**Collection**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Q. What is a Collection?**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Collection is a ready made implementation of data structure

**Q.What is the purpose of Collection?**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. Ability to store any kind of data.
2. We can extends its size at run time as well as shrink its size at run time

Means we can store infinite data in collection

1. Provide  ready made implementation of data structure

**Note:** if we think about array then we cannot modify its size at run time

Or we need to write manual logic to extend the array size.

When we use array we need to manual logic of data structure like as sorting ,searching etc

**Note:** when we create array of Object class then we can store any kind of data in array

package org.techhub;

public class TestcollApplication {

public static void main(String[] args) {

Object obj [] = new Object[5];

obj[0]=100;

obj[1]=false;

obj[2]=5.4f;

obj[3]=new java.util.Date();

obj[4]="good";

for(int i=0; i<obj.length;i++) {

System.out.println(obj[i]);

}

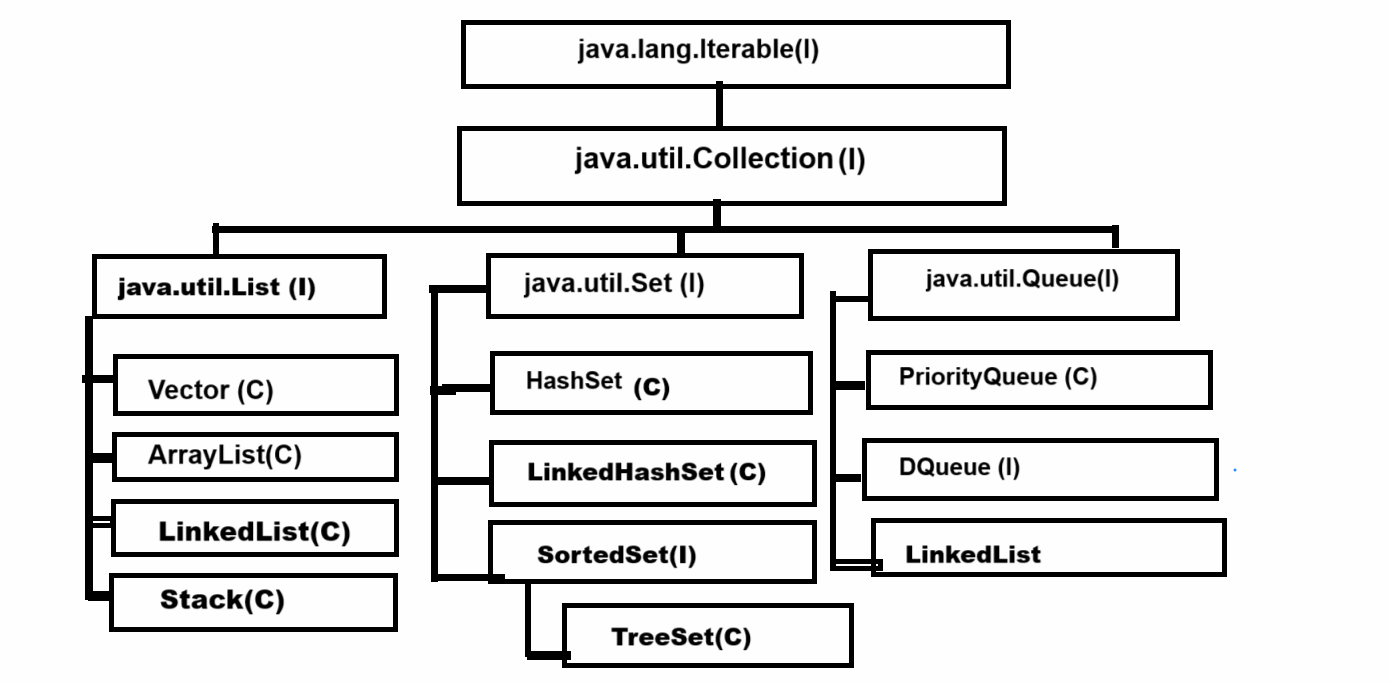
}

}

If we think about above code we have collection of different types of data using Object class array but there is limitation when we want to any data structure operation on above code we need to write manually like as searching, insertion of element , deletion of element etc

So Java Suggest we provide ready implementation of data structure to developer known as collection

If we want to work with Collection in JAVA we need to know the following classes and interface hierarchy

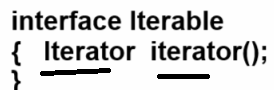


If we think about above diagram we have top most interface name as java.lang.Iterable

**Q. Why is java.lang.Iterable the parent of all Collection?**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

The major goal of Iterable interface is to fetch data from collection and data fetching is common operation in all collections so Iterable is parent of all collection because Iterable interface contain iterator() method which return reference of Iterator interface and using Iterator interface we can fetch data from collection



iterator() method return reference of Iterator  interface and Iterator interface contain some inbuilt methods which is used for fetch data from collection like as

**boolean hasNext() :** this method check data present in collection or not if present return true otherwise return false

**Object next() :** this method can fetch data from collection and move cursor on next element

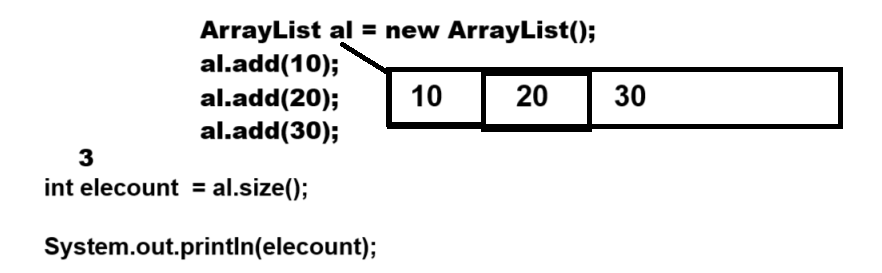
**void remove() :** this method can remove  data from collection using iterator

**Collection interface**

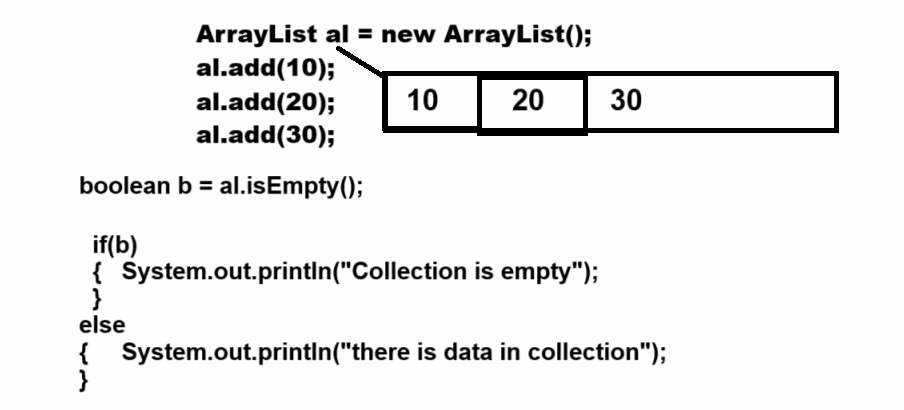
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Collection interface contain common methods which is required for perform daily operation on collection like as adding element, removing element ,checking size etc

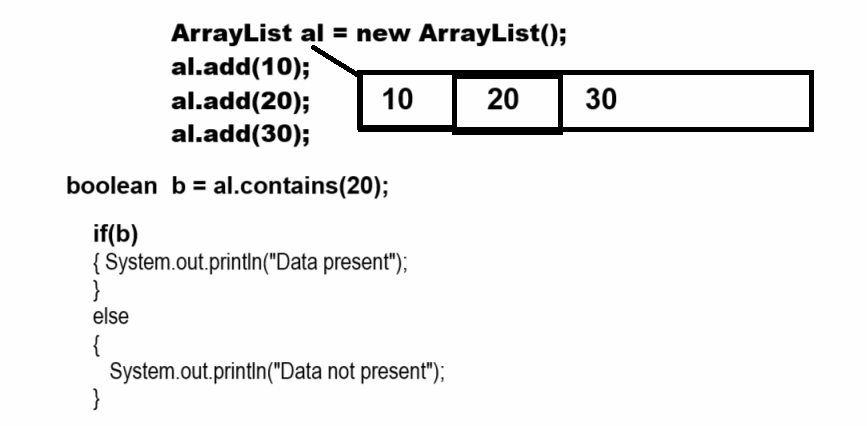
**public abstract int size():** this method is used for return size of collection  or return number of element present in collection



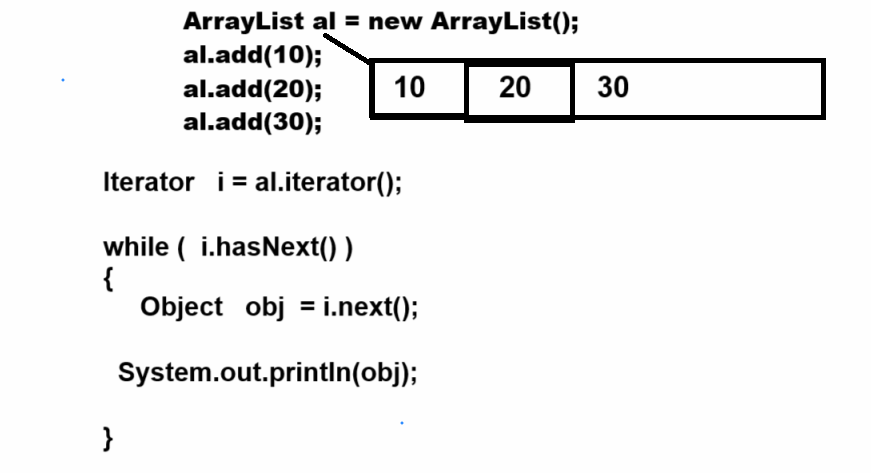
**public abstract boolean isEmpty():** this method checks if the collection is empty or not if empty return true otherwise return false.



