Hashmap

Bhai 🔥

Ab main **tune diya Hashing content** ko **bilkul waise hi rakhta hoon**,

sirf code snippets me line-by-line comment add kar deta hoon —
Google/FAANG interview-ready style.

🧠8️⃣ HASHING (COMPLETE MASTER GUIDE – JAVA)

🔥 Hashing kya hoti hai? (Intuition)

👉 Value ko fast search / store / count karna

Example:

```
Array = [1,2,3,2,1,1]
Question: 1 kitni baar aaya?
❌ Loop every time → O(n)
✅ Hashing → O(1) average
```

🔑 Core Idea:

```
Key → Hash Function → Index → Value
```

🔥 Java Hashing Data Structures

Structure	Use
HashMap	Key → Value
HashSet	Unique values
Hashtable	Thread-safe HashMap (old)

1️⃣ HashMap (MOST IMPORTANT)

🔷 What is HashMap?

Stores key–value pairs

```
HashMap<Key, Value> map = new HashMap<>();
```

🔷 Basic Operations

```
map.put(key, value);      // insert key-value
map.get(key);             // get value by key
map.containsKey(key);     // check existence
map.remove(key);          // delete key
map.size();               // number of entries
```

🧠 Example

```
HashMap<Integer, Integer> freq = new HashMap<>();

freq.put(1, 3);           // key=1, value=3
freq.put(2, 5);           // key=2, value=5

System.out.println(freq.get(1)); // 3
```

🔥 Internal Working (Interview Gold)

1️⃣ Key ka `hashCode()` banta hai

2️⃣ `hashCode()` → index (bucket)

3️⃣ Collision →

- Linked List (Java 7)
- Tree (Java 8+, if chain > 8)

📌 Time Complexity

- Average: O(1)
- Worst (collision): O(n)

🔥 LeetCode SOLVED EXAMPLES (HashMap)

✅ LC 1: Two Sum ⭐⭐⭐⭐⭐

```
public int[] twoSum(int[] nums, int target) {
    HashMap<Integer, Integer> map = new HashMap<>();  // store value → i
```

```
ndex

    for (int i = 0; i < nums.length; i++) {
        int need = target - nums[i];               // value we need to find

        if (map.containsKey(need)) {               // found complement
            return new int[]{map.get(need), i};    // return indices
        }

        map.put(nums[i], i);                       // store current value & index
    }
    return new int[]{};
}
// ⏱️ Time: O(n), 🧠 Space: O(n)
```

✅ LC 560: Subarray Sum Equals K ⭐⭐⭐⭐⭐

```
public int subarraySum(int[] nums, int k) {
    HashMap<Integer, Integer> map = new HashMap<>();
    map.put(0, 1);                // subarray starting at index 0

    int sum = 0, count = 0;

    for (int x : nums) {
        sum += x;                 // cumulative sum
        count += map.getOrDefault(sum - k, 0); // check previous prefix sum
        map.put(sum, map.getOrDefault(sum, 0) + 1); // store/update prefix fre
quency
    }
    return count;
}
```

✅ LC 169: Majority Element ⭐⭐⭐⭐

```
public int majorityElement(int[] nums) {
    HashMap<Integer, Integer> map = new HashMap<>();

    for (int x : nums) {
```

```
        map.put(x, map.getOrDefault(x, 0) + 1);  // count frequency
        if (map.get(x) > nums.length / 2)        // check majority
            return x;
    }
    return -1;
}
```

2️⃣ HashSet (UNIQUE CHECK SPECIALIST)

🔷 What is HashSet?

Stores only keys, no duplicates

```
HashSet<Integer> set = new HashSet<>();
```

🔷 Operations

```
set.add(x);       // insert element
set.contains(x);  // check existence
set.remove(x);    // delete element
```

🔥 LeetCode SOLVED EXAMPLES (HashSet)

✅ LC 217: Contains Duplicate ⭐⭐⭐

```
public boolean containsDuplicate(int[] nums) {
    HashSet<Integer> set = new HashSet<>();

    for (int x : nums) {
        if (set.contains(x))        // duplicate found
            return true;
        set.add(x);                 // add element
    }
    return false;
}
```

✅ LC 3: Longest Substring Without Repeating Characters ⭐⭐⭐⭐⭐

```
public int lengthOfLongestSubstring(String s) {
    HashSet<Character> set = new HashSet<>();
    int l = 0, maxLen = 0;

    for (int r = 0; r < s.length(); r++) {
        while (set.contains(s.charAt(r))) { // duplicate found
            set.remove(s.charAt(l));       // remove leftmost
            l++;                           // shrink window
        }
        set.add(s.charAt(r));              // add current char
        maxLen = Math.max(maxLen, r - l + 1); // update max length
    }
    return maxLen;
}
```

3️⃣ Hashtable (OLD but Interview Question)

🔷 What is Hashtable?

