



**CSE Department – Faculty of Engineering - MSA**

**Spring 2025**

**GSE122 GSE122i COM265 PROGRAMMING 2**

**Course Project**

**Course Instructor:Dr. Ahmed El Anany**

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

# **School Management System**



## Table of Contents

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



## 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:

- o **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.

- o **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.



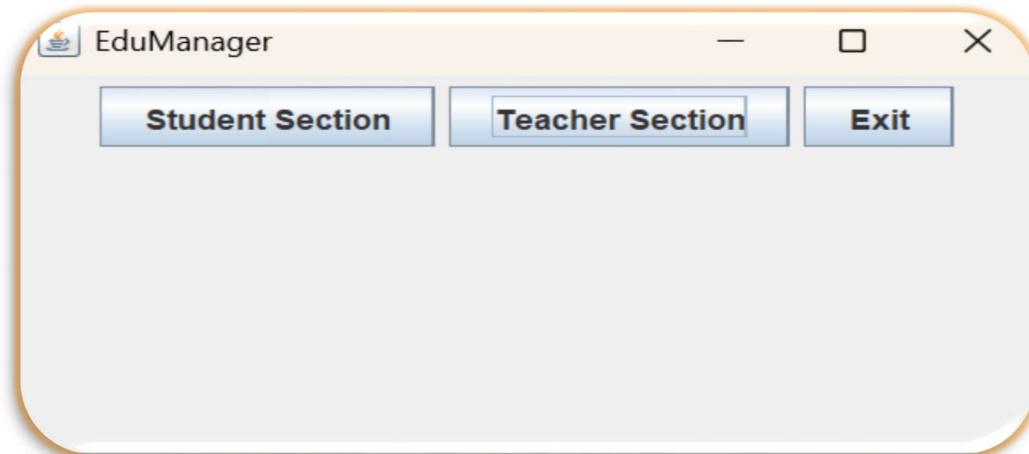
## 5. GUI and Database Usage

### 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:**

Teacher Section	
<b>Day of Joining:</b>	15
<b>Month of Joining:</b>	January
<b>Year of Joining:</b>	2025
<b>Name:</b>	Mohamed Ahmed
<b>Position:</b>	science teacher
<b>Salary:</b>	5000
<b>Buttons:</b>	<input type="button" value="Save"/> <input type="button" value="View Salary"/> <input type="button" value="List Teachers"/> <input type="button" value="Modify"/> <input type="button" value="Search"/> <input type="button" value="Delete"/>

### 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:**



Student Section

<b>Day of Admission:</b>	10
<b>Month of Admission:</b>	February
<b>Year of Admission:</b>	2025
<b>Name:</b>	Jane Khalad
<b>Class:</b>	10A
<b>Roll Number:</b>	101
<b>Fee Paid:</b>	800

## 5.2 Database Usage

- Database Designation:** The system utilizes the `school_dp` database, hosted on a local MySQL server.
- Table Structures:**
  - 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:**

The screenshot shows the 'Table structure' view for the 'student' table. The table has five columns:

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action		
1	<b>name</b>	varchar(100)	utf8mb4_general_ci		Yes	NULL	Student's name				
2	<b>studentClass</b>	varchar(50)	utf8mb4_general_ci		Yes	NULL	Class (e.g., 10A)				
3	<b>rollNo</b>	int(11)			No	None	Roll number (PK)				
4	<b>feePaid</b>	double			Yes	NULL	Fee paid amount				
5	<b>date</b>	varchar(50)	utf8mb4_general_ci		Yes	NULL					

Below the table structure, there is a section for 'Indexes' which states 'No index defined!'. At the bottom, there is a button to 'Create an index on 1 columns Go'.

## 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:**

The screenshot shows the 'Table structure' view for the 'teacher' table. The table has four columns:

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action		
1	<b>name</b>	varchar(100)	utf8mb4_general_ci		No	None	Teacher's name (PK)				
2	<b>Position</b>	varchar(50)	utf8mb4_general_ci		Yes	NULL	Position				
3	<b>salary</b>	double			Yes	NULL	Salary amount				
4	<b>date</b>	varchar(50)	utf8mb4_general_ci		Yes	NULL					

Below the table structure, there is a section for 'Indexes' which states 'No index defined!'. At the bottom, there is a button to 'Create an index on 1 columns Go'.



- **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).

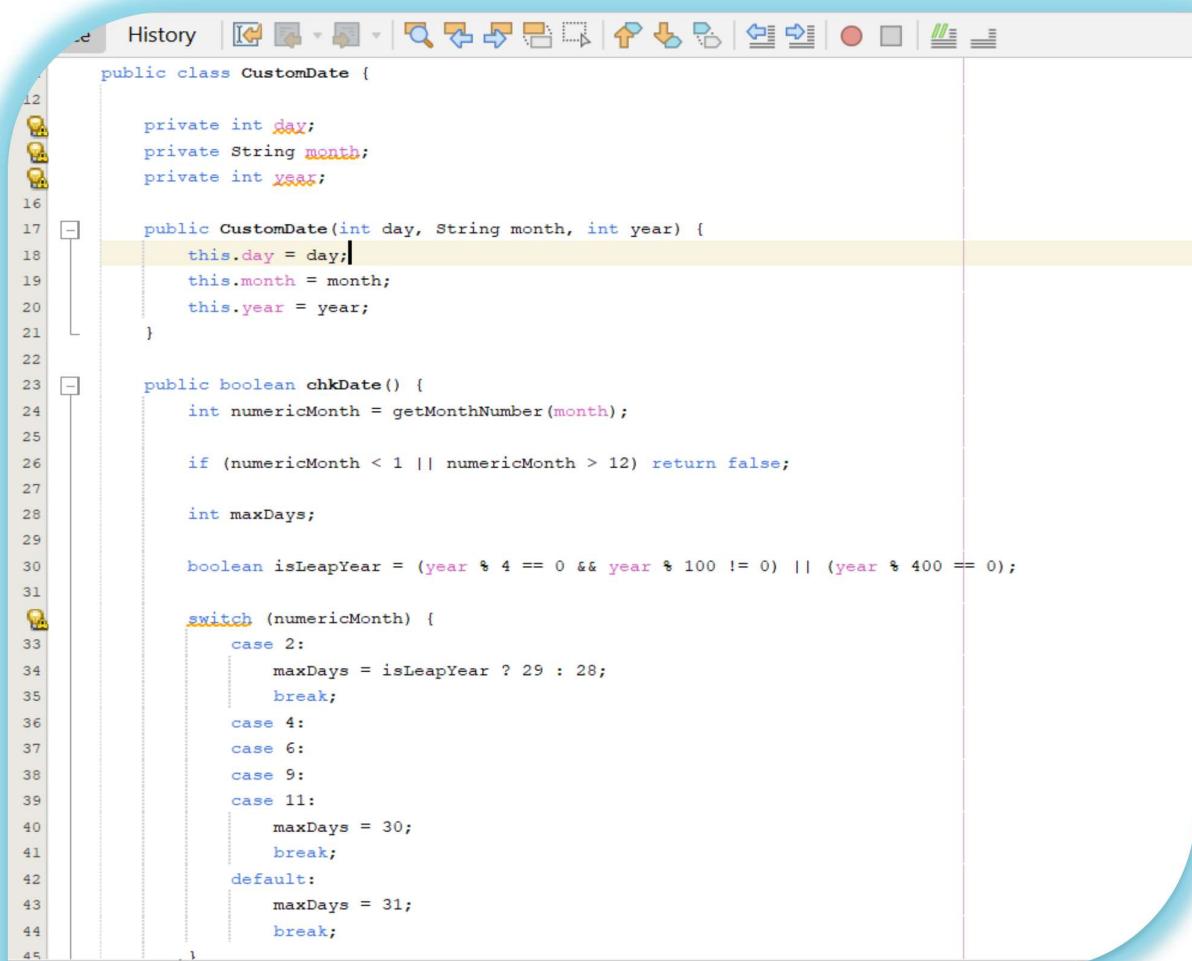


- If parsing fails (i.e., a `NumberFormatException` is thrown), converts `monthInput` to lowercase and matches it against predefined month names (e.g., "january" maps to 1, "february" to 2, etc.).
- Returns -1 for unrecognized inputs, indicating an invalid month.
- **Illustrative Example:** An input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



```

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);

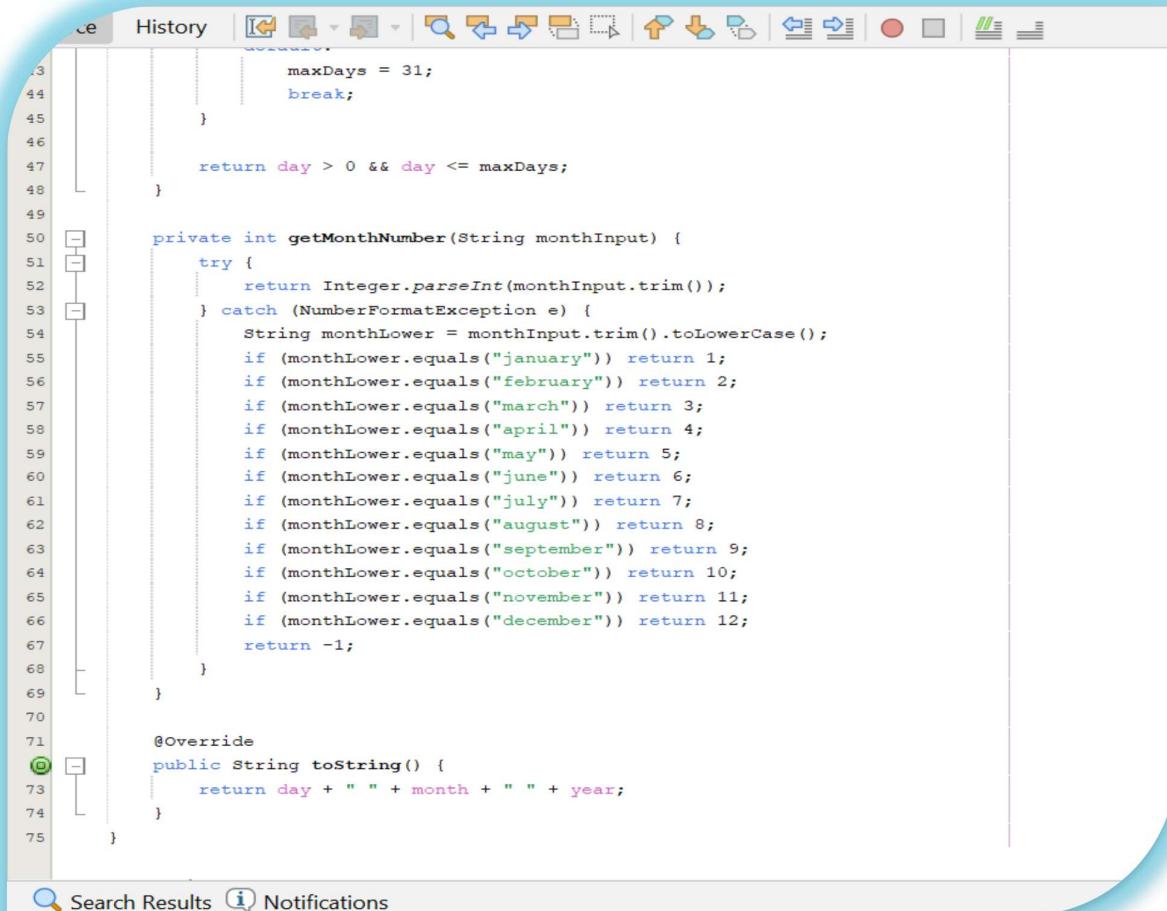
        if (numericMonth < 1 || numericMonth > 12) return false;

        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;
        }
    }
}

