PROGRAMMED BY

Raymund Klien Mañago
Lance Rimando

SUBMITTED TO

Ashlyn Kim Balangcod

Student Database

Machine Problem

COURSE

CMSC 12 - FG

Content

Introduction

MAXWELL UNIVERSITY
STUDENT DATABASE

Maxwell University introduces its **interactive offline database system**, in line with its goal to provide accessible student services. The system allows easy monitoring of student entries with the help of Java Swing GUI.

The system has five primary features: **viewing**, **adding**, **deleting**, **editing**, and **searching** entries. For the current version of the system, a maximum of ten (10) student entries may be added. Detailed descriptions of the features are given in the next pages, including the data structures used and algorithms implemented.

GUI

The main variable used in the program's GUI is the ***mainFrame***. It is the Swing frame that contains the entire Swing components and containers that compose the database interface.

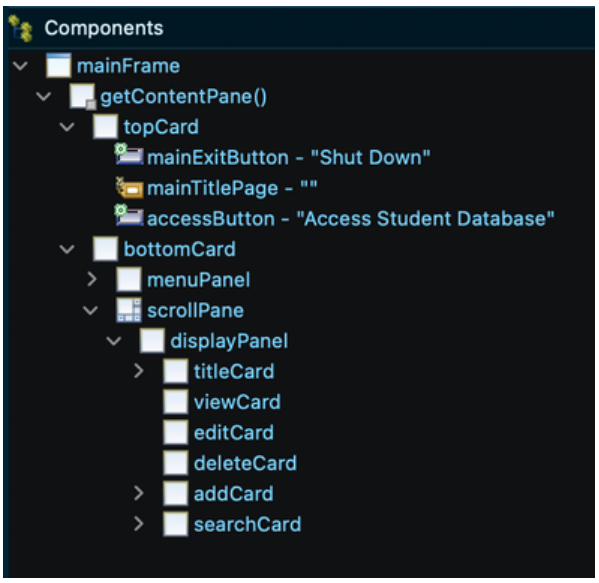


Figure 1. Swing components and containers hierarchy.

Aside from the main frame, Swing panels such as ***topCard*** and ***bottomCard*** were used together with a CardLayout to accommodate the two main panels . The top card contains the receiving homepage, while the bottom card contains all the features of the database.

```
CardLayout cl = new CardLayout(0, 0);

JFrame mainFrame = new JFrame();
mainFrame.setBounds(100, 100, 911, 607);
mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
mainFrame.setLocationRelativeTo(null);
mainFrame.getContentPane().setLayout(cl);

JPanel topCard = new JPanel(){
    protected void paintComponent(Graphics g) {
        Paint p = new GradientPaint(0.0f, 0.0f, new Color(255, 255, 255, 255),
                getWidth(), getHeight(), new Color(189, 175, 255, 255), true);
        Graphics2D g2d = (Graphics2D)g;
        g2d.setPaint(p);
        g2d.fillRect(0, 0, getWidth(), getHeight());
    }
};
mainFrame.getContentPane().add(topCard, "1");
topCard.setLayout(null);

JPanel bottomCard = new JPanel();
mainFrame.getContentPane().add(bottomCard, "2");
bottomCard.setLayout(null);
```

Figure 2. *mainFrame, topCard, bottomCard* initialization.

GUI

For every feature of the database, there exists a panel to contain them such as ***viewCard***, ***addCard***, ***editCard***, ***deleteCard***, and ***searchCard***. These panels are integrated in a CardLayout system so that it can display their features individually on the same parent container.

Another important variable is the ***entryPanel***. It contains the name, SAIS ID, student number, and address of each student entries in the database. This Swing panel is implemented inside the ***createPanel()*** function wherein it runs on a for loop. The panel's y-coordinate increments by 148 pixels each entries read by the ***showData()*** function while its x-coordinate remains the constant for all entries,

```
// ---------- A method to create GUI Entry Panels ---------- //

private void createPanel(JPanel entryCard, int state, JFrame mainFrame) throws FileNotFoundException{
    Dimension dim = new Dimension(672, 142);
    StudentData [] students = new StudentDB().showData();
    entryCard.removeAll();

    int studentCount = 1;
    for (StudentData student : students) {

        // Using a for loop to create panels for each student entries

        JPanel entryPanel = new JPanel();
        entryPanel.setBackground(new Color(230, 230, 250));
        entryPanel.setBounds(6, 6 + 148 * (studentCount - 1), 672, 142);
        entryPanel.setPreferredSize(dim);
        entryPanel.setLayout(null);
        entryCard.add(entryPanel);

        studentCount++;
```

Figure 3. Initialization and implementation of *entryPanel*.

