EECS 1021 – Lab E: Reading Text Documents a.k.a. Hydrometric & Air Quality CSV File-Reading

Dr. James Andrew Smith, PEng and Richard Robinson

Summary: In this lab, you will practice methods for reading the contents of real CSV files, whether from Hydrometric stations or Air Quality sensors.

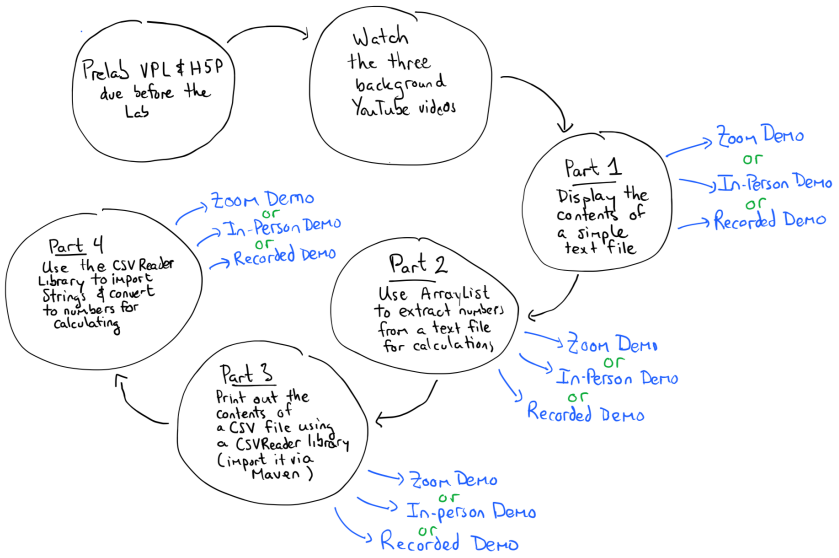


Figure 1 Lab E has four main parts after the pre-lab components. Make sure to do the pre-lab.

Intro

In this lab you will learn how to access files in general, but also Comma-Separated Value text files (a.k.a. CSV files) in particular.

Due dates.

- *Pre-Lab*: All of the interactive pre-lab activities are due on the Sunday night before the lab sessions.
- *Lab Demo*:
  - *Option 1*: live lab demo to the TA (Zoom or in-person)
  - *Option 2*: record a screen capture and submit a video to eClass.

Marking Guide:

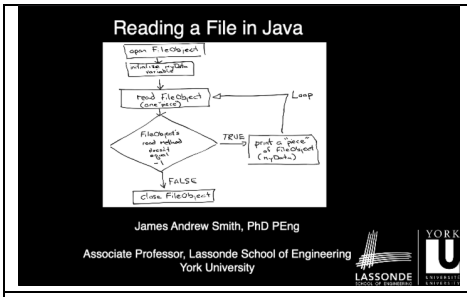
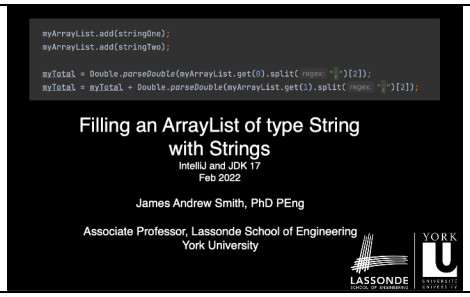
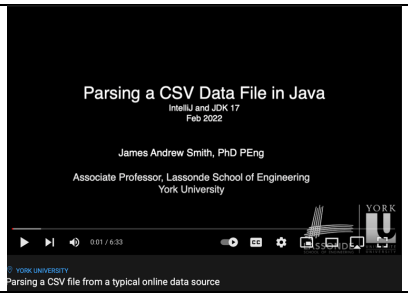
- All interactive Pre-lab activities are graded out of 1 and count towards your "interactive" activity grade. Any other pre-lab activity is not graded.
- Part 1: 0.3 marks. 0.2 if partially successful. 0 if not attempted.
- Part 2: 0.3 marks. 0.2 if partially successful. 0 if not attempted.
- Part 3: 0.2 marks. 0.1 if partially successful. 0 if not attempted.
- Part 4: 0.1 marks. 0.1 if partially successful. 0 if not attempted.
- Skill question: 0.1 marks. 0.05 if partially successful. 0 if wrong or not attempted.

Pre-lab

Check for pre-lab activities in Module 6. These could be either H5P activities or VPL activities or both. All pre-lab activities are due on the Sunday before the lab at 11:55pm.

Key Resources

Videos:

		
Reading a text file in Java (https://youtu.be/0CQXeCVvoC4)	Moving Strings into an ArrayList and parsing out numeric values for the purpose of calculation. https://youtu.be/re7-n3Eq5_0	Reading and parsing a CSV file (https://youtu.be/I0j7CMCXcfI)

Online resources:

1. FileInputStream:
   a. Geeksforgeeks: https://www.geeksforgeeks.org/java-io-fileinputstream-class-java/
   b. JavaTPoint: https://www.javatpoint.com/java-fileinputstream-class
   c. LinkedInLearning: https://www.linkedin.com/learning/advanced-java-programming-2/reading-console-input-with-a-scanner
2. ArrayList
   a. GeeksforGeeks: https://www.geeksforgeeks.org/arraylist-in-java/
   b. JavaTPoint: https://www.javatpoint.com/java-arraylist
   c. LinkedInLearning: https://www.linkedin.com/learning/learning-java-8/using-arraylists
3. Converting a CSV into an ArrayList:
   a. Java67 : https://www.java67.com/2017/09/how-to-convert-comma-separated-string-to-ArrayList-in-java-example.html
4. OpenCSV Library:
   a. Geeksforgeeks: https://www.geeksforgeeks.org/reading-csv-file-java-using-opencsv/
   b. Attacomsian.com: https://attacomsian.com/blog/read-write-csv-files-opencsv
5. Converting a String to an Integer (also works for *Doubles* if you think about it a bit)
   a. Freecodecamp: https://www.freecodecamp.org/news/java-string-to-int-how-to-convert-a-string-to-an-integer/
6. Throws IOException
   a. https://docs.oracle.com/javase/7/docs/api/java/io/FileInputStream.html
7. Try-catch Exceptions with file reading
   a. Jenkov: http://tutorials.jenkov.com/java-exception-handling/try-with-resources.html
   b. Geeksforgeeks: https://www.geeksforgeeks.org/java-io-fileinputstream-class-java/
   c. JavaTPoint: https://www.javatpoint.com/java-fileinputstream-class

Introduction

Climate change has increased the severity of weather and is having a noticeable impact on the bodies of water in and around cities, including Toronto, as well as European cities such as Strasbourg, on the border between France and Germany (where James did his sabbatical in 2018/19; https://bit.ly/3ouiwRu). To better understand the general trends, but also to make decisions on how to regulate the flow of water (in and around our cities, engineers and city officials measure the height and flow ("streamflow") of our rivers using hydrometric sensors such as the ones shown here: https://bit.ly/2Qjn5iY

Figure 2 York's Professor Usman Khan showing us a weir, which can be used to examine streamflow. (c/o Prof. Khan and CIVL 3220 class notes). More on his research here: https://bit.ly/2O7KS2v

Engineers like the Lassonde School's Professor Usman Khan use hydrometric data to examine the state of our watersheds. The information allows informs us about how water flows in and around first nations, cities, towns and rural areas so that we can make engineering and public policy decisions, like where to build dams, as well as how we should attempt to control water flow in our lakes and rivers. (https://bit.ly/2O7KS2v)

		
The Rhine River on the edge of Strasbourg, France[1]	Toronto's Don River @ Todmorden (historical painting)[2]	Attawapiskat First Nation[3]

Figure 3 Most cities and towns are built on bodies of water. Strasbourg, Toronto and Attawapiskat First Nation are examples of these.

