Map Interface in Java - HashMap

Map is an interface in Java that stores data in key-value pairs. Each key is unique and maps to exactly one value. The most commonly used implementation is HashMap.

Key Characteristics of HashMap

- Stores data as key-value pairs
- Keys are unique - no duplicate keys allowed
- Values can be duplicate - multiple keys can have the same value
- One null key allowed, multiple null values allowed
- Unordered - no guarantee of order
- Fast operations - O(1) average time complexity

---

1. Creating a HashMap

java

```java
import java.util.HashMap;
import java.util.Map;

public class HashMapDemo {
  public static void main(String[] args) {
    // Key type: String, Value type: Integer
    Map<String, Integer> map = new HashMap<>();
    // or
    HashMap<String, Integer> studentMarks = new HashMap<>();
  }
}
```

---

2. put(K key, V value) - Add/Update Key-Value Pair

Adds a key-value pair to the map. If the key already exists, it overwrites the old value with the new value.

java

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

// Adding key-value pairs
studentMarks.put("Alice", 85);
studentMarks.put("Bob", 90);
studentMarks.put("Charlie", 78);

System.out.println(studentMarks);
// Output: {Alice=85, Bob=90, Charlie=78}
```

---

3. Key is Unique - Overriding with Same Key

If you use put() with an existing key, it replaces the old value:

java

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

map.put("Alice", 85);
System.out.println(map); // {Alice=85}

// Using same key with different value - OVERWRITES!
map.put("Alice", 95);
System.out.println(map); // {Alice=95} - old value 85 is replaced
```

Important: The key "Alice" appears only once, but with the updated value!

---

4. Preventing Override Using if Condition

To avoid accidentally overwriting an existing value, check if the key exists first:

java

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

```
map.put("Alice", 85);
map.put("Bob", 90);

// Check before putting to prevent override
if (!map.containsKey("Alice")) {
  map.put("Alice", 95);
  System.out.println("Value updated");
} else {
  System.out.println("Key already exists! Value NOT updated");
}

System.out.println(map); // {Alice=85, Bob=90} - original value preserved
```

Alternatively, check and update in one step:

java

```
String key = "Charlie";
Integer newValue = 88;

// Only add if key doesn't exist
if (!map.containsKey(key)) {
  map.put(key, newValue);
}

System.out.println(map); // {Alice=85, Bob=90, Charlie=88}
```

---

5. putIfAbsent(K key, V value) - Add Only if Key Doesn't Exist

This method adds the key-value pair only if the key is not already present. It returns the existing value if the key exists, or null if it was added.

java

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

map.put("Alice", 85);
map.put("Bob", 90);

// Try to add - key doesn't exist, so it gets added
map.putIfAbsent("Charlie", 78);
System.out.println(map); // {Alice=85, Bob=90, Charlie=78}

// Try to add - key already exists, so nothing happens
map.putIfAbsent("Alice", 95);
System.out.println(map); // {Alice=85, Bob=90, Charlie=78} - Alice still 85!

System.out.println(map.putIfAbsent("Diana", 82)); // null (key added)
System.out.println(map.putIfAbsent("Bob", 100));  // 90 (key exists, returns old value)
```

Key Difference:

- put() - Always updates/replaces the value
- putIfAbsent() - Only adds if key doesn't exist, preserves existing values

---

6. Printing Each Entry Using Map.Entry and entrySet()

To iterate through all key-value pairs, use entrySet() which returns a Set of Map.Entry objects:

java

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

studentMarks.put("Alice", 85);
studentMarks.put("Bob", 90);
studentMarks.put("Charlie", 78);
studentMarks.put("Diana", 92);

System.out.println("Student Marks:");

// Using Map.Entry and entrySet()
for (Map.Entry<String, Integer> e : studentMarks.entrySet()) {
  System.out.println("Name: " + e.getKey() + ", Marks: " + e.getValue());
}

// Output:
// Name: Alice, Marks: 85
// Name: Bob, Marks: 90
// Name: Charlie, Marks: 78
// Name: Diana, Marks: 92
```

