SMP

Java

Need of collections :-

- Array is an indexed collection of fixed no. of homogeneous data elements.
- Advantages of array is we can represent multiple values with the help of single variable so that reusability of the code will be improved.

Limitation of object type array :-

- array is fixed in size. once the array is created with the some size there is no change of increasing or decreasing the size of array
- when we use array compulsory we should know the size in advance which may not possible always

② array can hold only homogeneous data elements

ex. Student [] s = new student [10000];
s[0] = new student(); (right)
s[1] = new customer(); (wrong)

But:
we can resolve above problem by using object array

object [] o = new object [1000];
o[0] = new student();
o[1] = new customer();

- Arrays concepts is not implemented based on some standard data structure hence readymade method support is not available for every requirement. we have to write the code explicitly. which is complexity of programming

To overcome above problem we can use Collection

- collection are growable in nature i.e. based on our requirement we can increase or decrease the size

Collections :-

- collection can hold the both homogeneous & heterogeneous elements.
- every collection class is implemented based on some standard ds.

Q = * Diff bet^n Arrays & collections *

Arrays	collection
① Arrays is fixed in size	collection is growable in size
② wrt to memory are not recommended to use	wrt to memory are recommended to use
③ wrt to performance are recommanded to use	wrt performance are not recommended to use
④ Arrays can hold only homogeneous data	collection can hold homogeneous & heterogeneous data
⑤ readymade method doesn't support	readymade method does support
⑥ primitive & object both are holds	in collection only object are holds.

Q. what is collection & collection framework.

Collection :-

collection is a group of single entity if want to represent group of individual object as a single entity. then we should go for collection.

collection framework :-

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

Java (equivalent in both)	C++
(I) collection	container
(II) collection framework	STL (standard template library)

9 key interface of collection framework.

(I) **collection** :- (1.2v)

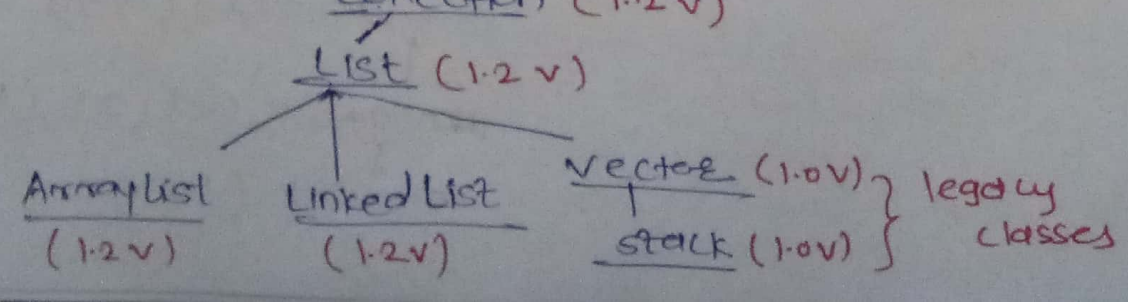
- If we want to represent group of individual object as a single entity then we should go for collection.
- collection interface defines the most common methods which are applicable for any collect^n object
- In general coll^n interface is considered as root interface of collect^n framework

* Diff bet^n collection & collections.

collection	collections
collection is an interface which can be used to represents a group of individual object as a single entity.	collections is an utility class present in java.util. package to define several utility methods (like sorting searching) for coll^n objects.

(II) **List** :- (1.2v)

- List is a child interface of collection.
- If we want to represent a group of individual object as a single entity. where duplicates are allowed & insertion order preserved then we should go for list



The classes which are comming from old generation is called as "legacy classes."

(III) set :-

- set is a child interface of collection.
- If we want to represent a group of individual object as a single entity where duplicates are not allowed & insertion order not preserved then we should go for set.

collection (1.2V)
↓
set (1.2V)
↓
Hashset (1.2V)
↓
LinkedHashset (1.4V)

* Diff betn List & set

List	set
(I) duplicates are allowed	duplicates are not allowed
(II) Insertion ordered preserved	Insertion order not preserved.

