CS203 Project Report Template

Homework #3

Justin Lewis, Jdlewis2

Introduction

The goal of this project was to add and design a student and employee tracking system for UAB hospitals. The system will offer automation for listing, deleting, and adding UAB students and employees, including staff like faculty, medical staff, office associates, IT professionals, and advisors. The project requires the use of inheritance to create various classes, for instance, UABPerson as a superclass and subclasses for each role and it requires putting these objects into arrays or ArrayLists. The program will read and update data in a text file (uabEmployee.txt), and the program will hold a basic text-based user interface to handle the data with the ability to add, delete, and display UAB personnel based on specific roles.

Code Explanation

In my Hw3 package, I made 10 different classes. I will list the names and functionality of each.

- UABPerson: This class is pretty much the foundation for all UAB people. It holds the shared info that applies to everyone such as name and ID , and the other classes build on it to add specific roles (like departments for Advisors, or majors for Students.

- Student: This is an extended version of UABPerson that represents students by putting in level, major, and GPA. It allows you to construct, modify, and print student data in a structured way.

- Advisors: This class copies an advisor at UAB with department and group data and allows you to add, retrieve, update, and print those records. It inherits the generic UABPerson class with advisor properties.

- Doctor: The Doctor class acts as a doctor at UAB and is a subclass of MedicalStaff, which further inherits UABPerson. It doesn't add any new variables or methods but uses the inherited properties like name, blazer ID, role, and department to construct and work with doctor objects, allowing me to handle doctors as a unique group.

- Nurse: The Nurse class acts as a nurse at UAB and is also a subclass of MedicalStaff with the addition of an onCall status specific to it. You can make nurse objects with all normal staff data, such as whether they're on call or not, and it prints that when you do.

- Faculty: This class manages and displays data specific to faculty members such as department and how many courses they teach.

- ITProfessional: This class is in charge of representing IT workers at UAB. It stores their basic info (name and BlazerID, from the UABPerson class) and adds extra info about which team they work on (like Developer or Networking).

- MedicalStaff: This is an abstract base class extending UABPerson for all medical personnel. It provides details and behavior for medical roles like doctors and nurses. This class can't be used to create objects directly; it must be extended by medical staff classes like my nurse and doctor classes

- OfficeAssociates: This simply adds role and department attributes specific to office personnel

- BlazerEMS: This class is where all the work is done. It Manages different types of personnel (Faculty, Students, Staff, Medical), It contains functions that displays, adds, and deletes members. It also saves/loads data from the file UABEmployeetxt. As you may be able to tell, the class is pretty long in lines and the primary reason for that would be because I also added navigation options to go back at any input stage if the user makes a mistake or just simply wants to go back. The go back feature is in the AddMembers function and the DeleteMembers function.

For OOP implementation, I used:

- Inheritance by having all the specific roles inherit from the UABPerson base class.

- Polymorphism by holding and handling different employee types in a single ArrayList.

- Encapsulation through proper getters and setters for class attributes.

- Method overriding, especially for the toString() method in each subclass.

The main BlazerEMS class is the system that manages everything, with methods for:

- Loading data from a file (getFromFile)

- Saving data to a file (saveToFile)

- Showing all members grouped by type (ShowAllMembers)

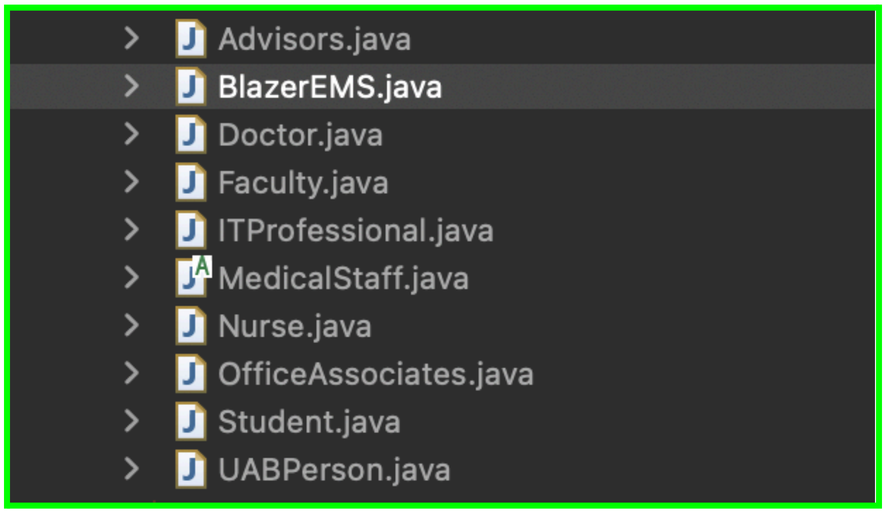
- Adding new members with validation (AddMembers)

- Deleting members (DeleteMembers)

I also added a simple text based menu in the main method so users can interact with the system like showing all members, adding or deleting people, saving data, or exiting the program. For saving and loading data, I used file input/output, with a custom format where each line in the file represents a person, starting with a letter for their role (like F for Faculty, S for Student), followed by their details. I added a lot of input checks and a way for users to go back if they make a mistake or want to change their choice, so they don’t have to restart the program every time

Result

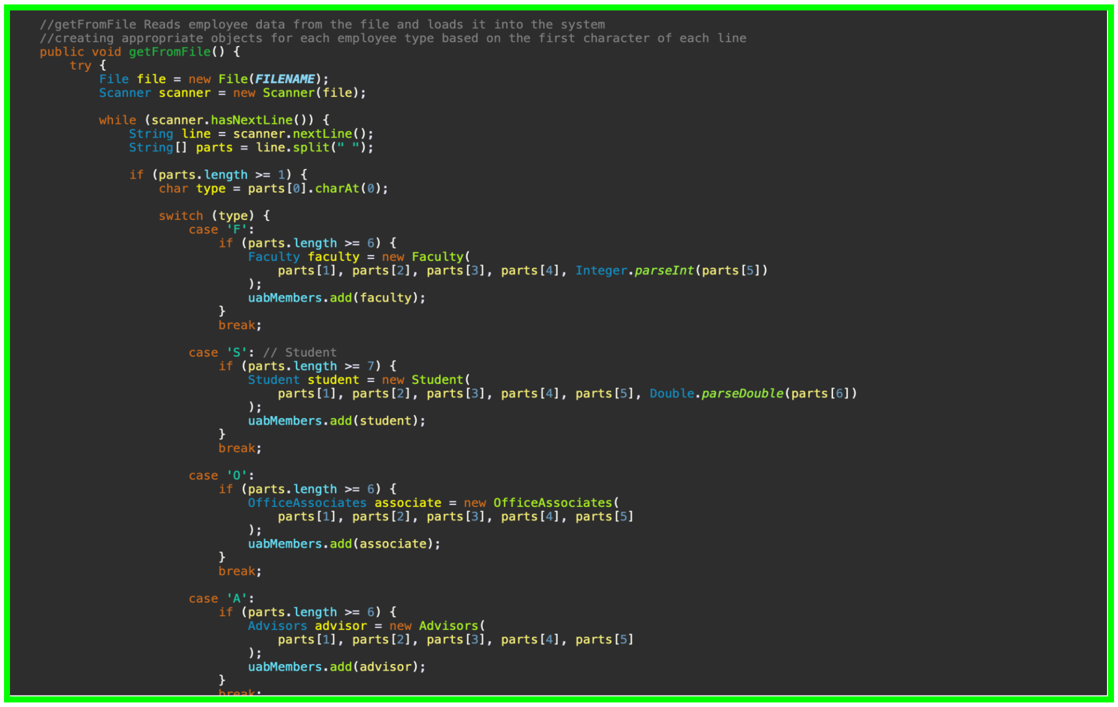
Class Setup:



```
Advisors.java
BlazerEMS.java
Doctor.java
Faculty.java
ITProfessional.java
MedicalStaff.java
Nurse.java
OfficeAssociates.java
Student.java
UABPerson.java
```

