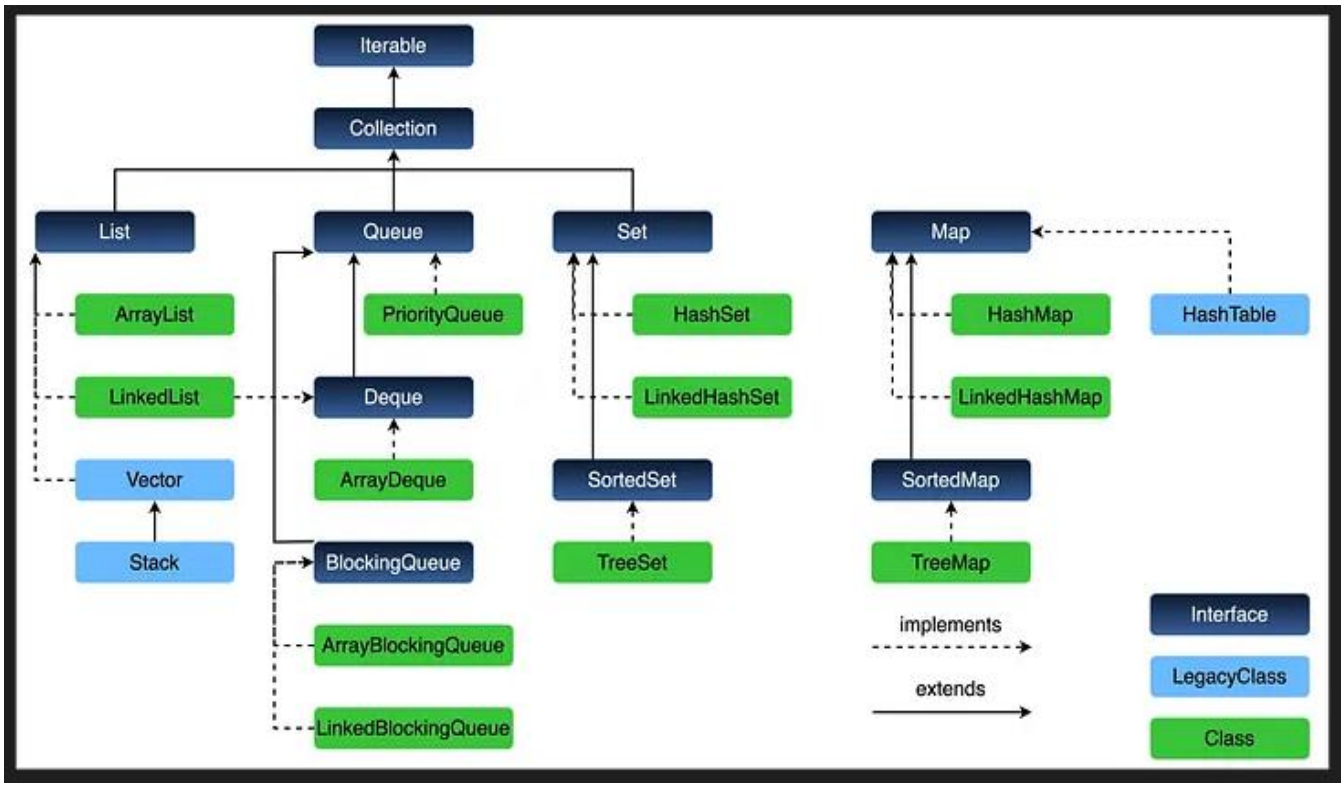
COLLECTION FRAMKEWORK

M.MUKESH



Public interface Iterable<T> {
Iterable<T> iterator();
}

Public interface Collection<E> {}

COLLECTION CONTAINS SOME METHODS :

int size();
boolean isEmpty();
boolean add(Type o);
boolean addAll(Collection c);
boolean contains(Object o);
boolean containsAll(Collection c);
boolean remove(Object o);
boolean removeAll(Collection c);//remove all the element in passed list
boolean retainAll(Collection c);//replace all the element in the list
void clear();//it will make list as empty

Public interface sequencedCollection<E> extends Collection<E> {}

SEQUENCEDCOLLECTION CONTAINS SOME METHODS :

default void addFirst(type o);
default void addLast(type o);
default E getFirst();
default E getLast();
default E removeFirst();
default E removeLast();
SequencedCollection<E> reversed();

