Introduction

I programmed an application which is utilized to store and work with information about a user’s Bollywood classes. The Netbeans IDE and a Java OOP approach is used to make a GUI interface for the user to most easily.

Word Count: 39

Summary of Programming Techniques

1. Flag Values

```java
private String name = "not set yet";
private String email = "not set yet";
private String location = "not set yet";
private String timeZone = "not set yet";
private String phoneNumber = "not set yet";
private String notes = "not set yet";
```

   a. Since the primitive type ‘int’ is only **32 bits**, the phoneNumber would only take up the range of values between -2,147,483,648 to 2,147,483,647
      i. thus either a Long, BigInteger or String could be chosen as a variable type.

2. User Defined Objects from a template class; eg “Student”
3. Parameter Passing
4. Default and Overload constructors

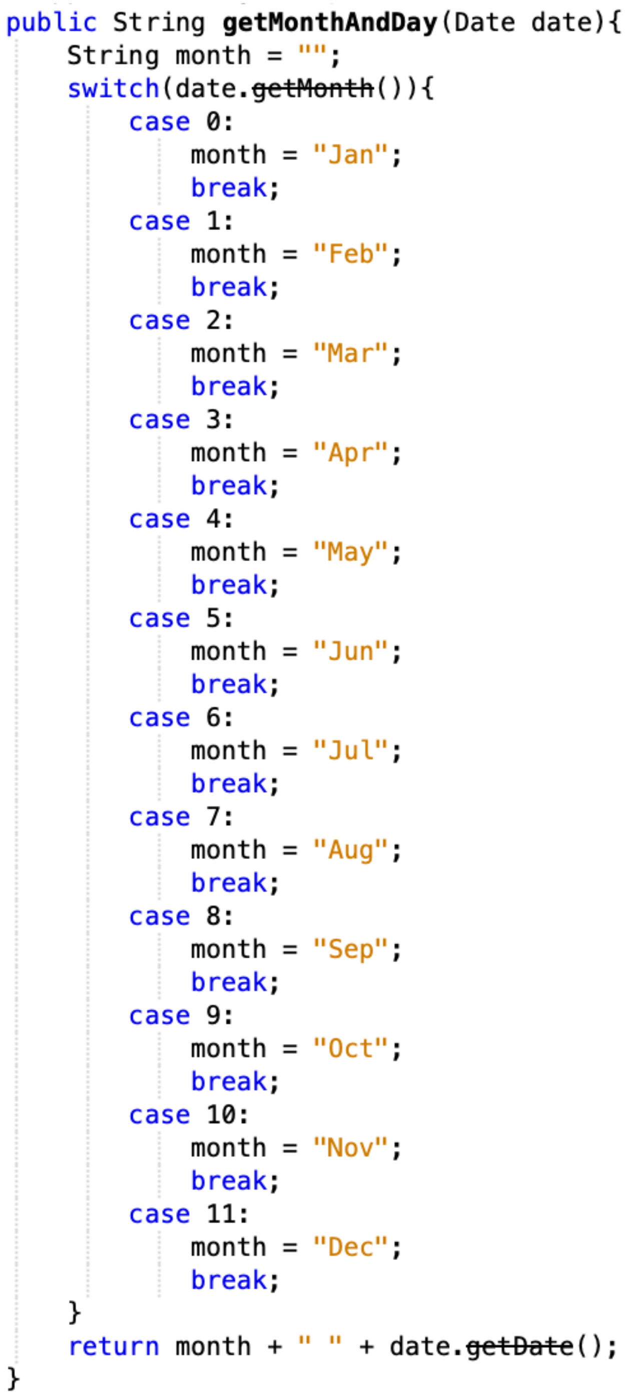
```java
public Student(){

}

public Student(String name, String email, String location, String timeZone, String phoneNumber, String notes){

    this.name = name;
    this.email = email;
    this.location = location;
    this.timeZone = timeZone;
    this.phoneNumber = phoneNumber;
    this.notes = notes;
}
```

5. Encapsulation: Protecting private attributes of a class by using public ‘set’ methods to work on those attributes, and using accessor ‘get’ methods to retrieve attributes.

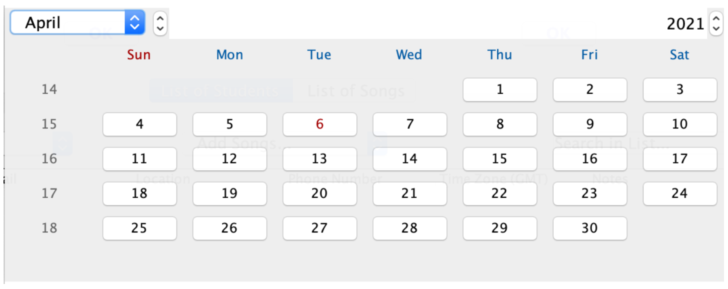
```java
public void setName(String name){
    this.name = name;
}
```

```java
public String getName(){
    return name;
}
```

6. Use of switch/case/break methods
    a. Method returns a value
    b. Switching the format of the date, from the month being an Integer, to something like “Feb” or “Jul”, it is more presentable to the user, especially since month with Integer ‘0’ is Jan, for example, when January is known for being Integer month ‘1’
    c. This implements the library “JCalendar” which gives the user a more friendly interface:
        i. Since the program utilizes “Maven”, the library is imported from a *repository*, with the specified version found as a *dependency* in the **pom.xml** file



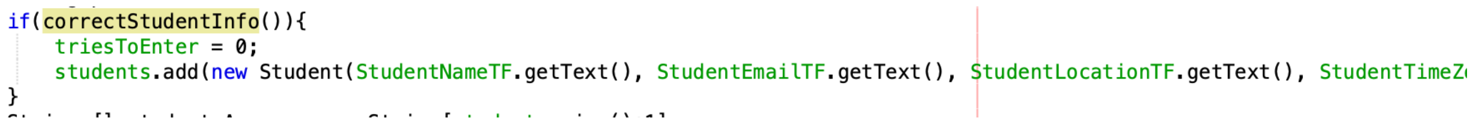
```
public String getMonthAndDay(Date date){
    String month = "";
    switch(date.~~getMonth~~()){
        case 0:
            month = "Jan";
            break;
        case 1:
            month = "Feb";
            break;
        case 2:
            month = "Mar";
            break;
        case 3:
            month = "Apr";
            break;
        case 4:
            month = "May";
            break;
        case 5:
            month = "Jun";
            break;
        case 6:
            month = "Jul";
            break;
        case 7:
            month = "Aug";
            break;
        case 8:
            month = "Sep";
            break;
        case 9:
            month = "Oct";
            break;
        case 10:
            month = "Nov";
            break;
        case 11:
            month = "Dec";
            break;
    }
    return month + " " + date.~~getDate~~();
}
```



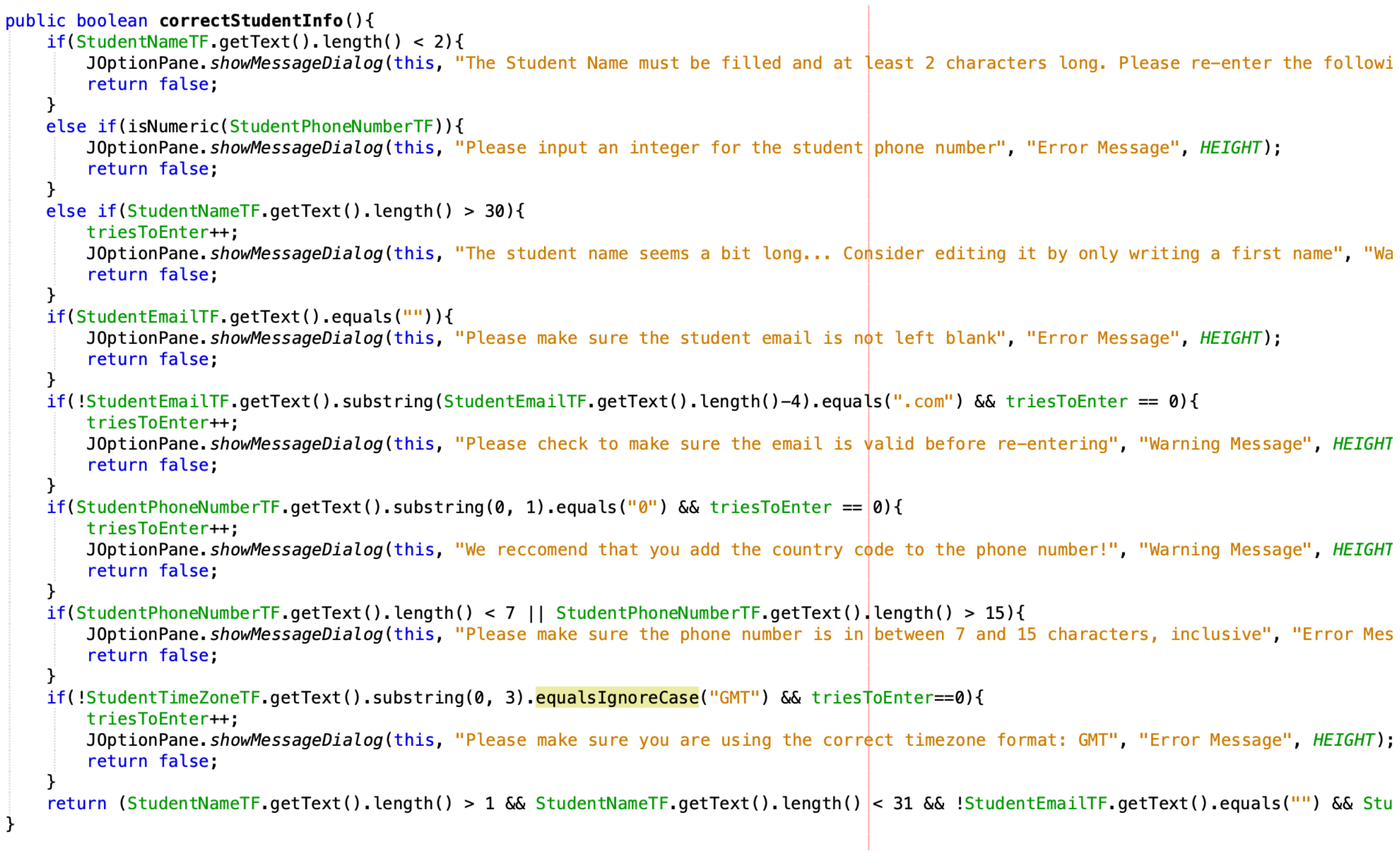
April 2021

	Sun	Mon	Tue	Wed	Thu	Fri	Sat
14					1	2	3
15	4	5	6	7	8	9	10
16	11	12	13	14	15	16	17
17	18	19	20	21	22	23	24
18	25	26	27	28	29	30	

7. “Checker” methods to make sure information is properly inputted
    a. Includes use of single and compound selection (if/else statements)



```
if(correctStudentInfo()){
    triesToEnter = 0;
    students.add(new Student(StudentNameTF.getText(), StudentEmailTF.getText(), StudentLocationTF.getText(), StudentTimeZ
}
```

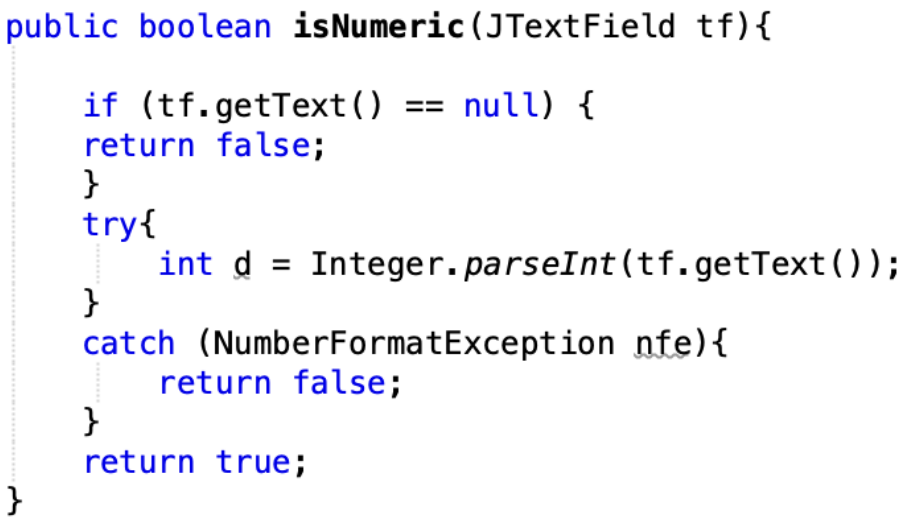


```java
public boolean correctStudentInfo(){
    if(StudentNameTF.getText().length() < 2){
        JOptionPane.showMessageDialog(this, "The Student Name must be filled and at least 2 characters long. Please re-enter the followi
        return false;
    }
    else if(isNumeric(StudentPhoneNumberTF)){
        JOptionPane.showMessageDialog(this, "Please input an integer for the student phone number", "Error Message", HEIGHT);
        return false;
    }
    else if(StudentNameTF.getText().length() > 30){
        triesToEnter++;
        JOptionPane.showMessageDialog(this, "The student name seems a bit long... Consider editing it by only writing a first name", "Wa
        return false;
    }
    if(StudentEmailTF.getText().equals("")){
        JOptionPane.showMessageDialog(this, "Please make sure the student email is not left blank", "Error Message", HEIGHT);
        return false;
    }
    if(!StudentEmailTF.getText().substring(StudentEmailTF.getText().length()-4).equals(".com") && triesToEnter == 0){
        triesToEnter++;
        JOptionPane.showMessageDialog(this, "Please check to make sure the email is valid before re-entering", "Warning Message", HEIGHT
        return false;
    }
    if(StudentPhoneNumberTF.getText().substring(0, 1).equals("0") && triesToEnter == 0){
        triesToEnter++;
        JOptionPane.showMessageDialog(this, "We reccomend that you add the country code to the phone number!", "Warning Message", HEIGHT
        return false;
    }
    if(StudentPhoneNumberTF.getText().length() < 7 || StudentPhoneNumberTF.getText().length() > 15){
        JOptionPane.showMessageDialog(this, "Please make sure the phone number is in between 7 and 15 characters, inclusive", "Error Mes
        return false;
    }
    if(!StudentTimeZoneTF.getText().substring(0, 3).equalsIgnoreCase("GMT") && triesToEnter==0){
        triesToEnter++;
        JOptionPane.showMessageDialog(this, "Please make sure you are using the correct timezone format: GMT", "Error Message", HEIGHT);
        return false;
    }
    return (StudentNameTF.getText().length() > 1 && StudentNameTF.getText().length() < 31 && !StudentEmailTF.getText().equals("") && Stu
}
```

i. A huge part of database programs, such as this one, is checking to make sure *information is valid to halt errors from occurring.*

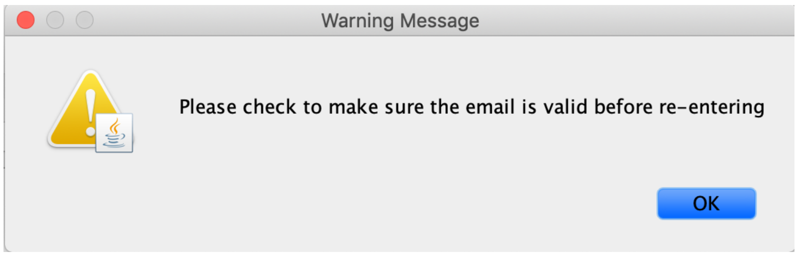
ii. ***'triesToEnter'*** is utilized, since some conditions are just characterized as 'Warning Messages', giving the user a chance to **bypass** the recommendation, if they click are to attempt to add the new Student

b. Use of **try/catch** to ensure the input value is a number.

