COMP0004 Coursework 2

Purpose: To develop an application with a GUI to display data.

Submission: Upload to Moodle before 16:00pm 22nd March 2021. To submit create a zip file of your IDE project and upload that.

Marking: This coursework is worth 30% of the module mark. Complete as many of the requirements as you can. It is better to submit a well-written, working program that covers a majority of the requirements, than a poorly written program that tries to do everything.

Inadequate (0-39)	Failed to demonstrate a basic understanding of the concepts used in the coursework, or answer the questions. There are fundamental errors, code will not compile, or nothing of significance has been achieved. (F)
Just Adequate (40-49)	Shows a basic understanding, sufficient to achieve a basic pass, but still has serious shortcomings. Code may compile but doesn't work properly. Not all concepts have been understood or all the core questions answered correctly. (D)
Competent (50-59)	Reasonable understanding but with some deficiencies. The code compiles and runs, the concepts are largely understood. This is the default for a straightforward, satisfactory answer. Most core questions answered. (C)
Good (60-69)	A good understanding, with possibly a few minor issues, but otherwise more than satisfactory. The code compiles, runs and demonstrates good design practice. Most expectations have been met. All core questions answered. (B)
Very Good (70-79)	A very good understanding above the standard expectations, demonstrating a clear proficiency in programming and design. All core questions plus at least one challenge question answered. (A)
Excellent (80-100)	Programming at a level well above expectations, demonstrating expertise in all aspects. This level is used sparingly only where it is fully justified. All questions answered. (A+, A++)

On Moodle you will find a link to some data files on GitHub in Comma Separated Value (CSV) format. The files contain synthetic patient data, with each line in the file representing one patient. The example below is the first two lines representing patients in the file patients100.csv:

3e9fd20e-b81c-4698-a7c2-0559e10361fa,1979-08-24,,999-46-5746,S99951588,X43863756X,Mrs.,Norah104,Stamm704,,Hagenes547,M,white,french,F,Hanover Massachusetts US,678 Trantow Overpass,New Bedford,Massachusetts,02740

71917638-8142-42da-87f1-fa0ba642f161,1989-10-21,,999-95-8365,S99995552,X3761822X,Mr.,Eloy929,Dickinson688,,,M,white,english,M,Dover Massachusetts US,431 Armstrong Forge,Upton,Massachusetts,01568

As you can see, each line is a sequence of strings separated by commas. Each string represents a piece of information about a patient. These are the fields or columns used to describe a patient, and are in the same order on every line. You can load a .csv file into a spreadsheet (e.g., Excel) to more easily view the data. In the data files provided commas are used only as separators between columns, which means a line can be easily split into a collection of column strings by splitting it at each comma.

The first line of the patients.csv file does not represent a patient, instead it gives the column name for each column, also separated by commas:

ID, BIRTHDATE, DEATHDATE, SSN, DRIVERS, PASSPORT, PREFIX, FIRST, LAST, SUFFIX, MAIDEN, MARITAL, RACE, ETHNICITY, GENDER, BIRTHPLACE, ADDRESS, CITY, STATE, ZIP

The meaning of the names is obvious, for example FIRST and LAST give the first and last name of each patient (note this is synthetic data so the names have a numerical code attached - just treat these as part of the name).

If you look through the data you will see that for some patients a column entry may be left blank, and you see two commas in a row (i.e., Stamm704,,Hagenes547). Beware that the final column (ZIP) can be left blank, so that there will be a comma followed by newline.