```

```

1    package com.example;
2
3    public class Date {
4
5        private int day;
6        private int month;
7        private int year;
8
9        public Date(int day, int month, int year) {
10            this.day = day;
11            this.month = month;
12            this.year = year;
13        }
14
15        public int getDay() {
16            return day;
17        }
18
19        public void setDay(int day) {
20            this.day = day;
21        }
22
23        public int getMonth() {
24            return month;
25        }
26
27        public void setMonth(int month) {
28            this.month = month;
29        }
30
31        public int getYear() {
32            return year;
33        }
34
35        public void setYear(int year) {
36            this.year = year;
37        }
38
39        public boolean isLeapYear() {
40            if (year % 400 == 0) {
41                return true;
42            } else if (year % 100 == 0) {
43                return false;
44            } else if (year % 4 == 0) {
45                return true;
46            } else {
47                return false;
48            }
49        }
50
51        public int getMaxDaysInMonth() {
52            int maxDays = 31;
53            if (month == 2) {
54                maxDays = 28;
55                if (isLeapYear()) {
56                    maxDays = 29;
57                }
58            }
59            return day > 0 && day <= maxDays;
60        }
61
62        private int getMonthNumber(String monthInput) {
63            try {
64                return Integer.parseInt(monthInput.trim());
65            } catch (NumberFormatException e) {
66                String monthLower = monthInput.trim().toLowerCase();
67                if (monthLower.equals("january")) return 1;
68                if (monthLower.equals("february")) return 2;
69                if (monthLower.equals("march")) return 3;
70                if (monthLower.equals("april")) return 4;
71                if (monthLower.equals("may")) return 5;
72                if (monthLower.equals("june")) return 6;
73                if (monthLower.equals("july")) return 7;
74                if (monthLower.equals("august")) return 8;
75                if (monthLower.equals("september")) return 9;
76                if (monthLower.equals("october")) return 10;
77                if (monthLower.equals("november")) return 11;
78                if (monthLower.equals("december")) return 12;
79                return -1;
80            }
81        }
82
83        @Override
84        public String toString() {
85            return day + " " + month + " " + year;
86        }
87    }

```

Search Results   Notifications

## 6.2 DatabaseHandler

- Purpose:** Serves as the central interface for all database interactions, managing connectivity and CRUD operations.
- Detailed Procedural Analysis:**
  - 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.

```
public class DatabaseHandler {
    private static final String URL = "jdbc:mysql://localhost:3306/type_school_dp";
    private static final String USER = "root";
    private static final String PASSWORD = "";

    private Connection conn;

    public DatabaseHandler() {
        try {
            Class.forName("com.mysql.jdbc.Driver");
            conn = DriverManager.getConnection(URL, USER, PASSWORD);
        } catch (ClassNotFoundException e) {
            System.err.println("MySQL JDBC Driver not found: " + e.getMessage());
            throw new RuntimeException("Driver initialization failed.");
        } catch (SQLException e) {
            System.err.println("Database connection failed: " + e.getMessage());
            throw new RuntimeException("Database initialization failed.");
        }
    }

    public void saveStudent(String name, String studentClass, int rollNo, double feePaid, String date) throws SQLException {
        String query = "INSERT INTO student (name, studentClass, rollNo, feePaid, date) VALUES (?, ?, ?, ?, ?)";
        try (PreparedStatement stmt = conn.prepareStatement(query)) {
            stmt.setString(1, name);
            stmt.setString(2, studentClass);
            stmt.setInt(3, rollNo);
            stmt.setDouble(4, feePaid);
            stmt.setString(5, date);
            stmt.executeUpdate();
        }
    }
}
```



```

37  public void saveTeacher(String name, String position, double salary, String date) throws SQLException {
38      String query = "INSERT INTO teacher (name, position, salary, date) VALUES (?, ?, ?, ?)";
39      try (PreparedStatement stmt = conn.prepareStatement(query)) {
40          stmt.setString(1, name);
41          stmt.setString(2, position);
42          stmt.setDouble(3, salary);
43          stmt.setString(4, date);
44          stmt.executeUpdate();
45      }
46  }
47
48  public ResultSet getStudents() throws SQLException {
49      String query = "SELECT * FROM student";
50      return conn.createStatement().executeQuery(query);
51  }
52
53  public ResultSet getTeachers() throws SQLException {
54      String query = "SELECT * FROM teacher";
55      return conn.createStatement().executeQuery(query);
56  }
57
58  public void updateStudent(int rollNo, String name, String studentClass, double feePaid, String date) throws SQLException {
59      String query = "UPDATE student SET name = ?, studentClass = ?, feePaid = ?, date = ? WHERE rollNo = ?";
60      try (PreparedStatement stmt = conn.prepareStatement(query)) {
61          stmt.setString(1, name);
62          stmt.setString(2, studentClass);
63          stmt.setDouble(3, feePaid);
64          stmt.setString(4, date);
65          stmt.setInt(5, rollNo);
66          stmt.executeUpdate();
67      }
68  }

```

 Search Results  Notifications

```

70  public void updateTeacher(String name, String position, double salary, String date) throws SQLException {
71      String query = "UPDATE teacher SET position = ?, salary = ?, date = ? WHERE name = ?";
72      try (PreparedStatement stmt = conn.prepareStatement(query)) {
73          stmt.setString(1, position);
74          stmt.setDouble(2, salary);
75          stmt.setString(3, date);
76          stmt.setString(4, name);
77          stmt.executeUpdate();
78      }
79  }
80
81  public void deleteStudent(int rollNo) throws SQLException {
82      String query = "DELETE FROM student WHERE rollNo = ?";
83      try (PreparedStatement stmt = conn.prepareStatement(query)) {
84          stmt.setInt(1, rollNo);
85          stmt.executeUpdate();
86      }
87  }
88
89  public void deleteTeacher(String name) throws SQLException {
90      String query = "DELETE FROM teacher WHERE name = ?";
91      try (PreparedStatement stmt = conn.prepareStatement(query)) {
92          stmt.setString(1, name);
93          stmt.executeUpdate();
94      }
95  }
96
97  public ResultSet searchStudent(int rollNo) throws SQLException {
98      String query = "SELECT * FROM student WHERE rollNo = ?";
99      PreparedStatement stmt = conn.prepareStatement(query);

```

 Search Results  Notifications



```

81     public void deleteStudent(int rollNo) throws SQLException {
82         String query = "DELETE FROM student WHERE rollNo = ?";
83         try (PreparedStatement stmt = conn.prepareStatement(query)) {
84             stmt.setInt(1, rollNo);
85             stmt.executeUpdate();
86         }
87     }
88
89     public void deleteTeacher(String name) throws SQLException {
90         String query = "DELETE FROM teacher WHERE name = ?";
91         try (PreparedStatement stmt = conn.prepareStatement(query)) {
92             stmt.setString(1, name);
93             stmt.executeUpdate();
94         }
95     }
96
97     public ResultSet searchStudent(int rollNo) throws SQLException {
98         String query = "SELECT * FROM student WHERE rollNo = ?";
99         PreparedStatement stmt = conn.prepareStatement(query);
100        stmt.setInt(1, rollNo);
101        return stmt.executeQuery();
102    }
103
104    public ResultSet searchTeacher(String name) throws SQLException {
105        String query = "SELECT * FROM teacher WHERE name = ?";
106        PreparedStatement stmt = conn.prepareStatement(query);
107        stmt.setString(1, name);
108        return stmt.executeQuery();
109    }
110}

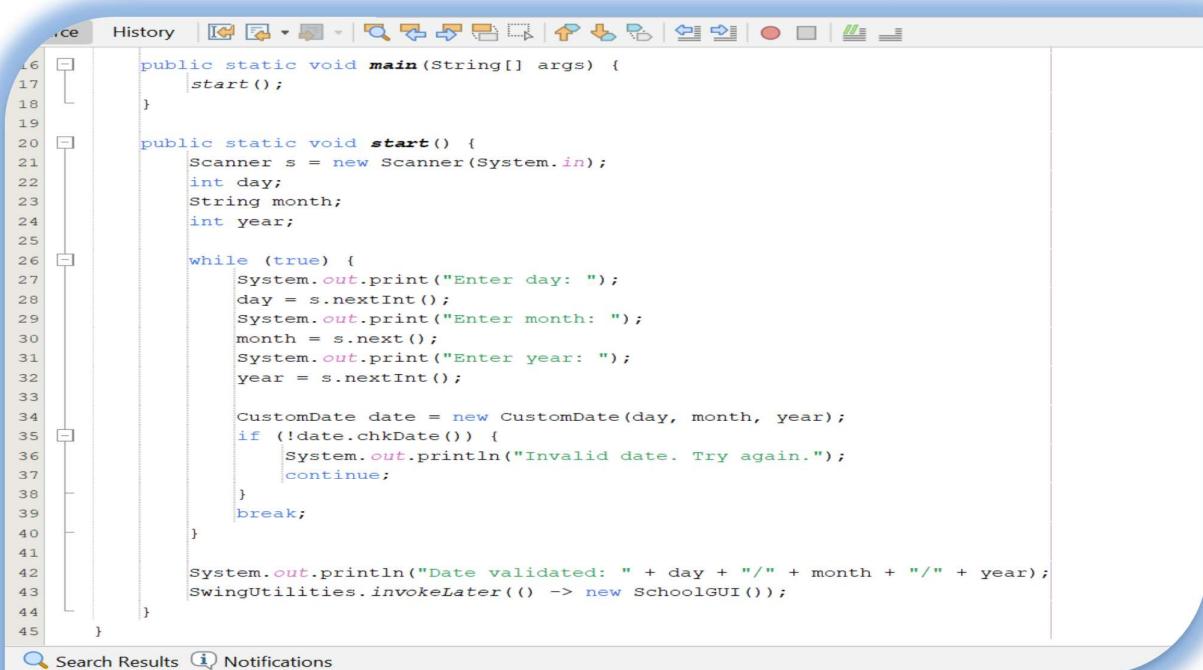
```

