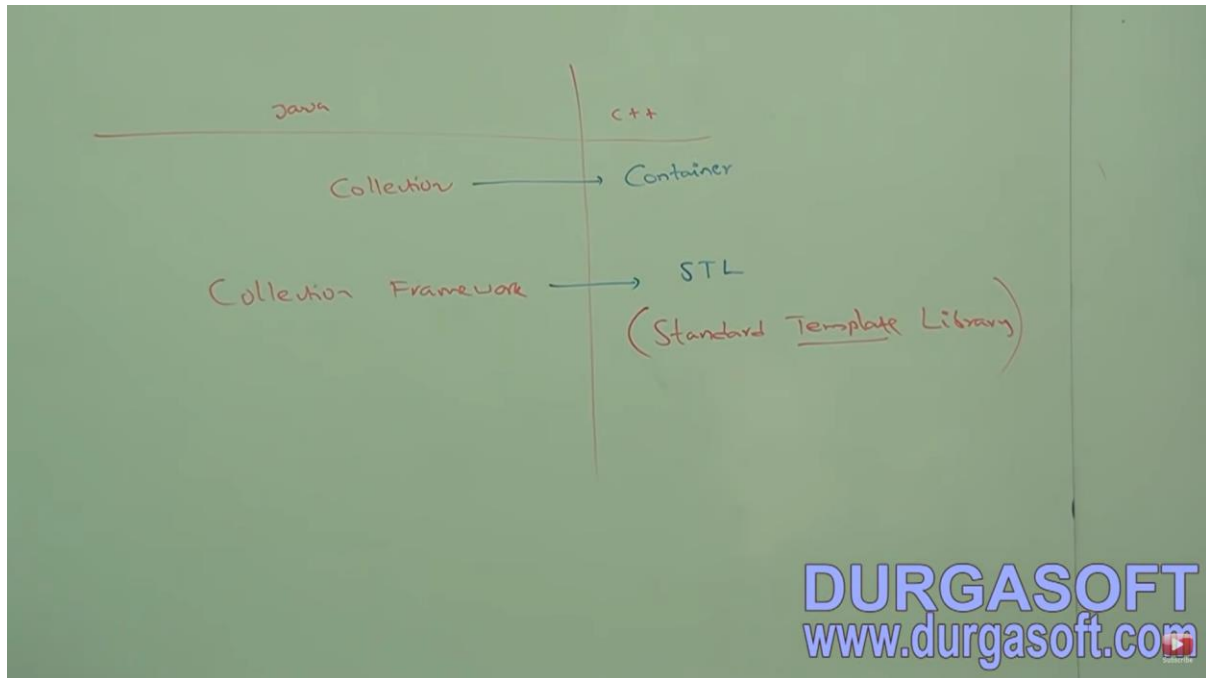


Collections are growable in nature.
Collections holds both homogeneous and non-homogeneous.

If we want to represent group of individual objects as a single entity than we should go with collection

Collection Framework: it contains several classes and interfaces which can be use to represent

a group of individuals as a single entity.



9 key interfaces of **Collection** Framework:

1. Collection
2. List
3. Set
4. SortedSet
5. NavigableSet
6. Queue
7. Map
8. SortedMap
9. NavigableMap

Collection(I) : if we want to represent a group of individual object as a
Single entity than we go for Collection.

Collection interface define the most common methods which are available

Any collection objects. In general collection interface is consider as root interface

Of collection framework.

There is no concrete class which implements collection interface directly.

Collection(I) vs Collections(class)

Collection is an interface if we want to represent individual Object as single entity then we should go for collection.

Collection is a utility class present in java.util package to define several utility methods for object like sorting searching etc.

list(I) : it is the child interface of collection. If we want to represent group of Individual object as a single entity where duplicates are allowed and insertion order must be preserved then we should go for list.

List Implemented class

1. ArrayList
2. LinkedList
3. Vector
4. Stack

Set(I): it is the child interface of collection. If we want to represent group of Individual object as a single entity where duplicates are not allowed and insertion order is not required then we should go for Set.