In EECS 1021we talk about "computational thinking", using sensors and measurement as a platform to explore the concept. In all engineering disciplines sensors applied to data logging and data analysis applications are ubiquitous. The sensors used in Hydrometric applications are typical of the kinds of sensors found in data logging applications, using SONAR, RADAR or encoders to measure depth, distance and flow. The output of these sensors needs to be processed and analyzed prior to reacting with actuators like the ones found in the dams or lock of our waterways.

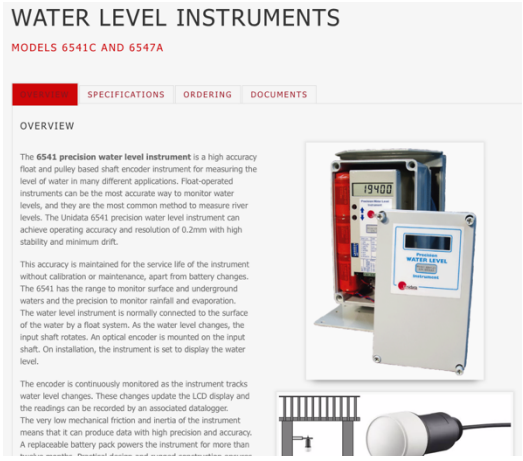


Figure 4 A example of an industrial water depth sensor (c/o https://www.unidata.com.au/products/water-monitoring-modules/precision-water-level-instrument/)

Engineers need to monitor the water levels, sometimes on a daily or hourly basis. This can be done by

- Visiting the water gauge
- Phoning for water gauge information
- Using the internet to monitor live data

In this lab we will explore how to access some of these water monitoring stations remotely, via the internet or other source. This is like we did with MATLAB in EECS 1011, but this time we're doing it with Java.

Two big differences between the MATLAB activity and this one:

1. The data has been pre-processed to make it more straight-forward to parse
2. A second source of data, an air quality sensor, is used for some of the activities to illustrate how alternative data sources could function.

[1] Rhine River photo: James Andrew Smith (http://drsmith.blog.yorku.ca/)
[2] Don River @ Todmorden historical painting (http://citiesintime.ca/toronto/story/remnants-tor/)
[3] Attawapiskat First nation photo c/o Wikimapia (http://photos.wikimapia.org/p/00/02/11/09/06_full.jpeg)

Another Data Source : Carbon Dioxide

In addition to the water monitoring station data, this lab also shows you how to read data from a typical air quality sensor. Air quality sensors can be used to sense things like the level of volatile organic compounds (VOC), particulate matter (e.g. PM2.5) and CO_2 levels. They're similar to the smoke detector in your house but designed to detect other things in the air.

In this lab, we'll present you with data from a CO_2 (Carbon Dioxide) sensor (AirQ). We use these to determine the level of carbon dioxide in the air. Too much carbon dioxide means that the air isn't being refreshed sufficiently when people are present. Too much CO2 means that you won't work as effectively as you should but, in COVID times, it also can indicate that the air isn't be cleansed enough to reduce the risk of transmission.

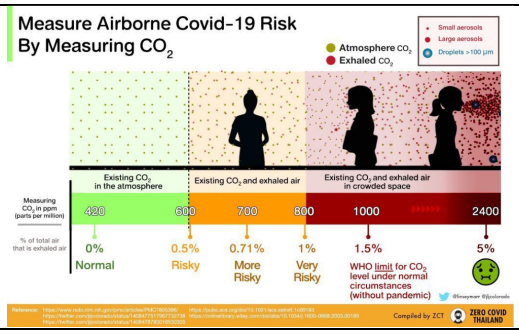
<table>
<tr>
<td> 
</td>
<td> 
</td>
</tr>
<tr>
<td>A pair of air quality sensors that can detect carbon dioxide levels.</td>
<td>In a COVID context, CO2 sensors can be used to determine if the air is fresh. (Source: Zero COVID Thailand et al; see image)</td>
</tr>
</table>

Figure 5 Air quality is an important thing to measure to determine if the air that you're breathing is healthy.

If you're interested in air quality research, check out

- Prof. Marina Freire-Gormaly in YorkU's Mechanical Engineering department (https://lassonde.yorku.ca/users/marina-freire-gormaly)
- Prof. Trevor Vandenboer in YorkU's Department of Chemistry (https://www.tcvandenboer.ca/contact)

Part 1: Display the Contents of a Simple Text File.

Watch the **video** linked to on the right and import a simple, one-line text file into IntelliJ. Call the file "Part1TextFile.txt". You can find it here: https://www.eecs.yorku.ca/~drsmith/eecs1021/labE/Part1TextFile.txt

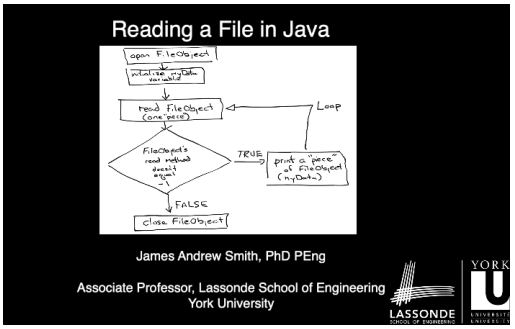
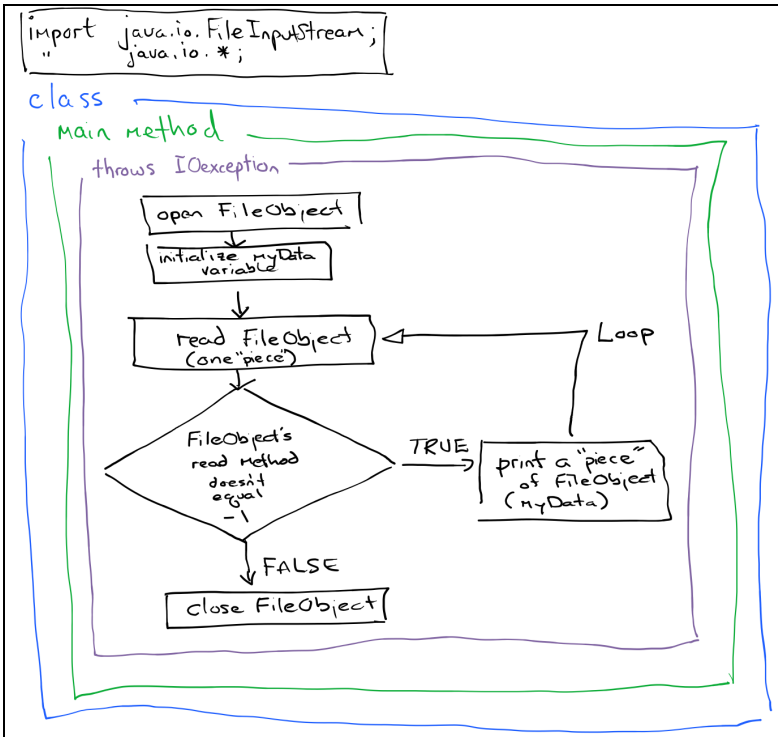


