Q. 7

HashMap internal working. How to arrange objects in a order in a HashMap?Difference between Ordering and Sorting

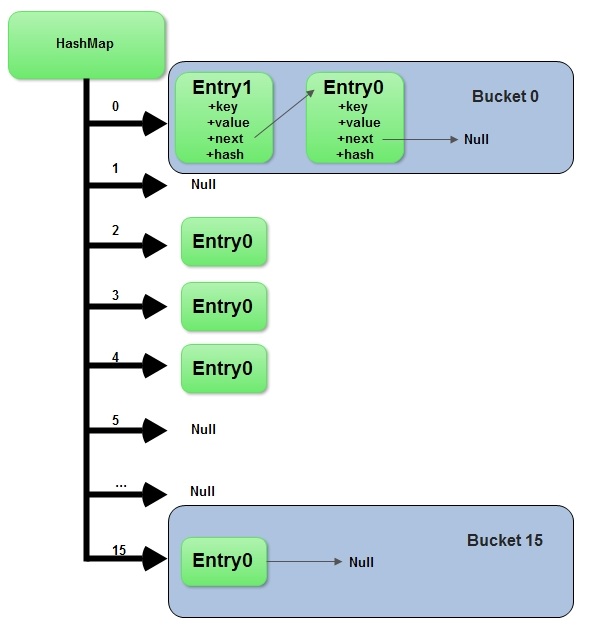
HashMap Internal working:

HashMap has Static class internally, named Entry. It Stores Key (k) as a final generic object and Value (v) as a generic objects.

The array of this Entry object is created,

While putting object:

1. It calculates hashcode of key.
2. Finds out specific bucket for that hashcode puts object in front of that bucket.



Object Ordering in HashMap:

In HashMap it maintains a doubly-linked list running through all of its objects (Elements). This linked list defines ordering of element, which is normally same as the order of insertion into the map.

Difference between Ordering and Sorting:

An ordered collection maintains the order of the elements based on the sequence of insertion into Collection.

A sorted collection keeps the elements sorted based on the basis of sorting criteria

Equals method.

Q. 8

We can sort arraylist with two techniques.

First is with comparable interface and other with comparator interface but there are some situation for when to use these interfaces.

* Class objects which you want to compare is your class means you are the owner of that class then use comparable interface to compare objects of that class
* Class objects which you want to compare is not your class that means it is from third party and you don’t have permission to change implementation of that class the go for comparator interface and compare objects of that class by implementing comparator with your own class.

We also have attached demos regarding this question.

Q 9.

Implementation of hashmap :

Generally initial size of hashmap is 16. When size of hashmap reaches 75% of its current size, i.e. 12 it will double its size by recomputing hashcode of existing data structures.

Since java is multi threaded it is possible that more than one thread might be using same hashmap and then they both realize the need for re-sizing the hashmap at the same time which leads to race condition.

When two or more threads see the need for resizing the same hashmap, they might end up adding the elements of old bucket to the new bucket simultaneously and hence might lead to infinite loops. In case of collision, i.e when there are different keys with same same hashcode, internally we use single linked list to store the elements. And we store every new element at the head of the linked list to avoid tail traversing and hence at the time of resizing the entire sequence of objects in linked list gets reversed, during which there are chances of infinite loops.

Java.util.HashMap.java

public V put(K key, V value) {

    if (key == null)

        return putForNullKey(value);

    int hash = hash(key.hashCode());

    int i = indexFor(hash, table.length);

    for (Entry<K,V> e = table[i]; e != null; e = e.next) {

        Object k;

        if (e.hash == hash && ((k = e.key) == key || key.equals(k))) {

            V oldValue = e.value;

            e.value = value;

            e.recordAccess(this);

            return oldValue;

        }

    }

    modCount++;

    addEntry(hash, key, value, i);

    return null;

}

Here it  
1. Regenerates the hashcode using hash(int h) method by passing user defined hashcode as an argument.  
2. Generates index based on the re-generated hashcode and length of the data structure.  
3. If key exists, it overrides the element else it will create a new entry in the hashmap at the index generated in Step-2

static int hash(int h) {

    // This function ensures that hashCodes that differ only by

    // constant multiples at each bit position have a bounded

    // number of collisions (approximately 8 at default load factor).

    h ^= (h >>> 20) ^ (h >>> 12);

    return h ^ (h >>> 7) ^ (h >>> 4);

}

/\*\*

 \* Returns index for hash code h.

 \*/

static int indexFor(int h, int length) {

    return h & (length-1);

}

here:  
‘h’ is hashcode because of its int data type, it is 32 bit.  
‘length’ is DEFAULT\_INITIAL\_CAPACITY because of its int data type, it is 32 bit.

HashMap usage:

While you could use a map as a list, there are some definite downsides of doing so.

**Maintaining order:** - A list by definition is ordered. You add items and then you are able to iterate back through the list in the order that you inserted the items. When you add items to a HashMap, you are not guaranteed to retrieve the items in the same order you put them in. There are subclasses of HashMap like LinkedHashMap that will maintain the order, but in general order is not guaranteed with a Map.

**Key/Value semantics:** - The purpose of a map is to store items based on a key that can be used to retrieve the item at a later point. Similar functionality can only be achieved with a list in the limited case where the key happens to be the position in the list.