Assignment: Computer Science II (CSCI 120) Project Submission

Instructor: Dr. Sajid Hussain

Due Date: April 28, 2022

Project Title: Bulldog Mailroom Notification System

**Group Members**: Oreoluwa Owoseeni

Henry Fowobaje

Shalom Akinbulumo

**Project Background**

After some inquiries from the mailroom at Fisk University, it has been realized that an average of 250 deliveries (both packages and mails) arrive at the Fisk University Mailroom located at 1000 17th Avenue North, Nashville, TN 37208. Of these 250 deliveries, about 150 are received by vigilant students on the day of delivery. This means that there is an average of 100 untended deliveries at the mailroom. If this is extended to the end of the working week, we can estimate that there will be about 500 packages and mails collecting dust in the mailroom.

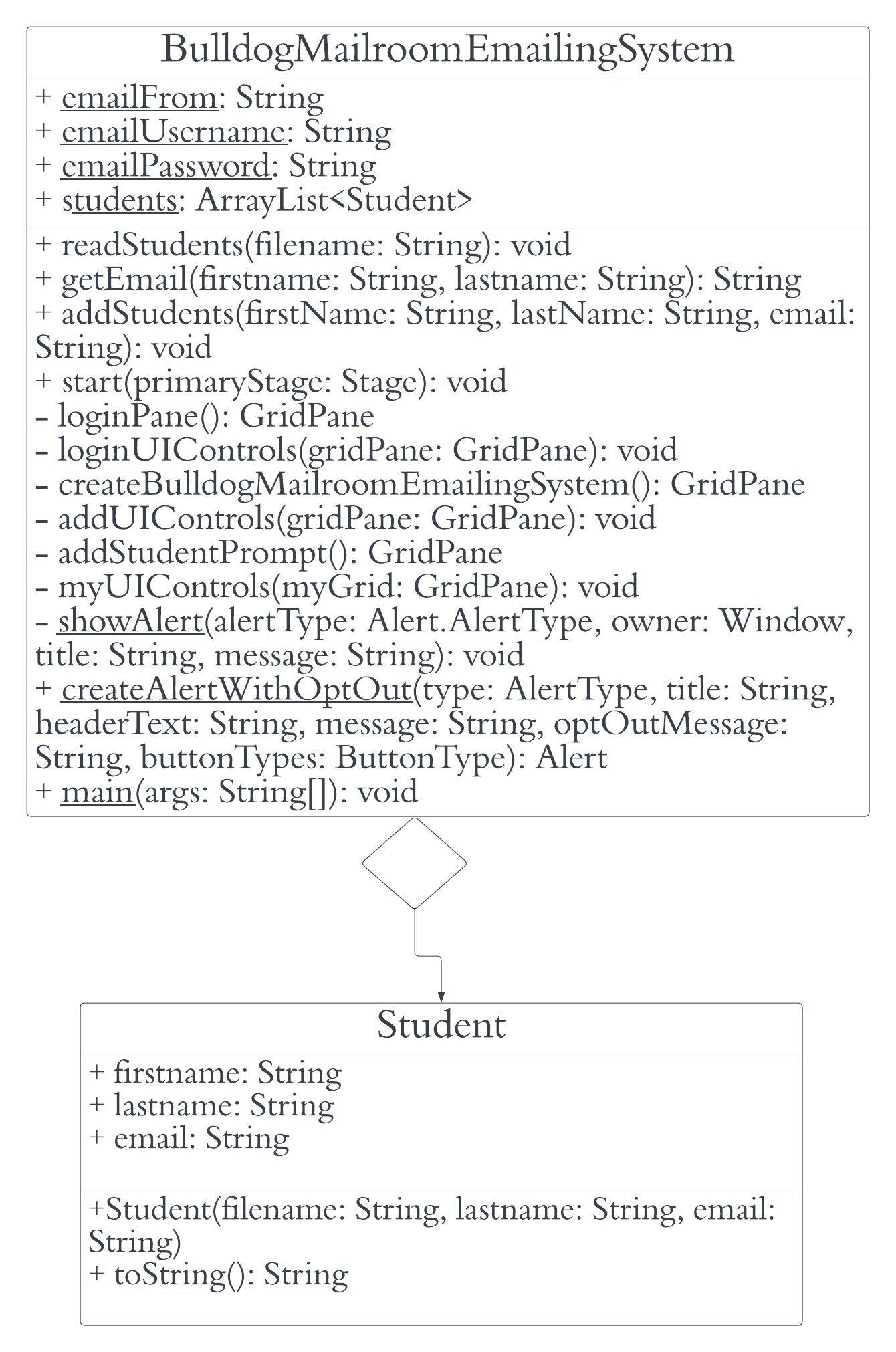
A very common but unfortunate attribute of young adults and college students is not remembering small and important details. However, this comes as no surprise, as there is a lot on our plate. Research has shown that most geniuses are forgetful people. In spite of this, forgetting your mail or package at your college mailroom is an inexcusable action. This problem was what spurred up the idea for our Computer Science II (CSCI 120) project, which we have titled: **Bulldog Mailroom Notification System**, specifically tailored for Fisk University.

