****

Collections

****

- Array has fixed size;
- Array list is not a fixed size;
- In an array if we want to add any element at any index place, we have to manually swift all elements by one and we have to insert.
- But in arrayList we have a built in method to insert at specified index.
- Collection —-> [1,2,3,4,5,6]
- List [1,2,3,4,5,6] with index this is the only difference.
- List is also allowed with duplicates.
- Set has no duplicates and no indexes, uniques values;
- Collection framework , classes and interfaces are present from java.util package.
-

Collection Interface :

- add(E e)
- addAll(collection<E> e)
- remove(E e)
- removeAll(Collection<E> e)
- retainAll(Collection<E> e)
  - C1 - [ABCDEF] c2 -[EF]
  - Output: c1- [EF] Only common elements will present.
- clear()
- isEmpty()
- contains(Object e)
- containsAll(Collection<E> e)
  - C1 - [ABCDEF] c2 -[EFGH]
  - Output: False because GH is not present in c1.
- equals(object o)
- size()

- iterator()
- toArray();

- This collection of classes, Interfaces are Generics.
- This Support Object of any class.

Interface List extends Collection:

```
3  import java.util.*;
4
5  public class Demo {
6      public static void main(String args[])
7      {
8          ArrayList al = new ArrayList();
9          al.add("durga");
10         al.add(10);
11         al.add('c');
12         System.out.println(al);
13     }
14 }
15
```

- add(int index, E e)
- addAll(int index, Collection<E> e)
- remove(index)
- get(int index)
- set(int index, E e)
- sublist(int from, int to)
- indexOf(Object o)
- lastIndexOf(Object o);
- listIterator()
- listIteratoR(int index);

- interface set extends Collection
  - No Extra Methods for Set.

Interface Queue extends Collection

- add(E e)
- poll()
  - If Queue is empty then return null
- remove() throws NoSuchElementException

- peek()
  - Returns first element
  - Returns null if Queue is empty.
- element() throws NoSuchElementException
  - Returns first element

```
5  public class Vector {
6      public static void main(String args[])
7      {
8          Vector al = new Vector();
9          System.out.println(al.capacity());
10         for (int i =1;i<10;i++)
11         {
12             al.addElement(i);
13         }
14         System.out.println(al.capacity());
15         al.addElement('A');
16         al.addElement('A');
17         System.out.println(al.capacity());
18     }
19 }
```

3- Types Cursors:

- Enumeration
  - We can use enumerator to get Objects one by one from legacy Colletion Object.
  - We can create Enumeration by using elements() of vector class
  - Public Enumeration elements()
  - EX: Enumeration e = V.elements();

```
 3 import java.util.*;
 4 public class Demo {
 5     public static void main(String args[])
 6     {
 7         Vector  al = new Vector();
 8         System.out.println(al.capacity());
 9         for (int i =1;i<10;i++)
10         {
11             al.addElement(i);
12         }
13         System.out.println(al.capacity());
14 //      Applicable for lagacy class.
15         Enumeration e =al.elements();
16         while(e.hasMoreElements())
17         {
18             Integer i = (Integer)e.nextElement();
19             System.out.println(i);
20             }
21     }
22 }
23
```

- Limitations:
- We can apply Enumeration concept only for legacy classes and it is not a universal cussor.
- By using Enumeration we can get only read access and we can perform remove Operation.
- To overcome above Limitations we should go for iterator.

- Iterator
- listIterator

Stack:

`boolean`
`empty()`
Tests if this stack is empty.

`E`
`peek()`
Looks at the object at the top of this stack without removing it from the stack.

`E`
`pop()`
Removes the object at the top of this stack and returns that object as the value of this function.

`E`
`push(E item)`

Pushes an item onto the top of this stack.

```
int
search(Object o)
```

Returns the 1-based position where an object is on this stack.

****

Collections-Durga

****

An Array is an Indexed Collection of Fixed Number of HomoGeneous data Elements,

The Main Advantages of Arrays is:

We can Represent multiple values by using Single Variable, so that readability of the code will be improved.

Limitations of Arrays:

- Arrays are Fixed in Size, i.e once we create an array there is no chance of increasing or decreasing size based on our requirement, due to this to use Array concept compulsory we should know the size in Advance, which may not possible always.

  —----------

- Array can hold only HomoGeneous Data Type Elements Ex:
- Student [] s = new Student[1000];
- S[0] = new Student(); //valid
- S[1] = new Customer(); // Invalid

  —----------

- We can solve this problem by using Object Type Arrays.
- Object[] s = new Object[1000];
- S[0] = new Student(); //valid
- S[1] = new Customer(); // valid

  —----------

- Arrays Concept is not implemented based on some Standard dataStructure and hence the readymade method support is not

Available, For every Requirement we have to write the code Explicitly which increases complexity of Programming.

To Overcome the above problems of arrays we should go for Collections Concept:

- ➔ **Collections Are Growable in nature, i.e Based on our requirement we can increase or decrease the size,**
- ➔ **Collections can hold both HomoGeneous And HeteroGeneous Elements,**
- ➔ **Every Collection Class Is implemented based on some Standard DataStructure Hence For Every Requirement ReadyMade method support is available. Being a programmer we are responsible to use those Methods and we Are not responsible to implement those methods.**

Differnce Between Arrays And Collections	
Arrays	Collections
Fixed in size	Growable in nature
Not recommended	Highly recommended
High Performance	Low Performance
only HomoGeneous	HomoGeneous And HeteroGeneous
No UnderLying DataStruture	UnderLying DataStruture
Arrays can hold Objects and primitives	Arrays can hold Objects

Collection:

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

Collection Framework:

It COntains Several Classes and interfaces which can be used to represent a group of individual Objects as a Single entity.

Collection in java —-----> Container in C++

Collection Framework in java—-----> STL(Standard Template Library) in C++

9 key Interfaces of Collection Framework:

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

1) Collection(I):

1. If we want to represent a group of Individual Objects as a single entity then we should go for Collection.
2. Collection Interface defines most Common methods which are applicable for any collection Objects.
3. In General Collection Interface consider as root Interface of Collection frameWork.
4. There is no-Concrete Class which implements the Collection interface directly.

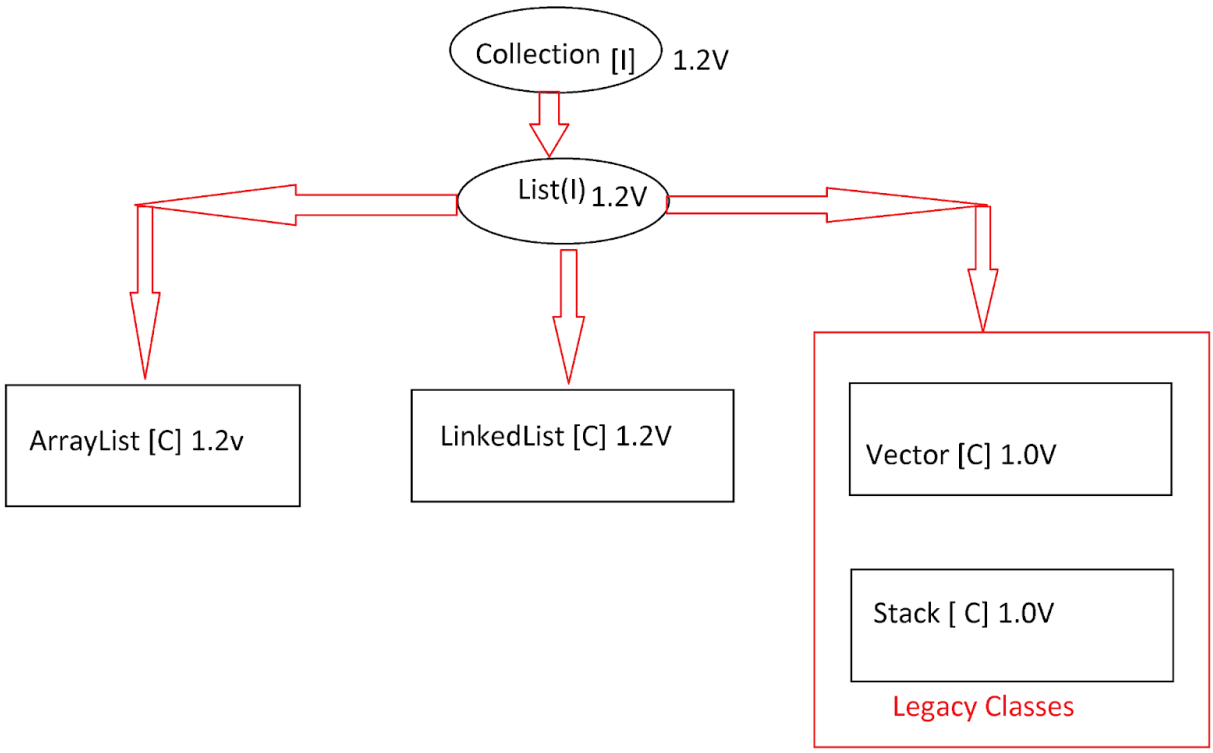
What is the Difference Between Collection[I] and Collections[C].

- Collection is an Interface , if we want to represent a group of Individual Objects as a single entity then we should go for Collection.
- Collections[C] is an Utility Class present in the Java.util package to define Several Utility methods for Collection Object like SOrting, Searching etc.

2) List[I]:

1. It is the child Interface of Collection.

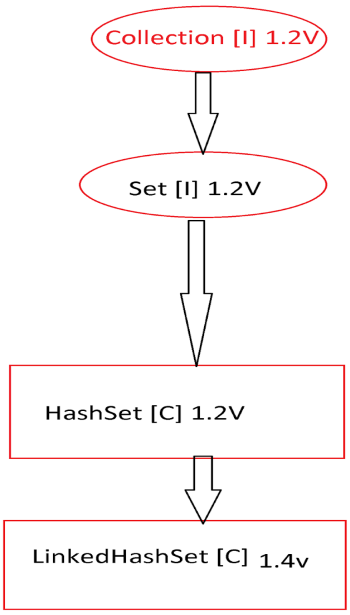
2. If we want to represent a group of individual objects as a SIngle Entity , where Duplicates are allowed and Insertion Order must be preserved, then We should go for a List.



3.

3) Set(I):

1. It is the child Interface of Collection
2. 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 a Set.



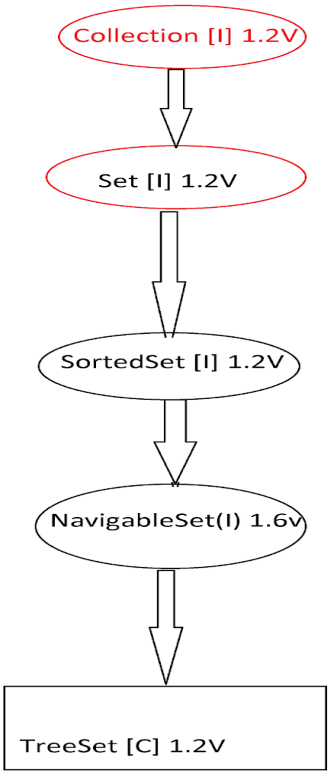
3.

4) SortedSet(I):

1. It is the child Interface of a set. If we want to represent a group of individual Objects as a single entity where duplicates are not allowed and all Objects Should be inserted according to some sorting Order, then we should go for Sorted Set.

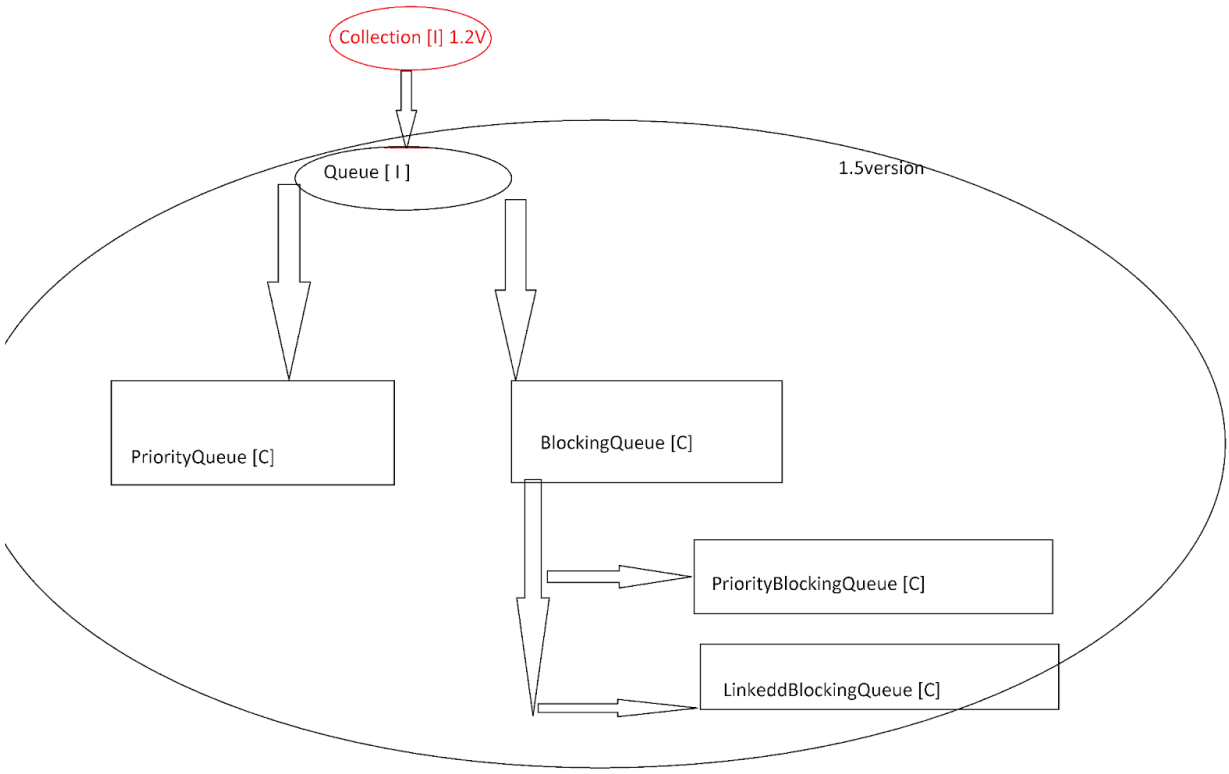
5) NavigableSet(I)

1. It is the child Interface of Sorted Set it COntains Several Methods for Navigation Purpose.

2. 

6) Queue[I]

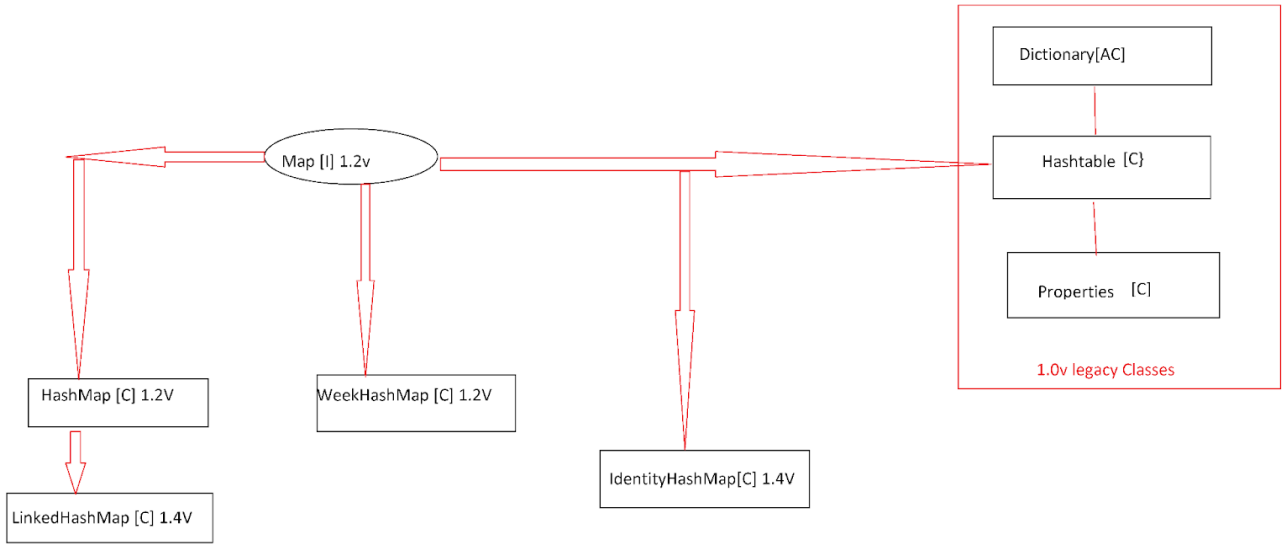
1. It is the child Interface of the collection. If you want to represent a Group of individual Objects prior to Processing then we should go for Queue.
2. Usually Queue follows first in First out Order, But based on our Requirement we can implement our own priority also.
3. EX: Before sending a mail all mail ids we have to store in some dataStructure in which Order we added mail ids in the same order only mail should be Delivered.For this requirement Queue is Best Choice.



4.

7) Map[I] :

1. All the Above Interfaces [Collection, List, Set, SortedSet, NavigableSet, Queue] Meant for representing a group of individual Objects, If you want to represent a group of Objects as Key-Value Pairs then we should go for Map.
2. Map is not the Child Interface of Collection.If you want to represent a group of Objects as Key-Value Pairs then we should go for Map.
3. Durga :101 (key:value)
4. Madhu :102
5. Both Key and Value are Object Only, Duplicates Keys are not Allowed but values can be Duplicated.



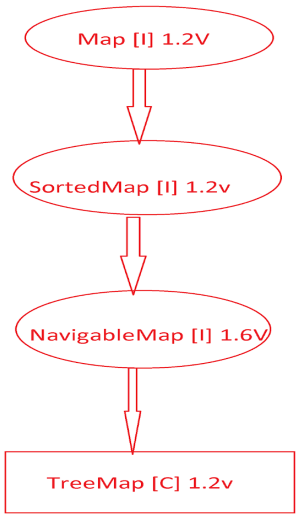
6.

