

Applause from David Smooke and 1 other



Adrian D. Finlay

@thewipprogrammer. Writer @hackernoon. Code, LOTS of it. Mangos, LOVE THEM! Barbering. Health. Travel. Business. & more! Network w/ me @ adriandavid.me/network

Mar 14 · 6 min read

java.lang.Object.finalize() is finally deprecated



GIPHY

And thus, `finalize()`, having been fed up with his own unreliability, walked out of `Object`'s boardroom, never to be seen again, or so we would like to believe.

Although many may not have realized, with the JDK 9 release came the **deprecation** of `finalize()`. Don't believe me, check [Java 9's documentation](#)! The reader be aware that **deprecation does not necessarily mean removal or future removal**. It is an indication used to note that the annotated element should (if possible) be avoided and may potentially be removed in future releases of the Java platform.

At the moment, it is not certain whether or not `finalize()` will be marked for removal. I personally don't believe that it will be removed, but I am not sure. All I can say is that Early Access Builds of JDK 11 have not removed it and as it is a method that is inherited by every single class in

the Java ecosystem, my gut is nudging me to believe it's going to stay put.

Nevertheless, if the metadata is any indication of what the API writer's intentions are, at least at the time of writing, it seems as if it is not going to be removed—at least not in the immediate future. We can induce this by running and analyzing the result of this code:

```
/* Note that this uses language features only available
in Java 10 and 11, namely, local variable type
inference. The use of this feature within Lambda
Expression arguments came about in Java 11 and is used
in this article. Java 10 is set to be released 6 days
following the publishing of this article. You may learn
more about local variable type reference @
bit.ly/VarJava */

import java.lang.annotation.Annotation;

...

var annos = Object.class.getDeclaredMethod("finalize",
(Class<?>[]) null ).getAnnotations();

for (var anno : annos) System.out.println(anno + "\n");

Output: @java.lang.Deprecated(forRemoval=false,
since="9")
```

We ran this code for fairly obvious reasons, which is to show, programmatically, that `java.lang.Object.finalize()` has not been marked for removal.

The deprecation of `finalize()` can be somewhat considered as ceremonial icing on a long-time proverbial cake.

Dating all the way back to 2001, in the first edition of his blockbuster classic “*Effective Java*” (which ought to be compulsory reading for serious Java developers), Joshua Bloch warned about the dangers of relying on `finalize()`.

Sun (and thereafter, Oracle) has long cautioned Java developers about making use of Finalization and why it is almost (if not outright) always a bad idea. A primary motivation in developing `finalize()` was to be able to dispose of “unreachable objects” in code, such as native resources.

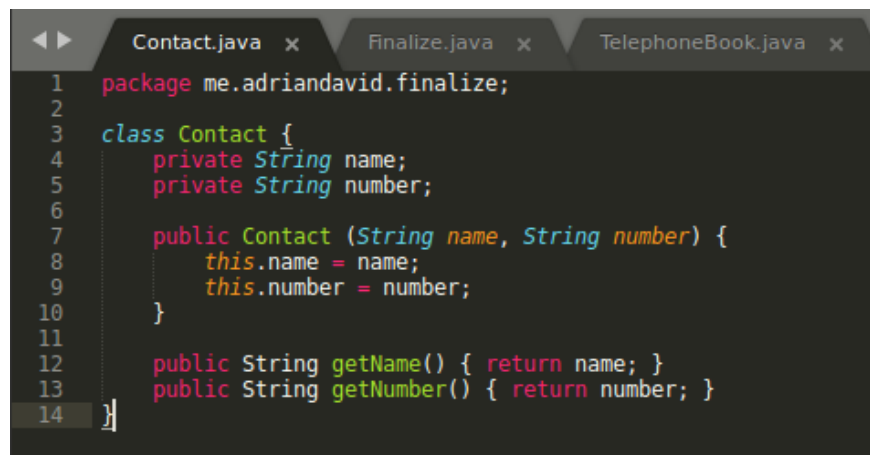
In the past, one might have been tempted to use `finalize()` as follows.

Some notes on my environment: I'll be using an early access build of JDK 11 on Linux Mint MATE.

