UNIT-4
Collections

2/3/20

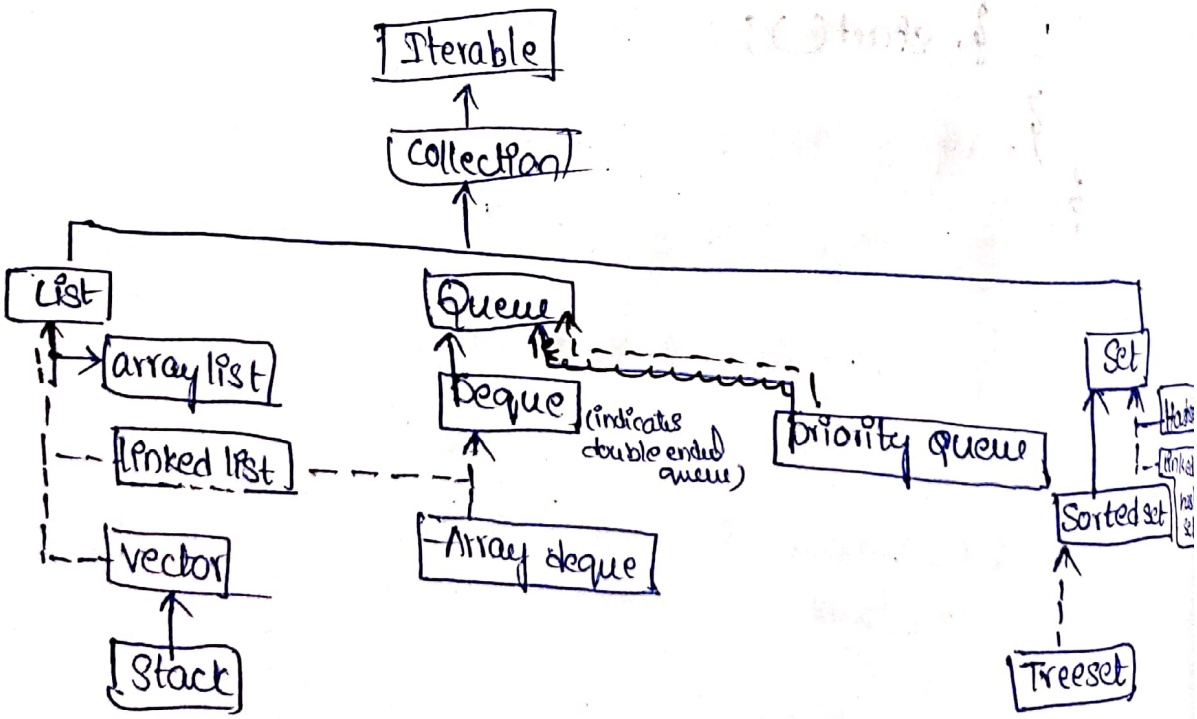
collection:- collection represents a single unit of objects or a group of objects of single unit.

→ collection in java is a frame work that provides an architecture to stored and manipulate a group of objects.

→ java collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation and deletion.

→ the collection frame work is a combination of interfaces and classes available in java.util.*;

Hierarchy of collection frame work:-



→ This Hierarchy represents the classes and Interfaces

→ list, queue, deque, set, sorted set are Interfaces [Bold lines]

→ array list, linked list, vector, stack, array deque, priority queue treeset, hashset & linked hashset are classes [- - -]

→ Here stack is a class which makes the use of the vector.

Collection Interface :-

This interface acts as a super interface to all the interface and it contains some methods to extends and implements to classes and interfaces.

(1) Boolean add (object obj) :- This method acts as the specified object to the collection.

(2) Boolean add(value) :- used to add the given value to the collection.

(3) Boolean addAll (collection c) :- it adds all the elements of c to the invoking collection.

(4) Void clear() :- removes all the elements from the invoking collection.

