

**CIS 494 – Business Systems Development with Java – Spring 2015**

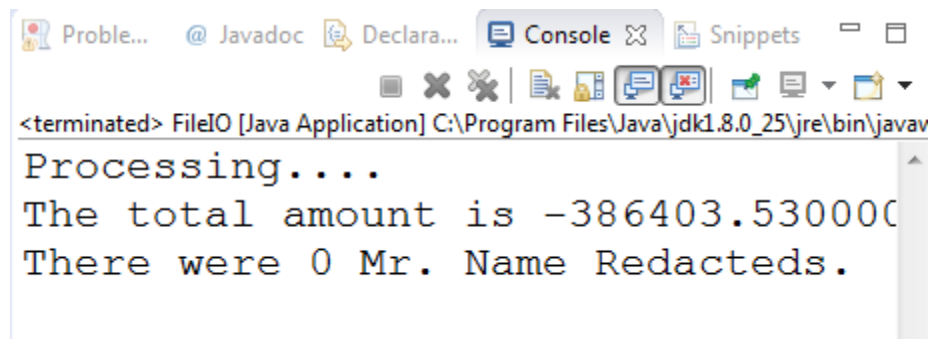
In-class Exercise 4: File I/O

Due on Blackboard: Tonight, February 11 by 11:59 PM

The purpose of this exercise is to gain basic proficiency in reading files and creating logic to process text input.

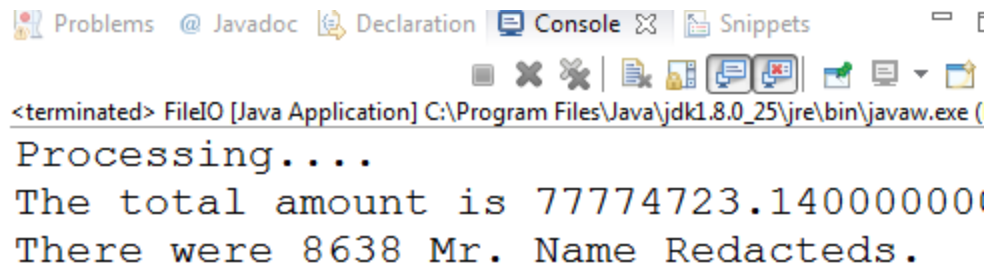
*Sample Output*

Shortened File



```
<terminated> FileIO [Java Application] C:\Program Files\Java\jdk1.8.0_25\jre\bin\javaw.exe  
Processing....  
The total amount is -386403.530000  
There were 0 Mr. Name Redacted.
```

Complete File



```
<terminated> FileIO [Java Application] C:\Program Files\Java\jdk1.8.0_25\jre\bin\javaw.exe  
Processing....  
The total amount is 77774723.14000000  
There were 8638 Mr. Name Redacted.
```

Create a new Eclipse project for Inclass4.

Copy the provided CSV data files in your project folder. These files will go into the root folder of your Eclipse workspace for the project. The files should NOT be put into either the bin or the src folders.

You should work with the shortened CSV file first. Once you are sure that your logic works, you can switch your program to the full version.

*Create variables of appropriate data types as you need to!*

**1) Study the CSV files, which have been downloaded from Arizona's official transparency website, [OpenBooks.az.gov](http://OpenBooks.az.gov).**

See which rows the data is included on. Study all the rows which do not have data, which will need to be excluded. Is there a pattern to knowing which rows are data vs non-data? Note down a precise test which works for ALL lines of data. Informally, scan through the text file and check if your test would work.

**2) Create a data model for Arizona's Revenue Transactions.**

One line represents one entire record, with attributes within a record separated by commas. For this limited exercise, you should track the Entity Name, Payee, Date, and Amount.

**Create a class *Transaction*** with the four attributes listed above. Provide get/set methods.

You will need a *data structure* to store the Transactions. Declare a ***static*** variable called transactionList of type ArrayList. Specify the data type of the ArrayList within <> symbols. For example, ArrayList<Student> is an ArrayList of Students. *Add an import statement for ArrayList using the full package name.*

You will need to create an instance of ArrayList just like arrays and other objects. However, you will not need to provide the constructor with a fixed length.

### 3) Create a method for parsing the file.

Within this method, use the Scanner class to read input from the CSV file.

You will need to declare and instantiate a Scanner object. However, instead of providing the System.in inputStream (which gets input from the Console), you will want to provide the Scanner constructor with an instance of a new **File**. You will need to call the File constructor and provide it with the filename as an argument.

To utilize the AutoCloseable interface, declare and instantiate your Scanner object using a Try-with-resources statement. The scanner object is the resource that will be declared within parentheses after the try keyword. The try block starts after the resources have been declared within parentheses.

