List of Concepts Involved:

- Why Collection?
- Collection Hierarchy
- ArrayList
- LinkedList
- ArrayDeque
- PriorityQueue
- TreeSet
- HashSet
- LinkedHashSet
- Iterator , List Iterator
- Legacy classes and Enumeration

Array: Syntax: <D.type> <array name>[] = new D.type<length>

ex: int arr[] = new int(5);

Student st[] = new Student[10];

☆ array to store 10 Student class objects.

Disadvantages of Array

① size is fixed

② only store homogeneous type of data

③ no inbuilts methods (for sorting, deleting, adding etc)

Collection: -① Set of interfaces and classes to store the data & and these classes has inbuilt methods to add, delete, retrive and manipulate the data

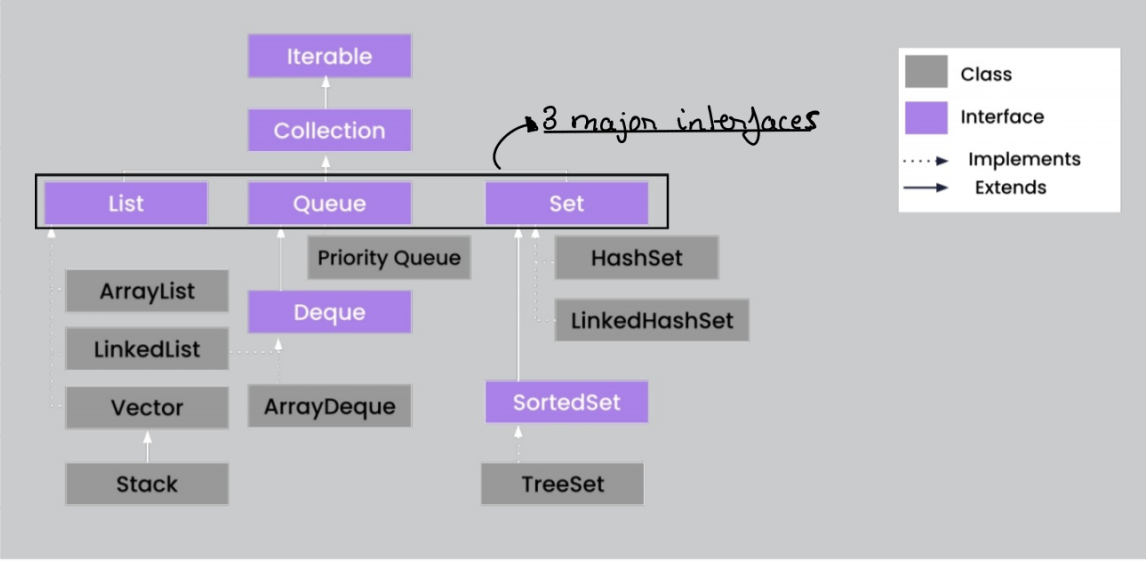
② Can store both homogenous & heterogeneous (mixed) data can be stored.

③ Dynamic in size

④ Introduced in java 1.2

⑤ all data stored as objects.

Collection Hierarchy :-



- List, Queue & Set :- are 3 major interfaces in collection class they extends Collection Class and Collection also has Iterable as its Parent class.

- Arraylist, Linkedlist & Vector : implements linked interface and Stack class extends vector class.

- Priority Queue : it implements Queue interface
- Dequeue : it extends Queue interface
- Arraydequeue :- implements Dequeue interface

- Hashset, Linkedhashset :- both classes implements set interface
  SortedSet, NavigableSet :- both interfaces extends set interface
  TreeSet :- this class implements SortedSet.

Arraylist, Linkedlist (List), Priorityqueue (Queue), Arraydequeue (Dequeue) Hashset, Linkedhashset (Set), TreeSet (SortedSet) : all these classes are important for exam and also for understanding collection. All these classes have their own specific data structures and algorithms.

and they also diff. inbuilt method. Some methods are common among them which are inherited from Collection interface.

1) Arraylist :-

To use a collection class 1st we have to make an object of that class

Suitable for adding data at the same end.

```java
public class Collection1 {
    Run | Debug
    public static void main(String[] args) {
        //creating Arraylist object
        ArrayList al1 = new ArrayList<>();

        //adding adata to ArrayList
        al1.add(e:100);
        al1.add(e:200);
        al1.add(e:300);
        System.out.println(al1);


        //creating another Arraylist
        ArrayList al2 = new ArrayList<>();

        //adding heterogenous data
        al2.add(e:"asit");
        al2.add(e:100);
        al2.add(e:'q');
        al2.add(e:20.213);
        System.out.println(al2);


        /*Adding one collection to another collection */
        al1.addAll(al2);
        System.out.println(al1);

        //adding data at a specific location
        al1.add(index:4,element:"Shastri");
        System.out.println(al1);

        //adding a Arraylist inside a Arraylist into a specific index
        ArrayList al3 = new ArrayList<>();
        al3.add(e:1);
        al3.add(e:2);
        al3.add(e:3);
        al3.add(e:4);

        al1.add(index:5,al3);
        System.out.println(al1);
    }
}
```

not recommended

```
[100, 200, 300]
[asit, 100, q, 20.213]
[100, 200, 300, asit, 100, q, 20.213]
[100, 200, 300, asit, Shastri, 100, q, 20.213]
[100, 200, 300, asit, Shastri, [1, 2, 3, 4], 100, q, 20.213]
PS C:\Users\Euphor\Documents\JAVA\15 CollectionFramework>
```

all though Array list can be used to add data on another Arraylist to a specific index but, it is not advisable.