First off, this is how all of my classes are setup, there are 10 of them.

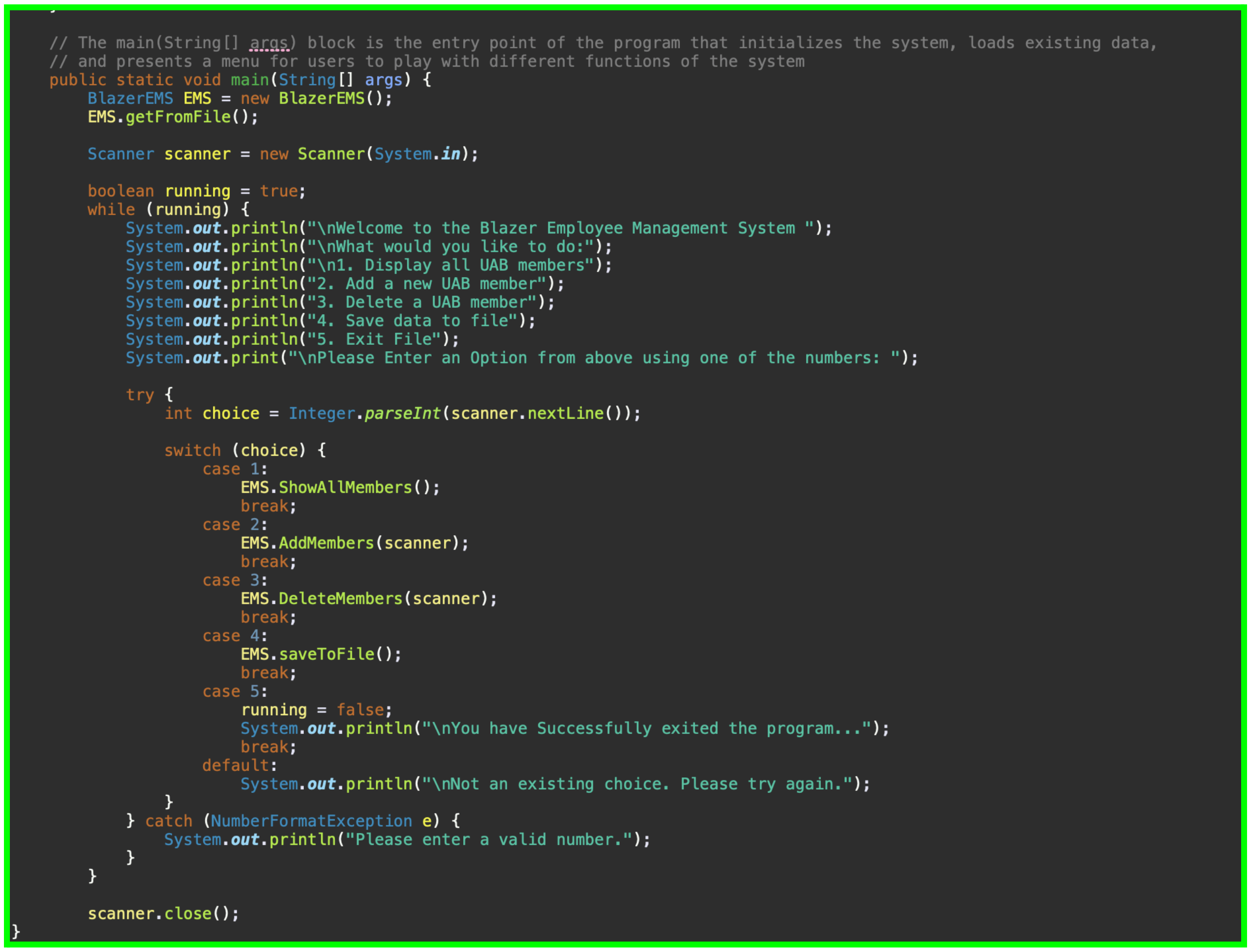
getFromFile:



```java
//getFromFile Reads employee data from the file and loads it into the system
//creating appropriate objects for each employee type based on the first character of each line
public void getFromFile() {
    try {
        File file = new File(FILENAME);
        Scanner scanner = new Scanner(file);

        while (scanner.hasNextLine()) {
            String line = scanner.nextLine();
            String[] parts = line.split(" ");

            if (parts.length >= 1) {
                char type = parts[0].charAt(0);

                switch (type) {
                    case 'F':
                        if (parts.length >= 6) {
                            Faculty faculty = new Faculty(
                                parts[1], parts[2], parts[3], parts[4], Integer.parseInt(parts[5])
                            );
                            uabMembers.add(faculty);
                        }
                        break;

                    case 'S': // Student
                        if (parts.length >= 7) {
                            Student student = new Student(
                                parts[1], parts[2], parts[3], parts[4], parts[5], Double.parseDouble(parts[6])
                            );
                            uabMembers.add(student);
                        }
                        break;

                    case 'O':
                        if (parts.length >= 6) {
                            OfficeAssociates associate = new OfficeAssociates(
                                parts[1], parts[2], parts[3], parts[4], parts[5]
                            );
                            uabMembers.add(associate);
                        }
                        break;

                    case 'A':
                        if (parts.length >= 6) {
                            Advisors advisor = new Advisors(
                                parts[1], parts[2], parts[3], parts[4], parts[5]
                            );
                            uabMembers.add(advisor);
                        }
                        break;
```