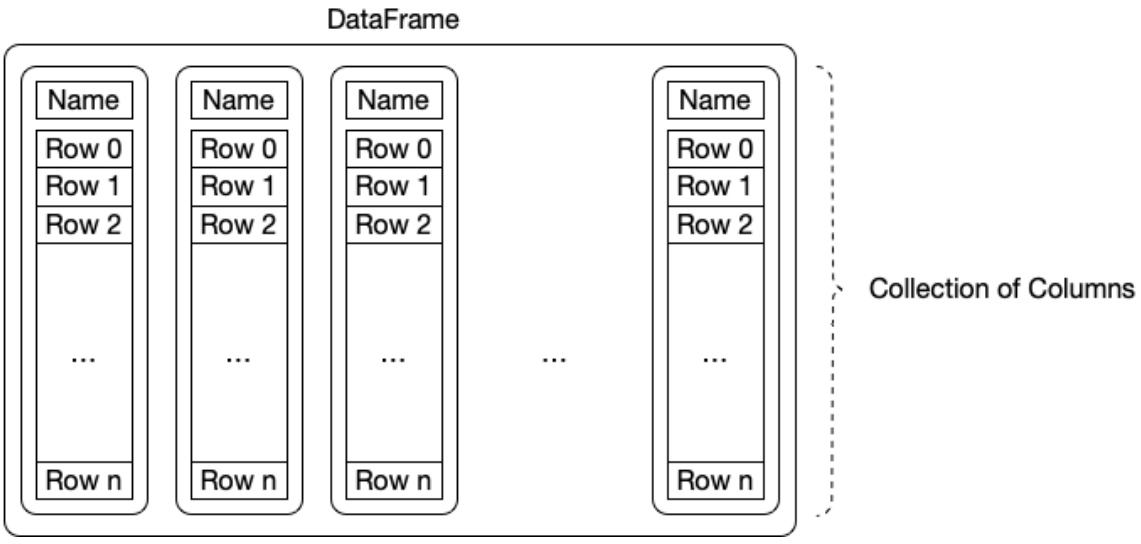
The first column, ID, is a unique id for each patient. The id is effectively the primary key for each patient and is what you use to uniquely identify each patient. A few other columns such as SSN and PASSPORT should be unique but are not usually used as primary keys. The rest of the columns will not contain unique values and you will certainly find that names and addresses appear multiple times.

There are four patients files on GitHub:
- patients100.csv - 100 patients
- patients1000.csv - 1000 patients
- patients10000.csv - 10000 patients
- patients100000.csv - 100000 patients

These are all in the same format. For development you want to work with the smallest file, but your code should work with the larger files as well.

In this coursework the aim is to create a general purpose data structure called a DataFrame, that can be loaded with the patient data (or potentially any other data) and used in a basic dashboard program with a graphical user interface (GUI). A dashboard is an application that allows you to view and manipulate data.

A DataFrame has this structure:



This is a collection of named columns where each column holds the data from a single column in the csv data, and the column name is the csv column name. For example, the first column would hold the ID for all patients, the second the BIRTHDATE and so on. A single row across all the columns holds the complete line of csv data for a single patient, so that row 0 is the data for the first patient and so on. This is the same kind of layout you would see if you open the .csv file in a spreadsheet application like Excel.

Requirements

Implement a program for the requirements listed below.

Assume that all the data held in a DataFrame or Column is always of type String even if there are numerical values or dates. The requirements form a specification for the dashboard program, so they don't have to be answered in order and you may well find it easier to work on each requirement in stages as you develop your program.

Requirement 1. Write a class Column, where a column has a name and an ArrayList of rows. Provide methods getName, getSize, getRowValue, setRowValue, and addRowValue (to add a new row). Determine what the parameters and return types should be. If you find additional methods are needed when answering later questions then add them, but *only* when you need them.

Requirement 2. Write a class DataFrame to hold a collection of Columns. It should have the following methods:

- addColumn
- getColumnNames (a list of names in the same order as they are stored in the frame).
- getRowCount (the number of rows in a column, all columns should have the same number of rows when the frame is fully loaded with data).
- getValue(columnName, row) to get a row value from a column.
- putValue(columnName, row, value) to put a value into a row in a column.
- addValue(columnName, value) to add a value to a column (at the end).

Again you can add methods if you find you need them when answering the later questions. Also remember that all the data is in String format.