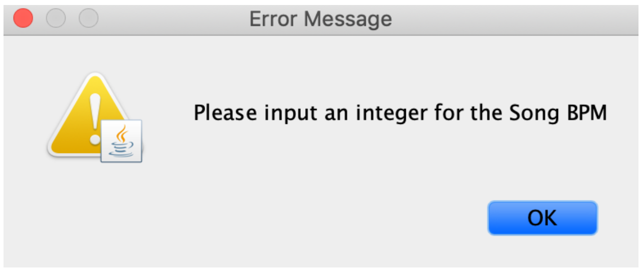


```java
public boolean isNumeric(JTextField tf){

    if (tf.getText() == null) {
    return false;
    }
    try{
        int d = Integer.parseInt(tf.getText());
    }
    catch (NumberFormatException nfe){
        return false;
    }
    return true;
}
```

c. Displays option pane to **communicate** to the user about the input error/ warning message

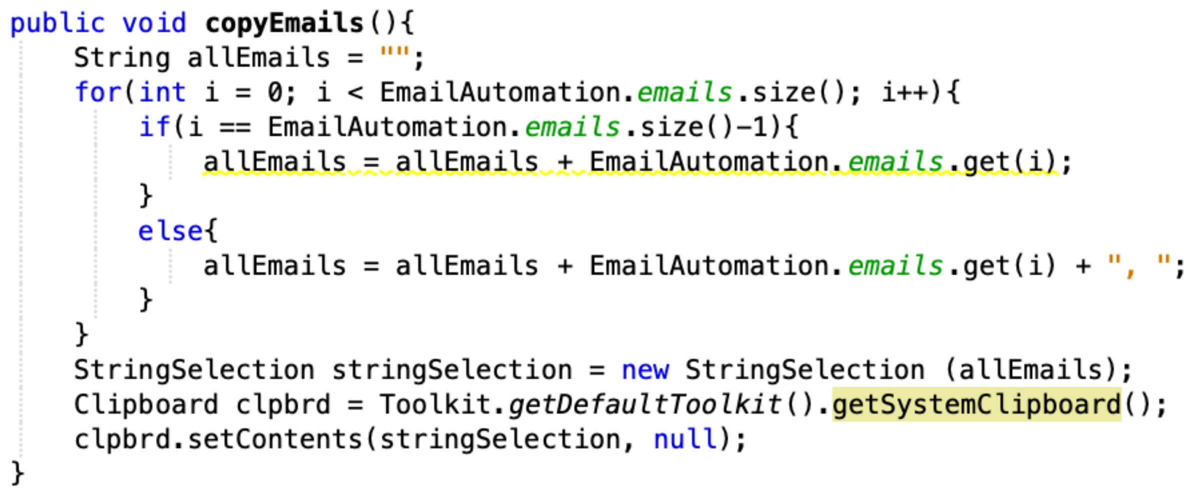


```
Warning Message

Please check to make sure the email is valid before re-entering

OK
```



```
Error Message

Please input an integer for the Song BPM

OK
```

8. For loops. Eg, used to iterate through an array ---- copyemails credit [link]



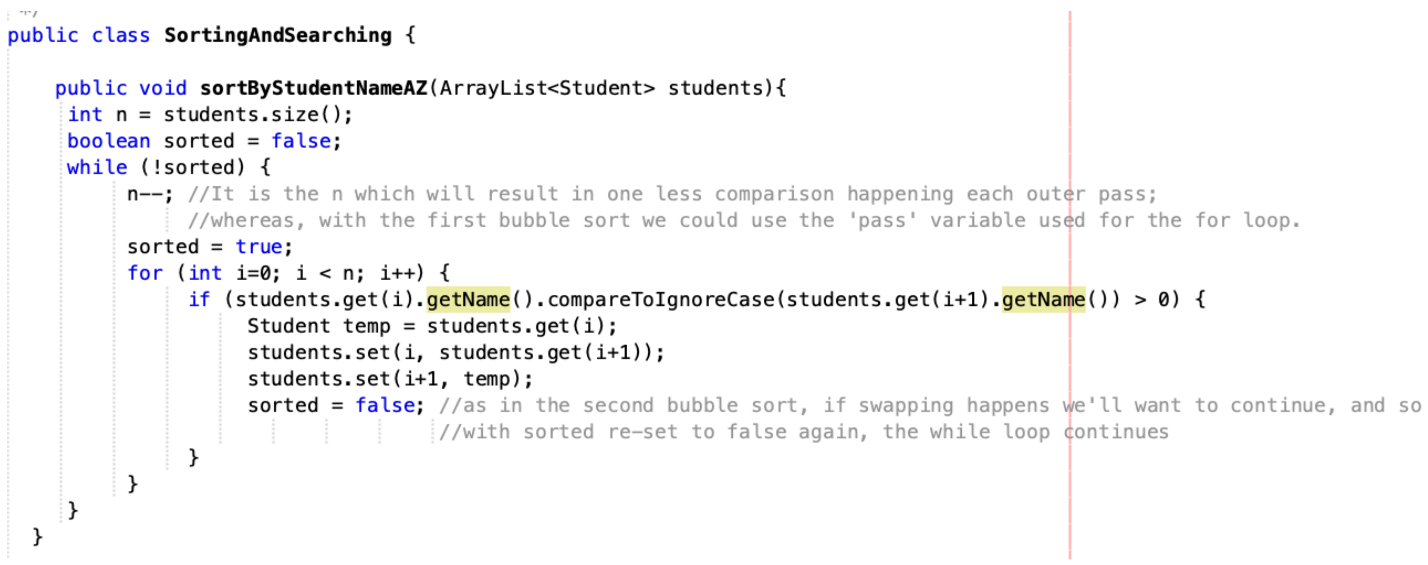
```java
public void copyEmails(){
    String allEmails = "";
    for(int i = 0; i < EmailAutomation.emails.size(); i++){
        if(i == EmailAutomation.emails.size()-1){
            allEmails = allEmails + EmailAutomation.emails.get(i);
        }
        else{
            allEmails = allEmails + EmailAutomation.emails.get(i) + ", ";
        }
    }
    StringSelection stringSelection = new StringSelection (allEmails);
    Clipboard clpbrd = Toolkit.getDefaultToolkit().getSystemClipboard();
    clpbrd.setContents(stringSelection, null);
}
```

The program would originally send an email to all the emails in the ‘emails’ list via Gmail on an automated Chrome browser (use discussed in a later section)

However, due to the client using the email software “Outlook”, copying the emails onto the systems clipboard in a simple copy/paste manner would provide the needed **versatility** for the client to use the emails in their respective context.

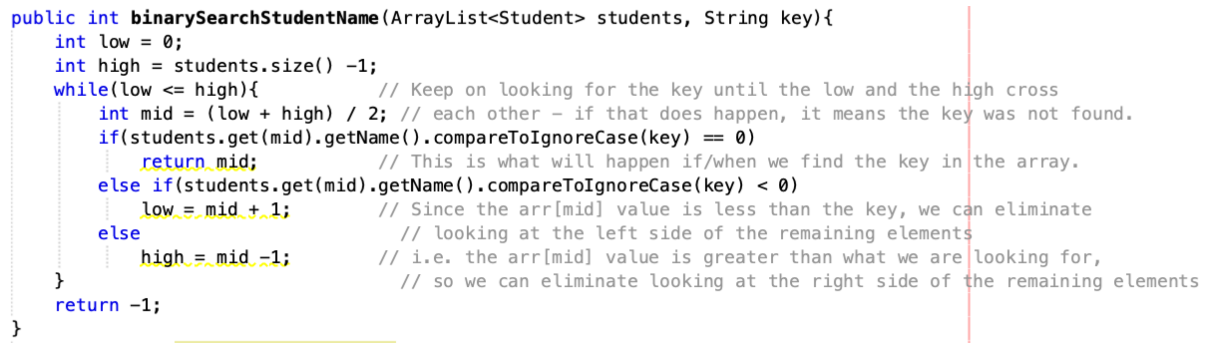
9. Bubble Sort on an Array of Objects based on a Key Attribute

SortingAndSearching.java

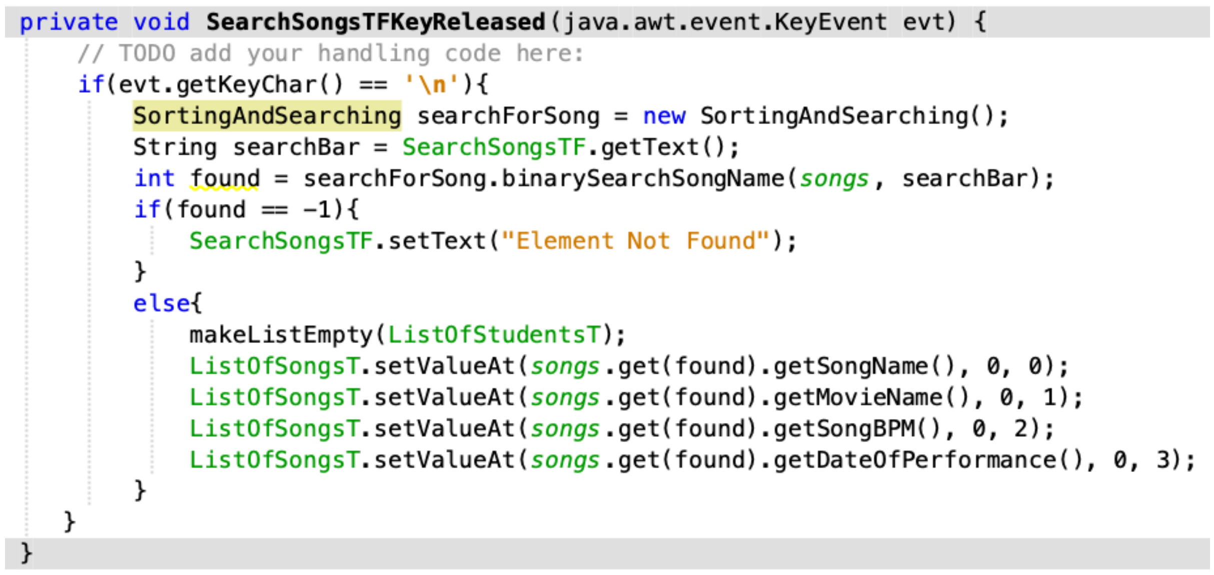


```java
public class SortingAndSearching {

    public void sortByStudentNameAZ(ArrayList<Student> students){
     int n = students.size();
     boolean sorted = false;
     while (!sorted) {
          n--; //It is the n which will result in one less comparison happening each outer pass;
               //whereas, with the first bubble sort we could use the 'pass' variable used for the for loop.
          sorted = true;
          for (int i=0; i < n; i++) {
               if (students.get(i).getName().compareToIgnoreCase(students.get(i+1).getName()) > 0) {
                    Student temp = students.get(i);
                    students.set(i, students.get(i+1));
                    students.set(i+1, temp);
                    sorted = false; //as in the second bubble sort, if swapping happens we'll want to continue, and so
                                    //with sorted re-set to false again, the while loop continues
               }
          }
     }
  }
```

10. Binary Search for the name of a Student/Song



```java
public int binarySearchStudentName(ArrayList<Student> students, String key){
    int low = 0;
    int high = students.size() -1;
    while(low <= high){           // Keep on looking for the key until the low and the high cross
        int mid = (low + high) / 2; // each other - if that does happen, it means the key was not found.
        if(students.get(mid).getName().compareToIgnoreCase(key) == 0)
            return mid;           // This is what will happen if/when we find the key in the array.
        else if(students.get(mid).getName().compareToIgnoreCase(key) < 0)
            low = mid + 1;        // Since the arr[mid] value is less than the key, we can eliminate
        else                        // looking at the left side of the remaining elements
            high = mid -1;        // i.e. the arr[mid] value is greater than what we are looking for,
    }                               // so we can eliminate looking at the right side of the remaining elements
    return -1;
}
```



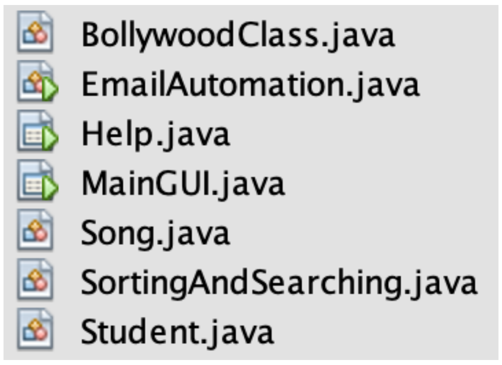
```java
private void SearchSongsTFKeyReleased(java.awt.event.KeyEvent evt) {
    // TODO add your handling code here:
    if(evt.getKeyChar() == '\n'){
        SortingAndSearching searchForSong = new SortingAndSearching();
        String searchBar = SearchSongsTF.getText();
        int found = searchForSong.binarySearchSongName(songs, searchBar);
        if(found == -1){
            SearchSongsTF.setText("Element Not Found");
        }
        else{
            makeListEmpty(ListOfStudentsT);
            ListOfSongsT.setValueAt(songs.get(found).getSongName(), 0, 0);
            ListOfSongsT.setValueAt(songs.get(found).getMovieName(), 0, 1);
            ListOfSongsT.setValueAt(songs.get(found).getSongBPM(), 0, 2);
            ListOfSongsT.setValueAt(songs.get(found).getDateOfPerformance(), 0, 3);
        }
    }
}
```

The following programming techniques will be discussed with examples in later sections:

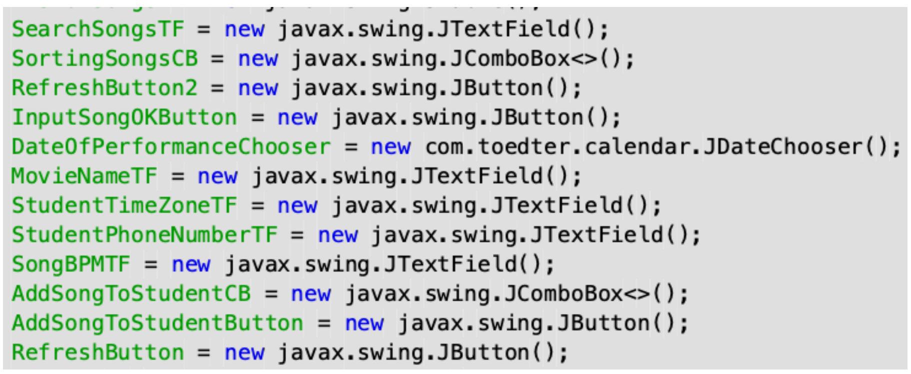
11. Saving/Opening Files
12. Use of ADT's
13. Arrays, Arrays of Objects, 2D Arrays
14. Nested loops
15. Converting a String to int, Character to int, int to String
16. Use of Event Listeners (key, action, mouse)
17. Concurrent Processing via Threads
18. GUI Elements (Tabs, TextFields, Buttons, etc…)
    a. Use of different GUI Layouts.
19. Use of external libraries: JCalendar
20. Use of open source API's for web automation: Selenium.

