CSE Department – Faculty of Engineering - MSA
Spring 2025
GSE122 GSE122i COM265 PROGRAMMING 2
Course Project
Course Instructor:Dr. Ahmed El Anany

Student Name	**Farah Mohamed Ahmed Habib**	**Student ID**	**243897**
Student Name	**Sarah Mohamed Ali**	**Student ID**	**248097**
Student Name	**Toni Wagdi Wadie**	**Student ID**	**244743**
Student Name	**Ahmed Hisham Mohsen**	**Student ID**	**242435**
Student Name	**Youssef Mohamed**	**Student ID**	**234833**
TA Name	**Eng. Gehad Ehab**	**Grade:**	**/**

School Management System

Table of Contents

1. Project Overview 3
2. Objectives 4
3. Roles and Responsibilities 5
4. Algorithm and external libraries 6
   - 4.1 Algorithmic Design 6
   - 4.2 External Libraries 7
5. GUI and Database Usage 8
   - 5.1 GUI Description 8
   - 5.2 Database Usage 10
6. Code explaining 12
   - 6.1 CustomDate 12
   - 6.2 DatabaseHandler 14
   - 6.3 SchoolSystemGSE122i 18
   - 6.4 SchoolGUI 19
   - 6.5 Person 21
   - 6.6 Teacher 22
   - 6.7 TeacherSectionGUI 25
   - 6.8 Student 30
   - 6.9 StudentSectionGUI 32
7. Output and results 36
8. GitHub 43
9. References 45

1. Project Overview

The project, herein referred to as **EduManager**, represents a sophisticated school management system meticulously designed to facilitate the administration of student and teacher records. Developed using the Java programming language, the system integrates a graphical user interface (GUI) constructed with the Swing framework and employs a MySQL database for persistent data storage. The architecture leverages advanced object-oriented programming (OOP) principles, Java Database Connectivity (JDBC) for seamless database interaction, and event-driven programming methodologies to ensure responsive user interactions. The database schema, as evidenced by phpMyAdmin screenshots, comprises two primary tables: student (with columns name, studentClass, rollNo, feePaid, and date) and teacher (with columns name, position, salary, and date), which have directly informed the system's design and functionality.

Technological Framework:

- **Programming Language:** Java (version 8 or higher, ensuring compatibility with Swing libraries).
- **GUI Framework:** Java Swing, offering a robust set of components such as `JFrame`, `JTextField`, and `JButton` for user interface development.
- **Database System:** MySQL, administered through phpMyAdmin for schema creation and data management.
- **Database Connectivity:** MySQL JDBC Driver (`com.mysql.jdbc.Driver`), enabling Java-MySQL integration.
- **Development Environment:** Likely IntelliJ IDEA or Eclipse, utilized for coding, debugging, and project management.
- **Version Control:** Git, optionally integrated with GitHub for collaborative development and source code management.

Core Technical Concepts:

- **Object-Oriented Programming:** The system employs encapsulation through private fields with public access methods, inheritance via class hierarchies (e.g., Teacher extending Person), and abstraction through the abstract Person class.
- **Event-Driven Programming:** User interactions, such as button clicks, are managed via ActionListener implementations, ensuring responsive GUI behavior.
- **JDBC Implementation:** Facilitates the execution of SQL queries and manages database transactions with efficiency and security.
- **Date Validation Logic:** A custom-designed algorithm validates date inputs, accounting for month-specific day limits and leap year considerations.
- **Exception Handling:** Comprehensive error management is achieved through try-catch blocks, addressing both user input errors and database connectivity issues.

2. Objectives

The EduManager project was undertaken with the following scholarly and practical objectives, aimed at demonstrating proficiency in software development and system design:

1. **Development of a Java-Based Application:** Engineer a cross-platform desktop application using Java's Java Virtual Machine (JVM) to manage educational records, ensuring portability across diverse operating systems.
2. **Design of an Intuitive User Interface:** Create a user-friendly GUI to support the creation, modification, retrieval, listing, and deletion of student and teacher records, providing immediate feedback through dialog boxes to enhance user experience.
3. **Integration with a MySQL Database:** Utilize JDBC to establish a secure connection to the type school_dp database, ensuring persistent storage of data within the student and teacher tables as defined in the phpMyAdmin schema.
4. **Application of Object-Oriented Principles:** Implement inheritance to encapsulate shared behaviors (e.g., Person as a superclass) and abstraction to enforce consistent method implementations across derived classes.
5. **Ensuring Robust Input Validation:** Develop comprehensive validation mechanisms for date inputs and form fields to maintain data integrity and prevent erroneous data entry, thereby enhancing system reliability.
6. **Provision of Comprehensive CRUD Operations:** Enable full-spectrum record management (Create, Read, Update, Delete) through secure database interactions, utilizing prepared statements to mitigate SQL injection risks.
7. **Demonstration of Collaborative Development:** Distribute tasks among five team members to foster coordinated development, integration, and testing of system components, reflecting real-world software engineering practices.

3. Roles and Responsibilities

The development of EduManager was a collaborative endeavor involving five team members, each assigned specific responsibilities to ensure comprehensive coverage of the project's requirements:

- **Farah Mohamed Ahmed Habib:**
  - **CustomDate Class:** Architected a date validation mechanism capable of processing both numeric (e.g., "1" for January) and textual (e.g., "January") month inputs, incorporating leap year calculations to ensure date integrity.
  - **DatabaseHandler Class:** Developed a JDBC-based interface to the MySQL database, implementing methods for all CRUD operations on the student and teacher tables, ensuring secure and efficient data transactions.
  - **phpMyAdmin Configuration:** Configured the type school_dp database, defined the student table (with rollNo as the primary key) and teacher table (with name as the primary key), and managed schema specifications, including column attributes and constraints.
- **Youssef Zeina:**
  - **TeacherSectionGUI Class:** Designed the graphical interface for teacher record management using a GridLayout(11, 2) configuration, incorporating input fields for day, month, year, name, position, and salary, and implementing button-driven functionalities for CRUD operations.
  - **Teacher Logic:** Formulated the business logic for teacher record operations, integrating database interactions and input validation to ensure data consistency and user feedback.
- **Toni Wagdi Wadie:**
  - **SchoolGUI Class:** Developed the primary navigation interface using FlowLayout, integrating buttons for accessing student and teacher sections, as well as an exit functionality to terminate the application.
  - **Teacher Class Contribution:** Collaborated on the implementation of teacher record management methods, ensuring alignment with the teacher table schema and supporting CRUD operations.
- **Ahmed hisham mohsen:**
  - **StudentSectionGUI Class:** Engineered the GUI for student record management with a GridLayout(12, 2) layout, including fields for day, month, year, name, class, roll number, and fee paid, alongside buttons for CRUD operations.
  - **Student Logic:** Developed student-specific features, including fee calculation logic (due amount, fine, and advance) and database synchronization for persistent storage.
- **Sarah Mohamed Ali:**
  - **Person Class:** Defined an abstract superclass establishing a contract for record management methods, ensuring consistent implementation across subclasses.
  - **Student Class:** Implemented student record logic, incorporating fee computation algorithms to calculate due amounts, fines, and advances based on payment status.
  - **SchoolSystemGSE122i Class:** Created the application's entry point, featuring a console-based date validation mechanism to initialize the system before launching the GUI.

4. Algorithm and external libraries

4.1 Algorithmic Design

1. **Date Validation (**CustomDate**):**
   - **Procedural Steps:**
     1. Convert the month parameter into a numeric value (1–12) using the getMonthNumber() method, which supports both textual and numeric inputs.
     2. Validate the month range; if the value is less than 1 or greater than 12, return false to indicate an invalid month.
     3. Determine leap year status using the formula `(year % 4 == 0 && year % 100 != 0) || (year % 400 == 0)` (e.g., 2024 is a leap year, while 2025 is not).
     4. Calculate the maximum allowable days for the given month: 29 for February in a leap year, 28 otherwise, 30 for April, June, September, and November, and 31 for all other months.
     5. Compare the day value against the computed maxDays; return `true` if the day is within the permissible range, `false` otherwise.
     6. Finalize the validation result based on the above checks.
   - **Computational Complexity:** O(1), as all operations are performed in constant time.
   - **Illustrative Example:** An input of "29 February 2024" passes validation (leap year); "29 February 2025" fails (not a leap year).

2. **Database Operations (**DatabaseHandler**):**
   - **Procedural Steps:**
     1. Establish a JDBC connection using `DriverManager.getConnection()` to connect to the `type school_dp` database.
     2. Construct prepared SQL statements with PreparedStatement objects to prevent SQL injection vulnerabilities and enhance query performance.
     3. Bind parameters to the prepared statements (e.g., ? placeholders for name, rollNo) and execute the query using methods such as stmt.executeUpdate() for INSERT, UPDATE, and DELETE operations, or stmt.executeQuery() for SELECT operations.
     4. Process the ResultSet object for SELECT queries, iterating over retrieved rows to extract data.
     5. Utilize try-with-resources blocks to ensure proper closure of database resources, mitigating memory leaks and connection exhaustion.
   - **Computational Complexity:** O(n) for SELECT operations, where n denotes the number of rows retrieved; O(1) for INSERT, UPDATE, and DELETE operations.
   - **Illustrative Example:** The method saveStudent("Jane", "10A", 101, 800, "10 Feb 2025") executes an INSERT query to add a student record to the database.

3. **GUI Event Handling:**
   - **Procedural Steps:**
     1. Register ActionListener instances on GUI buttons to handle user interactions (e.g., saveButton.addActionListener(e -> saveTeacher())).
     2. Parse and validate user inputs from text fields (e.g., Integer.parseInt(dayField.getText())), employing exception handling to manage invalid entries.
     3. Invoke the corresponding method associated with the user action (e.g., saveTeacher() for the Save button).
     4. Display the results or error messages to the user via JOptionPane.showMessageDialog(), providing immediate feedback.
   - **Computational Complexity:** O(1) per event, with additional overhead for input validation and processing.

4.2 External Libraries

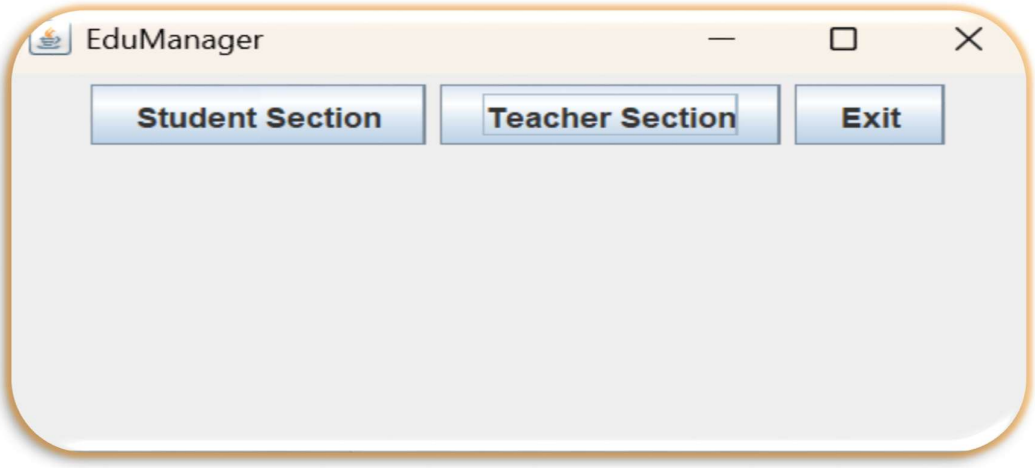
- **Java Util Package:** Provides essential utility classes, including Scanner and StringBuilder, supporting console input processing and efficient string manipulation for enhanced application functionality.
- **Java Swing:** Provides an extensive suite of GUI components, including `JFrame`, `JTextField`, `JButton`, and layout managers such as GridLayout and FlowLayout, facilitating the construction of a responsive user interface.
- **MySQL JDBC Driver:** Enables Java applications to interact with MySQL databases by providing the necessary driver (com.mysql.jdbc.Driver), loaded dynamically using Class.forName().
- **Java SQL Package:** Offers essential classes such as Connection, PreparedStatement, and ResultSet, ensuring type-safe database operations and efficient resource management through JDBC.

5. GUI and Database Usage

5.1 GUI Description

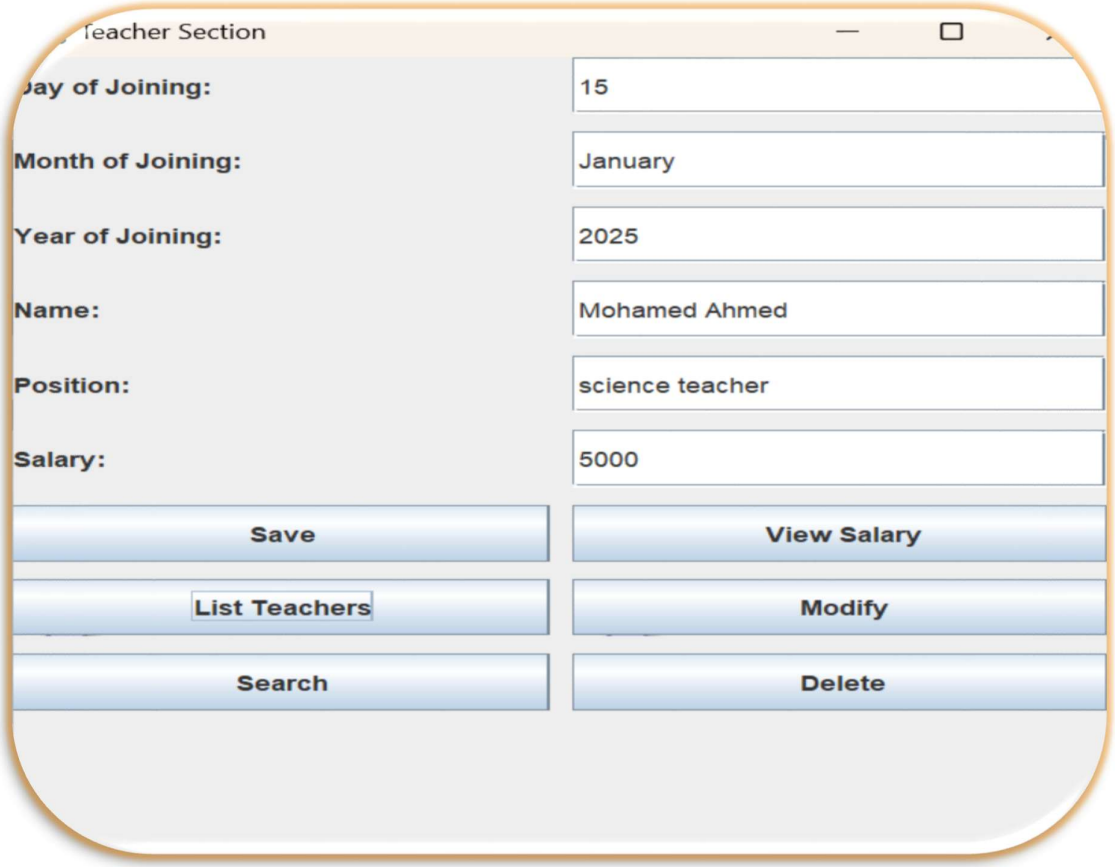
The system incorporates three primary GUI components, each designed with precision to enhance usability:

1. **Main Menu (SchoolGUI):**
   - **Layout Configuration:** Utilizes FlowLayout to arrange three `JButton` elements horizontally, ensuring a streamlined navigation experience.
   - **Component Composition:** Includes buttons labeled "Student Section," "Teacher Section," and "Exit" for accessing respective modules or terminating the application.
   - **Behavioral Response:** The "Student Section" button instantiates StudentSectionGUI, the "Teacher Section" button instantiates TeacherSectionGUI, and the "Exit" button invokes `System.exit(0)` to terminate the application.
   - **Dimensional Specifications:** The window is configured to 400x200 pixels, centrally positioned on the user's screen for optimal visibility.
   - **Illustrative Example:** Selecting the "Teacher Section" button launches the teacher management interface.



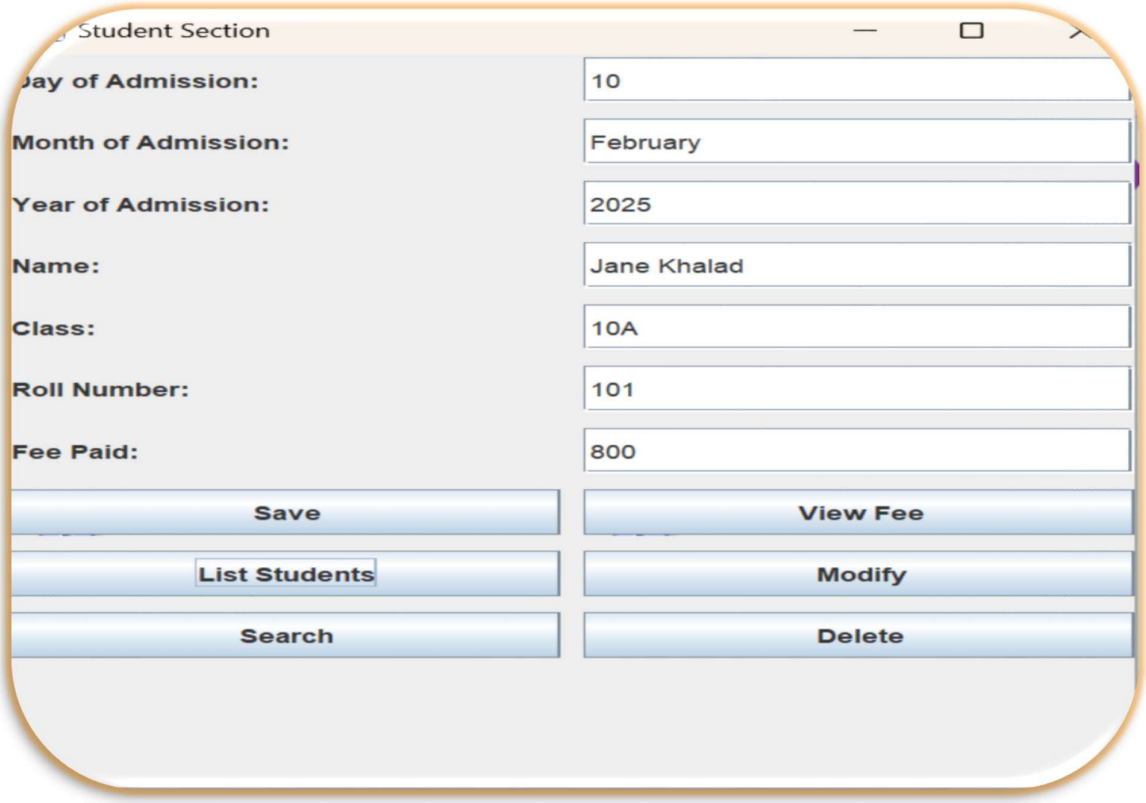
2. **Teacher Section (TeacherSectionGUI):**
   - **Layout Configuration:** Employs GridLayout(11, 2) with 10-pixel gaps between rows and columns, ensuring precise alignment of labels and input fields.
   - **Component Composition:**
     - Paired `JLabel` and `JTextField` elements for capturing day, month, year, name, position, and salary inputs.
     - Six `JButton` elements: "Save," "View Salary," "List Teachers," "Modify," "Search," and "Delete," each bound to specific action listeners.
   - **Behavioral Response:** Each button triggers a dedicated method (e.g., saveTeacher() parses inputs, validates them, and saves the record to the database).

