Dt:- 02/03/2022

COLLECTION FRAMEWORK

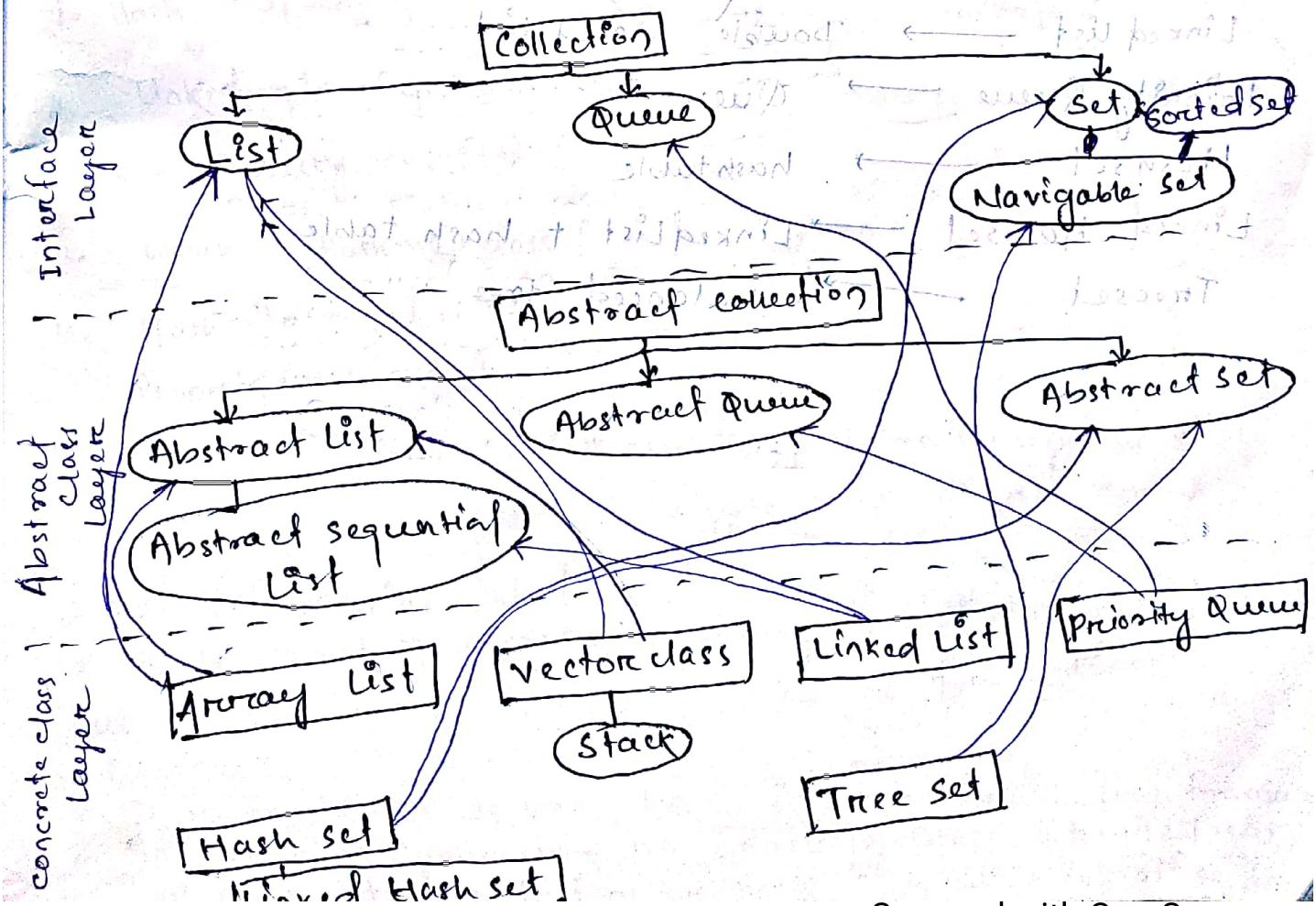
what is collection?

* A group of individual objects behaves as a single entity is called as collection.
* Collection framework contains several classes, interfaces which is use to represent a group of object as single entity.
* In C++ similar type of entity called as STL (Standard Templet Library)
* But collection framework is one of the popular technology use by Java. what ever classes available collection framework directly support some inbuild data structure, like: dynamic memory, linked list, stack, queue, balanced tree, hash table etc.
* collection framework contains 9 interfaces.

~~HIGHER~~ Hierarchy:-

It devided into 3 layer

(i) Interface Layer.
(ii) Abstract class Layer.
(iii) Concrete class Layer.



(i) ArrayList (inherit from abstract List class & List interface class)

(ii) Vector class (" " ")
↓
Stack (child class)

(iii) Linked List (inherit from abstract sequential list class & class list interface)

(iv) Priority Queue (inherite from abstract Queue class & Queue class inheritance)

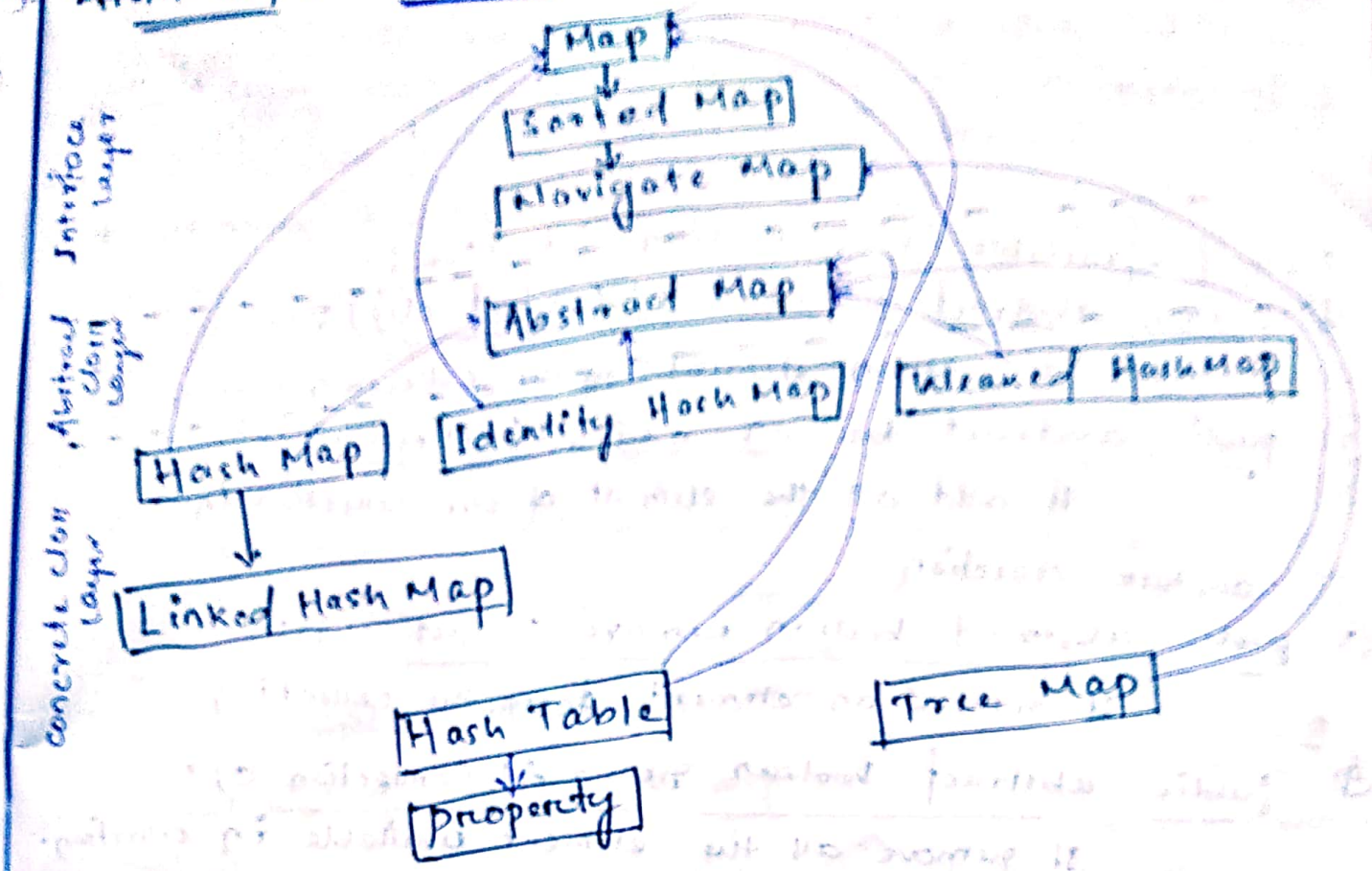
(v) Hash set class (inherite from abstract set class & set interface)
↓
Linked HashSet (child class)

(vi) Tree set (" " ")

Which datastructure which class support?

Class		datastructure
Arraylist	→	dynamic array
vector	→	dynamic array
stack	→	stack
Linked list	→	Double Linked list
Priority Queue	→	Queue
Hashset	→	hashtable
Linked Hashset	→	LinkedList + hash table
Treeset	→	balanced tree

~~Hash key~~ : Hierarchy :



1. Hash map class (inherite from abstract map class of map interface).
2. Linked Hash Map (child class)
2. ~~Ide~~ Identity Map class (")
3. Weaked Hash map class (").
4. Hash table class (")
   Property (child class)
5. Tree map class (inherite from abstract map class of navigate map interface).

What is Collection Interface ?

* If we want to represent a group of individual object as a single entity then we should go for collection interface.
* It is considered as root/base interface in collection framework
* In collection framework no concrete class inherite from [illegible] class inherite fr

the child class interface of the collection interface inherit also bound to override the methods of collection interface.

* In simple word every concrete class contains most of the common methods which is applicable for collection object.

Method available in collection interface :-

① public abstract boolean add(object obj) :-
It add an element in the collection.

② public abstract boolean addAll(collection c) :-
It add all the element of one collection in another collection.

③ public abstract boolean remove(object obj) :-
It remove an element from the collection.

④ public abstract boolean removeAll(collection c) :-
It remove all the element available in collection.

⑤ public abstract boolean isEmpty() :-
It check collection is empty or not.