(5) boolean contains(object obj) :- return true if the invoking collection contains the specified object obj.

6) Boolean containsAll(collection c) :- returns true if the invoking collections contains all the elements of c.

(7) Boolean equals(object obj) : returns true if the invoking collection and specified obj are equal.

(8) Int hashcode() :- return the hashcode for the invoking collection

(9) boolean isEmpty() : returns true if the invoking collection is empty.

(10) Iterator iterator() : returns an iterator for the invoking collections.

11. Boolean remove(object obj) : It removes the specified object from the invoking collection.

(12) Boolean removeAll(collection c) : It removes all the elements from the collection c.

(13) int size() : It returns the no. of elements available in the invoking collection.

(14) Object[] toArray() :- returns an array that contains the elements of invoking collection.

Sub Interfaces of Collection Interface

There are 5 interfaces in list (2) Queue (3) Deque interface (4) set (4) Sorted set interface interface.

Classes of collection Interface. There are 6 classes (1) array list (2). Linked list (3) priority queue (4) array deque (5) hash set (6) Tree set.

Accessing values of collection interface classes :-

→ The values of classes can be accessed in 2 ways
- by using an iterator interface
- using for each loop.

Iterator Interface :-

→ It provides the facility of iterating its elements in a forward direction only.

<Syntax> :- Iterator obj = collectionclassobj.iterator();

→ this Iterator interface contains 3 methods

i) public boolean hasNext() :- It returns true if the Iterator has more element.

ii) public object next() :- It returns the element and moves the cursor pointer to the next element.

iii) public void remove() :- used to remove the last element written by the Iterator.

for each loop :-

~~Syntax~~ : the following syntax is used to traverse through the elements.

Syntax :- for (datatype variable : collection class object)
{
// body of loop
}

Eg :- for (int i : a)
{ s.o.pln (i);
}

O/P 10
20
30
40

Array list :-

→ array list class implements list interface.

→ an array list is same as an array, except which is the size is, size can grows or shrinks.

→ Constructors of the array list class :-

i) ArrayList :- This constructor is used to build an empty array list

ii) ArrayList(Collection c) :- used to build an array list and initialised with the collection c.

iii) ArrayList (int capacity) :- an array list is ~~speci~~ built with a specified capacity of size

methods of array list class :-

i. add() method used to add all the collections existing in the

ii. addAll() :- used to add all the obj of collection c the invoking

iii. get(int index) : return written the element at specified index position

iv. set(int index, E element) : set an element at the specified index position

Eg :-

```
import java.io.*;
import java.util.*;
class Alist
{ p s v m ( )
  {
    ArrayList<String> o = new ArrayList<String>();
    o.add ("sreyas");
    o.add ("csed");
    o.add ("2nd");
    Iterator<String> itr = o.iterator( );
                           ↓
                      arraylist obj name
    while (itr.hasNext( ))
    { s.o.pln (itr.next( ));
    }
  }
}
```

accessing of array list element using for each loop:

```
class Alist
{
  p s v m(String args[])
  {
    ArrayList<String> al = new ArrayList<String>();
    for(String s : al)
    {
      System.out.println(s);
    }
  }
}
```

Linked List : This class implements the list interface and this is used to create a linked list. It contains some of the constructors and methods.

Constructors:

(1) LinkedList() : It is used to create an empty linked list.

(2) LinkedList(Collection c) :- used to create a linked list which contains the elements of collection.

Methods:

(1) addFirst(E e) :- this method used to insert a specified element at a first position.

(2) addLast(E e) : this method used to insert a specified element at the last position (or) at the end of list.

(3) removeFirst() : Removes the element available at the 1st position

(4) removeLast() : " " " " " " last "

(5) removeFirstOccurance(Object O) : It remove the 1st occurance of a specified object.

