

## Collections :

-->java.util package

-->It is a group of interfaces and classes where we define mechanisms to create different types of Collection Objects which follow different Data Structures such as List ,Set ,Queue ,Stack and maps

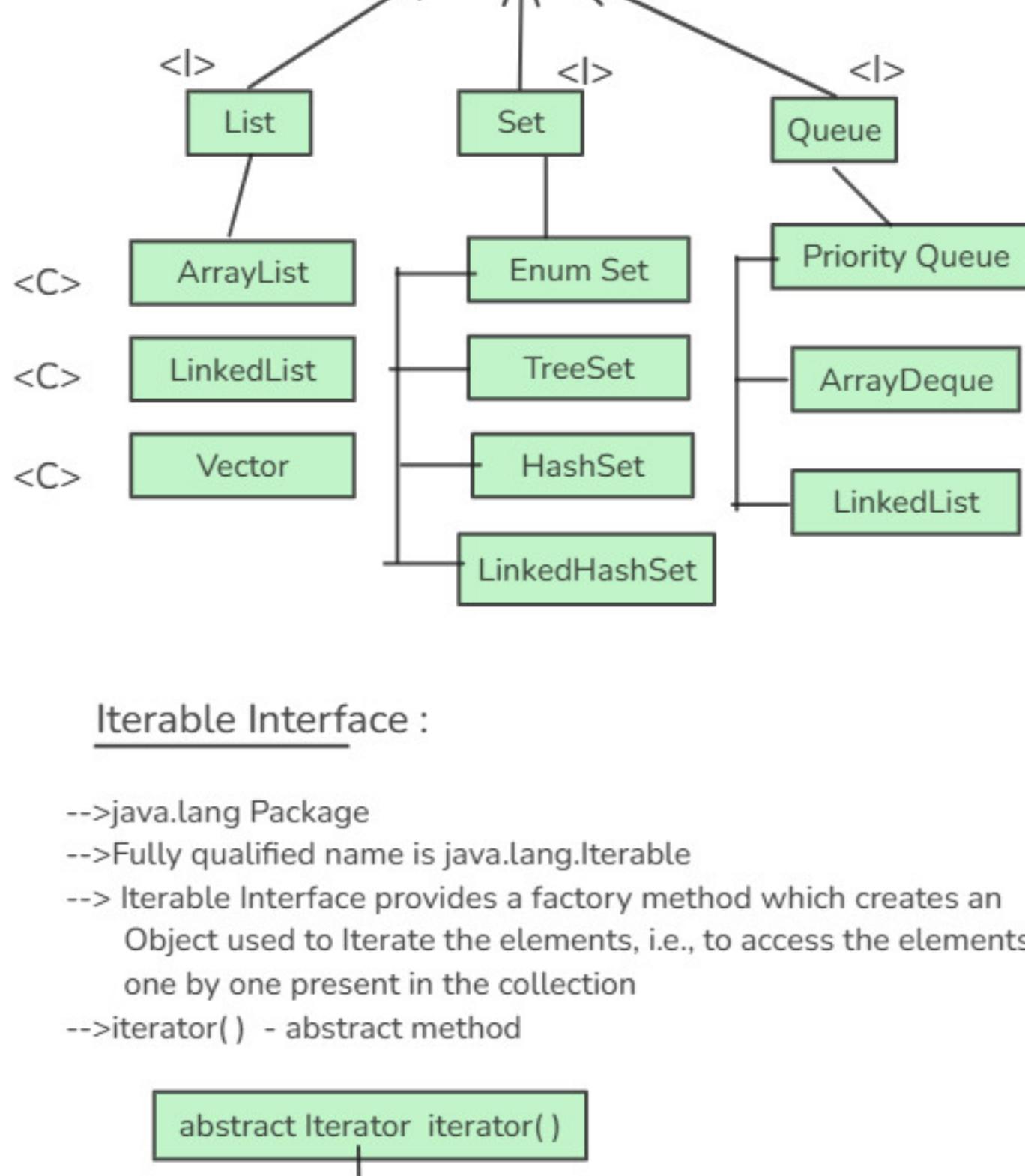
-->Collection Framework contains many Hierarchies , out of which

2 important Hierarchies are

1.Collection Hierarchy

2.Map Hieracrhy

## Collection Hierarchy



## Iterable Interface :

-->java.lang Package

-->Fully qualified name is java.lang.Iterable

--> Iterable Interface provides a factory method which creates an Object used to Iterate the elements, i.e., to access the elements one by one present in the collection

-->iterator( ) - abstract method

abstract Iterator iterator()

return type is Iterator which means this method creates an Instance of Iterator and returns Iterator

## Collection Interface :

-->1)it is an Interface defined in java.util package

-->java.util.Collection

-->2)it has abstract methods to perform operations such as

--Add Elements into Collection Object

--Remove

--Access

--Search

3)The Collection Interface is extending to the Iterable Interface

### To Add the Elements

1.add(Object)-it is used to add an element inside the Collection Object , return type boolean

2.addAll(Collection)-it is used to add all the elements in the given Collection to the Current Collection Object

### To Remove Elements

1.remove(Object) - it is used to remove a element from the given Collection ,return type is boolean

2.removeAll(Collection) - it removes all of the collection elements that are present in the given Collection , return type is boolean

3.retainAll(Collection) - it retains only the elements in the Collection that are present in the given Collection, return type is boolean

4.clear( )-removes all the elements from this Collection

### To Search Elements

1.contains(Object) - it return true if the Collection Contains the specified element , return type is boolean

2.containsAll(Collection) - it returns true if the Collection contains all of the elements in the given Collection , return type is boolean

3.equals(Object) - to compare the specified Object with Collection for equality , return type is boolean

4.size( ) - it returns the number of elements in the Collection , return type is int

5.isEmpty( ) -it returns true if the Collection is empty/ no elements present in it , return type is boolean

6.toArray( ) -it returns an array Containing all the elements in the Collection ,return array[] , object[]

### To Access Elements

-->we can access with the help of iterator( ) inside the Iterable Interface

1.equals(Object) - to compare the specified Object with Collection for equality , return type is boolean

2.hashCode( ) - it returns the hashCode value for the Collection , return type is int

3.size( ) - it returns the number of elements in the Collection , return type is int

4.isEmpty( ) -it returns true if the Collection is empty/ no elements present in it , return type is boolean

5.toArray( ) -it returns an array Containing all the elements in the Collection ,return array[] , object[]

## List :

- >it is an interface defined in java.util package
- >the fully qualified name is java.util.List
- >it is a child of Collection Interface
- >it inherits all the abstract methods from collection interface to perform some generic actions such as
  - 1.add elements
  - 2.remove elements
  - 3.access elements
  - 4.search elements

## Characteristics :

- 1.elements can be accessed/inserted by their position in the list using a zero based index
- 2.list may contain duplicate elements
- 3.list is ordered collection of elements

## to add elements inside list :

- 1.add(int index, Object o) - it inserts the specific element at the specified position in the list
- 2.addAll(int index, Collection c) - it inserts all the elements in the specified collection into this list at specified position

## to remove elements inside list :

- 1.remove(int index) - it removes the element at the specified position in the list
- 2.set(int index, Object o) - it replaces the element at the specified position in the list with a specified element

## to access elements inside list :

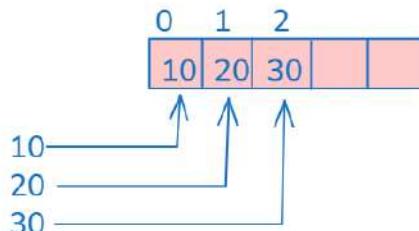
- 1.listIterator( ) - it returns a listIterator over the elements in the list
- 2.listIterator(int index) - it returns a listIterator over the elements in the list, starting at the specified position in the list.
- 3.get(int index) - it returns the element at the specified position in the list
- 4.indexOf(Object o) - it returns the index of first occurrence of the specified element in this list or -1 if this list doesn't contain the element.
- 5.lastIndexOf(Object o) - it returns the index of last occurrence of the specified element in this list or -1 if this list doesn't contain the element

## ArrayList :

- >It is a concrete implementing of ListInterface
- >it is used to store multiple elements
- >it is itself an object which can store multiple elements
- >it is defined in java.util package

### Characteristics of ArrayList :

- 1.it is a collection of objects
- 2.it is an ordered collection of objects(it maintains order of elements)



- 3.it internally uses an array to store the elements(initial capacity is 10)
  - >it is dynamic in growth
- 4.it has indexing therefore we can insert or remove elements with help of index
- 5.ArrayList can have duplicate objects/elements

### to create ArrayList :

constructors of ArrayList :

- 1.ArrayList( )
- 2.ArrayList(int initial Capacity)
- 3.ArrayList(Collection)

### to add elements inside ArrayList :

```
6 class Demo {  
7 public static void main(String[] args) {  
8     ArrayList a1 =new ArrayList();  
9     a1.add(10);  
10    a1.add(20);  
11    a1.add("hai");  
12    a1.add('k');  
13    a1.add(15);  
14    a1.add(true);  
15    System.out.println(a1);  
16}
```

