LinkedHashMap in Java

Difficulty Level : Easy Last Updated : 06 Dec, 2021

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.

Important Features of a LinkedHashMap are listed as follows:

- A LinkedHashMap contains values based on the key. It implements the Map interface and extends the HashMap class.
- It contains only unique elements.
- It may have one null key and multiple null values.
- It is non-synchronized.
- It is the same as HashMap with an additional feature that it maintains insertion order. For example, when we run the code with a HashMap, we get a different order of elements.

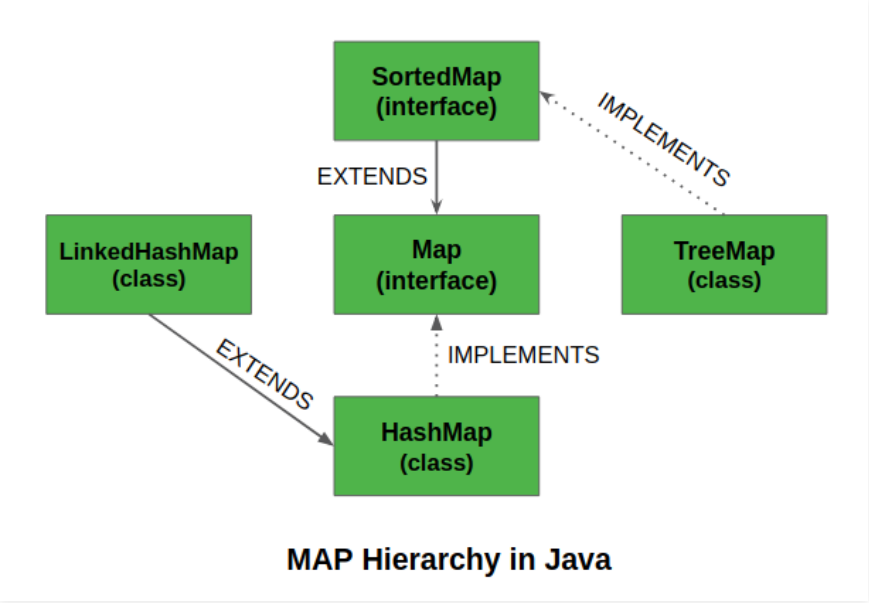
Declaration:

```
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.

It implements Map<K, V> interface, and extends HashMap<K, V> class. Though the Hierarchy of LinkedHashMap is as depicted in below media as follows**:**



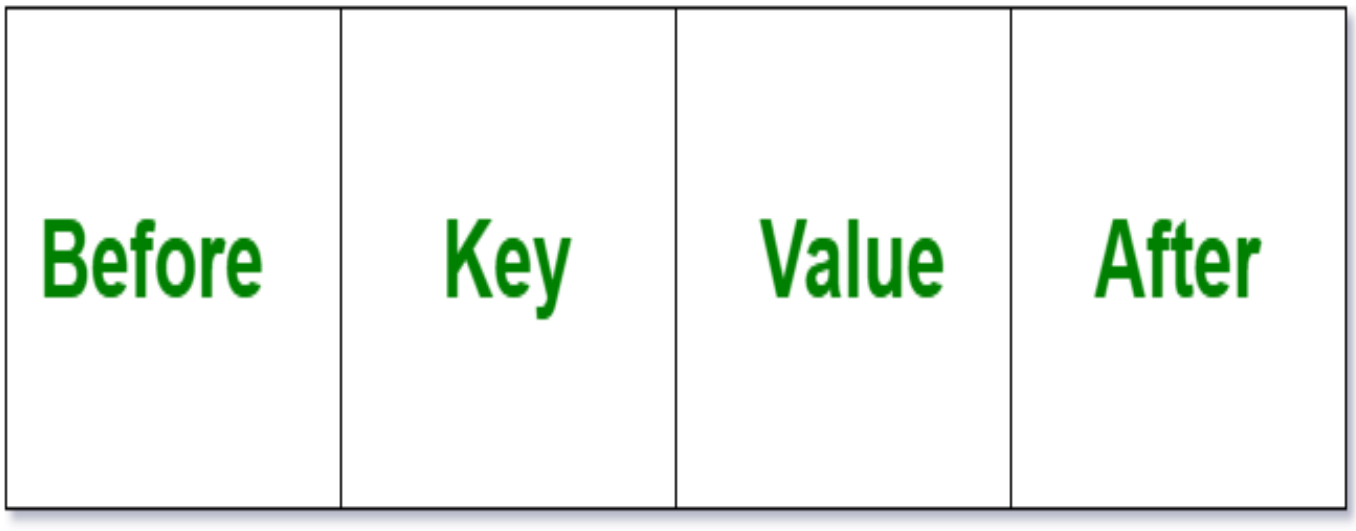
MAP Hierarchy in Java

How LinkedHashMap Work Internally?

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:



- **Hash:** All the input keys are converted into a hash which is a shorter form of the key so

that the search and insertion are faster.

- **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 implementation of LinkedHashMap not synchronized. If multiple threads access a linked hash map concurrently, and at least one of the threads modifies the map structurally, it must be synchronized externally. This is typically accomplished by synchronizing on some object that naturally encapsulates the map. If no such object exists, the map should be “wrapped” using the **Collections.synchronizedMap** method. This is best done at creation time, to prevent accidental unsynchronized access to the map:

