COLLECTION FRAMEWORK

Collection Framework:-

- It contains several classes and interface which can be used to represent a group of individual object as single entity.
- It provides good architecture to store and manipulate the group of object.
- Collection Framework related all classes and interfaces present java.util package.

The key Interfaces of collection Framework.

1) Collection 2) List 3) Set 4) SortedSet 5) NavigableSet 6) Queue 7)Map 8) SortedSet 9) Navigable Map

Hierarchy of Collection Framework:-

- Collection (Interface 1.2v)
    1. List(I 1.2v)
        - A. ArrayList(C 1.2v)
        - B. LinkedList(C 1.2v)
        - C. Vector(C 1.0v)
            - a. Stack(C 1.0v)
    2. Set(1.2v)
        - A. HashSet(C 1.2v)
        - B. LinkedHashSet(C 1.4v)
        - C. SortedSet(I 1.2v)
            - a) NavigableSet(I 1.2v)
            - b) TreeSet(C 1.2v)
    3. Queue(I)
        - A. PriorityQueue(C 1.5v)
        - B. BlockingQueue(I 1.5v)
            - a) PriorityBlockingQueue(C 1.5v)
            - b) LinkedBlockingQueue(C 1.5)
- Map(I 1.2)
    1. HashMap(C 1.2v)
        - A. LinkedHashMap(C 1.4v)
    2. SortedMap(I 1.2v)
        - A. NavigableMap(I 1.6v)
        - B. TreeMap(C 1.2v)
    3. IdentityHashMap(C 1.4)
    4. WeakHashMap(C 1.2v)
    5. Dictonary(AC 1.0v)
        - A. HashTable(C 1.0v)
        - B. Properties(c 1.0v)

Characteristics of Collection Frame Work classes:-

1) Classes introduce version:-Different classes introduce in different version of java
   Example:-vector class introduce in 1.0 version of java and TreeMap introduce in 1.2 version of java.
2) Collection class underlying data structures:-Every collection class contains
3) Heterogeneous data allowed or not.
4) Insertion order is preserved or not:-In which order we inserting element same order we get the result or not.

   Example:-
   Insertion order:-o1,o2,o3. Result:-o1,o2,o3 (Insertion order preserved)
   Insertion order:-o1,o2,o3.Result:-o3,o1,o2 (insertion order not preserved)
5) Duplicate object insertion is allowed or not:-we can insert same value more one times or not.
6) Null value insertion is allowed or not.
7) Collection class method is synchronized or not (Collection object is thread safe or not).
8) Implements interface Serializable , Cloneable and RandomAccess.
9) Collection class support cursor.

Collection Interface(1.2 version):-If We want to represent a group of individual object as a single entity then we used Collection class.

The most common method which are applicable for all collection class defined inside collection interface.

Method of Collection Interface:-

A) boolean add(Object o)
B) boolean addAll(Collection c)
C) boolean remove(Object o)
D) boolean removeAll(Collection c)
E) boolean contains(Object o)
F) boolean containAll(Collection c)
G) boolean reatainAll(Collection c)
H) boolean isEmpty()
I) void clear()
J) int size();
K) Object [] toArray()

L) Iterator iterator()

List(I 1.2v):-It is child interface of collection. We used list interface, If we want to represent a group of individual object as a single entity.

List interface common property:-

1) All List interface implementation classes allowed Heterogeneous element insertion.
2) All implementation classes allow null insertion.
3) All implementation classes allowed duplicate insertion.
4) All implementation classes insertion order preserved.

Implementation class of List interface:-

1) ArrayList 1.2v
2) LinkedList 1.2v
3) Vector 1.0v
4) Stack 1.0v

List interface contain following method:-

A) void add(int index,Object obj)
B) boolean addAll(int index,Collection c)
C) Object get(int index)
D) Object remove(int index)
E) Object set(int index, Object new):-Replace and return old values.
F) int indexOf(Object o):-Return index of first occurrence 'o' .
G) int lastIndex(Object o)
H) ListIterator listIterator()

java.util.ArrayList class characteristics :-

A) Introduce in 1.2 version of java.
B) The underlying data structure is resizable or growable array.
C) ArrayList stored Heterogeneous object.
D) Insertion order preserved.
E) Duplicate element object insertion is allowed.
F) In ArrayList it is possible to insert null Object.
G) ArrayList methods are non-synchronized.
H) ArrayList class implements Serializable, Cloneable and RandomAccess.
I) We use iterator() and listIterator() method to retrieve elements from ArrayList.

Constructor of ArrayList:-

1) ArrayList al =new ArrayList();create ArrayList object with default capacity 10 once it reach its maximum capacity then size is automatically increase by:-
   new capacity=(old Capacity*3/2)+1
2) ArrayList al=new ArrayList(int initial capacity);
3) ArrayList al =new ArrayList(Collection c);Create an equivalent ArrayList object the given collection.