Console  
[10, 20, hai, k, 15, true]

add(int index, Object o) -

```
4  
5 class Example {  
6 public static void main(String[] args) {  
7     ArrayList a=new ArrayList();  
8     a.add(10);  
9     a.add(2.5);  
10    a.add(true);  
11    System.out.println(a);  
12    a.add(1,"hai");  
13    System.out.println(a);  
14 }  
15 }
```

Console  
[10, 2.5, true]  
[10, hai, 2.5, true]

to add non-primitive objects :

The screenshot shows the Eclipse IDE interface. On the left, there is a 'Package Expl.' view with several files listed. In the center, there are two code editors. The left editor contains the 'Emp.java' code:

```
1 package arraylistt;
2
3 public class Emp {
4     int eid;
5     String name;
6     double sal;
7     public Emp(int eid, String name, double sal)
8     {
9         super();
10        this.eid = eid;
11        this.name = name;
12        this.sal = sal;
13    }
14 }
15
```

The right editor contains the 'Example.java' code:

```
1 package arraylistt;
2
3 import java.util.ArrayList;
4
5 class Example {
6     public static void main(String[] args) {
7         ArrayList a=new ArrayList();
8         a.add(10);
9         a.add(2.5);
10        a.add(true);
11        Emp e=new Emp(101,"Dinga",15.000);
12        Emp e1=new Emp(102,"Dingi",25.000);
13        a.add(e);
14        a.add(e1);
15        System.out.println(a);
16    }
17 }
18 }
```

Below the code editors is a 'Console' tab. The output window shows the results of running the 'Example' application:

```
[terminated> Example (15) [Java Application] C:\Users\OSP-Trainer\p2\pool\plugins\org.eclipse.jdt.core\openjdk\hotspot\jre\full\win32\x86_64_21.0.1\2020-03-17T11:45:00Z
[10, 2.5, true, arraylistt.Emp@4aa8f0b4, arraylistt.Emp@4aa8f0b4]
```

addAll(Collection) and addAll(int index,Collection c) -

The screenshot shows the Eclipse IDE interface. A single code editor displays the 'Menu.java' code:

```
5 class Menu {
6     public static void main(String[] args) {
7         ArrayList keralaMenu=new ArrayList();
8         keralaMenu.add("putt");
9         keralaMenu.add("idi appam");
10        keralaMenu.add("appam");
11        ArrayList HydMenu=new ArrayList();
12        HydMenu.add("Masala Dosa");
13        HydMenu.add("Poori");
14        HydMenu.add("Idly");
15        ArrayList Menu=new ArrayList(keralaMenu);
16        Menu.addAll(HydMenu);
17        System.out.println(Menu);
18        //addAll(int index,Collection c)
19        ArrayList a=new ArrayList();
20        a.add(1);
21        a.add(2);
22        a.addAll(1,HydMenu);
23        System.out.println(a);
24    }
25 }
26
```

Below the code editor is a 'Console' tab. The output window shows the results of running the 'Menu' application:

```
[terminated> Menu [Java Application] C:\Users\OSP-Trainer\p2\pool\plugins\org.eclipse.jdt.core\openjdk\hotspot\jre\full\win32\x86_64_21.0.1\2020-03-17T11:45:00Z
[putt, idi appam, appam, Masala Dosa, Poori, Idly]
[1, Masala Dosa, Poori, Idly, 2]
```

## Traverse the elements using Iterator :

iterator is used to access each and every element in the Collections

or ArrayList or any Collection object

-->it is present in java.util.Iterator

-->iterator has 2 abstract methods

1.next( ) - when we call 2 actions are performed

i)it returns the element on which the cursor is present

ii)it moves to the next position or one step forward

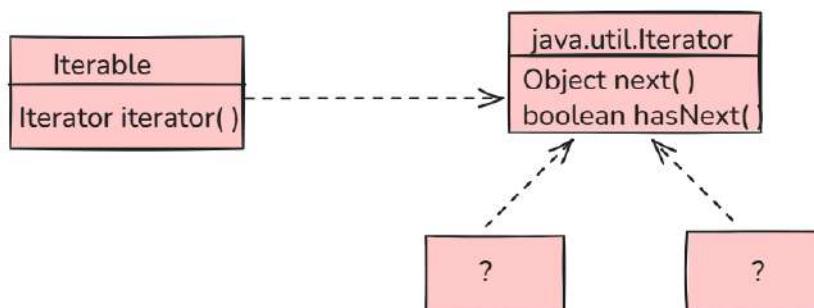
2.hasNext( ) - it returns true, if an element is available in the

iterator for accessing , else it returns false

-->we cannot create an object for iterator

-->this iterator( ) will create an instance of iterator using any of the implementing class

-->it returns the address of the object created

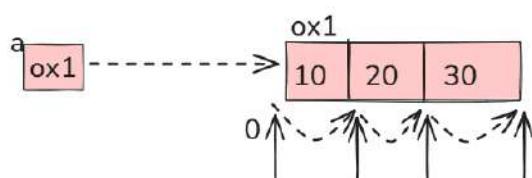


-->iterator( ) is called as Factory method because it is responsible to create the object and give the reference of that particular object

-->the return type is Iterator

```
ArrayList a=new ArrayList();
a.add(10);
a.add(20);
a.add(30);
//Iterator Object
```

Iterator i=a.iterator()



```
Sopln(i.next());//10
Sopln(i.next());//20
Sopln(i.next());//30
Sopln(i.next());//NoSuchElementException
```

### NoSuchElementException :

When we try to access the element using next( ),but we dont have any element at that particular position or index

or

When we call next( ) method on an iterator, and iterator doesnot haе an element to return we get NoSuchElementException

```
6 class Main {
7 public static void main(String[] args) {
8     ArrayList a=new ArrayList();
9     a.add(10);
10    a.add(20);
11    a.add(30);
12    Iterator i=a.iterator();
13    System.out.println(i.next());//10
14    System.out.println(i.next());//20
15    System.out.println(i.next());//30
16    System.out.println(i.next());//NoSuchElementException
17 }
```

10  
20  
30  
Exception in thread "main" java.util.NoSuchElementException  
at java.base/java.util.ArrayList\$ListIterator.next(ArrayList.java:1052)

## Limitations of Iterator :

-->we cannot access the elements in reverse order

-->we cannot modify the collection object such as adding a new element, remove element

```
6 class Main {  
7     public static void main(String[] args) {  
8         ArrayList a=new ArrayList();  
9         a.add(10);  
10        a.add(20);  
11        a.add(30);  
12        Iterator i=a.iterator();  
13        while(i.hasNext()) {  
14            System.out.println(i.next());  
15        }  
16    }  
17 }
```

Console X  
Main.java:13: error: cannot find symbol  
System.out.println(i.next());  
symbol: variable System.out  
location: class Main  
10  
20  
30

## Traverse the elements using ListIterator :

-->List Iterator is an interface defined in java.util package

-->methods are

- 1.next() > inherited from Iterator interface
- 2.hasNext()
- 3.hasPrevious()
- 4.previous()
- 5.nextInt()
- 6.previousIndex()
- 7.remove()
- 8.set(E e)
- 9.add(E e)

-->it is having 2 factory methods

- 1.listIterator()
- 2.listIterator(int index) > Factory methods helps to create instance of List Iterator

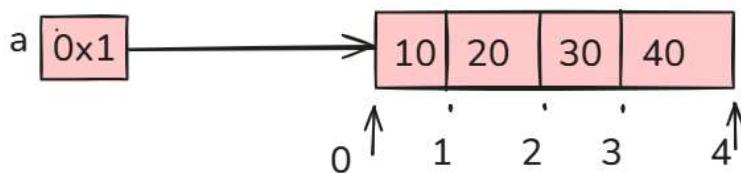
-->these methods can be only used for List type of Objects  
(cannot be used for Set,Queue...)

-->listIterator is available only for List

```
3 import java.util.ArrayList;  
4 import java.util.ListIterator;  
5  
6 class Main {  
7     public static void main(String[] args) {  
8         ArrayList a=new ArrayList();  
9         a.add(10);  
10        a.add(20);  
11        a.add(30);  
12        ListIterator i=a.listIterator();  
13        System.out.println(i.next());  
14        ListIterator il=a.listIterator(2);  
15        System.out.println(il.next());  
16    }  
17 }
```

## Characteristics of ListIterator :

ListIterator i1=a.listIterator()



-->whenever we use listIterator( ),by default the cursor will be pointing towards 0th index

-->listIterator(1) ->it points towards 1st index

## Advantage of ListIterator over Iterator :

-->it is both forward and backward traversal

```

6 public class Check {
7     public static void main(String[] args) {
8         ArrayList a=new ArrayList();
9         a.add(10);
10        a.add(20);
11        a.add(30);
12        //to traverse a in forward direction using ListIterator
13        ListIterator forward=a.listIterator();
14        while(forward.hasNext()) {
15            System.out.println(forward.next());
16        }
17        //to traverse in reverse order using ListIterator
18        ListIterator backward=a.listIterator();
19        while(backward.hasPrevious()) {
20            System.out.println(backward.previous());
21        }
22    }

```

Console

```

10
20
30

```

## Traversing the elements using for each loop :

```

for(datatype variable : array/Collection)
{
statements...
}

```

-->the elements present inside the collection

are stored inside the variable declared inside the loop

-->the loop iterates based on the no.of elements present inside the loop

## To Search Elements :

to search , keyelement is necessary



which elements we want to search

-->it is 2 types(Possibilities)