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.



```

16 public static void main(String[] args) {
17     start();
18 }
19
20 public static void start() {
21     Scanner s = new Scanner(System.in);
22     int day;
23     String month;
24     int year;
25
26     while (true) {
27         System.out.print("Enter day: ");
28         day = s.nextInt();
29         System.out.print("Enter month: ");
30         month = s.next();
31         System.out.print("Enter year: ");
32         year = s.nextInt();
33
34         CustomDate date = new CustomDate(day, month, year);
35         if (!date.chkDate()) {
36             System.out.println("Invalid date. Try again.");
37             continue;
38         }
39         break;
40     }
41
42     System.out.println("Date validated: " + day + "/" + month + "/" + year);
43     SwingUtilities.invokeLater(() -> new SchoolGUI());
44 }
45 }
```

Search Results Notifications

## 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.

```

11 import javax.swing.*;
12 import java.awt.*;
13
14 public class SchoolGUI extends JFrame {
15     public SchoolGUI() {
16         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 }

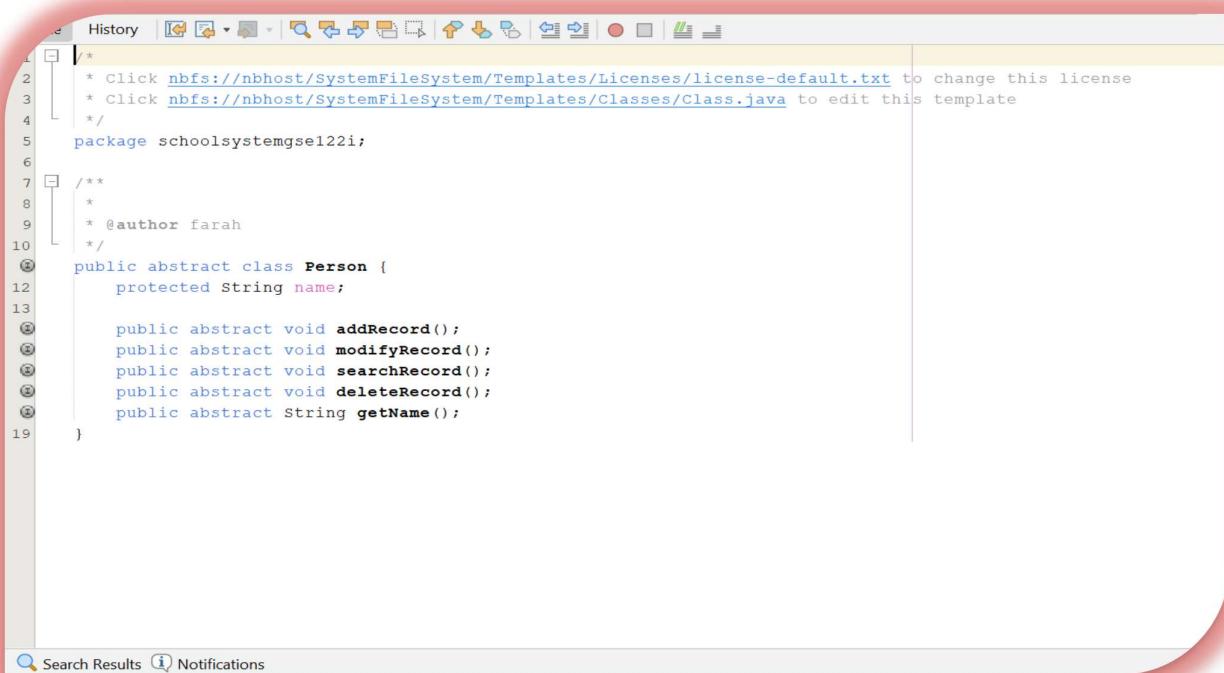
```

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.



```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package schoolsystemgse122i;

/**
 *
 * @author farah
 */
public abstract class Person {
    protected String name;

    public abstract void addRecord();
    public abstract void modifyRecord();
    public abstract void searchRecord();
    public abstract void deleteRecord();
    public abstract String getName();
}
```



## 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.



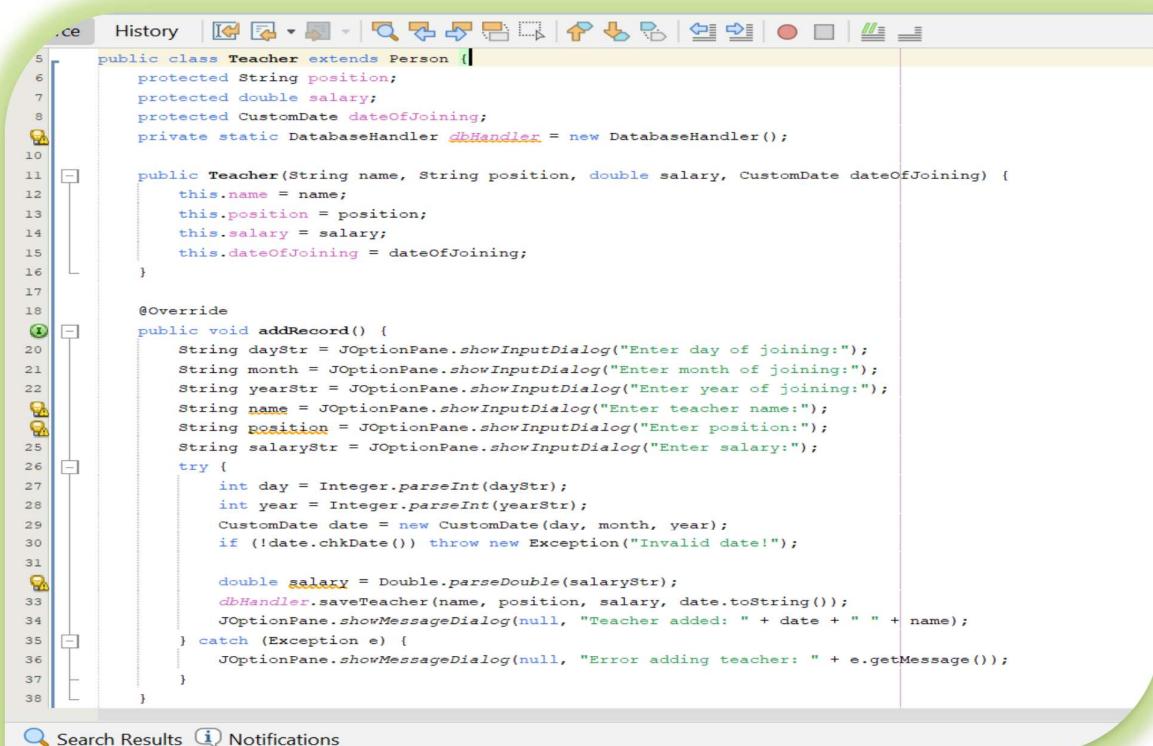
#### 6. `getName()` Method:

- 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.



```

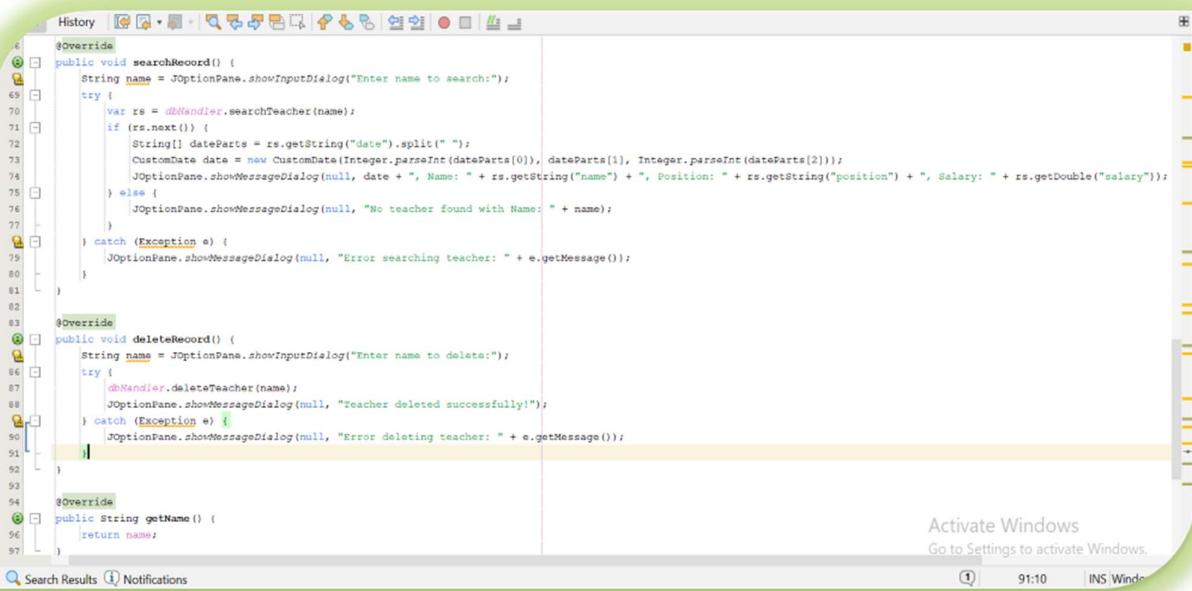
5  public class Teacher extends Person {
6      protected String position;
7      protected double salary;
8      protected CustomDate dateOfJoining;
9      private static DatabaseHandler dbHandler = new DatabaseHandler();
10
11     public Teacher(String name, String position, double salary, CustomDate dateOfJoining) {
12         this.name = name;
13         this.position = position;
14         this.salary = salary;
15         this.dateOfJoining = dateOfJoining;
16     }
17
18     @Override
19     public void addRecord() {
20         String dayStr = JOptionPane.showInputDialog("Enter day of joining:");
21         String month = JOptionPane.showInputDialog("Enter month of joining:");
22         String yearStr = JOptionPane.showInputDialog("Enter year of joining:");
23         String name = JOptionPane.showInputDialog("Enter teacher name:");
24         String position = JOptionPane.showInputDialog("Enter position:");
25         String salaryStr = JOptionPane.showInputDialog("Enter salary:");
26         try {
27             int day = Integer.parseInt(dayStr);
28             int month = Integer.parseInt(month);
29             int year = Integer.parseInt(yearStr);
30             CustomDate date = new CustomDate(day, month, year);
31             if (!date.chkDate()) throw new Exception("Invalid date!");
32
33             double salary = Double.parseDouble(salaryStr);
34             dbHandler.saveTeacher(name, position, salary, date.toString());
35             JOptionPane.showMessageDialog(null, "Teacher added: " + date + " " + name);
36         } catch (Exception e) {
37             JOptionPane.showMessageDialog(null, "Error adding teacher: " + e.getMessage());
38         }
}

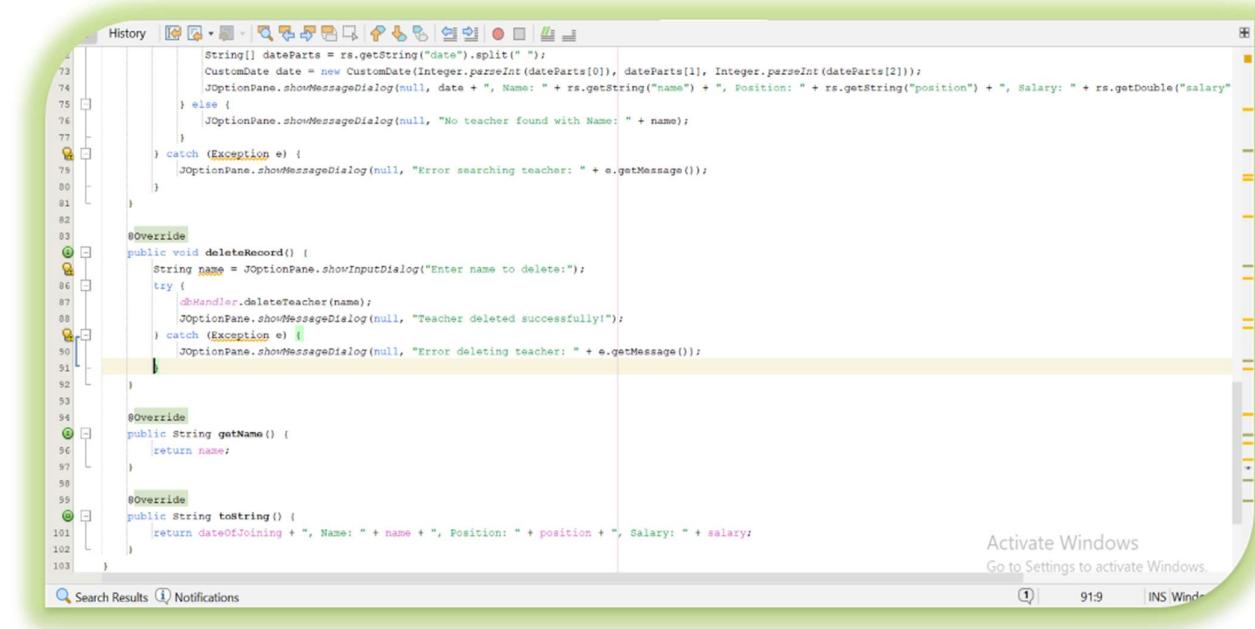
```

