* Collections *

DATE 07/12/24

Q. Why do we need Collection Framework in Java?

→ If we want to store multiple data or groups of objects together, we usually use Arrays, but it has some limitations.

* Disadvantages (limitation) of an Array:

1) We cannot store Heterogeneous data in an Array.
2) If we need to use Arrays we should know the base size in advance.
3) Arrays consume more memory but from performance point of view they are faster in execution.
4) Java doesn't provide much support for Arrays as they are not implemented using any dataStructures.
5) So we have to write complex code and logic which consumes time.

* Conclusion:

- So to overcome all the above problem we have Collections in Java.

* Difference Between Arrays and Collection:

Arrays	Collection
1) Arrays are index collection of fixed number of Homogeneous data elements.	1) Collections is a group of individual objects representing as a single entity.
2) Arrays are fixed in Size.	2) Collections are not fixed in Size.
3) To use Arrays we should know the size inadvance.	3) To use Collections we don't need the size inadvance.

DATE 07 12 24

4) We can store both primitive and non-primitive data init.	4) We can store non-prim in it.
5) Arrays are faster	5) Collection is slower.
6) It consumes more memory	6) It consumes less memor
7) Codes are complex in Array.	7) Codes are consize in Colle
8) It does not provides much support for builtin operation.	8) It provides support fo builtin operation.

* Collection Framework

- Collection Framework represents an Architecture for storing, arranging and manipulating group of objects In Ja
- It has two interfaces in it.
  i) Collection Interface.
  ii) Map Interface.

* collection :-

It is a group of individual object representing as a single entity is known as collection.

DATE 07 12 24

1) Enumeration
2) Iterator
3) ListIterator

Sorting → Comparator

Legacy Classes:
1) Vector
2) Stack
3) Hashtable
4) Dictionary (AC)
5) Properties

Interface → Enumeration

Iterable 1.5V
Collection 1.2V
Set 1.2V
HashSet 1.2V
LinkedHashSet
SortedSet 1.2V
NavigableSet 1.6V
TreeSet 1.2V
Queue 1.5V
PriorityQueue 1.5V
Deque 1.6V
ArrayDeque 1.6V
LinkedList 1.2V
List 1.2V
Vector 1.0V
Stack 1.0V
LinkedList 1.2V
ArrayList 1.2V
RandomAccess
Serializable
Cloneable

✗ insertion dup value ✗✗✗

✓ insertion order
✓ duplicate
multiple value

DATE 07/12/24

* Collection Interface

1) The Collection Interface is a member of Java's Collection Framework.
2) It is derived in java.util package.
3) The collection interface is used to pass collection objects.
4) It provides the standard methods and behaviours fo working with groups of objects.
5) It extends Iterable Interface and inherits Iterator enabling collections to be used inadvance forloop.
6) The Collection interface is not directly implemented by any other class.
7) However it is implemented indirectly via its subtyp (subinterfaces) like List, Set and Queue.

* Methods of Collection Interface.

1) public abstract int size();
2) public abstract boolean isEmpty();
3) public abstract boolean add(E ele);
4) public abstract boolean addAll(Collection c);
5) public abstract boolean remove(Object obj);
6) public abstract boolean removeAll(Collection c);
7) public abstract boolean retainAll(Collection c);
8) public abstract void clear();
9) public abstract Iterator<E> iterator();
10) public abstract T[] toArray(T[]);
11) public abstract object[] toArray();
12) public Stream<E> stream();

05 Date 12.24

1) add(E)e:-

This method is used to insert an element/insert an collection and the element is inserted at the end of this collection.

Ex:-
```
import java.util.*;
class Example
{   @SuppressWarnings("unchecked");
    psvm(S[]a)
    { Collection coll = new ArrayList();
      SOP(coll);
      for(int i=10; i<=100; i+=10)
      {   coll.add(i);
      }
      SOP(coll);
    }
}
```

Ex:-
```
import java.util.*;
class Ex
{   @SuppressWarning("unchecked");
    psvm(S[]a)
    { Collection coll = new Treeset();
      SOP(coll.add(3)); //T
      SOP(coll.add(4)); //T
      SOP(coll.add(1)); //T
      SOP(coll.add(2)); //T
      SOP(coll.add(3)); //ClassCastException
      SOP(coll.add("4")); //T
      SOP(coll);   //[3,4,1,2,"4"]
```

DATE 9/12/24

```
Collection coll = new HashSet();
SOP(coll);
SOP(coll.add(10)); //T
SOP(coll.add(20)); //T
SOP(coll.add(30)); //T
SOP(coll.add(20)); //F
SOP(coll.add("20")); //T
SOP(coll); // [10,20,30,"20"]
}
}
```

1) add() returns true if the collection accepts the eleme or else returns false if its refuse by the collection
2) Exceptions thrown by these method are ClassCastExcep NullPointerException, IllegalArgumentException.

2) addAll(Collection c) :-

1) This method is used to add group of objects insidea collection.
2) It will add all the elements at the end of this collectio
3) It returns true if collection accepts the elements or el returns false.
4) Exception thrown by this method are ClassCastException, NullPointException, IllegalArgumentException.

Ex:-

```
class Ex {
    psvm(String a)
    { Collection coll1 = new HashSet();
      coll1.add(10);
      coll1.add(20);
      SOP(coll1);
```

```
Collection coll1 = new ArrayList();
coll1.add(10);
coll1.add(20);
sop(coll1);

Collection coll1 = new TreeSet();
Coll1.add(10);
Coll1.add(20);
SOP(coll1);

Collection coll2 = new ArrayList();
coll2.add("fifty");
coll2.add(20);
coll2.add("sixty");
SOP(coll2);

SOP(coll1.addAll(coll2)); // [10, 20, fifty, Sixty]
SOP(coll1);
}
}
```

3) remove(Object obj):-

1) This method removes the single instance of specified element from this collection.
2) Return type of this method is boolean.
3) It returns true if the 1st instance of the specified element is been removed from this collection or else returns false.

Ex:- A Java Program to convert a Heterogeneous collection into Homogeneous.

→
```
Collection coll = new ArrayList();
coll.add(10);
coll.add("20");
```

```
coll.add(30);
coll.add("40");
coll.add(true);
coll.add(50);
coll.add('S');
coll.add("HELLOO");
coll.add(10.00);
SOP(coll);

Collection coll1 = new ArrayList();
for(Object ele : coll){
    if(ele instanceof Integer)
        coll1.add(ele);
}
}
```

4) removeAll (Collection c) :-

1) This method removes All the elements from this coll that are specified collection.
2) Return type of this method is boolean.
3) If the group of objects are present in it will remove them and will return true or else false.
4) This methods throws the following Exception :- Null Point Exception and ClassCastException.

Ex:-

```
Collection coll = new ArrayList();
coll.add(10);
coll.add("20");
coll.add(30);
coll.add("HELLOO");
coll.add(10.00);
SOP(coll);
```

Date 9/12/21

```
Collection coll1 = new ArrayList(~~int~~);
for(Object ele : coll) {
    if(ele instanceof Integer)
        coll1.add(ele);
    }
    sop(coll1);
    sop(coll.removeAll(coll1));
    sop(coll);
  }
}
```

Ex- Collection<String> coll = new ArrayList<String>();

```
coll.add("ramesh");
coll.add("nayan");
coll.add("suresh");
coll.add("nitin");
coll.add("mahesh");
coll.add("naman");
coll.add("mukesh");
sop(coll);

Collection pal = palindrome(coll);
sop(pal); // []
coll.removeAll(pal);
sop(coll);
}

public static Collection<String> palindrome(
                    Collection<String> coll)
{
Collection<String> pal = new ArrayList<>();
for(String ele : coll) {
```

```
        if(!(ele.contentEquals(new StringBuffer(ele).reverse()))){
            pal.add(ele);
        }
        return pal;
    }
}
```

5) contains() :-

1) This method returns true if this collection contains the specified element.

2) It returns true only if the collection contain the element or else returns false.

Ex:-

```
Collection coll = new ArrayList();
coll.add(10);
coll.add(20);
coll.add(null);
coll.add("30");

SOP(coll.contains(null));
SOP(coll.contains("30"));
SOP(coll.contains(true));

Collection coll1 = new TreeSet();
coll1.add(10);
coll1.add(20);
SOP(coll1.contains(null));
SOP(coll1.contains("30"));
SOP(coll1.contains(true));
}
```

9/12/24

containsAll() :-

This method returns true if this collection contains all the elements from the specified collection, or else returns false.

