Check if a file exists before calling openFileInput

Ask Question

21

10

To read a file in Android from your app's private storage area you use the function `openFileInput()` .

My question is, is there a way to check if this file exists before calling this function? The function can throw a `FileNotFoundException` , but I feel like calling this and then doing something based on a `try` - `catch` is a bad practice.

Using `File.exist()` seems like a strange thing to use also since it would require instantiating a class and I am not sure if just passing the name of the file to it would get it to find the file in the private area of my phone.

java android file file-exists

edited Aug 9 '17 at 17:47

Alexander Abakumov

5,089 5 48 75

asked Jan 15 '12 at 3:28

Pieces

1,507 4 18 39

3 Answers

```
public boolean fileExists(Context context, String filename
```

```
        }
        return true;
    }
```

EDIT:

Also, here is another way for files in external storage.

```
String fileUrl = "/appname/data.xml";
String file = android.os.Environment.getExternalStorageDir
File f = new File(file);

if(f.exists())
return;
```

edited Dec 7 '14 at 18:59

Michael Celey

10.8k 4 48 57

answered Jan 15 '12 at 3:35

coder_For_Life22

17.3k 19 75 114

1 Note: using File.exists() in this way 1) doesn't guard against all exceptions, 2) leaves you open to race conditions, and 3) makes you program slower on average ... unless non-existent files are a *likely* scenario. – Stephen C Sep 19 '12 at 2:35

1 i prefer the first method, as it keeps the opacity as to where the file actually is – njzk2 Jan 14 '13 at 16:57

30

The function can through a FileNotFoundException, but I feel like calling this and then doing something based on a try catch is bad practice.

I disagree. IMO, it is testing if the file exists before opening that is bad practice. Compare these two versions of the code:

```
File f = new File("someFile");
InputStream is;
```

Version #1

```
if (f.exists()) {
    is = new FileInputStream(f);
    ...
} else {
    System.err.println("Doesn't exist");
}
```

Version #2

```
try {
    is = new FileInputStream(f);
    ...
} catch (FileNotFoundException ex) {
    System.err.println("Doesn't exist");
}
```

There a number of problems with the first version:

- Version #1 makes an extra system call when you call `f.exists()` . This makes the first version slower on average, unless there is a high probability that the file *won't* exist.
- Version #1 has a race condition. If some external process deletes the file at roughly the same time, you could end up with `file.exists()` returning true, and then the `FileInputStream` constructor throwing `FileNotFoundException` . This is the kind of race condition that can be exploited to break security if the

created and `file.exists()` returns false when a subsequent attempt to open it would succeed. But that race condition is probably harmless.)

A further problem is that the `FileInputStream` is declared as throwing `IOException`. Testing to see if the file exists only deals with one of the possible failure modes. Your code is going to have to catch and deal with those other `IOException`s anyway.

@Pieces commented:

> Exceptions should be for when something actually goes wrong that you have no control over. In this case, I have complete control over it.

Actually, you don't have complete control over it. Certainly not in the general case. Even in your particular use case, the race condition is still possible in theory.

But the real problem with this line of thinking is that you end up jumping through hoops to avoid exceptions in situations where exceptions / exception handling are the *best* solution. This makes the code more complicated, less readable, and potentially slower and/or more fragile.

The normal dogma goes like this:

> *"Exceptions should only be used in exceptional situations"*.

This is not the same as saying what you said. The word "exceptional" really just means "not normal". This has a much broader meaning than "something actually goes wrong that you have no control over".

- Exceptions *should not* be used for normal flow control.
- Exceptions *should not* be used if they are going to prove too expensive *on average*.
- Exceptions *should* be used if the tests you would use to avoid them are not reliable.
- Exceptions *should* be used if the tests you would use to avoid them are too expensive *on average*.
- Exceptions *should* be used if they significantly simplify your code (modulo the above). And the criterion for simplicity is whether the code is readable by an average Java programmer.

(Note - "on average" and "*too* expensive" ...)

Now one can argue until the cows come home about *how exceptional* an event needs to be, but my take is that this is really a matter of balancing the relative simplicity of the approaches (in the context) versus the average performance costs (in the context). Any dogmatic rule that doesn't account for the trade-offs and the context is going to do you harm in some cases.

edited Jun 28 '13 at 7:22

answered Jan 15 '12 at 7:13

Stephen C

532k 72 592 954

2 If the file does not exist, probably because it is the first time the user has opened the app, I want to call a function that writes it with the default values (maybe I should have noted that in my question). What you are suggesting is what I would consider programming by exceptions. Exceptions should be for when something actually goes wrong that you have no control over.

race conditions if any occur, which would be a case that I don't have control over. – Pieces Jan 16 '12 at 15:40

This answer is exceptional. ;) However, in the case that the file not being found is not an exception, i.e. you expect there not to be a file sometimes (and, especially if you *only* do something when the file is *not* found), the other answer actually answers the question. You both get +1. – AlbeyAmakiir Sep 19 '12 at 2:31

@AlbeyAmakiir - The problem is that the solution proposed in the other Answer is unreliable. – Stephen C Mar 29 '13 at 13:53

1 This is not a good answer, for two reasons. 1st - the user is talking about private file storage, scoped to their application. So gold-plating safeguards against other application's accessing it is unnecessary, and just overcomplicates the issue. 2nd, as another commenter points out, there are plenty of perfectly valid reasons why a file might not yet exist. For example, if you are writing a game, you want to know if the save file exists, which for every new player it won't the first time they run the game. Completely legit. – user3690202 Jul 26 '15 at 9:44

Maybe. But the "gold plating" you refer to is simply using a `try ... catch` . Hardly seems like "gold plating" to me. Besides, you seem to be ignoring the race condition, and the bug and possibly the crash that results from it. (Depending on what you do about the exception ...) – Stephen C Jul 26 '15 at 10:44

0

This work for me.

```
try {
  FileInputStream fis = openFileInput(String filename);
  // ... do something
try {
    fis.close();
```

```
    }
  } catch (FileNotFoundException e) {
    e.printStackTrace();
  }
```

but happens, some times, it returns an Exception the same ... working with adb.

answered Feb 29 '16 at 16:55

rfellons

616 12 23