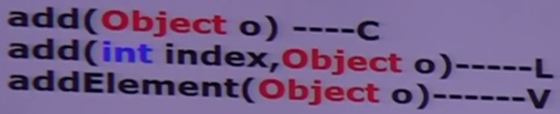
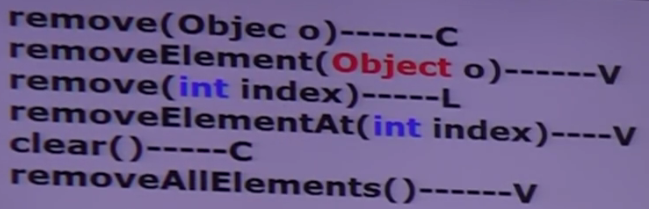
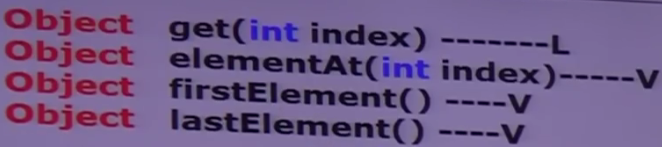
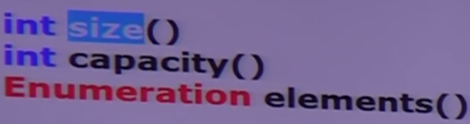
# Differences b/w ArrayList and LinkedList

