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

```
74      public void addStudentsToList() {
75          String rowData[][] = new String[myStudents.size()][4];
            String columnLabels[] = {"First Name", "Last Name", "Phone number", "Email address"};
77          for (int row = 0; row < myStudents.size(); row++) {
                for (int col = 0; col < 4; col++) {
79                  rowData[row][col] = (myStudents.get(row).getStudentInfo())[col];
80              }
            }
            createTable(rowData, columnLabels);
        }
```

2. 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.

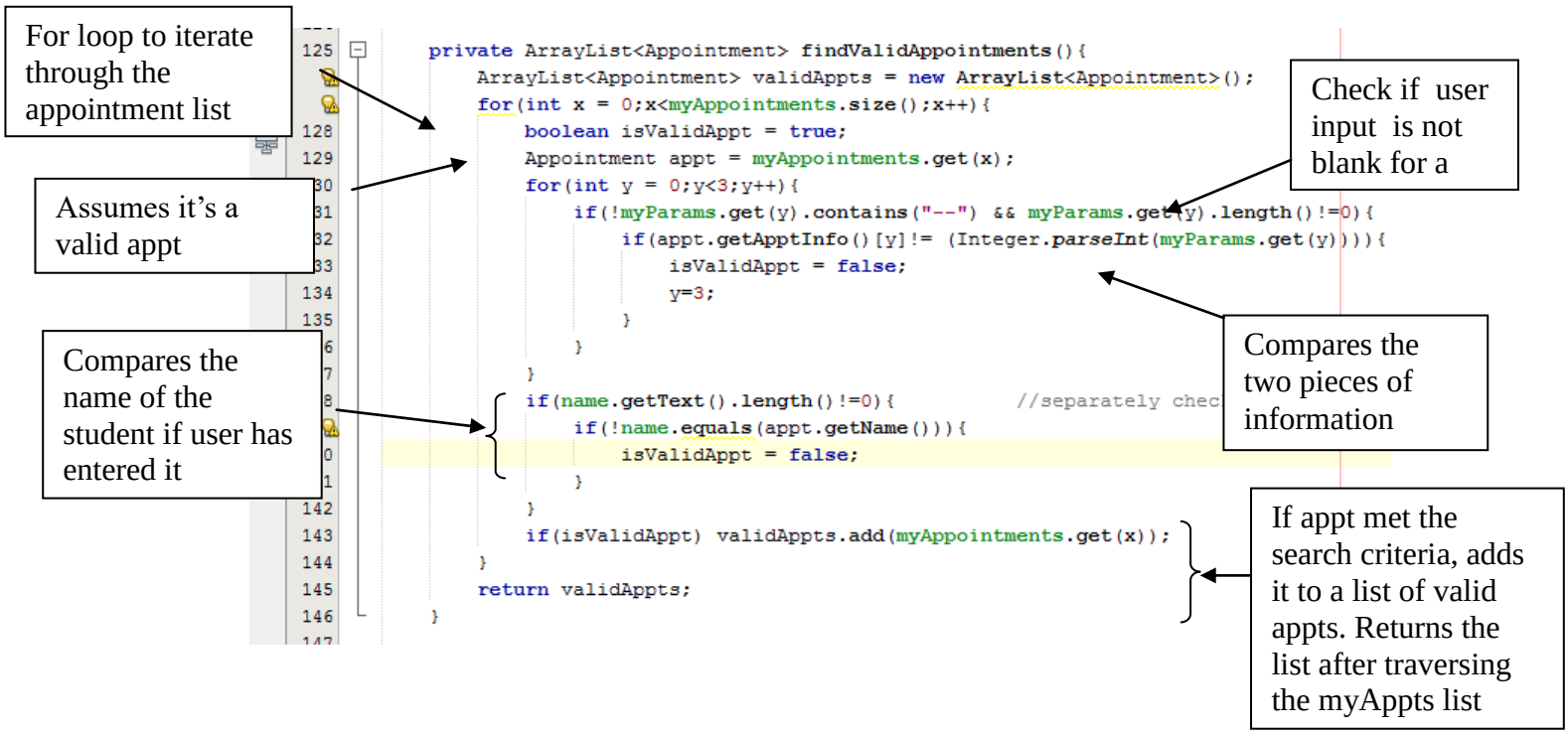
```
17  public class ControlSystem  extends Timer{
18
19      //creates all privates
20      private Scanner in;
        private ArrayList<Student> myStudents;
        private ArrayList<Integer> myYears;
        private ArrayList<Appointment> myFutureAppointments;
        private ArrayList<Appointment> myPastAppointments;
        private ArrayList<Appointment> myAllAppointments;
26      private static Timer myTimer;
```

