COLLECTION FRAME WORK

What is Collection ?

Representing Group of Object or elements is called A Collection

Best Example for Collection is an Array ?

Adv.Of.Arrays

> An Array is memory allocation which enables you to store 'N' no.of Homogeneous mixer of values

> For Array Variables Memory Location consequent thus reading the data from the Array is faster than reading values from an ordinary variable

Disadvantages .Of Arrays :

> Array can store only similar type of values .doesn't allow you store dis-similar types of values

```
Eg:   int[ ] x=new int[3];
            x[0]=10;
           x[1]=20;
           x[2]="Shashi";  // Error
```

> Array are static in size . we must specify the size of an array before Compilation

```
Eg:    int[ ]  x=new int[3];
       String[ ] y=new String[20];
```

> Once we define the size of an array it doesn't allow you increase or Decrease the size of any Array during the program execution.

>Based on our application requirements we can defined the array of primitive or reference type

```
int[ ] x=new int[3];   //int – Primitive type
      x[0]=10;
    x[1]=20;
```

```
Integer[ ] x=new Integer[10];  //Reference Type
           x[0]=new Integer(10);
        x[1]=new Integer(20);  // Integer Objects
```

Basically an Array is Homogeneous mixer of Elements

```
      Integer[ ] x=new Integer[10];
            x[0]=new Integer(10);
          x[1]=new Integer(20);
       x[2]=new String("Sai"); //Error
```

> Object is the super class of Every Class In Java

> the Super class reference variable can store the object of the same class or it can hold the object of any of its subclass implicitly

```
Object[ ] o=new Object[5];
        o[0]=new Integer(10);
        o[1]=new Float(3.14f);
        o[2]=new String("Sai");  //valid
```

> In Array No predefined Method support for manipulating the Array Collection

> Array's doesn't works based any underlying Data Structure concepts
Like : LIFO --> Stack | FIFO ---> Queue

To overcome the above limitation then we have go with Collection In Java

If you want represent a group of individual objects as a single entity then we need to work with collection

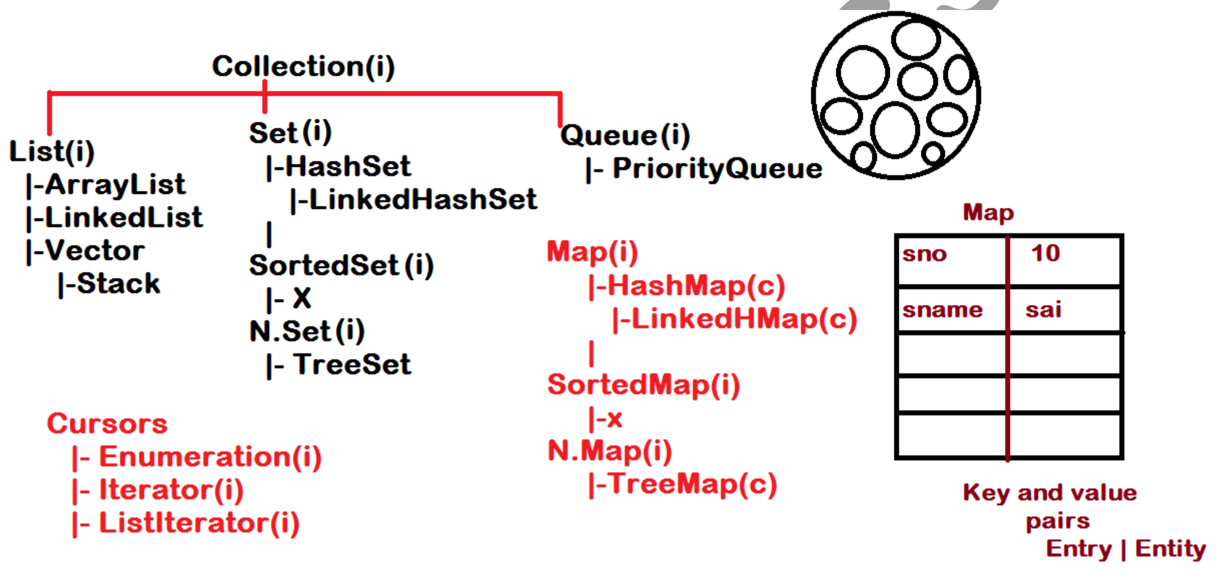
What is A Collection Frame Work ?