- **Dimensional Specifications:** The window dimensions are set to 500x500 pixels, providing ample space for all components.
- **Illustrative Layout Example:**



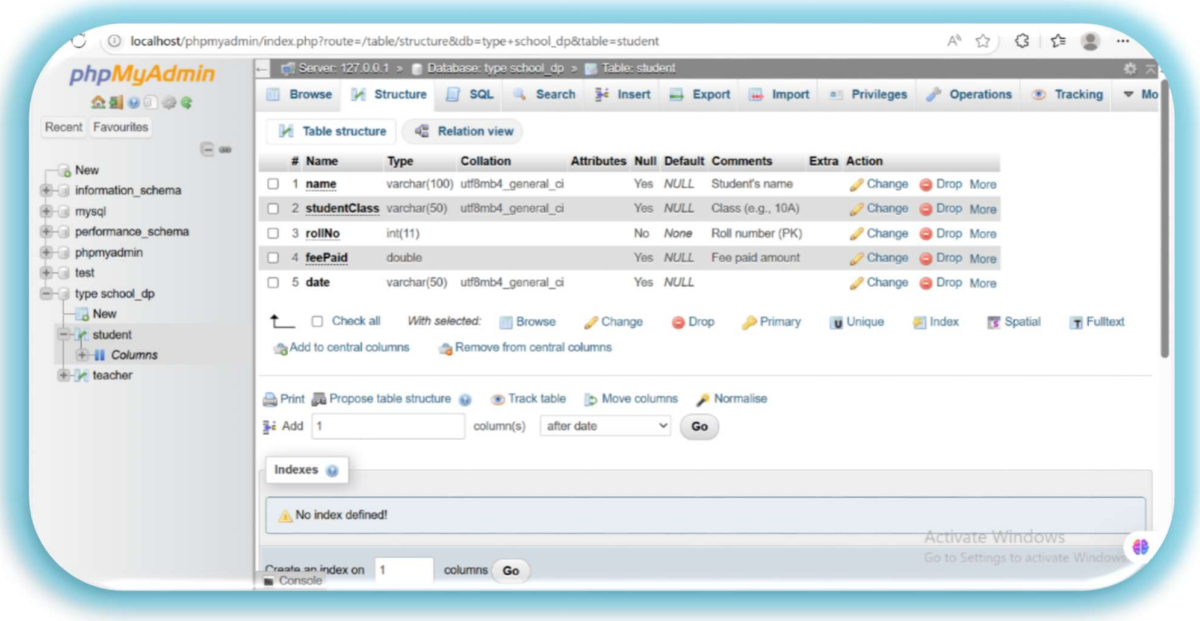
3. **Student Section (**StudentSectionGUI**):**

- **Layout Configuration:** Implements GridLayout(12, 2) with similar 10-pixel gaps, ensuring consistent alignment with the teacher section interface.
- **Component Composition:** Includes fields for day, month, year, name, class, roll number, and fee paid, alongside six buttons: "Save," "View Fee," "List Students," "Modify," "Search," and "Delete."
- **Behavioral Response:** Functions analogously to the teacher section GUI, with additional logic for handling student-specific fields such as roll number and fee calculations.
- **Dimensional Specifications:** The window is set to 500x500 pixels, matching the teacher section for uniformity.
- **Illustrative Layout Example:**

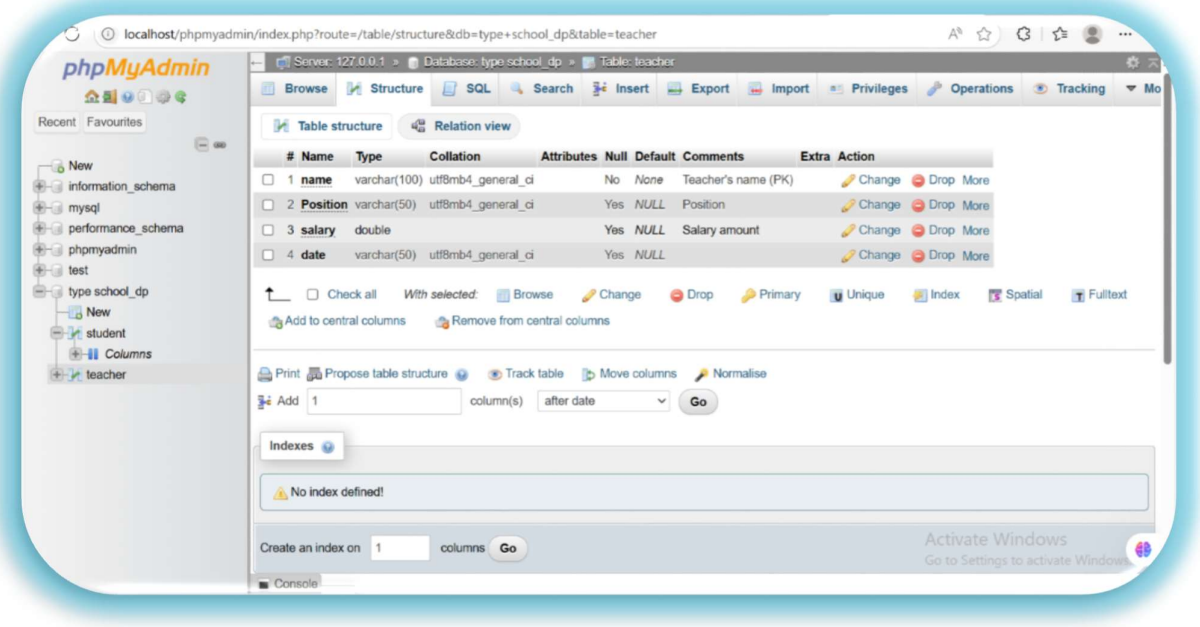


5.2 Database Usage

- **Database Designation:** The system utilizes the type school_dp database, hosted on a local MySQL server.
- **Table Structures:**
    1. **Student Table:**
        - **Column Definitions:** name (VARCHAR(100), nullable), studentClass (VARCHAR(50), nullable), rollNo (INT(11), non-nullable, primary key), feePaid (DOUBLE, nullable), date (VARCHAR(50), nullable).
        - **Functional Purpose:** Serves as a persistent storage mechanism for student records, with rollNo ensuring uniqueness.
        - **Illustrative Example:**



2. **Teacher Table:**
   - **Column Definitions:** name (VARCHAR(100), non-nullable, primary key), position (VARCHAR(50), nullable), salary (DOUBLE, nullable), date (VARCHAR(50), nullable).
   - **Functional Purpose:** Stores teacher records, with name as the unique identifier.
   - **Illustrative Example:**



- **Operational Procedures:**
  - **Insert Operation:** Executes prepared statements to populate table rows (e.g., `INSERT INTO student VALUES (?, ?, ?, ?, ?)`), ensuring secure data insertion.
  - **Select Operation:** Retrieves records using queries such as `SELECT * FROM teacher WHERE name = ?,` supporting both full-table retrieval and specific searches.
  - **Update Operation:** Modifies existing records with statements like `UPDATE student SET feePaid = ? WHERE rollNo = ?`, preserving data integrity.
  - **Delete Operation:** Removes records based on key constraints (e.g., `DELETE FROM teacher WHERE name = ?`).

6. Code explaining

This section provides a meticulous, step-by-step analysis of each class within the EduManager system, contextualized with the database schema and operational logic.

6.1 CustomDate

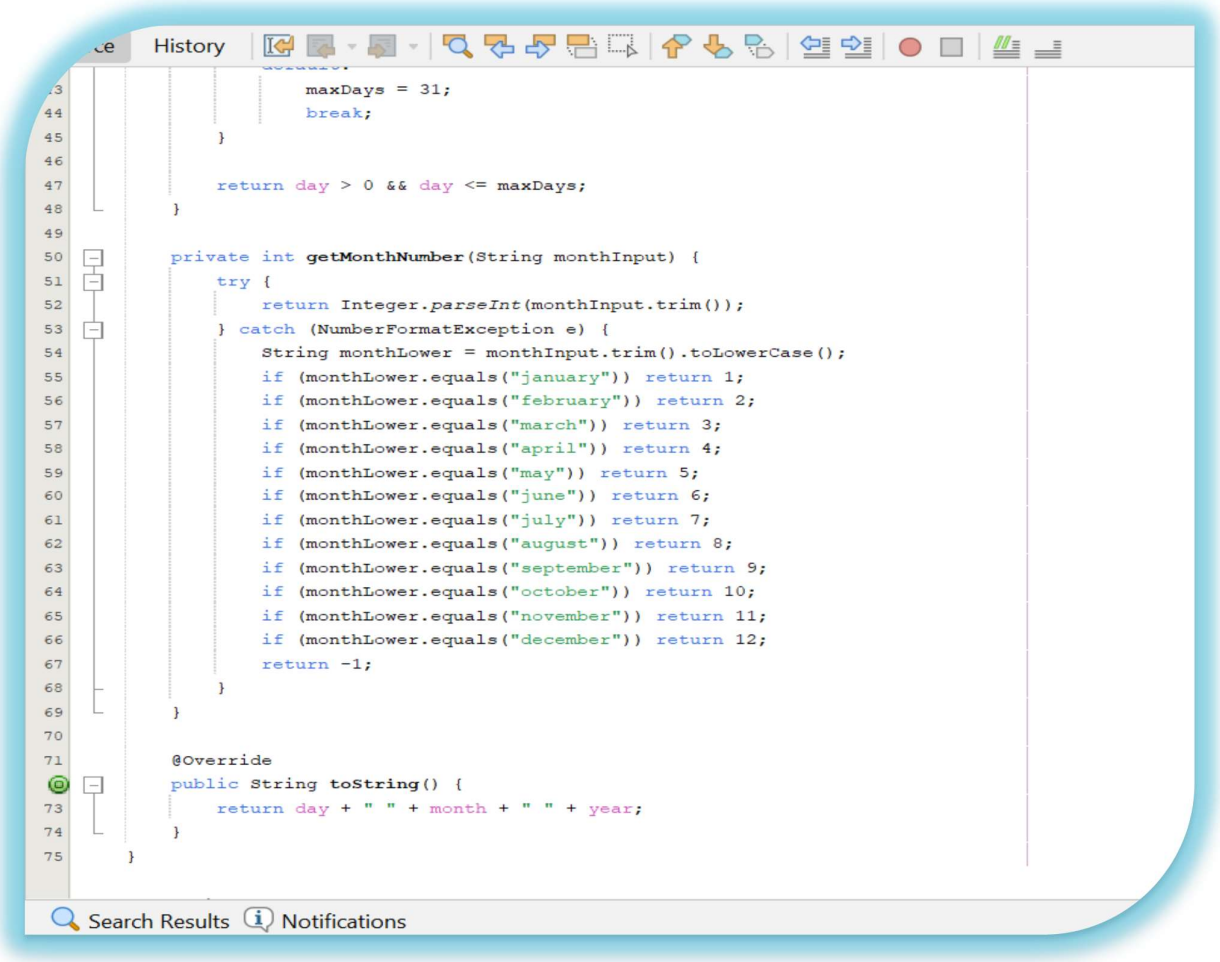
- **Purpose:** Validates and formats date inputs to ensure their correctness for record storage.
- **Detailed Procedural Analysis:**
  1. **Constructor (`CustomDate(int day, String month, int year)`):**
     - Initializes the instance variables day, month, and year with the provided values.
     - **Illustrative Example:** new CustomDate(15, "January", 2025) assigns day = 15, month = "January", and year = 2025.
  2. `chkDate()` **Method:**
     - Invokes getMonthNumber() to convert the month string into a numeric value ranging from 1 to 12.
     - Validates the month range; if the numeric month is less than 1 or greater than 12, the method returns `false`.
     - Assesses leap year status using the formula `(year % 4 == 0 && year % 100 != 0) || (year % 400 == 0)`, determining whether February has 29 or 28 days.
     - Determines the maximum allowable days (maxDays) for the given month: 29 for February in a leap year, 28 otherwise, 30 for April, June, September, and November, and 31 for January, March, May, July, August, October, and December.
     - Compares the day value against `maxDays`; returns true if day is within the valid range (i.e., `day > 0 && day <= maxDays`), `false` otherwise.
     - **Illustrative Example:** An input of "29 February 2024" passes validation (2024 is a leap year, allowing 29 days); "29 February 2025" fails (2025 is not a leap year, limiting February to 28 days).
  3. `getMonthNumber(String monthInput)` **Method:**
     - Attempts to parse `monthInput.trim()` as an integer using `Integer.parseInt()` to handle numeric month inputs (e.g., "1" for January).

- If parsing fails (i.e., a `NumberFormatException` is thrown), converts `monthInput` to lowercase and matches it against predefined month names (e.g., "january" maps to 1, "february" to 2, etc.).
- Returns -1 for unrecognized inputs, indicating an invalid month.
- **Illustrative Example:** An input of "jan" returns -1 (invalid); "january" returns 1; "1" also returns 1.

4. `toString()` **Method:**
   - Returns a string representation of the date in the format "day month year" (e.g., "15 January 2025"), which is used for both display purposes and database storage.
   - **Illustrative Example:** For day = 15, month = "January", year = 2025, the method returns "15 January 2025".

- **Database Relevance:** The `toString()` output aligns with the date column (VARCHAR(50)) in both the student and teacher tables, ensuring compatibility with the database schema.

```
History
public class CustomDate {

    private int day;
    private String month;
    private int year;

    public CustomDate(int day, String month, int year) {
        this.day = day;
        this.month = month;
        this.year = year;
    }

    public boolean chkDate() {
        int numericMonth = getMonthNumber(month);

        if (numericMonth < 1 || numericMonth > 12) return false;

        int maxDays;

        boolean isLeapYear = (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);

        switch (numericMonth) {
            case 2:
                maxDays = isLeapYear ? 29 : 28;
                break;
            case 4:
            case 6:
            case 9:
            case 11:
                maxDays = 30;
                break;
            default:
                maxDays = 31;
                break;
```



6.2 DatabaseHandler

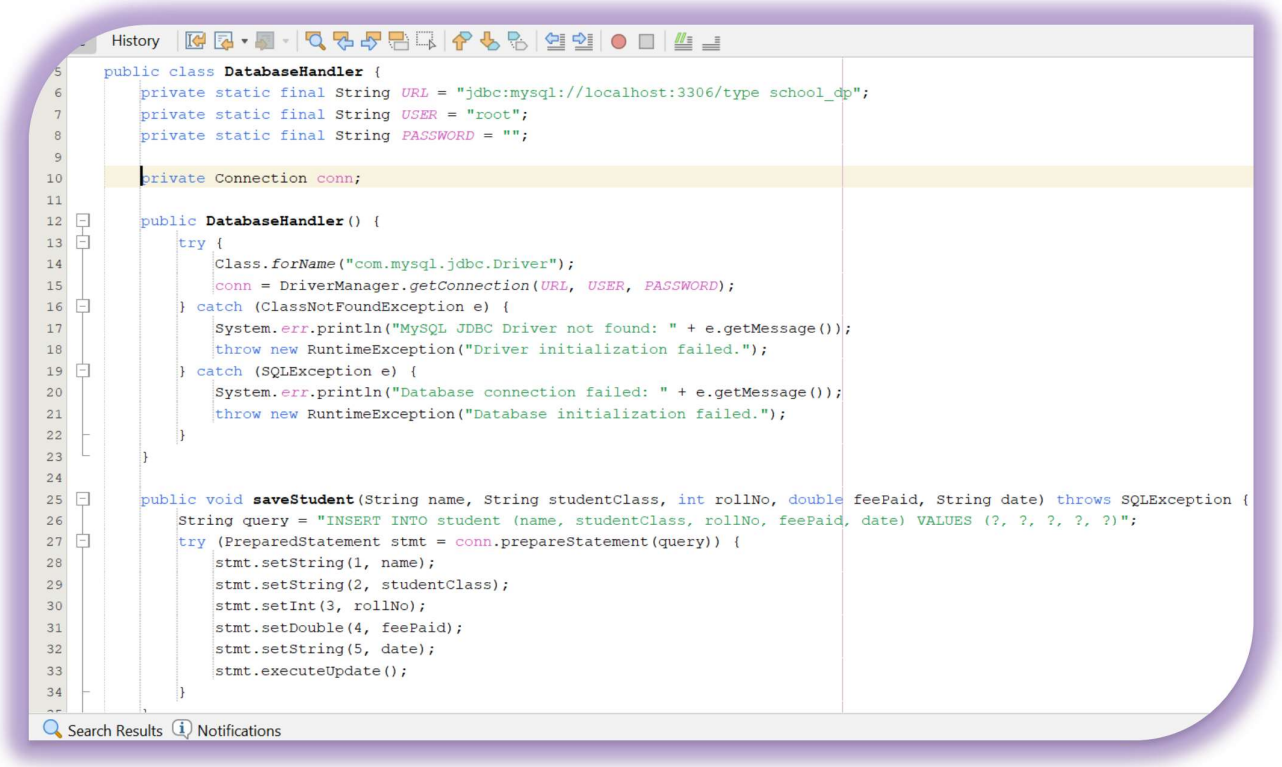
- **Purpose:** Serves as the central interface for all database interactions, managing connectivity and CRUD operations.
- **Detailed Procedural Analysis:**
  1. **Constructor:**
     - Dynamically loads the MySQL JDBC driver using `Class.forName("com.mysql.jdbc.Driver")` to enable Java-MySQL communication.
     - Establishes a connection to the database using `DriverManager.getConnection ("jdbc:mysql://localhost:3306/type school_dp", "root", "")`, specifying the database URL, username (root), and password (empty).
     - Handles potential exceptions: `ClassNotFoundException` (if the driver is unavailable) and `SQLException` (if the connection fails), rethrowing them as `RuntimeException` with descriptive messages.

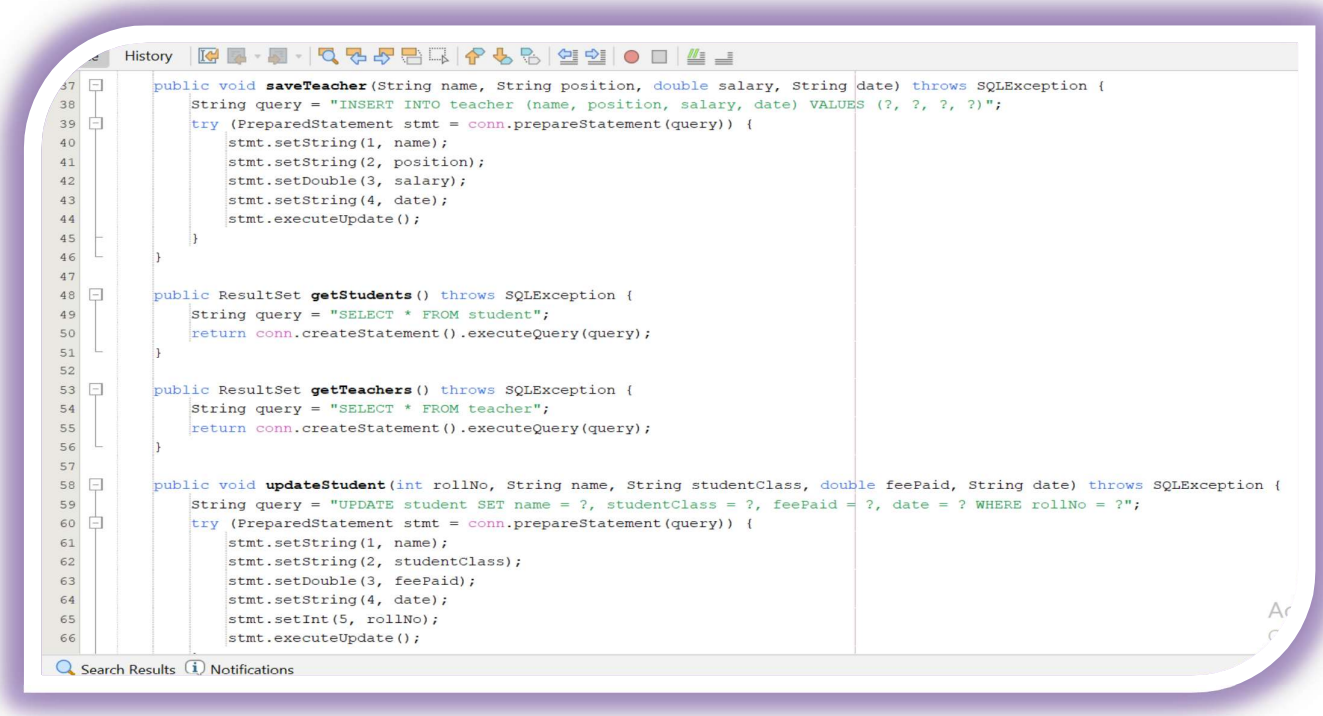
- **Illustrative Example:** If MySQL is running on localhost with the specified credentials, the connection is successfully established; otherwise, an exception is thrown.

2. `saveStudent(String name, String studentClass, int rollNo, double feePaid, String date)` **Method:**
   - Prepares an SQL INSERT statement: INSERT INTO student (name, studentClass, rollNo, feePaid, date) VALUES (?, ?, ?, ?, ?).
   - Binds the parameters in sequence: name to the first placeholder, studentClass to the second, rollNo to the third, feePaid to the fourth, and date to the fifth.
   - Executes the statement using `stmt.executeUpdate()` within a try-with-resources block to ensure the PreparedStatement is closed automatically.
   - **Illustrative Example:** Calling saveStudent("Jane", "10A", 101, 800, "10 Feb 2025") inserts a new row into the student table with the specified values.
3. `saveTeacher(String name, String position, double salary, String date)` **Method:**
   - Prepares an SQL INSERT statement: INSERT INTO teacher (name, position, salary, date) VALUES (?, ?, ?, ?).
   - Binds the parameters: name, position, salary, and date, in that order.
   - Executes the statement using `stmt.executeUpdate()` within a try-with-resources block.
   - **Illustrative Example:** Calling saveTeacher("John", "Professor", 5000, "15 Jan 2025") inserts a new row into the teacher table.
4. `getStudents()` **and** `getTeachers()` **Methods:**
   - Execute SELECT queries: SELECT * FROM student for `getStudents()` and SELECT * FROM teacher for `getTeachers()`.
   - Return a ResultSet object containing all rows from the respective table, which can be iterated to retrieve data.
   - **Illustrative Example:** `getStudents()` might return a ResultSet containing the row ("Jane Smith", "10A", 101, 800, "10 February 2025").
5. `updateStudent(int rollNo, String name, String studentClass, double feePaid, String date)` **Method:**
   - Prepares an SQL UPDATE statement: UPDATE student SET name = ?, studentClass = ?, feePaid = ?, date = ? WHERE rollNo = ?.
   - Binds the parameters: name, studentClass, feePaid, date, and rollNo, using rollNo as the primary key to identify the row to update.
   - Executes the statement with `stmt.executeUpdate()`.
   - **Illustrative Example:** updateStudent(101, "Jane Smith", "10A", 1000, "10 Feb 2025") updates the feePaid value to 1000 for the student with rollNo 101.
6. `updateTeacher(String name, String position, double salary, String date)` **Method:**
   - Prepares an UPDATE statement: UPDATE teacher SET position = ?, salary = ?, date = ? WHERE name = ?.
   - Binds the parameters: position, salary, date, and name, using name as the primary key.
   - Executes the statement with `stmt.executeUpdate()`.
   - **Illustrative Example:** updateTeacher("John Doe", "Senior Professor", 5500, "15 Jan 2025") updates the salary to 5500 for the teacher named "John Doe".

