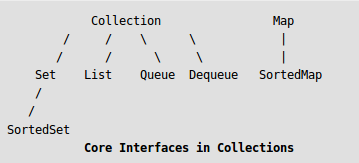
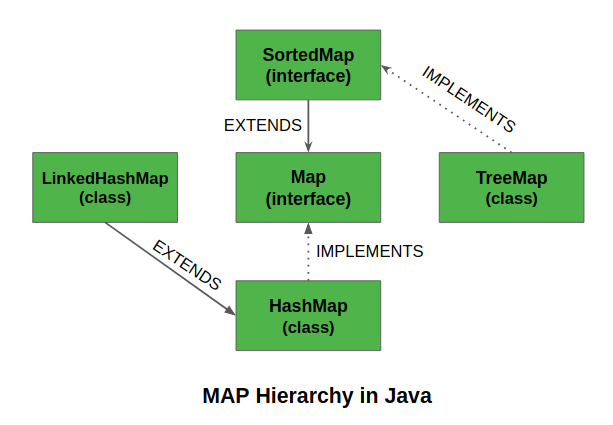
Map Interface in Java

The java.util.Map interface represents a mapping between a key and a value. The Map interface is not a subtype of the [Collection](https://www.geeksforgeeks.org/collections-in-java-2/)interface. Therefore it behaves a bit different from the rest of the collection types.  
  
Few characteristics of the Map Interface are:

1. A Map cannot contain duplicate keys and each key can map to at most one value. Some implementations allow null key and null value like the [HashMap](https://www.geeksforgeeks.org/java-util-hashmap-in-java/)and [LinkedHashMap](https://www.geeksforgeeks.org/linkedhashmap-class-java-examples), but some do not like the [TreeMap](https://www.geeksforgeeks.org/treemap-in-java/).
2. The order of a map depends on specific implementations, e.g [TreeMap](https://www.geeksforgeeks.org/treemap-in-java/)and [LinkedHashMap](https://www.geeksforgeeks.org/linkedhashmap-class-java-examples)have predictable order, while [HashMap](https://www.geeksforgeeks.org/java-util-hashmap-in-java/)does not.
3. There are two interfaces for implementing Map in java: Map and [SortedMap](https://www.geeksforgeeks.org/sortedmap-java-examples/), and three classes: [HashMap](https://www.geeksforgeeks.org/java-util-hashmap-in-java/), [TreeMap](https://www.geeksforgeeks.org/treemap-in-java/) and [LinkedHashMap](https://www.geeksforgeeks.org/linkedhashmap-class-java-examples/).



**Why and When to use Maps?**  
Maps are perfect to use for key-value association mapping such as dictionaries. The maps are used to perform lookups by keys or when someone wants to retrieve and update elements by keys. Some examples are:

* A map of error codes and their descriptions.
* A map of zip codes and cities.
* A map of managers and employees. Each manager (key) is associated with a list of employees (value) he manages.
* A map of classes and students. Each class (key) is associated with a list of students (value).

**Methods in Map Interface:**

1. public Object put(Object key, Object value): This method is used to insert an entry in this map.
2. public void putAll(Map map): This method is used to insert the specified map in this map.
3. public Object remove(Object key): This method is used to delete an entry for the specified key.
4. public Object get(Object key):This method is used to return the value for the specified key.
5. public boolean containsKey(Object key): This method is used to search the specified key from this map.
6. public Set keySet(): This method is used to return the Set view containing all the keys.
7. public Set entrySet(): This method is used to return the Set view containing all the keys and values.

Java - The TreeMap Class

Advertisements

[Previous Page](https://www.tutorialspoint.com/java/java_collections.htm)

[Next Page](https://www.tutorialspoint.com/java/java_collections.htm)

The TreeMap class implements the Map interface by using a tree. A TreeMap provides an efficient means of storing key/value pairs in sorted order, and allows rapid retrieval.

You should note that, unlike a hash map, a tree map guarantees that its elements will be sorted in an ascending key order.

Following is the list of the constructors supported by the TreeMap class.

|  |  |
| --- | --- |
| **Sr.No.** | **Constructors & Description** |
| 1 | **TreeMap( )**  This constructor constructs an empty tree map that will be sorted using the natural order of its keys. |
| 2 | **TreeMap(Comparator comp)**  This constructor constructs an empty tree-based map that will be sorted using the Comparator comp. |
| 3 | **TreeMap(Map m)**  This constructor initializes a tree map with the entries from **m**, which will be sorted using the natural order of the keys. |
| 4 | **TreeMap(SortedMap sm)**  This constructor initializes a tree map with the entries from the SortedMap **sm**, which will be sorted in the same order as **sm**. |

Apart from the methods inherited from its parent classes, TreeMap defines the following methods −

