## Criterion C: Development

1. Encapsulation

2. Complex Data Types

3. Sequential Search

4. Polymorphism

5. Recursive Sorting and Quick Sort

6. Error Handling

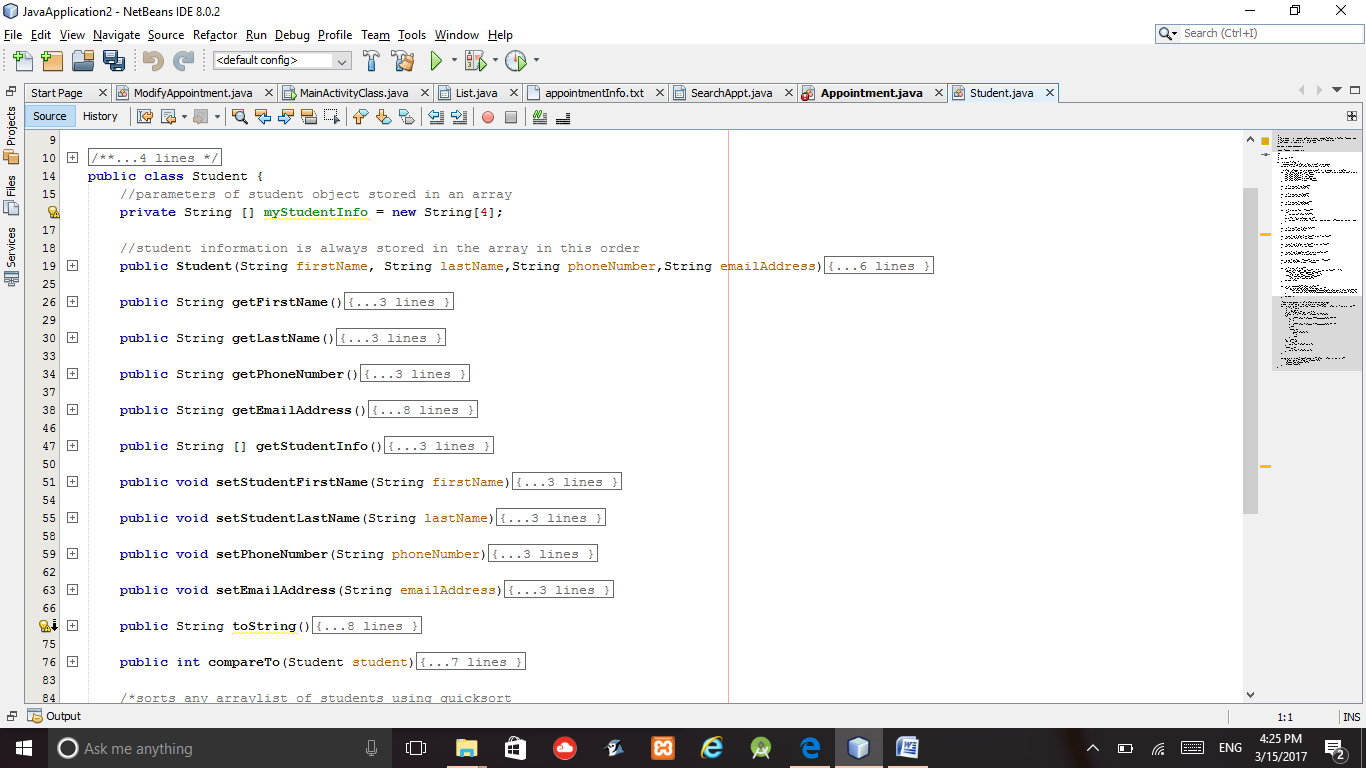
7. File Reading and Writing

8. Date Class

9. Inheritance

10. Timer and TimerTask Class

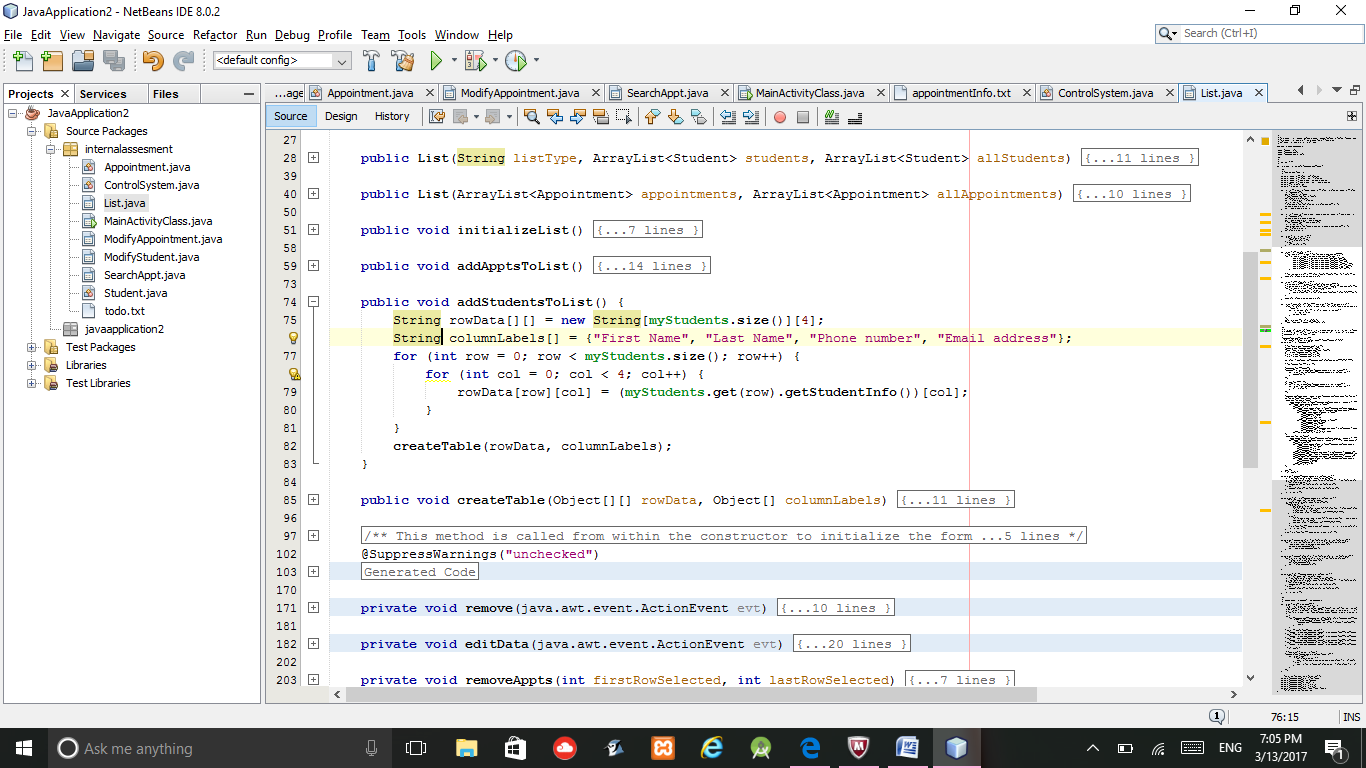
11. GUI

**1. Encapusulation**

The Student and Appointment classes were fully encapsulated by making all states private and giving them all public accessor and mutator methods. Encapsulation was a very useful feature because all private variables were easily accessible. The variables could represent the individual pieces of information of a student such as name and phone number and of an appointment such as date and time. This also allows me to add more features later on.