8) SortedMap[I] :

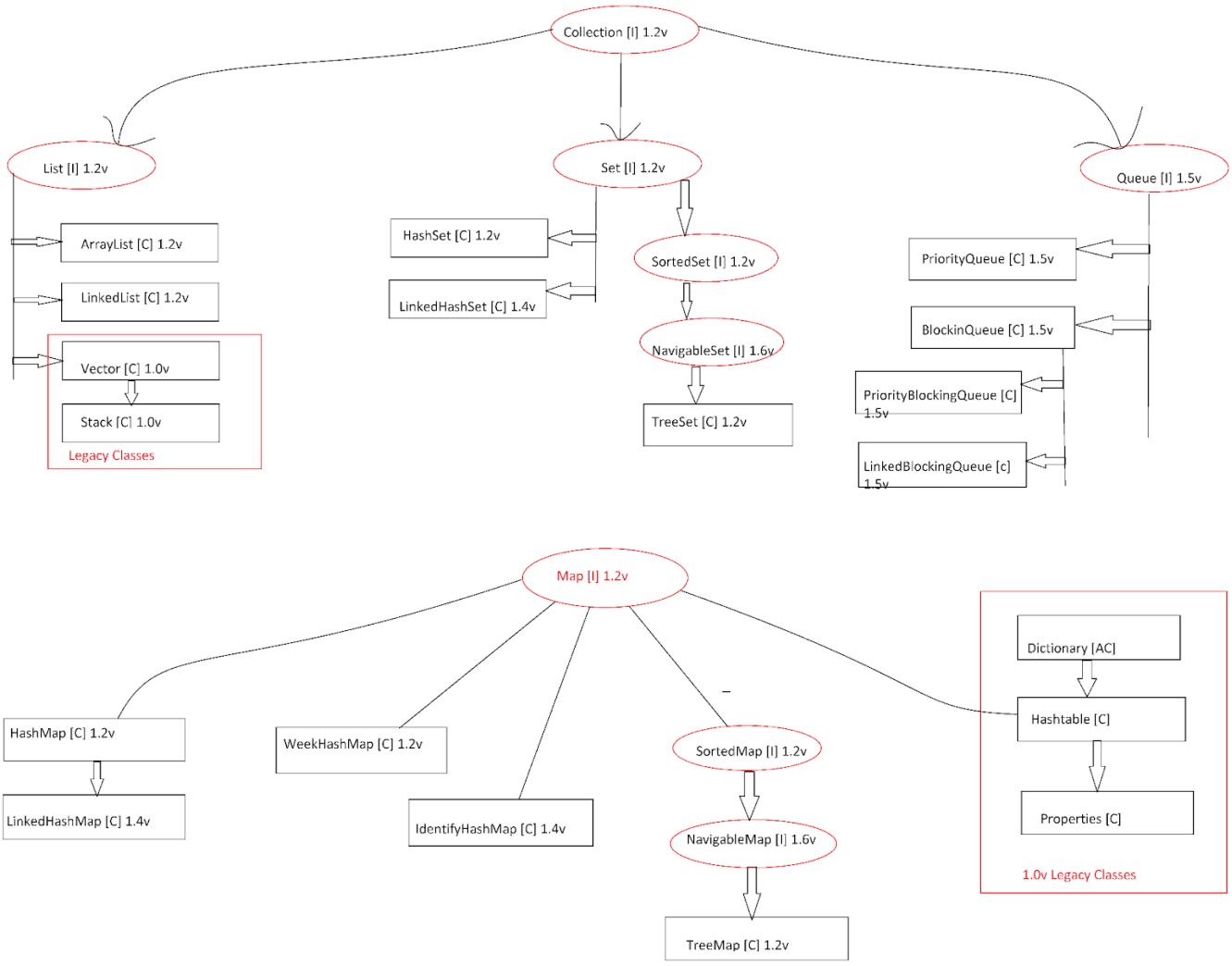
1. It is the child Interface of map. If we want to represent a group of Key-value pairs according to some sorting order of Keys, then we should go for a sorted map.
2. In SortedMap the Sorting should be based on Key but not based on value.

9)NavigableMap [I] :

1. It is the child Interface of SortedMap,
2. It defines Several methods for Navigation Purposes.



3.



Natural Sorting:

- **Comparable(x)**
- **Comparator(x)**

Object One by one then go to Cursor:

- **Enumeration**
- **Iterator**
- **ListIterator**

Utility Classes

- **Collections**
- **Arrays**

Legacy Classes

- **Enumeration(X)**
- **Dictionary(AC)**
- **Vector [C]**
- **Stack [C]**
- **Hashtable [C]**
- **Properties [C]**

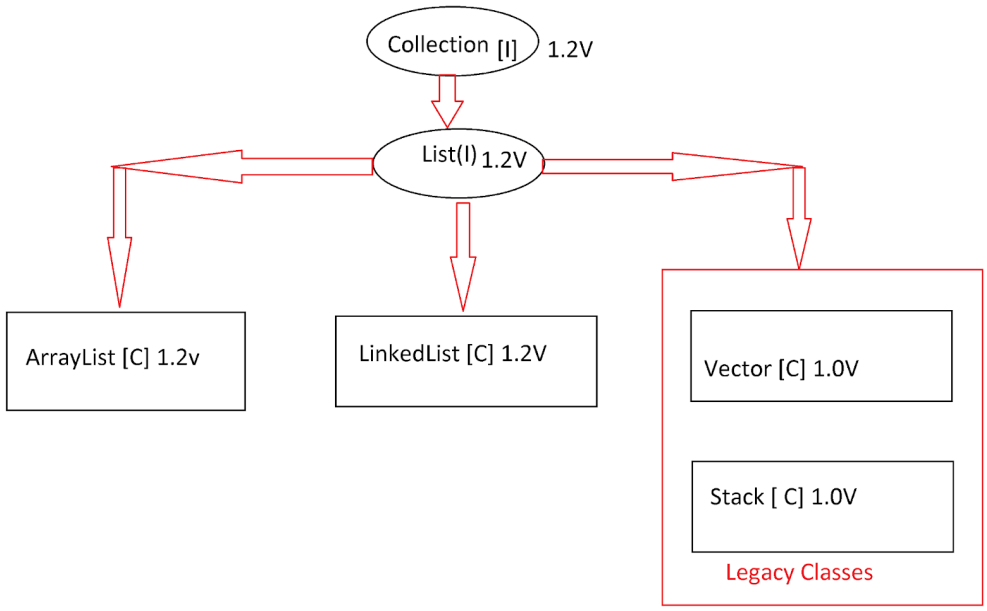
Collection:

- **Collection interface defines the most Common Methods which are applicable for any collection Objects.**
    - **Interface collection:**
        - **boolean add(E e)**
        - **boolean addAll(collection<E> e)**
        - **boolean remove(E e)**
        - **boolean removeAll(Collection<E> e)**
        - **boolean retainAll(Collection<E> e)**
            - **C1 - [ABCDEF] c2 -[EF]**
            - **Output: c1- [EF] Only common elements will present.**
        - **Void clear()**
        - **boolean isEmpty()**
        - **boolean contains(Object e)**
        - **boolean containsAll(Collection<E> e)**
            - **C1 - [ABCDEF] c2 -[EFGH]**
            - **Output: False because GH is not present in c1.**
        - **boolean equals(object o)**
        - **int size()**
        - **Ierator iterator()**
        - **Object[] toArray();**
    - **There is no-concrete class which implements Collection interface directly.**
    - **This collection of classes, Interfaces are Generics.**

- This Support Object of any class.
- interface List extends Collection:

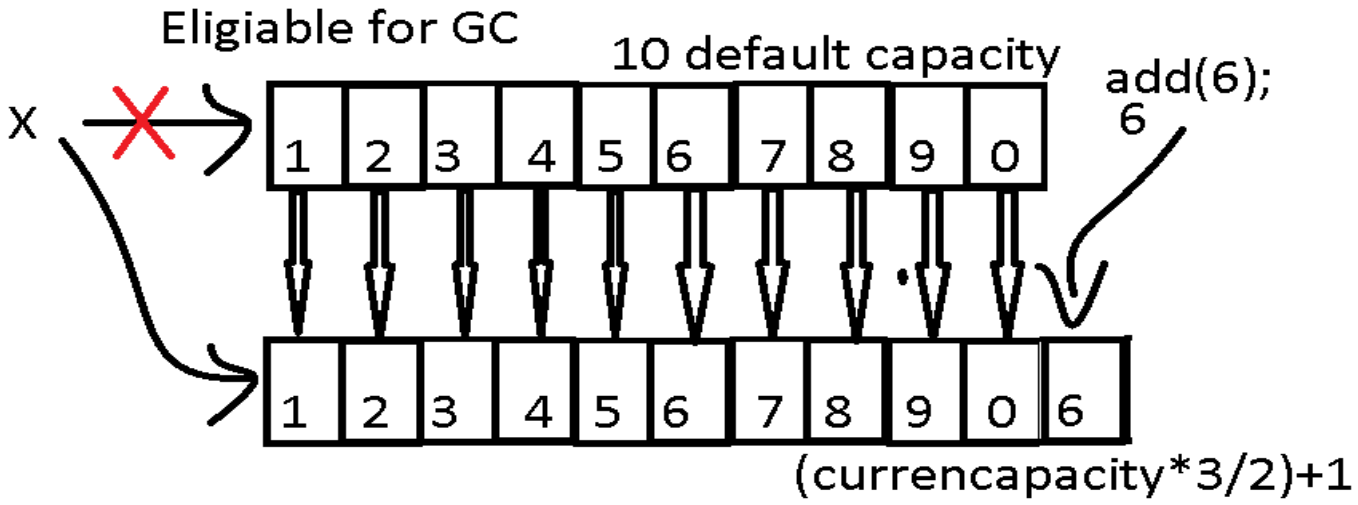
List (I):

- List is the Child interface of Collection.
- If we want to represent a group of individual Object as Single Entity and Duplicates are allowed and insertion order is pressed then we should go for List
- Insertion Order is preserved with respect to the index, and we can differentiate duplicated Objects by using the index. Hence index will play a very important role in List.
- List Interfaces defines the following specific methods [8]:
  - 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)
    - To replace the element present at specified index with provided Object and returns old Object.
  - Int indexOf(Object o);
  - int lastIndexOf(Object o)
  - ListIterator listIterator();



- The UnderLying data Structure is a resizable array or Growable Array.
- Duplicates are Allowed.Insertion Order is preserved.
- HeteroGeneous Objects are allowed, Except TreeSet and TreeMap everywhere heterogeneous Objects are allowed.
- Null Insertion is possible.

ArrayList [Constructor]:



1. ArrayList l = new ArrayList();

a. Create an empty arraylist Object with default initial capacity "10".
b. Once ArrayList Object reaches its max Capacity a new ArrayList Object will be Created with
c. NewCapatiy = (CurrentCapacity*$\frac{3}{4}$)+1;

2. ArrayList l = new ArrayList(int initialCapacity);
    a. Create an Empty ArrayList Object with Specified InitialCapacity.
3. ArrayList l = new ArrayList(Collection c);
    a. Create an equality ArrayList Object for the given COllection.

```
import java.util.*;

public class demo {

    public static void main( String args[] ) {
     ArrayList al = new ArrayList();
     al.add("Chandu");
     al.add(10);
     al.add(null);
     al.add(10);

     System.out.println(al);
     al.remove(3);
     System.out.println(al);
     System.out.println(al.get(1));
    }
}
```

```
Console  Problems  Debug Shell  Servers
<terminated> Demo [Java Application] G:\Eclipse Setup\eclipse\plugins\org.eclipse.ju
[Chandu, 10, null, 10]
[Chandu, 10, null]
10
```

Usually we can use Collection to hold and transfer objects from one location to Other Location [Container], To prove Support for this Requirement every Collection Class by default implements Serilizable and cloneable Interfaces.

ArrayList and Vector Classes.

- ArrayList and Vector classes Implements randomAccess interface, so that any random element we can access with the same speed

Random Access:

- Random access interface present in java.util package and it doesn't have any methods; it is a marker interface, where required ability will be provided automatically by the JVM.

```
import java.util.*;
import java.io.IOException;
import java.io.Serializable;
import java.lang.reflect.Method;
public class demo {


    public static void main( String args[] ) {
     ArrayList l1 = new ArrayList();
     LinkedList l2 = new LinkedList();

     System.out.println(l1 instanceof Serializable);
     System.out.println(l2 instanceof Serializable);
     System.out.println(l1 instanceof Cloneable);
     System.out.println(l2 instanceof Cloneable);
     System.out.println(l1 instanceof RandomAccess);
     System.out.println(l2 instanceof RandomAccess);
    }
}
```

```
Console  Problems  Debug Shell  Servers
<terminated> Demo [Java Application] G:\Eclipse Setup\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.j
true
true
true
true
true
false
```

- ArrayList is the best choice if our frequent Operation is retrieval Operation because arrayList implements RandomAccess interface.
- Array is the worst Choice if our frequency Operation is insertion and deletion in the middle.

ArrayList	Vector
EveryMethod present in ArrayList is non-Syncronized	Every method present in Vector is Synchronized
At a time Multiple thread are allowed to Operate on ArrayList Object and hence it is not thread safe	At a time only one thread is allowed to Operate on vector Object, Hence it is thread Safe.
Relatively Performance is High	Performance is low.
1.2v non-legacy	1.0v Legacy classes

Que: How to get a Synchronized version of ArrayList Object ?

- ➔ By default ArrayList is non-synchronized but we can get a Synchronized version of ArrayList Object by Using SynchronizedList() of the Collections class.
- ➔ Public static List SynchronizedList(list l);
- ➔ EX:
- ➔ ArrayList l = new ArrayList(); // this is the non-synchronized .
- ➔ List l1 = Collections.SynchronizedList(l) // Output is SynchronizedList.
- ➔ Similarly we can get a Synchronized version of Set and Map Object by using the Following methods of Collections Class.
- ➔ Public static Set SynchronizedLSet(Set s);Collections.SynchronizedLSet(l)
- ➔ Public static Map SynchronizedLMap(Map m)Collections.SynchronizedMap()

LinkedList

- Underlying DataStructure is a doubly Linked list.
- Insertion order is preserved.
- Duplicate Objects are Allowed.
- Heterogeneous Objects are allowed.
- Null insertion is possible.
- LinkedList Implements Serializable and cloneable interfaces, but not random access.

- LinkedList is the best choice for if our frequent Operation is insertion or deletion in the middle.
- LinkedList is the Worst Choice for if our frequent Operation is retrieval.
- Constructor:
    - LinkedList l = new linkedList();
        - Create a empty linked list Object
    - LinkedList l = new linkedList(collection c);
        - Creates an equivalent linkedlist Object for a given Collection.
- LinkedList class Specific methods:
    - Usually we can use LinkedList to develop Stacks and Queues to provide support for this requirement, LinkedList Class defines the following specific methods
    - Void addFirst(Object o);
    - Void addLast(Object o);
    - Object getFirst();
    - Object getLast();
    - Object removeFirst();
    - Object removeLast();

```java
import java.util.*;
public class demo {
    public static void main( String args[] ) {

     LinkedList l1 = new LinkedList();

     l1.add("chandu");     // chandu
     l1.add(10);           // chandu 10
     l1.add(null);         // chandu 10 null
     l1.add("chandu");     // chandu 10 null chandu
     l1.set(2, "Java");    // chandu 10 Java chandu
     l1.add(0,"String");   // String chandu 10 Java chandu
     System.out.println(l1);
     System.out.println(l1.getFirst());
     l1.addFirst("FirstString");  // FirstString String chandu 10 Java chandu
     l1.removeLast();             // FirstString String chandu 10 Java
     System.out.println(l1);
    }
}
```

```
Console  Problems  Debug Shell  Servers
<terminated> Demo [Java Application] G:\Eclipse Setup\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.5.v2022
[String, chandu, 10, Java, chandu]
String
[FirstString, String, chandu, 10, Java]
```

ArrayList	Vector
Best for Retrival	Best for insertion or deletion at middle
insertion or deletion at Best	Worst for Retrival
Contigious Memory	Not Contigious memory

-

Vector:

- **The underlying data structure is a resizable array or Growable array.**
- **Insertion Order is Preserved.**
- **Duplicates are allowed.**
- **Heterogeneous Objects are allowed.**
- **Null Insertion is Possible**
- **It implements Serilizable, cloneable and RandomAccess Interfaces.**
- **Every Method present in the Vector is Synchronized and hence Vector Object is Thread Safe.**
- ***COnstructor:***
  - **Vector v=new Vector();**
    - **Create a empty vector Object with default initial capacity "10", once Vector reaches its max-capacity then a new Vector Object will be created with [CurrentCapacity*2].**
  - **Vector v = new Vector(int initialCapacity);**
  - **Vector v = new Vector(int initialCapacity, int increment);**
    - **Ex: [1000,5] —> 1005,1010,1015...etc.**
  - **Vector v = new Vector(Collection c);**
- **Vector Specific methods.**
  - **To add Objects:**
    - **add(Object o) —- c**
    - **add(int index,Object o) —- L**
    - **addElement(Object o) —--v**
  - **To remove Objects**

- remove(Object o); —----C
- remove(int index);- —---L
- removeElement(Object o); —---V
- removeElementAt(int index) —---V
- clear() —---C
- removeAllElements() —-----V

- To get Objects
  - Object get(index); —--C
  - Object elementAt(index) —--V
  - Object firstElement() —----V
  - Object lastElement() —-----V
- Int size()
- int capacity();
- Enumeration elements();

Program:

```
2  import java.util.*;
3  public class demo {
4      public static void main( String args[] ) {
5          Vector v= new Vector();
6          System.out.println(v.capacity());
7          for(int i=0;i<10;i++)
8          {
9              v.addElement(i);
10         }
11         System.out.println(v.capacity());
12         v.add("chandu");
13         System.out.println(v.capacity());
14         System.out.println(v);
15
16
17
```

Console × Problems Debug Shell Servers

<terminated> Demo [Java Application] G:\Eclipse Setup\eclipse\plugins\org.eclipse.justj

```
10
10
20
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, chandu]
```

-

```java
import java.util.*;
public class demo {
    public static void main( String args[] ) {
        Vector v= new Vector(24,5);
        System.out.println(v.capacity());
        for(int i=0;i<24;i++)
        {
            v.addElement(i);
        }
        System.out.println(v.capacity());
        v.add("chandu");
        System.out.println(v.capacity());
        System.out.println(v);


    }
}
```

Console × Problems Debug Shell Servers

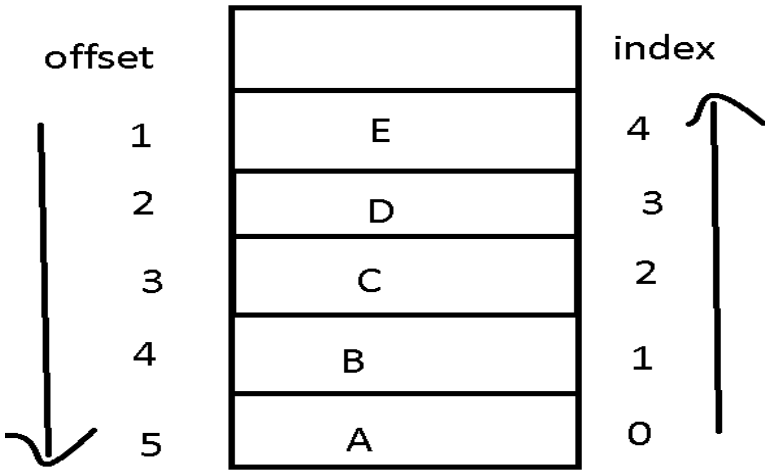
<terminated> Demo [Java Application] G:\Eclipse Setup\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.5.v20221102-0933\jre\bin\j

```
24
24
29
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, chandu]
```

-

Stack:

- It is the child class of vector
- It is a Specially Designed Class for last in firstOut Order[LIFO].
- Stack [Constructor]:
  - Stack s = new Stack();
- Stack Methods:
  - Object push(Object o);
    - To insert an Object into the Stack;
  - Object pop();
    - To remove and return top of the stack;
  - Object peek();
    - To return top of the stack
  - boolean empty()
    - Returns true if the stack is empty
  - int search(Object o)
    - Returns offset if the element is available otherwise returns -1;
-



s.search("A") --->5

s.search("Z") ---->-1

```
2  import java.util.*;
3  public class demo {
4      public static void main( String args[] ) {
5          Stack s =new Stack();
6          s.push("A");
7          s.push("b");
8          s.push("c");
9          System.out.println(s.search("A"));
10         s.pop();
11         System.out.println(s);
12
13     }
14 }
```

```
Console × Problems Debug Shell Servers
<terminated> Demo [Java Application] G:\Eclipse Setup\eclipse\plugins\org.eclipse.ju
3
[A, b]
```

3 Cursors of JAVA

1. **Enumeration**
2. **Iterator**
3. **ListIterator**

- **Enumeration:**
  - **We can use Enumeration to get Objects one by one from legacy Collection Objects.**

- We Can Create Enumerator Object by using elements() of Vector Class
- Eg : Enumeration e = v.elements();
- METHODS:
  - Public boolean hasMoreElements();
  - Public Object nextElements();

```
2  import java.util.*;
3  public class demo {
4      public static void main( String args[] ) {
5          Vector v=new Vector();
6          v.addElement("chandu");
7          v.add(10);
8          v.add(10.5);
9
10         Enumeration e = v.elements();
11         while(e.hasMoreElements())
12         {
13             System.out.println(e.nextElement());
14         }
15
16     }
17 }
```

```
Console × Problems Debug Shell Servers
<terminated> Demo [Java Application] G:\Eclipse Setup\eclipse\plugins\org.eclipse.justj.ope
chandu
10
10.5
```

-
-

Limitations of enumeration:

- We can apply the Enumeration concept only for legacy classes, and it is not a universal Cursor.
- By using Enumeration we can get only read access and we cannot perform remove operations.
- To Overcome the above limitations we should go for an iterator.