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.

```
    public ModifyAppointment(ArrayList<Appointment> appointments, String option) {...10 lines }
35
36  public ModifyAppointment(ArrayList<Appointment> appointments, Appointment apptToEdit, String option)
45  /** This method is called from within the constructor to initialize the form ...5 lines */
```

3. Sequential Search

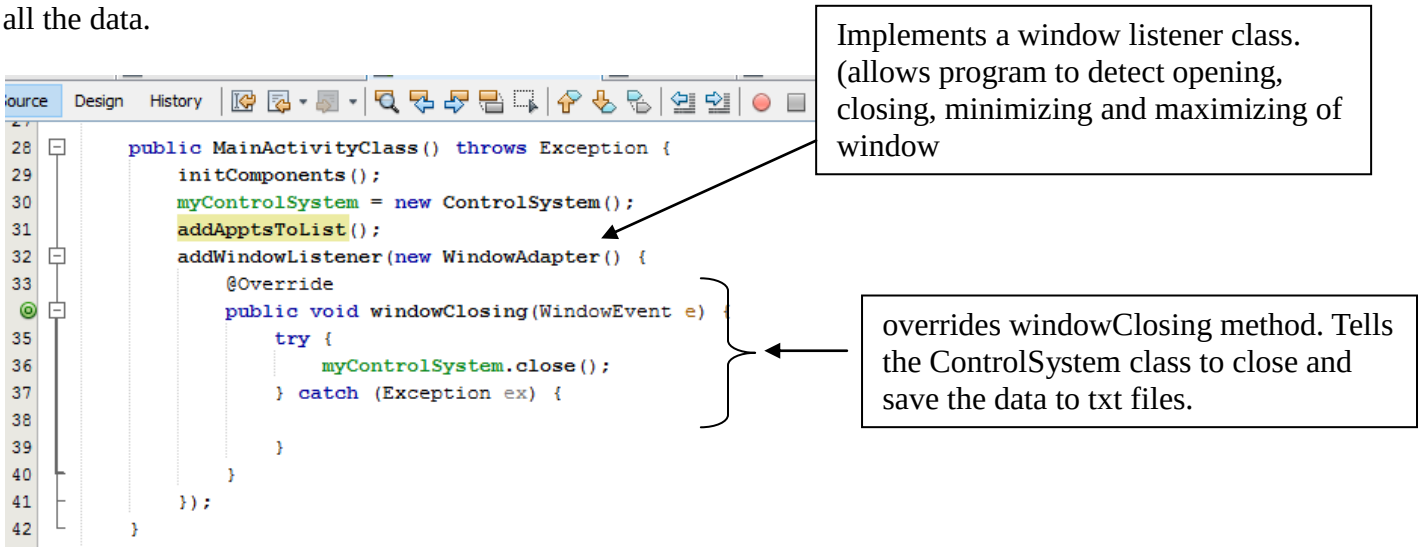
Sequential Search was the algorithm used to search for appointments or students. A for loop was used to iterate through the arraylist of objects. Sequential search is beneficial because the whole arraylist needs to be checked. This is because multiple object could meet the user's required criteria. Consequently, 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.




```
125     private ArrayList<Appointment> findValidAppointments(){
            ArrayList<Appointment> validAppts = new ArrayList<Appointment>();
            for(int x = 0;x<myAppointments.size();x++){
128             boolean isValidAppt = true;
129             Appointment appt = myAppointments.get(x);
130             for(int y = 0;y<3;y++){
131                 if(!myParams.get(y).contains("--") && myParams.get(y).length()!=0){
132                     if(appt.getApptInfo()[y]!= (Integer.parseInt(myParams.get(y)))){
133                         isValidAppt = false;
134                         y=3;
135                     }
6                   }
7               }
8               if(name.getText().length()!=0){          //separately chec
0                   if(!name.equals(appt.getName())){
1                       isValidAppt = false;
142                 }
143             }
144             if(isValidAppt) validAppts.add(myAppointments.get(x));
145         }
146         return validAppts;
147     }
```