Example:-

```
import java.util.*;
class ArrayListDemo
{
public static void main(String[]args)
{
ArrayList al=new ArrayList();
al.add("vikas");
al.add(null);
al.add(null);
al.add("vivek");
al.add(10);
System.out.println(al);
al.remove(1);
al.add(1,"bishal");
System.out.println(al);
}
}
```

Result:-

```
[vikas, null, null, vivek, 10]
[vikas, bishal, null, vivek, 10]
```

Remove the element from ArrayList:-

Example:-

In this example when we remove the data by passing numeric or character value that is by default treated as index value.

```
import java.util.*;
class ArrayListDemo
{
public static void main(String[]args)
{
ArrayList al=new ArrayList();
al.add('a');
```

```
al.add('b');
al.add(1);//1 is taken as index value
al.add(10.52);
al.add('a');
al.add('z');
System.out.println(al);
al.remove(1);
al.remove(10.52);
System.out.println(al);
}
}
```

Result:-

```
[a, b, 1, 10.52, a, z]
[a, 1, a, z]
```

Example:-

If we want to remove 1 Integer value from ArrayList then we passed wrapper form of 1(one).

```
import java.util.*;
class ArrayListDemo
{
public static void main(String[]args)
{
ArrayList al=new ArrayList();
al.add('a');
al.add('b');
al.add(1);
al.add(10.52);
al.add('a');
al.add('z');
System.out.println(al);
Integer i=new Integer(1);
al.remove(i);
al.remove(10.52);
System.out.println(al);
}
}
```

Result:-

```
[a, b, 1, 10.52, a, z]
[a, b, a, z]
```

Find the capacity of ArrayList:-

Example:-

```
import java.util.*;
import java.lang.reflect.Field;
class ArrayListDemo
{
public static void main(String[]args)throws Exception
{
ArrayList al=new ArrayList(5);
for(int i=0;i<10;i++)
{
al.add(i);
System.out.println("Size="+al.size()+"capacity="+getcapacity(al));
}
}
static int getcapacity(ArrayList l)throws Exception
{
Field f=ArrayList.class.getDeclaredField("elementData");
f.setAccessible(true);
return ((Object[])f.get(l)).length;
}
}
```

Result:-

```
Size=1capacity=5
Size=2capacity=5
Size=3capacity=5
Size=4capacity=5
Size=5capacity=5
Size=6capacity=7
Size=7capacity=7
Size=8capacity=10
Size=9capacity=10
Size=10capacity=10
```

Java.util.LinkedList:-

A) Introduce in 1.2 version of java.
B) The underlying data structure is double linkedlist.
C) LinkedList can store Heterogeneous object.
D) Insertion order is preserved.
E) Duplicate object insertion is allowed.
F) In LinkedList it is possible to insert null Object.

G) LinkedList methods are non-synchronized.
H) ArrayList class implements Serializable, Cloneable but not RandomAccess.
I) We use iterator() and listIterator() method to retrieve elements from LinkedList.

Note:-LinkedList is best choice if our frequent operation is insertion and deletion in middle and it is worst choice if our frequent operation retrieves.

Constructor of LinkedList class:-

LinkedList ll=new LinkedList();:-Empty LinkedList object will be created.

LinkedList ll=new LinkedList(Collection c):-LinkedList object will be created for given collection.

Method of LinkedList class :-

A) public void addFirst()
B) public void addLast()
C) public Object getFirst()
D) public Object getLast()
E) public Object removeFirst()
F) public Object removeLast()

Difference between ArrayList and LinkedList:-

ArrayList	LinkedList
1.Underlying data structure is resizable or growable Array.	1. Underlying data structure is double linkedList.
2.when our frequent operation is retrieval then then ArrayList is best choice.	2.when our frequent operation is insertion and deletion from the middle then LinkedList is best choice.
3.ArrayList is worst choice if our frequent operation is insertion or deletion in middle because when we delete or insert in the middle then several sift operation performed internally.	3.LinkedList is worst choice if our frequent operation is retrieval operation.
4.In ArrayList element will be stored in consecutive memory location so retrieval operation become simple.	4.In LinkedList element not stored in consecutive memory location so retrieval operation is very difficult.

Example:-

```
import java.util.*;
class LinkedListDemo
{
public static void main(String[]args)
{
LinkedList ll=new LinkedList();
ll.add("vikas");
ll.add(10);
ll.add("bishal");
ll.add("vikas");
ll.add(null);
System.out.println(ll);
ll.add(0,"software");
System.out.println(ll);
ll.removeLast();
ll.addFirst("ccc");
System.out.println(ll);
}
}
```

Result:-

```
[vikas, 10, bishal, vikas, null]
[software, vikas, 10, bishal, vikas, null]
[ccc, software, vikas, 10, bishal, vikas]
```

Java.util.Vector:-

A) Introduce in 1.0(legacy) version of java.
B) The underlying data structure is resizable or growable Array.
C) Vector can store Heterogeneous object.
D) Insertion order is preserved.
E) Duplicate object insertion is allowed .
F) In Vector it is possible to insert null Object.
G) Vector methods are synchronized so it is thread safe.
H) Vector class implements Serializable, Cloneable and RandomAccess.
I) We use enumeration(), iterator() and listIterator() method to retrieve elements from Vector.

Vector class constructor:-

1) Vector v=new Vector(); Create an empty vector object with default capacity 10 once its reaches its maximum capacity it

means when we trying to insert 11 element that capacity will become double[20]
New capacity=current capacity*2.

2) Vector v=new Vector(int initial capacity);
3) Vector v=new Vector(int initial capacity, int increment);
4) Vector v=new Vector(Collection c);Create an equivalent vector object for the given collection this constructor meant for inter conversion between collection object.