Figure 6 Example of reading a file, either in jShell or in IntelliJ : https://youtu.be/0CQXeCVvoC4

Open a Stream via FileInputStream and print out the contents of that file to your screen. Keep in mind these key concepts in this exercise:

1. Opening and closing file streams
   - Opening: var myFileObject = new FileInputStream("nameOfFile");
   - Closing: myFileObject.close();
   - https://www.linkedin.com/learning/advanced-java-programming-2/reading-console-input-with-a-scanner
2. Looping until the file contents are finished
   - While loop and verify (myData = myFileObject.read()) != -1)
3. Converting integers to characters (i.e. casting to char)
   - ex: var myCharacter = (char) myInteger;
4. Throwing Exceptions
   - Inserting an "throws IOException" command at the beginning of your method.
   - "throws IOException" is a simpler version of a "throw-catch" that you'll see in Parts 2, 3 and 4.

Resources:
• Youtube video: https://youtu.be/0CQXeCVvoC4
• Kishori's book "Lambda Expressions, … , I/O, Collections and Streams" (Chapter 7): https://bit.ly/3H3QgNK



Flowchart

```
import java.io.FileInputStream;
import java.io.*;

public class MyTextFileReader {

   public static void main(String[] args)
   throws IOException
   {
      int myData;

      // open the file here.

      // loop through the data in the while below.
      while(      /* read data & check for end of file */    ){
         // print a character here.
      }


      // close the file here.
   }
}
```

```
MyTextFileReader
/Users/drsmith/Library/Java/JavaVirtualMachines/openjdk-17.0.1/Cor
This is a text file.  It's got stuff in it.
Process finished with exit code 0
```

Skeleton of program & the program output.

Figure 7 The flowchart and skeleton for Part 1. It centers on reading the file, one row at a time until a -1 is returned by the read() method.

The looping through the file can be a bit tricky. You need to call the read() method on your FileInputStream object. Each time the read() method is called it will look at one 8-bit piece of information from the file. You want to keep doing that until there is no more data in the file. When that happens, the read() method will return -1 and you will know that it's time to stop.

Demonstrate this to your TA (Zoom or in-person) or submit a recorded video to eClass (under 1 minute).

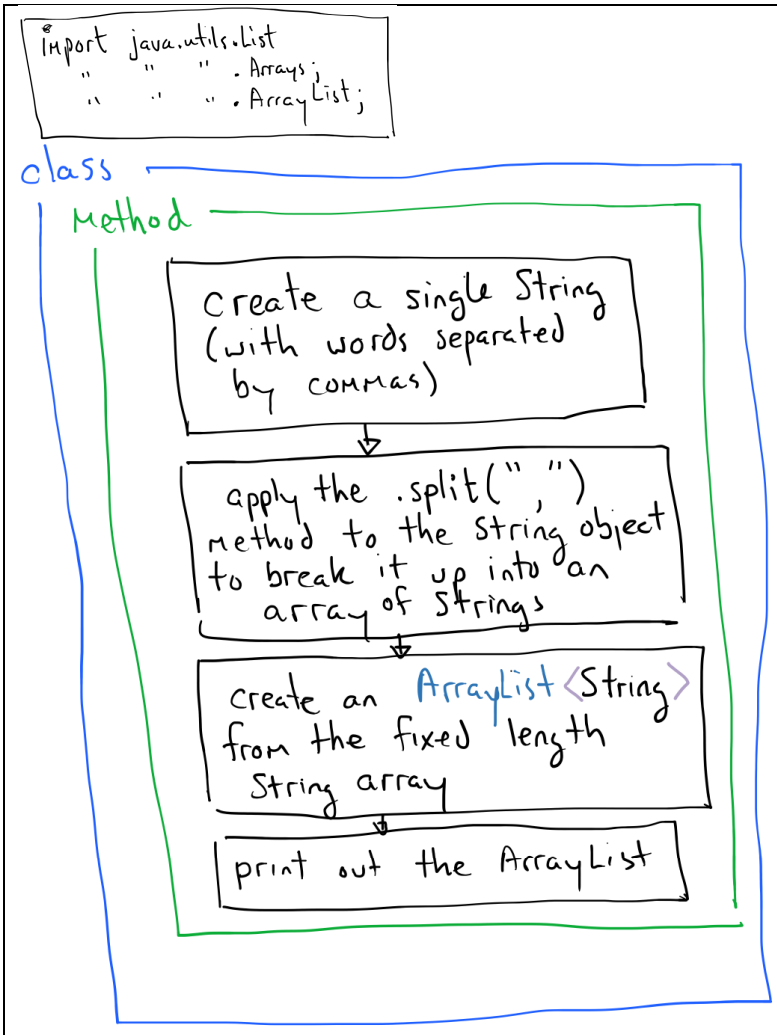
Part 2: Use ArrayList with Funny Angle Brackets to Store Text Data.

Moving information from a text file into your program is important. The most intuitive approach would be to store the file contents into an array in your program. However, even though arrays are great, you typically need to specify how long they are before using them. That's a problem with text files because you typically don't know how long they are. So we'll use ArrayList generics, instead.

I know. This sounds complicated. There's confusing < > "angle brackets". All you need to know, at this stage, is that ArrayList generics use the angle brackets to tell you the type of the data inside of the ArrayList. So, an ArrayList<*String*> means a variable-length Array with String elements in it.