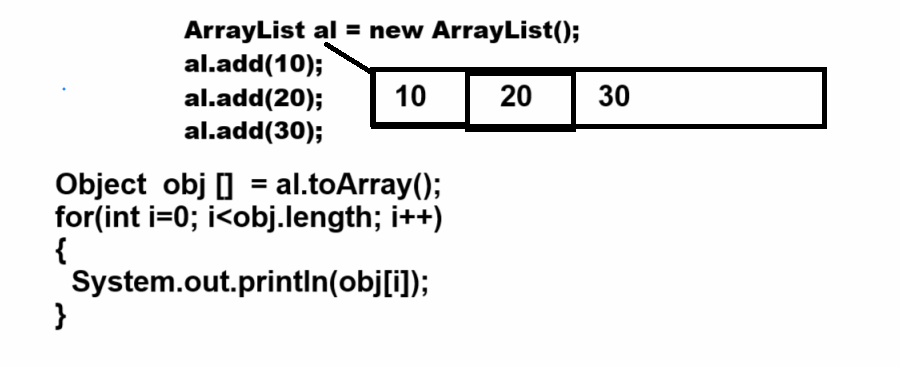
**public abstract boolean contains(java.lang.Object):** this method is used for checking data present in collection or not means normally we use this method for data searching purpose.

****

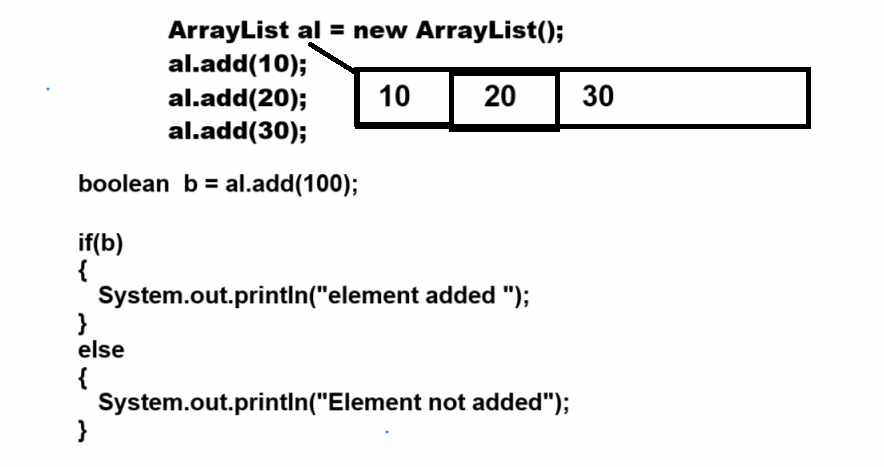
**public abstract java.util.Iterator<E> iterator():** it helps us to fetch data from collection and it is a universal iterator.



**public abstract java.lang.Object[] toArray():** this method can convert any collection in object class array.



**public abstract boolean add(E) :** this method can add a new element in collection and return true if element added otherwise return false.



**public abstract boolean remove(java.lang.Object):**  remove element from collection and element removed successfully return true otherwise return false.

  public abstract boolean containsAll(java.util.Collection<?>):



**public abstract boolean addAll(java.util.Collection<? extends E>):** this method helps us to add more than one element in collection at time.

**public abstract boolean removeAll(java.util.Collection<?>):** this method also helps us to remove more than one element at a time from collection.

**There are three types of collection we have**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**List :** List collection can store duplicate values or data and manage data by using indexing technique means we can say list collection is implementation of linear data structure internally.

**Set :** Set collection cannot allow duplicated data as well as can generate random data also provide data in sorted format using TreeSet collection etc

Means set collection is implementation of non linear and hash based data structure

**Queue :** Queue in data structure allow to store duplicate values and can manage data using first in first out format and provide indexing to data management etc

**Now we want to discuss about the List Collection**

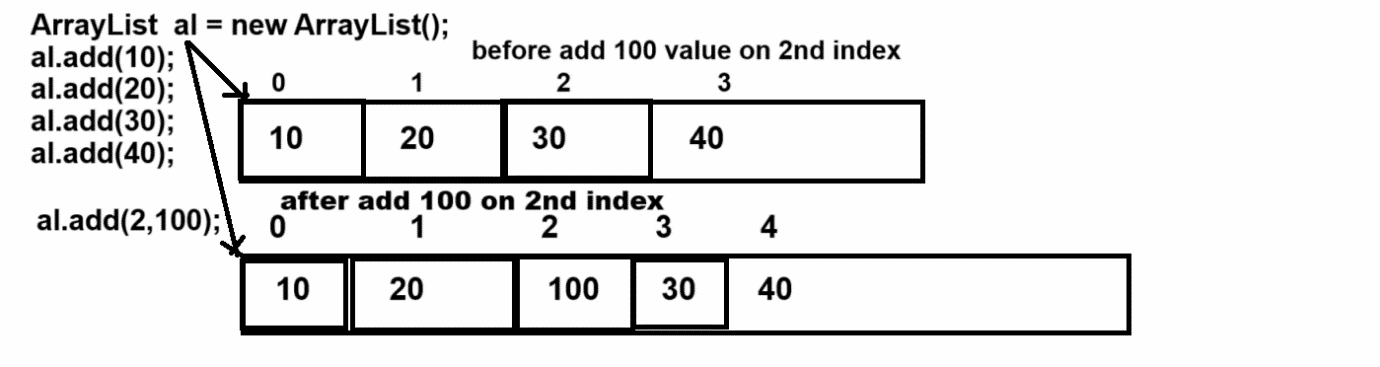
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Methods of List Collection**

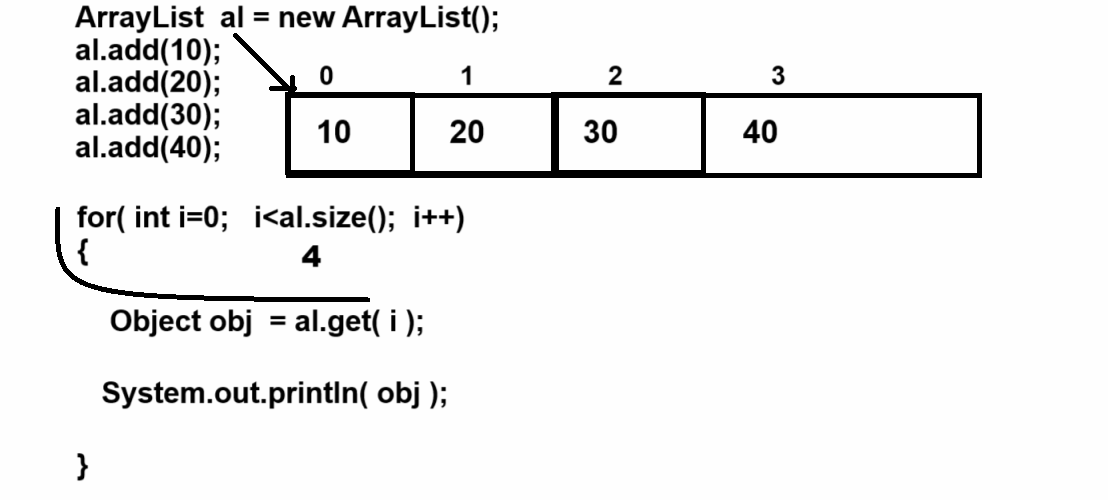
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

List Collection contains all methods from Collection interface as well as List contains its own additional method also.

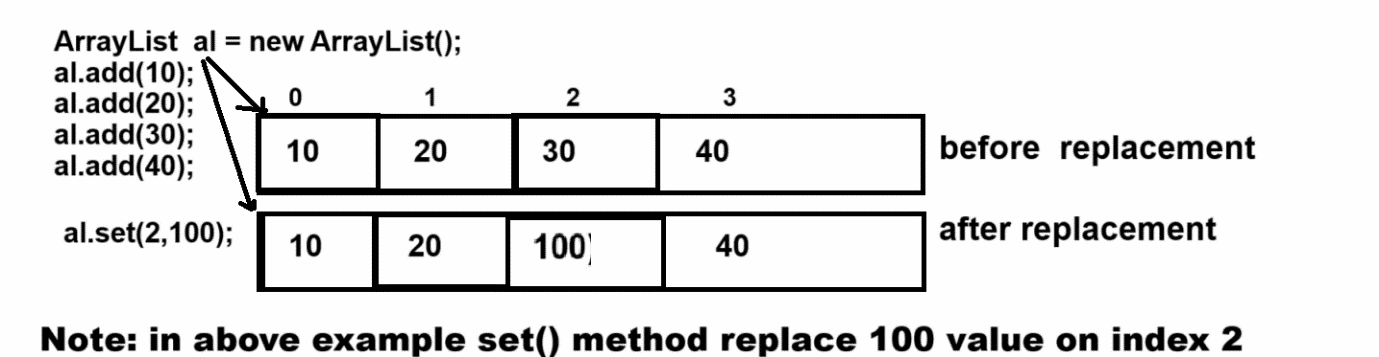
**public abstract void add(int, E):** this method is used for adding a new element on a specified index and moving previous values from that index onto the next index automatically.