Use "Linked List" for such purpose of adding data at a specific index.

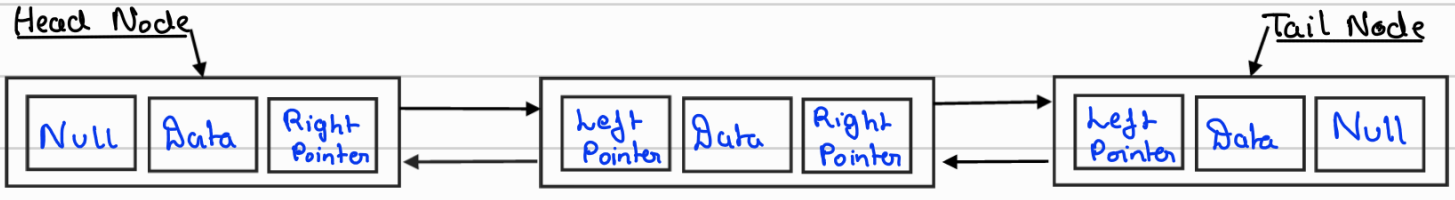
Linked List:-

- implements both Arraylist class and List interface
- follow "Doubly Linked List Data-structure"
- Therefore suitable for insertion type data manipulation
- Duplicates are allowed
- mentains insertion order

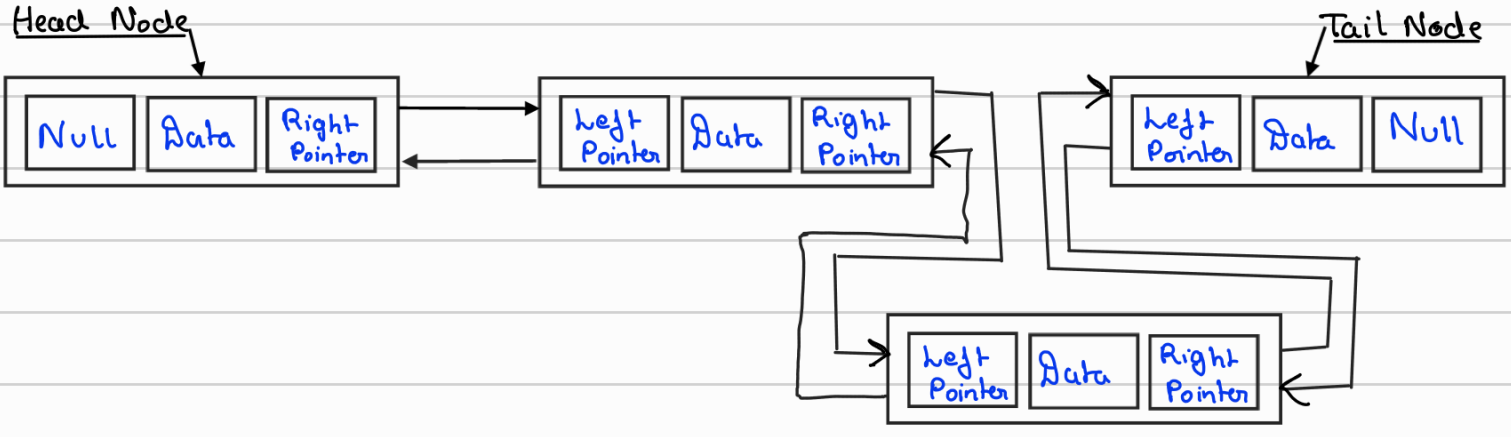
Doubly Linked List :- In doubly linked list data is stored in nodes. Each node contains 3 parts :-

1. Data
2. Left pointer/reference
3. Right pointer/reference

Ex:-



now adding another data inbetween



Observation :- As there is no need for shifting the entire data to insert a value ∴ LinkedList is best suited for insertion type of operation.

```java
import java.util.LinkedList;

/**Linked List 1:
 * LinkList implements both ArrayList class and List interface So common inbuilt methods can be found apart from some specialized methods
 * LinkList follow "Doubly LinkedList Data-Structure"
 * Suitable for insertion type manipulation cuz no shifting and data is stroeed as nodes.
 * Duplicates are allowed
 * mentains insertion order
 * mentains insertion order
 */
public class LL1 {
    public static void main(String[] args) {

        /*Creating LinkedList object*/
        LinkedList ll1 = new LinkedList();
        ll1.add(100);
        ll1.add(200);
        ll1.add("Asit"); //stores heterogenous data
        ll1.add(412.33);
        System.out.println(ll1);//order of insertion mentained


        LinkedList ll2=new LinkedList();
        ll2.add(10);
        ll2.add(20);
        ll2.add(30);
        System.out.println("ll2= "+ll2);

        /*add Fisrt and Last data */
        ll2.addFirst("Asit");
        ll2.addLast(2.36);
        System.out.println("First and Last data added: "+ll2);

        /*adding data at specific index */
        ll2.add(1, "Shastri");
        System.out.println("added data at index 1: "+ll2);

        /*peak() Method: Shows first data without removing it*/
        System.out.println("Peak at First data: "+ll2.peek());
        System.out.println("Peak at last data: "+ll2.peekLast());

        /*poll() Method: Shows first data and removes it from the list */
        Object a = ll2.poll(); //**imp**returns object cuz all collection framewors stroes data as objects
        System.out.println("poll() Method removed: "+a);
        System.out.println("ll2 after poll(): "+ll2);

        /*getting element from index*/
        System.out.println("get element in index 2: "+ll2.get(2));

        /*find index of an element*/
        System.out.println("Index of 'Asit': "+ll2.indexOf("Asit")); //-1 mean not in the list
        System.out.println("Index of 'Asit': "+ll2.indexOf("Shastri"));

        /*Last index of: if duplicate data is there */
        ll2.add(3,30);
        System.out.println("ll2= "+ll2);
        System.out.println("index of 30: "+ll2.indexOf(30));
        System.out.println("Last index of 30: "+ll2.lastIndexOf(30)); //this method shows last index if there are duplicate values

        /*Get first and last element */
        System.out.println("First element: "+ll2.getFirst()+" Last element: "+ll2.getLast());

        /*push() Method */
        ll2.push(11); //pushes 11 at the begining
        System.out.println("ll2 after pushing 11: "+ll2);

        /*pop() Method: pulls the recent pushed object */
        ll2.pop();
        System.out.println("pop: "+ll2);
        ll2.pop();
        System.out.println("pop again: "+ll2);




    }
}
```