**2. Complex Data Types**

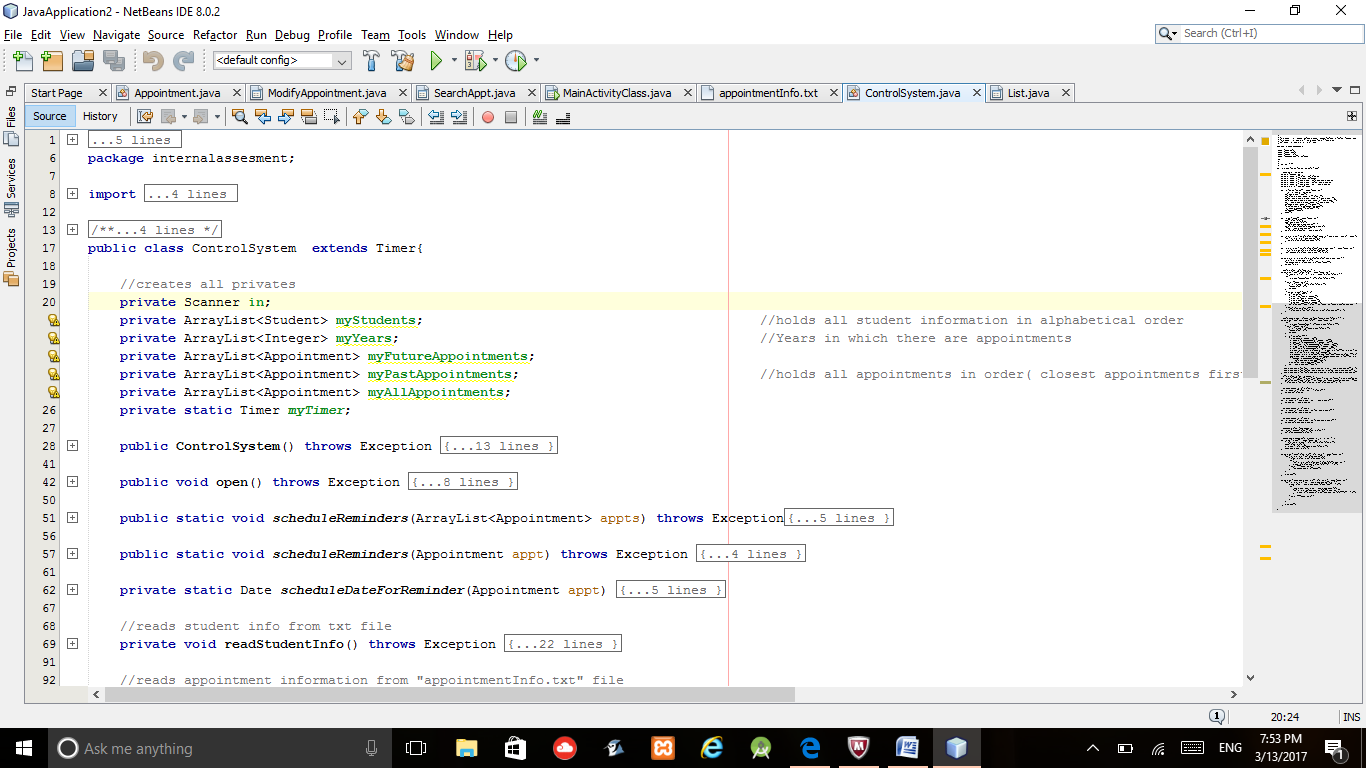
1. 2D Arrays

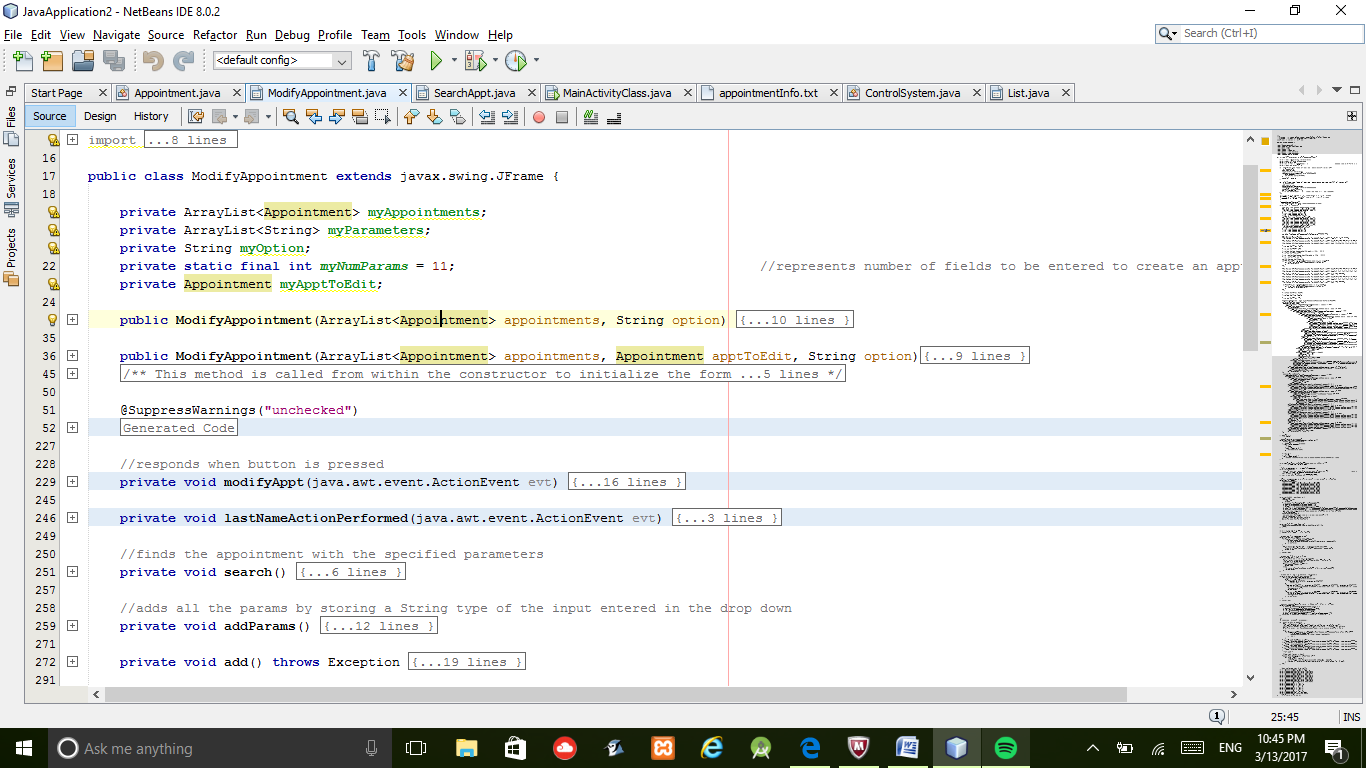
2D Arrays were used when displaying data in a tabular format. For example, the tables were used to display Student and Appointment information in the List.java class. This was useful because student information, for example, could be organized into five different columns and each row would contain one student and his/her information

Column size was 4 because four pieces of Student info is displayed.