Set Implemented Class

1. HashSet
2. LinkedHashSet

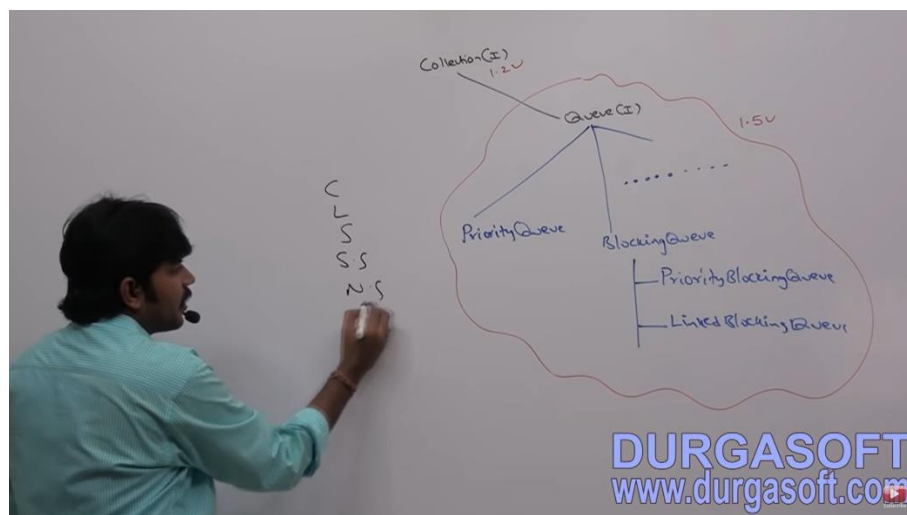
SortedSet(I) it is the child interface of set. If we want to represent group of Individual object as a single entity where duplicates are not allowed and insertion order requires some sorted order then we should go for SortedSet.

NavigableSet(I) it is the child interface of SortedSet. It contains several methods for navigation purposes

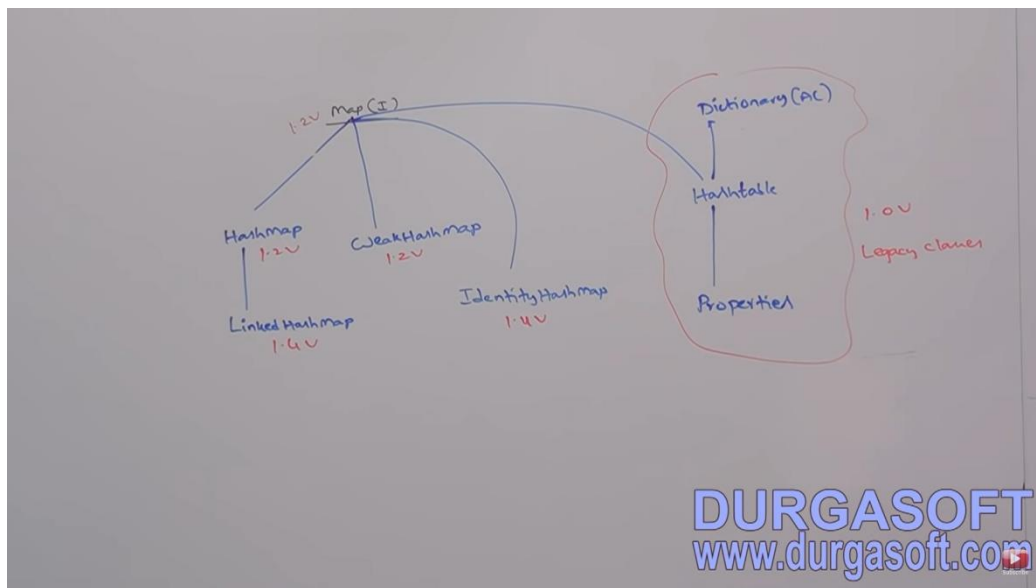
Implemented class of navigableSet

1. TreeSet

Queue(I): it is the child interface of Collection



Map(I)



SortedMap(I) sorted based on key.

NavigableMap it is child interface sortedMap. It Define several methods for navigation purposes

Implemented class:

1. TreeMap

Collection(i) Methods:

1. boolean add(Object o)
2. boolean addAll(Collection c)
3. boolean remove(Object o)
4. boolean removeAll(Collection c)
5. boolean retainAll(Collection c)
 - a. To remove all objects except those present in c
6. void clear()
7. boolean contains(Objects o)
8. boolean containsAll(Collection c)
9. boolean isEmpty()
10. Int size();

11. Object[] toArray();
12. Iterator iterator()

There is no contrite class which implement collection interface directly

List

List Interface define the following specific methods

1. void add(int index, Object o)
2. boolean addAll(int index , Collection c)
3. Object get(int index)
4. Object remove(int index)
5. Object set(int index, Object new)
To replace the element, present at specified index with provided Object and returns old object
6. int indexOf(Object o)
 - a. returns index of first occurrence of 'o'
7. int LastIndexOf(object o)
8. ListIterator listIterator();

ArrayList :

Constructor :