(IV) SortedSet :-

- It is the child interface of set
- If we want to represent a group of individual object as a single entity where duplicate are not allowed but all objects should be inserted according to some sorting order then we should go for SortedSet

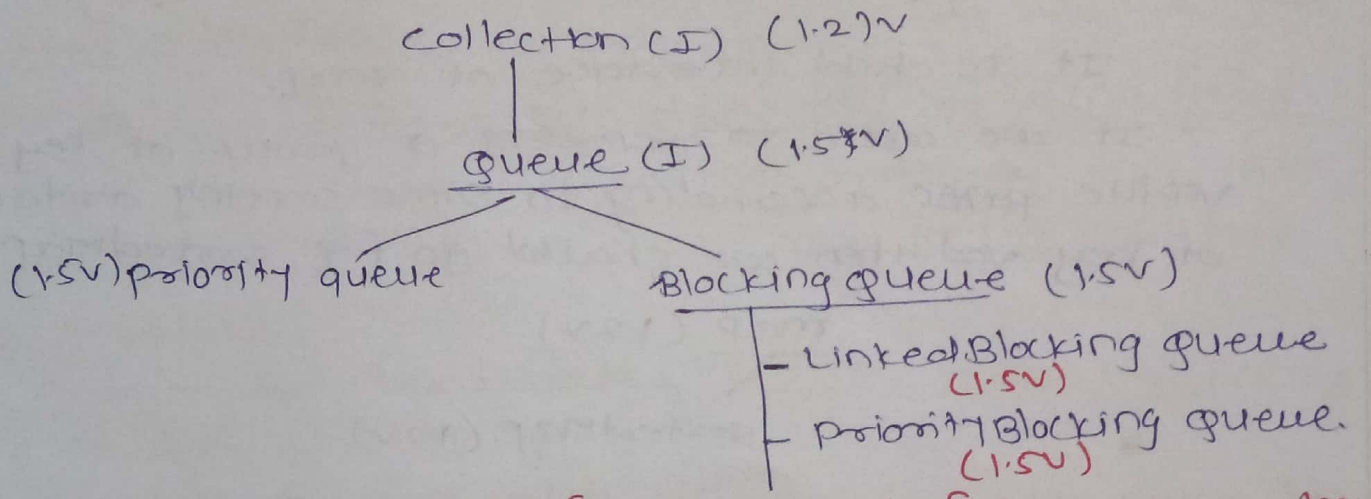
collection (1.2V) (I)
↓
set (1.2) (I)
↓
SortedSet (I) (1.2V)
↓
NavigableSet (I) (1.6V)
↓
Treeset (1.2V)

(V) Navigableset :-

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

(VI) Queue :-

- It is child interface of collection.
- If we want to represent group of individual object prior to processing then we should go for queue.