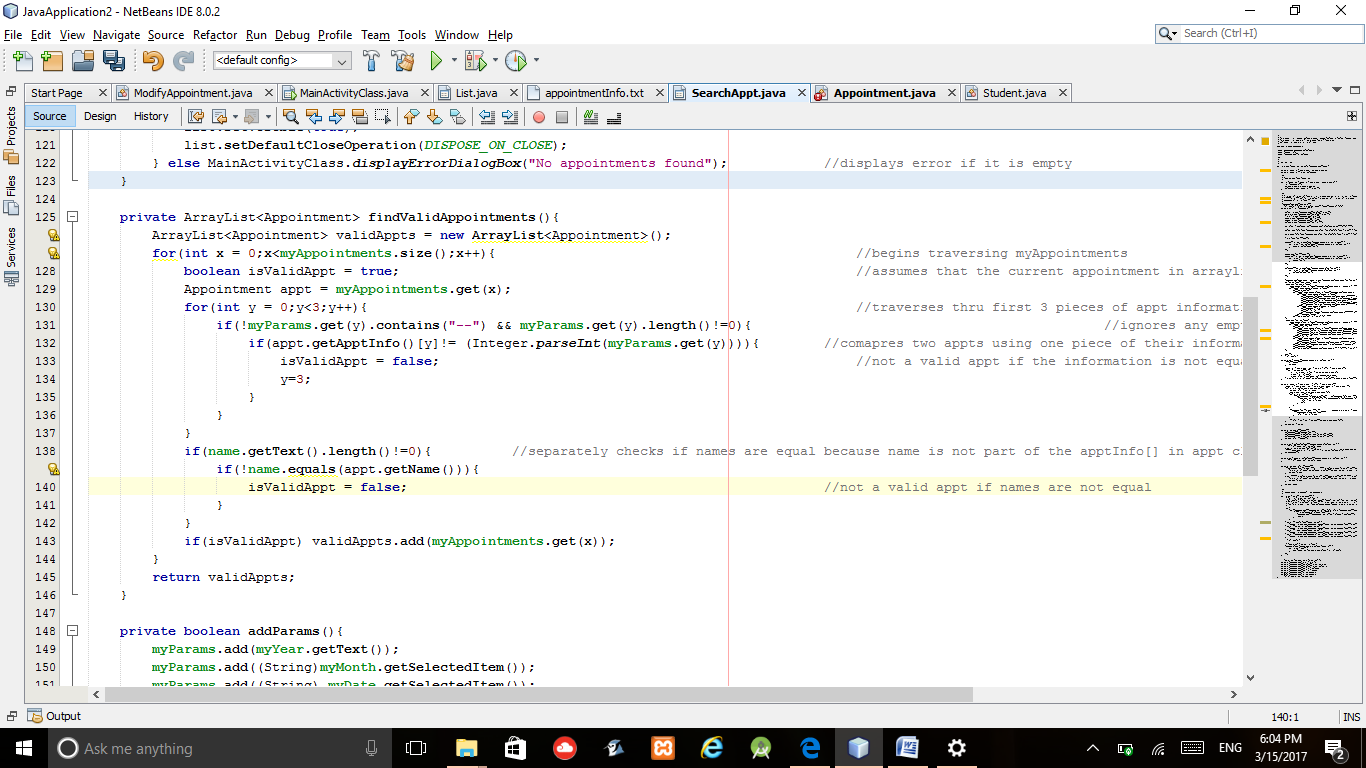
Column Labels used to organize the student information

1. Dynamic Data Structures: Arraylists

Arraylists were the most suitable data structure to use because multiple classes, such as ModifyStudent and ModifyAppointment, needed access to the same arraylist. Therefore, when a different class makes a change to the arraylist, the change only has to be made once because there is essentially only one arraylist.

For example, the constructors of ModifyAppointment requires an Arraylist<Appointments>. If myAllAppointments from ControlSystem.java is passed in, changes made to the list by this class will be made to myAllAppointments.

**3. Sequential Search**

 Sequential Search was the algorithm used to search for appointments or students. A for loop was used to iterate through the arraylist of objects. Sequential search is beneficial because the whole arraylist needs to be checked. This is because multiple object could meet the user's required criteria. Consequently, sequential search will allow multiple items to be added to an arraylist of valid objects whereas Binarysearch only finds one item. When searching for an appointment, if a text field is blank, it implies that the user does not want to search for that criteria. For example, if no email address is entered in student form, then the program will not compare student information based on email addresses.

For loop to iterate through the appointment list

Check if user input is not blank for a text field

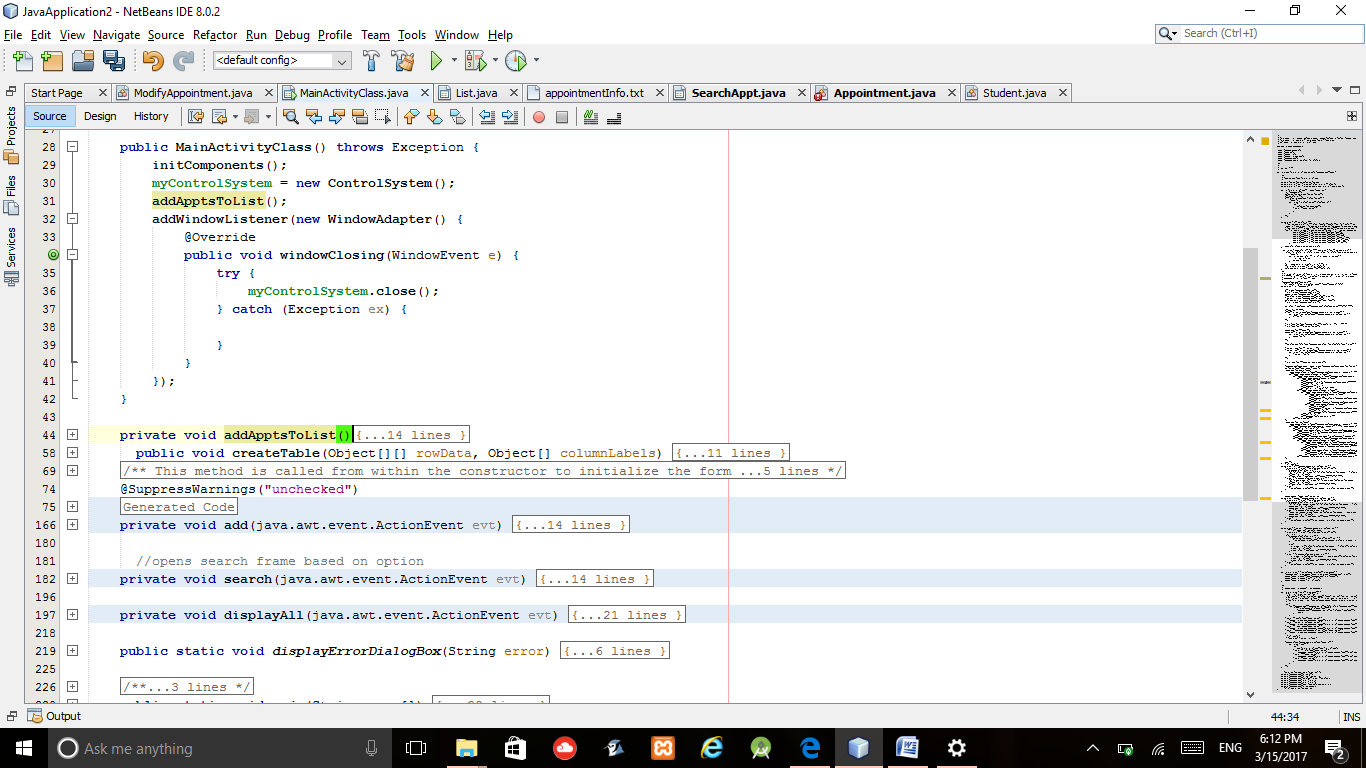
Assumes it’s a valid appt

Compares the two pieces of information

Compares the name of the student if user has entered it

If appt met the search criteria, adds it to a list of valid appts. Returns the list after traversing the myAppts list

