Map [Interface]: HashMap

Definition

HashMap provides the basic implementation of the Map interface of java. It stores the data in key/value pairs, and it can be access by index or another type. One Object is used as a key (index) to another Object(value). The implementation provides all of the optional map operations, and permits null values and the null key.

The HashMap class is roughly equivalent to hashTable, except that it is unsynchronized and permits nulls. This class no guarantees that the order will remain constant over time.

An instance of HashMap has two parameters that affect its performance: initial capacity and load factor. The capacity is the number of buckets in the hash table, and the initial capacity is simply the capacity at the time the hash table is created. The load factor is a measure of how full the hash table is allowed to get before its capacity is automatically increased.

Features of HashMap

- HashMap is similar to HashTable but it is unshynchronized
- It allows to store the null keys as well, but there should be only one null key object and there can be any number of null values.
- This class makes no guarantees as to the order of the map.
- To use this class and its methods, need to import java.util.HashMap package or its superclass.

Parameters

- The type of keys maintained by this map
- The type of mapped values

Constructor summary

HashMap()	It is the default constructor which creates an instance of HashMap with an initial capacity of 16 and load factor of 0.75.
HashMap(int initalCapacity)	Creates an hashMap that has an initial size specified by initialcapacity and the default load factor is 0.75.
HashMap(int size, **float** fillratio)	Creates a Hash Map with initial size specified by size and fill ratio specified by fillratio.
HashMap(**Map** map)	This creates a hash Map with the same mappings as the specified map.

Methods summary

Modifier and Type	Method and Description
V	**put**(**K** key, **V** value) Associates the specified value with the specified key in this map.
boolean	**remove**(**Object** key, **Object** value) Removes the entry for the specified key only if it is currently mapped to the specified value.
V	**replace**(**K** key, **V** value) Replaces the entry for the specified key only if it is currently mapped to some value.
void	**forEach**(**BiConsumer** < ? super **K**, ? super V> action) Performs the given action for each entry in this map until all entries have been processed or the actions throws an exception.
boolean	**containsValue**(**Object** value) Returns true if this map maps one or more keys to this value.

When is its use recommended?

- It recommended in web applications where the username is storage as a key and user data is stored as a value.
- Use HashMap if need a map structure to map keys to values
- When to Implementing Dijkstra's algorithm
- When need to implement topological sort.