Public abstract Class AbstractCollection<E> implements Collection<E> {}

ABSTRACTCOLLECTION CONTAINS SOME METHODS :

Public abstract int size();
Public abstract Iterator<E> iterator();

public interface List<E> extends SequencedCollection<E>{}

List CONTAINS SOME METHODS :

public E set(int index, E element){}//add a new element
public void add(int index, E element){}//replace the old to new element
public int indexOf(Object o){}
public int lastIndexOf(Object o){}
public boolean addAll(int index, Collection<? extends E> c){}
public E next(){}
public boolean hasNext(){}
public E previous(){}
public boolean hasPrevious(){}
public int nextIndex(){}
public int previousIndex(){}
public List<E> subList(int fromIndex, int toIndex){}

public class ArrayList<E> implements List<E>{}

ARRAYLIST CONTAINS SOME METHODS :
al.ensureCapacity(int);

public class LinkedList<E> implements List<E>{}
LINKEDLIST CONTAINS SOME METHODS :
ll.peek();// first value
ll.peekFirst();//take 0 ind value
ll.peekLast();
ll.poll();// remove the first element
ll.pollFirst();//take 0 ind value
ll.pollLast();
ll.pop();//remove the first element
ll.push(Object o);//add the element in front side
ll.add(Object o);//add the element in end side
ll.offer(Object o);
ll.offerFirst(Object o);
ll.offerLast(Object o);
ll.descendingIterator();
ll.removeFirstOccurrence(Object o);
ll.removeLastOccurrence(Object o);
ll.remove();

public class Vector<E> implements List<E>{}
VECTOR CONTAINS SOME METHODS :
v.setSize(int);
V.elements();
v.setElementAt(Oject, int);
v.insertElementAt(Object, int);
v.removeElement(Oject)
v.removeAll(Collection);
v.removeElementAt(int);
v.indexOf(Object, int);
v.copyInto(Object[]);

public class Stack<E> extends Vector<E> {}

STACK CONTAINS SOME METHODS :

s.push(s);
s.pop();//remove last object
s.peek();//retrieve the top of the element
s.search(s);//it will give reverse index value
s.empty();

public interface Set<E> extends Collection<E> {}

public abstract class AbstractSet<E> extends AbstractCollection<E> implements Set<E> {}

public class HashSet<E>
extends AbstractSet<E>
implements Set<E>{}

public class LinkedHashSet<E>
extends HashSet<E>
implements SequencedSet<E>, Cloneable, java.io.Serializable {}

WHAT IS COLLECTION ?

- COLLECTION IS A INTERFACE.
- COLLECTION MEANS AN OBJECT THAT GROUPS MULTIPLE ELEMENTS INTO A SINGLE UNIT.

WHAT IS FRAMEWORK ?

- FRAMEWORK IS A READYMADE ARCHITECTURE.
- FRAMEWORK IS A SET OF CLASSES AND INTERFACES.

WHAT IS COLLECTION FRAMEWORK ?

- IT REDUCES PROGRAMMING EFFORT(BY PROVIDING DATA STRUCTURE AND ALGORITHM) WHILE INCREASING PERFORMANCE.

• THE COLLECTION FRAMEWORK WAS RELEASED IN **1998 WITH JDK 1.2**.

• COLLECTION FRAMEWORK IS USED TO PERFORM **CRUD** OPERATIONS SUCH AS SEARCHING, SORTING, INSERTION, UPDATION AND DELETION.

• IN JAVA, THE COLLECTION INTERFACE (JAVA.UTIL.COLLECTION) AND MAP INTERFACE (JAVA.UTIL.MAP) ARE THE TWO MAIN **"ROOT"** INTERFACES OF JAVA COLLECTION CLASSES.

NOTE : IN COLLECTION FRAMEWORK ALL THE ELEMENTS SHOULD BE OBJECT TYPE, SO WE CANNOT STORE THE PRMITIVE DATATPES.

WHAT IS LIST ?

• LIST IS A SUB-INTERFACE OF COLLECTION INTERFACE.

• LIST INTERFACE PROVIDES A SPECIAL ITERATOR, CALLED A LISTITERATOR, THAT ALLOWS ELEMENT INSERTION AND REPLACEMENT.