Explanation:

- **entrySet() - Returns a Set view of all key-value pairs as Map.Entry objects**
- **e.getKey() - Gets the key from the entry**
- **e.getValue() - Gets the value from the entry**

---

7. keySet() - Get All Keys

Returns a Set containing all the keys in the map:

java

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

map.put("Alice", 85);
map.put("Bob", 90);
map.put("Charlie", 78);
```

```
// Get all keys
System.out.println("All Keys: " + map.keySet());
// Output: [Alice, Bob, Charlie]

// Iterate through keys
System.out.println("\nIterating through keys:");
for (String key : map.keySet()) {
   System.out.println("Key: " + key + ", Value: " + map.get(key));
}
```

---

8. values() - Get All Values

Returns a Collection containing all the values in the map:

java

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

map.put("Alice", 85);
map.put("Bob", 90);
map.put("Charlie", 78);

// Get all values
System.out.println("All Values: " + map.values());
// Output: [85, 90, 78]

// Iterate through values
System.out.println("\nIterating through values:");
for (Integer value : map.values()) {
   System.out.println("Value: " + value);
}
```

---

9. containsKey(Object key) - Check if Key Exists

Returns true if the map contains the specified key:

java

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


map.put("Alice", 85);

map.put("Bob", 90);


System.out.println(map.containsKey("Alice"));   // true

System.out.println(map.containsKey("Charlie")); // false


// Practical use

if (map.containsKey("Bob")) {

   System.out.println("Bob's marks: " + map.get("Bob")); // 90

}
```

---

10. containsValue(Object value) - Check if Value Exists

Returns true if the map contains the specified value:

java

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


map.put("Alice", 85);

map.put("Bob", 90);

map.put("Charlie", 85); // Same value as Alice


System.out.println(map.containsValue(90));  // true
System.out.println(map.containsValue(100)); // false
System.out.println(map.containsValue(85));  // true (found in Alice's entry)
```

Note: This is slower (O(n)) than containsKey() because it has to search through all values.

---

11. isEmpty() - Check if Map is Empty

Returns true if the map has no key-value pairs:

java

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


System.out.println("Is empty? " + map.isEmpty()); // true


map.put("Alice", 85);
System.out.println("Is empty? " + map.isEmpty()); // false
```

---

12. clear() - Remove All Entries

Removes all key-value pairs from the map:

java

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


map.put("Alice", 85);
map.put("Bob", 90);
map.put("Charlie", 78);


System.out.println("Before clear: " + map); // {Alice=85, Bob=90, Charlie=78}
System.out.println("Size: " + map.size());  // 3


map.clear();
```

```
System.out.println("After clear: " + map);  // {}

System.out.println("Size: " + map.size());  // 0

System.out.println("Is empty? " + map.isEmpty()); // true
```