Practice creating an ArrayList<*String*>

Here's an example of splitting a String into parts and then storing each part into an ArrayList<*String*>:



Flowchart for converting a String into an ArrayList<*String*>

```java
package eecs1021;

import java.util.List;  // for lists
import java.util.Arrays; // for arrays
import java.util.ArrayList; // for ArrayList



public class ReadCSVArrayList {
   public static void main(String[] args) {

      // A CSV string.  Usually, this is a file
      // but it's easier to just make a simple String here.
      String csvString = "apples, oranges, pears, plums";

      // step one : converting comma separate String to array of String
      String[] elements = csvString.split(",");

      // step two : convert String array to list of String
      List<String> fixedLengthList = Arrays.asList(elements);

      // step three : copy fixed list to an ArrayList
      ArrayList<String> listOfString = new ArrayList<String>(fixedLengthList);

      System.out.println("list from comma separated String : " + listOfString);
      System.out.println("size of ArrayList : " + listOfString.size());
   }
}
```

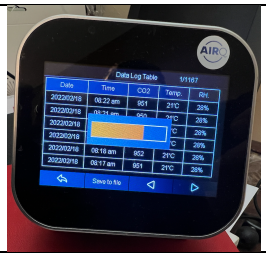
```
ReadCSVArrayList
/Users/drsmith/Library/Java/JavaVirtualMachines/openjdk-17.0.1/Contents/Ho
list from comma separated String : [apples,  oranges,  pears,  plums]
size of ArrayList : 4

Process finished with exit code 0
```

Code implementation: https://bit.ly/3JEp4H6 (Java67)

Part 2 continued: Demo ArrayList<*String*> on a CSV file from an Air Quality Sensor

We're going to break down a comma-separated list in three steps:

1. Split the String using String.split into an array of Strings
2. Convert the String array into a List of Strings, using the Arrays.asList() method.
3. Copy the String array into an ArrayList

			DATALOG_short.TXT cuments/EECS1021/labs/labE/CO2_Sensor_AirQ/D Date, Time, CO2, Temp, RH 2022/02/18, 8:21, 950, 20'C, 29% 2022/02/18, 8:20, 951, 20'C, 29% 2022/02/18, 8:19, 950, 20'C, 29% 2022/02/18, 8:18, 952, 20'C, 29%	Part2_CO2_DataLog_Modified_ drsmith:@indigo1.eecs.yorku.ca/www/eecs10 2,18,104.35,950,20,0.29 2,18,104.3333333,951,20,0.29 2,18,104.3166667,950,20,0.29 2,18,104.3,952,20,0.29 2,18,104.2833333,951,20,0.29
A Carbon Dioxide sensor showing 1000 ppm CO_2 in the room.	CO_2 history is available in a table	Data can saved & downloaded over USB	Data is now a text file, ready to be processed in Excel, Java, etc. However, it's a little complicated to parse.	Cleaned up data into a new file. This is easier for you to process in Java.

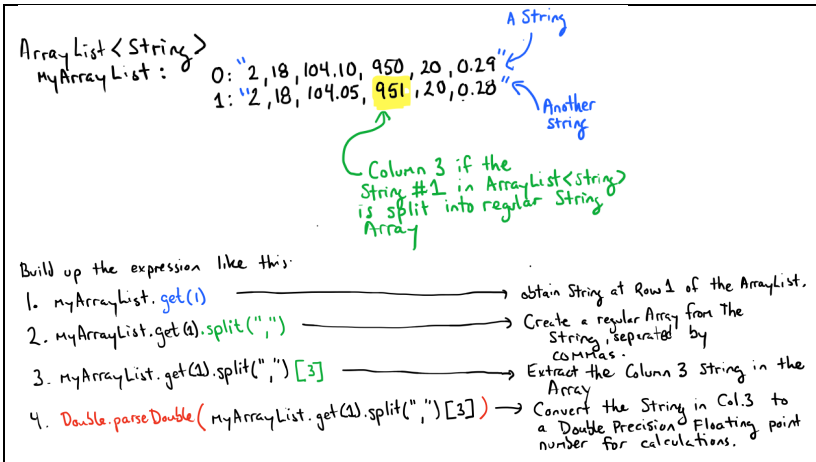
Download the file located here (the "Part 2 data file") and import into your IntelliJ project:
https://www.eecs.yorku.ca/~drsmith/eecs1021/labE/Part2_CO2_DataLog_Modified_short.csv

1. Monday: CO_2 average from the Part 2 data file. (Column 3; remember! count starts at 0)
2. Tuesday: Temperature average from the Part 2 data file (Column 4)
3. Wednesday: Humidity average from the Part 2 data file (Column 5)
4. Friday:
    a. Last Name A – G : CO_2 average from the Part 2 data file (Col. 3)
    b. Last Name H – O : Temperature average from the Part 2 data file (Col. 4)
    c. Last Name P – Z : Humidity average from the Part 2 data file (Col. 5)
5. Recorded to eClass:
    a. Last Name A – G : CO_2 average from the Part 2 data file (Col. 3)
    b. Last Name H – O : Temperature average from the Part 2 data file (Col. 4)
    c. Last Name P – Z : Humidity average from the Part 2 data file (Col. 5)

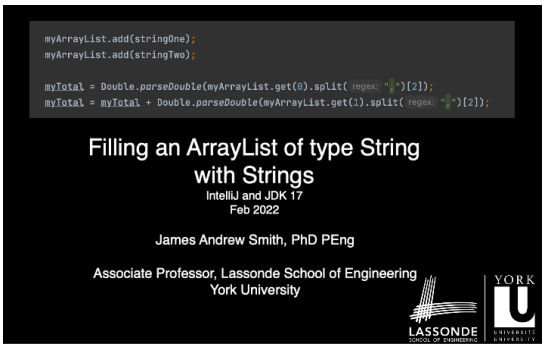
Data file	Column 0	Col. 1	Col. 2	Col. 3	Col. 4	Col. 5
Part2_CO2_DataLog_Modified_ drsmith:@indigo1.eecs.yorku.ca/www/eecs10 2,18,104.35,950,20,0.29 2,18,104.3333333,951,20,0.29 2,18,104.3166667,950,20,0.29 2,18,104.3,952,20,0.29 2,18,104.2833333,951,20,0.29	Month	Day	Time [hours]	CO2 [ppm]	Temp [°C]	Humid. [%]

Figure 8 Use the "Part 2 data file": https://www.eecs.yorku.ca/~drsmith/eecs1021/labE/Part2_CO2_DataLog_Modified_short.csv

Note: I've "cleaned up" the contents of the data files to get rid of the headers, as well as funny symbols like percentages, etc. The reason is that the header row and the funny symbols complicate the reading of the data. When you get more advanced you'll figure out ways around them. But for now, we'll keep it simple.