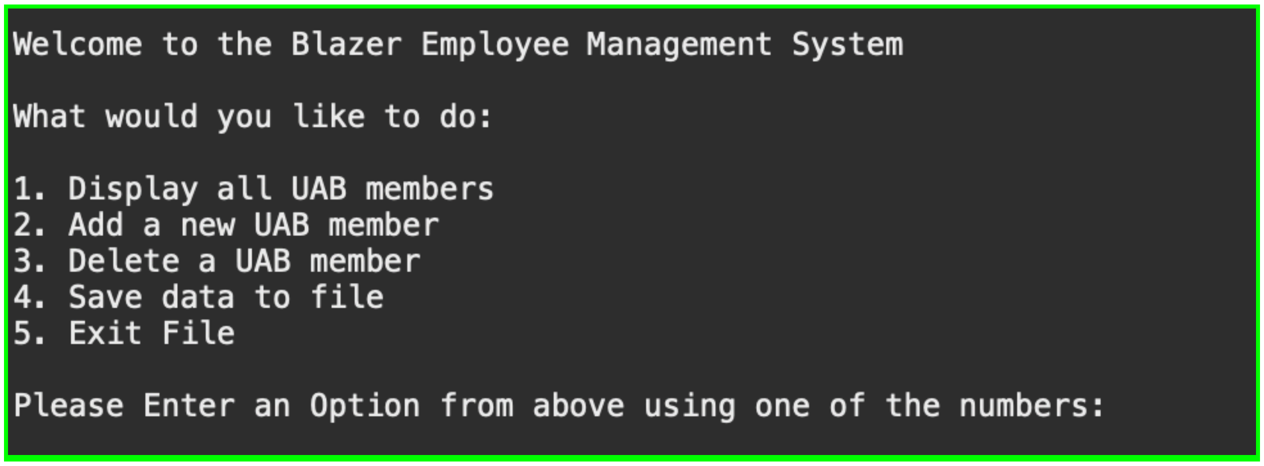
My getFromFile block in my BlazerEms class Reads employee data from the file and loads it into the system.

main(String[] args):



```java
    // The main(String[] args) block is the entry point of the program that initializes the system, loads existing data,
    // and presents a menu for users to play with different functions of the system
    public static void main(String[] args) {
        BlazerEMS EMS = new BlazerEMS();
        EMS.getFromFile();

        Scanner scanner = new Scanner(System.in);

        boolean running = true;
        while (running) {
            System.out.println("\nWelcome to the Blazer Employee Management System ");
            System.out.println("\nWhat would you like to do:");
            System.out.println("\n1. Display all UAB members");
            System.out.println("2. Add a new UAB member");
            System.out.println("3. Delete a UAB member");
            System.out.println("4. Save data to file");
            System.out.println("5. Exit File");
            System.out.print("\nPlease Enter an Option from above using one of the numbers: ");

            try {
                int choice = Integer.parseInt(scanner.nextLine());

                switch (choice) {
                    case 1:
                        EMS.ShowAllMembers();
                        break;
                    case 2:
                        EMS.AddMembers(scanner);
                        break;
                    case 3:
                        EMS.DeleteMembers(scanner);
                        break;
                    case 4:
                        EMS.saveToFile();
                        break;
                    case 5:
                        running = false;
                        System.out.println("\nYou have Successfully exited the program...");
                        break;
                    default:
                        System.out.println("\nNot an existing choice. Please try again.");
                }
            } catch (NumberFormatException e) {
                System.out.println("Please enter a valid number.");
            }
        }

        scanner.close();
}
```

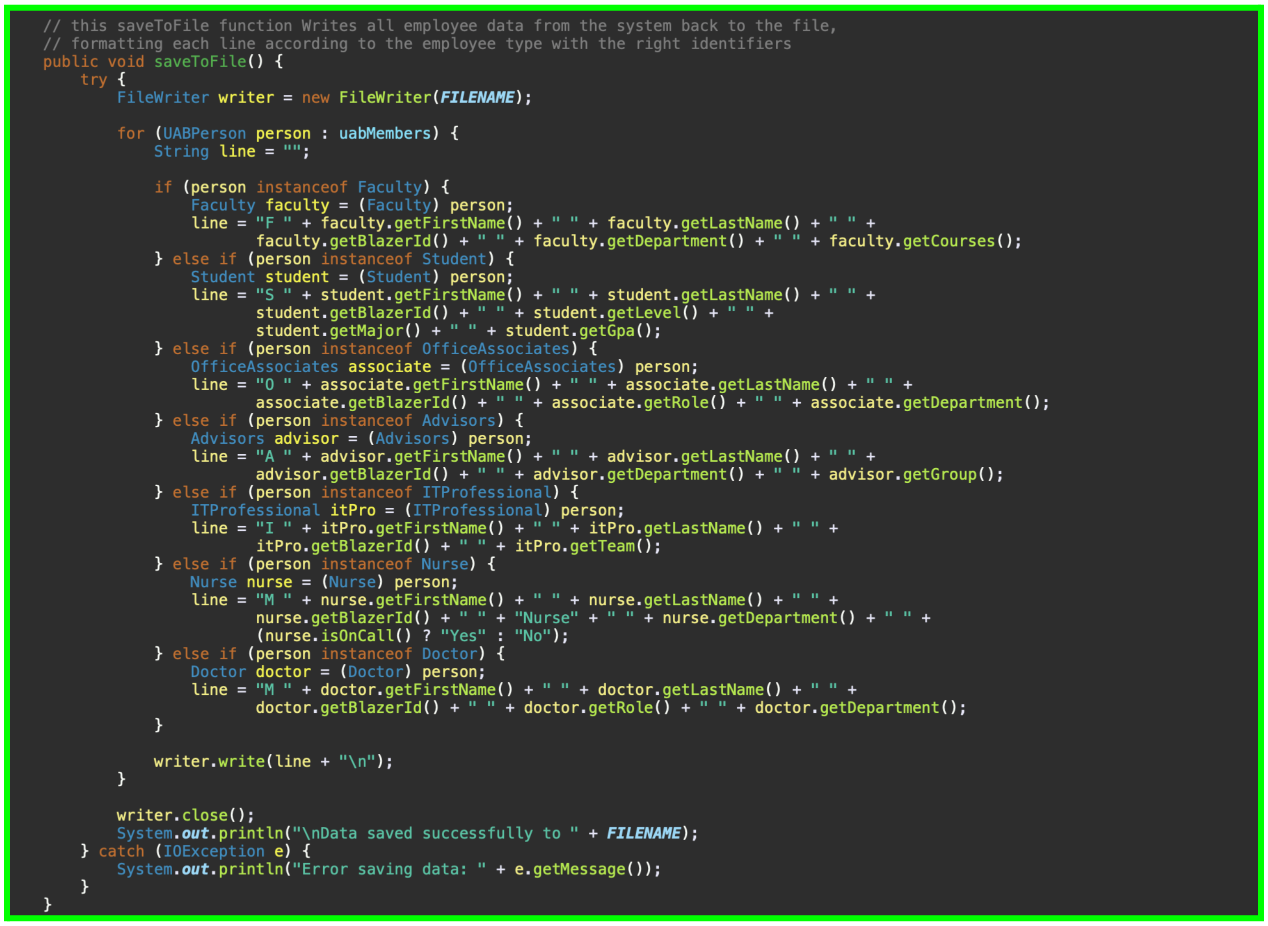
The main(String[] args) function is the entry point of the program that initializes the system, loads existing data, and presents a menu for users to play with different functions of the system. This function is the reason you see:



```
Welcome to the Blazer Employee Management System

What would you like to do:

1. Display all UAB members
2. Add a new UAB member
3. Delete a UAB member
4. Save data to file
5. Exit File

Please Enter an Option from above using one of the numbers:
```