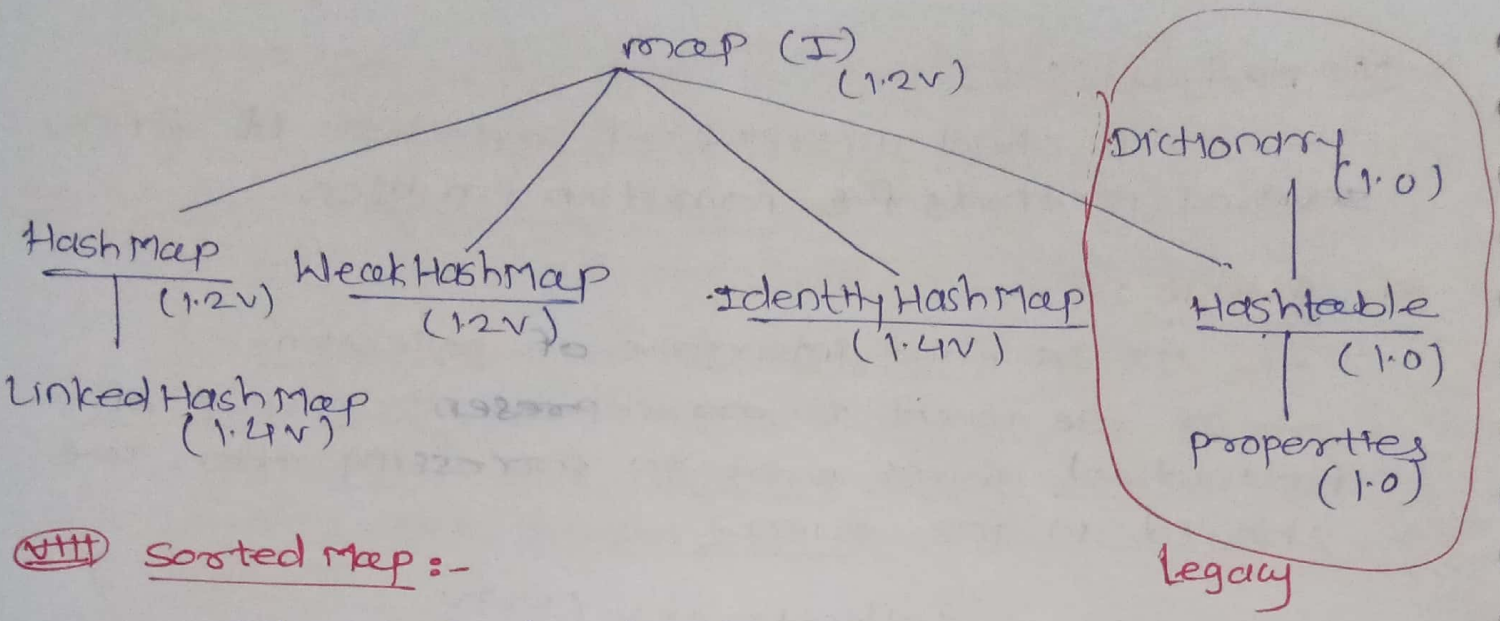
- All the above interfaces meant for representing a group of individual objects.
- If we want to represent a group of object as key value pairs then we should go for map interfaces

(VII) map :-

- map is not the child interface of collection
- If we want to represent group of individual object as key value pairs then we should go for map.
- Both key & values are objects, duplicates key are not allowed but values can be duplicated.

Ex

Roll no	Name
101	"ABC"
102	"DEF"
103	"GHI"



(VIII) **Sorted Map :-**

- It is child interface of Map.
- If we want to represent a group of key value pairs according to some sorting order of key then we should go for SortedMap.

Map (1.2V)

SortedMap (1.2V)

(IX) **Navigable Map :-**

- It is child interface of SortedMap.
- It represent the group of key value pairs. It defines several utility methods for navigation purpose.

Map (I) (1.2V)

SortedMap (1.2V)

NavigableMap (1.6V)

TreeMap (1.2V)

Sorting :-

(I) Comparable()
(II) Comparator()

Utility classes :-

(1) Collections
(2) Arrays

Cursor :-

(I) Enumerator()
(II) Iterator()
(III) ListIterator()

* Collection Interface :-

- In general coll^n interface is considered as root interface of coll^n framework
- coll^n interface defines the most common methods which are applicable for any coll^n object
- collection interface doesn't contain any method to retrieve objects there is no concrete class which implements collection class directly

* List Interface :-

- we can differentiate duplicates by using index.
- we can preserve insertion order by using index, hence index play very Imp role in list interface

(I) ArrayList :-

- The underlined data structure Resizable Array or Growable Array
- Duplicates are allowed.
- Insertion order is preserved.
- Heterogeneous object are allowed
- Null insertion is possible

* usually we can use collection to hold & transfer objects from one place to another place, to provide support for this requirement every collection already implements serializable & cloneable interfaces.