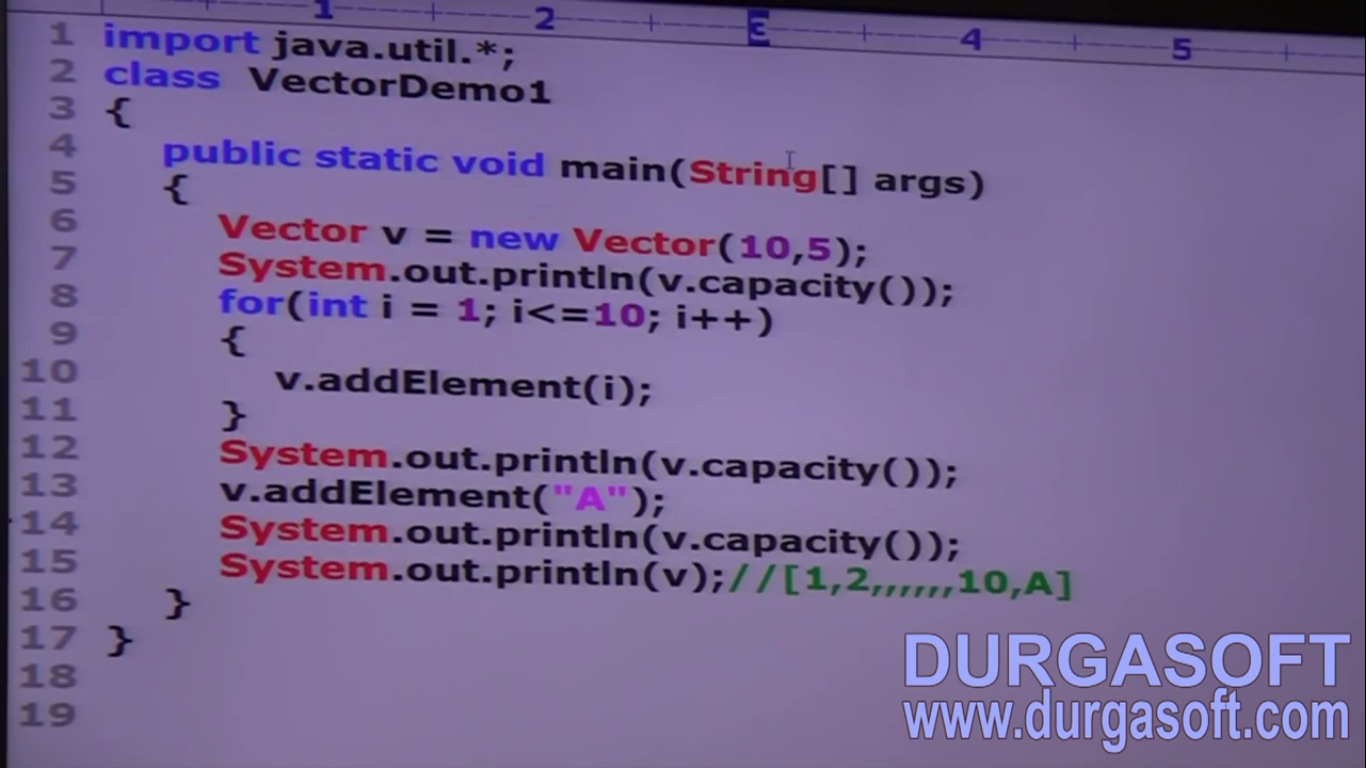
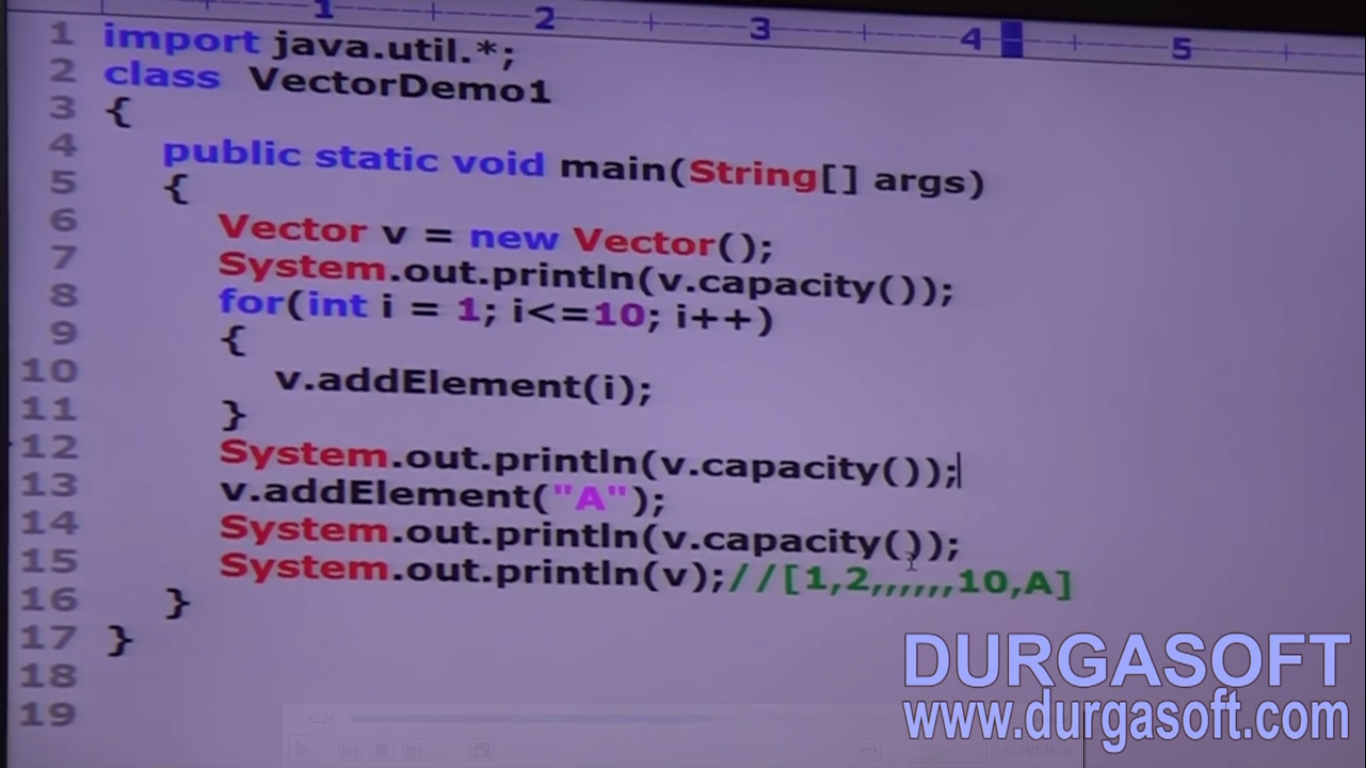
|  |  |
| --- | --- |
| **ArrayList** | **LinkedList** |
| **Best choice** for retrieval operation | **Best Choice** for Insertion and Deletion in the middle |
| **Worst Choice** for insertion, deletion in the middle because internally several shift operations are performed | **Worst Choice** for retrieval operation |
| Elements are stored in consecutive memory location so retrieval operation becomes easy | Elements are not stored in consecutive memory location so retrieval operation becomes complex. |
| **Jatin->** Implements RandomAccess | **Not** |

Vector(C)