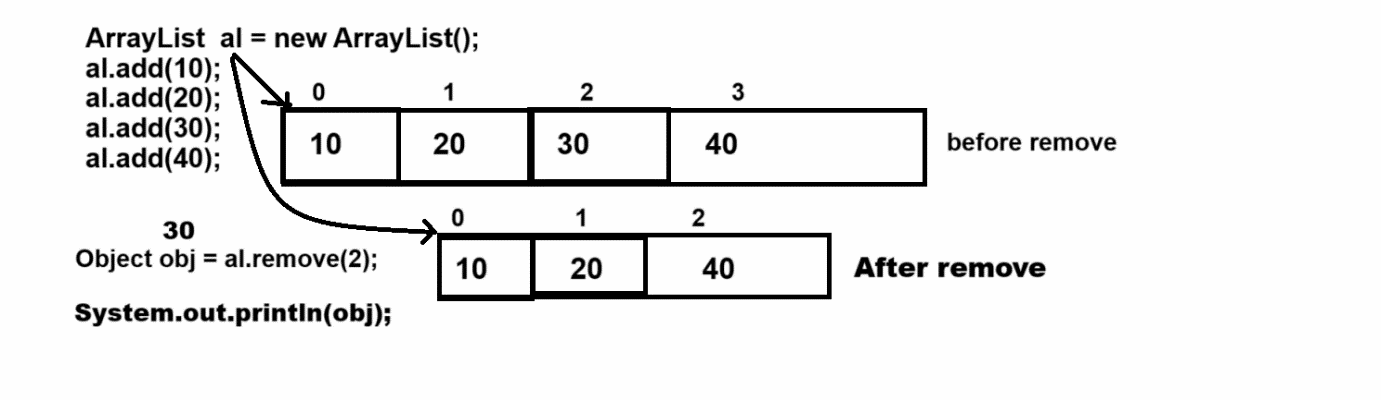
**public abstract E get(int):** this method can return data from ArrayList collection using its index

****

**public abstract E set(int, E):** this method is used to replace data on specified index.



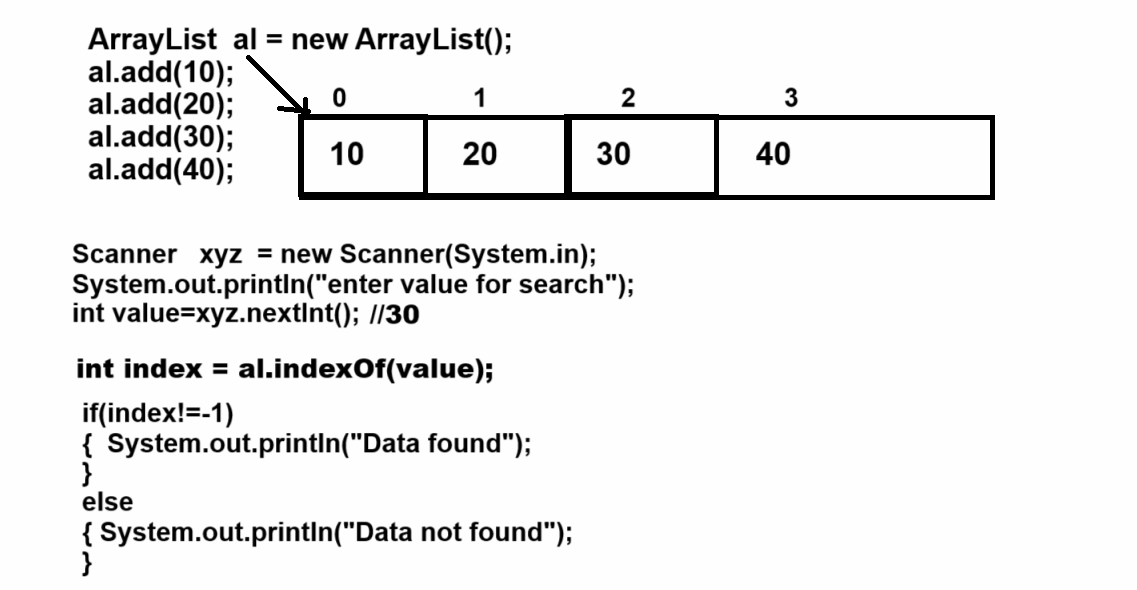
**public abstract E remove(int):** this method is used for remove data using its index and return removed value as output



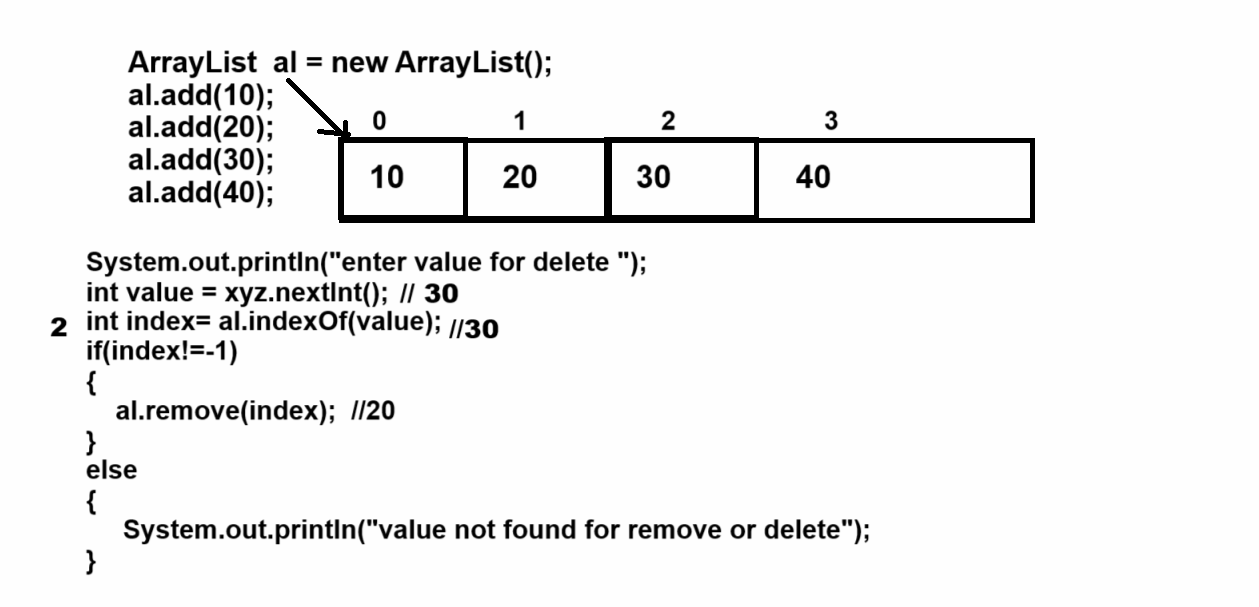
**public abstract int indexOf(java.lang.Object):** this method can return index of particular element if element or value not found in collection return -1

Note: this method can use for two purpose

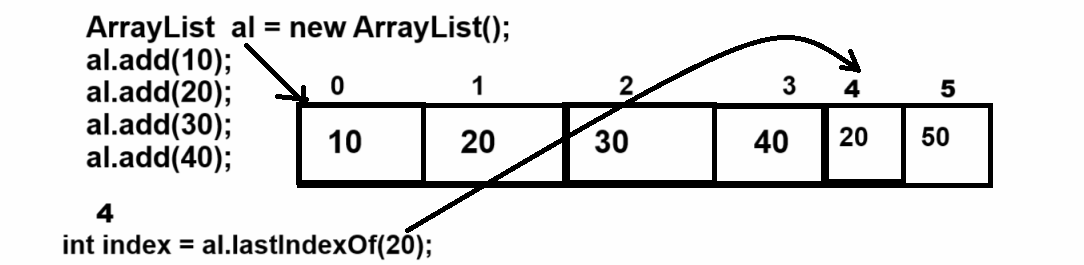
1. **Searching element from collection**



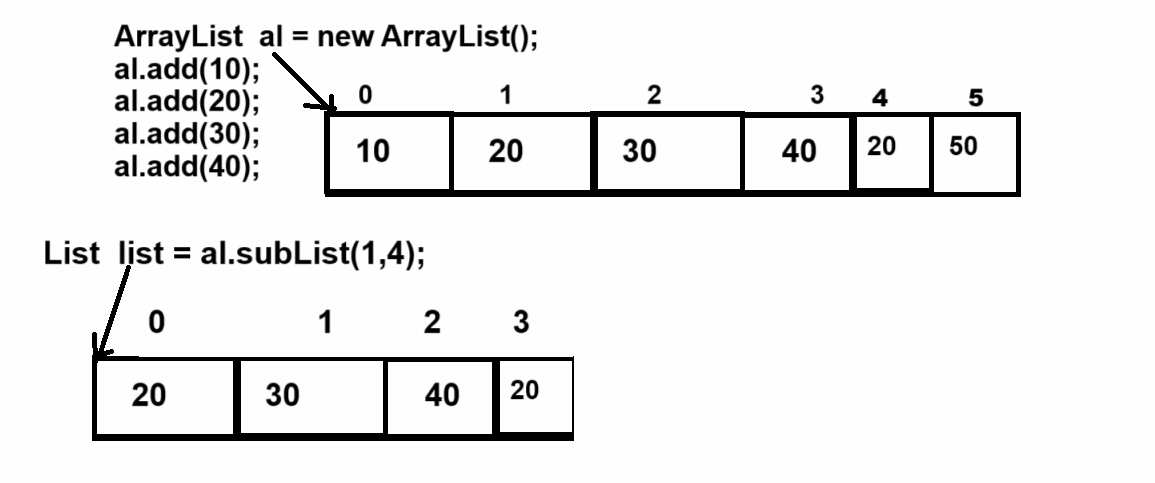
1. **Before call remove method for avoid IndexOutOfBoundsException**



**public abstract int lastIndexOf(java.lang.Object):** this method return the last occurrence of index



**public abstract java.util.List<E> subList(int, int):** this method helps us extract the some specified portion from List collection between two indexes.



**Vector class**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Vector is a dynamic array internally and it is known legacy collection

**Q. What is a legacy collection?**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 Legacy collection means a classes which is  part of previous version of JAVA before collection but later they are added as part of collection called as legacy collection

Vector was added as part of the collection in JDK 1.2 before that vector known as dynamic array in JAVA.

**Important points related with vector**

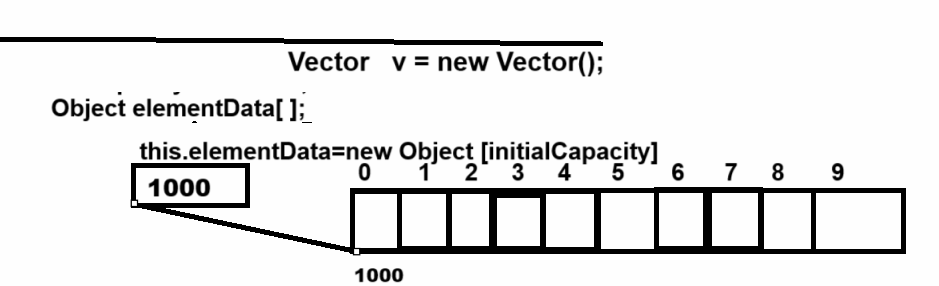
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Vector is thread safe collection
2. Vector legacy collection
3. Default capacity of Vector is 10
4. Vector allocate memory double thant its current capacity so the threshold of Vector is 1.0 or 100%

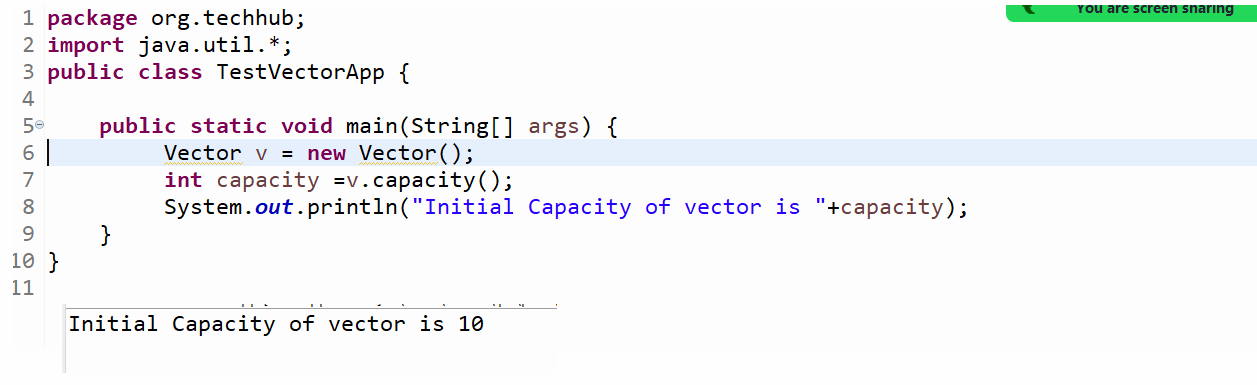
If we want to work with Vector we have some constructor provided by Vector to us

1. **Vector():** When we use this constructor creating Vector object then internally we get array of Object class with initial capacity 10

Means default capacity of Vector is 10.