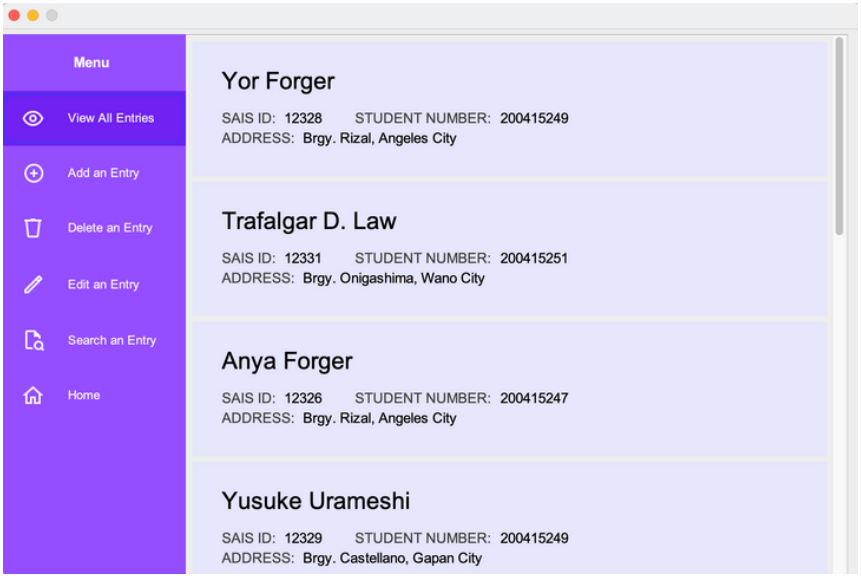


Figure 4. "View All Entries" tab calls createPanel() function and displays *entryPanel*.

Data Structures

Backend

Each information of an entry is stored in the String variables ***name*** and ***address*** and in the int variables ***SAISID*** and ***num*** (student number) that are accessible in StudentData class. An object of the StudentData class contains these variables as properties, along with an additional int variable ***fileCount*** which keeps track of the number of entries currently stored.

All objects of StudentData are collated in a StudentData array called ***students***, through the function ***showData()***. The function does exactly as its name suggests; it reads all text files currently stored in the directory and instantiates a StudentData for each one. Additionally, the function shortens the array to the number of entries available so null entries will be discarded.

```
// This function returns array of students that has been read from a list of text files.
public StudentData [] showData() {
    try {
        File fileHolder = new File(".");
        File [] files = fileHolder.listFiles(filter());
        StudentData [] studentList = new StudentData[10];
        int fileCount = 0;
        for (File file : files) {
            Scanner sc = new Scanner(new File(file.getName()));
            String name = sc.nextLine();
            String id = sc.nextLine();
            String num = sc.nextLine();
            String address = sc.nextLine();
            studentList[fileCount] = new StudentData(name, Integer.parseInt(id), Integer.parseInt(num), address);
            sc.close(); fileCount++;
        }
        StudentData [] students = new StudentData[fileCount];
        for (int fileCount2 = 0; fileCount2 < fileCount; fileCount2++) {
            students[fileCount2] = studentList[fileCount2];
        }
        return students;
    }
    catch (FileNotFoundException e) {
        return new StudentData[0];
    }

}
```

Figure 5. The function *showData()* returns the *students* array.

Algorithms

GUI

Looking at Figure 1 gives a general gist of the GUI algorithm. The algorithm starts by calling the function ***mainGUI()***. The database can be interacted by the user using buttons for navigation and text fields for input. The program starts by showing the top of the hierarchy - the topCard which contains the receiving homepage of the database. Clicking "Access Student Database" button will show the bottomCard panel which contains all the working services of the database.

