CSCI 2120:
Software Design & Development II

UNIT 2: Collections Framework & Generics

LinkedHashMap

Overview

1. Introduction
2. Hierarchy of LinkedHashMap class
3. LinkedHashMap class Declaration
4. Features of LinkedHashMap
5. Constructors of LinkedHashMap class
6. LinkedHashMap Methods
7. LinkedHashMap Examples
8. Use of LinkedHashMap
9. HashMap vs LinkedHashMap

Introduction

LinkedHashMap in Java is a concrete class that is HashTable and LinkedList implementation of Map interface. It stores entries using a doubly-linked list.

Java LinkedHashMap class extends the HashMap class with a linked-list implementation that supports an ordering of the entries in the map.

It is the same as HashMap (including constructors and methods) except for the following differences:

1. The underlying data structure of HashMap is HashTable whereas, the underlying data structure of LinkedHashMap is HashTable and LinkedList (Hybrid data structure).

2. Insertion order is not preserved in the HashMap because it is based on the hashCode of Key. But in the case of LinkedHashMap, the insertion order of elements is preserved because it is based on the Key insertion order, that is, the order in which keys are inserted in the map.

Hierarchy of LinkedHashMap class

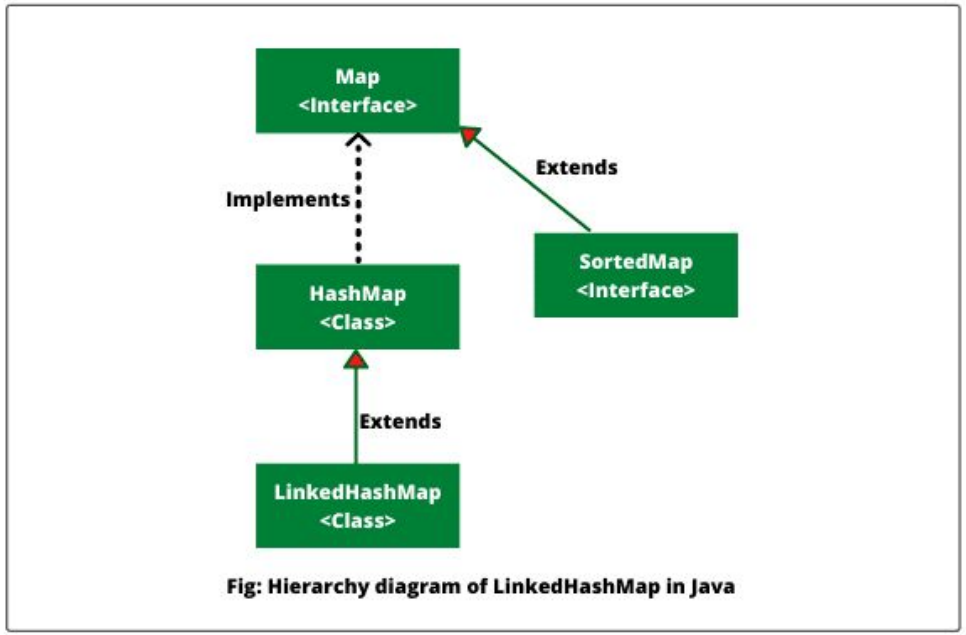

Fig: Hierarchy diagram of LinkedHashMap in Java

`LinkedHashMap` implementation in Java is a subclass of `HashMap` class. That is, `LinkedHashMap` class extends the `HashMap` class and implements `Map` interface.

The hierarchy diagram of `LinkedHashMap` is shown in the figure.

`LinkedHashMap` class Declaration

`LinkedHashMap` is a generic class that is present in `java.util.LinkedHashMap` package. It has the following declaration.

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

Here, K defines the type of keys, and V defines the type of values.