Array Deque:

```java
import java.util.ArrayDeque;

/** ArrayDeque:
 * This class implements Deque interface and Deque extends Queue interface so it contains bothd Deque and Queue interface's inbuilt methods
 * Index based accessing not allowed: can't access or add data based on index
 * Follows "Double ended queue Data-structure" : i.e inserction and deletion can be performed from both front and rare end.
 * dublicates are allowed
 * can store heterogenous type of data
 */
public class ADQ1 {
    public static void main(String[] args) {
        /*First ArrayDequeue object*/
        ArrayDeque adq1 = new ArrayDeque();

        /*add() Method: to add data */
        adq1.add(100);
        adq1.add(200);
        adq1.add(300);
        System.out.println("adq1= "+adq1);

        /*Inserting last and first data*/
        adq1.addFirst("Asit");      //index based insertion not allowed
        adq1.addLast("Shastri");   //So, can only add data in rare and front end

        /*can hold both homogenous and heterogenous data */
        System.out.println("adq1= "+adq1);

        /*Offer() Method: Data added using offer method might get added or not  */
        adq1.offer("Offer");
        adq1.offerLast("Last_Offer");
        adq1.offerFirst("First_Offer");
        System.out.println("adq1= "+adq1);
    }
}
```

Double Ended Queque:- it a type of queue where people can enter from front and rare end both, can't enter inbetween.

Priority Queue:

follows "Minimum Heap Data-structure" and stores data based on Complete Binary Tree (CBT).

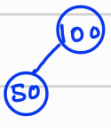
Q) Explaing the output if 100, 50, 150, 25, 75, 125, 175 are added to the Priority queue.

Sol^n → Priority queue uses Min-Heap structure to store the data and provide the highest priority data the the left most side.

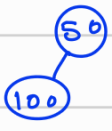
Using CBT to show Min Heap process.

add (100) :-

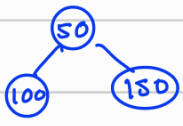
add (50) :-



as 50 < 100 ∴ swapping

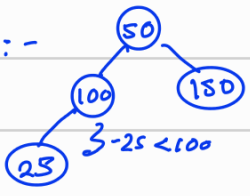


add (150) :-

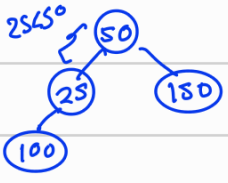


no swapping as 150 > 50

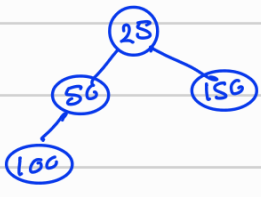
add (25) :-



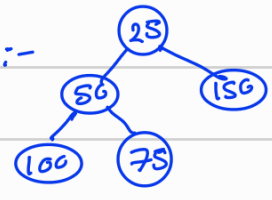
swap



swap

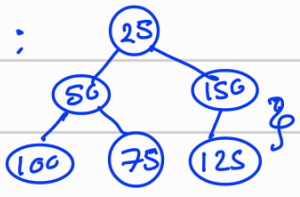


add (75) :-



, no swapping

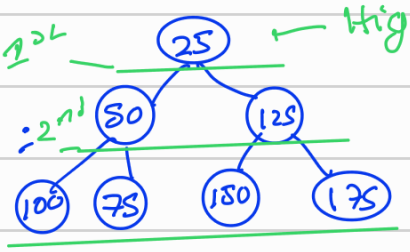
add (125) :



Swap

1st ← Highest Priority element

add (175) :



: no swapping

Op =

op= [25, 50, 125, 100, 75, 150, 175]

```java
import java.util.PriorityQueue;

/**PriorityQueue:
 * This class implements Queue interface.
 * follows: "Minimun Heap Data-Structure"
 * Order of insertion is not preserved. stores data bsed on "Complete Binary Tree(CBT)."
 * duplicates are allowed
 * only hopmogenous number data is allowed
 * "null" type of data is also not allowed
 */
public class PQ1 {
    public static void main(String[] args) {
        /*First PriorityQueue Object */
        PriorityQueue pq = new PriorityQueue<>();

        pq.add(100);
        pq.add(50);
        pq.add(150);
        pq.add(25);
        pq.add(75);
        pq.add(125);
        pq.add(175);
        System.out.println("pq= "+pq);

        /*duplicates are allowed */
        pq.add(50);
        System.out.println("Duplicate 50 added: pq= "+pq);

        /*try to add heterogenous data */
        try{
            pq.add("asit");
            System.out.println("asit added: "+pq);
        }
        catch(Exception e)
        {
            System.out.println("Can't give heterogenous value to PriorityQueue");
        }
    }
}
```

TreeSet :- TreeSet class implements SortedSet interface and SortedSet interface further extends to Set interface.