Iterator:

- We can apply the iterator concept for any Collection Object and hence it is a Universal Cursor.
- By Using iterator we can perform both read and remove operations.
  - Public Iterator iterator
  - Iterator itr = e.iterator();
- We can create an iterator Object by using iterator() of Collection Interface.

Methods:

- ☐ **Public boolean hasNext();**
- ☐ **Public Object next();**
- ☐ **Public void remove();**

```java
import java.util.*;
public class IteratorDemo {
    public static void main(String args[])
    {
        ArrayList l = new ArrayList();
        for(int i=0;i<10;i++)
        {
            l.add(i);

        }
        Iterator itr = l.iterator();
        while(itr.hasNext())
        {
            Integer i = (Integer)itr.next();
            if(i%2==0)
            {
                System.out.println(i);

            }
            else {
                itr.remove();
            }
        }
        System.out.println(l);

    }
}
```

Markers | Properties | Servers | Data Source Explorer | Snippets | Console

```
<terminated> IteratorDemo [Java Application] C:\eclipse\plugins\org.eclipse.justj.openjdk
0
2
4
6
8
[0, 2, 4, 6, 8]
```

Limitations Of Iterator:

- **By Using Enumeration and iterator we can only move toward forward direction, we cannot move towards backward direction these are single direction cursor but not bidirectional cursor.**
- **By Using iterator we can perform only read and remove operations but we cannot perform add and replacement operations.**
- **To Overcome above Limitations we should go for ListIterator.**

ListIterator.

- **By Using ListIterator we can Move either forward or backward direction and hence it is a bidirectional Cursor.**

- **By using ListIterator we can perform read and remove and add and replace OPerational.**
    - **Public ListIterator listIterator();**
    - **We can create a listIterator by using listIterator() of list Interface.**
    - **ListIterator ltr = l.listIterator();**

```java
import java.util.*;
public class IteratorDemo {
    public static void main(String args[])
    {
        ArrayList l = new ArrayList();
        for(int i=0;i<10;i++)
        {
            l.add(i);

        }
        ListIterator itr = l.listIterator();
        while(itr.hasNext())
        {
            Integer i = (Integer)itr.next();
            if(i%2==0)
            {
                System.out.println(i);

            }
            else {
                itr.remove();
            }
        }
        while(itr.hasPrevious())
        {
            Integer i = (Integer)itr.previous();
            if(i%2==0)
            {
                System.out.println(i);

            }
            else {
                itr.remove();
            }
        }
        System.out.println(l);

    }
}
```

Markers Properties Servers Data Source Explorer Sr

<terminated> IteratorDemo [Java Application] C:\eclipse\plugins\

```
0
2
4
6
8
8
6
4
2
0
[0, 2, 4, 6, 8]
```

- **ListIterator is the child interface of iterator and all methods presented in iterator by the default available with listIterator.**

- ListIterator defines the following 9 methods.
  - Public boolean hasNext();
  - Public Object next();
  - Public void nextIndex();
  -
  - Public boolean hasPrevious();
  - Public Object Previous();
  - Public void previousIndex();
  -
  - Public void remove();
  - Public void add(Object o);
  - Public void set(Object o);
  -

IteratorDemo.java

```
1  import java.util.*;
2  public class IteratorDemo {
3      public static void main(String args[])
4      {
5          ArrayList l = new ArrayList();
6          for(int i=0;i<10;i++)
7          {
8              l.add(i);
9
10         }
11         ListIterator itr = l.listIterator();
12         while(itr.hasNext())
13         {
14             Integer i = (Integer)itr.next();
15             if(i%2==0)
16             {
17                 itr.add("even:");
18
19             }
20             else {
21                 itr.set("odd");
22             }
23         }
24
25         System.out.println(l);
26
27     }
28 }
```

Markers | Properties | Servers | Data Source Explorer | Snippets | Console

<terminated> IteratorDemo [Java Application] C:\eclipse\plugins\org.eclipse.justj.openjdk.

[0, even:, odd, 2, even:, odd, 4, even:, odd, 6, even:, odd, 8, even:, odd]

- The Most Powerful cursor is ListIterator but its Limitations is applicable only for list Objects.

Properties	Enumeration	Iterator	ListIterator
where we can apply	Only for legacy class	Any Collection Object	Only for List Objects
Is It Legacy	Yes(1.0v)	No(1.2v)	No(1.2v)
Movements	Single Direction	Single Direction	bi-directional
Allowed Operation	Read	Read and Remove	Read,add,remove replace
How We can get	By Using elements() of Vector class	By Using iterator() using collection (I)	By using ListIterator method of list
Methods:	hasMoreElements() nextElements()	hasNext() next(), remove();	9 methods

Note:

- **Interface cannot create an Object enumeration is not an Object but is an implementation class Object is Created**

Enumeration:

- ☐ **Enumeration e = v.elements();**
- ☐ **System.out.println(e.getClass().getName());**

Iterator:

- ☐ **Iterator = v.iterator();**
- ☐ **System.out.println(e.getClass().getName());**

ListIterator:

- ☐ **ListIterator e = v.listIterator();**
- ☐ **System.out.println(e.getClass().getName());**

Output:

- **java.util.vector$1**
- **java.util.vector$Itr**
- **java.util.vector$ListItr**

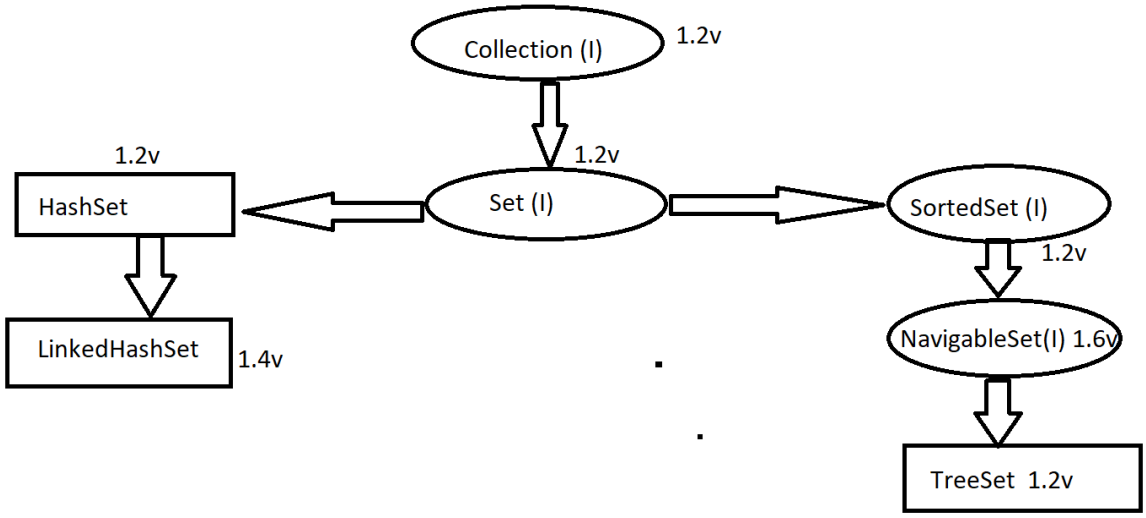
```java
import java.util.*;
public class IteratorDemo {
    public static void main(String args[])
    {
        Vector v= new Vector();
        v.add(10);
        v.add(20);
        v.add(30);

        Enumeration e = v.elements();
        Iterator i = v.iterator();
        ListIterator itr = v.listIterator();

        System.out.println(e.getClass().getName());
        System.out.println(i.getClass().getName());
        System.out.println(itr.getClass().getName());
    }
}
```

Markers Properties Servers Data Source Explorer Snippets Cons

<terminated> IteratorDemo [Java Application] C:\eclipse\plugins\org.eclipse.justj.

```
java.util.Vector$1
java.util.Vector$Itr
java.util.Vector$ListItr
```

SET:



- **Set is child interface of collection**
- **If we want to represent a group of individuals Objects as a Single entity where duplicates are not allowed and insertion order not Preserved.**

- Set interface does not contain any new methods, we have to use only Collection interface methods.

HashSet 👍

- ☐ The Underlying DataStructure is Hashtable.
- ☐ Duplicated objects are not allowed.
- ☐ Insertion Order is not preserved and it is based on the hashcode of Object.
- ☐ Null Insertion is possible (only Once)
- ☐ Heterogeneous Objects are allowed.
- ☐ Implements serializable and cloneable but not randomAccess.
- ☐ Hashset is used Frequently Operation is search Operation.

Note:In hashset() duplicates are not allowed. If we are trying to insert duplicated then they won't get any compiler or runtime errors and add() simply returns false.

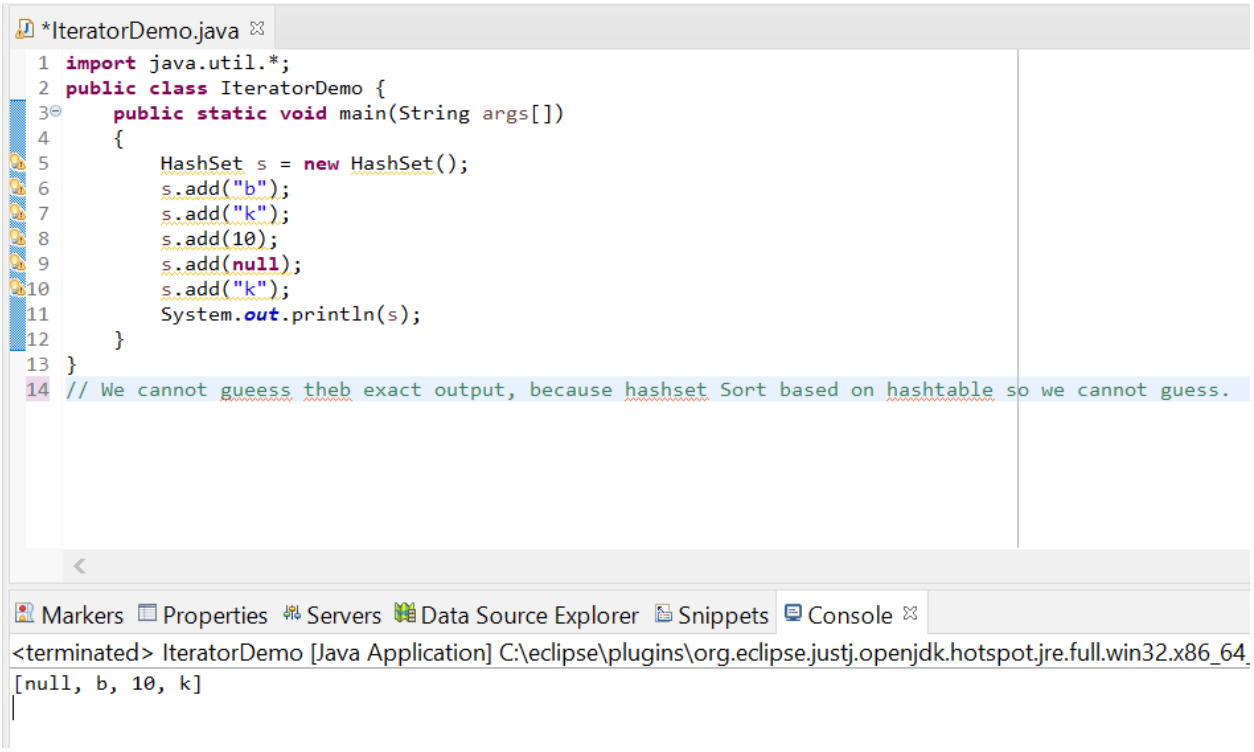
```
Hashset s = new HashSet();
s.add("a");
s.add("a");
```

Constructors:

- HashSet s = new HashSet();
  - Create an empty Hashset with default capacity of 16 ans default fill-ration is 0.75.
- HashSet s = new HashSet(int InitialCapacity)
  - Creates an empty HashSet Object with specified initial-Capacity and default fill-ratio.
- HashSet s = new HashSet(int initialcapacity, float fill-ratio);
  - Creates empty HashSet Object with SPecified with intial-Capacity and With Specified fill-ratio.
- HashSet s = new HashSet(Collection c);

Fill-Ratio:

- **After filling how much ration a new HashSet Object will be created, this ration is called fill-ratio or load-factor.**



fill -ration 0.75 means after filling 75% ratio a new Hashset Object will be created.

LinkedHashSet:

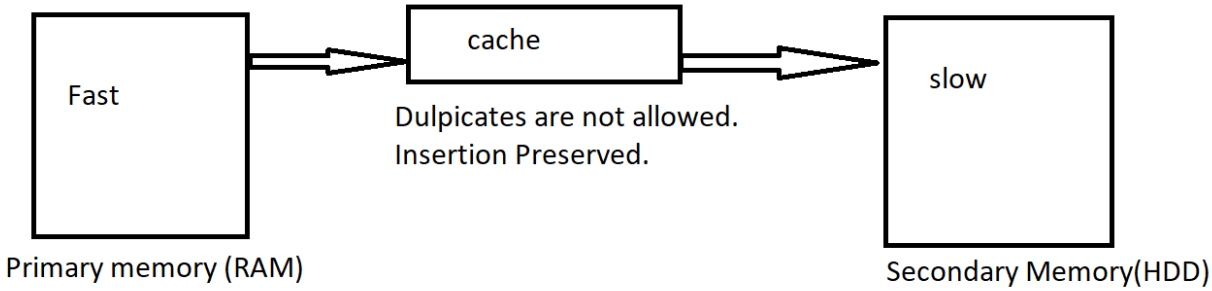
- **It is the child class of HashSet, it is exactly the same as hashset including constructors and methods except the following differences.**

HashSet	LinkedHashSet
The Underlying DataStructure is HashTable Insertion Order is not preserved. 1.2v	The Underlying DataStructure is LinkedList+HashTable Insertion Order is Preserved. 1.4v

-
- —----------

-

- If we replace HashSet with LinkedHashSet then we will get output in the insertion preserved. There is not much difference other than that.



- Note:
  - In general we can use LinkedHashSet to develop cache based applications where duplicates are not allowed and insertion Order is Preserved.

SortedSet:

- SortedSet is the child interface of Set interface. If we want to represent a group of individual objects According to some Sorting Order without duplicates then we should go for SortedSet.
- SortedSet Interface has some methods.
  - Object first()
  - Object last();
  - SortedSet headSet(Object 0);
  - SOrtedSet tailset(Object o);
  - SortedSet subSet(Object o1, Object o2);
  - Comparator Comparator();
- Default sorting Order:
  - Number — Ascending Order
  - String — Alphabetic Order

TreeSet:

- The Underlying DataStructure is a Balanced tree.
- Duplicated Objects are not Allowed.
- Heterogeneous Objects are not allowed otherwise we will get the runtime exception saying classCastexception.

- Null Insertion is Possible only once.
- Implements Serializable and Cloneable , but not randomAccess.
- All the Objects will be inserted based on SOme Sorting algorithm , it may be default natural Sorting order or Customized sorting Order.
- Constructors:
    - TreeSet t= new TreeSet();
        - Default natural Sorting Order.
        - Creates an Empty TreeSet Object where the elements will be inserted according to default natural sorting
    - TreeSet t = new TreeSet(Comparator c);
        - Creates an empty treeSet Object where the elements will be inserted according to Customized sorting Specified by Comparator Object.
    - TreeSet t = new TreeSet(Collection c)
    - TressSet ts = new TreeSet(SOrtedSet);
    -

```
1  import java.util.*;
2
3  public class TreeSetDemo {
4      public static void main(String args[])
5      {
6          TreeSet t = new TreeSet();
7          t.add("A");
8          t.add("b");
9          t.add("C");
10         t.add("B");
11         t.add("Z");
12         t.add(new Integer(10));
13         // java.lang.ClassCastException: class java.lang.String cannot be cast to class java.lang.Inte
14         t.add(null); // java.lang.NullPointerException
15
16         t.add("A");
17         t.add("A");
18
19     }
20 }
```

Markers | Properties | Servers | Data Source Explorer | Snippets | Console

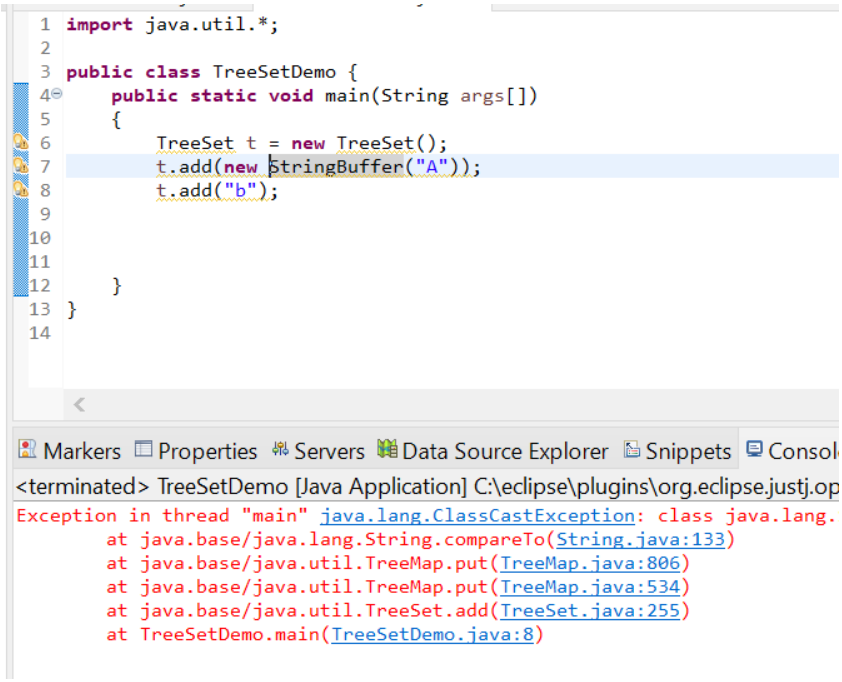
```
<terminated> TreeSetDemo [Java Application] C:\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_16.0
Exception in thread "main" java.lang.NullPointerException
        at java.base/java.util.Objects.requireNonNull(Objects.java:208)
        at java.base/java.util.TreeMap.put(TreeMap.java:801)
        at java.base/java.util.TreeMap.put(TreeMap.java:534)
        at java.base/java.util.TreeSet.add(TreeSet.java:255)
        at TreeSetDemo.main(TreeSetDemo.java:14)
```

Null Acceptance:

- FOr NonEmpty TreeSet if we are trying to insert Null then we will get NullPointerException.
- For Empty Treeset as the First element Null is Allowed, but after insertion Null if we try to insert any Object then we will get the NullPointerException.

