24 Prefer for-loop than while.. for(Iterator entries = myMap.entrySet().iterator(); entries.hasNext();) {...} With this syntax the 'entries' scope is reduced to the for loop only. – HanuAthena Oct 20 '09 at 13:20

3 @jpredham You are right that using the `for` construct as `for (Entry e : myMap.entrySet)` will not allow you to modify the collection, but the example as @HanuAthena mentioned it should work, since it gives you the `Iterator` in scope. (Unless I'm missing something...) – pkaeding Jan 10 '12 at 15:42

In Java 8 you can do it clean and fast using the new lambdas features:

```
Map<String,String> map = new HashMap<>();
map.put("SomeKey", "SomeValue");
map.forEach( (k,v) -> [do something with key and value] );

// such as
map.forEach( (k,v) -> System.out.println("Key: " + k + ": Value: " + v));
```

The type of `k` and `v` will be inferred by the compiler and there is no need to use `Map.Entry` anymore.

Easy-peasy!

edited Sep 7 '16 at 8:33

answered Oct 21 '13 at 10:15

Saint Hill
6,756 6 31 53

7 Depending on what you want to do with a map, you can also use stream API on the entries returned by `map.entrySet().stream()` docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html – Vitalii Fedorenko Jun 28 '14 at 12:46

Summarize other answers and what I known, I found 10 main ways to do this (see below). And I wrote some performance tests (see results below), for example, if we want to find sum of all keys and values of map, we can write :

1. Using **iterator** and **Map.Entry**

```
long i = 0;
Iterator<Map.Entry<Integer, Integer>> it = map.entrySet().iterator();
while (it.hasNext()) {
    Map.Entry<Integer, Integer> pair = it.next();
    i += pair.getKey() + pair.getValue();
}
```

2. Using **foreach** and **Map.Entry**

```
long i = 0;
for (Map.Entry<Integer, Integer> pair : map.entrySet()) {
    i += pair.getKey() + pair.getValue();
}
```

3. Using **foreach** from `Java 8`

```
final long[] i = {0};
map.forEach((k, v) -> i[0] += k + v);
```

4. Using **keySet** and **foreach**

```
long i = 0;
for (Integer key : map.keySet()) {
    i += key + map.get(key);
}
```

5. Using **keySet** and **iterator**

```
long i = 0;
Iterator<Integer> itr2 = map.keySet().iterator();
while (itr2.hasNext()) {
    Integer key = itr2.next();
    i += key + map.get(key);
}
```

6. Using **for** and **Map.Entry**

```
long i = 0;
for (Iterator<Map.Entry<Integer, Integer>> entries = map.entrySet().iterator();
entries.hasNext(); ) {
    Map.Entry<Integer, Integer> entry = entries.next();
    i += entry.getKey() + entry.getValue();
}
```

7. Using `Java 8` **Stream Api**

```
final long[] i = {0};
map.entrySet().stream().forEach(e -> i[0] += e.getKey() + e.getValue());
```

8. Using `Java 8` **Stream Api parallel**

```
final long[] i = {0};
map.entrySet().stream().parallel().forEach(e -> i[0] += e.getKey() + e.getValue());
```

9. Using **IterableMap** of `Apache Collections`

```
long i = 0;
MapIterator<Integer, Integer> it = iterableMap.mapIterator();
while (it.hasNext()) {
    i += it.next() + it.getValue();
}
```

10. Using **MutableMap** of `Eclipse (CS) collections`

```
final long[] i = {0};
mutableMap.forEachKeyValue((key, value) -> {
    i[0] += key + value;
});
```

Perfomance tests (mode = AverageTime, system = Win 8.1 64-bit, Intel i7-4790 3.60GHz 3.60GHz, 16 GB)