1.Object

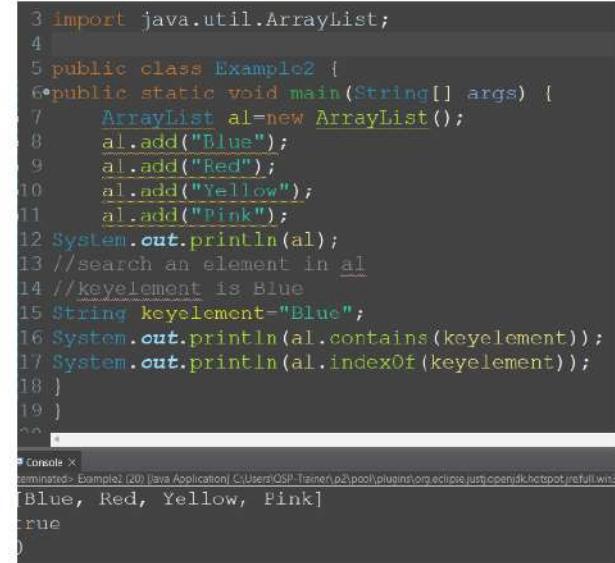
2.attribute/properties/fields

`contains(Object o) --> returns true/false`  
`indexOf(Object o) --> it returns position`



```
import java.util.ArrayList;

public class Example {
    public static void main(String[] args) {
        ArrayList al=new ArrayList();
        al.add(10);
        al.add(20);
        al.add(30);
        al.add(40);
        System.out.println(al);
        //search an element in al
        //keyelement is 30
        int keyelement =30;
        System.out.println(al.contains(keyelement));
        System.out.println(al.indexOf(keyelement));
    }
}
```



```
3 import java.util.ArrayList;
4
5 public class Example2 {
6     public static void main(String[] args) {
7         ArrayList al=new ArrayList();
8         al.add("Blue");
9         al.add("Red");
10        al.add("Yellow");
11        al.add("Pink");
12        System.out.println(al);
13        //search an element in al
14        //keyelement is Blue
15        String keyelement="Blue";
16        System.out.println(al.contains(keyelement));
17        System.out.println(al.indexOf(keyelement));
18    }
19 }
```

## to search elements (Custom Object)



```
public class Student {
    private int sid;
    private String sname;
    private int age;
    //setters and getters for all properties
    public int getSid() {
        return sid;
    }
    public void setSid(int sid) {
        this.sid = sid;
    }
    public String getSname() {
        return sname;
    }
    public void setSname(String sname) {
        this.sname = sname;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    //constructor
    Student(int sid, String sname, int age) {
        setSid(sid);
        setSname(sname);
        setAge(age);
    }
    //override toString method
    public String toString() {
        return sid+", "+sname+", "+age;
    }
}

import java.util.*;
import java.util.ListIterator;
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        ArrayList<Student> student = new ArrayList<Student>();
        student.add(new Student(101,"Rajesh",25));
        student.add(new Student(102,"Rakesh",20));
        student.add(new Student(103,"Naman",15));
        student.add(new Student(104,"Rishi",17));
        //remove student based on id
        //remove student id 102
        student.remove((Student)student.get(1));
        System.out.println("After removing student id : "+102);
        System.out.println("Total student : "+student.size());
        boolean NotFound = true;
        while(NotFound) {
            Scanner temp = new Scanner(System.in);
            System.out.print("Enter student id : ");
            int tempId = temp.nextInt();
            ListIterator<Student> listIterator = student.listIterator();
            for(Student student1 : listIterator) {
                if(student1.getId() == tempId) {
                    NotFound = false;
                    System.out.println("Student found with id : "+tempId);
                    System.out.println();
                }
            }
        }
        if(NotFound) {
            System.out.println("Student doesn't exist...!");
        }
    }
}
```

## creating own method to search element based on sid

```
import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;
import java.util.Scanner;

public class Test {
    //own method to search element based on student id
    public static Student searchByStudentId(Collection data, int keyelement) {
        //logic to search
        Iterator i=data.iterator();
        //iterate over students
        while(i.hasNext()) {
            Student temp=(Student)i.next();
            if(temp.getId()==keyelement)
                return temp;
        }
        //when student doesn't exist
        return null;
    }

    public static void main(String[] args) {
        ArrayList students=new ArrayList();
        students.add(new Student("Kavyaa",25));
        students.add(new Student("Hari",25));
        students.add(new Student("Anurita",24));
        //search student based on student id
        Scanner s=new Scanner(System.in);
        System.out.println("Enter student id :");
        int keyelement=s.nextInt();
        Student res=searchByStudentId(students, keyelement);
        if(res!=null) {
            System.out.println("Student Exist :");
            System.out.println(res.getName());
        } else {
            System.out.println("Student doesn't exist");
        }
    }
}
```

## To search element using non unique attribute

```
import java.util.ArrayList;
import java.util.Scanner;

public class Student {
    public static void main(String[] args) {
        ArrayList students=new ArrayList();
        students.add(new Student("Kavyaa",25));
        students.add(new Student("Hari",25));
        students.add(new Student("Anurita",24));
        //search student based on student id
        Scanner s=new Scanner(System.in);
        System.out.println("Enter student id :");
        int keyelement=s.nextInt();
        boolean notFound=true;
        for(Object element:students) {
            Student temp=(Student)element;
            if(temp.getId()==keyelement) {
                notFound=false;
                System.out.println(temp.getName());
            }
        }
        if(notFound) {
            System.out.println("Student doesn't exist for the given id");
        }
    }
}
```

## To remove the element - using index

```
3 import java.util.ArrayList;
4
5 public class Example2 {
6     public static void main(String[] args) {
7         ArrayList al=new ArrayList();
8         al.add("Blue");
9         al.add("Red");
10        al.add("Yellow");
11        al.add("Pink");
12        System.out.println(al);
13        //remove first element
14        al.remove(0);
15        System.out.println(al);
16        al.remove(al.size()-1);
17        System.out.println(al);
18    }
19 }
```

The screenshot shows the Java code running in an IDE. The console output shows the initial state of the list [Blue, Red, Yellow, Pink], then after removing the first element (index 0), the list becomes [Red, Yellow, Pink], and finally after removing the last element (index 2), the list becomes [Red, Yellow].

### Note :

if the index passed is greater than size of list , we get  
IndexOutOfBoundsException

## To remove the element - using Object

```
4
5 public class Example2 {
6     public static void main(String[] args) {
7         ArrayList<String> al=new ArrayList<String>();
8         al.add("Kavyaa");
9         al.add("Dingii");
10        al.add("Dingga");
11        al.add("Kuttyy");
12        System.out.println("****Before removing****");
13        for(String names:al) {
14            System.out.println(names);
15        }
16        //remove Kuttyy
17        System.out.println("****After removing****");
18        System.out.println(al.remove("Kuttyy"));
19        //after removing
20        for(String namesal) {
21            System.out.println(namesal);
22        }
23    }
24 }
```

The screenshot shows the Java code running in an IDE. The console output shows the initial state of the list [Kavyaa, Dingii, Dingga, Kuttyy] with the header '\*\*\*\*Before removing\*\*\*\*'. After removing the element 'Kuttyy', the list becomes [Kavyaa, Dingii, Dingga] with the header '\*\*\*\*After removing\*\*\*\*'. A boolean value 'true' is also printed, likely indicating the result of the remove operation.

## To remove the element - using attribute

### create Blueprint Driver Class

The screenshot shows a Java code editor with two panes. The left pane contains the following code:

```
* import java.util.ArrayList;
* import java.util.Iterator;
* import java.util.Scanner;
*
* import com.safal.*;
*
* public class EmployeeMain {
*     public static void main(String[] args) {
*         ArrayList<Employee> emp=new ArrayList<>();
*         emp.add(new Employee(401,"Kinggg",50000));
*         emp.add(new Employee(402,"Queen",96000));
*         emp.add(new Employee(403,"Soldier",175000));
*         emp.add(new Employee(404,"Rocky",70000));
*         System.out.println("Enter name of the emp to be removed from list");
*         Scanner s=new Scanner(System.in);
*         String name=s.nextLine();
*         //step 1 : search by attribute
*         //step 2 : if element found preserve the object and then remove using remove(object)
*         Iterator<Employee> i=emp.iterator();
*         Employee tobeRemoved=null;
*         while(i.hasNext()) {
*             Employee e=i.next();
*             if(e.getEname().equalsIgnoreCase(name)) {
*                 tobeRemoved=e;
*             }
*         }
*         if(tobeRemoved==null) {
*             emp.remove(tobeRemoved);
*             System.out.println("****After Removal****");
*             for(Employee e:emp) {
*                 System.out.println(e);
*             }
*         } else {
*             System.out.println("Emp doesn't exist");
*         }
*     }
* }
```

The right pane shows the output of the program:

```
enter name of the emp to be removed from list
Kinggg
****After Removal****
402,Queen,96000
403,Soldier,175000
401,Rocky,70000
```