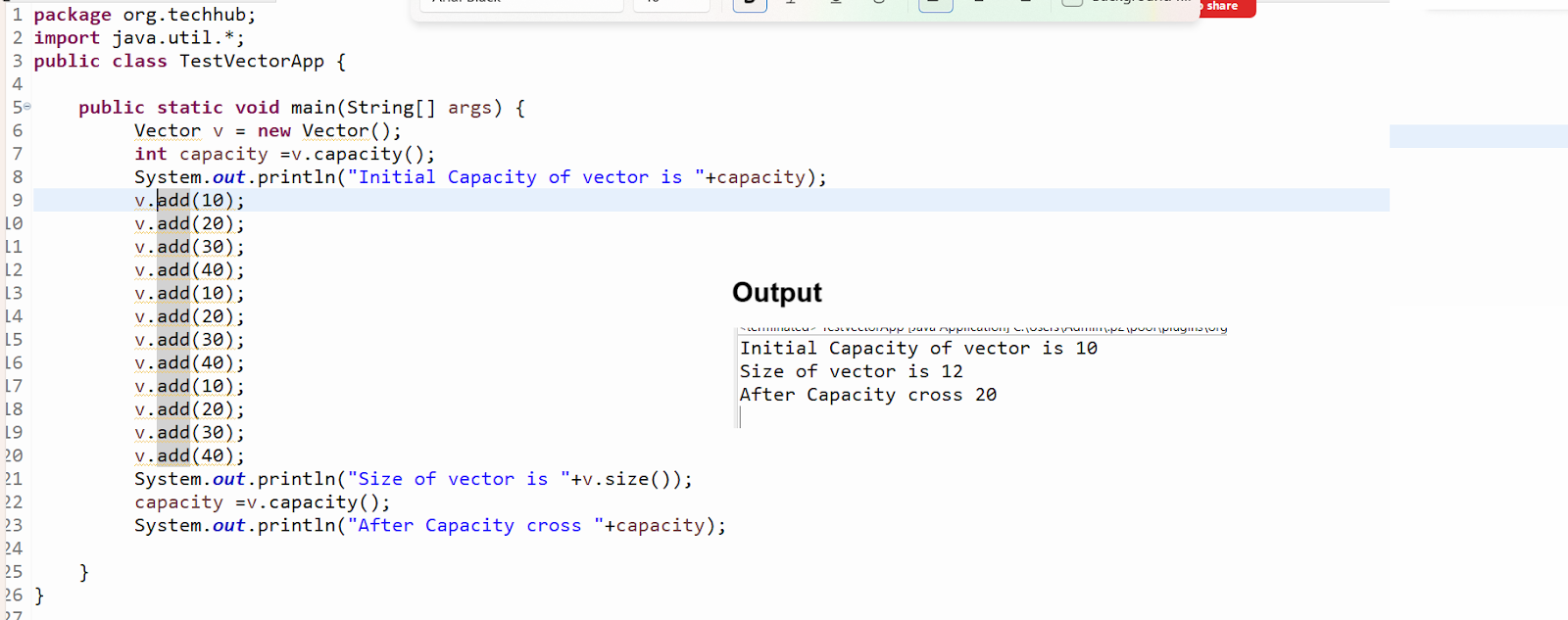
****

If we want to cross verify the default capacity of a vector is 10 then we have a method name as int capacity() using this method we can check the capacity of the vector.



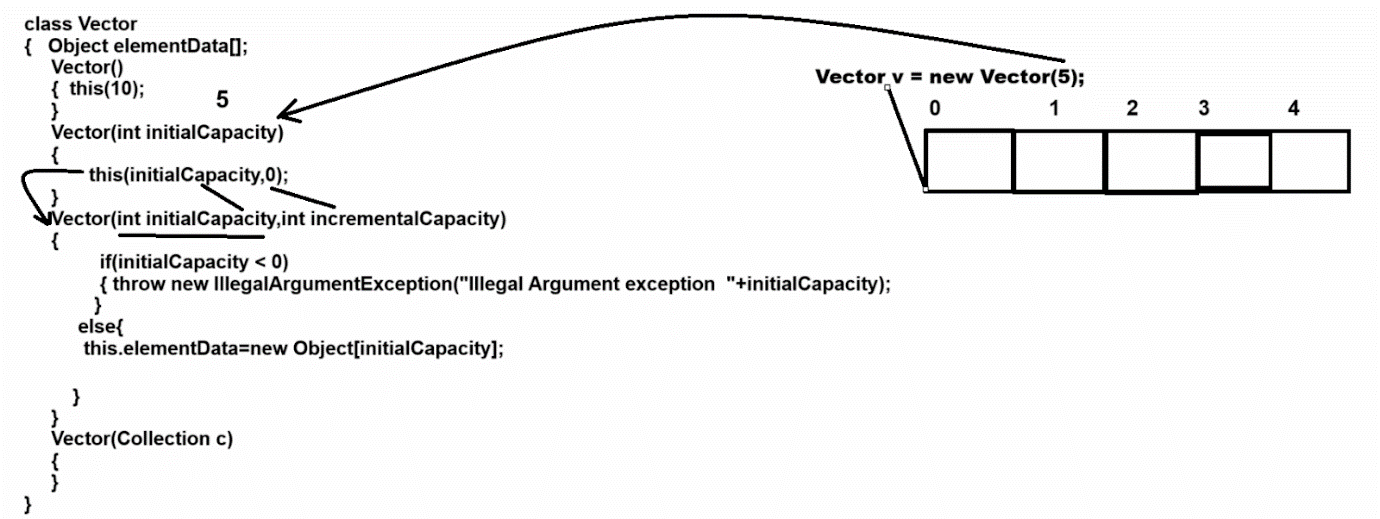
**Important point:** when we store the value more than the current capacity of a vector then the vector occupies double memory than its current capacity.

Example: if we create an object of Vector using default constructor then the default capacity of Vector is 10 and if we try to add the 11th element in vector then its new capacity is 20 and if we try to add 21th element then the new capacity is 40 and so on.



**Note:** user can set initial capacity of vector according to his requirement.

1. **Vector(int initialCapacity) :** this constructor can create a vector with initial capacity as per the user choice**.**

****

**Note:** if we set initial capacity of vector according to user choice and if vector cross its current capacity then vector allocate memory double than its current capacity

**Note:** if we want to decide the incremental capacity of Vector according to user choice not by default double then we have a constructor with two parameters.

1. **Vector(int initialCapacity,int incrementalCapacity)**

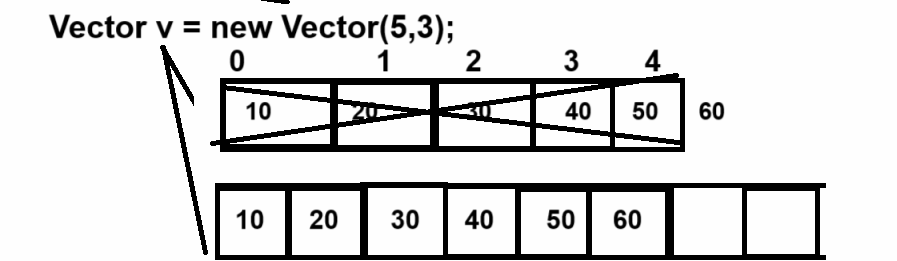
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Parameter Details**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**int initialCapacity:** this parameter is used for set initial capacity of vector.