* ArrayList & vector are any random access interfaces so that we can access any random element with the same speed

* Hence if our frequent operation is retrieval operation then ArrayList is the best choice

Random Access :-

- present in java.util package
- It doesn't contain any methods & it is marker interface

ArrayList :-

- ArrayList is best choice if our frequent operation is retrieval operation
  (Because ArrayList implements RandomAccess interfaces)
- Arraylist is the worst choice if our frequent operation is insertion or deletion in the middle
  (Because several shift operation are require)

Q- Difference bet" ArrayList & Vector

ArrayList	Vector
① every method present ArrayList is non-synchronize	every method present in Vector is synchronize
② At a time multiple thread allowed to operate on ArrayList object and hence ArrayList is not thread safe	At a time only one thread allowed to operate on ~~Vector~~ Vector object is thread safe.
③ Thread are not required to wait to operate on arraylist, hence relatively performance is high	Thread are required to wait to operate on Vector object and hence relatively performance is low
④ Introduced in 1.2 version & it is non legacy class	Introduced in 1.0 version and it is a legacy class.

length :- It is final variable applicable only for arrays.
It represent the size of array.

length() :- It is final method applicable for string object
- It returns the no. of characters present in string

Q- How to get synchronized version of ArrayList object?

→ By default ArrayList is object is non-synchronize but we can get synchronized version of ArrayList by using collections class synchronizedList() method.

* Public static List synchronizedList (List l)

Non-synchronized

ArrayList ll = new ArrayList();

Synchronized.

List l = Collections.synchronizedList(ll);

similarly we can get synchronized version of set, map object by using the following methods of collections class.

Public static Set synchronizedSet (set s);

public static map ——— map (map m);

(II) LinkedList :-

- Insertion order is preserved
- Duplicates are allowed
- Heterogeneous object are allowed
- Null insertion is possible
- The underlying data structure is double LinkedList
- LinkedList implements serializable & clonable interfaces but not RandomAccess interfaces.
- LinkedList is best choice if our frequent operation is insertion or deletion in the middle
- LinkedList is the worst choice if our frequent operation is retrieval operation.

- usually we can use LinkedList to implement stacks & queues to provide support for this requirement LinkedList class defines following specific methods

```
void addFirst();
void addLast();
Object getFirst();
object getLast();
```

LinkedList constructor :-

* LinkedList l1 = new LinkedList();
* creates an empty LinkedList object.

* LinkedList l1 = new LinkedList(collection c);
- creates an equivalent LinkedList object for the given collection.

* Difference bet^n ArrayList & LinkedList

ArrayList	LinkedList
① It is best choice if our frequent operation is retrieval.	It is best choice if our frequent operation is insertion & deletion
② ArrayList is the worst choice if our frequent operation is insertion & deletion	LinkedList is the worst choice if our frequent operation is retrival operation.
③ underlying data structure for ArrayList is resizable or growable array	underlying data structure is double linked List.
④ ArrayList implements RandomAccess interface	LinkedList doesn't implement RandomAccess interface

⑭ Vector :-

- underlying data structure for the vector is resizable array or growable array
- duplicate objects are allowed
- insertion order is preserved
- null insertion is possible
- Heterogeneous object are allowed
- Best choice if the frequent operation is retrieval
- Vector object is thread safe.
  most of the method present in vector are synchronized.

- vector class implemented serializable, ~~cloneable~~ cloneable & RandomAccess interfaces.

* constructors of vector class :-

① vector v = new vector ();

- creates an empty vector object with default initial capacity 10, once vector reaches its max capacity a new vector object will be created with new capacity = 2 * current capacity.

② vector v = new vector (int initial capacity);

- creates an empty vector object with specified initial capacity

③ vector v = new vector (int initialCapacity, int increment Capacity);

④ vector v = new vector (Collection c);

- creates an equivalent ~~obj~~ vector object for the given collection

* ⑤ Stack

- It is child class of vector.
- It is specially designed class for LIFO

Constructor of Stack

Stack s = new Stack ();

① object push (object obj);

- For inserting an object to the stack

② object POP ()

- For removes & returns top of the stack

③ object peek ();

to return the top of the stack without removal of object

④ int search (object obj)

- if the object is available it returns its offset from top of the stack.
- If the object is not available then it returns -1

* Three cursors of Java :-

Types:

① Enumeration ② Iterator ③ ListIterator.

- If we want to retrieve object one by one from the collection, then we should go for cursor.