Note:

- Until 1.6v Null is Allowed as the first element to the Empty Treeset.
- But from 1.7v onwards Null is not applicable even as the first element.
- "Null" Such type of Story not applicable for treeset from 1.7v;
- No Heterogeneous Objects are created else -> ClassCastException.
-



If we take default Sorting Order Treeset
Objects should be as below.

- Homogeneous
- Comparable
- Objects as above else ClassCastException.
- All Wrapper classes are comparable.

An Object is said to be comparable if and only if Corresponding class implements comparable interface.String CLass and all wrapper classes already implemented are comparable, but StringBuffer class doesn't implement comparable interface, hence we got classCastException in the above example.
Comparable (I):

- It is present in Java.lang package
- It contains only one method —> compareTo();
- Public int compareTo(Object obj)
    - Obj1.compareTo(obj2);

- **Return -1 if obj1 has to come before obj2**
- **Return 1 if obj1 has to come after obj2**
- **Return 0 if equal.**

```
public class comparetoDemo {
    public static void main(String args[])
    {
//      TreeSet t = new TreeSet();
        System.out.println("A".compareTo("Z"));
        System.out.println("Z".compareTo("A"));
        System.out.println("A".compareTo("A"));

    }
}
```

Markers Properties Servers Data Source Explorer Snipp

<terminated> comparetoDemo [Java Application] C:\eclipse\plugins\

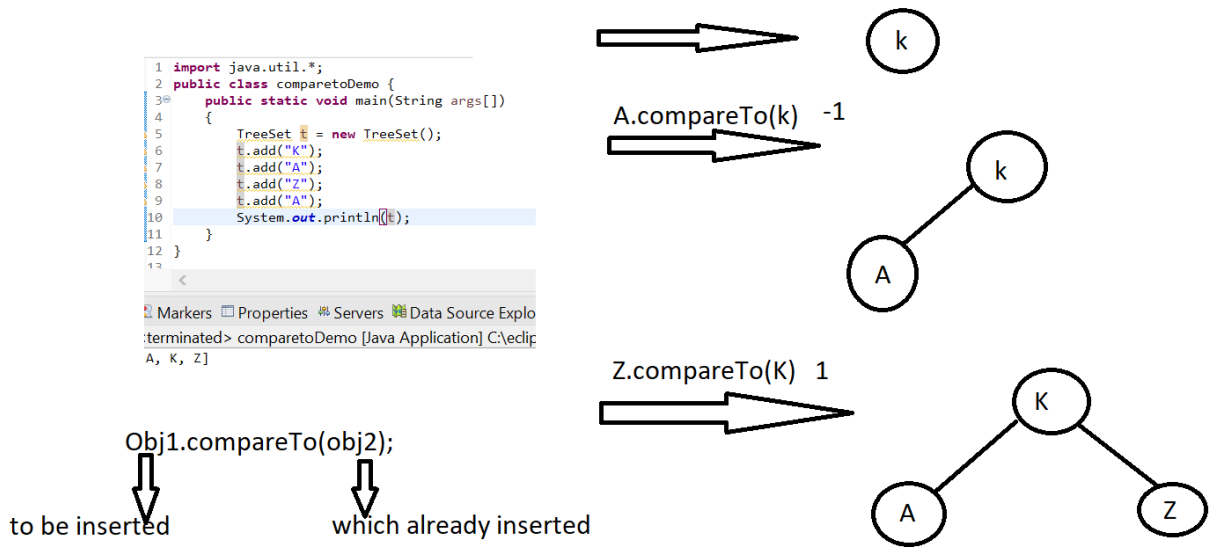
```
-25
25
0
```

```
import java.util.*;
public class comparetoDemo {
    public static void main(String args[])
    {
        TreeSet t = new TreeSet();
        t.add("K");
        t.add("A");
        t.add("Z");
        t.add("A");
        System.out.println(t);
    }
}
```

Markers Properties Servers Data Source Explo

terminated> comparetoDemo [Java Application] C:\eclip

```
A, K, Z]
```



If we are depending on Default natural sorting Order then while adding Objects into the treeset, JVM will call compareTo().

If default natural Sorting is not available or not satisfied with default natural sorting Order then we can go for Customized Sorting by using comparator.

COmparable —> Default Sorting Order.(java.lang package)

- [compareTo()]

Comparator(I) —> Customized Sorting Order.(java.util package)

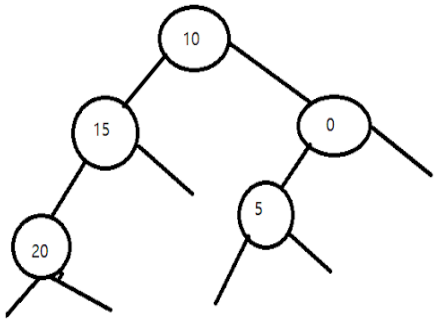
- [compare(),equals()]
- Public int compare(object obj1,Object obj2)
  - Return -1 if obj1 has to come before obj2
  - Return 1 if obj1 has to come after obj2
  - Return 0 if equal.
- Public boolean equals(obj);

Whenever we are implementing Comparator interface compulsory we should provide implementation only for compare method and we are not required to provide implementation for equals(), because it is already available to our class from Object class through inheritance.

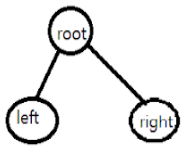
WRITE A PROGRAM TO INSERT INTEGER OBJECTS INTO TREESET, WHERE SORTING ORDER IS DESCENDING ORDER.

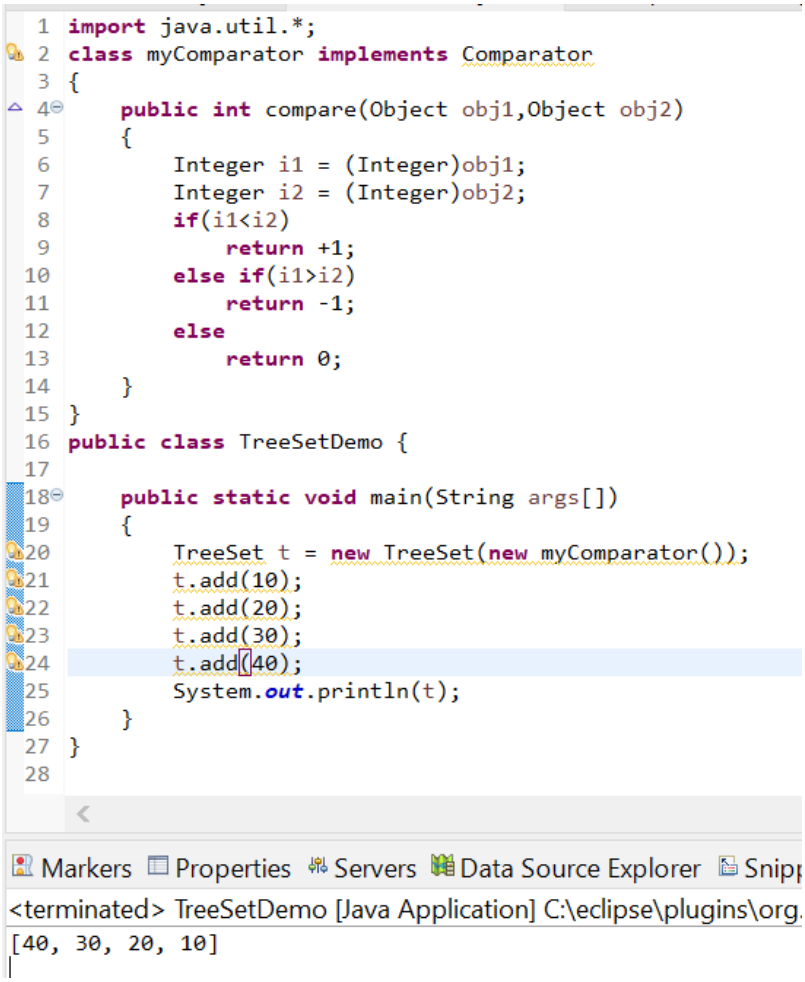
```
class myComparator implements Comparator
{
      public int compare(Object obj1,Object obj2)
      {
            Integer i1 = (Integer)obj1;
            Integer i2 = (Integer)obj2;
            if(i1<i2)
                  return +1;
            else if(i1>i2)
                  return -1;
            else
                  return 0;
      }
}
```

```
TreeSet t = new TreeSet(new mynewcompare);
t.add(10);

t.add(0);    compare(0,10)     i1<i2 ---> 1

t.add(15);   compare(15,10)    i1>i2 --->-1

t.add(5);    compare(5,10);    i1<i2 ---->1;
             compare(5,0)      i1>i2  ----> -1

t.add(20);   compare(20,10);   i1>i2 ---> -1
             compare(20,15)    i1>i2 ---> -1

t.add(20);   compare(20,10);   i1>i2 ---> -1
             compare(20,15)    i1>i2 ---> -1
             compare(20,20)    i1== i2 ---0
```



20,15,10,5,0 left,root,right (Example)





```
import java.util.*;
class myComparator implements Comparator
{
    public int compare(Object obj1,Object obj2)
    {
        Integer i1 = (Integer)obj1;
        Integer i2 = (Integer)obj2;
        if(i1<i2)
            return +1;
        else if(i1>i2)
            return -1;
        else
            return 0;
    }
}
public class TreeSetDemo {

    public static void main(String args[])
    {
        TreeSet t = new TreeSet(new myComparator());
        t.add(10);
        t.add(20);
        t.add(30);
        t.add(40);
        System.out.println(t);
    }
}
```


At line1 if we are not passing comparator Object then internally JVM will call compareTo() which is meant for default natural sorting order, In this case Output is [10,20,30,40]

At line1 if we are passing a comparator Object then JVM will call compare() which is meant for customized sorting order, In this case Output is [40,30,20,10].

Various Possible implementations of compare():

- **return i1.compareTo(i2); //Ascending order**
- **return -i2.compareTo(i1); //Ascending order**
- **return -i1.compareTo(i2); //Descending order**
- **return i2.compareTo(i1); //Descending order**

- **Return +1; [10,0,15,5,20,20] same as input // our comparator not telling it is a duplicate.**
- **Return -1; [20,20,5,15,0,15] same as input // our comparator not telling it is a duplicate.**
- **Return 0 [only first element will be inserted and remaining all are duplicate].**

WRITE A PROGRAM TO INSERT sTRING OBJECTS INTO TREESET, WHERE SORTING ORDER IS DESCENDING ORDER BY CUSTOMIZED SORTING oRDER.

```
1  import java.util.*;
2  class myComparator implements Comparator
3  {
4      public int compare(Object obj1,Object obj2)
5      {
6          String s1 = obj1.toString();
7          String s2 = (String)obj2;
8
9          return s2.compareTo(s1);
10     }
11 }
12 public class TreeSetDemo {
13
14     public static void main(String args[])
15     {
16         TreeSet t = new TreeSet(new myComparator());
17         t.add("Apple");
18         t.add("Mango");
19         t.add("Banana");
20         t.add("Orange");
21         t.add("Zeber");
22         System.out.println(t);
23     }
24 }
25
```

Markers Properties Servers Data Source Explorer Snip

<terminated> TreeSetDemo [Java Application] C:\eclipse\plugins\org

```
[Zeber, Orange, Mango, Banana, Apple]
```

WRITE A PROGRAM TO INSERT STRINGbUFFER OBJECTS INTO TREESET, WHERE SORTING ORDER IS ALPHABETIC ORDER.

```java
import java.util.*;
class myComparator implements Comparator
{
    public int compare(Object obj1,Object obj2)
    {
        String s1 = obj1.toString();
        String s2 = obj2.toString();

        return s2.compareTo(s1);
    }
}
public class TreeSetDemo {

    public static void main(String args[])
    {
        TreeSet t = new TreeSet(new myComparator());
        t.add(new StringBuffer("Apple"));
        t.add(new StringBuffer("Mango"));
        t.add(new StringBuffer("Banana"));
        t.add(new StringBuffer("Orange"));
        t.add(new StringBuffer("Zeber"));
        System.out.println(t);
    }
}
```

```
Markers Properties Servers Data Source Explorer Snippe
<terminated> TreeSetDemo [Java Application] C:\eclipse\plugins\org.ec
[Zeber, Orange, Mango, Banana, Apple]
```

Note:

- **If you are depending on default Natural Sorting Order compulsory Objects should be homogeneous and comparable otherwise we will get a runtimeException saying ClassCastException.**
- **If we are defining our own sorting by comparator then objects need not be comparable and homogeneous, i.e we can add heterogeneous and non-comparable Objects also.**

WRITE A PROGRAM TO INSERT STRINGbUFFER AND STRING OBJECTS INTO TREESET, WHERE SORTING ORDER IS BASED ON LENGTH OF STRING ORDER.

```java
import java.util.*;
class myComparator implements Comparator
{
    public int compare(Object obj1,Object obj2)
    {
        String s1 = obj1.toString();
        String s2 = obj2.toString();

        int l1 = s1.length();
        int l2 = s2.length();

        if(l1<l2)
            return -1;
        else if(l1>l2)
            return 1;
        else
            return s1.compareTo(s2);
    }
}
public class TreeSetDemo {

    public static void main(String args[])
    {
        TreeSet t = new TreeSet(new myComparator());
        t.add("chandu");
```

Markers Properties Servers Data Source Explorer Snippets Conso

<terminated> TreeSetDemo [Java Application] C:\eclipse\plugins\org.eclipse.justj.op

```
[AAA, Apple, Mango, Zeber, Banana, Orange, chandu, ABCDEFGHIJKLM]
```

Note:

- **For Our Own CLass Like Employee or Student class, who is writing the class he is responsible to define the Default natural sorting Orderby implementing Comparable interface. The Person who is using Our class, if he is not satisfied with default Natural sorting Order then he can define his own sorting by using comparator.**

```java
import java.util.TreeSet;

class employeede implements Comparable
{
	String name;
	int empid;
	int years;
	int grade;

	employeede(String name,int empid,int years,int grade)
	{
		this.name = name;
		this.empid = empid;
		this.years = years;
		this.grade = grade;
	}
	public String toString()
	{
		return name+" "+empid;
	}
	public int compareTo(Object obj)
	{
		int empid1 = this.empid;
		employeede e = (employeede)obj;
		int empid2 = e.empid;

		if(empid1<empid2)
			return 1;
		else if(empid1>empid2)
			return -1;
		else
			return 0;
	}

}
```

```java
public class employee {

	public static void main(String[] args) {
		employeede e1= new employeede("Chandu",101,3,10);
		employeede e2= new employeede("Madhu",301,1,2);
		employeede e3= new employeede("Kiran",202,0,5);
		//[employeede@2401f4c3, employeede@7637f22, employeede@4926097b]
		TreeSet t = new TreeSet();
		t.add(e1);
		t.add(e2);
		t.add(e3);

		System.out.println(t);

	}

}
```

OUTPUT:

```
[Madhu 301, Kiran 202, Chandu 101]
```

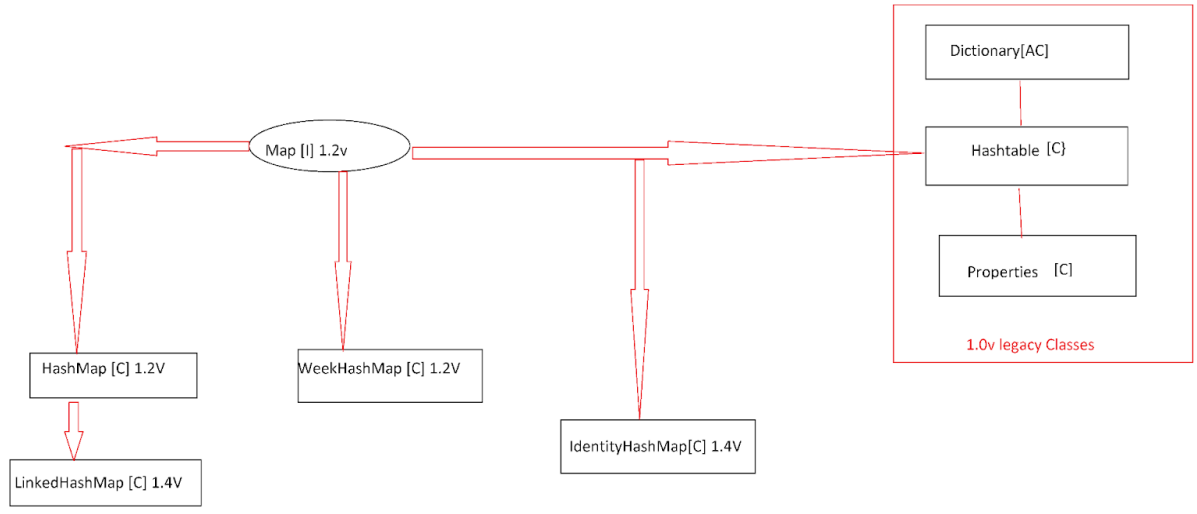
```java
//import java.util.TreeSet;
 import java.util.*;
class employeede implements Comparable
{
	String name;
	int empid;


	employeede(String name,int empid)
	{
		this.name = name;
		this.empid = empid;

	}
	public String toString()
	{
		return name+" "+empid;
	}
	public int compareTo(Object obj)
	{
		int empid1 = this.empid;
		employeede e = (employeede)obj;
		int empid2 = e.empid;

		if(empid1<empid2)
			return -1;
		else if(empid1>empid2)
			return 1;
		else
			return 0;
	}

}
```

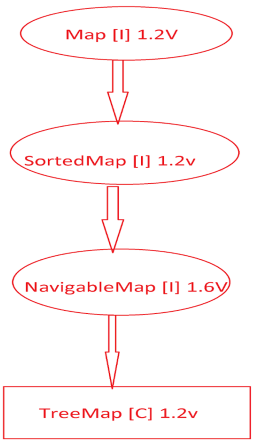
```java
public class employee {

	public static void main(String[] args) {
		employeede e1= new employeede("Chandu",101);
		employeede e2= new employeede("Madhu",301);
		employeede e3= new employeede("Kiran",202);
		//[employeede@2401f4c3, employeede@7637f22, employeede@4926097b]
		TreeSet t = new TreeSet();
		t.add(e1);
		t.add(e2);
		t.add(e3);
		System.out.println(t);
		TreeSet t1 = new TreeSet(new myComparator());
		t1.add(e1);
		t1.add(e2);
		t1.add(e3);
		System.out.println(t1);
	}

}


class myComparator implements Comparator
{
	public int compare(Object obj1, Object obj2)
	{
		employeede e1 = (employeede)obj1;
		employeede e2 = (employeede)obj2;

		String name1 = e1.name;
		String name2 = e2.name;

		return name1.compareTo(name2);
	}
}
```

Comparable	Comparator
Default Natural SOrting Order.	Customized Sorting Order
Presents in java.lang package	Presents in java.util package
Only One Method --> compareTo()	Contains two methods--> compare(), equal();
All Wrapper Classes and String Class already usesfmplements Comparable interface.	Collator and RuleBasedCollator classes

Properties	HashSet	LinkedHashSet	TreeSet
UnderLying Data Structure	HashTable	LinkedList+HashTable	Balanced Tree
Duplicates allowed	No	No	No
Insertion Order	No	Yes	No
Sorting Order	NA	NA	Applicable
Hetrogenious	Allowed	Allowed	Not Allowed
Null	Allowed	Allowed	Only empty Treeset as First element Null is allowed.

MAP:





- Map is not the Child interface of Collection.
- If we want to present a group of Objects as Key-Value pairs then we should go for a map.
    - Key Value
    - 101 madhu
    - 102 Chandu
    - 103 Samba
- Both Keys and Values are Objects only.
- Duplicates Keys are not allowed, Duplicates value is allowed.
- Each Key-Value pair is called entry, hence map is considered as a collection of entry Objects.

