



OCTOBER UNIVERSITY
FOR MODERN SCIENCES AND ARTS
جامعة أكتوبر للعلوم الحديثة والآداب

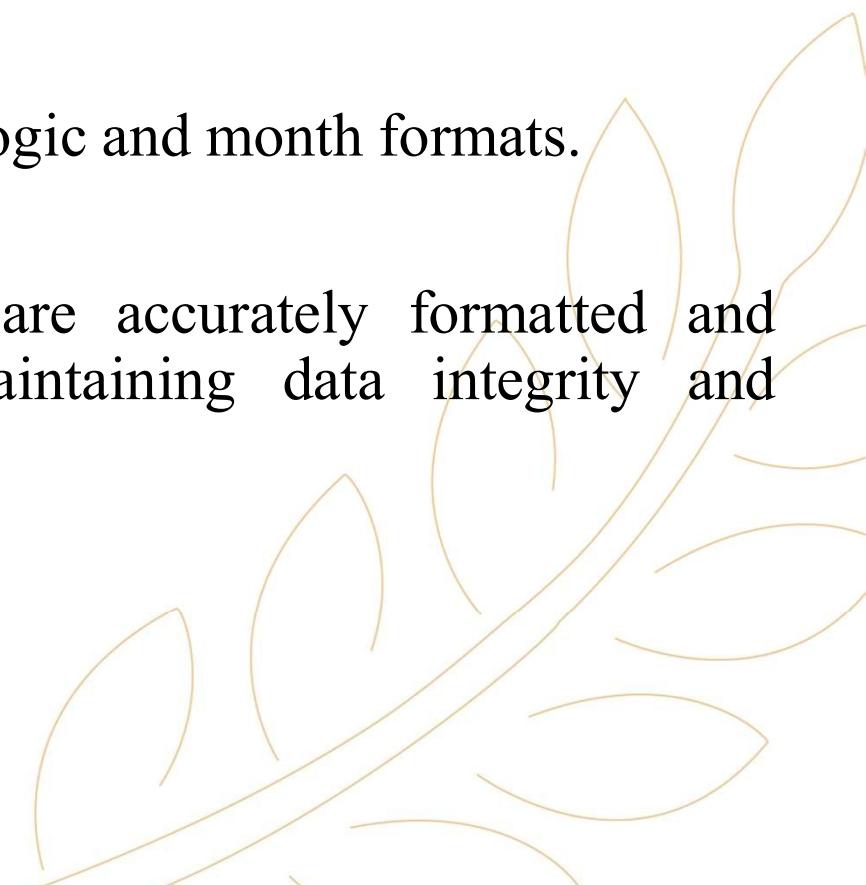


School Management System

- Student Name :Farah Mohamed Ahmed Habib ID :243897
Student Name :Sarah Mohamed Ali ID :248097
Student Name :Toni Wagdi Wadie ID:244743
Student Name :Ahmed Hisham Mohsen ID :242435
Student Name :Youssef Mohamed ID :234833