**Project Summary:**

This Project documentation specifies the functional requirements for an emailing notification program. This application was written using Java programming language, and more specifically JavaFX and the Java Mail Application Programming Interface package. Our notification system has been set up to notify Fisk University students when their parcels and mail arrive in the mailroom. This program is intended to function as an "*Administrative Desktop App that allows the Mailroom Manager to alert Fisk University Students of Package Arrival*." The Bulldog Mailroom Notification System has an authentication login page which will be used by the Mailroom Manager. It is also equipped with a tentative database of Fisk University students information, such as full names and email addresses, which will be used as the means for notifying. There is also a working active functionality to add more students, who are not presently saved in the system, to the database. When the Mailroom Manager enters the student's name and package details, our system immediately notifies the said student about the arrival of their package and its details.

**UML Diagram:**

Figure A below is a Unified Modeling Language (UML) diagram that visually represents architecture, design and implementation of this software. The respective Java classes and methods used are represented in the diagram below.



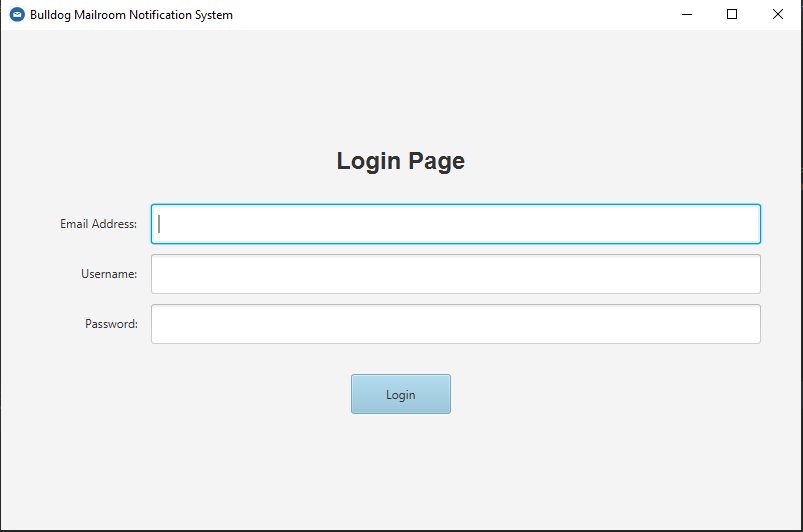
***Functionalities***

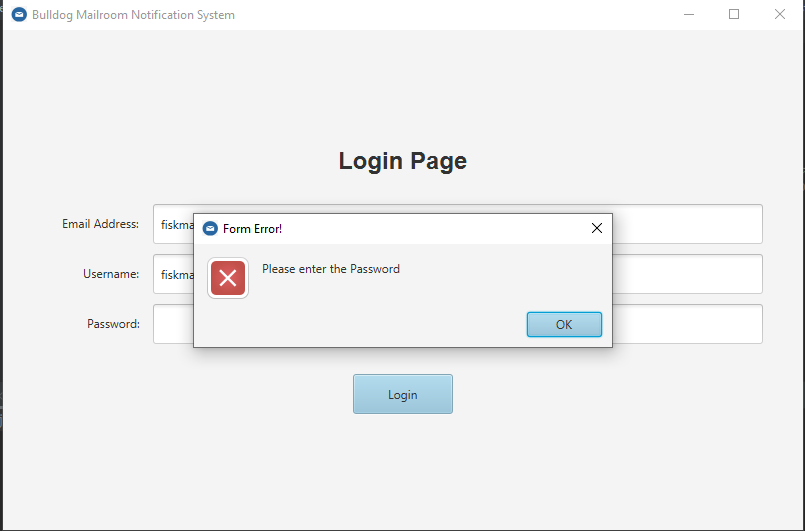
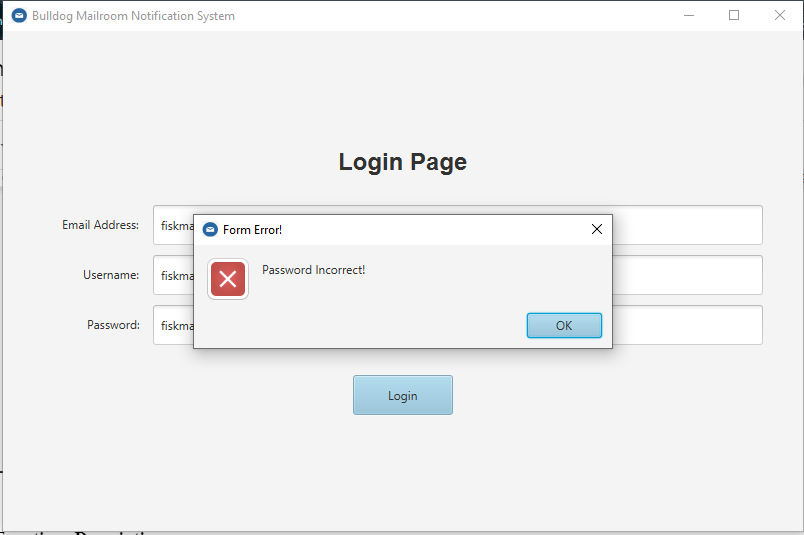
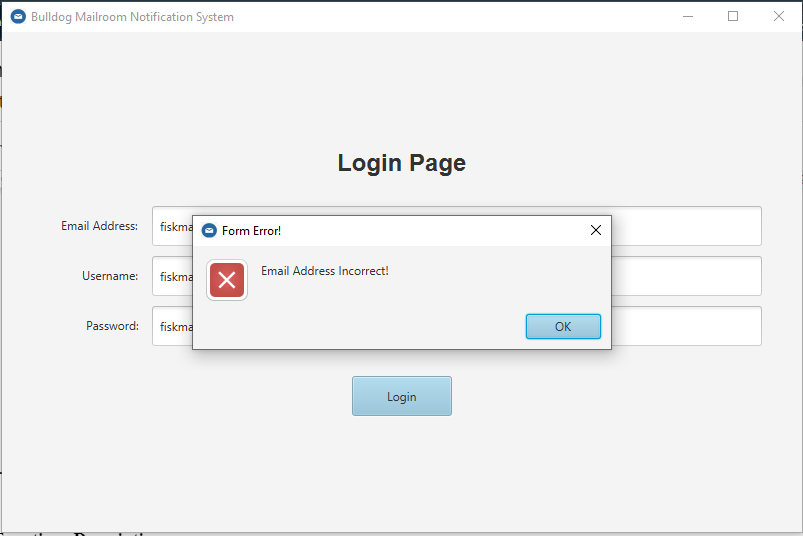
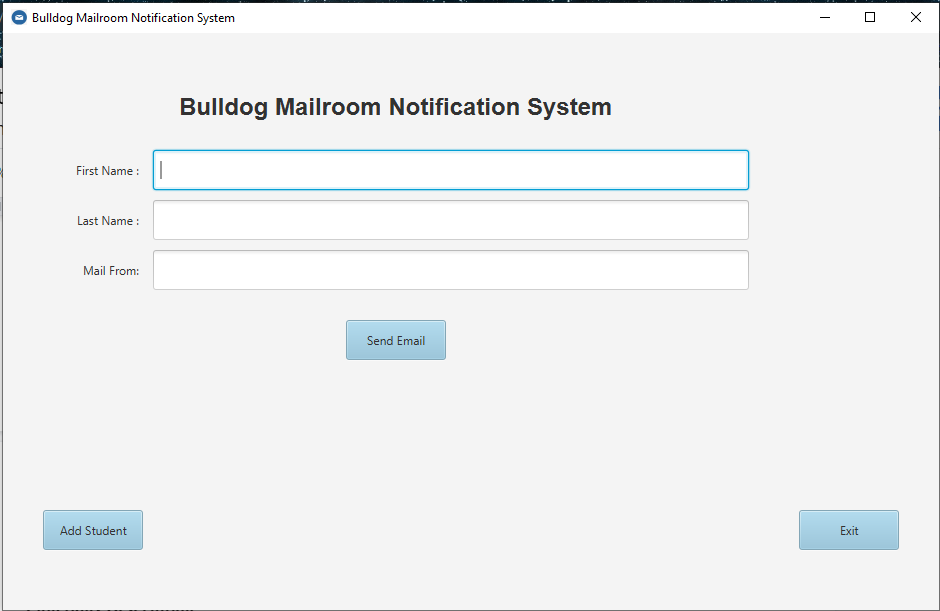
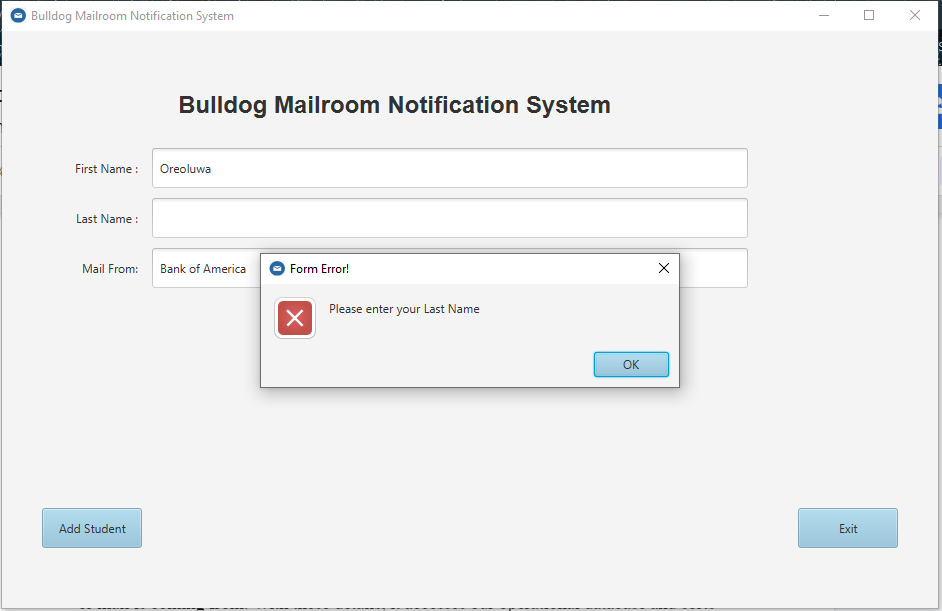
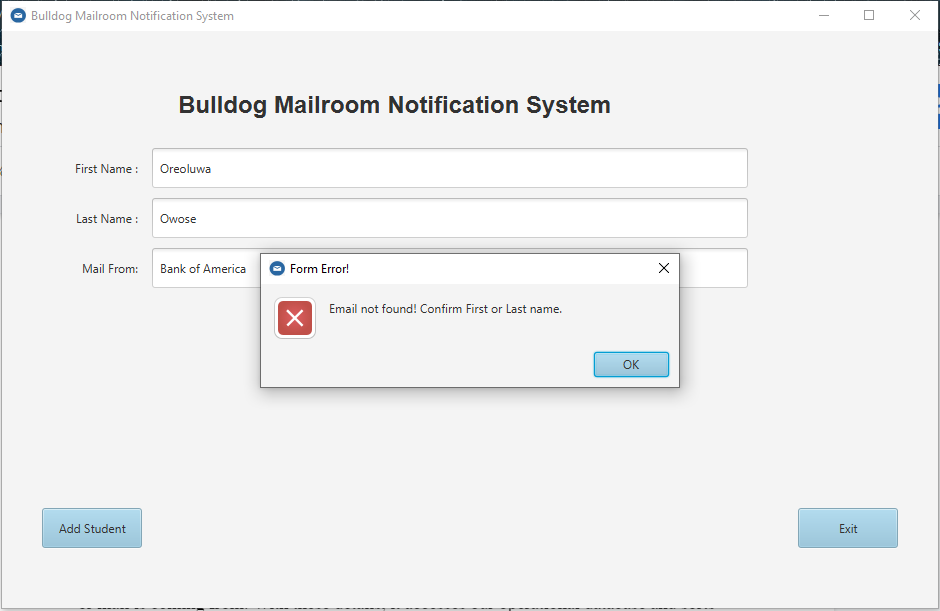
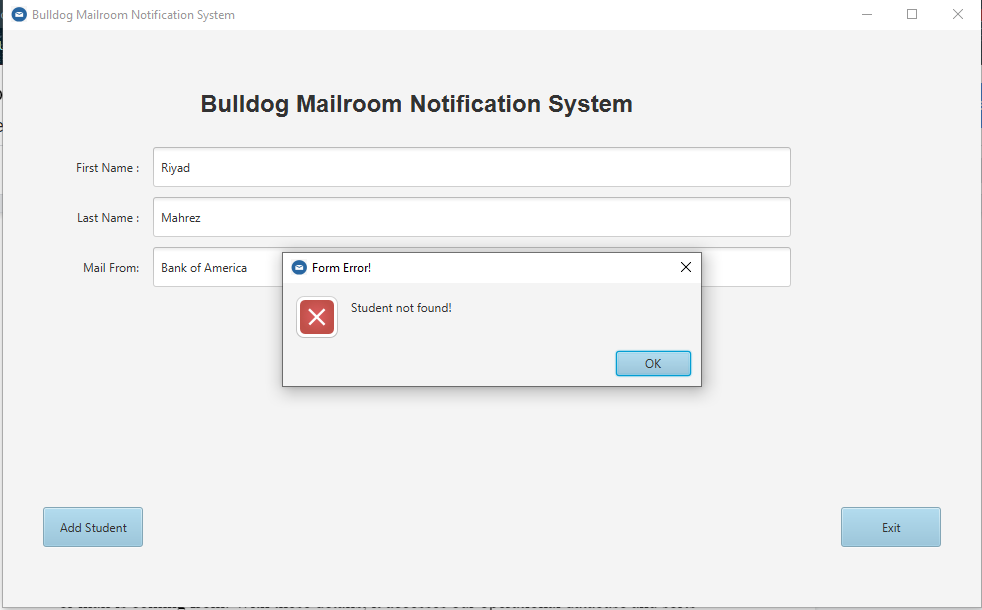
Below we will explain the classes and methods that were used in making this application:

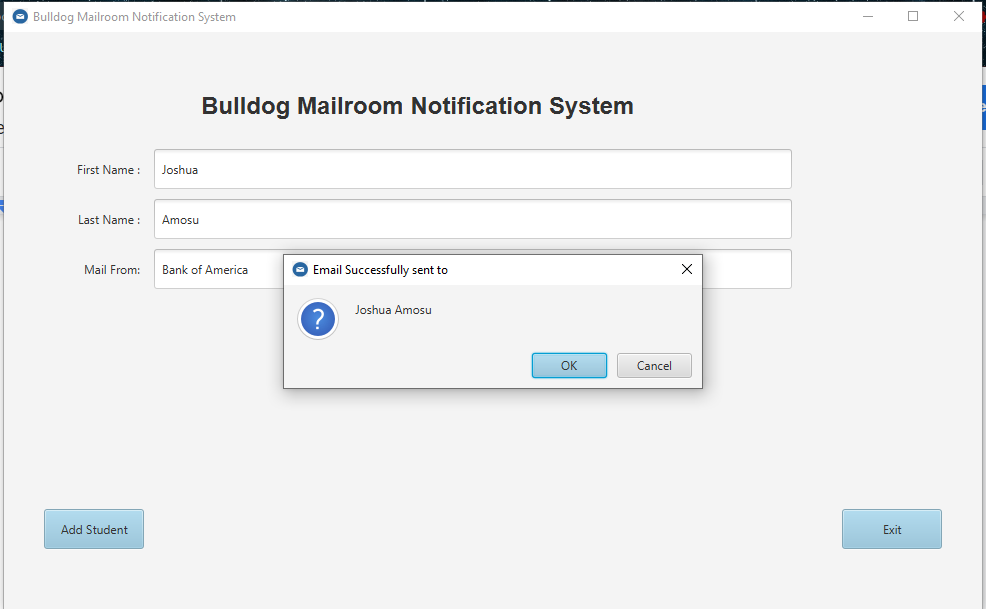
1. BulldogMailroomEmailingSystem: This file contains all the major data members and functions that help in launching and controlling the program as a whole. In this class we will find all the helper methods for this program. They are listed below:
2. readStudents(): The readStudents method is used to extract data from our central database and store student information in a Java ArrayList. In this way, any student information can be retrieved easily. This method doesn’t return anything.
3. getEmail(): The getEmail method, when called, takes in two parameters, which are the student’s first and last name. It then uses these values to sift through the data members of our ArrayList. Once both values can be found at the same position in the array list, the third member of that particular array element is the student’s email. This method returns the student’s email.
4. addStudents(): The addStudents method is used to add students to the central database. It was implemented to combat the *Student not found* problem. This method reads the file and adds the student as the newest member of our central database. This method doesn’t return anything.
5. start(): Our start method is where our window is initialized and called to display. This method doesn’t return anything.
6. loginPane(): The loginPane method determines the arrangement of the labels, buttons and textfields of the login page. This method returns a GridPane.
7. loginUIControls(): The loginUIControls method contains the eventhandler methods for all functionary buttons on the loginPane. It assigns functionality and alerts to the buttons in the login page. This method doesn’t return anything.
8. createBulldogMailroomEmailingSystem(): This method determines the arrangement of the labels, buttons, and textfields of the main menu. This method returns a GridPane.
9. addUIControls(): The addUIControls method contains the eventhandler methods for all functionary buttons on the main menu page. This method doesn’t return anything
10. addStudentPrompt(): This method determines the arrangement of the labels, buttons and textfields of the add student menu. This method returns a GridPane.
11. myUIControls(): The myUIControls method contains the eventhandler methods for all functionary labels and buttons on the add student menu. This method doesn't return anything.
12. showAlert(): This method contains the code used to engineer all the alerts that don’t involve an opt out option for memory use. This method doesn’t return anything.
13. createAlertWithOptOut(): This method contains the codes used to engineer all the alerts that include an opt out option for memory use. This method returns an alert.
14. Student: This file contains the data members and functions that were eventually used to construct the Student type and multiple Student instances in this app. It had two methods:
15. Student(): This served as the constructor in the class. It was used to store the values of an instance variable.
16. toString(): The toString method in this class was used to return subsequent Student objects. This method returned a string containing the data members of the Student at an instance.