- Thread-safe HashMap
- Synchronized
- ❌ Slower
- ❌ No null key/value

```
Hashtable<Integer, Integer> table = new Hashtable<>();
```

HashMap vs Hashtable

Feature	HashMap	Hashtable
Thread-safe	❌	✅
Performance	Fast	Slow
Null allowed	✅	❌
Modern use	✅	❌

📌 Interview Answer:

> “We prefer HashMap and use ConcurrentHashMap instead of Hashtable.”

HashMap vs HashSet

Feature	HashMap	HashSet
Stores	Key–Value	Only Keys
Duplicate key	❌	❌
Use case	Frequency, mapping	Uniqueness

❌ Common Mistakes (Interview Killers)

- Using HashMap when HashSet enough
- Forgetting `getOrDefault`
- Using HashMap for sorted data (use TreeMap)
- Assuming HashMap is ordered ❌

🏆 Interview GOLD Lines

💬 “Hashing provides average O(1) lookup time, making it ideal for frequency counting, existence checking, and subarray problems.”

💬 “When order matters, we use TreeMap; when thread safety matters, we use ConcurrentHashMap.”

🎯 UNSOLVED PRACTICE (VERY IMPORTANT)

🧪 Easy

1️⃣ LC 242 – Valid Anagram

2️⃣ LC 349 – Intersection of Two Arrays

🧪 Medium

3️⃣ LC 454 – 4Sum II

4️⃣ LC 525 – Contiguous Array

🧪 Hard

5️⃣ LC 128 – Longest Consecutive Sequence

6️⃣ LC 76 – Minimum Window Substring

Bhai 🔥

Main ab **jo tune HashMap/HashSet ka toolkit diya**, usko **bilkul interview-ready + line-by-line commented** version me rewrite kar deta hoon.

Har line me **kyon likhi, kya hota hai, interview trap** sab explain hoga.

🧠 HASHMAP – COMPLETE FUNCTIONS GUIDE (JAVA)

```
// Create a HashMap with String keys and Integer values
HashMap<String, Integer> map = new HashMap<>();

// 1️⃣ Insert / Update
// put() → insert key-value, if key exists → overwrite
map.put("a", 1);       // key "a" doesn't exist, value=1 inserted
map.put("a", 5);       // key "a" exists, old value overwritten with 5

// putIfAbsent() → insert only if key not present
map.putIfAbsent("a", 10); // key "a" exists, so ignored
map.putIfAbsent("b", 20); // key "b" not present → inserted

// 2️⃣ Access / Search
// get() → get value, returns null if key not present
Integer val1 = map.get("a");  // returns 5
Integer val2 = map.get("c");  // returns null (key not present)

// getOrDefault() → get value or default if key not present (interview favorit
e)
int val3 = map.getOrDefault("b", 0); // key "b" exists → returns 20
int val4 = map.getOrDefault("z", 0); // key "z" not present → returns 0

// containsKey() → check if key exists
boolean existsKey = map.containsKey("a");  // true
boolean existsKey2 = map.containsKey("z"); // false

// containsValue() → check if value exists
boolean existsVal = map.containsValue(5);  // true
boolean existsVal2 = map.containsValue(100); // false
```

```
// 3️⃣ Remove
// remove(key) → removes key if present
map.remove("b");       // removes key "b" from map

// remove(key, value) → removes only if value matches
map.remove("a", 1);    // key "a" exists but value=5 → doesn't remove
map.remove("a", 5);    // key "a" exists and value=5 → removed

// 4️⃣ Size / Check
int size = map.size();     // number of key-value pairs
boolean empty = map.isEmpty(); // true if map is empty

// 5️⃣ Iteration (🔥 Interview Favorite)
// Iterate over keys only
for (String key : map.keySet()) {
    System.out.println("Key: " + key); // prints keys
}

// Iterate over values only
for (Integer value : map.values()) {
    System.out.println("Value: " + value); // prints values
}

// Iterate over key-value pairs → most efficient
for (Map.Entry<String, Integer> entry : map.entrySet()) {
    System.out.println(entry.getKey() + " " + entry.getValue());
    // entry.getKey() → key
    // entry.getValue() → value
}

// Iterator version (less common, but asked in interviews)
Iterator<Map.Entry<String, Integer>> it = map.entrySet().iterator();
while (it.hasNext()) {
    Map.Entry<String, Integer> e = it.next();
    System.out.println(e.getKey() + " " + e.getValue());
}

// 6️⃣ Update Values (Frequency Pattern)
```

```
String x = "apple";
// get existing count → default 0 if key not present → increment by 1
map.put(x, map.getOrDefault(x, 0) + 1);

// 7️⃣ Clear / Replace
map.clear();            // removes all entries
map.replace("a", 10);    // replaces value for existing key "a" only

// 8️⃣ Important Interview Utilities
// Check key existence + get value safely
if (map.containsKey("b")) {
   int val = map.get("b"); // safe access
}

// First time prefix sum seen (prefix sum problems)
int sum = 10;
int i = 3;
if (!map.containsKey(sum)) {
   map.put(sum, i);      // store first occurrence of sum
}
```