Features of LinkedHashMap

There are several features of `LinkedHashMap` class that you need to keep in mind:

1. The underlying data structure of `LinkedHashMap` is `HashTable` and `LinkedList`.

2. `LinkedHashMap` maintains the insertion order. The entries in Java `LinkedHashMap` can be retrieved either in the order in which they were inserted into the map (known as insertion order) or in the order in which they were last accessed, from least to most recently accessed.

3. `LinkedHashMap` contains unique elements. It contains values based on keys.

4. `LinkedHashMap` allows only one null key but can have multiple `null` values.

5. `LinkedHashMap` in Java is non synchronized. That is, multiple threads can access the same `LinkedHashMap` object simultaneously.

6. The default initial capacity of `LinkedHashMap` class is `16` with a load factor of `0.75`.

Constructors of LinkedHashMap class

HashMap class in Java provides four constructors that are as follows on following slides.

Note: *The first four constructors of linkedhashmap are analogous to the four constructors of hashmap class. In each case, the created linkedhashmap maintains the insertion order.*

The last constructor of linkedhashmap allows us to specify whether elements will be stored in the linkedlist by insertion order, or by order of last access.

Constructors of LinkedHashMap class

1. LinkedHashMap(): This constructor is used to create a default LinkedHashMap object. It constructs an empty insertion-ordered LinkedHashMap object with the default initial capacity 16 and load factor 0.75.

The general syntax to construct default LinkedHashMap object is as follows:

```
LinkedHashMap lhmap = new LinkedHashMap();


// Generic form
LinkedHashMap<K,V> lhmap = new LinkedHashMap<>();
```

Constructors of LinkedHashMap class

2. LinkedHashMap(int initialCapacity): It is used to create an empty insertion-ordered LinkedHashMap object with the specified initial capacity and a default load factor of 0.75.

The general syntax in generic form is as follows:

```
LinkedHashMap<K,V> lhmap = new LinkedHashMap<K,V>(int initialCapacity);
```

Constructors of LinkedHashMap class

3. LinkedHashMap(int initialCapacity, float loadFactor): It is used to create an empty insertion-ordered LinkedHashMap object with the specified initial capacity and load factor.

The general syntax in generic form is given below:

```
LinkedHashMap<K,V> lhmap = new LinkedHashMap<>(int initialCapacity, float loadFactor);

//For example:
LinkedHashMap<String, Integer> lhmap = new LinkedHashMap<>(16, 0.75f);
```

Constructors of LinkedHashMap class

4. LinkedHashMap(Map m): This constructor is used to create an insertion-ordered LinkedHashMap object with the elements from the given Map m.

The general syntax is as follows:

```
LinkedHashMap<K,V> lhmap = new LinkedHashMap<K,V>(Map m);
```

Constructors of LinkedHashMap class

5. LinkedHashMap(int initialCapacity, float loadFactor, boolean accessOrder): This constructor is used to create an empty LinkedHashMap instance with the specified initial capacity, load factor, and ordering mode.

If accessOrder is true, access-order is used. If it is false, the insertion order is used. The general syntax to create LinkedHashMap object with three arguments of constructor is as follows:

```
LinkedHashMap<K,V> lhmap = new LinkedHashMap<int capacity, float loadFactor, boolean accessOrder>();

// For example:

// For access order.
LinkedHashMap<String, String> lhmap = new LinkedHashMap<>(16, 0.75f, true);

// For insertion order
LinkedHashMap<String, String> lhmap = new LinkedHashMap<>(16, 0.75f, false);
```

LinkedHashMap Methods

The methods of LinkedHashMap are exactly the same as HashMap class methods, except for one method that is added by LinkedHashMap. This method is removeEldestEntry().

The general syntax for this method is as follows:

```
protected boolean removeEldestEntry(Map.Entry<K,V> e)
```

The parameter e is the least recently added entry in the map, or if it is an access-ordered map, the least recently accessed entry.

This method returns true if the map removes this eldest (oldest) entry. If this entry is retained, or not removed, this method returns false.

LinkedHashMap Methods

The `removeEldestEntry()` method is called by `put()` and `putAll()` after adding a new entry into the map. It helps to remove the eldest entry each time when a new entry is added.

