**What is Hashtable in Java?**

* A **legacy class** (introduced in JDK 1.0, long before HashMap).
* Stores data in **key-value pairs** like HashMap.
* Part of java.util package.
* Implements **Map** interface (since JDK 1.2).
* **Thread-safe** → All its methods are synchronized.
* **Nulls are not allowed**:
  + ❌ Null key → Not allowed.
  + ❌ Null values → Not allowed.
* **1. Time Complexity**
* A **Hash Table** stores key-value pairs and uses **hashing** to map keys into buckets.

| **Operation** | **Average Case** | **Worst Case** |
| --- | --- | --- |
| **Create (Insert / put())** | **O(1)** | **O(n)** (when many collisions, all keys go in one bucket → linked list / tree traversal) |
| **Read (Get / get())** | **O(1)** | **O(n)** |
| **Update (Replace value for existing key)** | **O(1)** | **O(n)** |
| **Delete (Remove / remove())** | **O(1)** | **O(n)** |

**What is HashMap?**

* A **key–value pair** based collection class.
* Belongs to java.util package.
* Keys are **unique**, values can be **duplicate**.
* Uses **hashing** internally (like buckets + linked list / tree).

**🔹 Properties of HashMap**

1. **Null key** → Allows **one null key**.
2. **Null values** → Allows **multiple null values**.
3. **Not synchronized** (unlike Hashtable).
4. **Does not maintain insertion order**.
5. **Faster than Hashtable** (because it’s unsynchronized).

**What is TreeMap?**

* A **Map implementation** based on **Red-Black Tree** (a self-balancing binary search tree).
* Belongs to java.util package.
* Stores **key-value pairs** like HashMap, but with an important difference:

👉 **It maintains keys in sorted (ascending) order by default.**

**🔹 Properties of TreeMap**

1. **Sorted order** → Keys are always in natural order (Comparable) or by a custom Comparator.
2. **Null key** → ❌ **Not allowed** (throws NullPointerException).
3. **Null values** → ✅ Allowed.
4. **Not synchronized**.
5. **Backed by a Red-Black tree** → gives log(n) operations.

**🔹 Time Complexity of TreeMap**

* **Insert (put)** → O(log n)
* **Search (get)** → O(log n)
* **Update (put with existing key)** → O(log n)
* **Delete (remove)** → O(log n)
* **Space Complexity** → O(n)

👉 Slower than HashMap (O(1)), but TreeMap keeps the keys **sorted**.

**HashSet**

* **Internal structure:** Backed by a **HashMap** (uses hashing).
* **Order:** **Unordered** → does not guarantee insertion order or sorting.
* **Performance:**
  + add(), remove(), contains() → **O(1)** average, **O(n)** worst (or **O(log n)** in Java 8+ when treeified).
* **Nulls allowed?** Yes, at most one null.

👉 Example:

Set<Integer> hashSet = new HashSet<>();

hashSet.add(30);

hashSet.add(10);

hashSet.add(20);

System.out.println(hashSet); // Output: [20, 10, 30] (unordered)

**🔹 2. TreeSet**

* **Internal structure:** Based on a **Red-Black Tree** (self-balancing BST).
* **Order:** Maintains elements in **sorted (ascending) order** by default.
* **Performance:**
  + add(), remove(), contains() → **O(log n)**.
* **Nulls allowed?** No (throws NullPointerException).

👉 Example:

Set<Integer> treeSet = new TreeSet<>();

treeSet.add(30);

treeSet.add(10);

treeSet.add(20);

System.out.println(treeSet); // Output: [10, 20, 30] (sorted)

**🔹 3. LinkedHashSet**

* **Internal structure:** Combination of **HashSet + LinkedList**.
* **Order:** Maintains **insertion order**.
* **Performance:**
  + add(), remove(), contains() → **O(1)** average.
* **Nulls allowed?** Yes, one null.