Map Interface methods:

- Object put(Object key, Object value)
    - To add one key-value pair to the map.
    - If the Key is already present then Old Value will be replaced with new value and return old value.
        - m.put(101, "durga"); → returns null
        - m.put(102, "Shiva"); → returns null
        - m.put(101, "chandu"); → returns durga
    - Void putAll(map m);
    - Object get(Object key);
        - Returns the value wrt key.
    - Object remove(Object key);

        - Remove the value wrt key.
    - boolean containsKey(Object key)
    - boolean ContainsValue(Object value);
    - Boolean isEmpty();
    - int size();
    - Void clear();
    - —---------- Collection views of map —----------
    - Set keySet();
    - Collection values();
    - Set entrySet();
-

Entry (I):

- A map is a group of key-value pairs and each key-value pair is called an entry.Hence Map is considered as a collection of entry Objects.Without existing Map-Object there is no chance of existing entry-Object, Hence Entry interface is define inside map interface.

Entry Specific methods() → we can apply only on entry Objects.

```
Interface map{
        Interface entry{
                Object getKey();
                Object getValue();
                Object SetValue(Object value);

        }
}
```

HashMap:

- The Underlying dataStructure is HashTable.
- Insertion Order is Not Preserved and it is based on Hashcode of Keys.
- Duplicates Keys are not allowed but values can be duplicated.
- Heterogeneous Objects are allowed for both key and value .

- **Null is allowed for key,(Only once)**
- **Null is allowed for values (Any number of times)**
- **HashMap implements Serilizavble and clonable interfaces but not random access.**
- **HashMap is the best choice if our frequent Opertion is Search Operation.**

Constructors :

- **HashMap m = new HashMap();**
    - **Creates an empty hashmap object with default initial capacity 16 and default fill-ratio -0.75;**
- **HashMap m = new HashMap(int intialCapacity)**
    - **Creates an empty hashmap object with initialcapacity and default fill-ratio -0.75;**
- **HashMap m = new HashMap(int intialCapacity,float fillratio)**
    - **Creates an empty hashmap object with initialcapacity and fill-ratio;**
- **HashMap m = new HashMap(Map m)**

```java
import java.util.*;
public class demo {
    public static void main( String args[] ) {
        HashMap m = new HashMap();
        m.put("Chandu", 100);
        m.put("Madhu", 500);
        m.put("Jyothi", 50);
        m.put("VaraLaxshmi", 600);
        m.put("RamaKrishna",200);
        System.out.println(m);//{RamaKrishna=200, Jyothi=50, Madhu=500, Chandu=100, VaraLaxshmi=600}
        System.out.println(m.put("Chandu", 800)); //100
        System.out.println(m);//{RamaKrishna=200, Jyothi=50, Madhu=500, Chandu=800, VaraLaxshmi=600}
        Set ks = m.keySet();
        Set es = m.entrySet();
        System.out.println(ks);//[RamaKrishna, Jyothi, Madhu, Chandu, VaraLaxshmi]
        System.out.println(es);// [RamaKrishna=200, Jyothi=50, Madhu=500, Chandu=800, VaraLaxshmi=600]
        Iterator i = es.iterator();
        while(i.hasNext())
        {
            Map.Entry e = (Map.Entry) i.next();
            System.out.println(e.getKey()+ "---"+ e.getValue()); //RamaKrishna---200 etc..
            if(e.getKey().equals("Madhu"))
            {
                e.setValue(10000);
            }
        }
        System.out.println(m); //{RamaKrishna=200, Jyothi=50, Madhu=10000, Chandu=800, VaraLaxshmi=600}
    }
}
```

Console × Problems Debug Shell Servers

<terminated> Demo [Java Application] G:\Eclipse Setup\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.5.v20221102-0933\jre\bin\ja

```
{RamaKrishna=200, Jyothi=50, Madhu=500, Chandu=100, VaraLaxshmi=600}
100
{RamaKrishna=200, Jyothi=50, Madhu=500, Chandu=800, VaraLaxshmi=600}
[RamaKrishna, Jyothi, Madhu, Chandu, VaraLaxshmi]
[RamaKrishna=200, Jyothi=50, Madhu=500, Chandu=800, VaraLaxshmi=600]
RamaKrishna---200
Jyothi---50
Madhu---500
Chandu---800
VaraLaxshmi---600
[RamaKrishna=200, Jyothi=50, Madhu=10000, Chandu=800, VaraLaxshmi=600]
```

HashMap	HashTable
Not Synchronized	Synchronized
Multiple Threads are allowed	Only Single thread is allowed
No Thread Safe	Thread Safe
Relative Performance is High	Performance is low
Null key, Null Value	Null Key and Values(Not USED), NPE
It is not Legacy class 1.2v	It is Legacy 1.0v

How to get Synchronized version of HashMap Object:

- **By default HashMap is a non-Synchronized version.but we can get synchronized version of HashMap by using SynchronizedMap() of COllections Class.**
- **HashMap m = new HashMap();**
- **Map m1 = Collections.SynchronizedMap(m);**

LinkedHashMap:

- **It is the child class of HashMap.**
- **It is exactly same as HashMap (methods and constructors) expect the following Differences**
-

HASHMAP	LinkedHashMap
Underlying DS is HashTable Insertion Order is not preserved and it is based on HashCode of keys Introduced in 1.2v	Underlying DS is LS+HT Insertion order is preserved Introduced in 1.4v

```java
package com.high.code1;

import java.util.*;
public class cac {
      public static void main( String args[] ) {
            LinkedHashMap m = new LinkedHashMap();
            m.put("Chandu", 100);
            m.put("Madhu", 500);
            m.put("Jyothi", 50);
            m.put("VaraLaxshmi", 600);
            m.put("RamaKrishna",200);
            System.out.println(m);//{RamaKrishna=200, Jyothi=50, Madhu=500, Chandu=100, VaraLaxshmi=600}
            System.out.println(m.put("Chandu", 800)); //100
            System.out.println(m);//{RamaKrishna=200, Jyothi=50, Madhu=500, Chandu=800, VaraLaxshmi=600}
            Set ks = m.keySet();
            Set es = m.entrySet();
            System.out.println(ks);//[RamaKrishna, Jyothi, Madhu, Chandu, VaraLaxshmi]
            System.out.println(es);// [RamaKrishna=200, Jyothi=50, Madhu=500, Chandu=800, VaraLaxshmi=600]
            Iterator i = es.iterator();
            while(i.hasNext())
            {
                Map.Entry e = (Map.Entry) i.next();
                System.out.println(e.getKey()+ "---"+ e.getValue()); //RamaKrishna---200 etc..
                if(e.getKey().equals("Madhu"))
                {
                    e.setValue(10000);
                }
            }
            System.out.println(m); //{RamaKrishna=200, Jyothi=50, Madhu=10000, Chandu=800, VaraLaxshmi=600}
       }
    }
```

Markers | Properties | Servers | Data Source Explorer | Snippets | Console

```
<terminated> cac [Java Application] C:\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64
{Chandu=100, Madhu=500, Jyothi=50, VaraLaxshmi=600, RamaKrishna=200}
100
{Chandu=800, Madhu=500, Jyothi=50, VaraLaxshmi=600, RamaKrishna=200}
[Chandu, Madhu, Jyothi, VaraLaxshmi, RamaKrishna]
[Chandu=800, Madhu=500, Jyothi=50, VaraLaxshmi=600, RamaKrishna=200]
Chandu---800
Madhu---500
Jyothi---50
VaraLaxshmi---600
RamaKrishna---200
{Chandu=800, Madhu=10000, Jyothi=50, VaraLaxshmi=600, RamaKrishna=200}
```

- **That is, the insertion Order is preserved.**

- LinkedHashSet and LinkedHashMap are commonly used for Cache based Applications.
- Difference Between == Operator and .equal().
- == is for address or Reference Comparison
- .equals() is for content comparison.
    - Integer i1 = new Integer(10);
    - Integer i2 = new Integer(10);
    - SOP(I1==I2); // False
    - SOP(T1.equals(I2)); //True.

IdentityHashMap :

- JVM Uses .equals() to define duplicate or not.
- HashMap m=new HashMap();
- It is exactly same as HashMap including methods and constructions, except the following difference.
- In the Case of normal HashMap JVM will Use .equals() to identify duplicated Keys. which is meant for content comparison.
- But in the Case of IdentityHashMap JVM will use "==" to identify duplicates keys, which is meant for reference comparison.

```java
package com.high.code1;

import java.util.*;
public class cac {
	public static void main( String args[] ) {
		HashMap m = new HashMap();
		Integer i1 = new Integer(10);
		Integer i2 = new Integer(10);
		m.put(i1, "Chandu");
		m.put(i2, "Madhu");
		System.out.println(m);
	}
}
```

Markers Properties Servers Data Source Explorer Snippets Console

```
<terminated> cac [Java Application] C:\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win
{10=Madhu}
```

Overridden above.

```java
import java.util.*;
public class cac {
	public static void main( String args[] ) {
		IdentityHashMap m = new IdentityHashMap();
		Integer i1 = new Integer(10);
		Integer i2 = new Integer(10);
		m.put(i1, "Chandu");
		m.put(i2, "Madhu");
		System.out.println(m);
	}
}
```

Markers Properties Servers Data Source Explorer Snippets Console

```
<terminated> cac [Java Application] C:\eclipse\plugins\org.eclipse.justj.openjdk.hotsp
{10=Chandu, 10=Madhu}
```

WeekHashMap:

- ➔ It is exactly the same as HashMap except the following differences.
- ➔ In the case of HashMap even though Object does not have any reference it is not eligible for Gc if it is associated with HashMap,i.e HashMap dominated GC.
- ➔ But in the case of weekHashMap if the Object doesn't contain any references it is eligible for GC even though Object Associated with weekHashMap, i.e GC dominated weekHashMap.
- ➔ Shown in below screenshots.

```
 3 import java.util.*;
 4 public class cac {
 5       public static void main( String args[] ) throws Exception {
 6             HashMap m = new HashMap();
 7             Temp t = new Temp();
 8             m.put(t, "chandu");
 9             System.out.println(m);
10             t = null;
11             System.gc();
12             Thread.sleep(5000);
13             System.out.println(m);
14         }
15 }
16 class Temp
17 {
18     public String toString()
19     {
20         return "temp";
21     }
22     public void finalize(){
23         System.out.println("Finialzed method called");
24     }
25 }
26
```

Markers | Properties | Servers | Data Source Explorer | Snippets | Console

```
<terminated> cac [Java Application] C:\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_1
{temp=chandu}
{temp=chandu}
```

➔

```
2
3  import java.util.*;
4  public class cac {
5        public static void main( String args[] ) throws Exception {
6              WeakHashMap m = new WeakHashMap();
7              Temp t = new Temp();
8              m.put(t, "chandu");
9              System.out.println(m);
10             t = null;
11             System.gc();
12             Thread.sleep(5000);
13             System.out.println(m);
14       }
15 }
16 class Temp
17 {
18     public String toString()
19     {
20         return "temp";
21     }
22     public void finalize(){
23         System.out.println("Finialzed method called");
24     }
25 }
26
```

Markers Properties Servers Data Source Explorer Snippets Console

```
<terminated> cac [Java Application] C:\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_1
{temp=chandu}
Finialzed method called
{}
```

➔

➔ In above Example1 Temp Object not eligible for Gc It is Associated with HashMap;

➔ In above Example2 Temp Object eligible for Gc It is Associated withWeak HashMap;

➔

SortedMap:

- **It is the child interface of Map.**
- **If we want to represent a group of key-value pairs according to some sorting Order of Keys then we should form a sorted Map.**
- **Sorting is Based On Key but Based On Value.**
- **SortedMap defines the following specific methods.**
    - **Object firstKey();**
    - **Object lastkey();**
    - **SortedMap headMap(Object Key)**
    - **SortedMap tailMap(Object Key)**
    - **SortedMap subMap(Object key1,Object key2)**
    - **Comparator comparator();**

TreeMap:

- **UnderLying DataStructure is RED-BLACK Tree.**
- **Insertion Order is not preserved, and it is based on some sorting order of keys.**
- **Duplicate keys are not allowed,but values can be duplicated.**
- **If you are depending on the default natural sorting order, then keys should be Homogeneous and comparable otherwise we will be a runtimeexception saying class cast exception.**
- **If we are defining our own sorting by comparator then keys need not be Homogeneous and Comparable. We can take heterogeneous , non-comparable also.**
- **Whether we are depending on default natural sorting order or Customized sorting Order there are no restrictions for values we can take heterogeneous and non-comparable also.**
- **Null Acceptance:**
    - **For non-empty treemap if we are trying to insert an entry with Null key then we will get Runtime exception saying NullPointerException**
    - **FOr empty TreeMap as the first entry with Null key is allowed. But after inserting that entry if we are trying to insert any other entry then we will get a runtimeException saying NullPointerException.**
- **Note:**
    - **The Above Null acceptance rule applicable until 1.6v only, from 1.7v onwards null is not allowed for Key.**
    - **But For Values we can use Null any Number of times.**

Constructors:

- **TreeMap t = new TreeMap();**
    - **Default Natural Sorting Order**
- **TreeMap t = new TreeMap(Comparator c);**
    - **Customized Sorting Order.**

- **TreeMap t = new TreeMap(map m);**
- **TreeMap t = new TreeMap(SortedMap m);**

```
2
3  import java.util.*;
4  public class cac {
5        public static void main( String args[] ) throws Exception {
6              TreeMap t =  new TreeMap();
7              t.put(100,"Madhu");
8              t.put(800, "Kiranb");
9              t.put(600, "jyothiMadhuchandu");
10             t.put(500, "vararama");
11
12             System.out.println(t);
13       }
14 }
15
```

Markers Properties Servers Data Source Explorer Snippets Console

```
<terminated> cac [Java Application] C:\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x8
{100=Madhu, 500=vararama, 600=jyothiMadhuchandu, 800=Kiranb}
```

```
3  import java.util.*;
4  public class cac {
5        public static void main( String args[] ) throws Exception {
6              TreeMap t =  new TreeMap();
7              t.put(100,"Madhu");
8              t.put(800, "Kiranb");
9              t.put(600, "jyothiMadhuchandu");
10             t.put(500, "vararama");
11             t.put("XXX", "jyothiMadhuchandu");  //ClassCastException
12             t.put(null, "vararama");  // NuillPointerException
13
14             System.out.println(t);
15       }
```

Markers Properties Servers Data Source Explorer Snippets Console

```
<terminated> cac [Java Application] C:\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_16.0.1.v20210
Exception in thread "main" java.lang.ClassCastException: class java.lang.Integer cannot
        at java.base/java.lang.String.compareTo(String.java:133)
        at java.base/java.util.TreeMap.put(TreeMap.java:806)
        at java.base/java.util.TreeMap.put(TreeMap.java:534)
        at com.high.code1.cac.main(cac.java:11)
```

COmparator:(Our Own Sorting Order)

```java
import java.util.*;
public class cac {
      public static void main( String args[] ) throws Exception {
            TreeMap t = new TreeMap(new myComparator());
            t.put("XXX", 101);
            t.put("YYY", 595);
            t.put("SSS", 334);
            t.put("ZZZ", 126);
            t.put("RRR", 985);
            System.out.println(t);

    }
}
class myComparator implements Comparator
{
    public int compare(Object o1,Object o2)
    {
        String s1 = o1.toString();
        String s2 = o2.toString();
        return s2.compareTo(s1);
    }
}

```

Markers Properties Servers Data Source Explorer Snippets Console

```
<terminated> cac [Java Application] C:\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64
{ZZZ=126, YYY=595, XXX=101, SSS=334, RRR=985}
```

HashTable:

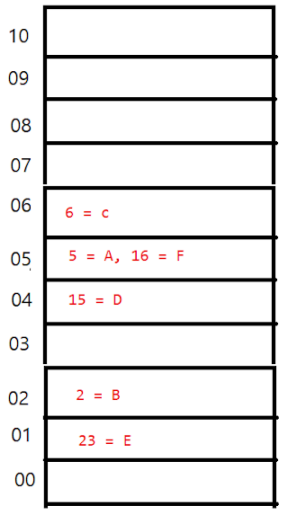
- The Underlying DataStructure for HashTable is HashTable
    - Java Class HashTable → HashTable
- Insertion Order is not Preserved.And It is Based On HashCode of Keys.
- Duplicate keys are not allowed, and values can be duplicated.
- Heterogeneous Objects are allowed for both Keys and Values.
- Null is not Allowed for Both Key and Value.Otherwise we will get a Runtime exception saying NullPointerException.
- It implements Serializable and cloneable interfaces but not random access.
- Every method present in HashTable is Synchronized and hence HashTable Object is ThreadSafe.
- HashTable is the Best CHoice if our frequent Operation is search Operation.

Constructors:

1. HashTable h = new HashTable();
    a. Default initial capacity -11; fill-ratio = 0.75
2. HashTable h = new HashTable(int intialCapacity);
3. HashTable h = new HashTable(int intialCapacity, float fillratio);
4. HashTable h = new HashTable(map m);

```
package com.high.code1;

import java.util.*;
public class cac {
        public static void main( String args[] )
throws Exception {
              Hashtable h = new Hashtable();
              h.put(new Temp(5),"A");
              h.put(new Temp(2),"B");
              h.put(new Temp(6),"C");
              h.put(new Temp(15),"D");
              h.put(new Temp(23),"E");
              h.put(new Temp(16),"F");
              System.out.println(h);


        }
}
```

```
class Temp
{
      int i;
      Temp (int i)
      {
            this.i = i;

      }
      public int hashCode()
      {
            return i;
      }
      public String toString()
      {
            return i+"";
      }
}
```



Output is from Top to bottom, And in same hash Right to left.

OutPut : {6=C, 16=F, 5=A, 15=D, 2=B, 23=E}

```
 3 import java.util.*;
 4 public class cac {
 5       public static void main( String args[] ) throws Exception {
 6             Hashtable h = new Hashtable();
 7             h.put(new Temp(5),"A");
 8             h.put(new Temp(2),"B");
 9             h.put(new Temp(6),"C");
10             h.put(new Temp(15),"D");
11             h.put(new Temp(23),"E");
12             h.put(new Temp(16),"F");
13 //          h.put("durga",null); // NullPointerException
14             System.out.println(h);
15
16
17       }
18 }
19 class Temp
20 {
21     int i;
22     Temp (int i)
23     {
24         this.i = i;
25
26     }
27     public int hashCode()
28     {
29         return i;
30     }
31     public String toString()
32     {
33         return i+"";
34     }
35 }
```

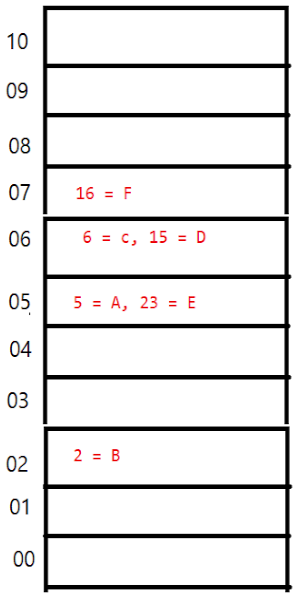
Markers Properties Servers Data Source Explorer Sr

<terminated> cac [Java Application] C:\eclipse\plugins\org.eclipse

```
{6=C, 16=F, 5=A, 15=D, 2=B, 23=E}
```

```java
package com.high.code1;

import java.util.*;
public class cac {
        public static void main( String args[] )
throws Exception {
            Hashtable h = new Hashtable();
            h.put(new Temp(5),"A");
            h.put(new Temp(2),"B");
            h.put(new Temp(6),"C");
            h.put(new Temp(15),"D");
            h.put(new Temp(23),"E");
            h.put(new Temp(16),"F");
            System.out.println(h);


        }
}
```

```java
class Temp
{
      int i;
      Temp (int i)
      {
            this.i = i;

      }
      public int hashCode()
      {
            return i%9;
      }
      public String toString()
      {
            return i+"";
      }
}
```

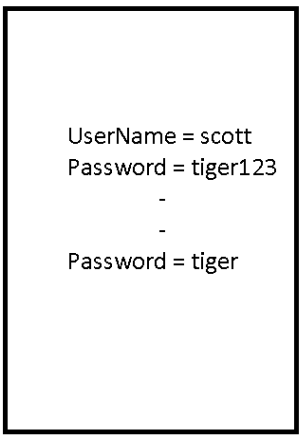


Output is from Top to bottom, And in same hash Right to left.