(6) removeLastOccurance(Object O) : It removes the last occurance of a Specified object

Eg:

```
import java.io.*;
import java.util.*;
class Ulex.
{
  P s v M(String args[])
  {
    LinkedList<Integer> ll = new LinkedList<Integer>();
```

```
ll.add(10);
ll.addFirst(5);
ll.addLast(20);
for(Integer i:ll)
{
  e.o.println(i);
}
}
```

Priority Queue:

→ It implements the queue interface, the priority queue is the elements which are inserted can be stored into queue are arranged either in ascending order or decending order. By default it follows ascending order.

→ Constructors of the priority queue:-

(1) PriorityQueue(): It creates a priority queue with the default initial capacity of 11

(2) PriorityQueue(int capacity)- It creates a priority queue with specified capacity.

(3) PriorityQueue(SortedSet<E> c): It creates a priority queue by storing the elements of sorted set in it.

methods:

(1) boolean add(E element): It add the element e into the priority queue

(2) Public peek(): This method returns the head of the queue. But it not removes it.

(3) public poll(): It returns the head of the queue and removes it.

Eg:-

```
import java.io.*;
import java.util.*;
class PriorityQ
{
  p s v m(String args[])
  {
PriorityQueue<Integer> pq = new PriorityQueue<>();
pq.add(10);
pq.add(20);
pq.add(30);
for(Integer i : pq)
{ s.o.pln(i);
}
s.o.pln("Head is"+pq.peek());
s.o.pln("Removed head is"+pq.poll());
} }
```

Array Deque: This class implements the deque interface

→ array deque means insertion and deletion of element into the queue by from both ends (front and rare end)

Constructors :-

~~(1) Array deque~~

(1) ArrayDeque: used to create an empty array deque.

(2) ArrayDeque(int capacity): it creates an array deque with a specified capacity.

(3) ArrayDeque(Collection c): it creates a arraydeque and stores the elements of collection c.

Methods :-

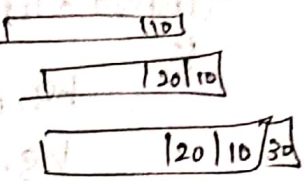
(1) addFirst(Element e): it add the specified element at the first.

(2) addLast(Element e): it add the specified element at the last.

3. getFirst():- Writtens the value available at the 1st position
4. getLast():- " " " " " " " " " last "

Eg:-

```
import java.io.*;
import java.util.*;
class Arrayfq
{
    p s v m(String args[])
    {
ArrayDeque<Integer> adq = new ArrayDeque<Integer>();
adq.add(10);
adq.add(20);
adq.addFirst(30);
for(Integer i : adq)
{ S.O.println(i);
}
}
}
```



Hash set:-

→ This class implements set interface in the hashset. In the hash set the duplicate values will not be allowed and it uses the hash table structure to store the elements.

Constructors:-

(1) HashSet():- It creates an empty hash set.

(2) HashSet(int capacity):- it creates an hash set with a specified capacity.

(3) HashSet(Collection c): It creates an hash set and storing the elements of collection c into it.

Methods():-

1) add(E e): It adds the element e into the hash set

2) Boolean contains(Object o): It writtens true if the element o is present in the set.

3) Boolean remove(Object o): It removes the object o from the hash set.

```
import Test
{
  psvm( )
  {
    TreeSet<Integer> ts = new TreeSet<Integer>( );
    ts.add(10);
    ts.add(20);
    ts.add(30);
    for(Integer i : ts)
    {
      s.o.println(i);
    }
  }
}
```

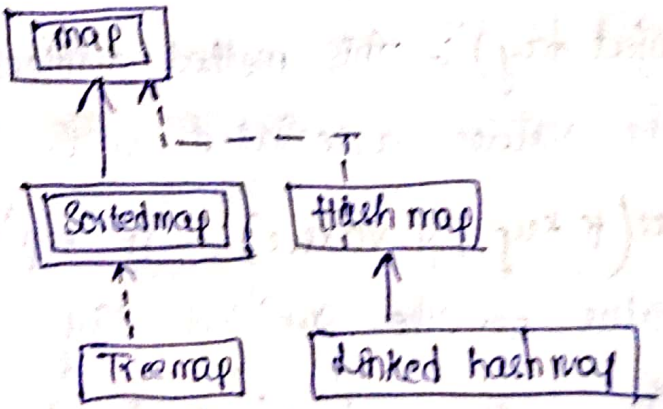
Map Interface:

→ map interface is used to store the data values as "key-value pair".