Exception thrown by containsAll() are ClassCastException and NullPointerException.

```
class Example {
    psvm(S[] a)
    { Collection coll = new ArrayList();
      coll.add(10);
      coll.add(20);
      coll.add(null);
      coll.add("80");

      ~~Sop(coll.containsAll(null)~~
      SOP(coll);

    Collection colll = new ArrayList();
      coll.add(10);
      coll.add(20);

      SOP(colll);

      SOP(coll.containsAll(colll));
      SOP(colll.containsAll(coll));

    }
}
```

DATE 9/12/24

7) size() :-

This method returns the no.of elements present in this collection.

Ex:- class Ex
{ @SW
psvm(S[]a)
{ List coll = new ArrayList();
Sop(coll.size()); //0
coll.add(10);
coll.add(20);
coll.add(30);
coll.add(40);
Sop(coll.size()); //4
for(int i=0; i<coll.size(); i++)
{ Sop(coll.get(i));
}
}
}

8) retainAll() :-

1) This Method retains only the elements that are conte in this specified collection.
2) In simple words it remove the collection of elements that are not present in specified collection argument.
3) This method throws NPE, CCE, and unsupported Operation Exception.

Q) WAJP to remove even elements from the collection
input:- [1,2,3,4,5,6,7,8,9]
o/p:- [1,3,5,7,9].

9/12/24

```
rt java.util.*;
: RemoveEvenElement
  @SuppressWarnings("unchecked")
  p s v m (S [] a) {
  Collection <Integer> coll = new ArrayList<>();
    coll.add(10);
    coll.add(2);
    coll.add(3);
    coll.add(4);
    coll.add(5);
    coll.add(6);
    coll.add(7);
    coll.add(8);
    coll.add(9);
     SoP(coll);
  Collection <Integer> coll1 = new ArrayList<>();
    for(int num : coll)
    { if(num % 2 != 0)
        coll1.add(num);
    }
    coll.retainAll(coll1);
      SOP(coll);
  }
}
```

clear() :- This method removes all the elements from this collection.

It throws UOE (unsupported operation Exception).

Ex

```
import java.util.*;
class Example {
    ps vm(S[]args) {
```

```
List<String> coll = new ArrayList<>();
        coll.add("Red");
        coll.add("Blue");
        coll.add("White");
        SOP(coll);
            coll.clear();
        ~~collection~~
            SOP(coll);
    }
}
```

10) isEmpty() :-

1) This method returns true if this collection contains no elements in it or else returns false.
2) It throws NPE, CCE

Ex:-
```
class Ex.
{   psvm(S[]a)
    {  List<String> coll = new Arraylist<>();

       if(coll.isEmpty())
       {  SOP("Empty");
       }
```

```
else{
    SOP("Not Empty").
}
}
}.
```

11) toArray() :-

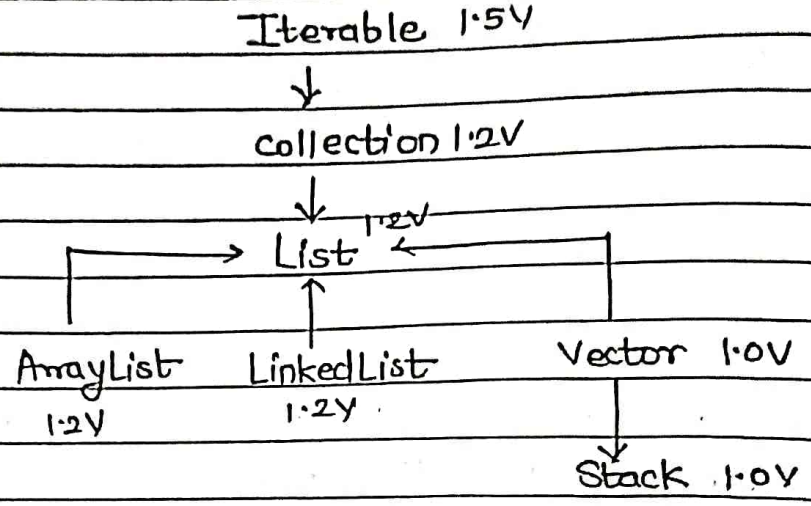
1) This method returns an object type of array containing all of the elements from this collection.
2) The return type of this method is an Object[] the reason behind it stores heterogeneous data elements.

Ex- import java.util.*;

```
class CharArray
{   p s v m(S[] a)
    { Collection coll = new ArrayList();
      coll.add(10);
      coll.add(20);
      coll.add(30);
      coll.add(40);
      SOP(coll);
    Object[] arr = coll.toArray();
    int arr1[] = new int[arr.length];
    for(int i=0; i<arr1.length; i++)
        arr1[i] = (Integer) arr[i];
        SOP(Arrays.toString(arr1));
    }.
}
```

10 12 21

* List :-



1) List is a child interface of collection.
2) Its an order collection of elements (Object).
3) It maintains the insertion order of elements.
4) It allow us to store duplicate elements and multiple null values.
5) It was introduced in JDK 1.2.
6) It has four implementation classes that are
   i) ArrayList
   ii) LinkedList
   iii) Vector
   iv) Stack

* Methods of List Interface

1) ~~public abstract boolean add(int E)~~ void

1) public abstract void add(int index, E ele);
2) public abstract boolean addAll(int index, Collection c);
3) public abstract E remove(int index);
4) public abstract int indexOf(Object obj);
5) public abstract int lastIndexOf(Object obj);

DATE 10 12 24

1) public abstract E get(int Index);
2) public abstract E set(int index, E ele);
3) public abstract ListIterator<E> listIterator();

1) add() :-
This method inserts the specified elements at a specified position in this List.

2) addAll() :-
It inserts all of the elements in the specified collection into this List at a specified position.

class Example {
psvm(S[]a) {
List list = new ArrayList();
list.add(10);
list.ad(20);
list.ad(30);
SOP(list);

List list1 = new ArrayList();
list1.add(40);
list1.add(50);

list.addAll(list1)
SOP(list);

list.add(2,80);
Sop(list);
}
}

DATE 10/12/24

3) remove() :-

- This method removes the elements which is at the specified position in this list.
- After removing it returns the same element.

Ex:- class Ex:

```
{   psvm(S[]a) {
        List list = new ArrayList();
        list.add(10);
        list.add(20);
        list.add(30);
        list.add(40);
        SOP(list);
        list.remove(1);
        .SOP(list);
        list.remove(4);
        SOP(list);
    }
}
```

4) set() :-

- This method replaces the element from a specified position in this List with the specified element.
- This method returns the previous elements which has been replaced.

5) get() :-

- This method returns the element from a specified position from this List.
- The return type of this method is E (type of element)

Ex:- class Example {

psvm(ST]a) {

```
List list = new ArrayList();
list.add(10);
list.add(20);
list.add(30);
    sop(list); // [10,20,30]

int list1 = list.get(0);
int list2 = list.get(1);
    sop(list1);

list.set(1,50);
    sop(list);

}
}
```

indexOf(Object obj) :-
This method returns the indexOf the 1st occurrence of the specified element in this List. or else returns -1 if the list does not contains the element.

lastIndexOf(Object obj) :-
This method returns the indexOf the last occurrence of the specified element in this List or else returns -1 if the list does not contains the element.

Ex:-
```
class Example {
    psvm(String[] a)
    {
List list = new ArrayList();
list.add(10);
list.add(20);
list.add(30);
list.add(40);
```

```
SOP(list);

SOP(list.indexOf(20));

SOP(list.lastIndexOF(30));

}
}
```

* ArrayList :-

1) ArrayList is an implementation class of List Interface.
2) Arraylist was introduced in version 1.2 and derived in java.util package.
3) The internal implementation of ArrayList is a Growable Array (Datastructure).
4) It stores the data using indexing.
5) It preserves the insertion order.
6) It allows us to store duplicate elements and also multiple null values.
7) Since, ArrayList is implemented using a growable array which uses indexing so we can perform fast searching of elements.
8) So for this fast searching of element ArrayList implements RandomAccess interface (Marker Interface).
9) ArrayList is not synchronized because of which multiple thread can work at a time and it is faster in performance compare to vector.

