An object that maps keys to values. A map cannot contain duplicate keys; each key can map to at most one value.

##### **HashMap**

In order to make it fast to locate the key-value pair in Big O one time. ****Hahsmap internally uses an array of linkedlist****.

Each element in this array is a bucket. Hashmap uses the ****hashcode()**** method to calculate the ****index**** of the target bucket.

After finding the bucket, it uses the ****equals()**** method to check if there is duplicate key.

if the operation is get(), then it will return the key-value paire,

if it is put(), it will overwrite the key-value with the new value.

If there are two keys having the same hashCode(), due to the nature of hashmap, those two keys will use the same bucket. this is ****hash collision****.

The solution of hash collision is to use linkedlist to save all k-v pair and use equals() method to check the duplication of key.

// Create a map of Char-occurency in a stringString str = "AABBCCDD";

HashMap<Character, Integer> data = new HashMap<>();

for (Character c : str.toCharArray()) {

data.put(c, data.getOrDefault(c, 1));

}

##### **LinkedHashMap**

Maintain the insertion order of the key-value pairs. This implementation differs from HashMap in that it maintains a doubly-linked list running through all of its entries. This linked list defines the iteration ordering, which is normally the order in which keys were inserted into the map (insertion-order). Note that insertion order is not affected if a key is re-inserted into the map.

##### **TreeMap**

The map is sorted according to the natural ordering of its keys, or by a Comparator provided at map creation time, depending on which constructor is used.

<https://www.youtube.com/watch?v=lgBEFMzu7zo>

<https://www.youtube.com/watch?v=yANcUu6nP94>