This method is useful if the map represents a cache. It allows the map to reduce memory consumption by deleting stale entries.

LinkedHashMap Examples

Let’s take different example programs where we will perform frequently asked operations based on the methods of `LinkedHashMap`.

Example 1: Adding entries

1. Let’s create a program where we will perform various operations such as adding elements, checking the size of linkedhashmap, and linkedhashmap is empty or not before adding elements into it. Look at the program source code.

Example 1: Adding entries

```
import java.util.LinkedHashMap;
public class LinkedHashMapTester1 {
   public static void main(String[] args) {
       // Create a LinkedHashMap instance.
       LinkedHashMap<String, Integer> lhmap = new LinkedHashMap<>();
       // Checking the size of linkedhashmap before adding entries.
       int size = lhmap.size();
       System.out.println("Size of LinkedHashMap before adding entries: " +size);
       // Checking linked hash map is empty or not before adding entries.
       boolean isEmpty = lhmap.isEmpty();
       System.out.println("Is LinkedHashMap empty: " +isEmpty);
       // Adding entries in linked hash map.
       lhmap.put("John", 30);
       lhmap.put("Peter", 25);
       lhmap.put("Ricky", 23);
       lhmap.put("Deep", 28);
       lhmap.put("Mark", 32);
       System.out.println("Display entries in LinkedHashMap");
       System.out.println(lhmap);
       int size2 = lhmap.size();
       System.out.println("Size of LinkedHashMap after adding entries: " +size2);
       // Adding null as key and value.
       lhmap.put(null, null);
       System.out.println(lhmap);
   }
}
```

Example 1: Adding entries

Output:

```
Size of LinkedHashMap before adding entries: 0
Is LinkedHashMap empty: true
Display entries in LinkedHashMap
{John=30, Peter=25, Ricky=23, Deep=28, Mark=32}
Size of LinkedHashMap after adding entries: 5
{John=30, Peter=25, Ricky=23, Deep=28, Mark=32, null=null}
```

Example 2: LinkedHashMap using accessOrder

2. Let’s take a program where we will create LinkedHashMap instance with the third argument of constructor set to true. The mappings En=English, Hi=Hindi, Ta=Tamil, De=German, Fr=French will be inserted into this object and entries will be displayed on the console. Look at the program source code to understand better.

Example 2: LinkedHashMap using accessOrder

```
import java.util.LinkedHashMap;
public class LinkedHashMapTester2 {
   public static void main(String[] args) {
       LinkedHashMap<String, String> lhmap = new LinkedHashMap<>(16, 0.75f, true);
       lhmap.put("En", "English");
       lhmap.put("Hi", "Hindi");
       lhmap.put("Ta", "Tamil");
       lhmap.put("De", "German");
       lhmap.put("Fr", "French");
       System.out.println("Initially, entries in LinkedHashMap lhmap: " +lhmap);
       System.out.println("Value corresponding to key Hi: " +lhmap.get("Hi"));
       System.out.println("Value corresponding to key En: " +lhmap.get("En"));
       System.out.println("After accessing entries Hi and En: " +lhmap);
       System.out.println("\n");

       LinkedHashMap<String, String> lhmap2 = new LinkedHashMap<>(16, 0.75f, false);
       lhmap2.put("En", "English");
       lhmap2.put("Hi", "Hindi");
       lhmap2.put("Ta", "Tamil");
       lhmap2.put("De", "German");
       lhmap2.put("Fr", "French");

       System.out.println("Initially, entries in LinkedHashMap lhmap2: " +lhmap2);
       System.out.println("Value corresponding to key Hi: " +lhmap.get("Hi"));
       System.out.println("Value corresponding to key En: " +lhmap.get("En"));
       System.out.println("After accessing entries Hi and En: " +lhmap2);
   }
}
```