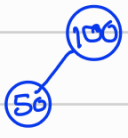
- insertion order is not preserved
- data inserted in sorted order
- follow :- Binary Search Tree internally

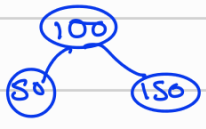
Q) Explain how TreeSet uses Binary Search Tree to store the data and Also show how this data is shown in sorted order.

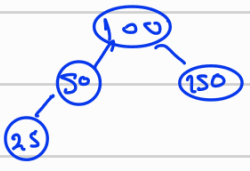
input data :- 100, 50, 150, 25, 75, 125, 175

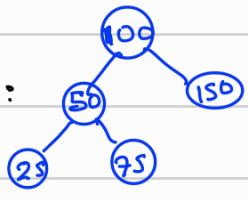
Sol^n -> "Binary Search Tree" : In BST the less value is added on the left while greate value is added on the right

add(100): 

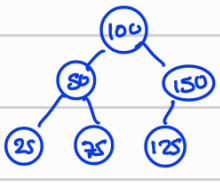
add(50): as 50<100 ∴ on right side 

add(150): as 150>100 ∴ on left side 

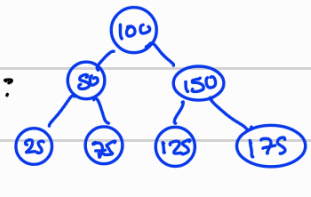
add(25) :- as 25<100; 25<50, => 

add(75): 

∴ 75<100 (right); 75>50 (Left)

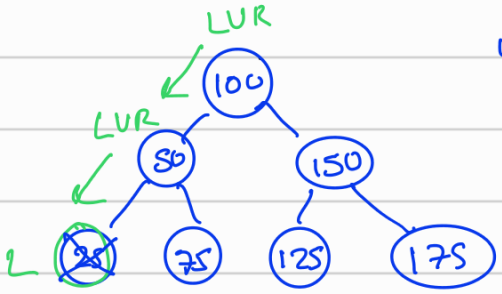
add(125): 

∴ 125>100 (Left)
125<150 (right)

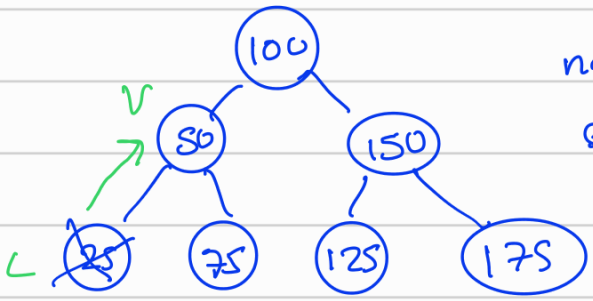
add(175): 

∴ 175>100 (left)
175>150 (left)

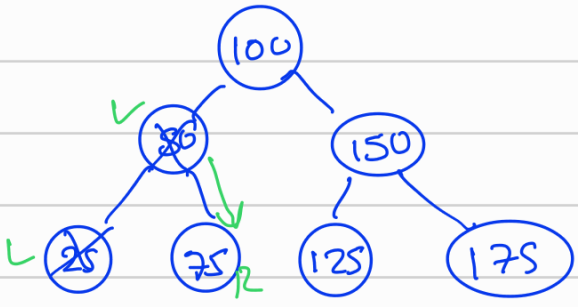
Now, to store this Complete Binary Tree in sorted order TreeSet uses "Left value Right (LVR)" method to extract sorted data



using LVR move left until reach last left value
=> store (25) - L

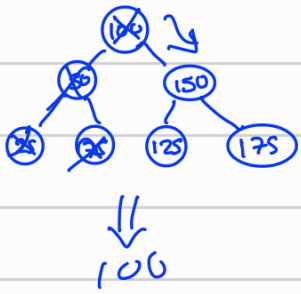


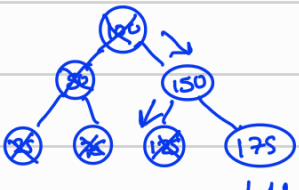
now move back and store value
=> store (50) - V



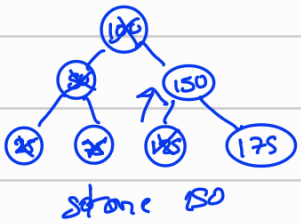
now from V move to right and store it
=> (75) - R

if you are standing on a value to which we have no value no value on left store that value and move right

