How does a HashMap work? What are its key properties?

Here is the deep dive into `HashMap` internals, structured for an interview setting.

1. How HashMap Works Internally

A `HashMap` stores data in an array of **buckets** (nodes). It uses the **key's hash code** to decide which bucket to use.

Step-by-Step Flow:

- **PUT Operation (`put(K key, V value)`):**
  1. **Calculate Hash:** It calls `key.hashCode()` and then applies its own hashing function (to spread bits and reduce collisions).
  2. **Calculate Index:** It calculates the index: `index = hash & (n - 1)` (where `n` is the array size).[1] [2]
  3. **Handle Collision:** It checks if the bucket at that index is empty.
     - **If empty:** It creates a new Node and stores the entry there.
     - **If not empty (Collision):** It iterates through the linked list (or Tree) at that bucket.
       - It compares keys using `equals()`.
       - If the key exists, it **updates** the value.
       - If the key doesn't exist, it **appends** the new node to the end of the list.[2] [3]
- **GET Operation (`get(K key)`):**
  1. Calculates index using the same hash logic.
  2. Goes to that bucket and traverses the list/tree.
  3. Compares keys using `equals()` to find the exact match and returns the value.[2]

2. Key Properties & Concepts

- **Collision Handling (Java 8+ Improvement):**
  - Traditionally, buckets were **Linked Lists**. Worst-case lookup was **O(n)**.
  - Since Java 8, if a bucket grows beyond **8 nodes** (`TREEIFY_THRESHOLD`), the Linked List converts into a **Red-Black Tree**. This improves worst-case performance from **O(n)** to **O(log n)**.[3]
  - It converts back to a list if nodes drop below **6**.

- **Capacity & Load Factor:**
    - **Initial Capacity:** Default is **16**.
    - **Load Factor:** Default is **0.75**.
    - **Resizing:** When the map gets 75% full (e.g., 12 entries in size 16), it triggers a **resize**. The array size doubles (16 → 32), and **all entries are re-hashed** to new buckets. This is an expensive operation.[4] [3]
- **Null Keys/Values:**
    - Allows **one null key** (always stored at index 0) and multiple null values.[1]
- **Thread Safety:**
    - `HashMap` is **not synchronized** (not thread-safe).
    - For multi-threaded environments, use `ConcurrentHashMap` (which uses bucket-level locking/CAS) or `Collections.synchronizedMap()`.[5]

3. The `equals()` and `hashCode()` Contract

- If two objects are equal (`a.equals(b)` is true), they **must** have the same `hashCode()`.
- If this contract is broken, the HashMap will fail to retrieve the value because it might look in the wrong bucket or fail the equality check inside the correct bucket.[2]

⁂

1. https://www.freecodecamp.org/news/how-java-hashmaps-work-internal-mechanics-explained/
2. https://www.theserverside.com/video/How-a-Java-HashMap-internal-implementation-works
3. https://www.youtube.com/watch?v=xKCdp0jjZAw
4. https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/HashMap.html
5. https://www.baeldung.com/java-hashmap-advanced
6. https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/images/81815274/429036f4-3bc3-4a7a-8c6f-90246e73b9b9/image.jpg
7. https://www.digitalocean.com/community/tutorials/java-hashmap
8. https://www.w3schools.com/java/java_hashmap.asp
9. https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/images/81815274/24f18c0a-b56e-4862-a619-e2959036a5c2/image.jpg
10. https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/images/81815274/97e9ed86-ed11-4035-a18b-1eb05c1bab4e/image.jpg
11. https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/images/81815274/87f62423-96f2-4071-9802-8f6699e0ecd8/image.jpg
12. https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html
13. https://www.turing.com/kb/implementing-hashmap-in-java
14. https://www.geeksforgeeks.org/java/java-util-hashmap-in-java-with-examples/