04/09/15

Collections Framework

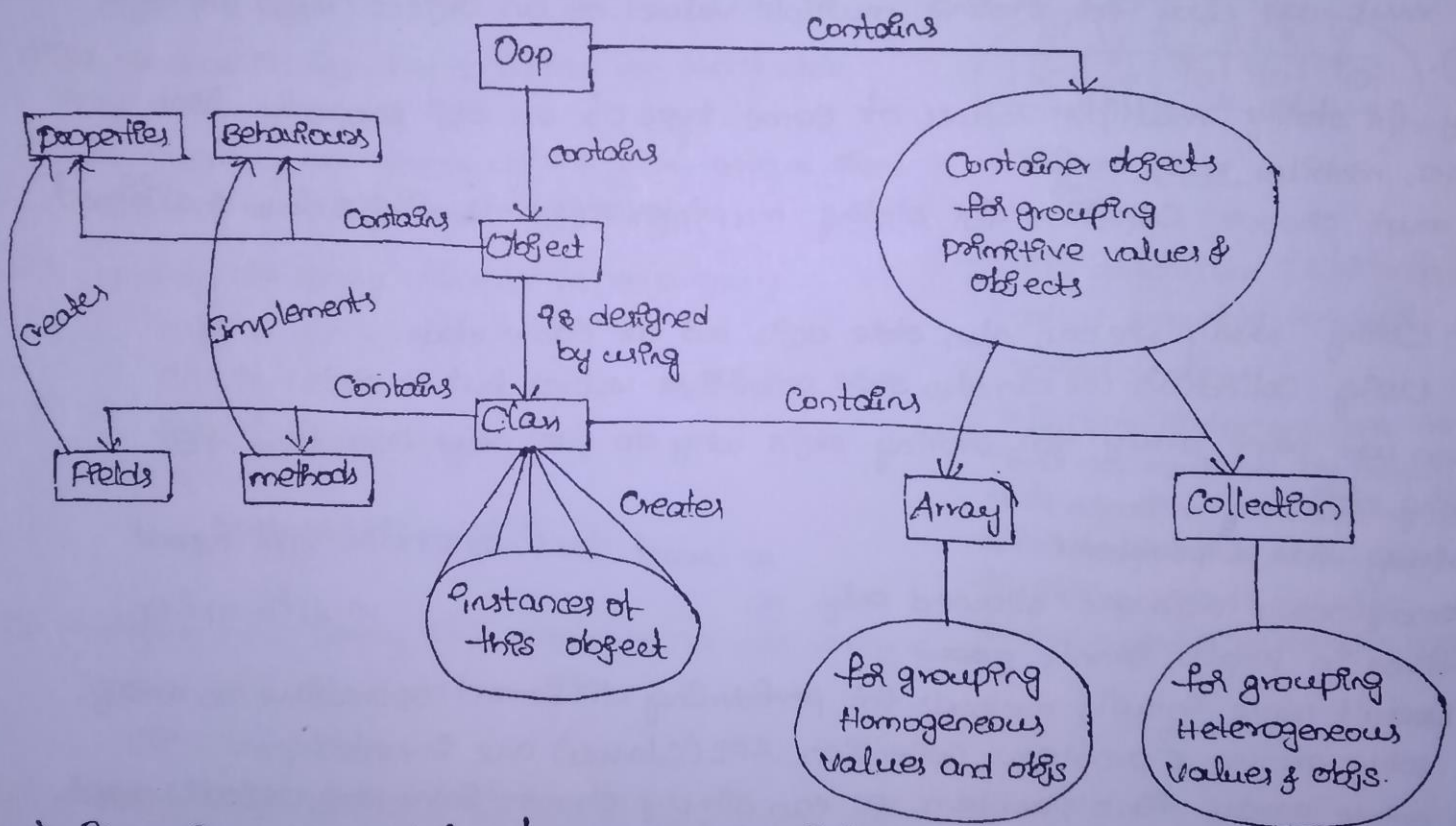
1) Oop terminology
2) Collection terminology
3) List operations
4) Set operations
5) Map Operations
6) Queue Operations
7) Cursor objects
8) utility objects

→ This chapter is meant for storing multiple objs as one group using one container obj that is collection.

1) Oop terminology:-

→ Object oriented programming concepts are invented for representing real world object in programming world by achieving security to data and dynamic method despatching in automating business operations.

→ Below diagram shows the basics of Oop.



→ By using any style of programming (Structured or Oop) our primary task is storing data in program.

→ In Java we can store data in 4 ways.

1) Variable
2) Array obj
3) Class obj
4) Collection obj

Problem:-

→ Using variable we can store only one value.

Sol:- Array.

→ Array can store fixed no. of similar type of values.

→ class obj can store fixed no. of different type of values.

Collection Definition:-

→ Collection is a container obj it is used for storing multiple homogeneous and heterogeneous, unique and duplicate objs without size limitation and further used for sending all these objs at a time from one class to another class as method argument and return value.

→ Collection is called container obj becoz it can contain other objs for storing and transfering to other class methods.

→ we must use variable for storing one value of an object of an operation.

→ we must use class for storing multiple values of an object (real world obj).

→ Array for storing multiple values of same type of an obj property like courses, mobiles nos, emails.

→ we must choose Collection for storing multiple objs of a single class or different classes.

Note:- Using array we can also store objs but of same class.
Using Collection we can also store primitive values but as objs.

Eg) When we have array for storing objs why do we need Cdlection for storing objs.

Ans:- Array has 5 problems.

1) Homogeneous objs are allowed only.
2) fixed in length, can't grow
3) Doesn't have inbuilt methods for performing different operations on array.
4) cannot store objects in diff. format
5) " " " in diff. order.

→ To solve above 5 problems Collection API (classes) are invented.

→ To solve array first problem we can simply choose java.lang.Object[] object.

→ Since it is superclass of all classes, we can store all types of java objects in this array.

→ But size problem and inbuilt operation methods problem can't be solved automatically we must define our own class with Object[] for storing objs, with methods for performing several operations on this array elements.

Algorithm for storing multiple objs without size limitation:-

1) Create a class with Object[] instance with required initial capacity, say 10.

2) Store elements in this array object.

3) before storing element we must check whether size reached its capacity or not.

4) If reached,
   a) create new array with more required capacity
   b) Copy elements from old array to this new array.
   c) Assign this new array obj reference to old referenced variable

5) Then store new element at end of all elements.

Sample code:-

program:-

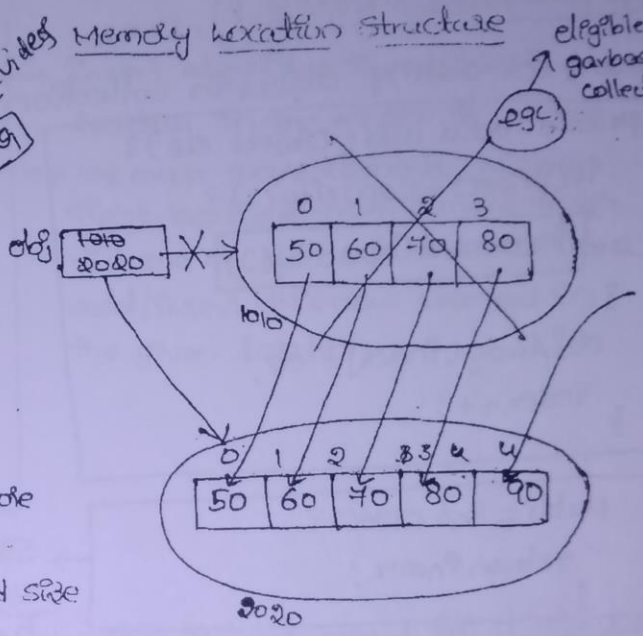
Array object creation

```
Object obj = new Object[4];
```

For complete programmatic explanation watch youtube video Collection by Hari Krishna

Storing elements

```
obj[0] = "50";
obj[1] = "60";
obj[2] = "70";
obj[3] = "80";
```



Size reached its capacity so further we cannot store any more elements,

1) So creating new array object with required size

```
Object[] tempobj = new Object[10];
```

2) Copying old array elements to new array

```
obj.length;
int i=0;
for (i; i<obj.length; i++)
 {
   tempobj[i] = obj[i];
 }
```

3) Assigning new array obj reference to old array referenced variable

```
obj = tempObj;
```

4) Storing new element at end of new array obj.

```
Obj[i] = "90";
```

→ this algorithm is technically called growable array (or) resizable array.

→ This code is tumair code becoz it is not reusable by following this algorithm we must develop a full-fledged class with Obj[] for storing elements and with several methods to perform operations.

08/09/15

→ Below program shows implementing our own collection like sun given collection.

// NITCollection.java

```
class NITCollection {
  //meant for storing/collecting objects.
  private Object[] objArray = new Object[10];
  // maintains array index and also size
  private int index=0;
  //meant for adding object in collection
  public void add(Object ele){
    if(size()==capacity()){
      incrementCapacity();
    }
    objArray[index] = ele;
    index++;
  }
  public int size(){
    return index;
  }
  public int capacity(){
    return objArray.length;
  }
  private void incrementCapacity(){
    Object[] tempArray = new Object[capacity()*2];
    for(int i=0; i<objArray.length; i++){
      tempArray[i] = objArray[i];
    }
    objArray = tempArray;
  }
```

For YouTube video explanation search Growable Array algorithm by Harikrishna

size means no. of objs we have stored in collection this number is maintain by index variable so we return it.

capacity means total no. of objs we can have stored in collection. this no. maint is nothing but length of array.

Above 4 methods are meant for solving Array size problem this method will be called when size = capacity to increase current capacity to double.

```
  public Object get(int i){
    return objectArray[i];
  }
```

meant for retreiving & returning given index object.

```
  public void replace(int i, Object ele){
    objArray[i] = ele;
  }
}
```

this method replaces given index obj with new given obj in this collection.

For youtube search [remaining element] by Hari krishna

```java
public void remove(int i) {
  for ( ; i< size()-1; i++) {
    objArray[i] = objArray[i+1];
  }
  objArray[i] = null;
  index--;
}
```

→ removing an element meaning not dropping in a location, instead dropping that element from this array for this purpose we must replace current element with next elements until end of all elements finally in last location we must add null, should decrease size by one.

search [inserting element] by Harikrishna

```java
public void insert(int i, object ele) {
    if(size()==capacity()) {
      incrementCapacity();
    }
    for (j=s.
    for (int j=size()-1;j>=i;j--) {
      objArray[j+1] = objArray[j];
    }
    objArray[i] = ele;
    index++;
}
```

→ Insert algorithm is not inserting a location in the middle of collection.

→ we must move elements to their right locations from current index till end of elements then we must add/insert this new element in the given location.

```java
@override
public String toString() {
  StringBuilder sb = new StringBuilder();
  sb.append("[");
  for(int i=0; i<size(); i++) {
    sb.append(objArray[i]);
    sb.append(", ");
  }
  int start=sb.lastIndexOf(", ");
  if (start != -1) {
    sb.delete(start, start+2);
  }
  sb.append("]");

  return sb.toString();
}
}
```

→ This method meant for displaying the objs those are added to collection.

→ delecting last , (comma) and space

→ We have completed our own development with collection operations adding, counting, retrieving, removing and inserting.

→ So this class has solved Object[] two problems.

1) size and
2) operations

→ Let us develop a test app for creating collection obj, adding objs without size limitation then further to perform replace, remove & insert operations.

```
//NITCollectionTest.java
class NITCollectionTest {
  public static void main(String[] args) {
    NITCollection col=new NITCollection();
    Sopln("capacity:"+col.capacity()); →10
    Sopln("size:"+col.size()); →0
    Sopln("elements:"+col); — []
    col.add("a");
    col.add("b");
    col.add("c");
    col.add("d");
    col.add(5);
    col.add(6);
    col.add(7);
    col.add(5);
    col.add(9);
    col.add(10);
    Sopln(col);
    Sopln("capacity:"+col.capacity()); →10
    Sopln("size:"+col.size()); →10
    sopln("elements:"+col);
    col.add(11);
    - - -
    - - - } → same *1
    - - -
```

*1 ←

```
//retrieving 3rd index object from collection
Object obj=col.get(2);
Sopln("\n2nd index element:"+obj);
```

Same as *1 ←

```
//inserting new element at 4th index
Col.insert(4,true);
Sopln("\n the element true is inserted");
```

Same as *1 ←

// removing 0 index element

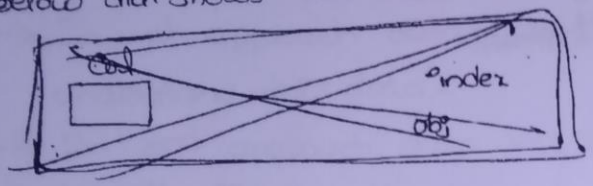
```
Col.remove(0);
SOpln("In the 0 index element is removed")
- - - - - } → same as #1
```

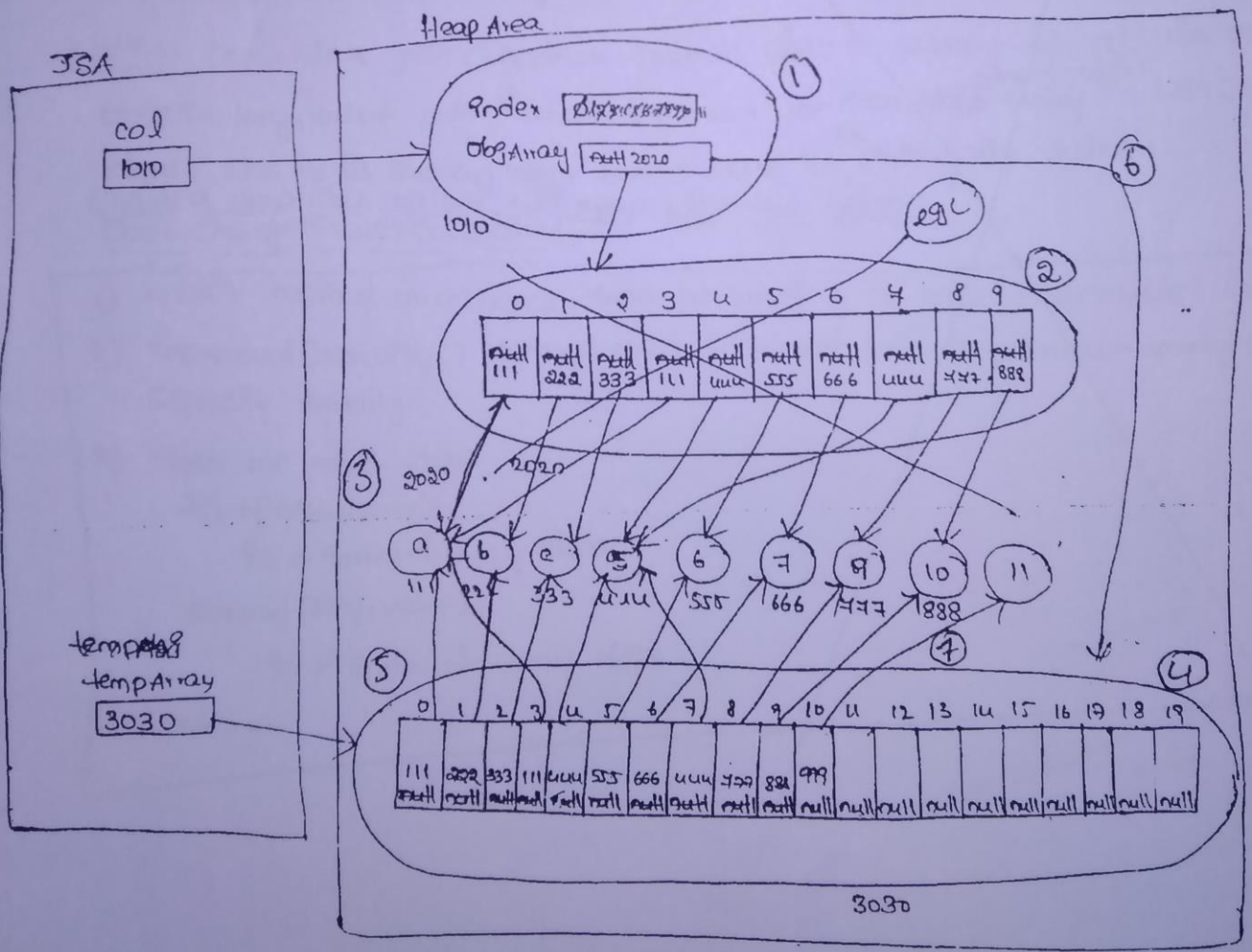
→ Converting primitive datatype into its associated object is called autoboxing. and its reverse process is called autounboxing.

// replacing 5th index obj with new object 6.9

```
Col.replace(5, 6.9);
            ↓
        primitive double
SOpln("In the 5th index obj is replaced");
= = = = } → same as #1
```

→ Below dia. shows JVM architecture (with above obj memory) with collection obj memory diagram.





Different operations we perform using collection object:-

→ Using collection obj we will perform below seven operations.

1) Adding objs
2) counting objs
3) searching objs
4) retrieving "
5) removing "
6) replacing "
7) Inserting "

for video explanation search in youtube

Sun given collection clases:-

→ To perform above 4 operations sun defined several collection clases.

→ All these clases are available in the package java.util.

