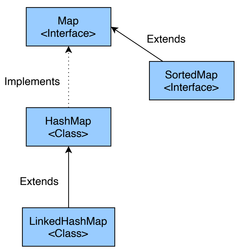
We have already discussed that a **HashMap** does not maintain insertion order and **TreeMap** stores the elements in sorted order. Now, if we want to store the elements in a Map in insertion order, then a **LinkedHashMap** can be used. **LinkedHashMap** is a class in the **java.utils** package that implements the **Map** interface and extends the **HashMap** class. It is similar to **HashMap** with the additional feature of maintaining the order of elements inserted into it.

Some of the important features of a **LinkedHashMap** are:

1. It does not allow duplicate keys.
2. It may have one null key and multiple null values.
3. It is non-synchronized.



LinkedHashMap hierarchy

Creating a LinkedHashMap

There are five different constructors available to create a LinkedHashMap object. We will discuss each of them here:

Using a no-arg constructor

The no-arg constructor, `LinkedHashMap()`, creates a Map with a default capacity of 16 and a default load factor of 0.75. The elements are stored in the insertion order.

Using the constructor that takes initial capacity

The `LinkedHashMap(int capacity)` constructor is used if we need to provide the initial capacity of the Map. This constructor is used if we are already aware of the number of elements that the Map will store. The initial capacity should be greater than zero; otherwise, `IllegalArgumentException` will be thrown.

Using the constructor that takes initial capacity and load factor

The `LinkedHashMap(int capacity, float loadFactor)` constructor is used if we need to provide both the initial capacity and the load factor of the Map. Both the initial capacity and load factor should be greater than zero. The load factor should be less than one.

Using the constructor that takes access order flag

The `LinkedHashMap(int capacity, float loadFactor, boolean accessOrder)` accepts three arguments: the initial capacity, the load factor, and the accessOrder flag. If the accessOrder is `false`, the elements will be stored in the order of insertion. If it is `true`, then the elements are stored in order of access. It means that the element that is accessed most recently is kept last.

Using the constructor that takes another Map

This constructor creates an insertion-ordered LinkedHashMap instance with the same mappings as the specified map. The LinkedHashMap instance is created with a default load factor (0.75) and an initial capacity sufficient to hold the mappings in the specified map.

Inserting into a LinkedHashMap

The **LinkedHashMap** class does not have a `put()` method, but since it extends the **HashMap** class, the `put()` method from the **HashMap** class is used. In the below example, we have created a **LinkedHashMap** in which few elements are inserted. On printing the elements, we can see that the elements were inserted as per the insertion order.

```
import java.util.LinkedHashMap;
import java.util.Map;

public class LinkedHashMapDemo {

    public static void main(String args[]) {

        Map<String, Integer> stocks = new LinkedHashMap<>();

        stocks.put("Apple", 123);
        stocks.put("BMW", 54);
        stocks.put("Google", 87);
        stocks.put("Microsoft", 232);
        stocks.put("Oracle", 76);

        System.out.println(stocks);
    }
}

```

Run Save Reset

In the below example, we have created a LinkedHashMap with the access order flag as `true`. Now the elements that were accessed most recently will be kept at the last position.

```
import java.util.LinkedHashMap;
import java.util.Map;

public class LinkedHashMapDemo {

    public static void main(String args[]) {

        Map<String, Integer> stocks = new LinkedHashMap<>(16, 0.75f, true);

        stocks.put("Apple", 123);
        stocks.put("BMW", 54);
        stocks.put("Google", 87);
        stocks.put("Microsoft", 232);
        stocks.put("Oracle", 76);

        System.out.println(stocks);
        stocks.get("Google");
        stocks.get("BMW");

        System.out.println(stocks);
    }

}

```

Run Save Reset