

## Section 4: Files

For the last time, you are to pair up with a new student, someone you have not paired with before. Next week, and thereafter, you can pair up with anyone who is agreeable to pairing up with you. Decide who will be the first driver, and let's get started.

### PART I: Writing Objects to a File

*In class Monday, we talked about how to make a Java program write to and read from a text file, and got a good start (I hope!) on writing to and reading from binary files. This part of today's section activity uses an existing example program to explore some details about exceptions and binary files.*

1. Visit the class web page, locate the program named `T03n07.java`, and load it into DrJava.
2. Read the comments and look carefully at the code. The program is storing the same integer value into two different files (`T03n07a.out` and `T03n07b.out`), but in different forms. What kinds (types) of things do the files contain?
3. Now compile and run the program. You'll see two lines of output explaining what the program did.
4. Open another terminal window and use the `ls -l` command to create a 'long listing' of the files in the current directory. Find the files that the program created (`T03n07a.out` and `T03n07b.out`).

(If you don't see them, it's probably because DrJava saved them in a different directory in your account. You need to look in the same directory you were in when you launched DrJava.)

5. In the file list, you'll see a column of integers, just ahead of the dates. These values are the sizes of the files, in bytes. How many times bigger is `T03n07b.out` than is `T03n07a.out`?

*(The lesson: Don't write objects to a binary file unless you have to!)*

6. Now let's address some of the problems with the program itself. Start by reading the error message printed in the `catch()` blocks.
7. Let's help out that lazy programmer by showing him/her how to catch exceptions correctly. (We'll just fix the first `try` block; the lazy programmer can fix the second one himself or herself.) Use your browser to find the Java API, and look up the three constructors and two methods used in that first `try` block (we've listed them below). What are the exceptions that each can throw, and which of those are checked exceptions?


- (a) `File (String)`
- (b) `FileOutputStream(File)`
- (c) `DataOutputStream(OutputStream)`
- (d) `writeInt(int)`
- (e) `close()`

(Continued ...)

8. The lazy programmer should have used a `catch` for each of the checked exceptions, so that appropriate messages can be displayed for each. Here's what you need to do:
  - (a) Remove the existing `catch` structure (again, just the first one).
  - (b) Add multiple catch blocks, one per type of checked exception that method calls in that block of code can throw. Use the following abbreviated form, replacing the bracketed parts with the appropriate statements for each exception:
 

```

catch ( [NAME OF THE CHECKED EXCEPTION] e ) {
    System.out.println( [BRIEF MESSAGE EXPLAINING THE SITUATION] );
    e.printStackTrace();
}
          
```
  - (c) Import any additional checked exception classes at the top of the program.
9. Compile and run your program. All of the checked exceptions should be caught, and the program should compile and run. If you have errors, you added those yourselves, and need to fix them before your SL will give you this checkpoint.

 **CHECKPOINT 1** Raise your hand. Your SL will come over and check your `try/catch` blocks.

*This is the end of what I plan to cover about exceptions in this class. It's far from all there is to know about exceptions. If you're curious, Oracle has a nice (long!) sequence of pages with additional information about exceptions: <https://docs.oracle.com/javase/tutorial/essential/exceptions/index.html>*

## PART II: Copying a Text File

*Remember: New part, new driver!*

1. Visit the class web page, find the program `CopyTextFile.java`, and load it into DrJava. `CopyTextFile` will, when complete, accept two file names from the command line, and will copy the content of the first to the second. Your job: Complete the program.
2. You probably already know how a Java program gets command-line arguments, but let's be sure: From DrJava's Interactions Pane, you can run a Java program and give it extra strings of information. Those strings are called "command-line arguments." For example:

```
java CopyTextFile input output
```

The strings "input" and "output" are given to the `main()` method. This is why we declare `main()` with `String [] args` as a formal parameter. `args[0]` references the first command-line argument, `args[1]` the second, etc. If you use DrJava's "Run" button, you can't provide command-line arguments.