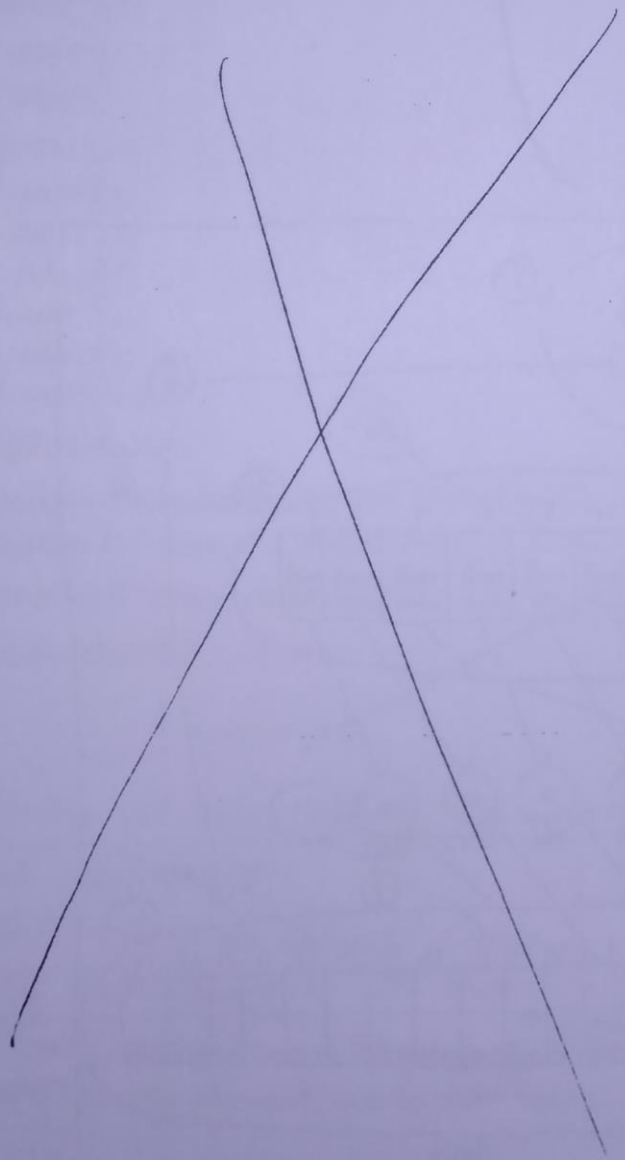
x[→ 8 formats]x

Two formats for colleting objects:-

→ (Collection API has)

→ we can store/collect objs in 8 different formats.

1) Anay format
2) key,value pair format



→ In Array format object doesn't have identity in key, value pair format objs will have identity. Here key is identity, value is the actual objs. For example to collect an employee data,

In Array format, it looks like as below.

0	1	2	3
4249	HariKrishna	NareshIT	Java

In key, value pair format, it looks like as below.

key	value
eno	4249
ename	HariKrishna
Institute	NareshIT
Teaches	Java

→ In 1st approach data doesn't have identity then end user may wrongly understand this data.

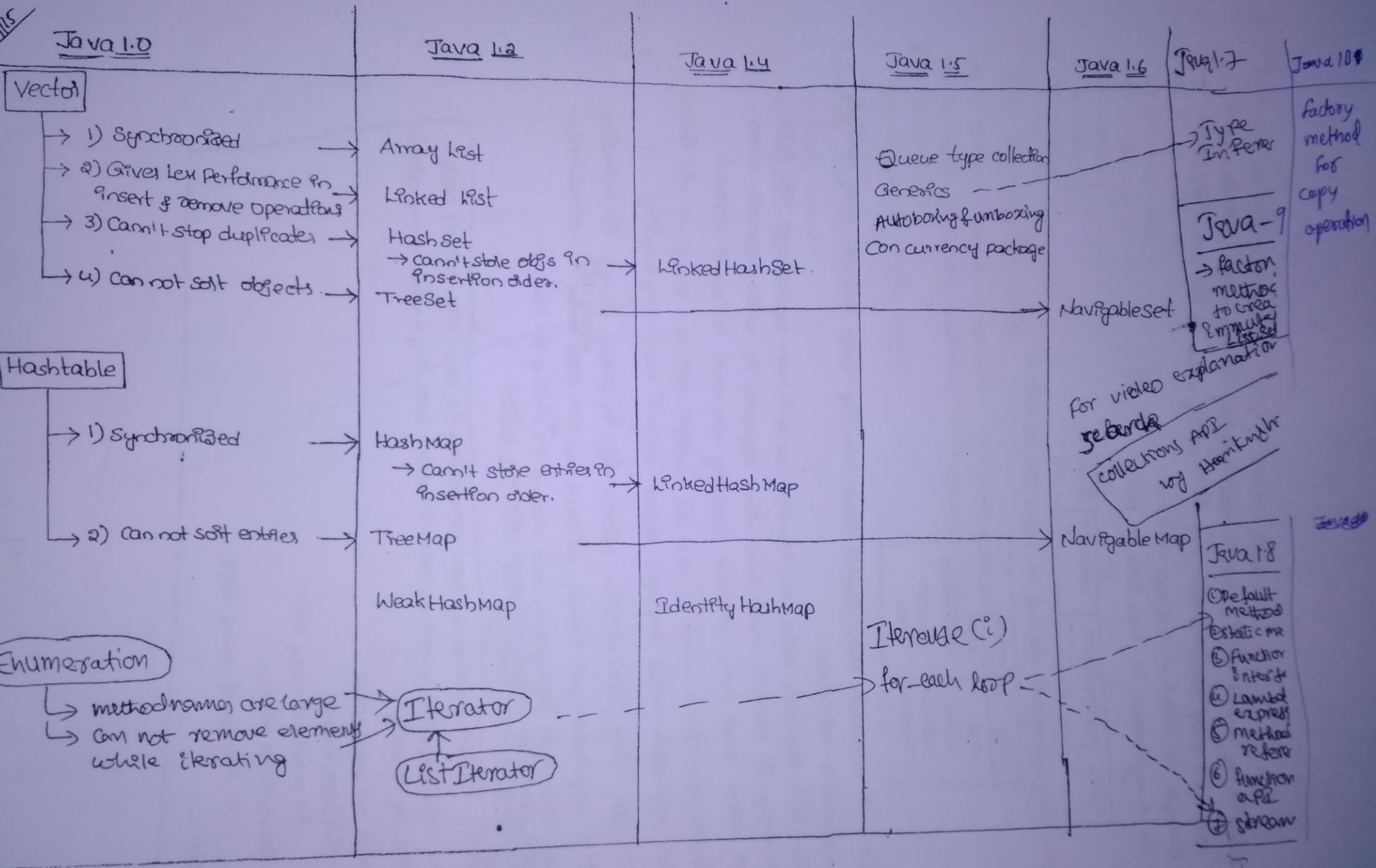
→ In 2nd approach data has identity so end user understands this data correctly. It doesn't mean that first approach is worst approach. When we want send data without identity we must follow 1st approach.

for example if we want to send all employee objs as a group we will follow first approach becoz employee object doesn't need any identity.

Note:- our class NITCollection collects objs in array format. for collecting objs in key, value pair format, we must use two array objs in NITCollection class. One is for storing keys and second is for storing the values.

Create a new class called NITMap with the below Code change in NITCollection:-

1) add() method must have two parameters to send (key, value) objs
2) incrementCapacity() method must increment both key & value array objects capacity equally.
3) Then we must store
   first(key, value)
      in 0 index of array objs
   second(key, value),
      in 1st index of array objs
   etc...

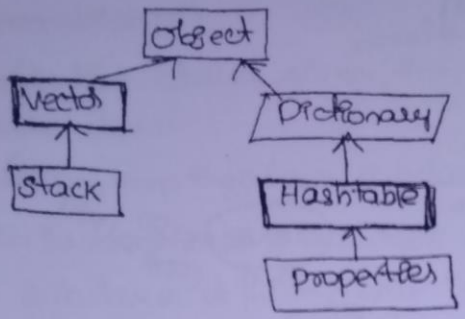

Java 1.0
Java 1.2
Java 1.4
Java 1.5
Java 1.6
Java 1.7
Java 1.8
Vector
1) Synchronized
2) Gives less Performance in insert & remove operations
3) Can't stop duplicates
4) Can not sort objects
Array List
Linked List
Hash Set
→ can't store objs in insertion order.
TreeSet
LinkedHashSet
Queue type collection
Generics
Autoboxing & unboxing
Concurrency package
Navigable set
Type Inferer
Java-9
→ factory methods to create immutable List, Set
factory method for copy operation
Hashtable
1) Synchronized
2) Can not sort entries
HashMap
→ Can't store entries in insertion order.
LinkedHashMap
TreeMap
NavigableMap
WeakHashMap
IdentityHashMap
for video explanation
Collections API by Harinath
Java 1.8
① Default method
② static me
③ Function interface
④ Lambda expression
⑤ method reference
⑥ functional API
⑦ stream
Enumeration
→ method names are large
→ Can not remove elements while iterating
Iterator
ListIterator
Iterable (i)
for-each loop

SUN given Collection classes :- in Java 1.0

→ For storing elements in array and (key, value) pair formats SUN defined below 2 classes in java.util package.

1) Vector → for storing elements in array format
2) Hashtable → for storing elements in key, value pair format.

→ These 2 classes are available from Java 1.0 version itself. These two classes are created with below hierarchy:



Collections framework classes :- given in Java 1.2

→ In Java 1.2 version java.util package is updated by adding many new classes & interfaces for adding more functionality in storing objs in diff. formats/ orders

Diagram :-

Below dia. shows all classes added in Java 1.2, 1.4, 1.5 & 1.6 refer dia. in previous page.

→ From Java 1.2 onwards java.util packages are called Collections framework classes. because all these classes are working for solving a particular problem that collecting objs without size limitation.

> A Framework is a semideveloped app that contains set of classes and interfaces for solving a particular problem.

→ We can interchange one Collection obj with another collection obj because collection API is a runtime polymorphic API.

→ To achieve Runtime polymorphic nature all collection classes are created under 2 superclasses

Collections framework hierarchies :-

→ Collections framework classes are grouped into 2 hierarchies.

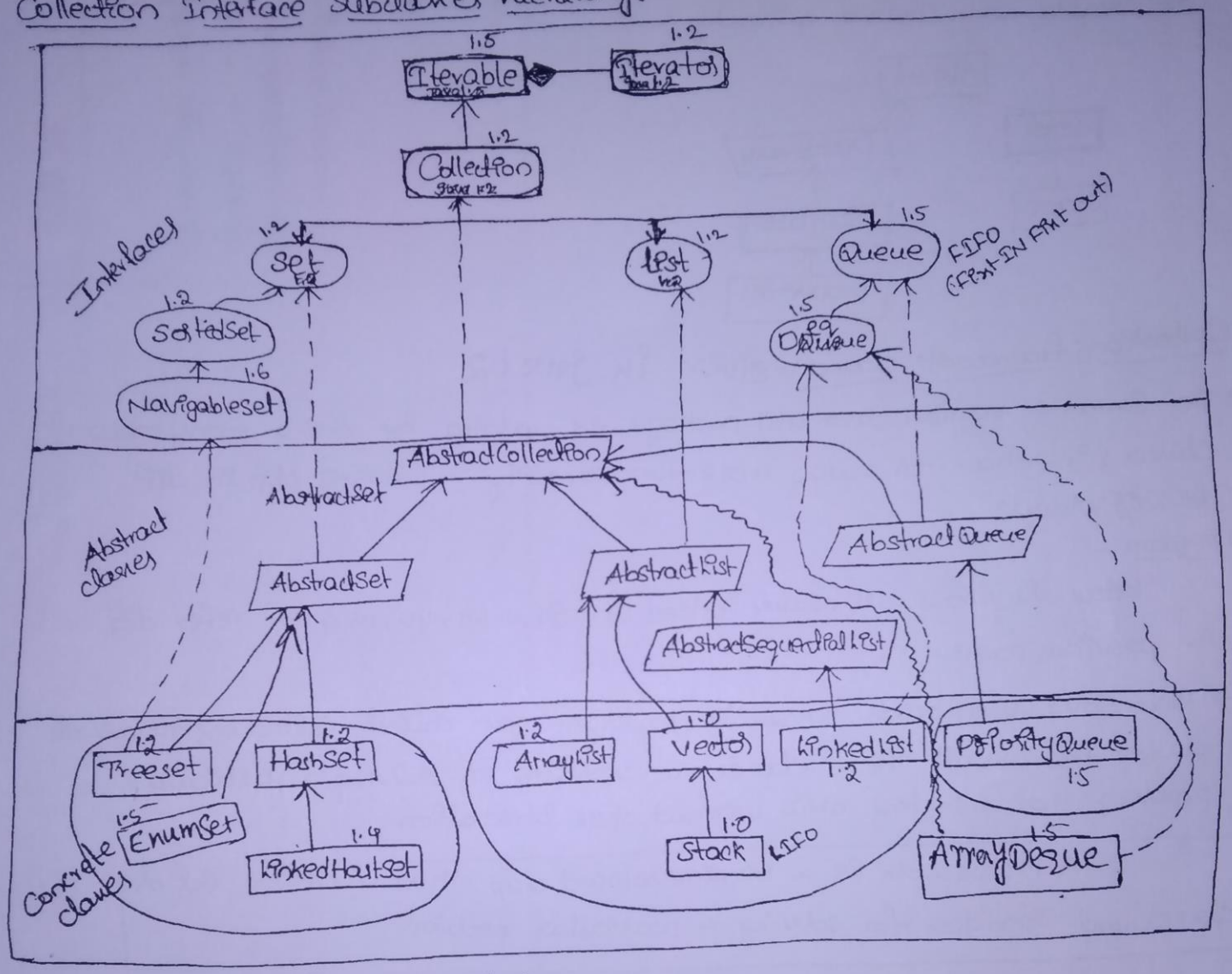
1) Collection hierarchy
2) Map hierarchy

→ Collection & Map are 2 root interfaces of collection framework classes.

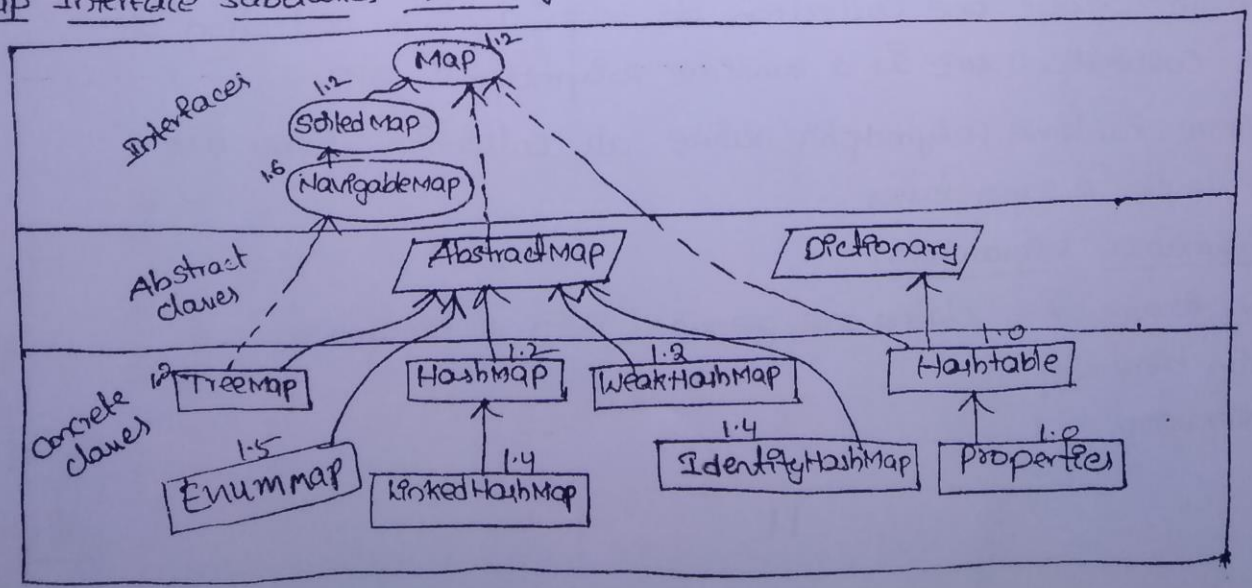
→ Collection Interface subclasses are meant for collecting objs in array format.

→ Map Interface subclasses are meant for collecting objs in key, value pair format.

Collection Interface subclasses hierarchy:-



Map Interface subclasses hierarchy:-



Q) What classes are called legacy classes, collection framework classes and generic framework classes?

Ans:- legacy means old version classes

→ The collection classes those are given in Java 1.0 are called legacy collection classes. They are:

1) Vector
2) Hashtable
3) Enumeration (I)

→ The collection classes given newly from Java 1.2 onwards are called Collection framework classes

Collection, Map interfaces subclasses are (added to) called Collection framework as classes.

Note:- In Java 1.2 Vector & Hashtable classes are added to collection framework as subclasses of List & Map interfaces.

→ From Java 5 onwards collection framework classes are also called Generic collection classes. In Java 5 version collection classes are rewritten with generic syntax for acting only homogeneous objs.

```
class ArrayList{}
class HashMap{}
```
→ are collection framework classes

```
class ArrayList<E>{}
class HashMap<K,V>{}
```
→ are collection framework classes with generic syntax.

→ From Java 5 onwards we will have collection classes only with Generic syntax but we have chance to use collection classes with Generic syntax or without generic syntax, it is our wish.

```
ArrayList al = new ArrayList();
```
→ collection obj without generic syntax

```
ArrayList<String> al = new ArrayList<String>()
```
→ collection obj with generic syntax.

→ The term legacy is only applicable to collection classes, because Vector & Hashtable classes got alternative collection classes.

Collection programming:-

→ To implement code using collection API we must follow below 3 steps.

1) add "import java.util.*;" statement
2) Create appropriate Collection obj based on storing and retrieving order.
3) Call appropriate methods to perform any one of the 4 operations.

→ for creating collection obj we must know the available constructors in all collection classes.

→ In every collection class we will have minimum 2 constructors.
In collection subclasses we have:
1) Non-parameterized constructor
   → for creating empty collection object.
2) Collection parameter constructor
   → for creating a collection with given collection elements (copied).

Note:- Stack has only noArgparam / zero-param constructor.

→ In Map subclasses we have:
1) Non-parameterized constructor
   → for creating empty map object
2) Map parameter constructor
   → for creating map objs with given map entries.

For more details check API documentation of each collection class.

Sample code:-

```
class ArrayList{
    ArrayList(){}
    ArrayList(Collection c){}
}
```

```
class HashMap{
    HashMap(){}
    HashMap(Map m){}
}
```

Q) Is Map subinterface Collection?