The screenshot shows a Java code editor with two panes. The left pane contains the following code:

```
1 package notesNLPN;
2
3 import java.util.ArrayList;
4
5 public class Employee {
6     private int eid;
7     private String ename;
8     private int sal;
9
10     //setters and getters for all properties
11     public int getEid() {
12         return eid;
13     }
14     public void setEid(int eid) {
15         this.eid = eid;
16     }
17     public String getEname() {
18         return ename;
19     }
20     public void setEname(String ename) {
21         this.ename = ename;
22     }
23     public int getSal() {
24         return sal;
25     }
26     public void setSal(int sal) {
27         this.sal = sal;
28     }
29
30     //conventions to initialize properties
31     @Override
32     public String toString() {
33         return eid+","+ename+","+sal;
34     }
35 }
```

The right pane shows the output of the program:

```
enter name of the emp to be removed from list
Kinggg
****After Removal****
402,Queen,96000
403,Soldier,175000
401,Rocky,70000
```

## To Remove element - using clear()

**clear()** - it is used to remove all the elements from arraylist  
**removeAll(Collection c)** - removes from this list all of its elements that are contained in the specified collection

The screenshot shows a Java code editor with two panes. The left pane contains the following code:

```
3 import java.util.ArrayList;
4
5 public class Test1 {
6     public static void main(String[] args) {
7         ArrayList a1=new ArrayList();
8         a1.add(10);
9         a1.add(20);
10        a1.add(30);
11        a1.add(20);
12        ArrayList a2=new ArrayList();
13        a2.add(40);
14        a2.add(20);
15        a2.add(30);
16        a2.add(50);
17        //removeAll removes all the elements of a1
18        //is present in a2
19        a1.removeAll(a2);
20        System.out.println(a1); // [10]
21        System.out.println(a2); // [40, 20, 30, 50]
22    }
23 }
24 }
```

`retainAll(Collection c)` : it returns all the elements in this list that are contained in the specified collection

```
1 // Java program to demonstrate
2
3 import java.util.ArrayList;
4
5 public class Test1 {
6     public static void main(String[] args) {
7         ArrayList a1=new ArrayList();
8         a1.add(10);
9         a1.add(20);
10        a1.add(30);
11        a1.add(20);
12        ArrayList a2=new ArrayList();
13        a2.add(40);
14        a2.add(20);
15        a2.add(30);
16        a2.add(50);
17        //retainsAll removes all the elements of a1
18        //which are not present in a2 |
19        a1.retainAll(a2);
20        System.out.println(a1); // [20, 30, 20]
21        System.out.println(a2); // [40, 20, 30, 50]
22    }
23 }
24 }
```

```
[20, 30, 20]
[40, 20, 30, 50]
```

## Comparable Interface :

Comparable is an interface used to define the natural ordering of objects of a class.

-->it is a functional Interface present inside java.lang package

-->Use Comparable when your class has a single natural order (e.g., String by lexicographic order).

--> to sort objects of your custom class (e.g., with Collections.sort() or in a TreeSet), you implement Comparable.

### method definition :

```
public interface Comparable<T> {  
    int compareTo(T o);  
}
```

-->if our class implements Comparable<T> we should define compareTo().

### Example :

```
public class Person implements Comparable<Person> {  
    private String name;  
    private int age;  
  
    public Person(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
  
    @Override  
    public int compareTo(Person p) {  
        return this.age-p.age;  
    }  
  
    @Override  
    public String toString() {  
        return name + " (" + age + ")";  
    }  
}
```

### Driver Class :

```
import java.util.*;  
  
public class Main {  
    public static void main(String[] args) {  
        List<Person> people = new ArrayList<>();  
        people.add(new Person("Kavya", 25));  
        people.add(new Person("Harika", 22));  
        people.add(new Person("Dingaa", 27));  
  
        Collections.sort(people);  
  
        for (Person p : people) {  
            System.out.println(p);  
        }  
    }  
}
```

Hari (22)  
Kavya (25)  
Dingaa (27)

### Note :

If compareTo(a, b) == 0, then ideally a.equals(b) should be true.

### Use comparable in following cases :

-->Sorting lists of objects

-->Using sorted collections such as:

TreeSet

TreeMap

-->Arrays sorted with Arrays.sort()

## Comparator Interface :

- >The Comparator interface is defined for custom sorting logics outside the class we want to sort
- >It compares two objects of a type and tells Java which one should come first when sorting.
- >it is Functional interface which is present inside java.util package
- >it contains method  
    int compare(T o1, T o2)

### Use cases :

- >to sort the same class in different ways
- > to avoid the modification of class everytime we do sorting
- > when we need sorting logic that's external and flexible
- > when we want to sort based on different fields (e.g., name, age, salary)

Example : sorting by trainer id

```
import java.util.Comparator;

public class SortById implements Comparator<Trainer>{
//override compare
    public int compare(Trainer t1, Trainer t2) {
        return t1.tid-t2.tid;
    }
}

public class Trainer {
    int tid;
    String tname;
    String sub;
    Trainer(int tid, String tname, String sub) {
        this.tid=tid;
        this.tname=tname;
        this.sub=sub;
    }
    //override toString()
    public String toString() {
        return tid+" , "+tname+" , "+sub;
    }
}
```

```
package demoCom;

import java.util.ArrayList;

public class TrainerDriver {
    public static void main(String[] args) {
        ArrayList<Trainer> a=new ArrayList<>();
        a.add(new Trainer(404, "Harika", "Java"));
        a.add(new Trainer(427, "Kavya", "SQL"));
        a.add(new Trainer(414, "Amruta", "Selenium"));
        System.out.println("*****Before Sorting*****");
        for(Trainer t:a) {
            System.out.println(t);
        }
        System.out.println("*****After Sorting with id*****");
        Collections.sort(a,new SortById());
        for(Trainer t:a) {
            System.out.println(t);
        }
    }
}
```

```
<terminated> TrainerDriver [Java Application] C:\Users\QSP-Trainer\d2\pool\plugins
*****Before Sorting*****
404 , Harika , Java
427 , Kavya , SQL
414 , Amruta , Selenium
*****After Sorting with id*****
404 , Harika , Java
414 , Amruta , Selenium
427 , Kavya , SQL
```

## Sorting by trainer name :

```
import java.util.Comparator;

public class SortByName implements Comparator<Trainer> {
    //override compare
    public int compare(Trainer t1, Trainer t2) {
        return t1.tname.compareTo(t2.tname);
    }

    public class Trainer {
        int tid;
        String tname;
        String sub;
        Trainer(int tid, String tname, String sub) {
            this.tid=tid;
            this.tname=tname;
            this.sub=sub;
        }
        //override toString()
        public String toString() {
            return tid+" , "+tname+" , "+sub;
        }
    }
}
```

```
package demoCom;

import java.util.ArrayList;

public class TrainerDriver {
    public static void main(String[] args) {
        ArrayList<Trainer> a=new ArrayList<>();
        a.add(new Trainer(404, "Harika", "Java"));
        a.add(new Trainer(427, "Kavya", "SQL"));
        a.add(new Trainer(414, "Amruta", "Selenium"));
        System.out.println("*****Before Sorting*****");
        for(Trainer t:a) {
            System.out.println(t);
        }
        System.out.println("*****After sorting with Name*****");
        Collections.sort(a, new SortByName());
        for(Trainer t:a) {
            System.out.println(t);
        }
    }
}
```

```
<terminated> TrainerDriver [Java Application] C:\Users\QSP-Trainer\p2\pool\plugins\or
*****Before Sorting*****
404 , Harika , Java
427 , Kavya , SQL
414 , Amruta , Selenium
*****After sorting with Name*****
414 , Amruta , Selenium
404 , Harika , Java
427 , Kavya , SQL
```

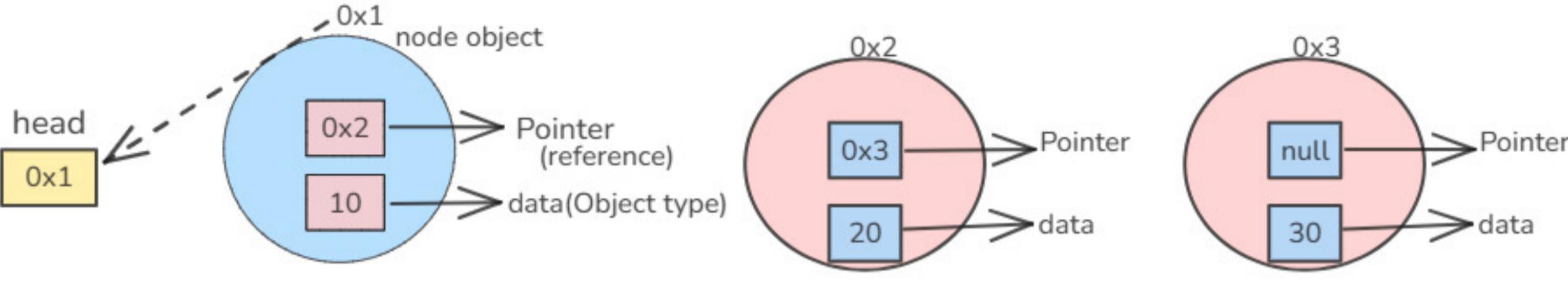
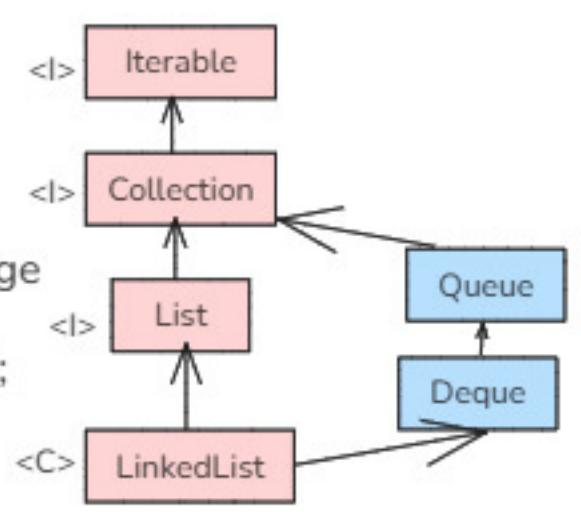
### LinkedList :