⑥ public abstract int size() :-
It extract size of the collection.

⑦ public abstract boolean contains(object obj) :-
It check the specified element is available or not.

⑧ public abstract Iterator iterator() :-
It return a reference of iterator interface

In collection framework iterator is a carser. this carser is use to retrive element from collection object in the forward direction.

⑨ Public abstract object[] toArray() :-
It retrieve all the element from collection object & store it in the array of object class.

Dt:- 04-03-22 ⑤

What is the advantage of using collection framework?

* To avoid the drawback of array collection framework was introduced.

* Array having some drawback :-

① Array is fixed in size.

In simple word array is a datastructure where we can't change the size of the array.

② Array holds homogeneous elements.

Ex:- new int[5]; // here size is fixed.

String s[] = new String[2];
s[0] = new String("java");
s[1] = new StringBuffer("python"); // error.

Advantage of collection framework :-

① collections are throwable in nature. As per our requirem we can increase and decrease the size of the array.

② collections holds both homogeneous & heterogeneous elemen

Arraylist al = new Arraylist();
↓
This Arraylist object size=0

al.add("Java");
al.add(new StringBuffer("python"));

③ Every collection implemented class based on some datastruct

Arraylist support dynamic Array
Array support Static Array

Drawback of Collection :-

From the performance point of view collections are not recomended over array.

Difference between Collection & Array :-

(i) Array is fixed in size. so it is static in nature, whereas collections are auto growable. so it is dynamic in nature.

(ii) from the memory point of view array is not recommended which as memory point of view collection is recommended.

(iii) from the performance point of view, array is recommended over collection.

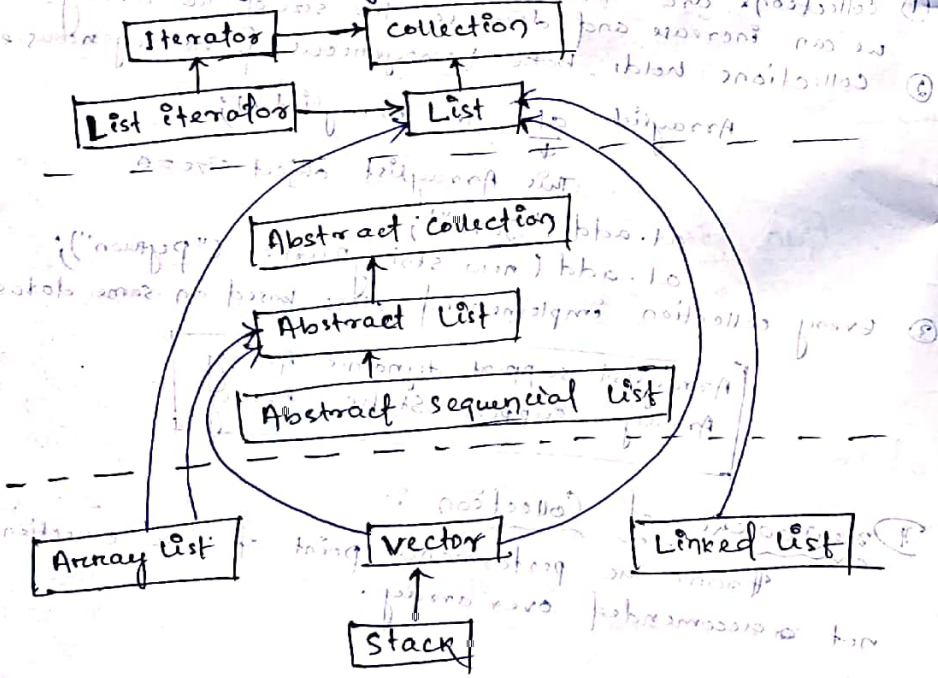
(iv) Array holds homogeneous element where as collection holds heterogeneous elements.

(v) Array elements are either premitive datatypes or array elements are object reference where as collection elements are only object reference.

* 1.2 Jdk version collection introduced.

LIST :-

* List is the child interface of collection it contains group of object as a single entity.



* It contains duplicate elements.
* It holds the elements in insertion order. As the elements are stored in insertion order. It takes an important role to differentiate the duplicate elements.
* The above diagram shows four classes inherit from list interface -
  1. ArrayList
  2. vector
  3. LinkedList
  4. stack (child class of vector)

Methods available in List interface :-

(1) public abstract boolean add(int index, Object obj) :-
It add the element in the specified index of the list.

(2) public abstract boolean addAll(int index, Collection c) :-
It add all the elements of a collection in the specified index of another collection.

(3) public abstract object remove(int index) :-
It remove an element from the list.

(4) public abstract int indexOf(Object obj) :-
It returns the index position where the specified object is stored.

(5) public abstract object get(int index) :-
It retreive the element from the list.

(6) public abstract int lastIndexOf(object obj) :-
It return the index position where the specified object is stored by searching last to first.

(7) public abstract ListIterator listIterator() :-
It returns the object of the ListIterator interface ListIterator is a cursor which is used to traverse the elements both in forward and backward iteration.

Arraylist class :-

* Arraylist is the pre-defined class present in java.util.package
* Arraylist class inherit from abstract list class and list interface.
* The underline datastructure is supported by arraylist object is dynamic array.
* If an arraylist object when we add new elements then the size of the arraylist object is auto expended and on the otherhand when we remove an element from the arraylist object then the size of the arraylist object is automatically shrink.
* As array list class inherit from list interface it is capable to hold duplicate elements and the elements are stored in insertion order.
* The constructor of the Arraylist class is overloaded
  ① Arraylist () → non parameterised
  new Arraylist ()

  → when we call this constructor construct an arraylist object then the size of the arraylist is 0. where as the default capacity of the arraylist object is 10.

* Once arraylist object is reached its maximum capacity then a new arraylist object is constructed where the capacity is: **current capacity * 3/2 + 1**

  ② Arraylist (int capacity)
  new Arraylist (1000);

  ③ Arraylist (Collection c)
  new Arraylist (new Arraylist (100));

* Arraylist class, override each and every method of list interface and collection interface.

NOTE :-
① An array list object can hold the null value.
② It override to string() method of object class.

PROGRAM :-

```
import java.util.*;
public class Test1
{
    public static void main (String args[])
    {
        ArrayList aa = new ArrayList ();
        System.out.println ("Size: " + aa.size ());
        aa.add (new String ("java"));
        aa.add (new String ("python"));
        aa.add (new String ("c++"));
        aa.add (new String ("c"));
        aa.add (new String ("perl"));
        System.out.println ("Size: " + aa.size ());
        int size = aa.size ();
        for (int i=0; i<size; i++)
        {
            String name = (String) aa.get (i);
            System.out.println (name);
        }
        String s = (String) aa.remove (3);
        sopln ("Remove elements is: " + s);
        sopln ("Size: " + aa.size ());
    }
}
```

Output :-

```
Size = 0
Size: 5
java
python
c++
c
perl
Remove element is: 0
Size: 4
```

Dt :- 05-03-2022

```
import java.io.*;
import java.util.*;
class Emp
{
    int empid;
    String empname;
    double sal;
    public Emp()
    {
        try{
            DataInputStream dis = new DataInputStream(System
            Sopln("Enter Emp Name");
            empname = dis.readLine();
            Sopln("Enter Emp ID.");
            empId = Integer.parseInt(dis.readLine());
            Sopln("Enter salary");
            sal = Double.parseDouble(dis.readLine());
            } catch(IOException ie)
                {
                    ie.printStackTrace();
                }
            }
            @Override
            public String toString()
            {
                Return "Name: "+empname+"ID:"+empid+
                                "Salary:"+sal;
            }
}
public class Demo
{
    Static ArrayList aa = new ArrayList();
    public static void addEmp()
    {
        aa.add(new Emp());
    }
```



```
public static void retrieveEmp()
{
    int size = aa.size();
    if(size>0)
    {
        Sopln("\nSee the info of the Employee.");
        for(int i=0; i<size; i++)
        {
            Emp ee = (Emp) aa.get(i);
            Sopln(ee);
        }
    }else{
        Sopln("Please add an Employee.");
    }
}
public static void deleteEmp()
{
    int size = aa.size();
    if(size>0)
    {
        Sopln("Remove the last employee...");
        for(int i=0; i<size; i++)
        {
            if(i==(size-1))
            {
                Emp aa1 = (Emp) aa.remove(i);
                Sopln("Remove Empdata is: "+aa1);
            }
        }
    }else{
        Sopln("please add an Employee.");
    }
public static void main(String argsTT)
{
    while(true)
    {
        Sopln("\t\t ArrayList Object...\n");
        Sopln("\t\t See the option...$");
        Sopln("\t\t 1> Add An Employee.");
        Sopln("\t\t 2> Retrive The Employee Info.");
```

```
Sopln ("\t\t 3> Delete the last Employee.");
Sopln ("\t\t 4> Quit.");
Sopln ("Enter Your choice:");
int ch = new Scanner (System.in).nextInt();
switch (ch)
{
    case 1 : addEmp();
             break;
    case 2 : retriveEmp();
             break;
    case 3 : deleteEmp();
             break;
    case 4 : Sopln ("Thankyou for using my applicatio
             System.exit (0);
    default : Sopln ("!!! Invalid choice !!!");
              break;
}
}
}
```

VECTOR CLASS :-

→ vector is a predefined class present in java.util package.
→ vector is the child class of abstract list class and list interface.
→ The underlying datastructure is supported by the vector class object is dynamic array.
→ vector class object performed like arraylist object. But the different is vector is threadsafe, which means vector object at a time permit only one thread to process the job.
→ In simple word vector object supports single thread model where as arraylist object supports multithreadi
→ As arraylist object supports multithreading. It's performance is better than vector. But arraylist object face deadlock situation where as vector object never face deadlock situation.
→ In vector class every methods are synchronized. As the methods are synchonized vector object suppo single thread model.
→ Vector is a legacy class.

NOTE :-

What is Legacy class?

→ In collection framework, legacy class means it is already introduced in jdk 1.0, But it is implemented in jdk 1.2 after the collection framework introduced.

The constructor of vector class is overloaded.