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.




```
ource  Design  History
28      public MainActivityClass() throws Exception {
29          initComponents();
30          myControlSystem = new ControlSystem();
31          addApptsToList();
32          addWindowListener(new WindowAdapter() {
33              @Override
                public void windowClosing(WindowEvent e) {
35                  try {
36                      myControlSystem.close();
37                  } catch (Exception ex) {
38
39                  }
40              }
41          });
42      }
```

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.

```
89      public static void quickSort(ArrayList<Student> info, int first,int last){
90          int f = first;
91          int l = last;
92          int midIndex = (first+last)/2;
93
94         Student obj = (Student) info.get(midIndex);
95          do{
96              while(((Student)info.get(f)).compareTo(obj)<0){
97               f++;
98              }
99              while(((Student)info.get(l)).compareTo(obj)>0){
100              l--;
101             }
102
103             if(f<=l){
104                 swap(info,f,l);
105                 f++;
106                 l--;
107             }
108         }while(f<l);
109         if(l>first){
110             quickSort(info,first,l);
111         }
112         if(f<last){
113             quickSort(info,f,last);
114         }
115     }
116
117     private static void swap(ArrayList<Student> info, int x, int y){
118         Student ex = (Student) info.get(x);
119         info.set(x,info.get(y));
120         info.set(y,ex);
121     }
122 }
123
```

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.

```
92      //reads appointment information from "appointmentInfo.txt" file
93      private void readAppointmentInfo() throws Exception {
94          try {
95              in = new Scanner(new File("appointmentInfo.txt"));
96          } catch (Exception e) {
97              File f = new File("appointmentInfo.txt");
98              f.createNewFile();
99              return;
100         }
```

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.

```
281     //returns true if all params are entered
282     private boolean isValidParam() {
            for (int x = 0; x < myParameters.size(); x++) {
                if (myParameters.get(x).equals("---")) {
                    MainActivityClass.displayErrorDialogBox("Please enter all parameters");
                    return false;
                }
            }
289         if(firstName.getText().isEmpty() && lastName.getText().isEmpty()){          //did t
290             MainActivityClass.displayErrorDialogBox("Please enter all parameters"); //error
291             return false;
292         }
293         return true;                                                                //retur
294     }
295
296     //returns true if date is in the future
297     private boolean isValidDate() {
298         LocalDateTime localDate = LocalDateTime.now();                              //
299         Instant instant = localDate.atZone(ZoneId.systemDefault()).toInstant();
300         Date local = Date.from(instant);                                            //creat
301         Date apptDate = createDate();                                               //creat
302         if(local.compareTo(createDate()) < 0)                                        //is t
303             return true;
304         else{
305             MainActivityClass.displayErrorDialogBox("This date has alread passed.");
306             return false;
307         }
308     }
309
```

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

Are both first and last name text fields empty?

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.

Figure 1

```
61  public void open() throws Exception {
62      readStudentInfo();
63      readAppointmentInfo();
```