Vector Specific Method:-

A) addElement(Object o)
B) removeElement(Object o)
C) removeElementAt(int index)
D) removeAllElements()
E) elementAt(int index)
F) firstElement()
G) lastElement()
H) int size()
I) int capacity()
J) Enumeration elements();

Example:-

```
import java.util.*;
class VectorDemo
{
public static void main(String[]args)
{
Vector v=new Vector();
System.out.println(v.capacity());
for(int i=1;i<=10;i++)
{
v.addElement(i);
}
System.out.println(v.capacity());
v.addElement(11);
System.out.println(v.capacity());
System.out.println(v);
}
}
```

```
Result:- 10
10
20
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

Difference between ArrayList and Vector:-

ArrayList	Vector
1.ArrayList introduce in 1.2 version of java	1.Vector introduce in 1.0 version of java and it is legacy.
2.Every method present inside Aarraylist is non-synchronized.	2.Every method present inside Vector is synchronized.
3.At a time multiple thread can operate on ArrayList object and hence it is not thread safe.	3.At a time only one thread can operate on Vector object and hence it is thread safe.
4.Relatiely performance is high because multiple thread allowed to execute on ArrayList Object.	4.Relatively performance is low because at a time only one thread allowed to execute on Vector object.

java.util.Stack:-

1) It is child class of Vector .
2) It is specially design class for last in first out order(LIFO).

Constructor:-

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

Method of Stack class:-

A) public Object push(Object o):-To insert an object into the stack.
B) public Object pop():-To Remove and return top of stack.
C) public object peek():-Return top of the stack without remove.
D) Int search(Object o):Return offset if the element is available other wise return -1.

Example:-

```
import java.util.*;
class StackDemo
{
public static void main(String[]args)
{
Stack s=new Stack();
s.push("vikas");
s.push("bishal");
s.push("vivek");
s.push("yogesh");
s.push("uday");
System.out.println(s);
```

```
System.out.println(s.search("vikas"));
System.out.println(s.size());
System.out.println(s.peek());
s.pop();
System.out.println(s);
}
}
```

```
Result:-
[vikas, bishal, vivek, yogesh, uday]
5
5
uday
[vikas, bishal, vivek, yogesh]
```

Cursor:- If want retrieve the object from the collection one by one then we used cursor. In java there are three types of cursor.

1) Enumeration
2) Iterator
3) ListIterator

Enumeration :-

- Enumeration interface introduce in 1.0 version of java so it is legacy class.
- We used Enumeration to retrieve the object only from legacy collection class like Vector, Stack etc so it is not universal cursor because we can use this cursor only for legacy class .
- We can create Enumeration class object by using elements() method.

Syntax of elements() method:-

```
public Enumeration elements()
```

Example:-

```
Vector v=new Vector();
Enumeration e=v.elements();
```

- It contain two method.
  1) public abstract boolean hasMoreElements()
  2) public abstract object nextElement()
- We can use this cursor only for read operation.
- Through this we can retrieve element only forward direction.

Example:-

```
import java.util.*;
class EnumerationDemo
{
public static void main(String[]args)
{
Vector v=new Vector();
for(int i=0;i<10;i++)
{
v.add(i);
}
System.out.println(v);
Enumeration e=v.elements();
while(e.hasMoreElements())
{
Integer I=(Integer)e.nextElement();
if(I%2==0)
{
System.out.print(I);
}
}
}
}
Result:- [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
         02468
```

Limitation of Enumeration cursor:-

1) We apply only Legacy collection class.
2) We can perform only read operation.
3) We can performed remove operation.

Overcome these problem we used Iterator cursor:-

Iterator:-

- Iterator interface introduce in 1.2 version of java and it is not legacy class.
- We can use Iterator to retrieve the object from all collection class so Iterator cursor is considered as universal cursor.
- We can create Iterator class object by using iterator() method.
  Syntax of iterator() method:
  public Iterator iterator()

Example:-
Vector v=new Vector();
Iterator itr=v.iterator();//v can be any collection class object.

- **It contain three method:-**
  1) public abstract boolean hasNext()
  2) public abstract Object next()
  3) public abstract void remove()
- We can use this cursor only for read and remove operation.
- Through this we can retrieve element only forward direction.

Example:-

```
import java.util.*;
class IteratorDemo
{
public static void main(String[]args)
{
Vector v=new Vector();
for(int i=0;i<10;i++)
{
v.add(i);
}
System.out.println(v);
Iterator itr=v.iterator();
while(itr.hasNext())
{
Integer I=(Integer)itr.next();
if(I%2==0)
System.out.print(I);
else
itr.remove();
}
System.out.println(v);
}
}
```

Result:-

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
02468
[0, 2, 4, 6, 8]
```

Limitation of Iteratror cursor:-

1) By using Enumeration and Iterator cursor we can retrieval element only forward direction and we can't retrieve element backward(reverse) direction. So Enumeration and Iterator are unidirectional cursor.
2) By using Enumeration and Iterator cursor we can performed read and remove operation only we can't performed replacement and add new element.
3) Overcome these limitation we used ListIterator cursor.

ListIterator:-

- By using ListIterator cursor we can perform read , remove ,set and insert operation.
- We can retrieve the element from in both direction (forward direction and backward(reverse)direction) .
- ListIterator introduce in 1.2 version of java so it is not legacy.
- We can create ListIterator class object by using listIterator() method.

  Syntax of listIterator() method:-

  public ListIterator listIterator()

  Example:-

```
Vector v=new Vector();
ListIterator itr=v.listIterator();//v can collection class object reference.
```

- **It contain total nine method:-**

  1) public abstract boolean hasNext()
  2) public abstract Object next() } Forward direction
  3) public abstract int nextIndex()
  4) public abstract boolean hasPrevious()
  5) public abstract Object previous() } Backward direction
  6) public abstract int previousIndex()
  7) public abstract void remove()
  8) public abstract void add(Object obj) } Extra activity
  9) public abstract void set(Object o)

Example:-

```
import java.util.*;
class ListIteratorDemo
{
public static void main(String[]args)
{
```

```
LinkedList ll=new LinkedList();
ll.add("vikas");
ll.add("vivek");
ll.add("bishal");
ll.add("uday");
ll.add("yogesh");
System.out.println(ll);
ListIterator itr=ll.listIterator();
while(itr.hasNext())
{
String str=(String)itr.next();
if(str.equals("uday"))
{
itr.remove();
System.out.println(ll);
}
else if(str.equals("yogesh"))
{
itr.set("ravi");
}
}
System.out.println(ll);
}
}
```

Result:-

```
[vikas, vivek, bishal, uday, yogesh]
[vikas, vivek, bishal, yogesh]
[vikas, vivek, bishal, ravi]
```

Retrieving object of collection classes:-There are three ways to retrieve the object from collection class.