**int incrementalCapacity:** this parameter is used for decide the incremental capacity of vector

****

**Note:** if we think about above constructor we create vector object using  Vector v=new Vector(5,3) here we set initial capacity 5 and we set incremental capacity 3 means when we try to add 6th element in vector then vector increase its capacity by 3 blocks only

1. **Vector(Collection) :** this constructor helps us to copy data from another collection and store it in a Vector object.

package org.techhub;

import java.util.\*;

public class TestVectorApplication {

public static void main(String[] args) {

ArrayList al = new ArrayList();

al.add(10);

al.add(20);

al.add(30);

al.add(40);

Vector v = new Vector(al);

System.out.println(v);

}

}

**Example by using Vector**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Case 1: Add New Element in Vector

Case 2: View All Elements

Case 3: Count number of element

Case 4: Search element by contains

Case 5: Search Element by index

Case 6: delete element by using index

Case 7: Fetch elements using get() method

Case 8: SubList

Case 9: Remove by element

**Example with source code**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

package org.techhub;

import java.util.\*;

public class TestVectorApplication {

public static void main(String[] args) {

Vector v = new Vector();

do {

Scanner xyz = new Scanner(System.in);

System.out.println("1: Add Element");

System.out.println("2: View All  ");

System.out.println("3: Count number of element ");

System.out.println("4:Search element by contains method");

System.out.println("5:Search Element by index");

System.out.println("6:Delete by using its index");

System.out.println("7:Fetch elements by using get method");

System.out.println("8: SubList");

System.out.println("9: Remove element by value");

System.out.println("Enter your choice");

int choice = xyz.nextInt();

switch (choice) {

case 1:

System.out.println("Enter data in vector");

int value = xyz.nextInt();

boolean b = v.add(value);

if (b) {

System.out.println("element added");

} else {

System.out.println("element not added");

}

break;

case 2:

Iterator i = v.iterator();

while (i.hasNext()) {

Object obj = i.next();

System.out.println(obj);

}

break;

case 3:

System.out.println("Number of element in vector " + v.size());

break;

case 4:

System.out.println("Enter value for search ");

value = xyz.nextInt();

b = v.contains(value);

if (b) {

System.out.println("Value found");

} else {

System.out.println("Value not found");

}

break;

case 5:

System.out.println("Enter value for search ");

value = xyz.nextInt();

int index = v.indexOf(value);

if (index != -1) {

System.out.println("Data found");

} else {

System.out.println("Data not found");

}

break;

case 6:

System.out.println("Enter value for delete ");

value = xyz.nextInt();

index = v.indexOf(value);

if (index != -1) {

Object obj = v.remove(index);

System.out.println("Data Deleted " + obj);

} else {

System.out.println("Data not found");

}

break;

case 7:

for (int k = 0; k < v.size(); k++) {

Object obj = v.get(k);

System.out.println(obj);

}

break;

case 8:

System.out.println("enter start index and end index");

int startIndex = xyz.nextInt();

int endIndex = xyz.nextInt();

if (startIndex >= 0 && endIndex < v.size()) {

List list = v.subList(startIndex, endIndex);

for (Object obj : list) {

System.out.print(obj + "\t");

}

}

break;

case 9:

System.out.println("Enter value for delete ");

value = xyz.nextInt();

v.remove((Object)value);

break;

case 10:

System.exit(0);

break;

default:

System.out.println("Wrong choice");

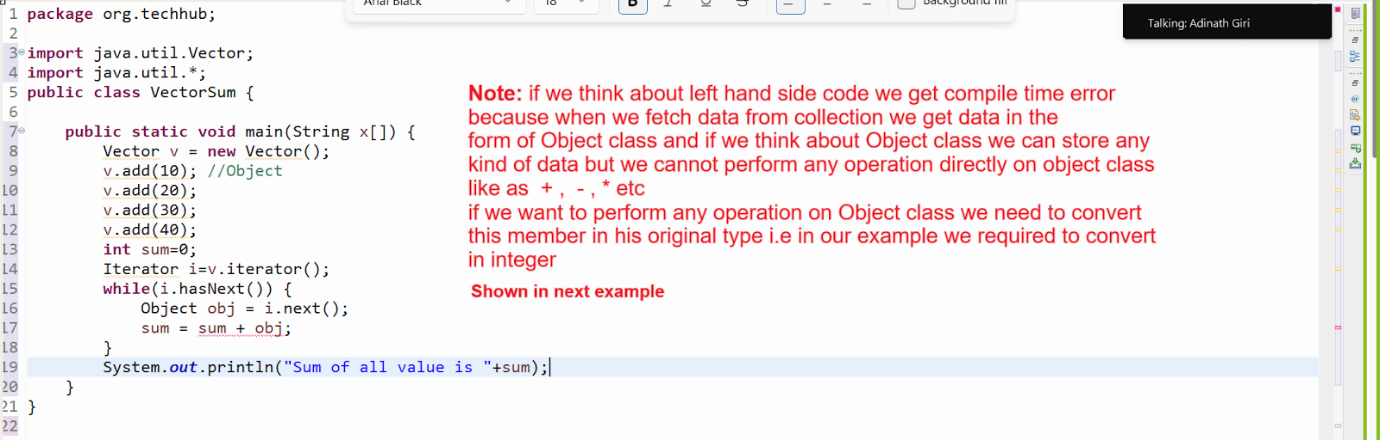
}

} while (true);

}

}

**Example:** WAP to create vector and store 5 values init and calculate its sum



package org.techhub;

import java.util.Vector;

import java.util.\*;

public class VectorSum {

public static void main(String x[]) {

Vector v = new Vector();

v.add(10); //Object

v.add(20);

v.add(30);

v.add(40);

int sum=0;

Iterator i=v.iterator();

while(i.hasNext()) {

Object obj = i.next();

sum = sum + (int)obj;

}

System.out.println("Sum of all value is "+sum);

}

}

**Example:** WAP to create Vector and store 5 integer values in it and perform sorting operations on it.

package org.techhub;

import java.util.\*;

public class SortVectorApplication {

public static void main(String[] args) {

Vector v = new Vector();

v.add(10);

v.add(2);

v.add(30);

v.add(5);

v.add(25);

System.out.println("Data Before Sorting");

Iterator itr= v.iterator();

while(itr.hasNext()) {

Object obj  = itr.next();

System.out.println(obj);

}

//perform sorting

for(int i=0; i<v.size();i++) {

for(int j=(i+1); j<v.size();j++) {

  Object prev=v.get(i);

  Object next=v.get(j);

  if((int)prev>(int)next) {

  v.set(i, next);

  v.set(j, prev);

  }

}

}

System.out.println("Data After Sorting");

  itr= v.iterator();

while(itr.hasNext()) {

Object obj  = itr.next();

System.out.println(obj);

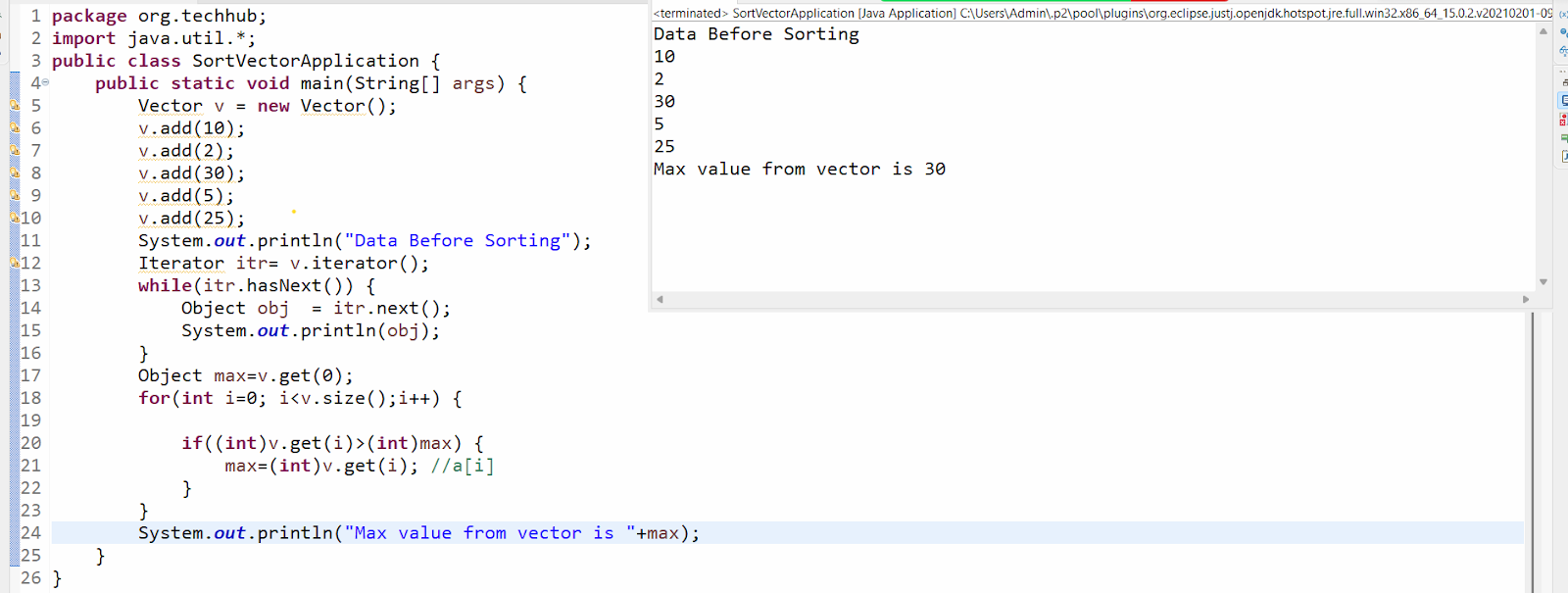
}

}

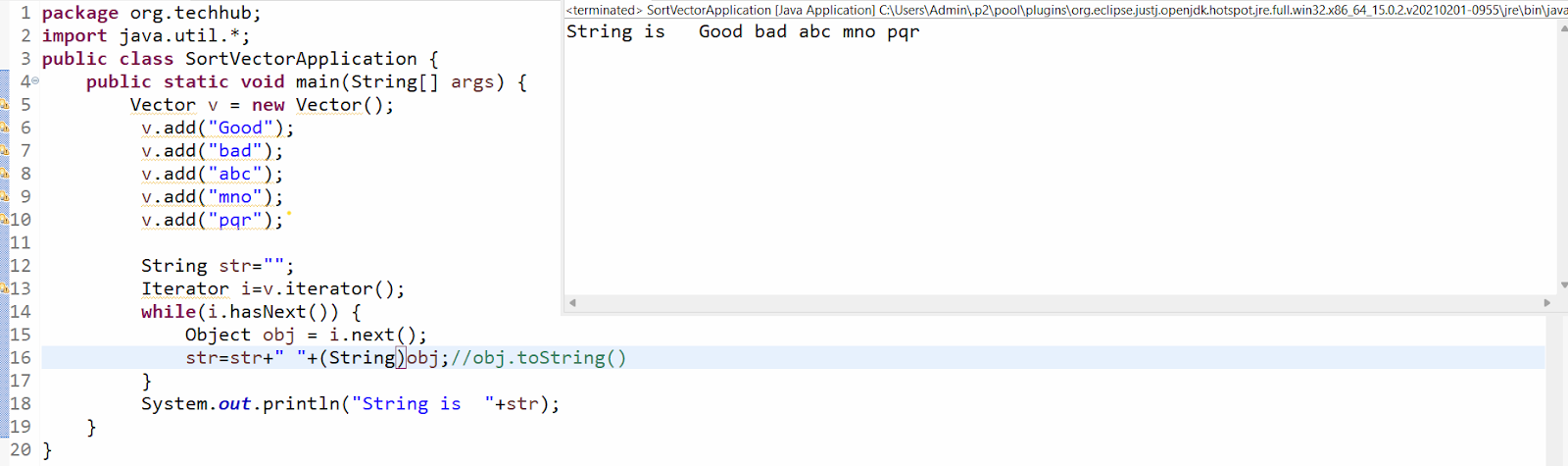
}

**Example:** WAP to create Vector and store 5 values in it and find the max value from Vector without using Collections.max()

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



**Example:** WAP to store five words in Vector and create single string or line of statement using Vector



**How to store user defined objects in Collection**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Q. Why do we need to store user-defined objects in collections?**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

When we want to add more than one element of different type as a single value in a collection then we can store user defined objects in the collection.

