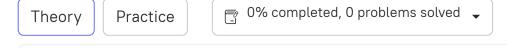
<u>Computer science</u> → <u>Programming languages</u> → <u>Java</u> → <u>Working with data</u> → <u>Multithreading</u> → <u>Thread-safe collections</u>

Collections and thread-safety



Theory () 4 minutes reading

Verify to skip

§1. Classic collections and multithreading

Start practicing

As you already know, several threads may access the same data concurrently that often leads to different problems if we do not use some kind of synchronization.

Similar problems occur when multiple threads access a collection:

- most of the classic collections like ArrayList, LinkedList, HashMap and others are non-synchronized and, as a consequence, they do not provide thread-safety;
- there is a set of old collections like Vector, Stack, and Hashtable that are totally synchronized and thread-safe, but they have low performance;
- when one thread is iterating over a standard totally synchronized collection and another thread tries to add a new element to it then we get a runtime exception called ConcurrentModificationException.

The following program demonstrates a race condition that appears when two threads add elements to the same collection.

```
import java.util.ArrayList;
1
2
3
     public class NeedOfConcurrentCollectionsDemo {
4
         public static void main(String[] args) throws
5
InterruptedException {
             ArrayList<Integer> numbers = new ArrayList<>();
6
7
8
             Thread writer = new Thread(() -> addNumbers(numbers));
 9
             writer.start();
 1
 0
1
             addNumbers(numbers); // add number from the main thread
1
 2
1
 3
             writer.join(); // wait for writer thread
4
1
 5
             System.out.println(numbers.size()); // the result can be any
         }
6
1
 7
1
         private static void addNumbers(ArrayList<Integer> target) {
 8
1
             for (int i = 0; i < 100 000; i++) {
9
                 target.add(i);
 0
 2
             }
 1
```

2 required topics What are collections Thread synchronization 3 dependent topics Thread-safe maps CopyOnWriteArrayList

Concurrent queues

https://hyperskill.org/learn/step/4984

The expected result is 200000 (100000 + 100000), but in fact, it changes every time you run this code. Some elements of a list are lost.

We started the program three times and get the following results:

```
1 196797
2 154802
3 181359
```

You may also try it yourself.

So, it is a bad idea to use standard collections in multithreaded environments without explicit synchronization. Again, though, such synchronization may lead to poor performance and hard-to-find errors in large programs.

§2. Concurrent collections

To avoid all the problems associated with custom synchronization, Java Class Library provides alternative collection implementations that are adapted to be used in multithreaded applications and they are fully thread-safe. You may find them in the <code>java.util.concurrent</code> package that includes lists, queues, maps and other collections which make it easier to develop modern Java applications.

These concurrent collections allow you to avoid custom synchronization in many cases as well as they have high performance close to classic collections.

Concurrent collections do not use the synchronized keyword but rely on more complex synchronization primitives and lock-free algorithm that allows them to be both thread-safe and high-performance. However, if you do not really need multithreading, use classic collections, since they are still more efficient than concurrent ones.

You will learn the different types of concurrent collections later on.

Report a typo

Table of contents:

multithreading

Discussion

↑ Collections and thread-safety

§1. Classic collections and

§2. Concurrent collections

154 users liked this piece of theory. 2 didn't like it. What about you?











Start practicing

Verify to skip

<u>Comments (6)</u>

<u>Useful links (0)</u>

Show discussion

JetBrains Academy

Tracks

About

😎 Become beta tester

Pricing

For organizations

Careers

Contribute

Be the first to see what's new

Terms Support





Made with ♥ by Hyperskill and JetBrains

https://hyperskill.org/learn/step/4984