Complete Example - Putting It All Together

java

```
import java.util.HashMap;

import java.util.Map;


public class CompleteHashMapDemo {

  public static void main(String[] args) {

    // Creating HashMap

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


    // 1. Adding entries using put()

    System.out.println("=== Adding Entries ===");

    studentMarks.put("Alice", 85);

    studentMarks.put("Bob", 90);

    studentMarks.put("Charlie", 78);

    studentMarks.put("Diana", 92);

    System.out.println(studentMarks);


    // 2. Overriding with same key

    System.out.println("\n=== Overriding Value ===");

    System.out.println("Before: Alice = " + studentMarks.get("Alice")); // 85

    studentMarks.put("Alice", 95); // Overwrites!

    System.out.println("After: Alice = " + studentMarks.get("Alice"));  // 95
```

```
// 3. Preventing override with if condition
System.out.println("\n=== Preventing Override ===");
if (!studentMarks.containsKey("Bob")) {
   studentMarks.put("Bob", 100);
} else {
   System.out.println("Bob already exists. Value NOT updated.");
}
System.out.println("Bob's marks: " + studentMarks.get("Bob")); // Still 90

// 4. Using putIfAbsent
System.out.println("\n=== Using putIfAbsent ===");
studentMarks.putIfAbsent("Eve", 88);     // Added (new key)
studentMarks.putIfAbsent("Charlie", 100); // Not added (key exists)
System.out.println(studentMarks);

// 5. Printing using Map.Entry and entrySet()
System.out.println("\n=== Printing All Entries ===");
for (Map.Entry<String, Integer> e : studentMarks.entrySet()) {
   System.out.println("Student: " + e.getKey() + ", Marks: " + e.getValue());
}

// 6. Using keySet()
System.out.println("\n=== All Keys ===");
System.out.println(studentMarks.keySet());

// 7. Using values()
System.out.println("\n=== All Values ===");
```

```
    System.out.println(studentMarks.values());


    // 8. containsKey()

    System.out.println("\n=== Contains Key ===");

    System.out.println("Contains 'Diana'? " + studentMarks.containsKey("Diana"));
// true

    System.out.println("Contains 'Frank'? " + studentMarks.containsKey("Frank"));
// false


    // 9. containsValue()

    System.out.println("\n=== Contains Value ===");

    System.out.println("Contains value 92? " + studentMarks.containsValue(92));
// true

    System.out.println("Contains value 100? " + studentMarks.containsValue(100));
// false


    // 10. isEmpty()

    System.out.println("\n=== Is Empty ===");

    System.out.println("Is map empty? " + studentMarks.isEmpty()); // false


    // 11. Size

    System.out.println("Size: " + studentMarks.size()); // 5


    // 12. clear()

    System.out.println("\n=== Clearing Map ===");

    studentMarks.clear();

    System.out.println("After clear: " + studentMarks);        // {}

    System.out.println("Is map empty? " + studentMarks.isEmpty()); // true

    System.out.println("Size: " + studentMarks.size());        // 0
    }
}
```

TreeMap in Java

TreeMap is a Map implementation that stores key-value pairs in **sorted order based on keys**. It uses a **Red-Black tree** data structure internally. Keys are automatically sorted in ascending (alphabetical/numerical) order.

Key Characteristics of TreeMap

- Stores data as **key-value pairs** (like HashMap)
- **Keys are sorted** - alphabetically for Strings, numerically for numbers
- **Keys are unique** - no duplicate keys allowed
- **No null keys** allowed (throws NullPointerException), but null values are allowed
- **Slower than HashMap** - O(log n) time complexity
- Keys must be **comparable** or use a custom Comparator

---

1. Creating a TreeMap

java

```
import java.util.TreeMap;
import java.util.Map;

public class TreeMapDemo {
    public static void main(String[] args) {
        // Key type: String, Value type: Integer
        Map<String, Integer> map = new TreeMap<>();
        // or
        TreeMap<String, Integer> studentMarks = new TreeMap<>();
```

```
   }
}
```

---

2. Letter-wise (Alphabetical) Key Arrangement

The most important feature - keys are **automatically sorted in alphabetical order** for Strings:

java

```
TreeMap<String, Integer> map = new TreeMap<>();


// Adding in random order
map.put("Diana", 92);
map.put("Alice", 85);
map.put("Charlie", 78);
map.put("Bob", 90);


System.out.println(map);
// Output: {Alice=85, Bob=90, Charlie=78, Diana=92} - SORTED BY KEY!
```

No matter what order you add entries, TreeMap always keeps them sorted alphabetically by key!