A:- No, because Map chars are different from Collection. Collections can be stored in the form of Array format while Map can be stored in the key, value pair format.

→ These are many classes working together to perform a particular task is called "Framework".

11/12/15

Collection Interface methods:-(16+5)

Collect

1) public boolean isEmpty()
2) public boolean add(Object obj)
3) public boolean addAll(Collection c)
4) public boolean remove(Object obj)
5) public boolean removeAll(Collection c)
6) public void clear()
7) public boolean contains(Object obj)
8) public boolean containsAll(Collection c) (+)
9) public boolean retainAll(Collection c)
10) public int size() → for counting
11) public Iterator iterator() → used for retrieving elements from Collection.
12) public int hashCode()
13) public boolean equals(Object obj)
14) public Object[] toArray()
15) public Object[] toArray(Object[] obj)

Java 8v added new Method

16
17
18
19
20

Refer API Documentation Collection Interface

11 Common interview questions asking on all type of Collections:-

1) What is the use of this Collection?
2) What is the implemented datastructure?
3) Default Capacity, Increament Capacity, load factor
4) What type of elements are allowed? (Homogeneous, heterogeneous like that)
5) Is null allowed how many?
6) What is the storing order?
7) What is the retrieving order?
8) Is it synchronized Collection?
9) IS it ordered collection?
10) Internally calling method?
11) Performance

→ These 4 questions we must learn for collections.

Working with Arraylist:-

Q1)-A) for storing multiple objs with index multip including duplicates without thread safety or synchronization we must use Array list.

Q2-A) Growable array or resizable array is the implemented datastructure of Arraylist.

Q3-A) Default capacity - 10

Incremental capacity = $\left(\frac{\text{current capacity} * 3}{2}\right) + 1$ => $\left(\frac{10*3}{2}\right) + 1$

load factor not applicable here => 15

Qu-A) All 4 type of elements are allowed are homogeneous, heterogeneous, unique and duplicate objs.

Q5-A) null insertion is possible, more than one null allowed beco duplicates are allowed.

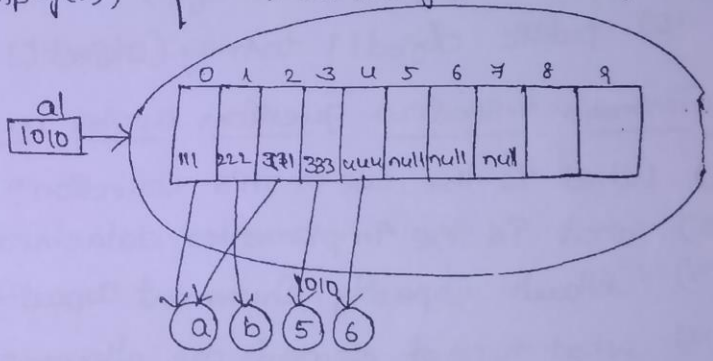
Q6-A) Insertion order with index. Insertion order means the order in which addmethods are called.

Q7-A) Random access ofcouse Sequencial access is also possible.

Q) Below program shows all above points with (a program) arraylist.

```
// ArrayListDemo.java
import java.util.*;
  Class ArrayListDemo {
    public static void main(String[] args) {
      Arraylist al = new Arraylist();
        Sopln("isEmpty:"+al.isEmpty());
    //adding all u types of objs.
        al.add("a");
        al.add("b");
        al.add("a");
        al.add(5);
        al.add(6);
        al.add(null);
        al.add(null);
        al.add(null);
        Sopln("size:"+al.size());
        Sopln("elements:"+al);
    //retrieving objs randomly
        Sopln("3rd index object:"+al.get(3));
      }
  }
```

Q8-A) not synchronized
Q9-A) ordered collection
Q10-A) equals() method in searching & removing operation
Q11-A) gives high performance in
→ adding & retrieving
gives low portance in
→ searching & removing & inserting



Compilation & execution:-

```
> javac ArrayListDemo.java
> java ArrayListDemo
        isEmpty : true
        size : 8
        elements : [a, b, a, 5, 6, null, null, null]
        3rd index object : 5
```

Working with Vector:-

→ Vector and ArrayList functionalities are same the only differences are there in

i) ArrayList not synchronized and Vector is synchronized.

ii) ArrayList incremental capacity half+1 ($\frac{1}{2}+1$) whereas vector incremental capacity is double.

Q1-A) Vector is used for storing homogeneous and heterogeneous unique and duplicate objs in insertion order with index in multithreading environment with thread safety & synchronization.

Q2-A) Growable array

Q3-A) default capacity - 10
Incremental capacity - double

Q4-A) All type of elements are allowed.

Q5-A) null insertion is possible, more than one.

Q6-A) Insertion order with index

Q7-A) Random access

Q8-A) it is synchronized collection

Q9-A) it is ordered collection

Q10-A) equals() method in searching & removing compared to AL

Q11-A) gives low performance → adding & retrieving
gives high performance → searching & removing & iterate

Q) Rewrite above program by replacing ArrayList obj with vector for testing above points.

Ans:-

```
//VectorDemo.java
import java.util.*;
Class VectorDemo{
  public static void main(String[] args){
    Vector v = new Vector();
    Sopln("IsEmpty:"+v.isEmpty());
    v.add("a");
    v.add("b");
    v.add("a");
    v.add(5);
    v.add(6);
    v.add(null);
    v.add(null);
    v.add(null);
    Sopln("Size:"+ v.size());
    Sopln("elements: "+v);
    Sopln("3rd index object:"+v.get(3));
  }
}
```

o/p:-

```
IsEmpty: ~~false~~ true
Size: 8
elements:[a,b,a,5,6,null,null,null]
3rd index object:6
```

12/09/15

Working with stack:-

Q1-A) We must use stack for storing multiple objs and further for retrieving them in "Last in First out manner (LIFO)".

→ Stack is a class in Java which is a subclass of vector. All methods of Vector are inherited to Stack. but we should not use Vector class methods on stack object because we can't retrieve objs in LIFO manner.

→ To treat vector as stack obj stack class has its own special methods.

1) push :-

```
public E push (E item)
```

This method will store obj in stack

2) public obj pop:-

```
public E pop();
```

→ Removes the obj at the top of this stack and returns that top object to our program.

Note:- top object means last added obj.

3) public obj peek:-

```
public E peek();
```

→ This method returns the top obj from this stack without removing.

Rule:- Above 2 method we should not call on empty collection both methods will throw java.util.EmptyStackException. It is unchecked exception optional to handle.

4) public int search(Object o):-

```
public int search(Object o)
```

→ It returns obj position inside stack the position (searching index) start with 1 from top to bottom top is last element and bottom is first element.

→ If obj not found this method returns -1

Q2-A):- stack is implemented datastructure (Growable array backed by vector obj).

Q3-A):- default capacity - 10
Incremental capacity - double

remaining points are same as vector

Q4-A):- all u type of objs

Q5-A):- null allowed, more than one

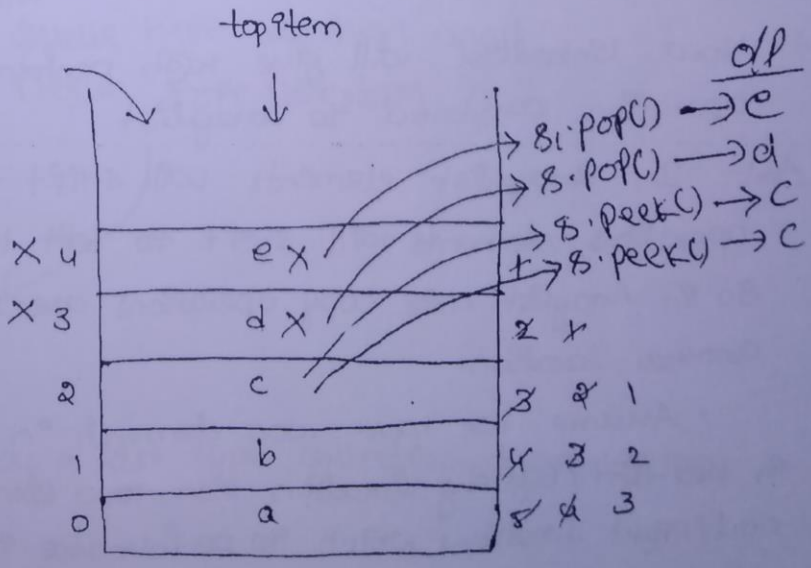
Q6-A):- insertion order

Q7-A):- LIFO order.

Q) Below prg shows all above points and working with stack.

```
import java.util.*;
class StackDemo{
  public static void main(String[] args){
    Stack s=new Stack();
    s.push("a");
    s.push("b");
    s.push("c");
    s.push("d");
    s.push("e");
    Sopln(s);
  // retrieving & removing top item
    Sopln(s.pop());
    Sopln(s);
    Sopln(s.pop());
    Sopln(s);
    Sopln();
  // only retrieving top item
    Sopln(s.peek());
    Sopln(s);
    Sopln(s.peek());
    Sopln(s);
    Sopln();
  // searching for an item
    Sopln(s.search("b"));
    Sopln(s.search("e"));  → -1
  }
}
```

O/P:- [a, b, c, d, e]
e
[a, b, c, d]
d
[a, b, c]
c
[a, b, c]
c
[a, b, c]
2
-1



Q) What is the diff. b/w pop & peek methods?

A:-→ pop() method will remove & return top item. peek() method will only retrieve and return will not remove top item.

→ Everytime when we call pop() method different object will return. Everytime when we call peek() method it returns same o/p.

Working with LinkedList:-

Q1-A) → If our operations are more insert and remove elements at the begining & in the middle of the collection then we must use LinkedList. It gives better performance Compared to other collections for insert and remove elements in the begining & middle of collection.

Q2-A) Implemented datastructure is Doubly-LinkedList

Q3-A) Default capacity-0
Increment capacity- 1

Q4-A) All u elements

Q5-A) null allowed, multiple

Q6-A) insertion

Q7-A) only sequential because no index

Q8-A) it is not synchronized

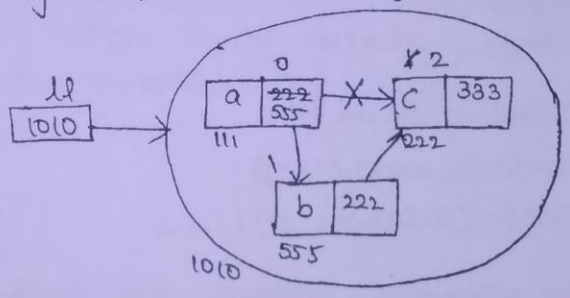
Q9-A) it is ordered collection, insertion order

Q10-A) it internally calls equals() method in searching & removing operations

Q11-A) gives high performance in insert & remove operations in the middle of collection (by object)

Below program shows working with LinkedList object.

```
LL ll=new LL();
ll.add("a");
ll.add("c");
ll.add(1,"b");
Sopln(ll);
```



→ How LinkedList will give high performance in insert and remove operation compared to arraylist.

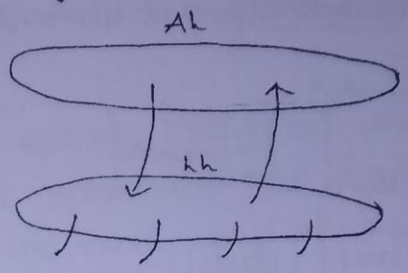
A:- In Arraylist elements will shift to right locations in insert operations, elements will shift to left locations in remove operation. So in Arraylist more copy operations are performed from one location to another location.

Assume we have 1000 elements in AL, want to insert 50 elements in beg-first / begining location then 1000 elements 50 times should copy to next/Right locations which is performance issue.

```
public class LinkedListTest
{
  p s v m
  {
    LinkedList<Object> li = new LinkedList<>();
    li.add("a");
```

→ In this case if we use LinkedList existing 1000 elements no need to shift Right side so 50000 copy operations only 100 copy operations need to perform for changing the links. So LinkedList execution is fast.

→ Algorithm for performing insertion operation using LL:-



1) Create AL object
2) Add objs into AL
3) For inserting objs in the middle
   i) Create LL obj
   ii) Copy obj from AL to LL
   iii) Insert missing obj in LL
   iv) clear AL (remove all objs from AL)
   v) Copy results obj from LL to AL.

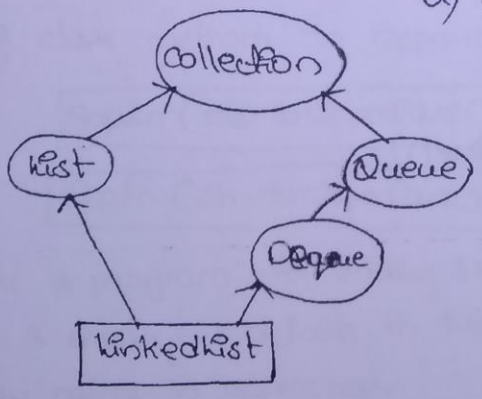
Note:- We shouldn't use LinkedList for adding objs and also shouldn't be used in LinkedList obj for inserting one or two elements it will gives less performance becos of the below reasons.

1) It consume more memory compared to ArrayList
2) If we directly insert ArrayList the no. of copy operations are less than the insert operation through LinkedList. So we must use LL only for more objs insert and remove operations.

→ In real world/Java world we can't use multiple inheritance with classes (becos (classes contains implemented) of reusability is not present in classes.)

Q) What is the collection class exist in collections implementing multiple inheritance?

Ans:- LinkedList class. It is the subclass of List Interface and also subclass of DQueue Interface.



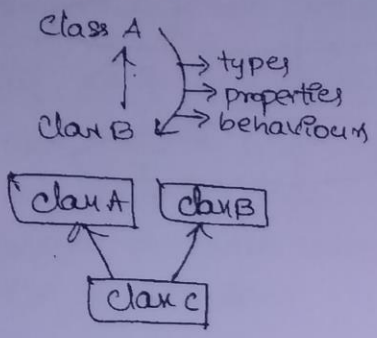
Q) How can we use LinkedList as List type collection and Queue type collection and Dequeue type collection ?

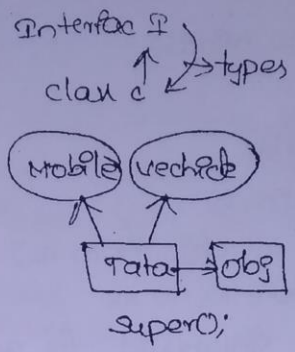
```
List l = new LinkedList();
Dequeue d = new LinkedList();
```

→ In first line LinkedList will act as a List type collection here we can only access List() methods.

→ In second statement LinkedList will act as Queue type hence we can access only Queue type methods.

→ Interface is used to provide only forcibility and class is used to provide only reusability. and abstract class have these two forcebility & reusability.





14/09/15

② write a program for adding String Object to ArrayList, display them in uppercase on console.

```
import java.util.*;
class ArrayList {
  P S V main(String[] args) {
    ArrayList al=new ArrayList();
    al.add("a");
    al.add("b");
    al.add("c");
    Sopln(al);
    for(int i=0;i<=al.size();i++){
    al.get(i);
```

Stmt1 → `Object obj=al.get(i);`

Stmt2 → `String str=(String)obj;`

```
    Sopln(str.toUpperCase());
  }
  Sopln();
  Sopln(al);
 }
}
```

⇒ understanding Retrieving objects from collection by using get(index) method

⇒ get method CE & RE, solutions

O/P:- [a, b, c]

A
B
C

[a, b, c]

Q) In above program why did u cast string obj to string type in stmt2?

Ans:- The obj comming out from the collection will have the type java.lang.Object, beco2 get method return type is Object, beco2 Collection is heterogeneous for returning any type of obj method return type should be java.lang.object.

→ We can't perform the current return obj specific operations when it is stored in Object type variable so we must convert its type from Object type to its own type for this purpose we must use cast operator.

Short ans:-

→ Get method returns every element from Collection as Object type, we must convert its element to its own type for calling its own specific operation methods so we must do cast operation.

Up casting & down casting:-

→ storing subclass Object reference in superclass variable is called upcasting.

for eg:- `Object obj = al.get(i);`

Here returning object is subclass

→ Converting subclass object type from superclass type to (subclass) its own type is called downcasting.

`String str = (String) obj;`

→ we must do downcasting only when we have to SubClass specific functionality methods.

→ In above program we have casted object to string type for calling string class methods to uppercase

`Sopln(obj.toUpperCase());` → X CE:

`Sopln(str.toUpperCase());` → ✓

Q) write a program for adding 3 employees salary and name. Display all these 3 employees details in table format with 2 columns name and sal. display names in uppercase.