Word Count: 67 (bullet points excluded)

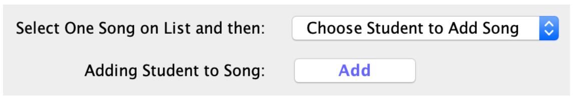
Structure of the Program

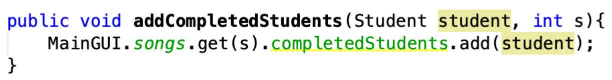

```
BollywoodClass.java
EmailAutomation.java
Help.java
MainGUI.java
Song.java
SortingAndSearching.java
Student.java
```

The MainGUI class is the central feature of the program, using OOP. It is **dependent** and thus **uses and collaborates** with the rest of classes seen above. It also contains all the Java Swing Tools allowing the design interface to function properly.


```
SearchSongsTF = new javax.swing.JTextField();
SortingSongsCB = new javax.swing.JComboBox<>();
RefreshButton2 = new javax.swing.JButton();
InputSongOKButton = new javax.swing.JButton();
DateOfPerformanceChooser = new com.toedter.calendar.JDateChooser();
MovieNameTF = new javax.swing.JTextField();
StudentTimeZoneTF = new javax.swing.JTextField();
StudentPhoneNumberTF = new javax.swing.JTextField();
SongBPMTF = new javax.swing.JTextField();
AddSongToStudentCB = new javax.swing.JComboBox<>();
AddSongToStudentButton = new javax.swing.JButton();
RefreshButton = new javax.swing.JButton();
```

BollywoodClass, Song, and Student are all *“template”* classes which MainGUI calls and makes new *instances* of to store and display. The SortingAndSearching class is used to sort by respective Student and Song elements on the table, as well as search for Student/Song name. The Help class is a JFrame window associated with the MainGUI by **composition**, containing *user documentation* about each tab. Moreover, the Song class **aggregates** the Student class, since a Song has an ArrayList of students who have completed it.



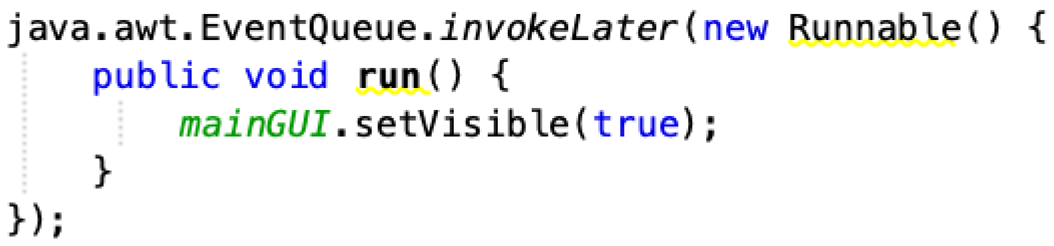

```
public void addCompletedStudents(Student student, int s){
    MainGUI.songs.get(s).completedStudents.add(student);
}
```

The BollywoodClass class **has a** Song **(aggregation)**. Overall, using OOP allows for the program to be more easily *debugged, reusable, visualized, managed, and extensible,* with relationships between the classes making sense in a logical manner.

Word Count: 160

Web Automation Algorithms Explained

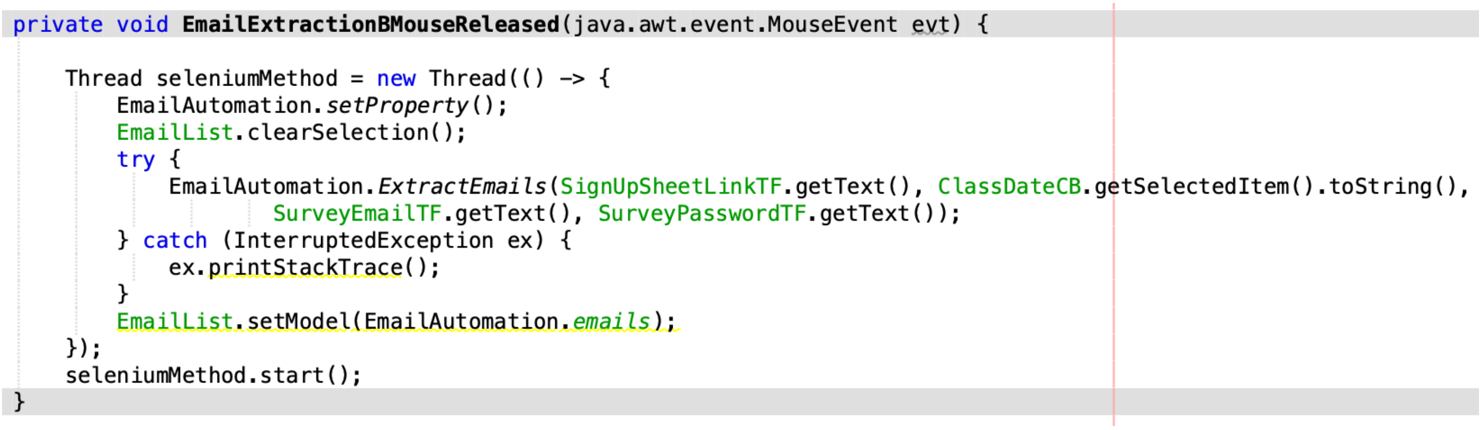
Multithreading allows for code methods to be executed synchronously rather than sequentially. The biggest and easiest to see implementation of this is in the main, where a new instance of “Runnable” is created, which is **inherited** by the class “Thread”.



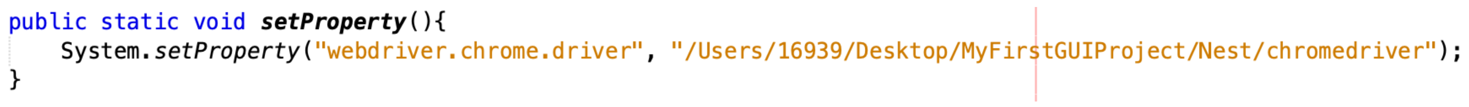
```
java.awt.EventQueue.invokeLater(new Runnable() {
    public void run() {
        mainGUI.setVisible(true);
    }
});
```

This allows for the mainGUI user interface to be responsive meaning the Java Swing tools respond to events in real time. In my case, a Thread is used for two reasons:

1. To start the process to use an automated Chrome browser, extracting emails from results of a ‘survey’ which the user’s students would use to sign up for up-coming Bollywood classes.
2. Updating a progress bar each time another email has been ‘extracted’, to let the user see the process update.



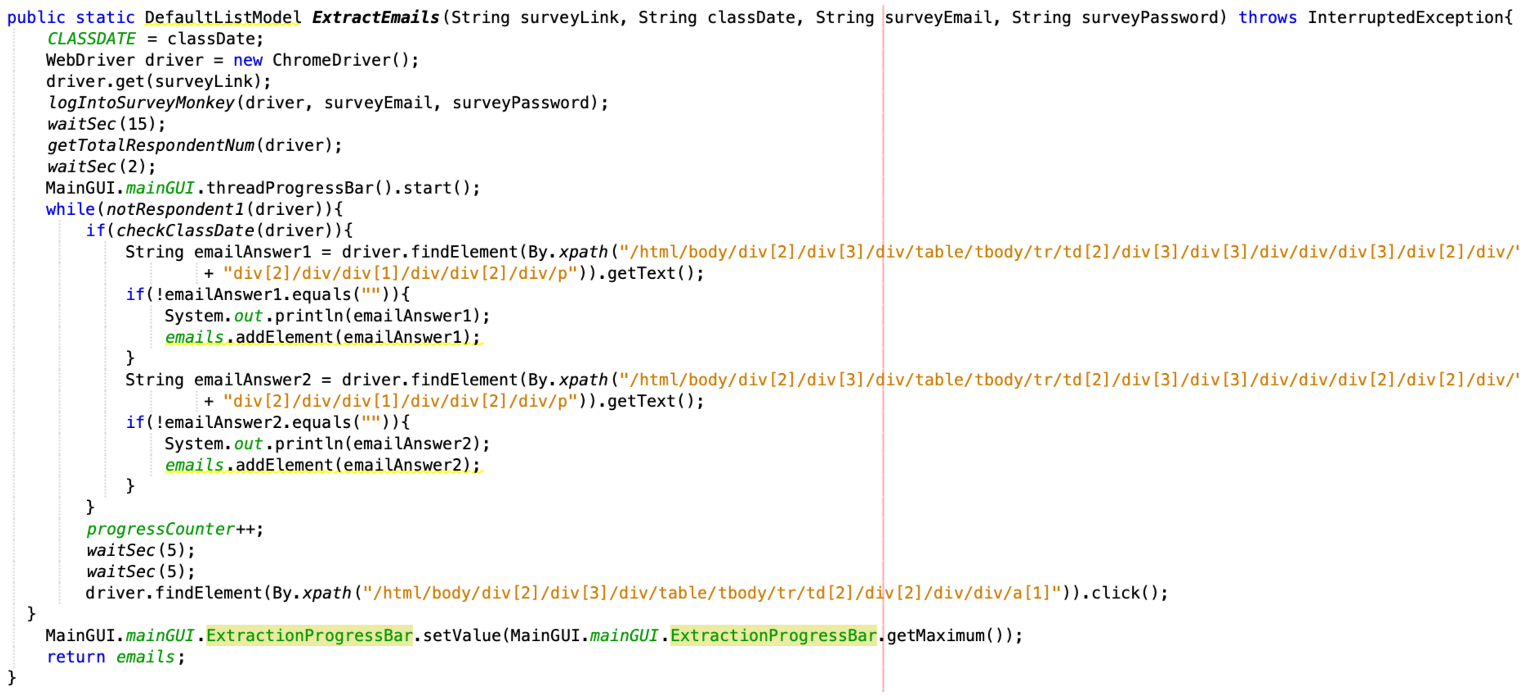
```
private void EmailExtractionBMouseReleased(java.awt.event.MouseEvent evt) {

    Thread seleniumMethod = new Thread(() -> {
        EmailAutomation.setProperty();
        EmailList.clearSelection();
        try {
            EmailAutomation.ExtractEmails(SignUpSheetLinkTF.getText(), ClassDateCB.getSelectedItem().toString(),
                    SurveyEmailTF.getText(), SurveyPasswordTF.getText());
        } catch (InterruptedException ex) {
            ex.printStackTrace();
        }
        EmailList.setModel(EmailAutomation.emails);
    });
    seleniumMethod.start();
}
```



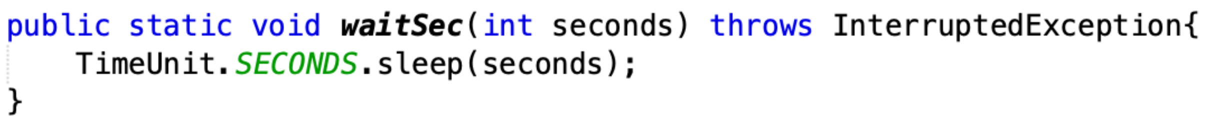
```
public static void setProperty(){
    System.setProperty("webdriver.chrome.driver", "/Users/16939/Desktop/MyFirstGUIProject/Nest/chromedriver");
}
```

Prerequisite Understanding.

- The “waitSec()’ method is used to ensure all web elements are loaded before the program attempts to move on to the next step of code.
    - Positives: limits errors and prevents inconsistencies due to network lag, slow internet connection, etc…
    - Negatives: potentially adds unnecessary ‘extra’ waiting time

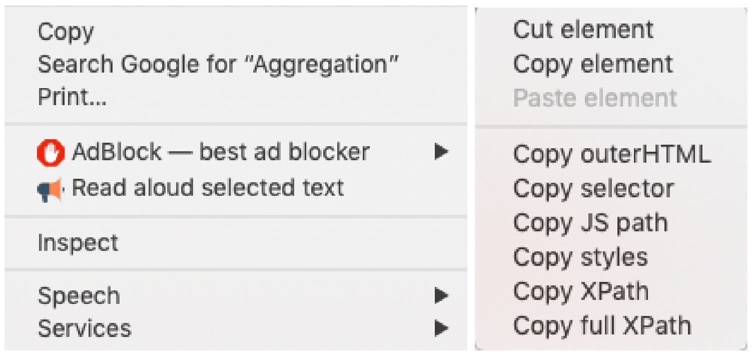


```java
public static DefaultListModel ExtractEmails(String surveyLink, String classDate, String surveyEmail, String surveyPassword) throws InterruptedException{
    CLASSDATE = classDate;
    WebDriver driver = new ChromeDriver();
    driver.get(surveyLink);
    logIntoSurveyMonkey(driver, surveyEmail, surveyPassword);
    waitSec(15);
    getTotalRespondentNum(driver);
    waitSec(2);
    MainGUI.mainGUI.threadProgressBar().start();
    while(notRespondent1(driver)){
        if(checkClassDate(driver)){
            String emailAnswer1 = driver.findElement(By.xpath("/html/body/div[2]/div[3]/div/table/tbody/tr/td[2]/div[3]/div[3]/div/div/div[3]/div[2]/div/"
                    + "div[2]/div/div[1]/div/div[2]/div/p")).getText();
            if(!emailAnswer1.equals("")){
                System.out.println(emailAnswer1);
                emails.addElement(emailAnswer1);
            }
            String emailAnswer2 = driver.findElement(By.xpath("/html/body/div[2]/div[3]/div/table/tbody/tr/td[2]/div[3]/div[3]/div/div/div[2]/div[2]/div/"
                    + "div[2]/div/div[1]/div/div[2]/div/p")).getText();
            if(!emailAnswer2.equals("")){
                System.out.println(emailAnswer2);
                emails.addElement(emailAnswer2);
            }
        }
        progressCounter++;
        waitSec(5);
        waitSec(5);
        driver.findElement(By.xpath("/html/body/div[2]/div[3]/div/table/tbody/tr/td[2]/div[2]/div/div/a[1]")).click();
  }
    MainGUI.mainGUI.ExtractionProgressBar.setValue(MainGUI.mainGUI.ExtractionProgressBar.getMaximum());
    return emails;
}
```



```java
public static void waitSec(int seconds) throws InterruptedException{
    TimeUnit.SECONDS.sleep(seconds);
}
```

- WebElements can be searched through different 'locators' on "Inspect"



  - Name tags, CSS Class, and shown above; outerHTML, selector, JS path, styles
  - The one that is used throughout my program is "*Full XPath*", which is used to navigate through the HTML/XML structure of the page.
    - **Why?** Provides a dynamic way to search for an element on the web page, allowing flexibility for changes in code by the developer.