**4. Polymorphism**

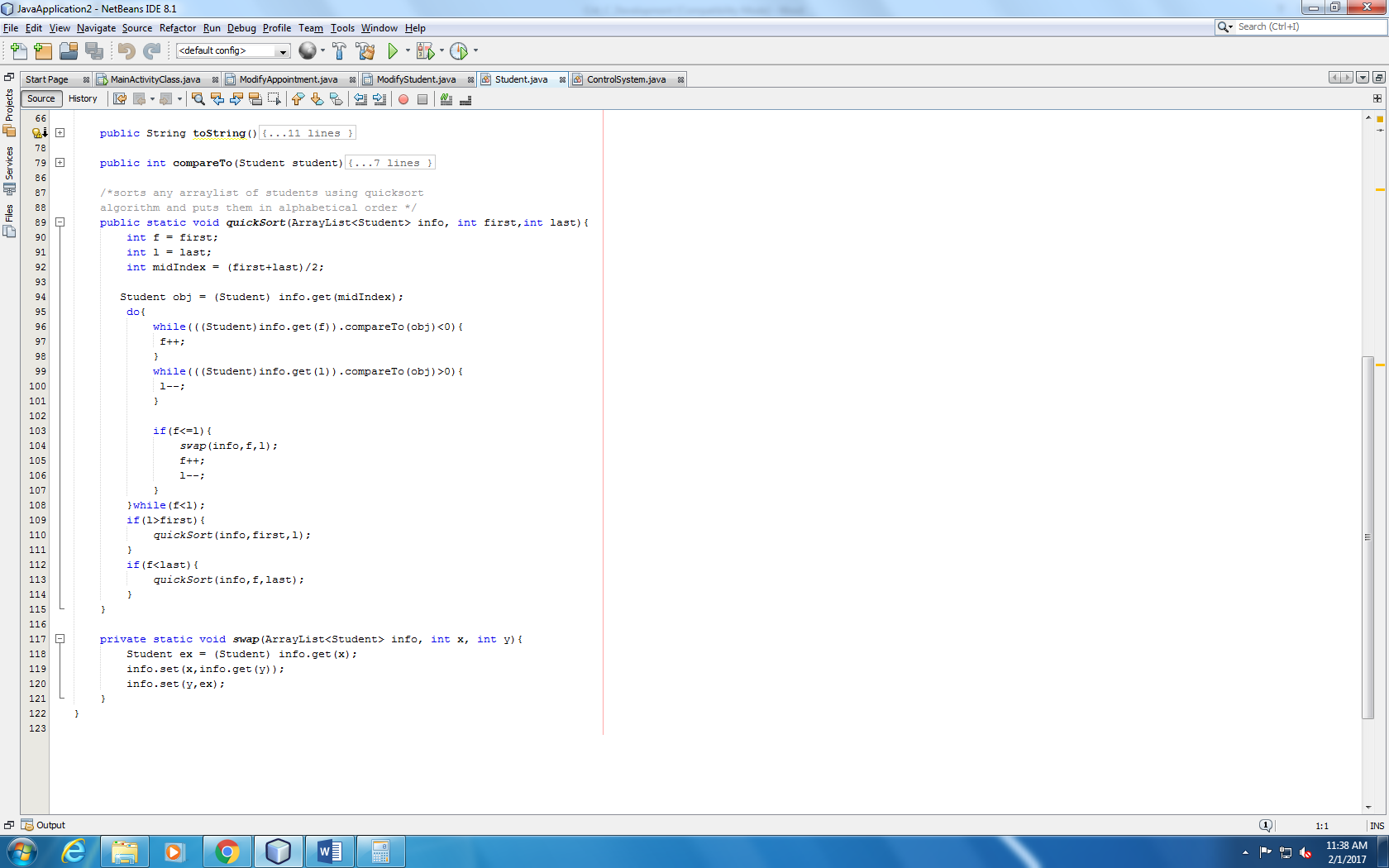
 Polymorphism was a useful implementation. When the Main Menu closes, a Window Listener can be added. Its default windowClosing method can be overriden and instead tell the control system to close and save all the data.

Implements a window listener class. (allows program to detect opening, closing, minimizing and maximizing of window

overrides windowClosing method. Tells the ControlSystem class to close and save the data to txt files.

**5. Recursive Sorting & QuickSort**

The Student and Appointment class both have static methods that sort an arraylist. Both implement the quick sort algorithm. Student arraylists are organized alphabetically by first name. Although, if the tutor added multiple items at once, this becomes the most efficient way to sort the arraylist. Therefore, QuickSort will ensure efficient sorting for any drastic changes to the data structure.