Collection Frame work define with several classes and interfaces which can be used to represent group of objects as a single entity

9 Key Interfaces of Collection Frame Work

----

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



sno	10
sname	sai

Collection (I) :

- If you want represent a group of individual Objects as a single entity then we have to work with Collection Interface
- This interface define with most commonly Used methods which are applicable for any Collection Object
- This interface act as an root interface of entire Collections Frame Work
- There is no concrete class which implements Collection interface directly

Collection Vs Collections

- Collection is an interface which can be used to represent group of individual objects as a single entity, Whereas Collections is utility class to define with several methods like Sorting, Searching for Collection Object

2. **List (I)** : It is a Child interface of Collection if you want represent a group of Collection individual Object where duplicates are allows and insertion order must is preserved

Note: Vector and Stack classes are reengineered 1.2 version to fit to Collection Frame Working by Implementing List Interface

3.**Set (I)** : It is child interface of Collection if we want represent a group of individual objects where duplicates are not allows and insertion order not preserved

4. **SortedSet (I)** : It is the child interface of Set (I) If you want represent a group of individual Object without duplicate but according to some Sorting Order

5. NavigableSet (I) : Child interface of SortedSet(I) with define several methods for navigation purpose

6.**Queue (I) :** It is a child interface of Collection if you want represent a group of individual Objects according to prior to process

Note : All above interfaces (Collection, List, Set,SortedSet, NavigableSet and Queue) designed for representing individual Objects, If you want represent Objects in the form of Key and Value pair then we have to go for Map(I)

7. **Map (I)** : If you want represent group of Objects in the form Key and Value pair we have to use Map interface. it is not Child interface of Collection , Duplicate keys are not allows but values can be duplicated

8.**SortedMap (i):** It is child interface of ?Map if you want represent a group of Objects in the form of key and value pair according to Some Sorting Order of keys then we should go for SortedMap

9.**NavigableMap (I)** : Child Interface of SortedMap which provides several methods for Navigation purpose

Collection :

-- If you want represent a group of individual objects as a single entity then we need to work with collection

-- Collection interface define with most common used methods which are applicable for any Collection Object

Methods:

boolean addObject(Object);

boolean addAll(Collection)

boolean remove(Object)

boolean removeAll(Collection)

void clear( ) : It will clear all Collection Objects

boolean contains(Object):

To ensure weather the specified Object is existed or not in the Collection

boolean retainAll(Object) :

To remove all Collection Objects except those which are presented in the Collection c

boolean containAll(Collection)

boolean isEmpty( ):

int size( )

Object[ ] toArray( )

Iterator iterator( )

List Interface :

It is a Child interface of Collection if you want represent a group of Collection individual Object where duplicates are allows and insertion order is preserved we can differentiate duplicate Object and preserve insertion order by Using Index hence index plays very important role in the list

Methods

boolean add(int index,Object o)

boolean addAll(int index,Collection c)

Object get(int index) : we can get an object

Object remove(int index)

Object set(int index,Object new) : to replace the element presented at specified index with provided object and return old object

int indexOf(Object) : returns the index of first occurrence of ' o ' in the list

Object lastIndexOf(Object)

ListIterator listIterator ( )

Implemented Classes :

ArrayList :

-- The underlying Data structure is Grow able Array
-- Insertion Order must be preserved Based On Index
-- Duplicate Objects are allowed
-- null insertion is possible
-- Heterogeneous Objects are allows

Note: All most all Collection classes are allows Heterogeneous Objects where as TreeSet and TreeMap Doesn't allow