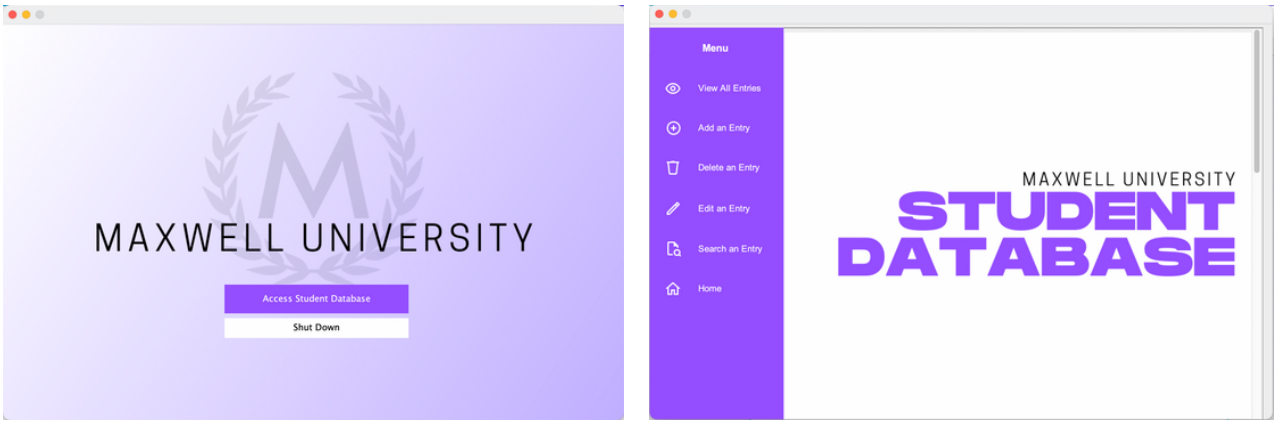


Figure 6. *topCard* (left) and *bottomCard* (right) display.

```
JButton mainExitButton = new JButton("Shut Down");
mainExitButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        System.exit(0);
    }
});


JButton accessButton = new JButton("Access Student Database");
accessButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        scrollPane.getViewport().setViewPosition(new Point(0,0));
        cl.show(mainFrame.getContentPane(), "2");
        cl2.show(displayPanel, "title");
    }
});
```

Figure 7. Code executed when clicking the two buttons in the topCard.

Algorithms

GUI

Swing text fields are heavily used in the "Add an Entry" menu section. It uses four text fields to accommodate for the incoming information to be given by the user. These text fields are equipped with error handling codes to be discussed later on. When the user input is proper and passes the error handling codes, it will be checked again if the entry is unique and if the total entries still has space for it (maximum of 10). As we can see from Figure 7, it also gives a preview whenever an entry is successfully added. If in the instance that an entry fails to be added, error messages would be displayed, shown in the error handling section.