When you first run the code.

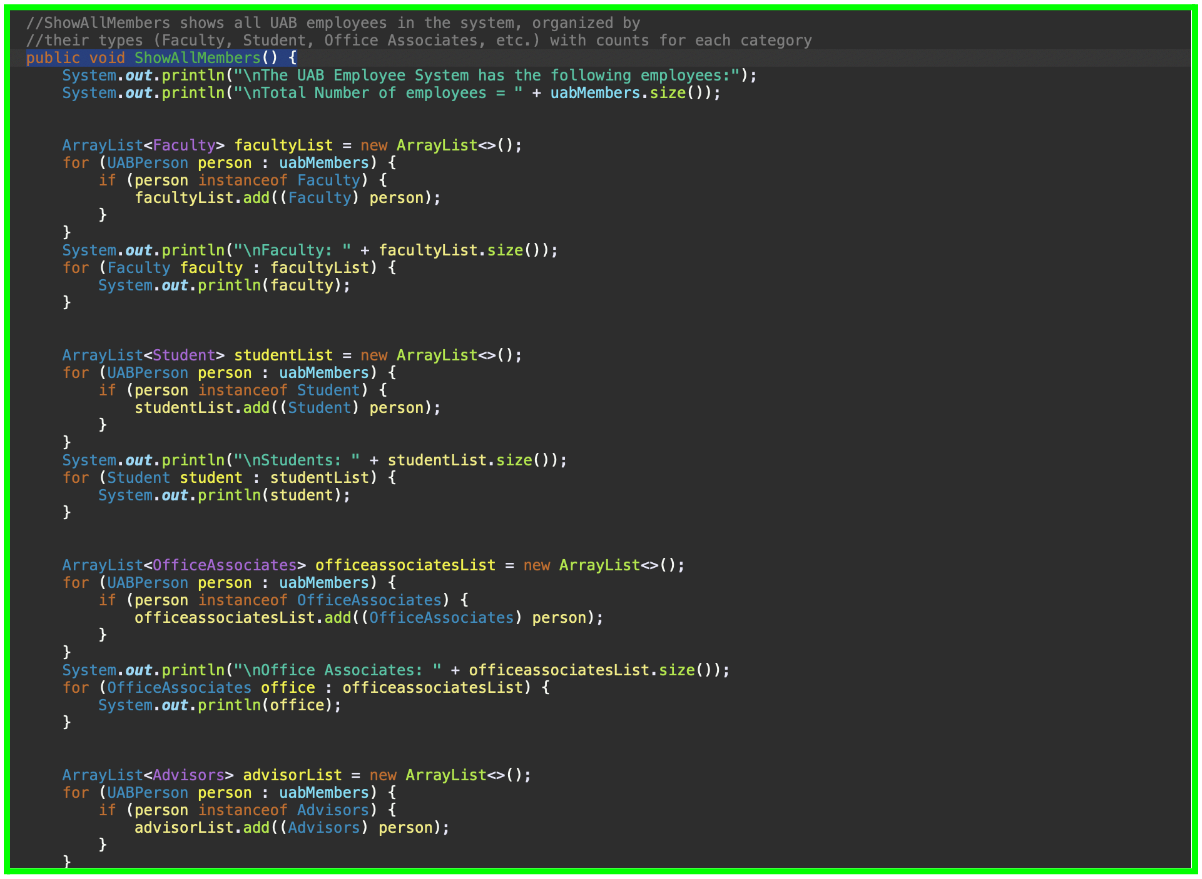
saveToFile:



```java
// this saveToFile function Writes all employee data from the system back to the file,
// formatting each line according to the employee type with the right identifiers
public void saveToFile() {
    try {
        FileWriter writer = new FileWriter(FILENAME);

        for (UABPerson person : uabMembers) {
            String line = "";

            if (person instanceof Faculty) {
                Faculty faculty = (Faculty) person;
                line = "F " + faculty.getFirstName() + " " + faculty.getLastName() + " " +
                       faculty.getBlazerId() + " " + faculty.getDepartment() + " " + faculty.getCourses();
            } else if (person instanceof Student) {
                Student student = (Student) person;
                line = "S " + student.getFirstName() + " " + student.getLastName() + " " +
                       student.getBlazerId() + " " + student.getLevel() + " " +
                       student.getMajor() + " " + student.getGpa();
            } else if (person instanceof OfficeAssociates) {
                OfficeAssociates associate = (OfficeAssociates) person;
                line = "O " + associate.getFirstName() + " " + associate.getLastName() + " " +
                       associate.getBlazerId() + " " + associate.getRole() + " " + associate.getDepartment();
            } else if (person instanceof Advisors) {
                Advisors advisor = (Advisors) person;
                line = "A " + advisor.getFirstName() + " " + advisor.getLastName() + " " +
                       advisor.getBlazerId() + " " + advisor.getDepartment() + " " + advisor.getGroup();
            } else if (person instanceof ITProfessional) {
                ITProfessional itPro = (ITProfessional) person;
                line = "I " + itPro.getFirstName() + " " + itPro.getLastName() + " " +
                       itPro.getBlazerId() + " " + itPro.getTeam();
            } else if (person instanceof Nurse) {
                Nurse nurse = (Nurse) person;
                line = "M " + nurse.getFirstName() + " " + nurse.getLastName() + " " +
                       nurse.getBlazerId() + " " + "Nurse" + " " + nurse.getDepartment() + " " +
                       (nurse.isOnCall() ? "Yes" : "No");
            } else if (person instanceof Doctor) {
                Doctor doctor = (Doctor) person;
                line = "M " + doctor.getFirstName() + " " + doctor.getLastName() + " " +
                       doctor.getBlazerId() + " " + doctor.getRole() + " " + doctor.getDepartment();
            }

            writer.write(line + "\n");
        }

        writer.close();
        System.out.println("\nData saved successfully to " + FILENAME);
    } catch (IOException e) {
        System.out.println("Error saving data: " + e.getMessage());
    }
}
```