Psuedocode for 'Extract Emails'

Initial Components:

Make a new object ChromeDriver type WebDriver (opens chrome)
Go to surveyLink using search bar
Log into survey monkey through Gmail (Figure1)

Save the number of total students who responded to the survey (Figure2)

- Used for updating progress bar

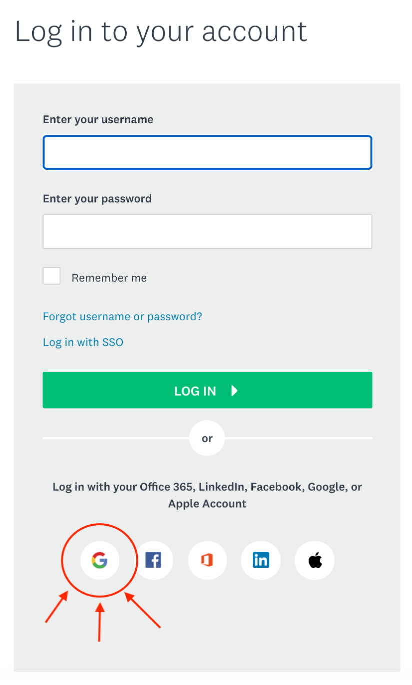
Start progress bar thread in mainGUI (Figure4)

Loop to Get all Valid Emails:

```
While the web page is not yet to the first respondent (Figure5)
        Check if the class date chosen by the student corresponds to the class date on the
        MainGUI chosen by the user (Figure6)
                Use XPath, to find email address
                (Note for this part, the XPath is inconsistent by survey monkey for each respondent,
                shown by the change in one number, and so each time, the program will attempt to
                look for the one element via 2 XPath's to save the email address, and then add it to
                the 'emails' list)
        Add 1 to progress counter
        Click onto the next respondent
(when all respondents have been checked…)
Set progress bar to full (extraction completed)
Return the Default List Model 'emails' to update the JList on the MainGUI.
```

Figure 1

```java
public static void logIntoSurveyMonkey(WebDriver driver, String username, String password) throws InterruptedException{
    waitSec(5);
    driver.findElement(By.xpath("/html/body/div[2]/div[2]/div/div/div[2]/div[1]/div[2]/div/div/p[1]/a")).click();
    waitSec(3);
    driver.findElement(By.xpath("/html/body/div[1]/div[1]/div[2]/div/div[2]/div/div/div[2]/div/div[1]/div/form/span/"
            + "section/div/div/div[1]/div/div[1]/div/div[1]/input")).sendKeys(username + Keys.ENTER); //Putting in email
    waitSec(3);
    driver.findElement(By.xpath("/html/body/div[1]/div[1]/div[2]/div/div[2]/div/div/div[2]/div/div[1]/div/form/span/"
            + "section/div/div/div[1]/div[1]/div/div/div/div[1]/div/div[1]/input")).sendKeys(password + Keys.ENTER);
}
```



Log in to your account

Enter your username

Enter your password

Remember me

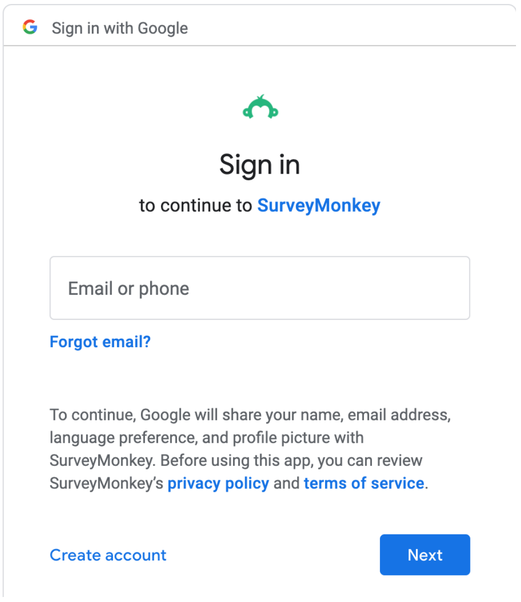
Forgot username or password?

Log in with SSO

LOG IN

or

Log in with your Office 365, LinkedIn, Facebook, Google, or Apple Account



Sign in with Google

Sign in

to continue to SurveyMonkey

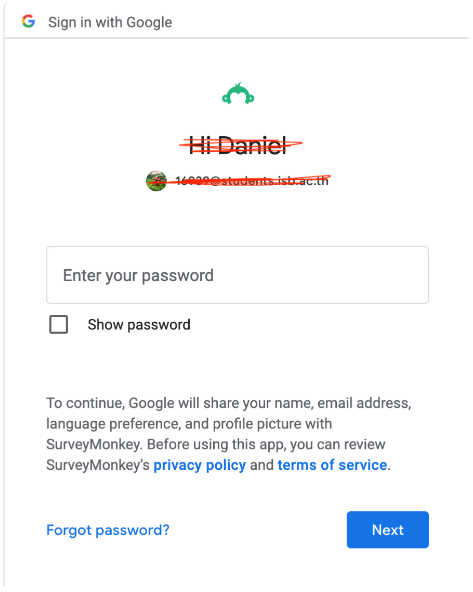
Email or phone

Forgot email?

To continue, Google will share your name, email address, language preference, and profile picture with SurveyMonkey. Before using this app, you can review SurveyMonkey's privacy policy and terms of service.

Create account

Next



Sign in with Google

~~Hi Daniel~~

~~16[illegible]@students.isb.ac.th~~

Enter your password

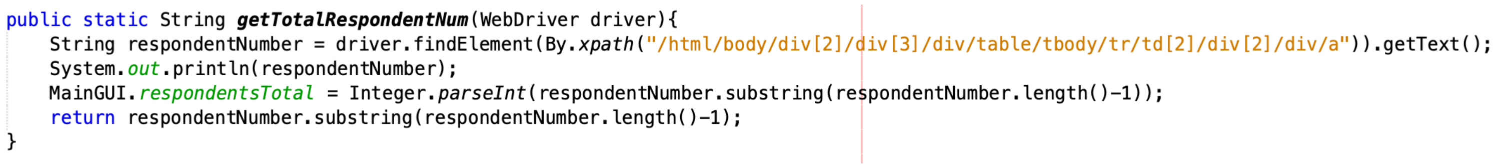
Show password

To continue, Google will share your name, email address, language preference, and profile picture with SurveyMonkey. Before using this app, you can review SurveyMonkey's privacy policy and terms of service.

Forgot password?

Next

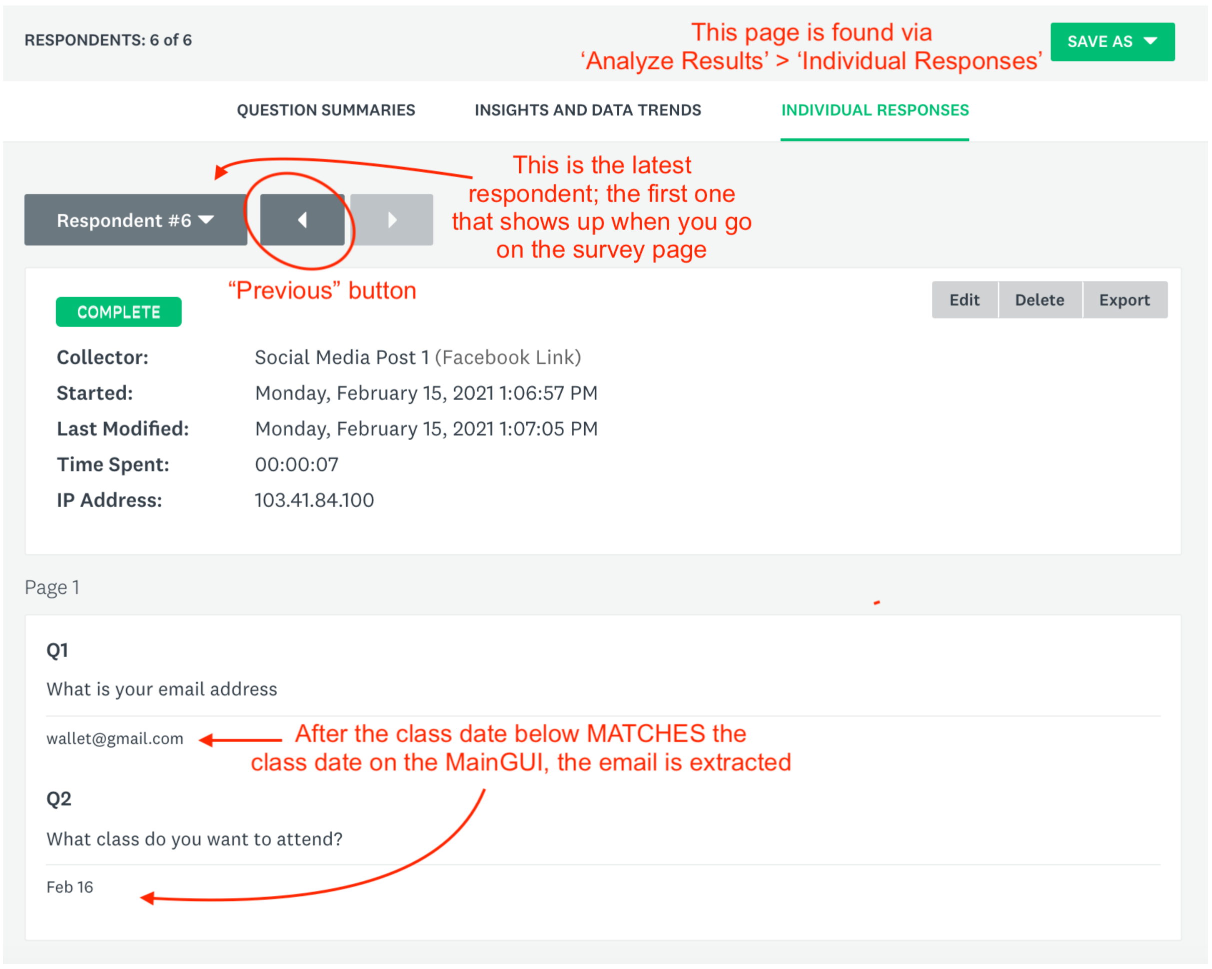
Figure 2



```
public static String getTotalRespondentNum(WebDriver driver){
    String respondentNumber = driver.findElement(By.xpath("/html/body/div[2]/div[3]/div/table/tbody/tr/td[2]/div[2]/div/a")).getText();
    System.out.println(respondentNumber);
    MainGUI.respondentsTotal = Integer.parseInt(respondentNumber.substring(respondentNumber.length()-1));
    return respondentNumber.substring(respondentNumber.length()-1);
}
```

At this point we are on the '**Individual Responses**' page shown on Figure3, with all the elements the program will be working with and looking for.

Figure 3



RESPONDENTS: 6 of 6

This page is found via 'Analyze Results' > 'Individual Responses'

This is the latest respondent; the first one that shows up when you go on the survey page

"Previous" button

Collector: Social Media Post 1 (Facebook Link)
Started: Monday, February 15, 2021 1:06:57 PM
Last Modified: Monday, February 15, 2021 1:07:05 PM
Time Spent: 00:00:07
IP Address: 103.41.84.100

Page 1

Q1

What is your email address

wallet@gmail.com

After the class date below MATCHES the class date on the MainGUI, the email is extracted

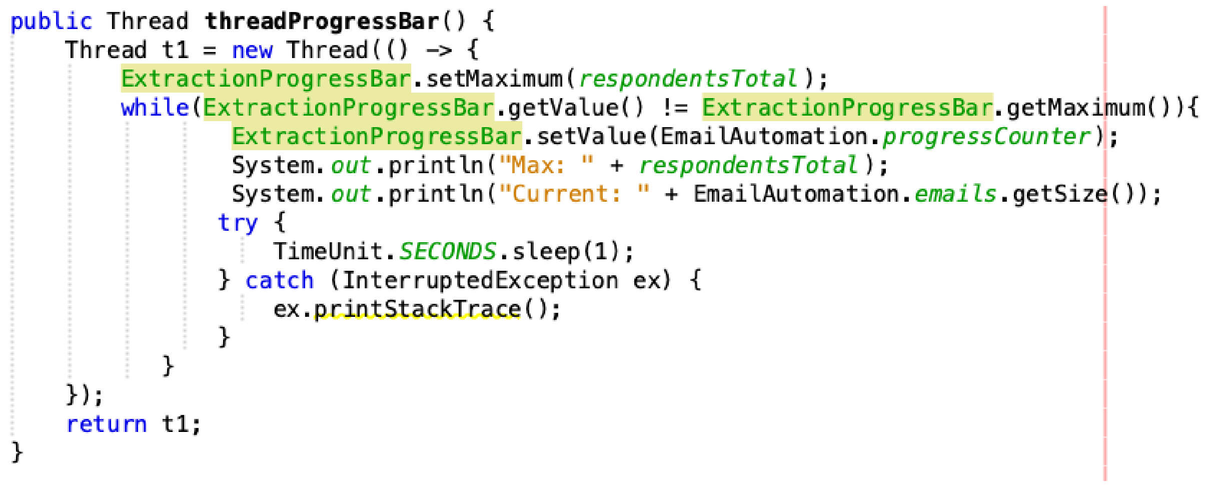
Q2

What class do you want to attend?

Feb 16

Total respondents is the number "**X**" at the end of "Respondent #**X**". This is sent back to the progress bar to set the maximum value of the progress bar. Using progressCounter, the progress bar will update by a fraction of the maximum value each time. In the example of Figure3, the maximum would be '6', and everytime the progress counter changes (by 'progressCounter++'), the progress bar would be filled by ⅙ more.