DATE 10 12 24

10) As it is not synchronized, its not thread safe.
11) The initial capacity of an ArrayList is 10.
12) We can store both homogeneous and heterogeneous data element in it.
13) Other interfaces which are implemented by ArrayList are Collection, Iterable, Serializable and ~~Set~~ Clonable.

* Constructors of Arraylist

1) public ArrayList();
2) public ArrayList(int capacity);
3) public ArrayList(Collection c);

1) public ArrayList():-
It constructs an empty list with an initial capacity 10.

Ex:-
```
import java.util.*;
class ArrayListConstructor
{   psvm(S[]a){
List<Integer> list = new ArrayList<>();
    SOP(list.size());

for(int i=10; i<=100; i+=10)
 list.add(i);
 SOP(list.size());

//(initial capacity * 3/2)+1
      10 × 3/2 +1 = 16

list.add(110);
 }
}
```

10/12/24

Note :- i) Initial Capacity of ArrayList is 10. once the is full if we try to store the next element it create a new ArrayList with a capacity of.
(initial capacity * 3/2)+1 = New Capacity.
i.e (10*3/2)+1 = 16

The new ArrayList have capacity 16 & it will copy all elements from old array list into newly created Array

2) ArrayList(int initial capacity) :-
It constructs an empty list with a specified capacity.

Ex :-
```
import java.util.*;
class Demo {
psvm(S[] a) {
List<Integer> list = new ArrayList<>();
for(int i=10; i<=100; i+=10)
{
list.add(i);
}
SOP(list.size());

//(initial cap * 3/2)+1
// (20 x 3/2)+1 → 60/2+1 = 31

list.add(210);
}
}
```

DATE 10/2/24

To alter the capacity of an ArrayList we can use following method.

i) ensureCapacity(int newcapacity):-
This method create new ArrayList with specified capacity.

ii) trimToSize():-
This method reduce the capacity of ArrayList equivalent to the no. of element present in it.

Ex:-
```
import java.util.*;
class Demo{
psvm(S[]a){
ArrayList<Integer> list = new ArrayList<>(1000);
for(int i=1; i<=900; i++)
list.add(i);
list.trimToSize();
SOP(list);
}
} //OP:- [1,2,3,...,900].
```

Ex:-
```
import java.util.*;
class Demo{
psvm(S[]a){
Arraylist<Integer> = new ArrayList<>(100);
for(int i=1; i<=100; i++)
list.add(i);
list.ensureCapacity(300);
for(int i=1; i<=200; i++)
list.add(i);
SOP(list);
}
}
```

DATE 10/12/24

3) ArrayList (Collection c) :-

It constructs a list containing elements of a specified collection.

- Basically it is used to convert any other type of into ArrayList.

Ex :-

```
import java.util.*;
class ArrayListConstructor {
    psvm(S[]a) {
Treeset<Integer> ts = new TreeSet<>();
    for(int i=1; i<=10; i++)
        ts.add(int)(Math.random()*10));
    SOP(ts);
Arraylist<Integer> al = new ArrayList<>(ts);
    SOP(al);

LinkedHashSet<Integer> lhs = new LinkedHashSet<>(
    lhs.addAll(ts);
ArrayList<Integer> al1 = new Arraylist<>(lhs);
    SOP(al1);
PriorityQueue<Integer> pq = new PriorityQueue<>(ts
ArrayList<Integer> al2 = new ArrayList<>(pq);
    SOP(al2);
}}
```

* Disadvantages of Arraylist :- i) If we want to perform like insertion & removing of an element from list it is going to consume lot of time because it performs shifting operation on removing an element/after inserti

Note :- To overcome this problem we have LinkedList.

12/12/24
Thu

1] tostring()

It returns the string representation of Arraylist.

If Arraylist containg no element it return [] or else returns enclosed [] or else return inside [] separated by ,.

2] add()

Element is add at the end and it's capacity is 10. If capacity is full then it increase by ((initialcap * 3) 2 + 1). add() use the indexing to add elements at index.

3] remove()

Arraylist remove and returns the element from any index and decreament Arraylist size by -1.

4] size()

It returns the number of elements i
the arraylist

5] indexOf()

It returns index of first occur
of specified element or not fou
then it return -1.

6] lastIndexOf()

It returns index of the last occ
of specified element.

7] get()

Retrieves the element at ~~th~~ specified
index.

8] set(int indx, E ele)

Replace the element at specied
with a new value.

4] removeAll()

It removes all elements from one list that exist in another list.

ote :

We cannot create generic type of array n java so by implementing ArrayList we can create object array convert n typecasted to generic array.

```
(E[]) new Object[size]
```

Random Access Interface

1] It is marker interface

2] There are few classes in collection that are implements Interface ie ArrayList, vector, stack.

3] It is use for fast searching of el with constant time complexity.

4] A class implementing RandomAcces provides meta data that it provid fast searching of element.

Note: RAI implemented by those clas which uses indexing for storing retrival of elements. which gener linear time complexity.

LinkedList 1.2V

] LinkedList is implementation class of List interface.

] Introduce in 1.2V & derived in java.util package.

] It preserve / maintain insertion order.

] We can store duplicate as well as null values.

] It is not synchronized.

] Internal implementation is DoublyLinked List in java.

] It store data in form of node. (inner class).

] LinkedList store data / element (node) at non contigenous location, & best suitable for insertion or removing for element bcoz it not shifting element. It generates O(1) constant time complexity for remove / insert.

10] other interface implemented by LinkedL
are Collection, Iterable, Seriasible, Clo
Deque.

11] LinkedList can implements by 3 way
singlyLinkedList, DoublyLinkedList.
& CircularLinkedList.

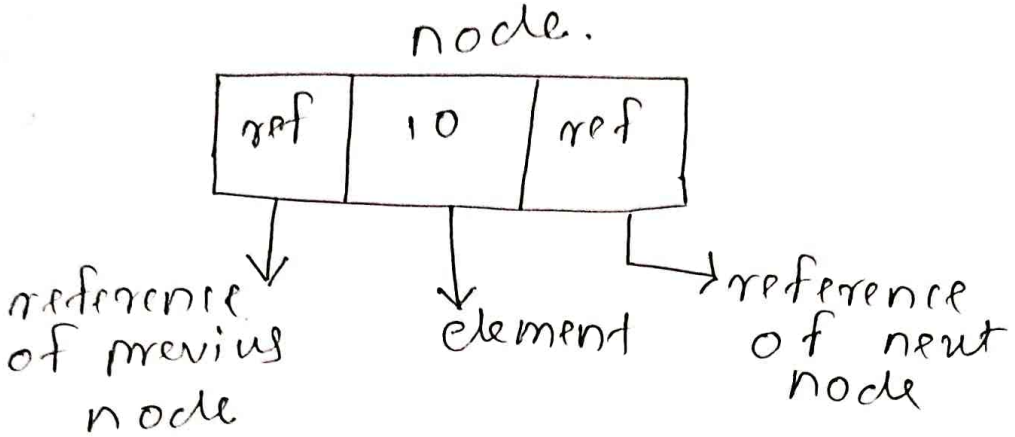
DoublyLinkedList

In computer science doubly linked
is link data structure that consis
of a set of sequencially linked r
~~nodes~~ called node.