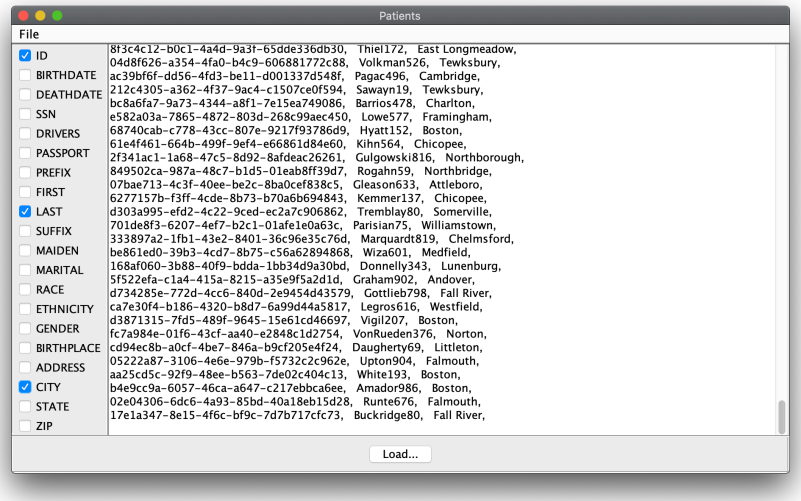
Requirement 3. Write a class DataLoader that can read a .csv data file and load the data into an empty DataFrame. The Column names are found as the first row in the .csv data file. It should have a method that returns a filled DataFrame.

Note that the file input methods will require you to manage IOExceptions that can be thrown (e.g., trying to open a file that doesn't exist). You will have to decide how to manage these. Your program should not crash or terminate if an exception is thrown!

Requirement 4. The program should have a Swing GUI which should communicate with a class Model to access the data to display. The GUI should *only* use the Model to get data and *not use or know about* DataFrames or Columns. Class Model provides a wrapper or *Facade* that hides how data is stored, providing a set of public methods the GUI can use as needed. While not part of this coursework the aim of having the model is to allow alternate data storage (e.g., a SQL database) to be used without having to change any of the code that uses the Model.

Write a class Model that manages a DataFrame internally and provides the methods that your GUI needs, and uses the DataLoader to load data into a frame.

Requirement 5. Write a Swing GUI that allows the data to be viewed in various ways. For example a very basic GUI might look like this (this is just an illustration, you don't need to copy this layout!):



Here the list of column names is displayed as check boxes that can be selected to determine what to display in the main panel, and there is a load data file button. You can design your own GUI and provide more (or better) ways of displaying the data!

Requirement 6. Write a Main class that contains the main method and any code to initialise the program, such as creating the GUI and Model objects. This class should contain the minimum code needed to get the program running, so will be quite short. The code for building the GUI out of Swing components should be in the GUI class.

Requirement 7. Add the ability to search the data and perform operations such as finding the oldest person, the number of people living in the same place, and so on. The GUI will need to be extended to give access to this functionality. Remember that the GUI only displays data it can get from the Model, so the implementation of search and other functions should be in the Model or other classes, not in the GUI code. Input from the GUI should be passed as parameters in method calls to the Model.

Challenge Requirements

Requirement 8. Write a class JSONWriter that given a DataFrame writes out the data to a file in JSON format. The JSON should be well-formed. You have to write all the code to generate the JSON yourself and cannot use a JSON library. As you will be doing a lot of String manipulation investigate the StringBuilder class, as this is much more efficient at building strings than the String class itself. Add a save to JSON format functionality to your GUI.

Also write a JSONReader that can read data files in JSON format, rather than the .csv format, and add the data into a DataFrame.

Requirement 9. Add the ability to display graphs or charts to show the data, for example distribution by age. You must write all the code to display a graph or chart yourself, and cannot use a library that you might find on the web somewhere.

The last two requirements assume you submit two versions of your program. One for requirements 1-9, the other for requirements 10 and 11. Each version should be the IDE project, both placed into the same zip file (you can only submit one zip file overall). Do not include the larger data files in your zip file.

Requirement 10. The requirements above do not state that any Java interfaces should be used (as in programming to an interface). Identify where interfaces can be introduced and add them, modifying the rest of the code as needed. Remember interfaces are used to separate specification (a set of methods) from implementation, allowing different implementation objects to be substituted. For example, to have DataFrame implementations that store the data in different ways

internally. Think carefully about how and where implementation objects are created, remember the Factory pattern.

Requirement 11. Assume that the program can also be used with very large data files, so that there is much more data than can be loaded into memory as objects all at the same time. Replace the DataFrame with one or more classes that access the data while it is still stored in a file. You can still have some data in memory or make use of a in memory data structure acting as an index to the data. You must write all the file handling code yourself using the classes provided in the standard JDK. You cannot use any other class libraries or a database.