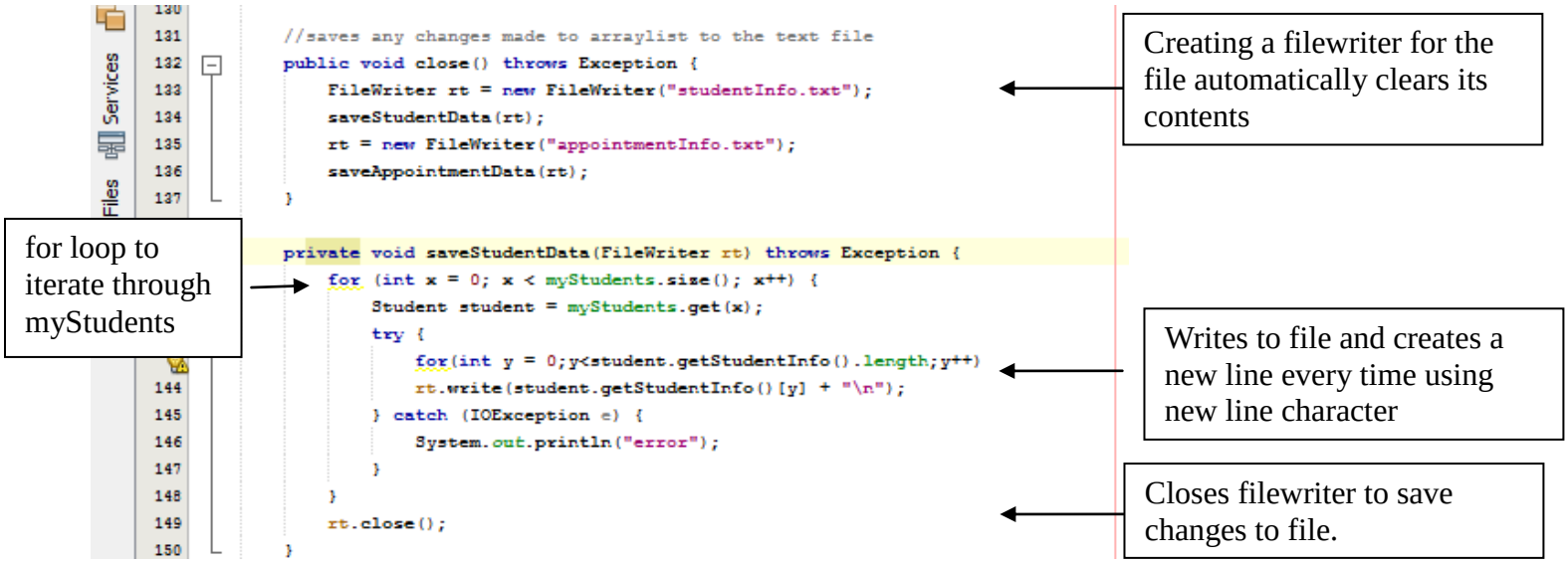
Calls readStudentInfo() and readAppointmentInfo()

```
92      //reads appointment information from "appointmentInfo.txt" file
93      private void readAppointmentInfo() throws Exception {
94          try {
95              in = new Scanner(new File("appointmentInfo.txt"));
96          } catch (Exception e) {
97              File f = new File("appointmentInfo.txt");
98              f.createNewFile();
99              return;
100         }
101         while (in.hasNext()) {
102             String name = in.nextLine();
103             int year = Integer.parseInt(in.nextLine());
104             int month = Integer.parseInt(in.nextLine());
105             int dayOfMonth = Integer.parseInt(in.nextLine());
106             int startTime = Integer.parseInt(in.nextLine());
107             int endTime = Integer.parseInt(in.nextLine());
108             Date apptDate = new Date(year - 1900, month-1, dayOfMonth, startTime/100, startTime%100);
109             LocalDateTime date = LocalDateTime.now();
110             Date currentDate = Date.from(date.atZone(ZoneId.systemDefault()).toInstant());
112             Appointment appt = new Appointment(name, year, month, dayOfMonth, startTime, endTime);
113             if (currentDate.compareTo(apptDate)<0)
114                 myFutureAppointments.add(appt);
115             else myPastAppointments.add(appt);
116             myAllAppointments.add(appt);
        }
```

