**How hash map works internally**

The HashMap in Java is a way to store key-value pairs and allows for fast retrieval of values based on keys. Here’s a detailed explanation of how HashMap works internally:

**Internal Structure**

1. **Array of Buckets**: HashMap uses an array of nodes (buckets) where each node represents a linked list or a tree (from Java 8 onwards).
2. **Node Class**: Each element in the bucket array is an instance of the inner class Node<K, V>, which stores the key, value, hash, and a reference to the next node.

**Important Components**

* **Initial Capacity**: The default initial capacity is 16.
* **Load Factor**: The default load factor is 0.75. This means the HashMap will be resized when 75% of the bucket array is filled.
* **Threshold**: The threshold is the product of the load factor and the current capacity. When the number of entries exceeds this threshold, the HashMap is resized.

**Operations**

1. **Hash Function**:
   * When a key-value pair is added to the HashMap, the hash code of the key is calculated using the key’s hashCode() method.
   * The hash code is then processed by a secondary hash function to minimize collisions.
   * The final hash is used to determine the index in the bucket array where the entry should be stored.
2. **Put Operation**:
   * Compute the hash code of the key.
   * Determine the index in the bucket array using (n - 1) & hash, where n is the array length.
   * If the bucket at the calculated index is empty, create a new node and place it there.
   * If the bucket is not empty, iterate through the linked list to check if the key already exists.
     + If the key exists, update the value.
     + If the key does not exist, add the new node at the end of the linked list.
   * If the size of the HashMap exceeds the threshold after the addition, resize the array.
3. **Get Operation**:
   * Compute the hash code of the key.
   * Determine the index in the bucket array.
   * Traverse the linked list at that index, if necessary, to find the node with the matching key.
   * Return the value associated with the key.
4. **Resize Operation**:
   * When the HashMap exceeds its threshold, it creates a new bucket array with double the capacity of the old array.
   * It rehashes all the existing entries and places them in the new array, preserving the order of nodes within each bucket.

**Treeify (From Java 8 Onwards)**

* If the number of elements in a bucket exceeds a certain threshold (default is 8), the linked list is converted to a balanced tree (red-black tree) to improve performance.
* This helps in reducing the time complexity for lookups from O(n) to O(log n).

**Summary**

* **Hashing**: Used to calculate the index for each key.
* **Buckets**: Array of nodes where each node contains a key-value pair.
* **Collision Handling**: Uses linked lists (or trees in Java 8+) to handle collisions.
* **Load Factor and Threshold**: Determines when to resize the HashMap to maintain performance.

Understanding these internals can help you use HashMap more effectively and troubleshoot any potential issues related to performance or memory usage.