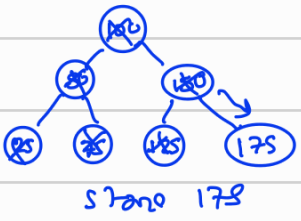




there is a left value of 150 :. move to 125 no further value after 125 :. store 125



store 150

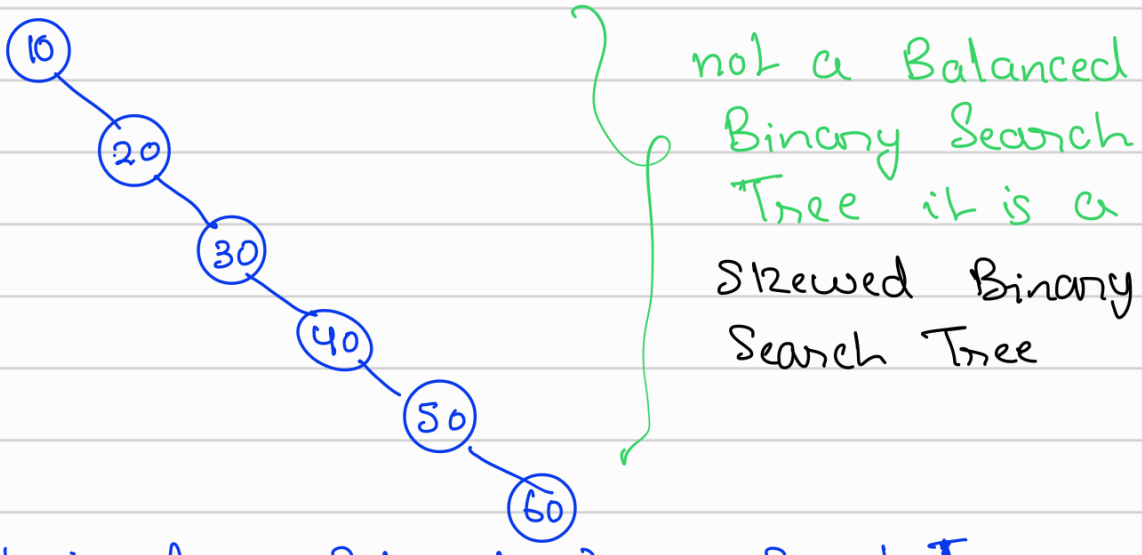


store 175

o/p = [25, 50, 75, 100, 125, 150, 175]

Q. What happens data added in order 10, 20, 30, 40, 50, 60?

Sol^{n} → Binary Search Tree :-



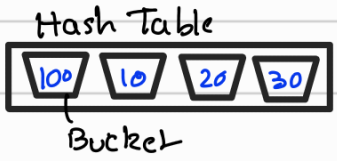
as it is not a Balanced Binary Search Tree
∴, searching process is not efficient as in case of Balanced Binary Search Tree
cuz

in BBST :- 7 elements → 3 components ≥ to search
in SBST :- 7 elements → 7 components to search

```java
import java.util.TreeSet;
/**TreeSet:
 * follows: Binary Search Tree
 * insertion order is not preserved
 * data inserted in sorted order(using "Left value right/LVM" method)
 * TreeSet on initializing can be made heterogenous(with string and int )
 */
public class TS1 {
    public static void main(String[] args) {
        /*First TreeSet Object: */
        TreeSet ts = new TreeSet();
        ts.add(100);
        ts.add(50);
        ts.add(150);
        ts.add(25);
        ts.add(75);
        ts.add(125);
        ts.add(175);
        System.out.println("First TreeSet: "+ts);

        /*Tress set with heterogenous value but only when initializing */
        TreeSet ts2 = new TreeSet();
        ts2.add("asit"); //ca add heterogenous value but only on initializing
        ts2.add("50");
        // ts2.add(3.14);
        System.out.println(ts2);

        /*TreeSet with only float value */
        TreeSet ts3 = new TreeSet();
        ts3.add(2.46);
        ts3.add(3.33);
        ts3.add(1.34);
        System.out.println(ts3);

        /*higher and lower Methods: */
        System.out.println("**************************higher and lower method");
        System.out.println("higher of 50: "+ts.higher(50));
        System.out.println("Lower of 50: "+ts.lower(50));
        System.out.println("higher of 40: "+ts.higher(40));
        System.out.println("lower of 40: "+ts.lower(40));

        /*ceiling and floor methods: Only Shows higher value if not present in list */
        System.out.println("**************************ceiling and floor
method***************************");
        System.out.println("ceiling of 50: "+ts.ceiling(50));   //shows same no
        System.out.println("floor of 50: "+ts.floor(50));       //shows same no.
        System.out.println("ceiling of 40: "+ts.ceiling(40));   //works
        System.out.println("floor of 40: "+ts.floor(40));       //works(cuz 40 is not in
treeset)
    }
}
```

HashSet and LinkedHashSet

- HashSet class implements set interface (java 1.2)
- LinkedHashSet extends HashSet class (java 1.4)
- index based accessing not allowed
- Order of insertion not preserved
- Duplicates values not allowed
- follows "Hash Table Algo" to store



Hash Table :- A hash table contains lots of locations that are also called "buckets" where data is stored. 1 bucket contains only one data and no dublicates in a bucket.

Default no. of Buckets in hash table are :- 16

load factor = 0.75 or 75%.