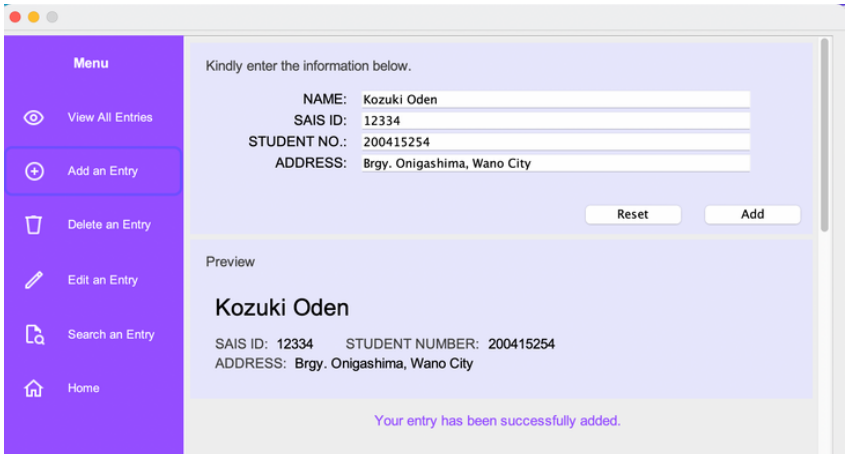


Figure 8. "Add an Entry" section of the database.

```
JButton addEntryBtn = new JButton("Add");
addEntryBtn.setBounds(549, 172, 117, 29);
addEntryBtn.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {

        // Running a series of boolean to handle user input errors

        boolean isIncomplete = nameAddField.getText().isEmpty() || saisAddField.getText().
        boolean isDigit = saisAddField.getText().matches("[0-9]+") && studNoAddField.getTe
        boolean isWhiteSpace = nameAddField.getText().trim().length() == 0 || saisAddField
        boolean isProperLength = saisAddField.getText().length() < 10 && studNoAddField.ge

        if (isIncomplete) {
            addFailLabel.setText("Incomplete entry!");
            addSuccessLabel.setText("");
        }
        else if (isWhiteSpace) {
            addFailLabel.setText("Entry can't be spaces!");
            addSuccessLabel.setText("");
        }
        else if (!isProperLength) {
            addFailLabel.setText("SAIS ID and Student Number must not exceed 9 digits.");
            addSuccessLabel.setText("");
        }
        else if (!isDigit) {
            addFailLabel.setText("SAIS ID and Student Number must be numbers!");
            addSuccessLabel.setText("");
        }
```

Figure 9. Code snippet of the error-handled add button.

Algorithms

GUI

Edit and delete menu options are special because it opens another window with a different set of options. Seen from Figure 7, when the "Edit" button is clicked, it pops a new window to ask which information of an entry the user wants to change. It also comes with two buttons for the user to confirm the change or to cancel it. Likewise in the add an entry section, it is also equipped with error handling codes.

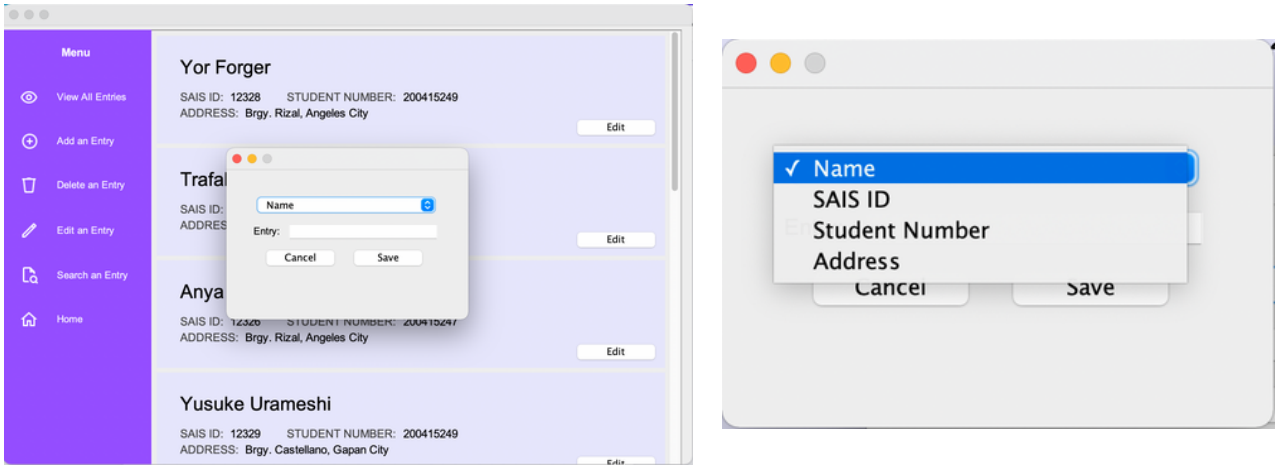


Figure 10. Edit button pops up a new window with a dropdown menu using Swing ComboBox.

Moreover, it is also shown in Figure 9 that clicking "Delete" button will also pop a new window. This is implemented so that user would not accidentally delete an entry - who wants that anyway?