Presented to:Eng. Gehad Ehab

1. CustomDate

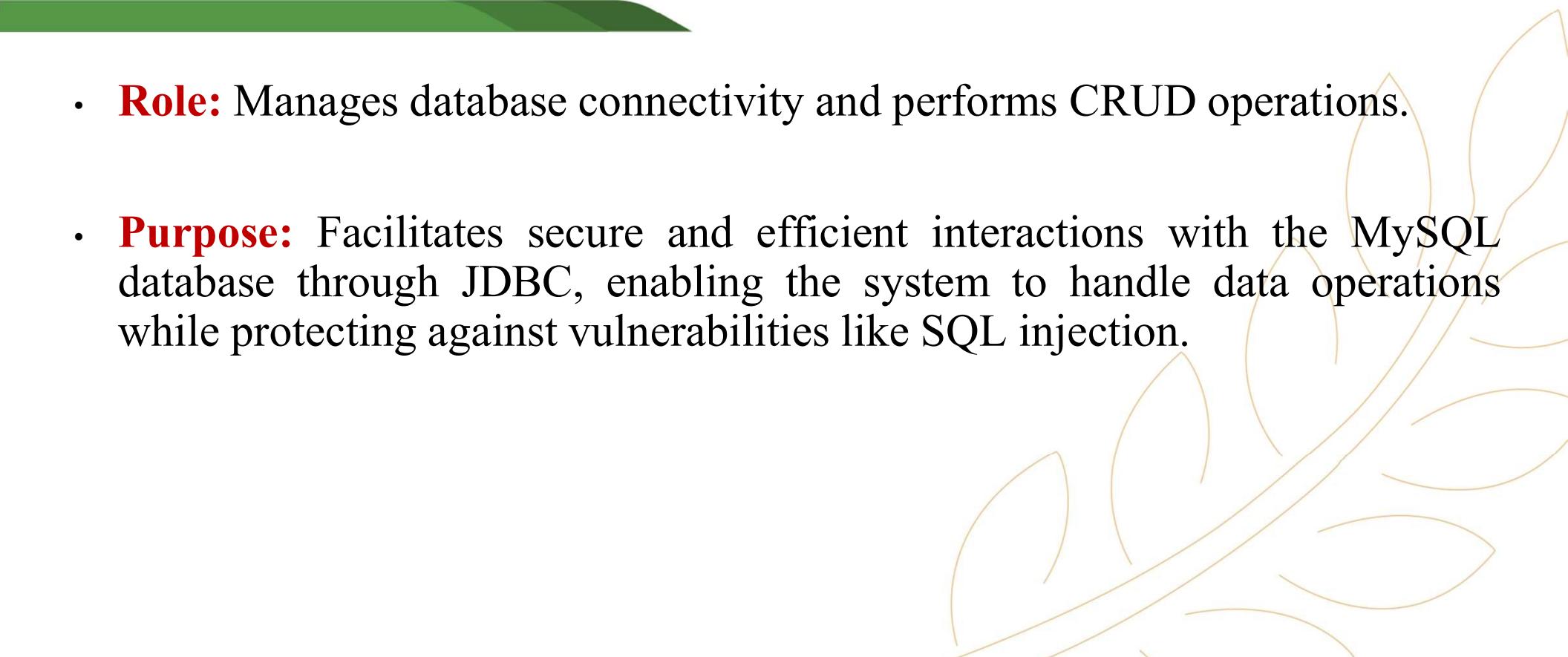


- **Role:** Validates date inputs with leap year logic and month formats.
- **Purpose:** Ensures that all date entries are accurately formatted and consistently stored in the database, maintaining data integrity and reliability across the entire application.

```
12
13
14
15
16
17 public class CustomDate {
18     private int day;
19     private String month;
20     private int year;
21
22     public CustomDate(int day, String month, int year) {
23         this.day = day;
24         this.month = month;
25         this.year = year;
26     }
27
28     public boolean chkDate() {
29         int numericMonth = getMonthNumber(month);
30
31         if (numericMonth < 1 || numericMonth > 12) return false;
32
33         int maxDays;
34
35         boolean isLeapYear = (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);
36
37         switch (numericMonth) {
38             case 2:
39                 maxDays = isLeapYear ? 29 : 28;
40                 break;
41             case 4:
42             case 6:
43             case 9:
44             case 11:
45                 maxDays = 30;
46                 break;
47             default:
48                 maxDays = 31;
49                 break;
50         }
51     }
52
53 }
```

```
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
```