Extracting numeric information from an ArrayList<*String*>

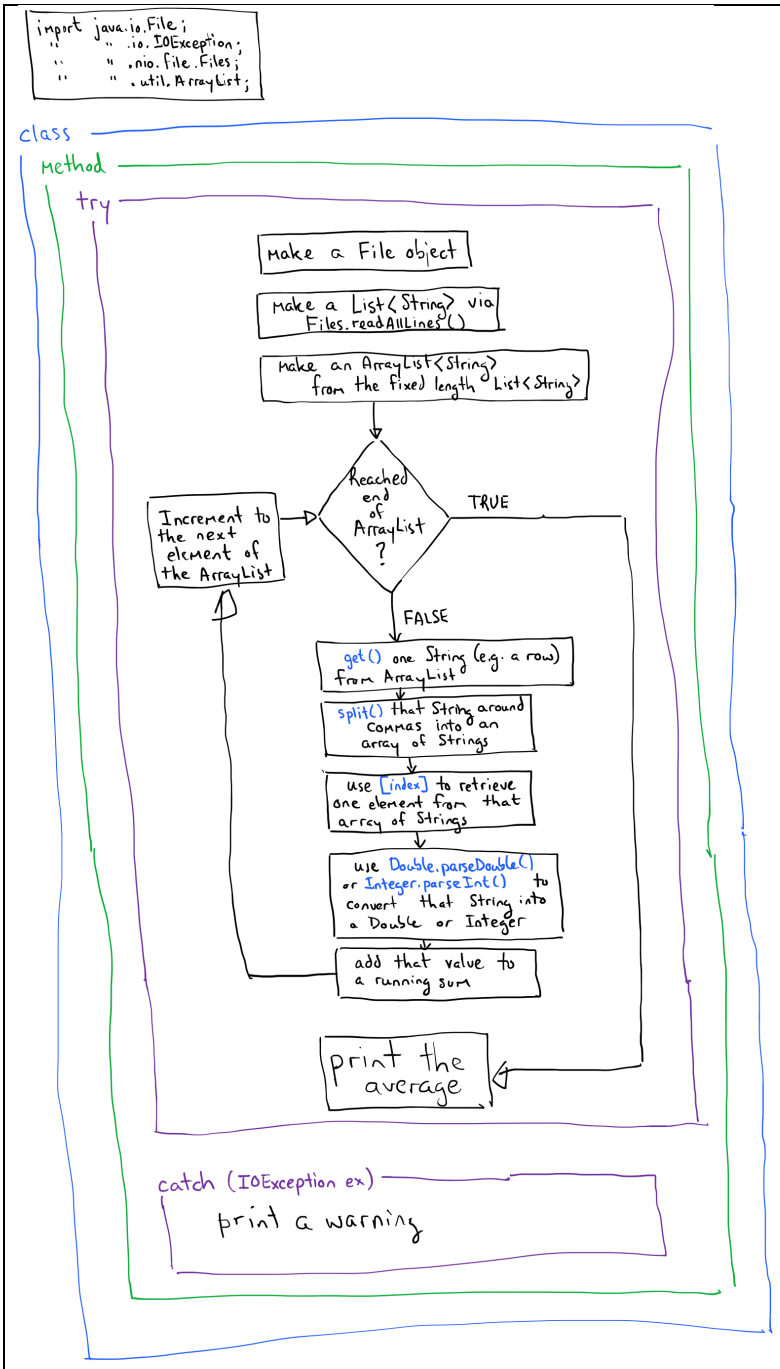


https://youtu.be/re7-n3Eq5_0

Video Demonstration of ArrayList<*String*> (https://youtu.be/re7-n3Eq5_0)

Figure 9 Building up an expression to extract numbers from an ArrayList of type String (e.g. ArrayList<String>)

In Part 1, we dealt with exceptions to the file reading by placing a little command just before the curly braces of the main method. Here, we'll introduce a **try-catch block** inside the main method.



Flowchart

```
import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.util.ArrayList;
// source: https://bit.ly/3BFixcx (Java67 reference)

public class Part2 {

   public static void main(String[] args) {
      try {



         // Step 1. Name the file and create a File object
         var theAirQualityFile = new
File("Part2_CO2_DataLog_Modified_short.csv");

         // Step 2. Real all lines in the File object
         var fixedLengthList =
Files.readAllLines(theAirQualityFile.toPath());

         // Step 3: copy fixed list to an ArrayList
         ArrayList<String> listOfString = new
ArrayList<String>(fixedLengthList);

         // Step 4. Create numeric variables to hold totals, increments, etc.

         // Step 5.  Loop through all rows of the ArrayList.  How many times?
         //          Apply the .size() method on the ArrayList.
         //          Double.parseDouble() or Integer.parseInt() as necessary.

         // Calculate the average & print to screen.









      } catch (IOException ex) {
         System.out.println("File reading exception");
      }
   }
}
```

Skeleton of program for Part 2

Good reference: https://bit.ly/3BFixcx (Java67 reference)

Part 3: Print out Contents of a CSV file using the CSV Parser Library

Interpreting the contents of a file is made easier when we use a "Parser" library. The Comma-Separated-Values format that is typically used in data files lends itself well to the use of parser libraries. Here, we'll use the "Open CSV" CSVReader library, which you should import into IntelliJ using Maven (search for com.opencsv:opencsv:3.8).

It's important to set up your main method with a try-catch block as you did in Part 2. Inside the "try" part, set up a FileReader object. Then, create a CSVReader object that reads that file and is told that the data in the file is separated by the "comma" character:

```
CSVReader csvReaderObject = new CSVReader(myFileObject,',');
```

Also, create a String array to capture one row at a time of the CSV file:

```
String[] oneLineRecord;
```

Then, create a while loop. Inside of the testing parentheses of the while loop write an expression that both

1. Transfers one line of the CSV obect into your String array, and
2. Tests to make sure that the thing that was read isn't "null" (empty… the end of the file)

```
thenameofyourstringarray = csvReader.readNext()) != null
```

Note, that in Parts 1 and 2, when you looped through the file, your while loop was checking for -1. For the csvReader object's readNext() method you have to check for *null*, and not -1.

Inside the while loop you'll want to print out the contents of your String array.

Finally, you'll want to close both the CSVReader object and the file object:

```
csvReader.close();
readMyFile.close();
```

Sources for more information:

- OpenCSV: https://www.geeksforgeeks.org/reading-csv-file-java-using-opencsv/
- OpenCSV: https://attacomsian.com/blog/read-write-csv-files-opencsv
- Covert String to Integer: https://www.freecodecamp.org/news/java-string-to-int-how-to-convert-a-string-to-an-integer/

The following shows the headers and sample data from a Water Station. Note that the data is slightly modified from the online file to make it easier for you to process in Java. I've removed the header row, leaving only data in each row. I've also modified the data and time data to make it easier to process. The earliest time value is 0 and increments in 5/60ths of an hour (5 minutes as a fraction of one hour).

Column 0	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8	Column 9	Column 10
Month	*Day*	*Hour*	*Water level [m]*	*Grade*	*Symbol*	*QA*	*Discharge rate [cubic meters per second]*	*Grade*	*Symbol*	*QA*
1	19	0	28.08			1	152			1

This example, shows that at midnight (Hour 0) on January 19, 2022 the water level (height) was 28.08 m and the discharge rate (flow) was 152 cubic meters per second.

For the Air Quality Sensor, the data is formatted like this in the modified data files:

Column 0	Column 1	Column 2	Column 3	Column 4	Column 5
Month	*Day*	*Hour*	*CO2 level (ppm)*	*Temperature (C)*	*Relative Humidity (%)*
1	2	23.0167	700	22	0.50

This example shows that at 23 hours and 1 minute after the first piece of saved data, on January 2, 2022, the CO2 level was moderate (700 parts per million) and the room temperature was 22 degrees C and the relative humidity was 50%.