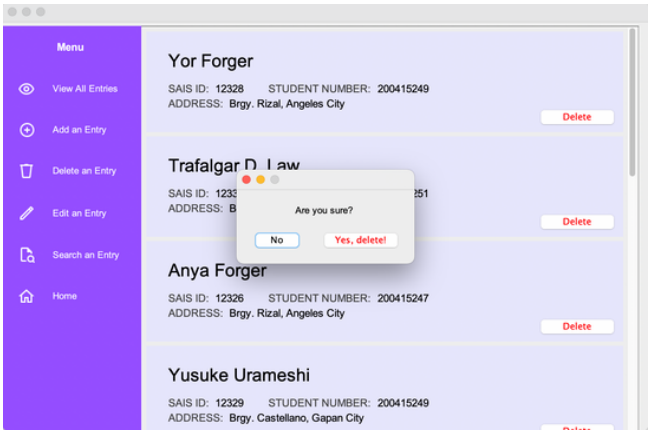


Figure 11. Delete button pops up a new window, asking a confirmation.

GUI

```
// ---------------- Delete Window GUI ---------------- //

private class openDeleteWindow{
    openDeleteWindow(String name, int SAISID, JFrame mainFrame){
        JFrame deleteFrame = new JFrame();
        deleteFrame.setBounds(100, 100, 251, 133);
        deleteFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        deleteFrame.setAlwaysOnTop(true);
        deleteFrame.getContentPane().setLayout(null);
```

Figure 12. *openDeleteWindow* class that contains the window properties after clicking Delete button.

```
// ---------------- Edit Window GUI ---------------- //

private class openEditWindow{
    openEditWindow(String name, int SAISID, JFrame mainFrame) {
        JFrame editFrame = new JFrame();
        editFrame.setBounds(100, 100, 320, 226);
        editFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        editFrame.setLocationRelativeTo(null);
        editFrame.setResizable(false);
        editFrame.setAlwaysOnTop(true);
        editFrame.getContentPane().setLayout(null);
```

Figure 13. *openEditWindow* class that contains the window properties after clicking Edit button.

```
// Using createPanel to create buttons for deletePanel and editPanel

if (state == 2) {
    deleteEntryBtn = new JButton("Delete");
    deleteEntryBtn.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            mainFrame.setEnabled(false);
            new openDeleteWindow(student.name, student.SAISID, mainFrame);
        }
    });
    deleteEntryBtn.setBounds(549, 107, 117, 29);
    deleteEntryBtn.setForeground(Color.RED);
    entryPanel.add(deleteEntryBtn);
}

else if (state == 3) {
    editEntryBtn = new JButton("Edit");
    editEntryBtn.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            mainFrame.setEnabled(false);
            new openEditWindow(student.name, student.SAISID, mainFrame);
        }
    });
    editEntryBtn.setBounds(549, 107, 117, 29);
    editEntryBtn.setForeground(Color.BLACK);
    entryPanel.add(editEntryBtn);
}
```

Figure 14. Code snippet inside *createPanel()* that creates an instance of the classes in Figures 12, 13.

Algorithms

GUI

```
// No paremeters
// Void
FUNCTION mainGUI()
    DISPLAY topCard
    IF accessButton is clicked
        THEN DISPLAY bottomCard
        IF viewButton is clicked THEN
            DISPLAY viewCard and CALL createPanel()
            CALL showData()
        IF addButton is clicked THEN
            DISPLAY addCard and CALL createPanel()
            GET name using nameAddField
            GET SAIS ID using saisAddField
            GET student number using studNoAddField
            GET address using addressAddField
            IF input is proper THEN
                CHECK IF entry is unique and
                total entries are less than 10 THEN
                    CALL addData()
                ELSE DISPLAY error message
            ELSE DISPLAY error message
        IF deleteButton is clicked THEN
            DISPLAY deleteCard and CALL createPanel()
            IF deleteEntryBtn is clicked THEN
                DISPLAY a new window with deleteFrame
                IF yesDeleteBtn is clicked THEN
                    CALL deleteData()
                IF noDeleteBtn is clicked THEN
                    DISPOSE the window
        IF editButton is clicked THEN
            DISPLAY editCard and CALL createPanel()
            IF editEntryBtn is clicked THEN
                CALL editData()
        IF searchButton is clicked THEN
            DISPLAY searchCard and CALL createPanel()
            IF searchEntryBtn is clicked THEN
                CALL searchData()
        IF homeButton is clicked THEN
            DISPLAY topCard
    IF mainExitButton is clicked THEN
        STOP the program
END FUNCTION
```

Figure 15. Frontend pseudocode.