****

Continues incrementing first a value in list is greater than mid's value

Continues decrementing last until a value in list is less than mid's value

Swaps the two if the first index is less than last

Recursively calls quicksort to sort new arraylist

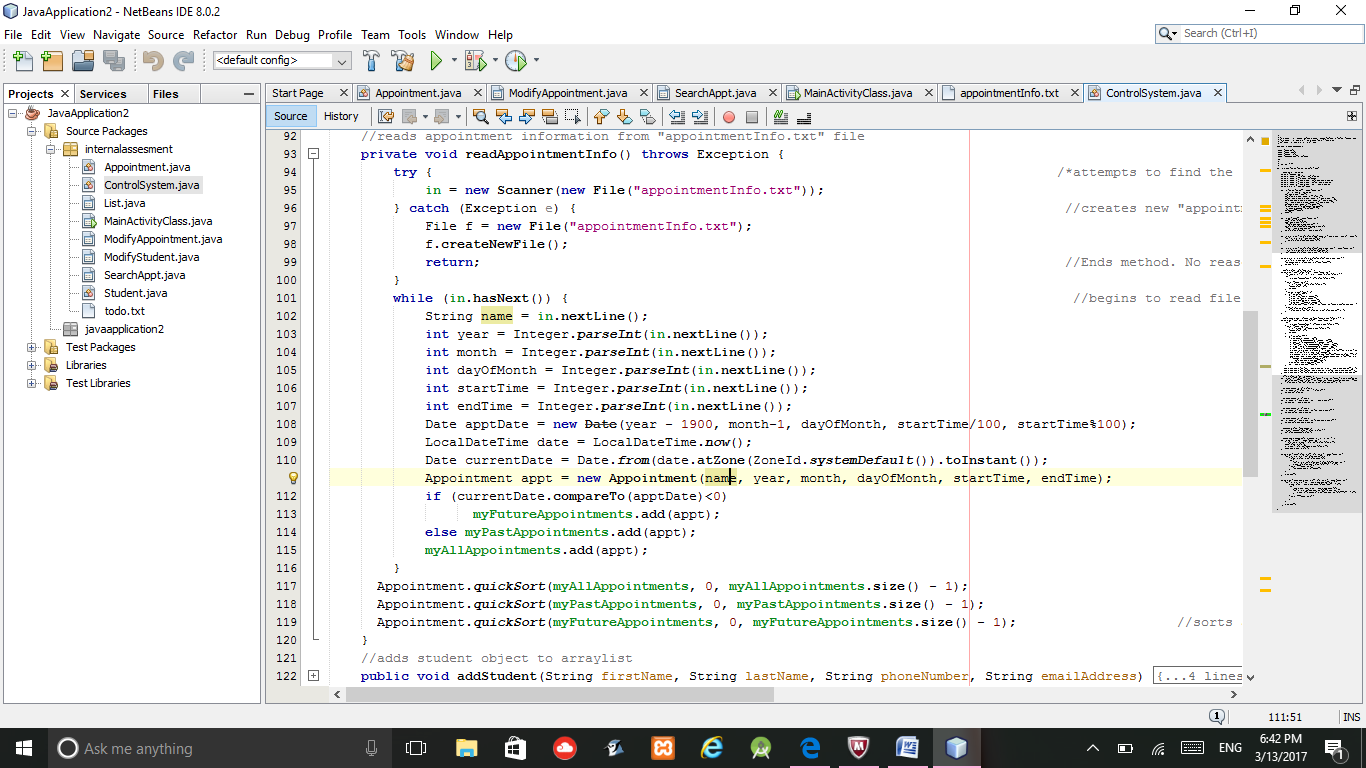
**6. Error Handling**

Error handling was a huge component of this project because there were many sources of runtime and logic errors.

1. Runtime Errors.

Runtime errors were most likely to occur when reading the files. A try-catch block will catch the only possible error: FileNotFoundException. The catch-clause allows me to

create a file if it does not exist and end the method call because the new file will be empty.

****

Attempts to initialize Scanner in to read the file.

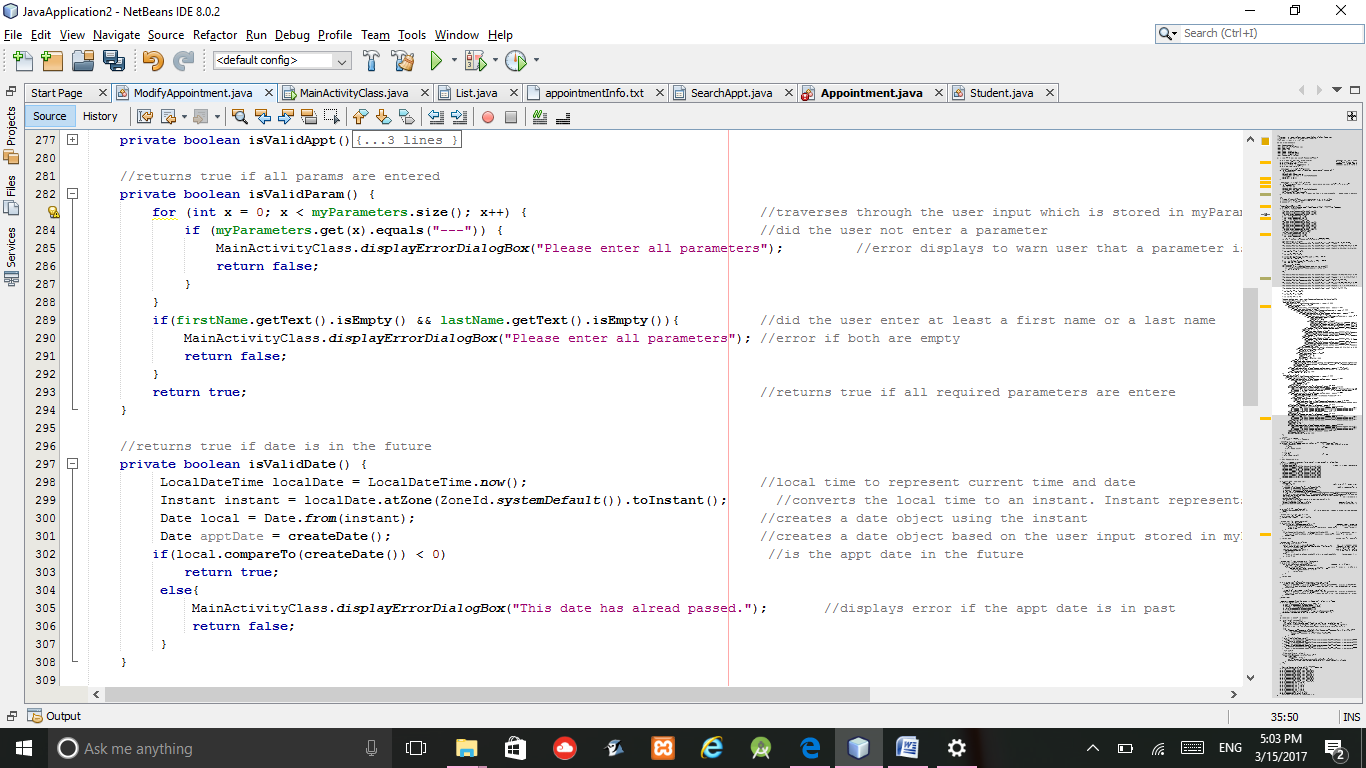
Creates a new file to be read and ends the method.

2. Logic Errors

The user can make logic errors in many places. For example, the user

* does not choose a data type to work with.
* tries to schedule an appointment in the past
* tries to schedule an appointment that overlaps with another appointment
* searches for a student or appointment that does not exist

Such errors were commonly handled by using an if-else statement. A static method, displayErrorDialogBox(String error), in the MainAcitivityClass instantiates a JDialogBox object and displays the error the user has made.

