Complete Guide to HashMap in Java

What is HashMap?

A HashMap in Java is a part of the java.util package and implements the Map interface. It stores data in key-value pairs, where each key is unique and maps to a specific value. HashMap uses a hash table as the underlying data structure to store the mappings. It allows constant time performance for the basic operations like get() and put() on average.

Basic Syntax & Initialization

To declare a HashMap in Java:

Map<KeyType, ValueType> map = new HashMap<>();

Example:

Map<String, Integer> map = new HashMap<>();

Adding Elements

Use put(key, value) to insert data into the HashMap. If the key already exists, it updates the value.

Example:

map.put("apple", 10);

Retrieving Elements

Use get(key) to retrieve the value associated with a key. Returns null if the key is not present.

Example:

Integer value = map.get("apple");

Checking Existence

Use containsKey(key) and containsValue(value) to check if a key or value exists.

Example:

boolean exists = map.containsKey("apple");

Removing Elements

Complete Guide to HashMap in Java

Use remove(key) to delete a mapping from the HashMap.

Example:

```
map.remove("apple");
```

Iterating Over HashMap

You can iterate using entrySet(), keySet(), or values().

Example:

```
for (Map.Entry<String, Integer> entry : map.entrySet()) {
    String key = entry.getKey();
    Integer value = entry.getValue();
}
```

Hashing & Collision

HashMap uses hashCode() method to determine the bucket location for a key.

If two keys return the same hashCode, a collision occurs. Collisions are handled using LinkedLists or TreeNodes.

From Java 8 onwards, LinkedList is converted to TreeNode when the list becomes large (>8 elements) for better performance.

Time and Space Complexity

Average time complexity:

- get(): O(1)
- put(): O(1)
- remove(): O(1)

Worst-case time complexity:

- O(n) when collisions are high

Space Complexity: O(n) for n key-value pairs.

Thread Safety

Complete Guide to HashMap in Java

HashMap is not thread-safe. For multithreaded environments, use:

- Collections.synchronizedMap(new HashMap<>());
- ConcurrentHashMap

Null Keys and Values

HashMap allows one null key and multiple null values.

Use Cases in DSA

- Counting frequency of elements (e.g., word frequency, number of occurrences)
- Caching results (Memoization)
- Implementing sets, maps, and lookups
- Grouping elements by certain attributes

Example Program

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Map<String, Integer> map = new HashMap<>();
        map.put("apple", 10);
        map.put("banana", 20);
        map.put("orange", 15);

        System.out.println("Value of apple: " + map.get("apple"));

        for (Map.Entry<String, Integer> entry : map.entrySet()) {
            System.out.println("Key: " + entry.getKey() + ", Value: " + entry.getValue());
        }
```

```
        map.remove("banana");
        System.out.println("Size of map: " + map.size());
    }
}
```

Conclusion

Mastering HashMap is crucial for solving many DSA problems efficiently. Practice problems involving frequency maps, prefix sums, grouping, and lookups to gain confidence.