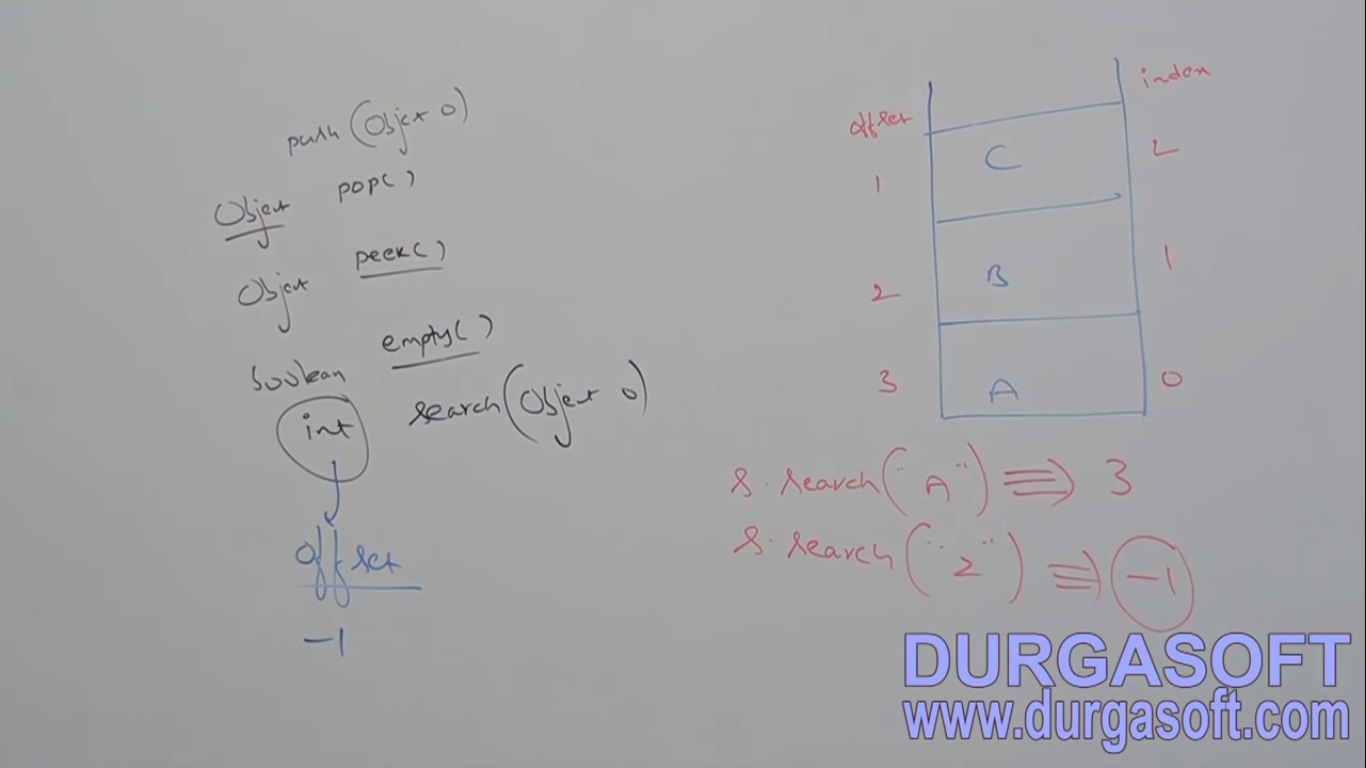
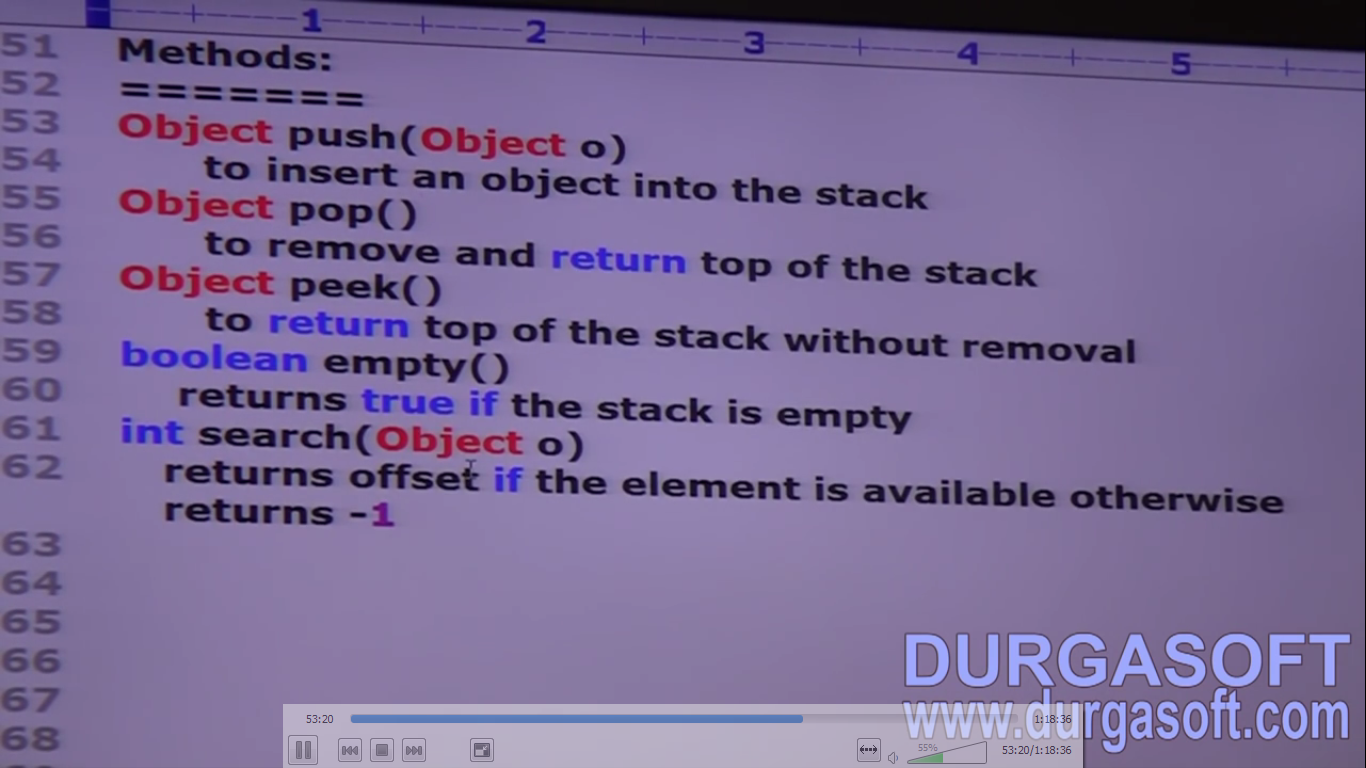
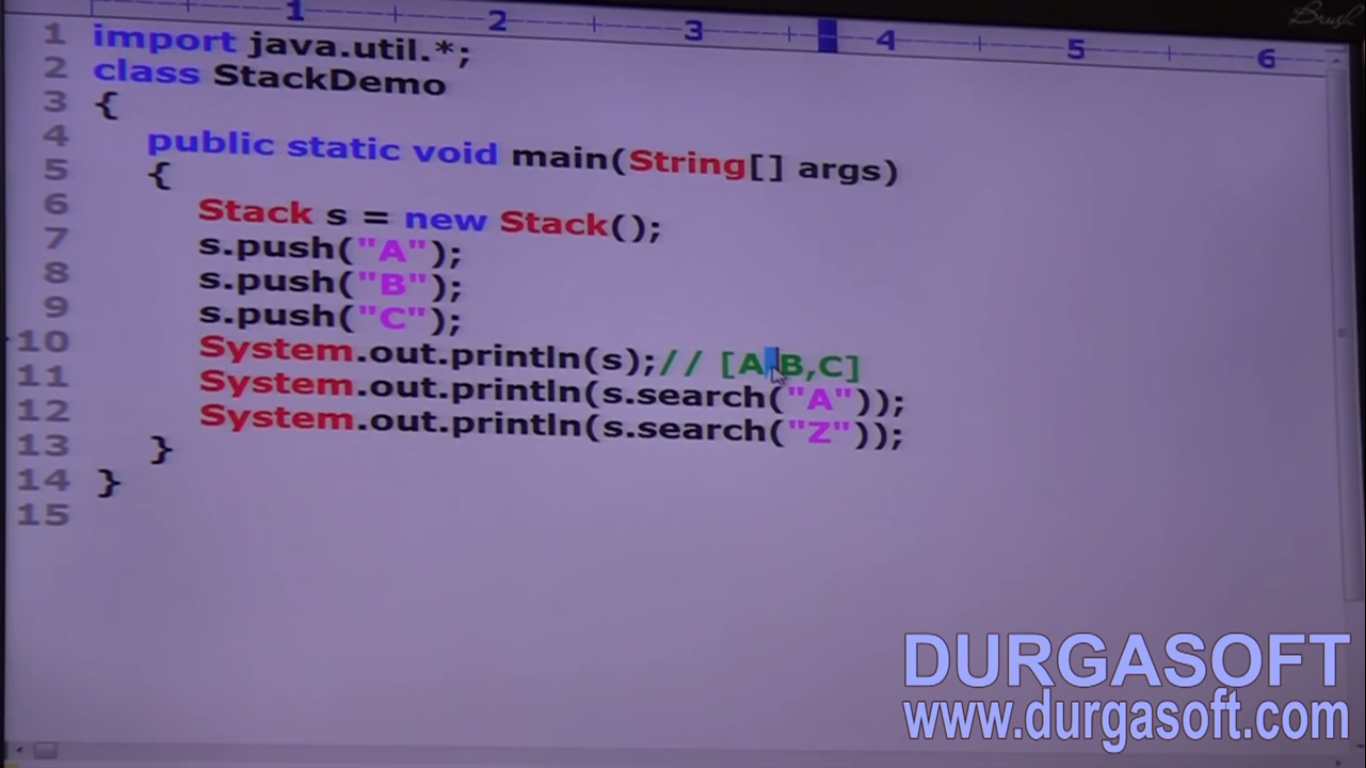
1. **Properties**:
   1. **Underlying Data Structure:** Resizable Array.
   2. **Insertion Order**: Preserved.
   3. **Duplicate objects**: Allowed.
   4. **Heterogeneous objects**: Allowed.
   5. **Null Insertion**: Allowed.
   6. **Implements**: Serializable, Cloneable, RandomAccess.
   7. **Thread-Safe**: Yes. Every method in the Vector is synchronized.
   8. **NOTE**: Except the thread-safe, all the properties of Vector and ArrayList are same.
2. **Constructor**:   
   1. **Vector();**
      1. Creates a new Vector object with default initial capacity = 10
      2. If it reaches its max capacity, then a new Vector object will created with   
         New Capacity = Current Capacity \* 2; 🡨 Just Double the capacity.
   2. **Vector(int intialCapacity)**
   3. **Vector(int initialCapacity, int incrementalCapacity);**
      1. When we know that I’m going to insert near about 1000 objects. Afterwards, if I needed to insert more, it would be 3-4 objects. Then use this constructor.   
         This incrementalCapacity is available in Vector but not in ArrayList.
   4. **Vector(Collection c)**
      1. Creates an equivalent Vector object for the given collection. This constructor is for inter-conversion b/w Collection objects.
3. **Vector Specific Methods:  
   NOTE:** Vector is old-generation code so its method names are lengthy.
   1. To add objects from different interfaces or classes.  
      
