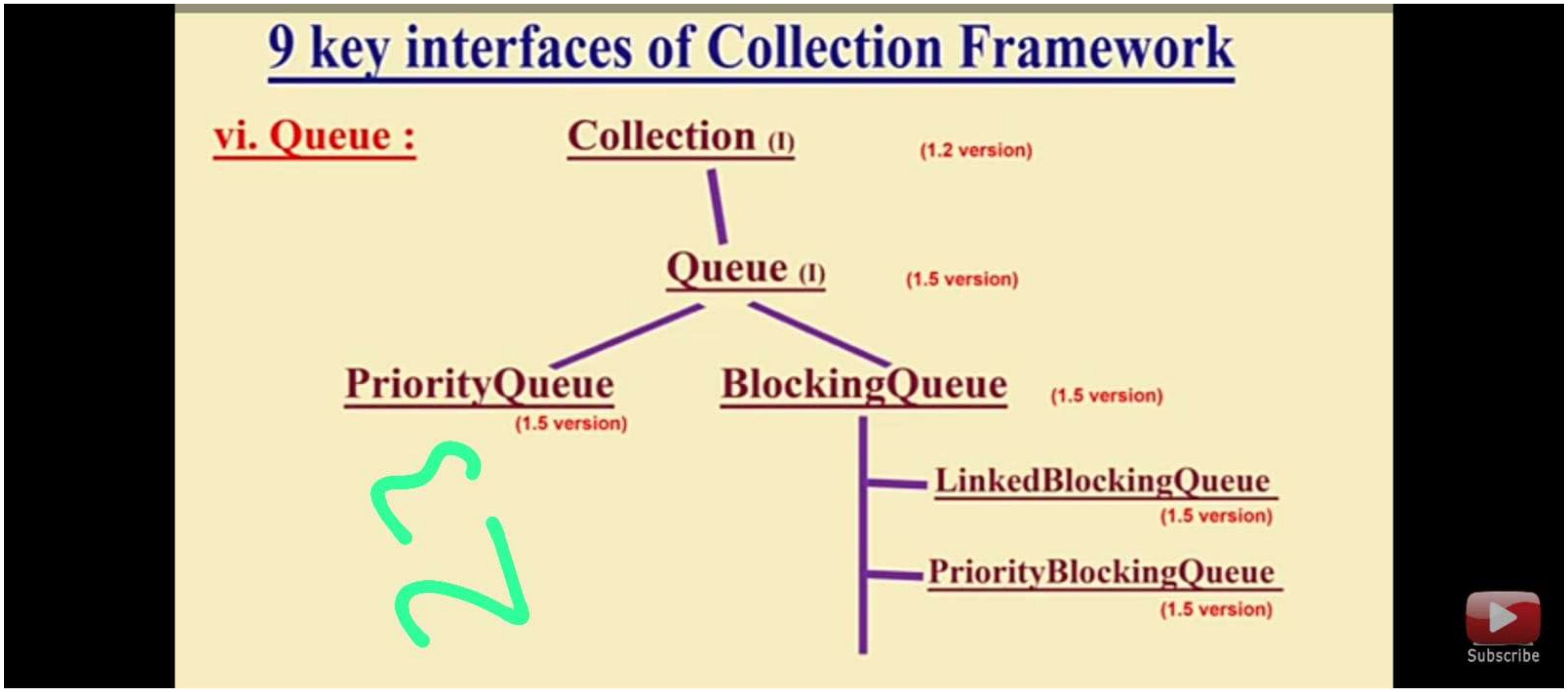
## 9 key interfaces of Collection Framework