Hashtable ht = new Hashtable(25);

- **If we configure initialcapacity as 25, There will be 24 buckets as in the previous example.**
-

Properties: → Properties file

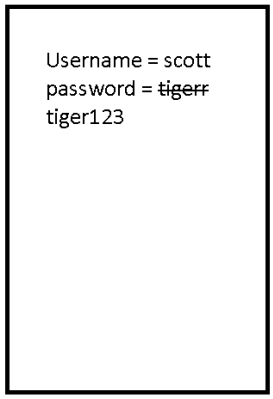


Main Java Soure

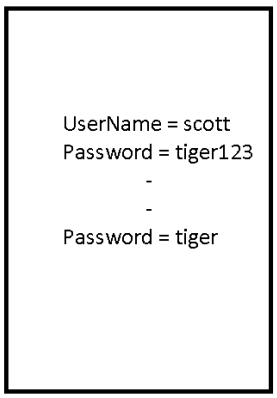
If we make any changes in main java source then you need to perform the following tasks.

Compile
rebuild
Redeploy
Restart server.

-



if we create a seperate properties files and we Change any details then we need to perform only redeploy



If we make any changes in main java source then you need to perform the following tasks.

Compile
rebuild
Redeploy
Restart server.

- abc.properties — Main Java Soure

- **In our program anything that changes frequently like username , password, mobileNumber, mail_id etc. are not recommended to hardcord in the java program because if there is any change to reflect that change, recompilation, rebuild, redeploy application are required even sometimes server restart is also required. Which creates a big business impact to the client.**
- **We can Overcome this program by using properties file such type variable things we have to configure in the properties file, from that properties file we have to read into java program and we can use those properties, the main advantage of this approach is if there is a change in properties file to reflect that change just redeployment is enough, which wont create any business impact to the client.**
- **We can use java properties Object to hold properties which are coming from properties files.**
- **In normal maps like HashMap, TreeMap, Hahstable , key and value can be anytype, but in the case of properties key-value should be String type.**

Constructor:

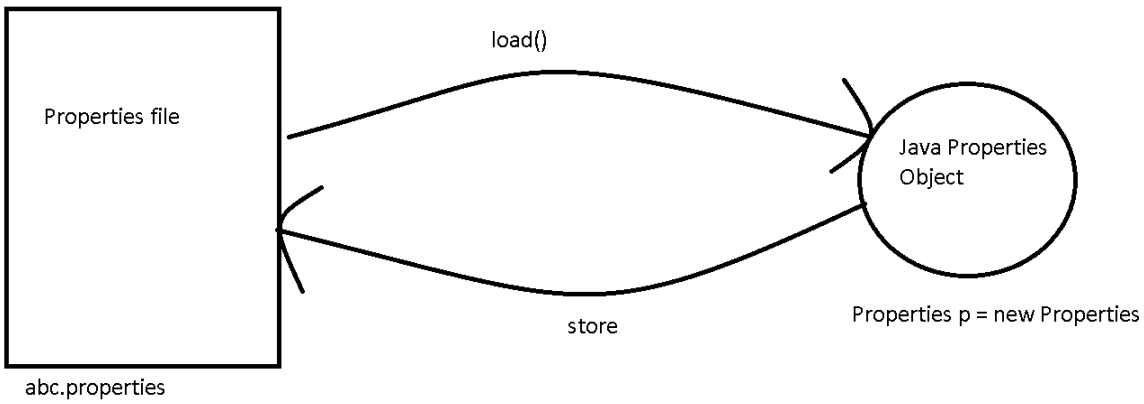
Properties p = new Properties();

Methods:

1. **String setProperty(String pname, String pvalue)**

a. To set a new Property.
b. If the specified property is already available then oldvalue will replace it with new value and return oldvalue.

2. String getProperty(String pname)
    a. To get value associated with the specified property.
    b. If the specified property is not available then this method returns null.
3. Enumeration PropertyNames():
    a. Returns all properties names in the properties object.



4. void load(InputStream is)
    a. To load properties from properties file into java properties object.
5. void store(OutputStream os, String comment)
    a. To store properties from java properties object onto properties file.

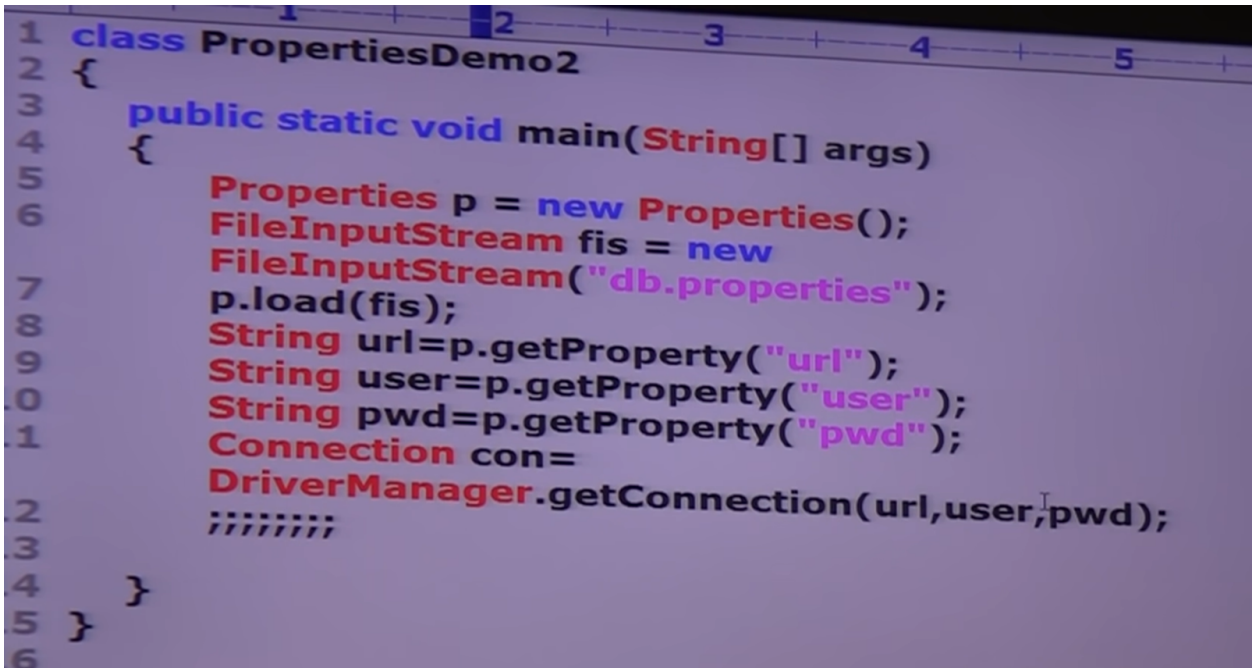
Abc.properties file

```
1  account=70545
2  password=Madhu@123
3  username=Chandu
4
```

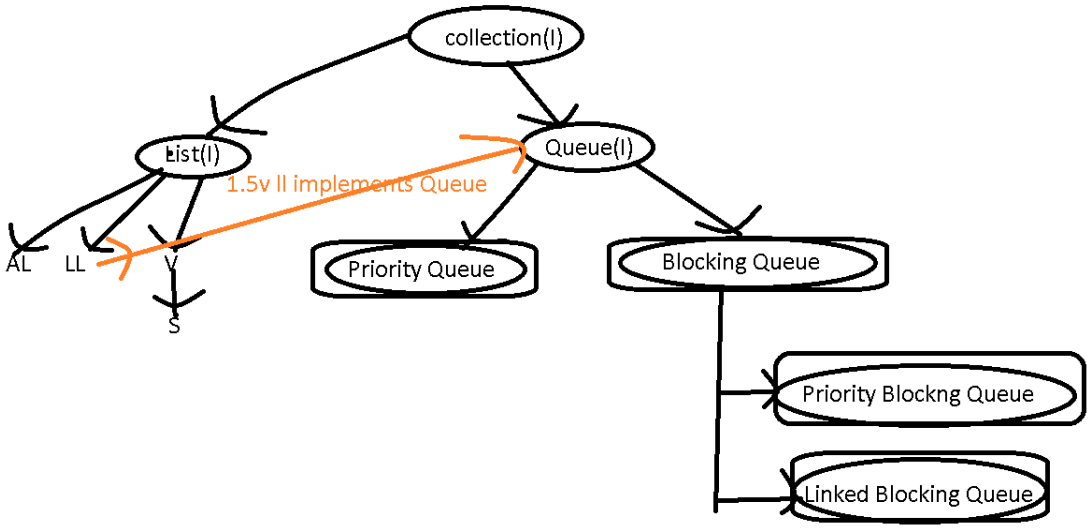
```java
import java.io.*;
import java.util.*;
public class properties {

    public static void main(String[] args)throws Exception {
        // TODO Auto-generated method stub
        Properties p = new Properties();
        FileInputStream fis = new FileInputStream("abc.properties");
        p.load(fis);
        System.out.println(p);
        String s = p.getProperty("username");
        System.out.println(s);
        p.setProperty("email", "ramireddy.reddy@highradius.com");
        FileOutputStream fos  = new FileOutputStream("abc.properties");
        p.store(fos, "updated by chandu");
        System.out.println(p);

    }

}
```

Output:

```
#updated by chandu
#Tue Feb 07 23:50:42 IST 2023
account=70545
email=ramireddy.reddy@highradius.com
password=Madhu@123
username=Chandu
```



Queue:



-

1.5V enhancements Queue Interface

- **It is the child interface of the collection.**
  - **Queue(I)**
    - **Priority Queue**
    - **Blocking Queue**
      - **Priority Blocking Queue**
      - **Linked Blocking Queue**
- **If we want to represent a group of individual Objects prior to processing then we should go for Queue.**
- **For EG: before sending an SMS message, all mobile numbers should store in some dataStructure, in which Order we added mobile numbers in the same order only msg should be delivered, for this first in first out requirement queue is best choice.**
- **Usually Queue follows FIFO order but based on our requirement we can implement our own priority order also (Priority Queue).**
- **From 1.5v onwards linkedList class also implements Queue interface.**
- **LinkedList based implementation of the queue always follows FIFO Order.**

QUEUE Interface Specific methods:

- boolean offer(Object o)
  - To add an Object into the queue.
- Object peek()
  - To return the head element of the queue. If the queue is empty then this method returns null.
- Object element()
  - To return the head element of the queue. If queue is empty then this method RE: NoSuchElementException
- Object poll()
  - To remove and return the head element of the queue. If the queue is empty then this method returns null.
- Object remove()
  - To remove and return the head element of the queue. If the queue is empty then this method raises RE: NoSuchElementException.

PriorityQueue:

- If we want to represent a group of individual Objects prior to processing according to some priority, then we should go for priority queue.
- The priority can be either default natural sorting order or customized sorting order defined by comparator.
- Insertion Order is not preserved and it is based on some priority.
- Duplicated objects are not allowed.
- If we are depending on default natural sorting order, compulsory objects should be homogenous and comparable otherwise we will get a runtime exception saying ClassCastException.
- If we are defining our own sorting by comparator, then objects need not be homogenous and comparable.
- Null is not allowed even as the first element also.

Constructors:

- PriorityQueue q = new PriorityQueue();

- Create an Empty priority queue with default initalcapacity- 11, and all Objects insert according to Default natural sorting Order.

- PriorityQueue q = new PriorityQueue(int intialcapacity);
- PriorityQueue q = new PriorityQueue(int intialcapacity,Comparator c);
- PriorityQueue q = new PriorityQueue(SortedSet s);
- PriorityQueue q = new PriorityQueue(Collection c);

```
1  import java.util.*;
2  class PriorityQueueDemo
3  {
4     public static void main(String[] args)
5     {
6        PriorityQueue q = new PriorityQueue();
7        //System.out.println(q.peek()); //null
8        //System.out.println(q.element()); //RE:NSEE
9        for(int i = 0; i<=10 ; i++)
10       {
11          q.offer(i);
12       }
13       System.out.println(q);//[0,1,2.....10]
14       System.out.println(q.poll());//0
15       System.out.println(q);//[1,2,3.....,10]
16    }
17 }
18
```

Output:

```
Note: Recompile with -Xlint:unchecked for
C:\durga_classes>java PriorityQueueDemo
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
0
[1, 3, 2, 7, 4, 5, 6, 10, 8, 9]

C:\durga_classes>
```

Expected Output : 123456789;

Some platforms won't provide proper support for thread priorities and priority queues.

```
import java.util.*;
class PriorityQueueDemo2
{
    public static void main(String[] args)
    {
        PriorityQueue q = new PriorityQueue(15,n
        MyComparator());
        q.offer("A");
        q.offer("Z");
        q.offer("L");
        q.offer("B");
        System.out.println(q); //[Z,L,B,A]
    }
}
class MyComparator implements Comparator
{
```

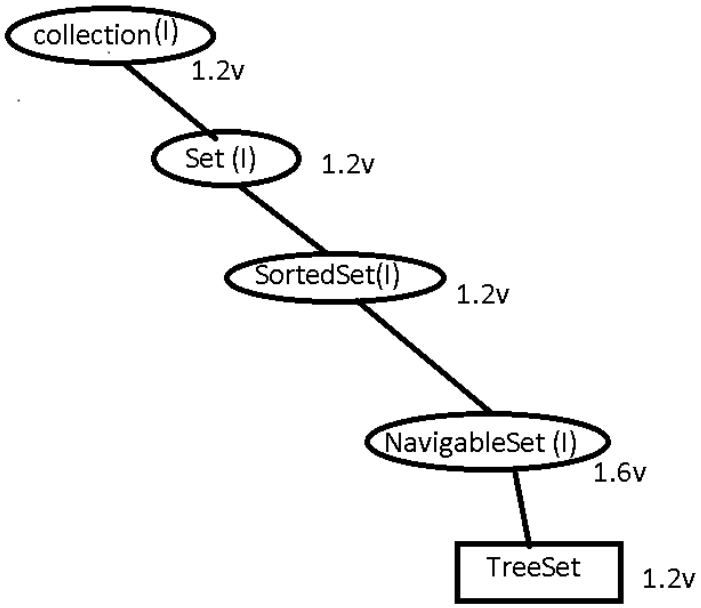
```
14 class MyComparator implements Comparator
15 {
16     public int compare(Object obj1,Object obj2)
17     {
18         String s1 = (String)obj1;
19         String s2 =obj2.toString();
20         return s2.compareTo(s1);
21     }
```

Output: ZLBA;

1.6v enhancements in collection frameworks.

As a part of 1.6v, the following 2 concepts are introduced in the collection framework.

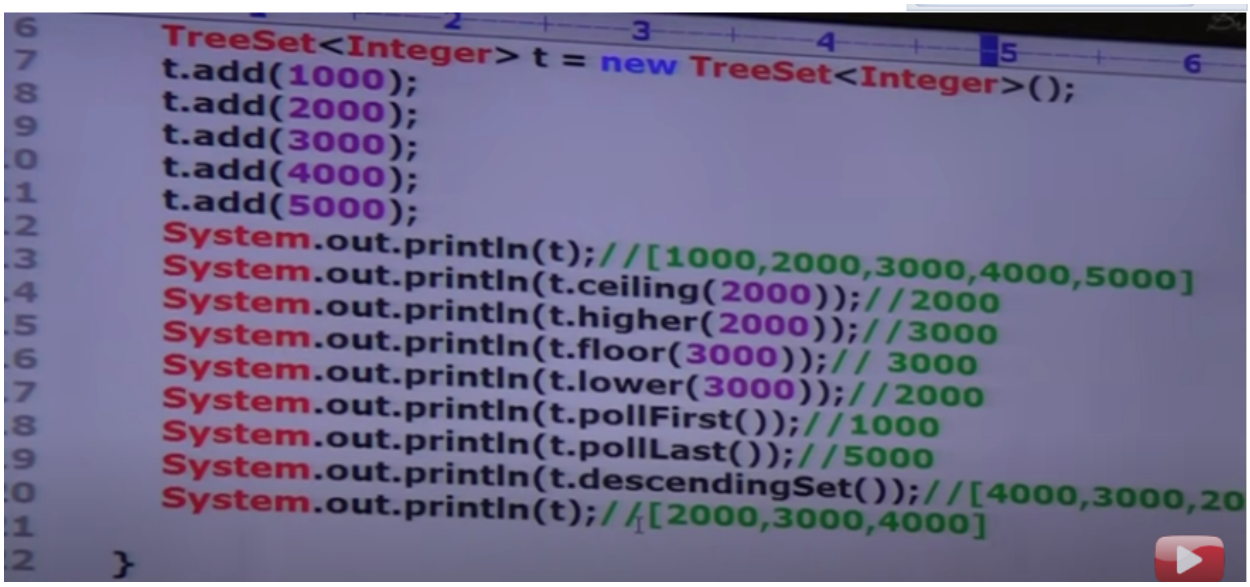
- **NavigableSet(I)**
- **NavigableMap(I)**



It is the child interface of sortedset and it defines several methods for navigation purposes.

NavigableSet defines the following methods:

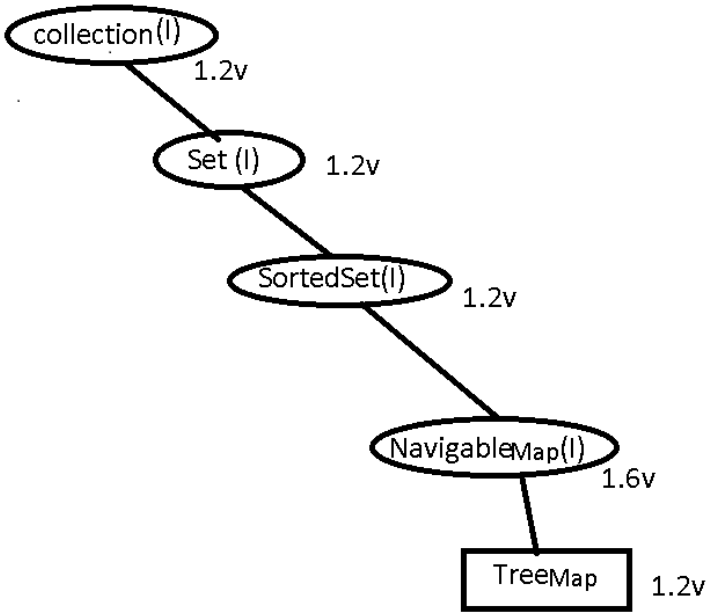
- **floor(e)**
  - **It returns highest element which is <=e**
- **lower(e)**
  - **It returns highest element which is <e**
- **ceiling(e)**
  - **It returns highest element which is >=e**
- **higher(e)**
  - **It returns highest element which is >e**
- **pollFirst()**
  - **Remove and return the first element.**
- **pollLast()**
  - **Remove and return last element**
- **descendingSet()**
  - **It returns NavigableSet in reverse Order.**



```
TreeSet<Integer> t = new TreeSet<Integer>();
t.add(1000);
t.add(2000);
t.add(3000);
t.add(4000);
t.add(5000);
System.out.println(t);//[1000,2000,3000,4000,5000]
System.out.println(t.ceiling(2000));//2000
System.out.println(t.higher(2000));//3000
System.out.println(t.floor(3000));// 3000
System.out.println(t.lower(3000));//2000
System.out.println(t.pollFirst());//1000
System.out.println(t.pollLast());//5000
System.out.println(t.descendingSet());//[4000,3000,20
System.out.println(t);//[2000,3000,4000]
}
```

NavigableMap(I):-

NavigableMap is chi;d interface of sortedMap, it defines several methods for navigation purposes.