Search Results Notifications



```
38
39
40     public void viewSalary() {
41         JOptionPane.showMessageDialog(null, dateOfJoining + "\nPosition: " + position + "\nSalary: " + salary);
42     }
43
44     @Override
45     public void modifyRecord() {
46         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
56             String position = JOptionPane.showInputDialog("Enter new position:");
57             String salaryStr = JOptionPane.showInputDialog("Enter new salary:");
58             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 }
```





```

1 package com.msa.cs;
2
3 import java.awt.GridLayout;
4 import java.awt.event.ActionEvent;
5 import java.awt.event.ActionListener;
6 import java.sql.Connection;
7 import java.sql.DriverManager;
8 import java.sql.PreparedStatement;
9 import java.sql.ResultSet;
10 import java.sql.SQLException;
11 import javax.swing.JButton;
12 import javax.swing.JFrame;
13 import javax.swing.JLabel;
14 import javax.swing.JOptionPane;
15 import javax.swing.JPanel;
16 import javax.swing.JTextField;
17
18 public class TeacherSectionGUI extends JFrame {
19     private JTextField dayField;
20     private JTextField monthField;
21     private JTextField yearField;
22     private JTextField nameField;
23     private JTextField positionField;
24     private JTextField salaryField;
25
26     public TeacherSectionGUI() {
27         setTitle("Teacher Section");
28         setSize(500, 500);
29         setLayout(new GridLayout(11, 2));
30
31         dayField = new JTextField();
32         monthField = new JTextField();
33         yearField = new JTextField();
34         nameField = new JTextField();
35         positionField = new JTextField();
36         salaryField = new JTextField();
37
38         add(new JLabel("Day"));
39         add(dayField);
40         add(new JLabel("Month"));
41         add(monthField);
42         add(new JLabel("Year"));
43         add(yearField);
44         add(new JLabel("Name"));
45         add(nameField);
46         add(new JLabel("Position"));
47         add(positionField);
48         add(new JLabel("Salary"));
49         add(salaryField);
50
51         JButton saveButton = new JButton("Save");
52         JButton viewSalaryButton = new JButton("View Salary");
53         JButton listTeachersButton = new JButton("List Teachers");
54         JButton modifyButton = new JButton("Modify");
55         JButton searchButton = new JButton("Search");
56         JButton deleteButton = new JButton("Delete");
57
58         saveButton.addActionListener(new ActionListener() {
59             public void actionPerformed(ActionEvent e) {
60                 saveTeacher();
61             }
62         });
63
64         viewSalaryButton.addActionListener(new ActionListener() {
65             public void actionPerformed(ActionEvent e) {
66                 viewSalary();
67             }
68         });
69
70         listTeachersButton.addActionListener(new ActionListener() {
71             public void actionPerformed(ActionEvent e) {
72                 listTeachers();
73             }
74         });
75
76         modifyButton.addActionListener(new ActionListener() {
77             public void actionPerformed(ActionEvent e) {
78                 modify();
79             }
80         });
81
82         searchButton.addActionListener(new ActionListener() {
83             public void actionPerformed(ActionEvent e) {
84                 search();
85             }
86         });
87
88         deleteButton.addActionListener(new ActionListener() {
89             public void actionPerformed(ActionEvent e) {
90                 deleteRecord();
91             }
92         });
93
94         add(saveButton);
95         add(viewSalaryButton);
96         add(listTeachersButton);
97         add(modifyButton);
98         add(searchButton);
99         add(deleteButton);
100    }
101
102    private void saveTeacher() {
103        String date = dayField.getText() + "/" + monthField.getText() + "/" + yearField.getText();
104        String name = nameField.getText();
105        String position = positionField.getText();
106        double salary = Double.parseDouble(salaryField.getText());
107
108        String query = "INSERT INTO teachers (date, name, position, salary) VALUES (?, ?, ?, ?)";
109
110        try {
111            Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/teachers_db", "root", "");
112            PreparedStatement stmt = conn.prepareStatement(query);
113            stmt.setString(1, date);
114            stmt.setString(2, name);
115            stmt.setString(3, position);
116            stmt.setDouble(4, salary);
117            stmt.executeUpdate();
118
119            JOptionPane.showMessageDialog(null, "Teacher saved successfully!");
120        } catch (SQLException e) {
121            JOptionPane.showMessageDialog(null, "Error saving teacher: " + e.getMessage());
122        }
123    }
124
125    private void viewSalary() {
126        String name = nameField.getText();
127
128        String query = "SELECT salary FROM teachers WHERE name = ?";
129
130        try {
131            Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/teachers_db", "root", "");
132            PreparedStatement stmt = conn.prepareStatement(query);
133            stmt.setString(1, name);
134            ResultSet rs = stmt.executeQuery();
135
136            if (rs.next()) {
137                double salary = rs.getDouble("salary");
138                JOptionPane.showMessageDialog(null, "Salary: " + salary);
139            } else {
140                JOptionPane.showMessageDialog(null, "No teacher found with Name: " + name);
141            }
142        } catch (SQLException e) {
143            JOptionPane.showMessageDialog(null, "Error searching teacher: " + e.getMessage());
144        }
145    }
146
147    private void listTeachers() {
148        String query = "SELECT * FROM teachers";
149
150        try {
151            Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/teachers_db", "root", "");
152            Statement stmt = conn.createStatement();
153            ResultSet rs = stmt.executeQuery(query);
154
155            while (rs.next()) {
156                String date = rs.getString("date");
157                String name = rs.getString("name");
158                String position = rs.getString("position");
159                double salary = rs.getDouble("salary");
160
161                JOptionPane.showMessageDialog(null, "Date: " + date + ", Name: " + name + ", Position: " + position + ", Salary: " + salary);
162            }
163        } catch (SQLException e) {
164            JOptionPane.showMessageDialog(null, "Error listing teachers: " + e.getMessage());
165        }
166    }
167
168    private void modify() {
169        String name = nameField.getText();
170        String position = positionField.getText();
171        double salary = Double.parseDouble(salaryField.getText());
172
173        String query = "UPDATE teachers SET position = ?, salary = ? WHERE name = ?";
174
175        try {
176            Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/teachers_db", "root", "");
177            PreparedStatement stmt = conn.prepareStatement(query);
178            stmt.setString(1, position);
179            stmt.setDouble(2, salary);
180            stmt.setString(3, name);
181            stmt.executeUpdate();
182
183            JOptionPane.showMessageDialog(null, "Teacher modified successfully!");
184        } catch (SQLException e) {
185            JOptionPane.showMessageDialog(null, "Error modifying teacher: " + e.getMessage());
186        }
187    }
188
189    private void search() {
190        String name = nameField.getText();
191
192        String query = "SELECT * FROM teachers WHERE name = ?";
193
194        try {
195            Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/teachers_db", "root", "");
196            PreparedStatement stmt = conn.prepareStatement(query);
197            stmt.setString(1, name);
198            ResultSet rs = stmt.executeQuery();
199
200            if (rs.next()) {
201                String date = rs.getString("date");
202                String position = rs.getString("position");
203                double salary = rs.getDouble("salary");
204
205                JOptionPane.showMessageDialog(null, "Date: " + date + ", Name: " + name + ", Position: " + position + ", Salary: " + salary);
206            } else {
207                JOptionPane.showMessageDialog(null, "No teacher found with Name: " + name);
208            }
209        } catch (SQLException e) {
210            JOptionPane.showMessageDialog(null, "Error searching teacher: " + e.getMessage());
211        }
212    }
213
214    private void deleteRecord() {
215        String name = nameField.getText();
216
217        String query = "DELETE FROM teachers WHERE name = ?";
218
219        try {
220            Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/teachers_db", "root", "");
221            PreparedStatement stmt = conn.prepareStatement(query);
222            stmt.setString(1, name);
223            stmt.executeUpdate();
224
225            JOptionPane.showMessageDialog(null, "Teacher deleted successfully!");
226        } catch (SQLException e) {
227            JOptionPane.showMessageDialog(null, "Error deleting teacher: " + e.getMessage());
228        }
229    }
230
231    private String getName() {
232        return nameField.getText();
233    }
234
235    @Override
236    public String toString() {
237        return dateOfJoining + ", Name: " + name + ", Position: " + position + ", Salary: " + salary;
238    }
239 }

```

## 6.7 TeacherSectionGUI

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

```

Source History | [File] [Edit] [View] [Search] [Help] [File] [Edit] [View] [Search] [Help]
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);
    }
}

```

Search Results Notifications

```

Source History | [File] [Edit] [View] [Search] [Help] [File] [Edit] [View] [Search] [Help]
    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);
}