-->it is the Implementing class  
for List and Deque Interface  
-->it is present in the java.util package

LinkedList l1=new LinkedList();

-->it internally uses "node"

Ex : 10  
20  
30



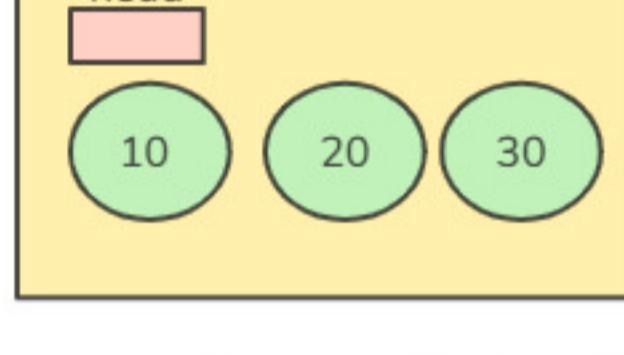
-->inside linkedlist one more component is there called as  
"head"



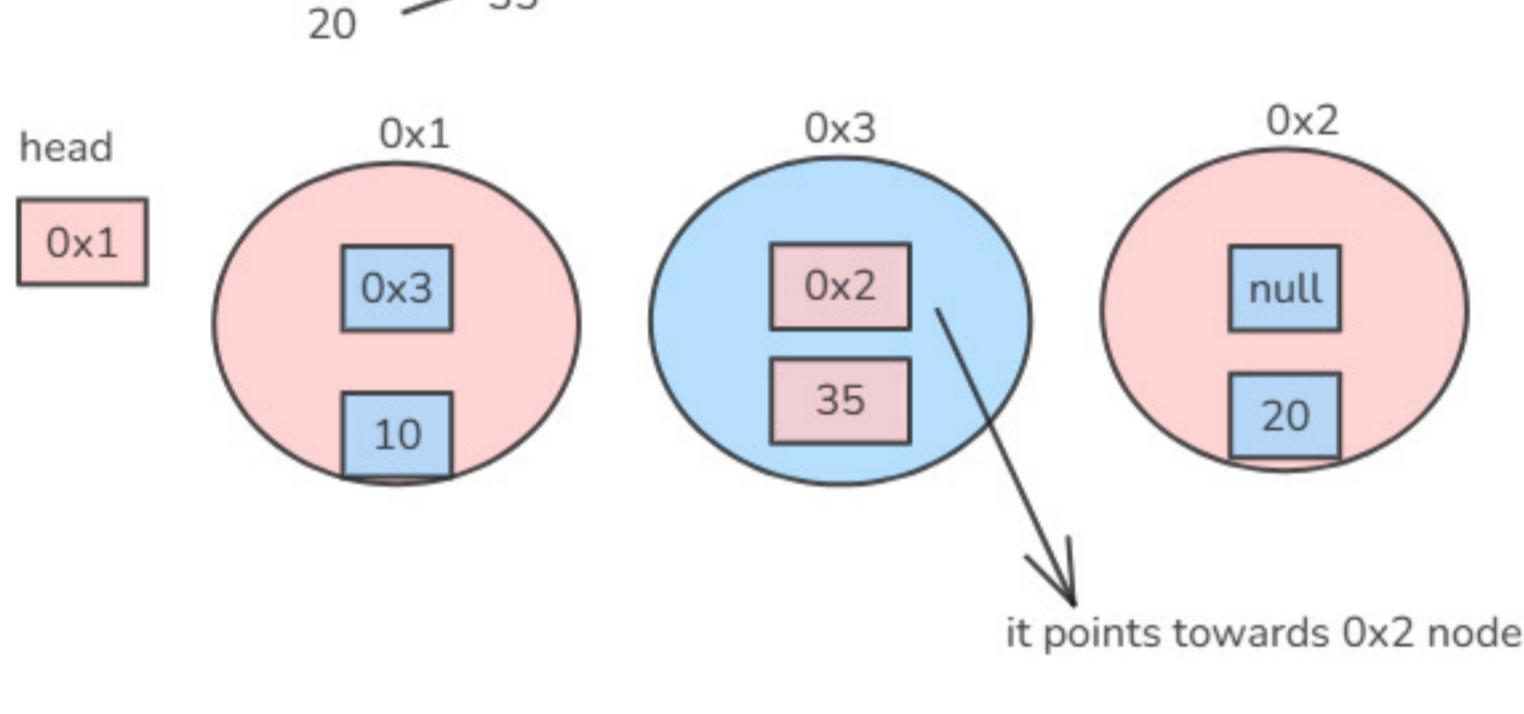
it is used to store the reference of the first node object  
created for LinkedList Object

-->every LinkedList will have one head and multiple nodes

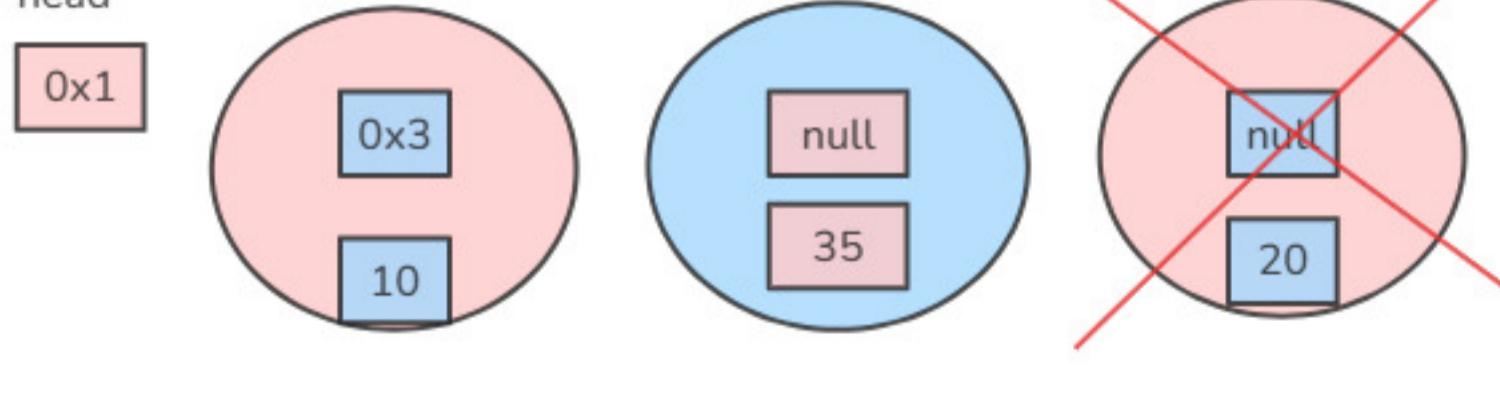
### LinkedList



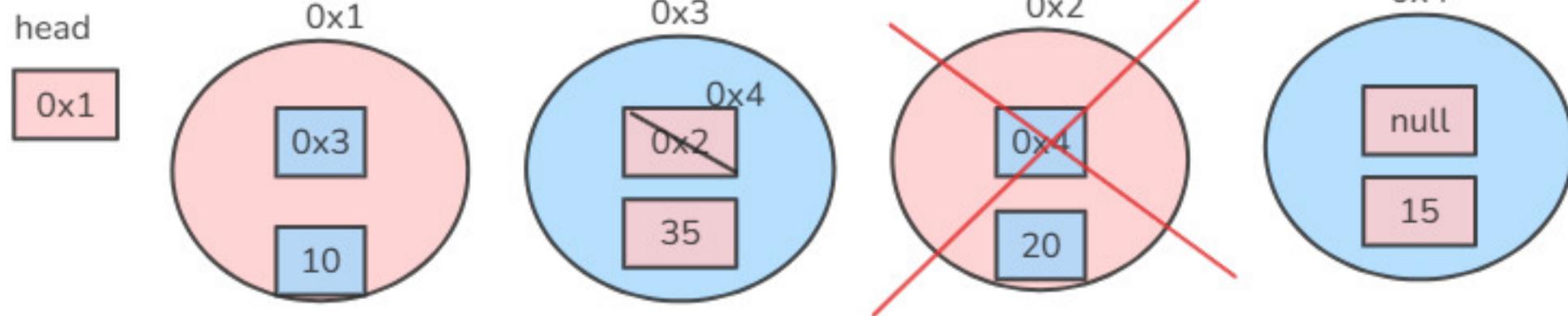
--> to increase the efficiency while inserting/removing the values  
from the collection , we can achieve the time complexity