When importing the OpenCSV library, make sure to use this within IntelliJ's Maven:
com.opencsv:opencsv:3.8

The CSV object allows you to capture data one row at a time. You'll do that inside a while loop. Each time you do it you will store the contents of that row in a String array. What's a String array? It looks like this when you create one:

```
String[] myStringArray = {"hello", "my", "string", "array"};
```

And you access an element like this:

```
System.out.println(myStringArray[3]);
```

But to print out the entire array at once, you do

```
System.out.println(Arrays.toString(myStringArray));
```

As discussed here: https://stackoverflow.com/questions/409784/whats-the-simplest-way-to-print-a-java-array

Here is what you need to demonstrate for Part 3:

Monday: Water Station 1 (Todmorden / Don Valley)
- One sample of data on Jan 19, 2022
- Data file: https://www.eecs.yorku.ca/~drsmith/eecs1021/labE/single_waterstation1_data_jan19.csv

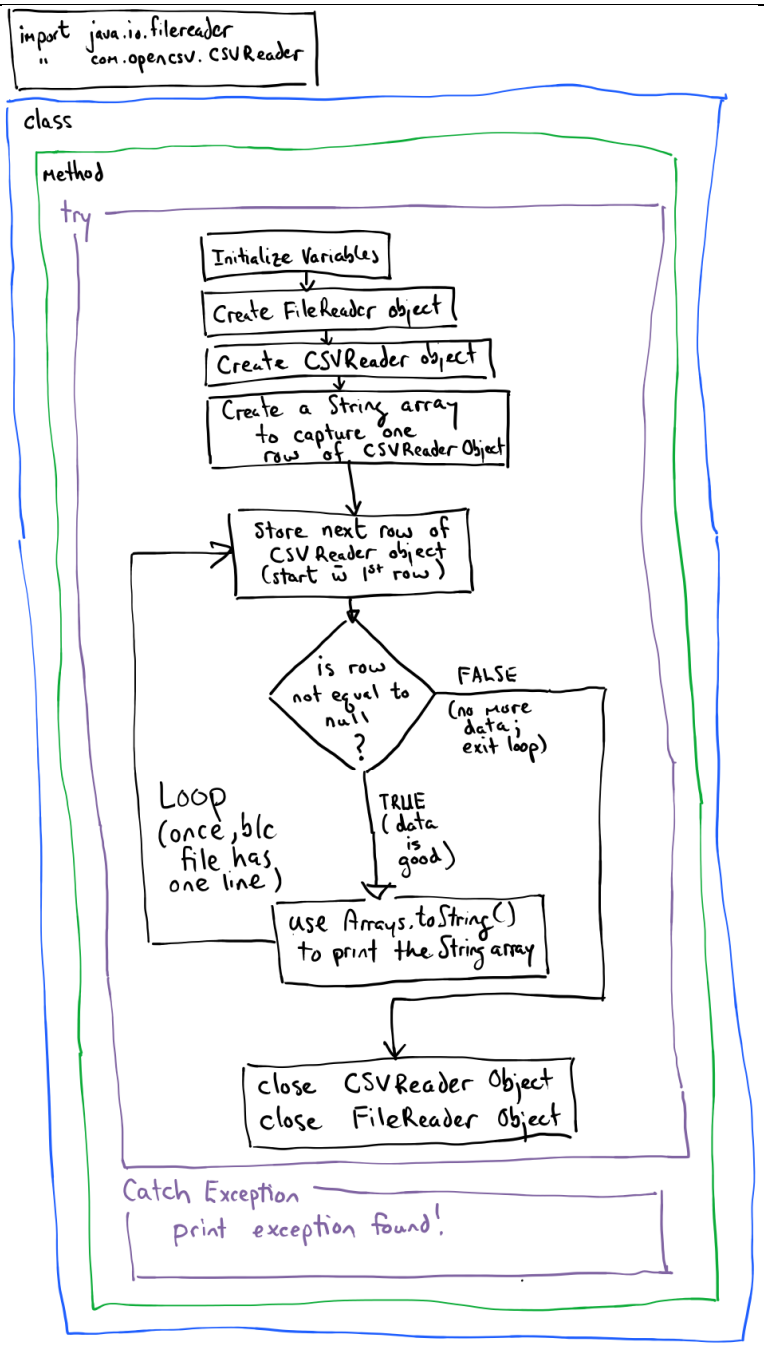
Tuesday: Water Station 1 (Todmorden / Don Valley)
- One sample of data on Jan 25, 2022
- Data file: https://www.eecs.yorku.ca/~drsmith/eecs1021/labE/single_waterstation1_data_jan25.csv

Wednesday: Water Station 2 (Attawapiskat)
- One sample of data on Feb 4, 2022
- Data file: https://www.eecs.yorku.ca/~drsmith/eecs1021/labE/single_waterstation2_data_feb4.csv

Friday: Water Station 2 (Attawapiskat)
- One sample of data on Feb 6, 2022
- Data file: https://www.eecs.yorku.ca/~drsmith/eecs1021/labE/single_waterstation2_data_feb6.csv

Recorded submissions: Air quality sensor
- Last Name A – G : One sample on Feb 17, 2022
  - Data file: https://www.eecs.yorku.ca/~drsmith/eecs1021/labE/single_co2_data_feb17.csv
- Last Name H – O : One sample on Feb 16, 2022
  - Data file: https://www.eecs.yorku.ca/~drsmith/eecs1021/labE/single_co2_data_feb16.csv
- Last Name P – Z : One sample on Feb 15, 2022

- Data file: https://www.eecs.yorku.ca/~drsmith/eecs1021/labE/single_co2_data_feb15.csv



Flowchart for Part 3

```
import java.io.FileReader;     // For reading files
import com.opencsv.CSVReader;  // Import with Maven: com.opencsv:opencsv:3.8
import java.util.Arrays;

public class LabEPart3 {
   public static void main(String[] args)
   {
      // try-catch is used in case reading the file fails.
      try
      {

         final String fileName = "single_waterstation2_data_feb6.csv";

         // Read the file.

         // Set up to process the file.



         // Read one line at a time in the CSV
         while(                                          ){
            // Extract the record and then use that copy elsewhere.
         }

         // Close the CSV and file objects.


      }

      // The catch part: catch the program if it fails.  Uncomment the catch.
      catch (Exception ex)
      {
         System.out.println("Exception reading the file!");
      }
   }
}
```

