# Java ConcurrentHashMap Example and Iterator

JULY 29, 2016 BY PANKAJ — 39 COMMENTS

Today we will look into Java ConcurrentHashMap Example. If you are a Java Developer, I am sure that you must be aware of `ConcurrentModificationException` that comes when you want to modify the Collection object while using iterator to go through with all its element. Actually Java Collection Framework iterator is great example of **iterator design pattern** implementation.

## Java ConcurrentHashMap


Java 1.5 has introduced `java.util.concurrent` package with Collection classes implementations that allow you to modify your collection objects at runtime.

### ConcurrentHashMap Example

`ConcurrentHashMap` is the class that is similar to HashMap but works fine when you try to modify your map at runtime.

Lets run a sample program to explore this:

`ConcurrentHashMapExample.java`

```
                myMap.put("3", "1");
                myMap.put("4", "1");
                myMap.put("5", "1");
                myMap.put("6", "1");
                System.out.println("ConcurrentHashMap before iterator: "+myMap);
                Iterator<String> it = myMap.keySet().iterator();

                while(it.hasNext()){
                        String key = it.next();
                        if(key.equals("3")) myMap.put(key+"new", "new3");
                }
                System.out.println("ConcurrentHashMap after iterator: "+myMap);

                //HashMap
                myMap = new HashMap<String,String>();
                myMap.put("1", "1");
                myMap.put("2", "1");
                myMap.put("3", "1");
                myMap.put("4", "1");
                myMap.put("5", "1");
                myMap.put("6", "1");
                System.out.println("HashMap before iterator: "+myMap);
                Iterator<String> it1 = myMap.keySet().iterator();

                while(it1.hasNext()){
                        String key = it1.next();
                        if(key.equals("3")) myMap.put(key+"new", "new3");
                }
                System.out.println("HashMap after iterator: "+myMap);
        }

}
```

When we try to run the above class, output is

```
ConcurrentHashMap before iterator: {1=1, 5=1, 6=1, 3=1, 4=1, 2=1}
ConcurrentHashMap after iterator: {1=1, 3new=new3, 5=1, 6=1, 3=1, 4=1, 2=1}
HashMap before iterator: {3=1, 2=1, 1=1, 6=1, 5=1, 4=1}
Exception in thread "main" java.util.ConcurrentModificationException
        at java.util.HashMap$HashIterator.nextEntry(HashMap.java:793)
        at java.util.HashMap$KeyIterator.next(HashMap.java:828)
        at com.test.ConcurrentHashMapExample.main(ConcurrentHashMapExample.java:44)
```

Looking at the output, its clear that **ConcurrentHashMap** takes care of any new entry in the map whereas HashMap throws `ConcurrentModificationException`.

```
String key = it1.next();
```

It means that the new entry got inserted in the HashMap but Iterator is failing. Actually Iterator on Collection objects are **fail-fast** i.e any modification in the structure or the number of entry in the collection object will trigger this exception thrown by iterator.

So How does iterator knows that there has been some modification in the HashMap. We have taken the set of keys from HashMap once and then iterating over it.

HashMap contains a variable to count the number of modifications and iterator use it when you call its next() function to get the next entry.

HashMap.java

```
/**
     * The number of times this HashMap has been structurally modified
     * Structural modifications are those that change the number of mappings in
     * the HashMap or otherwise modify its internal structure (e.g.,
     * rehash).  This field is used to make iterators on Collection-views of
     * the HashMap fail-fast.  (See ConcurrentModificationException).
     */
    transient volatile int modCount;
```

Now to prove above point, lets change the code a little bit to come out of the iterator loop when we insert the new entry. All we need to do is add a break statement after the put call.

```
if(key.equals("3")){
                                myMap.put(key+"new", "new3");
                                break;
                        }
```

Now execute the modified code and the output will be:

```
ConcurrentHashMap before iterator: {1=1, 5=1, 6=1, 3=1, 4=1, 2=1}
ConcurrentHashMap after iterator: {1=1, 3new=new3, 5=1, 6=1, 3=1, 4=1, 2=1}
HashMap before iterator: {3=1, 2=1, 1=1, 6=1, 5=1, 4=1}
HashMap after iterator: {3=1, 2=1, 1=1, 3new=new3, 6=1, 5=1, 4=1}
```

Finally, what if we won't add a new entry but update the existing key-value pair. Will it throw exception?

Change the code in the original program and check yourself.

```
//myMap.put(key+"new", "new3");
myMap.put(key, "new3");
```