-->if i want to remove 20 ,

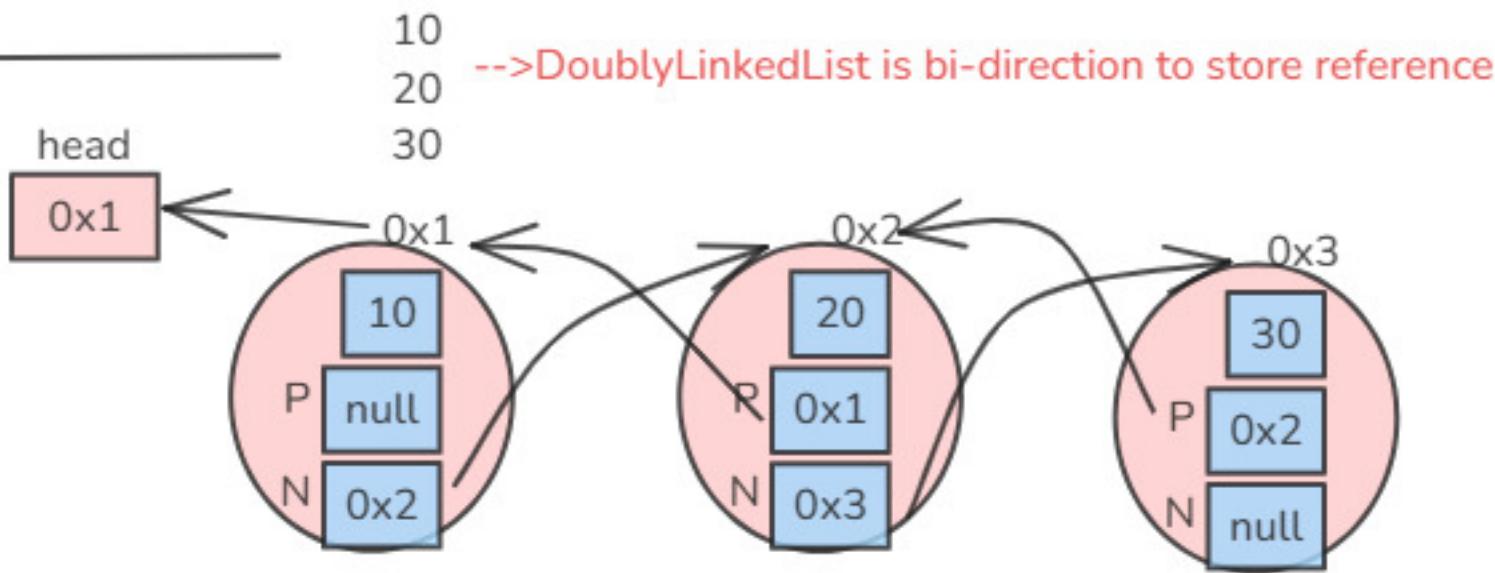


-->if we want to remove 20, the address 0x2 will be removed from 0x3 object  
,null will be stored as there is no next node and that node whatever got removed  
will be taken by Garbage Collector



-->if we want to remove 20, the address 0x2 will be removed from 0x3 object  
,next node address will be stored and that node whatever got removed  
will be taken by Garbage Collector

- >Index depends on number of nodes
- >LinkedList is more efficient than ArrayList
- >the initial capacity for LinkedList is zero



### Characteristics of LinkedList :

- 1.insertion order is maintained
- 2.indexing is there
- 3.it allows duplicate values
- 4.it allows null values
- 5.it accepts both Homogeneous and Heterogeneous type of data

### Vector :

- >after 1.5 they added in Collection Framework
- >the initial capacity for vector object is 10
- >the increasing factor is 50%(for arraylist).
- >the increasing factor for vector is 2x
- 16-->32
- 32-->64

### Stack :

Vector <--- Child of <--- Stack

**push( )** - whenever we call, it will add element in last position

**peek( )** - it access last element

**empty( )** - it will check whether it is empty or not

**pop( )** -it will remove the element

**search( )** - it will check whether the object is present or not  
if it is there it returns its position



it is 1 based index

(the last element you add in the Stack contains the 1st Index)

## Vector

-->it is a legacy synchronized class in Java that implements List and uses a dynamic array to store elements.

legacy class :

a class which is introduced in java 1.0 version

-->it is present inside java.util package

-->it implements List interface

characteristics :

-->maintains insertion order

-->it stores duplicate elements

-->it is thread safe(all methods are synchronized)

Internal working :

-->it uses Dynamic Array

-->the default capacity is 10

-->when capacity is full it dynamically increases the internal capacity

$$\text{New capacity} = \text{OldCapacity} * 2$$

Constructors of Vector :

1.Vector() - array is created with default capacity 10

2.Vector(int capacity) - we can give our own initial capacity

3.Vector(int capacity,int increment) - it grows the array of fixed capacity

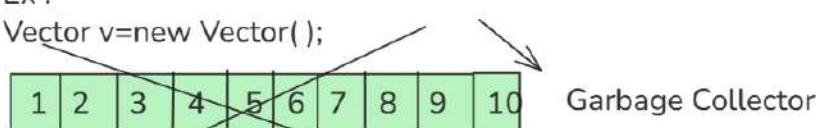
4.Vector(Collection c) - it accepts the another collection

Legacy methods :

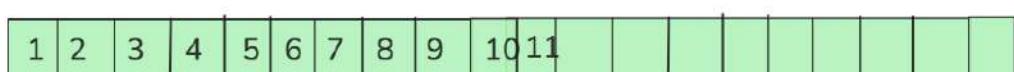
```
... addElement(Object obj)
        insertElementAt(Object obj, int index)
        removeElement(Object obj)
        removeElementAt(int index)
        removeAllElements()
        Object firstElement()
        Object lastElement()
```

Ex :

Vector v=new Vector();

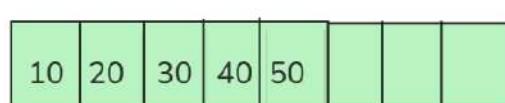
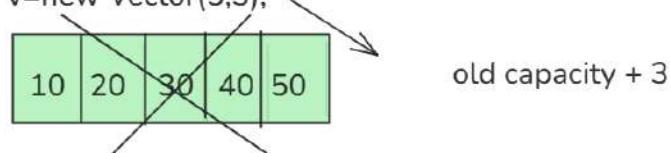


when we try to add 11th element , an array will be created internally with double capacity i.e., 20 and the old array will be collected by Garbage Collector



Vector with incremental capacity :

Vector v=new Vector(5,3);



now the capacity is  $5+3 = 8$

## Stack

-->Stack is a legacy synchronized class in Java  
that extends Vector and follows the LIFO principle.

LIFO - Last In First Out

-->it is present inside java.util package  
-->it is child of Vector

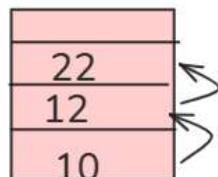
```
public class Stack<E> extends Vector<E>
```

-->Stack is Thread Safe  
-->Stack does not have its own storage it internally uses  
Dynamic Array of Vector  
-->so the capacity , growth and synchronization are same  
as Vector claass  
-->Stack does not have its own constructors,as it is extending to  
Vector we can use Vector Constructors  
-->Stack contains some extra methods  
1.push() - it adds element to top of stack  
2.pop() - it removes and returns top element  
3.peek() - it returns top element without removing  
4.empty() - it checks whether if stack is empty or not  
5.search() - it returns the position of element from the top of  
the Stack

Ex :

```
Stack s=new Stack();
```

```
    s.add(10);  
    s.push(12);  
    s.push(22);
```



```
import java.util.Stack;  
  
public class Test {  
public static void main(String[] ar  
Stack s=new Stack();  
s.add(10);  
s.push(12);  
System.out.println(s.pop());  
System.out.println(s);  
}
```

```
12  
[10]
```

```
import java.util.Stack;  
  
public class Test {  
public static void main(String[] ar  
Stack s=new Stack();  
s.add(10);  
s.push(12);  
System.out.println(s.peek());  
System.out.println(s);  
}
```

```
12  
[10, 12]
```

```
import java.util.Stack;  
  
public class Test {  
public static void main(String[] ar  
Stack s=new Stack();  
s.add(10);  
s.push(12);  
s.push(30);  
s.push(25);  
System.out.println(s.empty());  
System.out.println(s.search(25));  
}
```