① Vector(); → non-parameterized constructor.
when we call this constructor it creates an empty vector object which size is 0. for the default capacity is 10.
Once the vector object reach the maximum capacity then it creates a new vector object which capacity is :-

new capacity = 2 × current capacity

② Vector(int capacity) :-
By calling this constructor we can constructor a vector object where as a programmer we can fix the capacity

③ Vector (Collection c):-

By calling the constructor we can construct a vector object ~~whereas a programmer~~ by passing another collection object within it.

Difference between Vector & ArrayList :-

① In arraylist no methods are synchronized, whereas in Vector class every methods are synchronized.

② Arraylist Object is not thread safe where as the vector Object is threadsafe (means multithreading supports arraylist object & vector supports single thread model).

③ Arraylist Object performance is better than vector object.

④ Arraylist is not a legacy class where as vector is legacy class.

METHODS :-

① public synchronized void add (Object obj):-
→ It add an element in the vector object.

② public synchronized void add (int index, Object obj):-
→ It add an object in the vector Object in the specified position.

③ public synchronized void addElement (Object obj):-
→ It add an element.

④ public synchronized void remove (Object obj):-
→ It remove the element.

⑤ public synchronized void removeElement (Object obj):-

⑥ public synchronized Object remove (int index):-
→ Remove element by specified index position.

⑦ public synchronized Object removeElement (int index):-
→ It is used to remove all element in specified index

⑧ public synchronized void removeAllElement ():-
→ It is used to remove all element.

⑨ public synchronized Object get (int index).

⑩ public synchronized int size():-
→ It returns the size of object.

⑪ public synchronized int capacity:-
→ It is used to knows the capacity.

⑫ public synchronized Enumeration elements ():-
→ Enumeration is a cursor. It is use to traverse element & it is available in legacy class.

PROGRAM :-

```
import java.util.*;
public class Demo1
{
  public static void main (String args[])
  {
    Vector v1 = new Vector ();
    Sopln ("Size is : " + v1.size() + "Capacity is : "+v1.capa
    v1.add (new String ("One"));
    v1.add (new String ("One"));
    v1.add (new String ("One"));
    v1.add (new String ("One"));
    v1.add (new String ("One"));
    v1.add (new String ("One"));
    v1.add (new String ("One"));
       "
       "
    Sopln ("Size is : " +v1.size() + "Capacity is : " + v1.capacity

    System.out.println ("\n See the elements of vector object");
```

O/P :-
Size is : 0 Capacity is : 10
Size is : 11 Capacity is : 20
See the elements of Vector object :
One
One
One
One
One
one
One
One
One
One
One

CURSER USED IN COLLECTION FRAMEWORK :-

java used 3 types of curser in collection framework.

1) Iterator
2) ListIterator
3) EnumIterator

① These three cursers are pre-defined interface. In core java we know that interface are implicitly abstract. So it can't be instanciated. (never create the object of interface).

Why we use Cursers ?

→ cursers are used in collection framework to retrieve the elements.

→ All these 3 curser are interface.

② In every method within an interface implicitly public and abstract. when a method is public and abstract it bound to be override on the child class. If it not override then it must generate compile the error.

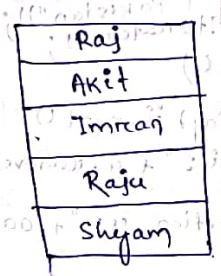
Iterator Interface :-

* Iteratator is a predeclared interface present in java.util package.
* Iterator interface is implemented on collection implemented class :-
  ① ArrayList ④ Stack ⑦ Treeset
  ② LinkedList ⑤ Hashset ⑧ Priority Queue
  ③ Vector ⑥ LinkedHashset

* As iterator is an interface it can't be instantiated. (You can't create object of Iterator).
* But if as a programmer if we wan't to create an obje of Iterator interface then we have to call a predeclared method of collection interface.

→ public abstract Iterator iterator(); // collection interfa

* When we create an object of the iterator interface then it always points to before the row :



METHODS AVAILABLE IN ITERATOR INTERFACE :-

① public abstract boolean hasNext() :-
When we call the method for the first time the compiler check the first row in available or not. If the first row is available then it returns true and curser moves to the first row.
But on the other hand if the row is not available then it returns false.

② public abstract Object next() :-
This method is used to extract the element whome th curser points.

③ public abstract Object next() :- remove
This method is used to ~~extract~~ the element from the collection whom the curser points.

* The main responsibility of iterator interface is if not only retrive the element from the curser but it is used to remove the elements from the collection.
* But iterator traverse the elements only in the forward direction.

PROGRAM :-

```
import java.util.*;
public class IteratorDemo
{
    public static void main (String args[])
    {
        ArrayList aa = new ArrayList();
        aa.add (new String ("India"));
        aa.add (new String ("Russia"));
        aa.add (new String ("Japan"));
        aa.add (new String ("Kuba"));
        aa.add (new String ("Pakistan"));
        Iterator ii = aa.Iterator ();
        while (ii.hashNext())
        {
            String s = (String) ii.next();
            Sopln ("Name is: " + ii.remove());
        }
        Sopln ("Size of collection is: " + aa.size());
    }
}
```

Output :-

```
Name is : India
Name is : Russia
Name is : Japan
Name is : Kuba
Name is : Pakistan
Size of collection is : 0
```

LIST ITERATOR :-

* List Iterator is a pre-defined interface present in java.util package.
* List iterator is a child interface of iterator. But it is implemented on list implemented classes.
  1. Arraylist
  2. LinkedList
  3. vector
  4. stack.

List iterator if is also a Cursor -

* But iterator only traverse the element on the forward direction but list iterator traverse the elements in both forward and backward direction.
* Iterator only remove the element and traverse the element where as List iterator remove element, add the elements and update the element.
* It is an interface can't be instanciated. If the programm wants to instanciated the List iterator interface then the programmer have to call ListIterator() method of List interface.

  → public abstract ListIterator (); // [List interface

METHODS AVAILABLE IN LIST ITERATOR INTERFACE :-

As ListIterator child interface of Iterator interfa
it was every method of iterator interface.

1. public abstract boolean hasPrevious():
   It check the previous element is available or n
2. public abstract Object previous():
   This method ~~abs~~extract the previous element from the Lis
3. public abstract void add (Object obj):
   It is used to add an element in the List (not in Collection)
4. public abstract void set (Object obj):
   This method update (set) the element in the list
   modify

PROGRAM :-

```
import java.util.*;
public class ListIteratorDemo
{
    public static void main (String args[])
    {
        ArrayList aa = new ArrayList();
        aa.add ("Java");
        aa.add ("Python");
        aa.add ("Perl");
        aa.add ("Velocity");
        aa.add ("PHP");
        ListIterator li = aa.ListIterator();
        Sopln ("\n Traverse the elements in forward direction
```

```
while (li.hasNext())
{
    String s1 = (String) li.next();
    Sopln(s1);
}
li.add("Spring Boot");
Sopln("\n Traverse the elements in backward direction \n");
while (li.hasPrevious())
{
    String s2 = (String) li.previous();
    Sopln(s2);
    Li.remove();
}
Sopln("size of list is:"+aa);
}
```

Output:- Traverse the element in forward direction

~~direction~~
Java
Python
Perl
velocity
PHP
~~springboot~~

Traverse the element in backward direction

Springboot
PHP
Velocity
Perl
Python
Java.

Difference between Iterator and ListIterator :- (21)

① Iterator is the base interface where as list iterator is the child interface.

② Iterator is implemented on collection implemented classes where as List iterator is implemented on list implemented classes.

③ Iterator traverse the elements in forward but list iterator traverse the elements in both forward and backward direction,

④ Iterator not only traverse the element but also remove the element where as list iterator traverse the elements, remove the element, add the element and update the element.

ENUMERATION :-

* Enumeration is a pre-declared interface present in java.util package
* Enumration is a curser which is implemented on legacy classes like:-
  * Vector → public synchronized Enumeration elements(
  * Stack →
  * Hashtable → public synchronized Enumeration keys()
  * Properties →
  * String Tokenizer →
* Enumeration is an interface, as it is an interface it can't be instanciated. If the programmer wants to instanciate the enumeration interface, then the programmer have to call either the method of vector class or hashtable class.
* Enumeration is a curser which is only use to retrive the element from the legacy class object only in forward direction.
* Enumeration curiser traverse the elements only in forward direction.

METHODS :-

① public abstract boolean hasMoreElements():-
The main responsibility of this method is it check the next element is available or not.

② public abstract Object nextElement():
this method is used to extract the element whom the ~~curi~~ curser point. It extract the next element available in the legacy class.

Program :-

```
import java.util.*;
public class EnumerationDemo
{
  public static void main (String args[])
  {
    Vector v = new Vector ();
    v.add (new String ("Java"));
    v.add (new String ("Structs"));
    v.add (new String ("Spring"));
    v.add (new String ("Hibernate"));
    Enumeration ee = v.elements ();
    Sopln ("In Traverse the elements \n");
    while (ee.hasMoreElements ())
    {
      String name = (String) ee.nextElement ();
      Sopln (name);
    }
  }
}
```

Output :-

```
Traverse the elements
Java
Structs
Spring
Hibernate
```

Program :

The previous program when the output came we have recived an warning message. if we want to avoid this warning (class cast Exception) we write this :-

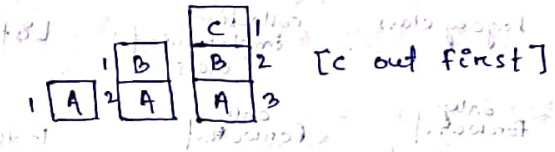
```
⇒ import java.util.*;
public class EnumerationDemo
{
  @SuppressWarnings ("Unchecked");
  public static void main (String args[])
  {
  }
}
```

STACK CLASS :- (23)

* Stack is a pre-defined class present in java.util. package.
* It is a child class of vector.
* It is a legacy class.
* It supports an algorithm LIFO (Last In First Out)
* This class having one constructor.

  stack() → non parameterized constructor.

* When we add an element in a stack object then it is added in this structure.



METHODS :

① public synchronized Object push (Object obj) :-
→ push in the method it is used to push an elemer in the stacks.

② public synchronized Object pop() :-
→ It is used to remove the last inserted element from stack.