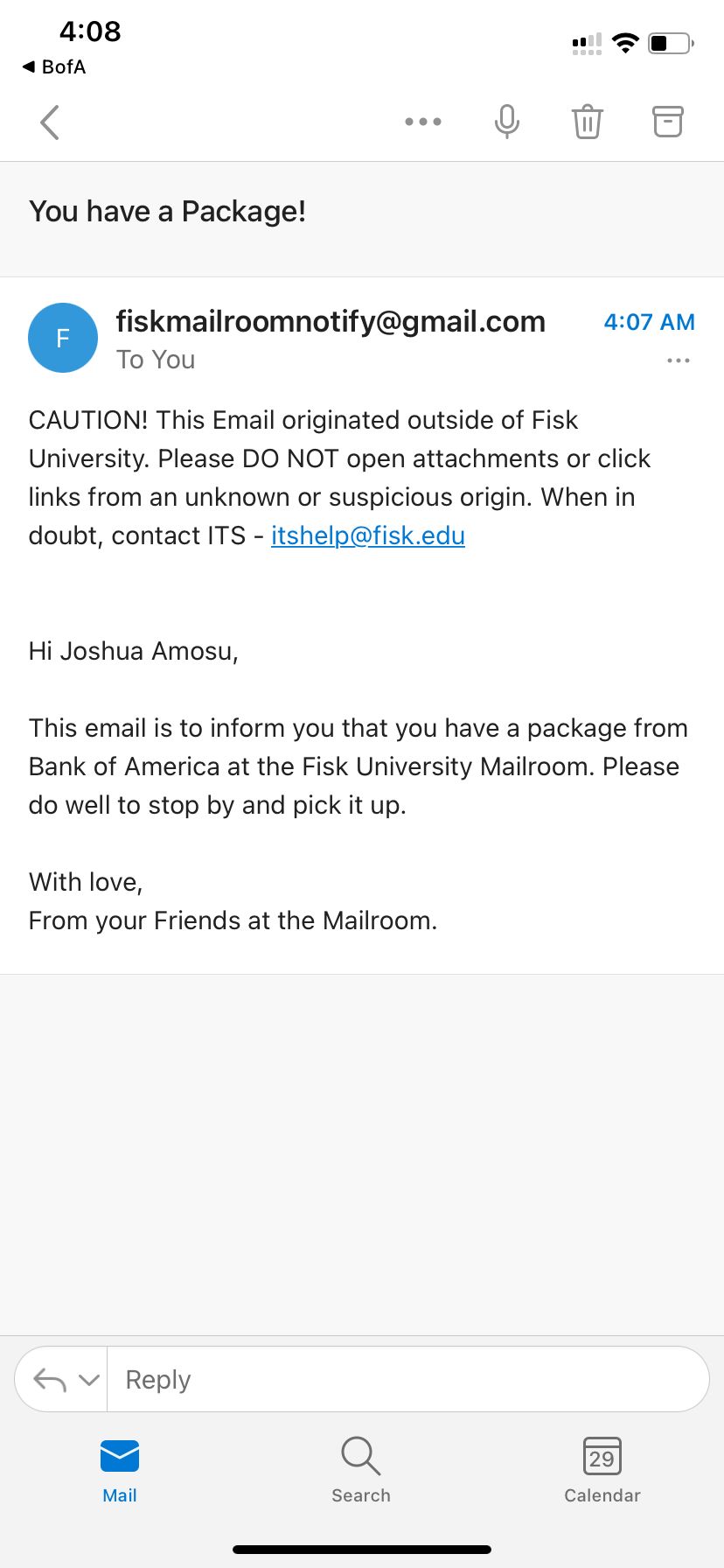
***Test Run***

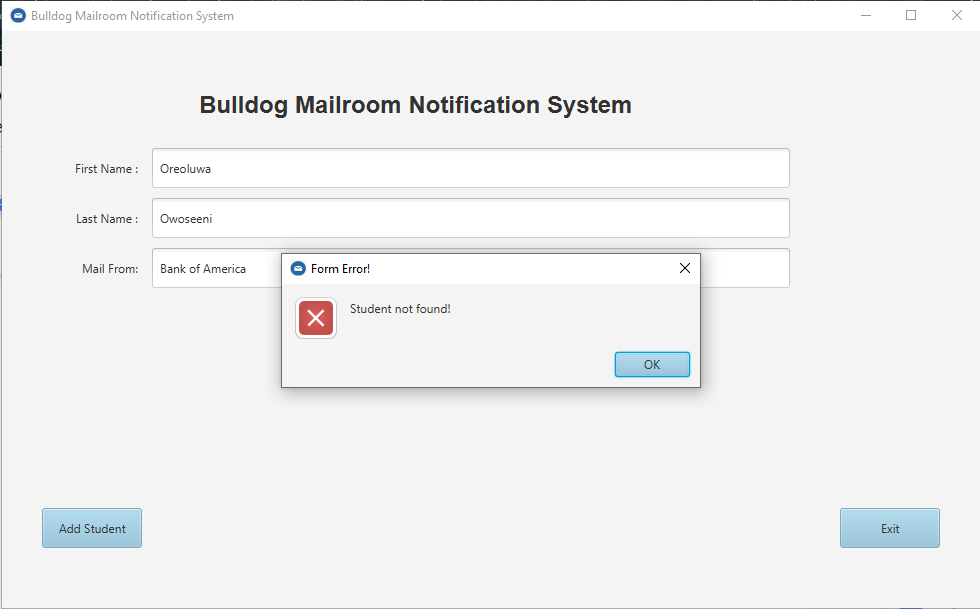
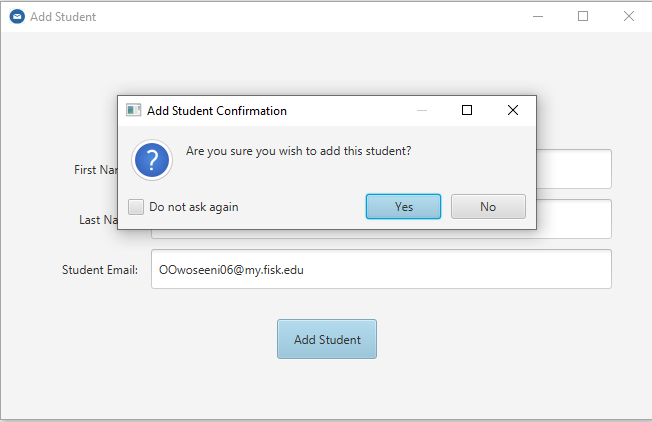
1. The first step we took in building this software was to create a Java file that helps build our student information database as a .csv file. This particular coding sequence took about 1007 lines.
2. The next thing, as we see, is observed when we run the program. The login page for the Mailroom Manager pops up. This login page requests for the login details to the central mailroom email account that will be used for sending notifications.  
   N.B: We have already created a tentative central email for the notification system, and that is what we will be using throughout this test run. The email address is [*fiskmailroomnotify@gmail.com*](mailto:fiskmailroomnotify@gmail.com).