Constructor:

ArrayList l=new ArrayList( );

Here empty ArrayList Object with default or initial capacity with 10 Once ArrayList reached to its max capacity then new ArrayList Object created with
new capacity=(current Capacity* 3/2) +1;

2.ArrayList l=new ArrayList(int initial Capacity) :

It will Crate an empty ArrayList Object with specified initial capacity

3.ArrayList l=new ArrayList(Collection c);

It will create equivalent ArrayList Objet for the given Collection Object. this is Used for inter conversion between Collection Object

```
eg:    import java.util.*;
class ArrayListDemo
{
  public static void main(String args[] )
  {
       ArrayList l=new ArrayList( );
                 l.add("A");    l.add(10);
```

```
                        l.add("A");     l.add(null);
                        System.out.println(l);
                        l.remove(2);
                        System.out.println(l);
                        l.add(2,"M");   l.add("N");
                        System.out.println(l);       }
}
```

ArrayList and Vector classes are implements RandomAccess Interface hence we can access any Object randomly with same speed. i.e Based Location

IQs:

```
                    ArrayList l1=new ArrayList( );
                         LinkedList l2=new LinkedList( );

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

Note : ArrayList is best choice if our frequent Operations is retrieving, Worst Choice if you want insert or remove the Objects in the middle bcoz number for adjustment is required

Difference Between ArrayList and Vector

ArrayList	Vector
1.All the methods of ArrayList is not Synchronized	1. All the methods of Vector is Synchronized
2.Multiple Threads Can perform operations ArrayListObject hence Which is not TS	2.Only One Thread can perform the operations on Vector hence Vector is Thread Safe
3.Threads are not required to be wait hence Thread relatively performance is high	3. In Vector increasing breaking time of hence performance is low
4.Non Legacy in jdk 1.2 version	4. It is from legacy Collection from JDK 1.0

IQ: How to get Synchronized Version of ArrayList Object

By default ArrayList is non Synchronized But we can get Synchronized Version of ArrayList Object is possible by Using the following method

```
public static List synchronizedList(List);
```

```
Eg:    ArrayList l=new ArrayList( );
       List l1=Collections.synchronizedList(l);
```

// here l1 is synchronized version where l is non synchronized

we can get synchronized version of Set and Map Objects by using the following methods Of Collections

```
public static Set synchronizedSet(set s);
public static Map synchronizedMap(Map m);
```

Linked List :

-- Underlying Data structure is Double Linked List
-- Heterogeneous Objects are Allows

-- insertion order is preserved
-- Duplicate Objects are allows and null insertion is possible
-- implemented by Serializable and Cloneable interfaces but not RandomAccess Interface

-- Best Choice to Use Linked List . if our frequent operations are insertions and deletion in the middle of the list but worst choice for retrieving the Objects from Linked list

-- Usually we can use linked list for implementing Stack and Queue

Methods

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

Constructor:

LinkedList l=new LinkedList( );

It will Create an empty LinkedList Object

LinkedList l=new LinkedList(Collection);

Crates an equivalent LinkedList object for the given Collection Object

Eg :

```
import java.util.*;
  class LinkedListDemo
  {
     public static void main(String args[] )
    {
       LinkedList l=new LinkedList( );
             l.add("Shashi"); l.add(30);  l.add(null);
             l.add("Shashi"); l.set(0,"SSSIT");
             l.add(0,"Java"); l.removeLast( );
```