∴, when Hash table 75% filled then the bucket size double of the current size. no need of automatically increasing size

LinkedHashSet :- (java 1.4)

follows both LinkedList & HashSet ∴,

Order of insertion is preserved

load factor = 0.75 / 75%.

Array List

for adding at rare end

index based insertion possible but frowned upon

duplicates allowed

maintains insertion order

Linked List

follows "doubly linked list"

fare index base insertion

duplicates allowed

maintains insertion order

Array Dequeue

follows doubly ended queue

index based accessing not allowed

dublicates allowed

maintain insertion order

Priority Queue

follows Heap Data Structure

index based accessing not allowed

duplicates allowed

doesn't maintain insertion order

Tree Set

follows "Binary Search Tree"

index based accessing not allowed

dubicates not allowed

insertion order not preserved

Hash Set & Linked Hash Set

follows "Hash Table Algo"

index based accessing allowed in case of LHS

dublicates not allowed

doesn't maintain insertion order

Iterator and ListIterator:

printing and accessing data from a collection is a different task

we use Iterator and ListIterator concept for accessing the data

for-loop can also be used for accessing collection data

Ex:1 using for-loop

```
for (int i=0; i<al.size(); i++)
{
    // creating object for fetching cuz collection stores data as objects.
    Object o = al.get(i);
    Sop (o);
    'or'
    Sop (al.get(i));
}
```

using for-each loop:-

```
for (Object o : al)
{
    Sop (o);
}
```

Disadvantages of using loop for fetching/accessing data:-

1. not all collection support index based accessing
2. If someone is accessing the data at the same time loop traversing is going on then in that case the loop will not stop and go into infinite loop. (Fail fast and fail safe concept)

Cursor Method:- also called Iterator is the best way for accessing collection data

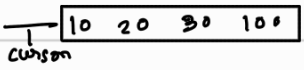
- Iterator itr = al.iterator(); → provides us with a cursor at the beginning of the collection.

- itr.hasNext(); Checks wheather there is data at the next position if Yes then returns true
- itr.Next(); fetches the data if itr.hasNext(); returns true.

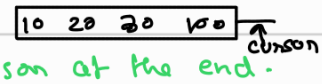
Example :- Iterator itr = al.iterator();

```
while (itr.hasNext())   // works while itr.hasNext() == true
{
    Integer i = (Integer) itr.Next();
                down casting itr.Next() object value to an integer.
    sop(i);
}
```

List-Iterator :- ListIterator litr = al.listIterator(); // cursor at begining



ListIterator litr = al.listIterator(al.size()); moves cursor at the end.



- It can access in reverse direction also
- only present in list List interface only.

Fail fast vs Fail Safe

Fail fast :- In this situation the moment we attempt to achieve concurrent modification while accessing the data the program will fail showing "Concurrent Exception"

Example:-

a) Concurent modification using for-loop :-

```
for(int i=0; i<al.size(); i++)
{
    Sop(al.get(i));
    al.add(100);
}
```

⇓ op

infinit loop

b) Concurrent modification using iterator :-

```
Iterator itr = al.iterator();
while (itr.hasNext())
{
    Sop(itr.Next());
    al.add(100);
}
```

⇓ o/p

Concurrent Exception

Fail Safe use Fail Safe when you want to achieve concorent modification without getting exception.

- Use Concurrent Package :- java.util.concurrent.*;
- Use CopyOnWrite <Collection_name> class to create collection object

Example :- concorrent modificat with failsafe on Array list.

```
CopyOnWriteArrayList al = new CopyOnWriteArrayList();
al.add(10);
al.add(20);
al.add(30);
al.add(40);
Iterator itr = al.iterator();
while (itr.hasNext())
{
    Sop(itr.Next());
    al.add(100);
}
```

output

10
20
30
40

☆ Showing result without giving exception and ignoring concurrent Exception.

List of Concepts Involved:

- Why Collection?
- Collection Hierarchy
- ArrayList
- LinkedList
- ArrayDeque
- PriorityQueue
- TreeSet
- HashSet
- LinkedHashSet
- Iterator , List Iterator
- Legacy classes and Enumeration

Why Collection?

1. They are growable in nature(we can increase and decrease)
2. They can hold both heterogeneous and homogeneous data elements
3. Every collection class is implemented using some standard data structure, so ready methods are available, as a programmer we need to implement rather we should just know how to call those methods.

Which one is prefered over Arrays and Collections?

Arrays is prefered, because performance is good.
Collections is not prefered because

1. List l=new ArrayList(); // default: 10 locations
   if 11th element has to added, then
   a. create a list with 11 locations
   b. copy all the elements from the previous collection
   c. copy the new reference into reference variable
   d. call the garbage collector and clean the old memory.

Note: To get something we need to compromise something, so if we use Collections performance is not upto the mark.
Array is language level concept(memory wise it is not good, performance is high)
Collection is API level(memory wise it is good,performance is low)

Difference b/w Arrays and Collection

Arrays => It is used only when Array size is fixed
Collection => It is used only when size is not fixed(dynamic)

Arrays => memory wise not recommended to use.
Collection => memory wise recommended to use.

Arrays => Performance wise recommended to use.
Collection => Performance wise it is recommended to use.

Arrays => It can hold only homogeneous objects
Collection => It can hold both heterogenous and homogenous Objects

Arrays => We can hold both primitive values and Objects
eg: int[] arr=new int[5];
Integer[] arr=new Integer[5];
Collection => It is capable of holding only objects not primitive types.