• BY USING LIST WE CAN STORE BOTH **HOMOGENEOUS** AND **HETROGENEOUS** OBJECTS.

CHARACTERISTICS OF LIST :

LIST CAN ALLOW DUPLICATE VALUES.
LIST CAN ALLOW NULL VALUES.
LIST CAN MAINTAIN INSERTION ORDER.
LIST CAN HAVE INDEXING CONCEPT. (BY USING INDEX WE CAN INSERT OR REMOVE AN OBJECT)

WHAT IS ARRAYLIST ?

• ARRAYLIST IS A CONCRETE IMPLEMENT CLASS OF LIST INTERFACE.

• ARRAYLIST IS ALSO KNOWN AS **GROWABLE / RESIZABLE / DYNAMIC** ARRAY.

• IN ARRAYLIST DEFAULT CAPACITY IS 10 AND THE INCREMENTAL CAPACITY IS (CURRENT CAPACITY X 3/2)+1.

EG =(10*3/2)+1.

• AS ELEMENTS ARE ADDED TO AN ARRAYLIST, ITS **CAPACITY GROWS AUTOMATICALLY**.

• ARRAYLIST IS ROUGHLY EQUIVALENT TO VECTOR, EXCEPT THAT IT IS **UNSYNCHRONIZED** AND NOT A THREAD SAFE.

• SO ARRAYLIST IS NON-SYNCHRONIZED, IT IS NOT A THREAD SAFE.

• ARRAYLIST INTERNALLY USE THE **DYNAMIC ARRAY** TO STORE THE OBJECT.

• ARRAYLIST IS BETTER FOR **STORING** AND **ACCESSING RANDOM OBJECT**.

• ARRAYLIST MAY NOT CONTAINS CONTIGUOUS MEMORY LOCATION.(BECAUSE IT STORE IN HEAP AREA)

• ARRAYLIST CONSUME **LESS MEMORY** COMPARE TO LINKEDLIST.

WHAT IS LINKEDLIST ?

• IN LINKEDLIST WE HAVE **SINGLY_LL , DOUBLY_LL , CIRCULAR_LL**.

• LINKEDLIST INTERNALLY USE THE **DOUBLY-LINKEDLIST** TO STORE THE OBJECT.

• LINKEDLIST IS BETTER FOR **INSERTING** AND **DELETING** THE OBJECT.

• LINKEDLIST NOT CONTAINS CONTIGUOUS MEMORY LOCATION.

• LINKEDLIST CONSUME **MORE MEMORY** COMPARE TO ARRAYLIST.

PEEK() & PEEKFIRST() & PEEKLAST() - WHEN THE LIST IS NULL IF WE ACCESS THE FIRST ELEMENT WE GET **NULL**.

POP() & GET() & GETFIRST() & GETLAST() - WHEN THE LIST IS NULL IF WE ACCESS THE FIRST ELEMENT WE GET **NOSUCHELEMENTEXCEPTION**.

POLL() & POLLFIRST() & **POLLLAST()** - WHEN THE LIST IS NULL IF WE ACCESS THE FIRST ELEMENT WE GET **[]**.

WHAT IS VECTOR ?

- **IN JAVA, THE** VECTOR **AND** HASHTABLE **CLASSES ARE SYNCHRONIZED.**
- VECTOR IS SAME AS ARRAYLIST.
- EXCEPT VECTOR IS SYNCHRONIZED, IT IS A THREAD SAFE .
- BUT ARRAYLIST IS NON-SYNCHRONIZED AND THAT IS NOT A THREAD SAFE .

WHAT IS STACK?

- THE STACK CLASS REPRESENTS A **LAST-IN-FIRST-OUT** (**LIFO-FILO**) STACK OF OBJECTS .
- STACK IS A SUB-CLASS OF VECTOR CLASS WITH FIVE OPERATIONS THAT ALLOW A VECTOR TO BE TREATED AS A STACK .
- BY USING POP() LAST ELEMENT WILL BE REMOVED IN STACK DSA BUT IN OTHERS POP() WILL REMOVE FIRST ELEMENT.

METHODS ARE :
s.push(s) ,s.pop() ,s.peek() ,s.search(s) ,s.empty();

DIFFERENCE BETWEEN ARRAY AND ARRAYLIST?
IN ARRAY WE CAN STORE HOMOGENEOUS ELEMENT.
ARRAY IS A FIXED IN SIZE.
ARRAY CANNOT HAVE A INBUILD METHOD (OR)API'S..
ARRAYS ARE MUTABLE IN JAVA.