1. ArrayList al = new ArrayList(); default size 10.
2. ArrayList al = new ArrayList(int intialCapacity);
3. ArrayList al = new ArrayList(Collection C);
 - a. ArrayList al new ArrayList(collection c)
create an equivalent ArrayList element
for the given collection

Methods :

Set(index,object) to replace object at index

Usually we can use collection to hold and transfer object from one location to another location(container) to provide support for this requirement every collection class by default implements serializable and cloneable interfaces

RandomAccess Interface

LinkedList

Constructor:

1. `LinkedList l = new LinkedList();`
2. `LinkedList l = new LinkedList(Collection c);`

Methods:

1. `add();`
2. `void addFirst(Object o);`
3. `void addLast(Object o);`
4. `Object getfirst();`
5. `Object getLast();`
6. `Object removeFirst();`
7. `Object removeLast();`

Vector

Resizable

Insertion order

Duplicate

Heterogeneous

Null insertion

Implement serializable cloneable, random access,

Thread safe

Constructors:

1. `Vector v = new Vector();` default size 10 ,
newcapacity=cc*2
2. `Vector v = new Vector(int initialCapacity);`
3. `Vector v = new Vector(int initialCapacity, int incremental capacity)`
4. `Vector V = new Vector(collection c);`

Methods:

`addElement(Object o);`
`removeElement(Object o);`
`removeElementAt(int index);`
`removeAllElements();`

`Object elementAt(int index)`

`Object firstElement();`

`Object lastElement();`

`Int size();`

`Int capacity();`

`Enumeration elements();`

Stack

It is child class of vector

Last in first out order.

Constructor:

1. Stack s = new Stack();

Methods:

1. push(Object o);

2. Object pop();

3. Object peek();

4. Bool empty();

5. int search(Object o); -1, return offset if the element is available otherwise return -1.

The 3 cursors of java.

Enumeration

Iterator

ListIterator

```
Enumeration e=v.elements();
```

```
While(e.hasMoreElements())
```

```
{
```

```
    Integer l = (Integer)e.nextElement();
```

```
    Sop(l);
```

```
}
```

ListIterator

We can create iterator object by using iterator() of collection object

Ex

```
Iterator itr = o.iterator();
```

Methods

1. Public Boolean hasNext();
2. Public Object next();
3. Public void remove();

Limitation

Single direction

Read and remove not add element not replace.

ListIterator

Bidirection

Read, remove, add and remove operation done.

```
public ListIterator ListIterator();
```

```
Ex: ListIterator its = l.listIterator();
```

l = Any list Object

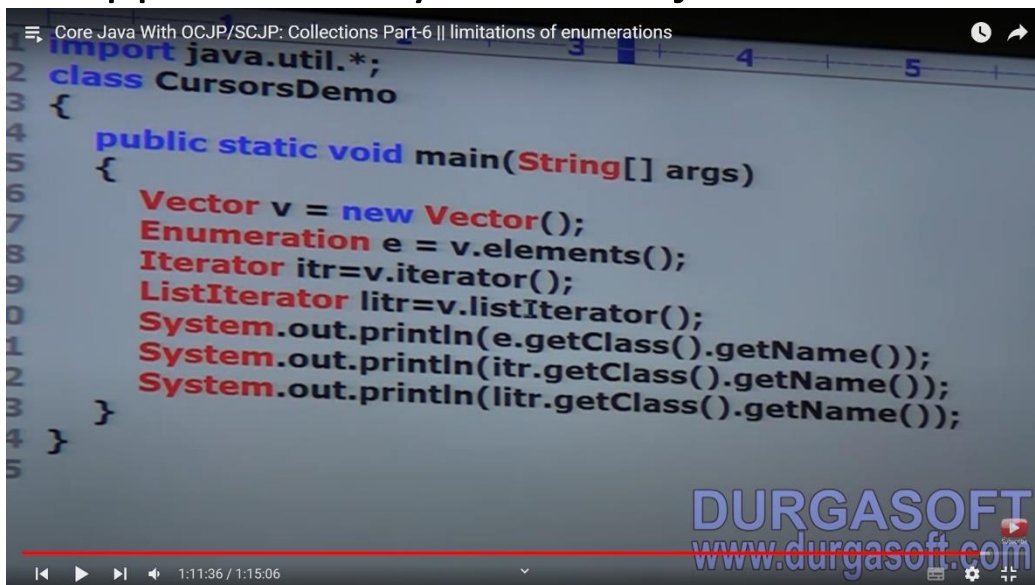
listIterator is child Interface of Iterator and hence all methods present in iterator by default available by listIterator.