1) By using for-each loop.
2) By using get() method.
3) By using cursors.

Example:-

```
import java.util.*;
class RetrieveObjectDemo
{
public static void main(String[]args)
{
ArrayList al=new ArrayList();
al.add("vikas");
```

```
al.add("vivek");
al.add("bishal");
al.add("ravi");
//Ist approach to retrieve Object from collection.
System.out.println("Ist approach to retrieve Object from
collection.");
for(Object o:al)
{
System.out.println((String)o);
}
//2nd approach to retrieve Object from collection.
System.out.println("2nd approach to retrieve Object from
collection.");
int size=al.size();
for(int i=0;i<size;i++)
{
System.out.println((String)al.get(i));
}
//3rd approach to retrieve Object from collection.
System.out.println("3rd approach to retrieve Object from
collection.");
Iterator itr=al.iterator();
while(itr.hasNext())
{
String s1=(String)itr.next();
System.out.println(s1);
}
}
}
```

Result:-

```
Ist approach to retrieve Object from collection.
vikas
vivek
bishal
ravi
2nd approach to retrieve Object from collection.
vikas
vivek
bishal
ravi
3rd approach to retrieve Object from collection.
vikas
vivek
```

bishal
ravi

Set interface:-

- Introduce 1.2 version of java.
- It is child interface of Collection interface.
- If we want to represent a group of individual object as single entity where duplicate are not and insertion order is preserved then we should go for Set interface.
- It is not contain any new separate method. We have to use only Collection interface method.

HashSet:-

A) Introduce in 1.0 version of java.
B) Underlying data structure is HashTable.
C) HashSet can store heterogeneous object.
D) Insertion order is not preserved it based on the hash code of the object(hashing mechanism).
E) Duplicate object insertion is not allowed.
F) Null insertion is possible but only once.
G) HashSet method is non-synchronized so it is not thread safe.
H) HashSet class implements Serializable, Cloneable interface but not implements RandomAccess interface.
I) We used only Iterator interface to retrieve the elements from HashSet.

HashSet class constructor:-

1) HashSet hs=new HashSet();:-Create HashSet Object with default capacity 16 and default fill ratio 0.75.
   Fill Ratio:-After filling how much ratio of an object a new object will be created this ratio is called Fill Ratio.
2) HashSet hs=new HashSet(int initialcapacity);
3) HashSet hs=new HashSet(int initialcapacity, float fillratio);
4) HashSet hs=new HashSet(Collection c);

Note:-Set interface and HashSet , LinkedHashSet class not contain any new method it used Collection interface method.

Note:-Duplicate object insertion is not allowed in HashSet when we try to insert duplicate value then we won't get any compile time or runtime error and add() method just return false.

Example:-Insert duplicate object inside HashSet.

```
import java.util.*;
class HashSetDemo
{
public static void main(String[]args)
{
HashSet hs=new HashSet();
hs.add("A");
hs.add("B");
System.out.println(hs.add("A"));
System.out.println(hs);
}
}
```

Result:-

```
false
[A, B]
```

LinkedHashSet:-

A) Introduce in 1.4 version of java.
B) Under lying data structure is LinkedList and HashSet.
C) LinkedHashset can store Heterogeneous objects.
D) Insertion order preserved.
E) Duplicate object insertion is not allowed.
F) Null insertion is possible but only once.
G) LinkedHashSet methods are non-synchronized.
H) LinkedHashSet class implements Serializable, Cloneable but not implements RandomAccess.
I) We can use only iterator() method to retrieve object from LinkedHashSet.

LinkedHashSet class constructor:-

1) LinkedHashSet lhs=new LinkedHashSet() Constructs a new, empty linked hash set with the default initial capacity (16) and load factor (0.75).
2) LinkedHashSet lhs=new LinkedHashSet(int initialCapacity)
   Constructs a new, empty linked hash set with the specified initial capacity and the default load factor (0.75).
3) LinkedHashSet lhs=new LinkedHashSet(int initialCapacity, float loadFactor)
   Constructs a new, empty linked hash set with the specified initial capacity and load factor.
4) LinkedHashSet lhs=new LinkedHashSet(Collection c)

Constructs a new linked hash set with the same elements as the specified collection

Note:-Set interface and HashSet , LinkedHashSet class not contain any new method it used Collection interface method.

Example:-In this example insertion order is preserved and duplicate insertion is not allowed.

```
import java.util.*;
class LinkedHashSetDemo
{
public static void main(String[]args)
{
LinkedHashSet lhs=new LinkedHashSet();
lhs.add("vikas");
lhs.add("vivek");
lhs.add("yogesh");
lhs.add(null);
lhs.add("ravi");;
System.out.println(lhs);
System.out.println(lhs.add(null));
System.out.println(lhs);
}
}
```

Result:-

```
[vikas, vivek, yogesh, null, ravi]
false
[vikas, vivek, yogesh, null, ravi]
```

Difference between HashSet and LinkedHashSet:-

HashSet	LinkedHashSet
1.HashSet introduce in 1.2 version of java	1.LinkedHashSet introduce 1.4 version of java
2. Underlying data structure HashTable.	2.Underlying data lying data structure is combination of LinkedList and HashTable.
3.Insertion order is not preserved.	3.Insertion order is preserved.