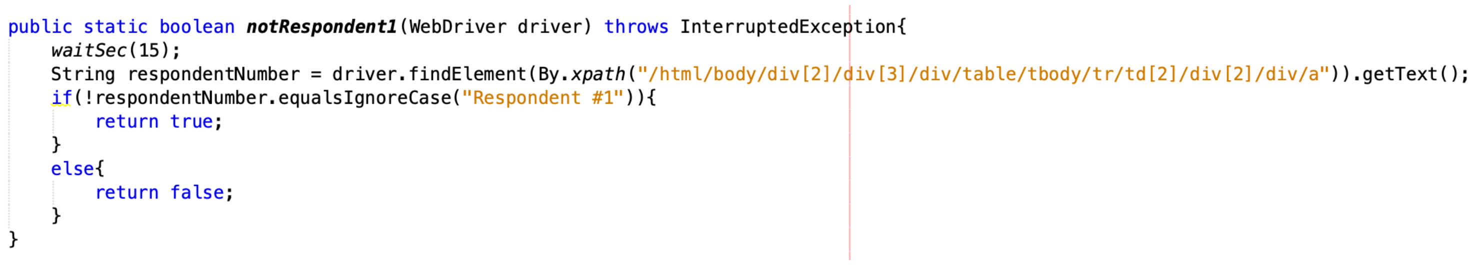
Figure 4



```java
public Thread threadProgressBar() {
    Thread t1 = new Thread(() -> {
        ExtractionProgressBar.setMaximum(respondentsTotal);
        while(ExtractionProgressBar.getValue() != ExtractionProgressBar.getMaximum()){
                ExtractionProgressBar.setValue(EmailAutomation.progressCounter);
                System.out.println("Max: " + respondentsTotal);
                System.out.println("Current: " + EmailAutomation.emails.getSize());
            try {
                TimeUnit.SECONDS.sleep(1);
            } catch (InterruptedException ex) {
                ex.printStackTrace();
            }
        }
    });
    return t1;
}
```

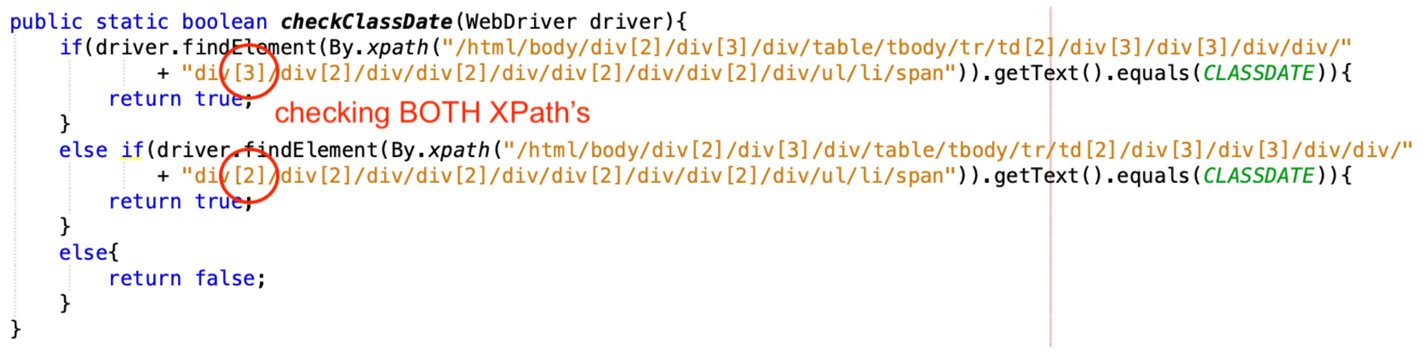
Note that a **thread** is used, since the update for progress bars would have been put on a stack of processes, waiting for the Selenium thread to finish, rather than updating concurrently and in live time.

Figure 5



```java
public static boolean notRespondent1(WebDriver driver) throws InterruptedException{
    waitSec(15);
    String respondentNumber = driver.findElement(By.xpath("/html/body/div[2]/div[3]/div/table/tbody/tr/td[2]/div[2]/div/a")).getText();
    if(!respondentNumber.equalsIgnoreCase("Respondent #1")){
        return true;
    }
    else{
        return false;
    }
}
```

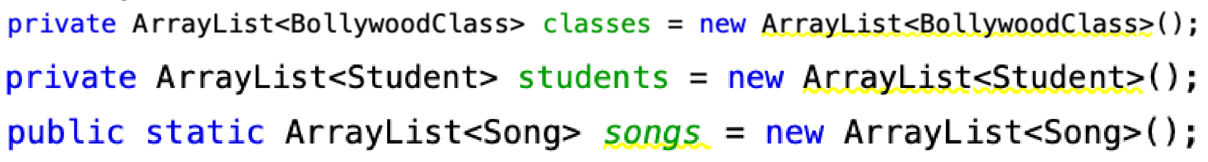
Figure 6



```java
public static boolean checkClassDate(WebDriver driver){
    if(driver.findElement(By.xpath("/html/body/div[2]/div[3]/div/table/tbody/tr/td[2]/div[3]/div[3]/div/div/"
            + "div[3]/div[2]/div/div[2]/div/div[2]/div/div[2]/div/ul/li/span")).getText().equals(CLASSDATE)){
        return true;
    }
    else if(driver.findElement(By.xpath("/html/body/div[2]/div[3]/div/table/tbody/tr/td[2]/div[3]/div[3]/div/div/"
            + "div[2]/div[2]/div/div[2]/div/div[2]/div/div[2]/div/ul/li/span")).getText().equals(CLASSDATE)){
        return true;
    }
    else{
        return false;
    }
}
```

Data Structures Used

ArrayList used to store new instances of the objects from the template classes (below), containing the attributes in the overload constructor of each class.



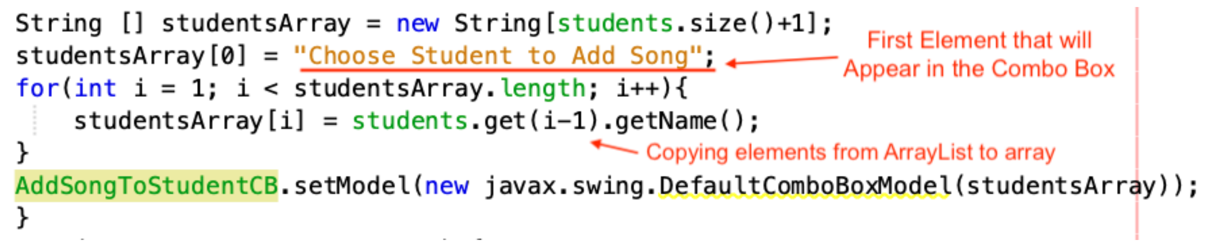
```java
private ArrayList<BollywoodClass> classes = new ArrayList<BollywoodClass>();
private ArrayList<Student> students = new ArrayList<Student>();
public static ArrayList<Song> songs = new ArrayList<Song>();
```

Linked lists are utilized, with each node having a payload of a String containing a coordinate of a grid, discussed later.

```
public LinkedList<String> selectedGridButtonList = new LinkedList<String>();
```

ADT's allow data to be kept in a sequential order, accessed via an index, as well as grow dynamically in size and thus is memory efficient compared to a static array which is seldom used.

However, an array of Strings is used as the model for combo boxes, since the first element displayed explains the use of the particular combo box.



```
String [] studentsArray = new String[students.size()+1];
studentsArray[0] = "Choose Student to Add Song";
for(int i = 1; i < studentsArray.length; i++){
    studentsArray[i] = students.get(i-1).getName();
}
AddSongToStudentCB.setModel(new javax.swing.DefaultComboBoxModel(studentsArray));
}
```

First Element that will Appear in the Combo Box
Copying elements from ArrayList to array

Also, there is use of a 2D array, array of arrays, of objects; a JButton and JToggleButton 2D array, used to emulate and store the grid boxes of a 15 x 15 size grid.

```
public static JButton[][] jbuttongrid = new JButton[15][15];
public static JToggleButton[][] jtogglegrid = new JToggleButton[15][15];
```

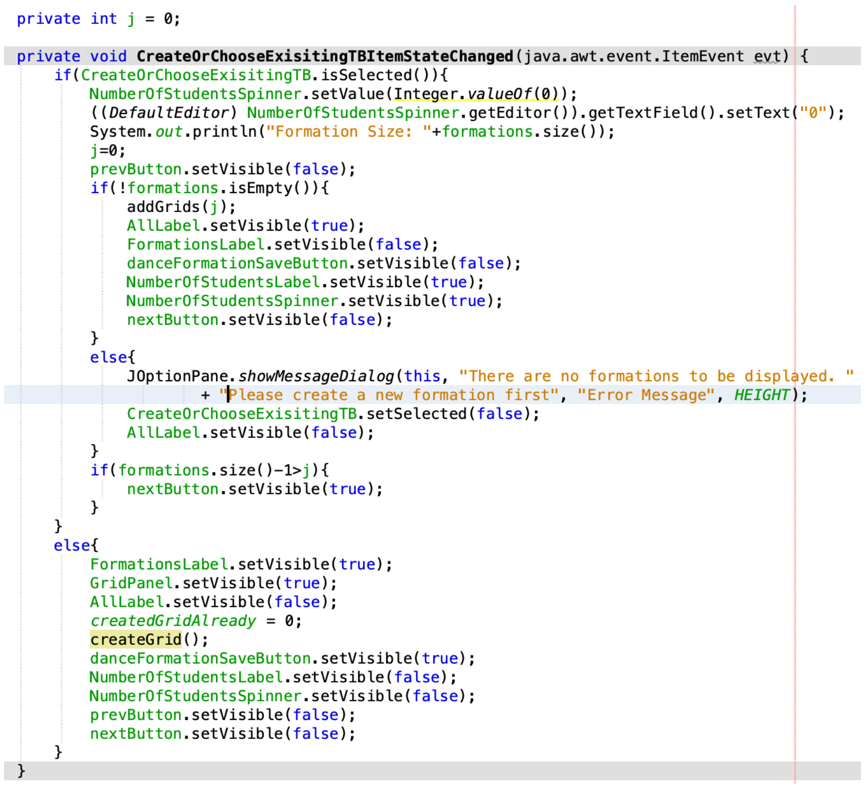
Word Count: 319 (excludes bullet points)

Main Unique Algorithms

Explanation of Dance Formation tab:

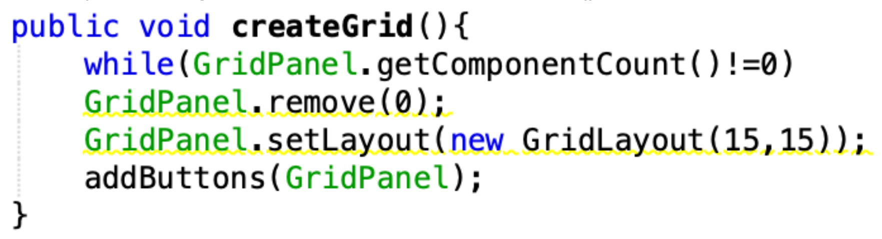
The toggle button on the bottom left toggles between 'creating' and 'choosing' a dance formation, each containing different actions:

Create a New Dance Formation
Choose an Existing Dance Formation

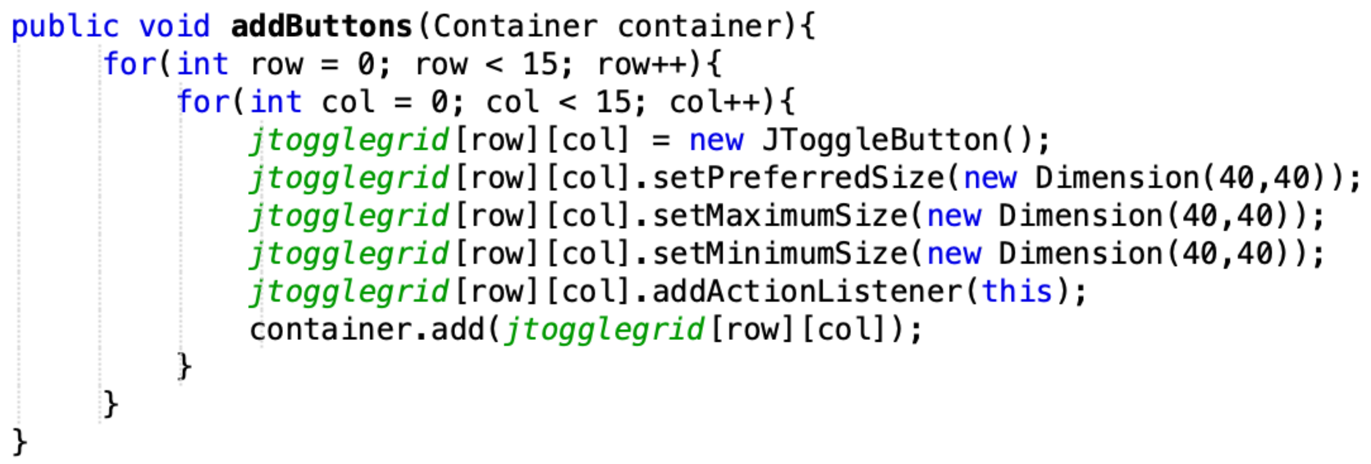


```
private int j = 0;

private void CreateOrChooseExisitingTBItemStateChanged(java.awt.event.ItemEvent evt) {
    if(CreateOrChooseExisitingTB.isSelected()){
        NumberOfStudentsSpinner.setValue(Integer.valueOf(0));
        ((DefaultEditor) NumberOfStudentsSpinner.getEditor()).getTextField().setText("0");
        System.out.println("Formation Size: "+formations.size());
        j=0;
        prevButton.setVisible(false);
        if(!formations.isEmpty()){
            addGrids(j);
            AllLabel.setVisible(true);
            FormationsLabel.setVisible(false);
            danceFormationSaveButton.setVisible(false);
            NumberOfStudentsLabel.setVisible(true);
            NumberOfStudentsSpinner.setVisible(true);
            nextButton.setVisible(false);
        }
        else{
            JOptionPane.showMessageDialog(this, "There are no formations to be displayed. "
                    + "Please create a new formation first", "Error Message", HEIGHT);
            CreateOrChooseExisitingTB.setSelected(false);
            AllLabel.setVisible(false);
        }
        if(formations.size()-1>j){
            nextButton.setVisible(true);
        }
    }
    else{
        FormationsLabel.setVisible(true);
        GridPanel.setVisible(true);
        AllLabel.setVisible(false);
        createdGridAlready = 0;
        createGrid();
        danceFormationSaveButton.setVisible(true);
        NumberOfStudentsLabel.setVisible(false);
        NumberOfStudentsSpinner.setVisible(false);
        prevButton.setVisible(false);
        nextButton.setVisible(false);
    }
}
```