Comparison with HashMap

java

```
// HashMap - Random/Unpredictable order
HashMap<String, Integer> hashMap = new HashMap<>();
hashMap.put("Diana", 92);
hashMap.put("Alice", 85);
hashMap.put("Charlie", 78);
hashMap.put("Bob", 90);
System.out.println("HashMap: " + hashMap);
// Output: {Bob=90, Alice=85, Diana=92, Charlie=78} - Random order
// TreeMap - Sorted order
TreeMap<String, Integer> treeMap = new TreeMap<>();
treeMap.put("Diana", 92);
treeMap.put("Alice", 85);
treeMap.put("Charlie", 78);
treeMap.put("Bob", 90);
System.out.println("TreeMap: " + treeMap);
// Output: {Alice=85, Bob=90, Charlie=78, Diana=92} - Alphabetically sorted!
```

---

3. put(K key, V value) - Add/Update Key-Value Pair

Adds a key-value pair in sorted position. Overwrites if key already exists.

java

```
TreeMap<String, Integer> map = new TreeMap<>();

map.put("Zebra", 100);
map.put("Apple", 50);
map.put("Mango", 75);

System.out.println(map);
// {Apple=50, Mango=75, Zebra=100} - Sorted alphabetically!

// Overwriting existing key
map.put("Apple", 60); // Updates Apple's value
System.out.println(map);
// {Apple=60, Mango=75, Zebra=100}
```

---

4. Preventing Override Using if Condition

Same as HashMap - check if key exists before updating:

java

```
TreeMap<String, Integer> map = new TreeMap<>();


map.put("Alice", 85);
map.put("Bob", 90);


// Prevent override
if (!map.containsKey("Alice")) {
   map.put("Alice", 95);
} else {
   System.out.println("Key 'Alice' already exists! Value NOT updated.");
}


System.out.println(map); // {Alice=85, Bob=90} - original preserved
```

---

5. putIfAbsent(K key, V value) - Add Only if Key Doesn't Exist

Adds key-value pair only if key is absent:

java

```
TreeMap<String, Integer> map = new TreeMap<>();


map.put("Alice", 85);
map.put("Bob", 90);


// Key doesn't exist - gets added
map.putIfAbsent("Charlie", 78);
System.out.println(map); // {Alice=85, Bob=90, Charlie=78}
```

```
// Key exists - nothing happens

map.putIfAbsent("Alice", 95);

System.out.println(map); // {Alice=85, Bob=90, Charlie=78} - Alice still 85!


// Adding more entries

map.putIfAbsent("Zebra", 100);

map.putIfAbsent("Diana", 88);

System.out.println(map);

// {Alice=85, Bob=90, Charlie=78, Diana=88, Zebra=100} - All sorted!
```

---

6. remove(Object key) - Remove Entry by Key

Removes the key-value pair for the specified key. Remaining entries stay sorted.

java

```
TreeMap<String, Integer> map = new TreeMap<>();


map.put("Alice", 85);

map.put("Bob", 90);

map.put("Charlie", 78);

map.put("Diana", 92);

map.put("Eve", 88);


System.out.println("Before: " + map);

// {Alice=85, Bob=90, Charlie=78, Diana=92, Eve=88}


// Remove entries

map.remove("Charlie");

System.out.println("After removing Charlie: " + map);

// {Alice=85, Bob=90, Diana=92, Eve=88} - Still sorted!
```

```
map.remove("Alice");
System.out.println("After removing Alice: " + map);
// {Bob=90, Diana=92, Eve=88}

// Try to remove non-existent key
System.out.println("Remove result: " + map.remove("Frank")); // null
```

---

7. Printing Using Map.Entry and entrySet()

Iterate through all entries in sorted order:

java

```
TreeMap<String, Integer> studentMarks = new TreeMap<>();

studentMarks.put("Charlie", 78);
studentMarks.put("Alice", 85);
studentMarks.put("Diana", 92);
studentMarks.put("Bob", 90);

System.out.println("Student Marks (Alphabetically):");

for (Map.Entry<String, Integer> e : studentMarks.entrySet()) {
   System.out.println("Student: " + e.getKey() + ", Marks: " + e.getValue());
}

// Output (sorted by key):
// Student: Alice, Marks: 85
// Student: Bob, Marks: 90
// Student: Charlie, Marks: 78
// Student: Diana, Marks: 92
```

