UCL Hospital Registry System

Dhruv Ghulati

December 5, 2015

1 Introduction

This is a Java based implementation of a hospital registry system. Doctors can register themselves, login, and maintain a list of patients they are responsible for, including editing, deleting, adding, exporting and importing, and managing scans and images. Implements the model view controller design pattern.

The final project is held in the 'finalproject' package. Other non-executable jar files contain prior attempts to build the system, and this code is visible at any point.

The video explainer can be viewed here:

`https://www.youtube.com/watch?v=p72jtbvLLxs`

2 Key Features

Implements a model which consists of Doctor, Patient and Person objects. Doctors and Patients inherit from Person by calliing the person constructor and having access to the global variables "First Name" and "Last Name".

Implements a View class which is built in Java Swing. This consists of one JFrame, a "login" panel, a "register doctor" panel, and then within the main system a "tabspanel" and a "container panel" of several panels. By clicking certain buttons in the tabspanel, the panels within "container panel" alternate to being visible/not visible (not technically cardlayout, but achieving the same outcome). There is one View class which controlls everything to do with the visuals of the system, and each panel is created via its own standalone method (rather than separate classes extending from JPanel).

Implements a Controller class which manages the database of the system. Data is passed around via two master ArrayLists of Doctor and Patient objects respectively. There are also only two CSVs, called "masterpatients" and "masterdoctors". When the system is initialised, the controller has a

method to generate these two arraylists from the two CSVs. When a doctor is logged in, a "session" is created, and the original patientlist is filtered and only the patients of that particular doctor ID are displayed within the JTable. However, when any edits, additions, deletions, imports or exports are made, the filtered patient list is removed from the original patientlist, and then added back with the required changes. This way, the entire system as a whole is maintained.

One feature that is unique is the generation of clickable Wikipedia URLs. In this system, a doctor can type in anything into the medical condition text field, and the text entered will be appended to a Wikipedia standard URL, if it exists (any white space is replaced by an underscore, matching the Wikipedia URL scheme. The user can then generate a clickable URL from any medical condition entered, and it will take him/her to the Wikipedia page. This was achieved via the Desktop and StringBuilder libraries within the Java SDK.

Uniqueness in the system of a Patient and Doctor is maintained via overriding the .equals() and hashcode() methods within Java. A Patient is unique via his/her first name, last name and the doctor ID who registered him/her. A Doctor is unique via a combination of his/her username, and password.

The system uses Abstract Classes and Interfaces. The use of Abstract Classes is via the creation of a class called PatientTableModel, an extension of AbstractTableModel which defines the JTable view of patients within the system, and fires change events to the table by altering the ArrayList of Patients underlying the table itself (the constructor of this class requires an ArrayList of Patients to be passed in). The use of Interfaces within the system is used by implementing MouseListeners, FocusListeners, DocumentListeners, ActionListeners and many other listeners on the text fields and search boxes of the JTable panel.

There are several exceptions documented and caught within the system, sometimes producing stack traces of the errors, and sometimes offering JOptionPane error messages within the system. For example, there are FileNotFoundExceptions, URISyntaxException IOExceptions and ParseExceptions/PatternSyntaxExceptions (for validating phone numbers and dates of birth, or for the search boxes).

Finally, the optional requirements are handled (importing a CSV of patients and filtering for any patients that are already in the system and not adding those in), as well as exporting a selection of patients from the JTable to a CSV at any point.

3 Prior attemps and issues encountered

Prior attempts to build this system encountered some major roadblocks. First of all, splitting the system into several classes for the view, for example with many "extends JPanel" classes for the patientTablePanel and the registerPanel was an issue. Even though it allowed me to design these components separately, repainting the components and being able to reset the text of components was an issue, as it mean coordinating from multiple places.

Second of all, again related to Java Swing, the use of a layout called GridBagLayout() for my registerPanel caused major issues. Because the GridBagLayout requires the individual specifications of where a component goes within a panel e.g. gridx=1, gridy=4, where it is aligned etc, and because these constraints are altered for each component, you cannot use the Design pane within Eclipse to see what is going on - you must manually reshape, and then run the program. By using this layout, and being frustrated, I wasted a lot of time and decided to rebuild my JPanel using MigLayout() instead of figuring out why a particular component was oddly sized and going through all the prior components one by one.

Third of all, using the cardlayout via the cards.show() method for the panels within containerPanel causes major issues, again with regards to understanding what was going wrong with repainting.

The biggest problem I faced, however, was related to the JTable and being able to repaint it. This caused me about 4 days of lost work.

`http://stackoverflow.com/questions/33926682/cant-get-jtable-to-refresh-repaint-a-defaulttablemodel-in-same-instance-of-gui?noredirect=1#comment55707937_33926682`

I initially started to keep generate my tablemodel within my Controller, returning a DefaultTableModel. However, when I added a patient, it would appear, but upon deletion I would face IndexOutOfBounds exceptions, and also general thread repainting issues. I realised that Swing is really not thread safe, and concurrency is difficult to maintain. By creating a very custom tablemodel extending from AbstractTableModel, I had visibility over the fireXXX methods, and then my tablemodel started to repaint.

4 Java Libraries used beyond Lecture Notes

- Desktop
- BufferedReader/BufferedWriter
- FileReader/FileWriter

- BufferedImage
- File
- JTable
- Image
- URI
- MigLayout
- AbstractTableModel

5 Script/Tests in Video

1. Register as a new Doctor, and add, edit, delete, export and import patients.
2. Login as an existing Doctor, and add, edit, delete, export and import patients.
3. The master file should be shown and updated in the correct way
4. The image files should be clearly changed or edited upon any edit
5. Upon login, only that doctors patients should be viewed
6. When adding, the patient should appear in the master file, and the JTable.
7. When editing, the patient should be reset in the master file, and the JTable. If adding an existing patient with same name, the system should ask you if you want to edit instead, if not tell you to add a new patient instead. If editing a patients first name or last name, it should prompt the user with a similar warning to add a new unique patient instead.
8. When deleting, the patient should disappear from the main file and the JTable.
9. When importing, the system should only import unique patients instead of importing all the patients, or missing patients out that should be added. When exporting, the file exported should match the selection made.

6 Potential Improvements

- Not store strings in the database, but hashed characters.
- Allow more advanced filters instead of text search for searching patients, for example JComboBox for the billing cycles

- Allow for many appointments to be stored for one patient, allowing many to one relationships (storing arrays of appointments into one cell of masterpatients)
- Displaying images themselves within the JTable
- Making the medical condition field within the JTable clickable to the URL, rather than having to click "Edit Patient" to view the URL.
- When appending the doctor username to the CSV, a few spaces are added before the start of the cell