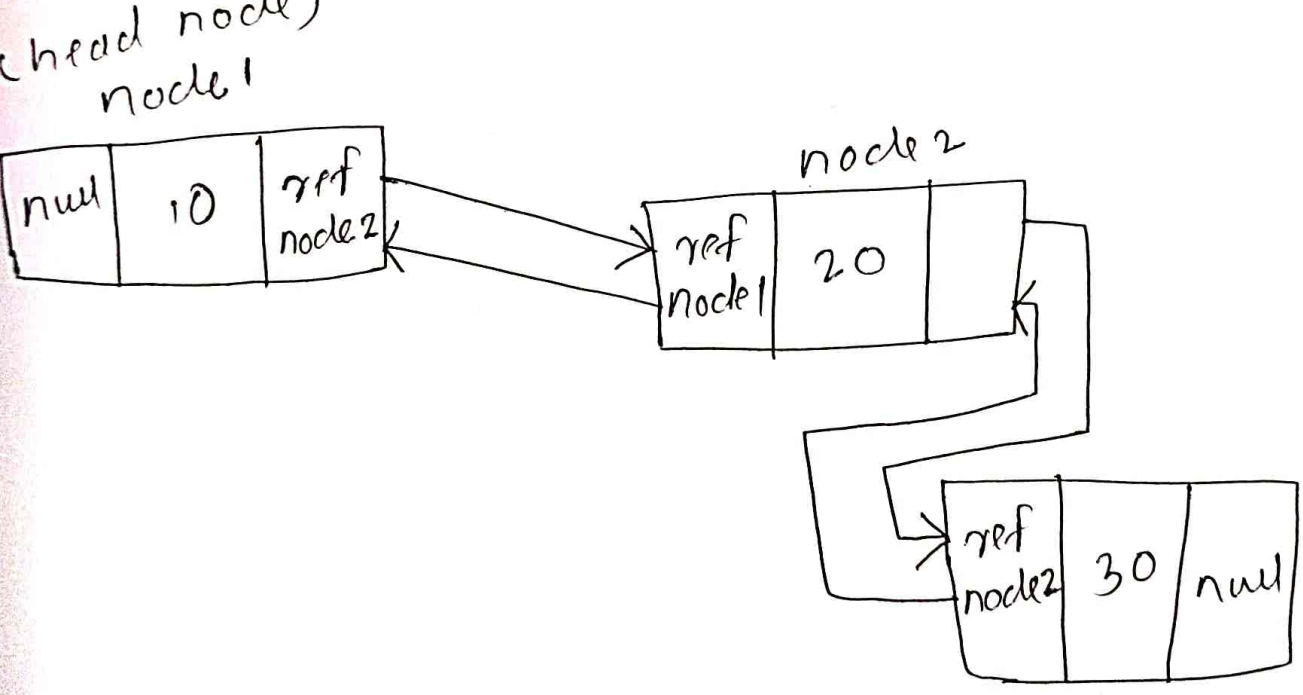
Node

A node in java inner class present
LinkedList

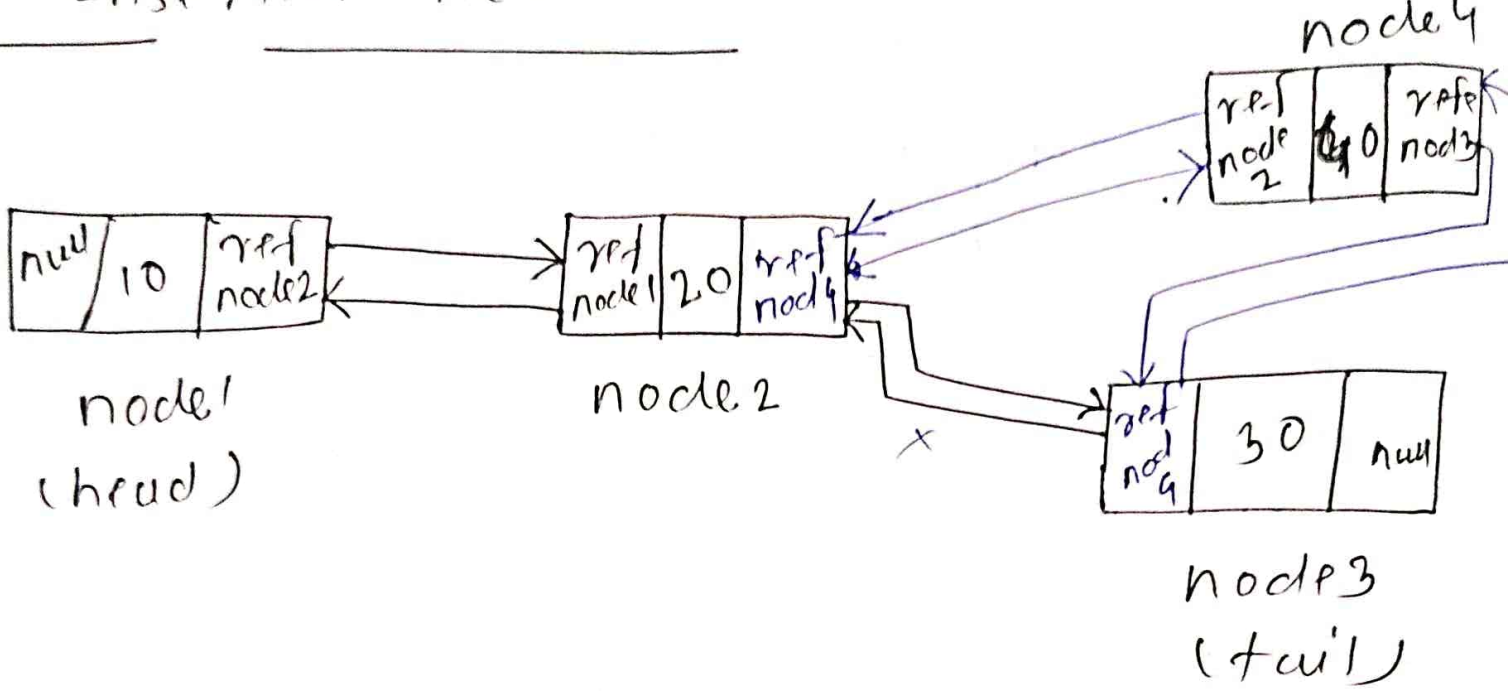
It mainly consist of 3 parts
i] Reference of previous
ii] element
iii] Reference of next node.



resentation



* Insertion new node.



Constructor

1] public LinkedList ();

2] public LinkedList (Collection c);

public LinkedList()

It is use to construct empty list which doesn't hewe element.

u.

```
main()
{
    LinkedList list = new LinkedList();
    list.add(10);
    list.add(20);
    list.add(30);
    SOP(list);
}
```

2] ~~Collection~~ LinkedList (Collection c)

It construct new list that cont
all the elements from specified
collection.

Eg.

```
main()
{
    ArrayList arr = new ArrayList
    arr.add(10);
    arr.add(20);
    arr.add(30);
    LinkedList list = new LinkedLis
    Sop(list);
}
```

methods of LinkedList

```
public E getFirst();
public E getLast();
public E removeFirst();
public E removeLast();
public void addFirst(E Ele);
public void addLast(E ele);
public boolean contains(Object obj);
```

DATE
13/12/24

LinkedList Program Implementation

```
package collection;
import java.util.LinkedList;
import java.util.NoSuchElementException;

class UserLinkedList<E> {
    int indx=0;
    Node<E> head;
    Node<E> tail;

    @SuppressWarnings("hiding")
    class Node<E>{
        E ele;
        Node<E> next;
        Node(E ele){
            this.ele = ele;
        }
    }

    @Override
    public String toString() {
        if(head == null)
            return "[]";
        String op = "[";
        Node<E> currentNode = head;
        while(currentNode.next != null) {
            op += currentNode.ele+", ";
            currentNode = currentNode.next;
        }
        op += currentNode.ele + "]";
        return op;
    }
```

DATE 13 12 21

```
public void addLast(E ele) {
    Node<E> newNode = new Node<E>(ele);
    tail = newNode;
    if(head == null) {
        head = newNode;
    } else {
        Node<E> currentNode = head;
        while(currentNode.next != null) {
            currentNode = currentNode.next;
        }
        currentNode.next = newNode;
    }
    indx++;
}

public void addFirst(E ele){
    Node<E> newNode = new Node<E>(ele);
    if(head == null) {
        head = newNode;
    } else {
        newNode.next = head;
        head = newNode;
    }
    indx++;
}

public int size(){
    return indx;
}

public E getFirst(){
    return head.ele;
}
```

DATE 18/12/24

```
public E getLast() {
    return tail.ele;
}

public E removeFirst() {
    if(head == null)
        throw new NSEE("User the Linked List
                        is empty");
    E temp = head.ele;
    head = head.next;
    indx--;
    return temp;
}

public E removeLast() {
    if(head == null)
        throw new NSEE("User the Linked list
                        is empty");
    Node<E> currentNode = head;
    for(int i=0; i<size()-2; i++) {
        currentNode = currentNode.next;
    }
    currentNode.next = null;
    E temp = tail.ele;
    tail = currentNode;
    indx--;
    return temp;
}
```

```
public void add(int indx, E ele) {
    if(indx<0 || indx>=size())
        throw new IOOBE("User wrong index"+In

    Node<E> newNode = new Node<E>(ele);
    Node<E> currentNode = head;
    for(int i=0; i<indx-1; i++) {
        currentNode = currentNode.next;
    }
    newNode.next = currentNode.next;
    currentNode.next = newNode;
    indx++;
}


public E remove(int indx) {
    if(indx<0 || indx>=size())
        throw new IOOBE("User wrong index"+in
    if(size()==1) {
        E temp = head.ele;
        head = null;
        return temp;
    }
    Node<E> currentNode1 = head;
    Node<E> currentNode2 = null;
    for(int i=0; i<indx-1; i++) {
        currentNode1 = currentNode1.next;
    }
    currentNode2 = currentNode1.next;
    currentNode1.next = currentNode2.next;
    this.indx--;
    return currentNode2.ele;
}
```