① Enumeration:-

We can create Enumeration object by using elements() method of Vector class.

public Enumeration elements();

Ex Enumeration e = v.elements();

Enumeration has two methods.

① public boolean hasMoreElements();

② public object nextElement();

Limitation:-

- Enumeration concept is applicable only for legacy classes & hence it is not universal cursor
- By using enumeration we can get only read access & we can't perform remove operation.

To overcome above limitations of enumeration we should go for Iterator.

② Iterator :

- We can apply Iterator concept for any collection object hence it is universal cursor.
- By using iterator we can perform both read & remove operations.
- we can create Iterator object by using iterator() method of collection interface.

public Iterator iterator();

Ex iterator itr = c.iterator();

* where c is any collection object.

Iterator interface def^n the following three methods.

(I) public boolean hasNext()
(II) public object next()
(III) public void remove().

* Limitation:-

- By using enumeration & Iterator we can move only forward direction. & we can't move to the backward direction & hence these are single direction cursors.
- By using iterator we can perform only read & remove operations & we can't perform replacement of new objects.

To overcome above limitations of iterator we should go for ListIterator

(III) ListIterator:-

→) By using ListIterator we can move either to the forward direction or to the backward direction, & hence ListIterator is bidirectional cursor.

) By using ListIterator we can perform replacement & addition of new objects in addition to read & remove operations.

- we can create ListIterator object by using listIterator method of List interface.

Public ListIterator listIterator()

Ex: ListIterator itr = l.listIterator();

* l is any List object.

- ListIterator is the child interface of Iterator & hence all the method of Iterator

Methods of ListIterator:

Forward direction:-

1. public boolean hasNext()
2. public void next()
3. public int nextIndex()

Backward direction:-

4. public boolean hasPrevious()
5. public void previous()
6. public int previousIndex()

other capability methods:-

7. public void remove()
8. public void set(object new)
9. public void add(object new)

Note:-

List Iterator is the most powerful cursor but it's ~~limitation~~ limitation is, it is applicable only for List implemented class object & it is not a universal cursor.

* comparision of three cursors.

Iterable	Iterator
- It is related to for-each loop.	It is related to collection
- The targate element in foreach loop should be iterable	we can use iterator to get objects one by one from the collection.
- Iterator present in java.lang package	It is present in java.util package
- It contains only one method Iterator()	It contains three methods, hasNext(), next(), remove().

* Set *

- set is a child interface of collection
- If we want to group of individual object as a single entity, where duplicates are not allowed & insertion order is not preserved then we should go for set.
- set interface doesn't contain any new methods. So we have to use only collection interface methods.

* Hashset (I):-

- The underlying data structure is Hashtable.
- Duplicates are not allowed. If we are trying to inserting duplicates, we wont get any compiletime or runtime error. add() method simply returns false
- Insertion order is not preserved and all objects will be inserted based on hash-code of objects.
- Heterogeneous objects

* Constructors of HashSet.

① Hashset h = new Hashset();
- creates an empty Hashset object with default initial capacity 16 & default fill ratio 0.75

② HashSet h = new HashSet(int initialCapacity);
- creates an empty Hashset object with specified initial capacity & default fill ratio 0.75.

③ Hashset h = new Hashset(int initialCapacity, float loadFactor);
- creates an empty HashSet object with specified initial capacity & specified loadFactor (or Fill Ratio)

④ Hashset h = new HashSet(Collection c);
- For inter conversion bet^n coll^n objects.

Load Factor / Fill Ratio :-

After loading the how much factor, a new Hashset object will be created, that factor is called as load Factor or Fill Ratio.

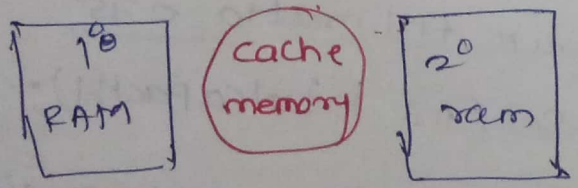
① LinkedHashSet :-

- It is child class of hashset.
- introduced in 1.4V
- It is exactly same as Hashset except the following differences.