```

Search Results Notifications



Source History

```

67     private void saveTeacher() {
68         try {
69             int day = Integer.parseInt(dayField.getText().trim());
70             String month = monthField.getText().trim();
71             int year = Integer.parseInt(yearField.getText().trim());
72             CustomDate date = new CustomDate(day, month, year);
73             if (!date.chkDate()) throw new Exception("Invalid date!");
74
75             String name = nameField.getText().trim();
76             String position = positionField.getText().trim();
77             double salary = Double.parseDouble(salaryField.getText().trim());
78
79             if (name.isEmpty() || position.isEmpty() || salary < 0) {
80                 throw new Exception("Invalid input: All fields must be valid.");
81             }
82
83             var dbHandler = new DatabaseHandler();
84             dbHandler.saveTeacher(name, position, salary, date.toString());
85             JOptionPane.showMessageDialog(this, "Teacher saved: " + date + " " + name);
86             dispose();
87         } catch (Exception e) {
88             JOptionPane.showMessageDialog(this, "Error: " + e.getMessage());
89         }
90     }
91
92     private void viewSalary() {
93         try {
94             int day = Integer.parseInt(dayField.getText().trim());
95             String month = monthField.getText().trim();
96             int year = Integer.parseInt(yearField.getText().trim());
97             CustomDate date = new CustomDate(day, month, year);
98             if (!date.chkDate()) throw new Exception("Invalid date!");
99
100            String name = nameField.getText().trim();

```

Search Results Notifications

Source History

```

100        String name = nameField.getText().trim();
101        String position = positionField.getText().trim();
102        double salary = Double.parseDouble(salaryField.getText().trim());
103        Teacher teacher = new Teacher(name, position, salary, date);
104        teacher.viewSalary();
105    } catch (Exception e) {
106        JOptionPane.showMessageDialog(this, "Error: " + e.getMessage());
107    }
108
109
110    private void displayRecordList() {
111        try {
112            var dbHandler = new DatabaseHandler();
113            var rs = dbHandler.getTeachers();
114            StringBuilder display = new StringBuilder("Teacher Records:\n");
115            boolean hasRecords = false;
116
117            while (rs.next()) {
118                hasRecords = true;
119                String[] dateParts = rs.getString("date").split(" ");
120                CustomDate date = new CustomDate(Integer.parseInt(dateParts[0]), dateParts[1], Integer.parseInt(dateParts[2]));
121                display.append(date)
122                    .append(", Name: ").append(rs.getString("name"))
123                    .append(", Position: ").append(rs.getString("position"))
124                    .append(", Salary: ").append(rs.getDouble("salary"))
125                    .append("\n");
126            }
127
128            if (!hasRecords) {
129                display.append("No records found.");
130            }
131
132            JOptionPane.showMessageDialog(this, display.toString());
133        } catch (Exception e) {

```

Search Results Notifications



The screenshot shows a Java application interface with a toolbar at the top containing icons for file operations, search, and other functions. The main area displays Java code for modifying and searching teacher records. The code uses JOptionPane dialogs for input and DatabaseHandler for database operations. It includes exception handling for invalid dates and search results.

```
134     } catch (Exception e) {
135         JOptionPane.showMessageDialog(this, "Error displaying records: " + e.getMessage());
136     }
137 }
138 private void modifyRecord() {
139     try {
140         String name = JOptionPane.showInputDialog(this, "Enter name to modify:");
141         int day = Integer.parseInt(JOptionPane.showInputDialog(this, "Enter new day of joining:"));
142         String month = JOptionPane.showInputDialog(this, "Enter new month of joining:");
143         int year = Integer.parseInt(JOptionPane.showInputDialog(this, "Enter new year of joining:"));
144         CustomDate date = new CustomDate(day, month, year);
145         if (!date.chkDate()) throw new Exception("Invalid date!");
146
147         String position = JOptionPane.showInputDialog(this, "Enter new position:");
148         double salary = Double.parseDouble(JOptionPane.showInputDialog(this, "Enter new salary:"));
149         var dbHandler = new DatabaseHandler();
150         dbHandler.updateTeacher(name, position, salary, date.tostring());
151         JOptionPane.showMessageDialog(this, "Record modified: " + date + " " + name);
152     } catch (Exception e) {
153         JOptionPane.showMessageDialog(this, "Error: " + e.getMessage());
154     }
155 }
156
157 private void searchRecord() {
158     try {
159         String name = JOptionPane.showInputDialog(this, "Enter name to search:");
160         var dbHandler = new DatabaseHandler();
161         var rs = dbHandler.searchTeacher(name);
```

The screenshot shows a Java IDE interface with the following code:

```
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.



```

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 : 0));
    }

    public String getName() {
        return name;
    }
}

```

Activate Windows  
Go to Settings to activate Windows.

(1) 1:1 INS Win<sup>3</sup>

```

History | File | Edit | View | Insert | Tools | Options | Help | Run | Save | Print | Copy | Paste | Find | Replace | Cut | Copy | Paste | Delete | Undo | Redo | New | Open | Save As | Properties | Refresh | Help | About | Exit | History | File | Edit | View | Insert | Tools | Options | Help | Run | Save | Print | Copy | Paste | Delete | Undo | Redo | New | Open | Save As | Properties | Refresh | Help | About | Exit |

```

```

0  public void viewFee() {
1  	double due = totalFee - feePaid;
2  	double fine = due > 0 ? due * 0.05 : 0;
3  	double advance = feePaid > totalFee ? feePaid - totalFee : 0;
4  	JOptionPane.showMessageDialog(null, dateOfAdmission + "\nTotal Fee: " + totalFee + "\nFee Paid: " + feePaid + "\nDue: " + (due > 0 ? due : 0) + "\nFine: " + fine + "\nAdvance: " + advance);
5  }

6  public String getName() {
7  	return name;
8  }

9  public String getStudentClass() {
10 	return studentClass;
11}

12 public int getRollNo() {
13 	return rollNo;
14}

15 public double getFeePaid() {
16 	return feePaid;
17}

18 public CustomDate getDateOfAdmission() {
19 	return dateOfAdmission;
20}

21 @Override
22 public String toString() {
23 	return dateOfAdmission + ", Name: " + name + ", Class: " + studentClass + ", Roll No: " + rollNo + ", Fee Paid: " + feePaid;
24}

```

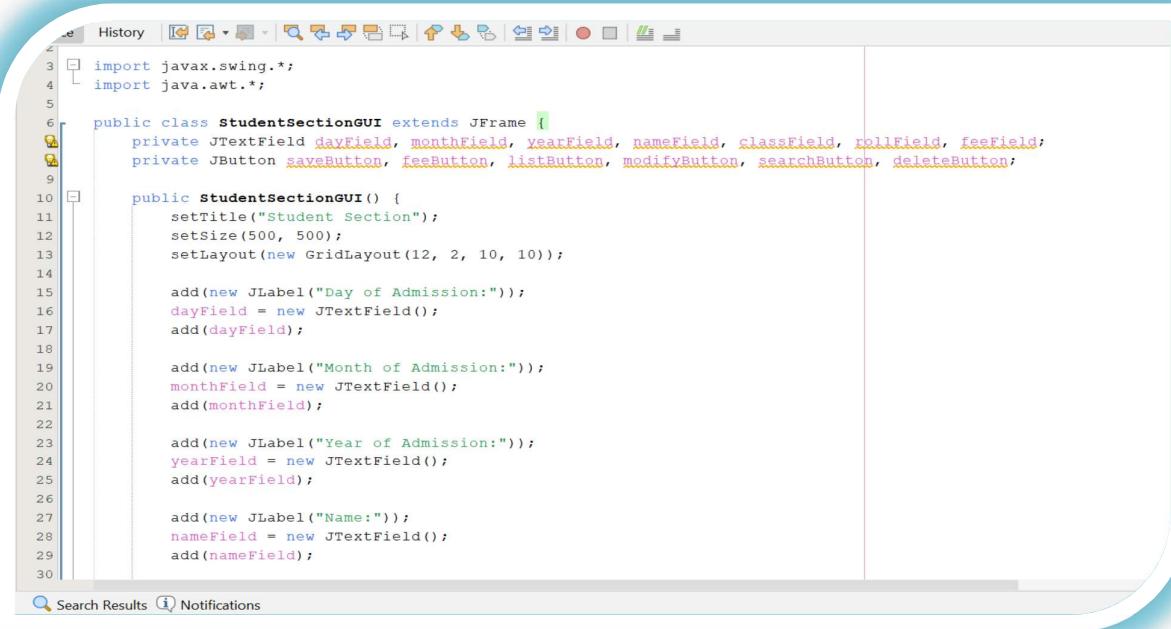
Activate Windows  
Go to Settings to activate Windows.

(1) 1:1 INS Win<sup>3</sup>



## 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.



```
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));

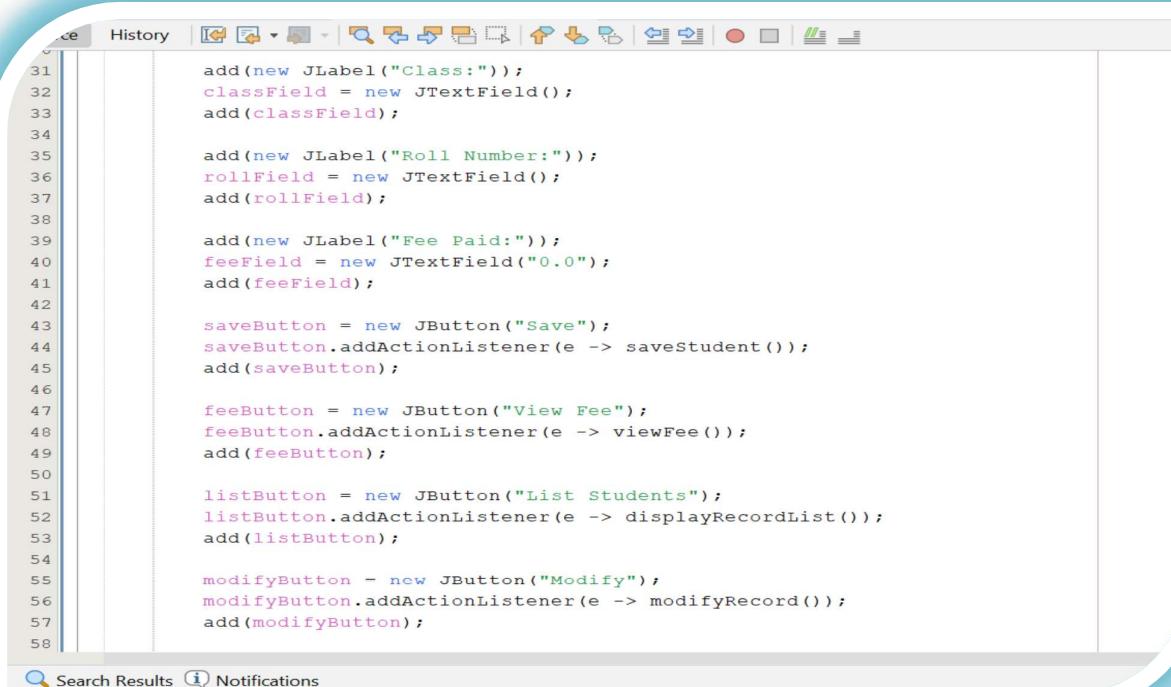
        add(new JLabel("Day of Admission:"));
        dayField = new JTextField();
        add(dayField);