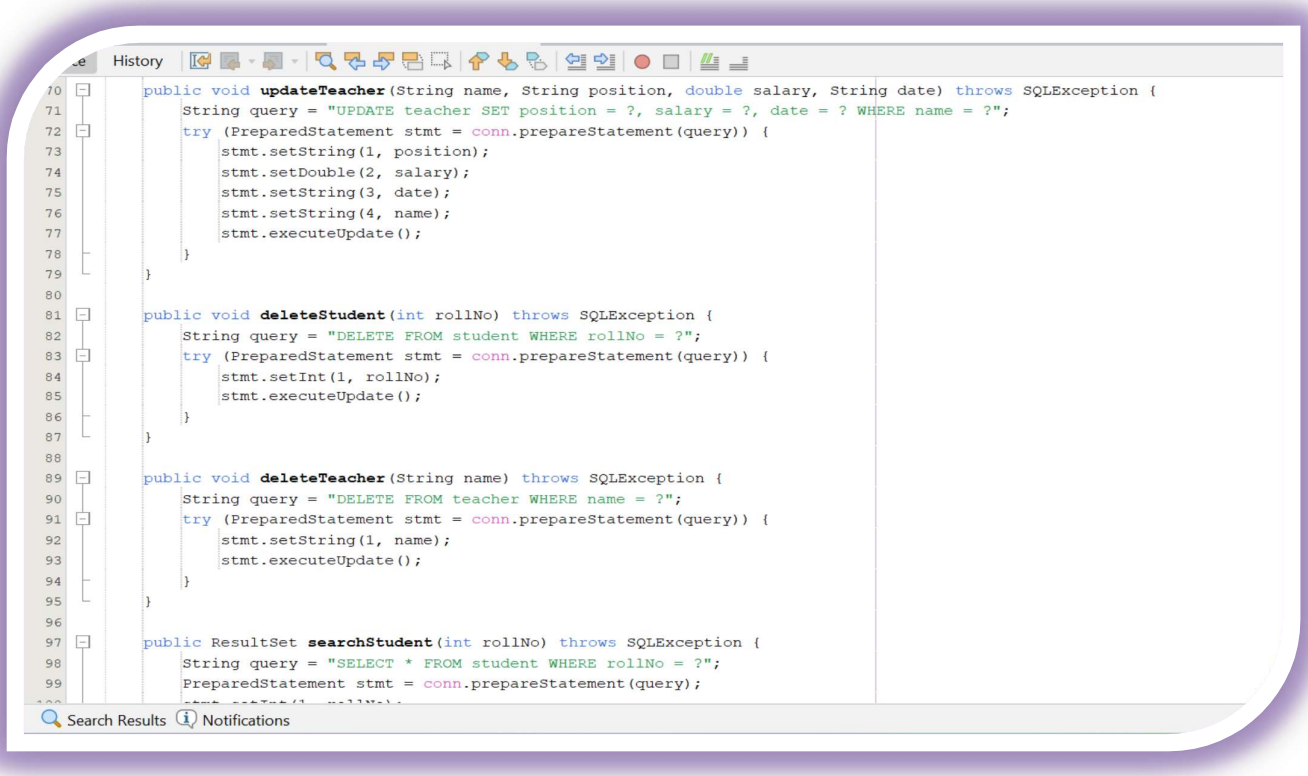
7. `deleteStudent(int rollNo)` **and** `deleteTeacher(String name)` **Methods:**
   - Prepare SQL DELETE statements: DELETE FROM student WHERE rollNo = ? for `deleteStudent()` and DELETE FROM teacher WHERE name = ? for `deleteTeacher()`.
   - Bind the respective key (rollNo or name) and execute the statement with `stmt.executeUpdate()`.
   - **Illustrative Example:** `deleteStudent(101)` removes the student record with rollNo 101 from the student table.
8. `searchStudent(int rollNo)` **and** `searchTeacher(String name)` **Methods:**
   - Prepare SQL SELECT statements: SELECT * FROM student WHERE rollNo = ? for `searchStudent()` and SELECT * FROM teacher WHERE name = ? for `searchTeacher()`.
   - Bind the respective key and execute with `stmt.executeQuery()`, returning a ResultSet containing the matching row (if any).
   - **Illustrative Example:** `searchStudent(101)` returns a ResultSet with the row for "Jane Smith" if present.

- **Database Relevance:** The methods are designed to align with the student table (where rollNo is the primary key) and the teacher table (where name is the primary key), ensuring consistency with the phpMyAdmin schema.

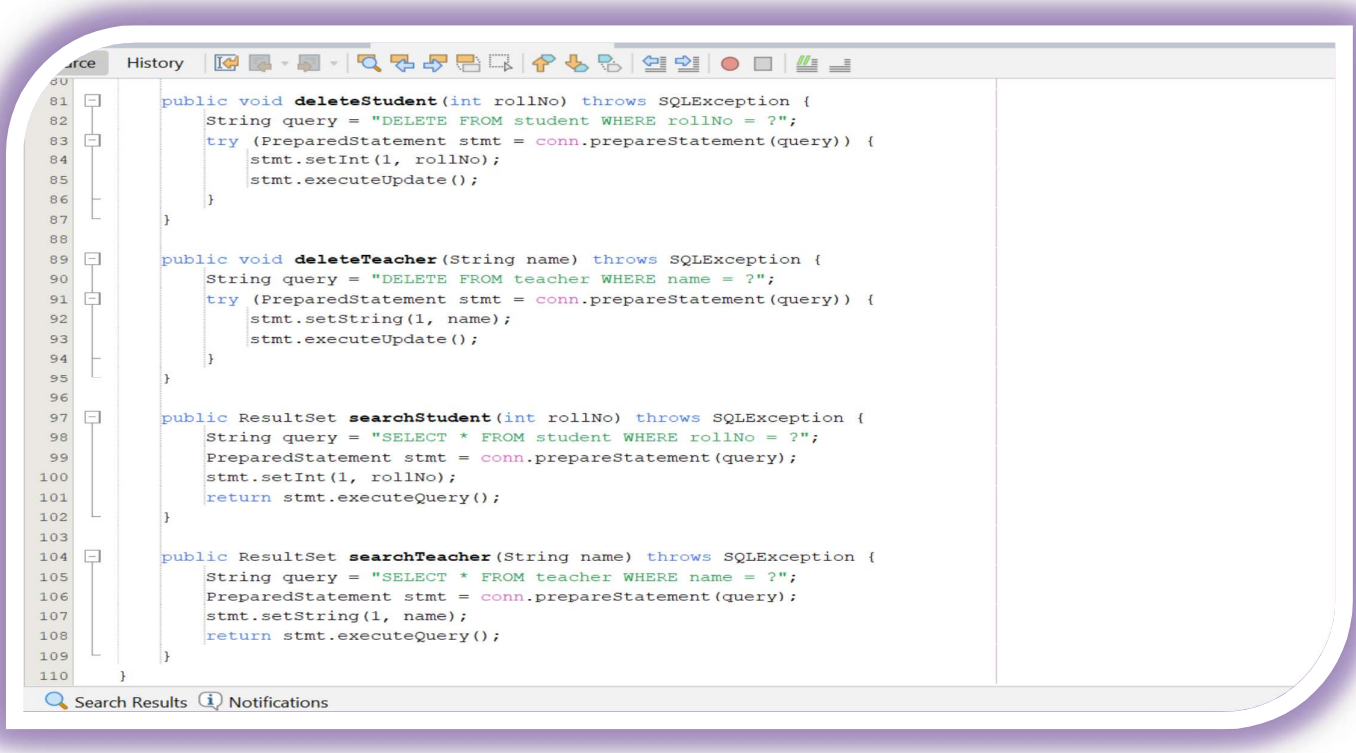




```
History
    public void saveTeacher(String name, String position, double salary, String date) throws SQLException {
        String query = "INSERT INTO teacher (name, position, salary, date) VALUES (?, ?, ?, ?)";
        try (PreparedStatement stmt = conn.prepareStatement(query)) {
            stmt.setString(1, name);
            stmt.setString(2, position);
            stmt.setDouble(3, salary);
            stmt.setString(4, date);
            stmt.executeUpdate();
        }
    }

    public ResultSet getStudents() throws SQLException {
        String query = "SELECT * FROM student";
        return conn.createStatement().executeQuery(query);
    }

    public ResultSet getTeachers() throws SQLException {
        String query = "SELECT * FROM teacher";
        return conn.createStatement().executeQuery(query);
    }

    public void updateStudent(int rollNo, String name, String studentClass, double feePaid, String date) throws SQLException {
        String query = "UPDATE student SET name = ?, studentClass = ?, feePaid = ?, date = ? WHERE rollNo = ?";
        try (PreparedStatement stmt = conn.prepareStatement(query)) {
            stmt.setString(1, name);
            stmt.setString(2, studentClass);
            stmt.setDouble(3, feePaid);
            stmt.setString(4, date);
            stmt.setInt(5, rollNo);
            stmt.executeUpdate();
Search Results  Notifications
```



```
History
    public void updateTeacher(String name, String position, double salary, String date) throws SQLException {
        String query = "UPDATE teacher SET position = ?, salary = ?, date = ? WHERE name = ?";
        try (PreparedStatement stmt = conn.prepareStatement(query)) {
            stmt.setString(1, position);
            stmt.setDouble(2, salary);
            stmt.setString(3, date);
            stmt.setString(4, name);
            stmt.executeUpdate();
        }
    }

    public void deleteStudent(int rollNo) throws SQLException {
        String query = "DELETE FROM student WHERE rollNo = ?";
        try (PreparedStatement stmt = conn.prepareStatement(query)) {
            stmt.setInt(1, rollNo);
            stmt.executeUpdate();
        }
    }

    public void deleteTeacher(String name) throws SQLException {
        String query = "DELETE FROM teacher WHERE name = ?";
        try (PreparedStatement stmt = conn.prepareStatement(query)) {
            stmt.setString(1, name);
            stmt.executeUpdate();
        }
    }

    public ResultSet searchStudent(int rollNo) throws SQLException {
        String query = "SELECT * FROM student WHERE rollNo = ?";
        PreparedStatement stmt = conn.prepareStatement(query);
Search Results  Notifications
```



6.3 SchoolSystemGSE122i

- **Purpose:** Acts as the entry point of the application, initiating the system with a console-based date validation step before launching the GUI.
- **Detailed Procedural Analysis:**
  1. **Main Method (`public static void main(String[] args)`):**
     - Serves as the application's entry point, invoking the `start()` method to begin execution.
     - **Illustrative Example:** The JVM calls `main()` upon program launch.
  2. `start()` **Method:**
     - Instantiates a Scanner object to read input from System.in, enabling console-based interaction.
     - Enters a loop to prompt the user for day, month, and year inputs, reading them using `s.nextInt()` for day and year, and `s.next()` for month.
     - Constructs a CustomDate object with the inputs and validates it using `chkDate()`. If validation fails, displays an error message ("Invalid date. Try again.") and continues the loop.
     - Upon successful validation, prints a confirmation message ("Date validated: day/month/year") and proceeds to launch the GUI.

- Uses `SwingUtilities.invokeLater(() -> new SchoolGUI())` to ensure the GUI is created and displayed on the Event Dispatch Thread (EDT), adhering to Swing's thread-safety requirements.
- **Illustrative Example:** Inputs of "15", "January", "2025" pass validation, leading to the launch of the SchoolGUI interface.

```
public static void main(String[] args) {
    start();
}

public static void start() {
    Scanner s = new Scanner(System.in);
    int day;
    String month;
    int year;

    while (true) {
        System.out.print("Enter day: ");
        day = s.nextInt();
        System.out.print("Enter month: ");
        month = s.next();
        System.out.print("Enter year: ");
        year = s.nextInt();

        CustomDate date = new CustomDate(day, month, year);
        if (!date.chkDate()) {
            System.out.println("Invalid date. Try again.");
            continue;
        }
        break;
    }

    System.out.println("Date validated: " + day + "/" + month + "/" + year);
    SwingUtilities.invokeLater(() -> new SchoolGUI());
}
}
```

6.4 SchoolGUI

- **Purpose:** Provides the main navigation interface for the application, allowing users to access the student and teacher management sections or exit the system.
- **Detailed Procedural Analysis:**
  1. **Constructor (`SchoolGUI()`):**
     - Delegates the setup of the interface to the `displayMainMenu()` method.
     - **Illustrative Example:** Instantiating SchoolGUI triggers the creation of the main menu window.
  2. displayMainMenu() **Method:**
     - Configures the JFrame with the title "EduManager" and dimensions of 400x200 pixels, ensuring a compact and focused interface.
     - Sets the layout to FlowLayout, which arranges components horizontally in the order they are added.
     - Creates three JButton objects: "Student Section," "Teacher Section," and "Exit."
     - Binds ActionListener implementations to each button:

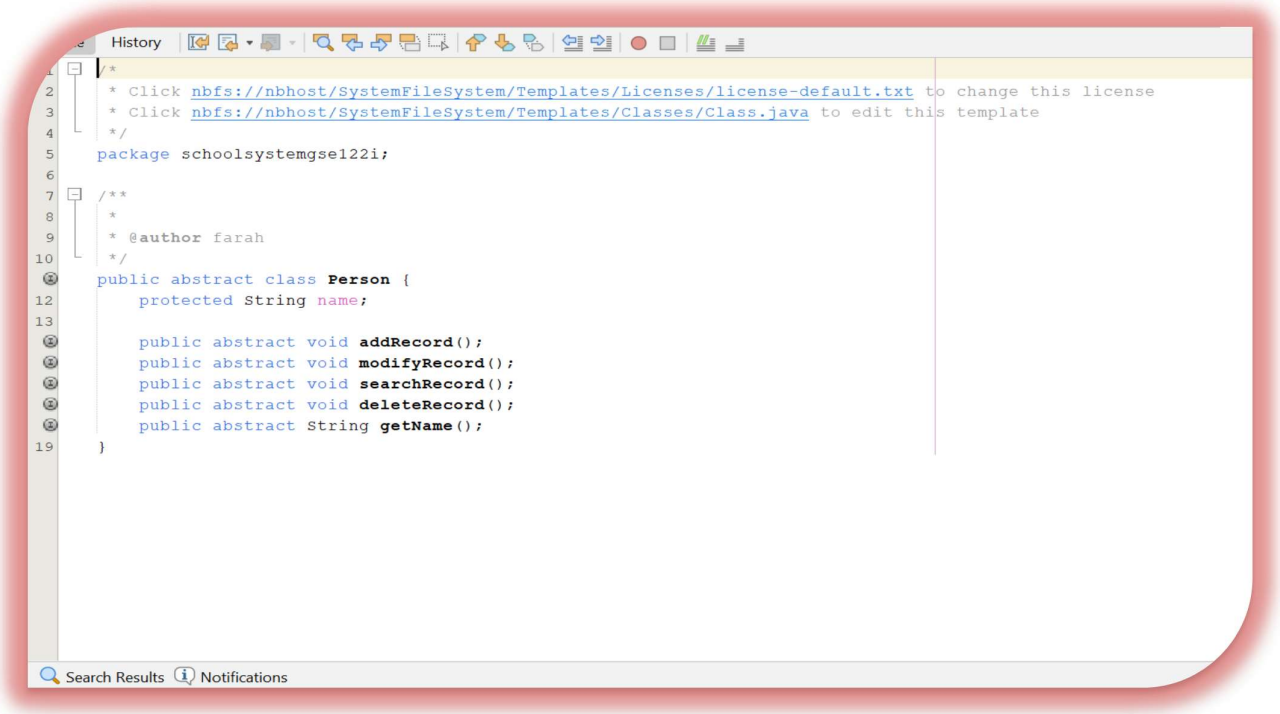
- "Student Section" button: `studentBtn.addActionListener(e -> new StudentSectionGUI().setVisible(true))`, opening the student management GUI.
- "Teacher Section" button: `teacherBtn.addActionListener(e -> new TeacherSectionGUI().setVisible(true))`, opening the teacher management GUI.
- "Exit" button: `exitBtn.addActionListener(e -> System.exit(0))`, terminating the JVM and closing the application.

- Adds the buttons to the JFrame using `add()` in the order specified, ensuring they appear sequentially from left to right.
- Sets the default close operation to EXIT_ON_CLOSE, ensuring the application terminates when the window is closed.
- Calls `setVisible(true)` to render the window on the screen.
- **Illustrative Example:** Clicking the "Exit" button immediately terminates the application, while clicking "Student Section" opens a new StudentSectionGUI window.

```
History
11  import javax.swing.*;
12  import java.awt.*;
13
14  public class SchoolGUI extends JFrame {
15      public SchoolGUI() {
            displayMainMenu();
17      }
18
19      public void displayMainMenu() {
20          setTitle("EduManager");
21          setSize(400, 200);
22          setLayout(new FlowLayout());
23
24          JButton studentBtn = new JButton("Student Section");
25          JButton teacherBtn = new JButton("Teacher Section");
26          JButton exitBtn = new JButton("Exit");
27
28          studentBtn.addActionListener(e -> new StudentSectionGUI().setVisible(true));
29          teacherBtn.addActionListener(e -> new TeacherSectionGUI().setVisible(true));
30          exitBtn.addActionListener(e -> System.exit(0));
31
32          add(studentBtn);
33          add(teacherBtn);
34          add(exitBtn);
35
36          setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
37          setVisible(true);
38      }
39  }
Search Results   Notifications
```

6.5 Person

- **Purpose:** Defines an abstract superclass that establishes a common interface for record management operations, serving as the foundation for Student and Teacher classes.
- **Detailed Procedural Analysis:**
  1. **Field Declaration:**
     - Declares a protected String name field, accessible to subclasses but encapsulated from external access.
     - **Illustrative Example:** Subclasses like Teacher can access and modify the name field directly.
  2. **Abstract Method Definitions:**
     - Declares five abstract methods: `addRecord()`, `modifyRecord()`, `searchRecord()`, `deleteRecord()`, and `getName()`, requiring implementation by any concrete subclass.
     - Ensures that all subclasses adhere to a consistent interface for record management operations.
     - **Illustrative Example:** The Teacher class must provide concrete implementations for all five methods.



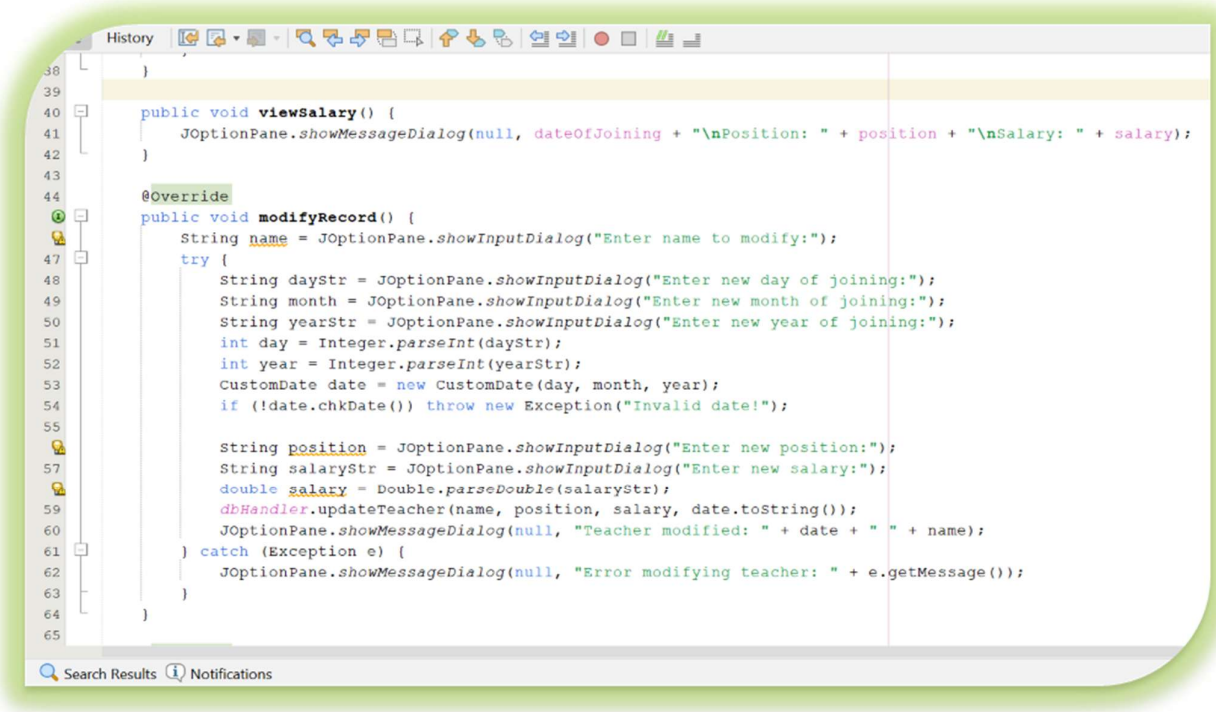
6.6 Teacher

- **Purpose:** Encapsulates the logic for managing teacher records, including database interactions and user interface feedback.
- **Detailed Procedural Analysis:**
  1. **Constructor (`Teacher(String name, String position, double salary, CustomDate dateOfJoining)`):**
     - Initializes instance variables: assigns the provided name to the inherited name field, and sets position, salary, and dateOfJoining with the provided values.
     - Instantiates a static DatabaseHandler object (dbHandler) for database operations, shared across all Teacher instances.
     - **Illustrative Example:** new Teacher("John Doe", "Professor", 5000, new CustomDate(15, "January", 2025)) creates a teacher object with the specified attributes.
  2. `addRecord()` **Method:**
     - Prompts the user for input using `JOptionPane.showInputDialog()` to collect day, month, year, name, position, and salary.
     - Parses the day and year inputs as integers using `Integer.parseInt()`, and the salary as a double using `Double.parseDouble()`.
     - Constructs a CustomDate object and validates it using `chkDate()`; throws an exception if the date is invalid.
     - Invokes `dbHandler.saveTeacher()` to persist the teacher record to the database.
     - Displays a success message ("Teacher added: [date] [name]") or an error message via JOptionPane if an exception occurs.
     - **Illustrative Example:** Inputs of "15", "January", "2025", "John Doe", "Professor", "5000" result in a new teacher record being added.
  3. `viewSalary()` **Method:**
     - Uses `JOptionPane.showMessageDialog()` to display the teacher's dateOfJoining, position, and salary in a formatted dialog.
     - **Illustrative Example:** For a teacher with dateOfJoining = "15 January 2025", position = "Professor", and salary = 5000, the dialog displays these values.
  4. `modifyRecord()` **Method:**
     - Prompts the user for the name of the teacher to modify and new values for day, month, year, position, and salary using JOptionPane.
     - Validates the new date and updates the record using `dbHandler.updateTeacher()`.
     - Displays a success or error message.
     - **Illustrative Example:** Modifying "John Doe" with a new salary of 5500 updates the database accordingly.
  5. `searchRecord()` **and** `deleteRecord()` **Methods:**
     - Prompt for the teacher's name, then invoke `dbHandler.searchTeacher()` or `dbHandler.deleteTeacher()` respectively.
     - Display the results or confirmation via JOptionPane.
     - **Illustrative Example:** Searching for "John Doe" displays the teacher's details; deleting "John Doe" removes the record.