Ans:-

```
import java.util.*;
class DisplayALHetroEleInCx{
    p s v main(String[] args){
        ArrayList al = new ArrayList();
```

Samp project & on adding, retrieving & modifying object in collection

```
al.add("Hari");
al.add(10000);

al.add("Balayya");
al.add(20000);
al.add("Nani");
al.add(30000);
sopln(al);
for(int i=0;i<al.size();i++){
Sopln();

Sopln("NAME  SAL");
Sopln("==========");

for(int i=0; i<al.size(); i++){
  Object ele=al.get(i);

  if(ele instanceof String){
    String name=(String)ele;
     Sopln(name.toUpperCase()+"17");
  }else{
    Sopln(ele);
  }
}
}
```

for avoiding CCE

checking the ele type if ele is string type casting it to string

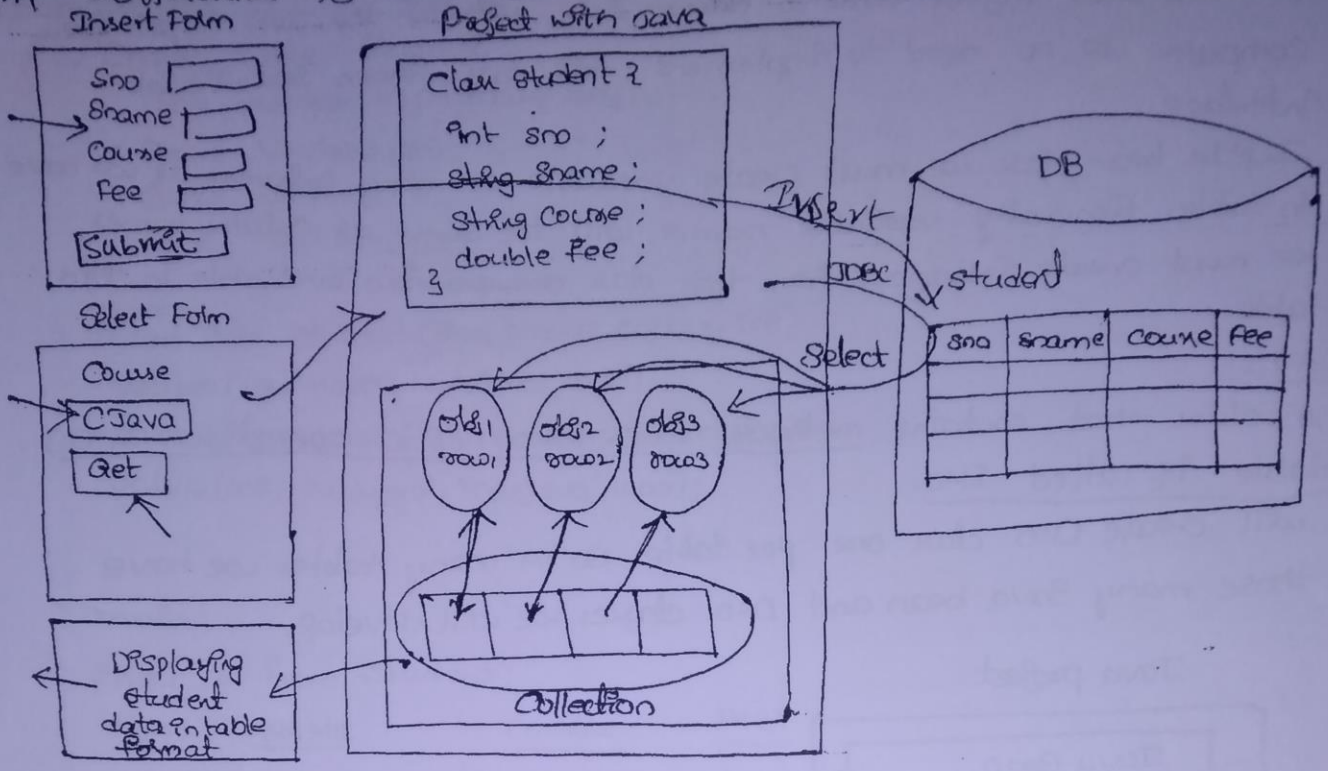
→ If Collection contains different class objects, while retrieving these objs from collection we must (do casting) cast these objs to their own type inside instanceof operator if condition as shown in above program otherwise we will get ClassCastException, program terminated in the middle of execution.

→ In above program comment if & else lines u will get CCE

1) Adding → L.add(obj); (C)
2) Counting → L.size(); (Collection)
3) Retrieving → L.get(index); (L)
4) Inserting → L.add(index, obj) (L)
5) Removing → L.remove(obj) (C)
L.remove(index) (L)
6) replacing L.set(index, obj) (List)
4) searching → L.contains(obj) (C)
L.indexOf(obj) (L)

~~8) Develop a project for inserting student data from database to textfile based on given id.~~

Q) Develop a project for transfering student data from database to end user.



15/09/15

Java DB mapping :-

DB	Java
table	class
columns	variables
rows	instances

Object data

n objects → n rows in database
→ n instances in Java Heap.

Java bean :-

→ As many tables we have in database those many bean classes we develop in Java.

→ bean is also a class created with the tablename.

→ A class that contains only private variables and public setter, getter methods for storing and retrieving an object is called bean class.

→ bean class is implemented from java.lang.io.SerializableInterface. Then only this class obj data will travel through n/w.

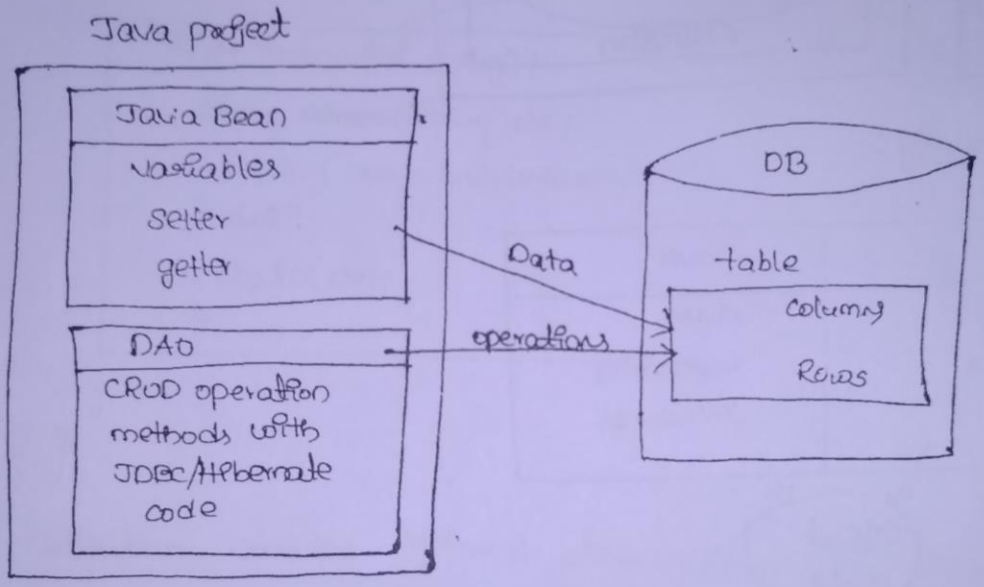
→ If this class object data is using only within the JVM in single computer we no need to implement this class from serializable interface.

→ Inside bean class we must create variable as many columns as we have in table. Generally variable names will be same as column names. we must create instances from this class one per row available in this table.

DAO:-

→ A class that contains methods to perform CRUD operations on DB table is called DAO.

→ will create DAO class one per table so as many tables we have those many Java bean and DAO classes we will develop.



→ In order to develop NareshIT students information system. we must develop below components.

1) One new user SIMS/SIMS
2) One table - Student (Sno, Sname, Course, fee)
3) Insert Sample rows with some student data (H, B, N)
4) One Bean class - [Student.java] StudentBean.java
5) One DAO class - StudentDAO.java
6) Main method class - FrontOffice.java

User creation Command:-

1) Login to system / manager user
2) Run below commands at SQL prompt
   > Create user sims identified by sims;
   > Grant DBA to sims;
   > Connect sims/sims;

Table creation command:-

```
> Create table Student(
    Sno number(5) primary key,
    Sname varchar2(20),
    Course varchar2(10),
    fee number(7,2)
);
```
(7 = total, 2 = fractional part)

```
> Insert into student (Sno, sname, course, fee)
  values (101, 'Hari', 'corejava', 1000);
  Insert into student (Sno, Sname, Course, fee)
  values (102, 'Balayya', 'corejava', 1000);
                    :
> commit;
> Select * from student;
```

SNO	SNAME	COORSE	FEE
101	Hari	Corejava	1000
102	Balayya	Corejava	1000
:	:	:	:

Java side coding:-

```
// StuderdBean.java
package com.naveshit.bean;
import java.io.Serializable;
  public class StuderdBean implements Serializable {

      private int sno;
      private String sname;
      private String Course;
      private double fee;
  // generate getter and setter methods using eclipse
  //Rightclick -> source -> generate getters and setters
    public int getSno() {
      return sno;
    }
     public void setSno(int sno) {
     this.
```

DAO Class design:-

1) JDBC objects variables declaration
2) A constructor for creating Con and Pstmt objects
3) A select methods for executing Selectquery
   i) obtain RS object
   ii) Create one bean instance for each row
   iii) read row values and store in that bean
   iv) store bean instance reference in Collection objects
   v) Repeat above 2,3,4 steps for every row available in Rs object
   vi) return this Collection object

```
package com.naresh.dao;
Package
import java.sql.*;
import com.naresh.bean.*;
  public class StudentDAO {

    privateConnection        con;
    private PreparedStatement pstmt;
    private ResultSet         rs;

     Public StudentDAO() {
      try {
        Class.forName("oracle.jdbc.driver.OracleDriver");

      Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:
                                                    1521:xe", "sims", "sims");

      String squery = "select * from student" +
                      "where course = ?";

      Pstmt = con.prepareStatement(query);
    } catch (ClassNotFoundException e) {
       e.printStackTrace();
    } catch (SQL Exception e) {
       e.printStackTrace();
    }
  } //constructor close
// method for retrieving students information from DB
public ArrayList getStudents(String course) {
  ArrayList studentsList = new ArrayList();
```

```
try {
  pstmt.setString(1, course);
  rs = pstmt.executeQuery();
  while(rs.next()) {
    StudentBean bean = new StudentBean(); - Create this line outside of the while loop and test
    bean.setSno(rs.getInt(1));
    bean.setSname(rs.getString(2));
    bean.setCourse(rs.getString(3));
    bean.setFee(rs.getDouble(4));

    studentslist.add(bean); → Adding this bean obj to collection.
  }
 } catch(SQLException e) {
   e.printStackTrace();
 }
  finally {
    try { → if(rs!=null) {
if(rs!=null) } rs.close();
    } catch(SQLException e) {
      e.printStackTrace();
    }

    return studentslist;
  }
 // @Override
  public void closeObjects() {
    try {
     if(pstmt!=null) { pstmt.close(); }
     if(con!=null) { con.close(); }
    } catch(SQLException e)
      e.printStackTrace();
    }
  }
// FrontOffice.java
package com.nareshit.user;

  import java.util.*;
  import com.nareshit.bean.*;
  import com.nareshit.dao.*;

    public class FrontOffice {
      public static void main(String[] args) {
      x [Scanner scn = new Scanner(System.in);]x
```

```
StudentDAO dao = new StudentDAO();
Scanner scn = new Scanner(System.in);
while(true){
  Sop ln("Enter Course:");
  String course = scn.nextLine();
  ArrayList studentsList = dao.getStudents(course);

    Sop ln ("Sno\t\tSname\t\t\tcourse\t\t\tfee");
    sop ln("_____");
    for(int i=0; i<studentsList.size(); i++){
      StudentBean bean = (StudentBean) studentsList.get(i);
        Sop ln System.out.print(bean.getSno()+"\t");
            System.out.print(bean.getSname()+"\t");
            System.out.print(bean.getCourse()+"\t");
            System.out.print(bean.getFee+"\t");
        }
    }
  }
}
```

Note :- To run this project we must add Ojdbc6.jar file to eclipse buildpath. Oracle driver classes are available in this jar file

Rightclick on project → build path → configure build path → libraries → Add external jars. ⇒ End of project ⇐

(3) Searching an object in Collection :-

→ For searching an object in collection, we must use contains method.

* prototype :- public boolean contains(Object obj)

→ contains method will search an obj in the collection with the given object.
argument

→ For this purpose inside contains method logic, given argument object will be compared with objects available in the Collection. by using java.lang.Object class equals() method

→ If match found, it returns "true" else returns "false".

→ for comparing objs, inside contains method, Equals() method is used.

→ Equals() method compares 'o' objects, using either "reference" or using "data" based on its implementation.

→ If equals method not overridden in added objects class, the two objects are compared using reference.

→ If it is overridden in added (using) object class, then objs are compared using their data.

Note :- In strings & all "wrapper classes" equals method has overridden so, their objects are compared using data.

→ In String Buffer & String Builder classes "equals method" not overridden, so their objs are compared using reference.

Another Ex:- Searching StringBuilder object in AL :-

```
AL al=new AL();
al.add(new StringBuilder("a"));
al.add(new StringBuilder("b"));
StringBuilder sb3 = new StringBuilder("c");
al.add(sb3);
Sopln(al.contains(new StringBuilder("a")); // false
Sopln(al.contains(sb3)); // true
```

→ Since equals() method not OVR in StringBuilder class only same referenced SB objects are found in collection.

→ In above program in first contains() method called we have passed new referenced StringBuilder object so it returns false becoz obj not found with this reference.

→ In 2nd contains method called we passed sb3, same referenced obj so contains method returns true becoz obj found in Collection.

→ Below is the searching algorithm

```
Sopln(al.contains(new StringBuilder("a"));
    sb1.equals(sb1); → executed from SB class
        └false          → not OVR
    sb1.equals(sb2);    → so executed from object class
        └false              → two SB objs are compared
    sb1.equals(sb3);          with reference
        └false              → returns false
```

```
Sopln(al.Contains(sb3));
    sb3.equals(sb1);
        └false
    sb3.equals(sb2);
        └false
    sb3.equals(sb3);
        └true
```

Sample project on searching operation
→ Adding and searching custom objs in

- Employee
- Student
- BankAccount etc. are custom objs

→ Develop a project to add Employee object List type collection. Those added employee objects must be found by using new referenced objects with same data.

(P.T.O)

```
//AddingSearchingCOEmp.java
import java.util.*;
class AddingSearchingCOEmp {
    public static void main(String[] args) {
        ArrayList al=new ArrayList();
        al.add(new Emp(101,"AB",10000,"Java"));
        al.add(new Emp(102,"BB",20000,"Java"));
        al.add(new Emp(103,"MB",30000,".NET"));
        Sopln(al.contains(new Emp(102,"BB",20000,"Java")));
        //equals() not OVR    OVR
        //e1.equals(e1); →false   false
        //e1.equals(e2); →false   true
        //e1.equals(e3); →false
        Sopln(al.contains(new Emp(102,"BB",50000,"Java")));
    }
}
```

matched with this second object hence returns true ⑤

```
//Emp.java
class Emp {
    int eno;
    String ename;
    double sal;
    String dept;
    Emp(int eno, String ename, double sal, String dept) {
        this.eno=eno;
        this.ename=ename;
        this.sal=sal;
        this.dept=dept;
    }
    @Override
    public boolean equals(Object obj) {
        if(obj instanceof Emp) {
            Emp e=(Emp)obj;
            return (this.eno==e.eno) && (this.dept.equals(e.dept));
        }
        return false;
    }
}
```

Run above program:- with the below test cases

Case(1):- remove equals method from Emp class:-

O/P :- false
false

Ans:- Becoz equals method will execute from Object class so Emp objs are compared with reference.

Case(2):- override equals method in Emp class by comparing eno & dept data in both objs

O/P :- true
true

Ans :- Employee objs are compared with data. In equals method

Case(3) :- Add (Condition) an expression for comparing ename.

O/P :- true
false

Summary :- on contains, indexOf & remove methods of WTE :

1) When we see contains method in exam, called on any of the list implemented classes [Vector, Stack, ArrayList, LinkedList], we must check below 2 things.

i) equals method overridden or not in this searching obj class if not overriden compare with reference, [Case(1)]. If overriden compare objs with data [Case(2)].

ii) we must compare only the variables data which is used in equals method comparision [Case(3)].

Q) What will happen if we don't override equals method in our subclass?

Ans:- No Compile-time error, No Runtime error but the problem is the obj added to collection will not find in Collection with new referenced object.

→ It is mandatory to override equals method in a class whose objs will add to collection.

[Sample program on searching operation

Q) write a program for adding BankAccount objs to ArrayList, these objs must find by their account no. and branch.

```
import java.util.*;
Adding n Searching CO BankAccount {
class BankAccount1{
    p s v m(String[] args){

    ArrayList al=new ArrayList();

    al.add(new BankAccount(32u501,"HB",10000,"Ammerpet"));
    al.add(new BankAccount(32u502,"BB",20000,"Ammerpet"));
    al.add(new BankAccount(32u503,"MB",30000,"Reddigudem"));

    Sopln(al.contains(new BankAccount(32u503,"MB",30000,"Reddigudem")));

    //eu.equals(e1);
    //eu.equals(e2);      } Internal calling order
    //eu.equals(e3);

    Sopln(al.contains(new BankAccount(32u503,"LB",50000,"Reddigudem")));
    }
}
```

```
// BankAccount.java
class BankAccount{
    long accno;
    String accHoldername;
    double balance;
    String branch;
BankAccount(long accno, String accHoldername, double balance, String branch){
    this.accno = accno;
    this.accHoldername = accHoldername;
    this.balance = balance;
    this.branch = branch;
}

public boolean equals(Object obj){
    if(obj instanceof BankAccount){
     BankAccount b=(BankAccount) obj;
    return (this.accno=b.accno)&&(this.branch.equals(b.branch));
    }
    return false;
}
```

23/09/15

Q) Difference b/w contains and indexOf() in searching an object:-

```
public boolean contains(Object obj)
public int     indexOf(Object obj)
```

→ Contains method will return boolean value to tell us given object found or not. where as indexOf() will return the index of given object in collection if found.

→ If not found it returns -1.

→ So using contains method we can only find obj in collection, further we can't retrieve it.

→ whereas using indexOf method we can retrieve the obj from collection with the help of get method by passing returned index.

for example:- we have an ArrayList with BankAccount objs. A customer want to know account exist in this branch or not. In this case we must use contains method.

→ customer wanted to know account available or not, if available he wanted to get balance information and last transaction information etc. from this account. In this situation indexOf() and get() method refer project no. 2 in fb.com\javatutorial.

→ literal concept (pooling) is applicable only these 3 classes are String, wrapper class & java.lang. class apart from these classes the obj can be created only by using new keyword.

Eg:- al.add("a");

al.add(new A());

→ when we display ArrayList object it internally calls the toString() method.

→ for searching we can use equals() method.

→ if we can't override toString() method then it can displays classname@hashcode.

(4) Working with remove method:-

→ Remove() method is used for removing object from the collection. we have two overloaded remove methods in List collection.

→ they are

1) public boolean remove(Object obj)
2) public int/boolean removeAt(int index)
   object

→ remove(Object) parameter method will remove given objs matched objs from arraylist.

→ remove method also will use equals method for finding matched objs in collection for the given objs.

→ If obj found, it removes the obj from collection, returns true.

