NOTE: SCENARIO IS WHAT THE PUZZLE TAKER SEES.

SCENARIO:

You are developing a library for file operations (e.g., copy, move). For the copy operation, you write a method that takes three arguments: a source file's path, a destination file's path, and a `boolean` flag indicating if the destination file should be overwritten. The following is an implementation of the `copy` method. Consider the snippet of code below and answer the following questions, assuming that the code has all required permissions to execute.

```java
// OMITTED: Import whatever is needed.
public final class FileUtils {
  /**
   * This method copies the contents of the specified source file
   * to the specified destination file. The source file and the
   * directory holding the destination file have to exist.
   * If 'overwrite' argument is true, then the method will
   * overwrite the destination file if it exists.
   *
   * @param source
   *          the path to an existing file
   * @param destination
   *          the path of a destination file
   * @param overwrite
   *          whether the copy method should overwrite existing
   *          destination file
   * @return true, if the operation succeeded, otherwise false
   * @throws NullPointerException if source or destination is null
   *         IOException if an I/O error occurs
   */
  public static boolean copy (File source, File destination,
      boolean overwrite) throws IOException {
    InputStream input = null;
    OutputStream output = null;
    try {
      if (!overwrite && destination.exists())
        return false;

      if (!destination.exists())
        destination.createNewFile();

      input = new FileInputStream(source);
      output = new FileOutputStream(destination);

      // OMITTED: By using 'input' and 'output', read all bytes
      // from 'source' and write them into 'destination'. At the
      // end, 'source' and 'destination' contain the same data.
38
39         return true;
40        }
41        // OMITTED: In finally block, close all streams.
42       }
43      }
```

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

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

a. No existing file is overwritten.
b. If the destination file exists, an exception is thrown and no data is overwritten.
c. If the destination file exists, the method fails to prevent overwriting the existing destination file. However, it is possible to implement a method such that the file is not overwritten.
d. An implementation that guarantees existing files will not be overwritten is impossible in Java.
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:
1. The method makes a copy of a file by copying its content to a new file, or by overwriting its content to an existing file, if `overwrite = true`.
2. c
In the implementation above, checking if the file exists and creating the file (lines 29 & 30) are not atomic. Since the code does not check the return value of the `createNewFile` method after the call (line 30), it is not clear if the file is newly created or if it existed before the call. So, there is a possibility that right after checking for the file's existence (line 29), but before creating the file (line 30), another process/thread creates a file with the same name. This means that the rest of the code is about to write data on an existing file, even though the overwrite flag is set to `false`.

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, old-api

CATEGORIES:
Blindspot - YES
Type - File
Number of distinct functions - 4
Number of total functions - 5
Blindspot function - `File.createNewFile()`
Function call omitted - NO
Blindspot type - Return type ignored
Number of Parameters in the blindspot function - 0 parameters
Cyclomatic complexity - 3

NAME:
File class, createNewFile method - An abstract representation of file or directory pathnames, creates a new file.

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 `createNewFile` method returns `true` if the named file does not exist and is successfully created: `false` if the named file already exists. It may also throw `IOException` if an I/O error occurs.

BLINDSPOT:
In the implementation above, checking on a file's existence and creating the file (lines 29 & 30) are not atomic. Since the code does not check the return value of the `createNewFile` method after the method call statement (line 30), it is not clear if the file is newly created or it existed before the method call. So, there is a possibility that right after checking for the file's existence (line 29), but before creating the file (line 30), another process/thread creates a file with the same name, meaning that the rest of the code is about to write data on an existing file, even though the overwrite flag is set to `false`.

The method reporting some errors via exceptions and others via a return value may cause a blindspot. When the `createNewFile` method is called, the return value specifies whether the method has created a new file, or found an existing file with the same pathname. For example, if the call of the `createNewFile` method returns `false`, that means the file could not be created because a file with the same pathname already exists. Any attempt to write on such a file destroys the current content of the file and then writes new data over it.

CORRECT USE EXAMPLE:

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

        if (foo.createNewFile())
            System.out.println("The new file is created!");
        else
            System.out.println("The file already exists!");
    }
}
```

MORE INFORMATION:
To see the correct way to use the API, look at the JX01-File.createNewFile puzzle.

REFERENCES:

1. File#createNewFile