Algorithms

Functions

The primary functions of the program are defined in *StudentDB* class. There are seven primary functions used in the program, namely: ***addData()***, ***deleteData()***, ***editData()***, ***showData()***, ***printData()***, ***read()***, and ***searchData()***.

The function showData() has already been described in the previous section. Refer to the Backend section for details.

The boolean function ***addData()*** has only one parameter: an instance of StudentData. The function calls the ***printData()*** function to store the data into a file. This function is used in adding entries and editing entries. If the entry has been added successfully, the function returns a **true** value and **false** otherwise. Before the function gets called, every property of StudentData gets verified first using boolean variables.

The void function ***printData()*** stores the data into a file using PrintWriter. It has only one parameter similar to ***addData()***. Each entry is printed into a standard text document or a TXT file. The filenames follow the format [Student Name <space> SAIS ID]. This is to make sure that no entries will be repeated. All files are saved in the same directory.

```
1  // Parameter: StudentData
2  // Returns a boolean
3  FUNCTION addData()
4      CALL printData(StudentData)
5      IF successful return true
6      ELSE return false
7  END FUNCTION
8
9  // Parameter: StudentData
10 // Void
11 FUNCTION printData()
12     OPEN PrintWriter
13     THEN set filename to name + SAISID
14     THEN print to file:
15         name + \n +
16         SAISID + \n +
17         num + \n +
18         address
19     CLOSE PrintWriter
20 END FUNCTION
```

Figure 16. *addData()* and *printData()* pseudocode.

Functions

The boolean function ***deleteData()*** has two parameters: String ***name*** and int ***SAISID***. The parameters are used to identify the file that needs to be deleted. If the entry has been deleted successfully, the function returns a true value and false otherwise.

```
1  // Parameters: name, SAISID
2  // Returns a boolean
3  FUNCTION deleteData()
4      set filename to name + SAISID
5      THEN delete file
6      IF successful return true
7      ELSE return false
8  END FUNCTION
9
```

Figure 17. *deleteData()* pseudocode.

The StudentData array function ***searchData()*** checks the occurrence of the parameter, String ***toSearch***, in the existing entries. It calls the function ***showData()*** to retrieve the array of StudentData and evaluates each property of each StudentData. If a property of StudentData contains String ***toSearch***, that is if the character or string is present, it adds the entire object to another StudentData array ***searches***. The function returns StudentData array ***searches*** to be displayed.

```
1  // Parameter: String toSearch
2  // Returns an array
3  FUNCTION searchData()
4      INIT array students and call showData()
5      INIT array searches
6      LOOP FOR every student:
7          INIT line: name + SAISID + num  + address
8          IF toSearch in line THEN
9              ADD student in searches
10     return searches
11 END FUNCTION
```

Figure 18. *searchData()* pseudocode.

Algorithms

Functions

The function ***createPanel()*** allows the display of entries using JPanels. For every entry (or StudentData), it creates a panel with the entry details. It collates these panels to entryCard which is triggered by clicking the "View All Entries" button.

```
// Parameters: JPanel, JFrame, int state (to identify if edit or delete)
// Void
FUNCTION createPanel()
    SET dimensions
    THEN SET entryCard
    INIT array students and call showData()
    LOOP FOR every student:
        INIT panel THEN INIT panel properties
        ADD panel to entryCard
        IF called from deleteCard THEN ADD deleteEntryBtn
        IF called from editCard THEN ADD editEntryBtn
        GET name THEN SET nameLabel and nameInfo THEN ADD to panel
        GET SAISID THEN SET SAISIDLabel and SAISIDInfo THEN ADD to panel
        GET num THEN SET numLabel and numInfo THEN ADD to panel
        GET address THEN SET addressLabel and addressInfo THEN ADD to panel
    REPAINT entryCard
END FUNCTION
```