   2. To remove objects   
      
   3. To get objects  
      
   4. Mislleneous Methods:



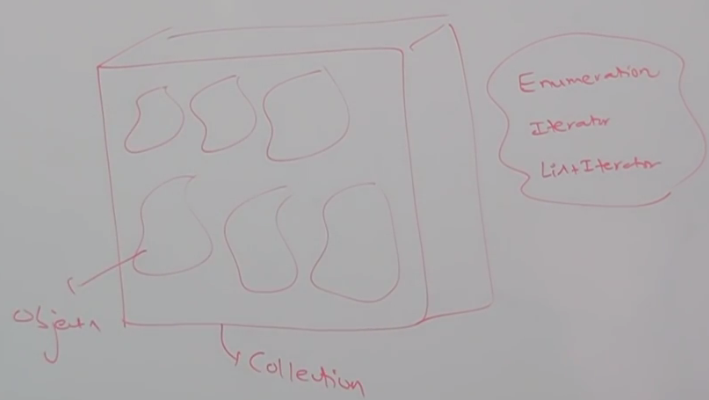
* 1. NOTE: When there was no List(I) which came in 1.2v, Vector was implemented with method working with index. So vector also contains methods related to index of its own.

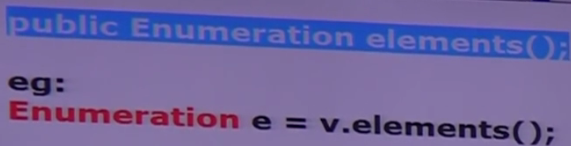
1. **Demonstrating the vector method capacity()**
2. d

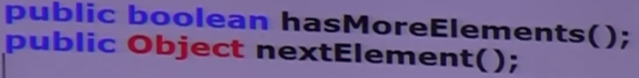
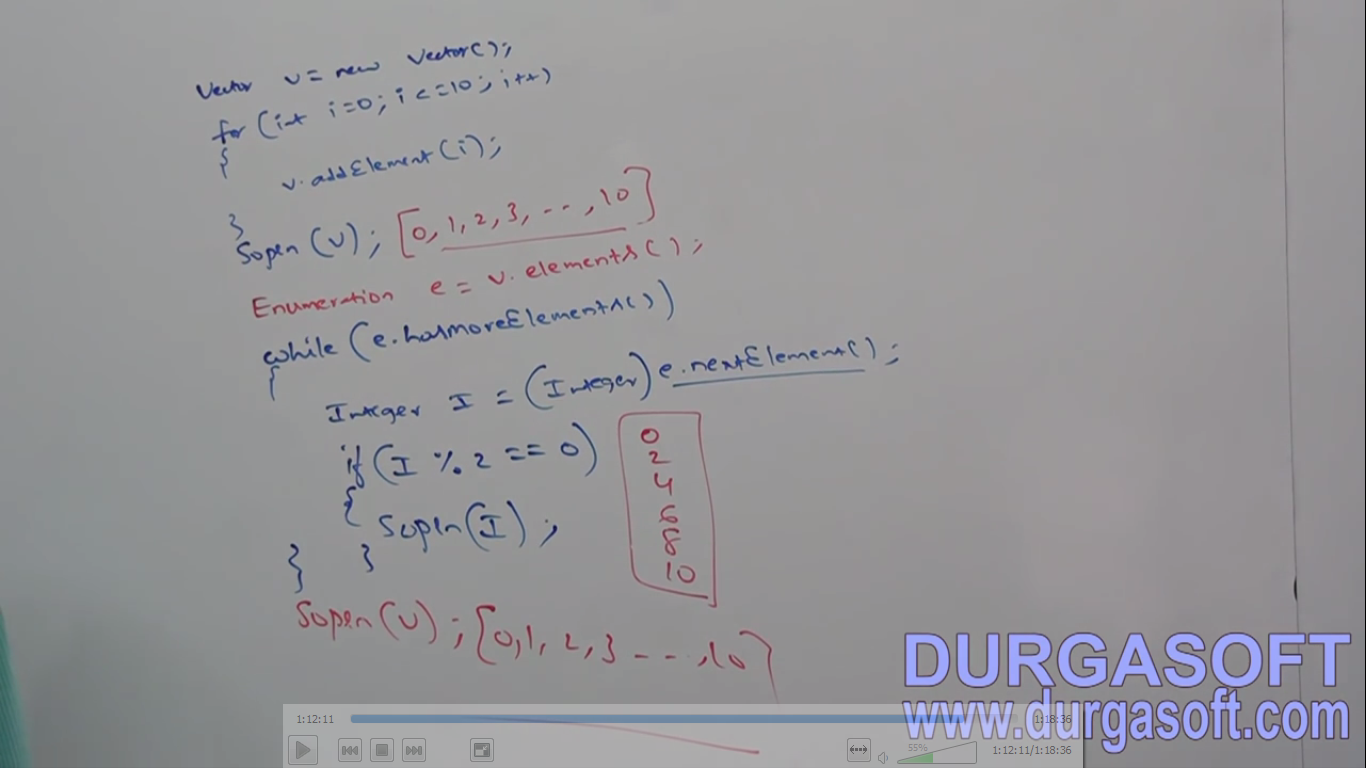
Stack(C)

1. Child class of Vector(C)
2. Specially designed class for LIFO order.
3. **Constructors:**
   1. **Stack()**
4. **Stack Specific Methods**:
   1.   
      
5. **Example**:  
   
6. d

3 Cursors of Java

1. This is next sub-topic we are going to study
2. **Scenario**:   
   We have a box of mangoes. I want to each mango not together but one by one.  
     
   **Cursors**: If we want to get objects **one by one** from collection then we should go for cursor.   
   There are 3 kinds of cursors in java.
   1. **Enumeration:**
   2. **Iterator:**
   3. **ListIterator**:
3. **Enumeration:**
   1. We can use Enumeration to get object one by one from legacy collection object.
   2. We can create Enumeration object by using **elements() : Enumeration** of Vector class.



* 1. **Methods**: only 2 methods.
     1. **hasMoreElements(): boolean**
     2. **nextElement():Object**
  2. **Example:**

1. **d**
2. d