→ If obj not found it doesn't remove any obj, returns false.

→ We must override equals method in the class whose objs we added to the collection.

→ If we don't override equals method obj not found, obj not removed eventhough it is available in collection.

→ remove of index method will remove the given index obj from this collection, and returns this removed obj reference.

Rule:- the passed index value must be in the range of 0 to al.size()-1 if it is less than 0 or greater than size()-1 we will get exception IndexOutOfBoundsException

Q) what is the difference b/w al.remove(5) and al.remove(new Integer(5))?

Ans:- al.remove(5) will remove 5th index obj because the given argument 5 is matched with remove of int parameter method so 5th index obj removed.

→ al.remove(new Integer(5)) will remove integer obj with data 5, by comparing objs in collection from 0 index until (Object) this integer obj found using integer class equals method.

Q) What is the o/p from below program?

Ans:-
```
Al al = at new Al();
al.add("a");
al.add("b");
al.add("a");
al.add("c");
sopln(al);
al.remove(new String("a"));
sopln(al);
```

o/p:- [a, b, a, c]
[b, a, c]

Here equals method executed from string class, data compared, first string obj is removed from collection becos their data is same.

al.remove(new String("a"));
→ sele(searchingelement)
→ sele.equals(e1);
↳ executed from string class
↳ returns true, e1 is removed from al.

Case(2):- Removing Integer objs from ArrayList

```
Al al=new Al();
                    Index
al.add("5");  - 0
al.add(5);    - 1
al.add(6);    - 2
al.add(7);    - 3
al.add(5);    - 4
al.add(8);    - 5
al.add(9);    - 6
al.add(10);   - 7
Sopln(al);
al.remove(5); → 5th index obj (8) is removed
// al.remove(10); → Here 10th index obj should remove, since 10th index
Sopln(al);          not available IOBE raised
al.remove(new Integer(5)); → Integer obj with data 5 is removed,
Sopln(al);                   only 1st occurance.
al.remove(new Integer(10)); → Integer obj with data 10 is removed.
Sopln(al);
```

It is String obj 5

o/p:- [5,5,6,7,5,8,9,10]
[5,5,6,7,5,9,10]
[5,6,7,5,9,10]
[5,6,7,5,9]

Case(3):- Removing character objs

```
Al al = new Al();
al.add('a');
al.add('b');
al.add('c');
Sopln(al);
// al.remove('b'); → 98 index obj should be removed becoz this method
Sopln(al);          call will be matched with remove of index method
al.remove(new Character('b'));  for removing character obj('b') we must create character
Sopln(al);                      class obj explicitly as shown next line
```

o/p:- [a,b,c]
Exception:
[a,c]

Note:- The given argument matching with remove of index only. to the types byte, short, char, int. Becoz these type arguments are matched with remove of int method.

→ So for calling remove method, for removing Byte, Short, Character, Integer objs from Collection we must create their objs explicitly as shown in above case(2) & case(3).

→ For other datatypes long, float, double, boolean we no need to create explicit obj for remove method call becoz these types are not matched with remove of int, they are matched with remove of obj.

Case(iv) :- removing long, double, float, boolean objs.

```
Al al = new Al();
al.add(5);      0
al.add(5L);     1
al.add(7.0);    2
al.add(8.0f);   3
al.add(true);   4
al.add(false);  5
al.add('a');    6
al.add(true);   7
Sopln(al);
```

O/P :- [5, 5, 7.0, 8.0, true, false, a, true] (false: 5th index obj)
[5, 5, 7.0, 8.0, true, a, true]
[5, 7.0, 8.0, true, a, true]
Exception: IOOBE
[5, 7.0, 8.0, true, a, true]
[5, 7.0, 8.0, true, a, true]
[5, 8.0, true, a true]
[5, 8.0, a, true]

```
al.remove(5); → remove of 5 5th index obj removed.
Sopln(al);
al.remove(5L); → remove of long obj with data 5L is removed.
Sopln(al);
// al.remove('a'); → RE: IOOBE beco9 99 index not existed.
// Sopln(al);
al.remove("a"); → String 'a' not available, no element removed.
Sopln(al);
al.remove(8.0); → double type 8.0 not available so no element removed.
Sopln(al);
al.remove(7.0); → double 7.0 is removed.
Sopln(al);
al.remove(true); → first true is removed.
Sopln(al);
```

Case(v) :- Removing StringBuilder objs

```
Al al = new Al();
al.add(new StringBuilder("a"));
al.add(new StringBuilder("b"));
StringBuilder sb=new StringBuilder("c");
al.add(sb3);
Sopln(al);
al.remove(new StringBuilder("a")); → no obj removed from collection
Sopln(al);                          beco9 equals method not overriden
                                    in StringBuilder class so reference are
                                    comparing.
al.remove(sb3); → sb3("c") is removed beco9 same referenced obj.
Sopln(al);
```

O/P :- [a, b, c]
[a, b, c]
[a, b]

Case(vii):- removing userdefined objs (A class objs).

```
Ah al = new Ah();
al.add(new A(5));
al.add(new A(6));
al.add(new A(7));
Sop1n(al);
al.remove(new A(7));
Sop1n(al);
```

O/P:-

24/09/15

//case(i): toString() and equals() not overridden

```
class A {
    int x;
    A(int x) {
        this.x=x;
    }
}
```

→ with this 'A' class 'A' obj not removed from collection and classname@Hashcode is displayed.

//case(2): toString() overridden

```
class A {
    int x;
    A(int x) {
        this.x=x;
    }
    public String toString() {
        return "A("+x+")";
    }
}
```

→ with this changed A class also A obj is not removed from collection. obj data is displayed as below [A(5), A(6), A(7)].

//Case(3): toString() and equals() method overridden

```
class A2
  int x;
  A(int x)2
  this.x=x;
  }

 public boolean equals (Object obj)2
//   Sop(n(this+"and"+obj+"compared");
  if (obj instanceof A)2
     A a=(A)obj;
     return(this.x=a.x);
   }
    return false;
  }
   public String toString()2
    return "A("+x+")";
  }
}
```

→ with this changed code obj is removed from collection

//Case(4): equals() is overridden by returning true literal directly.

```
class A2
  int x;
  A(int x)2
  this.x=x;
 }
  public boolean equals(Object obj)2
   return true;
 }
}
```

→ with this equals() code 1st obj in the collection will be removed not the exact object

→ In above program A(5) is removed with A(7)

(5) Replacing an object in ArrayList:-

→ In list implemented classes V, S, Ah, Lh we can replace one obj with new object. for this purpose we have below method.

public Object set(int index, Object obj)

It replaces given index object with the given new object in collection returns old object reference.

Q) Write a prg with Arraylist with 3 String objects replace and object with its upper case.

Prg:-

```
//ReplaceTest.java
import java.util.*;
class ReplaceTest{
  public static void main(String[] args){
    Arraylist al=new Arraylist();
    al.add("a");
    al.add("b");
    al.add("c");
    Sopln("Original Elem:"+al);
    String s1=(String)al.get(1);
    String s2=s1.toUpperCase();
    al.set(1,s2);
    Sopln("Changed Elem:"+al);
  }
}
```

O/P:-

Original Elem:[a, b, c]

Changed Elem:[a, B, c]

(6) Inserting new objects in AL:-

→ We can insert new obj in list implemented collections V, S, AL, LL using below method.

```
public void add(int index, Object obj)
```

Q) Write a prg with (collection) String objs "A", "B", "C", "D". After adding these String objs insert String obj "B" before "c".

Prg:-

```
//InsertTest.java
import java.util.*;
class InsertTest{
  P S V m(S[] args){
    Arraylist al=new Arraylist();
    al.add("a");
    al.add("c");
    al.add("d");
    Sopln("Original Elem:"+al);
    al.add(1,"b");
    Sopln("Changed Elem:"+al);
  }}
```

O/P:-

Original Elem:[a, c, d]

Changed Elem:[a, b, c, d].

al.add(size, "a") will append this element means stored end of collection.

Rule:- Insertion index value must be in b/w 0, size else it leads to IndexOutOfBoundsException.

```
import java.util.*;
class InsertTestCase {
    p s v m(S[] args) {
    Al al=new Al();
    Sopln(al);
    al.add(0,"b"); → appended
    al.add(1,"c"); → appended.
    Sopln(al);
    al.add(6,"d"); → XRE:
    Sopln(al);
    }
}
```

What is the output from the below program:

```
//RemoveTestCase.java
import java.util.*;
class RemoveTest{
  P S V main(S[] args){
    AL al = new AL();
    for(int i=1; i<10; i++){
      Sopln(i*10);
    }
    sopln(al);
    al.remove(1);
    al.remove(2);
    Sopln(al);
  }
}
```

→ All 4 operations of collection we performed on list implemented classes.

Operations	list methods	Internal method
1) adding	l.add(o); (c)	Not Applicable (NA)
2) counting	l.size() (c)	NA
3) retrieving	l.get(index) (l)	NA
4) searching	l.contains(obj) (c)	obj.equals(ele)
	l.indexof(obj) (l)	obj.equals(ele)
5) removing	l.remove(obj) (c)	obj.equals(ele)
	l.remove(index) (l)	NA
6) replacing	l.set(index, obj) (l)	NA
7) inserting	l.add(index, obj) (l)	NA

What is the o/p from below prg.

```
//RemoveTestCase.java
import java.util.*;
class RemoveTestCase{
  p s v m(S[] args){
  Al al=new Al();
  for(int i=1; i<=10; i++){
    al.add(i*10);
  }
```

10, 20, 30, 40, 50, 60, 70, 80, 90, 100
10, 30, 40, 50

```
        Sopln(al);
        al.remove(1);
        al.remove(2);
        Sopln(al);
    }
}
```

O/P :- [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

[10, 30, 40, 50, 60, 70, 80, 90, 100]

[10, 30, 50, 60, 70, 80, 90, 100]

Set Implemented classes :-

If we want to store only unique elements in collection we must use set implemented classes Hashset, LinkedHashset, Treeset.

→ Hashset doesn't maintain insertion order it will store objs in hashcode based order it willn't maintain insertion order.

→ If we want to store objs in insertion order we must use LinkedHashset class.

→ If we want to store objs in sorting order we must use Treeset.

```
import java.util.*;
class HSLHSTS {
    p s v m(S[] args) {
        HashSet hs = new HashSet();
        LinkedHashSet lhs = new LinkedHashset();
        TreeSet ts = new Treeset();
        hs.add("a");
        hs.add("c");
        hs.add("b");
        hs.add("5");
        hs.add("d");
        Sopln(hs); → [a, b, c, d, 5] → not insertion order

        store(hs);
        Sopln(hs);
        store(lhs);
        Sopln(lhs);
        store(ts);
        Sopln(ts);
    }
    static void store(Set s) {
        s.add("a");
        s.add("c");
        s.add("b");
        s.add("5");
        s.add("d");
    }
}
```

Because collections are Runtime polymorphic classes, we can call add() common to all collections as shown in this store() method

```
lhs.add("a");
lhs.add("c");
lhs.add("b");
lhs.add("5");
lhs.add("d");
Sopln(lhs); → [a, c, b, 5, d] → insertion order

ts.add("a");
ts.add("c");
ts.add("b");
ts.add("5");
ts.add("d");
Sopln(ts); → [5, a, b, c, d] → Sorting order
```

→ Below table shows the 4 collection operations and appropriate method to perform these operations on HashSet & LinkedHashSet.

Operations	HashSet, LinkedHashset	Internal method
1) adding	hs.add(obj); (C)	→ obj.hashCode(), → == operator, → obj.equals(ele)
2) counting	hs.size() (C)	–
3) retrieving	hs.iterator() (C)	–
4) searching	hs.contains(obj) (C)	→ obj.hashcode(), → == operator, → obj.equals(ele)
5) removing	i.remove(obj)	→ obj.hashCode(), → == operator, → obj.equals(ele)
6) replacing & 7) inserting	are not supported because Set dont have index	

HashSet Interview Questions:- (refer questions in previous pages before this list).

Q1-A):- HashSet is used for storing only unique elements without considering storing order.

Q2-A):- Hashtable backed by HashMap instance

when we create HashSet obj, internally HashMap obj is created, elements are stored in this HashMap

Q3-A):- Default capacity - 16

Incremental capacity - Double

load factor - 0.75

load factor is a measurement that specifies when Hashtable capacity should increment, that measurement is 75% i.e after adding 75% entries (elements) its capacity will be increased.

Q4-A):- Homogeneous, heterogeneous and only unique elements.

Q5-A):- null allowed, only one, Becoz duplicate not allowed.

Q6-A):- ~~obj~~ x Hashcode order

Q7-A):- Only sequential

Q8-A): Not synchronized collection

Q9-A): Not ordered collection

Q10-A): hashCode(), == operator, equals() in adding searching & removing operations

LinkedHashset IQs :-

Q1-A) :- When we want to store only unique objs in insertion order we must use LinkedHashset.

Q2-A) :- A Doubly-linked list & Hashtable backed by LinkedHashMap instance

Q3-A) :- Same as Hashset

Q4-A) :- Same as Hashset

Q5-A) :- Same as Hashset

Q6-A) :- Insertion order

Q7-A) :- Sequential order.

Q8-A) same as HS

Q9-A) Same as HS

Q10-A) Same as HS

Q) How Hashset & LinkedHashset can stop duplicate objs ?

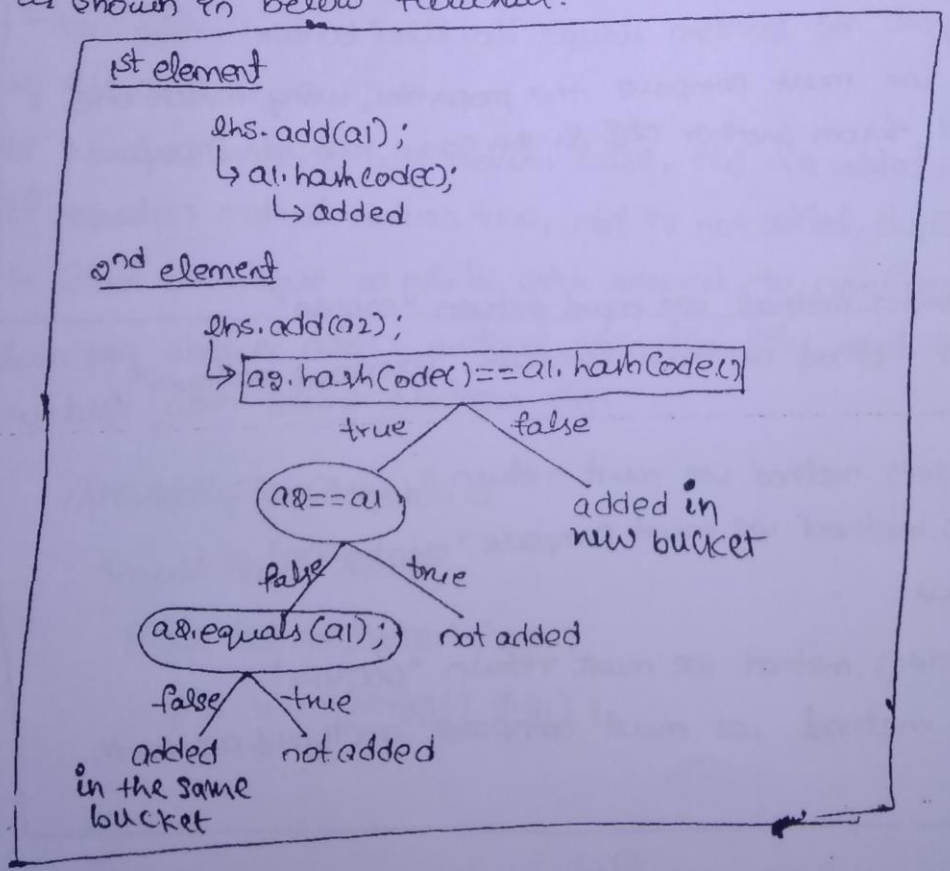
Ans :- By using Hashcode, == operator and equals methods

→ The classes those contains a word hash will use HashCode, == operator & equals methods for stoping duplicate objs and also used for searching and removing correct element from the collection.

The procedure is :-

1) First hashCode() method is called on the current adding obj,
2) Next "==" operator is used
3) Finally equals() method is called

as shown in below flowchart.



25/09/15

→ hashCode() method is used for storing same hashCode objs as one group

→ "==" operator is used for comparing the adding objs with its hashcode objs using reference.

→ equals() methods is used for comparing the adding objs with its hashcode objs using data.

Q) What will happen if we don't override hashCode() and equals() methods in adding objs class?

Ans:- Datawise duplicate objs are added, and also these objs will not found and remove further from collection with new referenced objs.

→ If we don't override hashCode() methods in subclass they will executed from Object class. Then hashcode is generated using reference and objects are compared using reference. Then only same referenced duplicate objs are not added.

→ So we must override hashCode() and equals() methods to adding object class for stopping data wise duplicate objs.

Right procedure for overriding hashCode() & equals() method?

→ From hashCode() method we must return the property, using which one group of objs are differentiated from another group of objs of same class.

→ In equals() method we must compare the properties, using which one obj is differentiated from another obj in the same hashCode group.

For example:

In Student class

→ From hashCode() method we must return "course"

→ In equals() method we must compare "course & rollnum".

In Employee class

→ From hashCode() method we must return "dept".

→ In equals() method we must compare "dept & eno".

In BankAccount class

→ From hashCode() method we must return "accType".

→ In equals() method we must compare "accType & accNum".

→ Examplain HM/HT Internals:

→ add(), put(), compare(), remove() methods algorithm in HS/LHS/HM/LHM/HT collection objs:-

1) All above 5 objs internally uses "hashtable" data structure for storing objs.

2) "hash table" is internally uses another data structure called "bucket".

3) bucket is a collection of elements there have same hashCode. this bucket is also a collection obj.

4) bucket is used for storing same hashCode objs as one group. so that comparing & searching will be fast bcoz currently adding objs will be compared only with this objs hashcode group of elements.