Note:-In general we can used LinkedHashSet to developed cache based application where duplicate are not allowed and insertion order is preserved.

SortedSet:-

A) Introduce in 1.2 version of java.
B) It is child interface of Set interface.
C) If we want represent a group of individual object according to some sorting order without duplicate then we used SortedSet.

SortedSet specific method:-

1) public abstract Object first() return first element of SortedSet.
2) public abstract Object last() return last element of SortedSet.
3) public abstract SortedSet headset(Object obj) return SortedSet whose elements are strictly less than obj.
4) public abstract SortedSet tailSet(Object obj) return SortedSet whose elements are greater than or equal obj(>=obj).
5) public abstract SortedSet subset(Object obj1,Object obj2):- return SortedSet whose elements are >= obj1 and < obj2.
6) public abstract Comparator comparator() return Comparator object that describes underlying sorting technique . If we are using default natural sorting order then we will ge null.

Note:-Default natural sorting order.
For Number:-Ascending order.
For String:- Alphabetical order.

NavigableSet Interface:-

A) It is introduce 1.6 version of java.
B) It is child interface of SortedSet.
C) It contain several method which can be used for navigable pupose.

Method of NavigableSet:-

1) floor(e) Return greatest element which is < =e or return null if there is no such element present.
2) ceiling(e) Return lowest element which is >= e or return null if there is no such element.
3) lower(e) Return highest element which is <e or return null if there is no such element.
4) higher(e) Return lowest element which is >e or return null if there is no such element.
5) pollFirst() Retrieves and removes the first(lowest) element, or return null if this set is empty.

6) pollLast() Retrieves and removes the last(highest) element, or return null if this set is empty.
7) descendingSet() Returns a reverse order view of the elements contain in this set.

TreeSet:-

A) Introduce in 1.2 version of java.
B) The underlying data structure is all object inserted based on some sorting order it may be default natural sorting or customized sorting order.
C) TreeSet can store only homogeneous object if we try to insert heterogeneous object then we will get runtime exception saying classCasteException.
D) Insertion order is not preserved.
E) Duplicate object insertion is not allowed.
F) Null insertion is possible but only once but this rule is applicable until 1.6version but java from 1.7 version onward null insertion is not allowed .
G) TreeSet methods are non-synchronized.
H) TreeSet class implements Serializable Cloneable but not RandomAccess.
I) We can use only Iterator Cursor to retrieve the object from TreeSet.

Example:-

```
import java.util.*;
class TreeSetDemo
{
public static void main(String[]args)
{
TreeSet ts=new TreeSet();
ts.add(12);
ts.add(4);
ts.add(9);
ts.add(21);
ts.add(17);
ts.add(88);
System.out.println(ts);
System.out.println(ts.ceiling(9));
System.out.println(ts.higher(9));
System.out.println(ts.lower(17));
System.out.println(ts.floor(9));
System.out.println(ts.pollFirst());
System.out.println(ts.pollLast());
```

```
System.out.println(ts.descendingSet());
}
}
```

Result:-

```
[4, 9, 12, 17, 21, 88]
9
12
12
9
4
88
[21, 17, 12, 9]
```

Comparable interface:-

- If we depends on default natural sorting our object must be comparable and homogeneous.
  An object said to comparable when corresponding class must be implements Comparable interface. All wrapper classes and String objects are implementing Comparable interface hence it is possible to performed sorting.
- Comparable interface present in java.lang package and it contain only one method compareTo().
- public abstract int compareTo(Object obj)
  obj1compareTo(obj2):-Return –ve if obj1 has come before obj2.
  obj1compareTo(obj2):-Return +ve if obj1 has to come after obj2.
  obj1compareTo(obj2):-Return 0(zero) if obj1 and obj2 are equal.

Example:-

```
class TestDemo
{
public static void main(String[]args)
{
System.out.println("1".compareTo("2"));
System.out.println("3".compareTo("2"));
System.out.println("3".compareTo("3"));
}
}
```

Result:-

```
-1
1
0
```

Note:-Comparable means default natural sorting order.

Comparator interface:-

- For customized sorting order we use Comparator interface.
- When we performed customized sorting order then corresponding class not need implements Comparator interface some third party class can also implements Comparator interface.
- Comparator interface present java.util package.
- It contains two method
  A) public void int compare(Object obj1,Object obj2)
     Return –ve if obj1 has come before obj2.
     Return +ve if obj1 has to come after obj2.
     Return 0(zero) if obj1 and obj2 are equal.
  B) public boolean equals(Object obj)

Example:-In this example we insert String object into the TreeSet object and performed reverse sorting order.

```
import java.util.*;
class ComparatorDemo
{
public static void main(String[]args)
{
TreeSet ts=new TreeSet(new MyComparator());
ts.add("vikas");
ts.add("ravi");
ts.add("yogesh");
ts.add("vivek");
ts.add("bishal");
System.out.println(ts);
}
}
class MyComparator implements Comparator
{
public int compare(Object obj1,Object obj2)
{
String s1=(String)obj1;
String s2=(String)obj2;
return -s1.compareTo(s2);
}
}
```

Result:- [yogesh, vivek, vikas, ravi, bishal]

Note:-In the above example we performed different sorting operation as follows:-

1) return s1.compareTo(s2);
   Result:- [bishal, ravi, vikas, vivek, yogesh](ascending order)
2) return -s1.compareTo(s2);
   Result:- [yogesh, vivek, vikas, ravi, bishal](descending order)
3) return s2.compareTo(s1);
   Result:- [yogesh, vivek, vikas, ravi, bishal](descending order)
4) return s2.compareTo(s1);
   Result:- [bishal, ravi, vikas, vivek, yogesh](ascending order)
5) return +1;
   Result:- [vikas, ravi, yogesh, vivek, bishal](insertion order)
6) return -1;
   Result:- [bishal, vivek, yogesh, ravi, vikas](reverse insertion order)
7) return 0;
   Result:-[vikas] (In this case only object will be printed and all remaining are duplicate.)

