https://techvidvan.com/tutorials/java-concurrentmodificationexception/

How to Avoid ConcurrentModificationException in Java

In this article, we will study a predefined Exception in Java which is ConcurrentModificationException. It is a very common exception that occurs while working with Collections in Java. You must know this exception and how to handle the same.

We will study this Exception with examples and the ways to avoid this exception.

ConcurrentModificationException in Java

Whenever we try to modify an object concurrently without permission, then the ConcurrentModificationException occurs. We often face this exception usually when we are working with Collection classes of Java. This exception is present in the java.util package.

The Collection classes of Java are very fail-fast and if we try to change them while some thread is traversing over it using an iterator, then we will get a ConcurrentModificationException from iterator.next() method.

The Concurrent modification exception can occur in the multithreaded as well as a single-threaded Java programming environment.

Let’s take an example. A thread is not permitted to modify a Collection when some other thread is iterating over it because the result of the iteration becomes undefined with it.

Some implementations of the Iterator class throw this exception, including all those general-purpose implementations of Iterator that JRE provides.

Such Iterators that do this are called fail-fast because they throw the exception as soon as they encounter such a situation rather than facing undetermined behavior of the Collection any time in the future.

***Note:***  
It is not necessary that this exception will occur only when some other thread tries to modify a Collection object. It can also occur if the methods of a single thread are trying to violate the contract of the object.

### Example of Java ConcurrentModificationException

Now, let us discuss the Java concurrent modification exception with the help of an example.

**Code 1 to explain ConcurrentModificationException:**

package com.techvidvan.concurrentmodificationexception;

import java.awt.List;

import java.util. \* ;

public **class** ConcurrentModificationException {

public static **void** main(String[] args) {

ArrayList < Integer > list = **new** ArrayList < >();

list.add(1);

list.add(2);

list.add(3);

list.add(4);

list.add(5);

Iterator < Integer > iterator = list.iterator();

**while** (iterator.hasNext()) {

Integer value = iterator.next();

System.out.println("The value of the List is:" + value);

**if** (value.equals(3)) list.remove(value);

}

}

**Output**

The value of the List is:1  
The value of the List is:2  
The value of the List is:3  
Exception in thread “main” java.util.ConcurrentModificationException  
at java.util.ArrayList$Itr.checkForComodification(ArrayList.java:901)  
at java.util.ArrayList$Itr.next(ArrayList.java:851)  
at ConcurrentModificationException.main(ConcurrentModificationException.java:18)

The above output message says that the exception occurs when we call the next method as the iterator is iterating the list and we are making modifications in it simultaneously.

But if we make changes in the hashmap like the code below, then it will not throw any such exception as the size of the hashmap won’t change. So let’s see this example.

Constructors of ConcurrentModificationException

There are 4 types of constructors of ConcurrentModificationException which are given in the following table:

|  |  |  |
| --- | --- | --- |
| **S.N** | **Constructor** | **Description** |
| 1. | public ConcurrentModificationException() | This constructor creates a ConcurrentModification Exception with no parameters. |
| 2. | public ConcurrentModificationException(String message) | This constructor creates a ConcurrentModification Exception with a detailed message of the Exception. |
| 3. | public ConcurrentModificationException(Throwable cause) | This constructor creates a ConcurrentModification Exception with a cause of the exception and a message which is (cause==null?null:cause.toString()). We can later retrieve this  message by using Throwable.getCause(). |
| 4. | public ConcurrentModificationException(String message, Throwable cause) | This  constructor creates a ConcurrentModification Exception with a cause and detailed message. We can retrieve this  message using Throwable.getMessage() and the cause by Throwable.getCause(). |

How to avoid ConcurrentModificationException in a multi-threaded environment?

We can follow some precautions or ways to avoid ConcurrentModificationException in a multi-threaded environment:

**1.** We can iterate over the array instead of iterating over the collection class. Following this way, we can work very well with small-sized lists, but this will degrade the performance if the array size is very large.