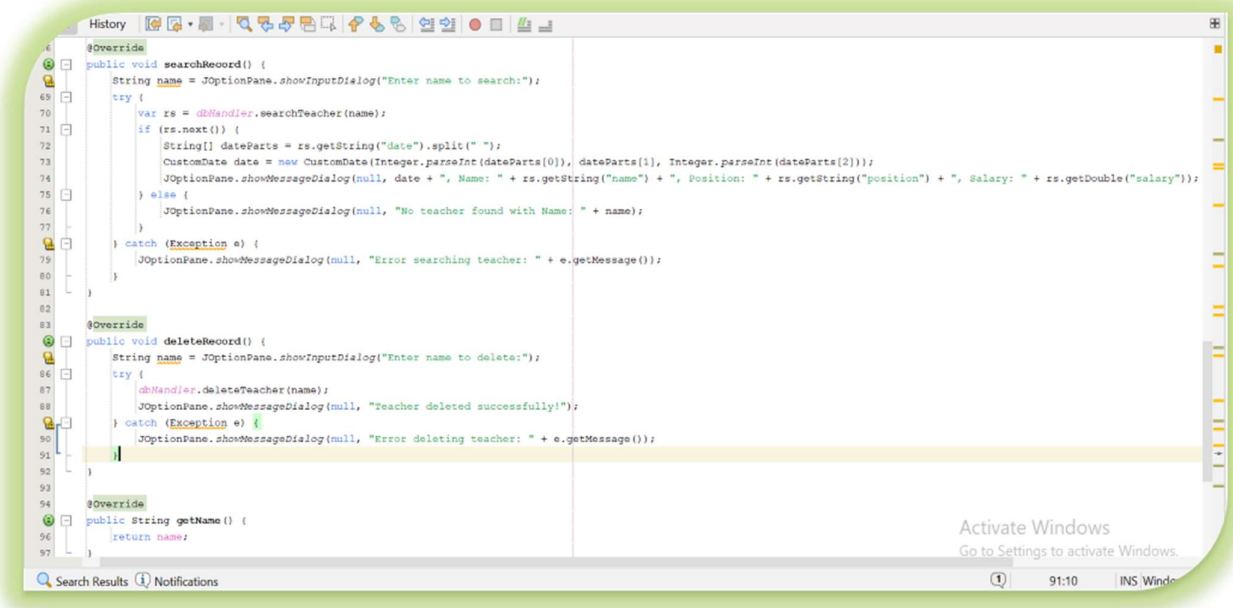
6. `getName()` **Method:**
    - Returns the name field.
    - **Illustrative Example:** Returns "John Doe" for the teacher instance.
7. `toString()` **Method:**
    - Returns a formatted string containing dateOfJoining, name, position, and salary (e.g., "15 January 2025, Name: John Doe, Position: Professor, Salary: 5000").

- **Database Relevance:** Maps directly to the teacher table, where name is the primary key, ensuring unique identification of records.

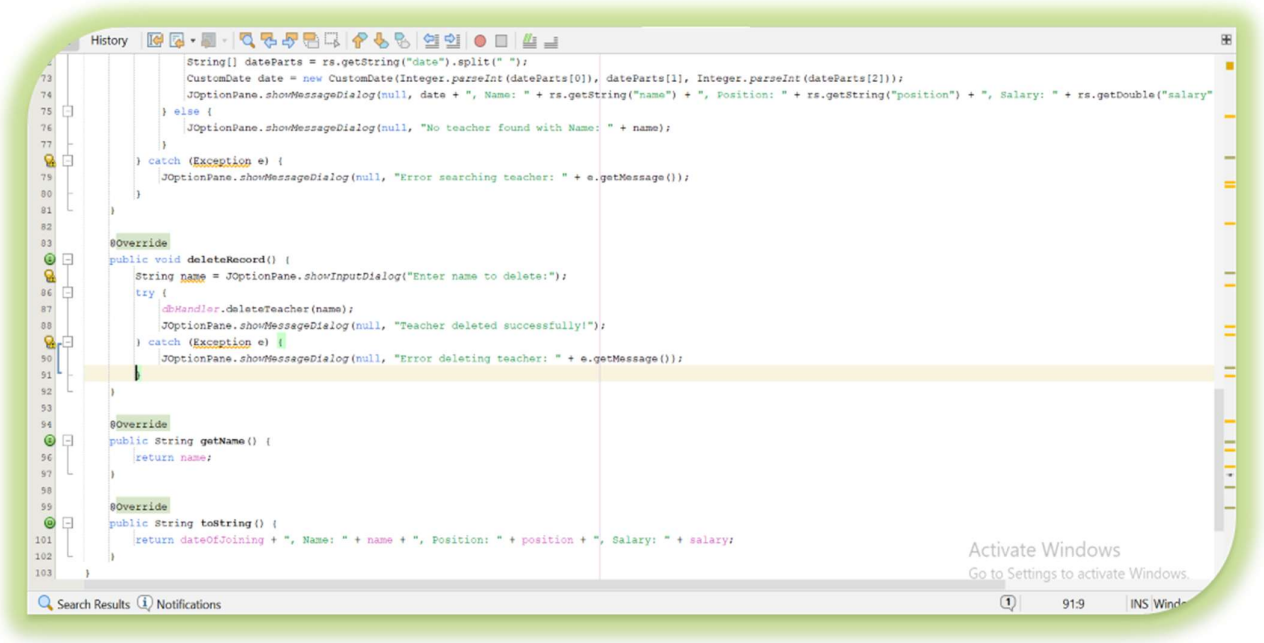
```
History
public class Teacher extends Person {
    protected String position;
    protected double salary;
    protected CustomDate dateOfJoining;
    private static DatabaseHandler dbHandler = new DatabaseHandler();

    public Teacher(String name, String position, double salary, CustomDate dateOfJoining) {
        this.name = name;
        this.position = position;
        this.salary = salary;
        this.dateOfJoining = dateOfJoining;
    }

    @Override
    public void addRecord() {
        String dayStr = JOptionPane.showInputDialog("Enter day of joining:");
        String month = JOptionPane.showInputDialog("Enter month of joining:");
        String yearStr = JOptionPane.showInputDialog("Enter year of joining:");
        String name = JOptionPane.showInputDialog("Enter teacher name:");
        String position = JOptionPane.showInputDialog("Enter position:");
        String salaryStr = JOptionPane.showInputDialog("Enter salary:");
        try {
            int day = Integer.parseInt(dayStr);
            int year = Integer.parseInt(yearStr);
            CustomDate date = new CustomDate(day, month, year);
            if (!date.chkDate()) throw new Exception("Invalid date!");

            double salary = Double.parseDouble(salaryStr);
            dbHandler.saveTeacher(name, position, salary, date.toString());
            JOptionPane.showMessageDialog(null, "Teacher added: " + date + " " + name);
        } catch (Exception e) {
            JOptionPane.showMessageDialog(null, "Error adding teacher: " + e.getMessage());
        }
    }
Search Results  Notifications
```



```
History
38      }
39
40      public void viewSalary() {
41          JOptionPane.showMessageDialog(null, dateOfJoining + "\nPosition: " + position + "\nSalary: " + salary);
42      }
43
44      @Override
        public void modifyRecord() {
            String name = JOptionPane.showInputDialog("Enter name to modify:");
47          try {
48              String dayStr = JOptionPane.showInputDialog("Enter new day of joining:");
49              String month = JOptionPane.showInputDialog("Enter new month of joining:");
50              String yearStr = JOptionPane.showInputDialog("Enter new year of joining:");
51              int day = Integer.parseInt(dayStr);
52              int year = Integer.parseInt(yearStr);
53              CustomDate date = new CustomDate(day, month, year);
54              if (!date.chkDate()) throw new Exception("Invalid date!");
55
                String position = JOptionPane.showInputDialog("Enter new position:");
57              String salaryStr = JOptionPane.showInputDialog("Enter new salary:");
                double salary = Double.parseDouble(salaryStr);
59              dbHandler.updateTeacher(name, position, salary, date.toString());
60              JOptionPane.showMessageDialog(null, "Teacher modified: " + date + " " + name);
61          } catch (Exception e) {
62              JOptionPane.showMessageDialog(null, "Error modifying teacher: " + e.getMessage());
63          }
64      }
65
Search Results  Notifications
```



```
History
@Override
public void searchRecord() {
    String name = JOptionPane.showInputDialog("Enter name to search:");
69      try {
70          var rs = dbHandler.searchTeacher(name);
71          if (rs.next()) {
72              String[] dateParts = rs.getString("date").split(" ");
73              CustomDate date = new CustomDate(Integer.parseInt(dateParts[0]), dateParts[1], Integer.parseInt(dateParts[2]));
74              JOptionPane.showMessageDialog(null, date + ", Name: " + rs.getString("name") + ", Position: " + rs.getString("position") + ", Salary: " + rs.getDouble("salary"));
75          } else {
76              JOptionPane.showMessageDialog(null, "No teacher found with Name: " + name);
77          }
79      } catch (Exception e) {
79          JOptionPane.showMessageDialog(null, "Error searching teacher: " + e.getMessage());
80      }
81  }
82
83  @Override
    public void deleteRecord() {
        String name = JOptionPane.showInputDialog("Enter name to delete:");
86      try {
87          dbHandler.deleteTeacher(name);
88          JOptionPane.showMessageDialog(null, "Teacher deleted successfully!");
89      } catch (Exception e) {
90          JOptionPane.showMessageDialog(null, "Error deleting teacher: " + e.getMessage());
91      }
92  }
93
94  @Override
    public String getName() {
96      return name;
97  }
Search Results  Notifications
```



6.7 TeacherSectionGUI

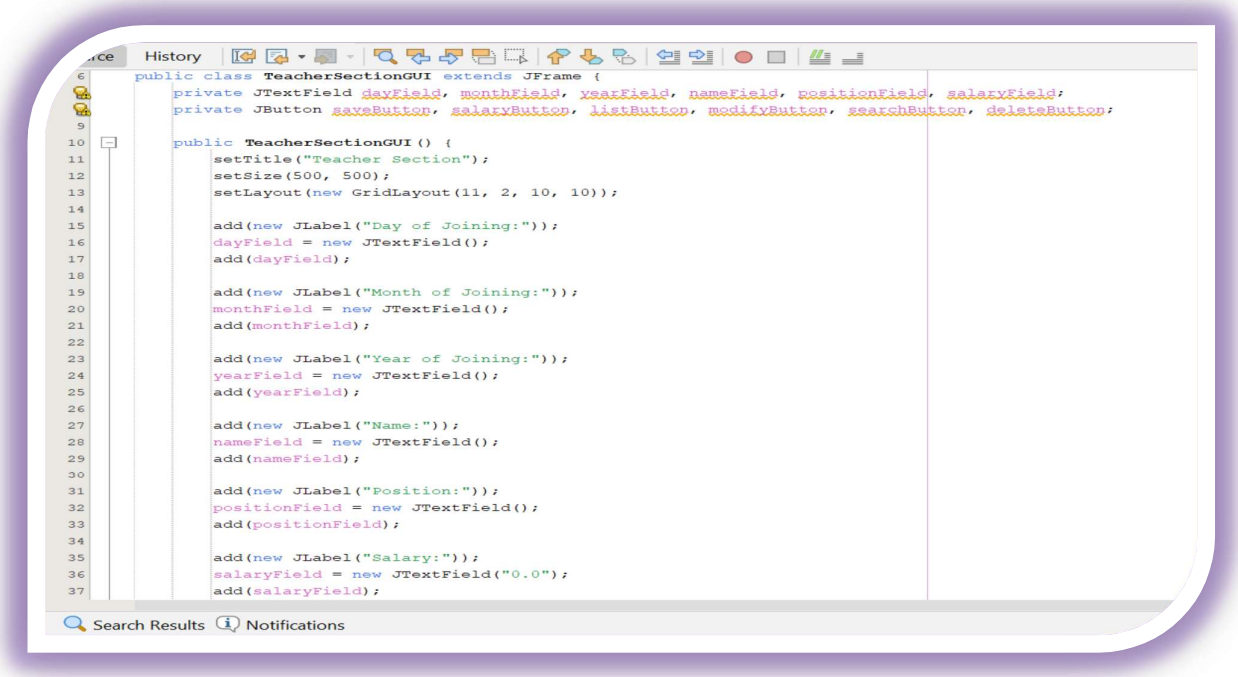
- **Purpose:** Provides a graphical interface for managing teacher records, integrating user input with database operations.
- **Detailed Procedural Analysis:**
    1. **Constructor (`TeacherSectionGUI()`):**
        1. Configures the JFrame with the title "Teacher Section", dimensions of 500x500 pixels, and a GridLayout(11, 2) layout with 10-pixel gaps.
        2. Adds JLabel and JTextField pairs for day, month, year, name, position, and salary inputs.
        3. Adds six JButton objects ("Save," "View Salary," "List Teachers," "Modify," "Search," "Delete") with associated ActionListener implementations.
        4. Sets the default close operation to DISPOSE_ON_CLOSE, ensuring the window closes without terminating the application.
        5. Calls `setVisible(true)` to display the window.
        6. **Illustrative Example:** Instantiating TeacherSectionGUI opens a window with the specified layout and components.
    2. `saveTeacher()` **Method:**
        1. Parses the text field inputs: day, month, and year as integers and strings, name and position as strings, and salary as a double.
        2. Constructs a CustomDate object and validates it using chkDate(); throws an exception if invalid.

3. Validates other inputs: ensures name and position are non-empty and salary is non-negative, throwing an exception if invalid.
4. Instantiates a DatabaseHandler and calls saveTeacher() to persist the record.
5. Displays a success message ("Teacher saved: [date] [name]") or an error message via JOptionPane.
6. Closes the window using dispose() upon successful save.
7. **Illustrative Example:** Inputs of "15", "January", "2025", "John Doe", "Professor", "5000" save a new teacher record and close the window.

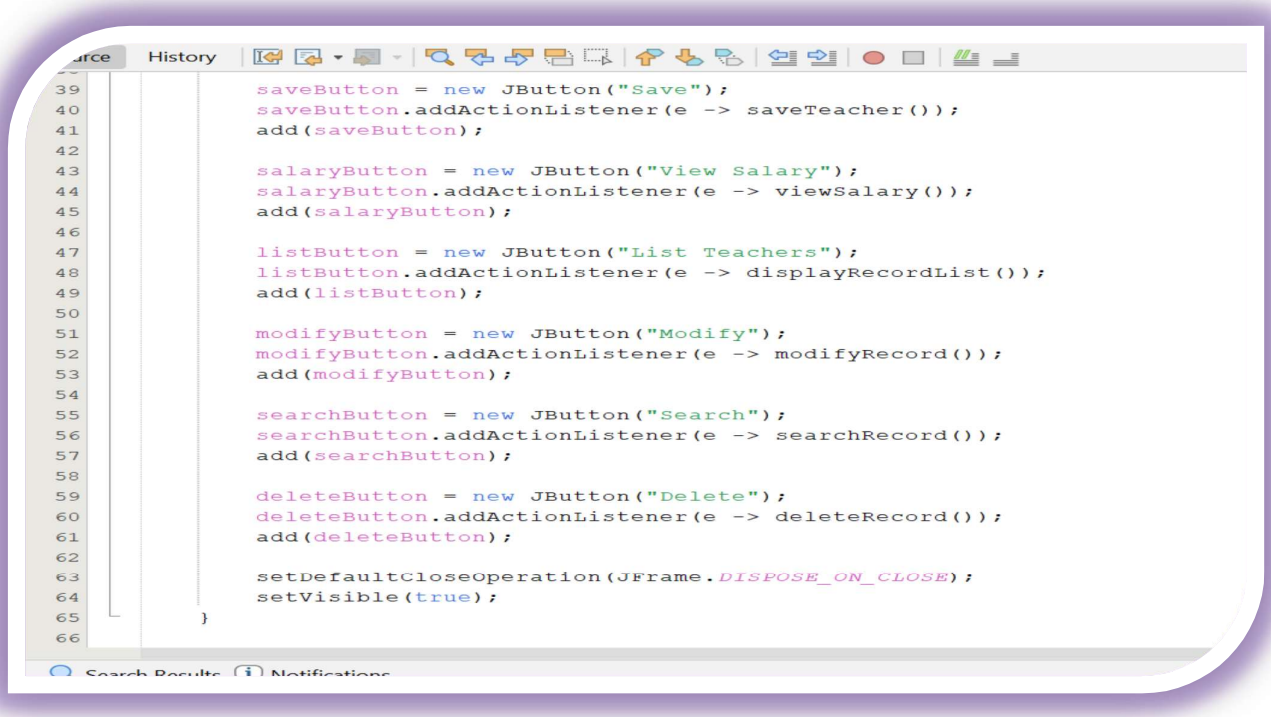
3. `viewSalary()` **Method:**
   1. Parses the text field inputs to create a CustomDate and validates it.
   2. Constructs a Teacher object with the parsed name, position, salary, and date.
   3. Calls `teacher.viewSalary()` to display the salary details in a dialog.
   4. Handles exceptions by displaying error messages.
   5. **Illustrative Example:** Displays "15 January 2025\nPosition: Professor\nSalary: 5000" for the given inputs.
4. `displayRecordList()` **Method:**
   1. Instantiates a DatabaseHandler and retrieves a ResultSet using `getTeachers()`.
   2. Iterates over the ResultSet, parsing the date string into a CustomDate object and formatting each record with a StringBuilder.
   3. Displays the formatted list in a JOptionPane dialog; if no records exist, displays "No records found."
   4. **Illustrative Example:** Shows "Dennis: "15 January 2025, Name: John Doe, Position: Professor, Salary: 5000".

5. `modifyRecord()`, `searchRecord()`, **and** `deleteRecord()` **Methods:**
   1. Use JOptionPane to prompt for the teacher's name (for modify and delete) or search criteria.
   2. Call the corresponding DatabaseHandler methods (`updateTeacher()`, `searchTeacher()`, `deleteTeacher()`).
   3. Display the results or confirmation messages via JOptionPane.
   4. **Illustrative Example:** Modifying "John Doe" updates the record; searching for "John Doe" displays the record; deleting "John Doe" removes it.

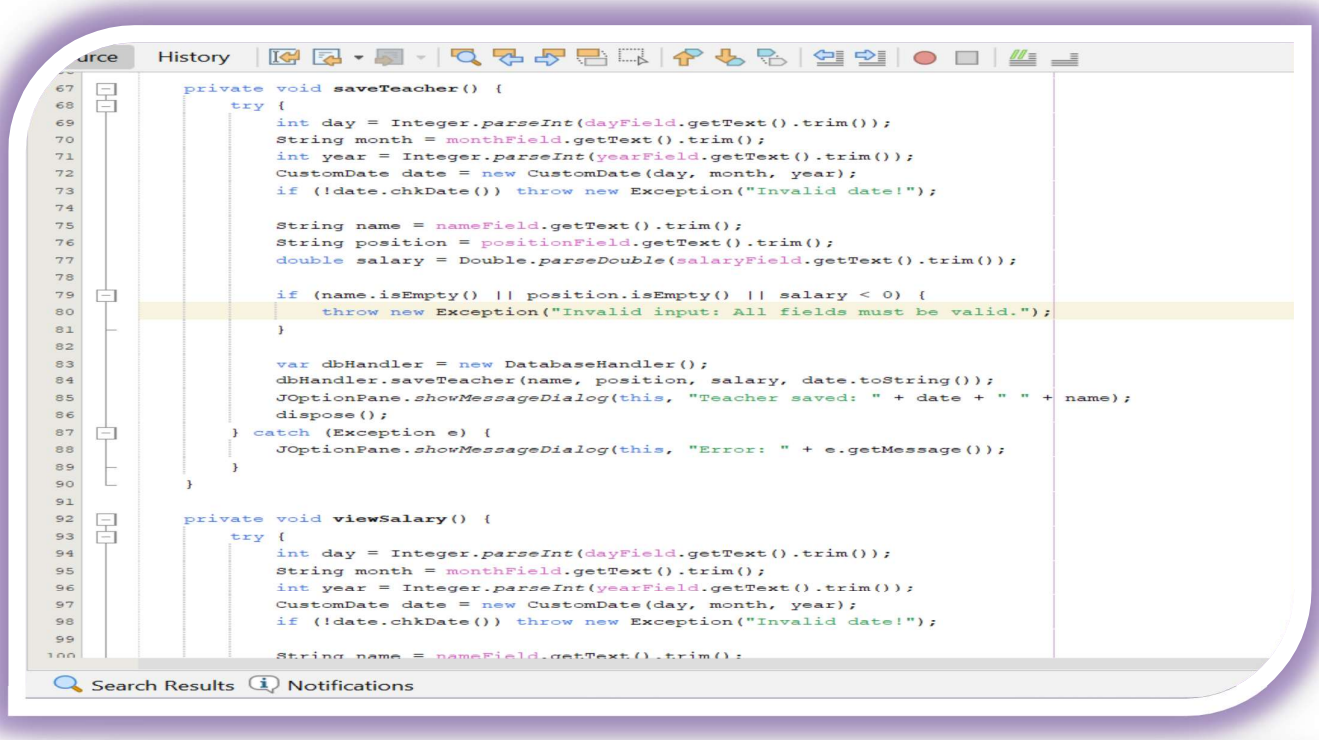
- **Database Relevance:** Interacts with the teacher table, ensuring all operations respect the name primary key constraint.