IN ARRAYLIST WE CAN STORE HETROGENEOUS AND HOMOGENEOUS ELEMENT.
ARRAYLIST IS A RESIZABLE/GROWABLE/DYNAMIC ARRAY.
ARRAYLIST CAN HAVE A INBUILD METHOD (OR)API'S.

WHAT IS SET ?

- SET IS A SUB INTERFACE OF COLLECTION INTERFACE .
- BY USING SET WE CAN STORE BOTH HOMOGENEOUS AND HETROGENEOUS OBJECTS.

CHARACTERISTICS OF SET:

SET CANNOT ALLOW DUPLICATE VALUES.
SET CAN ALLOW ONLY ONE NULL VALUE.
SET CANNOT MAINTAIN INSERTION ORDER.
SET CAN HAVE INDEXING CONCEPTS.

WHAT IS HASHSET ?

- HASHSET DOES NOT MAINTAIN INSERTION ORDER.

WHAT IS LINKEDHASHSET ?

- LINKEDHASHSET MAINTAIN THE INSERTION ORDER.

WHAT IS TREESET ?

- TREESET **CANNOT ALLOW NULL** VALUES. IF NULL VALUE IS PRESENT WE GET **NULLPOINTEREXCEPTION**.
- TREESET **CAN STORE HOMOGENEOUS** ELEMENTS ONLY IF WE STORE HETROGENEOUS ELEMENTS WE GET **CLASSCASTEXCEPTION.**
- TREESET MAINTAIN ASCENDING ORDER.

WHAT IS QUEUE?

- QUEUE PROVIDE ADDITIONAL INSERTION, EXTRACTION, AND INSPECTION OPERATIONS.

• EACH OF THESE METHODS EXISTS IN TWO FORMS: ONE THROWS AN EXCEPTION IF THE OPERATION FAILS, THE OTHER RETURNS A SPECIAL VALUE (EITHER NULL OR FALSE, DEPENDING ON THE OPERATION).

WHEN WE USE STACK AND WHEN WE USE QUEUE?

QUEUE:

• QUEUE IS A SUB-INTERFACE 0F COLLECTION INTERFACE.

• QUEUE IS LINEAR DATA STRUCTURE.

• QUEUE FOLLOWS THE **FIFO** (FIRST IN FIRST OUT).

• IN QUEUE INSERTION AND DELETION ARE DONE FROM TWO END.

• THE ELEMENT FIRST ENTERED IN THE QUEUE IS REMOVED FIRST AS WELL.

• IN QUEUE WE CANNOT ADD OR REMOVE AN OBJECT INBETWEEN THE QUEUE(ARRAYDEQUE).

EG-TASK SCHEDULING IN OPERATING SYSTEM.

CHARACTERISTICS OF QUEUE:

PRIORITYQUEUE:

• QUEUE CAN ALLOW DUPLICATE VALUES.

• QUEUE CANNOT MAINTAIN INSERTION ORDER.

• QUEUE **CANNOT ALLOW NULL** VALUES. IF NULL VALUE IS PRESENT WE GET **NULLPOINTEREXCEPTION**.

• QUEUE **CAN STORE HOMOGENEOUS** ELEMENTS ONLY IF WE STORE HETROGENEOUS ELEMENTS WE GET **CLASSCASTEXCEPTION**.

ARRAYDEQUE :

• QUEUE CAN ALLOW DUPLICATE VALUES.

• QUEUE CAN MAINTAIN INSERTION ORDER.

• QUEUE **CANNOT ALLOW NULL** VALUES. IF NULL VALUE IS PRESENT WE GET **NULLPOINTEREXCEPTION**.

• QUEUE CAN STORE BOTH HOMOGENEOUS AND HETROGENEOUS OBJECTS.

STACK:

• STACK IS A SUB-CLASS OF VECTOR CLASS.
• STACK IS LINEAR DATA STRUCTURE.
• STACK FOLLOWS THE **LIFO/FILO** (LAST IN FIRST OUT).
• STACK CAN STORE BOTH HOMOGENEOUS AND HETEROGENEOUS OBJECTS.
• IN STACK WE CAN DO INSERTION AND DELETION INBETWEEN.
• STACK IS USED WHENEVER WE FACE UNDO AND REDO PROCESS, BECAUSE **PUSH()** WILL **ADD** THE ELEMENT IN LAST AND **POP()** WILL **REMOVE** THE LAST ELEMENT.