You can then use the Scanner class's standard methods to read a line of text, no different than the way you would read it if the input was coming from the Console. Find out which Exceptions the *nextLine()* method *throws*. Find out if it the Exceptions are *checked* or *unchecked* (if an Exception inherits from RuntimeException, then it is *unchecked*.) Checked exceptions must be caught. Catch Exceptions using a catch block.

You can use the hasNext() method of the Scanner to test if there are more lines to be processed. When you have a line of *data* that has been read from the file (you need to test if the line contains data), call your processLine method to process that line of text.

Keep calling nextLine() and processLine() methods *while* there are more lines to be read!

### 4) Create a processLine method that accepts a String as a parameter.

Within the method, create temporary variables for your four attributes.

Use the split method of the String class to split one line. Provide it with a comma as a separator. It will return to you an array of strings, which you should store. Access elements of the array and assign them to your appropriate temporary variables.

While you are assigning values from the array to your temporary variables, within the same line use the replaceAll method of the String class to replace quotes with blank strings. This should only delete the extra quotation marks.

Create a new instance of Transaction and set all four attributes. Add the new instance of Transaction to the transactionList by using the add method of ArrayList.

***5) Calculate total Amount.***

Provide a method for looping through the transactionList and adding up all the amounts. You can loop through an ArrayList just like an array except that you should use the size() method instead of length to get the value of how many elements there are. You can use the get() method of ArrayList to “get” to the element, e.g. arrayList.get(index).

You might find that if you create a double variable to store the total of all the amounts, that data type is too small in size. So, once you encounter the error, you might like to revise that variable data type to a BigDecimal. BigDecimal is a class, so you will need to use constructors and provide it with an argument that should be used as the value. Since BigDecimal is not a primitive type, you cannot use the “+” operator to add. You will need to use its in-built method called add().

***6) Can you find out how many Mr. Name Redacted are listed as the payee?***

## Submitting Files

Submission should be made using a zip file that contains the entire Eclipse project folder. You will need to **zip the entire project folder**. The folder will automatically contain the class source files as well as the compiled .class files.

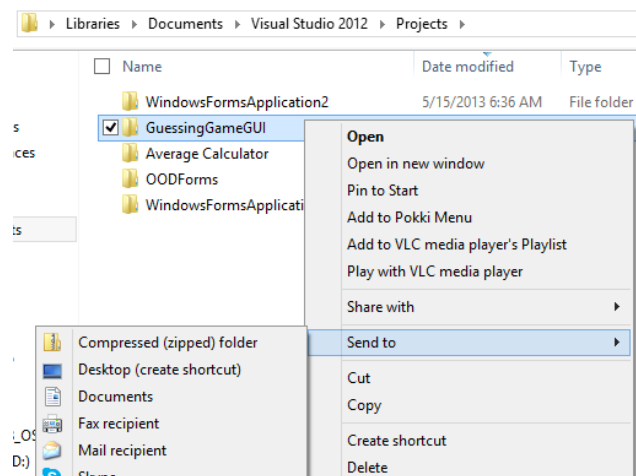
ZIP file should be named: **AX.zip or IC.zip**

where X is the in-class assignment number, e.g., A1.zip is the submission file for Assignment 1 and IC1.zip is the submission file for InClass1.

*Note: The ZIP filename is independent of the Project name. Do not name your project A1.*

## How to Properly ZIP and Submit Your Project Files:

- Go to the folder within {Eclipse Workspace}\
- ZIP the entire top-level folder for your project by right-clicking your project folder and selecting Send to | Compressed (zipped) folder.
- Finally, submit the ZIP file using the submission link on Blackboard by the due-date and time listed on the assignment. Upload the ZIP file.



Using built-in windows zip tools: <http://windows.microsoft.com/en-us/windows/compress-uncompress-files-zip-files>

Verify your files BEFORE and AFTER submission:

- Check for actual class files being present in the folder before you zip it.
- ***Ensure that you are not zipping a short-cut to the folder.***
- After zipping, check file size. A file size under 4K likely does not contain all the files.
- Unzip, extract all files, and verify you see actual files, not a solitary short-cut.
- Uncompress your zip file before submitting and verify that files are present.
- Make sure you have submitted your file and not just saved a draft on Blackboard. ***A blue clock indicates a submission in progress, i.e. a draft, not a submission. The draft is accessible only to you. You will get a ZERO if you only ever save a draft on Blackboard and never submit your files.***
- Download your zip file after submitting, uncompress, and again verify that your files are present. Test your files in Visual Studio after uncompressing.

*This takes an extra couple of minutes. Please do it if your grade is important to you. If you do this, you will not end up submitting a bad file. If you submit an empty file, or one containing only a shortcut, or a bad zip file, or a bad project file, you will receive a score of zero and your only recourse will be to do the makeup assignment at the end of the semester.*