```
Map m = Collections.synchronizedMap(new LinkedHashMap(...));
```

Constructors of LinkedHashMap Class

In order to create a **LinkedHashMap**, we need to create an object of the LinkedHashMap class. The LinkedHashMap class consists of various constructors that allow the possible creation of the ArrayList. The following are the constructors available in this class:

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

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

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

```
LinkedHashMap<K, V> lhm = new LinkedHashMap<K, V>(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>
```

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 fillRati
```

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 fillRati
```

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

Methods of LinkedHashMap

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.
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.

METHOD	DESCRIPTION
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.*

Example:

```
// Java Program to Demonstrate Working of LinkedHashMap

// Importing required classes
import java.util.*;

// LinkedHashMapExample
public class GFG {

    // Main driver method
    public static void main(String a[])
    {

        // Creating an empty LinkedHashMap
        LinkedHashMap<String, String> lhm
            = new LinkedHashMap<String, String>();

        // Adding entries in Map
        // using put() method
        lhm.put("one", "practice.geeksforgeeks.org");
        lhm.put("two", "code.geeksforgeeks.org");
        lhm.put("four", "quiz.geeksforgeeks.org");

        // Printing all entries inside Map
        System.out.println(lhm);
```

```
        // Note: It prints the elements in same order
        // as they were inserted

        // Getting and printing value for a specic key
        System.out.println("Getting value for key 'one': "
                           + lhm.get("one"));

        // Getting size of Map using size() method
        System.out.println("Size of the map: "
                           + lhm.size());

        // Checking whether Map is empty or not
        System.out.println("Is map empty? "
                           + lhm.isEmpty());

        // Using containsKey() method to check for a key
        System.out.println("Contains key 'two'? "
                           + lhm.containsKey("two"));

        // Using containsKey() method to check for a value
        System.out.println(
            "Contains value 'practice.geeks"
            + "forgeeks.org'? "
            + lhm.containsValue("practice"
                                + ".geeksforgeeks.org"));

        // Removing entry using remove() method
        System.out.println("delete element 'one': "
                           + lhm.remove("one"));

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

Output

```
{one=practice.geeksforgeeks.org, two=code.geeksforgeeks.org, four=quiz.geeksfo
Getting value for key 'one': practice.geeksforgeeks.org
Size of the map: 3
Is map empty? false
Contains key 'two'? true
Contains value 'practice.geeksforgeeks.org'? true
delete element 'one': practice.geeksforgeeks.org
Mappings of LinkedHashMap : {two=code.geeksforgeeks.org, four=quiz.geeksforgee
```

Various operations on the HashMap class

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

Operation 1: Adding Elements

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 Program to Demonstrate Adding
// Elements to a LinkedHashMap

// Importing required classes
import java.util.*;

// Main class
// AddElementsToLinkedHashMap
class GFG {

    // Main driver method
    public static void main(String args[])
    {

        // Initialization of a LinkedHashMap
        // using Generics
        LinkedHashMap<Integer, String> hm1
            = new LinkedHashMap<Integer, String>();

        // Add mappings to Map
        // using put() method
        hm1.put(3, "Geeks");
        hm1.put(2, "For");
        hm1.put(1, "Geeks");

        // Printing mappings to the console
        System.out.println("Mappings of LinkedHashMap : "
                           + hm1);
    }
}
```

Output

```
Mappings of LinkedHashMap : {3=Geeks, 2=For, 1=Geeks}
```

Operation 2: Changing Elements

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

Example

```
// Java Program to Demonstrate Updation of Elements
// of LinkedHashMap

import java.util.*;

// Main class
// UpdatingLinkedHashMap
class GFG {

    // Main driver method
    public static void main(String args[])
    {

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

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

        // Printing mappings to the console
        System.out.println("Initial map : " + hm);

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

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

Output

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

Operation 3: Removing Element

In order to remove an element from the TreeMap, we can use the remove() method. This method takes the key value and removes the mapping for the key from this treemap 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

```
// Java program to Demonstrate Removal of Elements
// from LinkedHashMap

// Importing utility classes
import java.util.*;

// Main class
// RemovingMappingsFromLinkedHashMap
class GFG {

    // Main driver method
    public static void main(String args[])
    {
        // Initialization of a LinkedHashMap
        // using Generics
        LinkedHashMap<Integer, String> hm
            = new LinkedHashMap<Integer, String>();

        // Inserting the Elements
        // using put() method
        hm.put(3, "Geeks");
        hm.put(2, "Geeks");
        hm.put(1, "Geeks");
        hm.put(4, "For");

        // Printing the mappings to the console
        System.out.println("Initial Map : " + hm);
```

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

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

Output

```
Initial Map : {3=Geeks, 2=Geeks, 1=Geeks, 4=For}
Updated Map : {3=Geeks, 2=Geeks, 1=Geeks}
```

Operation 4: Iterating through the LinkedHashMap

There are multiple ways to iterate through the Map. The most famous way is to use a for-each loop and get the keys. The value of the key is found by using the *getValue()* method.

Example

```
// Java program to demonstrate
// Iterating over LinkedHashMap

// Importing required classes
import java.util.*;

// Main class
// IteratingOverLinkedHashMap
class GFG {

    // Main driver method
    public static void main(String args[])
    {

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

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

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

            Integer key = mapElement.getKey();

            // Finding the value
            // using getValue() method
            String value = mapElement.getValue();

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

Output

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

Related Articles:

- LRU Cache Implementation
- Differences between TreeMap, HashMap, and LinkedHashMap in Java