2. DatabaseHandler



- **Role:** Manages database connectivity and performs CRUD operations.
- **Purpose:** Facilitates secure and efficient interactions with the MySQL database through JDBC, enabling the system to handle data operations while protecting against vulnerabilities like SQL injection.

```
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();
        }
    }
}
```

```
17 public void saveTeacher(String name, String position, double salary, String date) throws SQLException {
18     String query = "INSERT INTO teacher (name, position, salary, date) VALUES (?, ?, ?, ?)";
19     try (PreparedStatement stmt = conn.prepareStatement(query)) {
20         stmt.setString(1, name);
21         stmt.setString(2, position);
22         stmt.setDouble(3, salary);
23         stmt.setString(4, date);
24         stmt.executeUpdate();
25     }
26 }
27
28 public ResultSet getStudents() throws SQLException {
29     String query = "SELECT * FROM student";
30     return conn.createStatement().executeQuery(query);
31 }
32
33 public ResultSet getTeachers() throws SQLException {
34     String query = "SELECT * FROM teacher";
35     return conn.createStatement().executeQuery(query);
36 }
37
38 public void updateStudent(int rollNo, String name, String studentClass, double feePaid, String date) throws SQLException {
39     String query = "UPDATE student SET name = ?, studentClass = ?, feePaid = ?, date = ? WHERE rollNo = ?";
40     try (PreparedStatement stmt = conn.prepareStatement(query)) {
41         stmt.setString(1, name);
42         stmt.setString(2, studentClass);
43         stmt.setDouble(3, feePaid);
44         stmt.setString(4, date);
45         stmt.setInt(5, rollNo);
46         stmt.executeUpdate();
47     }
48 }
```

This screenshot shows a Java code editor with a purple header bar. The code implements methods for updating teacher records:

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

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

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

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

This screenshot shows a Java code editor with a purple header bar. It contains methods for deleting and searching records from student and teacher tables:

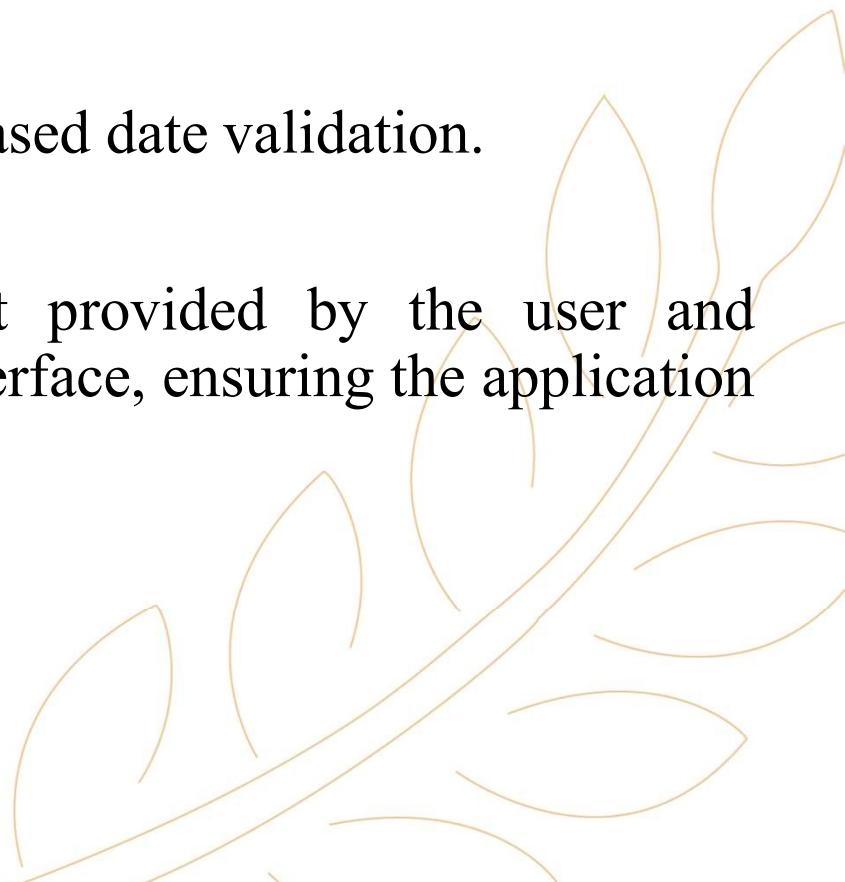
```
public void deleteStudent(int rollNo) throws SQLException {
    String query = "DELETE FROM student WHERE rollNo = ?";
    try (PreparedStatement stmt = conn.prepareStatement(query)) {
        stmt.setInt(1, rollNo);
        stmt.executeUpdate();
    }
}

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

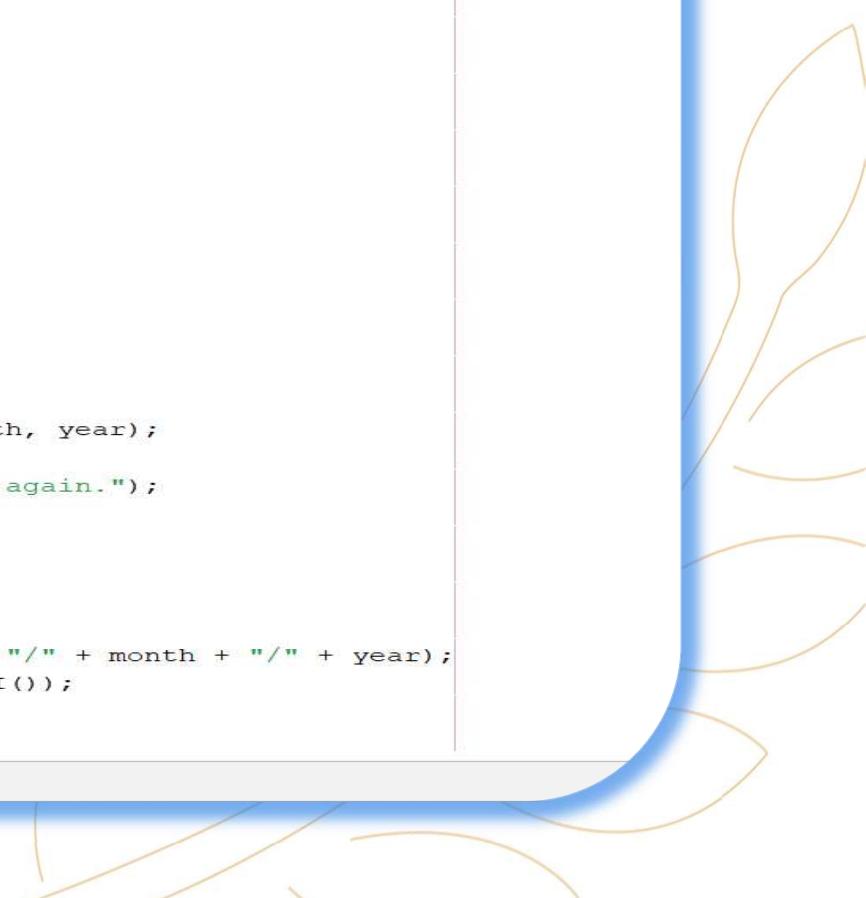
public ResultSet searchStudent(int rollNo) throws SQLException {
    String query = "SELECT * FROM student WHERE rollNo = ?";
    PreparedStatement stmt = conn.prepareStatement(query);
    stmt.setInt(1, rollNo);
    return stmt.executeQuery();
}

public ResultSet searchTeacher(String name) throws SQLException {
    String query = "SELECT * FROM teacher WHERE name = ?";
    PreparedStatement stmt = conn.prepareStatement(query);
    stmt.setString(1, name);
    return stmt.executeQuery();
}
```

3. SchoolSystemGSE122i



- **Role:** Acts as the entry point with console-based date validation.
- **Purpose:** Validates the initial date input provided by the user and subsequently launches the graphical user interface, ensuring the application starts with a verified foundation.