→ each "key-value pair" is called as an "entry"

→ the keys are unique. A map contains unique keys

→ Hierarchy of a Java map:-



here ▭ → interface

▭ → class.

↑ → extends

⇡ → implements.

methods of map interface:

(1) V put(object key, object value) :- (key-value)

this method inserts an entry into the map.

(2) void putAll(Map map) :-

this is used to insert the specified map into the new map.

(3) V remove(object key) : used to delete the entry of specified key.

(4) boolean remove(object key, object value) : used to delete the entire entry contains key and value

(5) set keySet() :- it writtens the set view containing all the key

(6) Set<Map.Entry<K,V>> entrySet() :-

this is used to access the entries available on a map interface.

(7) V get(object key) :- this method writtens the object contains the value associated with the key.

(8) V replace(K key, V value) :- it replaces the specified value for the specified key.

Eg V replace(101, "33")

Map.Entry Interface:

→ entry is a subinterface of map it will be accessed as "map.Entry".

→ It writtens the collection view of the map elements

Syntax:-

→ This contains a method called entryset used to access entries of map interface

Sorted Map:

→ This interface provides a mechanism to its subclasses as the entry should be stored in a sorted order

Tree map:-

→ This class implements sorted map interface and the elements are stored in a tree structure, contained in a sorted order

Constructors of the tree map:-

(1) TreeMap(): construct an empty tree map that will be sorted by using the natural order of its keys.

→ By default natural order is in ascending order

(2) Treemap(map m): This constructor constructs a tree map and stored with the elements of map m into it.

(3) Treemap(sortedmap sm):- this constructs the treemap and initialized with the entries of sortedmap sm.

(4) Treemap(comparator cm): It constructs an empty tree map that will be sorted using comparator cm.

↓ used to sort the elements

methods

(1) boolean containsKey(object key) :- returns true if specified key is available in map

(2) boolean containsValue(object value) : returns true if specified value is available in map.

(3) ~~boolean~~ object firstKey() : it written the key of the ~~first~~ entry available on sorted map

(4) object lastKey() : returns key of the last entry made on the sorted map.

(5) SortedMap subMap(k startKey, ~~k~~ endKey) :- it returns the portion of map containing map from start key to end key.

Eg :-
```
class Tmap
{
  psvm()
  {
    TreeMap<Integer, String> tm = new TreeMap<Integer, String>();

    tm.put(501, "Abhi");
    tm.put(502, "Ankita");
    tm.put(503, "akshu");
    for(Map.Entry m : tm.entrySet())
    {
      S.O.pln(m);
    }
  }
}
```

Inter communication Threads

```
import java.io.*;
import java.util.*;
class first extends Thread
{
    public void run()
    {
      for(int i=1 ; i<=5;i++)
      { int roll;
        Random d = new Random();
        roll = d.nextInt(10);
        S.o.pln("random number is"+roll);
        Thread t2 = new Second(roll);
        Thread t3 = new Third(roll);
        if(roll%2==0)
        { t2.start();
        }
        else
        t3.start();
      }
    }
}
class Second extends Thread
{ int r1;
  Second(int r)
  { r1=r;
  }
  public void run()
  { s.o.pln("Square of number "+r1+" is:"
                +r1*r1);
```

```
}
}
class Third extends Thread
{
  int r1
  Third(int r)
  {
    r1 = r;
  }
  public void run()
  {
    S.o.pln("the cube of number" + r1 + "is:" +
                                    r1*r1*r1);
  }
}
class Eqmain
{
  p s v m(string args[])
  {
    Thread t1 = new First();
    t1.start();
  }
}
```