**Example:** we want to create a collection of students with data id,name and per here id has integer data type, name has string data type and per has float data type.

If we want to solve above problem then we can create POJO class name as Student with field id, name and per and store data in Student object and store in collection i.e in Vector according to our example and when we want to fetch data from collection then we required convert that data in Student object from Object class.

**Example with source code**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

package org.techhub;

public class Student {

private int id;

private String name;

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public float getPer() {

return per;

}

public void setPer(float per) {

this.per = per;

}

private float per;

}

package org.techhub;

import java.util.\*;

public class SortVectorApplication {

public static void main(String[] args) {

Vector v = new Vector();

  Student s1 = new Student();

  s1.setId(1);

  s1.setName("ABC");

  s1.setPer(90.5f);

  Student s2 = new Student();

  s2.setId(2);

  s2.setName("PQR");

  s2.setPer(70.5f);

  Student s3 = new Student();

  s3.setId(3);

  s3.setName("XYZ");

  s3.setPer(85.5f);

  v.add(s1);

  v.add(s2);

  v.add(s3);

  //Object - generalize format  - Student

  Iterator i=v.iterator();

  while(i.hasNext()) {

  Object obj = i.next();

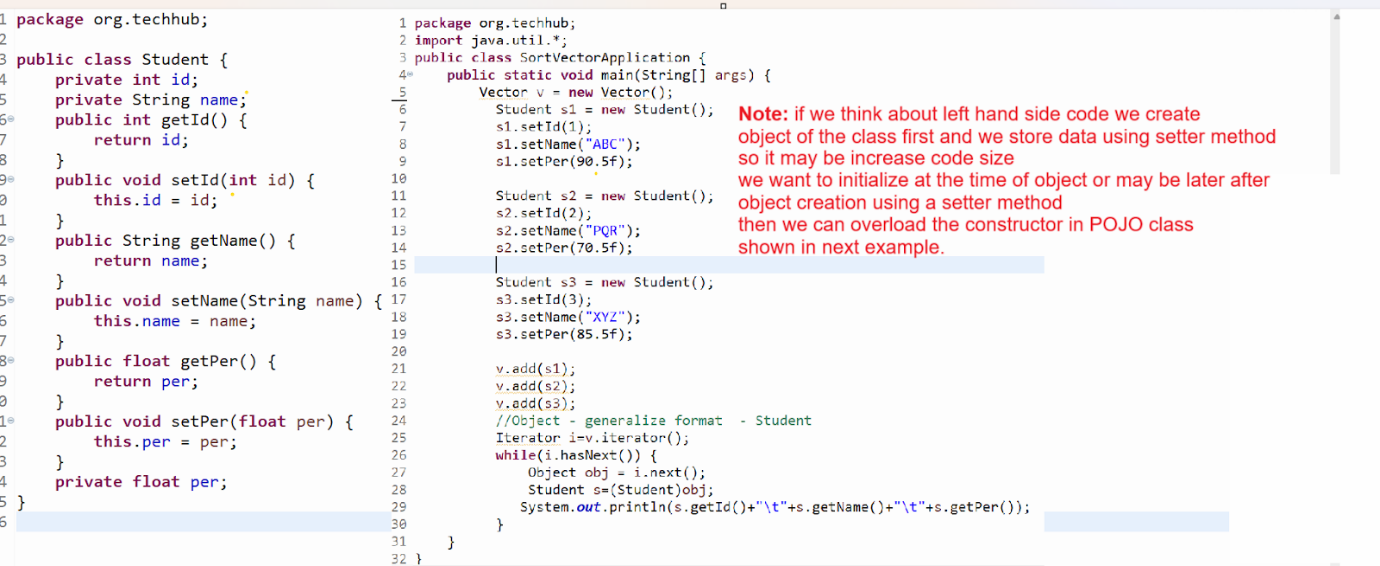
  Student s=(Student)obj;

System.out.println(s.getId()+"\t"+s.getName()+"\t"+s.getPer());

  }

}

}

****

**Example :** Source code with parameterized constructor

package org.techhub;

public class Student {

private int id;

private String name;

public Student() {

}

public Student(String name,int id,float per) {

this.name=name;

this.id=id;

this.per=per;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public float getPer() {

return per;

}

public void setPer(float per) {

this.per = per;

}

private float per;

}

package org.techhub;

import java.util.\*;

public class SortVectorApplication {

public static void main(String[] args) {

Vector v = new Vector();

  Student s1 = new Student("ABC",1,90.5f);

  Student s2 = new Student("PQR",2,70.5f);

  Student s3 = new Student("XYZ",3,85.0f);

  v.add(s1);

  v.add(s2);

  v.add(s3);

  //Object - generalize format  - Student

  Iterator i=v.iterator();

  while(i.hasNext()) {

  Object obj = i.next();

  Student s=(Student)obj;

System.out.println(s.getId()+"\t"+s.getName()+"\t"+s.getPer());

  }

}

}

Or

package org.techhub;

public class Student {

private int id;

private String name;

public Student() {

}

public Student(String name,int id,float per) {

this.name=name;

this.id=id;

this.per=per;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public float getPer() {

return per;

}

public void setPer(float per) {

this.per = per;

}

private float per;

}

package org.techhub;

import java.util.\*;

public class SortVectorApplication {

public static void main(String[] args) {

Vector v = new Vector();

  v.add(new Student("ABC",1,90.5f));

  v.add(new Student("PQR",2,70.5f));

  v.add(new Student("XYZ",3,85.0f));

  Iterator i=v.iterator();

  while(i.hasNext()) {

  Object obj = i.next();

  Student s=(Student)obj;

System.out.println(s.getId()+"\t"+s.getName()+"\t"+s.getPer());

  }

}

}

Example: WAP to create Vector and store five book objects in vector and search book using id from collection.

package org.techhub;

import java.util.\*;

class Book{

private int id;

private String name;

public Book() {

}

public Book(String name,int id,float price) {

this.name=name;

this.id=id;

this.price=price;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public float getPrice() {

return price;

}

public void setPrice(float price) {

this.price = price;

}

private float price;

}

public class BookApplication {

public static void main(String[] args) {

Vector v = new Vector();

Book b1 = new Book("ABC",1,100);

Book b2 = new Book("MNO",2,200);

Book b3 = new Book("PQR",3,300);

Book b4 = new Book("STV",4,400);

v.add(b1);

v.add(b2);

v.add(b3);

v.add(b4);

System.out.println("Display book data");

Iterator itr=v.iterator();

while(itr.hasNext()) {

Object obj = itr.next();

Book b =(Book)obj;

System.out.println(b.getId()+"\t"+b.getName()+"\t"+b.getPrice());

}

Scanner xyz = new Scanner(System.in);

System.out.println("Enter book id for search");

int bid=xyz.nextInt();

boolean flag=false;

for(int i=0;i<v.size();i++) {

Book b=(Book)v.get(i);

if(b.getId()==bid) {

flag=true;

break;

}

}

if(flag) {

System.out.println("Book found");

}

else {

System.out.println("Book not found");

}

  }

}

**Cursor in Collection**

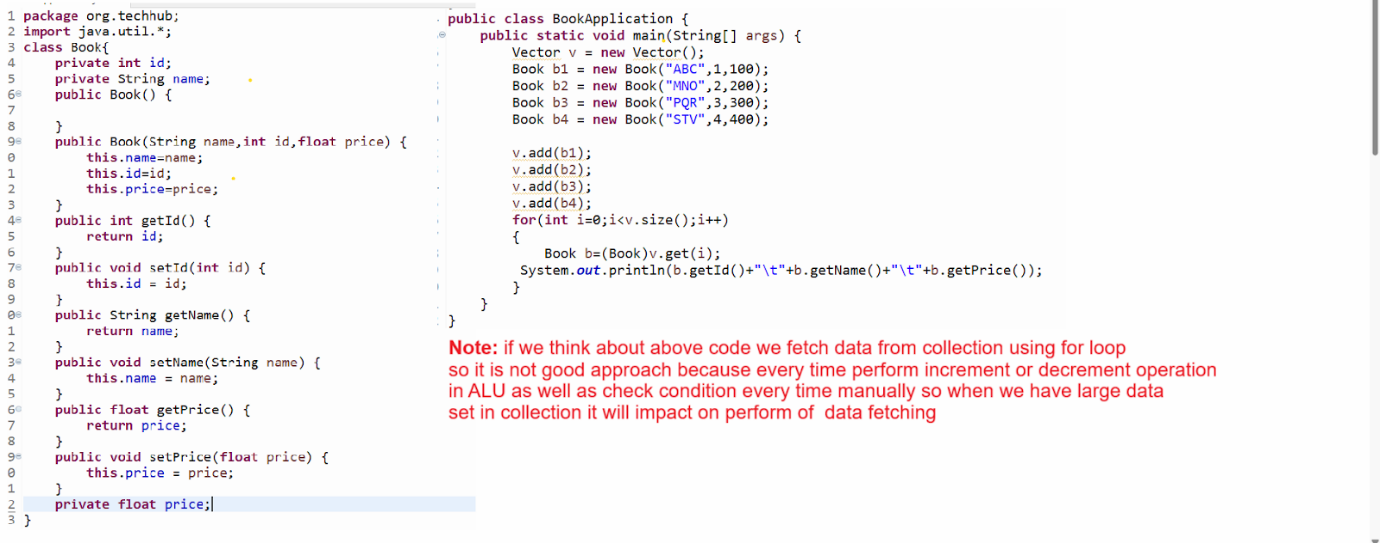
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Cursor are the some  iterators which help us to fetch data from collection

**Types of Cursor in Collection**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **For loop :** but normally we fetch using for loop from collection who maintain the index like as List Collection, Queue collection etc

****

1. **Enumeration :** Enumeration is a cursor in collection which helps us to fetch data from legacy collections only like as Vector etc

If we want to work with Enumeration or create reference of Enumeration we have elements() method of Collection

Syntax: Enumeration ref = collref.elements();

Note: Enumeration is read only cursor means using Enumeration we can fetch data from collection only not perform any other operation on Collection using Enumeration like as removing element or adding element etc

**Methods of Enumeration**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**boolean hasMoreElements():** this method can help us to check data present in collection or not if present return true otherwise return false.