Hashset	LinkedHashSet.
① the underlying data structure is Hashtable	- The underlying data structure is Hashtable + Linked List (It is hybrid ds)
② Insertion order is not preserved	- Insertion order is preserved.
③ Introduced in 1.2V	- Introduced in 1.4V.

* where we use ~~this~~ LinkedHashSet

Ex Cache based



Note :-

LinkedHashSet is the best choice to develop cache based applications, where duplicates are not allowed & insertion order must be preserved

* SortedSet (I) :-

- (I) It is the child interface of set
- If we want to represent group of individual objects then according to some sorting order & duplicates are not allowed then we should go for sortedSet.

- Default natural sorting order for number ascending order & for string alphabetical order.
- We can apply the above methods only on sortedset implemented class objects. that is on the Treeset object.

* Treeset :-

- The underlying ds for Treeset is Balanced Tree.
- Duplicate object are not allowed.
- Insertion order not preserved, but all objects will be inserted according to some sorting order.
- Heterogeneous objects are not allowed. If we are trying to insert heterogeneous objects then we will get RE. saying ClassCastException.
- NULL insertion is allowed but only once

Treeset constructor.

(1) Treeset t = new Treeset ();
- creates an empty treeset object where elements will be inserted according to default natural sorting order

(2) Treeset t = new Treeset (Comparator c);
- creates an empty treeset object where elements will be inserted according to customized sorting order.

(3) Treeset t = new Treeset (SortedSet s)

(4) Treeset t = new Treeset (Collection c);

Null Acceptance :-

- For empty Treeset as the first element null insertion is possible. But after inserting that null if we are trying to insert any another element we will get NullPointerException
- For Non-empty treeset if we are trying to insert Null then we will get NullPointerException

Ex

```
import java.util.Treeset;
class TreesetDemo1{
    psvm(String[] args){
    Treeset t = new Treeset();
     t.add(new StringBuffer("A"));
     t.add(new ----------("Z"));
     t.add(----------("L"));
     t.add(----------("B"));
     So.pln(t); // ClassCastException
    }
}
```

Note:-

If we are depending on default natural sorting order then objects should be homogeneous & comparable other wise we will get runtime exception saying ClassCastException.

- If object is said to be comparable if and only if the corresponding class implements java.lang.Comparable interface
- String class & all wrapper classes already implement Comparable interface. But StringBuffer doesn't implement Comparable interface.

* Comparable (Interface):-

This interface present in java.lang.package it contains only one method CompareTO().

Public int CompareTO (object obj)

Ex

obj1.CompareTO(obj2)

- returns -ve if obj1 has to come before obj2
- returns +ve " to " after "
- return 0 if obj1 & obj2 are equal.

Ex

```
class Test {
    psvm(string[] arg){
        s.o.pln("A".compareTO("Z")); // -ve
        s.o.pln("Z".compareTO("A")); // +ve
        s.o.pln("A".compareTO("A")); // 0
        s.o.pln("A".compareTO(null)); // NPE
    }
}
```

If we depending on default natural sorting order internally jvm will call compareTO() method while inserting object to the treeset. Hence the objects should be Comparable

```
Treeset t = new Treeset();
t.add("B");
t.add("Z"); // "Z".compareTO("B"); +ve
t.add("A"); // "A".compareTO("B"); -ve
t.add("a");
s.o.pln(t); // [A, B, Z]
```

Note:

If we are not satisfy with default natural sorting order or if the default natural sorting order is not already available then we can def" our own "customized sorting" by using "Comparator"

Comparable	Comparatee
① It is meant for default natural sorting order	It is meant for customized sorting order
② It is present in java.lang package	It is present in java.util package.
③ This interface defn only one method CompareTO()	This interface defn two methods compare() & equals().
④ All wrapper classes & string class implements Comparable interface	The only implemented classes of comparatee are Collatee & RuleBased Collatee

* Treeset *

WAP to insert integer objects into the treeset where the sorting order is descending order.