③ public synchronized boolean isEmpty() :-
→ It check the stack is empty or not.

④ public synchronized int search (Object obj) :-
→ It search the position of the element in the sta If the element is not available then it return "-1".

Program :-

```
import java.util.*;
public class stackDemo
{
  public static void main (String args[])
  {
    Stack s = new stack ();
    S.push ("India");
    S.push ("Japan");
    S.push ("Canada");
    S.push ("China");
    S.push ("Cuba");
    Sopln ("position of India is: " + s.search ("India"));
    Sopln ("position of Cuba is: " + s.search ("Cuba"));
    String name = (String) s.pop();
  } Sopln ("Remove country name is : " + name);
```

Output:

position of India is : 5
position of Cuba is : 1
position of Pakistan is : -1
Remove country name is : cuba

Property	Enumeration	Iterator	ListIterator
It is legacy class	Yes	NO	NO
It is applicable	Legacy class	collection implemented class	List implemented class
Movements	only forward	only Forward	Both Direction (Forward / Backward)
How to get it (object)	elements() of vector / key() of Hashtable	iterator() of collection	ListIterator() of List Interface.
accessibility	read only (Retrive)	read & remove (Retrive)	read, remove, add & replace (modify) [read means retrive]

LINKEDLIST CLASS:-

* Linked list is a predefined class present in java.util package.
* LinkedList is the child class of AbstractSequencedList class & List Interface.
* The underline datastructure supported by linkedlist Object is double linkedlist.
* Here the data are stored in node ~~from~~ form and every node having three field
  ① is data field.
  ② is two linked field.
* LinkedList object hold the address of next and previous node.
* As array node hold the address of next and previous node have adding the element in any position, remove the element in any position, remove the element in any position is better than arraylist and vector.
* The constructor of the class is
  ① LinkedList()
  → It creates an empty linkedlist object class where size is Zero. (There is no capacity)

② LinkedList (Collection c)

METHODS:-

It override every method of list and collection interface.

① public void addFirst (Object obj):-
→ It add the element at first position.

② public void addLast (Object obj):-
→ It add element in the last position.

③ public Object getFirst():-
→ It extract the element at first position.

④ public Object getLast():-
→ It extract the element at last position.

⑤ public Object removeFirst():-
→ It remove the element at the First position.

⑥ public Object removeLast():-
→ It remove the element at the Last position.

Program:-

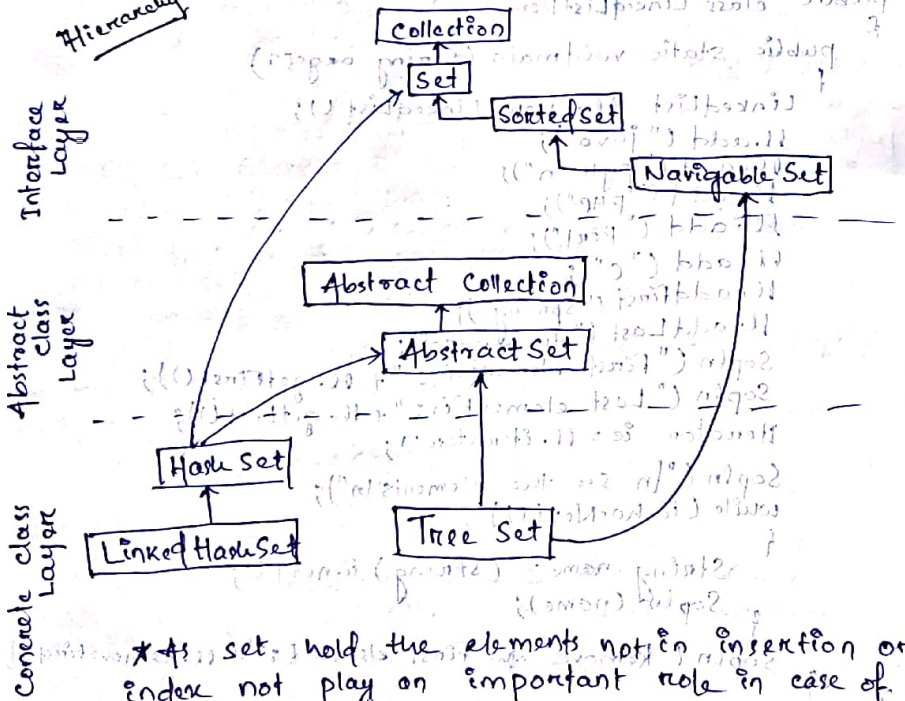
```
import java.util.*;
public class LinkedListDemo
{
    public static void main (String args[])
    {
        LinkedList ll = new LinkedList();
        ll.add ("java");
        ll.add ("Python");
        ll.add ("PHP");
        ll.add ("Perl");
        ll.add ("C");
        ll.addFirst ("Spring");
        ll.addLast ("Hibernate");
        Sopln ("First element is: " + ll.getFirst());
        Sopln ("Last element is: " + ll.getLast());
        Iterator ie = ll.iterator();
        Sopln ("\n See the elements \n");
        while (ie.hasNext())
        {
            String name = (String) ie.next();
            Sopln (name);
        }
        Sopln ("Remove the first element: " + ll.removeFirst());
    }
}
```

Output :-

See the elements
Spring
Java
Python
PHP
Perl
C
Hibernate
First elements is : Spring
Last elements is : Hibernate
Remove the first element : Spring

SET INTERFACE :-

* Set is a pre-declared interface present in java.util package.
* Set is the child interface of collection.
* Set also hold object as an element but set hold unique element.
* Set hold the elements not in insertion order.



* As set hold the elements not in insertion order index not play on important role in case of set.
* Set never hold duplicate elements, it always hold unique elements.

HASHSET CLASS :-

* It is a pre-defined class present in java.util package.
* It inherite from Abstractset class & set interface.
* The underlying data-Structure supported by Hashset Object is hashtable. → data structure
* As hashset object stored the elements in the data structure called as hashtable. it store the elements in key and value format.
* The key value must be unique for each element present in the hashtable data structure.
* The key value associated with a pointer or memory addres
* When the data is retrived from the hashtable data structur it is retrived through the key value.
* Hash set object stores the information by using the mechanis hashing.
* Hash set object never hold the duplicate elements as it inherit from set.
* In hashing the informational content of the key is used to determine an unique key which is called as hashcode.
* The hashcode is then used as an index at which the dat associated with key is stored.
* The advantage of hashing is it allows the execution time the request for addition, searching, remove & size of the set remains same for large number of elements availab in the set.
* Hashset object not hold the elements in insertion order

CONSTRUCTOR :-

(1) Hashset() :-

It construct a hashset object whose size is '0' but initial capacity is 16. when size exceed the capacity then it construct a new hashset object whose fill ratio is 75% (3/4).

capacity ↑ 16 × 3/4

16 + (16 × 3/4) = 16 + 12 = 28

when exceed the capacity

28 + (28 × 3/4) = 28 + 21 = 49

② HashSet (int capacity):-
It is used to construct a hashset object where the programmer can fix the capacity.

③ HashSet (Collection c):-
We can pass the collection object.

DIFFERENCE BETWEEN SET AND LIST:-

* List hold the duplicate elements whereas set hold the unique element.
* List hold the elements in insertion order but set never hold the elements in insertion order.

Program:-

```
import java.util.*;
public class SetDemo
{
  public SetDemo()
  {
    Collection cc=new ArrayList();
    cc.add("Rashmi");
    cc.add("Ankit");
    cc.add("Rashmi");
    cc.add("Faran");
    cc.add("Ankit");
    cc.add("Rani");
    cc.add("Raj");
    Sopln("\n See the elements of List.\n");
    Iterator ii=cc.iterator();
    while(ii.hasNext())
    {
      String name=(String)ii.next();
      Sopln(name);
    }
    call(cc);
  }
  void call(Collection aa)
  {
    HashSet set=new HashSet(aa);
    Sopln("\n See the element of set.\n");
    Iterator ss=set.iterator();
    while(ss.hasNext())
    {
      String j=(String)ss.next();
      Sopln(j);
    }
  }
  public static void main(String args[])
  {
    new SetDemo();
  }
}
```

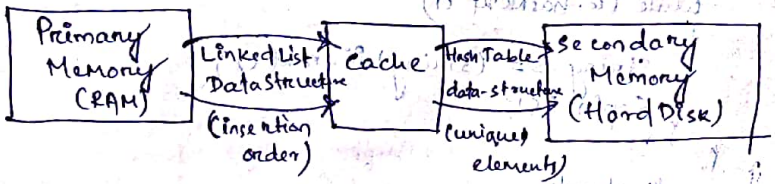
Output:-

See the elements of list
Rashmi
Ankit
Rashmi
Faran
Ankit
Rani
Raj

See the elements of set.
Rashmi
Raj
Faran
Rani
Ankit

LINKED HASHSET CLASS:-

* It is a pre-defined class present in java.util package.
* It is the child class of Hashset class.
* The underlying data-structure supported by LinkedHashSet Object is LinkedList & hashtable.
* The insertion order is preserved by the object of LinkedHashSet.
* This class introduced in collection framework in jdk 1.4
* The object of LinkedHashSet is implementing cache applic
* LinkedHashSet object store the elements in insertion ord but it hold the unique elements. Duplicate elements are not allowed to insert.



* The constructor of the class is overloaded:-
  1. LinkedHashSet()
     size=0, capacity=16, fillratio=75% (3/4)
  2. LinkedHashSet(int capacity)
  3. LinkedHashSet(Collection c)

DIFFERENCE BETWEEN HASHSET & LINKED

* In case of HashSet the underlying datastructure is HashTable where as in case of LinkedHashSet the underlying data structure is Hash Table & LinkedList.
* Insertion order is not preserved in case of Hashset object. where as in case of LinkedHashset the insertion order is preserved..
* Hashset introduce JDK 1.2 where as LinkedHashSet introduce in jdk 1.4 .
* Hashset is the base class where as LinkedHashSet is the child class.