```
        l.addFirst("Core");
    System.out.println(l);       }    }
```

Vector :

-- Underlying Data Structure is resizable Array or grow able Array
-- Duplicates are allows insertion order is preserved
-- Heterogonous Objects are allows and null insertion is possible
-- implemented Serializable , Clonable and RandomAccess
-- All methods are synchronized hence Vector Objects Thread Safe

Vector is best choice for retrieval operations worst choice for if your frequent operations in insertion and deleting in the middle

Constructors:

Vector v=new Vector( );

It will Create an empty Vector Object with default initial capacity with 10 . Once Vector is reached to max capacity then a new Vector Object is Created with Double capacity

new capacity=(Current capacity*2);

Vector v=new Vector(int initial Capacity);

It will Create an empty Vector Object with specified initial capacity

Vector v=new Vector(int initial Cap, int incremental cap);
Vector v=new Vector(Collection c);

Methods

For Adding Object

add(Object) from Collection (I)
add(int index,Object) From List(I)
addElement(Object) From Vector

For Removing Objects

remove(Object) From Collection
removeElement(int index) from List
removeElementAt(int index) from Vector

clear( ); From collection it will remove all objects from Vector
removeAllElements( ) From Vector

For Accessing:

Object get(index) From List
Object elementAt(index) From Vector
Object firstElement( ) From Vector
Object lastElement( ) From Vector

Other methods

int size( ) : It will return no.of. Objects are existed in the Vector Object

int capacity( ): It will return capacity of Vector Object i.e How many number of Objects can be placed

Enumeration elements(); To return the elements one by one from Vector Object

```
  import java.util.*;
 class VectorDemo
 {     public static void main(String args[])
     {        Vector v=new Vector( );  // Vector v=new Vector(10,3);
                System.out.println(v.capacity( ));
              for(int i=0;i<=10;i++)
                 {v.addElement(i);}
             System.out.println(v.capacity( ));
             v.add("A");
             v.add(v.capacity());
             System.out.println(v);      }
 }
```

Stack:

-- It is Sub class of Vector
-- It is specially designed Class for LIFO Order

Constructor

Stack s=new Stack( );

methods

Object push(Object)

Object pop( )

Object peek( ) : It will return Object which is existed on Top of Stack without removal

boolean empty( ) : return true if stack is empty

int search(Object) returns index or offset if the element is existed otherwise it will return -1 if specified Object is not existed

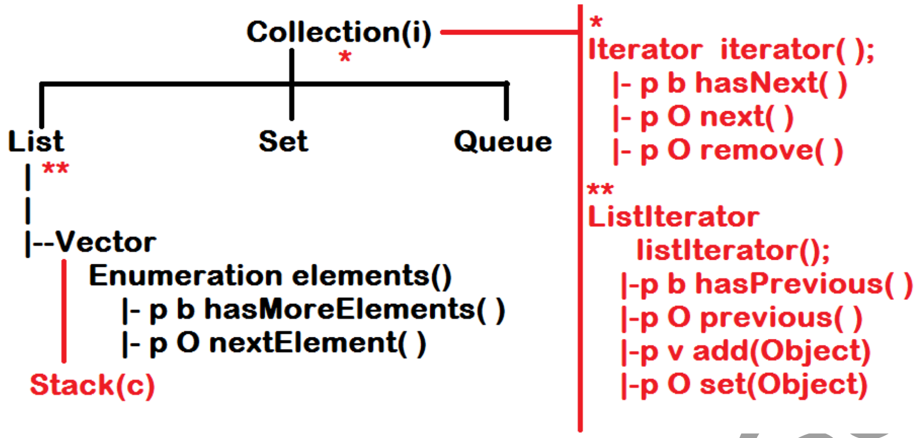
prg:

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

IQ. What is Cursor in Java Collection

Cursor are used to get or read One by One Object from the Collections. In java There are three Types of Cursors

1.Enumeration
2.Iterator
3.ListIterator



Enumeration :

We can use enumeration to get one by one objects from the legacy Collection objects. we can create enumeration Object using elements ( ) from Vector class

public Enumeration elements( );

Enumeration e=v.elements( );

Enumeration(I) : Define with two methods i.e

public boolean hasMoreElements();

public Object nextElement( );

Prg:

```
import java.util.*;
class EnumerationDemo
 {
            public static void main(String args[])
            {
                     Vector v=new Vector( );
                      for(int i=0;i<=10;i++)
                       {v.addElement(i);}
                   System.out.println(v);