→
```
import java.util.*;
class TreesetDemo{
    P S V m(String[] args){
    Treeset t = new Treeset(new MyComparatee());
      t.add(10);
      t.add(0);
      t.add(15);
      t.add(20);
      t.add(20);
      S.o.pln(t);
}}
```

```
class comparatee implements comparatee {
    Public int compare (object obj1, object obj2) {
        integer i1 = (integer) obj1;
        Integer i2 = (integer) obj2;
        if (i1 < i2)
            return +1;
        elseif (i1 > i2)
            return -1;
        else
            return 0;
    }
}
```

db → a
jave
project
dls

Integer objects into Treeset, descending order.

```
Treeset t = new Treeset(new myComparatee[]);   line (1)
t.add(10);
t.add(0);  → +ve  compare(0,10);
t.add(15); → -ve  ——→ (15,10);
t.add(20); → +ve  ——→ (20,10);
t.add(20)  → -ve  ——→ (20,15);

t.add(20); → +ve → compare(20,10);
t.add(20); → -ve → ——→ (20,15);
t.add(20); — 0 → ——→ (20,20);
s.o.pln(t); // [20,15,10,0]
```

- At line-1 if we are not passing comparatee object then internally jvm will call compareTo() method which meant for default natural sorting order (ascending)
  - In this case o/p is [0, 10, 15, 20].
- If we are passing comparatee object then internally jvm will call compare() method which is meant for customized sorting (Descending order)
  In this case o/p is [20, 15, 10, 0]

Various possible implementations of compare() method:

```
class MyComparatee implements Comparatee {
    public int compare(object obj1, object obj2) {
        Integee I1 = (Integee) obj1;
        Integee I2 = (Integee) obj2;
        // return I1.compareTo(I2); [0,10,15,20] ascending
        // return -I1.compareTo(I2); [20,15,10,0] descending
        // return I2.compareTo(I1): [20,15,10,0] descending
        // return -I2.compareTo(I1); [0,10,15,20] ascending
        // return +1; [10,0,15,20,20] insertion order.
        // return -1; [20,20,15,0,10] reverse of insertion
        // return 0; [10]
    }
}
```

* WAP to insert String objects into the Treeset where sorting order is Reverse of alphabetical order.

```
Import java.util.*;
class Treeset {
    public static void main(String[] args) {
        Treeset t = new Treeset(new MyComparatee());
        t.add("Akshay");
        t.add("Roja");
        t.add("shobharani");
        t.add("Rajakumar");
        t.add("Gangabhavani");
        t.add("Ramulamma");
        Sop(t);
    }
}
```

```
class MyComparatee implements Comparatee {
    public int compare(object obj1, object obj2) {
        String s1 = obj1.toString();
        String s2 = (String) obj2;
        // return s2.compareTo(s1);
        return -s1.compareTo(s2);
    }
}
```

If we are defining our own sorting by comparator, the objects need not be comparable

WAP to insert string & StringBuffer objects into the Treeset where sorting order is -increasing length order if two objects having the same length then consider their alphabetical order ?

```
import java.util.*;
class Treeset{
P S V M {String [] ,args){
Treeset t = new Treeset (new comparatoe());
t.add ("A");
t.add (new StringBuffer ("ABC"));
t.add (new StringBuffer ("AA");
t.add ("xx");
t.add ("ABCD");
t.add ("A");
S.O.Pln(t);
}
}
```

```
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;
elseif (l1>l2)
    return +1;
else
return S1.compareTO(S2);
}
}
```

O/P- [A, AA, xx, ABC, ABCD]

- If we are depending on default natural sorting order then objects should be homogeneous & comparable otherwise we will get runtime exception.

- But if we are defining our own sorting by comparator then objects need not be homogeneous & comparable. We can insert heterogeneous non-comparable objects also.

- For predefined comparable classes like String default natural sorting order already ~~already~~ available. If we are not satisfied with that, we can defined our own sorting comparator object

- For predefined non-comparable classes like String Buffer, default natural sorting order is not already available. We can defined required sorting by implementing comparator interface.