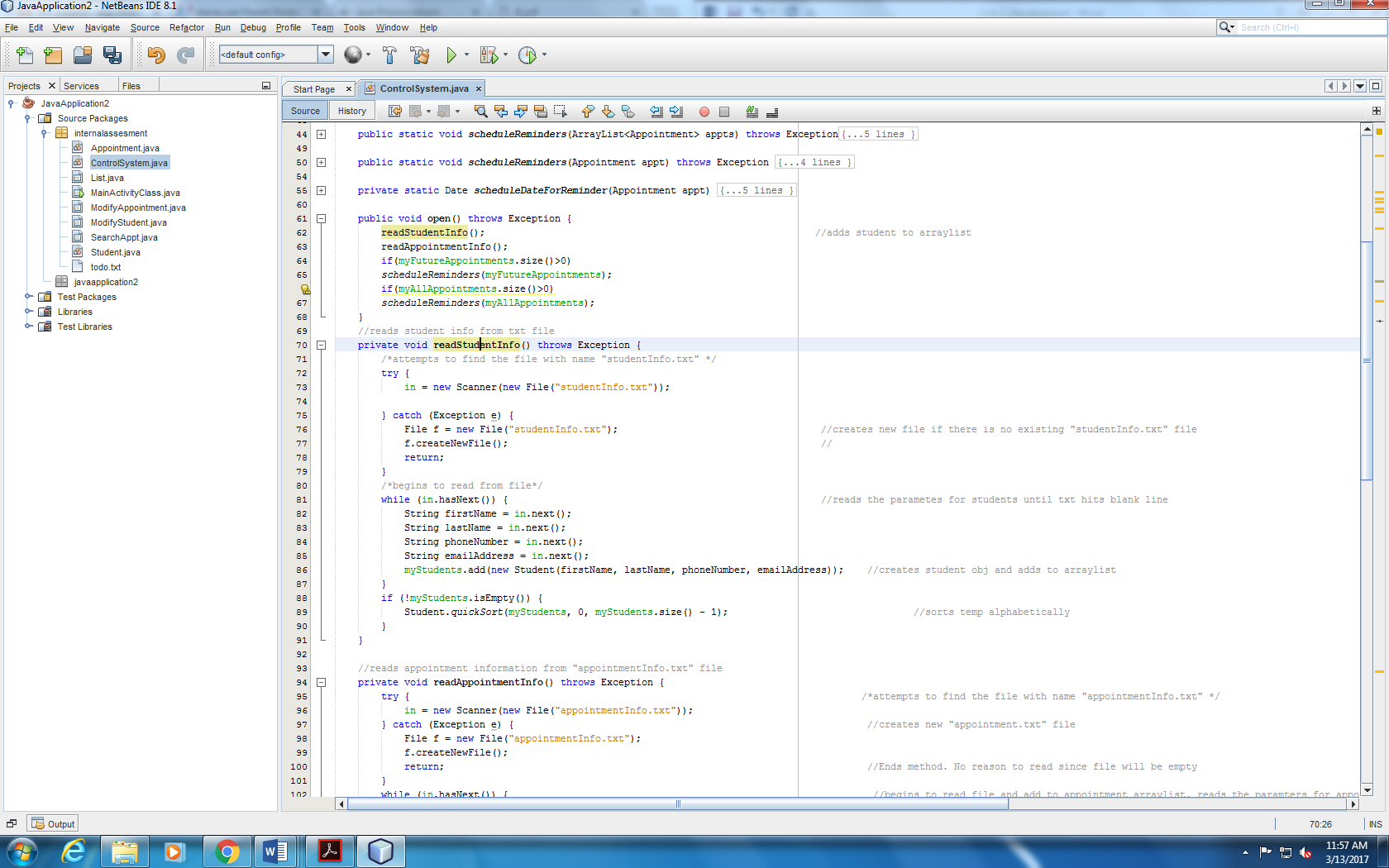
****

Are both first and last name text fields empty?

Is the parameter empty? \*An empty parameter in a combo box is "---"

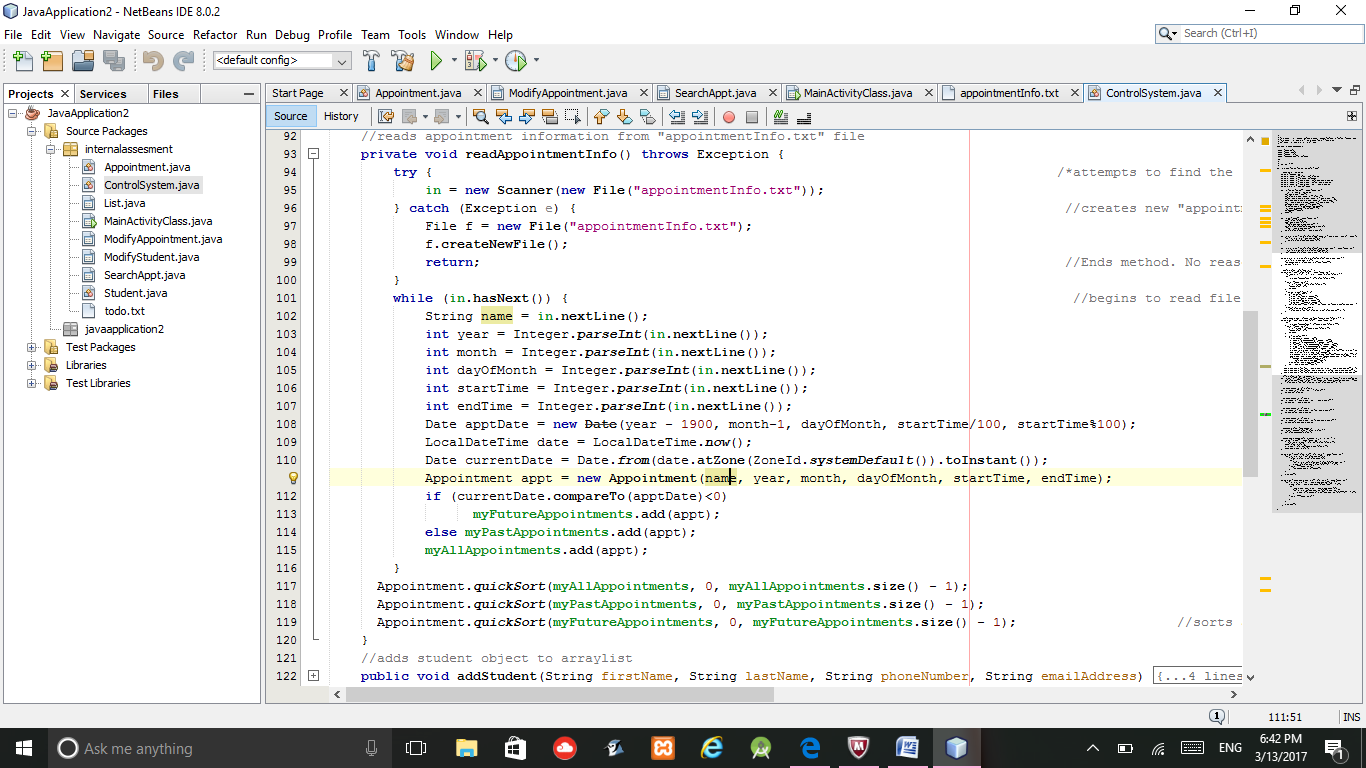
Is the date and time of the appointment in the future?

**7.File Reading and Writing**

The ControlSystem class handles all file reading, writing and data storage in the open() and close() by utilizing the Scanner and FileWriterclass from java.util package. FileReading and writing is important to create and save Student and Appointment objects. I used simple txt files because of their simplicity. readStudentInfo() methods as shown below in Figures 1 and 2. Using a while loop, it traverses arraylists to get a student or appointment object and write that objects information to the file.