        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);
    }
}
```

Search Results Notifications



```
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);
}
```

Search Results Notifications

```

59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
    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 saveStudent() {
    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 studentClass = classField.getText().trim();
        int rollNo = Integer.parseInt(rollField.getText().trim());
        double feePaid = Double.parseDouble(feeField.getText().trim());

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

```

```

88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
    var dbHandler = new DatabaseHandler();
    dbHandler.saveStudent(name, studentClass, rollNo, feePaid, date.toString());
    JOptionPane.showMessageDialog(this, "Student saved: " + date + " " + name);
    dispose();
} catch (Exception e) {
    JOptionPane.showMessageDialog(this, "Error: " + e.getMessage());
}

private void viewFee() {
    try {
        int rollNo = Integer.parseInt(rollField.getText().trim());
        double feePaid = Double.parseDouble(feeField.getText().trim());
        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!");

        Student student = new Student("", "", rollNo, feePaid, date);
        student.viewFee();
    } catch (Exception e) {
        JOptionPane.showMessageDialog(this, "Error: " + e.getMessage());
    }
}

private void displayRecordList() {
    try {
        var dbHandler = new DatabaseHandler();
        var rs = dbHandler.getStudents();
    }
}

```

 Search Results  Notifications



The screenshot shows a Java code editor with the following code:

```
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!");
    }
}
```

The screenshot shows a Java application window with the title 'History'. The code is organized into several methods:

- updateRecord() (lines 50-158):** Prompts for new name, class, and fee paid, then updates the database and displays a success message.
- searchRecord() (lines 160-176):** Prompts for a roll number, searches the database, and displays the student's details or an error message if not found.
- deleteRecord() (lines 178-180):** Prompts for a roll number to delete, which is then handled by the database handler.

The code uses JOptionPane for user input and output, and DatabaseHandler for database operations. It includes exception handling for all operations.