5) new bucked will be created for every new hashcode obj. this obj will add to this bucket directly without any comparisons.

6) If bucket is alaeady available with this currently adding obj's hashCode. then add() method will check whether this obj is unique or duplicate by using "==" operator and equals() method.

7) If "==" operator returns "true", this obj is reference wise duplicate obj, not added to collection.

8) If "==" operator returns false. adding obj is unique based on reference. but it may be duplicate based on state.

Then add() method will call equals method for comparing objs using its content.

9) If equals() method also returns false, obj is added, unique on data wise.

10) If equals() method return true, obj is not added, duplicate on data wise.

Note:- Read all above 10 points with respect to previous page flowchart

Q) below prg shows add() method functionality and hashmap memory structure with String & Integer objs.

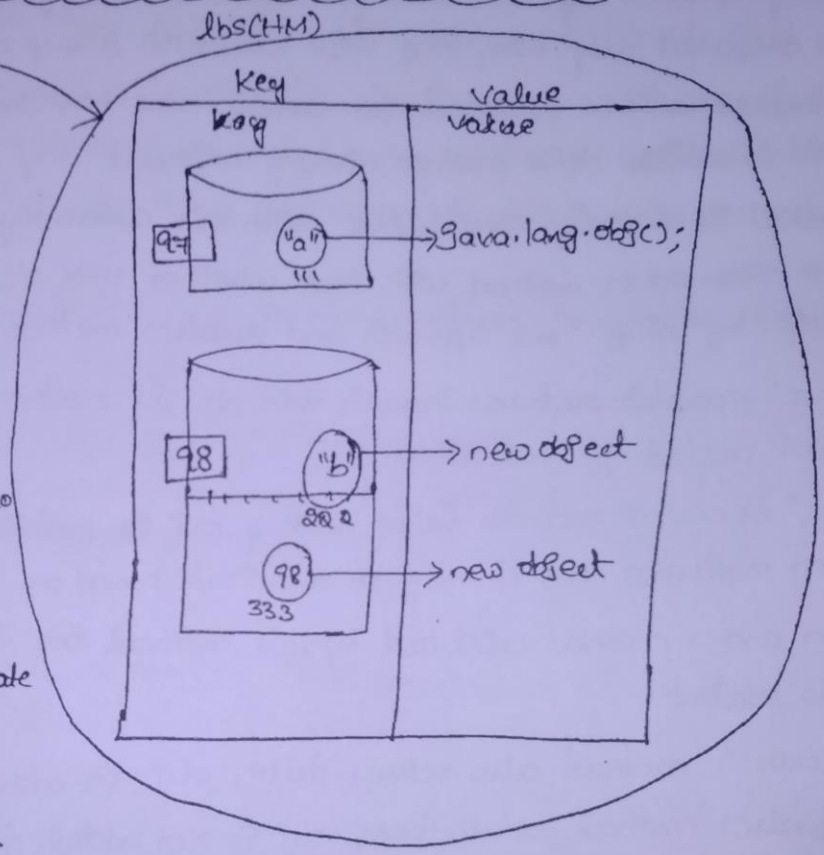
Ans:-

```
//LHSAddingTestCases.java

import java.util.*;

Class LHSAddingTestCases {

    p s v m(String[] args) {
```

// case(1): adding string & Integer objs.

```
LinkedHashSet lhs1 = new LinkedHashSet();
lhs1.add("a");          ← Added in new bucket
lhs1.add("a");          ← Not added, duplicate(ref)
lhs1.add(new String("a")); ← Not added, duplicate (data)
lhs1.add("b");          ← added in new bucket
lhs1.add(98);           ← added in "b" bucket
Sop ln(lhs1); → [a, b, 98]
```

Execution flow as per the 10 points explained in previous page




```
lhs1.add("a");
  ↳ s1.hashCode();
      ↳ 97 → bucket not found
      ↳ new bucket created
      "a" added in this obj. bucket

lhs1.add("a");
  ↳ s2.hashCode();
      ↳ 97 → bucket found
      ↳ comparision done with objs exist in this bucket
  ↳ (s2 == s1) true, not added, same reference obj, duplicate

lhs1.add(new String("a"))
  ↳ s3.hashCode()
      ↳ 98 → bucket found.
  ↳ (s3 == s1) false, unique, reference wise
  ↳ s3.equals(s1) true, duplicate data wise

lhs1.add("b");
  ↳ s4.hashCode()
      ↳ 98 → bucket not found.
      ↳ new bucket created.

lhs1.add(98);
  ↳ i0.hashCode(); → 98
      bucket found.
  ↳ i0 == s4 → false, unique reference wise
  ↳ i0.equals(s4) → false, unique type wise
```

// Case(2): adding StringBuilder objs [In SB class hc() & equ() methods are not overridden]

```
LinkedHashSet lhs2 = new LinkedHashSet()
StringBuilder sb1 = new StringBuilder("a");
"      sb2 = "      "      ("a");
"      sb3 = "      "      ("b");
"      sb4 = "      "      ("c");
StringBuilder sb5 = sb4;
lhs2.add(sb1); ← added in new bucket
   // → sb1.hashCode() → 31168322 → new object bucket
lhs2.add(sb2); ← added in another newBucket
   // → sb2.hashCode() → 1422537 2 → new bucket
lhs2.add(sb3); ← added in another new Bucket
   // → sb3.hashCode() → 5u3363u → new bucket
lhs2.add(sb4); ← added in another new bucket
   // → sb4.hashCode() → 2u30287 → new bucket
lhs2.add(sb5) ← not added be cause same ref obj
   // → sb5.hashCode() → 2u30287 → bucket found
   // → sb5==sb4 → true, not added (same ref. obj).
sopln(lhs2);
```

// Case(3): adding custom objs to set.

→ For adding custom objs [our own class objs], we must override hashcode & equals() methods in our class.

→ Then only data wise duplicate objs are not added, further added obj will found in Collection & removed from collection with new referenced obj.

→ we have 6th cases in adding & removing obj with equals() & hashcode() methods.

From the Below Test cases find out

1) How many objs are added
2) " buckets are created
3) " objs are removed.

Given

```
import java.util.*;
class ExCollection {
    p s v m(S[] args) {
    LinkedHashSet lhs = new LinkedHashSet();
     lhs.add(new Ex(5,6));
     lhs.add(new Ex(5,6));

    Ex e3 = new Ex(5,6);
    Ex e4 = new Ex(5,6);
    Ex e5 = new Ex(6,5);
    Ex e6 = new Ex(7,8);
    Ex e7 = e6;

    lhs.add(e3);
    lhs.add(e4);
    lhs.add(e5);
    lhs.add(e6);
    lhs.add(e7);
```

verify hashCode() and equals() method overridden (or) not

```
    Sopln("No.of objs added:" + lhs.size());
    Sopln(lhs);
    Sopln();
    Sopln("It is removed:" + lhs.remove(new Ex(6,5));
    Sopln("It is removed:" + lhs.remove(e7));
    Sopln("No of objs after remove:" + lhs.size());
    Sopln(lhs);
  }
}
```

// below Ex class

```
class Ex {
   int x,y
    Ex(int x, int y) {
      this.x=x;
      this.y=y;
    }
    public String toString() {
      return "Ex("+x+","+y+")";
    }
}
```

Find o/p in below cases:-

Case(i):- Both hashCode() & equals() methods are not overriden in Ex class.

Case(ii):- Only Equals() hashcode() method is overriden by returning 5;

case(iii):- Only Equals() method is overriden by either true/false;

Case(iv):- Hashcode() method is overiden by returning 5; also equals() method is overriden by returning true;

Case(v):- In above case return false from equals() method.

Case(vi):- hashCode() method is overriden by returning x+y & and equals() method is Overraiden by comparing data.

Case(i)-A):- 6 buckets are created (all objs have different references)
6 objs are added in different buckets
1 obj is removed i.e; e6.

Case(ii)-A):- 1 bucket is created(hash code() returning same val 5)
6 objs are added in the same bucket
1 obj is removed i.e, e6

Case(iii)-A):- 6 buckets are created (different objs)
6 objs are added in different buckets
1 obj is removed i.e; e6

Case(iv)-A):- 1 bucket is created (hashcode returning same value).
1 object is added i.e; first added object
1st object is removed irrespective of reference & state.

Case v)-A) 1 bucket is created
6 objs are added in the same bucket
1 obj is remove i.e; e6.

Case vi)-A):- 2 buckets are created
Ex(5,6) Ex(6,5) Ex(7,8) are added
2 objs are removed, i.e; Ex(6,5) Ex(7,8)

Contract in Overriding hashCode() & Equals() method:-

→ we must override hashcode() & Equals() method in subclass to satisfy below Contract.

distinct Case:-

hc is same ⇒ Equals() returns t/f

* hc is diff ⇒ Equals() should return false

Eq() return true :-

* Eq() compare '2' objs return true ⇒ hc should be same.

Eq() compare 2 objs return false ⇒ hc() should be either same or differed.

⇒ It is rule should be override Equals() & hashcode() methods in every class.

Ans :- No, we must override hashcode() & Equals() methods in the classes whose objs were adding to Collection.

For high performance :-

x(generate different hashcode per each object.)x

generate different hashcode per each object, so no comparisions, object is directly added in different new bucket.

But According to business :-

we must generate same hashcode for related objs, and different hashcode for unrelated objs of the same class. so that searching element is fast minimum comparisions.

So According to business :-

Develop a project for adding Student objs to lhs Collections and further search and remove the given student obj from L.H.S.

Note :- you must built Student class, for storing its objects in the pattern as per real time school, classroom based arrangement. that means, student obj in LHS should group on course wise, should not allow two students objs those have same course & rollnum.

So From hashcode() method we must return course & in Equals() method we must compare course & rollnum;

Coding :-

Number of class : 3

1. student class
2. School class
3. Course Map class.

In student class: we must

1) create fields for storing students data
2) create a param constructor for initializing
3) create HashMap obj for storing Course name and its nums
(this program will be in CourseMap class)
4) Override hashCode() & equals() methods.
5) From hashCode() method we must return course num
6) In equals() method we must compare rollnum & course.

In School class:- we must

1) Define main method
2) In main method create HS object
3) create student objs with unique & duplicate content.
4) Add student objs to HS.

In CourseMap class:- we must

1) create HashMap obj with private static ref variable.
2) Define static {} inside static {} initialized HashMap obj with coursename & its number entries.
3) Define getCourseNum() method for retrieving given course number

project:-

```
package com.nareshit.helper;

public class CourseMap {
    private static HashMap courseMap = new HashMap();
// static Block
    static {
        courseMap.put("CRT",1);
        courseMap.put("C",2);
        courseMap.put("CORE JAVA",3);
        courseMap.put("Adv JAVA",4);
        courseMap.put("ORACLE",5);
    }
    public int getCourseNum(String course) {
        return (Integer) courseMap.get(course.toUpperCase());
    }
}
```

```
package com.nareshit.bean;
public class Student {
    private int rollnum;
    private String sname;
    private String course;
    private double fee;

public Student(int sno, String sname, String course, double fee) {
    this.sno = sno;
    this.sname = sname;
    this.course = course;
    this.fee = fee;
}
```

```
@override
public int hashCode() {
    return courseMap.getCourseNum(course);
}
```

check previous page for logic

```
public static void main(String[] args) {
}
```

```
@override
public boolean equals(Object obj) {
    if (obj instanceof student) {
        student s = (student) obj;
        return (this.course.equals(s.course)) && this.rollnum == s.rollnum);
    }
    return false;
}
@override
public String toString() {
    return "InStudent(" + rollnum + ", " + sname + ", " + course + ", " + fee + ");
}
}
```

```
package com.nareshit.user;
public class School {
    p s v m(S[] args) {
        LinkedHashSet lhs = new LinkedHashSet();
```

```
lhs.add(new Student(101,"Hari","CoreJava",1000);
lhs.add(new Student(102,"Balayya","CoreJava",1000);
        :
lhs.add(new Student(101,'Hari","Oracle",1000);
lhs.add(  "      "      "      "      "      );
lhs.add(new Student(101,"Balayya","Oracle",1000);
lhs.add(new Student(102,'Balayya","CoreJava",1000);
}
  Sop.ln(lhs);
  lhs.remove(new Student(102,"Balayya","Core Java",1000);
  Sopln(lhs);
}
```

2nd project:-

Project title:- using collection object as mini database

```
//AccountBean.java
package com.naresh.bean;
public class AccountBean{
 private long accNum;
 private String accHName;
 private double balance;
 private String accType;
 public long getAccNum(){
  return accNum;
}
 public void setAccNum(long accNum){
  this.accNum=accNum;
}
 public String getAccHName(){
  return accHName;
 }
 public void setAccHName(String accHName){
  this.accHName=accHName;
 }
 public double getBalance(){
  return balance;
 }
```

```
public void setBalance (double balance) {
  this.balance = balance;
 }
   public String getAccType() {
    return accType;
   }
    public void setAccType(String accType) {
     this.accType = accType;
    }
    @Override
    public boolean equals (Object obj) {
    if(obj instanceof AccountBean) {
     AccountBean bean = (AccountBean) obj;
     return (this.accNum == bean.accNum)
                    &&
            (this.accType.equals(bean.accType));
    }
      return false;
     }
  }
```

```
//CollectionDB.java
package com.nareshit.db;
import java.util.ArrayList;
import class Collection
import com.nareshit.bean.AccountBean;
public class CollectionDB {
private ArrayList accList = new ArrayList();
public void addAccount (AccountBean bean) {
 accsList.add(bean);
}

 public AccountBean getAccount(long AccNum, String accType) {
 AccountBean resAccBean=null;
 AccountBean searchBean = new AccountBean();
 searchBean.setAccNum(accNum);
 SearchBean.setAccType(accType);
 int index = accsList.indexOf(searchBean);
 if(index!=-1) {
 resAccBean = (AccountBean) accsList.get(index);
 }
 return resAccBean;
 }
```

```
// Clerk.java
package com.naseehit.user;
import java.util.Scanner;
import com.naseehit.bean.AccountBean;
import com.naseehit.db.CollectionDB;
public class Clerk {
public static void main(String[] args) {
Scanner scn = new Scanner(System.in);
CollectionDB db = new CollectionDB();
while(true){
Sopln("\n Choose option");
Sopln("1. Create Account");
Sopln("2. Get Balance");
Sopln("3. Exit");
Sopln("Enter option");
int option = scn.nextInt();
scn.nextLine();
switch(option) {
case(1):
AccountBean bean = new AccountBean();
Sopln("accNum:");
bean.setAccNum(scn.nextLong());
scn.nextLine();
Sopln("accHName");
bean.setAccHName(scn.nextLine());
sop("balance");
bean.setBalance(scn.nextDouble());
scn.nextLine();
SOP("accType");
bean.setAccType(scn.nextLine());
db.addAccount(bean);
Sopln("Account created");
break;
```

Case(2):-

```
Sop("accNum");
long accNum = scn.nextLong();
scn.nextLine();
Sop("accType");
String accType = scn.nextLine();
bean = db.getAccount(accNum, accType);
if(bean!=null){
Sopln("Current Balance:"+bean.getBalance());
} else{
Sopln("Account Number is wrong");
}
break;
```

Case(3):-

```
break loop;
default:
Sopln("Invalid option");
}
} //while(true)
Sopln("\n*****Thank you, Visit again *****");
}
}
```

28/09/15

Tree Set; Comparable, Comparator:-

→ Treeset is used for storing elements in sorting order either in ascending or descending order based on the adding objs

→ Treeset doesn't have any sorting order this sorting order should be define by the class whose objs are adding to treeset.

→ the default sorting order of Treeset is that objs those are adding to the default sorting order of object those are adding to treeset.

→ For storing objs in sorting order treeset internally uses either of the below two interfaces

1) Comparable (or)
2) Comparator.

→ Treeset uses these 2 interfaces becoz the objs sorting logic will be changed from one type of obj to another type of obj so the sorting order logic should be supplied by adding object class programmer. Through interface we can force subclass programmer to implement a method logic.

→ TS.add() method internally calls these interfaces methods. Based on the value returned from these methods, TS will sort objs.

→ add() method will sort objs based on return value as below.

- If return number is 0,
  - ↳ element is not added.
- If return number is -ve
  - ↳ element is added LEFT to ts-ele.
- If return number is +ve
  - ↳ element is added RIGHT to ts-ele.

If there is already element existed RIGHT to ts-ele, Comparision is repeated with the next element & then decides the position of currently adding element.

Adding Custom objects to Treeset with Comparable Interface:-

→ Comparable interface has a method CompareTo(Object) for defining / implementing objs comparision logic

```
public int compareTo(Object obj)
```

→ For adding our class objs to TS, we must also derive our class from Comparable interface and should implement CompareTo() method with sorting order logic.

→ Inside TS.add() method given obj will be cast to Comparable interface.

→ Algorithm / procedure for adding Custom objs to TS:

1) we must derive class from Comparable interface.

2) we must implement compareTo() method.

Inside compareTo() method sorting logic. we must use "-" operator for Comparing objs to generate a number.

3) When we add objs to TS class, inside add() method

i) Given obj will be cast to Comparable type

ii) Invokes compareTo() method using this obj by passing already added objs.

iii) then this adding obj will be stored in TS per the value returned from compareTo() method as shown above.

→ Below diagram shows sample code of TS.add() method, and compareTo() method logic for storing Emp obj in employee.

```
class TS {
public boolean add(Object obj){
Comparable cmp=(Comparable)obj;
if(this.isEmpty()) {
    first obj added
} else {
int pos = cmp.compareTo(ele);
if(pos<0) {
    obj added LEFT to ele
    break;
} else if(pos>0) {
    obj added RIGHT to ele
}
```

e2, e1

0/+ve/-ve

```
class Employee implements Comparable {
    int eno;
    String ename;
    String dept;
    double sal;
    double exp;
public int compareTo(Object o) {
    Emp e = (Emp)o;
    double ed = this.exp - e.exp;
    if(ed==0) {
        return this.eno - e.eno;
    } else if (ed<0) {
        return -5;
    } else {
        return 5;
    }
}
```

Assignment

① Date & Time working functionalities in Java

29/09/15

Note :- String class, all wrapper classes are subclasses of Comparable interface so we can add these classes objs in treeset directly. In these classes compareTo() method has overriden to store their objs in ascending order [logic will be current obj - Argument obj].

