# CS 273 Laboratory 12B: File I/O

This lab gives you experience with reading, writing, and manipulating files. It has a total of 50 points.

### Overview

Examine the file FileHandler.java, which defines the class FileHandler. This class contains one piece of state: a string that gives a message about the previous file operation that it performed. It defines seven (presently dummied-up) file-manipulation methods:

* createEmptyFile - creates an empty file
* deleteFile - deletes an empty file
* countBytes - counts the number of bytes in a file
* listContents - lists the contents of a file
* copyFile - makes a copy of a file
* copyFileInvertCase - makes a copy of a file, but upper-case letters changed to lower-case, and vice versa.
* appendString - appends text to an existing file

Most of the above methods set the lastMessage instance variable to a value that either

* reports the result of the operation (e.g., whether it was successful), or
* contains the data that was requested by the user.

The instance variable lastMessage is declared as private, but other classes can get its contents by invoking the getMessage operation.

The FileFrame class provides a graphical user interface (GUI) that allows you to test these operations. The GUI consists of the following parts:

* a text-field where an input file can be specified, via a Browse... button.
* a text-field where an output file can be specified, via a Browse... button.
* one button for each method you’ll be implementing
* a text-field for typing a string to append to a file
* a message area that displays the message from the most recent operation (i.e., the result of calling getMessage on its FileHander object).
* a text-field that is labeled Call:, which displays the most recent file-manipulation method call.

Each operation requires that either the *input file* or the *output file* field (or both) be filled in. If you attempt to apply the operation (by pressing a button), and the appropriate field has not been set, an error message will appear in the message area.

In its present state, all the operations result in a "... not implemented" message being displayed in the message area. This is because the methods in FileHandler.java are dummied up.

Your job will be to implement methods in FileHandler.java so that they behave correctly.

Compile FileHandler and run FileFrame (using void main String[]) so that you can see what this looks like.

### Part 1: Implement the createEmptyFile method

Edit FileHandler.java so that the createEmptyFile method creates a file that contains zero bytes, whose path/name is given by its String parameter. (This is taken directly from the Output file: text-field of the FileFrame interface.) You should be able to do this by:

1. creating an instance of type FileOutputStream class. (This automatically opens the file for writing.)
2. immediately closing the file.

If you need more details on the FileOutputStream class, refer to the online Java documentation.

The above operations should be "wrapped" in a try clause. If an error occurs, the lastMessage instance variable should be set to a String that reports the type of error that occurred; if no error occurs, the lastMessage instance variable should be set to a String that reports that the file was created successfully***. Do not print the error message to the console***. Do not use the generic Exception type to catch the exceptions. Instead, catch the specific type of exception that could be thrown. That means you may have to use multiple catch blocks. If you don’t know why kind of exception could be thrown, refer to the online Java documentation.

To test that your method works correctly, do the following:

1. Press the Browse button for the Output File. Select the folder where you want to create the file. The Browse button will open the previously browsed folder on your system, so the first time you run the program, you will need to navigate to your lab12B folder. Type a name for a file that does not already exist in your chosen directory. **Note:** You cannot type a filename directly into the output field. It must be typed in via the file selection dialog window that appears when you click the Browse button.
2. Press the “Create empty file” button. You should see in File Explorer (or Finder in MacOs) that the file has been created and has size 0.

You should also test your method works when unsuccessful. To do this, select an existing directory as the name of the file (i.e. the Backup directory in your lab12B directory). You should be able to test the file not found error, but you don’t need to test the I/O error (that’s harder to simulate).

**checkpoint 1 (10 points): Show your lab instructor or assistant your code and the executing program. You should be able to use the File Explorer (or Finder in MacOs) to verify that the file is in the directory, and that its length is zero. Also demonstrate that it has appropriate behavior when unsuccessful.**

### Part 2: Implement the deleteFile method

Edit FileHandler.java so that the deleteFile method deletes the file that is named by its String parameter. (This is taken directly from the Output file: text-field.) Do this using Java’s File class. The lastMessage instance variable should be set so that it reports whether the deletion was successful. You should be able to do this by:

1. Creating an instance of type File class.
2. Using the instance to delete the file.

If you need more details on the File class, refer to the online Java documentation.

To test that your method works correctly, do the following:

1. From your Java GUI, select deleteThisFileFirst.txt in your lab12B directory as your Output file. Click “Delete File”. Confirm that the file has disappeared from the File Explorer (or Finder in MacOs).
2. From your Java GUI, create an empty file. Then press the Delete File button to delete it. Press the Delete File button again to attempt to delete it for a second time. The second deletion attempt should result in an error message. If it does not, you may not have properly taken advantage of the boolean value returned from the delete() method.
3. Attempt to delete a directory (e.g., Backup) that has something inside of it; this should be unsuccessful, but it is not expected to throw an exception.

**checkpoint 2 (10 points): Show your lab instructor or assistant your code and the executing program. You should be able to use File Explorer (or Finder in MacOs) to verify that the file is actually deleted. Also demonstrate that it behaves correctly when unsuccessful.**

### Part 3: Implement the listContents method

Edit FileHandler.java so that the listContents method sets the lastMessage variable to be a String whose contents are the contents of the file named by listContents' String parameter. (This is taken directly from the Input file: text-field.) You should be able to do this by:

1. Creating a File object using the fileName passed into listContents.
2. Creating an instance of type Scanner class, but instead of passing in System.in as the argument to the constructor, pass in the File object created above.