Example 2: LinkedHashMap using accessOrder

Output:

```
Initially, entries in LinkedHashMap lhmap: {En=English, Hi=Hindi, Ta=Tamil, De=German, Fr=French}
Value corresponding to key Hi: Hindi
Value corresponding to key En: English
After accessing entries Hi and En: {Ta=Tamil, De=German, Fr=French, Hi=Hindi, En=English}


Initially, entries in LinkedHashMap lhmap2: {En=English, Hi=Hindi, Ta=Tamil, De=German, Fr=French}
Value corresponding to key Hi: Hindi
Value corresponding to key En: English
After accessing entries Hi and En: {En=English, Hi=Hindi, Ta=Tamil, De=German, Fr=French}
```

As you can see in the above program, the mapping Hi=Hindi is accessed, followed by the mapping En=English. After accessing entries, the mappings are displayed in the order that they were last accessed.

Another LinkedHashMap instance is created with the third argument set to false. The same steps are followed here as followed by previous steps. After accessing entries, the insertion order is maintained.

Example 3: Removing entries

3. Let’s take an example program where we will perform remove, and replace operations. We will also check that a particular value or key is present in linkedhashmap or not. Look at the following source code.

Example 3: Removing entries

```
import java.util.LinkedHashMap;
public class LinkedHashMapTester3 {
   public static void main(String[] args) {
       LinkedHashMap<String, String> lhmap = new LinkedHashMap<>();
       lhmap.put("En", "English");
       lhmap.put("Hi", "Hindi");
       lhmap.put("Ta", "Tamil");
       lhmap.put("De", "German");
       lhmap.put("Fr", "French");

       System.out.println("Entries in LinkedHashMap lhmap: " +lhmap);

       // Call remove() method to delete an entry for specified key.
       lhmap.remove("De");
       System.out.println("Updated Entries in LinkedHashMap: " +lhmap);

       // Call replace() method to replace specified value for a specified key.
       lhmap.replace("En", "English-US");
       System.out.println("After replacing, updated entries in LinkedHashMap: " +lhmap);

       // Call containsValue() method to determine specified value for specified key.
       boolean value = lhmap.containsValue("Hindi");
       System.out.println("Is Hindi present in LinkedHashMap: " +value);
   }
}
```

Example 3: Removing entries

Output:

```
Entries in LinkedHashMap lhmap: {En=English, Hi=Hindi, Ta=Tamil, De=German, Fr=French}
Updated Entries in LinkedHashMap: {En=English, Hi=Hindi, Ta=Tamil, Fr=French}
After replacing, updated entries in LinkedHashMap: {En=English-US, Hi=Hindi, Ta=Tamil, Fr=French}
Is Hindi present in LinkedHashMap: true
```

Example 4: Iterate LinkedHashMap with Iterator

4. Let’s take an example program where we will iterate entries, keys, and values of LinkedHashMap using java iterator concept. Look at the source code.

Example 4: Iterate LinkedHashMap with Iterator

```
import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.Map.Entry;
public class LinkedHashMapTester4 {
   public static void main(String[] args) {
       LinkedHashMap<Character, String> lhmap = new LinkedHashMap<>();
       lhmap.put('R', "Red");
       lhmap.put('G', "Green");
       lhmap.put('B', "Brown");
       lhmap.put('O', "Orange");
       lhmap.put('P', "Pink");
       System.out.println("Entries in LinkedHashMap lhmap: " +lhmap);
       Iterator<Entry<Character, String>> itr = lhmap.entrySet().iterator(); // entrySet used to get entries of a linked hash map
       System.out.println("Iterating Entries of LinkedHashMap");
       while(itr.hasNext()) {
           Object key = itr.next();
           System.out.println(key);
       }
       System.out.println("\n");

       Iterator<Character> itr2 = lhmap.keySet().iterator(); // keySet used to get view of keys of a linked hash map.
       System.out.println("Iterating Keys of LinkedHashMap");
       while(itr2.hasNext()) {
           Object keyView = itr2.next();
           System.out.println(keyView);
       }
       System.out.println("\n");

       Iterator<String> itr3 = lhmap.values().iterator(); // values is used to get values of keys of a linkedhashmap.
       System.out.println("Iterating Values of LinkedHashMap");
       while(itr3.hasNext()) {
           Object valuesView = itr3.next();
           System.out.println(valuesView);
       }
   }
}
```