```

History [File] [Edit] [View] [Search] [Tools] [Help]
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();
        dbHandler.deleteStudent(rollNo);
        JOptionPane.showMessageDialog(this, "Record deleted.");
    } catch (Exception e) {
        JOptionPane.showMessageDialog(this, "Error: " + e.getMessage());
    }
}

```

Activate Windows  
Go to Settings to activate Windows.

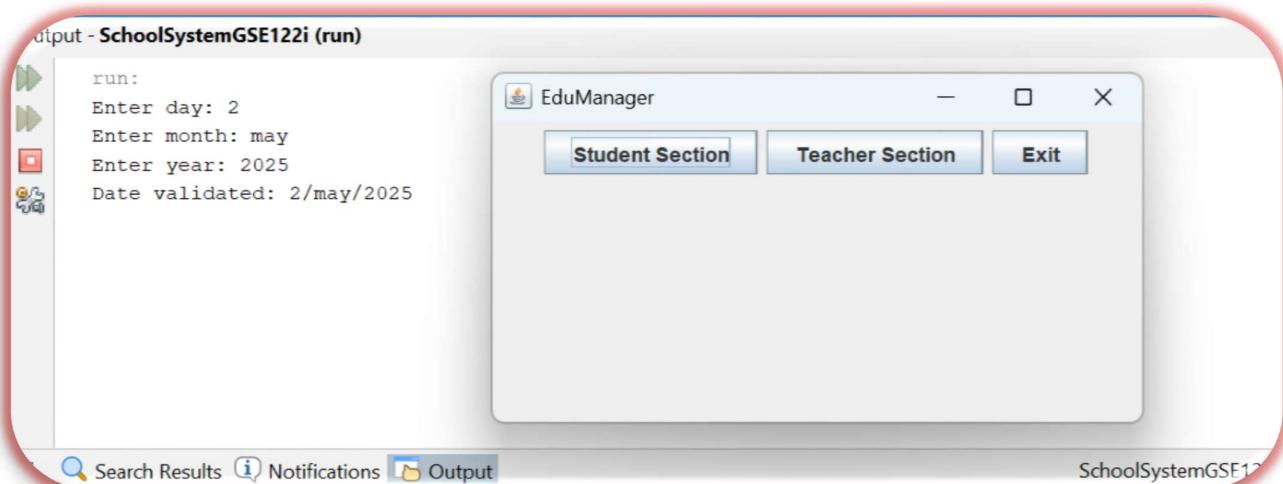
① 187:2 INS Win+1

Search Results Notifications Output

## 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:

Teacher Section

Day of Joining: 15

Month of Joining: January

Year of Joining: 2025

Name: Mohamed Ahmed

Position: Science Teacher

Salary: 5000

Save View Salary

List Teachers Modify

Search Delete

Message

Teacher saved: 15 January 2025 Mohamed Ahmed

OK

- View Salary Operation:

Teacher Section

Day of Joining: 15

Month of Joining: January

Year of Joining: 2025

Name: Mohamed Ahmed

Position: Science Teacher

Salary: 5000

Save View Salary

List Teachers Modify

Search Delete

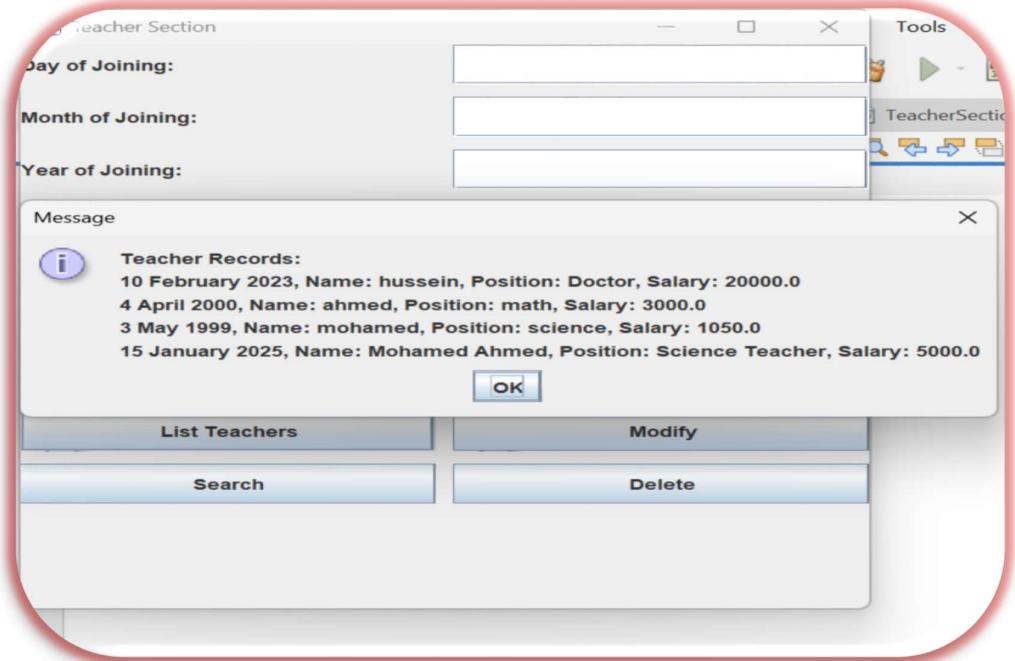
Message

15 January 2025  
Position: Science Teacher  
Salary: 5000.0

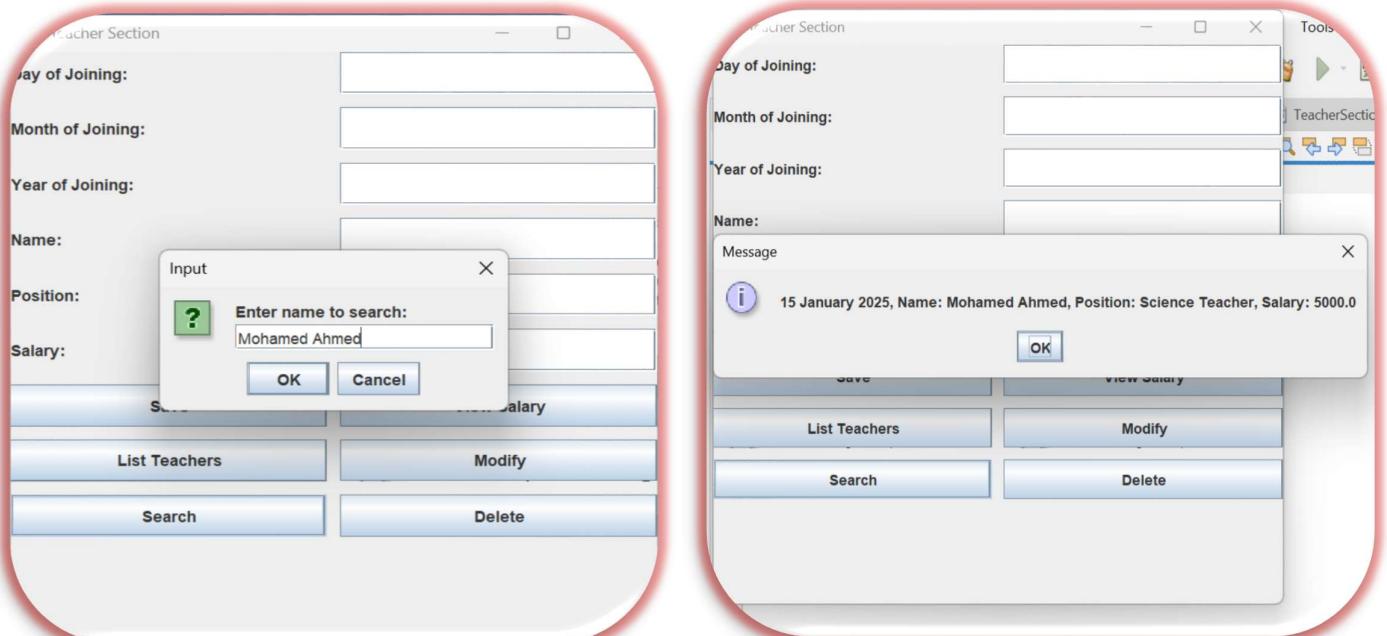
OK



- o List Teachers Operation:



- o Search Operation:





- o Modify Operation:

The screenshot shows a software application window titled "Teacher Section". It contains fields for "Day of Joining", "Month of Joining", and "Year of Joining". Below these are fields for "Name", "Position", and "Salary", each with an associated input field and a dropdown menu. A modal dialog box titled "Input" is open, asking "Enter name to modify:" with the value "Mohamed Ahmed" entered. At the bottom of the application are buttons for "List Teachers", "Modify", "Search", and "Delete". To the right of the application, a second window shows a message dialog box with an information icon and the text "Record modified: 15 January 2025 Mohamed Ahmed". There is an "OK" button at the bottom of this message box.

- o Delete Operations:

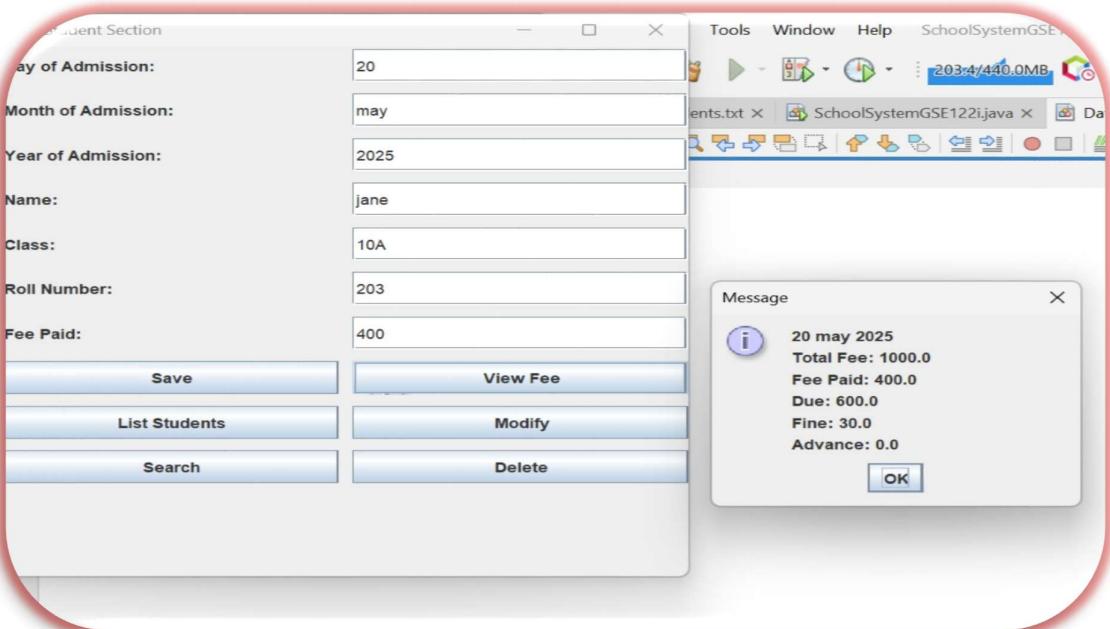
The screenshot shows a software application window titled "Teacher Section". It contains fields for "Day of Joining", "Month of Joining", and "Year of Joining". Below these are fields for "Name", "Position", and "Salary", each with an associated input field and a dropdown menu. A modal dialog box titled "Input" is open, asking "Enter name to delete:" with the value "Mohamed Ahmed" entered. At the bottom of the application are buttons for "List Teachers", "Modify", "Search", and "Delete". To the right of the application, a second window shows a message dialog box with an information icon and the text "Record deleted successfully!". There is an "OK" button at the bottom of this message box.

- Student Section (StudentSectionGUI):

- Save Operation:

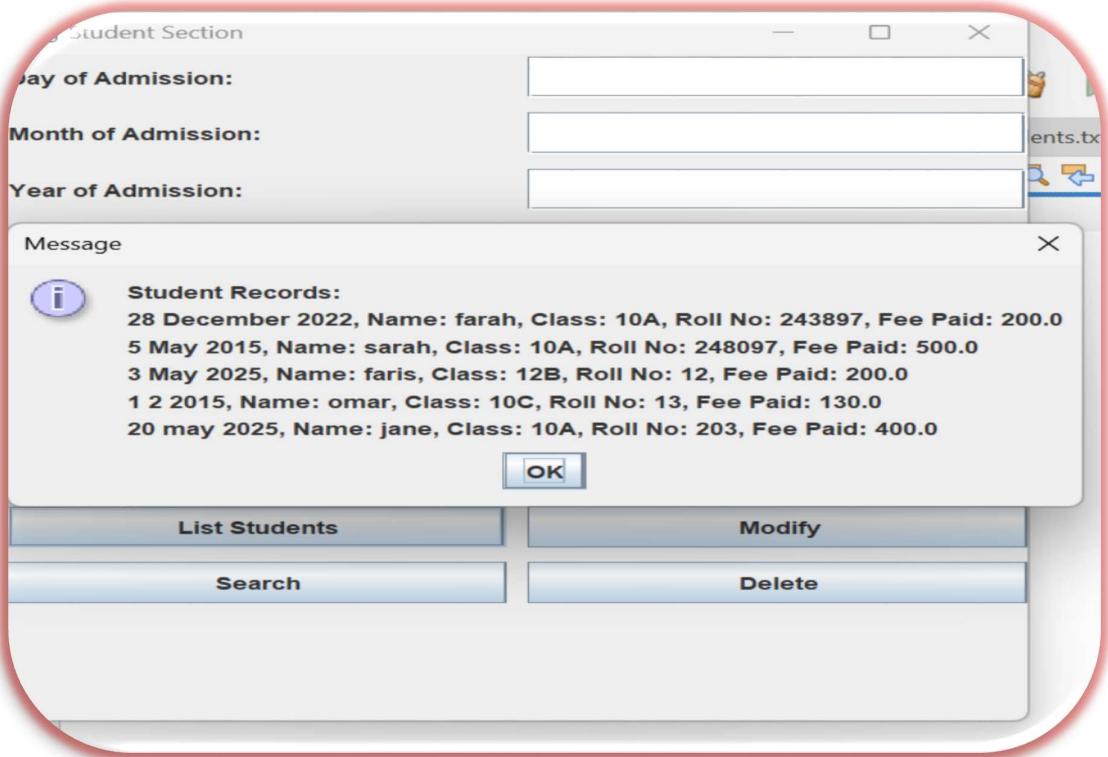


- View Fee Operation:

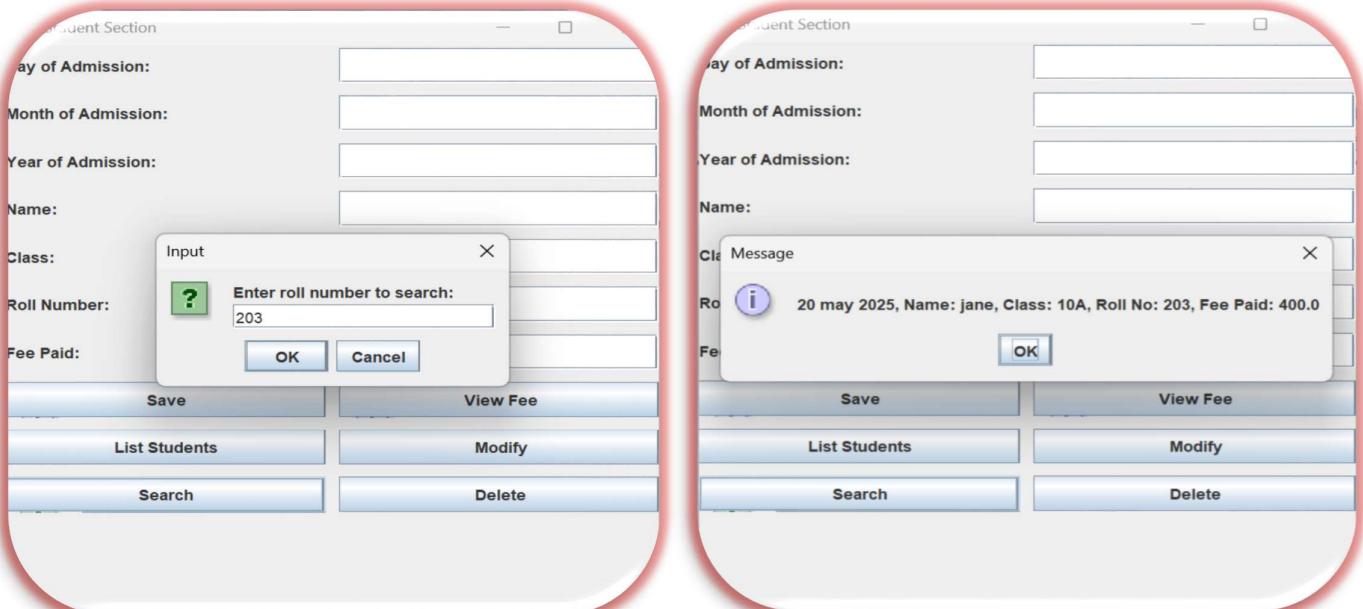




- List Students Operation:



- Search Operation:





- o Modify Operation:

Student Section

Day of Admission:

Month of Admission:

Year of Admission:

Name:

Class:

Roll Number:  ? Enter roll number to modify:  
203

Fee Paid:

**OK** **Cancel**

**Save** **View Fee**

**List Students** **Modify**

**Search** **Delete**

Student Section

Day of Admission:

Month of Admission:

Year of Admission:

Name:

Class:

Roll Number:  ? Record modified: 2 may 2025 mona

Fee Paid:

**OK**

**Save** **View Fee**

**List Students** **Modify**

**Search** **Delete**

- o Delete Operations:

Student Section

Day of Admission:

Month of Admission:

Year of Admission:

Name:

Class:

Roll Number:  ? Enter roll number to delete:  
203

Fee Paid:

**OK** **Cancel**

**Save** **View Fee**

**List Students** **Modify**

**Search** **Delete**

Student Section

Day of Admission:

Month of Admission:

Year of Admission:

Name:

Class:

Roll Number:  ? Record deleted.

Fee Paid:

**OK**

**Save** **View Fee**

**List Students** **Modify**

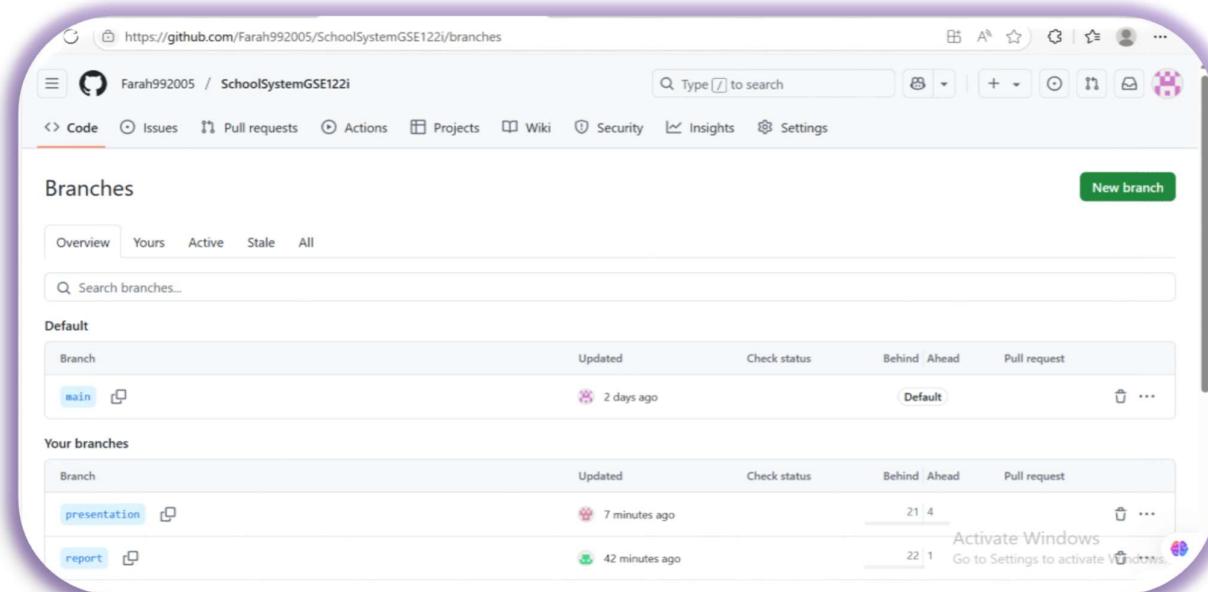
**Search** **Delete**



## 8. GitHub

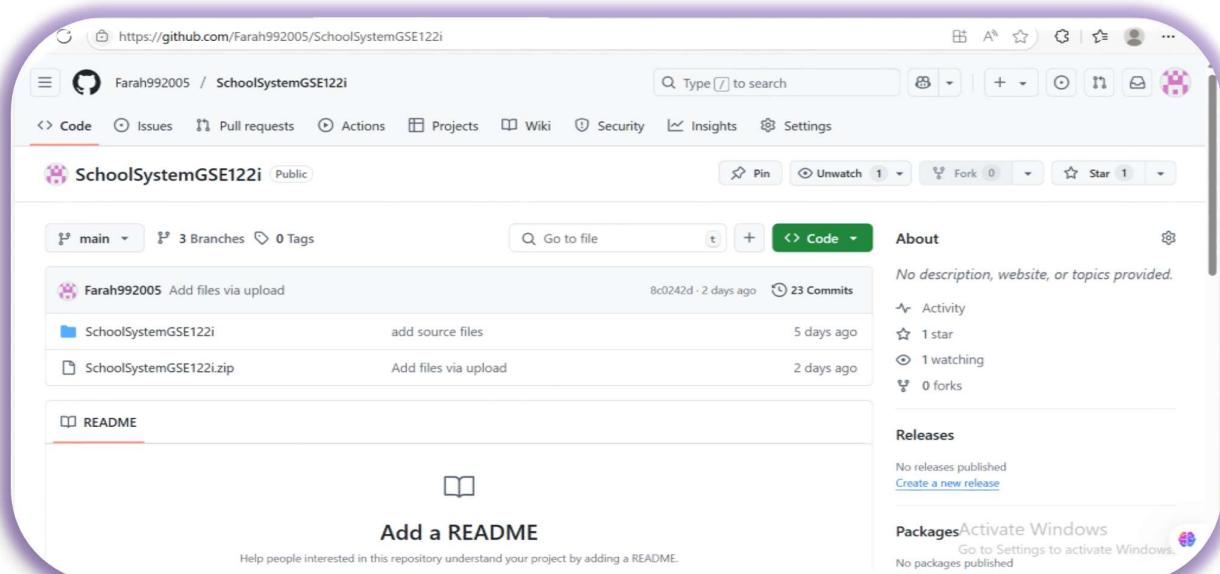
Repository URL: <https://github.com/Farah992005/SchoolSystemGSE122i>

3 Branches:



The screenshot shows the GitHub interface for the repository `SchoolSystemGSE122i`. The main navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. The 'Code' tab is selected. In the top right corner, there is a green button labeled 'New branch'. Below the navigation, the 'Branches' section is displayed. Under 'Default', the 'main' branch is listed with an update timestamp of '2 days ago' and a status of 'Default'. Under 'Your branches', the 'presentation' and 'report' branches are listed, both updated '7 minutes ago'. A message 'Activate Windows' is visible next to the 'report' branch.

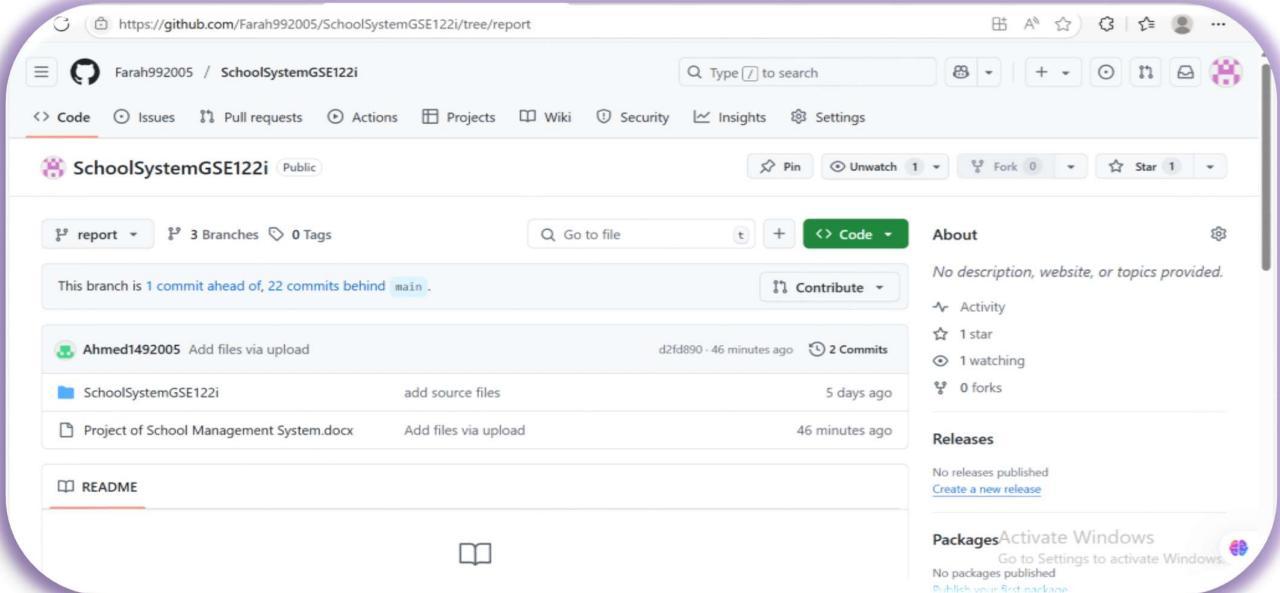
Main Branch:



The screenshot shows the GitHub interface for the repository `SchoolSystemGSE122i`. The main navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. The 'Code' tab is selected. The repository name `SchoolSystemGSE122i` is shown in bold, with a public status indicator. The repository has 3 branches and 0 tags. Recent activity includes a commit from `Farah992005` adding files via upload, a commit adding source files, and a commit adding files via upload. The 'About' section notes 'No description, website, or topics provided.' and lists 1 star, 1 watching, and 0 forks. The 'Releases' section indicates 'No releases published' and provides a link to 'Create a new release'. The 'Packages' section shows 'Activate Windows' and a note about no packages published.



## Report Branch:



This branch is 1 commit ahead of, 22 commits behind main.

Ahmed1492005 Add files via upload · d2fd890 · 46 minutes ago · 2 Commits

SchoolSystemGSE122i add source files · 5 days ago

Project of School Management System.docx Add files via upload · 46 minutes ago

README

About

No description, website, or topics provided.

Activity

1 star

1 watching

0 forks

Releases

No releases published

Create a new release

Packages

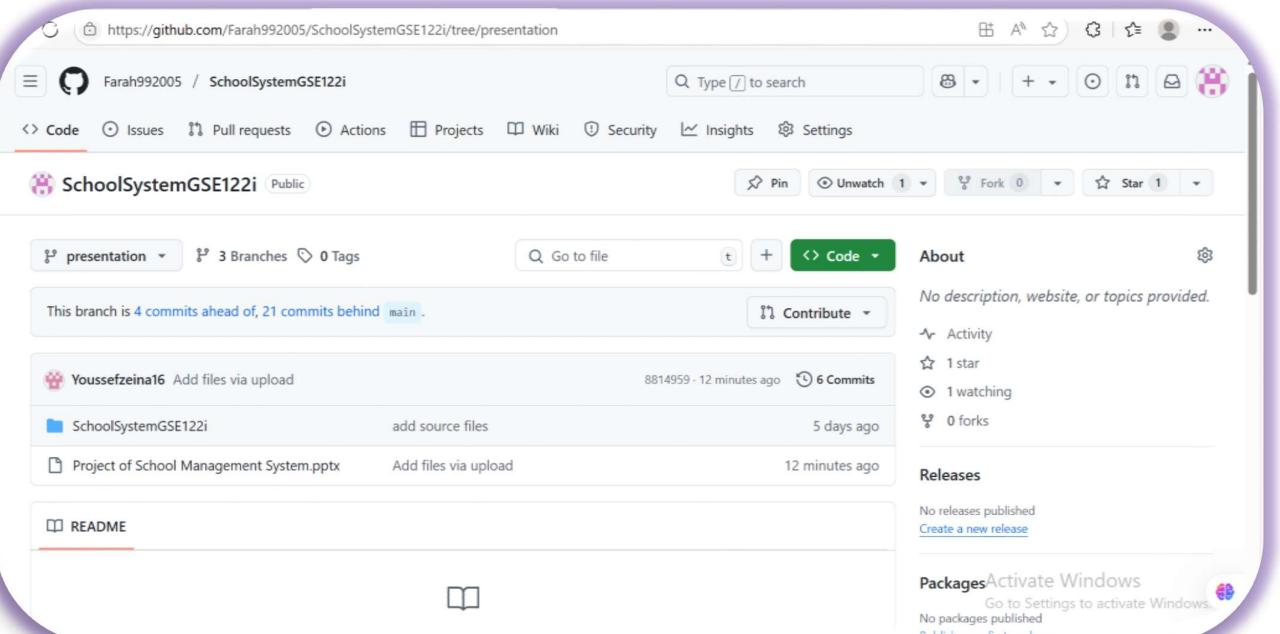
Activate Windows

Go to Settings to activate Windows.

No packages published

Click here to publish your first package

## Presentation Branch:



This branch is 4 commits ahead of, 21 commits behind main.

Youssefzeina16 Add files via upload · 8814959 · 12 minutes ago · 6 Commits

SchoolSystemGSE122i add source files · 5 days ago

Project of School Management System.pptx Add files via upload · 12 minutes ago

README

About

No description, website, or topics provided.

Activity

1 star

1 watching

0 forks

Releases

No releases published

Create a new release

Packages

Activate Windows

Go to Settings to activate Windows.

No packages published

Click here to publish your first package



## 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/>