Arrays => It is not implemented using any standard data structure, so no ready made methods For our requirement,it increases the complexity of programming.

Collection => It is implemented using standard data structure, so ready made methods are available for our requirement,it is not complex.

What is a Collection?

In Order to represent a group of individual object as a single entity then we need to go for Collection.

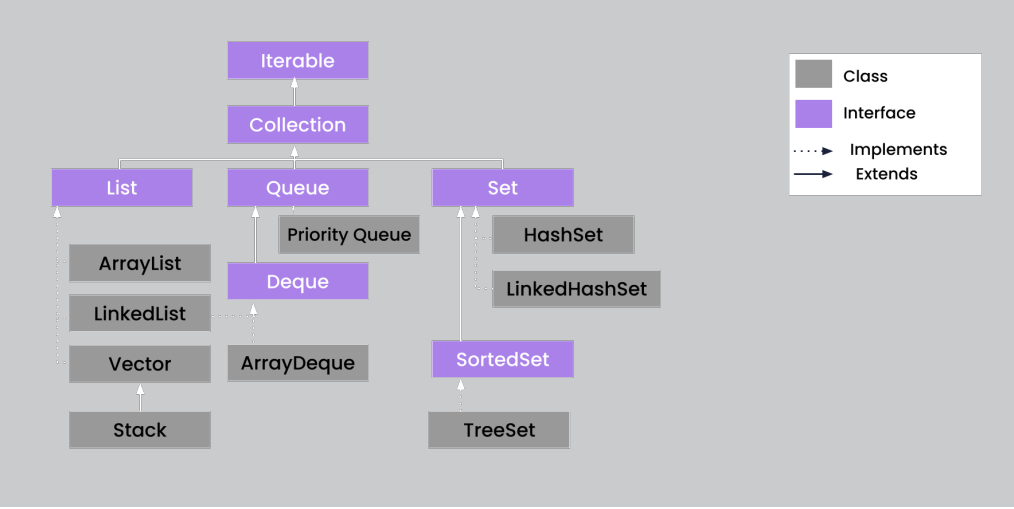
CollectionFramework
Group of classes and interface, which can be used to represent a group of individual object as a single entity, then we need to go for "CollectionFramework".

Java C++
Collection => container
CollectionFramework => STL(standard template library)

Collection
1. In order to represent a group of individual object, then we need to go for "Collection".
2. It is a root interface of collection framework
3. All the commonly used method required for all the collections are a part of Collection(I).

Note: There is no concrete class which would implement the interface Collection(I) directly.

Collection Hierarchy



To know more information about the framework,then we need to know the specification(interface)
9 key interfaces of Collection framework

1. Collection(I)
2. List(I)
3. Set(I)
4. SortedSet(I)
5. NavigableSet(I)
6. Queue(I)

MAP(I) → We will see in future lecture

7. Map(I)
8. SortedMap(I)
9. NavigableMap(I)

ArrayList(C)

1. DataStructure: GrowableArray /Sizeable Array
2. Duplicates are allowed through index
3. insertion order is preserved through index
4. Heterogenous objects are allowed.
5. null insertion is also possible.

Constructors

a. ArrayList al=new ArrayList()
Creates an empty ArrayList with the capacity to 10.
    a. if the capacity is filled with 10, then what is the new capacity?
        new capacity=> (currentcapacity * 3/2)+1
        so new capacity is =>16,25,38,.....
    b. if we create an ArrayList in the above mentioned order then it would result in performance issue.
    c. To resolve this problem create an ArrayList using the 2nd way approach.

b. ArrayList al=>new ArrayList(int initialCapacity)
c. ArrayList l=>new ArrayList(Collection c)
    It is used to create an equivalent ArrayList Object based on the Collection Object

When to use ArrayList and when not to use?

ArrayList => it is best suited if our frequent operation is "retrieval operation",because it implements RandomAccess interface.

ArrayList => it is the worst choice if our frequent operation is "insert/deletion" in the middle because it should perform so many shift operations.To resolve this problem we should use "LinkedList".

LinkedList

- Memory management is done effectively if we work with LinkedList.
- memory is not given in continuous fashion.

1. DataStructure is :: doubly linked list
2. heterogenous objects are allowed
3. null insertion is possible
4. duplicates are allowed

Usage

1. If our frequent operation is insertion/deletion in the middle then we need to opt for "LinkedList".

```
LinkedList l=new LinkedList();
  l.add(a);
  l.add(10);
  l.add(z);
 l.add(2,'a');
 l.remove(3);
```

2. LinkedList is the worst choice if our frequent operation is retrieval operation

Constructors

a. LinkedList l=new LinkedList();
   It creates an empty LinkedList object.

b. LinkedList l=new LinkedList(Collection c);
   To convert any Collection object to LinkedList.

ArrayDeque

- The ArrayDeque class implements the Deque interface.
- It facilitates us to use the Deque. Unlike queue, we can add or delete the elements from both the ends.
- ArrayDeque is faster than ArrayList and Stack and has no capacity restrictions.

PriorityQueue

- The PriorityQueue class implements the Queue interface.
- It holds the elements or objects which are to be processed by their priorities. PriorityQueue doesn't allow null values to be stored in the queue.