DATE 13/12/24

```
public boolean removeFirstOccurrence(E ele){
    Node<E> currentNode = head;
    boolean flag = false;
    for(int i=0; i<size(); i++){
        if(ele.equals(currentNode.ele)){
            remove(i);
            flag = true;
            break;
        }
        currentNode = currentNode.next;
    }
    return flag;
}
public boolean removeLastOccurrence(E ele){
    Node<E> currentNode = head;
    boolean flag = false;
    int indx = 0;
    for(int i=0; i<size(); i++){
        if(ele.equals(currentNode.ele)){
            flag = true;
            indx = i;
        }
        currentNode = currentNode.next;
    }
    if(flag){
        remove(indx);
    }
    return flag;
}
}
```

DATE 13/12/24

```
public class LinkedListDriver {
    p s v m (S[]args) {
    LinkedList list1 = new LinkedList();
    list1.addLast(10);
    list1.addLast(20);
    list1.addLast(30);
    list1.addLast(40);
    SOP(list1); // [10,20,30,40].

UserLinkedList list2 = new UserLinkedList();
    list2.addLast(10);
    list2.addLast(20);
    list2.addLast(30);
    list2.addLast(40);
    list2.addLast(50);
    list2.addLast(60);
    list2.addLast(40);
    list2.addFirst("hello");
    list2.addFirst("easy");
    list2.addFirst("Java is");
    SOP(list2);
    SOP(list2.removeFirst());
    SOP(list2);
    SOP(list2.removeLast());
    SOP(list2);
    SOP(list2.remove(40));
    SOP(list2);
    SOP(list2.removeFirstOccurrence(40));
    SOP(list2);
    SOP(list2.removeLastOccurrence(40));
    SOP(list2);
}
}
```

* Explaination of methods

DATE 18/12/24

1) toString() :-

- Converts the linked list to a string representation.
- if the LinkedList contains no elements in it returns empty [] or else returns the elements enclosed in [] separated by comma.

2) addFirst() :-

- Adds a new element at the beginning of the list
- Create a new Node with the given element.
- Set the next of the newNode to point to the current head.
- Update head to the new node. and increment the index

3) addLast() :-

- Adds a new element at the end of the list.
- Create a new Node with the given element.
- If the list is empty (head == null), set head and tail to the new node.
- Update tail to the newnode and increment the index.

4) removeFirst() :-

- Removes the first element in the list.
- If the list is empty, throw a NSEE.
- Save the ele of the head node.
- Update head to point to head.next.
- Decrement the size and return element.

5) removeLast() :-
- Removes the last element in the list.
- If the list is empty, throw a NSEE.
- If the list has only one element (head == tail), set head and tail to null.
- Update tail and decrement the size.

6) size() :-
- Returns the current size of the list.

7) getFirst() :-
- Returns the element at the head node.

8) getLast() :-
- Returns the element at the tail node.

9) remove(int indx) :-
- Removes the element at the specified index.
- Validate the index (between 0 and size()-1)
- Update the size and return the element.

10) add(int indx, E ele) :-
- Inserts an element at the specified index.
- Validate the index (must be betn 0 & size())
- if the index is 0, call addFirst.
- if the index is size(), call addLast.
- Update the size.

11) removeFirstOccurrence(E ele) :-

- Removes the first occurrence of the given Element.
- Traverse the list while checking each node's element.
- When found, call remove(indx) and return true.
- if the element is not found, return false.

12) removeLastOccurrence(E ele) :-

- Removes the last occurence of the given element.
- Traverse the list to track the index of the last occurenc
- Call remove(index) if the element is found.

14.12.24
Saturday

Vector

1] Vector is implemented class of list interface.

2] It is present inside java.util package.

3] Introduce in 1.0v.

4] It internally implements growable array data structure.

5] It maintain insertion order.

6] It allows to store duplicate values.

7] It allows you to store multiple null value

8] It is provide the implementation for the RandomAccess interface bcoz that is faster for searching.

9] It can provide implementation for interface like Clonable, Serializable & RandomAccess.

10] Vector is synchronized bcoz of that we perform multithreading.

11] It also called Legacy class.

12] It can store both hetrogenous & homogeneous type of data.

13] Vector have a built-in capacity 10, Vector capacity is used doubled when we insert more than 10.

* Constructor of Vector

1] public vector()
2] public Vector(int initialCapacity);
3] public vector(int initialCap, int cap
4] public vector(Collection c)
1] public Vector()

It constructs an empty vector with no element in it & with capacity 10.

Ex.

```
main {
Vector v = new vector(); //10
for(int i=0; i<=10; i++)
    v.add(i);
Sop(v);    // capacity 10
}
```

2] public Vector(int initialCapacity)

It constructs with an empty vector object, it specified initial capacity.

```
import java.util.*;
class Vector {
    main() {
        Vector v = new Vector(20);
        for(int i=10; i<=21; i++)
            v.add(i);
        sop( v.capacity());
        sop(v);
    }
}
```

Vector(int initialCap, int capIncrement)

It ~~count~~ constructs an empty vector with specified initial capacity & increment capacity

Parameters: i) initial capacity : initial capacity of vector

ii) capacity increment : The amount by which capacity set when vector overflow.

Ex. import java.util.*;

```
class Vector {
    main() {
        Vector v = new Vector(10, 20);
        for(int i=0; i<=10; i++)
            v.add(i);
        sop(v.capacity()); //30
    } } sop(v);
}
```

4] Vector(Collection c)

It constructs a vector containing elem of specified collection.

Ex. import java.util.*;

```
class Vector {
    main() {
        Vector v = new Vector();
        for(int i=0; i<=10; i++)
            v.add(i);
        sop(v);
        ~~Vector v = new Vec~~
        ArrayList arr = new ArrayList(
        sop(arr);
    } }
```

methods

public synchronized int capacity()

public synchronized E elementAt(int indx)

public synchronized E firstElement();

public synchronized E lastElement();

public synchronized void setElementAt(E, in

public synchronized void removeElementAt(int indx

public synchronized void
insertElementAt(E, int indx)

public Enumeration elements()

public synchronized void trimToSize()

public synchronized void ensureCapacity(int

public synchronized void addElement(E ele)

public synchronized boolean removeElement(obj

16.12.24
Mon

Cursors

Jawa cursor is an iterator is use to iterate (traverse) over the collection, stream of object one by one.

Advantages

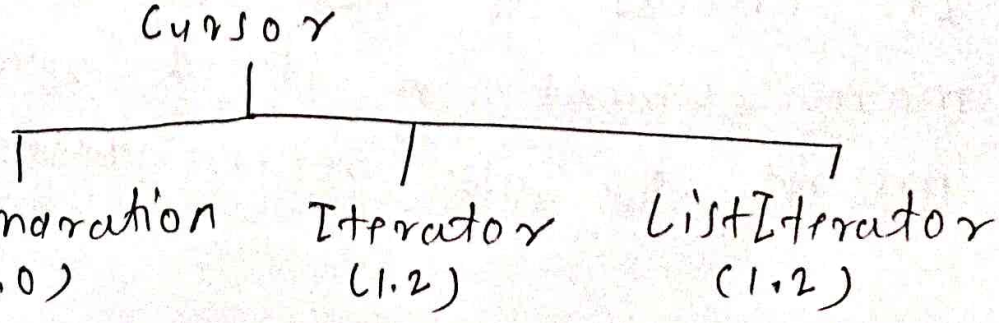
We can use it to iterate through elements of collection or dataset without worry about underline data structure.

Cursors are design to work with various data structure list, set & map.

It allows us to safely remove the elements from collections, preventing ConcurrentModificationException.

Cursor also allow us traverse a collection in both forward & backward direction.

Types



1] Enumaration (1.0)

It legacy interface in jawa is introd in 1.0 v & derive in jawa.util package.

An object that implements Enumaration interface generates series of item element at a time.