                 Enumeration e=v.elements( );

                 while(e.hasMoreElements())
```

```
                        {Integer i=(Integer)e.nextElement();
                                if(i%2==0)
                                   System.out.println(i);
                        }
                        System.out.println(v);
                  }
         }
```

Limitations of Enumerations

-- Enumeration is applicable for only legacy Collection classes and it is not universal Cursor
-- By using Enumeration we can perform only read operations, we can't perform remove operations
-- To overcome this limitation the we need to go for Iterator

Iterator: we can Apply Iterator Concept for any type of Collection object hence it is universal cursor , by using this cursor we can perform read and remove operations

-- we can create Iterator Object by using iterator( ) of Collection interface

```
                  public Iterator iterator( )
                        Iterator i=c.iterator( );
```

methods

```
          public boolean hasnext( )
          public Object next( )
          public Object remove( )
```

prg:

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

          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);
   }
}
```

ListIterator:

-- It is bidirectional Cursor it will move in both either forward or backward Directions

-- By using ListIterator we can perform replacement and addition of new Object in addition to read and remove operations also possible

-- ListIterator is Sub interface of Iterator

-- we can create an object ListIterator by using the listIterator( ) of List(I)

public ListIterator listIterator( );
ListIterator ltr=l.listIterator( ); // here l is Any list Object

methods

```
public boolean hasNext( );
public Object next( );
public int nextIndex( );
public boolean hasPrevious( );
public Object previous( );
public int previousIndex( );

public void remove( );
public void set(Object new ); //replace object
public void add(Object new); // for add new Object
```

Prg:

```
import java.util.*;
class ListIteratorDemo
{
      public static void main(String[] args)
        {
                LinkedList l=new LinkedList();
                  l.add("Roja"); l.add("Kooja");
                  l.add("Manasa"); l.add("Samatha");
                  l.add("Ramesh");
               System.out.println(l);

           ListIterator ltr=l.listIterator( );
             while(ltr.hasNext())
                 {
                   String s=(String)ltr.next();
                      if(s.equals("Manasa"))
                           ltr.remove();
                     // if(s.equals("Ramesh"))
                            ltr.set("Shashi");
                 }
            System.out.println(l);
        }
  }
```

DIFFERENCE BETWEEN ENUMERATION,ITERATOR AND LISTITERATOR

Properites	Enumerations	Iterator	ListIterator
Is it Legacy	Yes	No	No
Applicable	Only for Legacy	Any Collection Obj	Only for List Objects
Moment	Single Direction	Single Direction	Bi-Directional
How to get	By using elements( )	By Using iterator( )	By Using ListIterator( )
Accessibility	Only readAccess	read( ) and remove( )	read( ),remove(), set( ),add( )

Set (I) :

-- It is a child interface of Collection

-- if you want represent a group of individual Objects where duplicates are not allows and insertion order is not preserved

Set interface doesn't contains any new methods we have to use only the methods which are existed in Collection interface methods

HashSet

- -- The Underlying DS HashSet is HashTable
- -- Duplicates are not allows
- -- Insertion order not preserved and it is based on HashCode of Objects
- -- Heterogeneous Objects are allows
- -- null insertion is possible
- -- HashSet implement Serializable , Clonable interfaces

Constructors

HashSet h=new HashSet( );

It will create an empty HashSet Object with default initial capacity 16 and default fill ration is 0.75

HashSet h=new HashSet(int initialCap) :
Creates an empty HashSet Object with specified initial capacity and default fill ration is 0.75 only

HashSet h=new HashSet(int initialCap, float fillRation) :
HashSet h=new HashSet(Collection c);

Example:

```
import java.util.*;
class HashSetDemo
{  public static void main(String[] args)
   { HashSet h=new HashSet( );
                h.add("B"); h.add("C"); h.add("D");
                h.add("Z"); h.add(null); h.add(10);
      System.out.println(h.add("Z")); //false
      System.out.println(h);
   }
}
```

we doesn't give an order by a programmer , duplicates are not allowed in the Set if you trying to add duplicate we won't any CE/RE add( ) which return false

