Collections:

* List 🡺 Interface

1. Duplicates are allowed
2. Null values are allowed
3. Asynchronous
4. ArrayList
5. Whenever we have fetching operations, we need to use Arraylist 🡺 internally implements Random Access Interface (Marker Interface 🡺 there wont be any methods)
6. Adding and deleting operations at the middle should not be used.
7. By default size is 10. 🡺 ArrayList<T> list = new ArrayList<>();
8. Load factor or threshold limit🡺 75% or 0.75
9. new capacity = (current capacity\*3/2)+1
10. LinkedList
11. Adding and deleting operations in the middle we will be using Linkedlist
12. Fetching operations, we should not use linkedlist
13. Vector
14. Synchronous
15. Load factor 75%
16. Default size is 10 🡺 New capacity = 2 \* current capacity

* Set 🡺 Interface

1. Duplicates are not allowed
2. Null values are not allowed
3. Asynchronous
4. Search operations is best choice
5. HashSet: 🡺 hashMap working 🡺 Hashset values will go and store in HashMap keys
6. TreeSet 🡺 TreeMap 🡺 sorting
7. LinkedHashSet 🡺 insertion

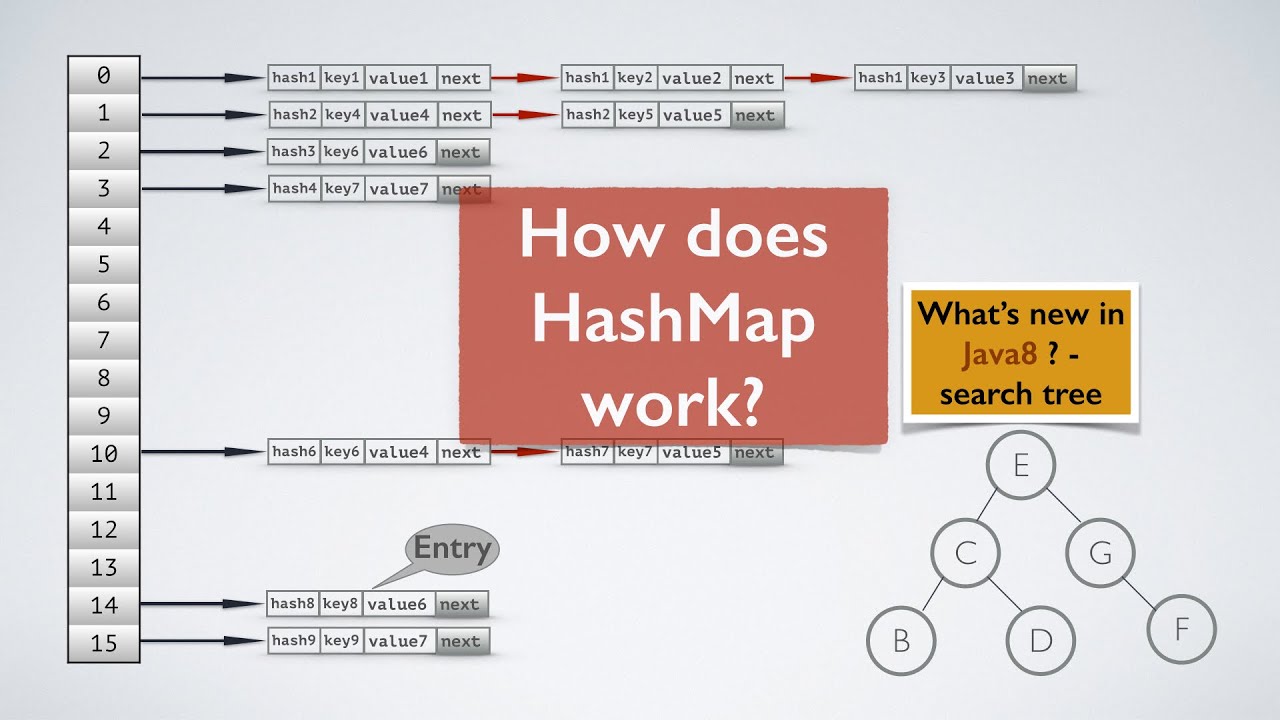
* Map🡺 Interface

1. Key and Value pairs
2. Keys 🡺 unique data and 1 null value allowed
3. Values 🡺 duplicate data and null values also allowed
4. Asynchronous
5. Directly we cant iterate maps in forloop, we need to convert it to Set Interface.
6. HashMap: random order
7. TreeMap: sorting order
8. LinkedHashMap: insertion order

HashTable 🡺 totally synchronous 🡺 no null values and keys allowed🡺 duplicates not allowed

ConcurrentHashMap: ADD, DELETE data is synchronous 🡺 get operation 🡺Asynchronous

Equals and Hash code contract:



Map<Integer, String> hashMap = new HashMap<>(); // 16 size

hashMap.put(1,”ABC”); 🡺 hashcode calculation for key 1 🡺 12345678 🡺12345678%16 🡺 3 index

hashMap.put(2,”ABC”); 🡺 hashcode calculation for key 2 🡺 12345679 🡺12345679%16 🡺 1 index

hashMap.put(3,”ABC”); 🡺 hashcode calculation for key 3 🡺 12345677 🡺12345677%16 🡺 1 index

hashMap.get(3) 🡺 hashcode calculation for key 3 🡺 12345677 🡺12345677%16 🡺 1 index 🡺 ABC

* Threshold limit 🡪 8 🡺