**Object nextElement():** this method can fetch data from collection and move cursor on next element

package org.techhub;

import java.util.\*;

class Book{

private int id;

private String name;

public Book() {

}

public Book(String name,int id,float price) {

this.name=name;

this.id=id;

this.price=price;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public float getPrice() {

return price;

}

public void setPrice(float price) {

this.price = price;

}

private float price;

}

public class BookApplication {

public static void main(String[] args) {

Vector v = new Vector();

Book b1 = new Book("ABC",1,100);

Book b2 = new Book("MNO",2,200);

Book b3 = new Book("PQR",3,300);

Book b4 = new Book("STV",4,400);

v.add(b1);

v.add(b2);

v.add(b3);

v.add(b4);

Enumeration enm = v.elements();

while(enm.hasMoreElements()) {

Object ele=enm.nextElement();

Book b=(Book)ele;

System.out.println(b.getId()+"\t"+b.getName()+"\t"+b.getPrice());

}

  }

}

1. **Iterator :**  Iterator helps us to fetch data from collection and provide some inbuilt methods to fetch data from collection.

**boolean  hasNext():** this method checks if an element is present in collection or not if present return true otherwise return false.

**Object next():** this method fetches data from collection and moves the cursor on the next element.

**void remove():** this method can remove data from collection at the time data traveling or traversing.

**How does Iterator works internally with ArrayList,Vector,Stack and LinkedList?**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

ArrayList,Vector and Stack are the dynamic arrays internally i.e Object [ ] to store elements.

So if we think about Iterator with these three classes

The iterator maintains the index to track the current element position.

next() method return elementData[index++] and hasNext() check index!=size

**Show in following diagram**

****

**Internal working of Iterator for LinkedList**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

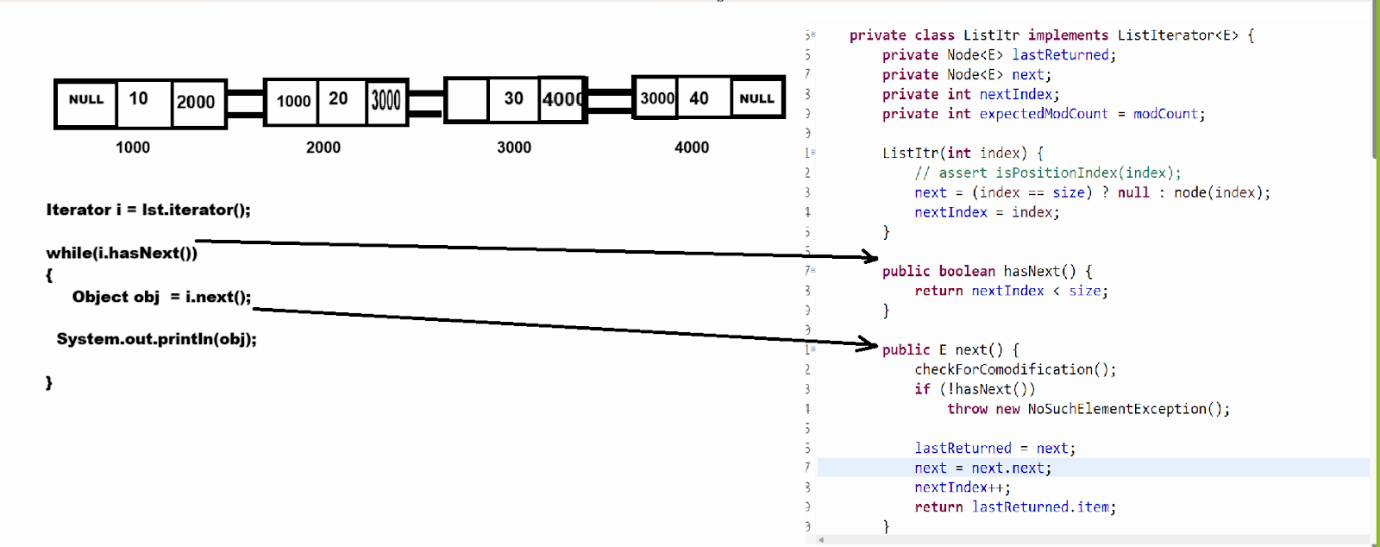
If we think about linked list in Java internally it is doubly LinkedList where each node holds references to its previous and next element  as well as item or data  for that purpose LinkedList has one inner class name as Node which contain three data i.e prev,next, E item

**boolean hasNext():** method check the index<size in linked list for check data availability if index<size return true otherwise return false.

**Object next():** this method can point one pointer on current node i.e lastReturn=next and move cursor on next node using ponter or reference of next node i.e next= next.next and increase index by 1

Index++ for move cursor on next node for comparing with size in hasNext() method and return data using lastReturn.item

Shown in following diagram

****

**If we want to work with iterator we need to know two major important points**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. ModCount concept
2. Fail Fast Concept

**ModCount concept in Java Collection**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

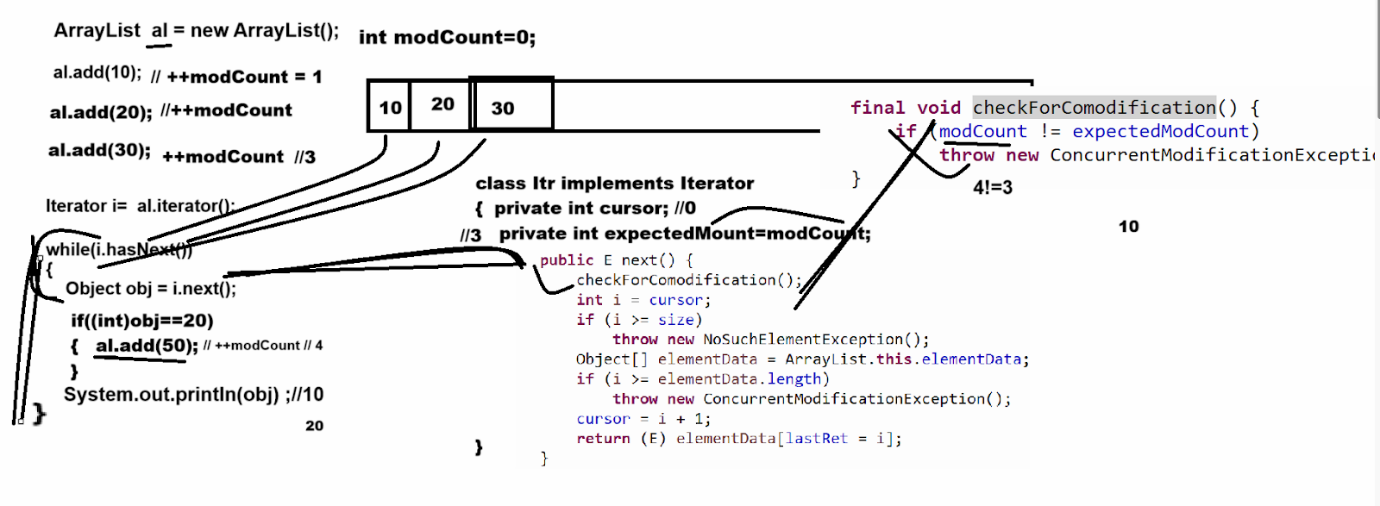
ModCount stands for modification count. It is an integer field to use track number of structural modifications made to collection. It plays a crucial role in ensuring fail-fast behaviour of iterators by detecting concurrent modification duration iteration

ModCount integer field present in collection classes  like as ArrayList, LinkedList , HashMap or Vector etc

It keep track of structural modification to the collection such as

1. Adding elements
2. Removing elements
3. Clearing the collection

Note: structural modification are operations that  alter the collection size or its internal structure



**Example with source**

package org.techhub;

import java.util.Vector;

import java.util.\*;

public class VectorSum {

public static void main(String x[]) {

ArrayList v  = new ArrayList();

  v.add(10);

  v.add(20);

  v.add(30);

  v.add(40);

  Iterator i = v.iterator();

  while(i.hasNext()) {

  Object obj = i.next();

  if((int)obj==30) {

i.remove();

  }

  }

  System.out.println(v);

}

}

1. **ListIterator :** ListIterator is the child interface of Iterator and it is used for travel the collection in forward direction as well as in backward direction and ListIterator only works with List Collection, not other means it is not an universal cursor.