EG -TEXT EDITOR, UNDO MECHANISMS, BACK AND FORWARD BUTTONS ON BROWSERS AND FUNCTION CALL STACK.

CHARACTERISTICS OF STACK:

• STACK CAN ALLOW DUPLICATE VALUES.
• STACK CAN ALLOW NULL VALUES.
• STACK CAN MAINTAIN INSERTION ORDER.

• USE A STACK WHEN YOU NEED TO **REVERSE** THE ORDER OF ELEMENTS OR TRACK FUNCTION CALLS**(FILO)**.

I. • USE A QUEUE WHEN YOU NEED TO **MAINTAIN** THE ORDER OF ELEMENTS AND PROCESS THEM IN SAME ORDER**(FIFO)**.

WHAT IS MAP?

- IN MAP OBJECTS CAN BE STORED IN THE FORM OF **KEY** AND **VALUE** PAIR.
- KEY-VALUE PAIR IS ALSO KNOW AS **ENTRY**.
- BY USING MAP WE CAN STORE BOTH **HOMOGENEOUS** AND **HETROGENEOUS** OBJECTS.

MAP METHODS :

map.put(K,V);
map.get(Object key)
map.values()//it return list of values
map.keySet()//it return list of keys
map.entrySet()//return map list
map.containsKey(Object o)//return list of keys
map.containsValue(Object o)//return list of values

CHARACTERISTICS OF MAP :

- IN MAP **VALUE** CAN BE DUPLICATE BUT **KEY** CANNOT BE **DUPLICATE**.
- IN MAP **KEY AND VALUE** CAN BE **NULL**.
- IN MAP INSERTION ORDER IS NOT MAINTAINED.

NOTE : IF THE KEY IS SAME THEN THE LAST UPDATED VALUE WILL BE TAKEN ON THAT KEY.

Set ks = map.keySet();//return.t set type of data
Set es = map.entrySet();//return.t set type of data
Collection ks = map.values();//return.t Collection type of data
ArrayList ks =(ArrayList) map.values();//return.t Collection type of data
System.out.println(Collections.max(a));

WHAT IS HASHMAP?

- **HASHMAP** IS ROUGHLY EQUIVALENT TO **HASHTABLE**, EXCEPT THAT IT IS **UNSYNCHRONIZED** AND ACCEPT **NULL**.

• HASHMAP USES A HASHTABLE TO STORE THE ELEMENTS.

• AN INSTANCE OF HASHTABLE HAS TWO PARAMETERS THAT IS INITIAL CAPACITY AND LOAD FACTOR.

• HASHMAP CANNOT HAVE INDEXING CONCEPTS.

• THE INITIAL DEFAULT CAPACITY OF HASHMAP CLASS IS **16** WITH A LOAD FACTOR OF **0.75**.

• HASHMAP CONTAINS AN ARRAY OF THE NODES,

BUCKETS: ARRAY OF THE NODE IS CALLED BUCKETS. EACH NODE HAS A DATA STRUCTURE LIKE A LINKEDLIST. MORE THAN ONE NODE CAN SHARE THE SAME BUCKET.

WHAT IS LINKEDHASHMAP?

LINKEDHASHMAP CAN MAINTAIN INSERTION ORDER

- **HashMap**: Iteration order is not guaranteed and may change when the map is modified.
- **LinkedHashMap**: Iteration order is predictable and corresponds to the insertion order.

WHAT IS TREEMAP?

• TREEMAP **CANNOT ALLOW NULL** VALUES. IF NULL VALUE IS PRESENT WE GET **NULLPOINTEREXCEPTION**.

• TREEMAP **CAN STORE HOMOGENEOUS** ELEMENTS ONLY IF WE STORE HETROGENEOUS ELEMENTS WE GET **CLASSCASTEXCEPTION**.

• TREEMAP SORT THE LIST IMPLICITLY IN ASCENDING ORDER.

• TREEMAP DOES NOT MAINTAIN INSERTION ORDER.

WHAT IS HASHTABLE?