==========

floorKey(e)

lowerKey(e)

ceilingkey(e)

higherKey(e)

pollFirstEntry()

pollLastEntry()

descendingMap();

==========

```
        TreeMap<String,String> t = new
        TreeMap<String,String>();
        t.put("b","banana");
        t.put("c","cat");
        t.put("a","apple");
        t.put("d","dog");
        t.put("g","gun");
        System.out.println(t);//{a=apple,b=banana,c=cat
        g,g=gun}
        System.out.println(t.ceilingKey("c"));//c
        System.out.println(t.higherKey("e"));// g
        System.out.println(t.floorKey("e"));// d
        System.out.println(t.lowerKey("e")); //d
        System.out.println(t.pollFirstEntry()); //a=apple
        System.out.println(t.pollLastEntry());//g=gun
        System.out.println(t.descendingMap());//{d=dog,
        ,b=banana}
        System.out.println(t);//{b = ba
    }
}
```


Collections:

Collections class defines several utility methods for collection Objects like sorting ,searching, reversing.etc..

Collections class defines 2 sort methods.

Sorting elements of lists :

- public static void sort (List l):
    - To sort based on Default natural sorting order.
    - In this case the list should compulsorily contain homogeneous and comparable else ClassCastException.
    - List should not contain null, otherwise we will get NullPointerException.
- Public static void sort(list l,Comparator c)
    - To sort based on customized Sorting Order.

Demo Programs:

```
public static void main(String[] args)
{
    ArrayList l = new ArrayList();
    l.add("Z");
    l.add("A");
    l.add("K");
    l.add("N");
   // l.add(new Integer(10));//---CCE
    //l.add(null);//---NPE
    System.out.println("Before Sorting:" +l);//[Z,A,K,N
    Collections.sort(l);
    System.out.println("After  Sorting:" +l);//[A,K,N,Z]
  }
}
```



```
import java.util.*;
class CollectionsSortDemo2
{
   public static void main(String[] args)
   {
      ArrayList l = new ArrayList();
      l.add("Z");
      l.add("A");
      l.add("K");
      l.add("L");
      System.out.println("Before Sorting:" +l);//[Z,
      Collections.sort(l, new MyComparator());
      System.out.println("After  Sorting:" +l);

   }
}
```

```
      System.out.println("After  Sorting:" +l);

   }
}
class MyComparator implements Comparator
{
   public int compare(Object obj1,Object obj2)
   {
      String s1 = (String)obj1;
      String s2 =obj2.toString();
      return s2.compareTo(s1);
   }
}
```

Searching element of List :

- **Collections class defines the following binary search methods,**

- Public static int binarySearch(list l, Object target)
    - If the list is sorted according to the default natural sorting order then we should use this method.
- Public static int binarySearch(list l, Object target,Comparator c)
    - We have to use this method if the list is sorted according to customized sorting order.

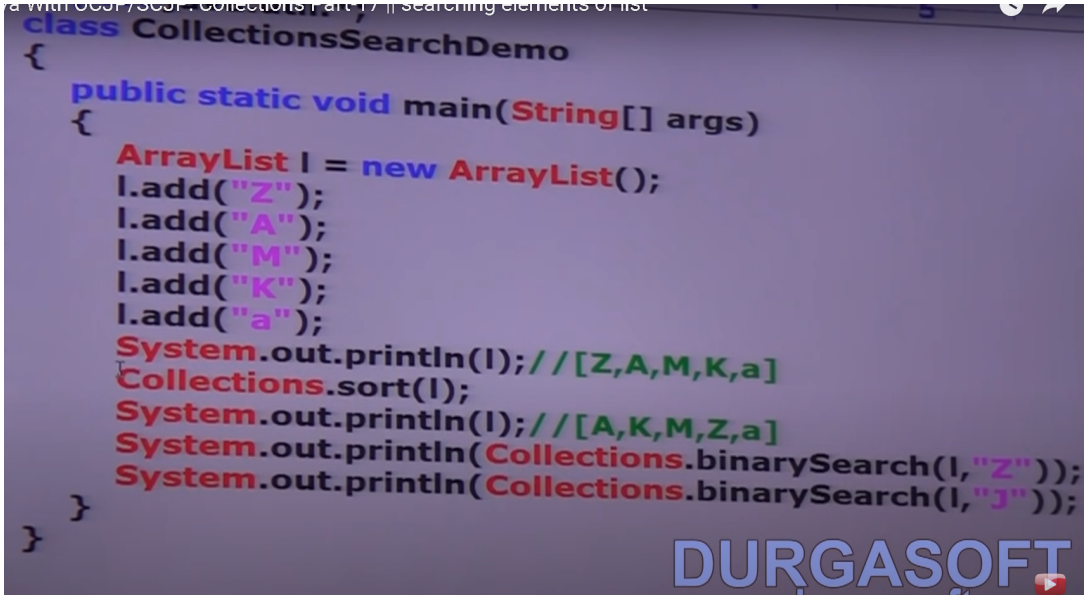
Conclusions :

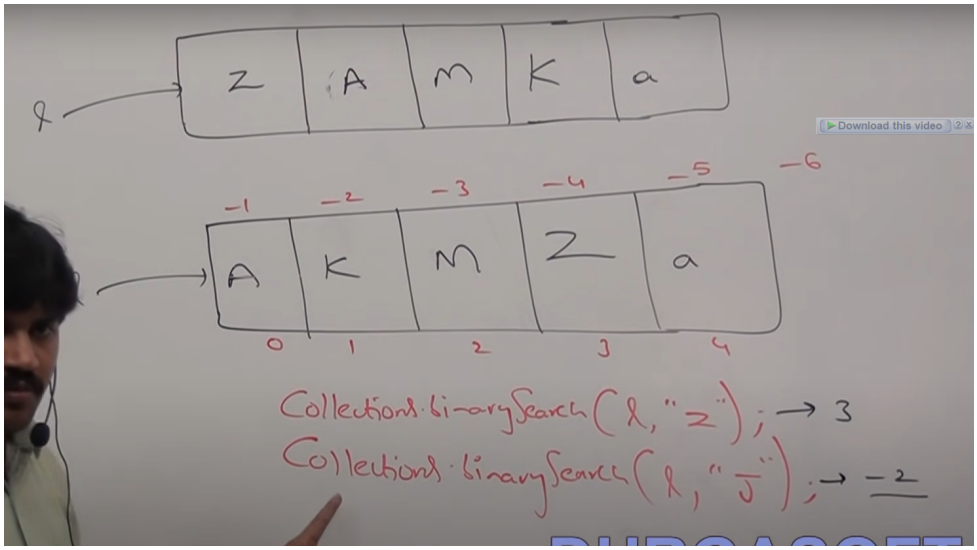
The above search methods internally will use binarySearchAlgorithm.

Successful search return → index

Unsuccessful search return → insertion Point

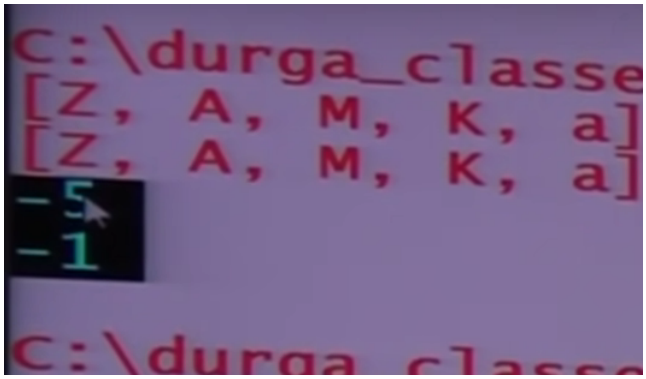
- Insertion Point is the location where we can place the target element in a sorted list.
- Before calling the binary search method, the compulsory list should be sorted, otherwise we will get unpredictable results.
- If the list is sorted according to comparator then at the time of search operation we have to pass the same comparator Object, otherwise we will get unpredictable results.



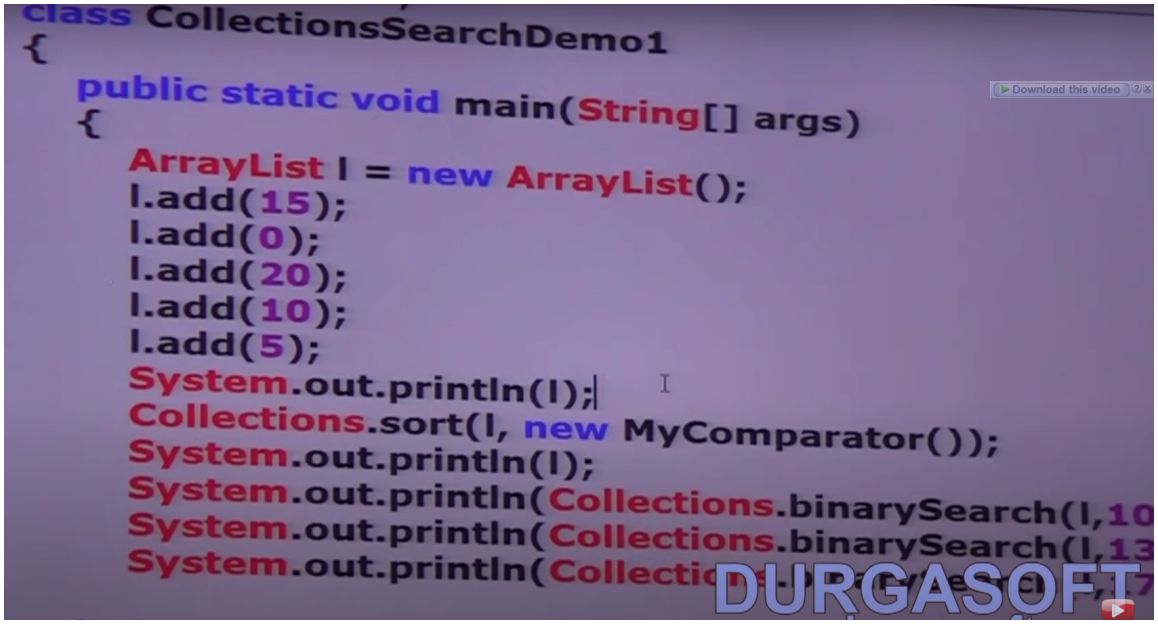
- 

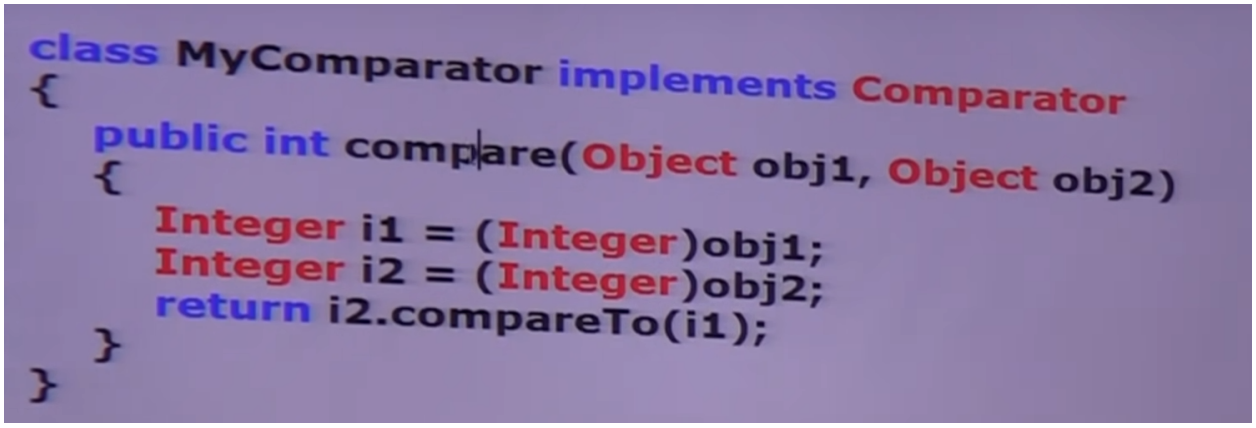
- **unsortedList**

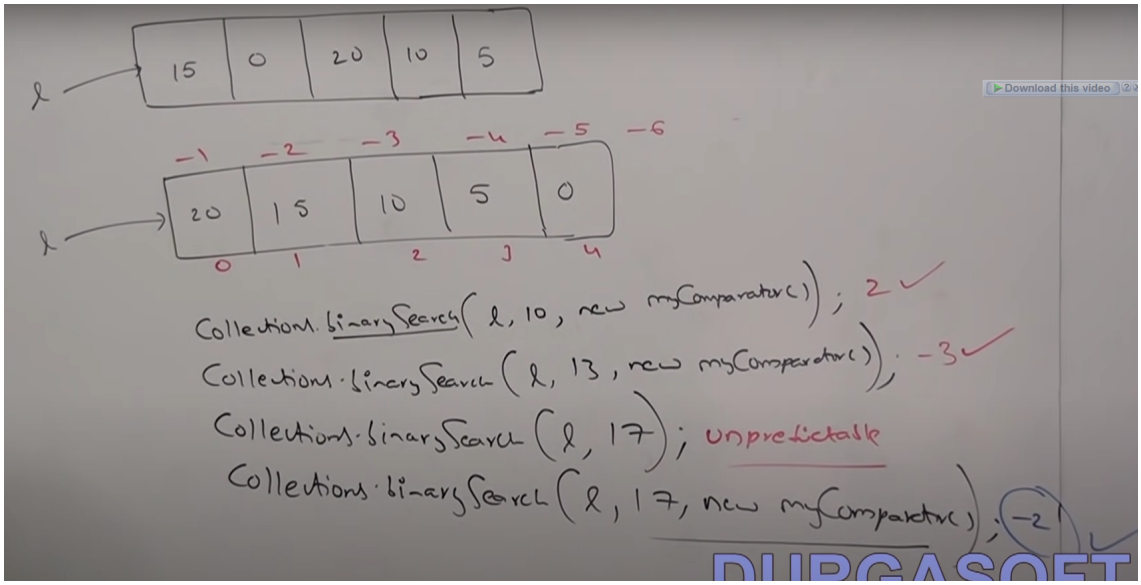
```
public static void main(String[] args)
{
    ArrayList l = new ArrayList();
    l.add("Z");
    l.add("A");
    l.add("M");
    l.add("K");
    l.add("a");
    System.out.println(l);//[Z,A,M,K,a]
    //Collections.sort(l);
    System.out.println(l);//[A,K,M,Z,a]
    System.out.println(Collections.binarySearch(l,"Z"
    System.out.println(Collections.binarySearch(l,"J"
}
}
```

- 

One more Example :

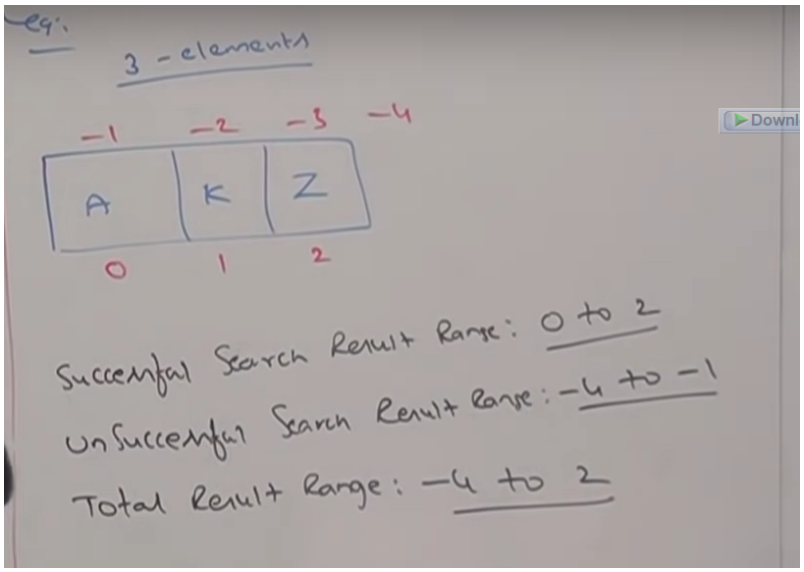

class CollectionsSearchDemo1
{
public static void main(String[] args)
{
ArrayList l = new ArrayList();
l.add(15);
l.add(0);
l.add(20);
l.add(10);
l.add(5);
System.out.println(l);
Collections.sort(l, new MyComparator());
System.out.println(l);
System.out.println(Collections.binarySearch(l,10
System.out.println(Collections.binarySearch(l,13
DURGASOFT


class MyComparator implements Comparator
{
public int compare(Object obj1, Object obj2)
{
Integer i1 = (Integer)obj1;
Integer i2 = (Integer)obj2;
return i2.compareTo(i1);
}
}


15 0 20 10 5
-1 -2 -3 -4 -5 -6
20 15 10 5 0
0 1 2 3 4
Collections.binarySearch(l, 10, new myComparator()); 2
Collections.binarySearch(l, 13, new myComparator()); -3
Collections.binarySearch(l, 17); unpredictable
Collections.binarySearch(l, 17, new myComparator()); -2
DURGASOFT

Note :

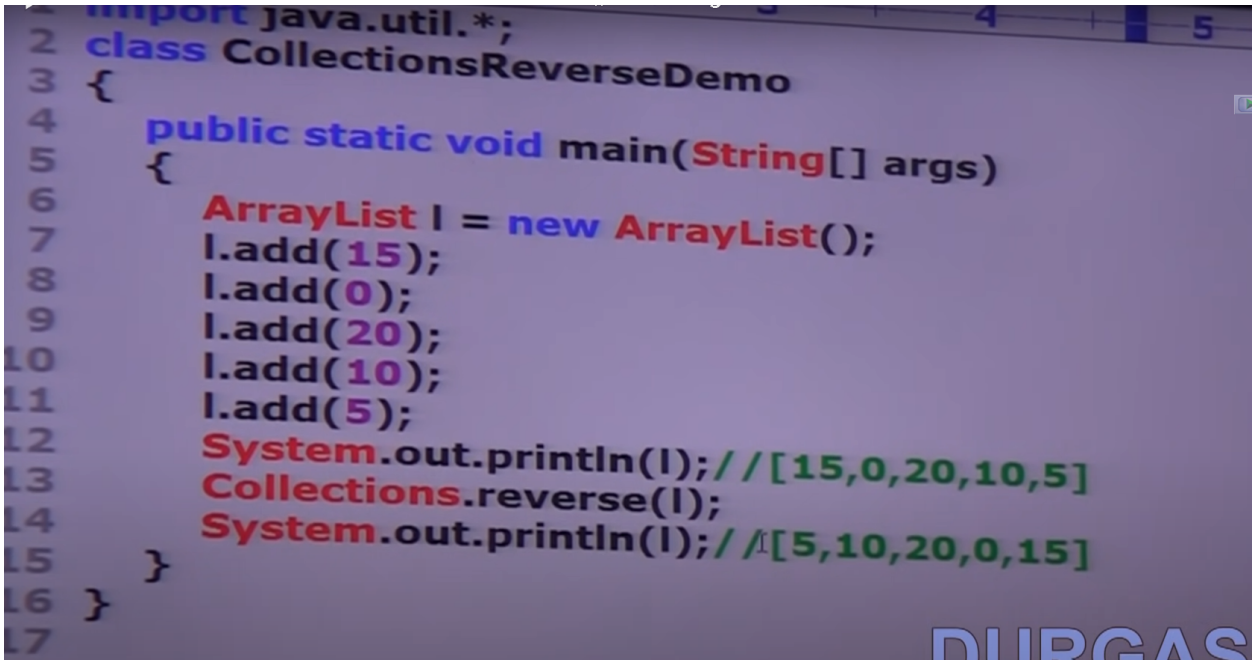
- For the list of n- elements in the binary search method
    - Successful search result range :- 0-n-1
- Unsuccessful search result range :- -(n+1) to -1;
- Total result range :- -(n+1) to n-1



Reversing Elements of List:

Collections class defines the following reverse method, to reverse elements of a list.

