NOTE: SCENARIO IS WHAT THE PUZZLE TAKER SEES.

SCENARIO:

You are developing a library for file operations (e.g., copy, move). For the move operation, you write a static method that takes three arguments: `File source` (the path of an existing file), `File destination` (the path of destination file with an existing parent directory), `boolean overwrite` (whether the `move` method should overwrite the existing destination file), and returns a `boolean`: `true` if the operation succeeded, `false` otherwise. Consider the snippet of code below and answer the following questions, assuming that the code has all required permissions to execute.

```
01   // OMITTED: Import whatever is needed
02   public final class FileUtils {
03     public static boolean move (File source, File destination,
04         boolean overwrite) {
05       if (overwrite) {
06         // OMITTED: if the file exists, delete it.
07         // Assume the deletion operation succeeds.
08
09         source.renameTo(destination);
10         return true;
11       }
12       else if (!destination.exists()) {
13         source.renameTo(destination);
14         return true;
15       }
16       else {
17         return false;
18       }
19     }
20   }
```

Questions:
1. What will the `move` method do when executed?

2. If a program calls the `move` method with arguments such that `source` exists, `destination` may or may not exist (but the `destination`'s parent directory exists), and `overwrite` is `true`, which one statement is correct?

a. If the move method returns `true`, the source file is moved. Further, if the destination file existed, it has been overwritten.
b. If the destination file existed, it has not been overwritten: the source file is not moved, even if the `move` method returns `true`.
c. If the destination file exists, an exception is thrown and no data is overwritten.

d. The move operation may fail, no matter what the `move` method returns.
e. None of the above.

[Other statistical questions will be imported here while creating survey.]

NOTE: ANSWER IS TO BE SHOWN TO THE PUZZLE TAKER AT THE END OF THE SESSION.

ANSWER:
d
In the implementation above, checking if the destination file exists, deleting it, and moving the source file (lines 6~9) are not atomic (likewise lines 12 and 13). Thus there is a possibility that right after checking for the file's existence, and deleting it if required, but before renaming the file (line 9 or 13), another process/thread creates a file with the same name (of the destination). This means that the rest of the code is about to write data on an existing file. Note that the behavior of the `renameTo` method when the destination file exists is platform-dependent. So that it may or may not succeed to overwrite the destination file. Since the code does not check the return value of the `renameTo` method after the call (line 9 or 13), it is not clear if the file is successfully moved/renamed to the destination or if it failed due to existence of the destination. After calling the `renameTo` method, the return value must be checked to see if the file is moved to the destination or not.

NOTE: THE REST OF THIS DOCUMENT CONTAINS EXTRA INFORMATION FOR THE PROJECT RESEARCHERS. IT IS NOT TO BE SHOWN TO THE PUZZLE TAKERS.

TAGS:
java, file-class, io-operation, file-operation, ignoring-return-value, deprecated-api

CATEGORIES:
Blindspot - YES
Type - File
Number of distinct functions - 6
Number of total functions - 9
Blindspot function - `File.renameTo()`
Blindspot type - Return value ignored
Number of Parameters in the blindspot function - 1 parameters
Cyclomatic complexity - 7

NAME:
File class, renameTo method - An abstract representation of file and directory pathnames.

DESCRIPTION:

Operating systems use system-dependent pathname strings to name files and directories. The Java `File` class is an abstract representation of file and directory pathnames. This class presents an abstract, system-independent view of hierarchical pathnames and is used for various operations, such as creating and deleting files and directories. Some of the operations throw exceptions in case of failure, while others do not. The `renameTo` method returns a `boolean` value; `true` if and only if the operation succeeded, `false` otherwise.

BLINDSPOT:
The platform-dependent behaviors of some methods may cause a blindspot. As an example when the `renameTo` method is called, whether the operation succeeded to overwrite the destination file or not depends on the execution environment. The only way to see if the operation failed or not is to check the return value of the method call. For example, if the call of the `renameTo` method returns `false`, that means the file could not be moved because, for example, a file with the same pathname already exists and the method does not support overwriting in this specific execution environment.

CORRECT USE EXAMPLE:

```
import java.io.File;
public class Application {
    public static void main ( String... args ) {
        File foo = new File("foo.txt");
        File bar = new File("bar.txt");

        if (foo.renameTo(bar))
            System.out.println("File is renamed!");
        else
            System.out.println("File cannot be renamed!");
    }
}
```

MORE INFORMATION:
To see the correct way to use the API, look at the JX02-File.renameTo puzzle.

REFERENCES:
1. File#renameTo