Hash map:

- It is a class which implements map interface and it allows one null key,value pair

constructors of the hash map

1. HashMap(): It constructs an empty hashmap.
2. HashMap(int capacity): it creates an empty hashmap of a specified capacity.
3. HashMap(Map m): it creates hash map and copies the values of hash map m into it.

methods:

1) void clear(): used to remove all mappings from a map
2) boolean containskey(object key): written true if specified key is available in hash map.
3) boolean containsvalue(object value): writtens true if specified value is available in hash map
4) object get(object key): it retrives the key value pair for the specified key.

prgm:

```
class Hmap
{
  ps vm ( )
  {
    Hashmap<Integer,String> hm = new Hashmap<
                                 Integer,string>();
     hm.put (501,"mani");
     hm.put (502, "mahi");
     hm.put (Null, "Null");
     for( S.o.pln("Hash map Ps" +hm);
  }
}
```

Linked hash map :-

⇒ This class inherits hashmap class and it is used to stored the data as a key value pair and it allows a single null key and multiple null values

Constructors :-

(1) LinkedHashMap() :
Creates an empty linked hash map

(2) LinkedHashMap(int capacity) : creates an empty linked hash map with specified capacity

(3) LinkedHashMap(Map m) : It creates a linked hash map and copies the values of map m into it.

Methods :-

(1) Object get(Object key) :- It writtens the value associated with specified key

(2) Object get(Object key, Object value) :- it writtens the key value pair if the specified pair is available otherwise it writtens null.

(3) Object removeEldestEntry(Map.Entry Eldest) : It removes the eldest entry of a map and writtens true.

prgm:

```
class Lmap
{
  p s v m ( )
  {
LinkedHashMap<Integer, String> Lhm = new LinkedHashMap
                                    <Integer, String>( );
Lhm.put(501, "manu");
Lhm.put(502, "mahi");
Lhm.put(null, "nlull");
Lhm.put(504, "Null");
S.O.pln("Linked hash map is" + Lhm);
  }
}
```

Legacy classes :-

→ Legacy classes are nthng but the classes available in the older versions of java.

→ Legacy classes are :- (1) Dictionary - It is abstract class
(2) Hash Table
(3) properties
(4) vector
(5) Stack
→ these are concreate classes.

1) Dictionary class :-

→ It is a abstract class for which object cannot be created
→ ~~Here which~~ data is stored as like a key-value pair.

2) Hash Table :-

→ It is one of the legacy class and inherits from Dictionary class in which data is stored as key-value pair and it doesnot allow any null keys or null values.

constructors :-

(1) HashTable( ) : creates an empty hash table
(2) Hash Table(int capacity) : creates an empty hash table with specified capacity

methods :-

1) put(key K, value V) :- It inserts the key-value pair into hash table
2) remove(object key) : It removes a key value pair for specified key.
3) IsEmpty( ) :- It checks whether hash table is empty or not
4) size( ) :- writtens the size of the hash table
5) capacity( ) :- written the capacity of the hash table

Eg: class ht
{
psvm( )
{

Hashtable<Integer, String> ht = new Hashtable<Integer, String>();

ht.put(501, "manu");
ht.put(502, "mahi");
S.o.pln("hash table is" + ht);
}
}

(3) Properties class :-

→ In which data is stored as a key value pair and both key and values are of type string

Constructors :-

(1) Properties() :- used to create a property to store a key value pair.