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | **void clear()**  Removes all mappings from this TreeMap. |
| 2 | **Object clone()**  Returns a shallow copy of this TreeMap instance. |
| 3 | **Comparator comparator()**  Returns the comparator used to order this map, or null if this map uses its keys' natural order. |
| 4 | **boolean containsKey(Object key)**  Returns true if this map contains a mapping for the specified key. |
| 5 | **boolean containsValue(Object value)**  Returns true if this map maps one or more keys to the specified value. |
| 6 | **Set entrySet()**  Returns a set view of the mappings contained in this map. |
| 7 | **Object firstKey()**  Returns the first (lowest) key currently in this sorted map. |
| 8 | **Object get(Object key)**  Returns the value to which this map maps the specified key. |
| 9 | **SortedMap headMap(Object toKey)**  Returns a view of the portion of this map whose keys are strictly less than toKey. |
| 10 | **Set keySet()**  Returns a Set view of the keys contained in this map. |
| 11 | **Object lastKey()**  Returns the last (highest) key currently in this sorted map. |
| 12 | **Object put(Object key, Object value)**  Associates the specified value with the specified key in this map. |
| 13 | **void putAll(Map map)**  Copies all of the mappings from the specified map to this map. |
| 14 | **Object remove(Object key)**  Removes the mapping for this key from this TreeMap if present. |
| 15 | **int size()**  Returns the number of key-value mappings in this map. |
| 16 | **SortedMap subMap(Object fromKey, Object toKey)**  Returns a view of the portion of this map whose keys range from fromKey, inclusive, to toKey, exclusive. |
| 17 | **SortedMap tailMap(Object fromKey)**  Returns a view of the portion of this map whose keys are greater than or equal to fromKey. |
| 18 | **Collection values()**  Returns a collection view of the values contained in this map. |

Java - The LinkedHashMap Class

Advertisements

[Previous Page](https://www.tutorialspoint.com/java/java_collections.htm)

[Next Page](https://www.tutorialspoint.com/java/java_collections.htm)

This class extends HashMap and maintains a linked list of the entries in the map, in the order in which they were inserted. This allows insertion-order iteration over the map. That is, when iterating a LinkedHashMap, the elements will be returned in the order in which they were inserted.

You can also create a LinkedHashMap that returns its elements in the order in which they were last accessed.

Following is the list of constructors supported by the LinkedHashMap class.

|  |  |
| --- | --- |
| **Sr.No.** | **Constructor & Description** |
| 1 | **LinkedHashMap( )**  This constructor constructs a default LinkedHashMap. |
| 2 | **LinkedHashMap(Map m)**  This constructor initializes the LinkedHashMap with the elements from the given Map class **m**. |
| 3 | **LinkedHashMap(int capacity)**  This constructor initializes a LinkedHashMap with the given capacity. |
| 4 | **LinkedHashMap(int capacity, float fillRatio)**  This constructor initializes both the capacity and the fill ratio. The meaning of capacity and fill ratio are the same as for HashMap. |
| 5 | **LinkedHashMap(int capacity, float fillRatio, boolean Order)**  This constructor allows you to specify whether the elements will be stored in the linked list by insertion order, or by order of last access. If Order is true, then access order is used. If Order is false, then insertion order is used. |

Apart from the methods inherited from its parent classes, LinkedHashMap defines the following methods −

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | **void clear()**  Removes all mappings from this map. |
| 2 | **boolean containsKey(Object key)**  Returns true if this map maps one or more keys to the specified value. |
| 3 | **Object get(Object key)**  Returns the value to which this map maps the specified key. |
| 4 | **protected boolean removeEldestEntry(Map.Entry eldest)**  Returns true if this map should remove its eldest entry. |

WeakHashMap is an implementation of the Map interface that stores only weak references to its keys. Storing only weak references allows a key-value pair to be garbage-collected when its key is no longer referenced outside of the WeakHashMap.

This class provides the easiest way to harness the power of weak references. It is useful for implementing "registry-like" data structures, where the utility of an entry vanishes when its key is no longer reachable by any thread.

The WeakHashMap functions identically to the HashMap with one very important exception: if the Java memory manager no longer has a strong reference to the object specified as a key, then the entry in the map will be removed.

**Weak Reference** − If the only references to an object are weak references, the garbage collector can reclaim the object's memory at any time.it doesn't have to wait until the system runs out of memory. Usually, it will be freed the next time the garbage collector runs.

Following is the list of constructors supported by the WeakHashMap class.

|  |  |
| --- | --- |
| **Sr.No.** | **Constructor & Description** |
| 1 | **WeakHashMap()**  This constructor constructs a new, empty WeakHashMap with the default initial capacity (16) and the default load factor (0.75). |
| 2 | **WeakHashMap(int initialCapacity)**  This constructor constructs a new, empty WeakHashMap with the given initial capacity and the default load factor, which is 0.75. |
| 3 | **WeakHashMap(int initialCapacity, float loadFactor)**  This constructor constructs a new, empty WeakHashMap with the given initial capacity and the given load factor. |
| 4 | **WeakHashMap(Map t)**  This constructor constructs a new WeakHashMap with the same mappings as the specified Map. |

Apart from the methods inherited from its parent classes, TreeMap defines the following methods −

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | **void clear()**  Removes all mappings from this map. |
| 2 | **boolean containsKey(Object key)**  Returns true if this map contains a mapping for the specified key. |
| 3 | **boolean containsValue(Object value)**  Returns true if this map maps one or more keys to the specified value. |
| 4 | **Set entrySet()**  Returns a collection view of the mappings contained in this map. |
| 5 | **Object get(Object key)**  Returns the value to which the specified key is mapped in this weak hash map, or null if the map contains no mapping for this key. |
| 6 | **boolean isEmpty()**  Returns true if this map contains no key-value mappings. |
| 7 | **Set keySet()**  Returns a set view of the keys contained in this map. |
| 8 | **Object put(Object key, Object value)**  Associates the specified value with the specified key in this map. |
| 9 | **void putAll(Map m)**  Copies all of the mappings from the specified map to this map. These mappings will replace any mappings that this map had for any of the keys currently in the specified map. |
| 10 | **Object remove(Object key)**  Removes the mapping for this key from this map if present. |
| 11 | **int size()**  Returns the number of key-value mappings in this map. |
| 12 | **Collection values()**  Returns a collection view of the values contained in this map. |