Program :-

```
import java.util.*;
public class LinkedHashSetDemo
{
    public static void main (String args[])
    {
        LinkedHashSet set = new LinkedHashSet();
        set.add("Java");
        set.add("Python");
        set.add("PHP");
        Set.add("Python");
        Set.add("Java");
        Set.add("CGI");
        Set.add("Spring");
        Sopln("In See the elements of LinkedHashSet Object \n");
        Iterator it = set.Iterator();
        while (it.hasNext())
        {
            String s = (string) it.next();
            Sopln(s);
        }
    }
}
```

Output :- See the elements
Java
Python
PHP
CGI
Spring

SORTEDSET INTERFACE :-

* It is the child interface of set interface.
* As it is the child interface of set interface, it contains unique elements but the elements are sorted in a sorted order. * It is a pre-declared interface present in java.util package.

(Sorted order means Ascending order)

METHODS :-

1. public abstract Object first();
   It extract the first element from the sortedset object.
2. public abstract Object Last();
   It extract the last element from the sortedset object.
3. public abstract sortedset headset (Object obj);
   It returns sortedset object whose elements are less than specified Object.
4. public abstract Sortedset tailset (Object obj);
   It returns sortedset Object whose elements are greater than or equals to specified object.
5. public abstract SortedSet subset (Object obj1, Object obj
   It returns a sortedset object whose elements are greater than & equal obj1 and less than obj2.
6. public abstract Comparator comparator();

   First Comparator = interface
   Second Comparator = Method ()

   It returns a comparator interface object that which the elements are stored in a sorted order (& the prog describe underly sorting technique

```
Treeset ts = new TreeSet();
ts.add (new Integer(100));
ts.add (new Integer(101));
ts.add (new Integer(102));
ts.add (new Integer(103));
ts.add (new Integer(104));
ts.first(); // 100
ts.last(); // 109
ts.headSet(104); // 100, 101, 103
ts.tailSet(104); // 104, 107, 109
ts.subset(101, 107); // 101, 103, 104
```

NAVIGABLESET INTERFACE :-

★ It is a pre-declared interface present in java.util package.
★ It is the child interface of sortedSet interface.
★ As it is the child interface of sortedset interface it, hold the unique element and elements are stored in a sorted order (by default asending order)
★ This interface contains several methods to provide support for navigation for treeset Object.

METHODS:-

① public abstract NavigableSet ceiling (Object obj) :-
It returns the lowest element which is, greater than or equal to obj.

② public abstract NavigableSet higher (Object obj) :-
It returns the lowest element which is greater than obj

③ public abstract NavigableSet floor (Object obj) :-
It returns the highest element which is less than, equal to obj.

④ public abstract Navigableset lower (Object obj) :-
It returns the highest element which is less than obj.

⑤ public abstract object pollfirst () :-
It ~~returns~~ remove the first element.

⑥ public abstract Object pollLast () :-
It remove the last element.

⑦ public abstract NavigableSet descendingSet () :-
It return a navigableset Object where elements are are stored in descending order.

Program :-

```
import java.util.*;
public class NavigableDemo
{
    public static void main (String args[])
    {
        TreeSet ts = new TreeSet ();
        ts.add (new Integer (1000));
        ts.add (new Integer (5000));
        ts.add (new Integer (3000));
        ts.add (new Integer (2000));
        ts.add (new Integer (4000));
        Sopln (ts); // it implicitly call toString method of object
        Sopln (ts.higher (2000));
        Sopln (ts.floor (2000));
        Sopln (ts.lower (2000));
        Sopln (ts.pollFirst ());
        Sopln (ts.pollLast ());
        Sopln (ts.descendingSet ());
    }
}
```


Output :-

```
[1000, 2000, 3000, 4000, 5000]
2000
3000
2000
1000
1000
5000
[4000, 3000, 2000]
```

TREESET CLASS :-

★ It is a pre-declared class present in java.util.package.
★ Treeset class inherit from AbstractSet class & set interface
★ The underlying datastructure supported by Treeset Objec is Balanced tree.
★ As the class interface from NavigableSet interface the duplicate elements are not allowed here to insert.
★ Here the elements are by default insert in sorted order or ascending order.
★ Constructor of the class is Overloaded.

Treeset ():-

Treeset → Comparable interface - present in java.lang packa
When we call this constructor it creates an empty Treeset Objec where the sorting order is natural sorting order.

Note:-

→ In simple word when we insert elements in the Treeset object in ascending order by using compareTo() method of comparable interface.
→ But for storing the element in asending order Treeset object imposses some restriction :-

1. Elements must be homogeneous otherwise the program terminates at the runtime by generating ClassCastException.

2. Don't add null value within TreeSet object otherwise the program terminates at the runtime by generating NullPointerException.

3. Don't add the element or object which is not inherit from Comparable interface otherwise the program terminates at the runtime by generating ClassCastException

Program:-

1.
```
import java.util.*;
public class TreeDemo1
{
    public static void main (String args[])
    {
      TreeSet ts = new TreeSet();
       ts.add("Java");
       ts.add("Python");
       ts.add("PHP");
       ts.add(new Integer(1));
       Sopln(ts);
    }
}
```

O/p- Exception in thread "main" java.lang.ClassCastException

2.
```
import java.util.*;
public class TreeDemo2
{
    public static void main (String args[])
    {
       TreeSet ts = new TreeSet();
       ts.add("Java");
       ts.add("Python");
       ts.add("PHP");
       ts.add(null);
    } Sopln(ts);
}
```

Output :- Exception in thread "main" java.lang.NullPointerException

3.
```
import java.util.*;
public class TreeDemo3
{
    public static void main (String args[])
    {
       TreeSet ts = new TreeSet();
       ts.add(new StringBuffer("Java"));
       ts.add(new StringBuffer("Python"));
       Sopln(ts);
    }
}
```

Output :- Exception in thread "main" java.lang.ClassCastExceptio
java.lang.StringBuffer cannot be cast to java.lang.Com

4.
```
import java.util.*;
public class TreeDemo
{
    public static void main (String args[])
    {
       TreeSet ts = new TreeSet();
       ts.add("Ankit");
       ts.add("Amit");
       ts.add("Ayush");
       ts.add("Alka");
       ts.add("Aviman");
       ts.add("Anu");
       Sopln("\n See the elements of TreeSet object.");
       Iterator ii = ts.iterator();
       while(ii.hasNext())
       {
          String s = (String) ii.next();
          Sopln(s);
       }
}
```

Output :- See the elements of Treeset object.
Alka
Amit
Ankit
Anu
Aviman
Ayush

COMPARABLE INTERFACE :-

* It is a pre-declared interface present in java.lang package.
* In java it is an functional interface. As it is an functional interface it contains only one abstract method.

```
public abstract int compareTo (object Obj);
obj.compareTo (obj2);
```

* If it return negative (-ve) value then obj1 comes before obj2.
* "Alka".compareTo ("Anu");

  Alka
  Anu
* If it return positive (+ve) value the obj2 comes before obj1.

```
"Binit".compareTo ("Anu");
```

  Anu
  Binit

* On the otherhand if it return '0' then it is a duplicate object.

```
"JT".compareTo ("JT");
```

duplicate object.

Program :-

```
import java.util.*;
public class TreeDemo
{
  public static void main (String args[])
  {
    TreeSet ts = new TreeSet ();
    ts.add ("Z");
    ts.add ("D");   // "Z".compareTo ("D"); // +ve value
    ts.add ("R");   // "D".compareTo ("R"); // -ve value.
                    // "Z".compareTo ("R"); // +ve value.
    Sop ln (ts);
  }
}
```

Output :- [D, R, Z]

NOTE :-

* If we are not satisfied with the default natural sorting order or ascending order then we can defined our own customized sorting order by inheriting from java.util.comparator interface.
* By using the comparator interface programmer can sort the elements as per the requirement of the client.
* No pre-defined class inherit from comparator interface.
* Comparator interface having 2 methods.

1. public abstract boolean equals (Object obj) :-
   When a class inherit from comparator interface the we are bound a override compare method but we may override the equals method or may not.
2. public abstract int Compare (Object obj1, Object obj2) :-
   * If this method returns -ve value then obj1 comes before obj2.
   * If this method return +ve value then obj2 comes before obj1.
   * If it return 0 then both are duplicate elements.

DIFFERENCE BETWEEN COMPARE & COMPARATOR :-

* Comparable is a pre-declared interface present in java.lang package. where as comparator is a pre-declared interface in java.util package.
* Comparable supports natural sorting order whereas comparator supports customized sorting order.
* Comparable having only one method where as comparator having 2 methods.
* String and all wrapper classes inherit from comparable interface whereas no-concrete class inherit from comparator interface.

Treeset :-

* When we construct an object of the treeset interface by calling non-parameterized constructor then by-default it supports Comparable Interface.
* When it supports comparable interface then each element are compared by using CompareTo() method of comparable interface. so each elements in the treeset object is stored in ascending order.

NOTE:-

* Don't add the elements in the Treeset object when we call non-parameterial constructor of Treeset class which is not inherit from comparable interface otherwise compareTo() never implemented on that object. So program Terminates at the runtime by generating classCastException.
* Treeset object also holds the elements in ~~sort~~ customized sorting order.
* When we add comparator interface object as a parameter in the treeset constructor then it supports customized sorting order.