**How to create reference of ListIterator**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**boolean hasNext():** check element present in collection or not in forward direction traveling

**boolean hasPrevious():** this method check element present in collection or not in backward direction

**Object next():  this method can fetch data from collection and move cursor on next element**

**Object previous():** this method can fetch data from collection and move cursor previous element

**void add(Object):** add new element in collection at the time of data adding

         void remove(): this method can remove data from collection using ListIterator

**void set(Object):** this method can replace element in collection using ListIterator

Int previousIndex(): return the index of previous element

Int nextIndex(): return the index of next element

**Example of ListIterator using Vector class**

package org.techhub;

import java.util.Vector;

import java.util.\*;

public class VectorSum {

public static void main(String x[]) {

Vector v  = new Vector();

  v.add(10);

  v.add(20);

  v.add(30);

  v.add(40);

  ListIterator listItr=v.listIterator(v.size());

  while(listItr.hasPrevious()) {

  Object previous =listItr.previous();

  System.out.println(previous);

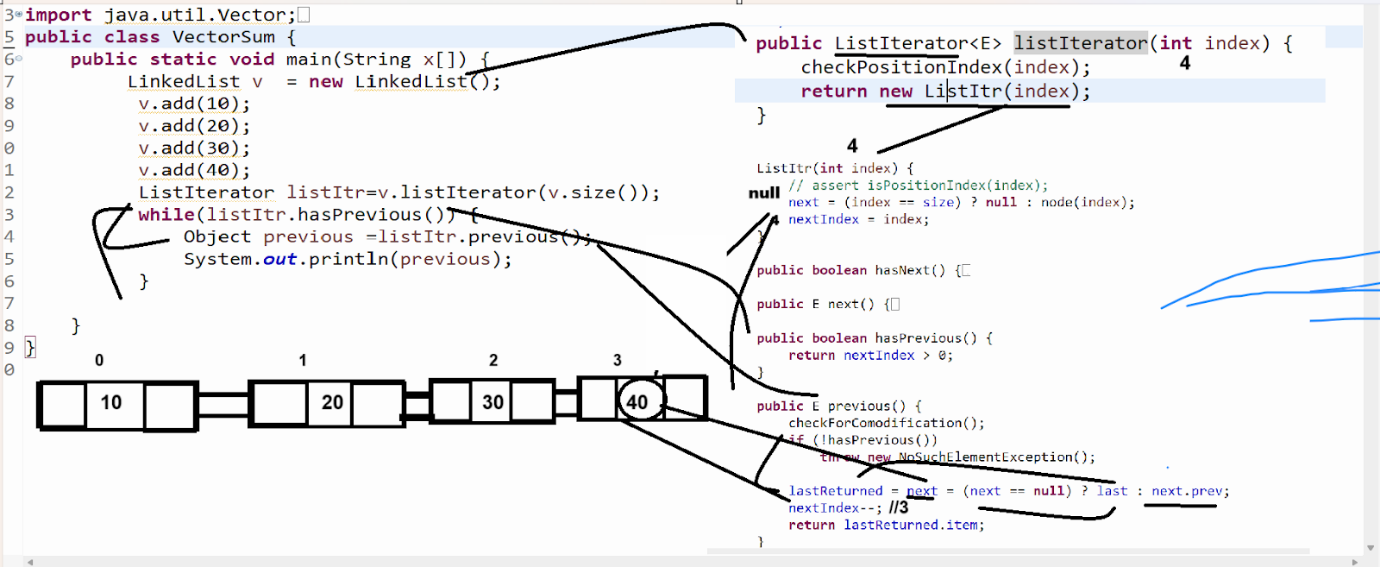
  }

}

}

**Example of ListIterator using LinkedList**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



**Q. What is the difference between Iterator and List Iterator?**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **Travelling capabilities /Traversal capabilities**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Iterator:** Travel collection or fetch data from collection using forward direction or in only one direction.

**ListIterator:** Travel collection or fetch data from collection using forward direction as well as using backward direction.

1. **Element Modification:**

**Iterator:** Iterator allow only removal operation at the time of travelling using remove() method

**ListIterator:** ListIterator can allo add(),remove(), set() -replace element in collection at the time of travelling.

1. **Index Access:**

**Iterator:** Iterator not provide method to access the index of collection at the time of traveling

**ListIterator:** ListIterator provides two methods to us: int previousIndex() , int nextIndex() for fetch  index from collection at the time to traversing.

1. **Methods available**

Iterator: Iterator contain methods boolean hasNext(),Object next(), void remove()

ListIterator: ListIterator contain methods boolean hasNext(),boolean hasPrevious(),Object next(),Objext previous(), int previousIndex(),int nextIndex() , void add(E),void set(E),void remove() etc

1. **Performance Consideration**
2. For ArrayList both Iterator and ListIterator are the fast due to the index access
3. For LinkedList ListIterator is preferable because it avoids redundant travel when moving backward as LinkedList is optimized for bidirectional iteration.

How to create Fail safe collection

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

If we want to create fail safe collection we have CopyOnWriteArrayList class

Basically CopyOnWriteArrayList is member of java.util.concurrent package and design to handle the concurrent modification like as  fail safe collection means not throws ConcurrentModification Exception

package org.techhub;

import java.util.Vector;

import java.util.concurrent.CopyOnWriteArrayList;

import java.util.\*;

public class VectorSum {

public static void main(String x[]) {

CopyOnWriteArrayList v=new CopyOnWriteArrayList();

  v.add(10);

  v.add(20);

  v.add(30);

  v.add(40);

  Iterator i = v.iterator();

  System.out.println("Before Adding "+v);

  while(i.hasNext()) {

  Object obj = i.next();

  if((int)obj==30) {

  v.add(50);

  }

  }

  System.out.println("After adding  "+v);

}

}

**How CopyOnWriteArrayList works**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **Snapshot mechanism :** when iterator is created it work on a snapshot (copy) of the underlying array at the time of iterator creation

1. **Structural modification:** operations like add(),remove() or set() create a new copy of the entire underlying array with the modification applied.
2. **Iterator behaviour :** Iterator operator on snapshot so they are unaffected by concurrent modification to the list during iteration.

1. **Foreach loop :** for each is an enhancement for loop introduced by java in JDK 1.5 version and it specially designed for fetch data from array or collection in forward directions only.

**Important points related with for each**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. Fetch data only by using forward direction
2. Not need to manually
3. Not need to check condition direct fetch from 0 to n-1 index
4. By default increment by 1 internally
5. Cannot travel data using backward direction
6. Not need to use index for fetch data directly we get data from array or collection

**Syntax:** for(data type variable:array/collection)

              {

    }

**Example with source code**

package org.techhub;

import java.util.Vector;

import java.util.concurrent.CopyOnWriteArrayList;

import java.util.\*;

public class VectorSum {

public static void main(String x[]) {

ArrayList v=new  ArrayList();

  v.add(10);

  v.add(20);

  v.add(30);

  v.add(40);

  for(Object obj:v) {

  System.out.println(obj);

  }

}

}

1. **For each method from JDK 1.8 version of java**

package org.techhub;

import java.util.Vector;

import java.util.concurrent.CopyOnWriteArrayList;

import java.util.\*;

public class VectorSum {

public static void main(String x[]) {

ArrayList v=new  ArrayList();

  v.add(10);

  v.add(20);

  v.add(30);

  v.add(40);

v.forEach((val)->System.out.println(val));

}

}

**ArrayList**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ArrayList is a internally dynamic array  of Object class and it is implementer class of List collection and it is asynchronous collection

Means not thread safe means multiple thread object can use ArrayList simultaneously

**Constructor of ArrayList**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

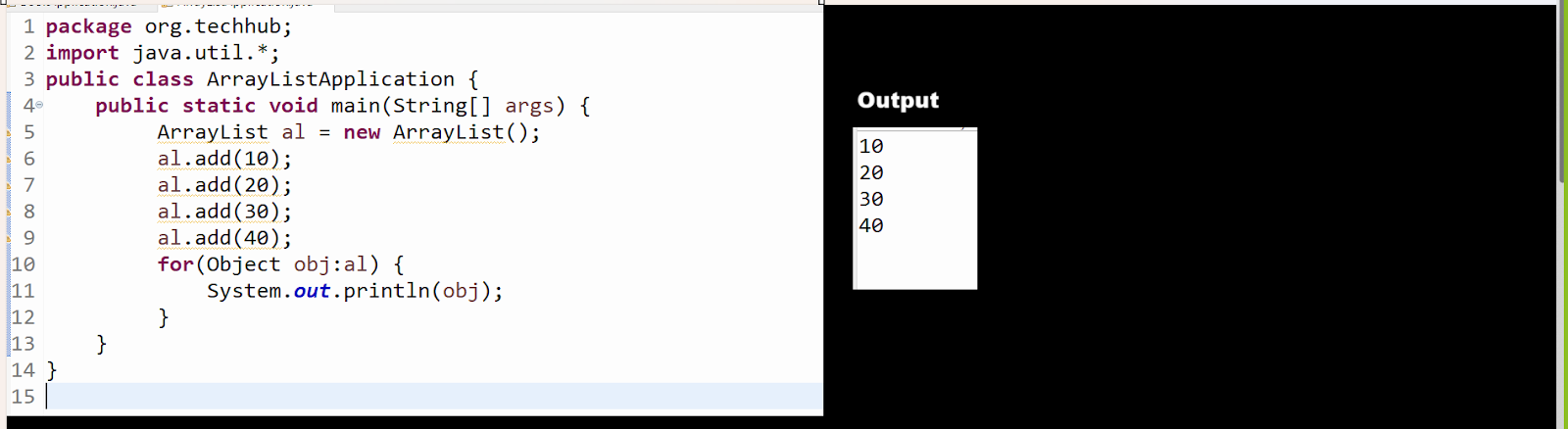
**ArrayList():** this constructor help us to create ArrayList with default size i.e 10

**ArrayList(int initialCapacity):** this constructor can create ArrayList object with initial capacity provided by user

**ArrayList(Collection):** This constructor can be used to copy data from another collection and store it in the ArrayList Collection.

**Example using ArrayList**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



**Q. What is the difference between ArrayList and Vector?**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

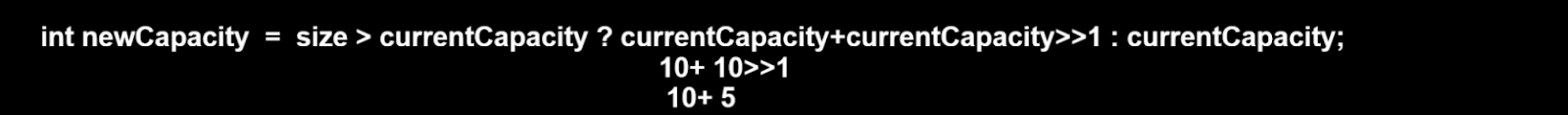
1. Vector is legacy collection and ArrayList is not legacy collection
2. Vector is thread safe or synchronized and ArrayList is not thread safe or non synchronized collection

Means performance wise ArrayList is faster than Vector but not thread safe

1. Vector allocates double memory than its current capacity when capacity crosses and ArrayList occupies half memory than its current capacity when capacity crosses.

**Logic of ArrayList capacity increment**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



1. Threshold of Vector is 1.0 or 100% and Threshold of ArrayList is 0.5

**LinkedList**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

If we think about LinkedList it is internally implementation of doubly LinkedList  and doubly linked list is combination of node with three field

Prev,next and item and prev is reference of Node which is help us to hold address of previous node and next is a reference of Node which help us to  store address of Next node and item is Object data type variable which represent internally by E generic notation which help us to store any kind of data or object in LinkedList

