

Java Multithreading for Senior Engineering Interviews / ... / ConcurrentModificationException

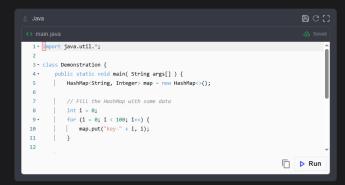
ConcurrentModificationException

This lesson explains why ConcurrentModificationExceptions occurs and how it can be avoided.

If you are interviewing, consider buying our number#1 course for <u>Java Multithreading</u> Interviews.

Single Thread Environment

The name ConcurrentModificationException may sound related to concurrency, however, the exception can be thrown while a single thread operates on a map. In fact, ConcurrentModificationException isn't even part of the java.util.concurrent package. The exception occurs when a map is modified at the same time (concurrently) any of its collection views (keys, values or entry pairs) is being traversed. The program below demonstrates the exception being thrown as the main thread traverses the map entries and also attempts to insert new entries.



Multithread Environment

In case of a single threaded environment it is often trivial to diagnose ConcurrentModificationException cause, however, in multithreaded scenarios, it may be difficult to do so as the exception may occur intermittently depending on how threads are scheduled for execution. Concurrent modification occurs when one thread is iterating over a map while another thread attempts to modify the map at the same time. A usual sequence of execution is as follows:

- 1. Thread A obtains an iterator for the keys, values or entry set of a map.
- 2. Thread A begins to iterate in a loop.
- Thread B comes along and attempts to delete, insert or update a key/value pair in the map.
- Concurrent Modification Exception is thrown when thread A attempts to retrieve the next item in the collection it is iterating.

Since the map has been modified from the time the iterator for the map was created, the events is as follows:

- 1. Thread A obtains an iterator for the keys, values or entry set of a map.
- 2. Thread A begins to iterate in a loop.
- Thread B comes along and attempts to delete, insert or update a key/value pair in the map.
- ConcurrentModificationException is thrown when thread A attempts to retrieve the next item in the collection it is iterating.

Since the map has been modified from the time the iterator for the map was created, the thread iterating over the collection can observe inconsistent data and a ConcurrentModificationException is thrown. The program below demonstrates interaction between two threads that results in the exception.

```
Java

The port java util. HashMap;

import java. util. HashMap;

import java. util. concurrent. Executor Service;

import java. util. concurrent. Executor Service;

import java. util. concurrent. Executor Service;

import java. util. concurrent. Future;

7 - class Demonstration {

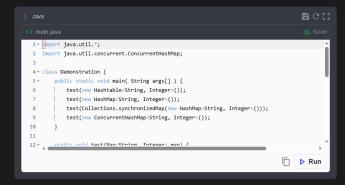
8 - public static void main( String args[] ) throws Exception {

10 | HashMap String, Integer> map - new HashMap(>)();

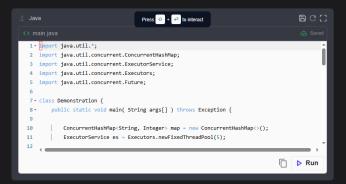
11 | Executor Servictor manufice with TheadMonal(S).
```

□ ▶ Run

It is not only the HashMap that suffers from ConcurrentModificationException, other maps exhibit same behavior. The only map that is designed to be concurrently modified while being traversed is the ConcurrentHashMap. The program below demonstrates the behavior of all the maps.



Even though the concurrentHashaMap can undergo concurrent modifications (additions, deletions, updates) at the same time as its elements are being traversed, the modifications may not be reflected during the traversal. Consider the program below in which a reader thread starts traversing a map's entries while the map is being written to by a writer thread. The reader thread only observes a limited number of entries reflecting the state of the map at some point at or since the creation of the iterator/enumeration.



In the above program, the writer inserts 1000 entries into the map but the reader only sees a

As a user of ConcurrentHashMap one has to be cognizant of the limitation of iterators/enumerators, which may return a snapshot of the map taken at the time of creation of the iterator/enumeration or later.

```
Technical Quiz

Consider the program below and answer:

static void quiz() {

MapcString, Integer> map = new HashMapc>();
Random random = new Random(System.currentTimeMillis());

// Put some data in the map
for (int i = 0; i < 10; i++) {

| map.put("key-" + i, i);
}

Iterator it = map.entryset().iterator();

while (it.hasNext()) {

| it.next();

| int k = random.nextInt(10);

| map.put("key-" + k, k);
}
}

A Program throws ConcurrentModificationException

B Program doesn't throw any exception
```

The above scenario is very interesting since we are modifying the map but we are essentially overwriting the same key/value pair and no exception is thrown. In some cases such as handling duplicates (e.g. key/value pairs received from a message bus), a program

can overwrite the same key/value pair twice (an example of *idempotent write*) and continue to function correctly but under different conditions may throw

${\sf ConcurrentModificationException.}$

The program from the quiz is reproduced in the widget below.

