Queue Interface

- The implementation classes for Queue Interface are LinkedList & PriorityQueue.
- In Queue elements are stored in FIFO order.
- If we are creating an object for LinkedList with LinkedList reference variable, then we can access complete functionality of Queue & List.
- From 1.5v onwards LinkedList also implements Queue interface.

Queue Interface Methods

Method	Description
Offer(Object o);	Add an element in to Queue
Object poll() ;	To remove and return first element of the Queue (returns null if the queue is empty)
Object remove();	To remove and return first element of the Queue (NoSuchElementException when the queue is empty)
Object peek();	To return first element of the Queue without removing it

PriorityQueue

- It doesn’t maintains insertion order and returns the elements in ascending order (Smallest Number first).
- In PriorityQueue the top element is always the smallest element.
- It doesn’t accept null.
- PriorityQueue is available since jdk1.5V.
- It allows duplicate values, default capacity is 11.

```
PriorityQueue q=new PriorityQueue();
PriorityQueue q=new PriorityQueue(int initialcapacity);
```

Understanding Map Interface

- In Map elements are stored in the form of Key-Value pairs.
- Keys are unique in Map interface, duplicate keys are not allowed but duplicate values are allowed.
- Each key can map to at most one value.
- Each key-value pair is called "one entry“.
- Duplicates will be replaced but they are not rejected.
- Map interface is not child interface of Collection interface.
- Some map implementations have restrictions on the keys and values they may contain.

Map Interface Methods

Method	Description
int size();	Returns the number of key-value mappings in this map
boolean isEmpty();	Returns true if this map contains no key-value mappings
boolean containsKey (Object key);	Returns true if this map contains a mapping for the specified key.
boolean containsValue (Object value);	Returns true if this map maps one or more keys to the specified value
V get(Object key)	Returns the value to which the specified key is mapped
V put(K key, V value)	Associates the specified value with the specified key in this map
V remove(Object key)	Removes the mapping for a key from this map if it is present
Set<K> keySet()	Returns a Set view of the keys contained in this map

HashMap:

- HashMap is available since jdk1.2V.
- It is available in java.util package.
- It allows duplicate values with unique keys & insertion order is not maintained. (Duplicate keys are replaced)
- Default capacity is 16 & load factor is 0.75.
- HashMap is not synchronized by default.

```
HashMap m=new HashMap();

HashMap m=new HashMap(int initialcapacity);
```


LinkedHashMap:

- The only difference between HashMap & LinkedHashMap is HashMap doesn't maintain the insertion order where as LinkedHashMap maintains it.
- LinkedHashMap is available since jdk 1.4

TreeMap:

- In TreeMap keys are maintained in ascending order.
- TreeMap uses balanced binary trees algorithm internally.
- Insertion order is not maintained.
- TreeMap is available since jdk1.2V.
- Null keys are not allowed where as null values are accepted.
- Duplicates values are accepted, If duplicate key is there the previous key-value will be replaced.

♪ HashMap doesn't maintain insertion order

♪ Linked HashMap maintains Insertion Order

♪ TreeMap Maintains Ascending order of keys

Hashtable

- Hashtable is available since jdk1.0V.
- Insertion order is not preserved.
- Heterogeneous objects are allowed for both keys and values.
- Null key (or) null value is not allowed.
- It allows duplicate values with unique keys.
- Every method present inside Hashtable is synchronized
- Default capacity is 11 & load factor is 0.75.

```
Hashtable h=new Hashtable();

Hashtable h=new Hashtable(int initialcapacity);
```

HashMap Vs LinkedHashMap Vs TreeMap Vs Hashtable

Property	HashMap	LinkedHashMap	TreeMap	Hashtable
Insertion order	Not Maintained	Maintained	Not Maintained (But keys are sorted in ascending order)	Not Maintained
Null Keys/Values	Allowed	Allowed	Not Allowed (Null Values are allowed)	Not Allowed
Synchronization	Not Synchronized	Not Synchronized	Not Synchronized	Synchronized

Important Differences in ‘Collections’

Set (Vs) List

Set	List
A set represents a collection of elements Order of the elements may change in the set.	A List represents ordered collection of elements. List preserves the order of elements in which they are entered.
Set will not allow duplicate values to be stored.	List will allow duplicate values.
Accessing elements by their index (position number) is not possible in case of sets.	Accessing elements by index is possible in lists.

ArrayList (Vs) Vector

ArrayList	Vector
ArrayList object is not synchronized by default.	Vector object is synchronized by default.
Incase of a single thread, using ArrayList is faster than the Vector.	In case of multiple threads, using Vector is advisable. With a single thread, Vector becomes slow.
ArrayList increases its size every time by 50 percent (half).	Vector increases its size every time by doubling it.

HashMap (Vs) HashTable

HashMap	HashTable
HashMap object is not synchronized by default.	Hashtable object is synchronized by default.
In case of a single thread, using HashMap is faster than the Hashtable.	In case of multiple threads, using Hashtable is advisable, with a single thread, Hashtable becomes slow.
HashMap allows null keys and null values to be stored.	Hashtable does not allow null keys or values.