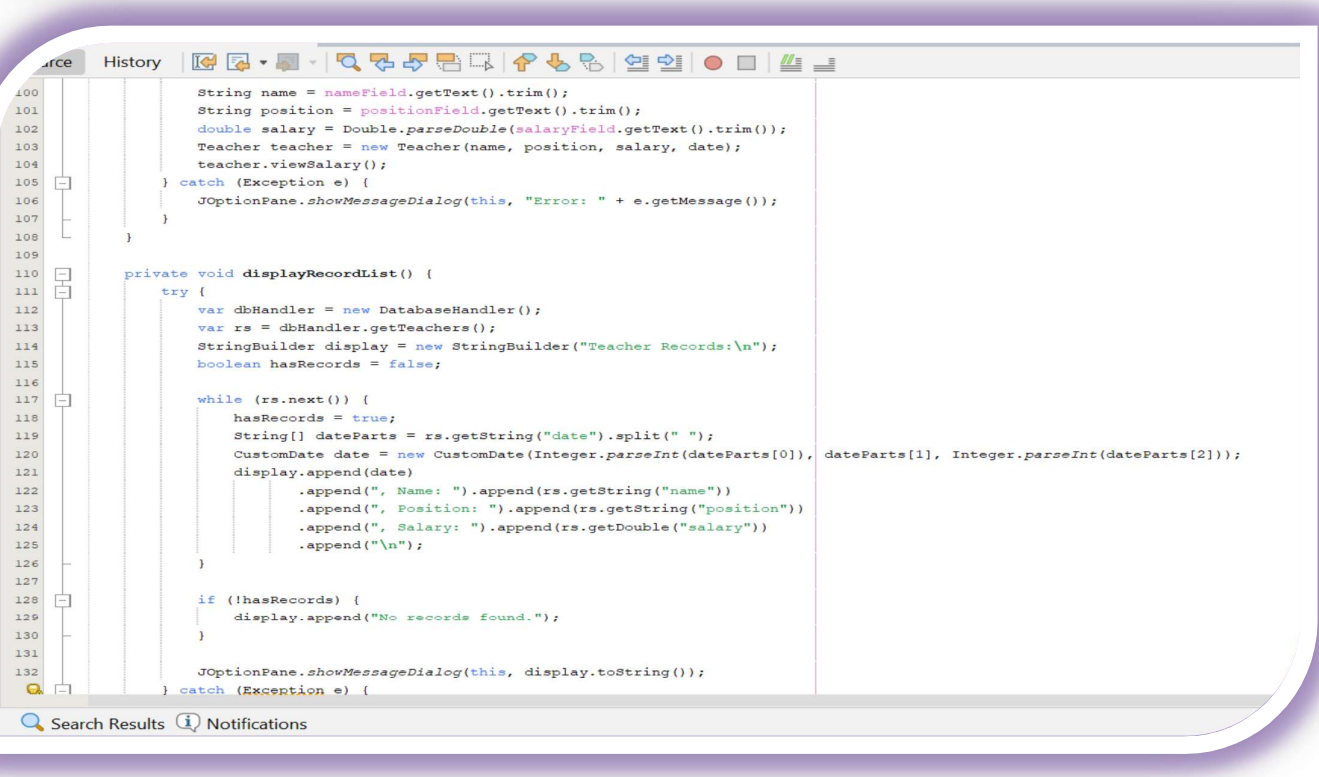
```
public class TeacherSectionGUI extends JFrame {
    private JTextField dayField, monthField, yearField, nameField, positionField, salaryField;
    private JButton saveButton, salaryButton, listButton, modifyButton, searchButton, deleteButton;

    public TeacherSectionGUI() {
        setTitle("Teacher Section");
        setSize(500, 500);
        setLayout(new GridLayout(11, 2, 10, 10));

        add(new JLabel("Day of Joining:"));
        dayField = new JTextField();
        add(dayField);

        add(new JLabel("Month of Joining:"));
        monthField = new JTextField();
        add(monthField);

        add(new JLabel("Year of Joining:"));
        yearField = new JTextField();
        add(yearField);

        add(new JLabel("Name:"));
        nameField = new JTextField();
        add(nameField);

        add(new JLabel("Position:"));
        positionField = new JTextField();
        add(positionField);

        add(new JLabel("Salary:"));
        salaryField = new JTextField("0.0");
        add(salaryField);
```



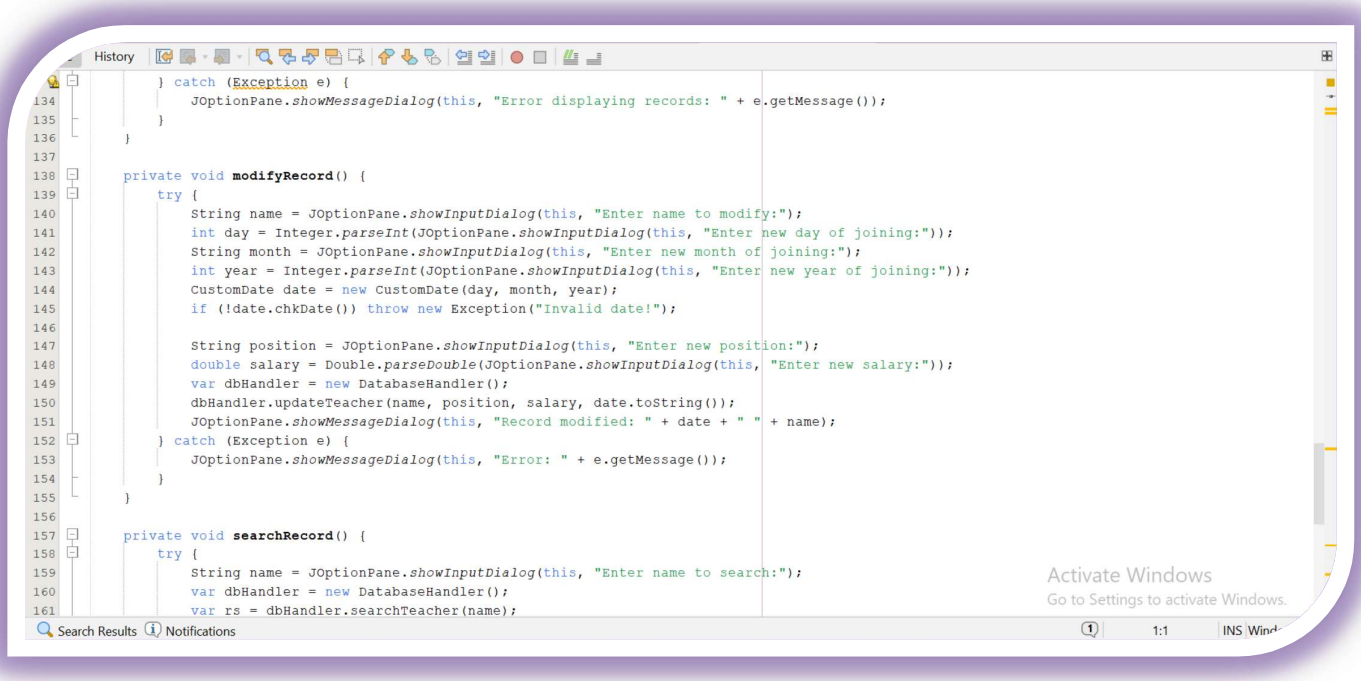
```
        saveButton = new JButton("Save");
        saveButton.addActionListener(e -> saveTeacher());
        add(saveButton);

        salaryButton = new JButton("View Salary");
        salaryButton.addActionListener(e -> viewSalary());
        add(salaryButton);

        listButton = new JButton("List Teachers");
        listButton.addActionListener(e -> displayRecordList());
        add(listButton);

        modifyButton = new JButton("Modify");
        modifyButton.addActionListener(e -> modifyRecord());
        add(modifyButton);

        searchButton = new JButton("Search");
        searchButton.addActionListener(e -> searchRecord());
        add(searchButton);

        deleteButton = new JButton("Delete");
        deleteButton.addActionListener(e -> deleteRecord());
        add(deleteButton);

        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        setVisible(true);
    }
```



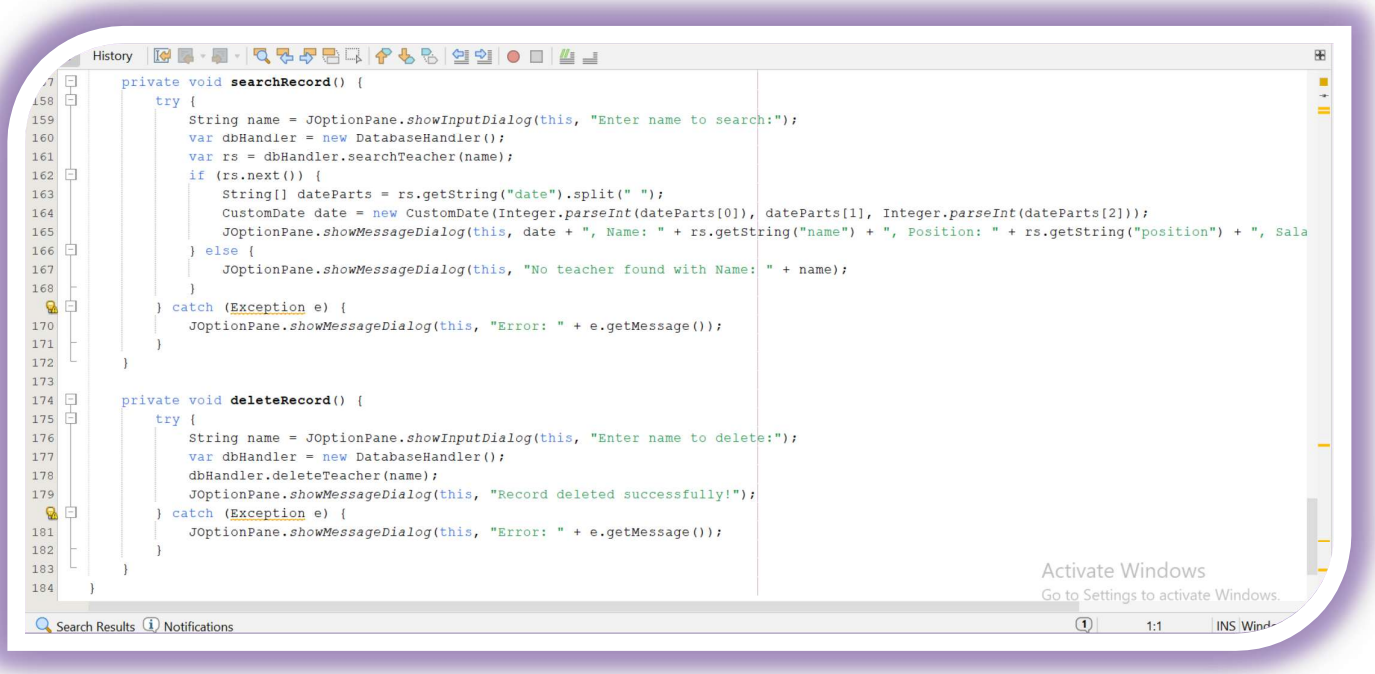
```
    private void saveTeacher() {
        try {
            int day = Integer.parseInt(dayField.getText().trim());
            String month = monthField.getText().trim();
            int year = Integer.parseInt(yearField.getText().trim());
            CustomDate date = new CustomDate(day, month, year);
            if (!date.chkDate()) throw new Exception("Invalid date!");

            String name = nameField.getText().trim();
            String position = positionField.getText().trim();
            double salary = Double.parseDouble(salaryField.getText().trim());

            if (name.isEmpty() || position.isEmpty() || salary < 0) {
                throw new Exception("Invalid input: All fields must be valid.");
            }

            var dbHandler = new DatabaseHandler();
            dbHandler.saveTeacher(name, position, salary, date.toString());
            JOptionPane.showMessageDialog(this, "Teacher saved: " + date + " " + name);
            dispose();
        } catch (Exception e) {
            JOptionPane.showMessageDialog(this, "Error: " + e.getMessage());
        }
    }

    private void viewSalary() {
        try {
            int day = Integer.parseInt(dayField.getText().trim());
            String month = monthField.getText().trim();
            int year = Integer.parseInt(yearField.getText().trim());
            CustomDate date = new CustomDate(day, month, year);
            if (!date.chkDate()) throw new Exception("Invalid date!");
```



```
            String name = nameField.getText().trim();
            String position = positionField.getText().trim();
            double salary = Double.parseDouble(salaryField.getText().trim());
            Teacher teacher = new Teacher(name, position, salary, date);
            teacher.viewSalary();
        } catch (Exception e) {
            JOptionPane.showMessageDialog(this, "Error: " + e.getMessage());
        }
    }

    private void displayRecordList() {
        try {
            var dbHandler = new DatabaseHandler();
            var rs = dbHandler.getTeachers();
            StringBuilder display = new StringBuilder("Teacher Records:\n");
            boolean hasRecords = false;

            while (rs.next()) {
                hasRecords = true;
                String[] dateParts = rs.getString("date").split(" ");
                CustomDate date = new CustomDate(Integer.parseInt(dateParts[0]), dateParts[1], Integer.parseInt(dateParts[2]));
                display.append(date)
                        .append(", Name: ").append(rs.getString("name"))
                        .append(", Position: ").append(rs.getString("position"))
                        .append(", Salary: ").append(rs.getDouble("salary"))
                        .append("\n");
            }

            if (!hasRecords) {
                display.append("No records found.");
            }

            JOptionPane.showMessageDialog(this, display.toString());
        } catch (Exception e) {
```



```
        } catch (Exception e) {
            JOptionPane.showMessageDialog(this, "Error displaying records: " + e.getMessage());
        }
    }

    private void modifyRecord() {
        try {
            String name = JOptionPane.showInputDialog(this, "Enter name to modify:");
            int day = Integer.parseInt(JOptionPane.showInputDialog(this, "Enter new day of joining:"));
            String month = JOptionPane.showInputDialog(this, "Enter new month of joining:");
            int year = Integer.parseInt(JOptionPane.showInputDialog(this, "Enter new year of joining:"));
            CustomDate date = new CustomDate(day, month, year);
            if (!date.chkDate()) throw new Exception("Invalid date!");

            String position = JOptionPane.showInputDialog(this, "Enter new position:");
            double salary = Double.parseDouble(JOptionPane.showInputDialog(this, "Enter new salary:"));
            var dbHandler = new DatabaseHandler();
            dbHandler.updateTeacher(name, position, salary, date.toString());
            JOptionPane.showMessageDialog(this, "Record modified: " + date + " " + name);
        } catch (Exception e) {
            JOptionPane.showMessageDialog(this, "Error: " + e.getMessage());
        }
    }

    private void searchRecord() {
        try {
            String name = JOptionPane.showInputDialog(this, "Enter name to search:");
            var dbHandler = new DatabaseHandler();
            var rs = dbHandler.searchTeacher(name);
```

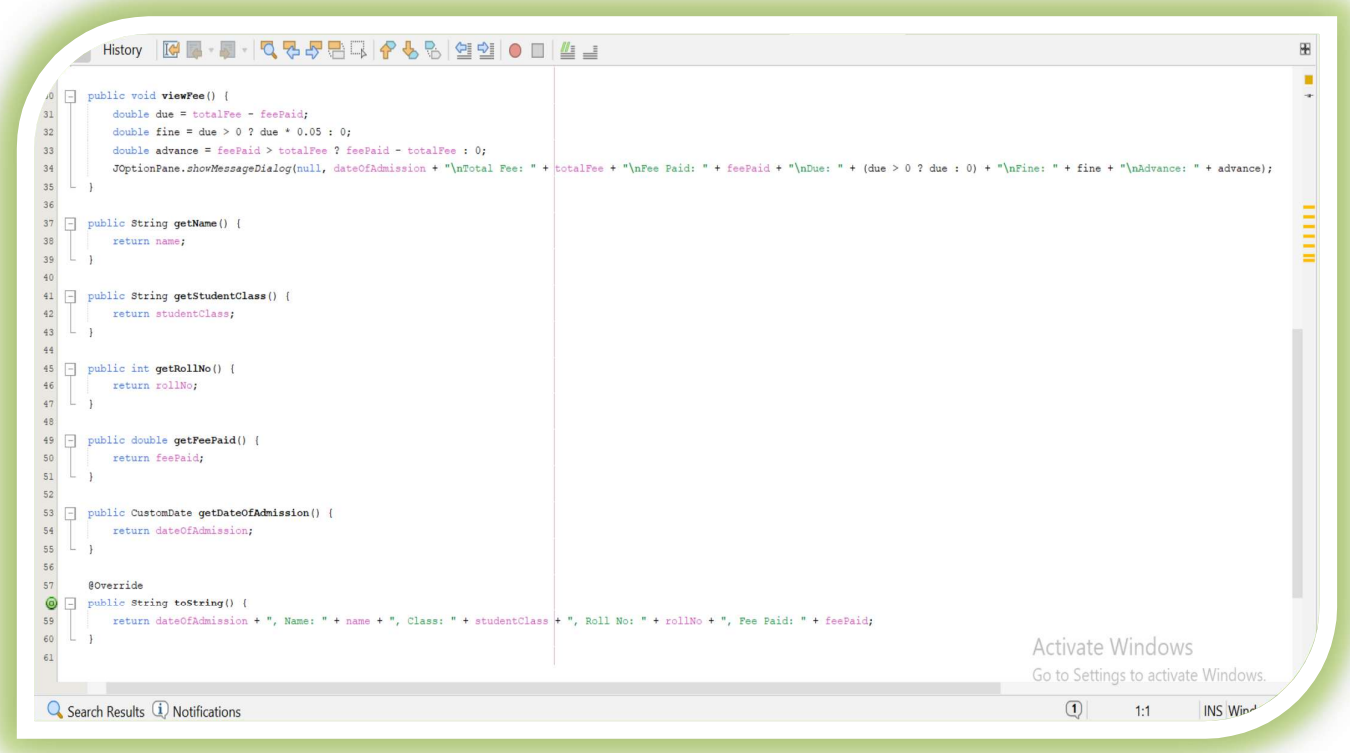
```
    private void searchRecord() {
        try {
            String name = JOptionPane.showInputDialog(this, "Enter name to search:");
            var dbHandler = new DatabaseHandler();
            var rs = dbHandler.searchTeacher(name);
            if (rs.next()) {
                String[] dateParts = rs.getString("date").split(" ");
                CustomDate date = new CustomDate(Integer.parseInt(dateParts[0]), dateParts[1], Integer.parseInt(dateParts[2]));
                JOptionPane.showMessageDialog(this, date + ", Name: " + rs.getString("name") + ", Position: " + rs.getString("position") + ", Sala
            } else {
                JOptionPane.showMessageDialog(this, "No teacher found with Name: " + name);
            }
        } catch (Exception e) {
            JOptionPane.showMessageDialog(this, "Error: " + e.getMessage());
        }
    }

    private void deleteRecord() {
        try {
            String name = JOptionPane.showInputDialog(this, "Enter name to delete:");
            var dbHandler = new DatabaseHandler();
            dbHandler.deleteTeacher(name);
            JOptionPane.showMessageDialog(this, "Record deleted successfully!");
        } catch (Exception e) {
            JOptionPane.showMessageDialog(this, "Error: " + e.getMessage());
        }
    }
}
```

6.8 Student

- **Purpose:** Encapsulates the logic for managing student records, including fee-related calculations.
- **Detailed Procedural Analysis:**
  1. **Constructor (`Student(String name, String studentClass, int rollNo, double feePaid, CustomDate dateOfAdmission)`):**
     - Initializes instance variables: name, studentClass, rollNo, feePaid, dateOfAdmission, and a static totalFee set to 1000.
     - Instantiates a static DatabaseHandler for database operations.
     - **Illustrative Example:** new Student("Jane Smith", "10A", 101, 800, new CustomDate(10, "February", 2025)).
  2. `viewFee()` **Method:**
     - Calculates the due amount: due = totalFee - feePaid.
     - Computes a fine: fine = due > 0 ? due * 0.05 : 0, applying a 5% fine on any due amount.
     - Computes advance: advance = feePaid > totalFee ? feePaid - totalFee : 0, determining any overpayment.
     - Displays the results in a JOptionPane dialog: dateOfAdmission, totalFee, feePaid, due, fine, and advance.
     - **Illustrative Example:** For feePaid = 800, displays "10 February 2025\nTotal Fee: 1000\nFee Paid: 800\nDue: 200\nFine: 10\nAdvance: 0".
  3. **Getter Methods (`getName()`, `getStudentClass()`, `getRollNo()`, `getFeePaid()`, `getDateOfAdmission()`):**
     - Provide access to the instance variables.
     - **Illustrative Example:** `getRollNo()` returns 101 for the student.
  4. `toString()` **Method:**
     - Returns a formatted string: dateOfAdmission, name, studentClass, rollNo, feePaid.
     - **Illustrative Example:** "10 February 2025, Name: Jane Smith, Class: 10A, Roll No: 101, Fee Paid: 800".
- **Database Relevance:** Maps to the student table, with rollNo as the primary key.

```java
public class Student {
    private String name;
    private String studentClass;
    private int rollNo;
    private double feePaid;
    private double totalFee = 1000.0;
    private CustomDate dateOfAdmission;
    private static DatabaseHandler dbHandler = new DatabaseHandler();

    public Student(String name, String studentClass, int rollNo, double feePaid, CustomDate dateOfAdmission) {
        this.name = name;
        this.studentClass = studentClass;
        this.rollNo = rollNo;
        this.feePaid = feePaid;
        this.dateOfAdmission = dateOfAdmission;
    }

    public void viewFee() {
        double due = totalFee - feePaid;
        double fine = due > 0 ? due * 0.05 : 0;
        double advance = feePaid > totalFee ? feePaid - totalFee : 0;
        JOptionPane.showMessageDialog(null, dateOfAdmission + "\nTotal Fee: " + totalFee + "\nFee Paid: " + feePaid + "\nDue: " + (due > 0 ? due :
    }

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


Search Results Notifications 1:1 INS



History