Try it now! Type the above command into DrJava's Interactions pane and see what happens. Then try it with less than two command-line arguments.


3. Of course, we don't want to just print the two command-line arguments the program is expecting; instead, we want the local variables `sourceName` and `destinationName` to reference the first and the second command-line arguments, respectively. Replace the `for` loop with appropriate assignment statements.

(Continued ...)

4. Now that the program has the file names, we need to open the first for reading and the second for writing. For this program, we need a `BufferedReader` object for reading (to be referenced by the already-declared variable `inFile`) and a `BufferedWriter` object for writing (`outFile`). Their constructors need to receive a `FileReader` object and a `FileWriter` object, respectively. We learned how to do this in class on Monday. Dig out your notes (or look at Monday's slides) and remind yourselves how to do this.
5. Together, we have four constructors that need to be called, and `FileReader` and `FileWriter` both can throw checked exceptions. As you learned in Part I, we need the following:
  - (a) A `try` block surrounding the calls to the constructors (that is, around the statements that assign a `BufferedReader` object to `inFile` and a `BufferedWriter` object to `outFile`).
  - (b) A `catch()` block to handle `FileReader`'s exception.
  - (c) A `catch()` block to handle `FileWriter`'s exception.

We've supplied the outline of these `try` and `catch()` blocks in the program. Note that you are to replace `Exception` and `Throwable` with the proper exception names, and import them, too. Between Monday's lecture and the exception work you did in Part I, you know what these blocks need to contain; make it happen!

6. Make sure your program compiles and runs. (It won't do much, but it should compile and run.)

 **CHECKPOINT 2** Raise your hand. Your SL will come over and verify that you've completed these first two modifications.

*Switch drivers!*


7. To read the content of `sourceName`, the best way is an infinite `while()` loop, which we've provided. The loop will just read a character and write it, until `sourceName` is exhausted.

We'll use `BufferedReader`'s `read()` method to read characters. Can `read()` throw an exception? Is it a checked exception? If so, you'll need `try/catch` blocks to deal with it. Use the API to answer those two questions, and add the appropriate statements at the top of the `while()` loop to read one character into the variable `currentChar`.
8. We've supplied the `if` statement that will get the program to leave the infinite loop; you should leave it as-is. But, you do need to supply the code that writes the character to `destinationName`. You need to do the same sort of things as you did for reading: Write to the file, and handle any exceptions that could be thrown. What's new is that we haven't told you how to write a character to a text file; you'll have to look up `BufferedWriter` in the Java API to find the appropriate method. Make it happen!
9. Last step! The program needs to close both of the files. From Part I, you know what to do; do it! (It's fine to close both files in the same `try` block.)
10. Make sure your program works. You can use DrJava to create and save a small text file to use as your input file, or you can open a terminal window and use a text editor such as `nano` to create one.

To verify that your output file is the same as the input file, you can use a UNIX utility program called `diff`. Open a terminal window, if necessary `cd` to the directory containing your files, and type this:

```
diff <firstfilename> <secondfilename>
```


with `<firstfilename>` `<secondfilename>` replaced by your input and output file names, of course. If `diff` prints nothing, the files are the same. If they are different, you'll see the lines that aren't the same, and you have some debugging to do.

 **CHECKPOINT 3** Raise your hand. Your SL will come over and verify that you've completed the program.

(Continued ...)

## PART III: Clean Up!

1. Log out of your computer.
2. Pick up your papers, writing implements, cell phones, trash, etc.
3. Push in your chair(s).

 **CHECKPOINT 4** Raise your hand. Your SL will come over and listen for the tiny cries of bacteria dying all around your work area.

You're free to go! But, if you have time, we recommend that you use it to work some more on the programming assignment, if you're not already done.

In fact, if you and your partner are agreeable, you could spend the remaining time getting started on Program #3.