LinkedHashSet :

-- It is the Child class of HashSet
-- It is Exactly Same as HashSet except the following differences

HashSet	***LinkedHashSet***
Underlaying Data Structure is Hash Table	Underlaying DS is HashTable + Linked List hence it is a Hybrid Collection
Insertion order is not preserved	Insertion order is preserved
introduced in 1.2 version	introduced in 1.4 version

```
import java.util.*;
class LinkedHashSetDemo
{  public static void main(String[] args)
   { LinkedHashSet h=new LinkedHashSet( );
                h.add("B"); h.add("C"); h.add("D");
```

```
                    h.add("Z"); h.add(null); h.add(10);
             System.out.println(h.add("Z")); //false
             System.out.println(h);
          }
       }

    output : [B,C,D,Z,null,10]
```

Where Duplicates are not allowed but insertion order must be preserved

SortedSet :

-- It is Child Interface of Set

-- If we want represent a group of unique Objects According to Some Sorting Order

-- Sorting Order can be default or Custom Sorting Orders

Normal Set Eg: {101,103,102} or {102,103,101} or {101,102,103} all are same only

SortedSet is as follows Example

100
101
102
104
106
108
110

SortedSet is Having Six methods

Object first( ) --> Returns the first element from the SortedSet

Object last( ) --> Returns the last element from the SortedSet

SortedSet headSet(Object) -->returns SortedSet whose elements are less Than Object

SortedSet tailSet(Object) --> returns SortedSet whose elements are >=Object

SortedSet subSet(Object1, Object2) --> returns Sorted elements which are >=Obj1 and <=Obj2

Comparator comparator( ) --> returns Comparator Object That describes Underlying Sorting technique. If we are using default natural SortingOrder then this method will return null

TreeSet:

-- The Underlying Ds Balanced Tree
-- Duplicates are not allowed
-- Insertion order is Not preserved
-- It is Based on SortingOrder
--Heterogeneous Objects are not allowed by mistake if we are trying to Insert Heterogeneous Object then we will get Run time Exception ClassCastException
-- null insertion is possible but only once

Constructor

TreeSet t=new TreeSet( ): It will Create simple TreeSet Object with default natural Sorting Order i.e Accending

TreeSet t=new TreeSet(Comparator c) : An empty TressSet Object is Created where Sorting order is customized Sorted Order

TreeSet t=new TreeSet(SortedSet);
TreeSet t=new TreeSet(Collection);

Example :

```
import java.util.*;
class TreeSetDemo
{
  Public static void main(String args[])
  {
      TreeSet t=new TreeSet( );
                t.add(“A”); t.add(“a”); t.add(“B”);
                t.add(“Z”); t.add(“L”);
                t.add(new Integer(10)); //CCE
                t.add(null);
               System.out.println(t);
  }
}
```

Java.lang.NullPointerException : For non empty TreeSet if you trying to insert null then we will get NullPointerException. For empty TreeSet null insertion is possible becoz there is no comparison with any. If any we will get Exception

```
import java.util.*;
class TreeSetDemo2
{ public static void main(String args[])
   {
        TreeSet t=new TreeSet();
                t.add(new StringBuffer("A"));
                t.add(new StringBuffer("Z"));
                t.add(new StringBuffer("l"));
                System.out.println(t);
   }
 }
```

Note : if we are depending on default natural Sorting order then compulsory Object should be homogenous and comparable otherwise we will get ClassCastException

An Object is Set to be comparable if and only if the Corresponding interfaces. All the wrapper classes are implemented comparable but not StringBuffer Class

Comparable Interface : It is presented in java.lang.package it existed with only one method

java.lang.Comparable
public int compareTo(Object);

Obj1.compareTo(Obj2) : this method return +ve if obj1 is > obj2 then obj1 has to come after obj2. return –ve if obj1<obj2 then obj1 has to come before obj2. it will return 0 if both are same