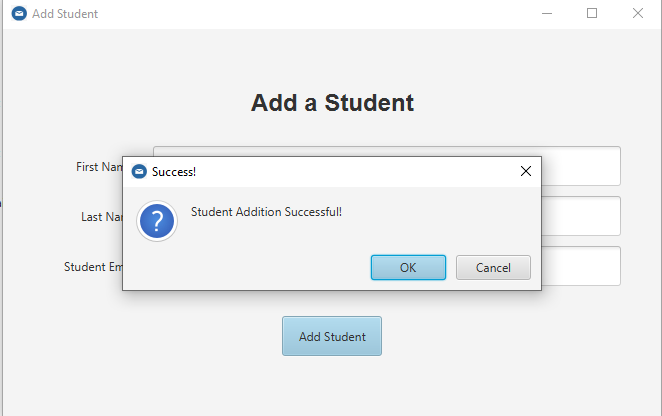


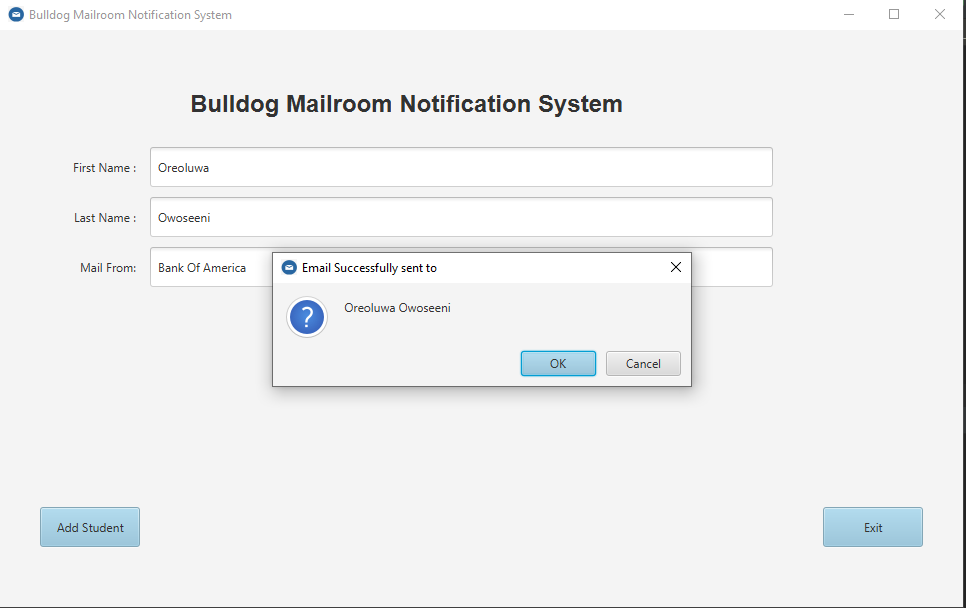
1. An error alerting system has also been set up for separate instances of error e.g. using a wrong password, wrong email address, or leaving a field blank. The figures below show some common errors that could occur and their corresponding alerts.  
   *Error 1*: Password field blank  
     
     
   *Error 2*: Password Incorrect!  
     