Example 4: Iterate LinkedHashMap with Iterator

Output:

```
Entries in LinkedHashMap lhmap: {R=Red, G=Green, B=Brown, O=Orange, P=Pink}
Iterating Entries of LinkedHashMap
R=Red
G=Green
B=Brown
O=Orange
P=Pink


Iterating Keys of LinkedHashMap
R
G
B
O
P


Iterating Values of LinkedHashMap
Red
Green
Brown
Orange
Pink
```

Example 5: Iterate LinkedHashMap with forEach()

5. Let’s iterate entries of linked hashmap using forEach() method.

Example 5: Iterate LinkedHashMap with forEach()

```
import java.util.LinkedHashMap;
public class LinkedHashMapTester5 {
   public static void main(String[] args){
       LinkedHashMap<Character, String> lhmap = new LinkedHashMap<>();
       lhmap.put('R', "Red");
       lhmap.put('G', "Green");
       lhmap.put('B', "Brown");
       lhmap.put('O', "Orange");
       lhmap.put('P', "Pink");

       System.out.println("Iterating Entries of LinkedHashMap");

       // Iteration over map using forEach() method.
       lhmap.forEach((k,v) -> System.out.println("Color code: "+ k + ", Color name: " + v));
   }
}
```

Example 5: Iterate LinkedHashMap with forEach()

Output:

```
Iterating Entries of LinkedHashMap
Color code: R, Color name: Red
Color code: G, Color name: Green
Color code: B, Color name: Brown
Color code: O, Color name: Orange
Color code: P, Color name: Pink
```

Example 6: removeEldestEntry()

6. Let’s take an example program based on removeEldestEntry() method.

Example 6: removeEldestEntry()

```
import java.util.LinkedHashMap;
import java.util.Map;
public class LinkedHashMapTester6 {
   public static void main(String[] args) {
       final int max = 5;
       LinkedHashMap<String, String> lhmap = new LinkedHashMap<String,String>(){
           protected boolean removeEldestEntry(Map.Entry<String, String> e) {
               return size() > max;
           }
       };
       lhmap.put("R", "Red");
       lhmap.put("G", "Green");
       lhmap.put("B", "Brown");
       lhmap.put("O", "Orange");
       lhmap.put("P", "Pink");

       System.out.println("Initial Entries of LinkedHashMap");
       System.out.println(lhmap);

       // Adding more entry into linkedhashmap.
       lhmap.put("W", "White");
       System.out.println("Displaying Map after adding more entry: " +lhmap);

       lhmap.put("Y", "Yellow"); // Adding one more entry into linkedhashmap.
       System.out.println("Displaying Map after adding one more entry: " +lhmap);
   }
}
```

Example 6: removeEldestEntry()

Output:

```
Initial Entries of LinkedHashMap
{R=Red, G=Green, B=Brown, O=Orange, P=Pink}
Displaying Map after adding more entry: {G=Green, B=Brown, O=Orange, P=Pink, W=White}
Displaying Map after adding one more entry: {B=Brown, O=Orange, P=Pink, W=White, Y=Yellow}
```

Use of LinkedHashMap

`LinkedHashMap` can be used when you want to preserve the insertion order.

`LinkedHashMap` is slower than `HashMap` but it is suitable when more number of insertions and deletions happen.

HashMap vs LinkedHashMap

Both `HashMap` and `LinkedHashMap` classes provide comparable performance but `HashMap` is a natural choice if the ordering of elements is not an issue.

Adding, removing, and finding entries in a `LinkedHashMap` is slightly slower than in `HashMap` because it needs to maintain the order of doubly linked list in addition to the hashed data structure.

END