implements Closeable or implements AutoCloseable

I'm in the process of learning Java and I cannot find any good explanation on the `implements Closeable` and the `implements AutoCloseable` interfaces.

When I implemented an `interface Closeable` , my Eclipse IDE created a method `public void close() throws IOException` .

I can close the stream using `pw.close();` without the interface. But, I cannot understand how I can implement the `close()` method using the interface. And, what is the purpose of this interface?

Also I would like to know: how can I check if `IOstream` was really closed?

I was using the basic code below

```
import java.io.*;

public class IOtest implements AutoCloseable {

public static void main(String[] args) throws IOException  {

    File file = new File("C:\\test.txt");
    PrintWriter pw = new PrintWriter(file);

    System.out.println("file has been created");

    pw.println("file has been created");

}

@Override
public void close() throws IOException {


}
```

java java-io

edited Aug 29 '16 at 17:56
Kirby
5,636 5 38 55

asked Oct 30 '12 at 14:36
malas
263 1 4 5

1 I think all has already been said, but maybe you are interested in the following article about try on ressources: docs.oracle.com/javase/tutorial/essential/exceptions/… . This might also be helpful to understand the given answers. – crusam Oct 30 '12 at 14:51

4 Answers

It seems to me that you are not very familiar with interfaces. In the code you have posted you don't need to implement `AutoCloseable` .

You will only have to (or should) implement `Closeable` or `AutoCloseable` if you are about to implement an own `PrintWriter` which handles with files or any other resources which needs to

be closed.

In your implementation it is enough to call `pw.close()` . You should do this in a finally block:

```
PrintWriter pw = null;
try {
   File file = new File("C:\\test.txt");
   pw = new PrintWriter(file);
} catch (IOException e) {
   System.out.println("bad things happen");
} finally {
   if (pw != null) {
      try {
         pw.close();
      } catch (IOException e) {
      }
   }
}
```

The code above is Java 6 related. In Java 7 this can be done more elegant (see this answer).

edited Mar 15 '16 at 11:50 — Arnaud — 1,413 2 14 40

answered Oct 30 '12 at 14:45 — Kai — 22.6k 8 58 86

1 Why only with a `PrintWriter` ? Especially `AutoClosable` objects can be used in many more circumstances than just `PrintWriter` s... – glglgl Jan 26 '15 at 14:53

You are absolutely right. The question was about `PrintWriter` so I mentioned it to be more specific. – Kai Jan 26 '15 at 19:48

This is SO cumbersome, but it works. – Josh Dec 7 '15 at 15:06

`AutoCloseable` (introduced in Java 7) makes it possible to use try-with-resources idiom:

```
public class MyResource implements AutoCloseable {

    public void close() throws Exception {
        System.out.println("Closing!");
    }

}
```

now you can say:

```
try(MyResource res = new MyResource()) {
    //use res here
}
```

and JVM will call `close()` automatically for you. `Closeable` is an older interface. ~~For some reason~~ To preserve backward compatibility language designers decided to create a separate one. This way not only all `Closeable` classes (like streams throwing `IOException`) can be used in try-with-resources but also those that wish throwing more general checked exception from `close()` .

When in doubt, use `AutoCloseable` , users of your class will be grateful.

edited Oct 30 '12 at 15:29

answered Oct 30 '12 at 14:41 — Tomasz Nurkiewicz — 215k 34 494 531

48 The reason is simple: `Closeable.close()` throws `IOException` . A lot of `close()` methods that could benefit of try-with-resources throw other checked exceptions (eg `java.sql.Connection.close()` so `AutoCloseable.close()` throws `Exception` . Changing the existing `Closeable` contract would break all existing applications/library relying on the contract that `close()` only throws `IOException` and not all (checked) exceptions. – Mark Rotteveel Oct 30 '12 at 14:49

3 @MarkRotteveel: +1, thanks. I corrected my answer to reflect your suggestions and comments. – Tomasz Nurkiewicz Oct 30 '12 at 14:57

5 And also: `Closeable.close()` is required to be idempotent. `AutoCloseable.close()` is not, although it is still strongly recommended. – Lukas Eder Jan 25 '14 at 17:00

1 Also, don't use the default `public void close( ) throws Exception` -- use a more specific exception if you can (e..g IOException) – gerardw Nov 15 '14 at 20:23

Except this breaks backwards compatibility, because things that used to be Closable (like `javax.sql.Connection`) are now *only* AutoClosable, so won't work with libraries (like commons-io) anymore, because they only support Closable (Because they need to compile with non-Java 7 JDK) . – Mikkel Løkke Feb 20 '15 at 11:46

`Closeable` extends `AutoCloseable` , and is specifically dedicated to IO streams: it throws IOException instead of Exception, and is idempotent, whereas AutoCloseable doesn't provide this guarantee.

This is all explained in the javadoc of both interfaces.

Implementing AutoCloseable (or Closeable) allows a class to be used as a resource of the try-with-resources construct introduced in Java 7, which allows closing such resources automatically at the end of a block, without having to add a finally block which closes the resource explicitely.

Your class doesn't represent a closeable resource, and there's absolutely no point in implementing this interface: an IOTest can't be closed. It shouldn't even be possible to instantiate it, since it doesn't have any instance method. Remember that implementing an interface means that there is a *is-a* relationship between the class and the interface. You have no such relationship here.

edited May 30 '15 at 3:41

rmtheis
3,783 8 32 56

answered Oct 30 '12 at 14:46

JB Nizet
421k 33 583 756

i keep coming across your answers (you seem to be omnipresent like Jon Skeet) and they're spot-on! i will be copy-pasting your first paragraph into my notes for the OCP exam i am taking next Monday. – Peter Perháč Nov 27 '13 at 11:19

Just implement **Closable** for streams-related classes, and **AutoClosable** for others which requires autoclose feature. – lospejos Jul 13 '16 at 15:13

Do 1 one thing!wite an o/p console stateement explicitly inside the close method.in another class do try-with-resource i.e make an instance of yours resource and put inside the try-with-resource statement clause.you will notice that after the execution of the resource is automatically closed!!the statement is printed!

answered Jul 14 '14 at 17:15

user3201264
22 8