---

8. keySet() - Get All Keys in Sorted Order

Returns a Set of all keys in alphabetical order:

java

```
TreeMap<String, Integer> map = new TreeMap<>();


map.put("Zebra", 100);
map.put("Apple", 50);
map.put("Mango", 75);
map.put("Banana", 60);


System.out.println("All Keys (sorted): " + map.keySet());
// [Apple, Banana, Mango, Zebra]


// Iterate through keys
for (String key : map.keySet()) {
   System.out.println(key + " -> " + map.get(key));
}
// Apple -> 50
// Banana -> 60
// Mango -> 75
// Zebra -> 100
```

---

9. values() - Get All Values

Returns a Collection of all values (in the order of sorted keys):

java

```
TreeMap<String, Integer> map = new TreeMap<>();
```

```
map.put("Charlie", 78);
map.put("Alice", 85);
map.put("Bob", 90);

System.out.println("All Values: " + map.values());
// [85, 90, 78] - order follows sorted keys (Alice, Bob, Charlie)
```

---

10. containsKey(Object key) - Check if Key Exists

java

```
TreeMap<String, Integer> map = new TreeMap<>();

map.put("Alice", 85);
map.put("Bob", 90);

System.out.println(map.containsKey("Alice"));   // true
System.out.println(map.containsKey("Charlie")); // false
```

---

11. containsValue(Object value) - Check if Value Exists

java

```
TreeMap<String, Integer> map = new TreeMap<>();

map.put("Alice", 85);
map.put("Bob", 90);
map.put("Charlie", 85); // Duplicate value

System.out.println(map.containsValue(90));  // true
System.out.println(map.containsValue(100)); // false
System.out.println(map.containsValue(85));  // true
```

---

12. isEmpty() - Check if Map is Empty

java

```
TreeMap<String, Integer> map = new TreeMap<>();

System.out.println("Is empty? " + map.isEmpty()); // true

map.put("Alice", 85);
System.out.println("Is empty? " + map.isEmpty()); // false
```

---

13. size() - Get Number of Entries

java

```
TreeMap<String, Integer> map = new TreeMap<>();

map.put("Alice", 85);
map.put("Bob", 90);
map.put("Charlie", 78);

System.out.println("Size: " + map.size()); // 3
```

---

14. clear() - Remove All Entries

java

```
TreeMap<String, Integer> map = new TreeMap<>();

map.put("Alice", 85);
map.put("Bob", 90);
map.put("Charlie", 78);
```

```
System.out.println("Before clear: " + map); // {Alice=85, Bob=90, Charlie=78}
map.clear();
System.out.println("After clear: " + map);  // {}
System.out.println("Is empty? " + map.isEmpty()); // true
```

Additional TreeMap-Specific Methods

TreeMap has special navigation methods due to sorted nature:

firstKey() and lastKey() - Get First/Last Key

java

```
TreeMap<String, Integer> map = new TreeMap<>();
map.put("Charlie", 78);
map.put("Alice", 85);
map.put("Diana", 92);

System.out.println("First key: " + map.firstKey()); // Alice
System.out.println("Last key: " + map.lastKey());   // Diana
```

firstEntry() and lastEntry() - Get First/Last Entry

java

```
System.out.println("First entry: " + map.firstEntry()); // Alice=85
System.out.println("Last entry: " + map.lastEntry());   // Diana=92
```

higherKey() and lowerKey() - Get Next/Previous Key

java

```
TreeMap<String, Integer> map = new TreeMap<>();
map.put("Alice", 85);
map.put("Bob", 90);
map.put("Charlie", 78);
map.put("Diana", 92);
```

```
System.out.println("Higher than 'Bob': " + map.higherKey("Bob"));    // Charlie
System.out.println("Lower than 'Charlie': " + map.lowerKey("Charlie")); // Bob
```