Calls readStudentInfo() and readAppointmentInfo()

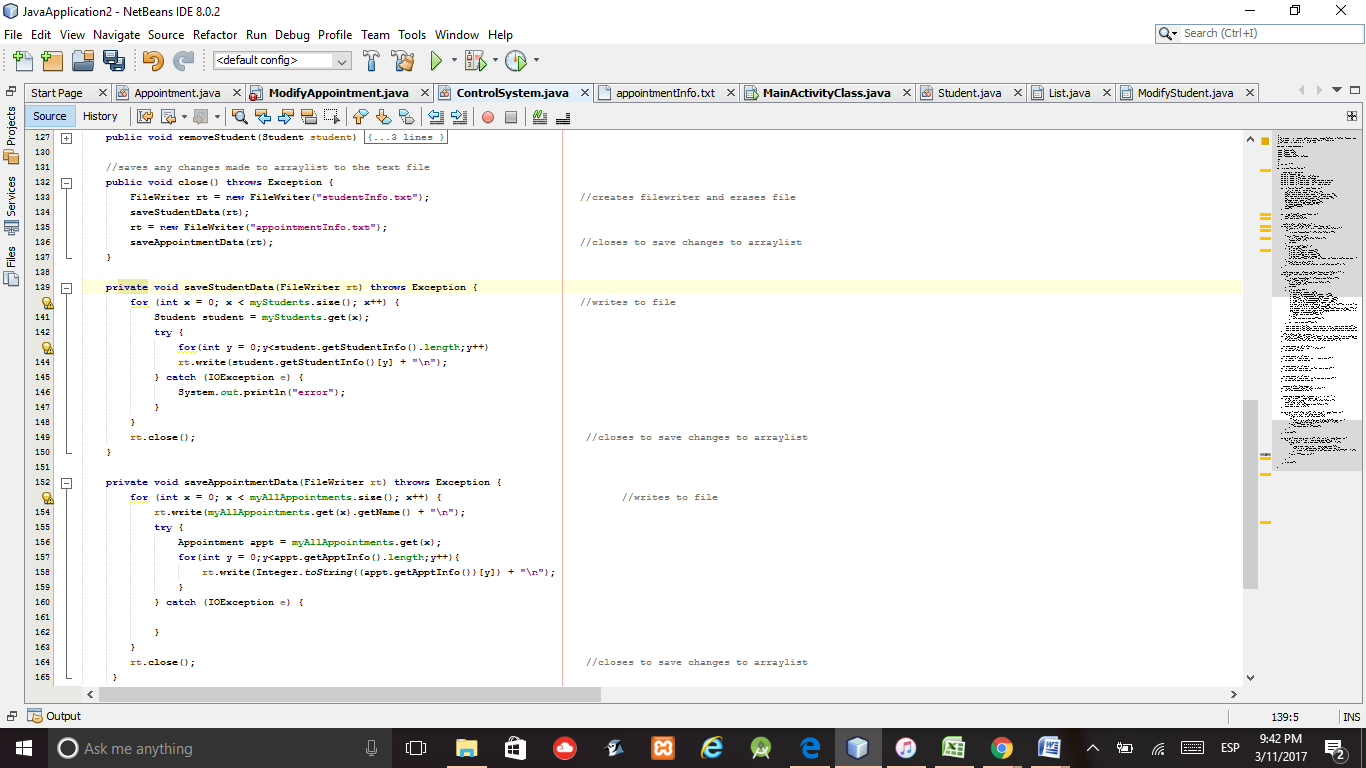
**Figure 1**



Continues to read file until reaches blank line

private Scanner *in* used to read file. Creates a new file and ends method if it does not exist.

The close() method is responsible for saving all the changes to the data by writing to the appointmetn.txt and student.txt files.



for loop to iterate through myStudents

Closes filewriter to save changes to file.

Writes to file and creates a new line every time using new line character

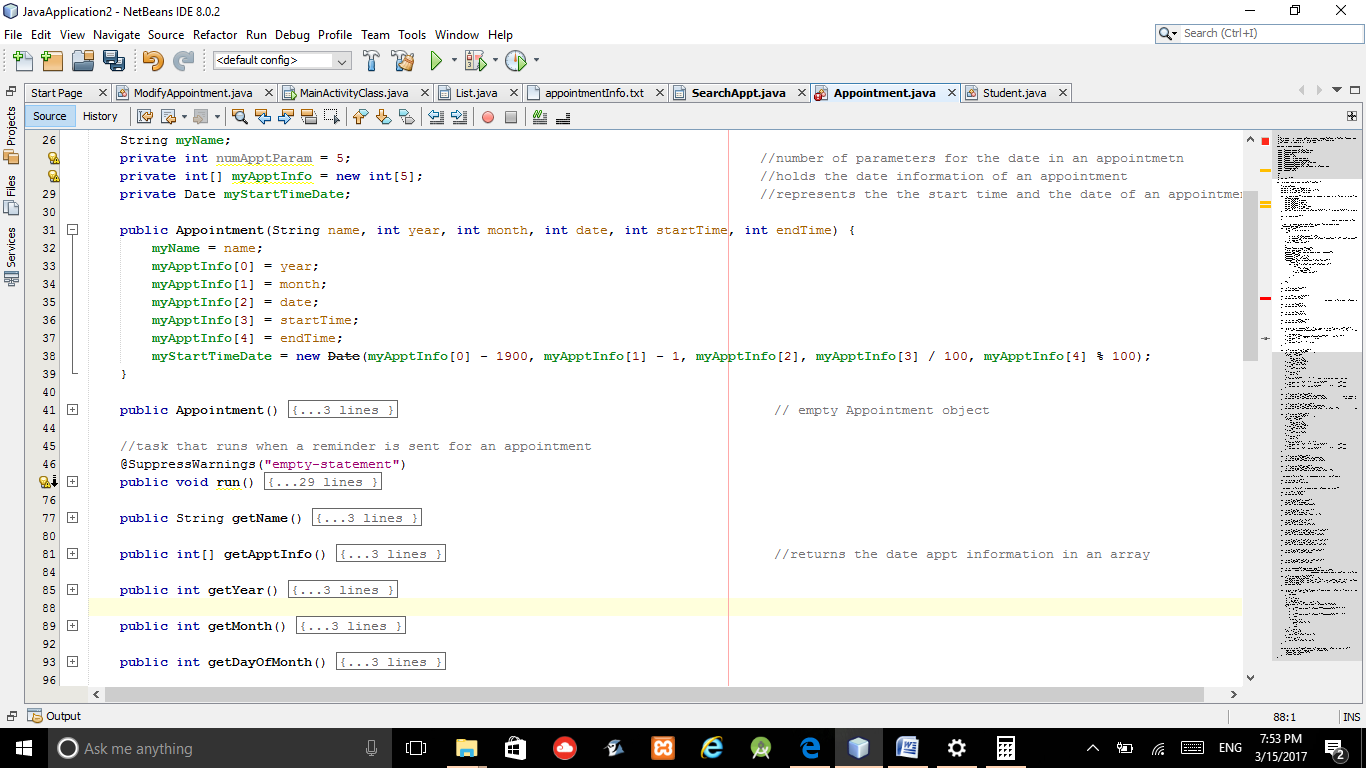
Creating a filewriter for the file automatically clears its contents

**8. Date API**

To represent the start time of an appointment, a Date object was used. The link below was used to learn how to create Date objects. This was the best API to use because when creating a reminder, the schedule method in the Timer API (discussed in next section) requires a Date object