Program:-

```
import java.util.*;
class Demo implements Comparator
{
  @Override
  public int compare(Object obj1, Object obj2)
  {
    Integer i1 = (Integer) obj1;
    Integer i2 = (Integer) obj2;
    if (i1<i2)
      return 1;
    else if (i1>i2)
      return -1;
    else
      return 0;
  }
}
class Treeset2
{
  public static void main (String args[])
  {
    Treeset ts = new Treeset (new Demo());
    ts.add(new Integer(17));
    ts.add(new Integer(11));
    ts.add(new Integer(15));
    ts.add(new Integer(19));
    ts.add(new Integer(14));
    ts.add(new Integer(18));
    Sopln("\n See the elements.\n");
    Iterator ii = ts.iterator();
    while(ii.hasNext())
    {
      Integer aa = (Integer) ii.next();
      Sopln(aa);
    }
  }
}
```

Output: See the elements

```
19
18
17
15
14
11
```

Program:-

```
import java.util.*;
class Demo1 implements Comparator
{
  @Override
  public int compare(Object obj1, Object obj2)
  {
    String s1 = obj1.toString();
    String s2 = obj2.toString();
    int a = s1.length();
    int b = s2.length();
    if(a<b)
      return -1;
    else if(a>b)
      return 1;
    else
      return (s1.compareTo(s2));
  }
}
public class Treeset3
{
  public static void main(String args[])
  {
    Treeset ts = Treeset(new Demo1());
    ts.add(new String("India"));
    ts.add(new String("Pakistan"));
    ts.add(new String("Afganistan"));
    ts.add(new String("china"));
    ts.add(new String("Japan"));
    ts.add(new StringBuffer("Cuba"));
    ts.add(new StringBuffer("Nepal"));
    ts.add(new String("USA"));
    ts.add(new String("Russia"));
    Sopln("\n See the elements.\n");
    Iterator ii = ts.iterator();
    while(ii.hasNext())
    {
      System.out.println(ii.next());
    }
  }
}
```

QUEUE :-

* Queue is a pre-declared interface present in java.util package.
* It is the child interface of collection, if we want to represent a group of individual objects prior to processing for holding the elements then we should go for queue object.
* Queue usually supports FIFO but in collection framework based on our requirement we can change the order by using comparator interface.
* As queue is the child interface of collection it also hold object as an elements.

METHODS :-

1. public abstract boolean offer (Object obj) :-
   → It add the elements in the queue object.
2. public abstract Object peek() :-
   → It extract the head element from the queue object if the queue object is empty then this method return null.
3. public abstract Object element();
   → It also performed like peek() method used to extract the element but when the queue object is in empty then program terminate at the runtime by generating NoSuchElementException.
4. public abstract Object poll() :-
   → It is used to remove the head element from the queue
5. public abstract Object remove();
   → It also remove the element from the queue.

PRIORITY QUEUE :-

* It is a pre-defined class present in java.util package.
* This class inherit from abstract queue class of queue interface
* This underlying datastructure supported by priority queue Object is queue.
* This is the datastructure which hold a group of individual object prior to processing according to same priority.
* The priority can be natural sorting order or customized sorting order.

* Insertion sorting order are not preserved.
* We can't add null values are within it.
* Duplicate elements are not be hold be priority queue object.

CONSTRUCTOR :-

The constructor of the class is overloaded.

1. PriorityQueue() :-
   It construct the priorityQueue object whose initial size 0, but default capacity is 11 and fill ratio 75%. (3/4)
2. priority Queue (int capacity)
3. priority Queue (comparator obj)
4. priorityQueue (collection obj)

METHODS :-

It override each and every method of Queue interface

Program :-

```
import java.util.*;
public class QueueDemo
{
    public static void main (String args[])
    {
        priorityQueue p = new priorityQueue ();
        Sopln ("Element is :" +p.peek());
        for (int i=1; i<=10; i++)
        {
            P. offer (new Integer(i));
        }
        System.out.println (p);
        System.out.println (p.poll());
        System.out.println (p);
    }
}
```

Output :-

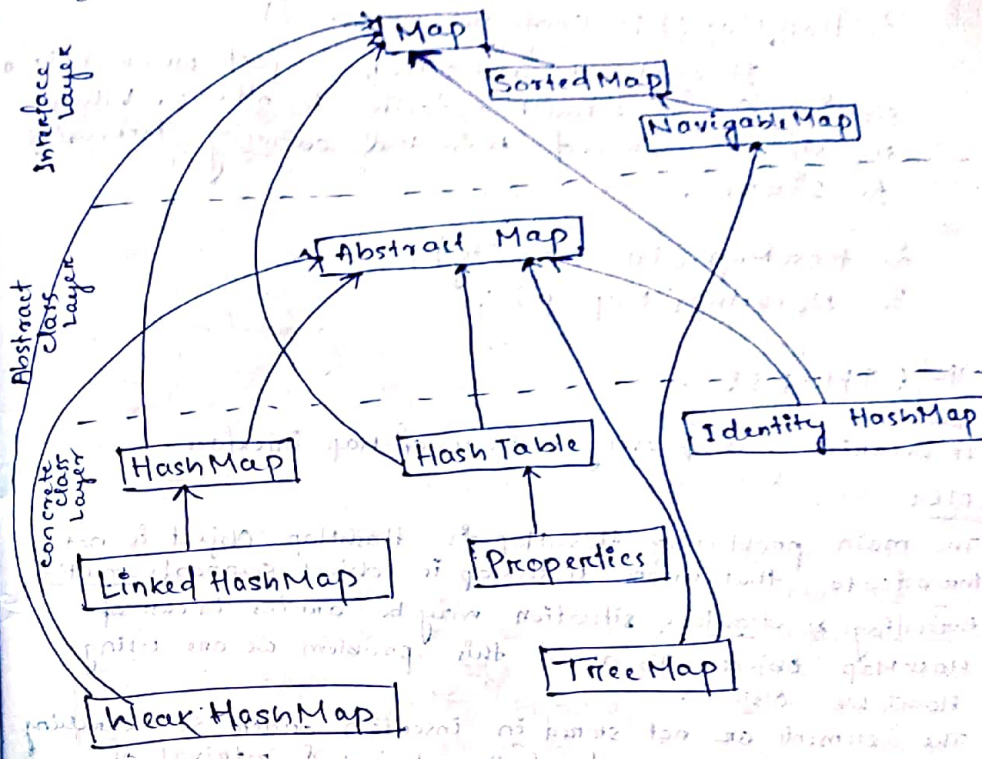
```
Element is : null
[1,2,3,4,5,6,7,8,9,10]
1
[2,4,3,8,5,6,7,10,9]
```

MAP INTERFACE :-

* If we want to represent the group of objects as sorted in key and value pair then we construct the object of map interface.
* Map is a pre-defined interface present in java.util package.
* Map is a core interface in collection framework which is popularly known as entity.
* Map object never contains duplicate key but it contains duplicate value.
* In case of map interface we extract the values by using key and we remove the value by using key and we check the values by using key.

METHODS :-

1. public abstract Object put(Object key, Object value);
   → It is used to insert an elements in the map.
2. public abstract Object get(Object key);
   → It is used to extract the value from the map.
3. public abstract Object remove(Object key);
   → It is used to remove the key from the map.
4. public abstract boolean containsKey(Object key);
   → It check the specified key is available or not.
5. public abstract boolean containsValue(Object value);
   → It check the specified value is available or not.
6. public abstract int size();
   → It extract the size of the map.
7. public abstract boolean isEmpty();
   → It check the map is empty or not.
8. public abstract Set keySet();
   → It extract all the key available in the map & store it in the set object. As we know set contains unique elements and in the map the key are unique.
9. public abstract Collection values();
   → It extract all the values from map and store it in the collection.
10. public abstract void putAll(Map p);
    → It extract all the entity from the map & store it in another map.



HashMap Class :-

* It is a pre-defined class present in java.util pack
* It is the child class of Abstract map class of Map interface.
* The underlying datastructure supported by HashMap class in HashTable.

HashTable :-

* It is a collection that stores the elements in key and v pair.
* In HashMap object both the key and value are of Object type.
* We retrieve the value from the HashTable datastru by the help of key.
* In hashmap object key are must be unique where a the values can be identif different.

* Insertion order is not preserved because the values are stored by comparing the hashcode values of the key.
* The constructor of the class is overloaded.

1. HashMap() :- (non-parameterised)
   It construct a HashMap object whose initial size is '0' & default capacity is $2^4 = 16$. When the size is exceed then the capacity increased to $2^5 = 32$.
2. HashMap (int capacity);
3. HashMap (Map p);

METHODS :-

* It override each & every methods of Map interface.

NOTE :

* The main problem of HashMap is HashMap Object is not threadsafe that means HashMap is object supports multi-threading so deadlock situation may be arrises in case of HashMap object. To avoid this problem we are using HashTable Object.
* The elements are not stored in insertion order so searching is faster but removal of the element & retrival of element is not faster than LinkedHashMap object.
* LinkedHashMap stored the element in insertion order by the help of key.

LINKED HASHMAP CLASS :-

* It is a pre-defined class present in java.util package.
* It is the child class of HashMap.
* The underlying datastructure supported by LinkedHashMap Object is LinkedList and HashTable.
* As it supports HashTable datastructure it store the unique elements by comparing the key but by the help of LinkedList datastructure the elements are stored in insertion order.
* It is introduced in JDK 1.4.
* The main job of LinkedHashMap object it supports cache application where the duplicate elements are not allowed & insertion order is reserved.

CONSTRUCTOR :-

1. LinkedHashMap() :-
   → When we construct the object by calling this construct it construct a LinkedHashMap object whose initial size is '0' & default capacity is $2^4 = 16$ & fill ratio is 75% (3/4)
2. LinkedHashMap (int capacity);
3. LinkedHashMap (Map p);

METHODS :-

* It uses every method of HashMap class.

IDENTITY HASHMAP CLASS :-

* The main problem of HashMap object is it hold unique k but the uniqueness of the keys are matched by using a metho of object class.

```
public boolean equals (Object obj);
```

* The equals() method of Object class match the addres the object but as we know in java every class implici inherit from object class, so the child class is able t override the method of object class (base class).
* It is introduced in JDK 1.5 for avoiding the problem arising in the case of HashMap.

Program :-

(on HashMap)

```
import java.util.*;
public class Test
{
    public static void main (String args[])
    {
        HashMap map = new HashMap();
        map.put (new String("java"), new String("A"));
        map.put (new String("java"), new String("B"));
        map.put (new String("java"), new String("C"));
        System.out.println (map);
    }
}
```

Keys are matched by equals(). so it match the content because in override the method of object class.