Complete Example

java

```
import java.util.TreeMap;
import java.util.Map;

public class CompleteTreeMapDemo {
    public static void main(String[] args) {
        // Creating TreeMap
        TreeMap<String, Integer> studentMarks = new TreeMap<>();

        // 1. Adding entries (in random order)
        System.out.println("=== Adding Entries ===");
        studentMarks.put("Diana", 92);
        studentMarks.put("Alice", 85);
        studentMarks.put("Charlie", 78);
        studentMarks.put("Bob", 90);
        studentMarks.put("Eve", 88);
        System.out.println(studentMarks);
        // {Alice=85, Bob=90, Charlie=78, Diana=92, Eve=88} - SORTED!

        // 2. Overriding with same key
        System.out.println("\n=== Overriding Value ===");
        studentMarks.put("Alice", 95);
        System.out.println("Alice's new marks: " + studentMarks.get("Alice")); // 95
```

```
// 3. Preventing override with if condition
System.out.println("\n=== Preventing Override ===");
if (!studentMarks.containsKey("Bob")) {
    studentMarks.put("Bob", 100);
} else {
    System.out.println("Bob already exists. Not updated.");
}

// 4. Using putIfAbsent
System.out.println("\n=== Using putIfAbsent ===");
studentMarks.putIfAbsent("Frank", 82);  // Added
studentMarks.putIfAbsent("Charlie", 100); // Not added
System.out.println(studentMarks);
// Still alphabetically sorted!

// 5. Printing using Map.Entry and entrySet()
System.out.println("\n=== All Entries (Sorted) ===");
for (Map.Entry<String, Integer> e : studentMarks.entrySet()) {
    System.out.println(e.getKey() + " -> " + e.getValue());
}

// 6. keySet() - sorted keys
System.out.println("\n=== All Keys (Sorted) ===");
System.out.println(studentMarks.keySet());

// 7. values()
System.out.println("\n=== All Values ===");
```

```
    System.out.println(studentMarks.values());

    // 8. containsKey()
    System.out.println("\n=== Contains Key ===");
    System.out.println("Contains 'Diana'? " + studentMarks.containsKey("Diana")); // true
    System.out.println("Contains 'George'? " + studentMarks.containsKey("George")); // false

    // 9. containsValue()
    System.out.println("\n=== Contains Value ===");
    System.out.println("Contains 92? " + studentMarks.containsValue(92));  // true
    System.out.println("Contains 100? " + studentMarks.containsValue(100)); // false

    // 10. TreeMap specific methods
    System.out.println("\n=== TreeMap Specific Methods ===");
    System.out.println("First key: " + studentMarks.firstKey());   // Alice
    System.out.println("Last key: " + studentMarks.lastKey());     // Frank
    System.out.println("Higher than 'Charlie': " + studentMarks.higherKey("Charlie")); // Diana
    System.out.println("Lower than 'Eve': " + studentMarks.lowerKey("Eve"));         // Diana

    // 11. remove()
    System.out.println("\n=== Removing Entries ===");
    studentMarks.remove("Charlie");
    studentMarks.remove("Eve");
    System.out.println("After removals: " + studentMarks);
    // Still sorted!
```

```
        // 12. isEmpty() and size()
        System.out.println("\n=== Size and Empty Check ===");
        System.out.println("Size: " + studentMarks.size());         // 4
        System.out.println("Is empty? " + studentMarks.isEmpty());    // false

        // 13. clear()
        System.out.println("\n=== Clearing Map ===");
        studentMarks.clear();
        System.out.println("After clear: " + studentMarks);         // {}
        System.out.println("Is empty? " + studentMarks.isEmpty());    // true
    }
}
```

TreeMap with Numbers (Numerical Sorting)

java

```
TreeMap<Integer, String> map = new TreeMap<>();

// Adding in random order
map.put(50, "Fifty");
map.put(10, "Ten");
map.put(30, "Thirty");
map.put(20, "Twenty");

System.out.println(map);
// {10=Ten, 20=Twenty, 30=Thirty, 50=Fifty} - Sorted numerically by key!
```