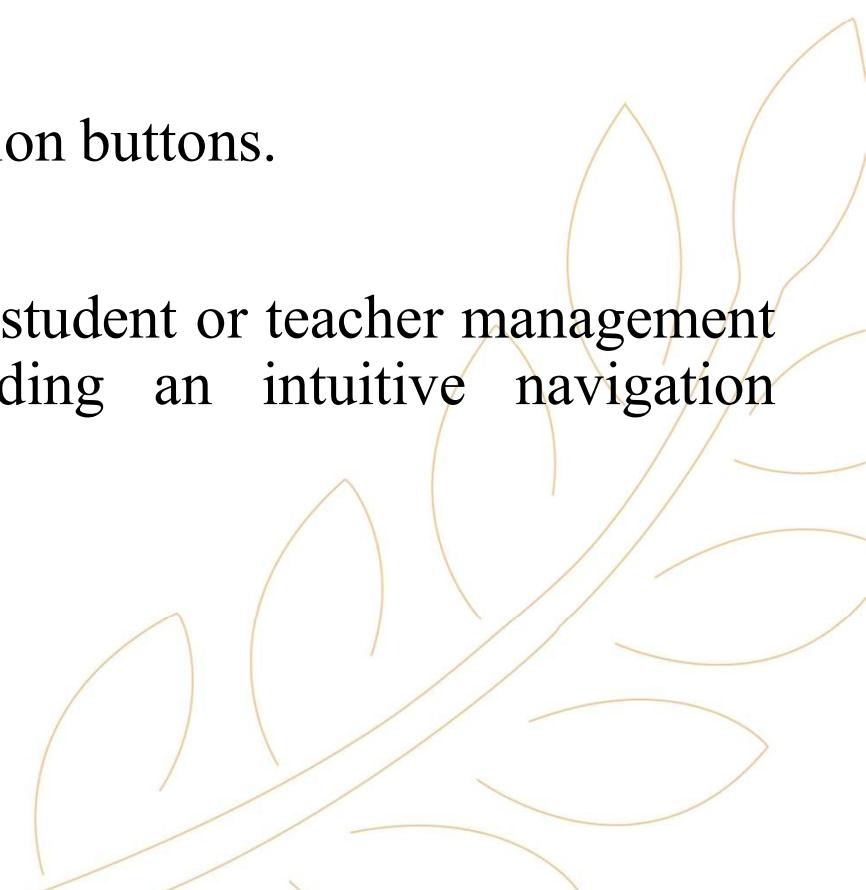


The image shows a Java code editor window with a blue border. The title bar includes tabs for 'Source' and 'History', along with various icons for file operations like opening, saving, and navigating. The code itself is a simple application for validating dates. It uses a Scanner to read input from the user and a CustomDate class to perform validation. If the date is invalid, it prints an error message and loops back to ask for another date. Once a valid date is entered, it prints the date in the format DD/MM/YYYY and creates a new SchoolGUI instance.

```
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 }
```

At the bottom left, there are two small buttons: 'Search Results' and 'Notifications'.