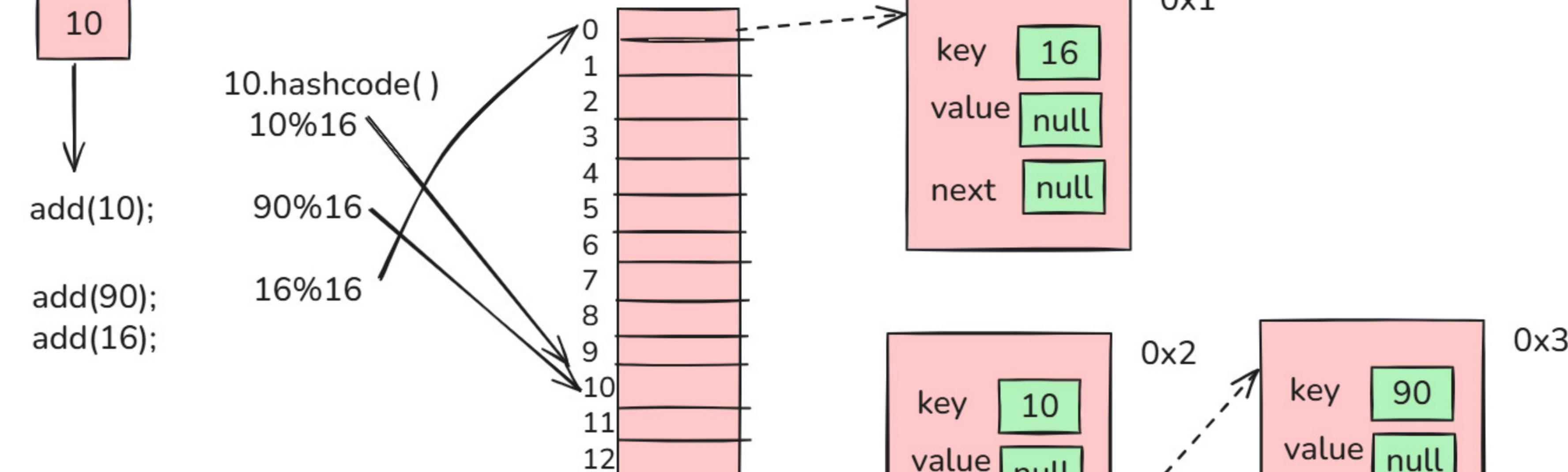
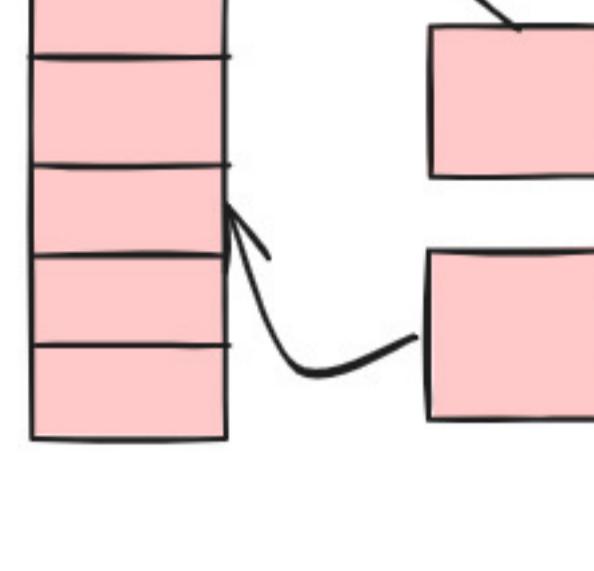
```
false  
1  
25  
30  
12  
10
```

### Set :

- >no indexing
- >insertion order is not there
- >we can store both heterogeneous and homogeneous
- >only 1 null value is allowed
- >no duplicates

### HashSet :

- >The internal implementation is HashMap
- >data gets stored in form of Array of Buckets



- >the default size of an array inside hashset is 16
- >HashSet calls Hashing Function
- >it receives the object and call hashCode()
- $10.\text{hashCode}()$
- >where to store the data will be decided by hashing function
- >if we remove any bucket ,if any other reference is present inside next then that will be replaced in particular index

### HashSet Usage :

- >if we want to remove duplicates then we go for it
- >if we want to shuffle the elements
- >to remove elements , but not shuffled but in same order then we can by using LinkedHashSet

methods :

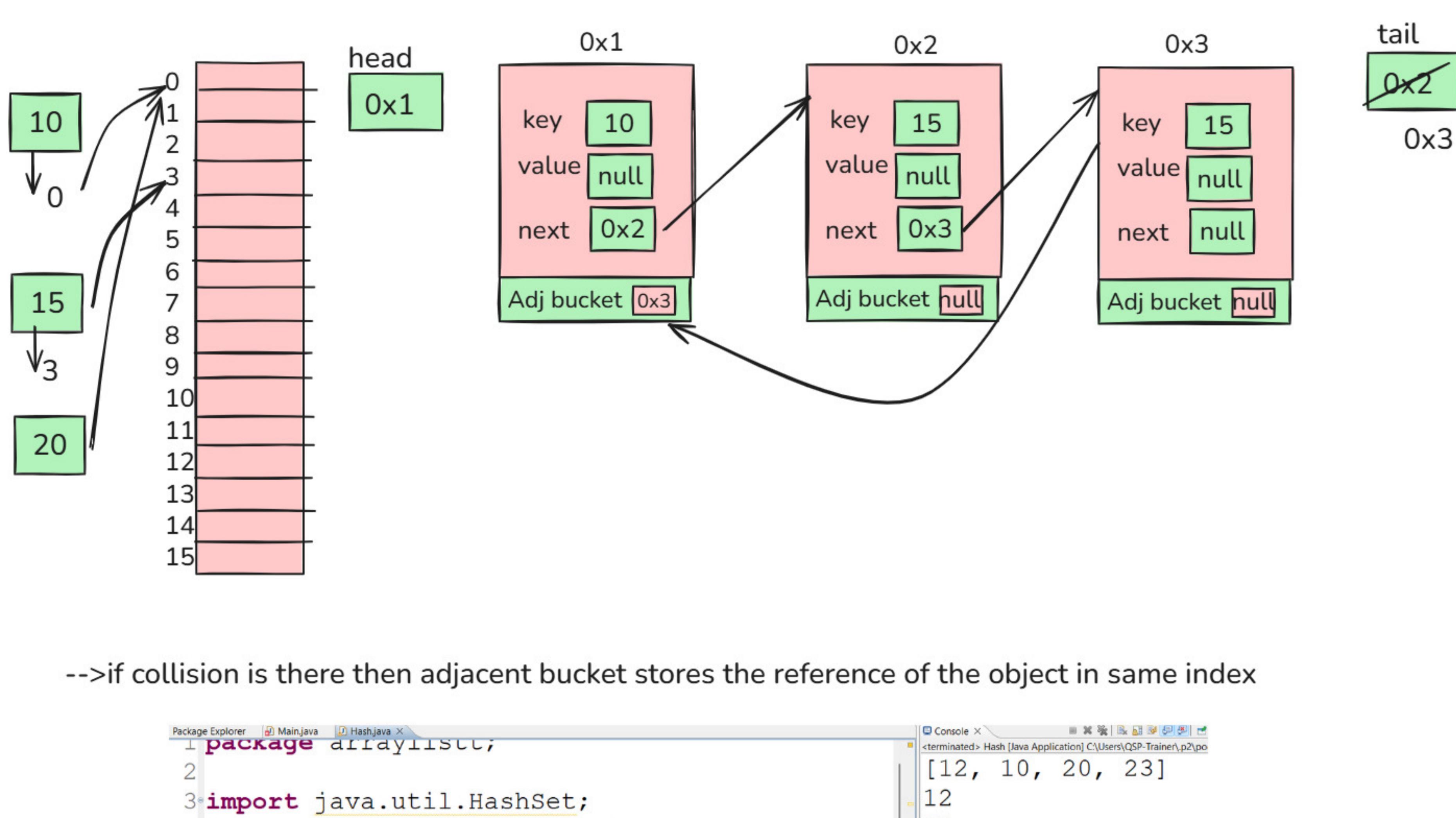
- 1.addFirst()
- 2.addLast()
- 3.getFirst()
- 4.getLast()
- 5.removeFirst()
- 6.removeLast()
- 7.reversed()

### LinkedHashSet :

- >LinkedHashSet will preserve the order of Insertion

head  
Ref of first bucket inserted

tail  
address of bucket inserted



- >if collision is there then adjacent bucket stores the reference of the object in same index

```

package arrays;
import java.util.HashSet;
import java.util.LinkedHashSet;
class Hash {
    public static void main(String[] args) {
        LinkedHashSet h=new LinkedHashSet();
        h.add(10);
        h.add(20);
        h.addFirst(12);
        h.addLast(23);
        System.out.println(h);
        System.out.println(h.getFirst());
        System.out.println(h.getLast());
        System.out.println(h.removeFirst());
        System.out.println(h.removeLast());
        System.out.println(h.reversed());
    }
}

```

```

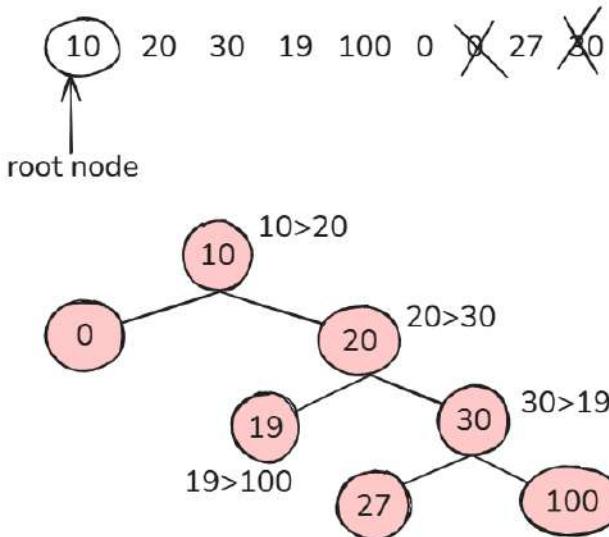
[12, 10, 20, 23]
12
23
12
23
[20, 10]