Can `finalize()` reliably close your stream?

| *Let's write a telephone book modeled with the MVC Architectural pattern.*

First, the Model, a POJO bean called Contact.



```
1 package me.adriandavid.finalize;
2
3 class Contact {
4     private String name;
5     private String number;
6
7     public Contact (String name, String number) {
8         this.name = name;
9         this.number = number;
10    }
11
12    public String getName() { return name; }
13    public String getNumber() { return number; }
14 }
```

Second, the Controller, a class called TelephoneBook.

```
TelephoneBook.java x Contact.java x Finalize.java x TelephoneBook.txt x Test.java
1 package me.adriandavid.finalize;
2
3 import java.util.Map;
4 import java.util.HashMap;
5 import java.nio.file.Files;
6 import java.nio.file.Paths;
7 import java.io.IOException;
8 import java.io.PrintWriter;
9 import java.io.OutputStreamWriter;
10
11
12 class TelephoneBook {
13     //Member Variables
14     private java.io.PrintWriter record = null;
15     private Map<String,String> contacts = new HashMap<>();
16     //Provides minor advantage over System.out in that it is Internationally
17     //friendly and better suited for writing Characters to the console.
18     private PrintWriter out = new PrintWriter(new OutputStreamWriter(System.out, "UTF-8"), true);
19
20     //Constructor
21     public TelephoneBook() throws IOException {
22
23         //If TelephoneBook.txt exists, delete it.
24         if (Files.deleteIfExists(Paths.get("TelephoneBook.txt"))) {
25             out.println("Overwriting the file.");
26         }
27         else {
28             out.println("Creating the file.");
29         }
30
31         //Instantiate The Stream
32         try { record = new java.io.PrintWriter("TelephoneBook.txt"); }
33         catch (IOException e) {
34             out.println("The attempt to create a Telephone Book failed.");
35             System.exit(0);
36         }
37     }
38 }
```

```
TelephoneBook.java x Contact.java x Finalize.java x TelephoneBook.txt x
36
37 @Override /*Overriding Finalize*/
38 protected void finalize() {
39     out.println("Finalize will close your stream for you....");
40     if (record != null) record.close();
41 }
42
43 /*Utility Methods */
44
45 //Write to the File
46 void writeToFile() throws IOException {
47     // The Pre-Java 8 Way --->
48     // for (Map.Entry<String,Long> contact : contacts.entrySet())
49
50     //Java 8+ Functional way of for-each over a Map
51     contacts.forEach( (var NA, var NM) -> {
52         String EY = "\"" + NA + "\"" + ":\t+1-" + NM + "\n";
53         record.write(EY);
54     });
55 }
56
57 //Add Contacts
58 void addContacts(Contact ... contacts) {
59     for (var i=0; i<contacts.length; i++) {
60         this.contacts.put(contacts[i].getName(), contacts[i].getNumber());
61     }
62 }
63 //Print Map
64 public void printMap() { out.println(contacts); };
65 }
```

Third, the View, our main class, called Finalize.

```
Finalize.java x TelephoneBook.java x Contact.java x TelephoneBo
1 package me.adriandavid.finalize;
2
3 public class Finalize {
4     public static void main (String[] args) throws java.io.IOException {
5         var tpb = new TelephoneBook();
6         tpb.addContacts(
7             new Contact ("Michio", "8007823456"),
8             new Contact ("Albert", "8556723401"),
9             new Contact ("Stephen", "3052938712")
10        );
11
12        tpb.printMap();
13        tpb.writeToFile();
14
15        // Let's nudge the GC
16        tpb = null;
17        System.gc();
18    }
19 }
```