When you first go on the tab, the “createGrid()” method is called.



```
public void createGrid(){
    while(GridPanel.getComponentCount()!=0)
    GridPanel.remove(0);
    GridPanel.setLayout(new GridLayout(15,15));
    addButtons(GridPanel);
}
```



```
public void addButtons(Container container){
     for(int row = 0; row < 15; row++){
         for(int col = 0; col < 15; col++){
             jtogglegrid[row][col] = new JToggleButton();
             jtogglegrid[row][col].setPreferredSize(new Dimension(40,40));
             jtogglegrid[row][col].setMaximumSize(new Dimension(40,40));
             jtogglegrid[row][col].setMinimumSize(new Dimension(40,40));
             jtogglegrid[row][col].addActionListener(this);
             container.add(jtogglegrid[row][col]);
         }
     }
}
```

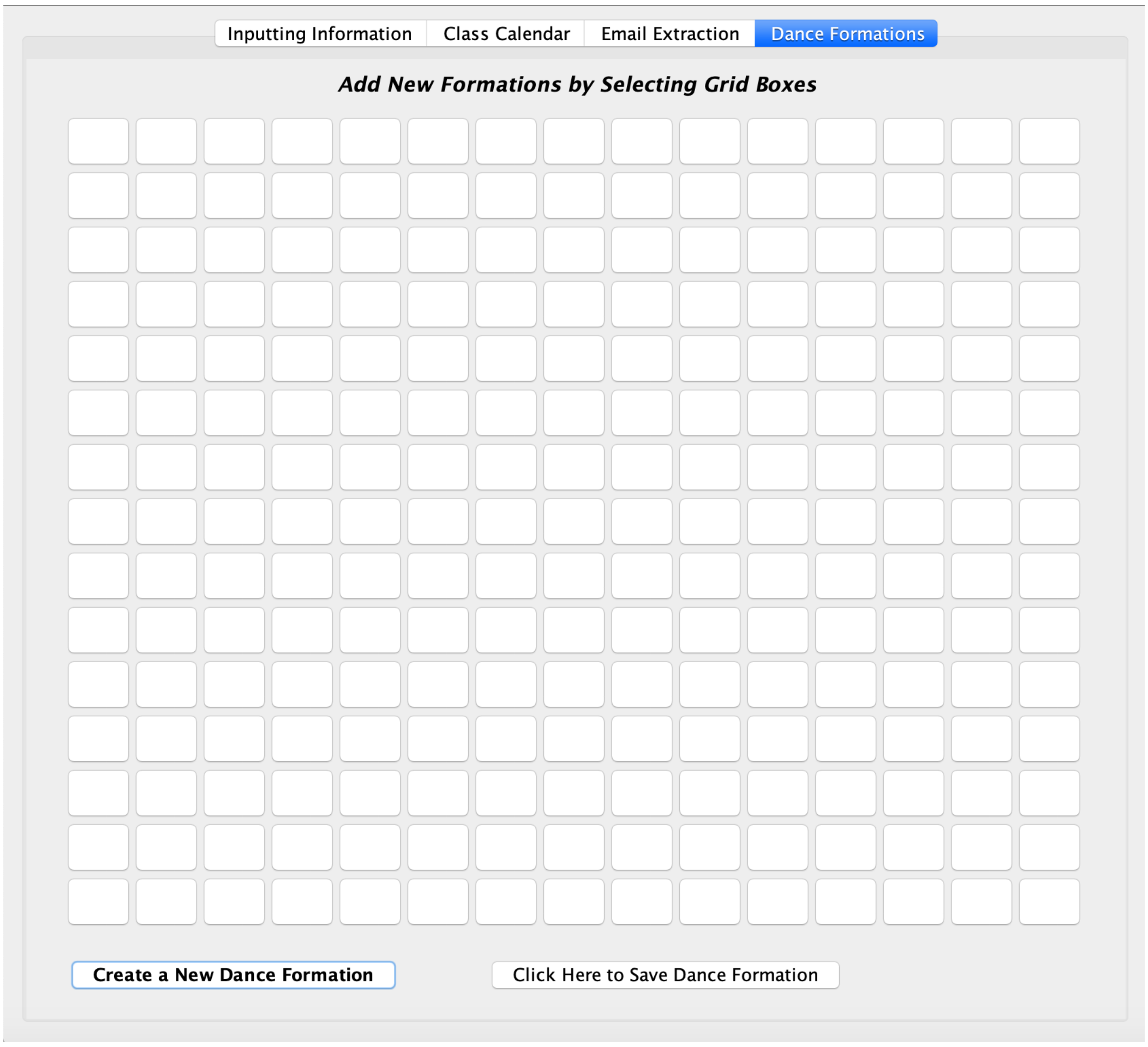
A JPanel is created within the tabbed pane, with a **Grid Layout** manager. There is a specified number of rows and columns (15x15). This layout is *suitable* for the users purpose because:

1. The grid will represent a stage, of which each toggle button within it represents a student, and thus a dance formation can be created.
2. The GridLayout container divides its space into equal-sized rectangles, with each component (TButton) placed in a single rectangle.

Pseudocode for createGrid()

```
Remove all components from GridPanel
Set layout to GridLayout of dimension 15 x 15
Loop through the rows in GridPanel AND ‘jtogglegrid’ 2D array of JToggleButtons
    Loop through the columns in GridPanel AND ‘jtogglegrid’ 2D array of JToggleButtons
        Initialize a new instance of JToggleButton in the 2D array
        Set preferred, max, minimum size to 40,40
        Note that the height of the buttons will actually be set automatically to fill the
        panel, however it is the width that is important to be set.
        Add action listener
        Add the instance of the button to the container (GridPanel)
```

See the result below.



Add New Formations by Selecting Grid Boxes

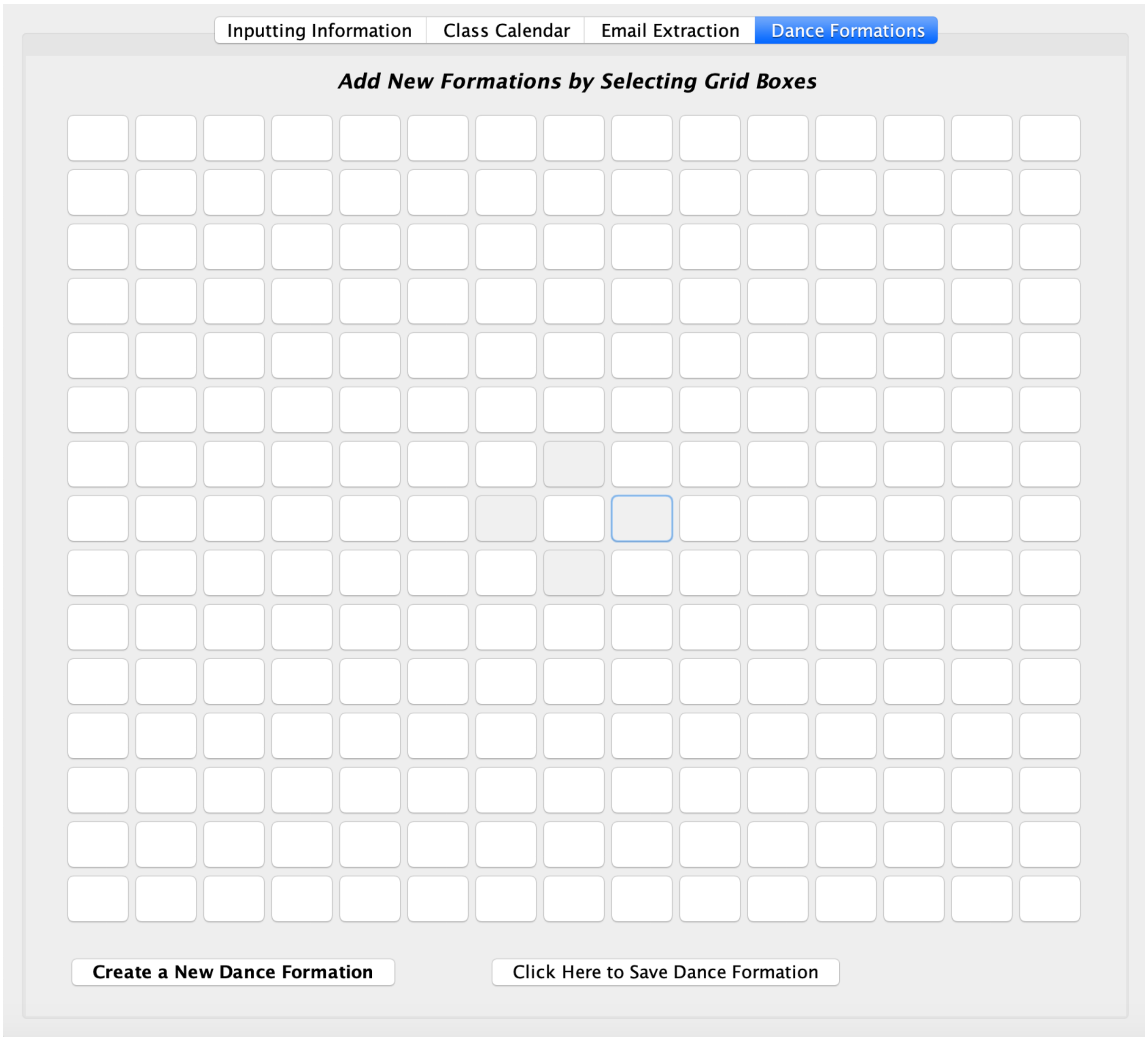
Now, the action listener will wait for the user to click the buttons.

```java
@Override
public void actionPerformed(ActionEvent e) {
    boolean selected = true;
    for(Integer j = 0; j < 15; j++){
        for(Integer k = 0; k < 15; k++){
            if (e.getSource() == jtogglegrid[j][k]){
                if(selectedGridButtonList.size()>0){
                    for(int check = 0; check < selectedGridButtonList.size(); check++){
                        if(selectedGridButtonList.get(check).equals(j.toString() + "," + k.toString())){
                            selectedGridButtonList.remove(check);
                            System.out.println("Element is Removed");
                            selected = false;
                        }
                    }
                }
                else if(selectedGridButtonList.isEmpty()){
                        selectedGridButtonList.add(j.toString() + "," + k.toString());
                        System.out.println(selectedGridButtonList.getLast());
                        System.out.println("First Element is Added");
                        selected = false;
                }
                if(selected){
                    selectedGridButtonList.add(j.toString() + "," + k.toString());
                    System.out.println(selectedGridButtonList.getLast());
                    System.out.println("Element is added");
                }
            }
        }
    }
}
```

Originally, I was under the notion that I could use the action listener and simply save the value of the variables 'row' and 'col', dictating which button had been pressed. However, I came to the realization that since the original nested loops had already been completed, the value of 'row' and 'col' would have already been 15.

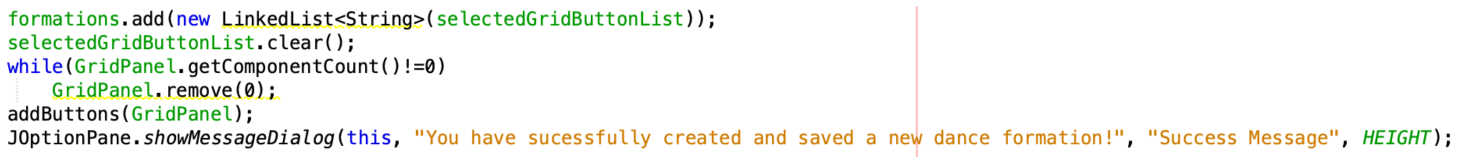
So, after trial and error, I found that I could loop through all the jtogglebuttons again, each time an action was performed, and compare it to **e.getSource()** → gets the object component of which the event occurred**.** Then, if the **respective button and source matched,** its 'coordinate' was saved in a Linked List in the format → ***row* + *","* + *col*** such as "**14, 7**" OR "**1, 2**" OR "**9, 13**"

Complications with this step arised too, since if the user had already pressed the button, toggling it '**on**', then pressing the button again would toggle it '**off**' on the GUI, yet unwillingly add another instance of the same button to the Linked List.


Inputting Information
Class Calendar
Email Extraction
Dance Formations
Add New Formations by Selecting Grid Boxes
Create a New Dance Formation
Click Here to Save Dance Formation

So, a 'checker' was added checking the existing clicked buttons in 'selectedGridButtonList' to make sure that there was no overlap. If there was, that element would be removed since the user had unselected it from the formation.