```java
    public void viewFee() {
        double due = totalFee - feePaid;
        double fine = due > 0 ? due * 0.05 : 0;
        double advance = feePaid > totalFee ? feePaid - totalFee : 0;
        JOptionPane.showMessageDialog(null, dateOfAdmission + "\nTotal Fee: " + totalFee + "\nFee Paid: " + feePaid + "\nDue: " + (due > 0 ? due : 0) + "\nFine: " + fine + "\nAdvance: " + advance);
    }

    public String getName() {
        return name;
    }

    public String getStudentClass() {
        return studentClass;
    }

    public int getRollNo() {
        return rollNo;
    }

    public double getFeePaid() {
        return feePaid;
    }

    public CustomDate getDateOfAdmission() {
        return dateOfAdmission;
    }

    @Override
    public String toString() {
        return dateOfAdmission + ", Name: " + name + ", Class: " + studentClass + ", Roll No: " + rollNo + ", Fee Paid: " + feePaid;
    }
```

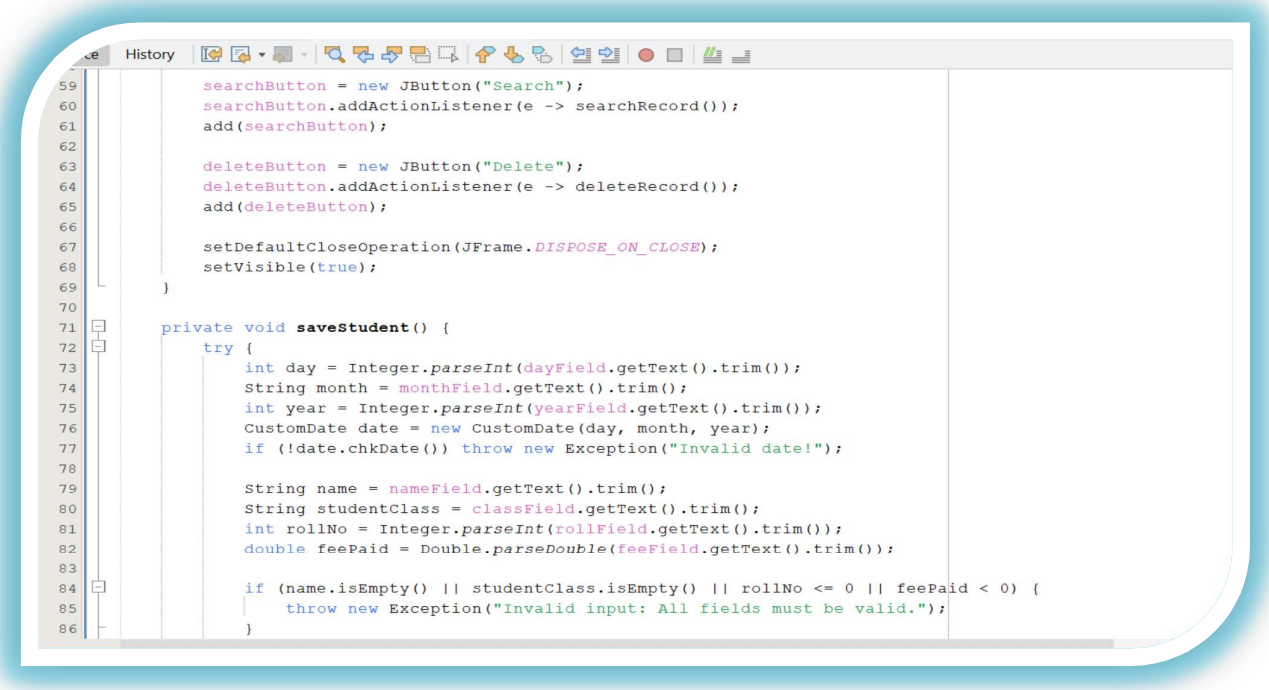

Search Results Notifications 1:1 INS

6.9 StudentSectionGUI

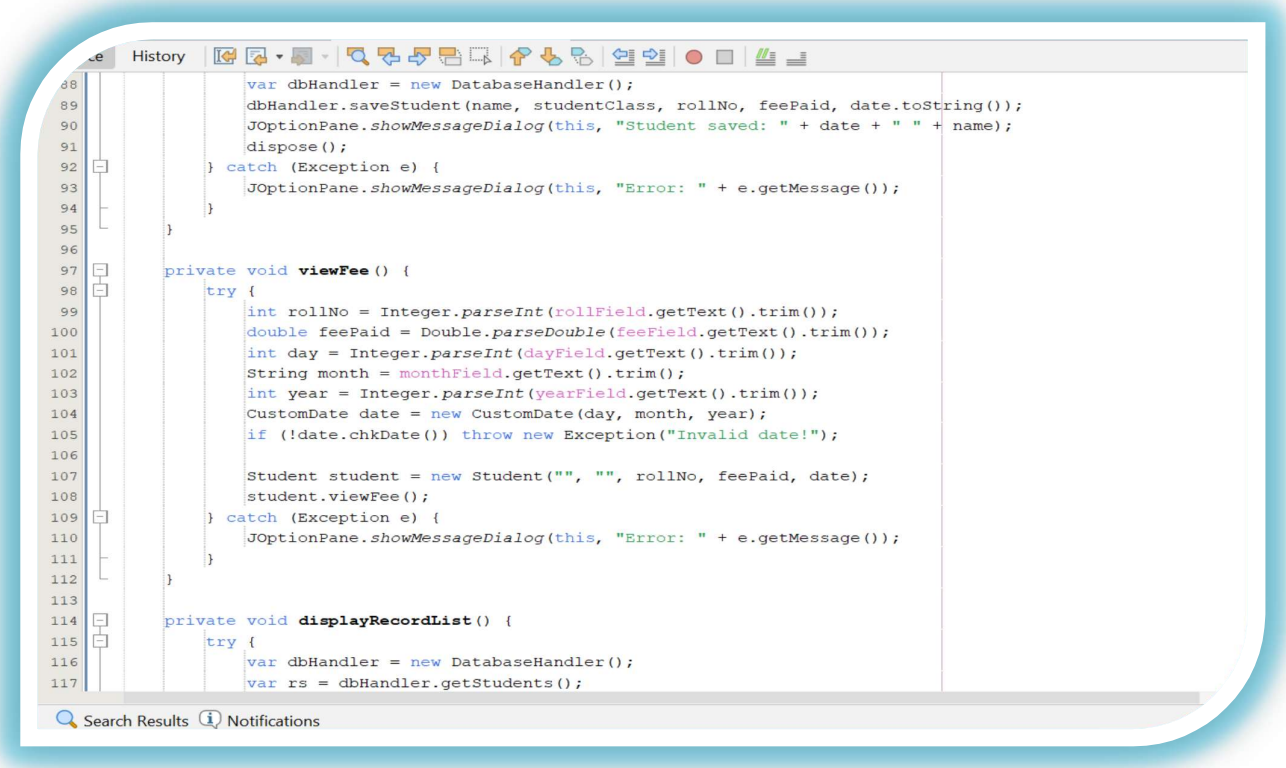
- **Purpose:** Provides a graphical interface for managing student records, integrating user input with database operations.
- **Detailed Procedural Analysis:**
  1. **Constructor (`StudentSectionGUI()`):**
     - Configures the JFrame with the title "Student Section", dimensions of 500x500 pixels, and a GridLayout(12, 2) layout with 10-pixel gaps.
     - Adds `JLabel` and `JTextField` pairs for day, month, year, name, class, roll number, and fee paid.
     - Adds six `JButton` objects ("Save," "View Fee," "List Students," "Modify," "Search," "Delete") with `ActionListener` implementations.
     - Sets the default close operation to DISPOSE_ON_CLOSE.
     - **Illustrative Example:** Opens a window with the specified layout.
  2. `saveStudent()` **Method:**
     - Parses inputs, validates the date with `chkDate()`, and ensures name, studentClass, rollNo > 0, and feePaid >= 0.
     - Calls `dbHandler.saveStudent()` to persist the record.
     - Displays a success message or error via JOptionPane, closing the window on success.
     - **Illustrative Example:** Saves "Jane Smith" with rollNo 101.
  3. `viewFee()` **Method:**
     - Parses inputs, constructs a Student object, and calls `viewFee()` to display fee details.
     - **Illustrative Example:** Shows "10 February 2025\nTotal Fee: 1000\nFee Paid: 800\nDue: 200\nFine: 10\nAdvance: 0".
  4. `displayRecordList()`, `modifyRecord()`, `searchRecord()`, `deleteRecord()` **Methods:**
     - Retrieve and display student records, or perform modify/search/delete operations using dbHandler.
     - **Illustrative Example:** `displayRecordList()` shows "10 February 2025, Name: Jane Smith, Class: 10A, Roll No: 101, Fee Paid: 800".
- **Database Relevance:** Interacts with the student table, respecting the rollNo primary key.

```java
import javax.swing.*;
import java.awt.*;

public class StudentSectionGUI extends JFrame {
    private JTextField dayField, monthField, yearField, nameField, classField, rollField, feeField;
    private JButton saveButton, feeButton, listButton, modifyButton, searchButton, deleteButton;

    public StudentSectionGUI() {
        setTitle("Student Section");
        setSize(500, 500);
        setLayout(new GridLayout(12, 2, 10, 10));

        add(new JLabel("Day of Admission:"));
        dayField = new JTextField();
        add(dayField);

        add(new JLabel("Month of Admission:"));
        monthField = new JTextField();
        add(monthField);

        add(new JLabel("Year of Admission:"));
        yearField = new JTextField();
        add(yearField);

        add(new JLabel("Name:"));
        nameField = new JTextField();
        add(nameField);
```

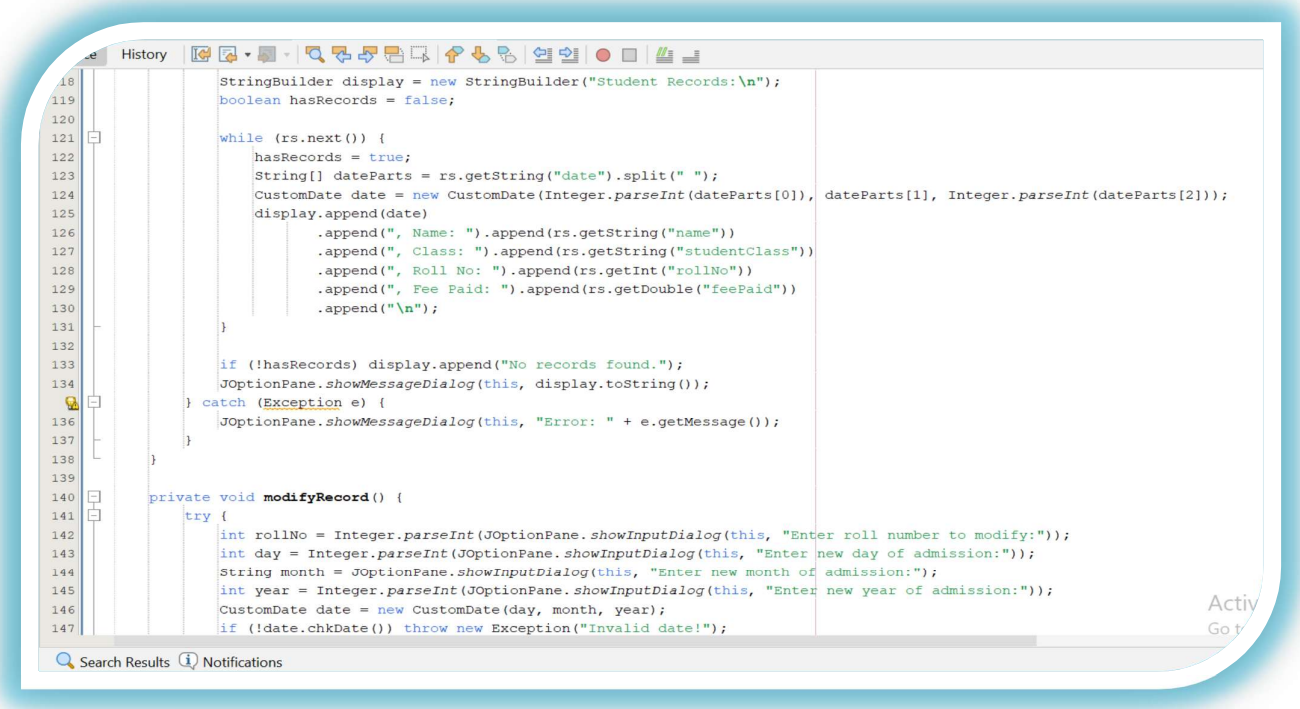
```java
        add(new JLabel("Class:"));
        classField = new JTextField();
        add(classField);

        add(new JLabel("Roll Number:"));
        rollField = new JTextField();
        add(rollField);

        add(new JLabel("Fee Paid:"));
        feeField = new JTextField("0.0");
        add(feeField);

        saveButton = new JButton("Save");
        saveButton.addActionListener(e -> saveStudent());
        add(saveButton);

        feeButton = new JButton("View Fee");
        feeButton.addActionListener(e -> viewFee());
        add(feeButton);

        listButton = new JButton("List Students");
        listButton.addActionListener(e -> displayRecordList());
        add(listButton);

        modifyButton = new JButton("Modify");
        modifyButton.addActionListener(e -> modifyRecord());
        add(modifyButton);
```



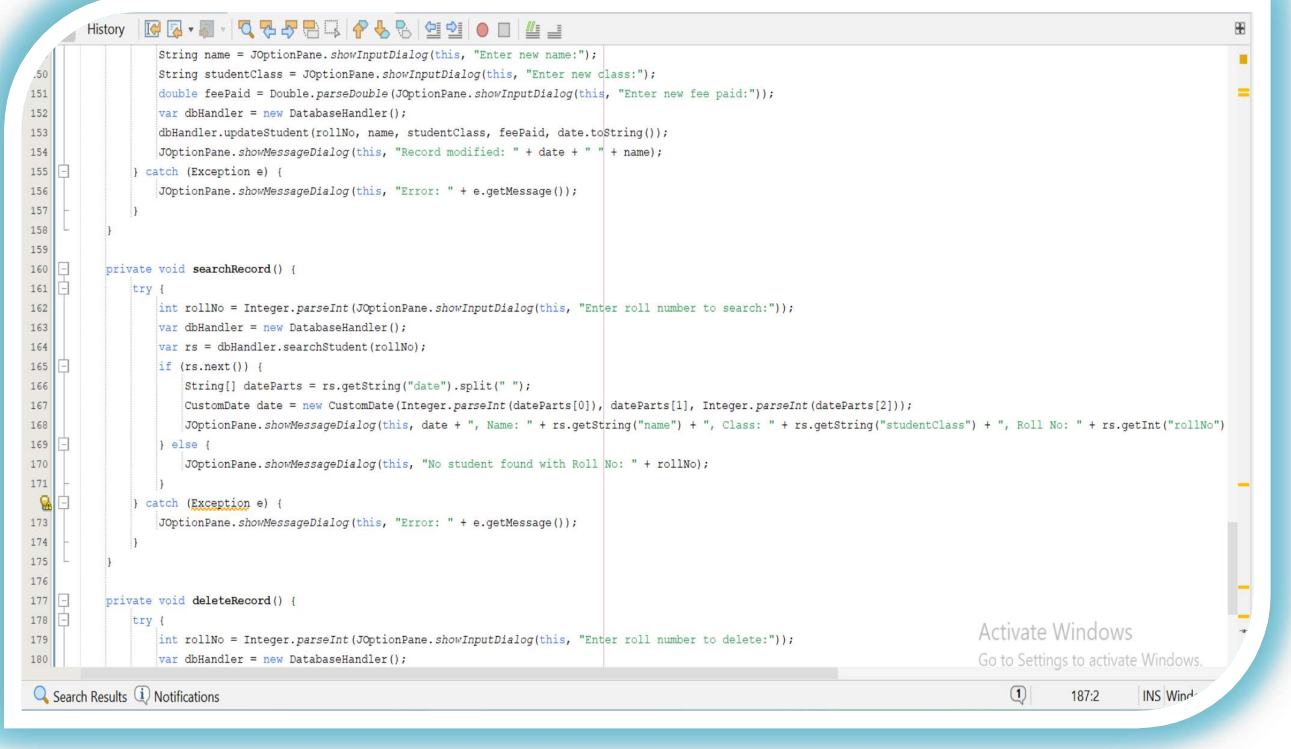
```
59          searchButton = new JButton("Search");
60          searchButton.addActionListener(e -> searchRecord());
61          add(searchButton);
62
63          deleteButton = new JButton("Delete");
64          deleteButton.addActionListener(e -> deleteRecord());
65          add(deleteButton);
66
67          setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
68          setVisible(true);
69      }
70
71      private void saveStudent() {
72          try {
73              int day = Integer.parseInt(dayField.getText().trim());
74              String month = monthField.getText().trim();
75              int year = Integer.parseInt(yearField.getText().trim());
76              CustomDate date = new CustomDate(day, month, year);
77              if (!date.chkDate()) throw new Exception("Invalid date!");
78
79              String name = nameField.getText().trim();
80              String studentClass = classField.getText().trim();
81              int rollNo = Integer.parseInt(rollField.getText().trim());
82              double feePaid = Double.parseDouble(feeField.getText().trim());
83
84              if (name.isEmpty() || studentClass.isEmpty() || rollNo <= 0 || feePaid < 0) {
85                  throw new Exception("Invalid input: All fields must be valid.");
86              }
```



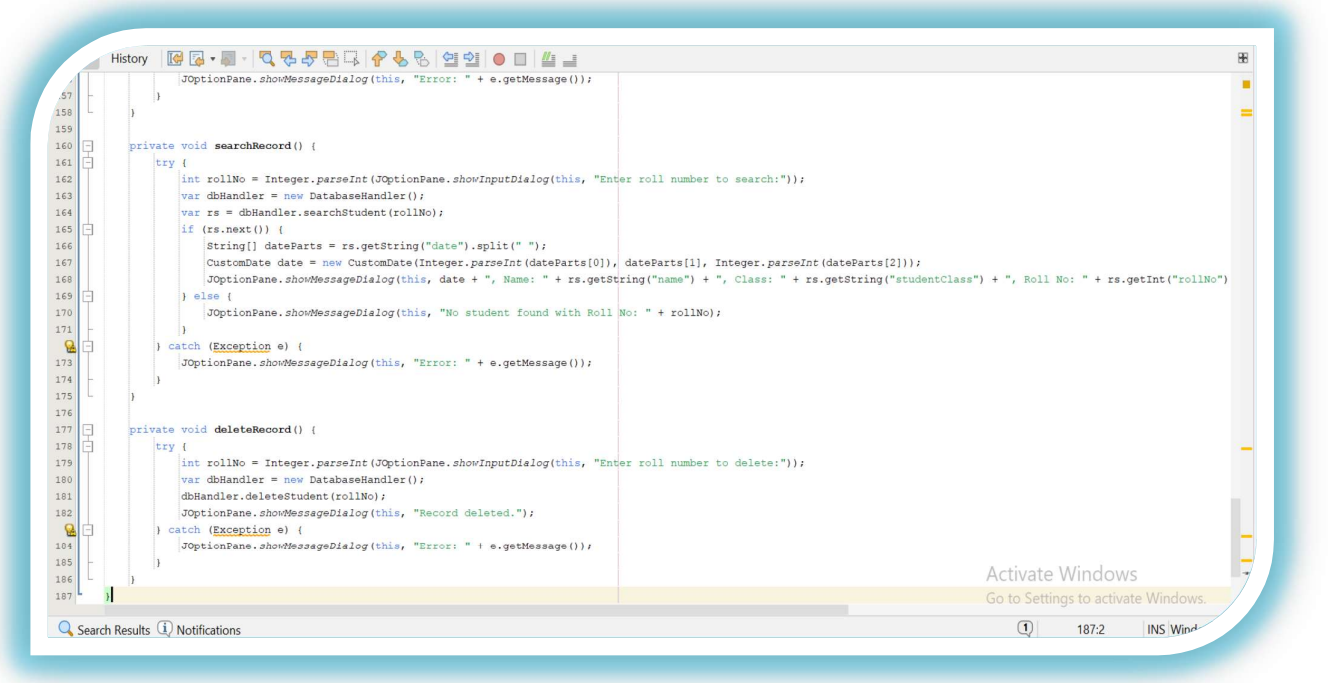
```
88              var dbHandler = new DatabaseHandler();
89              dbHandler.saveStudent(name, studentClass, rollNo, feePaid, date.toString());
90              JOptionPane.showMessageDialog(this, "Student saved: " + date + " " + name);
91              dispose();
92          } catch (Exception e) {
93              JOptionPane.showMessageDialog(this, "Error: " + e.getMessage());
94          }
95      }
96
97      private void viewFee() {
98          try {
99              int rollNo = Integer.parseInt(rollField.getText().trim());
100             double feePaid = Double.parseDouble(feeField.getText().trim());
101             int day = Integer.parseInt(dayField.getText().trim());
102             String month = monthField.getText().trim();
103             int year = Integer.parseInt(yearField.getText().trim());
104             CustomDate date = new CustomDate(day, month, year);
105             if (!date.chkDate()) throw new Exception("Invalid date!");
106
107             Student student = new Student("", "", rollNo, feePaid, date);
108             student.viewFee();
109         } catch (Exception e) {
110             JOptionPane.showMessageDialog(this, "Error: " + e.getMessage());
111         }
112     }
113
114     private void displayRecordList() {
115         try {
116             var dbHandler = new DatabaseHandler();
117             var rs = dbHandler.getStudents();
```