Example:-In this program to insert String and StringBuffer object into the TreeSet where sorting order is increasing length order. If two object having same length then considered there alphabetical order.

```
import java.util.*;
class ComparatorDemo
{
public static void main(String[]args)
{
TreeSet ts=new TreeSet(new MyComparator());
ts.add("vikas");
ts.add("narendramodi");
ts.add("bishal");
ts.add(new StringBuffer("yogesh"));
ts.add(new StringBuffer("ravi"));
System.out.println(ts);
}
}
class MyComparator implements Comparator
```

```
{
public int compare(Object obj1,Object obj2)
{
String s1=obj1.toString();
String s2=obj2.toString();
int l1=s1.length();
int l2=s2.length();
if(l1<l2)
{
return -1;
}
else if(l1>l2)
{
return 1;
}
else
{
return s1.compareTo(s2);
}
}
}
```

Result:- [ravi, vikas, bishal, yogesh, narendramodi]

Map interface:-

- **It is not child interface of Collection. If we want to represent a group of object as key-value pair then we should go for Map.**
  Entry=Key+Value

KEY	VALUE
1118	VIKAS
1119	VIVEK
1120	YOGESH

- **Each key-value pair is called Entry and Map is collection of Entry. without exiting of Map object there is no chance of existing Entry object so Entry object defined inside Map object.**
- **Internal implementation of Entry interface.**
  interface Map
  {

```
interface Entry
{
public abstract Object getKey();
public abstract Object getValue();
public abstract Object setValue();
}
}
```

Note:-Entry specific method we can apply only on Entry object.

Method of Map interface:-

1) public abstract Object put(Object key, Object value) To add key-value pair in Map if key already present then old value replace with new value and return old value.
2) public abstract void putAll(Map m)
3) public abstract Object get(Object key) Return the value associate with specify key.
4) public abstract Object remove(Object key) Remove entry associate with specified key.
5) public abstract containsValue(Object value)
6) public abstract boolean isEmpty()
7) public abstract int size()
8) public abstract void clear()

HashMap:-

A) Introduce in 1.2 version of java.
B) Underlying data structure is HashTable.
C) HashMap can store Heterogeneous object both key and value.
D) Insertion order is not preserved it based on hashcode.
E) Duplicate key insertion is not allowed but value can be duplicate.
F) Null insertion is allowed but key level only once but value level we can insert any number of times.
G) HashMap every methods are non-synchronized.
H) HashMap class implements Serializable, Cloneable but not RandomAccess.
I) We use iterator() method to retrieve the elements from HashMap.

Constructor of HashMap:-

1) HashMap hm=new HashMap(); Create an Empty HashSet object with default initial capacity 16 and default fill ratio 0.75.

2) HashMap hm=new HashMap(int initialcapacity); Create an Empty HashMap object with specified initial capacity and default fill ratio.
3) HashMap hm=new HashMap(int initialcapacity, float fillRatio); Create an Empty HashMap object with specified initial capacity and fill ratio.
4) HashMap hm=new HashMap(Map m);

- To get all the keys from Map by using keySet() method.
  public java/util/Set<K>keyset();

- To get all the values from Map by using values() method.
  public java/util/Collection<V>values()

- To get all entries from Map by using entrySet() method.
  public java/util/Set<java/util/Map$Entry<K,V>>entrySet();

Example:-

```
import java.util.*;
class HashMapDemo
{
public static void main(String[]args)
{
HashMap hm=new HashMap();
hm.put(1118,"vikas");
hm.put(1119,"vivek");
hm.put(1120,"yogesh");
hm.put(4093,"ravi");
Set s1=hm.keySet();
System.out.println("All Key="+s1);
Collection c=hm.values();
System.out.println("Total value="+c);
Set s2=hm.entrySet();
System.out.println("All Entry"+s2);
Iterator itr=s2.iterator();
while(itr.hasNext())
{
Map.Entry me=(Map.Entry)itr.next();
System.out.println(me.getKey()+"..."+me.getValue());
}
}
}
Result:- All Key=[1120, 4093, 1118, 1119]
```

```
Total value=[yogesh, ravi, vikas, vivek]
All Entry[1120=yogesh, 4093=ravi, 1118=vikas, 1119=vivek]
1120...yogesh
4093...ravi
1118...vikas
1119...vivek
```

LinkedHashMap:-

- It is child interface of HashMap it is exactly same as HashMap including method and constructor except following.
  1) Introduce 1.4 version of java.
  2) LinkedHashMap underlying data structure is combination Linked and HashTable.
  3) Insertion order is preserved.