System.out.println("A".compareTo("z")); -ve
System.out.println("Z".compareTo("A")); +ve
System.out.println("A".compareTo("A")); 0
System.out.println("A".compareTo(null)); RE NullPointerException
System.out.println("A".compareTo(new Integer(10)); RE //ClassCast Exception

If we are depending on default natural sorting order then Jvm internally It will use ***compareTo( )*** to arrange Object in Sorting Order

Note: Default sorting order doesn't support of StringBuffer class. If we are not satisfied with default natural sorting order java provides u that we can also define our own sorting order
By using Comparator

Comparable meant for default Sorting Order where as Comparator is meant for customized Sorting order

Comparator (I) : Which is existed in java.util.Package and contain 2 methods

java.util.Comparator(i)

public int compare(Object ob1,Object obj2);
return –ve iff Obj1>obj2 the it must be before Obj2
return +ve iff Obj1<Obj2 then it must be after Obj2
return 0 iff Obj1 and Obj2 equals no changes in it's positions

```
public boolean equals(Object obj)

        interface Comparator
       {    compare( );
            equals( );}

public class Test implements Comparable
{
    public int Compare( )
       {
       }
}

class Test
{
    Public static void main(String…x)
    {
        TreeSet t=new TreeSet(new Mycomparator());
                     t.add(10);  // not compare
                     t.add(0);   // compare(0,10); +ve  10, 0
                     t.add(15); // compare(15,10); +ve  15,10,0
                     t.add(5); // compare(5,15); +ve   15,5,10,0 (5,10); 15,10,5,0  (5,0)
                                          15,10,5,0
                     t.add(20); // compare(20,15);□ 20,15,5,10,0
                     t.add(20); //compare(20,20); □0
         System.out.print(t);  // 20,15,10,5,0
    }
}
public class Mycomparator implements Comparator
{
  public int compare(Object o1,Objct o2)
    {
         Integer i1=(Integer)o1;
         Integer i2=(Integer)o2;

          if(i1<i2)
              return +1;
        else if(i1>i2)
               return -1;
        else
            return 0;
    }
}
```

Difference between Comparable and Comparator

Comparable	Comparator
1. it is used for default Sorting Order	1.It is used for Customized Sorting Order
2. presented int java.lang package	2. presented in java.util. package

3. contain with only one function
compareTo( )
	3. contain with two function
	compare( )
	equals( )

4.All wrapper class and String class implemented by comparable interface	4. Only one predefined class implements comparator interface java.text.Collector

Differences between HashSet, LinkedHashSet, TreeSet

Properties	HashSet	LinkedHashSet	TreeSet
Underplaying Data Structure	HashTable	HashTable + LL	Balanced Tree
Insertion Order	Not	Preserved	Non preserved

	preserved		
Sorting Order	Not applicable	Not Applicable	Applicable
Heterogeneous Object	allowed	allowed	By default not allowed
Null Objects Acceptance	allowed	allowed	For empty TreeSet as first element allowed
Duplicate Objects	Not allowed	Not allowed	Not allowed

Map(I) :

- If you want represent a group of values in the form key and value pair then we need to go for Maps.
- Map is not child interface of Collection
- Both key and values are Object only
- Duplicate keys are not allowed but values can be
- Each key an value pair is called One entity

Methods Of Map:

1. ***Object put(Object key,Object Value); To insert One key value pair in the*** Map . if specified Key already existed then the Old values replaced with new value and Old value is return
2. void putAll(Map m) : To insert a group of key and value pair into map
3. Object get(Object key) : it return the value associated with specified key. If key is not existed then it will return null
4. Object remove(Object Key) : To remove the entry which is existed with specified key and return corresponding Values
5. boolean containKey(Object Key)
6. boolean containValues(Object value)
7. boolean isEmpty( )
8. int size( )
9. void clear()
10. set keySet( ) it will return only key's