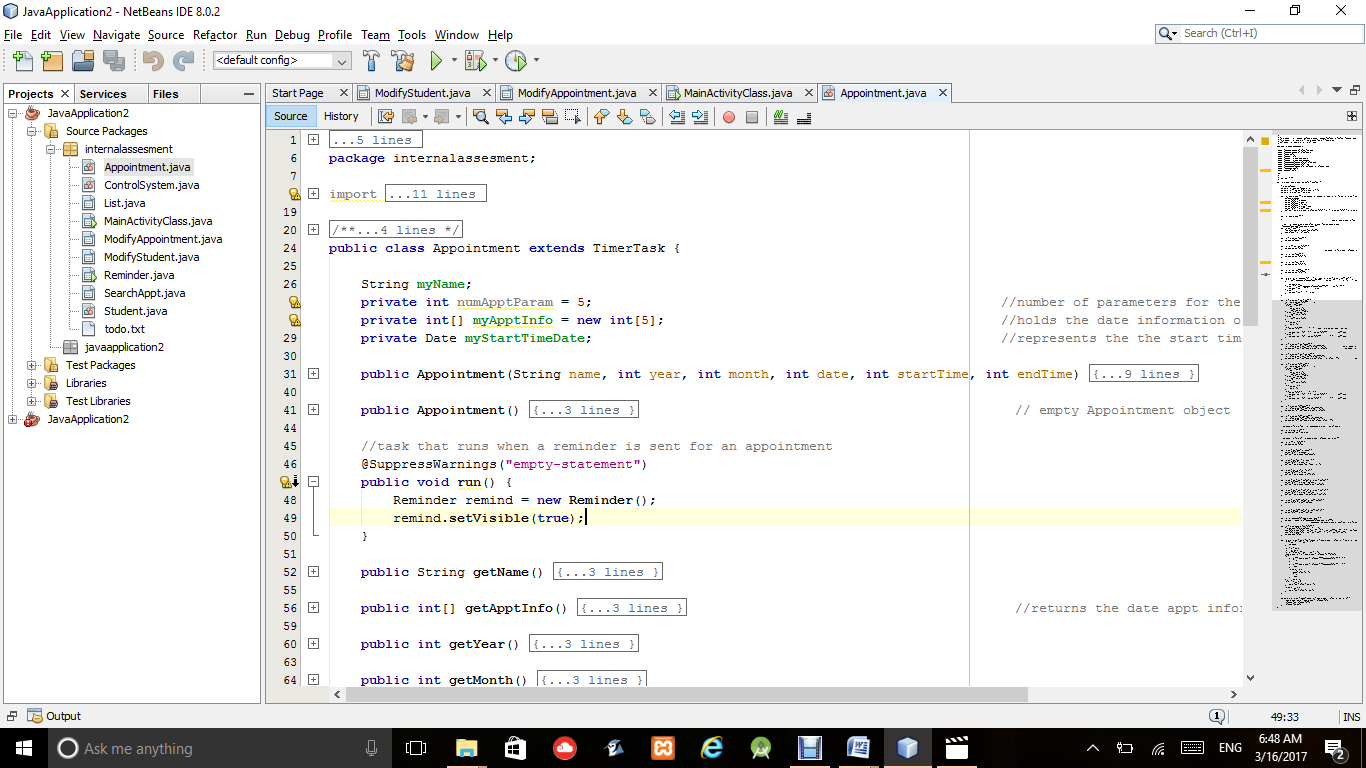
<https://docs.oracle.com/javase/7/docs/api/java/util/Date.html>

Line 38 in the figure below shows an example of how to instantiate a date object



**9. Timer and TimerTask**

The Timer and TimerTask class were needed to set reminders for an appointment a day in advance. Inheritance was used to implement the TimerTask. The run method that was implemented will play an alarm to notify the tutor that she has an appointment tomorrow



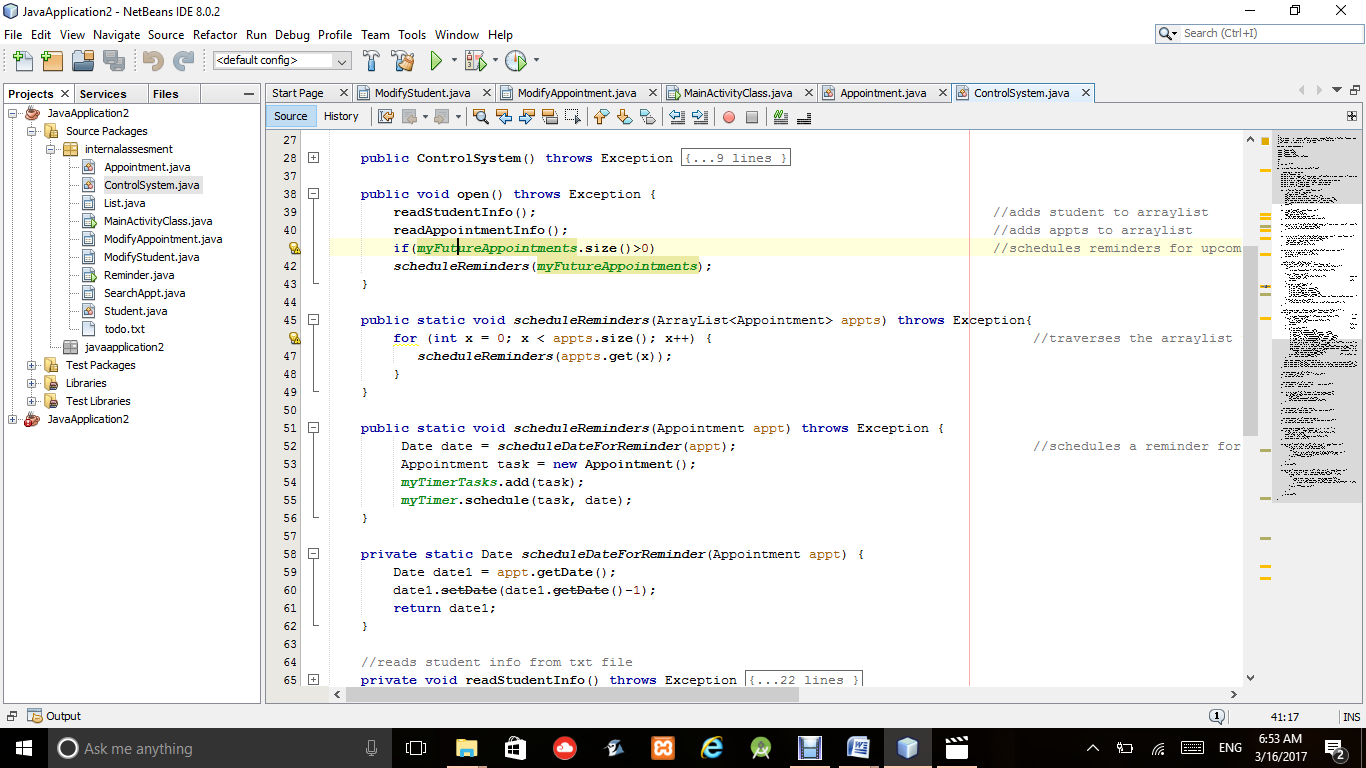
JFrame form that is created and is visible when TimerTask is executed

Implements the run method of TimerTask. The run method will execute when a reminder for an appointment is scheduled

Appointment IS-a TimerTask

The Timer class was used in ControlSystem to schedule the reminder one day before the date of the appointment.

Are there upcoming appointments



Action to execute on that date

Schedules the date for reminder to be 1 day prior to appointment's date

Traverse through the list of upcoming appointments to schedule an a reminder for each one

Schedules reminders for upcoming appointments after the information has been read from text file

**11.Graphical User Interface**

The java.swing package was imported to use the Java Swing library. The following links were used to learn how to implement the different features such as buttons, interactive lists, dialog boxes, error handling, and user inputs.

<https://docs.oracle.com/javase/tutorial/uiswing/><https://netbeans.org/kb/docs/java/gui-functionality.html>

<https://www3.ntu.edu.sg/home/ehchua/programming/java/j4a_gui.html.>