TreeSet

- Underlying Data Structure: BalancedTree
- duplicates : not allowed
- insertion order : not preserved
- heterogeneous element: not possible,if we try to do it would result in "ClassCastException".
- null insertion : possible only once
- Implements Serializable and Cloneable interface,but not RandomAccess.
- All Objects will be inserted based on "some sorting order" or "customised sorting order".

Constructor

TreeSet t=new TreeSet();//All objects will be inserted based on some default natural sorting order.

TreeSet t=new TreeSet(Comparator); //All objects will be inserted based on some customized sorting order.

Set

- It is the Child Interface of Collection.
- If we want to Represent a Group of Individual Objects as a Single Entity where Duplicates are Not Allowed and Insertion Order is Not Preserved then we should go for Set.
- Set Interface doesn't contain any New Methods and Hence we have to Use Only Collection Interface Methods

HashSet

1. Duplicates are not allowed,if we try to add it would not throw any error rather it would return false.
2. Internal DataStructure: Hashtable
3. null insertion is possible.
4. heterogeneous data elements can be added.
5. If our frequent operation is search, then the best choice is HashSet.
6. It implements Serializable,Cloneable, but not random access.

Constructors

HashSet s=new HashSet(); Default initial capacity is 16
Default FillRation/load factor is 0.75

Note: In case of ArrayList, default capacity is 10, after filling the complete capacity then a new ArrayList would be created.
In the case of HashSet, after filling 75% of the ratio only new HashSet will be created.

HashSet s=new HashSet(int initialCapacity);//specified capacity with default fill ration=0.75
HashSet s=new HashSet(int initaliCapacity,float fillRatio)
HashSet s=new HashSet(Collection c);

LinkedHashSet

- It is the child class of "HashSet".
- DataStructure: HashTable + linkedlist
- duplicates : not allowed
- insertion order: preserved
- null allowed : yes

All the constructors and methods which are a part of HashSet will be a part of "LinkedHashSet",but except "insertion order will be preserved".

Difference b/w HashSet and LinkedHashSet

HashSet => underlying data structure is "HasTable"
LinkedHashSet => underlying data structure is a combination of "Hashtable + "linkedlist" .
HashSet=> Duplicates are not allowed and insertion order is not preserved
LinkedHashSet => Duplicates are not allowed,but insertion order is preserved.
HashSet => 1.2V
LinkedHashSet => 1.4v

The 3 Cursors of Java

- If we want to get Objects One by One from the Collection then we should go for Cursors.
- There are 3 Types of Cursors Available in Java.

1. Enumeration
2. Iterator
3. ListIterator

1. Enumeration:

We can Use Enumeration to get Objects One by One from the Collection.
We can Create Enumeration Object by using elements().

public Enumeration elements();
Eg: Enumeration e = v.elements(); //v is Vector Object.

Methods:

1. public boolean hasMoreElements();
2. public Object nextElement();

```
import java.util.*;
public 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);//[0,1,2,3,4,5,6,7,8,9,10]
  Enumeration e = v.elements();
  while(e.hasMoreElements()) {
   Integer I = (Integer)e.nextElement();
   if(I%2 == 0)
    System.out.println(I);//0 2 4 6 8 10
  }
  System.out.println(v);//[0,1,2,3,4,5,6,7,8,9,10]
 }
}
```

Limitations of Enumeration:

- Enumeration Concept is Applicable Only for Legacy Classes and it is Not a Universal Cursor.
- By using Enumeration we can Perform Read Operation and we can't Perform Remove Operation.

To Overcome Above Limitations we should go for Iterator.

2. Iterator

- We can use an Iterator to get Objects One by One from Collection.
- We can Apply Iterator Concept for any Collection Object. Hence it is a Universal Cursor.
- By using Iterator we can Able to Perform Both Read and Remove Operations.
- We can Create Iterator Object by using the iterator() of Collection Interface.
  public Iterator iterator();
  Eg:Iterator itr = c.iterator(); //c Means any Collection Object.

Methods:

1) public booleanhasNext()
2) public Object next()
3) public void remove()

Limitations:

- By using Enumeration and Iterator we can Move Only towards Forward Direction and we can't Move Backward Direction. That is, these are Single Direction Cursors but NotBiDirection.
- By using Iterator we can Perform Only Read and Remove Operations and we can't Perform Addition of New Objects and Replacing Existing Objects.

To Overcome these Limitations we should go for ListIterator.

3. ListIterator:

- ListIterator is the Child Interface of Iterator.
- By using ListIterator we can Move Either to the Forward Direction OR to the Backward Direction. That is it is a Bi-Directional Cursor.
- By using ListIterator we can Able to Perform Addition of New Objects andReplacing existing Objects. In Addition to Read and Remove Operations.
- We can Create ListIterator Object by using listIterator().

public ListIteratorlistIterator();
Eg:ListIteratorItr = l.listIterator(); //l is Any List Object

Methods:

- ListIteratoris the Child Interface of Iterator and Hence All Iterator Methods by Default Available to the ListIterator.

Iterator(I)
|
|
ListIterator(I)

ListIteratorDefines the following 9 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)
public void add(Object new)
```

Legacy classes

- Legacy classes refers to the older classes that were included in the early versions of Java and have since been replaced by newer, more efficient classes. One such class is Enumeration, which is a legacy interface that was used to traverse collections before the introduction of the Iterator interface.

The Legacy classes are Dictionary, Hashtable, Properties, Stack, and Vector. The Legacy interface is the Enumeration interface.