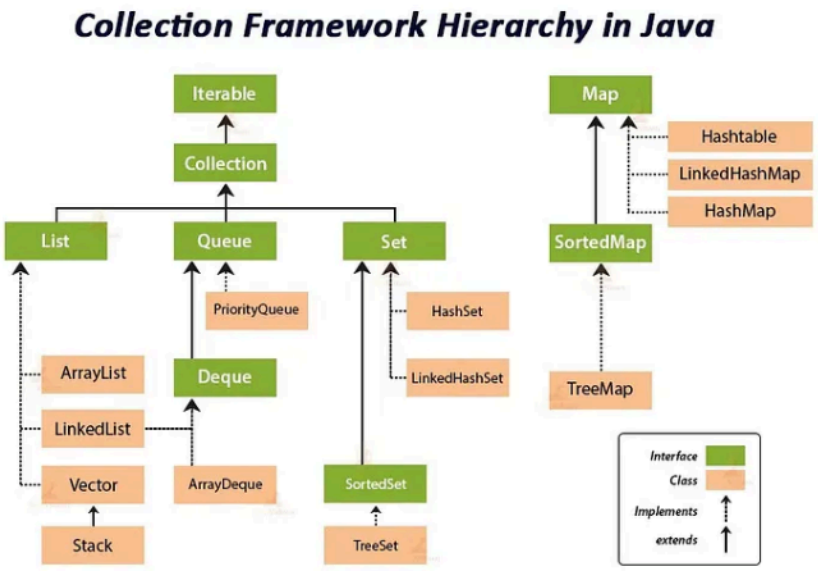
Map



HashMap

- `HashMap` is part of the **Java Collections Framework**.
- Stores data as **key-value pairs**.
- Keys must be **unique**; values can be **duplicated**.
- Allows **one** `null` **key** and **multiple** `null` **values**.
- Does **not maintain insertion order**
- Not thread safe.

⚙️ Internal Working of `HashMap`

1. When `put(key, value)` is called:
   - The **hashCode()** of the key is calculated.
   - That hash is converted into a **bucket index**.
   - If the bucket is empty, the entry is placed there.
   - If not, Java checks for:

- Duplicate key (replace the value if same)
- Collision (different keys but same bucket) – stored as a **linked list** or **tree** (Java 8+)

2. Retrieval uses the key's hash to find the bucket and the **equals()** method to find the exact match.

```java
import java.util.HashMap;

public class HashMapExample {
    public static void main(String[] args) {
        HashMap<Integer, String> map = new HashMap<>();

        map.put(1, "Java");
        map.put(2, "Spring");
        map.put(3, "Boot");
        map.put(1, "Updated"); // Overwrites previous value
        map.put(null, "NullKey");
        map.put(4, null); // null value

        System.out.println("Map: " + map);
        System.out.println("Value for key 2: " + map.get(2));
        System.out.println("Contains key 3? " + map.containsKey(3));
        System.out.println("All keys: " + map.keySet());
        System.out.println("All values: " + map.values());

        map.remove(2);
        System.out.println("After removing key 2: " + map);
    }
}

```

```
Map: {null=NullKey, 1=Updated, 3=Boot, 4=null}
Value for key 2: Spring
Contains key 3? true
All keys: [null, 1, 3, 4]
All values: [NullKey, Updated, Boot, null]
After removing key 2: {null=NullKey, 1=Updated, 3=Boot, 4=null}

```

LinkedHashMap :

📌 **What is `LinkedHashMap`?**

- `LinkedHashMap` is a part of the **Java Collections Framework**.
- It **extends** `HashMap` and implements the `Map` interface.
- Stores **key-value pairs**, like `HashMap`, but **preserves insertion order**.
- Allows **one** `null` **key** and **multiple** `null` **values**.
- **Faster iteration** compared to `HashMap` due to predictable order

- not thread safe

⚙️ Internal Working of `LinkedHashMap`

- Inherits from `HashMap`, so it uses **hashing** for key placement.
- Maintains a **doubly linked list** to keep track of **insertion order** or **access order** (optional).
- When you `put()`:
  - Hash is calculated for the key → bucket chosen
  - Entry is placed in hash table and also linked at the **end** of the linked list
- Access order mode: If enabled (`accessOrder=true`), recently accessed entries are moved to the end (used in LRU cache implementations).

```java
import java.util.LinkedHashMap;

public class LinkedHashMapExample {
    public static void main(String[] args) {
        LinkedHashMap<Integer, String> map = new LinkedHashMap<>();

        map.put(101, "Java");
        map.put(102, "Spring");
        map.put(103, "Boot");
        map.put(101, "Updated Java"); // Overwrites
        map.put(null, "NullKey");     // One null key allowed
        map.put(104, null);           // Null value allowed

        System.out.println("Map: " + map);
        System.out.println("Value for key 103: " + map.get(103));
        System.out.println("All keys: " + map.keySet());
        System.out.println("All values: " + map.values());

        map.remove(102);
        System.out.println("After removing key 102: " + map);
    }
}

```

```
Map: {101=Updated Java, 102=Spring, 103=Boot, null=NullKey, 104=null}
Value for key 103: Boot
All keys: [101, 102, 103, null, 104]
All values: [Updated Java, Spring, Boot, NullKey, null]
After removing key 102: {101=Updated Java, 103=Boot, null=NullKey, 104=null}

```

TreeMap :

📌 What is `TreeMap`?

- `TreeMap` is part of the **Java Collections Framework**.
- Implements `NavigableMap`, `SortedMap`, and `Map` interfaces.
- Stores **key-value pairs**, like `HashMap`, but in **sorted order of keys**.
- Uses **natural ordering** or a **custom** `Comparator`.
- **Does not allow** `null` **keys**, but allows **multiple** `null` **values**.
- not thread safe

⚙️ Internal Working of `TreeMap`

- Uses a **Red-Black Tree**, which is a self-balancing **binary search tree**.
- Keys are stored in **sorted order**, based on:
  - Their **natural ordering** (must implement `Comparable`), or
  - A provided `Comparator` at map creation
- All operations like `put`, `get`, and `remove` maintain **logarithmic time** complexity (`O(log n)`).

```java
import java.util.TreeMap;

public class TreeMapExample {
    public static void main(String[] args) {
        TreeMap<Integer, String> map = new TreeMap<>();

        map.put(3, "Boot");
        map.put(1, "Java");
        map.put(2, "Spring");
        map.put(4, null); // null value allowed
        // map.put(null, "NullKey"); // ❌ NullPointerException

        System.out.println("Sorted Map: " + map);
        System.out.println("First key: " + map.firstKey());
        System.out.println("Last key: " + map.lastKey());
        System.out.println("Ceiling of 2: " + map.ceilingKey(2));
        System.out.println("Higher than 2: " + map.higherKey(2));

        map.remove(3);
        System.out.println("After removing key 3: " + map);
    }
}

```

```
Sorted Map: {1=Java, 2=Spring, 3=Boot, 4=null}
First key: 1
Last key: 4
Ceiling of 2: 2
Higher than 2: 3
After removing key 3: {1=Java, 2=Spring, 4=null}

```

HashMap, LinkedHashMap and TreeMap -

Feature	HashMap	LinkedHashMap	TreeMap
Maintains order	❌ NO	✅ Insertion order	✅ Sorted by keys
Allows null key	✅ One	✅ One	❌ No
Allows null value	✅ Yes	✅ Yes	✅ Yes
Backing structure	HashTable	HashTable + List	Red-black tree
Use case	Fast lookup	Ordered lookup	Sorted lookup

Document by ***Suyash*** 😇