11. Collection values( ) return the values of map
12. set entrySet( )

Note : getKey( ), getValue( ),setValue(object new) are inner interface of Map

```
  interface Map
      {
          interface Entry
           {
                Object getKey();
               Object getValue( );
               Object setValue(Object new);
           }
      }
```

HashMap:

- Underlying data structure is HashTable
- Duplicate keys are not allowed but values can be duplicate
- Insertion order is not preserved and it is bases of Hashcode of the Key
- Null key is allowed (only once) and null values are allowed
- Heterogeneous key and values are allowed

Differenced between HashMap and HashTable

HashMap	*HashTable*
1. These methods are not synchronized	1.Every method is HashTable is Synchronized
2.HashMap Objects are not ThreadSafe	2.HashTable Objects are ThreadSafe
3.null objects are allowed in both key and value	3.null is not allowed key and values
4.performance is high	4.Performance is low
5.non legacy version from 1.2	5.Legacy Collection from 1.0

Constructor

HashMap m=new HashMap( ): Creates an empty HashMap Object with default initial capacity 19 and default fill ratio Is 0.75

1. ***HashpMap m=new HashMap(int initialCapacity);***
2. ***HashMap m=new HashMap(initial cap,float ratio);***
3. ***HashMap m=new HashMap(Map m);***

```
import java.util.*;
class HashMapDemo
{ public static void main(String args[])
        {
HashMap m=new HashMap();
            m.put("Ramesh",35);
            m.put("Nagesh",30);
            m.put("Venky",50);
                                        System.out.println(m);
System.out.println(m.out("Ramesh",1000));
Set s=m.KeySet();
System.out.println(s);

        Collection c=m.values();
        System.out.println(c);
        Set s1=m.enterySet();
        System.out.println(s1);

         Iterator itr=s1.iterator();
 while(itr.hasnext())
        {Map.Entry m1=(Map.Entry)itr.next();
                                System.out.println(m1.getKey( )+"-"+m1.getValue());
         if(m1.getKey().equals("venky"))
                                                m1.setValue(2000);
         }
   System.out.println(m);
  }
}
```

LinkedHashMap: It is child class if HashMap it is exactly Same as HashMap Except the following Differences

HashMap	***LinkedHashMap***
1. Underlying DS is HashTable	1. Underlying DS is HashTable+LinkedList
2. Insertion order is not preserved	2. Insertion order is preserved
3.introduced in 1.2 version	3. Introduced in 1.4 version

Queue(I) : It is child interface of Collection if you want represent group of individual Objects prior to processing then we need to use Queue

Usually Queue Follows FIFO Order but based on our programming requirements we can implements our own priority Order also

Methods

Boolean offer(Object o) : To add an Object into Queue

Object peek( ) : It return head element of Queue if Queue is empty it will return null

Object element( ) : Return head element of Queue if Queue is empty is will raise Runtime Exception NoSuchelement Exception

Object poll( ) : It will remove head Object and return . If queue is empty it will return null

Object remove( ): it will remove head object and return. If queue is empty it will raise Exception NoSuchElementException

PriorityQueue

- It is Data Structure which can be used to represent a group of individual prior to Processing according to Some prority

- Priority Can be either default natural sorting order or customized Sorting Order.
- Duplicate Objects are not allowed, Insertion order is not preserved

Note: If you are depending default natural Sorting order then the Object must be homogeneous and comparable otherwise ClassCastException

Null insertion is not possible for even has first element also

Constructor

1. ***PriorityQueue q=new PriorityQueue( ); Crate an empty priority Queue with*** default initial capacity with 11 objects and sorting is iff netural Sorting Order
2. ***PriorityQueue q=new priorityQueue(int cap)***

```
class PriorityQueueDemo
{
  Public static void main(String args[])
  {
   PriorityQueue p=new PriorityQueue();
      System.out.println(q.peek());
System.out.println(q.element());

   for(int i=0;i<=10;i++)
    { q.offer(i);}
  System.out.println(q.poll());
System.out.println(q);
  }
}
```