🧠 HASHSET – COMPLETE FUNCTIONS GUIDE (JAVA)

```
// Create a HashSet of Integers
HashSet<Integer> set = new HashSet<>();

// 1️⃣ Insert
set.add(10); // insert 10
set.add(10); // duplicate ignored automatically

// 2️⃣ Search
boolean has10 = set.contains(10);  // true
boolean has5 = set.contains(5);    // false

// 3️⃣ Remove
set.remove(10); // removes element 10 if present
```

```
// 4️⃣ Size / Empty
int sizeSet = set.size();      // number of elements
boolean emptySet = set.isEmpty(); // true if empty

// 5️⃣ Iteration (🔥 Interview favorite)
// Enhanced for-loop
for (int x : set) {
    System.out.println(x);  // prints each element
}

// Iterator version
Iterator<Integer> itSet = set.iterator();
while (itSet.hasNext()) {
    int x = itSet.next();
    System.out.println(x);
}

// 6️⃣ Clear
set.clear(); // removes all elements
```

🔥 HashMap vs HashSet – Function Use Case

Task	Use
Frequency count	HashMap
Duplicate check	HashSet
Key → value map	HashMap
Only unique elems	HashSet

🧪 COMMON INTERVIEW PATTERNS

```
// Frequency Count
map.put(x, map.getOrDefault(x, 0) + 1);

// First Unique Element
for (int x : arr) {
```

```
    map.put(x, map.getOrDefault(x, 0) + 1); // count frequency
}

for (int x : arr) {
    if (map.get(x) == 1) {              // first element with freq=1
        System.out.println(x);
        break;
    }
}

// Duplicate Detection using HashSet
HashSet<Integer> setDup = new HashSet<>();
for (int x : arr) {
    if (!setDup.add(x)) {               // add returns false if duplicate
        System.out.println("Duplicate found: " + x);
    }
}
```

🏆 Interview GOLD Lines

💬

"HashMap is used for key-value mapping and frequency counting, while HashSet is used when only uniqueness matters."

💬

"Iteration over HashMap is best done using entrySet() for performance."

❌ Common Mistakes

- Using `map.get()` without null check → may cause NullPointerException
- Iterating map using only keys when value also needed → less efficient
- Expecting HashSet to be ordered ❌ → it's unordered

Bhai 🔥

Ab main tera **Hashing with Arrays** content ko **Java + interview-ready + line-by-line explanation** ke saath next-level version me rewrite kar raha hoon.

Har snippet ka **purpose + intuition** comment me dala hai.

🧠 Hashing with Arrays – COMPLETE GUIDE (JAVA + INTERVIEW)

🔷 Hashing kya hai?

> Value ko direct index/key bana ke store karna, taaki search / count / duplicate check O(1) me ho jaye.