1) For small map (100 elements), score 0.308 is the best

Benchmark	Mode	Cnt	Score	Error	Units
test3_UsingForEachAndJava8	avgt	10	0.308 ±	0.021	us/op
test10_UsingEclipseMap	avgt	10	0.309 ±	0.009	us/op
test1_UsingWhileAndMapEntry	avgt	10	0.380 ±	0.014	us/op
test6_UsingForAndIterator	avgt	10	0.387 ±	0.016	us/op
test2_UsingForEachAndMapEntry	avgt	10	0.391 ±	0.023	us/op
test7_UsingJava8StreamApi	avgt	10	0.510 ±	0.014	us/op
test9_UsingApacheIterableMap	avgt	10	0.524 ±	0.008	us/op
test4_UsingKeySetAndForEach	avgt	10	0.816 ±	0.026	us/op
test5_UsingKeySetAndIterator	avgt	10	0.863 ±	0.025	us/op
test8_UsingJava8StreamApiParallel	avgt	10	5.552 ±	0.185	us/op

2) For map with 10000 elements, score 37.606 is the best

Benchmark	Mode	Cnt	Score	Error	Units
test10_UsingEclipseMap	avgt	10	37.606 ±	0.790	us/op
test3_UsingForEachAndJava8	avgt	10	50.368 ±	0.887	us/op
test6_UsingForAndIterator	avgt	10	50.332 ±	0.507	us/op
test2_UsingForEachAndMapEntry	avgt	10	51.406 ±	1.032	us/op
test1_UsingWhileAndMapEntry	avgt	10	52.538 ±	2.431	us/op
test7_UsingJava8StreamApi	avgt	10	54.464 ±	0.712	us/op
test4_UsingKeySetAndForEach	avgt	10	79.016 ±	25.345	us/op
test5_UsingKeySetAndIterator	avgt	10	91.105 ±	10.220	us/op
test8_UsingJava8StreamApiParallel	avgt	10	112.511 ±	0.365	us/op
test9_UsingApacheIterableMap	avgt	10	125.714 ±	1.935	us/op

3) For map with 100000 elements, score 1184.767 is the best

Benchmark	Mode	Cnt	Score	Error	Units
test1_UsingWhileAndMapEntry	avgt	10	1184.767 ±	332.968	us/op
test10_UsingEclipseMap	avgt	10	1191.735 ±	304.273	us/op
test2_UsingForEachAndMapEntry	avgt	10	1205.815 ±	366.043	us/op
test6_UsingForAndIterator	avgt	10	1206.873 ±	367.272	us/op
test8_UsingJava8StreamApiParallel	avgt	10	1485.895 ±	233.143	us/op
test5_UsingKeySetAndIterator	avgt	10	1540.281 ±	357.497	us/op
test4_UsingKeySetAndForEach	avgt	10	1593.342 ±	294.417	us/op
test3_UsingForEachAndJava8	avgt	10	1666.296 ±	126.443	us/op
test7_UsingJava8StreamApi	avgt	10	1706.676 ±	436.867	us/op
test9_UsingApacheIterableMap	avgt	10	3289.866 ±	1445.564	us/op

Graphs (perfomance tests depending on map size)

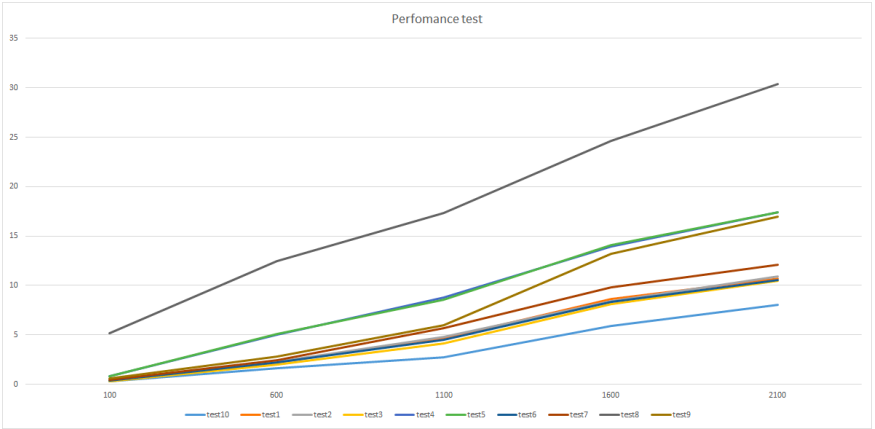

Table (perfomance tests depending on map size)