Java LinkedHashMap

Last Updated : 17 Dec, 2024

LinkedHashMap in Java implements the Map interface of the Collections Framework. It stores key-value pairs while maintaining the insertion order of the entries. It maintains the order in which elements are added.

- Stores unique key-value pairs.
- Maintains insertion order.
- Allows one null key and multiple null values.
- Fast performance for basic operations.

Example:

```java
import java.util.LinkedHashMap;

public class Geeks {
    public static void main(String[] args) {

        // Create a LinkedHashMap of
        // Strings (keys) and Integers (values)
        LinkedHashMap<String, Integer> lhm = new LinkedHashMap<>();

        // Displaying the LinkedHashMap
        System.out.println("" + lhm);
    }
}
```

Output

```
{}
```

Declaration of LinkedHashMap

public class LinkedHashMap<K,V> extends HashMap<K,V> implements Map<K,V>

Here, **K** is the key Object type and **V** is the value Object type

- **K:** The type of the keys in the map.
- **V:** The type of values mapped in the map.

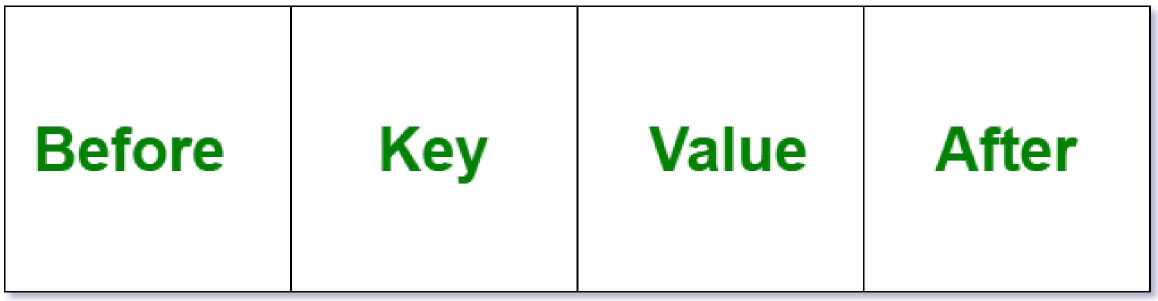
The **LinkedHashMap Class** is just like HashMap with an additional feature of maintaining an order of elements inserted into it. HashMap provided the advantage of quick insertion, search, and deletion but it never maintained the track and order of insertion, which the LinkedHashMap provides where the elements can be accessed in their insertion order.

Internal Working of LinkedHashMap

A **LinkedHashMap** is an extension of the **HashMap** class and it implements the **Map** interface. Therefore, the class is declared as:

public class LinkedHashMap
  extends HashMap
    implements Map

In this class, the data is stored in the form of nodes. The implementation of the LinkedHashMap is very similar to a doubly-linked list. Therefore, each node of the LinkedHashMap is represented as:



- **Key:** Since this class extends HashMap, the data is stored in the form of a key-value pair. Therefore, this parameter is the key to the data.

- **Value:** For every key, there is a value associated with it. This parameter stores the value of the keys. Due to generics, this value can be of any form.
- **Next:** Since the LinkedHashMap stores the insertion order, this contains the address to the next node of the LinkedHashMap.
- **Previous:** This parameter contains the address to the previous node of the LinkedHashMap.

Synchronized LinkedHashMap

The **LinkedHashMap** class is not synchronized. If it is used in a multi-threaded environment where structural modifications like adding or removing elements are made concurrently then external synchronization is needed. This can be done by wrapping the map using Collections.synchronizedMap() method.

Map<K, V> synchronizedMap = Collections.synchronizedMap(new LinkedHashMap<>());

Constructors of LinkedHashMap Class

LinkedHashMap class provides various constructors for different use cases:

1. LinkedHashMap(): This is used to construct a default LinkedHashMap constructor.

LinkedHashMap<K, V> lhm = new LinkedHashMap<>();

2. LinkedHashMap(int capacity): It is used to initialize a particular LinkedHashMap with a specified capacity.

LinkedHashMap<K, V> lhm = new LinkedHashMap<>(int capacity);

3. LinkedHashMap(Map<? extends **K**,? extends **V**> map**):** It is used to initialize a particular LinkedHashMap with the elements of the specified map.

> *LinkedHashMap<K, V> lhm = new LinkedHashMap<K, V>(Map<? extends K,? extends V> map);*

4. LinkedHashMap(int capacity, float fillRatio): It is used to initialize both the capacity and fill ratio for a LinkedHashMap. A fillRatio also called as **loadFactor** is a metric that determines when to increase the size of the LinkedHashMap automatically. By default, this value is 0.75 which means that the size of the map is increased when the map is 75% full.

> *LinkedHashMap<K, V> lhm = new LinkedHashMap<K, V>(int capacity, float fillRatio);*

5. LinkedHashMap(int capacity, float fillRatio, boolean Order): This constructor is also used to initialize both the capacity and fill ratio for a LinkedHashMap along with whether to follow the insertion order or not.

> *LinkedHashMap<K, V> lhm = new LinkedHashMap<K, V>(int capacity, float fillRatio, boolean Order);*

Here, for the ***Order attribute***, true is passed for the last access order and false is passed for the insertion order.