Public static void reverse(List l);



reverse() VS reverseOrder()

Comparator c1 = Collections.reverseOrder(Comparator l)

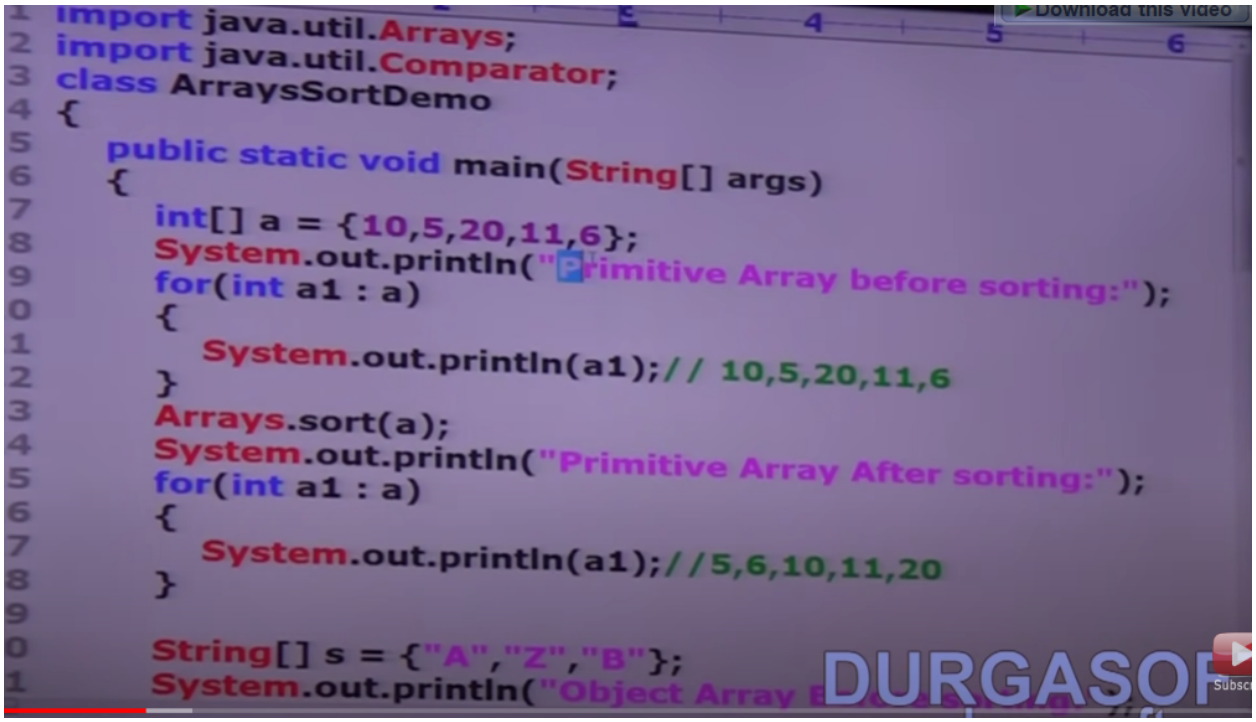
Desc **Asce**

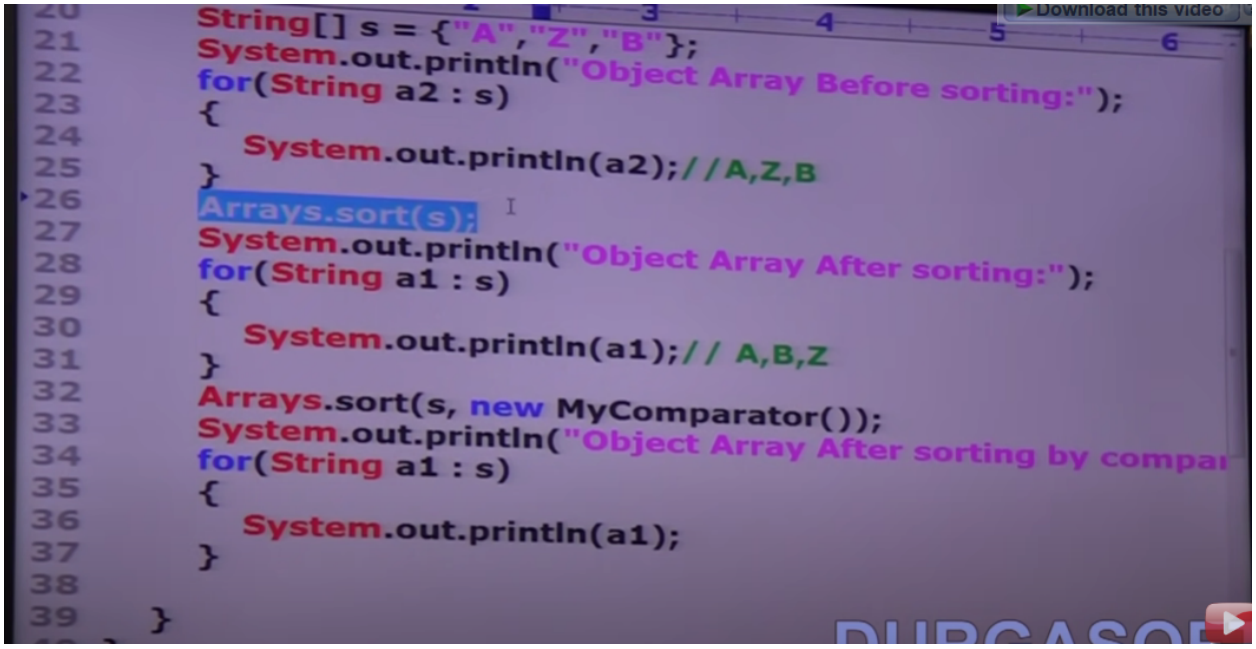
We can use the reverse method to reverse order of elements of the list, whereas we can use the reverse order method to get a reverse comparator.

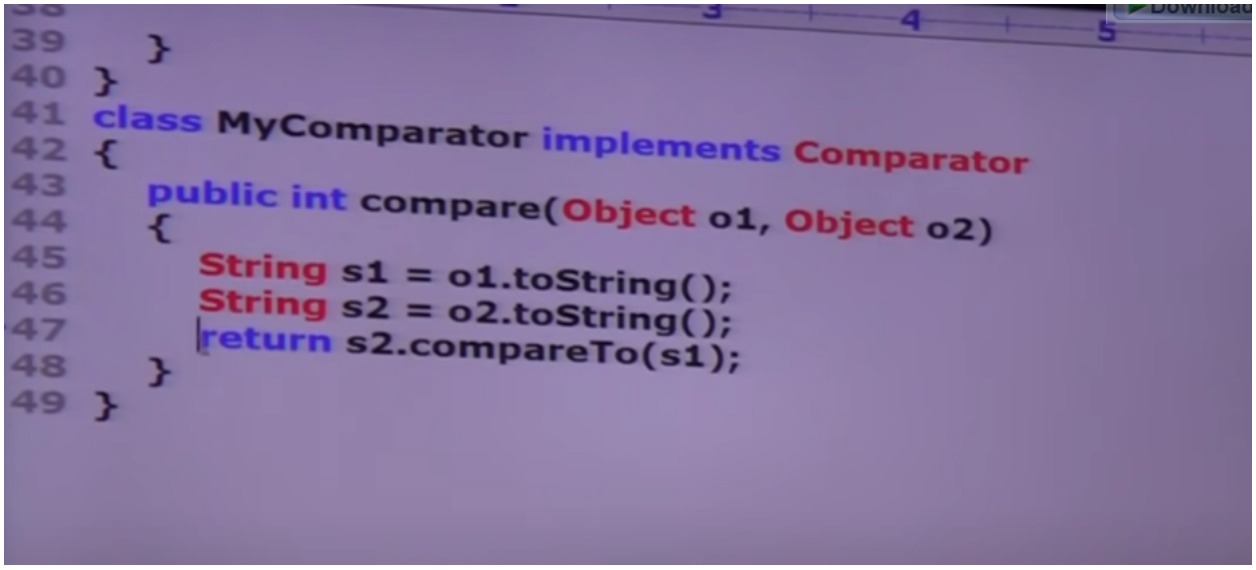
Arrays Class is an utility class to define several utility methods for array objects.

Sorting elements of array

- **Public static void sort(primitive[ ] p)**
  - **DNSO**
- **Public static void sort(Object[ ] o)**
  - **DNSO**
- **Public static void sort(Object[ ] o,comparator c)**
  - **CSO – > Customized Sorting Order …**







Note :

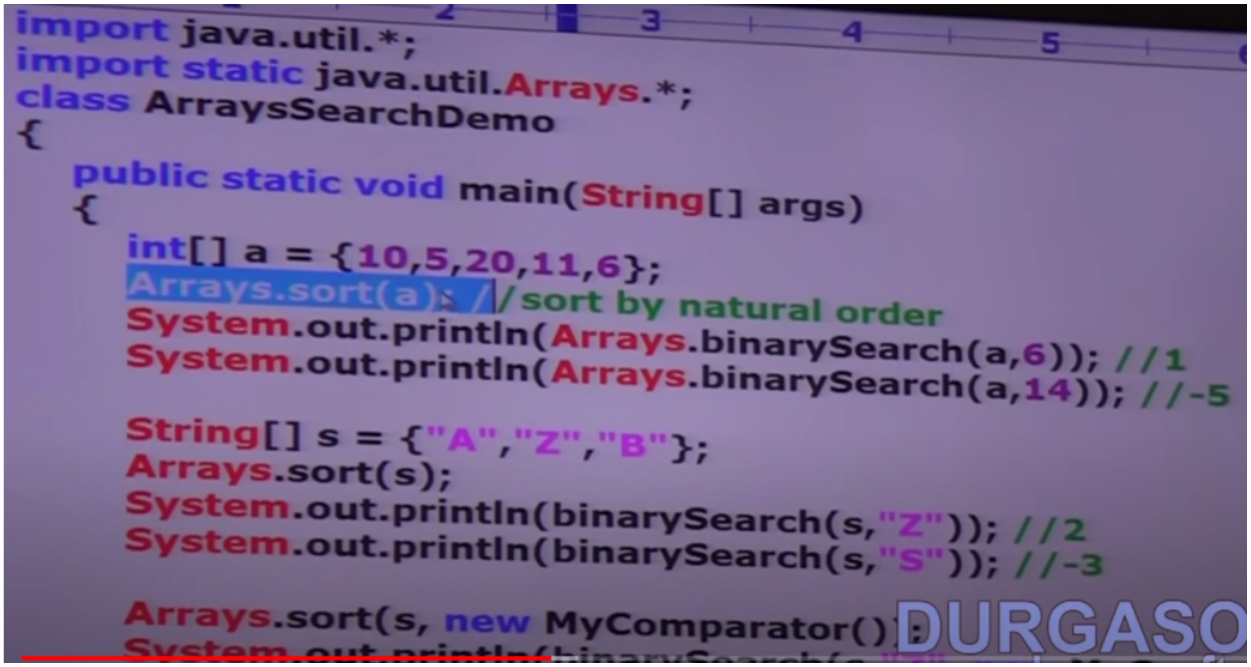
We can sort primitive arrays only based on default natural sorting order, whereas we can sort object arrays either based on default natural sorting order, or based on customized sorting order.

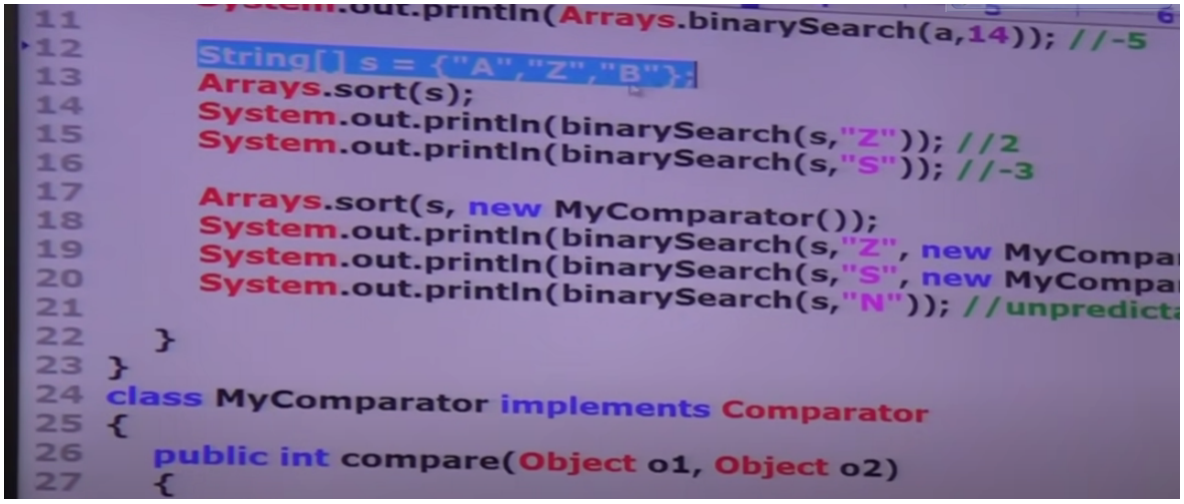
Searching Elements Of array:

Arrays Class defines the following binary search methods.

Public static int binarySearch(primitive[] p, primitive target);
Public static int binarySearch(Object[] p, Object target);
Public static int binarySearch(Object [] p, Object target, comparator c);

Note:
All rules of array class binary search methods are exactly the same as collections class binary Search methods.

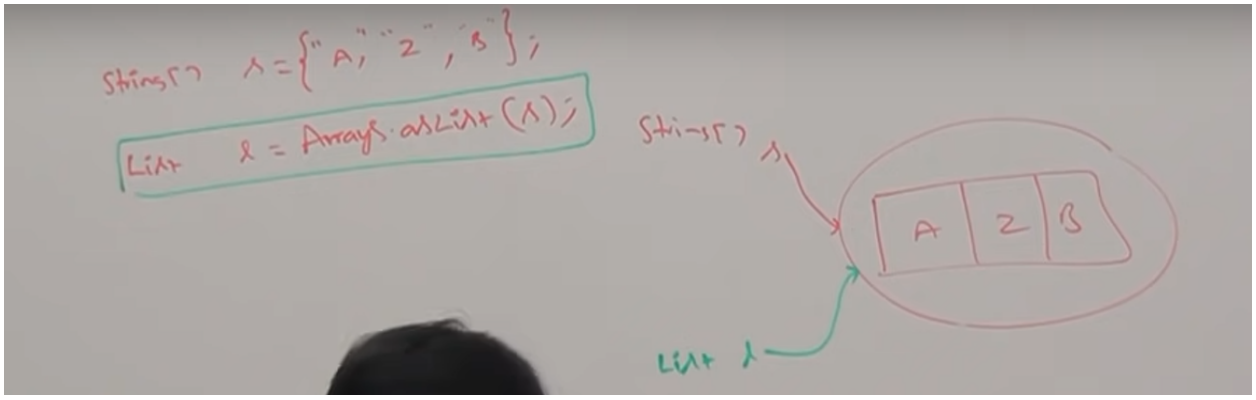




```
        Arrays.sort(s, new MyComparator());
        System.out.println(binarySearch(s,"Z", new MyComparat
        System.out.println(binarySearch(s,"S", new MyComparat
        System.out.println(binarySearch(s,"N")); //unpredictabl
    }
}
class MyComparator implements Comparator
{
    public int compare(Object o1, Object o2)
    {
        String s1 = o1.toString();
        String s2 = o2.toString();
        return s2.compareTo(s1);
    }
}
```

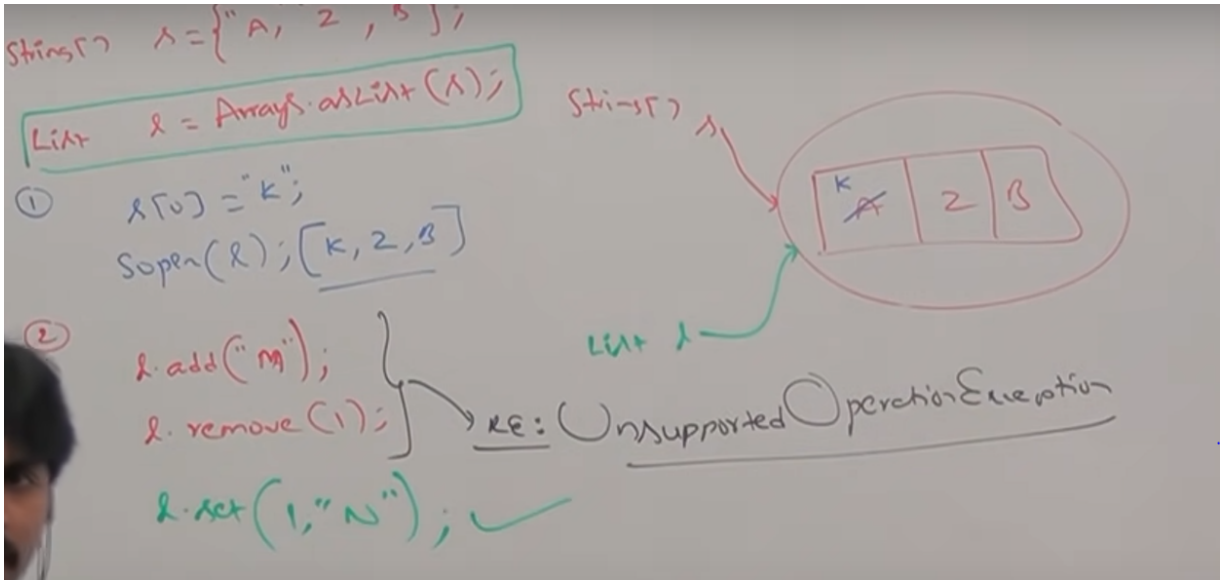
Conversion of Array to List...

Public static list asList(Object [ ] a)



By Using Array reference if we perform any change then automatically that change will be reflected to the array.

Similarly by using list reference if we perform any change, that change will be reflected automatically to the array.



By using list reference we cannot perform any operation which varies the size otherwise we will get runtime exception saying unsupportedOperationException

l.set(1,new Integer(10)) , → internally it is a string Array→ RE : ArrayStoreException.