(2) properties(int capacity) :- used to create a property of specified capacity

(3) properties(properties p) :- creates a property and copies the property p into it

methods() :-

(1) String getProperty(String key) :- it writtens the value for a specified key

(2) String setproperty(String key, String value) :- it inserts the specified value for the specified key.

class Pclass
{
psvm( )
{

Properties p = new Properties();
p.put(501, "manu");
p.put(502, "mahi");

S.O.P("properties values are" + p);
}
}

(2) vector :-

→ A vector is a dynamic array and it is synchronized
→ It is similar to the array list except dynamic in nature.

constructors :-

(1) vector() :- used create an empty vector to store the elements
(2) vector(int capacity) : creates a vector of a specified capacity.
(3) vector(vector v) : creates a vector and copies the elements of vector v into it.

Methods() :-

(1) object addElement(Element e) :- used to add a element to the vector.
(2) object firstElement() :- writtens 1st element of the vecto
(3) Boolean remove(object o) :- removes the specified object
(4) object get(index) :- written the element available in the specified position.

Eg :-

```
class vclass
{
  P S V M (String args[])
  {
  vector<Integer> v = new vector<Integer>();
   v.addElement(10);
   v.addElement(20);
   S.O.P("vector values are" + v);
  }
}
```

(5) Stack :-

→ It is one of the legacy class in java used to store & retrive the data in LIFO-Fashion and push() & pop() are used to insert & delete an operation resp.

Constructors :-

(1) Stack() :- creates an empty stack

(2) Stack(int capacity) :- creates an empty stack with specified capacity

(3) Stack (Stack s) :- creates a stack and copies stack s elements into it

Methods :-

(1) void push(element e) :- used to insert an element into the Stack

(2) object pop() :- used to delete an element available at the top of the stack

(3) Boolean isEmpty() :- used to check whether the stack is empty or not

(4) object peek() :- it returns the head of the stack also as stack top.

prgm :-

```
class Sclassex
{
  P S V M( )
  {
  Stack<Integer>  s = new Stack<Integer>();
  s.push (10);
  s.push (20);
  s.push (30);
// s.pop();
  s.o.pln("Stact elements are " +s);
  }
}
```

Utility classes:-

→ The utility classes are (1) String Tokenizer
(2) Bitset
(3) Date
(4) Calender
(5) Random
(6) Scanner
(7) formatter

} These all are concreate classes used to perform some specific operations.

String Tokenizer:

→ It breaks the given string into Tokens.

→ these are available in java.util.*; package.

Constructors:

(1) String Tokenizer(String str): creates a stringtokenizer for a specified string.

(2) String Tokenizer (String Str, String delimiter) creates a stringtokenizer for a specified string with a specified delimiter.

methods():

(1) string nextToken(): writtens the next Token from a string Tokenizer by taking "space" as a default delimiter.

(2) String nextToken(String Delimiter):- this method writtens the next ~~method~~ Token from a string Tokenizer based on a specific delimiter.

(3) object nextElement(): it is similar to next token. its written type is object.

(4) Boolean hasMoreTokens():- writtens true if more tokens are available.

(5) Boolean hasMoreElements():- same as the hasmoretokens.

(6) int countTokens():- It counts the no. of times the next token method is called.

Prgm: class stoken
{
p s v m( )
{
StringTokenizer st = new StringTokenizer("Sca
Year cse d", " ");
while(st.hasMoreTokens())
{ s.o.pln(st.nextToken());
}
}
}

2) BitSet :-
→ It creates an array of bits represented by boolean values.

Constructors :-
1) BitSet() :- creates an empty BitSet.
2) BitSet(int capacity) :- " " " " specified capacity

Methods :-
1) Set(int value) :- used to set the index position to the bit

Prgm :- class Bclass Beg
{
p s v m( )
{
BitSet bs1 = new BitSet();
BitSet bs2 = new BitSet(4);
bs1.set(0);
bs1.set(1);
bs1.set(2);
bs2.set(10);
bs2.set(5);
bs2.set(3);
bs2.set(6);
s.o.pln("bitset values are of bs1 is" + bs1);
s.o.pln(" " " " " bs2 is" + bs2);
}
}