Methods :

1. Public Boolean hasNext();
2. Public Boolean next();
3. Public int nextIndex();
4. Public Boolean hasPrevious();
5. Public Object previous();
6. Public int previousIndex();
7. Public void remove();
8. Public void add(object o);
9. Public void set(object);

Limitation:

Its applicable only for list objects.

A screenshot of a video player interface. The video title is "Core Java With OCPJP/SCJP: Collections Part-6 || limitations of enumerations". The video content shows a Java code snippet for a class named "CursorsDemo". The code imports "java.util.*" and defines a "main" method. Inside the "main" method, it creates a "Vector" object, an "Enumeration" object from its elements, an "Iterator" object, and a "ListIterator" object. It then prints the class names of these three objects using "System.out.println". The video player has a progress bar at the bottom showing "1:11:36 / 1:15:06" and a "DURGASOFT" watermark with the website "www.durgasoft.com".

```
1 import java.util.*;  
2 class CursorsDemo  
3 {  
4     public static void main(String[] args)  
5     {  
6         Vector v = new Vector();  
7         Enumeration e = v.elements();  
8         Iterator itr=v.iterator();  
9         ListIterator litr=v.listIterator();  
10        System.out.println(e.getClass().getName());  
11        System.out.println(itr.getClass().getName());  
12        System.out.println(litr.getClass().getName());  
13    }  
14 }
```


HashSet

1. Underling data structure is hash Table.
2. Duplicates are not allowed.
3. Insertion order is not preserved, and it is based on hash code of object.
4. Null insertion is possible only once.
5. Heterogenous object are allow;
6. Implements serializable and cloneable but not RandomAccess interface
7. HashSet is best choices of search operation.

Constructors:

1. `HashSet h = new HashSet();`
2. `HashSet h = new HashSet(int initialCapacity);`
3. `HashSet h = new HashSet(int initialCap, float fillRatio);`
4. `HashSet h = new HashSet(Collection c);`

LinkedHashSet

LinkedHashSet is child class of hashSet.

It's is exactly same as HashSet Including constructors and methods excepts the following differences

1. Underling data structure is Linked List + Hash Table
2. Insertion order is preserved

```
LinkedHashSet h = new LinkedHashSet();
```

Common use LinkedHashSet to develop cache-based application

SortedSet

SortedSet is a child interface of set

If we want to represent a group of individual objects according to some sorting order without duplicate, then we should go for SortedSet.

SortedSet Interface define the following specific method,

1. Object **first()**;
2. Object **last()**;
3. SortedSet **headSet**(Object obj)
 - a. Return SortedSet whose elements are less than object.
4. SortedSet **tailSet**(object obj)
 - a. Returns SortedSet whose elements are \geq obj
5. SortedSet **subSet**(object obj1, Object obj2)
 - a. Returns SortedSet whose elements re \geq object and $<$ obj2
6. Comparator **comparator()**

7. Return Comparator object that describes underlying sorting technique. If we are using default natural sorting order then we will get null.

TreeSet

Underlying data structure is balanced tree

1. Duplicate not
2. Inserting order not preserve
3. Not heterogeneous objects are not allows
4. Only once null

Constructor:

1. `TreeSet t = new TreeSet();`
2. `TreeSet t = new TreeSet(Comparator c);`
 - a. Create an empty TreeSet where the element will be insert according to customising sorted order specified by comparator object.
3. `TreeSet t = new TreeSet(Collection c);`
4. `TreeSet t = new TreeSet(SortedSet s);`

Null acceptance:

- 1 `t.add(null);`

Re:NPE

2 t.add(null);

t.add("a"); return null pointer exception

```
import java.util.TreeSet;
public class TreeSetEx {
    public static void main(String args[])
    {
        TreeSet<String> t = new TreeSet<String>();
        t.add("A");
        t.add("B");
        t.add("C");
        System.out.println(t);
    }
}
```

If we are depending on default natural sorting order compulsory the object should be homogenous and comparable otherwise, we get classCastException

Comparable(Interface)

1. Java.lang package
2. compareTo();

obj1.compareTo(obj2);

-	iff obj1 comes before obj2
+	iff obj1 comes after obj2
0	Iff obj1 & obj2 equal

```
public class ComparableEx {
    public static void main(String args[]) {
        System.out.println("A".compareTo("Z"));
        System.out.println("A".compareTo("A"));
        System.out.println("Z".compareTo("A"));
    }
}
```

```
}  
}
```

Comparable means for Default natural sorting Order

Where as

Comparator means for customized Sorting order.