Example:-

In above example if we replace HashMap with LinkedHashMap then insertion order is preserved.

```
import java.util.*;
class LinkedHashMapDemo
{
public static void main(String[]args)
{
LinkedHashMap lhm=new LinkedHashMap();
lhm.put(1118,"vikas");
lhm.put(1119,"vivek");
lhm.put(1120,"yogesh");
lhm.put(4093,"ravi");
Set s1=lhm.keySet();
System.out.println("All Key="+s1);
Collection c=lhm.values();
System.out.println("Total value="+c);
Set s2=lhm.entrySet();
System.out.println("All Entry"+s2);
Iterator itr=s2.iterator();
while(itr.hasNext())
{
Map.Entry me=(Map.Entry)itr.next();
System.out.println(me.getKey()+"..."+me.getValue());
}
}
}
```

```
Result:-
All Key=[1118, 1119, 1120, 4093]
Total value=[vikas, vivek, yogesh, ravi]
All Entry[1118=vikas, 1119=vivek, 1120=yogesh, 4093=ravi]
1118...vikas
1119...vivek
1120...yogesh
4093...ravi
```

IdentityHashMap:-

- Introduce in 1.4 version of java.
- It is child class of Map.
- It exactly same as HashMap including method and constructor except one difference, In case of HashMap jvm will used equals() method to identify duplicate key(it performed content comparison).
  But in case of IdentityHashMap jvm will use = = (double equal) operator to identify the duplicate key(performed reference comparison).

Example:-

```
import java.util.*;
class IdentityHashMapDemo
{
public static void main(String[]args)
{
HashMap hm=new HashMap();
Integer I1=new Integer(12);
Integer I2=new Integer(12);
hm.put(I1,"vikas");
hm.put(I2,"vivek");
System.out.println(hm);//last entry show.
IdentityHashMap ihm=new IdentityHashMap();
ihm.put(I1,"vikas");
ihm.put(I2,"vivek");
System.out.println(ihm);
}
}
```

Result:-

```
{12=vivek}
{12=vikas, 12=vivek}
```

WeakHashMap:-

- Introduce in 1.2 version of java.
- It is child class of Map.
- It exactly same as HashMap including method and constructor except following difference.
  If an object is associate with HashMap that object is not destroyed even though it does not contain any reference type.
  But in case of WeakHashMap if the object does not contains reference type that object is eligible for garbage collector even though it associated with WeakHashMap.

Example:-

In this example we used HashMap.

```
import java.util.*;
class Demo
{
public String toString()
{
return "demo";
}
public void finalize()
{
System.out.println("Object destoyed");
}
}
class HashMapDemo
{
public static void main(String[]args)
{
HashMap hm=new HashMap();
Demo d=new Demo();
hm.put(d,"vikas");
System.out.println(hm);
d=null;
System.gc();
System.out.println(hm);
}
}
```

Result:-

```
{demo=vikas}
{demo=vikas}
```

Example:-In this example we used WeakHashMap in place of HashMap.

```
import java.util.*;
class Demo
{
public String toString()
{
return "demo";
}
public void finalize()
{
System.out.println("Object destroyed");
}
}
class WeakHashMapDemo
{
public static void main(String[]args)
{
WeakHashMap whm=new WeakHashMap();
Demo d=new Demo();
whm.put(d,"vikas");
System.out.println(whm);
d=null;
System.gc();
System.out.println(whm);
}
}
```

Result:-

```
{demo=vikas}
{}
Object destroyed
```

SortedMap:-

- It child interface of Map.
- Introduce in 1.2 version of java.
- If we want to represent a group of object as a group of key-value pair according to some sorting order of keys then we used SortedMap.
- Sorting always based on key not value.

SortedMap specific Method:-

1) public abstract Object firstKey() Return the first key in this Map.

2) public abstract Object lastKey() Return the last key in this Map.
3) public abstract SortedMap headMap(Object KeyE) Return a view of the portion of this map whose keys are strictly less than keyE.
4) public abstract SoretedMap tailMap(Object keyE) Return of the portion of this map whose key are greater than or equal to key.
5) public abstract Comparator comparator()

NavigableMap:-

- Introduce in 1.6 version of java.
- It is child interface of SortedMap(I).
- It contain several method for navigable purpose.

NavigableMap specific Method:-

1 floorKey()
2 lowerKey()
3 ceilingKey()
4 higherKey()
5 pollFirstEntry()
6 pollLastEntry()
7 descendingMap()

TreeMap:-

- Introduce in 1.2 version of java.
- The underlying data structure is RED-BLACK tree.
- Insertion order is not preserved it is based on some sorting order of key.

  If our sorting order is default natural sorting order then key should be homogeneous and comparable otherwise we will get runtime exception saying classCasteException.

  If we defining our own sorting by comparator then keys need not be homogeneous and comparable object.

  But the above rule is applicable only on key not on values, value can be homogeneous and non comparable.
- Duplicate are key not allowed but value can be duplicate.
- For Empty TreeMap it is possible to insert null key only once, If we are trying to insert null key in non empty TreeSet then we will runtime exception saying NullPointerException but for the value any number of null values can be inserted.

But this rule is applicable only until 1.6 version of java from 1.7 version of java null key insertion is not possible.

Constructor of TreeMap:-

1) TreeMap tm=new TreeMap(); Create empty TreeMap object that will be stored by using natural sorting order.
2) TreeMap tm=new TreeMap(Comparator c); Create TreeMap object that will be stored by using customized sorting order.
3) TreeMap tm=new TreeMap(SortedMap m); It creates the TreeMap object by initializing SortedMap.
4) TreeMap tm=new TreeMap(Map m); It creates TreeMap object with specified Map.