Skeleton of the class for Part 3

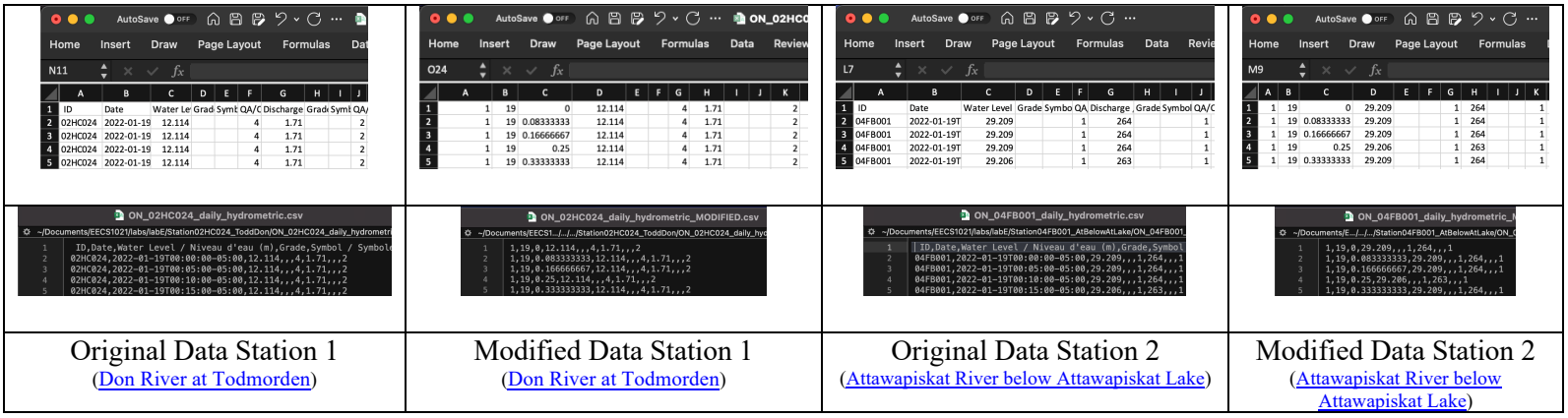
Part 4: Perform Math on data from a CSV file using CSV Parser Library.

Reading and printing data from a CSV file (Part 3 of the lab) is good, but being able to perform calculations on that data is vital.

Build on the work you did in Part 3. Inside of your while loop you'll want to do the following:

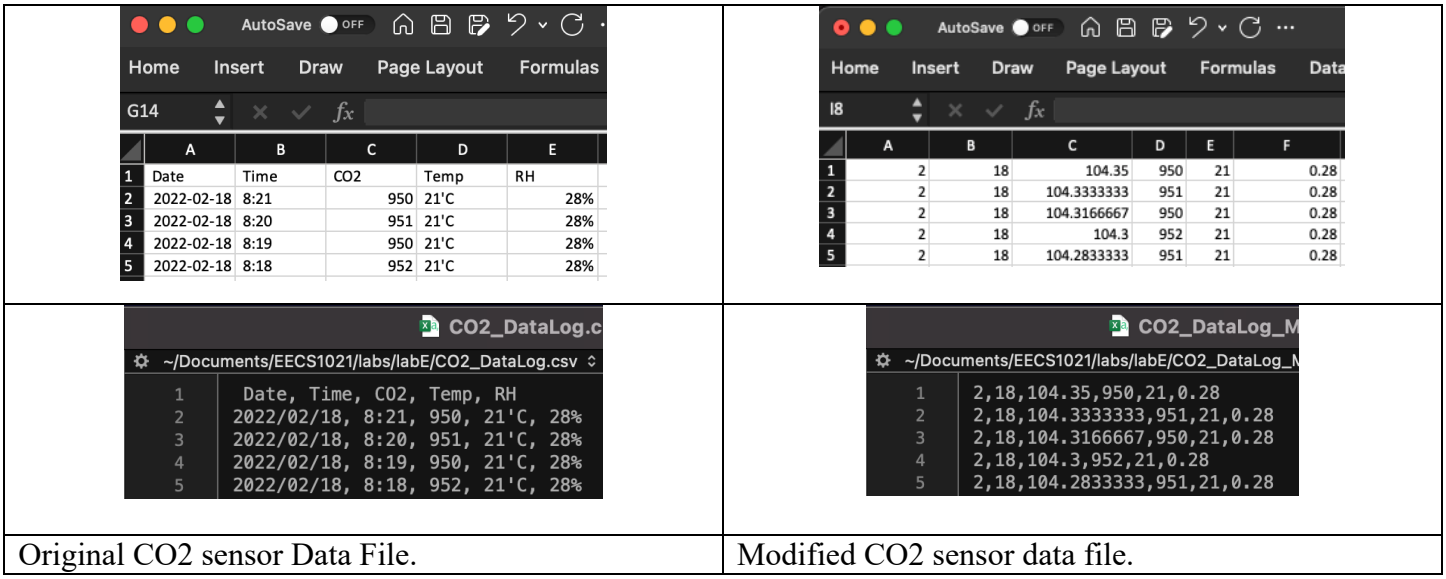
1. Extract an individual element out of the String array and place it in its own String.
2. Feel free to print out that String, just so that you can see it.
3. Convert the String into a Double Precision Floating Point value:
    a. Double.parseDouble(theStringName)

The data that you will be examining looks like this:



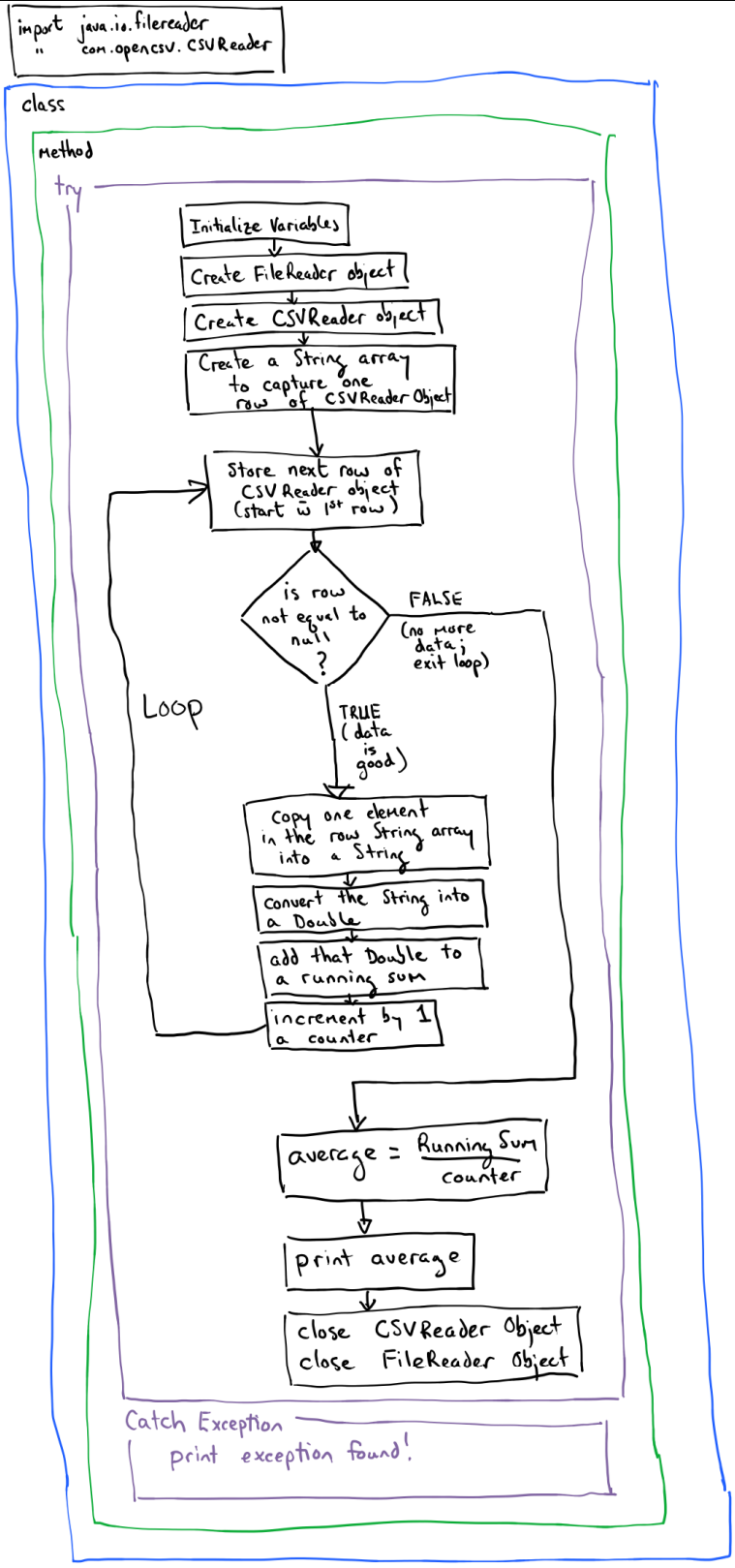
Original Data Station 1 (Don River at Todmorden)	Modified Data Station 1 (Don River at Todmorden)	Original Data Station 2 (Attawapiskat River below Attawapiskat Lake)	Modified Data Station 2 (Attawapiskat River below Attawapiskat Lake)