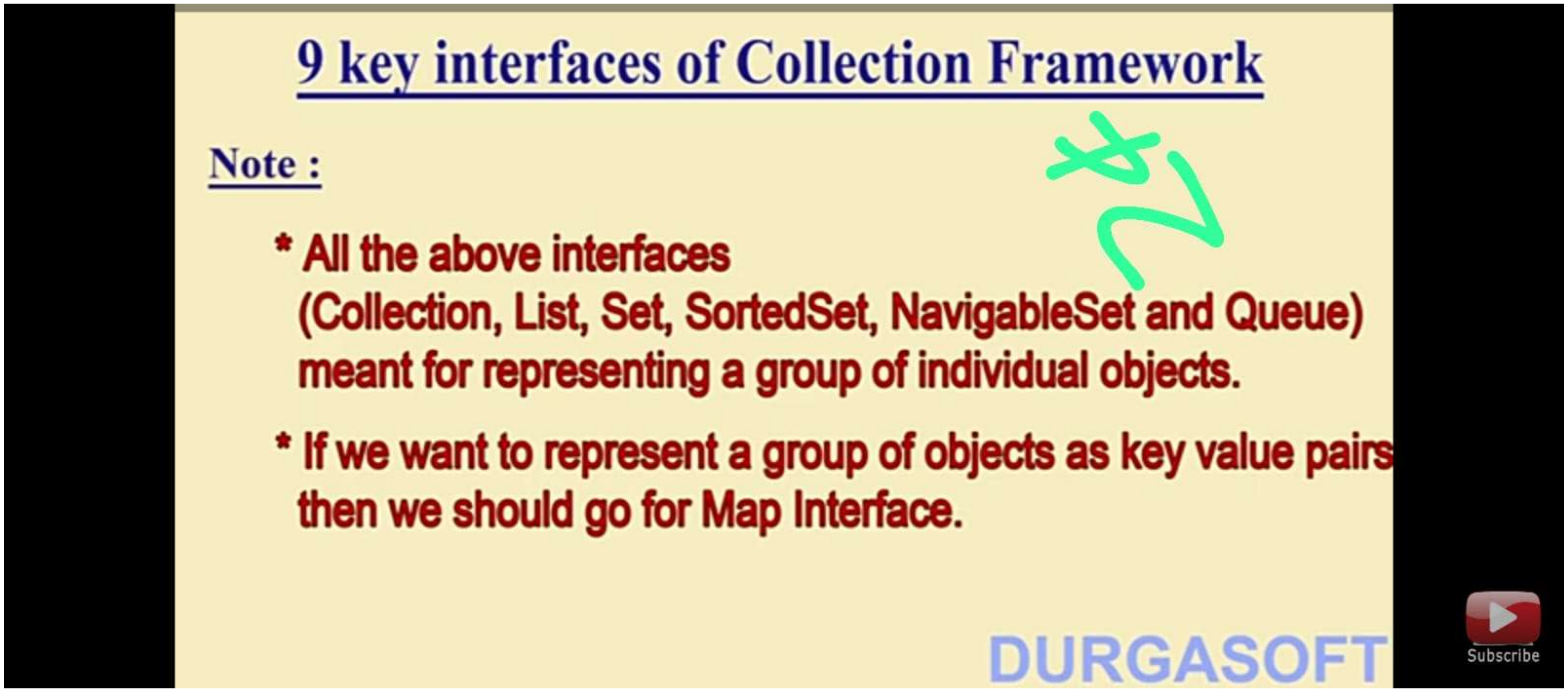
**vi. Queue :**

* It is child interface of Collection.
* If we want to represent a group of individual objects prior to processing then we should go for Queue.

Ex: before sending a mail all mail id's we have to store somewhere and in which order we saved in the same order mail's should be delivered (First in First out) for this requirement Queue concept is the best choice.