```

### TreeSet :

- >it internally implements Tree Data Structure
- >it is also known as Self Balanced Binary Tree

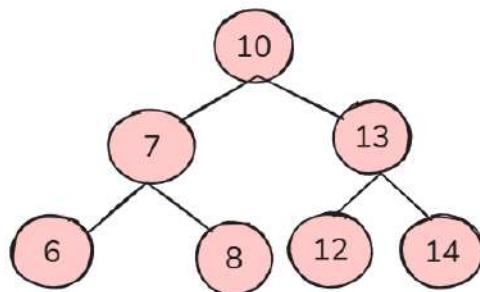
### Binary Tree :



- >no duplicates
- >while printing Left Parent Right

### Characteristics of TreeSet :

- >No Heterogeneous values are allowed
- >we cannot insert null values
  - if we try to insert null values(Null Pointer Exception)
- >no indexing order is there
- >sorting elements is less time (more efficient )than any other Collection object
- >it removes duplicates while sorting/inserting
- >for searching elements it is not good
- >if Left Side node = right Side node then it is Self Balanced Binary Tree

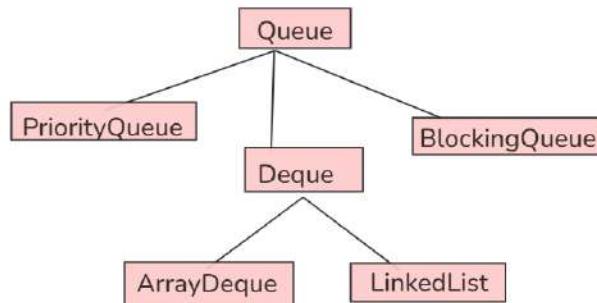


```

public class Example {
public static void main(String[] args) {
    TreeSet t=new TreeSet();
    t.add(10);
    t.add(30);
    t.add(5);
    t.add(30);
    t.add(45);
    t.add(11);
    System.out.println(t); // [5, 10, 11, 30, 45]
}
}
  
```

## Queue

-->it is an interface present inside java.util package  
-->it follows FIFO (First In First Out)



### extra methods in Queue :

offer( ) - inserts element in the Queue  
poll( ) - removes element on the top of the Queue  
element( ) - it retrieves element on the top of the Queue  
throws exception if Queue is empty  
peek( ) - it retrieves top element of the Queue and  
returns null if there Queue is empty

### PriorityQueue :

-->here elements are based on priority  
-->no FIFO here  
-->Default priority is given to minimum element inside the Queue  
Characteristics :

-->no null elements are allowed  
-->order depends on natural ordering or comparator  
-->not synchronized

### Ex : PriorityQueue Natural Ordering - Min Heap

```
1 import java.util.PriorityQueue;
2 public class PriorityQueueExample {
3     public static void main(String[] args) {
4         PriorityQueue<Integer> pq = new PriorityQueue<>();
5         pq.add(40);
6         pq.add(10);
7         pq.add(30);
8         pq.add(20);
9         System.out.println("Priority Queue elements:");
10        while (!pq.isEmpty()) {
11            System.out.println(pq.poll());
12        }
13    }
14 }
```

Output:  
Priority Queue elements:  
10  
20  
30  
40

-->here smallest element has highest priority  
-->poll( ) removes the elements in priority order

### Ex : PriorityQueue Custom Priority(Max Heap)

```
3 import java.util.Collections;
4 import java.util.PriorityQueue;
5 public class PriorityQueueExample {
6     public static void main(String[] args) {
7         PriorityQueue<Integer> pq =
8             new PriorityQueue<>(Collections.reverseOrder());
9         pq.add(40);
10        pq.add(10);
11        pq.add(30);
12        pq.add(20);
13
14        System.out.println("Max Priority Queue:");
15        while (!pq.isEmpty()) {
16            System.out.println(pq.poll());
17        }
18    }
19 }
```

Output:  
Max Priority Queue:  
40  
30  
20  
10

-->here Collections.reverseOrder( ) changes the priority  
-->it behaves as Max Heap here

### Ex : PriorityQueue with Objects

```

import java.util.PriorityQueue;

class Student implements Comparable<Student> {
    int id;
    String name;

    Student(int id, String name) {
        this.id = id;
        this.name = name;
    }

    public int compareTo(Student s) {
        return this.id - s.id; // priority based on id
    }
}

```

```

public class PriorityQueueObject {
    public static void main(String[] args) {
        PriorityQueue<Student> pq = new PriorityQueue<>();
        pq.add(new Student(3, "Hari"));
        pq.add(new Student(1, "Kavya"));
        pq.add(new Student(2, "Dinga"));
        while (!pq.isEmpty()) {
            Student s = pq.poll();
            System.out.println(s.id + " " + s.name);
        }
    }
}

```

output
1 Kavya
2 Dinga
3 Hari

### Deque :

- >Double Ended Queue
- >here Insertion and removal from both ends

### ArrayDeque :

-->ArrayDeque is a resizable-array implementation of the Deque interface in Java that allows insertion and removal of elements from both ends.

- >it is present inside java.util package
- >it implements Deque
- >it allows both FIFO and LIFO operations
- >no null elements are allowed
- >it is faster than Stack and LinkedList
- >it is not thread safe

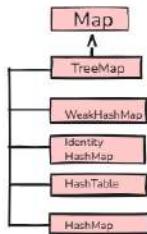
### BlockingQueue :

-->BlockingQueue is a synchronized queue used in multithreaded environments where threads wait when the queue is full or empty.

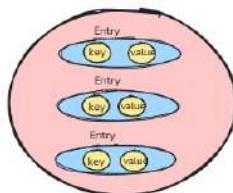
- >it is used in multi threading
- >Thread waits if queue is Full(while inserting)
- Empty(while removing)

## Map

- >Map is an interface in java
- >it is present inside java.util package
- >it stores data in form of key-value pairs



- >data will be stored in form of Entries
- >in Entry key and values will be there



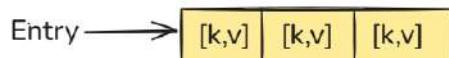
- >when we want group of key-value pairs, we go for Map

### Characteristics of Maps :

- >the data should be stored in key-value pair(entry)
- >only unique keys are allowed
- >only one key is allowed
- >we can have multiple null values
- >no order of insertion
- >no indexing
- >we can store both heterogeneous & homogeneous values , keys and values

### methods :

1.put(key,value)



- 2.keySet ( ) - to fetch only keys
- 3.values( ) - it returns collection of values
- 4.replace(key k,value v)
- 5.replace(key k,old value,new value)
- 6.containsKey(Object key)
- 7.containsValue(Object value)

### HashMap :

- > it is unordered and it is hash-based implementation
- >it allows one null key and multiple null values
- >it is not synchronized
- >it internally uses hash table
- >default capacity is 15 and the load factor is 0.75

```
3 import java.util.HashMap;
4
5 class Hash {
6     public static void main(String[] args) {
7         HashMap h=new HashMap();
8         h.put(10, 2);
9         h.put(12, 5);
10        h.put("name","kavya");
11        h.put(10, 3);
12        System.out.println(h);
13        System.out.println(h.keySet());
14        System.out.println(h.values());
15
16    }
17 }
```

[name=kavya, 10=3, 12=5]  
[name, 10, 12]  
[kavya, 3, 5]

### LinkedHashMap :

- >it maintains insertion order
- >only null key allowed , multiple null values are allowed
- >slightly slower than HashMap
- >we can use this when order matters

```
import java.util.LinkedHashMap;

public class LinkedHashMapDemo {
    public static void main(String[] args) {
        LinkedHashMap<Integer, String> map = new LinkedHashMap<>();
        map.put(3, "C");
        map.put(1, "A");
        map.put(2, "B");
        System.out.println("LinkedHashMap maintains insertion order: " + map);
    }
}
```

output :

LinkedHashMap maintains insertion order: {3=C, 1=A, 2=B}

### TreeMap :

- >it stores keys in sorted order
- >no null keys are allowed
- >internally it uses Red-Black Tree

```
TreeMap<Integer, String> map = new TreeMap<>();
map.put(3, "C");
map.put(1, "A");
map.put(2, "B");
System.out.println(map); // {1=A, 2=B, 3=C}
```

```
import java.util.TreeMap;
public class TreeMapDemo {
    public static void main(String[] args) {
        TreeMap<Integer, String> map = new TreeMap<>();

        map.put(5, "E");
        map.put(2, "B");
        map.put(8, "H");
        System.out.println("TreeMap sorted keys: " + map);
    }
}
```

output :

TreeMap sorted keys: {2=B, 5=E, 8=H}

### TreeMap with Custom Comparator - Reverse Sorting :

```
import java.util.TreeMap;
import java.util.Comparator;
public class TreeMapCustomComparator {
    public static void main(String[] args) {
        TreeMap<Integer, String> map = new TreeMap<>(Comparator.reverseOrder());

        map.put(1, "A");
        map.put(4, "D");
        map.put(2, "B");

        System.out.println("TreeMap with reverse order: " + map);
    }
}
```

output :

TreeMap with reverse order: {4=D, 2=B, 1=A}

### Hashtable :

- >it is synchronized
- >it is thread safe
- >it does not allow null key or value
- >it is slower than HashMap