Figure 10 The Water Station data, viewed in Excel (top) and a text editor (bottom). Oldest data at the top, newest at the bottom. Data begins on January 19, 2022 and ends on February 18, 2022.



Original CO2 sensor Data File.	Modified CO2 sensor data file.

Figure 11 Data files for the CO2 sensor. Newest data is at the top. Oldest is at the bottom. Time starts at Midnight on Feb. 14, 2022. Ends at 8:21am on Feb. 18, 2022.



Flowchart for Part 4

```
import java.io.filereader // For reading files
import  com.opencsv.CSVReader // Import with Maven: com.opencsv:opencsv:3.8

public class LabEPart4WaterProcess {
   public static void main(String[] args)
   {
      // try-catch is used in case reading the file fails.
      try
      {


         // Read the file.
         FileReader readMyFile = new FileReader(fileName);

         // Set up to process the file.









         // Read one line at a time in the CSV
         while((                                    ){
            // Extract the record and then use that copy elsewhere.



            // Convert the String into a Double Float.

            //System.out.println(Double.parseDouble(myString));

         }

         // Calculate average & print result to the screen.
         // Discharge rate
        // System.out.println("Discharge @ Attawapiskat: " + value + " m^3 per second.");









         // Close the CSV and file objects.
         csvReader.close();
         readMyFile.close();

      }


      // The catch part: catch the program if it fails.  Uncomment the catch.
      catch (Exception ex)
      {
         System.out.println("Exception reading the file!");
      }
   }
}
```

Skeleton for Part 4

For the Part 4 demonstration, please use the data files, as indicated here:

- Monday: Water Station 1
  - Station 02HC024 (Don River at Todmorden)
  - Dates: Jan 17 – Feb 18, 2022
  - Use this file *https://www.eecs.yorku.ca/~drsmith/eecs1021/labE/ON_02HC024_daily_hydrometric_MODIFIED.csv* [4]
  - Calculate the **average** of the Water **Level** (Column 4, where 1st column is Column 0)
- Tuesday: Water Station 1
  - Station 02HC024 (Don River at Todmorden)
  - Dates: Jan 17 – Feb 18, 2022
  - Use this file *https://www.eecs.yorku.ca/~drsmith/eecs1021/labE/ON_02HC024_daily_hydrometric_MODIFIED.csv*
  - Calculate the **average** of the **Discharge** Rate (Column 7, where 1st column is Column 0)
- Wednesday: Water Station 2
  - Station 04FB001 (Attawapiskat River below Attawapiskat Lake)
  - Dates: Jan 17 – Feb 18, 2022
  - Use this file https://www.eecs.yorku.ca/~drsmith/eecs1021/labE/ON_04FB001_daily_hydrometric_MODIFIED.csv [5]
  - Calculate the **average** of the Water **Level** (Column 4, where 1st column is Column 0)
- Friday: Water Station 2
  - Station 04FB001 (Attawapiskat River below Attawapiskat Lake)
  - Dates: Jan 17 – Feb 18, 2022
  - Use this file https://www.eecs.yorku.ca/~drsmith/eecs1021/labE/ON_04FB001_daily_hydrometric_MODIFIED.csv
  - Calculate the **average** of the **Discharge** Rate (Column 7, where 1st column is Column 0)

- Recorded submissions:
  - Use the AirQ Air Quality sensor data file (Feb 14 – 18, 2022)
  - Use this file: http://www.eecs.yorku.ca/~drsmith/eecs1021/labE/CO2_DataLog_Modified.csv [6]
    - Last Name A – G : Air Quality Sensor: Calculate Average CO2
    - Last Name H – O : Air Quality Sensor: Calculate Average Temperature
    - Last Name P – Z : Air Quality Sensor: Calculate Average Humidity

[4] Original daily data for Don River at Todmorden: https://www.eecs.yorku.ca/~drsmith/eecs1021/labE/ON_02HC024_daily_hydrometric.csv (don't use this in your Java program. Use the modified data, instead)

[5] Original daily data for Attawapiskat: https://www.eecs.yorku.ca/~drsmith/eecs1021/labE/ON_04FB001_daily_hydrometric.csv (don't use this file in your Java program. Use the modified data, instead)

[6] Original air quality file: https://www.eecs.yorku.ca/~drsmith/eecs1021/labE/CO2_DataLog.csv (don't use this file in your Java program. Use the modified data file, instead)

Part 5: Skill Question

We've noticed that a number of students are simply copying and pasting each others' code. This is a recurring problem in programming classes, not just this one. To encourage you to try to understand the lab exercise content a bit more we're having TAs ask you a question during the lab demonstrations (Zoom or in-person).

If you are submitting a pre-recorded video, then you will have to answer one of ten questions as a text submission to Turn-it-in. If the answer appears to be plagiarized, based on the Turn-it-in score, you will be given a grade of 0 for the answer.

- Last Name A – J: Explain why we use try-catch in this lab.
- Last Name K -- R: Explain why we would want to use an ArrayList instead of an Array.
- Last name S – Z : What kinds of separators are typically used in a CSV file? (Not just the obvious one)