If you need more details on the File or Scanner classes, refer to the online Java documentation.

If the operation is successful, the variable lastMessage should be set to a String that gives the contents of the file. If the operation is unsuccessful, the variable lastMessage should be set to an appropriate error message.

It is recommended that you use the hasNextLine and nextLine methods from the Scanner class. However, nextLine will remove the newline character from each line it reads. You’ll have to add it back. The EOL instance variable at the top of FileHandler.java contains this character.

You should close the scanner when you are done reading the lines.

To test that your method works correctly, do the following:

1. From the File Explorer, locate the file myFile.txt in your lab12B directory. From your Java GUI, select this as your Input file. Click “List contents”. Compare the contents of the message window to the contents of the file.
2. Test the “unsuccessful” case. From your Java GUI, select an existing directory (e.g., Backup). Clicking “List contents” should result in an error message.
3. Extra challenge (optional): If you press List Contents repeatedly, does it keep adding the file’s content to the textbox? If so, can you update listContents so that it shows only the contents of the file once, no matter how many times you press it?

Try it with a couple of other files as well (e.g., a Java file, README.TXT).

**checkpoint 3 (15 points): Show your lab instructor or assistant your code and the executing program. You should be able to view the contents of a text file in the message area.**

### Part 4: Implement the appendString method

Edit FileHandler.java so that the appendString method appends a String to the end of the file. (This should increase the size of the file by a few bytes.) The file to be modified is the one that is named by appendString's second String parameter. (This is taken directly from the Output file: text-field.) The String to be appended is given in the first parameter.

1. Open the file for reading, using an instance of the FileOutputStream class in “append mode”.
2. Create an instance of type PrintStream using the FileOutputStream object as its source.
3. Write to the file using the println() method of the PrintStream class.

If you need more details on the FileOutputStream or PrintStream classes, refer to the online Java documentation.

The variable lastMessage should be set to a String that gives a message that tells whether the operation is successful.

**checkpoint 4 (15 points): Show your lab instructor or assistant your code and the executing program. You should be able to create an empty file (createEmptyFile), append strings to it, and then display it using listContents.**

### EXTRA CREDIT (optional)

### You can do any of the following for extra credit, up to 10 points total.

### EC1: Implement the countBytes method

Edit FileHandler.java so that the countBytes method counts the number of bytes in the file that is given by its String parameter. (This is taken directly from the Input file: text-field.) You could do this by creating a File object and calling its length() method, but that’s too easy, and you need some practice reading a file. Instead, you should be able to do this by:

1. Opening the file for reading, using an instance of the FileInputStream class.
2. Successively reading one byte at a time from the file until there are no more bytes. Keep count of how many bytes you read as you do this. Note, you do not have to store the bytes in a byte variable because the type returned from read() is int.
3. Return your count to the caller.

Use the file myfile.txt in your lab12B directory as an initial test case. As the properties of the file will tell you, the number of bytes in the file is 67. Run a test case on an empty file too.

If you need more details on the FileInputStream class, refer to the online Java documentation.

**checkpoint EC1 (5 points): Show your lab instructor or assistant your code and the executing program. You should be able to check the properties of a file through File Explorer to verify that byte-lengths of files are reported correctly.**

### EC2: Implement the copyFile method

Edit FileHandler.java so that the copyFile method creates a file with a name given by its *second* parameter (taken from Output file) and copies all the data into it from the file that is named by its *first* parameter (taken from Input file:). When the operation is complete, the contents of the two files should be identical.

You should be able to pattern this after the countBytes method, except that:

* The copyFile method should not allow a file to be copied onto itself because this may result in the file getting truncated to zero bytes. If both file-strings are identical, don't perform the operation; instead, set lastMessage to an appropriate error message and return.
* you open a second file (for output) using FileOutputStream
* instead of just counting the bytes, you write it to the output file using the method write(byte b). If you need more information about the FileOutputStream class, see the online Java documentation.

The lastMessage variable should be set to a message that indicates whether the operation was successful.

For a test case, use myFile.txt as an input file and another file as output. You can then use a text editor (e.g., Notepad) to examine the new file.

### checkpoint EC2 (5 points): Show your lab instructor or assistant your code and the executing program. The second file should be created and have contents that are identical to the first. Also demonstrate the scenario where the input and output files are the same.

### EC3: Implement the copyFileInvertCase method

Edit FileHandler.java so that the copyFileInvertCase method creates a file with a name given by its *second* parameter (taken from Output file) and copies all the data into it (with all letters case-inverted) from the file that is named by its *first* parameter (taken from Input file:). When the operation is over, the contents of the two files should be identical, except that any character that is a letter (a-z, A-Z) has its case changed. For example, “Where am I going, George?” would become “wHERE AM i GOING, gEORGE?”. Other than case-conversion, this method should have the same behavior as the copyFile method.

You may want to examine the Java API documentation in the Character class for methods that do case-testing and case-conversion. Note the following:

* 1. You can type cast the value returned from the FileInputStream read() method to a char and use that to create a new Character object.
  2. Most of the methods in the Character class are static, which means you have to call them like so: Character.methodName() and pass in any arguments as applicable. The names of the methods you will use are like those in the String class, but you will have to use them differently because they are static in the Character class and non-static in the String class.

The lastMessage variable should be set to a message that indicates whether the operation was successful.

**checkpoint EC3 (5 points): Show your lab instructor or assistant your code and the executing program. The second file should be created and have contents that are identical to the first except with inverted case.**