```
StringBuilder display = new StringBuilder("Student Records:\n");
boolean hasRecords = false;

while (rs.next()) {
    hasRecords = true;
    String[] dateParts = rs.getString("date").split(" ");
    CustomDate date = new CustomDate(Integer.parseInt(dateParts[0]), dateParts[1], Integer.parseInt(dateParts[2]));
    display.append(date)
           .append(", Name: ").append(rs.getString("name"))
           .append(", Class: ").append(rs.getString("studentClass"))
           .append(", Roll No: ").append(rs.getInt("rollNo"))
           .append(", Fee Paid: ").append(rs.getDouble("feePaid"))
           .append("\n");
}

if (!hasRecords) display.append("No records found.");
JOptionPane.showMessageDialog(this, display.toString());
} catch (Exception e) {
    JOptionPane.showMessageDialog(this, "Error: " + e.getMessage());
}
}

private void modifyRecord() {
    try {
        int rollNo = Integer.parseInt(JOptionPane.showInputDialog(this, "Enter roll number to modify:"));
        int day = Integer.parseInt(JOptionPane.showInputDialog(this, "Enter new day of admission:"));
        String month = JOptionPane.showInputDialog(this, "Enter new month of admission:");
        int year = Integer.parseInt(JOptionPane.showInputDialog(this, "Enter new year of admission:"));
        CustomDate date = new CustomDate(day, month, year);
        if (!date.chkDate()) throw new Exception("Invalid date!");
```



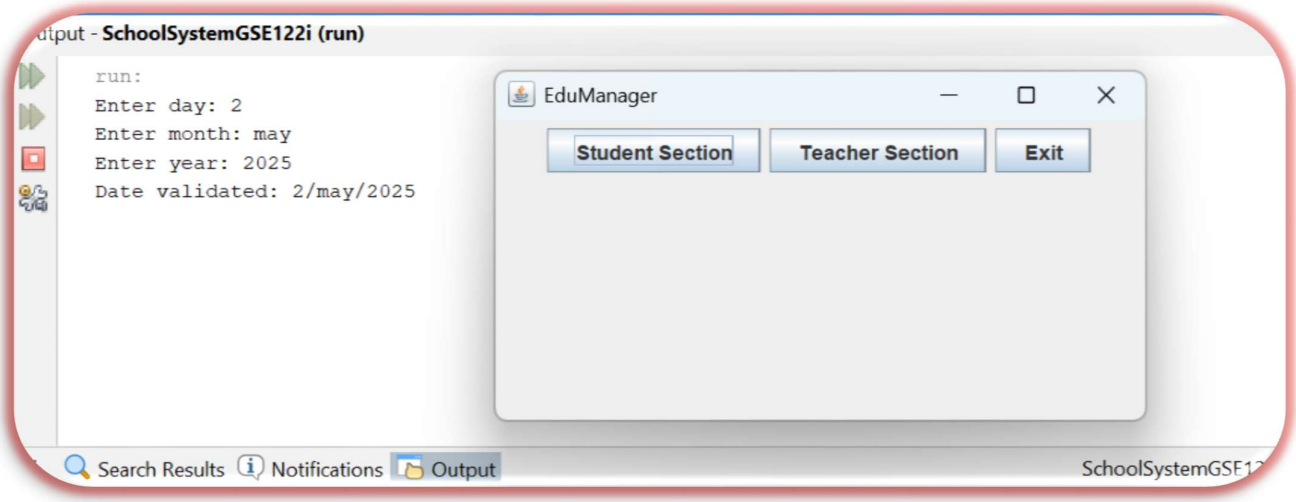
```
String name = JOptionPane.showInputDialog(this, "Enter new name:");
String studentClass = JOptionPane.showInputDialog(this, "Enter new class:");
double feePaid = Double.parseDouble(JOptionPane.showInputDialog(this, "Enter new fee paid:"));
var dbHandler = new DatabaseHandler();
dbHandler.updateStudent(rollNo, name, studentClass, feePaid, date.toString());
JOptionPane.showMessageDialog(this, "Record modified: " + date + " " + name);
} catch (Exception e) {
    JOptionPane.showMessageDialog(this, "Error: " + e.getMessage());
}
}

private void searchRecord() {
    try {
        int rollNo = Integer.parseInt(JOptionPane.showInputDialog(this, "Enter roll number to search:"));
        var dbHandler = new DatabaseHandler();
        var rs = dbHandler.searchStudent(rollNo);
        if (rs.next()) {
            String[] dateParts = rs.getString("date").split(" ");
            CustomDate date = new CustomDate(Integer.parseInt(dateParts[0]), dateParts[1], Integer.parseInt(dateParts[2]));
            JOptionPane.showMessageDialog(this, date + ", Name: " + rs.getString("name") + ", Class: " + rs.getString("studentClass") + ", Roll No: " + rs.getInt("rollNo")
        } else {
            JOptionPane.showMessageDialog(this, "No student found with Roll No: " + rollNo);
        }
    } catch (Exception e) {
        JOptionPane.showMessageDialog(this, "Error: " + e.getMessage());
    }
}