→ String buffer and String builder classes are not subclasses of Comparable interface so we can't add these class objs in treeset it leads to CCE

Below program shows adding String objs to treeset.

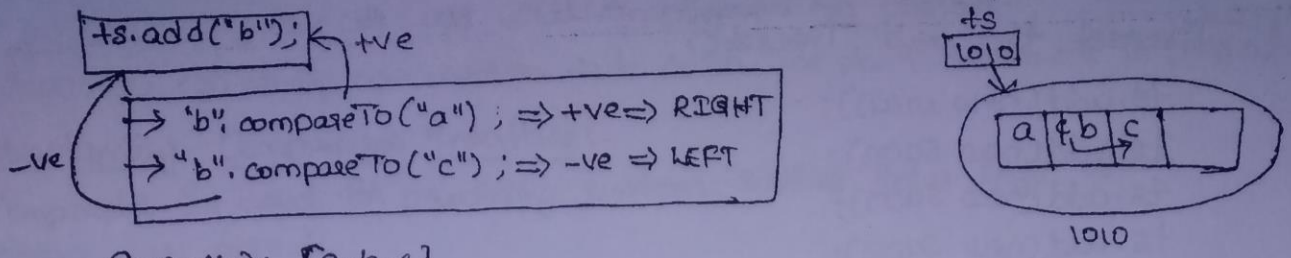
```
Class TSwithStringObj {
    P S V main(String[] args) {
        TS ts = new TS();
        ts.add("a");
        ts.add("c");
```

→ "c".compareTo("a");

"c" - "a"

99 - 97

+ve



SOPln(ts); [a,b,c]

}

}

Adding custom objs to treeset:-

Step(1):- Implement class declaring from Comparable interface

Step(2):- Implement compareTo() method by returning int value 0 to -ve number & +ve number.

→ we have 5 test cases in implementing compareTO() method

Case(i):- return ZERO directly

Only first element is stored

Case(ii):- return +ve number directly

All objs are stored in insertion order including duplicates

Case(iii):- return -ve number directly

All elements are added in reverse insertion order including duplicates.

Case(iv):- return (this.field - arg.field)

Only unique elements are added in ascending order.

Case(v):- return (arg.field - this.field)

Only unique elements are added in descending order.

Rule in implementing compareTO() method:-

→ we shouldn't use "instanceof" operator in condition for casting argument obj to current class type. becoz treeset shouldn't allow heterogeneous object.

→ Below Example shows adding custom obj class to TS find o/p for all above 5 cases.

```
//TS with Cust Obj Java
import java.util.*;
class TSwithCustObj2
{
    P S V main(String[] args)2
```

```
TreeSet ts = new TreeSet();
    ts.add(new Sa(4));
    ts.add(new Sa(3));
    ts.add(new Sa(5));
    ts.add(new Sa(2));
    ts.add(new Sa(2));
     Sopln(ts);
  }
}
```

//Sa.java

```
class Sa implements Comparable?
    int x;
     Sa(int x)?
      this.x=x;
    }
      public String toString()?
        return "Sa["+x+"]";
    }
```

```
public int compareTo(Object obj)?
  //Insert code here
}
}
```

(1) From the below cases identify how many objs are added to TS, and in which order?

Case(1) :- return directly ZERO, i.e; "return 0".

Ans:- [Sa(4)]

Case(2) :- return directly 5, i.e; "return 5;"

O/P:- [Sa(4), Sa(3), Sa(5), Sa(2), Sa(2)]

Case(3) :- return directly -5, i.e; "return -5;"

O/P :- [Sa(2), Sa(2), Sa(5), Sa(3), Sa(4)]

Case(4) :- (or) AO-CO state is returned i.e; return this.x-s.x;

O/P:- [Sa(3), Sa(3), Sa(4), Sa(5)]

Case(5):- AO-CO state is returned i.e; return s.x-this.x;

O/P :- [Sa(5), Sa(4), Sa(3), Sa(2)]

Note :- Case(4) & Case(5) we will implement in project.
→ Case(1), (2), (3) they are written test cases we don't implement in project.

Understanding Comparator interface :-
→ Comparator is used for providing custom sorting order logic of an existing class objects.
→ Using comparator we can provide three logics
  i) for storing comparable objs in their reverse natural sorting order
  ii) for storing comparable objs on different property
  iii) for storing non-comparable objs.
→ Assume Employee class has sorting order logic for storing its objs on name based ascending order. This sorting logic is called natural sorting order for providing natural sorting order Employee class should be subclass of Comparable interface then Employee class is called Comparable obj.
→ If you want to store Employee obj name wise descending order or want to store different property wise let us say on eid/sal/joining date, we must develop custom comparator using java.util.Comparator interface.
→ The subclass of Comparator interface is called CustomComparator. This CC objects should pass to TS obj. Then Employee objs are stored as per CC sorting order.
Rule :- So, for adding an object to treeset
  1) either it should be a subclass of Comparable interface (or)
  2) It should be supplied with custom Comparator.
     else TS throws CCE.
→ If we want to store string & all wrapper class obj in descending order we must develop CustomComparator.

Storing String objs in descending order in TS using customComparator :-
Procedure :-
Step(1) :- Define a class from Comparator interface.
Step(2) :- Implement compare(o1,o2) method in this method cast parameters to the class whose objs we want to compare. Then return result value. this case cast to String class.
Step(3) :- pass this customComparator obj as argument to TS object Constructor. Then add method will compare adding object with tseles using Compare method of this CC.

Compare() will be called internally as below

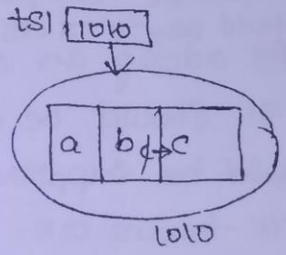
Customcomparator.Compare(addingobject, tsele)

Code:- //StringComparator.java

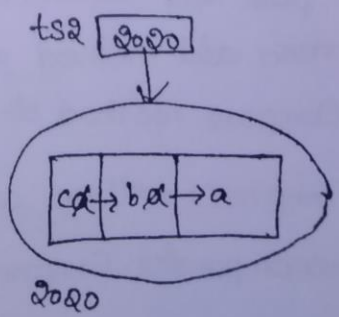
```
class StringComparator implements java.util.Comparator{
    public int compare(Object o1, Object o2){
        String s1 = (String)o1;
        String s2 = (String)o2;
        return s2.compareTo(s1);
    }
}
```

→ Below program shows adding String objs with NSO (Natural sorting order), CSO (custom sorting side).

```
// TSwithStringObjs.java
import java.util.*;
class TSwithStringObjs{
    p s v main(String[] args){
        Treeset ts1 = new Treeset();
        ts1.add("a");
        ts1.add("c");
            // "c".compareTo("a");
        ts1.add("b");
            // "b".compareTo("a");
            // "b".compareTo("c");
        sopln(ts1);  // [a, b, c]
    }
    Treeset ts2 = new Treeset(new StringComparator());
    ts2.add("a");
    ts2.add("c");
        // sc.compare("c","a"); a-c => 97-99.
            // executed from sc class
    ts2.add("b");
        // sc.compare("b","c"); c-b => 99-98
        // sc.compare("b","a"); a-b => 97-98
        sopln(ts2); // [c, b, a]
    }
}
```



// executed from String class



→ Below prg shows defining custom Comparator for storing Non-Comparable objs for example StringBuilder.

```
//SBComparator.java
class SBComparator implements java.util.Comparator{
    public int compare(Object o1, Object o2){
        StringBuilder sb2 = (StringBuilder)o1;
        StringBuilder sb1 = (StringBuilder)o2;
        String s1 = sb1.toString();
        String s2 = sb2.toString();
        return s1.compareTo(s2);
    }
}
```

→ Comparable method is not available in StringBuilder class. Comparing StringBuilder data is equal to Comparing String data so converted to String

```
//TSwithSBobjs.java
import java.util.*;
class TSwithSBobjs{
    P S V M(S[] args){
//  TreeSet ts = new TreeSet(); → this obj causes CCE
    TreeSet ts = new TreeSet(new SBComparator());
    ts.add(new StringBuilder("a"));
    ts.add(new StringBuilder("c"));
        //sbc.compare(sb2,sb1);
    ts.add(new StringBuilder("b"));
        //sbc.compare(sb3,sb1);
        //sbc.compare(sb3,sb2);
        Sopln(ts); → [a,b,c]
    }
}
```

30/09/15

Difference b/w Comparable, Comparator interface:-

D.	Comparable	Comparator
→ 1)	It is used for developing natural sorting comparison logic.	1) It is used for developing custom sorting comparison logic.
2)	It has CompareTo(Object) method.	2) It has compare(Object,Object) and equals(Object) methods.

3) It is available in java.lang package

4) Treeset use Comparable.compareTo() method if its object is created using non-parameterized constructor.

5) TS calls compareTo() method using adding object by passing already we added object in TS.

Ex:- e2.compareTo(e1);

3) It is available in java.util package

4) TS uses Comparator.compare() method if its object is created using Comparator parameter constructor by passing custom Comparator object.

5) TS calls compare() method using the registered Comparator obj by passing current adding obj as first argument and already added element as second argument.

Ex:- Cmpr.compare(e2,e1);

Q) Why compareTo() has one parameter and compare() has two parameters?

Ans:- CompareTo() method will be implemented in the same adding obj class, so among two objs if one obj is send as current obj, another obj is sent as argument object. Hence compareTo() method has only one parameter.

→ compare() method will be implemented in the other class, not in adding object class. So the two objs must be passed as arguments. Hence Compare() method has two parameters.

Q) Develop a project for storing Student object in sorting order class wise, within the class sort students in rollno wise if student rollnumbered class is equal to an existing student rollno class don't add finally display all students details.

→ In this project we must develop 3 classes.

1) CourseMap.java
2) Student.java
3) School.java.

```
//courseMap.java
   Collect code from previous project.

//student.java
package com.naresh.it.bean;
import com.naresh.it.helper.CourseMap;
   Public class Student implements Comparable {
      private int rollnum;
      private String sname;
      private string course;
      private double fee;
      private double height;
```

```
Public Student(int rollnum, String sname, String course, double fee, double height){
    this.rollnum = rollnum;
    this.sname = sname;
    this.course = course;
    this.fee = fee;
    this.height = height;
}

@Override
  Public int compareTo(Object o){
    Student s = (Student)o;
    int foCourseNum = CourseMap.getCourseNum(this.course);
    int soCourseNum = CourseMap.getCourseNum(s.course);
    int courseDiff = foCourseNum - soCourseNum;
    if(courseDiff!=0){
      return courseDiff;
    }else{
      return this.rollnum - s.rollnum;
    }
  }

@Override
  public String toString(){
    return "\nStudent("+rollnum+","+name+","+course+","+fee+","+height+");";
  }
}

//School.java.

package com.naveshit.user;
import java.util.TreeSet;
import com.naveshit.bean.Student;
public class School{
  public static void main(String[] args){

    TreeSet ts=new TreeSet();
    ts.add(new Student(101,"Hari","corejava",1000,6));
    ts.add(new Student(101,"Hari","Oracle",1000,9));
    ts.add(new Student(101,"Hari","CoreJava",1000,6));
    ts.add(new Student(104,"sampu","CoreJava",1000,5.9));
    ts.add(new Student(103,"Balayya","CoreJava",1000,5.9));
    ts.add(new Student(102,"Hari","Oracle",1000,5);
    ts.add(new Student(103,"Mahesh","CoreJava",1000,6.1));
    ts.add(new Student(101,"Balayya","Oracle",1000,8));
```

Sopln(ts);

Sopln(); → ts.remove(new student(101,"Hasi","oracle"coreJava",1000,6));
Sopln(ts);

→ Create a custom Comparator for storing student objs in the reverse of above natural sorting order.

Stucls.

```
package com.nareshit.customcmpr;
import java.util.Comparator;
public class StudentRNSOComparator implements Comparator{
@override
public int compare(Object o1, Object o2){
  Student s1=(Student)o1;
  Student s2=(Student)o2;
return s2.compareTo(s1);
}
}
```

→ for testing this comparator in school.java pass this class obj to TreeSet constructor as shown below.

```
TreeSet ts = new TreeSet(new StudentRNSOComparator());
```

→ Create customComparator to store student objs on height wise.
2 students can have same height if their height is same sort them on rollnum wise

```
package com.nareshit.Customcmpr;
import java.util.Comparator;
import com.nareshit.bean.Student;
import com.nareshit.helper.CourseMap;
public class StudentHeightComparator implements Comparator{
@override
public int compare(Object o1, Object o2){
  Student s1=(Student)o1;
  Student s2=(Student)o2;
  int soCourseNum = CourseMap.getCourseNum(s1.getCourse());
  int soCourseNum = CourseMap.getCourseNum(s2.getCourse());
```

```
int courseDiff = FoCourseNum - SoCourseNum;
if(courseDiff!=0){
 return courseDiff;
}else{
  int rollnumDiff=S1.getRollnum()-S2.getRollnum();
  if(rollnumDiff!=0){
    return 0;
    }else{
     double heightDiff=S1.getHeight()-S2.getHeight();
     if(heightDiff!=0){
       if(heightDiff<0){
         return -5;
        }else{
          return 5;
        }
     }else{
       return rollnumDiff;
     }
   }
 }
}
```

→ In school class replace Treeset obj by passing this obj Comparator obj as shown below.

```
TreeSet ts = new TreeSet(new StudentHeightComparator());
```

→ Below table shows TS operations & mapping methods.

Operations	TreeSet	Internal method
1) adding	ts.add(obj);	→ obj.compareTo(ele) → cmpr.compare(obj, ele)
2) counting	ts.size().	—
3) retrieving	ts.iterator()	—
4) searching	ts.contains(obj)	→ obj.CompareTo(ele) → cmpr.Compare(obj, ele)
5) removing	ts.remove(obj)	→ obj.compareTo(ele) → cmpr.compare(obj, ele)
6) replacing		
7) Inserting		

TS FAQs :-

Q1-A) :- for storing objs in sorting order Red-

Q2-A) :- Red-black-tree

Q3-A) :- default capacity = 1
increment " = 1

Q4-A) :- homogeneous, unique, comparable type objs allowed and homogeneous, unique, non-comparable objs with Custom Comparator.

Q5-A) :- null is not allowed, it leads to NPE.
Solo null is allowed only till Java 6.

Q6-A) :- Given objs sorting order.

Q7-A) :- sequential.

Q-8A) NOT Synchronized

Q-9A) ordered collection, sorting order

Q-10A) Comparable.compareTo(-)
(or)
Comparator.compare(-,-)

01/10/15

Q) What are the methods we must override in subclass for adding its objs in anyone of the collection and further searching & removing from Collection?

Ay :- 3 methods.

1) equals()
2) Hashcode()
3) CompareTo()

→ Hashcode() & equals() method for hashset & linkedhashset and compareTo() for Treeset list all u subclasses use only equals methods for searching & removing elements.

→ In previous project in student class we have overriden in only compareTo() method, Copy Hashcode() & equals() methods from project-3 into this class. Then we can say it is fullfledged implemented class for adding, searching & removing its objs from any one of the collection including map collection also.

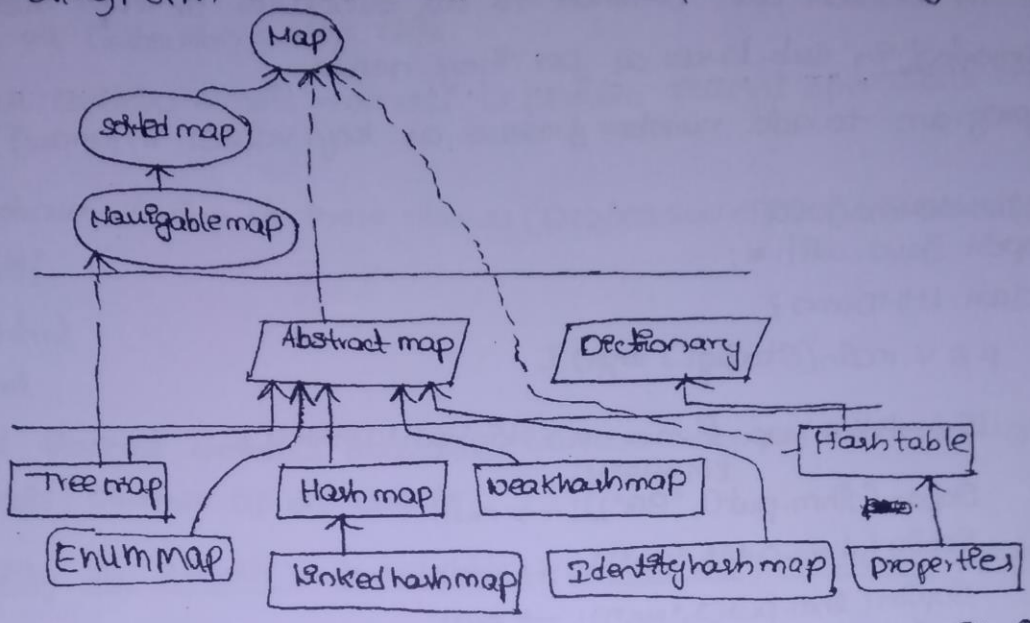
Note :-) In string and all wrapper classes equals(), hashcode(), compareTo() all these 3 methods are overriden so we can add these classes objs in any of the collection including map.

→ In StringBuffer and in StringBuilder classes hashCode(), equals() & compareTo() all three methods are not overridden. Hence we can not add and can not find StringBuffer and StringBuilder objects in Set, Map, SortedSet and SortedMap Collections.

Working with Map class:-

→ When we want store objs in key, value pair format we must choose map type collections.