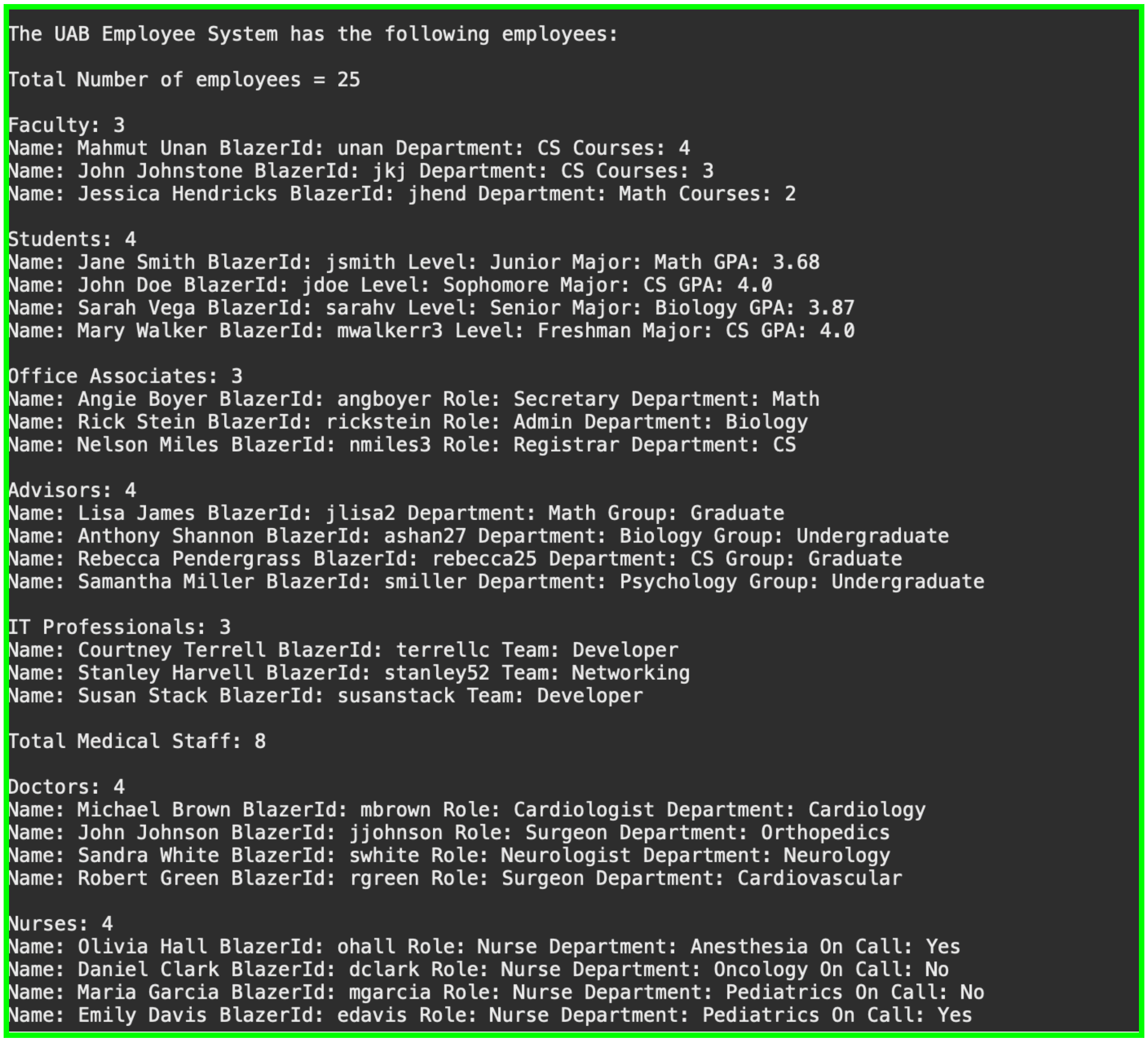
The saveToFile function writes all employee data from the system to the UABEmployee.txt file. It formats each employee record by their type using letters like F for Faculty, etc., It converts all employee objects into properly formatted text strings, and then saves everything to the external file for permanent storage. saveToFile allows the program to restore the data when it's launched again later.

ShowAllMembers:



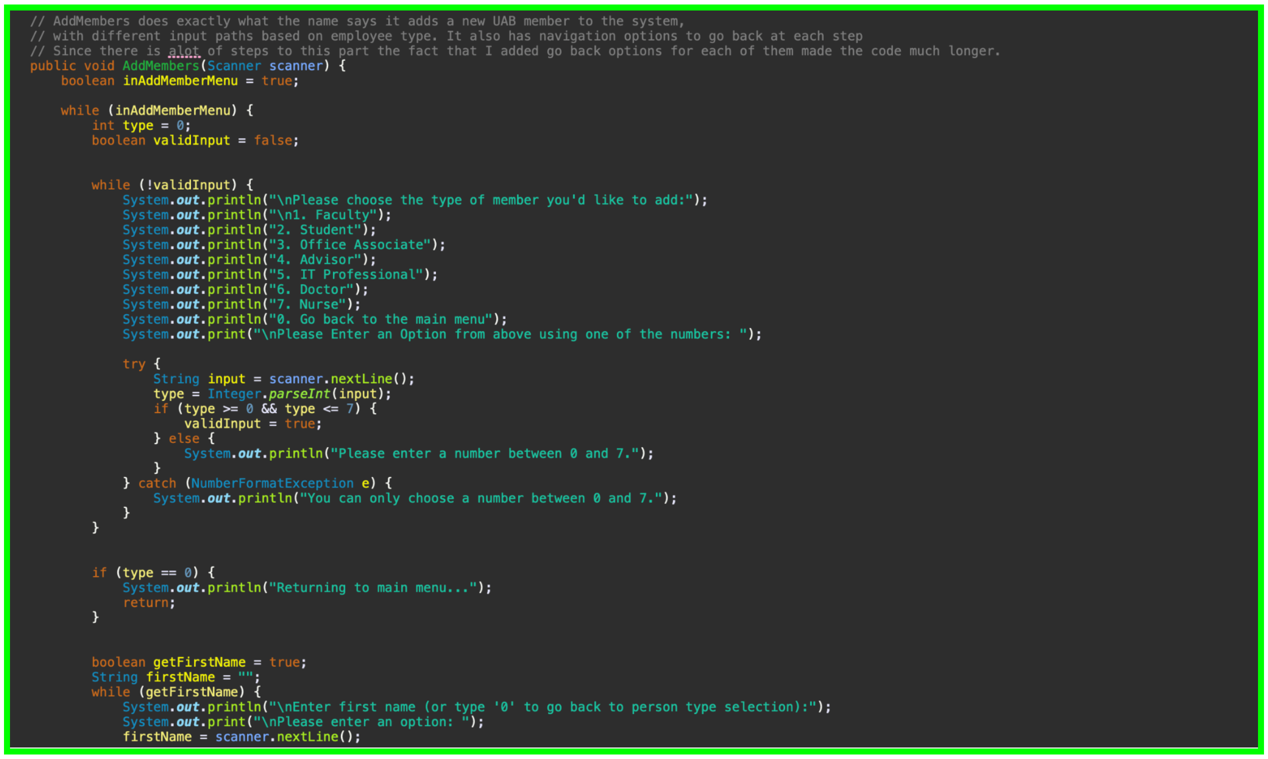
```java
//ShowAllMembers shows all UAB employees in the system, organized by
//their types (Faculty, Student, Office Associates, etc.) with counts for each category
public void ShowAllMembers() {
    System.out.println("\nThe UAB Employee System has the following employees:");
    System.out.println("\nTotal Number of employees = " + uabMembers.size());


    ArrayList<Faculty> facultyList = new ArrayList<>();
    for (UABPerson person : uabMembers) {
        if (person instanceof Faculty) {
            facultyList.add((Faculty) person);
        }
    }
    System.out.println("\nFaculty: " + facultyList.size());
    for (Faculty faculty : facultyList) {
        System.out.println(faculty);
    }


    ArrayList<Student> studentList = new ArrayList<>();
    for (UABPerson person : uabMembers) {
        if (person instanceof Student) {
            studentList.add((Student) person);
        }
    }
    System.out.println("\nStudents: " + studentList.size());
    for (Student student : studentList) {
        System.out.println(student);
    }


    ArrayList<OfficeAssociates> officeassociatesList = new ArrayList<>();
    for (UABPerson person : uabMembers) {
        if (person instanceof OfficeAssociates) {
            officeassociatesList.add((OfficeAssociates) person);
        }
    }
    System.out.println("\nOffice Associates: " + officeassociatesList.size());
    for (OfficeAssociates office : officeassociatesList) {
        System.out.println(office);
    }


    ArrayList<Advisors> advisorList = new ArrayList<>();
    for (UABPerson person : uabMembers) {
        if (person instanceof Advisors) {
            advisorList.add((Advisors) person);
        }
    }
```

The ShowAllMembers method shows an organized list of all employees in the UAB system. It shows the total number of employees, followed by a breakdown of each category (Faculty, Students, Office Associates, etc.) with counts and individual listings for each person. This is how it looks once ran:



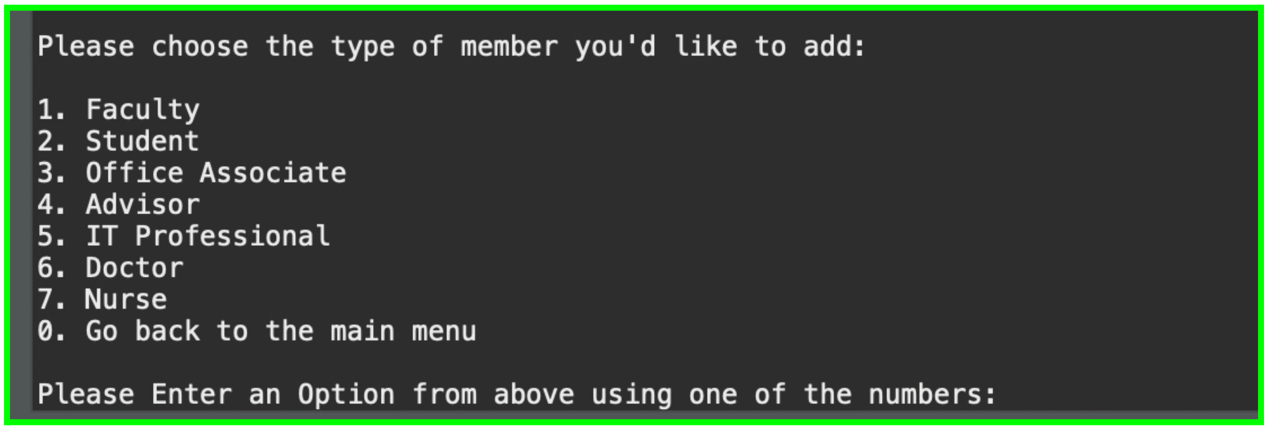
```
The UAB Employee System has the following employees:

Total Number of employees = 25

Faculty: 3
Name: Mahmut Unan BlazerId: unan Department: CS Courses: 4
Name: John Johnstone BlazerId: jkj Department: CS Courses: 3
Name: Jessica Hendricks BlazerId: jhend Department: Math Courses: 2

Students: 4
Name: Jane Smith BlazerId: jsmith Level: Junior Major: Math GPA: 3.68
Name: John Doe BlazerId: jdoe Level: Sophomore Major: CS GPA: 4.0
Name: Sarah Vega BlazerId: sarahv Level: Senior Major: Biology GPA: 3.87
Name: Mary Walker BlazerId: mwalkerr3 Level: Freshman Major: CS GPA: 4.0

Office Associates: 3
Name: Angie Boyer BlazerId: angboyer Role: Secretary Department: Math
Name: Rick Stein BlazerId: rickstein Role: Admin Department: Biology
Name: Nelson Miles BlazerId: nmiles3 Role: Registrar Department: CS

Advisors: 4
Name: Lisa James BlazerId: jlisa2 Department: Math Group: Graduate
Name: Anthony Shannon BlazerId: ashan27 Department: Biology Group: Undergraduate
Name: Rebecca Pendergrass BlazerId: rebecca25 Department: CS Group: Graduate
Name: Samantha Miller BlazerId: smiller Department: Psychology Group: Undergraduate

IT Professionals: 3
Name: Courtney Terrell BlazerId: terrellc Team: Developer
Name: Stanley Harvell BlazerId: stanley52 Team: Networking
Name: Susan Stack BlazerId: susanstack Team: Developer

Total Medical Staff: 8

Doctors: 4
Name: Michael Brown BlazerId: mbrown Role: Cardiologist Department: Cardiology
Name: John Johnson BlazerId: jjohnson Role: Surgeon Department: Orthopedics
Name: Sandra White BlazerId: swhite Role: Neurologist Department: Neurology
Name: Robert Green BlazerId: rgreen Role: Surgeon Department: Cardiovascular

Nurses: 4
Name: Olivia Hall BlazerId: ohall Role: Nurse Department: Anesthesia On Call: Yes
Name: Daniel Clark BlazerId: dclark Role: Nurse Department: Oncology On Call: No
Name: Maria Garcia BlazerId: mgarcia Role: Nurse Department: Pediatrics On Call: No
Name: Emily Davis BlazerId: edavis Role: Nurse Department: Pediatrics On Call: Yes
```