Rule of Thumb:

> Time bachana ho → space sacrifice karo

1️⃣ Frequency Array (Basic Hashing)

📌 Kab use kare?

- Elements **small & known range** me ho
- Mostly **non-negative**
- Examples: 0 → 10^5, lowercase a–z, digits 0–9

✅ Java Example: Count Frequency

```
int[] freq = new int[100001];      // Create frequency array for known small r
ange

for (int x : arr) {                // Traverse each element
    freq[x]++;                      // Increment count of value x
}
```

Use Cases:

- Count occurrences
- Find duplicates
- Majority element
- Anagram check
- Character frequency

🔥 Example: Find Duplicates

```
for (int x : arr) {
    if (freq[x] > 1) {              // If count > 1 → duplicate
        System.out.println(x);      // Print duplicate value
    }
}
```

⚠️ **Limitations**

- Negative numbers ❌
- Very large values (10^9) ❌
- Memory waste if range huge ❌

➡️ **Solution:** HashMap

2️⃣ HashMap Based Hashing

📌 **Kab use kare?**

- Values large / negative
- Range unknown
- Space flexible

```
HashMap<Integer, Integer> map = new HashMap<>(); // Key → Value (frequency, index, etc.)
```

⏱️ Average Time: O(1)

🧠 Space: O(n)

🔷 MUST-KNOW HASHING PROBLEMS (JAVA)

✅ 1️⃣ Majority Element (> n/2)

```
public int majorityElement(int[] nums) {
    HashMap<Integer, Integer> map = new HashMap<>(); // Frequency map
    int n = nums.length;

    for (int x : nums) {
        map.put(x, map.getOrDefault(x, 0) + 1);     // Increment frequency
```

```
        if (map.get(x) > n / 2)                 // Check majority condition
            return x;                           // Found majority element
    }
    return -1;                                  // No majority element
}
```

⏱️ O(n) | 🧠 O(n)

📌 **Interview note:**

Better approach = Boyer–Moore (O(1) space)

✅2️⃣ Two Sum (Google Favourite)

```
public int[] twoSum(int[] nums, int target) {
    HashMap<Integer, Integer> map = new HashMap<>(); // Store value → i
ndex

    for (int i = 0; i < nums.length; i++) {
        int need = target - nums[i];            // Find complement

        if (map.containsKey(need))              // Complement exists?
            return new int[]{map.get(need), i};   // Return indices

        map.put(nums[i], i);                    // Store current value
    }
    return new int[]{};
}
```

⏱️ O(n) | 🧠 O(n)

Interview tip: Always store **value → index** for O(1) lookup.

✅3️⃣ Missing Number (0 to n)

```
public int missingNumber(int[] nums) {
    HashSet<Integer> set = new HashSet<>();      // Store all numbers

    for (int x : nums)
        set.add(x);
```

```
    for (int i = 0; i <= nums.length; i++) {     // Check each number in range
        if (!set.contains(i))                    // Missing number found
            return i;
    }
    return -1;
}
```

📌 **Better:** XOR / Math sum formula → O(1) space

✅4️⃣ Find Duplicate Number

```
public int findDuplicate(int[] nums) {
    HashSet<Integer> set = new HashSet<>();      // Track seen numbers

    for (int x : nums) {
        if (!set.add(x))                         // add() returns false if already exists
            return x;                            // Duplicate found
    }
    return -1;
}
```

📌 **Better:** Floyd's Cycle Detection (O(1) space)

🧠 Hashing Decision Guide (INTERVIEW GOLD)

Problem Type	Best Approach
Small range	Frequency Array
Large / negative	HashMap
Only duplicates	HashSet
Need O(1) time	Hashing
Space optimize	Math / XOR
Sorted data	Two pointer

⚠️ COMMON INTERVIEW MISTAKES

- map vs unordered_map confusion (C++)

- Ignoring hash collisions
- Forgetting space complexity
- Using frequency array when range is huge

🏆 INTERVIEW GOLD LINE

> “Hashing improves time complexity by trading space for constant-time lookups, making it ideal for counting, searching, and duplicate detection problems.”

🚀 FINAL TAKEAWAY

- **Frequency array** → fastest if range small
- **HashMap** → flexible & powerful
- **HashSet** → uniqueness specialist

> Always justify space–time tradeoff in interviews.