As it's legacy interface it only applicable for vector & stack.

It cannot perform any operation rather than iterating (retrieving) element.

It traverse element in only one direction.

Methods

① public abstract boolean hasMoreElement

It checks if these enumaration contains more element in it & return true if present or else return false.

② public abstract E nextElement()

It returns next element of these enumeration and moves the cursor one element ahead.

```
import java.util.~~Enum~~ Enumeration;
import java.util.Vector;
class Exp {
    main(){
    Vector<Integer> v = new Vector<Integer>();
    for(int i=10; i<=100; i+=10)
            v.add(i);
    sop(v);
    Enumeration<Integer> en = v.elements();
    while (en.hasMoreElements())
    {
     Iterator ele = en.nextElement();
      sop(ele);
} } }
```

Note: We cannot remove elements using Enumeration to overcome these we have Iterator.

2] Iterator

The functionality of these interface duplexation of Enumeration.

It takes places of Enumeration in java frmWk collection.

Iterator differ from Enumeration by follow

Iterator allows us to remove eleme from undereine collection while trave with well define implementation.

It has improve method names

Iterator is universal cursor which c operate any classes of collection framew iterator() is define in Iterable interf (root of collection).

There are 2 child classes of Iterab ① ListIterator ② EventIterator

* Methods

1] public abstract boolean hasNext()

2] public abstract E next()

3] ~~default void remove()~~

3] public default void remove()

1] public abstract boolean hasNext()

It return true if iteration has more elements in it or else false.

2] public abstract E next()

It return next element from iteration.

3] public default void remove()

It remove element from underlying colle

The method called once per call next(

If try to call more than once metho

throughs exception such as

Unsupported ~~Except~~ OperationException,

IllegalStateException.

17-12-24
Tue

3] ListIterator

It is a child interface of iterator.

It is not universal cursor as it only operates on List implementation classe

This iterator allows programmer to traverse the list in both direction ie forward & backword.

With the help of this cursor we can perform reversed operation such as insertion of an element, removing element, updating element & fetch element.

* Methods

1] public abstract boolean hasNext()

It returns true if this iterator has more elements by traversing this list in forward direction or else return false.

2] public abstract boolean hasPrevious()

It return true if this iteration has more element while traversing list in buckword direction or else return false.

3] public abstract E next()

It returns the next element from this list & moves the cursor position ahead.

4] public abstract E previous()

It return previous element from this list & moves cursor position backward.

5] public abstract void remove()

It removes the last element last was returned by next() or previous() (based on operation).
This call to remove() can be made only if remove() or add() have been called after the last call to next() or previous().

6] public abstract void set (E ele)

It replaces element return by next() or previous() with specified element.

7] public abstract void add (E ele)

It inserts the specified element into list.

Ex
```
List<Integer> list = new ArrayList<>();
for(int i=0; i<=100; i+=10)
    list.add(i);

sop(list);    // [10, 20, 30, 40, 50, 60, 70, 80, 90
                                              100]

ListIterator<Integer> list2 = list.iterator();
while (list2.hasPrevious())
{
    sop(list2.previous);
}

while (list2.hasNext())
{
    sop(list2.next());
}
while (list2.hasPrevious())
{
    Integer currEle = list2.previous();
    sop(currEle);
}
```

* Collections class

This is part of java.util package, it provides static utility method for working with collection framework.

The method of these class all throw a NPE if collections or class objects provided to them are null.

* methods

public static int frequency (Collection c, Object frqObj)

public static Comparator <T> reverseOrder ()

public static T max (Collection c)

public static T min (Collection c)

public static void rotate (List list, int noofrot)

public static void fill (List list, T type)

public static void shuffle (List list)

public static void swap (List list, int elefirst, int elesecond)

public static void sort (List list, Comparator comp)

public static Comparable <T> void sort (List list)

Comparable Interface

- Comparable interface was introduced in jdk 1.2 & derived in java.lang package.
- The comparable interface is used for natural ordering of elements (ie use)
- Collections.sort() internally uses Comparable interface for sorting.
- Comparable is functional interface that is it contains a single abstract method in it which is compareTo().

Steps to use Comparable Interface

1. The class must implement Comparable interface.
2. We need to override compareTo().

Comparable can perform single sorting sequence at a time ie we can sort the elements base on the single data members

1] public abstract int compareTo()

This method is used to compare the current object (this) with the specified object and return some value based on condition.

This method returns:

Positive integer: if the current object data member is greater than specified object data member.

Negative integer: if the current object data member is less than the specified object data member if it return zero, if current object data member same as specified object data member.

Comparator Interface

① The Comparator interface in jawa is used to order the element based on user define implementation.

② It oftenly used when we need to sort object in way that is different from natural ordering (Comparator).

③ Comparator introduced in jdk 1.2 & derived in jawa.util package.

④ We can use it for multiple sorting of data members at a time.

* Steps

① Define a class for sorting, implement it with comparator interface.

② override compare().

③ Invoke Collections.sort() & pass first arg as list, second arg as instance & comparator type.

~~# Stack~~

public abstract int compare (T obj1, T ob

It compares both the specified args object & return value based an condn

It return -1 : If object 1 data member is less than object 2 data member.

It return 1 : If obj1 is greater than obj 2

It return 0 : if both are same.

Comparable	Comparator
① use to sort the obj with natural order	① use to sort obj based on custom or
② derived in java.lang	② derived in java.u
③ use for both built-in class as well as user defined class	③ use for user defined class
④ It can perform single sorting attribute at a time.	④ perform multip attribute
⑤ It contains abstract method ie compareTo()	⑤ It contains 2 abstract metho compare() & eq

19.12.24
Thu

Set interface

① Set is a subinterface of Collection.

② It is none order collection of object (ie doesn't maintain insertion order).

③ We can't store duplicate value.

④ We can't store multiple null value.

⑤ It contains methods inherites from Collection interface & add functionality that restriction of insertion of duplicate element.

⑥ Introduce in 1.2 version & derived in java.util package.

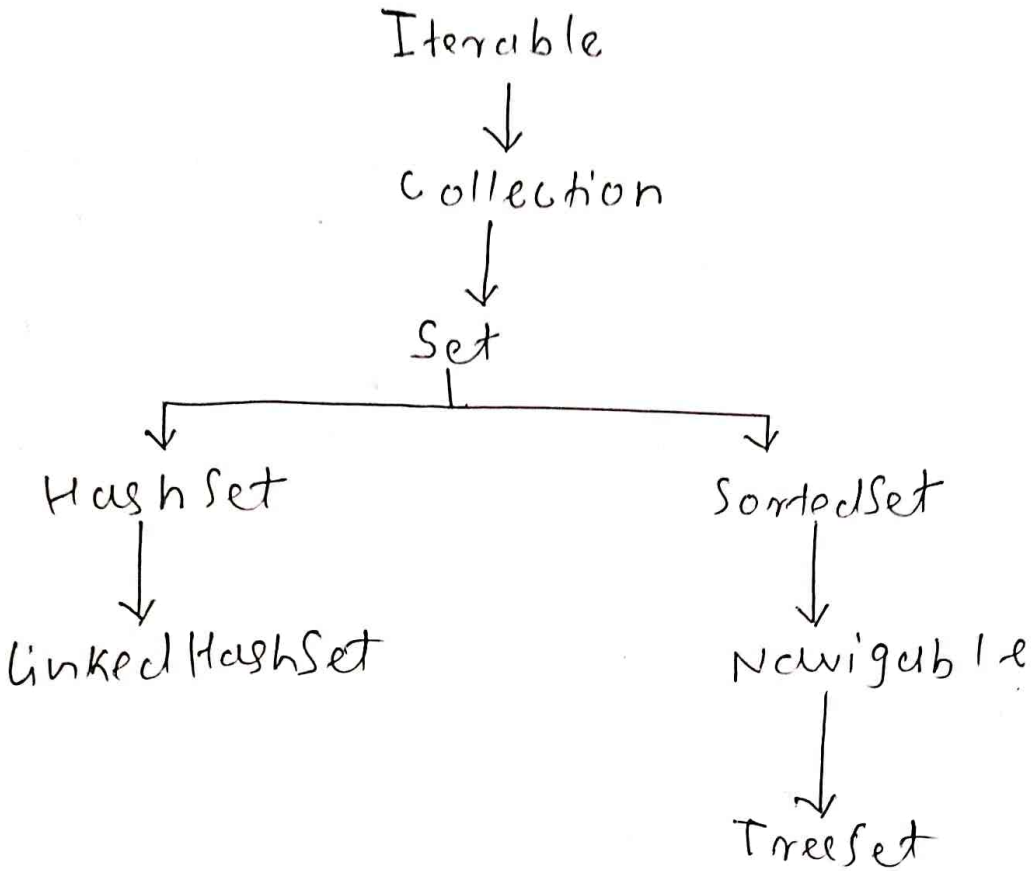
⑦ There are two interfaces that are extends Set ie. NavigableSet & SortedSet.