**2.** Locking the list by putting it in the synchronized block is another way to avoid the concurrent modification exception. This is not an effective approach; it is not using the sole purpose of multi-threading.

**3.** The classes ConcurrentHashMap and CopyOnWriteArrayList of JDK 1.5 or higher versions can also help us in avoiding concurrent modification exceptions.

How To Avoid Concurrent Modification Exception in a single-threaded environment?

We can also avoid the Concurrent Modification Exception in a single threaded environment. We can use the remove() method of Iterator to remove the object from the underlying collection object. But in this case, you can remove only the same object and not any other object from the list.

**Code to understand above concept:**

package com.technvidvan.concurrentmodificationexception;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import java.util.concurrent.ConcurrentHashMap;

import java.util.concurrent.CopyOnWriteArrayList;

public **class** AvoidConcurrentModificationException {

public static **void** main(String[] args) {

List < String > myList = **new** CopyOnWriteArrayList < String > ();

myList.add("1");

myList.add("2");

myList.add("3");

myList.add("4");

myList.add("5");

Iterator < String > it = myList.iterator();

**while** (it.hasNext()) {

String value = it.next();

System.out.println("List Value:" + value);

**if** (value.equals("3")) {

myList.remove("4");

myList.add("6");

myList.add("7");

}

}

System.out.println("List Size:" + myList.size());

Map < String,

String > myMap = **new** ConcurrentHashMap < String,

String > ();

myMap.put("1", "1");

myMap.put("2", "2");

myMap.put("3", "3");

Iterator < String > it1 = myMap.keySet().iterator();

**while** (it1.hasNext()) {

String key = it1.next();

System.out.println("Map Value:" + myMap.get(key));

**if** (key.equals("1")) {

myMap.remove("3");

myMap.put("4", "4");

myMap.put("5", "5");

}

}

System.out.println("Map Size:" + myMap.size());

}

}

**Output**

List Value:1  
List Value:2  
List Value:3  
List Value:4  
List Value:5  
List Size:6  
Map Value:1  
Map Value:2  
Map Value:4  
Map Value:5  
Map Size:4

Using of loop to avoid ConcurrentModificationException

We can also use the for loop rather than an iterator to avoid concurrent modification exceptions in a single-threaded environment. We can ensure that our code takes care of the extra added objects in the list.

**Code to use for loop to remove Concurrent Modification Exception:**

package com.technvidvan.concurrentmodificationexception;

import java.util.List;

import java.util.Map;

import java.util.concurrent.ConcurrentHashMap;

import java.util.concurrent.CopyOnWriteArrayList;

public **class** AvoidConcurrentModificationException {

public static **void** main(String[] args) {

List < String > myList = **new** CopyOnWriteArrayList < String > ();

myList.add("1");

myList.add("2");

myList.add("3");

myList.add("4");

myList.add("5");

**for** (int i = 0; i < myList.size(); i++) {

System.out.println("List value: " + myList.get(i));

**if** (myList.get(i).equals("3")) {

myList.remove(i);

i--;

myList.add("6");

}

}

System.out.println("List Size:" + myList.size());

}

}

**Output**

List value: 1  
List value: 2  
List value: 3  
List value: 4  
List value: 5  
List value: 6  
List Size:5

The ConcurrentModificationException will occur if you try to modify the structure of the original list with the subList. Let’s see this with a simple example.

**Example:**

package com.technvidvan.concurrentmodificationexception;

import java.util.ArrayList;

import java.util.List;

public **class** ConcurrentModificationExceptionWithArrayListSubList {

public static **void** main(String[] args) {

List < String > names = **new** ArrayList < >();

names.add("Java");

names.add("PHP");

names.add("SQL");

names.add("Angular 2");

List < String > first2Names = names.subList(0, 2);

System.out.println(names + " , " + first2Names);

names.set(1, "JavaScript");

// check the output below. :)

System.out.println(names + " , " + first2Names);

// Let's modify the list size and get ConcurrentModificationException

names.add("NodeJS");

System.out.println(names + " , " + first2Names); // this line throws exception

}

}