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

Continues to read file until reaches blank line

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

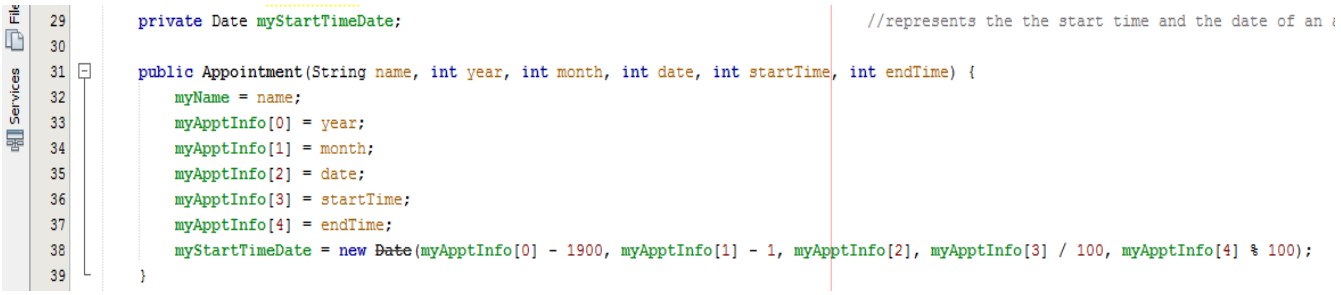


8. Date API

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

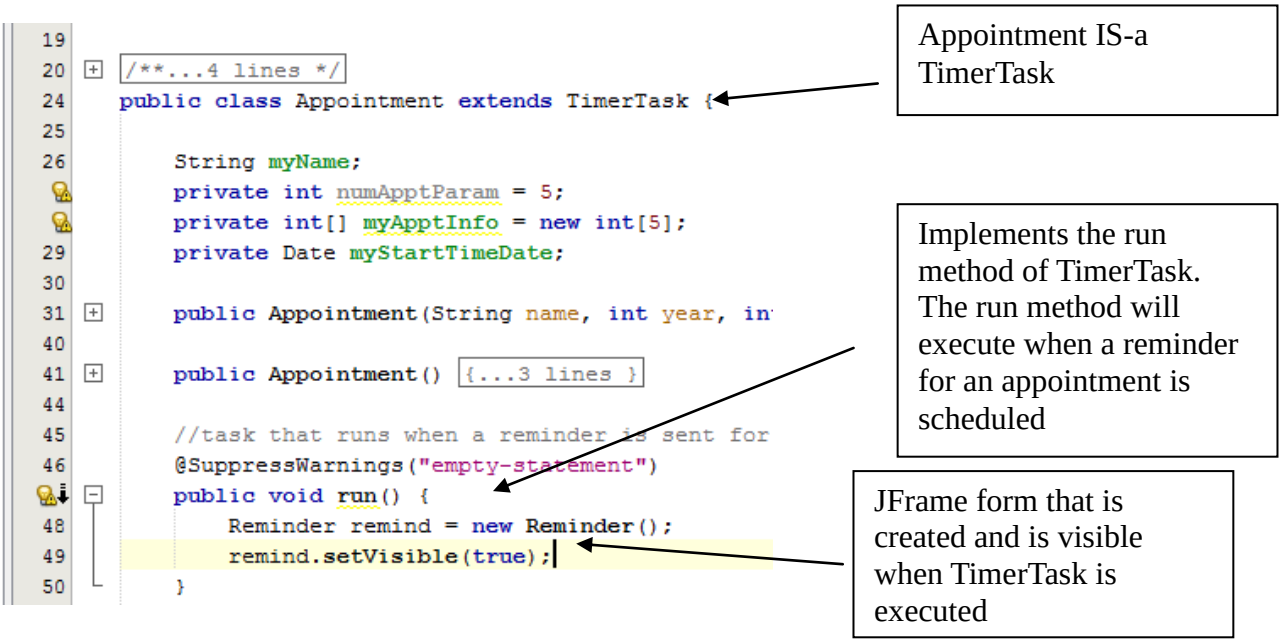
https://docs.oracle.com/javase/7/docs/api/java/util/Date.html