DURGASOFT

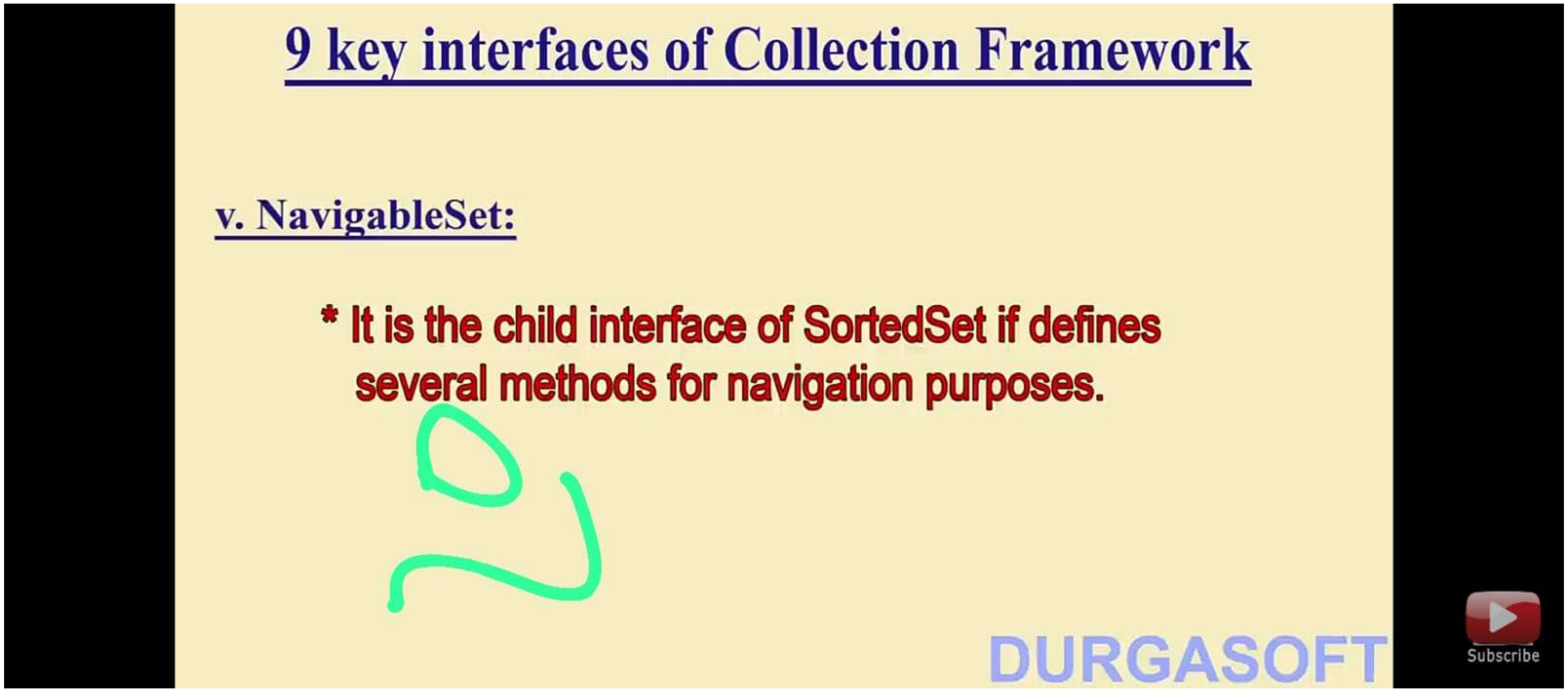

9 key interfaces of Collection Framework
vi. Queue :
Collection (I)
(1.2 version)
Queue (I)
(1.5 version)
PriorityQueue
(1.5 version)
BlockingQueue
(1.5 version)
LinkedBlockingQueue
(1.5 version)
PriorityBlockingQueue
(1.5 version)
23
Subscribe



## 9 key interfaces of Collection Framework

**Note :**

* All the above interfaces
  (Collection, List, Set, SortedSet, NavigableSet and Queue)
  meant for representing a group of individual objects.
* If we want to represent a group of objects as key value pairs
  then we should go for Map Interface.



## 9 key interfaces of Collection Framework

**v. NavigableSet:**

* It is the child interface of SortedSet if defines several methods for navigation purposes.

20