Figure 19. *createPanel()* pseudocode

The function ***editData()*** is defined in *StudentDB* class but it is also incorporated in *StudentDBDemo* class. The verification of inputs happens in *StudentDBDemo* class and the modification happens in *StudentDB* class. Once the modification is done, the file is reprinted with the changes by calling ***addData()***.

```
// Paramaters: JComboBox, name, SAISID, num, address
// Void
FUNCTION editData()
    GET choice FROM ComboBox
    GET name, SAISID, num, address from file
    IF choice is name THEN
        IF input is proper THEN
            CHANGE name to input
            CHANGE filename
    IF choice is SAISID THEN
        IF input is proper THEN
            CHANGE SAISID to input
            CHANGE filename
    IF choice is num THEN
        IF input is proper THEN
            CHANGE num to input
    IF choice is address THEN
        IF input is proper THEN
            CHANGE address to input
    INIT StudentData(name, SAISID, num, address)
    CALL addData(StudentData)
END FUNCTION
```

```
// No paramater
// Returns an array
FUNCTION showData()
    INIT array students [10]
    SET file directory
    THEN INIT array fi[illegible]
    LOOP FOR every file:
        OPEN Scanner
        GET name
        GET SAISID
        GET num
        GET address
        INIT StudentData(name, SAISID, num, address)
        CLOSE Scanner
    INIT array students2
    IF students < 10 THEN
        LOOP FOR every student:
            IF STUDENT is not NULL THEN
                ADD student to students2
            ELSE BREAK
        return students2
    ELSE return students
END FUNCTION
```

Figure 20. *editData()* (left) and *showData()* (right) pseudocodes

Error Handling

Booleans

- The boolean ***isIncomplete*** verifies if an entry is empty. This is to make sure that there will be no missing properties in the student data.
- The boolean ***isWhiteSpace*** verifies if an entry is composed only of whitespaces. Similar to ***isIncomplete***, this is to prevent missing properties.
- The boolean ***isProperLength*** limits the length of integer inputs. The integers have a maximum possible value of 2,147,483,647, thus the inputs are limited to nine place values.
- The boolean ***isDigit*** confirms that the entry is only composed of numerics using the regular expression "[0-9]+".
- The boolean ***isIdentical*** checks if there is an existing entry with the same name and SAIS ID.
- The boolean ***isEntryListMax*** checks if the incoming entry can still be accommodated in the list of all entries as it has a limit of 10.

Input Errors

Input errors are handled using the boolean variables mentioned above. For example. to make sure that entries (or inputs) are not assigned to incompatible data types, boolean ***isDigit*** is utilized. Input errors were also minimized with the use of GUI. Instead of having to input a character to perform a function, simple buttons are employed.

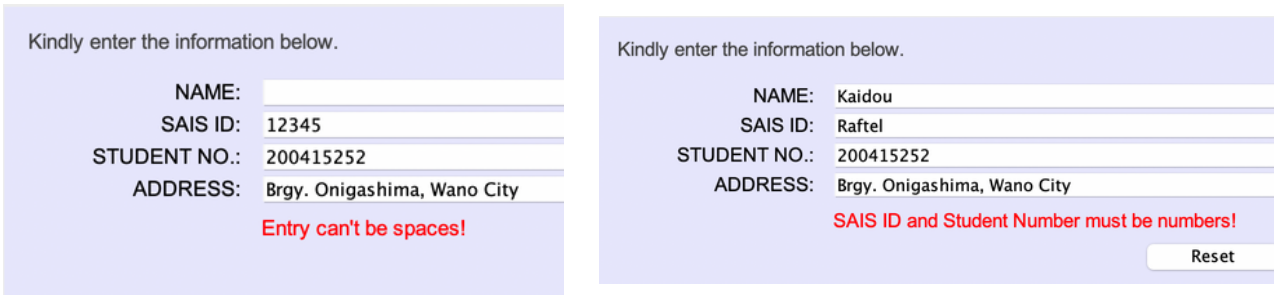


Figure 21. Error handling in action.