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



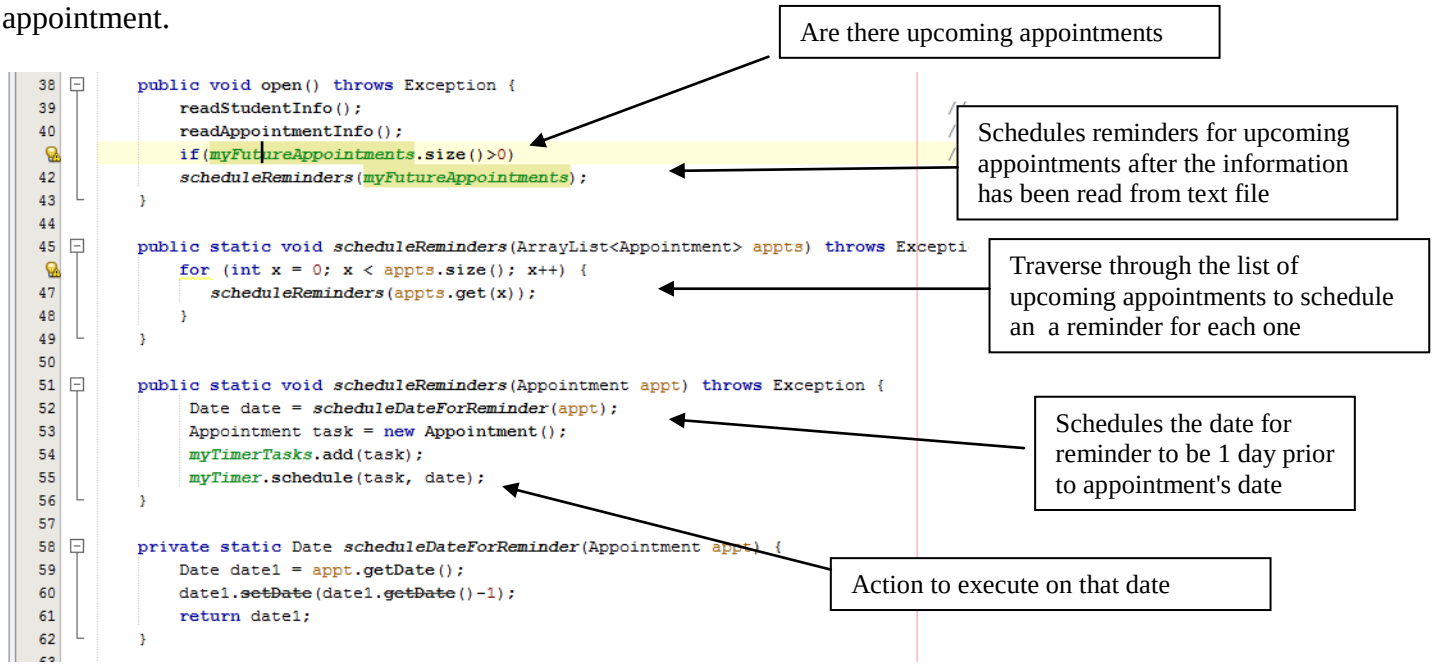
9. Timer and TimerTask

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




```
19
20  /**...4 lines */
24  public class Appointment extends TimerTask {
25
26      String myName;
        private int numApptParam = 5;
        private int[] myApptInfo = new int[5];
29      private Date myStartTimeDate;
30
31      public Appointment(String name, int year, in
40
41      public Appointment() {...3 lines }
44
45      //task that runs when a reminder is sent for
46      @SuppressWarnings("empty-statement")
        public void run() {
48          Reminder remind = new Reminder();
49          remind.setVisible(true);
50      }
```

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




```
38  public void open() throws Exception {
39      readStudentInfo();
40      readAppointmentInfo();
        if(myFutureAppointments.size()>0)
42      scheduleReminders(myFutureAppointments);
43  }
44
45  public static void scheduleReminders(ArrayList<Appointment> appts) throws Excepti
        for (int x = 0; x < appts.size(); x++) {
47          scheduleReminders(appts.get(x));
48      }
49  }
50
51  public static void scheduleReminders(Appointment appt) throws Exception {
52      Date date = scheduleDateForReminder(appt);
53      Appointment task = new Appointment();
54      myTimerTasks.add(task);
55      myTimer.schedule(task, date);
56  }
57
58  private static Date scheduleDateForReminder(Appointment appt) {
59      Date date1 = appt.getDate();
60      date1.setDate(date1.getDate()-1);
61      return date1;
62  }
```

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.