⑧ And it has some implimentation classes such as LinkedHashSet, HashSet, TreeHashSet

Iterable 1.5
↓
Collection 1.2
↓
Set

HashSet

LinkedHashSet


Iterable
Collection
Set
HashSet
SortedSet
LinkedHashSet
Navigable
TreeSet

Hashset

① Hashset is implementation class of Set interface.

② Introduce in jdk 1.2 & derived in java.util pack.

③ Internal implementation of Hashset is a Hashmap.

④ Whenever we create Hashset Object, one Hashmap object associate with it is created internally.

⑤ The element which we add into the Hashset is stored as a key inside the Hashmap object

⑥ And value associated with those keys will be constant "PRESENT".

⑦ We can not store duplicate key but we can store duplicate values.

⑧ It provides no guarantee of ordering element (it means doesn't maintain insertion order).

(9) We can't store duplicate elements & also also one NULL value is allowed.

(10) Hashset ~~often~~ offers constant time performance complexity for basic operation like add(), remove(), contains (search), size.

(11) Every Hash implementation classes containing default load factor ie 0.75.
(HashSet, LinkedHashSet, HashMap, LinkedHashMap, HashTable)

(12) Hashset has default capacity 16 which will double one object reaches it's threshold value (load factor).

(13) HashSet has child class called ~~linked~~ LinkedHashSet.

(14) It implements other interfaces like Collection, Iterable, Serializable, Clonable,

(15) It not Synchronized.

* Constructors

1] public HashSet ()

2] public HashSet (Collection c)

3] public HashSet (int initialCapacity)

4] public HashSet (int capacity, float loadFac

1] public HashSet ()

It create empty set with initial capacity (16) and with load factor 0.75 (default)

Ex

```
main(){
    HashSet <Integer> set = new HashSet ();
    set.add(10);
    set.add(40);
    set.add(20);
    set.add(30);
    set.add(10);
    sop(set);          // [20, 10, 40, 30]
}
```

2] public HashSet(Collection c)

① It create set a specified collection in it

② Hashmap which is created will have default load factor of 0.75 & capacity will be equals to no. of elements inside the specified collection.

③ It ~~thr~~ throw NPE if specified collection is null.

Ex main() {

```
Arraylist<Integer> list = new Arraylist();
list.add(null);
list.add(10);
list.add(40);
list.add(20);
list.add(null);
sop(list);   // [null, 10, 40, 20, null]

Hashsed<Integer> set = new Hashset(set);
sop(set);  // [null, 40, 10, 20]
```

24.12.24

ceiling() : It returns lowest element from Set which is greater than or equal to given element.

method return null if there is no such element.

Ex.

```
main()
{
    TreeSet<Integer> set = new TreeSet<Integer>();
    for(int i=10; i<100; i+=10)
        set.add(i);
    sop(set);
    sop(set.ceiling(52));   // 60
    sop(set.ceiling(55));   // 60
    sop(set.ceiling(57));   // 60
    sop(set.floor(52));     // 50
    sop(set.floor(55));     // 50
}
```

subSet() : It return view of portion of set whose elements ranges from element till to element.

If from element and to element are equals it will return an empty set.

Return type of these method is SortedSet.

```
Ex  main()
    {
      TreeSet<Integer> set = new TreeSet<Integer>();
      for(int i=10; i<=100; i+=10)
      {
            set.add(i);
      }
      sop(set);

      SortedSet<Integer> subset1 = set.subSet(30,70);

      TreeSet<Integer> subset2 = (TreeSet<Integer>)
                                  set.subSet(30,70);

      sop (subset1);   // 30, 40, 50, 60

      sop(subset2);    // 30, 40, 50, 60
    }
```

If first args greater args it throw IllegalArgumentException.

If we provide argument which is not present it will use its ceiling value.

headSet() : It return view of portion from the set whose elements are strictly less than specified element.

tailSet(E ele) : It return a view of portion from the set whose elements are ~~eqq~~ greater than or equals to specified object.

Ex

```
TreeSet<Integer> set = new TreeSet<Integer>();
for(int i = 10 ; i<=100; i+=10)
        set.add(i);
sop(set);
sop( set.headSet(54)); // 10, 20, 30, 40
sop( set.headSet(50)); //  10 20 30 40
sop( set.headSet(10)); //  []
sop( set.headSet(5)); //  []

sop(set.tailSet(50)); //  50, 60, 70, 80, 90, 100
sop( set.tailSet(100)); //  100
sop( set.tailSet(10)); //  10, ..... 100
}
```

Iterable (interface)

① It is root of collection interface.

② Introduce in 1.5v and derive in java.lang package.

③ These interface represents collection of objects that can be iterated using iterator.

④ It provides ability to traverse elements sequencially collection.

⑤ It has single abstract method in it i.e. iterator().

⑥ It is example of functional interface.

⑦ It containg default method forEach()

⑧ The interfaces which extends Iterable are Queue, List, Set, Collection, SortedSet, etc.

* forEach()

It perform given action by iterating the elements of iterable until all elements are not processed.

It introduce in 1.8v

Ex

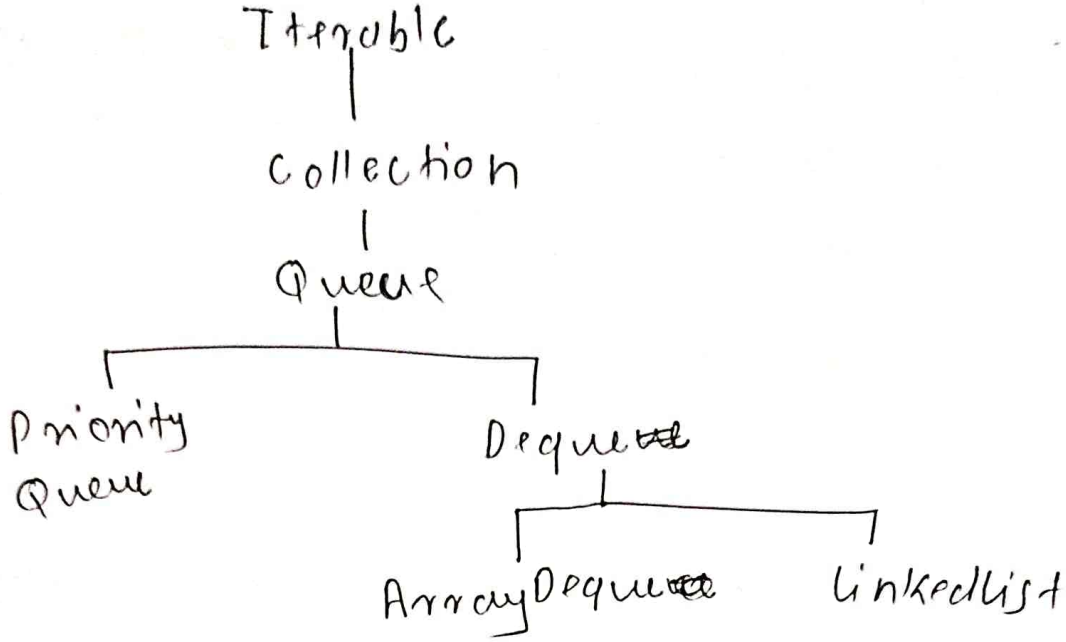
```
TreeSet<Integer> set = new TreeSet<Integer>();
for(int i=1; i<=10; i++)
    set.add(i);

set.stream().filter(ele->ele%2==0).
    forEach(ele->System.out.println(ele+" "));  // 2,4,6,8,10
```

Ex.

```
main(){
ArrayList list = new ArrayList();
for(int i=10; i<=100; i+=10)
    list.add(i);
list.stream().forEach(ele->System.out.println(ele+" "));
```

Queue Interface



① It is sub-Interface of Collection.

