COMP0004 Coursework 2024

Purpose: To develop a Java web application based on the content covered in lectures.

Submission: Upload to Moodle before 16:00pm 20th March 2024. Create a zip file of your IDE project and upload that, including your report document file (in pdf).

Marking: This coursework is worth 10% 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. Very poor report.(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 requirements implemented correctly. Report is just good enough but not well written.(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 requirements answered, report is reasonable. (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, and most requirements implemented. Quite good, clear report. (B)
Very Good (70-79)	A very good understanding above the standard expectations, demonstrating a clear proficiency in programming and design. All requirements implemented very well, very good report showing good insight into the design and programming process. (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 requirements answered excellently, excellent report with real depth and insight. (A+, A++)

Tasks

1. Implement a Java Web Application using the requirements specification below (80%).

2. Write a report on your work with this content (20%):

- Section 1: Summarise what features your program implements, half a page maximum.
- Section 2: Describe and evaluate your design and programming process. You should reflect on how you went about designing your classes, why they are appropriate classes, whether you have used good OO design practice (e.g., good use of abstraction, cohesive classes), and the overall quality of your work (you decide the criteria for this).
- The overall report should be a *maximum* of 2 pages (sides of A4 paper) in length.

Requirements Overview

Implement a Java web application to access, update, and display patient data for as many of the requirements listed below that you have time to complete. You can use any of the classes provided in the standard JDK, plus the Tomcat and servlet classes that are provided via the example code. Other than a library for reading and writing JSON or CSV data (covered in lectures), you should *not* add any other additional third-party libraries libraries or use a database such as MySQL. Use JDK 21 if possible.

The requirements form a specification for the program, 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.

The Data

On Moodle you will find a link to some data files on GitHub in Comma Separated Value (CSV) format, or use this URL https://github.com/UCLComputerScience/COMP0004Data. 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 row of .csv file does not represent actual data, instead it gives the column name for each column in the rows below. These names are also separated by commas. For the patients100.csv the column names are:

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, not real data, and 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 there are 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

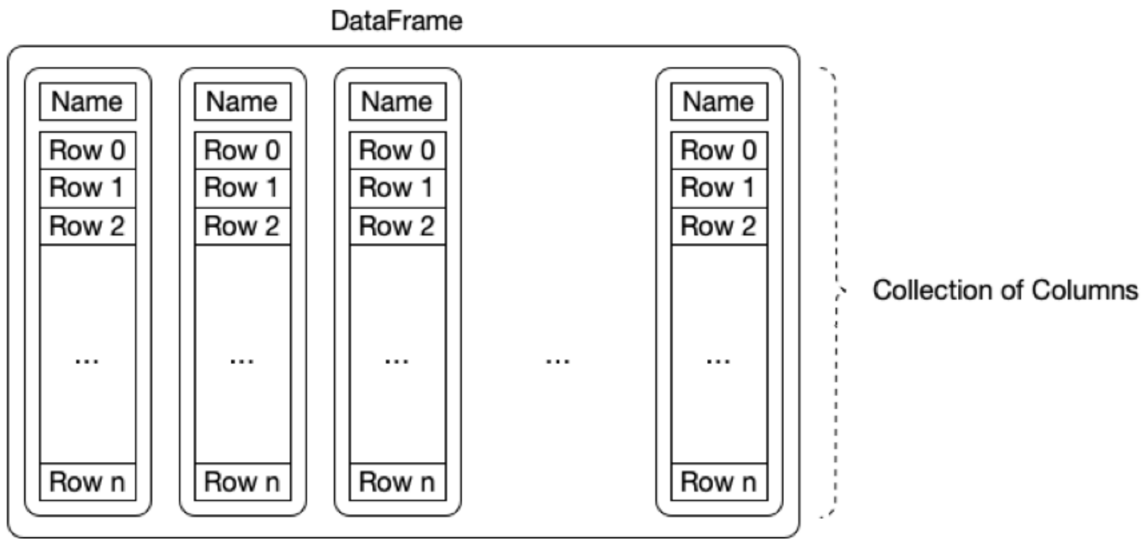
patients10000.csv - 10000 patients

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. There are other csv files on the GitHub site but you don't need to use them.

In this coursework you should create a general purpose data structure called a DataFrame, that will be by the Model part of your web application (remember the MVC pattern). The DataFrame can be loaded with the patient data (or potentially any other data) so that the View and Controller parts of the application can provide a dashboard to display and manipulate the data.

A DataFrame has this structure:



This is a series 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.

Assume that all the data held in a DataFrame is always of type String even if there are numerical values or dates.

The dashboard user interface should be implemented as web pages viewed in a web browser, using standard html and css. You don't need to implement a sophisticated user interface, this module is primarily about Java programming not creating complex front-ends for web applications! If you want to you can use a CSS framework like BootStrap, which makes it quick and easy to create nice looking webpages, but this is entirely optional. Don't get side-tracked in to spending too much time on the appearance of web pages.

Requirements

It is up to you to interpret the requirements and decide the details of what you actually implement.

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. Write the web application Model class that manages a DataFrame and related classes such as the DataLoader internally, providing the methods that your Controllers (servlets) need.

Requirement 5. Develop the servlets and JSPs needed to view the data in various ways to create a working web application. Make your own decisions on how the data is displayed and the operations provided on it.

Requirement 6. Add the ability to search the data by entering a string or keywords to match against the data. The implementation of search should be in the Model not in servlets or JSPs. A search servlet should pass the search words as parameters in method calls to the Model.

Requirement 7. Add a variety of operations such as finding the oldest person, the number of people living in the same place, and so on.

Challenge Requirements

Requirement 8.

Add the ability to add a new patient row, edit an existing row, or delete a row. The data will need to be saved by writing out a new CSV file.

Requirement 9. Write a class JSONWriter that given a DataFrame writes out the data to a file in JSON format. The JSON should be well-formed. Add the ability to save to JSON option to the web application.

Requirement 10. Add the ability to display graphs or charts to show the data, for example distribution by age.

Design Notes

Remember the Model-View-Controller (MVC) pattern. A Java servlet is a controller so should contain only the code to receive a http request (i.e., when the user clicks an URL or link on the web page, or submits data in a form), call methods on the model to carry out actions on the data, and forward the result data to a JSP to display the results. Web pages (the html and css) should be generated by Java Server Pages (JSPs).

A servlet will need to get a reference to the Model via the Singleton Pattern. This means that the Model provides a static method to return a reference to a Model object - see the example code for how this works.

The example code and content on the Moodle site shows how to set up a web app project in the IntelliJ IDEA IDE. You do need to use Maven to manage, build and run applications as shown. The best way to start is to clone the example project and then start extending it.

When using the IntelliJ IDEA IDE, you need to make sure that you the JDK version settings are correct. In the Project Structure configuration dialog make sure that Java 21 is selected everywhere.