When run, this is the output.

```
adrian@adrian-ThinkPad-T520 ~/Desktop/Test
File Edit View Search Terminal Help

adrian@adrian-ThinkPad-T520 ~/Desktop/Test $ javac me/adriandavid/finalize/Finalize.java
adrian@adrian-ThinkPad-T520 ~/Desktop/Test $ java me.adriandavid.finalize.Finalize
Creating the file.
{Albert=8556723401, Michio=8007823456, Stephen=3052938712}
Finalize will close your stream for you...
adrian@adrian-ThinkPad-T520 ~/Desktop/Test $ java me.adriandavid.finalize.Finalize
Overwriting the file.
{Albert=8556723401, Michio=8007823456, Stephen=3052938712}
Finalize will close your stream for you...
adrian@adrian-ThinkPad-T520 ~/Desktop/Test $ java -version
java version "11-ea" 2018-09-18
Java(TM) SE Runtime Environment 18.9 (build 11-ea+4)
Java HotSpot(TM) 64-Bit Server VM 18.9 (build 11-ea+4, mixed mode)
adrian@adrian-ThinkPad-T520 ~/Desktop/Test $ javac -version
javac 11-ea
adrian@adrian-ThinkPad-T520 ~/Desktop/Test $
```

Analysis.

Take a moment and look over the code. The main thing to grasp here is that we are relying on `finalize()` to do some of our clean up: closing the `java.io.PrintWriter` stream. Now draw your attention to lines 16 and 17 of the `Finalize` class. Note that we both set an instance of `TelephoneBook` to null as well as meekly suggested to the Garbage Collector that now might be a good time to run. Recall that `System.gc()` is not a direct trigger of the garbage collector and `finalize()` is not a destructor. **Neither are required to run.** At the heart of all this is the fact that that Garbage Collector will run when it pleases and we can't (for our practical purposes) control when it does.

Did you notice that the Garbage Collector ran and that `finalize()` was called in our example? Try this experiment. Remove either line 16 or

line 17 of the `Finalize` class and see if the `finalize()` method was called. I'm willing to bet it didn't, although I can't be sure. For me, it repeatedly did not. Remember, we cannot (in most practical cases at least) deterministically control when the Garbage Collector is run.

Thus, if we rely on `finalize()` to close our `OutputStream`, we are engaging in sloppy programming.

Do you see why `finalize()` is a bad idea? This is just one example of many but the message is clear: **because we can't guarantee that `finalize()` will be called or determine when the GC will be run, we can't reliably use `finalize()` to finish closing tasks, such as closing I/O streams.** What if we had to close an `SSLSocket`? Or a `JDBC Connection`? Or a `JNI resource`? I'll leave it to the reader's imagination (or nightmares) to muse about the mayhem that could be caused.

Closing Thoughts

Java 9 is the long awaited ceremonializing an old truism: you can't rely on `finalize()` to clean up after you. Behave, and clean up your resources explicitly and by other means. As also hinted to, there are other issues with `finalize()`, namely: **performance, deadlocks, and hangs**[1].

To understand why `System.gc()` is bad practice in more depth, visit [here](#):

Why is it bad practice to call `System.gc()`?

This is a very bothersome question, and I feel contributes to many being opposed to Java despi...
stackoverflow.com



Despise `finalize()` like the rest of us? Or do you think it's still a good idea? Let me know in the comments below :)

Want the source? Grab it here.

You will need JDK 11 which you can find, [here](#).

afinlay5/finalize-

finalize- - Gradle source code repository for the
java.lang.Object.finalize() Java 11 examples on...
[github.com](#)



Goodbye, finalize(). You will not be missed.

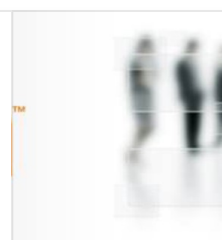


Interested in Java? Join my Java group on Facebook:

Join My Java Facebook Group

Interested in Java? Check out my Facebook Group:
Java Software Development Group!

[medium.com](#)



Like my Content? Subscribe to my mailing list:

This embedded content is from a site that does not comply with the Do Not Track (DNT) setting now enabled on your browser.

Please note, if you click through and view it anyway, you may be tracked by the website hosting the embed.

[Learn More about Medium's DNT policy](#)

Don't forget to give it a.... ;)



!Emoji.com

Works Cited

[1] —[JDK 9 API Documentation: java.lang.Object.finalize\(\)](#)



HOW HACKERS START THEIR AFTERNOON

[READ TODAY'S TOP STORIES](#)