     
   *Error* 3: Wrong Email Address  
   
2. When login details are correct, access is granted to the app main menu. It looks like this:  
     
   The main menu takes in the first and last name of the student, and also where the package or mail is coming from. With these details, it accesses our operational database and sorts through to retrieve the student’s email address. Necessary alerts are in place for errors such as:  
   *Error 1*: Empty Field  
     
   *Error* *2*: Incomplete details  
     
   *Error 3*: Student not found in database  
   

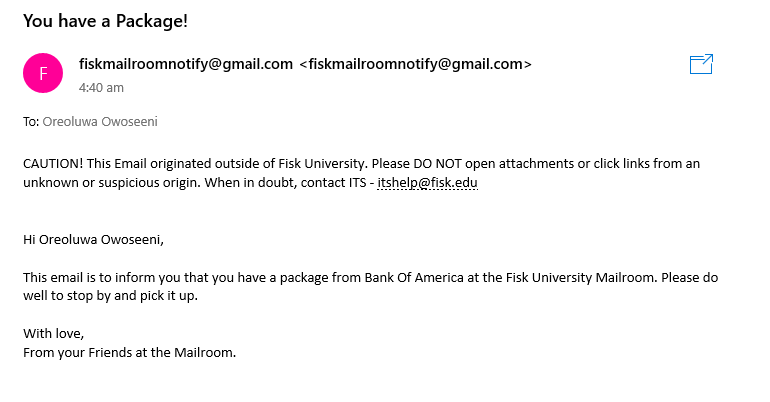
When the details are correct and are imputed correctly, a confirmation alert is used to indicate a successful notification.  
*Confirmation*  


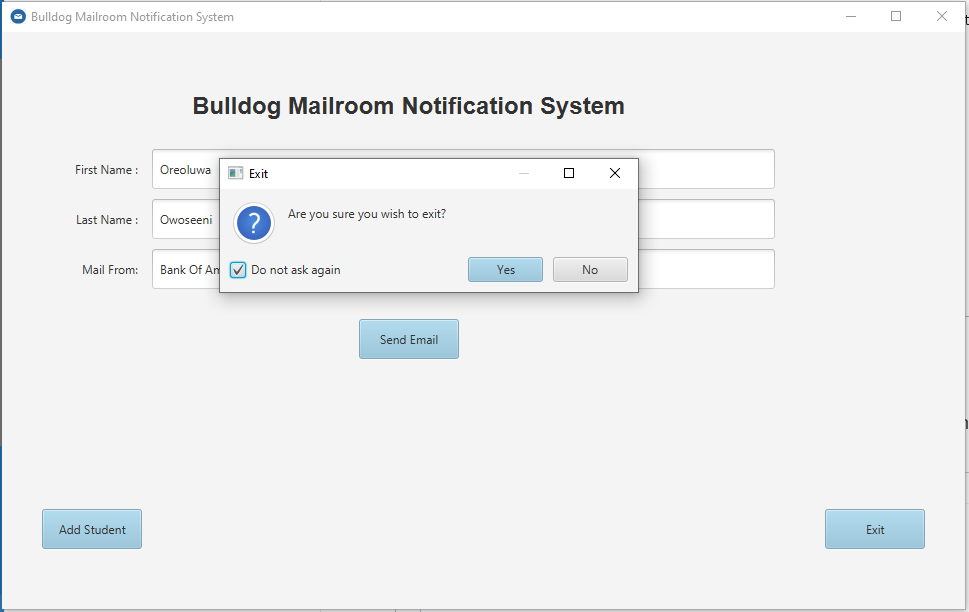
1. Below is a screenshot of the notification received by a fellow student in this test run.  
   *Notification screenshot*:
2. In response to instances where the student isn’t found in the database, we have added a ‘*Add Student*’ button, which can be used to add the said student to the central database. An example is shown below:

*Error*: Student Not Found!  
*Confirmation Alert for Student Addition  
*

*Student Addition Successful!*  


*Email Can Now Send to Student*

*Email Result*  


1. The exit button is designed to store user preferences:  
   

***Test Run Complete!***

Total number of lines of code used was about **3,800 lines**.

**Environment and Interface Requirements:**

We would absolutely discuss the environment behind our program before concluding our documentation. The program should function as closely as feasible like an administrative management desktop tool. The program is a simple and user-friendly desktop application. The prompted program displays a screen that requires the user to authenticate themselves using the manager's username and password. The manager is then requested to notify the student of the delivery of their package based on their first and last name via a prompted screen. Finally, a popup would appear to confirm whether or not the email notification was successful.

**Prospective Future Additions**

We would like to document some thoughtful and intuitive additions that we will subsequently make to this application:

1. The use of the Java calendar function to record the date a package arrived
2. The use of an integer variable that counts how many days a package has stayed at the mailroom.
3. The incorporation of the integer variable to the body of the email to let the student know how long it has been since the package has been delivered.
4. The use of analysis to determine the number of packages sent by a particular company.
5. The use of analysis to determine the number of packages a student has received within a specific time period.
6. The use of a text messaging function to couple with the email notifying function.