• **HASHTABLE** IS ROUGHLY EQUIVALENT TO **HASHMAP** , EXCEPT THAT IT IS **SYNCHRONIZED** AND CANNOT ACCEPT **NULL** AND **DUPLICATE VALUE**.

• THE INITIAL DEFAULT CAPACITY OF HASHTABLE CLASS IS **11** WHEREAS LOADFACTOR IS **0.75**.

• THE CAPACITY IS THE NUMBER OF **BUCKETS** IN THE **HASH TABLE**.

• A HASHTABLE IS AN ARRAY OF A LIST. EACH LIST IS KNOWN AS A BUCKET.

COLLECTION FRAMEWORK INTERNAL WORKING

- ARRAYLIST : **Structure**: Internally uses an array to store elements.
- **Growth**: When the array becomes full, it creates a new array with a larger size (usually 1.5 times the old size) and copies the elements to the new array.
- **Access**: Provides fast random access to elements (O(1) time complexity).
- **ArrayList**: Uses a dynamic array for fast random access.

- LINKEDLIST : **Structure**: Uses a doubly linked list to store elements.
- **Node**: Each element is stored in a node, which contains references to the previous and next node.
- **Access**: Provides sequential access (O(n) time complexity for accessing elements by index).
- **LinkedList**: Uses a doubly-linked list for efficient insertion and deletion.

- HASHMAP : **Structure**: Uses an array of buckets, where each bucket is a linked list (or tree in case of high hash collisions).
- **Hashing**: Elements are stored based on the hash code of keys.
- **Resize**: Automatically resizes the array when the load factor threshold is exceeded (default load factor is 0.75).
- **HashMap**: Uses a hash table

- TREEMAP : **Structure**: Uses a Red-Black tree to store key-value pairs.
- **Ordering**: Maintains natural ordering of keys or a specified comparator.
- **Balance**: Red-Black tree properties ensure balanced tree operations, providing O(log n time complexity for get, put, and remove operations.

M.MUKESH

WHAT IS COLLECTION?

WHAT IS FRAMEWORK?

WHAT IS COLLECTION FRAMEWORK?

DIFFERENCE BETWEEN ARRAY AND ARRAYLIST?

DIFFERENCE BETWEEN ARRAYLIST AND LINKEDLIST?

DIFFERENCE BETWEEN ARRAYLIST AND VECTOR?

DIFFERENCE BETWEEN HASHSET AND LINKEDHASHSET?

DIFFERENCE BETWEEN HASHSET AND TREESET ?

DIFFERENCE BETWEEN ARRAY AND COLLECTION?

DIFFERENCE BETWEEN COLLECTION & COLLECTIONS?

DIFFERENCE BETWEEN LIST AND SET?

DIFFERENCE BETWEEN ITERATOR AND LISTITERATOR?

DIFFERENCE BETWEEN SET AND MAP?

DIFFERENCE BETWEEN HASHSET AND HASHMAP?

DIFFERENCE BETWEEN HASHMAP AND HASHTABLE?

DIFFERENCE BETWEEN HASHMAP AND TREEMAP?

DIFFERENCE BETWEEN COMPARABLE AND COMPARATOR?

WRITE A PROGRAM TO ITERATE THE LIST USING THE LAMBDA EXPRESSION?

METHODS

List list = Arrays.*asList*(arr);//it will convert the array into List.

Collections.shuffle(list);//shuffle the list

ArrayList al1 = new ArrayList();
ArrayList al2 = (ArrayList)al1.clone();//arraylist_clone

Collections.unmodifiableList(list);//we make Collection can be readable only in set,map etc..,we cannot add values.

map.putIfAbsent("1","newval");//add if map not cantains this value.

ArrayList res = new ArrayList<>(hash_map.entrySet());//we can add map key values in array list

```
Set l = hm.entrySet();
Iterator itr = l.iterator();
while (itr.hasNext()) {//we use iterator or
        System.out.println(itr.next() + " yes");//we use this or
        Map.Entry entry = (Map.Entry) itr.next();//we use thid
        System.out.println(entry);
}
for (Map.Entry map : hm.entrySet())//map entry
```

```
        System.out.println(map.getKey() + " " + map.getValue());
for (Map.Entry m : hm.entrySet()) {//foreach loop
        System.out.println(m.getKey() + " :" + m.getValue());
}
```