Output = {java = C}

(updated value is print, last element is extract)

(On IdentityHashMap)

```
import java.util.*;
public class Test1
{
   public static void main(String args[])
   {
      IdentityHashMap map=new IdentityHashMap();
      map.put(new String("java"),new String("A"));
      map.put(new String("java"),new String("B"));
      map.put(new String("java"),new String("C"));
      System.out.println(map);
   }
}
```

Output :- {java=B, java=C, java=A}

(Here, keys are matched by == operator it check the address so keys must unique & output is not in insertion order because it supports Hashtable datastructure)

HashMap :-

```
import java.util.*;
public class HashMapDemo
{
   public static void main(String args[])
   {
      HashMap map=new HashMap();
      map.put(new String("AS"),new String("Java"));
      map.put(new String("AB"),new String("Spring"));
      map.put(new String("UC"),new String("Hibernate"));
      map.put(new String("RD"),new String("struct s"));
      map.put(new String("OE"),new String("Servlet"));
      map.put(new String("IF"),new String("Jsp"));
      map.put(new String("AG"),new String("EJB"));
      Sopln("\n See the Elements\n");
      Set ss=map.keySet(); // it extract all the unique keys and store it in set.
      Iterator ii=ss.iterator();
      while(ii.hasNext())
      {
         String key=(String)ii.next();
         String value=(String)map.get(key);
         Sopln("key value is:
```

Output :-

```
key value is : AB  value is : Spring
key value is : AS  value is : Java
key value is : RD  value is : struct s
key value is : OE  value is : EJB
key value is : IF  value is : Jsp
key value is : UC  value is : Hibernate
```

* On the above example, justified that the elements are not stored in insertion Order. But advantage is here Searchin an element is very fast it supports multithreading.
* To avoid the problem in JDK 1.9 java introduced LinkedHashMap.
* LinkedHashMap is a pre-defined class, present in java. package.
* It is the child class of Hashmap.
* The underlying datastructure supported by LinkedHash object is hashtable and LinkedList.

BENIFITS :-

1. This class object avoid the drawback of hashmap by storing the elements in the elements in the insertion o of the main area of LinkedHashMap is cache applicati
2. While the duplicate elements are not allowed and insertion order is preserved.

The constructor of the class is overloaded

(1) LinkedHashMap() :-

It construct a linkedHashMap object whose initial size is '0' where as default capacity = 16 & fill rat is 75% (3/4).

(2) LinkedHashMap(int capacity) :-

As a programmer. we can set the capacity.

(3) LinkedHashMap(Map map)

METHODS :-

* It used every method of HashMap.

PROGRAM :-

```
import java.util.*;
public class LinkedHashMapDemo
{
    public static void main (String args[])
    {
        LinkedHashMap map = new LinkedHashMap ();
        map.put (new String ("As"), new String ("Java"));
        map.put (new String ("AB"), new String ("Spring"));
        map.put (new String ("UC"), new String ("Hibernate"));
        map.put (new String ("RD"), new String ("Structs"));
        map.put (new String ("OE"), new String ("Servlet"));
        map.put (new String ("IF"), new String ("JSP"));
        map.put (new String ("AG"), new String ("EJB"));
        Sopln ("\n See the Elements\n");
        Set ss = map.keyset ();  // it extract all the unique keys & store it in set object
        Iterator ii = ss.Iterator ();
        while (ii.hasNext ())
        {
            String key = (String) ii.next ();
            String value = (String) map.get (key);
            Sopln ("Key Value:" + key + "Value is:" + value);
        }
    }
}
```

Output :-

```
Key value is : As  value is : Java
Key value is : AB  value is : Spring
Key value is : UC  value is : Hibernate
Key value is : RD  value is : Structs
Key value is : IF  value is : Servelet
Key value is : AG  value is : EJB
```

② Another drawback of HashMap is, it store the unique elements by comparing the keys values. But it compare the key values by the help of a pre-defined methods of object class.

```
public boolean equals (Object obj)
{
    // Object class
}
```

→ In object class equals() method match the address so when the keys points to different object even if the contents are same they are considered as the unique key.

Program :-

```
import java.util.*;
public class HashMapDemo1
{
    public static void main (String args[])
    {
        HashMap map = new HashMap ();
        map.put (new StringBuffer ("Java"), new String ("Indi
        map.put (new StringBuffer ("Java"), new String ("Jap
        map.put (new StringBuffer ("Java"), new String ("Chi
        // here the key comparision is happend by calling equals() method of object class.
        // the object class 'equals()' method match the address as address are different.
        // they are treated as different keys.
        Sopln ("In see the elements\n");
        Set ss = map.keySet ();
        Iterator ii = ss.iterator ();
        while (ii.hasNext ())
        {
            StringBuffer key = (StringBuffer) ii.next ();
            String value = (String) map.get (key);
            Sopln ("Key is:" + key + "Value is:" + value);
        }
    }
}
```

Output :-

```
See the elements
Key is : java  value is: India
Key is : Java  value is: China
Key is : Java  value is: Japan
```

Program:-
On the above program instead of StringBuffer if we write String:-

Output:- See the elements
Key is: Java value is: china

* To avoid the problem, Java introduced IdentityHashMap class in JDK - 1.4
* IdentityHashMap class inherite from AbstractMap class and Map interface.
* In case of identity hashmap object the keys are matched by equals operator. So here equals operator matched address in each and every key.
* The underlying datastructure supported by IdentityHashMap object is hashtable.
* The constructor of the class is Overloaded.
  (1) IdentityHashMap():-
  → When we call this constructor it consists identity hashMap object whose initial size is '0', capacity is 16 and fill ratio is 75% (3/4).
  (2) IdentityHashMap (int capacity);
  (3) IdentityHashMap (Map map);

METHODS:-
* It override each and every methods of Map interface.

Program:-

```
import java.util.*;
public class IdentityHashMapDemo
{
  public static void main(String args[])
  {
    IdentityHashMap map = new IdentityHashMap();
    map.put(new String("Java"), new String("India"));
    map.put(new String("Java"), new String("Japan"));
    map.put(new String("Java"), new String("china"));
    // here the key comparision is happed by calling
       == operator.
    Sopln("\n See the elements \n");
    Set ss = map.keySet();
    Iterator ii = ss.iterator();
    while(ii.hasNext())
    {
      String key = (String) ii.next();
      String value = (String) map.get(key);
      Sopln("Key is: " + key + " Value is: " + value);
    }
  }
}
```

Output:-
Key is: Java value is: Japan
Key is: Java value is: china
Key is: Java value is: India

WEAKHASHMAP CLASS:-
* It is a pre-defined class present in java.util package
* WeakHashMap inherit from AbstractMap class & Map interface.
* The underlying data-structure supported by WeakHashMap Object is HashTable.
* One of the problem of HashMap Object is it is not eligible for garbage collection even if the reference is null.

Program:-
HashMap Object Permanent Object.

```
import java.util.*;
class Temp
{
  @Override
  public String toString() // It is implicitly call by JVM when we print the value of a reference.
  {
    return "Java Technocrat";
  }
  @Override
  public void finalize() // It is implicitly called by garbage collector before it finalize the object.
  {
    Sopln("My job is over.");
  }
}
public class HashMapDemo3
{
  public static void main(String args[])
```

```
HashMap map = new HashMap();
Temp tt = new Temp();
map.put(tt, "Rashmi");
Sopln(map); // it call toString() method of Temp class
tt = null;
System.gc(); // I explicitly call garbage collector.
try{
    Thread.sleep(2000);
}catch(InterruptedException ie)
  {
  }
Sopln(map);
}
```

Output :- { Java Technocrat. = Rashmi }
{ Java Technocrat. = Rashmi }

Program :-
WeakHashMap Object

```
import java.util.*;
class Temp
{
  @Override
  public String toString()
  {
    return "Java Technocrat.";
  }
  @Override
  public void finalize()
  {
    Sopln("My job is Over");
  }
}
public class WeakHashMap
{
  public static void main(String args[])
  {
    WeakHashMap map = new WeakHashMap();
    Temp tt = new Temp();
    map.put(tt, "Rashmi");
    Sopln(map);
    tt = null;
    System.gc();
    try{
      Thread.sleep(2000);
    }catch(InterruptedException ie)
      {
      }
    Sopln(map);
  }
}
```

Output :-
{ Java Technocrat. = Rashmi }
My job is Over

* HashMapObject dominates garbage collection but when we construct WeakHashMap object then it is eligible for garbage collection, if the object have any external reference.
* Garbage collection dominates WeakHashMap object.
* The constructor of the class is overloaded.
  1. WeakHashMap() :-
     size = 0
     initial capacity = 16
     fill ratio = 75% (3/4)
  2. WeakHashMap(int capacity);
  3. WeakHashMap(Map p);

METHODS :-
It overrides each & every methods of Map interface.

HASHTABLE CLASS:-

* Another drawback of HashMap class is it not threadsafe because it supports multithreading.
* To avoid the problem of deadlock HashTable class is introd
* HashTable is a legacy class.
* HashTable object performed like HashMap object but the difference is HashTable supports single Thread model whereas HashMap object supports multithreading.
* HashTable Object is Threadsafe where as HashMap object is not threadsafe.
* From the performance point of view HashMap object performs is better than HashTable Object.
* HashTable is a legacy class where as HashMap is not a legacy class.
* Every methods of HashTable is synchronized where as every methods of HashMap is not synchronized.
* In HashTable both key & value are unique where as HashMap keys are unique, value can be duplicate.
* This class inherit from Abstractmap class & Map interface.
* The underlying data structure supported by HashTable object is HashTable.
* The constructor of the class is overloaded:
  1. HashTable ():-
     → It construct a hashtable object whose initial size is '0' & initial capacity is 11 & fill ratio is 75% (3/4)
  2. HashTable (int capacity):-
     → It construct a hashtable object by fixing the capacity.
  3. HashTable (Map p);

METHOD:-

public Synchronized Enumeration keys();

Program:-

```
import java.util.*;
public class HashtableDemo
{
    public static void main(String args[])
    {
        Hashtable table = new Hashtable();
        table.put(new String("Abc"), new String("India"));
        table.put(new String("ABD"), new String("Nepal"));
        table.put(new String("Bbc"), new String("China"));
        table.put(new String("Cbc"), new String("Japan"));
        table.put(new String("Dbc"), new String("Cuba"));
        Sopln("In Retrive the elements From hashtable Object.In
        Enumeration ee = table.keys();
        while(ee.hasMoreElements())
        {
            String key = (String)ee.nextElement();
            String value = (String) table.get(key);
            Sopln("Key is :"+key+" Value is :"+value);
        }
    }
}
```

PROPERTIES CLASS:-

* It is a pre-defined class present in java.util package.
* It is the child class of HashTable.
* It is also a legacy class.
* In our application if we want to change some sensitive information frequently.