Example:-

In this program we performed default natural sorting (ascending).

```
import java.util.*;
class TreeMapDemo
{
public static void main(String[]args)
{
TreeMap tm=new TreeMap();
tm.put(1120,"yogesh");
tm.put(1118,"vikas");
tm.put(1119,"vivek");
tm.put(4021,"bishal");
tm.put(1233,null);
tm.put(1254,1456);
System.out.println(tm);
}
}
```

Result:-

{1118=vikas, 1119=vivek, 1120=yogesh, 1233=null, 1254=1456, 4021=bishal}

Example:-In this program we performed customized sorting (descending order).

```
import java.util.*;
class TreeMapDemo
{
public static void main(String[]args)
{
```

```
TreeMap tm=new TreeMap(new MyComparator());
tm.put(1120,"yogesh");
tm.put(1118,"vikas");
tm.put(1119,"vivek");
tm.put(4021,"bishal");
tm.put(1233,null);
tm.put(1254,1456);
System.out.println(tm);
}
}
class MyComparator implements Comparator
{
public int compare(Object obj1,Object obj2)
{
String s1=obj1.toString();
String s2=obj2.toString();
return s2.compareTo(s1);
}
}
```

Result:-

{4021=bishal, 1254=1456, 1233=null, 1120=yogesh, 1119=vivek, 1118=vikas}

Hashtable:-

A) Introduction in 1.0(legacy) version of java.
B) Underlying data structure is Hashtable.
C) We can store heterogeneous data both key and value.
D) Insertion order is not preserved it based on hash code of key.
E) Duplicate key are not allowed but value can be duplicate.
F) Null insertion is not allowed but key and value if we are insert then we will get Runtime Exception nullPointerException.
G) Hashtable all method is synchronized.
H) Hashtable class implements Serializable, Cloneable but RandomAccess.
I) It is best choice if our frequent operation is searching.

Constructor of Hashtable:-

1) Hashtable ht=new Hashtable(); Create Empty HashSet object with default capacity 11 and fill ratio 0.75.
2) Hashtable ht=new Hashtable(int initialcapacity);
3) Hashtable ht=new Hashtable(int initialCapacity,float fillRatio);

4) Hashtable ht=new Hashtable(Map m);

Example:-

```
import java.util.*;
class HashTableDemo
{
public static void main(String[]args)
{
Hashtable ht=new Hashtable();
ht.put(new Emp(1118),"vikas");
ht.put(new Emp(4021),"bishal");
ht.put(new Emp(1120),"yogesh");
ht.put(new Emp(1119),"vivek");
System.out.println(ht);
}
}
class Emp
{
int eid;
Emp(int eid)
{
this.eid=eid;
}
public int hashCode()
{
return eid;
}
public String toString()
{
return eid+"";
}
}
```

Result:-

```
{1120=yogesh, 1119=vivek, 1118=vikas, 4021=bishal}
```

Properties class:-

- It is child class of Map interface.
- In our program anything repeatedly changes like username, password, contact number, address etc it is not recommended to write hard code because for every change we must be recompile redeploy, and some time there may be chance of restart the server it create performance problem and big business impact.

Overcome these problem we use property file, inside property file we configure such types of variable things. and in our java program we read that variable from property file and use it.

The main advantage of this approach is we have not need re-compile our java program for every changes, we just performed redeploy.

- We can use Properties object to hold properties which coming from properties file.
- In Properties class key and value must be String type.

Properties class constructor:-

A) Properties p=new Properties();

Important method of Properties class:-

1) public void load(InputStream is) load the data from InputStream Object.
2) public void store(OutputStream os, String comment) To store properties from java properties object into properties file.
3) public String getProperty(String key) To get value based on key.
4) public void setProperty(String key,String value) To set new property in Properties object.
5) public Enumeration propertyName()

Example:-

```
import java.io.*;
import java.util.*;
class PropertiesDemo
{
public static void main(String[]args)throws Exception
{
FileInputStream fis=new FileInputStream("abc.properties");
Properties p=new Properties();
p.load(fis);
System.out.println(p);
System.out.println("userID="+p.getProperty("userId"));
System.out.println("contact no="+p.getProperty("contact"));
p.setProperty("Secondary contact no","8527033103");
System.out.println(p);
}
```

```
}
```

Result:-

```
{contact=8130535652, userId=vikas}
userID=vikas
contact no=8130535652
{contact=8130535652, userId=vikas, Secondary contact no=8527033103}
```

Dictionary class:-It is an abstract class, similar to the Map representing a key-value relation. by using key we can store value and based on our requirement we can retrieve the value by using key. Thus , it is list of key-value pair.

Note:-Highly recommended to implementing Map interface rather than extending Dictionary class. this is deprecated.

Constructor of Dictionary class:-

A) Dictionary()

Method of Dictionary class:-

1) public abstract V put(Object key, Object value)
2) public abstract Enumeration elements()
3) public abstract V get(Object key)
4) public abstract boolean isEmpty()
5) public abstract Enumeration keys()
6) public abstract V remove(Object key)
7) public abstract int size()

Example:-

```
import java.util.*;
class DictionaryDemo
{
public static void main(String[]args)
{
Dictionary di=new Hashtable();
di.put("1118","vikas");
di.put("1119","vivek");
di.put("1120","yogesh");
di.put("4021","bishal");
for(Enumeration i=di.elements();i.hasMoreElements();)
{
System.out.println("Value in Dictionary="+i.nextElement());
}
System.out.println("Value at key=1118:"+di.get(1118));
```

```
for(Enumeration i=di.keys();i.hasMoreElements();)
{
System.out.println("keys in dictonary="+i.nextElement());
}
System.out.println("Dictionary is Empty="+di.isEmpty());
}
}
```

```
Result:- Value in Dictionary=yogesh
Value in Dictionary=vivek
Value in Dictionary=vikas
Value in Dictionary=bishal
Value at key=1118:null
keys in dictonary=1120
keys in dictonary=1119
keys in dictonary=1118
keys in dictonary=4021
Dictionary is Empty=false
```