② It is ordered list of object used to insert element at end of list. And removing from start of list.

③ Queue is data structure follows a principle first In First Out (FIFO).

④ It introduce in 1.5v & derived in java.util pack.

⑤ Classes which implements Queue are PriorityQueue, LinkedList, ArrayDeque, PriorityBlockingQueue, LinkBlockingQueue, etc

⑥ There are some sub interface like Dequeue, BlockingQueue.

⑦ Queue is implemented using LinkedList, BinaryHeap (is nothing but complete BinaryTree)

⑧ Initial capacity is 11

* methods of Queue

1] public abstract boolean add(E);

2] public abstract boolean offer(E ele);

3] public abstract E remove();

4] public abstract E poll();

5] public abstract E element();

6] public abstract E peek();

1] add(E ele)

It insert the specified element into this queue.

It return true if element added else false.

2] offer(E ele)

It insert the element in queue.

It return true if element added else false.

method throws ClassCastException if specified element prevents it from be added in queue.

It throw NullPointerException if specified element null. And also throw IllegalArgumentException if queue prevent it from be added in Queue.

3] remove()

It retrive and remove head of Queue.
It different from poll() as it throws
an Exception if Queue is Empty.
It throw ~~AU~~ ~~sth~~ NoSuchElementException.

4] poll()

It retrieve & remove head of Queue.
It return null if Queue is empty.

5] element()

It retrieve but doesn't remove head
of Queue it throws NoSuchElementException
if Queue is empty.

6] peek()

It retrieve but doesn't remove head
of Queue it throw null if Queue
is empty.

PriorityQueue

① It is implementation class of Queue interface.

② Introduce in 1.5v & derived in java.util

③ Used when object been processed based on certain priority

④ It called Queue follows FIFO algo, but sometimes element of Queue needed to process according to the priority that's when priority Queue comes into picture.

⑤ Priority Queue is implemented using priority Heap, it nothing but a BinaryHeap (BinaryTree).

⑥ 2 Types of BinaryHeap

1] MaxHeap (descending)

2] Min Heap (Ascending)

⑦ In MaxHeap the root element will be largest element & it stores data in descending order.

(8) In minHeap the root element will d be smallest element & it storey ents data in Ascending order.

[Default implementation of PriorityQueue is minHeap]

(9) It uses Comparable for natural sorting of elements (Ascending)

(10) If we want to customize the sorting we use Comparator.

(11) Null cant add.

Constructors

1] public PriorityQueue()

2] public PriorityQueue(int initialCapacity)

3] public PriorityQueue(Collection c)

4] public PriorityQueue(Comparator c)

5] public PriorityQueue(int initialCap, Comparator c)

2]

It create priorityQueue with specified initial capacity that order its elements according to nature order.

3]

It constructor priori
containing eleming in specified collecth
capacity is equal to no. of elements in
specified coll^n

3] It creates a PriorityQue with default initial capacity (Comparator c) that order it's element according to specified comparator (customer order).

4] It creates PriorityQueue with specified initial capacity that order its elements according to specified comparator.

Deque interface.

1] It is sub interface of Queue which introduce in 1.6V & derive in java.util pack.

2] The name Deque short form "Double Ended Queue" & usually pronounce as Deque.

3] It is linear collection of elements that supports ~~operat~~ insertion & removal operation both end.

4] Java Deque is data structure combing an ordinary stack & Queue.

5] Basically it follow both principle of stack & Queue ie. LIFO & FIFO resply.

6] It has some implementation class like ArrayDeque & LinkedList.

* Methods of Deque

1] public abstract void addFirst(E ele)

2] public abstract ~~boolean~~ void addLast(E ele)

3] public abstract boolean offerFirst(E ele)

4] public abstract boolean offerLast(E ele)

5] public abstract E removeFirst(~~E ele~~)

6] public abstract E removeLast(~~E ele~~)

7] public abstract E pollFirst(~~E ele~~)

8] public abstract E pollLast(~~E ele~~)

9] public abstract E getFirst

10] public abstract E getLast()

11] public abstract E peekFirst()

12] public abstract E peekLast()

13] public abstract boolean
removeFirstOccurrence(Object obj)

14] public abstract boolean
removeLastOccurrence(Object obj)

1] addFirst(E ele)

It insert specified element at front

2] addLast(E ele)

It insert specified element at last (end of Deque)

3] offerFirst(E ele)

It insert specified element at First

4] offerLast(E ele)

It insert element at last of deque

5]

It remove & retrieves the first element of Deque.

6] removeLast()

Remove & retrieve last element

7] pollFirst()

Retrieves & remove the first element of these Deque.
It return null if Deque Empty.

8] ~~ret~~ pollLast()

It remove ~~thod~~ retrive last element.
Return null if Dequeue empty.

9] getFirst()

It retrive but don't remove element.(1st).

10] getLast()

It retrieve but don't remove last element.

11] peekFirst()

It retrieve but don't remove first element return null if Deque is empty.

12] peekLast()

It retrieve last element but don't remove last element
Return null if Deque is empty.

13] removeFirstOccurance(Object obj)

It remove 1st occurance of specified element from Deque.

14] removeLastOccurance(Object obj)

It remove last occurance of specified element from deque.

Ex.

```
class Driver
{
    main()
    {
        Deque d = new ArrayDeque();
        for(int i=10; i<=50; i+=10)
            d.add(i);
        SOP(d);                    // [10,20,30,40,50]
        d.addFirst(70);            // [70,10,20,30,40,50]
        d.addLast(80);             // [70,10,20,30,40,50,80]
        d.offerFirst(90);          // [90,70,10,20,30,40,50,80]
        d.offerLast(100);          // [90,70,10,20,30,40,50,80,100]
        d.removeFirst();           // [70,10,20,30,40,50,80,100]
        d.removeLast();            // [70,10,20,30,40,50,80]
    }
```

Eg.

```
class Emp
{
  main()
  {
    Deque<Integer> d = new ArrayDeque<Integer>();
    for(int i=10; i<=50; i+=0)
    {
        d.add(i);
    }
    d.pollFirst(); // 10   [20, 30, 40, 50]
    d.pollLast(); // 50 [20, 30, 40]
    d.getFirst(); // [20]
    d.getLast(); // [40]
    d.peekFirst(); //
    d.peekLast(); //
  }
}
```

✱ Array Deque

1] It is implementation class of Deque interface.

2] Introduce in 1.6V & in java.util

3] Internal implementation is growable Array where we can perform operation from both end.

4] It not synchronize so it's not thread safe.

5] It faster than stack.

6] Capacity is 16

✱ Constructor

1] public ArrayDeque

It constructs an empty ArrayDeque with initial capacity 16.

2] public ArrayDeque(Collection c)

It constructs new ArrayDeque with specified collection of element.

3] public ArrayDeque(int initialCapacity)

It constructs new ArrayDeque with specified initial capacity

Collections

Collections Method

1] Collections.frequency (Collection c, Object obj)

It return no. of element in the specified Collection equal to specified Object.

Q. Highest frequency of element

```
main(){
  int[] arr = {5,4,6,2,5,2,6,2,7,3,7,8,9,3};
  ArrayList<Integer> list = new ArrayList<Integer>();
  for( int i : arr )
      list.add(i);
  TreeSet<Integer> set = new TreeSet<Integer>(list);
  Integer ele = 0;
  int max = 0;
  for(Integer integer : set)
  {
     int freCurr = Collections.frequency(list, integer)
     if (max< freCurr) {
        max = freCurr;
     } ele = integer;
  }
} sop ( ele +"  " + max);
```

Map

Q. W.A.J.P. To find ~~unique~~ frequency of element. from Array using stream() API.

```
-> {
int [] arr = { 5,4,6,2,5,2,6,2,7,3,7,3,8,9,3};
int [] distArr = Arrays.stream(arr).distinct().
                                    toArray();

for( int i: distArr) {
    int cnt = 0;
    for( int j : arr )
    {
        if (i==j)
            cnt++;
    }
    SOP (i + " " + cnt);
}
}
```

2] Collections.fill (List list, T obj)

It replaces all the elements of specified list with specified element.

Ex

```
main()
{
    List list = new ArrayList();
    list.add(10);
    list.add("Hello");
    list.add("true");
    list.add('A');
    list.add(false);
    SOP(list);
    Collections.fill(list, "Java");
    SOP(list); // [Java, Java, Java, Java]
}
```