Methods of LinkedHashMap

Below are some commonly used methods of the LinkedHashMap class:

Method	Description
containsValue(Object value)	Returns true if this map maps one or more keys to the specified value.
entrySet()	Returns a Set view of the mappings contained in this map.

Method	Description
get(Object key)	Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.
keySet()	Returns a Set view of the keys contained in this map.
removeEldestEntry (Map.Entry<K,V> eldest)	Returns true if this map should remove its eldest entry.
values()	Returns a Collection view of the values contained in this map.

Application: *Since the LinkedHashMap makes use of Doubly LinkedList to maintain the insertion order, we can implement LRU Cache functionality by overriding the removeEldestEntry() method to impose a policy for automatically removing stale when new mappings are added to the map. This lets you expire data using some criteria that you define.*

Performing Various Operations on LinkedHashMap

Let's see how to perform a few frequently used operations on the LinkedHashMap class instance.

1. Adding Elements in LinkedHashMap

In order to add an element to the LinkedHashMap, we can use the put() method. This is different from HashMap because in HashMap, the insertion order is not retained but it is retained in the LinkedHashMap.

Example:

```java
// Adding Elements to a LinkedHashMap
import java.util.*;

class Geeks {
    public static void main(String args[]) {

        // Initialization of a LinkedHashMap
        // using Generics
        LinkedHashMap<Integer, String> lhm
            = new LinkedHashMap<Integer, String>();

        // Add mappings to Map
        // using put() method
        lhm.put(3, "Geeks");
        lhm.put(2, "For");
        lhm.put(1, "Geeks");

        // Printing mappings to the console
        System.out.println(""
                           + lhm);
    }
}
```

Output

```
{3=Geeks, 2=For, 1=Geeks}
```

2. Updating Elements in LinkedHashMap

After adding elements if we wish to change the element, it can be done by again adding the element using the put() method. Since the elements in the LinkedHashMap are indexed using the keys, the value of the key can be changed by simply re-inserting the updated value for the key for which we wish to change.

Example:

```
// Updation of Elements of LinkedHashMap
import java.util.*;

class Geeks {

    public static void main(String args[]) {

        // Initialization of a LinkedHashMap
        // using Generics
        LinkedHashMap<Integer, String> lhm
            = new LinkedHashMap<Integer, String>();

        // Inserting mappings into Map
        // using put() method
        lhm.put(3, "Geeks");
        lhm.put(2, "Geeks");
        lhm.put(1, "Geeks");

        // Printing mappings to the console
        System.out.println("" + lhm);

        // Updating the value with key 2
        lhm.put(2, "For");

        // Printing the updated Map
        System.out.println("Updated Map: " + lhm);
    }
}
```

Output

```
{3=Geeks, 2=Geeks, 1=Geeks}
Updated Map: {3=Geeks, 2=For, 1=Geeks}
```

3. Removing Element in LinkedHashMap

In order to remove an element from the LinkedHashMap, we can use the remove() method. This method takes the value of key as input, searches for existence of such key and then removes the mapping for the key from this LinkedHashMap if it is present in the map. Apart from that, we can also remove the first entered element from the map if the maximum size is defined.

Example:

```
// Removal of Elements from LinkedHashMap
import java.util.*;

class Geeks {

    public static void main(String args[]) {

        // Initialization of a LinkedHashMap
        // using Generics
        LinkedHashMap<Integer, String> lhm
            = new LinkedHashMap<Integer, String>();

        // Inserting the Elements
        // using put() method
        lhm.put(3, "Geeks");
        lhm.put(2, "Geeks");
        lhm.put(1, "Geeks");
        lhm.put(4, "For");
        // Printing the mappings to the console
        System.out.println("" + lhm);

        // Removing the mapping with Key 4
        lhm.remove(4);

        // Printing the updated map
        System.out.println("" + lhm);
    }
}
```

Output

```
{3=Geeks, 2=Geeks, 1=Geeks, 4=For}
{3=Geeks, 2=Geeks, 1=Geeks}
```

4. Iterating through the LinkedHashMap

There are multiple ways to iterate through the LinkedHashMap. The most famous way is to use a for-each loop over the set view of the map (fetched using map.entrySet() instance method). Then for each entry (set element) the values of key and value can be fetched usingthe getKey() and the *getValue()* method.

Example:

```
// Iterating over LinkedHashMap
import java.util.*;

class Geeks {

    public static void main(String args[]) {

        // Initialization of a LinkedHashMap
        // using Generics
        LinkedHashMap<Integer, String> lhm
            = new LinkedHashMap<Integer, String>();

        // Inserting elements into Map
        // using put() method
        lhm.put(3, "Geeks");
        lhm.put(2, "For");
        lhm.put(1, "Geeks");

        // For-each loop for traversal over Map
        for (Map.Entry<Integer, String> mapElement :
             lhm.entrySet()) {

            Integer k = mapElement.getKey();
            // Finding the value
            // using getValue() method
            String v = mapElement.getValue();

            // Printing the key-value pairs
            System.out.println(k + " : " + v);
        }
    }
}
```

Output

```
3 : Geeks
2 : For
1 : Geeks
```

Advantages of LinkedHashMap

- It maintains insertion order.
- Faster iteration.
- Allows null values.

Disadvantages of LinkedHashMap

- Higher memory usage.
- Slower insertion.
- Less efficient for large datasets.

Comment More info

Placement Training Program

Next Article

Hashtable in Java