→ Below diagram shows Map subclasses hierarchy



→ Interface map is the root interface of all map type collection. It is not subinterface of Collection bcoz both interfaces functionalities are different.

→ Map contains only methods related to general operations adding, searching, removing, retrieving, counting.

→ the methods required to store and retrieve objs on sorting order are given in sortedmap & navigablemap.

Map methods:-

1) boolean isEmpty()
2) Object put(Object key, Object value)
   ↓
   Why return type?
3) void putAll(Map m)
4) Object remove(Object key)
5) void clear()
6) boolean containsKey(Object key)
7) Boolean containsValue(Object value)
8) int size()
9) Object get(Object key)
10) Collection values()
11) Set keySet()
12) Set entrySet()
13) boolean equals(Object o)
14) int hashCode()

Q) Why put() method return type is object?

Ans:- If key already existed in this map it replaces existed value object with this new value obj, returns old value object.

→ All above 14 methods are common to all subclasses of map. These methods are implemented in subclasses as per they needs.

Q) Write a program to add number & name as key, values in (array) LHM collection?

Ans:-

```
//LHMDemo.java
import java.util.*;
class LHMDemo {
  p s v main(String[] args) {

  LinkedHashMap lhm = new LinkedHashMap();
                          (5,"MB");
  Sopln(lhm.put(1,"BB")); → null
  Sopln(lhm.put(4,"PB")); → null
  Sopln(lhm.put(3,"MB")); → null
  "     "     "  (1,"CB"); → BB
  "     "     "  (2,"BB"); → null
/*Object vObj = lhm.put(1,"CB"); → Sopln(lhm.put(5,"MB"));
  Sopln(vObj);

  vObj=lhm.put(2,vObj);
  Sopln(vObj); */

  Sopln(lhm); → {1=CB, 4=PB, 3=MB, 2=BB, 5=MB}
  }
}
```

Map object memory structure:-

Key	Value
1	~~BB~~ CB
4	PB
3	MB ←
2	BB
~~5~~	MB ←

Note:- In a map object we can't store duplicate keys but we can store duplicate values. If we try to duplicate key old value will be replaced with new value of this key.

Q) What is the use of Collections and Arrays?

Ans:- These two classes are called utility classes.

→ These two classes contains only static methods

→ Collections class contains several static methods to perform operations common to all Collection & Map objs.

→ Arrays class contains static methods to perform several operations on Array objs.

→ open API documentation of these classes (DVD/03SUN API Docs/ Java SE 1.6 docs (01)/ api/java/util/

/Collections.html

/Arrays.html

then find several methods available in these classes, call them by passing Collection objs, observe o/p on Console.

05/10/15

Functionality of addAll(), containsAll(), removeAll(), retainAll(), addAll() method:-

addAll() method:-

AddAll will copy given Collection element into current collection

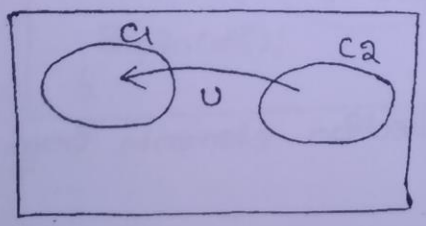
for example

```
C1.addAll(C2);
```

All elements of C2 will copy into C1. then the elements of C2 will have references from C2 and also from C1.

Note:- If C1 is list type Collection, duplicate objs can copy from C2 to C1.

If C1 is set type Collection, if C2 elements already available in C1, elements are not copied.



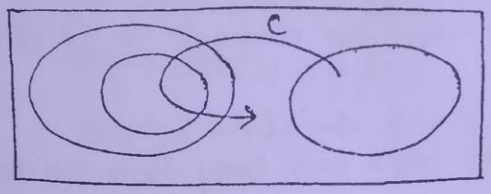
As per set theory addAll() method perform union (∪).

```
import java.util.*;
class addAllTest{
public static void main(String[] args){
    AL al= new AL();
     al.add("a");
     al.add("b");
```

```
LHS lhs1 = new LHS();
lhs1.add("a");
lhs1.add("b");
AL al2 = new AL();
al2.add("d");
al2.add("e");
al2.add("a");
al2.add("b");
al1.addAll(al2);
lhs1.addAll(al2);
Sopln(al1); ---> [a,b,d,e,a,b]
Sopln(lhs1); ---> [a,b,d,e]
}
}
```

containsAll():-

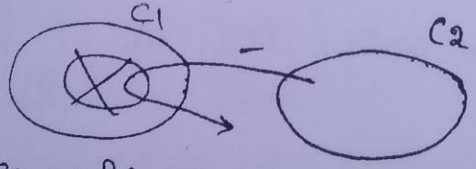
It is used for checking all elements of given collection are available in this collection or not. If available it returns true else returns false. Atleast one element missing also returns false.



containsAll() method perform subset(c) operation. When we pass empty collection containsAll() method returns true.

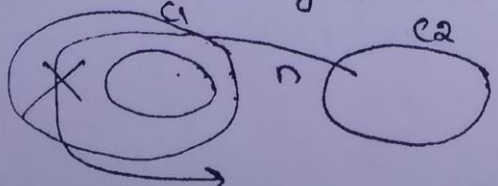
removeAll():-

It is used for removing given collection elements from current collection all occurances.



This performs '-' operation of set theory

retainAll():- This method will retain given collection elements in current collection, removes remaining all other objs in current collection.



This meth performs intersection operation.

Retrieving objects from Collection & Map

08/10/15

List, Set (Collection) :-

→ There are 8 ways to retrieve objects from Collections (List, Set).

1) using index based → l.get(index)
2) using Enumeration → Collections.enumeration(c)
3) using Iterator → c.iterator()
4) using ListIterator → l.listIterator()
5) using for-each loop → for(:)
6) forEach(-) method
7) stream API
8) SplIterator

+ available from 1.8v

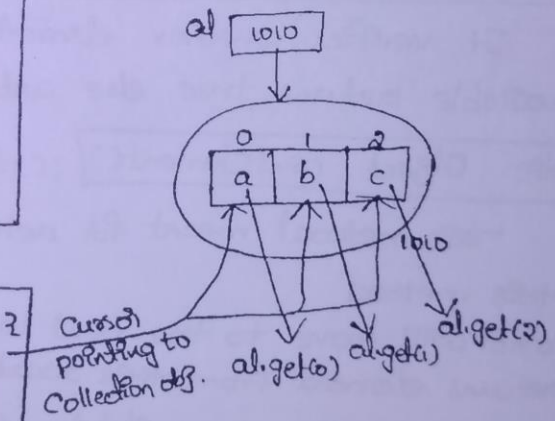
Map :-

→ Retrieving objects from map we have 4 ways for retrieving objs from Collection & map.

1) using get(key)
2) using m.values()
3) using m.keySet()
4) using m.entrySet()

Retrieving objects from List using index :-

```
import java.util.*;
class RetrievingUsingIndex {
  P S V main(String[] args) {
    ArrayList al = new ArrayList();
    al.add("a");
    al.add("b");
    al.add("c");
    Sopln(al);
    for(int i=0; i<siz(al.size()); i++) {
      Object obj = al.get(i);
      Sopln(obj);
    }
  }
}
```



Retrieving objs from List using Enumeration :-

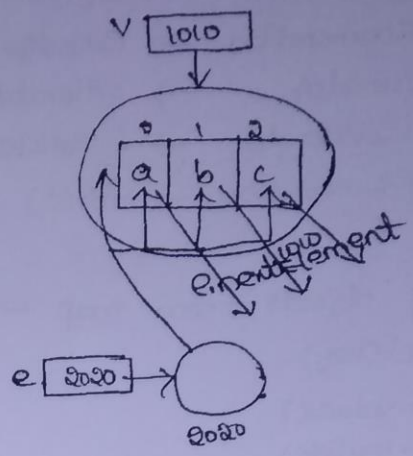
→ Enumeration is an interface bcoz e.hasMoreElements() and e.nextElement() methods are having abstract class.

→ Enumeration is a legacy interface given in Java 1.0. It is used for retrieving objs from vector and hashtable.

→ For checking element available or not & for retrieving element from Collection it has below 2 methods.

Program:-

```
import java.util.*;
class RetrievingUsingEnumeration {
    p s v m(SI args) {
        Vector v=new Vector();
        v.add("a");
        v.add("b");
        v.add("c");
        Sop ln(v);
        Enumeration e=v.elements();
        while(e.hasMoreElement()) {
            Object obj=e.nextElement();
            Sop ln(obj);
        }
    }
}
```



i) `public boolean hasMoreElements()`

It verifies whether element available in the next location or not if available returns true else returns false.

ii) `public Object nextElement()`

this method meant for retrieving objs from collection when we call this method

→ cursor will move to the next location
→ returns element from this location.

Rule:- If element not available in this location this method will throw NoSuchElementException. So we must not call this method on empty location or on empty collection.

→ Enumeration is an Interface its subclasses are defined by sun inside Collection classes has inner class. To obtain this subclass obj we have a factory method elements defined in vector class.

`public Enumeration elements`

- for retrieving elements from vector obj using Enumeration we must call 3 things

1) v.elements() for obtaining enumeration obj
2) e.hasMoreElements() for checking element available or not
3) e.nextElement() for retrieving obj.

Q) How can we obtain Enumeration obj on Collection framework Collections like ArrayList, HashSet.

Ans:- Factory method given in Collections class.

we have 3 words

Collections, Collection, Collections

Collection:→ It represents one group of objs (In English word)

Collection:→ It is the root interface of all types of collections it supplies methods for performing collection operation.

Collections:→ It is a utility class, Contains methods to perform operations common to all Collection and map objs. such as obtaining Enumeration, sorting, Searching etc..

The Method for obtaining Enumeration:-

public static Enumeration enumeration(Collection c)

→ we can call this method by passing list or set or queue objs.

Q) write a program for retrieving elements from ArrayList using Enumeration.

Ans:-

```
import java.util.*;
class RetrievingEnumerationFromAL{
  P S V main(String[] args){
    ArrayList al=new ArrayList();
    al.add("a");
    al.add("b");
    al.add("c");
    Sopln(al);
    Enumeration e=Collections.enumeration(al);
    while(e.hasMoreElements()){
      Object obj=e.nextElement();
      Sopln(obj);
    }
    Sopln(al);
  }
}
```

→ memory diagram copy from previous page.

Working with Iterator:- → It is replace of Enumeration from Java 1.2 onwards. 2007 - 2015

```
import java.util.*;
class RetrievingUsingIterator {
  p s v main(String[] args) {
    ArrayList al = new ArrayList();

    al.add("a");
    al.add("b");
    al.add("c");
    sop.ln(al);
    Iterator itr = al.iterator();
     while(itr.hasNext()) {
      Object obj = itr.next();
       sopln(obj); → itr.remove();
    }

       sopln(al);
     }
}
```

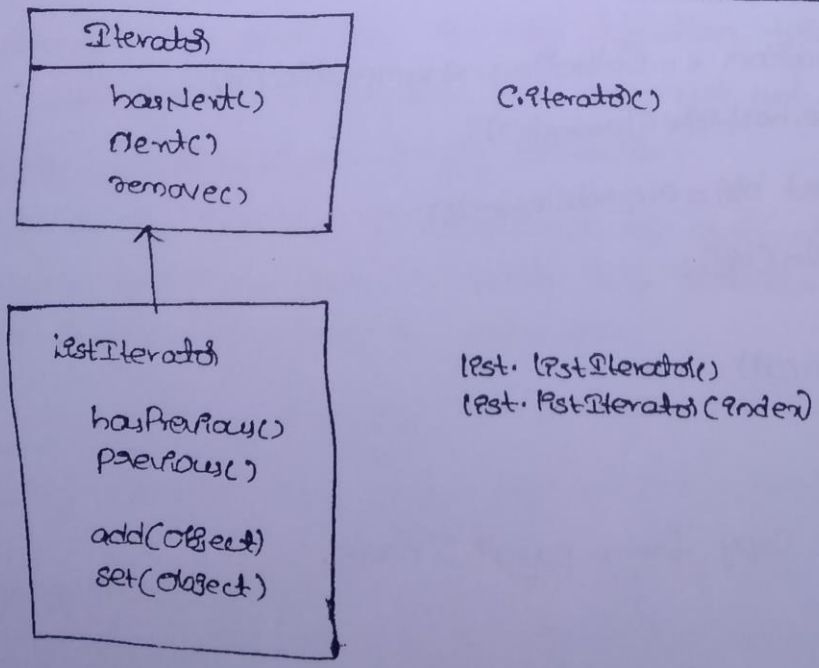
Q) Diff?

→ Enumeration is a legacy Interface used for only retrieving elements from collection.

→ Iterator is a Collection framework Interface used for retrieving and also remove the elements from Collection. [for retrieving get method is used only for list objs.]

→ Iterator is unidirectional (forward only) cursor.

→ ListIterator is a subclass of Iterator it is bidirectional cursor.



Q) write a prog for creating ArrayList with 3 Integer String objs and 3 Integer objs add string & Integer objs alternatively then while iterating insert Integer 20 after 1st Integer value, replace all strings with their uppercase, display result elements in reverse order.

Ans:-

```
import java.io.*;
Class ListIteratorDemo {
  p s v main(String[] args) {
    ArrayList al = new ArrayList();
    al.add("a");
    al.add("A");
    al.add("b");
    al.add(2);
    al.add("C");
    al.add(3);
    Sopln(al);
  ListIterator ltr = al.listIterator();
    int count=1;
    while(ltr.hasNext()) {
      Object obj = ltr.next();
      if(obj instanceof String) {
        ltr.set((CString)obj).toUpperCase());
      }
      else if(obj instanceof Integer) {
        if(count==1) {
          ltr.add(20);
          count++;
        }
      }
    }
    Sopln(al);
    while(ltr.hasPrevious()) {
      Sopln(ltr.previous());
    }
  }
```

Op:- [a, 1, b, 2, C, 3]
[A, 1, B, 2, C, 3]
[A, 1, B, 20, C, 3]
3
C
20
B
1
A

For-each loop :- It can be used from java 5 onwards in place of using Iterator

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

Enhanced for loop :-

→ for loop syntax is enhanced for retrieving objs from array and Collection objs without using regular for loop and iterators.

→ Enhanced for loop is also called for-each loop because it iterates for each obj available in array or Collection its syntax

```
for(variable declaration : obj)
{
  ---
  ---
}
```

↑ should be either array obj or Iterable interface subclass obj.

→ Collection interface is subclass of Iterable interface so for-each loop is allowed to apply on Collection type objs.

→ Map is not subclass of Iterable so we cannot apply/use for-each loop on Map type objs.

→ Why Enhanced for loop :-

→ To reduce no. of lines of code development in retrieving objs from array and collection enhanced for loop is given.

→ for example. for retrieving objs from any Collection same code used Iterator. To eliminate this code enhanced for loop given. Below diagram shows retrieving objs from ArrayList using Iterator (lengthy code) its equivalent for-each loop (short code).

```
Iterator itr = al.iterator();
while(itr.hasNext())
{
  Object obj = itr.next();
  Sop(obj);
}
```

Common code eliminated using for-each loop

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

3 rules on for-each loop:-

1) object type should be array obj or Iterable type.

i) String s1="a";
for(Object obj : s1) {} → XCE: String is not Iterable type

ii) String[] sa={"a","b"};
for(String s:sa) {} → ✓

iii) LHS lhs=new LHS();
for(Object obj:lhs) {} → ✓

iv) LHM lhm=new LHM();
for(Object obj:lhm) {} → ✓ XCE: (map is not subclass of Iterable)

v) AL al=new AL();
Iterator itr=al.iterator();
for(Object obj:itr) {} → XCE: (because for & itr are both cursors can't be open one in another)

2) Variable should declare inside parantheis

Object obj;
for(obj:al) {} → XCE:

3) Variable type should be same type or superclass type of the objs retrieving from Collection.

AL al=new AL();
al.add("a");
al.add("b");
for(String s:al) {} → XCE:
for(Object obj:al) {} → ✓
for(String s:(String)obj) {} → XCE:

for(Object obj:al){
String s=(String)obj; → ✓
}

→ get() method will not throw Exception if an object is not available while Iterator throw an Exception.

→ ("a" = 1) It is → Map.Entry type object
(key) (value)

Q) Develop a program for storing employee number and its complete obj as key and value. for the employee #280 increase salary by 1000 Rs. then display all employees key and values.

06/10/13

Generics:-

→ This concept is introduced in Java 5.0.

→ To avoid 'Type casting' and further to avoid CCE by providing compile time type safety.

→ It's nothing but, by using generics we are informing to compiler the type of objs are used and must use at execution time.

→ Mainly generics concept was introduced for collection framework objs for creating homogeneous collections, means to store only same type objs.

Eg:- ArrayList al = new Anaylist();

→ It is a heterogeneous collection, allows us to store different type of objs. So at retrieving we must use cast operator and if (instance of) condition for converting type & avoid CE.

```
ArrayList<String> al = new ArrayList<String>();
```

It is generic collection, homogeneous collection

ArrayList<Integer> al = new ArrayList<Integer>();

ArrayList<Employee> al = new ArrayList<Employee>();

Q) Below prog shows adding & retrieving elements from ArrayList with & without generics.

Ans:-

```
import java.util.*;
class ALWithOutGenerics {
 P S V m(S[] args){
  AL al=new AL();
  al.add("a");
  al.add("b");
  al.add(1);
  Sopln(al);
 Iterator itr=al.iterator();
 while(itr.hasNext()){
 Object obj=itr.next();
 if(obj instanceof String){
  String s=(String)obj;
  Sopln(s.toUpperCase());
 }}
}
```

```
import java.util.*;
class ALWithGenerics {
 P S V m(S[] args){
 AL<String> al=new AL<String>();
  al.add("a");
  al.add("b");
  Sopln(al);
Iterator<String> itr=al.iterator();
while(itr.hasNext()){
  String s=itr.next();
   Sopln(s.toUpperCase());
  }
 }
```

```
// Retrieving using for:each loop
  for(String s:al){
   Sopln(s.toUpperCase());
  }
```