When the ‘Click Here to Save Dance Formation’ button is clicked:



```
formations.add(new LinkedList<String>(selectedGridButtonList));
selectedGridButtonList.clear();
while(GridPanel.getComponentCount()!=0)
    GridPanel.remove(0);
addButtons(GridPanel);
JOptionPane.showMessageDialog(this, "You have sucessfully created and saved a new dance formation!", "Success Message", HEIGHT);
```

- The reason a LinkedList of Strings was utilized, rather than creating another array of toggle buttons, was for the **ease of saving files later**.
    - Since the *FileWriter* class utilized can only write characters from a String, it would easily be able to **save** the coordinates of each and every button in each respective formation, ready to be opened and interpreted by a *BufferedReader.*

Choosing an Existing Dance Formation:

The spinner will always be set to the default value 0; showing all formations

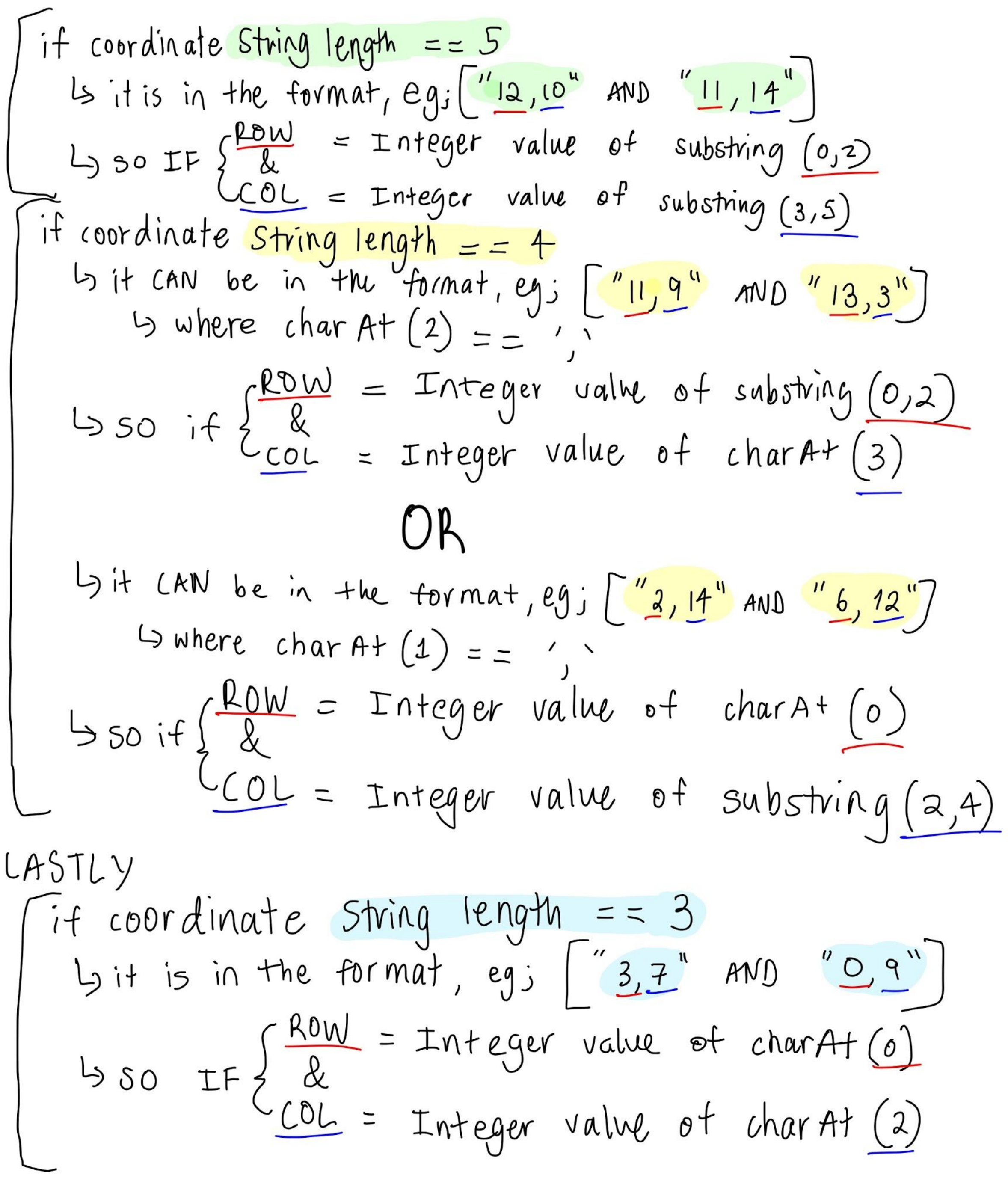
Pseudo-code for Displaying Existing Formations:

```
Previous button visibility set false
If formations (ArrayList of LinkedLists) is empty
    Go back to ‘creating’ a dance formation
    Communicate error with user via JOptionPane
Else (not empty…)
    addGrids(0) method (Pseudo-code found below)
    next Button visibility is false;
```

Pseudo-code for addGrids() method: → takes in index of formation which is wanted to be displayed

```
Remove all components from panel
Loop through the rows in GridPanel AND ‘jbuttongrid’ 2D array of JButtons
    Loop through the columns in GridPanel AND ‘jbuttongrid’ 2D array of JButtons
    Note that JButtons is used rather than JToggleButtons since this is simply for
    displaying and there should be no user interaction with the buttons
    Make new instance of JButton in the 2D array
    Set preferred, min, max size to dimension 40,40
    Assume visibility is false
    Loop through all coordinates in String format of index of formation
        (CHECKER METHOD FOR WHETHER ROW/COL CAN BE FOUND)
        SEE FIGURE 7 FOR CLEAR DIAGRAM EXPLANATION
            Set visibility true of conditions are true
    Add the instance of the button in 2D array to GridPanel
```

Figure 7



```
if coordinate String length == 5
  ↳ it is in the format, eg; ["12,10" AND "11,14"]
  ↳ so IF { ROW & COL  ROW = Integer value of substring (0,2)
                       COL = Integer value of substring (3,5)
if coordinate String length == 4
  ↳ it CAN be in the format, eg; ["11,9" AND "13,3"]
     ↳ where charAt (2) == ','
  ↳ so if { ROW = Integer value of substring (0,2)
            &
            COL = Integer value of charAt (3)
                    OR
  ↳ it CAN be in the format, eg; ["2,14" AND "6,12"]
     ↳ where charAt (1) == ','
  ↳ so if { ROW = Integer value of charAt (0)
            &
            COL = Integer value of substring (2,4)
LASTLY
if coordinate String length == 3
  ↳ it is in the format, eg; ["3,7" AND "0,9"]
  ↳ so IF { ROW = Integer value of charAt (0)
            &
            COL = Integer value of charAt (2)
```

I was dumbfounded when after implementing all this, I was still having issues with buttons being shown where they shouldn't have been. To find the root cause of the problem, I implemented…

1. A print line every time a button was set visible
2. A counter which would allow only a limited number of buttons to be set visible, depending on the size of the formation (# of selected buttons)
3. Removing adding any buttons at all, to see that after all the components had initially been removed, if any buttons remained. I found none remained.

None of these ended up fixing the issue.

After lots of time attempting to fix this issue, I found that when I was on the program, then clicked to another application such as Chrome or Netbeans (instead of the Jar), and came back, the problem **VANISHED.** I then figured out that if I switched tabs from the "Dance Formation" to any other tab and then back, it solved the problem as well.

So the last two lines of code implemented which made the algorithm work can be seen below:

```
Input.setSelectedIndex(0);
Input.setSelectedIndex(3);
```

Despite the REASON for the issue not being apparent I came up with 1 of 2 conclusions:

1. **Netbeans was having a glitch**, solved simply by a "*turn off, turn on*" type of method
2. The IDE was attempting to **save memory,** because I found that the buttons that were not supposed to show and did remain, were only BELOW all the buttons that were supposed to be there, on the GridLayout.
    a. Therefore, the assumption is that the program didn't do a full "***garbage collection***" of the buttons after all the elements in the Linked List that were supposed to be there were added.

Lastly, the spinner in the bottom right hand corner of the GUI can be changed to show only formations with a certain number of selected grids/students.

The difference in the algorithm can be seen below:

```
private ArrayList<Integer> found = new ArrayList<Integer>();
```

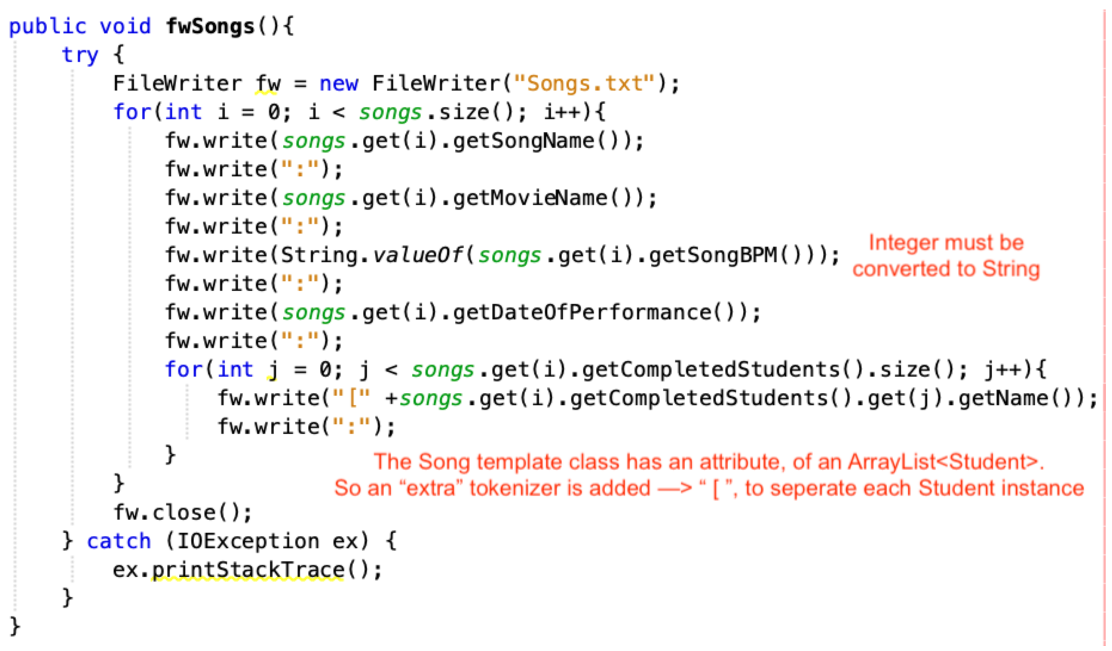
```
found.clear();
for(int i = 0; i < formations.size(); i++){
    if(formations.get(i).size() == (Integer) NumberOfStudentsSpinner.getValue()){
        System.out.println("Found formation at:" + i);
        found.add(i); // Stores the index of where the spinner is the same value as
        //the number of coordinates/selectedGrid/students in the LinkedList in formations
    }
}
```

- Then instead of the method addGrids() taking the parameter **'j'** (simply a counter, starting at 0), the addGrids() method takes **'found.get(j)'**, storing the indexes in **formations ArrayList** where the .size() == value of the spinner.

So, the next/prev button works by simply **adding / subtracting 1** from the **counter 'j'**

Word Count: 385

Code for Saving/Opening Files (see example below):

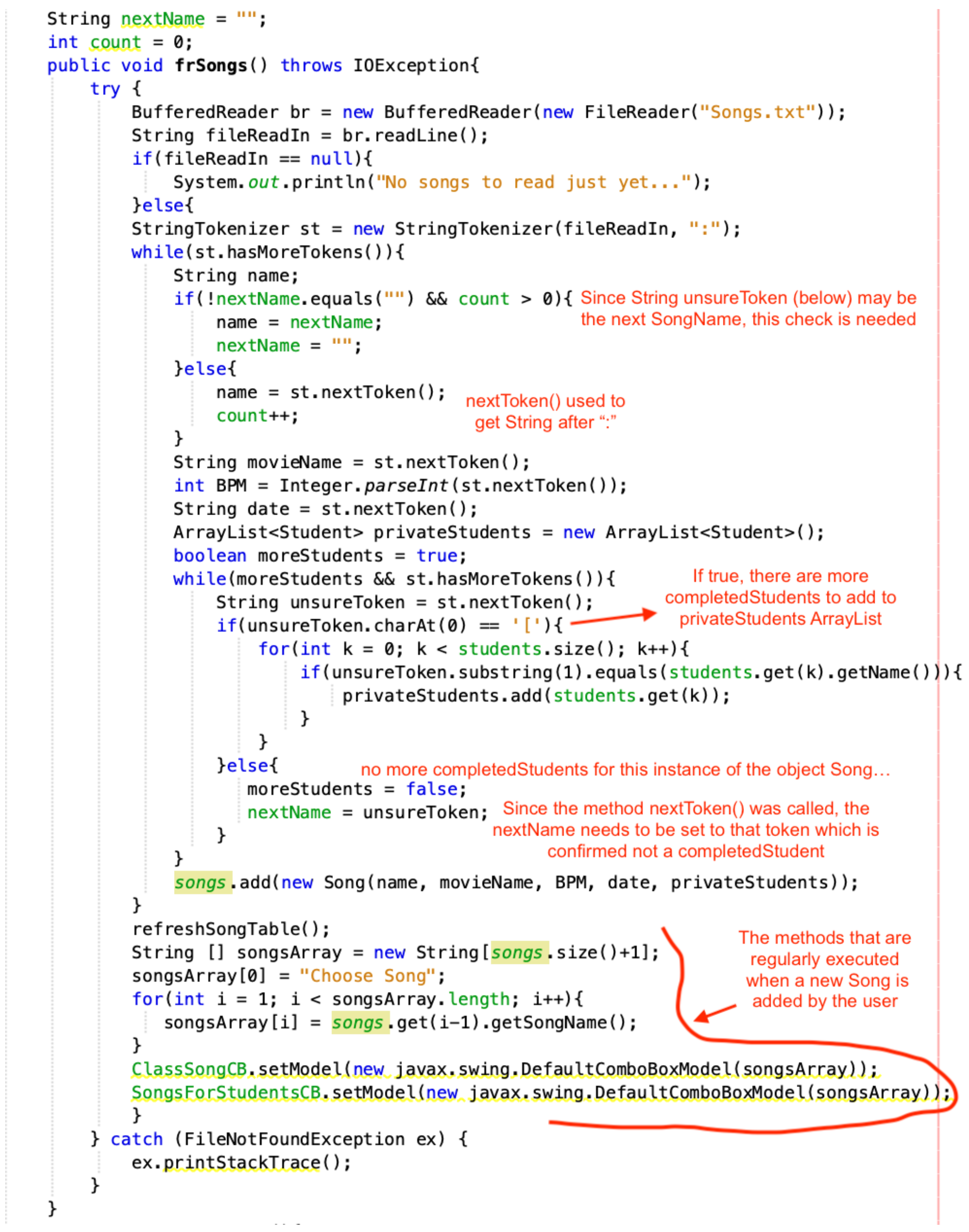


```
public void fwSongs(){
    try {
        FileWriter fw = new FileWriter("Songs.txt");
        for(int i = 0; i < songs.size(); i++){
            fw.write(songs.get(i).getSongName());
            fw.write(":");
            fw.write(songs.get(i).getMovieName());
            fw.write(":");
            fw.write(String.valueOf(songs.get(i).getSongBPM()));
            fw.write(":");
            fw.write(songs.get(i).getDateOfPerformance());
            fw.write(":");
            for(int j = 0; j < songs.get(i).getCompletedStudents().size(); j++){
                fw.write("[" +songs.get(i).getCompletedStudents().get(j).getName());
                fw.write(":");
            }
        }
        fw.close();
    } catch (IOException ex) {
        ex.printStackTrace();
    }
}
```

Integer must be converted to String

The Song template class has an attribute, of an ArrayList<Student>.
So an "extra" tokenizer is added —> " [", to seperate each Student instance