AddMembers:



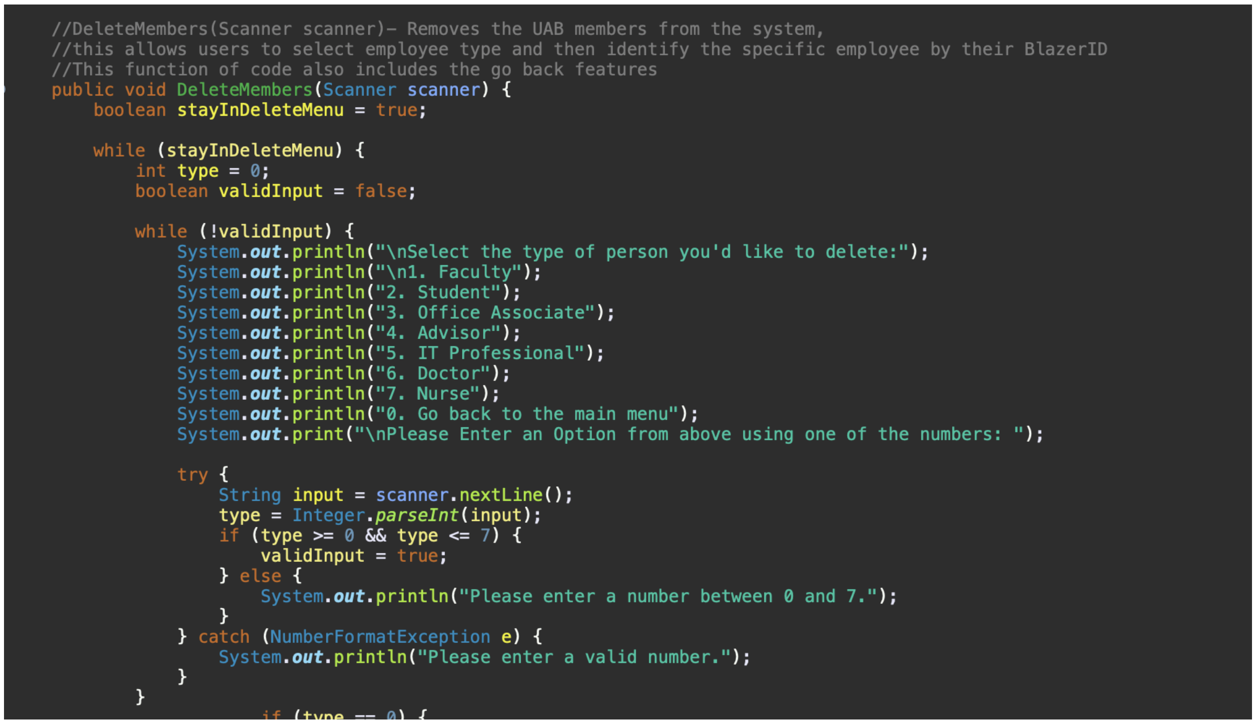
```java
// AddMembers does exactly what the name says it adds a new UAB member to the system,
// with different input paths based on employee type. It also has navigation options to go back at each step
// Since there is alot of steps to this part the fact that I added go back options for each of them made the code much longer.
public void AddMembers(Scanner scanner) {
    boolean inAddMemberMenu = true;

    while (inAddMemberMenu) {
        int type = 0;
        boolean validInput = false;


        while (!validInput) {
            System.out.println("\nPlease choose the type of member you'd like to add:");
            System.out.println("\n1. Faculty");
            System.out.println("2. Student");
            System.out.println("3. Office Associate");
            System.out.println("4. Advisor");
            System.out.println("5. IT Professional");
            System.out.println("6. Doctor");
            System.out.println("7. Nurse");
            System.out.println("0. Go back to the main menu");
            System.out.print("\nPlease Enter an Option from above using one of the numbers: ");

            try {
                String input = scanner.nextLine();
                type = Integer.parseInt(input);
                if (type >= 0 && type <= 7) {
                    validInput = true;
                } else {
                    System.out.println("Please enter a number between 0 and 7.");
                }
            } catch (NumberFormatException e) {
                System.out.println("You can only choose a number between 0 and 7.");
            }
        }


        if (type == 0) {
            System.out.println("Returning to main menu...");
            return;
        }


        boolean getFirstName = true;
        String firstName = "";
        while (getFirstName) {
            System.out.println("\nEnter first name (or type '0' to go back to person type selection):");
            System.out.print("\nPlease enter an option: ");
            firstName = scanner.nextLine();
```

The AddMembers method guides users through adding new members to the system by a series of prompted inputs. It first asks for the type of member to add (Faculty, Student, etc.), then collects required information like name and ID, followed by role-specific details. The method includes a "go back" feature at each step, allowing users to correct mistakes by returning to previous inputs before committing the new member to the system, since the AddMembers function had so many steps adding the go back feature to all the steps made the line count super long… This process was truly exhausting but I'm happy with how it came out. Here's how it looks once you choose this option:



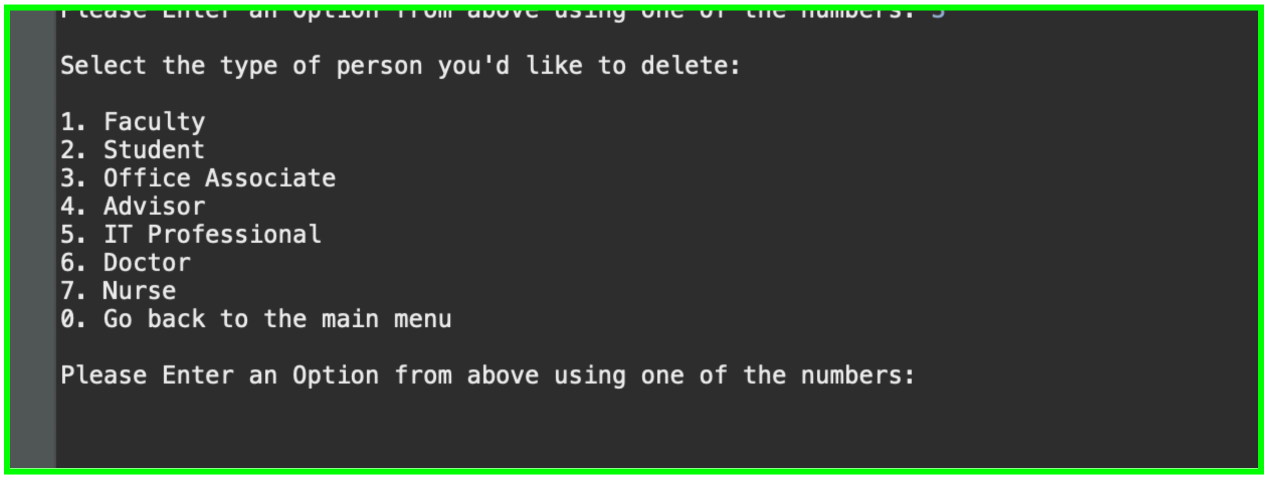
```
Please choose the type of member you'd like to add:

1. Faculty
2. Student
3. Office Associate
4. Advisor
5. IT Professional
6. Doctor
7. Nurse
0. Go back to the main menu

Please Enter an Option from above using one of the numbers:
```

DeleteMembers:



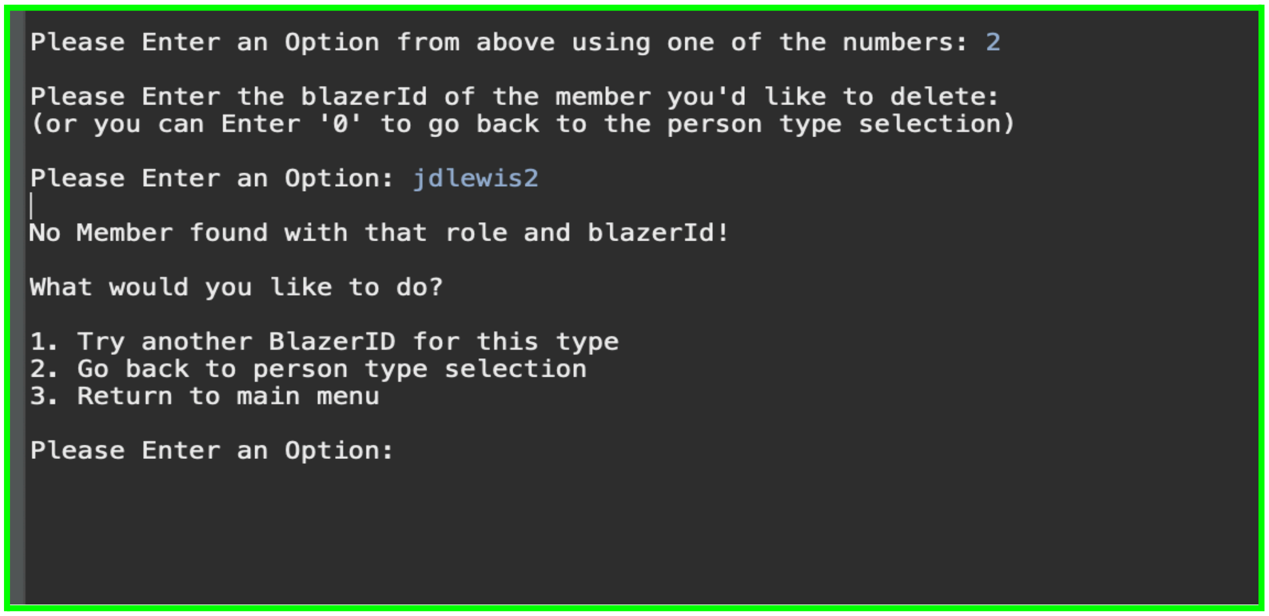
```java
//DeleteMembers(Scanner scanner)- Removes the UAB members from the system,
//this allows users to select employee type and then identify the specific employee by their BlazerID
//This function of code also includes the go back features
public void DeleteMembers(Scanner scanner) {
    boolean stayInDeleteMenu = true;

    while (stayInDeleteMenu) {
        int type = 0;
        boolean validInput = false;

        while (!validInput) {
            System.out.println("\nSelect the type of person you'd like to delete:");
            System.out.println("\n1. Faculty");
            System.out.println("2. Student");
            System.out.println("3. Office Associate");
            System.out.println("4. Advisor");
            System.out.println("5. IT Professional");
            System.out.println("6. Doctor");
            System.out.println("7. Nurse");
            System.out.println("0. Go back to the main menu");
            System.out.print("\nPlease Enter an Option from above using one of the numbers: ");

            try {
                String input = scanner.nextLine();
                type = Integer.parseInt(input);
                if (type >= 0 && type <= 7) {
                    validInput = true;
                } else {
                    System.out.println("Please enter a number between 0 and 7.");
                }
            } catch (NumberFormatException e) {
                System.out.println("Please enter a valid number.");
            }
        }
                if (type == 0) {
```

The DeleteMembers function is the complete opposite of the AddMembers function; it allows users to remove members from the system using 2 steps. Users first select the type of employee to delete (Faculty, Student, etc.), then enter the BlazerID of the specific person they wish to remove. This method also includes the navigation options to try another ID, return to type selection, or exit to the main menu if the specified person isn't found so the user won't have to run the code all over again. Here's how it looks once selected:



```
Please Enter an Option from above using one of the numbers: 5

Select the type of person you'd like to delete:

1. Faculty
2. Student
3. Office Associate
4. Advisor
5. IT Professional
6. Doctor
7. Nurse
0. Go back to the main menu

Please Enter an Option from above using one of the numbers:
```

And here's what it looks like if the person you are trying to delete isn't found:



```
Please Enter an Option from above using one of the numbers: 2

Please Enter the blazerId of the member you'd like to delete:
(or you can Enter '0' to go back to the person type selection)

Please Enter an Option: jdlewis2

No Member found with that role and blazerId!

What would you like to do?

1. Try another BlazerID for this type
2. Go back to person type selection
3. Return to main menu

Please Enter an Option:
```

References:

- Lecture Slides

-