Error Handling

Exceptions

There is one main exception encountered in the program: FileNotFoundException. The exception was either thrown or enclosed in a try-catch block. In the instances where the exception wouldn't terminate the program, a throws clause was used. In the other instances, try-catch blocks were used. A corresponding error message is printed to the terminal when the exception is caught. These are, however, only safety nets. The program is expected to run smoothly without ever triggering the exception.

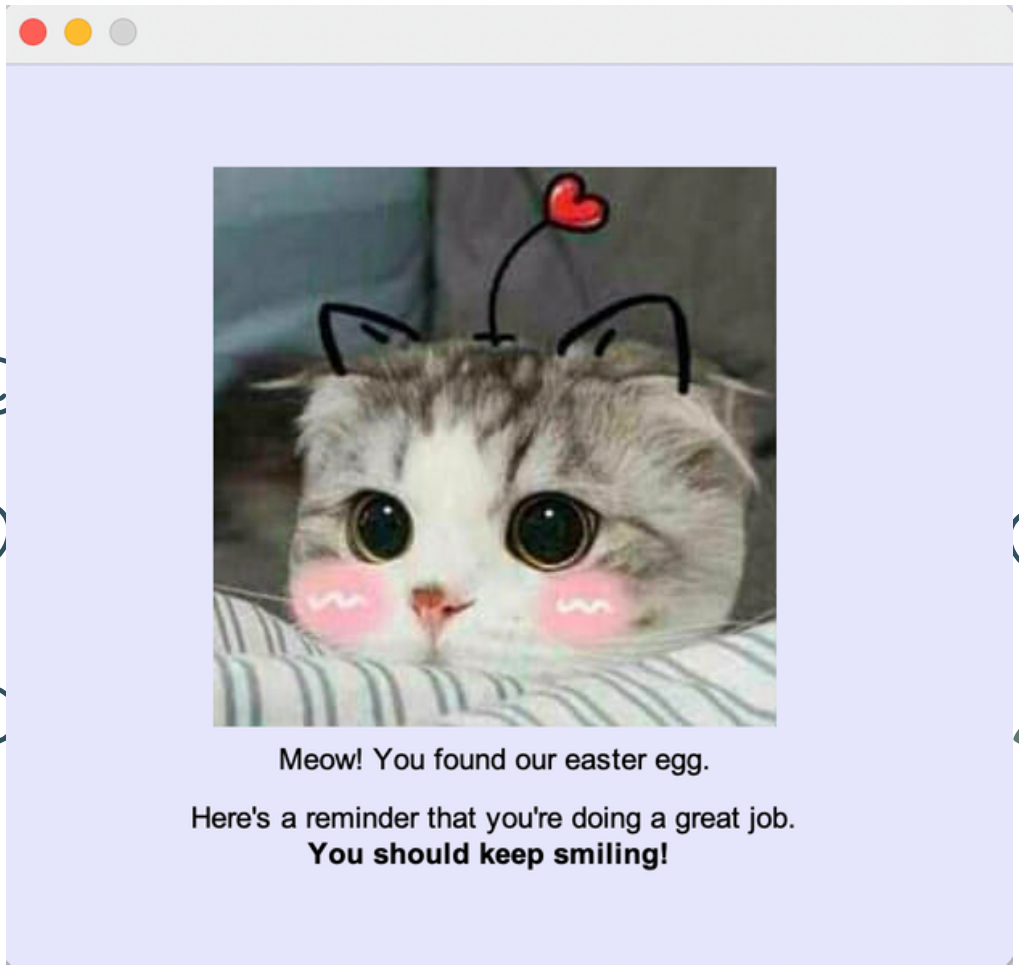
```
public static void main(String[] args) {
    EventQueue.invokeLater(new Runnable() {
        public void run() {
            try {
                new StudentDBDemo();
            } catch (Exception e1) {
                e1.printStackTrace();
            }
        }
    });
}

public StudentDBDemo() throws FileNotFoundException {

    // Calls the function that starts the GUI algorithm
    mainGUI();
}
```

Figure 22. An example of how throwing exceptions and try-catch blocks are implemented.

Meow!

Easter Egg

I mean... why not? This can be triggered using the "Add an Entry" menu option in which certain user input is accepted.



Hint: Call me by my sound twice and I'll be happy.
Then, increment SAIS and make it a binary.