private void deleteRecord() {
    try {
        int rollNo = Integer.parseInt(JOptionPane.showInputDialog(this, "Enter roll number to delete:"));
        var dbHandler = new DatabaseHandler();
```



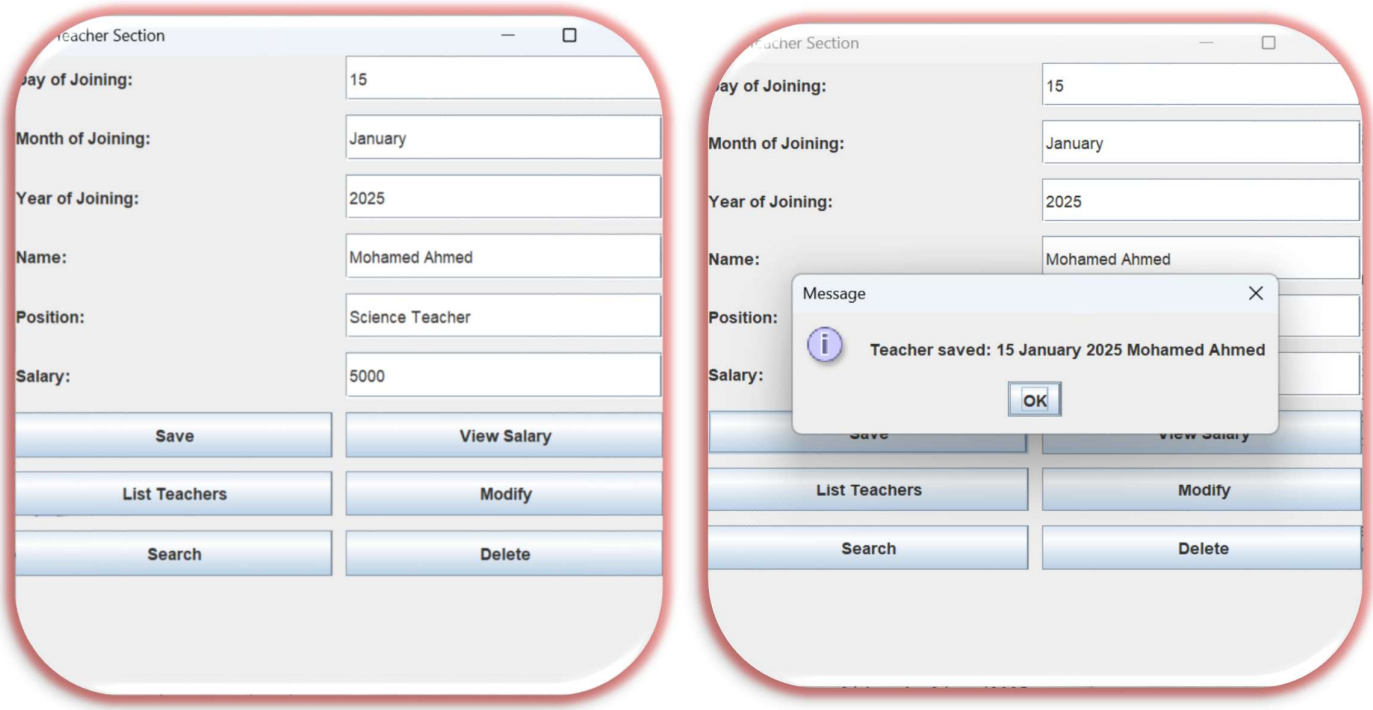
7. Output and results

The following outputs were derived from testing the EduManager system across various use cases, with detailed explanations:

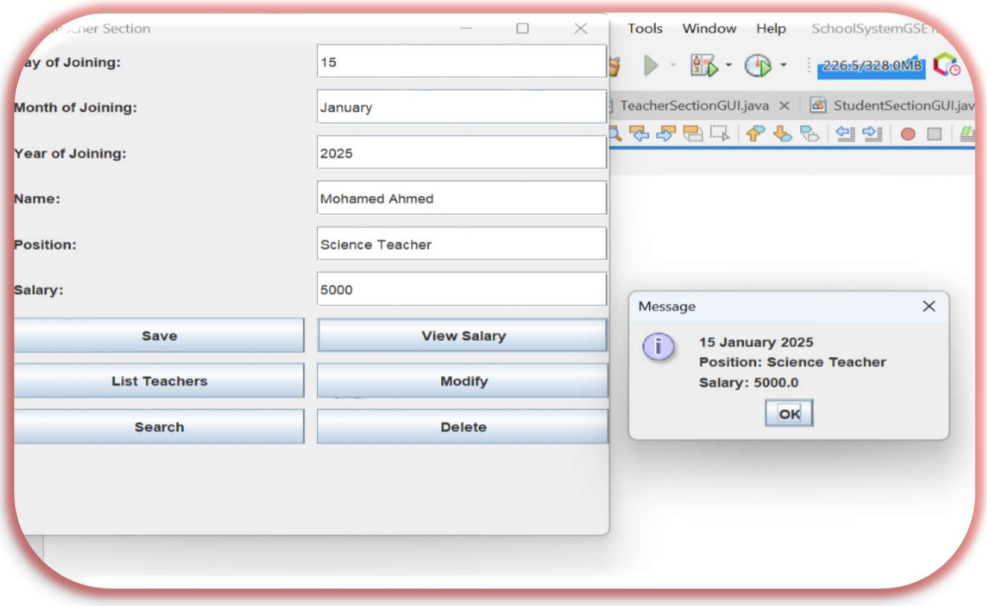
- **Main Menu (**SchoolGUI**):**



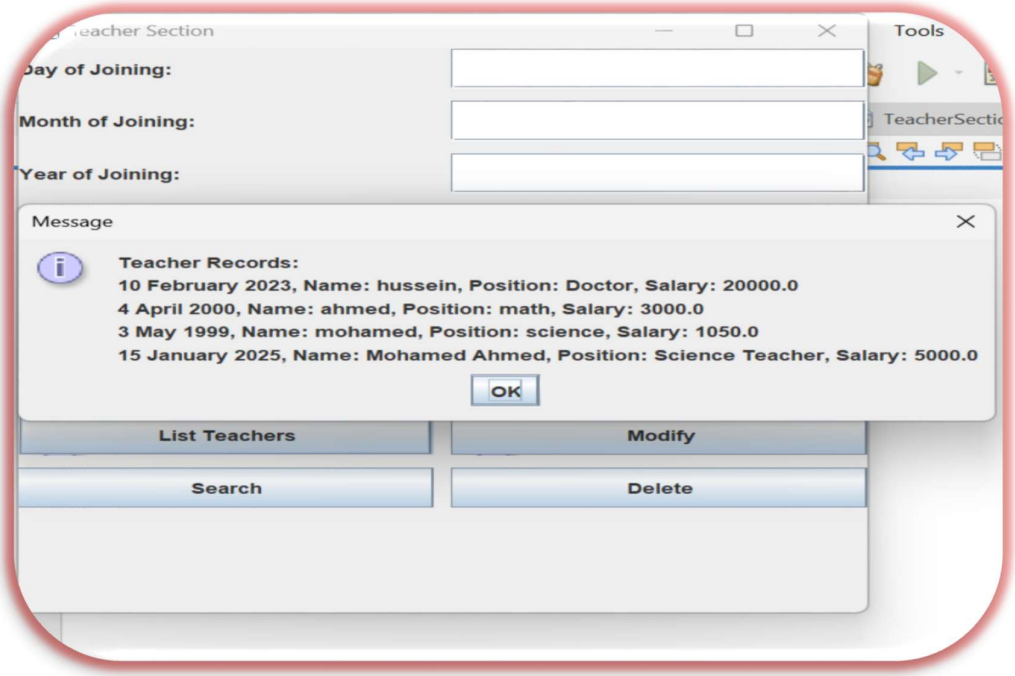
- Teacher Section (TeacherSectionGUI):
  - Save Operation:



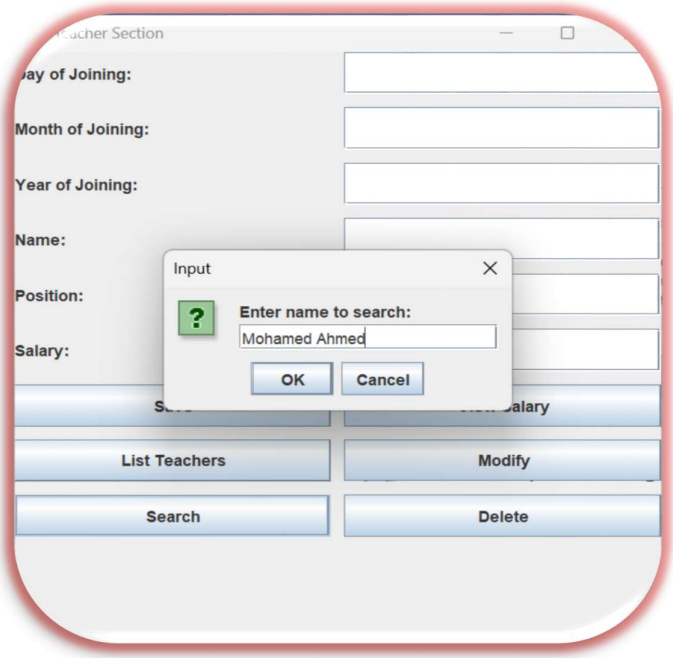
  - View Salary Operation:

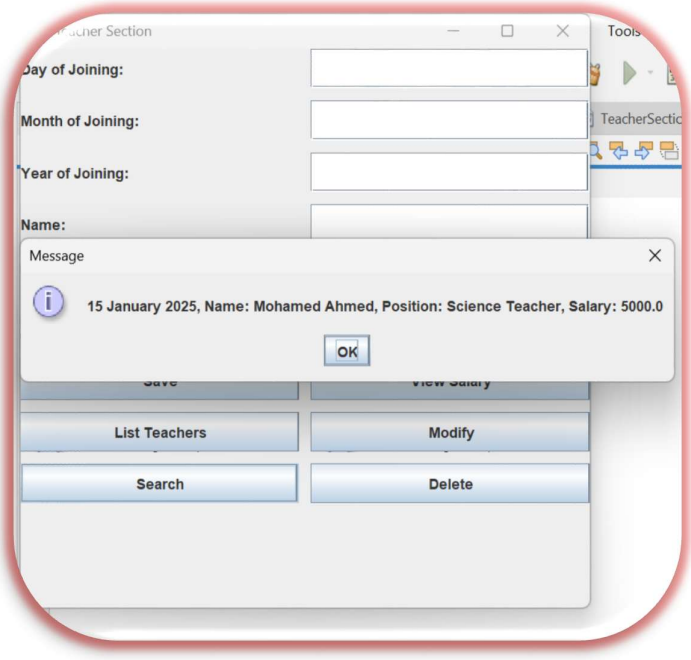


- List Teachers Operation:

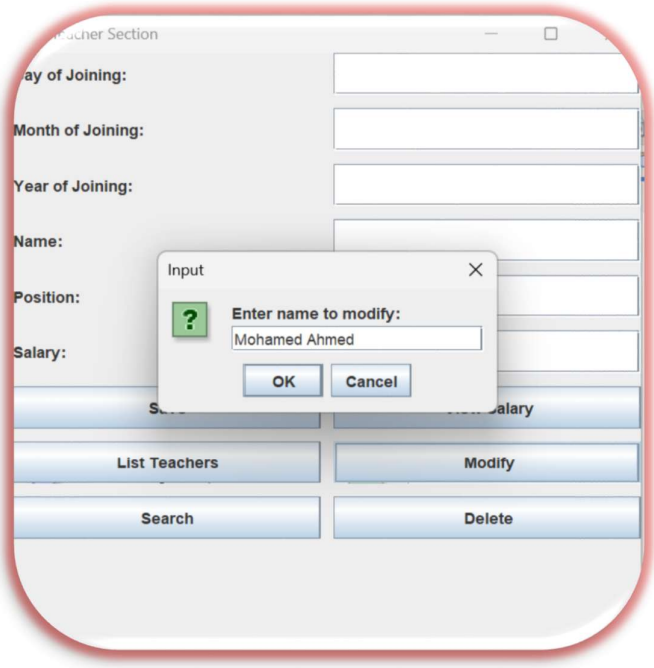


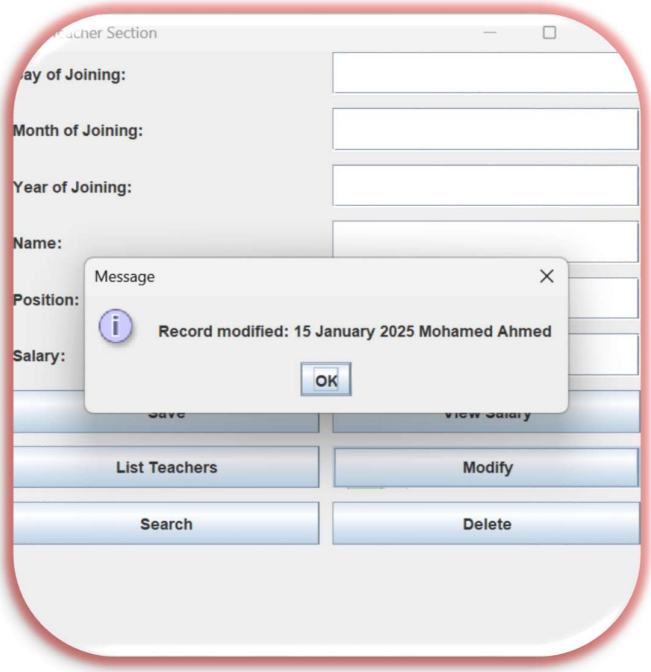
- Search Operation:



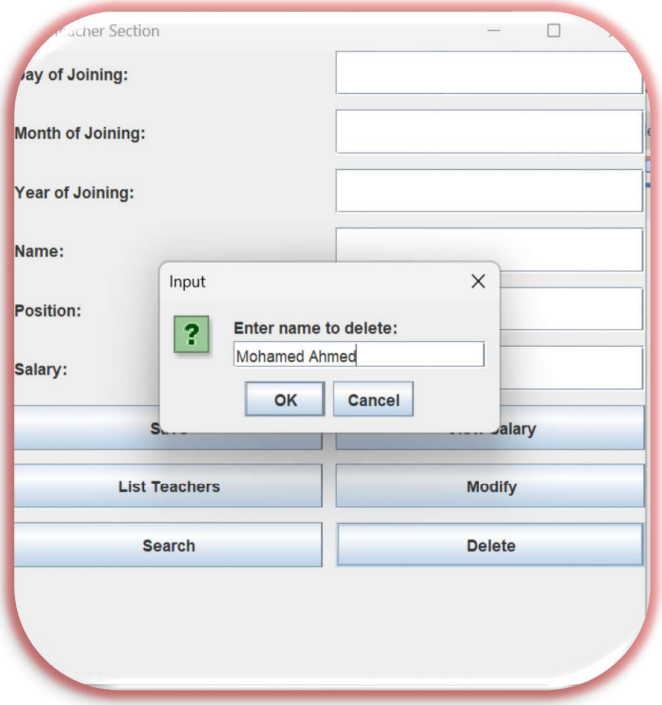


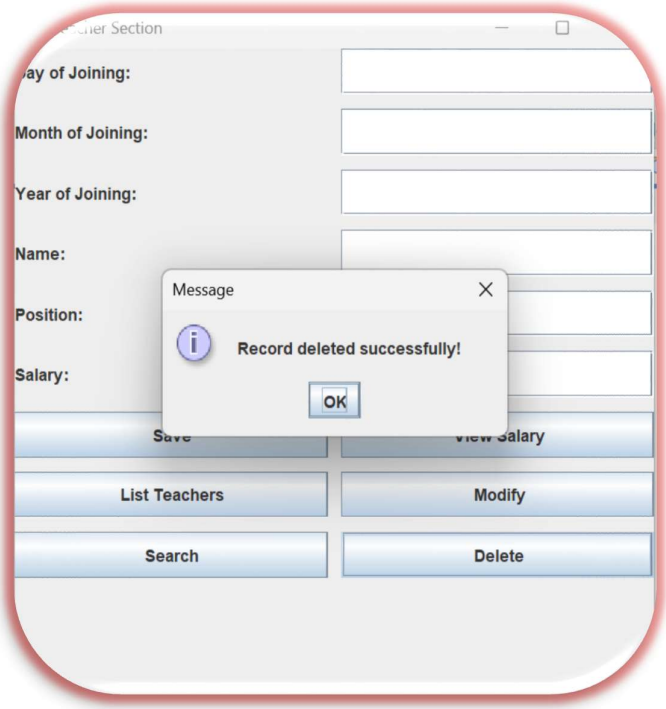
- Modify Operation:





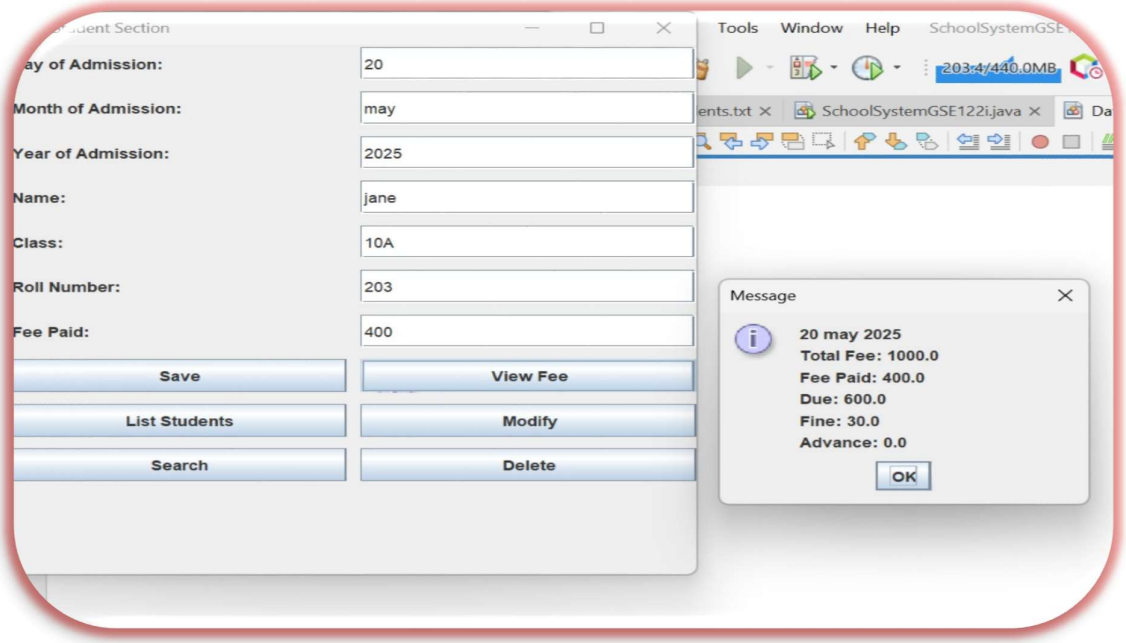
- Delete Operations:



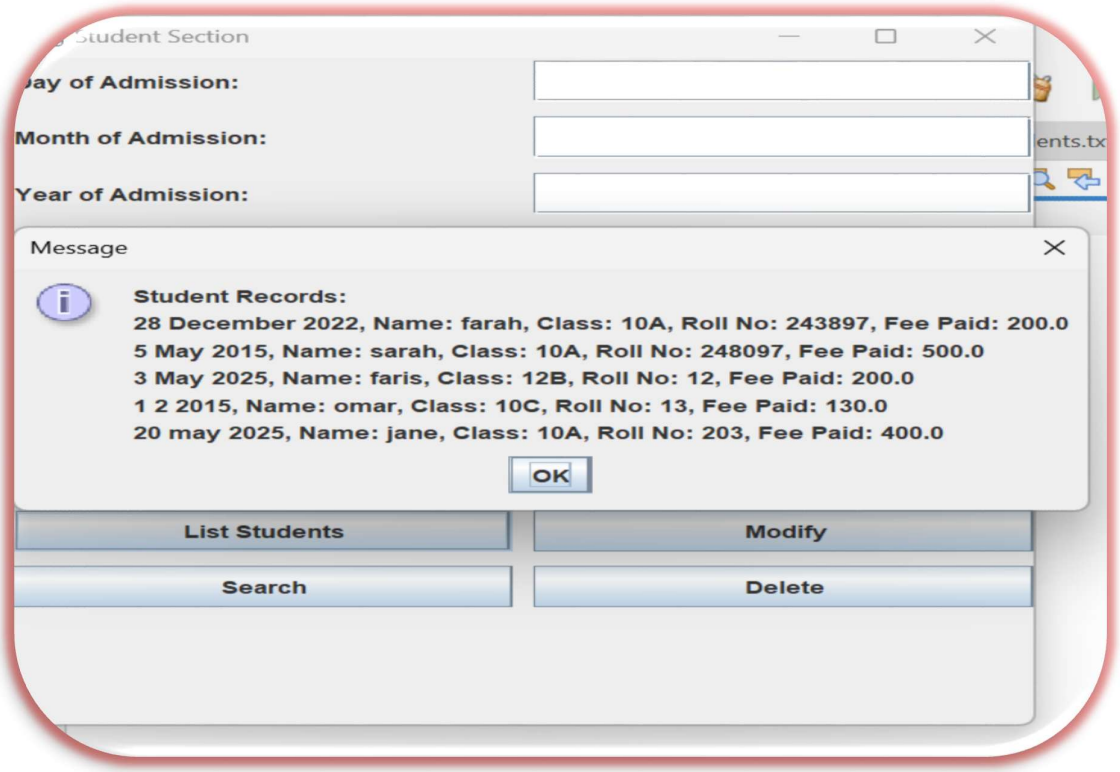


- **Student Section (**StudentSectionGUI**):**
  - **Save Operation:**

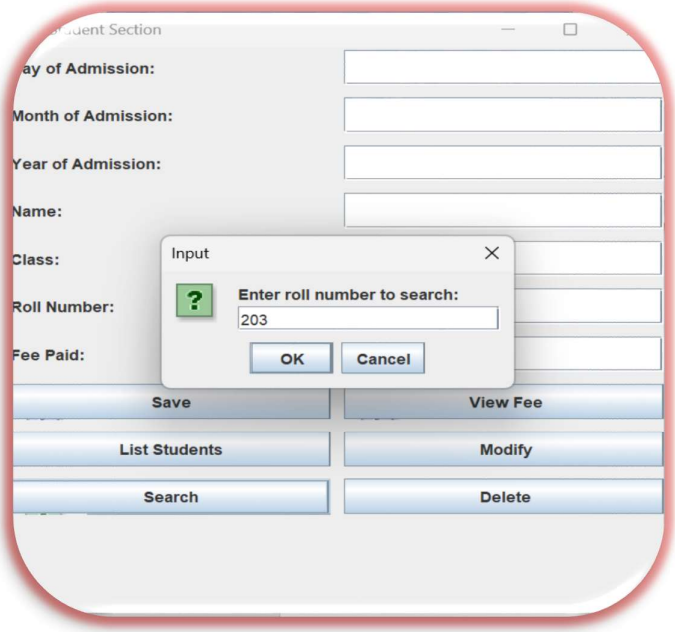
  - View Fee Operation:

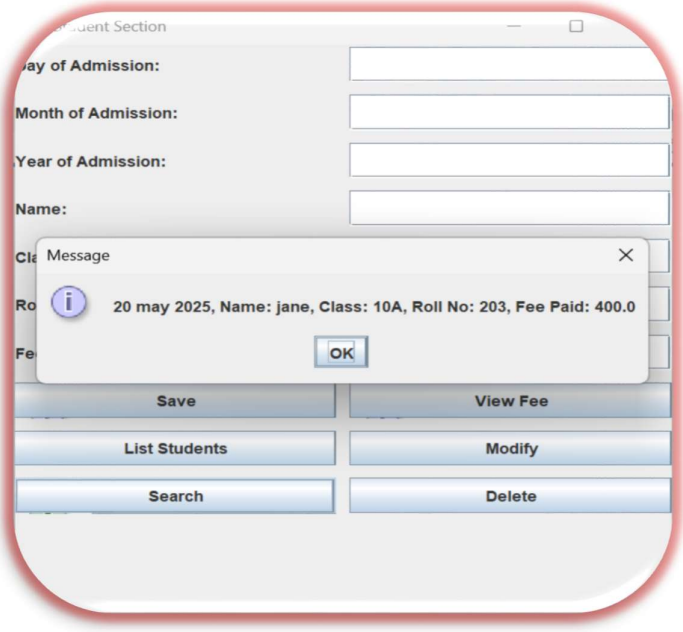


- List Students Operation:

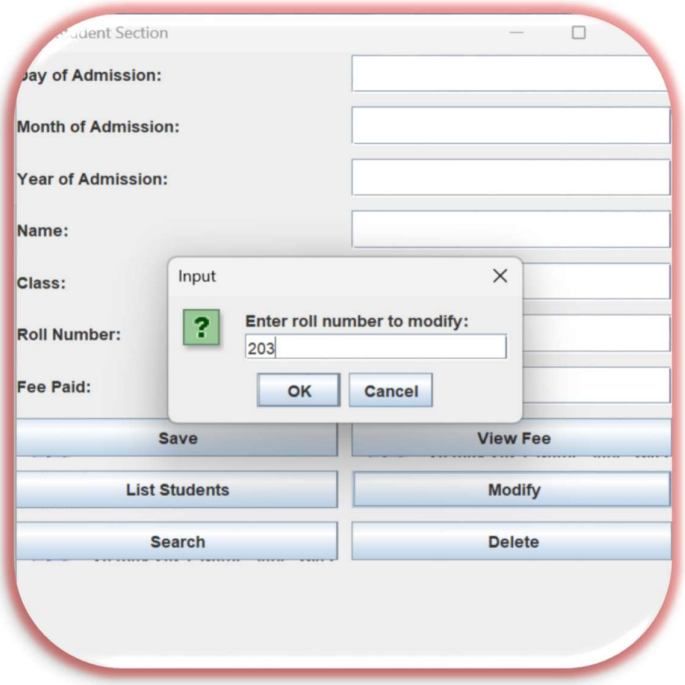


- Search Operation:

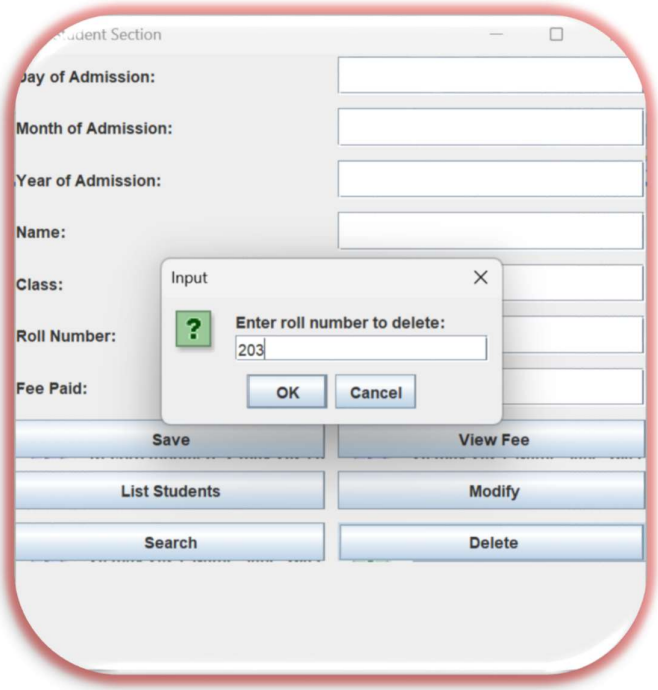


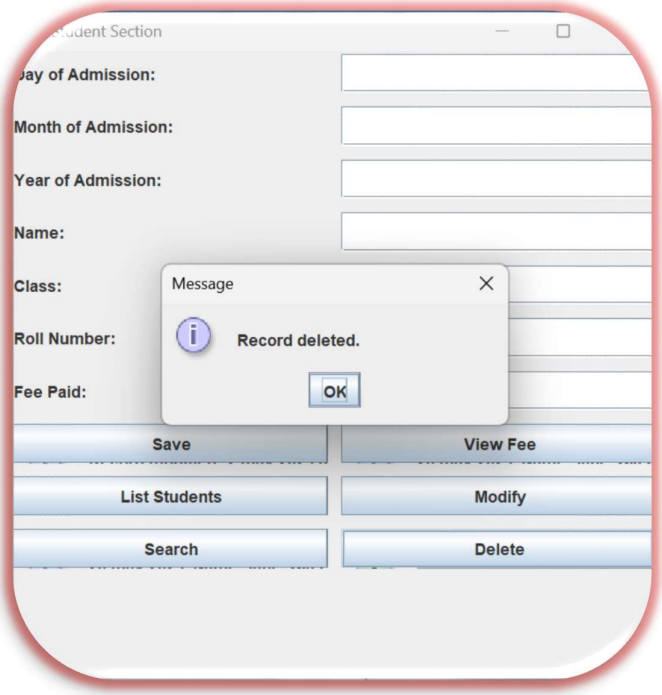


- Modify Operation:



- Delete Operations:

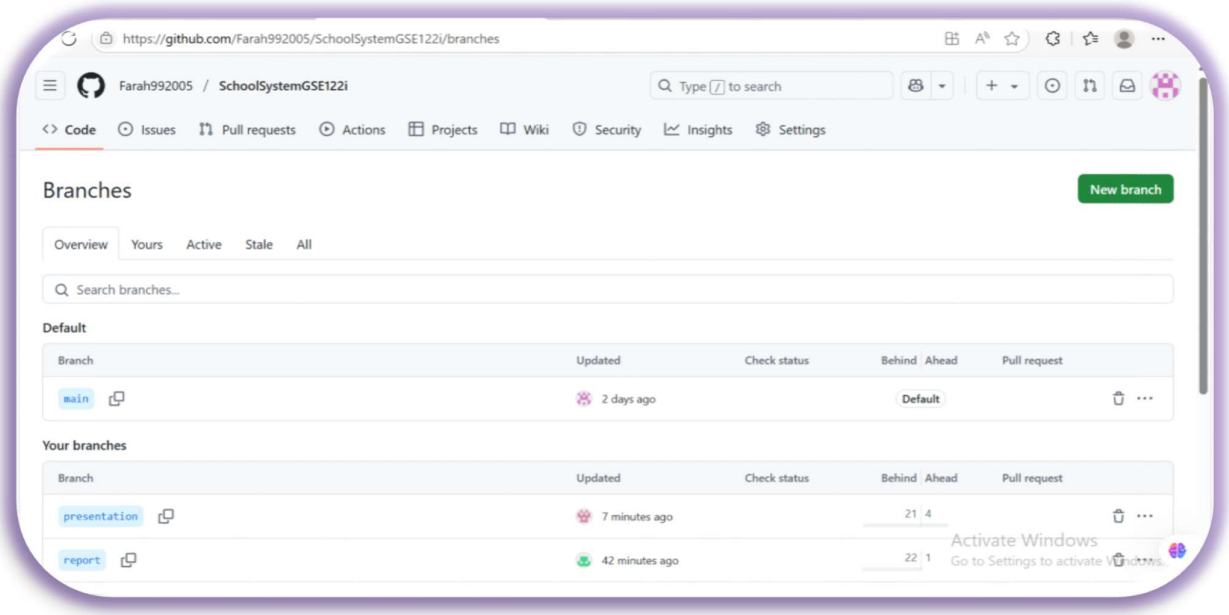




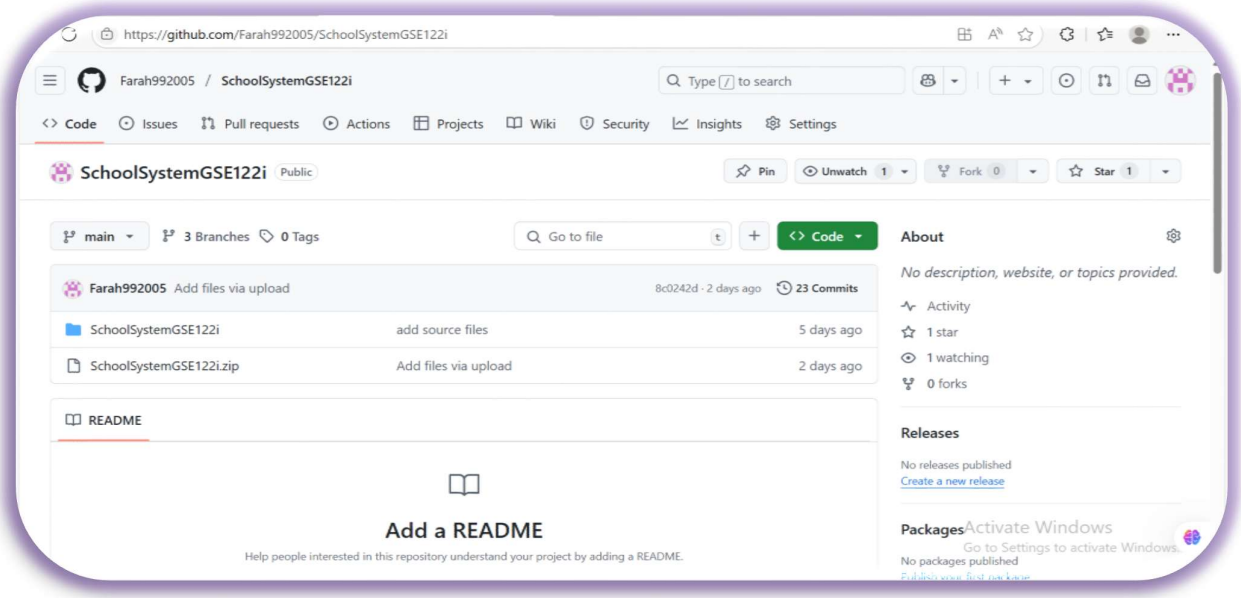
8. GitHub

Repository URL: https://github.com/Farah992005/SchoolSystemGSE122i

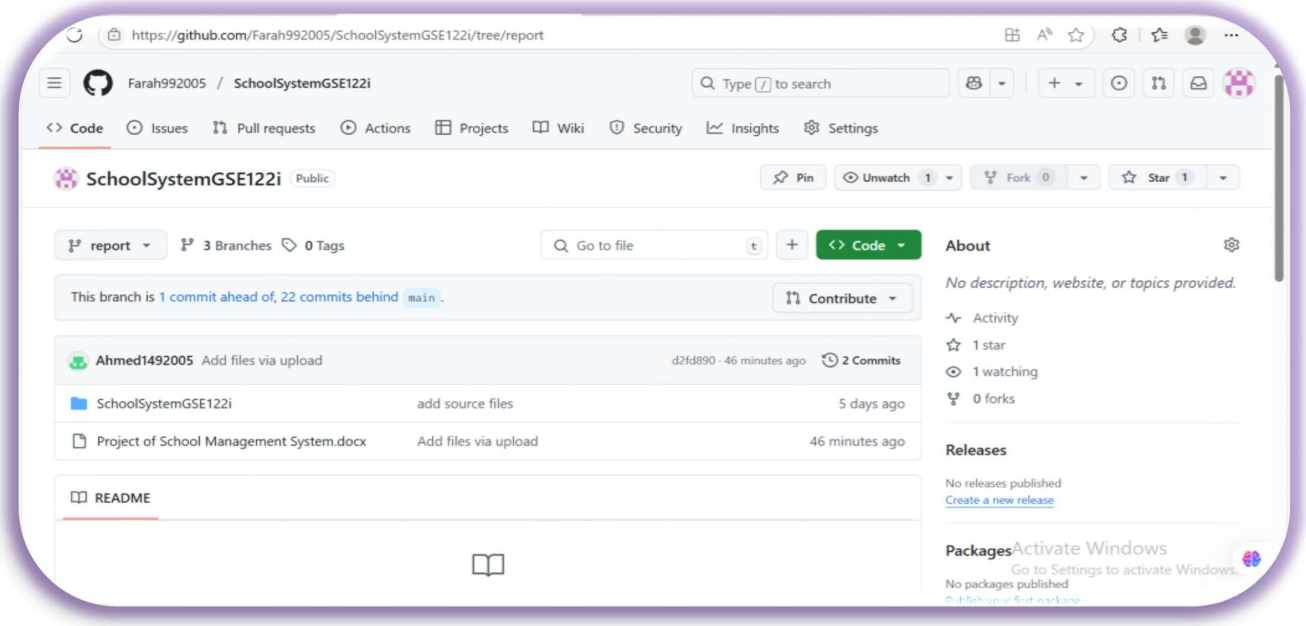
3 Branches:



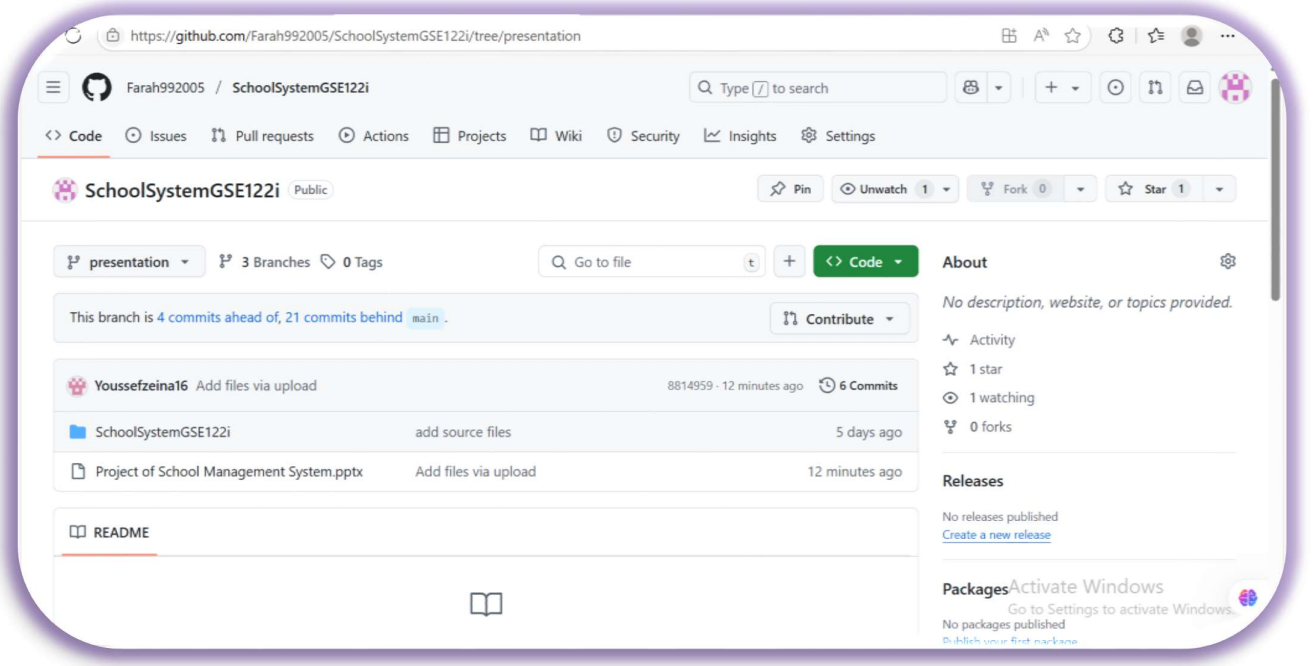
Main Branch:



Report Branch:



Presentation Branch:



9. References

Bracha, G., Odersky, M., Stoutamire, D., & Wadler, P. (1998). Making the future safe for the past: Adding genericity to the Java programming language. Acm sigplan notices, 33(10), 183-200.

Finlayson, M. (2014, January). Java libraries for accessing the princeton wordnet: Comparison and evaluation. In Proceedings of the Seventh Global Wordnet Conference (pp. 78-85).

Joy, B., Steele, G., Gosling, J., & Bracha, G. (2000). The Java language specification.

MySQL. (2025). *MySQL JDBC Driver Documentation*. Retrieved from https://dev.mysql.com/doc/connector-j/8.0/en/

W3Schools. (2025). *SQL Tutorial*. Retrieved from https://www.w3schools.com/sql/