Example:-

Credit card pin number, password, admin name etc. then it is always recommended to implement properties class.

* In our application if we specify the admin name & passwor in the Hardcode.

Hardcode
```
String username = "Rashmi";
String password = "Java Technocrat";
```

Then any changes in the hardcode must required recompilation where as if we specify the information in the properties file then the changes of the data never required recompilation.

* The extension of properties files are .properties
* The Constructor of the class is non-parameterised property

METHODS :-
It uses all the methods of HashTable class beyond that it has some methods.

1. public String getProperty (String name);
   → By specifying the property name we extract the value.
2. public void setProperty (String name, String value);
   → It store the property name & value in property file.
3. public Enumeration propertyNames();
   → It extract all the property are available in the property file.
4. public void store (OutStream os, String comment);
   → It store the information in the property.
5. public void load (InputStream is);
   → It extract the information from the property file

Program :-

```
import java.io.*;
import java.util.*;
public class PropertyDemo
{
    public static void main (String args[])
    {
        try{
            Properties p = new Properties();
            FileInputStream fis = new FileInputStream("
                                                  properties");
            p.store (Fos, "This is Java property class.");
        }catch (Exception ee)
            {
                ee.printStackTrace();
            }
    }
}
```

SORTEDMAP INTERFACE :- (57)

* It is a pre-declared interface present in java.util package
* It is the child interface of map interface.
* If you want to represent a group of entitys (key & values) according to sorting order than we should go for SortedMap interface.
* The sorting should be done based on keys.
* SortedMap interface bydefault stored the elements in ascending order by comparing the keys.

METHODS :-

1. public abstract Object firstKey();
   → It is used to extract the firstKey.
2. public abstract Object LastKey();
   → It extract the last key.
3. public abstract SortedMap headMap (Object key);
   → It compared the keys which is less than headd key.
4. public abstract SortedMap tailMap (Object key);
   → It check the key which is equal or more than the specified key.
5. public abstract SortedMap subMap (Object key1, Object k
   → It is more than key1 but equals & less than key2.
6. public abstract ~~construct~~ comparator comparator();
   → It is used construct a comparator object through which we can compare the elements as per customiz order.

NAVIGABLEMAP INTERFACE :-

* It is the child interface of sortedMap.
* As it is the child interface of sortedMap it also hold the elements in sorted order by comparing the keys.
* It also hold the unique elements by comparing the keys.
* This interface having some pre-declared Methods for navigation purpose.

METHODS :-

1. public abstract NavigableMap ceilingKey (Object key);
2. public abstract NavigableMap higherKey (Object key);
3. public abstract NavigableMap floorKey (Object key);
4. public abstract NavigableMap lowerKey (Object key);
5. public abstract NavigableMap pollFirstEntry();
   → It delete the first element.
6. public abstract NavigableMap pollLastEntry();
   → It delete the last element.
7. public abstract NavigableMap descendingMap();
   → It store the element in descending order.

PROGRAM :-

```
import java.util.*;
public class NavigableMap Demo
{
  public static void main (String args[])
  {
    TreeMap map = new TreeMap();
    map.put (new Integer (101), new String ("Java"));
    map.put (new Integer (102), new String ("Python"));
    map.put (new Integer (103), new String ("PHP"));
    map.put (new Integer (104), new String (".Net"));
    map.put (new Integer (105), new String ("Velocity"));
    map.put (new Integer (106), new String ("Android"));
    Sopln (map);
    Sopln (map.ceilingKey (103));
    Sopln (map.higherKey (103));
    Sopln (map.floorKey (103));
    Sopln (map.lowerKey (103));
    Sopln (map.pollFirstEntry());
    Sopln (map.pollLastEntry());
    Sopln (map);
  }
}
```

Output :-

{101 = Java, 102 = Python, 103 = PHP, 104 = .Net, 105 = velocity, 106 = And
103
104
103
102
101 = Java
106 = Android
{102 = Python, 103 = PHP, 104 = .Net, 105 = Velocity}

TREEMAP CLASS :-

* It is a pre-defined class present in java.util package.
* This class inherit from AbstractMap class & NavigableMap interface.
* As the class inherit from NavigableMap interface it not only hold the unique element by comparining the keys but also it store the elements bydefault in ascending order by comparing the keys.
* The underlying data-structure supported by TreeMap object is RED-BLACK TREE.
* We can't add the null values for keys otherwise the program is terminated at runtime by generating NullPointerException
* Don't add heterogenous elements for the keys otherwise the program is terminated at the runtime by generating classcastException.
* Don't add the key which is not inherit from comparable interface otherwise the program is terminated at the runtime by generating ClassCastException.
* The constructor of the class is Overloaded.

1. TreeMap () :-
   non-parameterized constructor.
   It construct a TreeMap object where the comparision of keys are possible by using compareTo() method of comparable interface
2. TreeMap (Map m) :-

Program :-

```
import java.util.*;
public class TreeMapDemo
{
  public static void main (String args[])
  {
    TreeMap map = new TreeMap();
    map.put (new Integer(105), "Sagar");
    map.put (new Integer(107), "Tanmay");
    map.put (new Integer(101), "Rahul");
    map.put (new Integer(103), "Ankit");
    map.put (new Integer(102), "Binit");
    // map.put (null, "Vivek"); // NullPointerException.
    // map.put ("Java", "Avik"); // ClassCastException
    Set ss = map.keyset();
    Iterator ii = ss.iterator();
    while (ii.hasNext())
    {
      Integer key = (Integer) ii.next();
      String value = (String) map.get (key);
      System.out.println ("key is :" +key+ "Value is:" +va
    }
  }
}
```

Output :-

Key is: 101 value is: Rahul
Key is: 102 value is: Binit
Key is: 103 value is: Ankit
Key is: 105 value is: Sagar
Key is: 107 value is: Tanmay

Program :-

```
import java.util.*;
class Demo implements Comparator
{
    @Override
    public int compare (Object obj1, Object obj2)
    {
        String s1 = obj1.toString();
        String s2 = obj2.toString();
        return s1.compareTo(s2);
        //return s2.compareTo(s1);
    }
}
public class TreeMapDemo1
{
    public static void main (String args[])
    {
        TreeMap map = new TreeMap (new Demo());
        map.put (new String ("abc"), "Java");
        map.put (new StringBuffer ("xyz"), "Python");
        map.put (new String ("aaa"), "Android");
        map.put (new String ("zzz"), "Spring");
        map.put (new StringBuffer ("zzz"), "Hibernate");
        Sopln (map);
    }
}
```

Output :-

{zzz = Hibernate, xyz = Python, abc = Android, zzz = Spring}

GENERICS :-

* It is a powerful feature introduced in JDK 1.5.
* It is introduced for collection framework to avoid the drawback of collection framework.
* Array having some advantage over collection framework.
* Array are always type shape application.
* In array we can give the guaranty for the type of elements available in inside the array.

for example = if the programmer requirement is to store String Object then we can choose the array of String type.

```
String str[] = new String[5];
```

* But if you want to add any other element in the String type array than the program must terminate at the compiler time because the compiler check the type of array element.

* In array we give the guaranty that String array contains only String elements because compiler check the elements at the time of compilation of Java compilation of Java program so, array is typeshape.

DRAWBACK OF COLLECTION :-

* But on the otherhand collections are not typeshape because collection hold heterogeneous elements. At the time compilation of the collection application the compiler never check the type of elements so, collection applications are not typeshape. When we retrieve the elements in case of collection application program may terminate at runtime by generating ClassCastException.
* When we retrive the elements from the array we never required type casting because array holds homogeneous elements but on the otherhand when we retrive the element from the collection we must required type casting.

Program :-

```
import java.io.*;
public class Demo1
{
    public static void main (String args[])
    {
        String str[] = new String[3];
        Str[0] = "Red";
        Str[1] = "Green";
        Str[2] = "Blue";
        String s1 = Str[0];
        String s2 = Str[1];
        String s3 = Str[2];
        Sopln (s1 + "\t" + s2 + "\t" + s3);
    }
}
```

Output :- Red Green Blue

Program :-

```
import java.util.*;
public class Demo2
{
    public static void main (String args[])
    {
        ArrayList aa = new ArrayList();
        aa.add ("Red");
        aa.add ("Green");
        aa.add (new Integer (1));
        Sopln ("\nSee the elements of Arraylist object.\n");
```

```
for (int i=0; i<aa.
{
    String s = (String) aa.get (i);
    Sopln (s);
}
}
}
```

* Two overcome two problems of collection java introduced Generics in collection framework.

Program :-

```
import java.util.*;
public class Demo3
{
    public static void main (String args[])
    {
        ArrayList <String> aa = new ArrayList <String> ();
        aa.add ("Red");
        aa.add ("Green");
        aa.add ("Blue");
        Sopln ("\nSee the elements of ArrayList Object:");
        for (int i=0; i<aa.size(); i++)
        {
            String s= aa.get (i);
            Sopln (s);
        }
    }
}
```

* Generics is a powerful feature of java introduced only for collection.
* In simple words generics are parameterised types.
* Generics are important because they enable the programmer for dealing the classes, interface & methods.
* In java generics parameters are only object type.
* In java generics parameters can't be primitives type:

```
ArrayList <int> aa = new ArrayList <int> (); // error
ArrayList <Integer> aa = new ArrayList <Integer> ();
```

* In generics whatever the parameter we specify we are bound to add that type of parameter.
* If we add some other elements then compiler generics in collection applications the collection applications are types-sharing.
* When we retrieve the elements from the collection object we never required typecasting.

Program :-

```
import java.util.*;
public class Demo4
{
    public static void main (String args[])
    {
        HashMap <Integer, String> map = new HashMap <Integers

        map.put (new Integer (104), new String ("Python"));
        map.put (new Integer (101), new String ("Java"));
        map.put (new Integer (103), new String (".Net"));
        map.put (new Integer (102), new String ("PHP"));
        Set <Integer> ss = map.keySet ();
        Iterator <Integer> ii = ss.iterator ();
        while (ii.hasNext())
        {
            Integer key = ii.next ();
            String value = map.get (key);
            Sopln ("Key: "+key+ "Value: "+value);
        }
    }
}
```