Reading the File "Song.txt" is quite complicated because of the intricate relationship between



```
String nextName = "";
int count = 0;
public void frSongs() throws IOException{
    try {
        BufferedReader br = new BufferedReader(new FileReader("Songs.txt"));
        String fileReadIn = br.readLine();
        if(fileReadIn == null){
            System.out.println("No songs to read just yet...");
        }else{
        StringTokenizer st = new StringTokenizer(fileReadIn, ":");
        while(st.hasMoreTokens()){
            String name;
            if(!nextName.equals("") && count > 0){
                name = nextName;
                nextName = "";
            }else{
                name = st.nextToken();
                count++;
            }
            String movieName = st.nextToken();
            int BPM = Integer.parseInt(st.nextToken());
            String date = st.nextToken();
            ArrayList<Student> privateStudents = new ArrayList<Student>();
            boolean moreStudents = true;
            while(moreStudents && st.hasMoreTokens()){
                String unsureToken = st.nextToken();
                if(unsureToken.charAt(0) == '['){
                    for(int k = 0; k < students.size(); k++){
                        if(unsureToken.substring(1).equals(students.get(k).getName())){
                            privateStudents.add(students.get(k));
                        }
                    }
                }else{
                   moreStudents = false;
                   nextName = unsureToken;
                }
            }
            songs.add(new Song(name, movieName, BPM, date, privateStudents));
        }
        refreshSongTable();
        String [] songsArray = new String[songs.size()+1];
        songsArray[0] = "Choose Song";
        for(int i = 1; i < songsArray.length; i++){
           songsArray[i] = songs.get(i-1).getSongName();
        }
        ClassSongCB.setModel(new javax.swing.DefaultComboBoxModel(songsArray));
        SongsForStudentsCB.setModel(new javax.swing.DefaultComboBoxModel(songsArray));
        }
    } catch (FileNotFoundException ex) {
        ex.printStackTrace();
    }
}
```

Since String unsureToken (below) may be the next SongName, this check is needed

nextToken() used to get String after ":"

If true, there are more completedStudents to add to privateStudents ArrayList

no more completedStudents for this instance of the object Song…

Since the method nextToken() was called, the nextName needs to be set to that token which is confirmed not a completedStudent

The methods that are regularly executed when a new Song is added by the user

User Interface/GUI Work

Key features of Java Swing components:

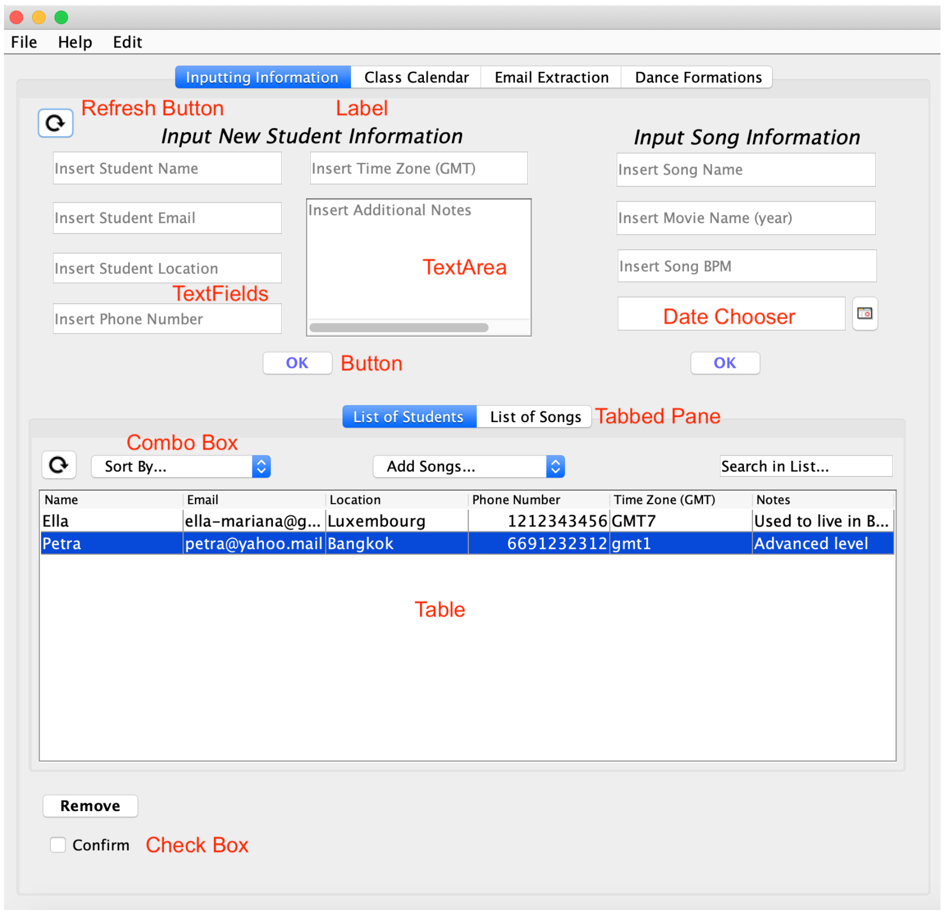
- TextFields
- TextAreas
- Buttons, Toggle Buttons
- Combo Box
- Check Box
- Menu Items
- Labels
- Spinner
- Tabbed Pane
- Popup Menu
- Progress Bar
- Date Chooser (fromJCalendar library)
- Table, List
- Scroll Pane

Key Layouts Utilized:

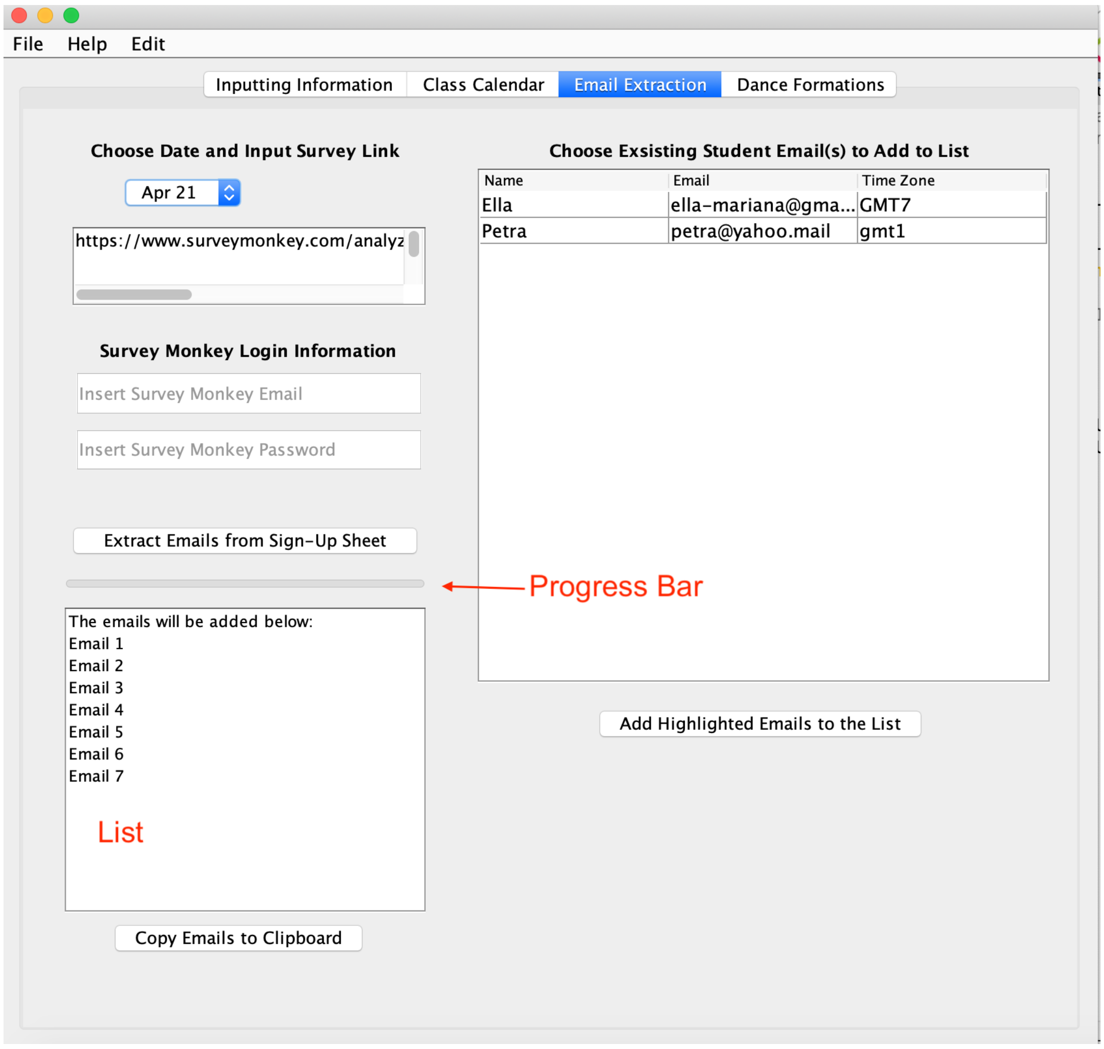
- Absolute Layout
  - Provides needed limited flexibility for elements not to shift around
- Grid Layout
  - Used for dance formations

Other:

- Custom refresh button (refreshing inputting information and JTables)
- Remove button to remove element from Student/Song JTable



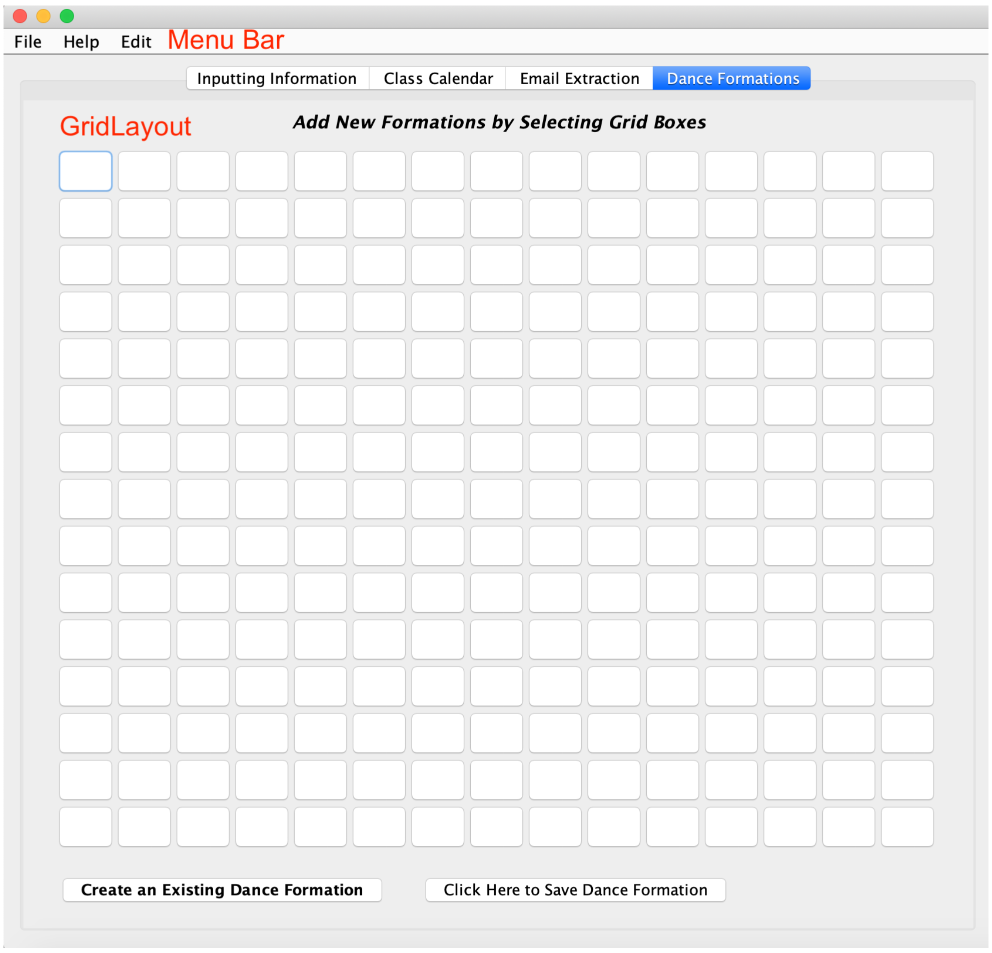
Name	Email	Location	Phone Number	Time Zone (GMT)	Notes
Ella	ella-mariana@g...	Luxembourg	1212343456	GMT7	Used to live in B...
Petra	petra@yahoo.mail	Bangkok	6691232312	gmt1	Advanced level


File
Help
Edit
Inputting Information
Class Calendar
Email Extraction
Dance Formations
Choose Date and Input Survey Link
Apr 21
Survey Monkey Login Information
Insert Survey Monkey Email
Insert Survey Monkey Password
Extract Emails from Sign-Up Sheet
Progress Bar
List
Copy Emails to Clipboard
Choose Exsisting Student Email(s) to Add to List
Add Highlighted Emails to the List

```
https://www.surveymonkey.com/analyz
```

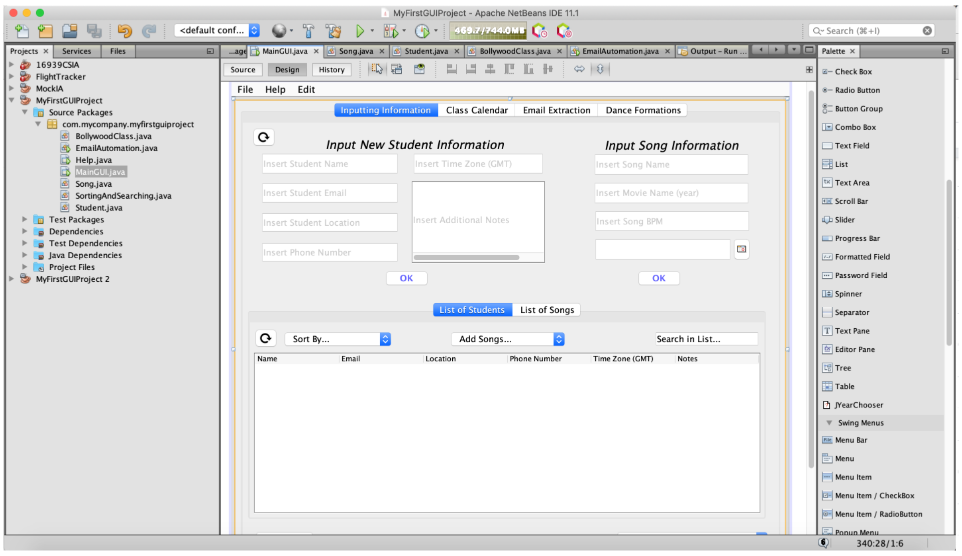
Name	Email	Time Zone
Ella	ella-mariana@gma...	GMT7
Petra	petra@yahoo.mail	gmt1

```
The emails will be added below:
Email 1
Email 2
Email 3
Email 4
Email 5
Email 6
Email 7
```


File
Help
Edit
Menu Bar
Inputting Information
Class Calendar
Email Extraction
Dance Formations
GridLayout
Add New Formations by Selecting Grid Boxes
Create an Existing Dance Formation
Click Here to Save Dance Formation

Software Tools Used

This program is appropriately created on the popular Integrated Development Environment (IDE) Netbeans, which contains a plethora of convenient features and GUI tools, boosting my efficiency and productivity. Moreover, it requires a suitable level of Java understanding for a high schooler, and packs a compiler, spell checker and debugger all into one.



Total Word Count: 1,093