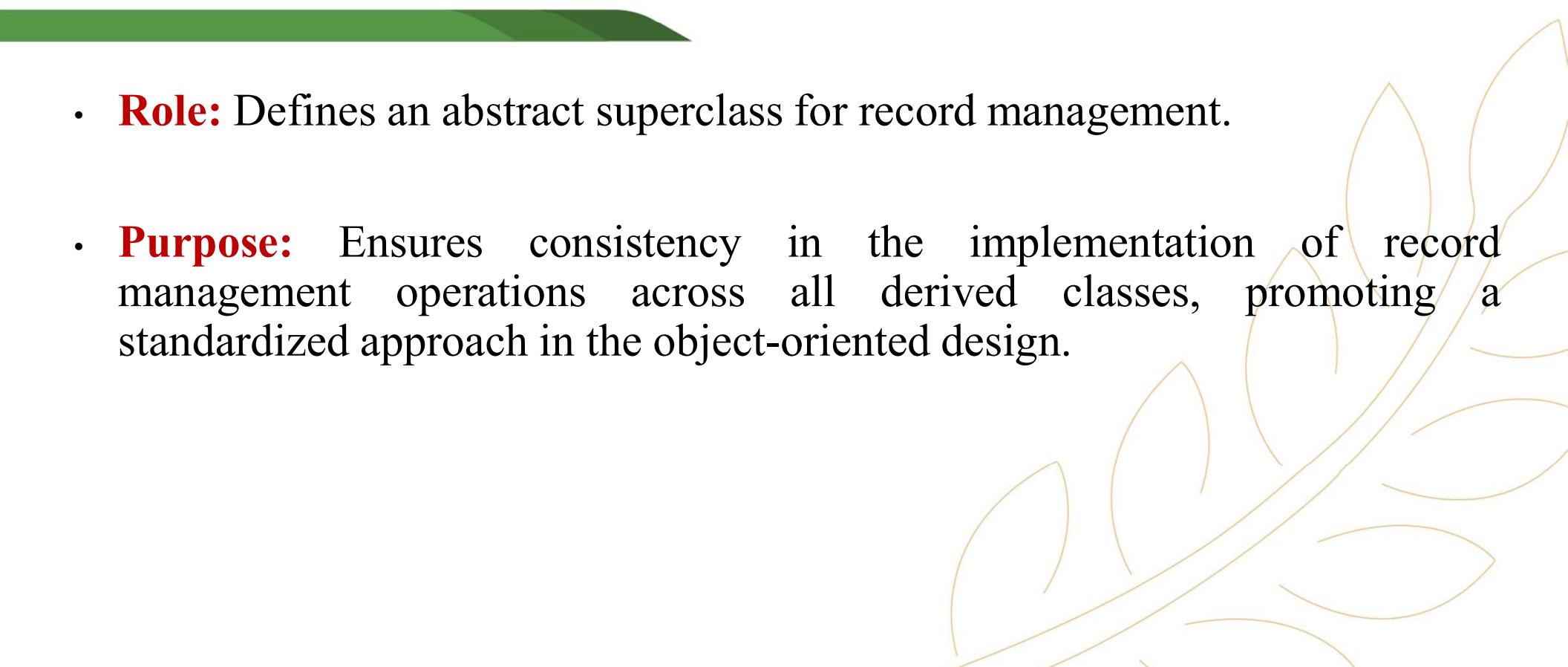
4. SchoolGUI



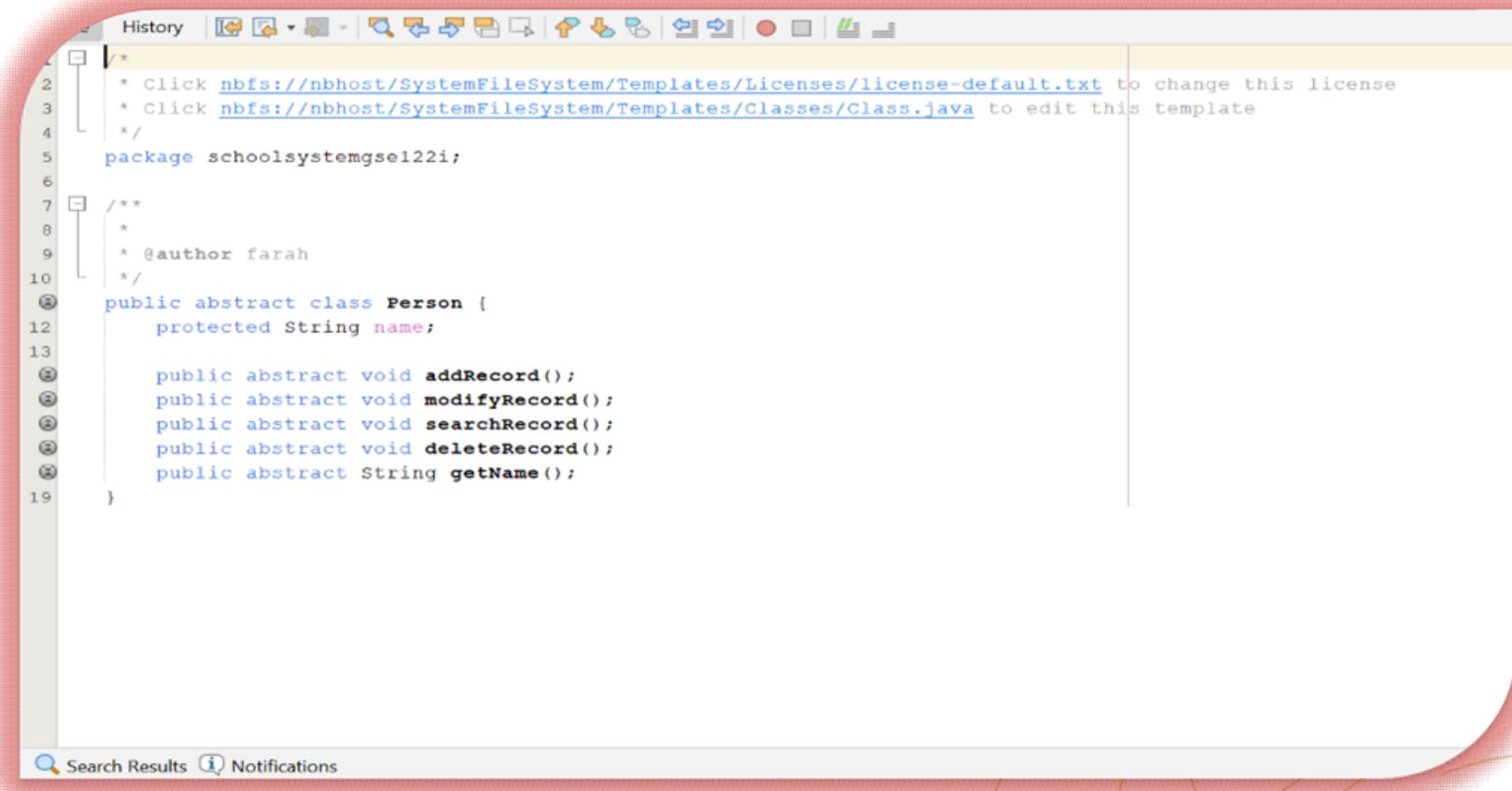
- **Role:** Displays the main menu with navigation buttons.
- **Purpose:** Guides users to access either the student or teacher management sections or exit the application, providing an intuitive navigation experience to enhance usability.

```
1  import javax.swing.*;
2  import java.awt.*;
3
4  public class SchoolGUI extends JFrame {
5      public SchoolGUI() {
6          displayMainMenu();
7      }
8
9      public void displayMainMenu() {
10         setTitle("EduManager");
11         setSize(400, 200);
12         setLayout(new FlowLayout());
13
14         JButton studentBtn = new JButton("Student Section");
15         JButton teacherBtn = new JButton("Teacher Section");
16         JButton exitBtn = new JButton("Exit");
17
18         studentBtn.addActionListener(e -> new StudentSectionGUI().setVisible(true));
19         teacherBtn.addActionListener(e -> new TeacherSectionGUI().setVisible(true));
20         exitBtn.addActionListener(e -> System.exit(0));
21
22         add(studentBtn);
23         add(teacherBtn);
24         add(exitBtn);
25
26         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
27         setVisible(true);
28     }
29
30 }
```

5. Person

A decorative element consisting of a solid green horizontal bar at the top, followed by a cluster of stylized gold-colored leaves and branches on the right side.

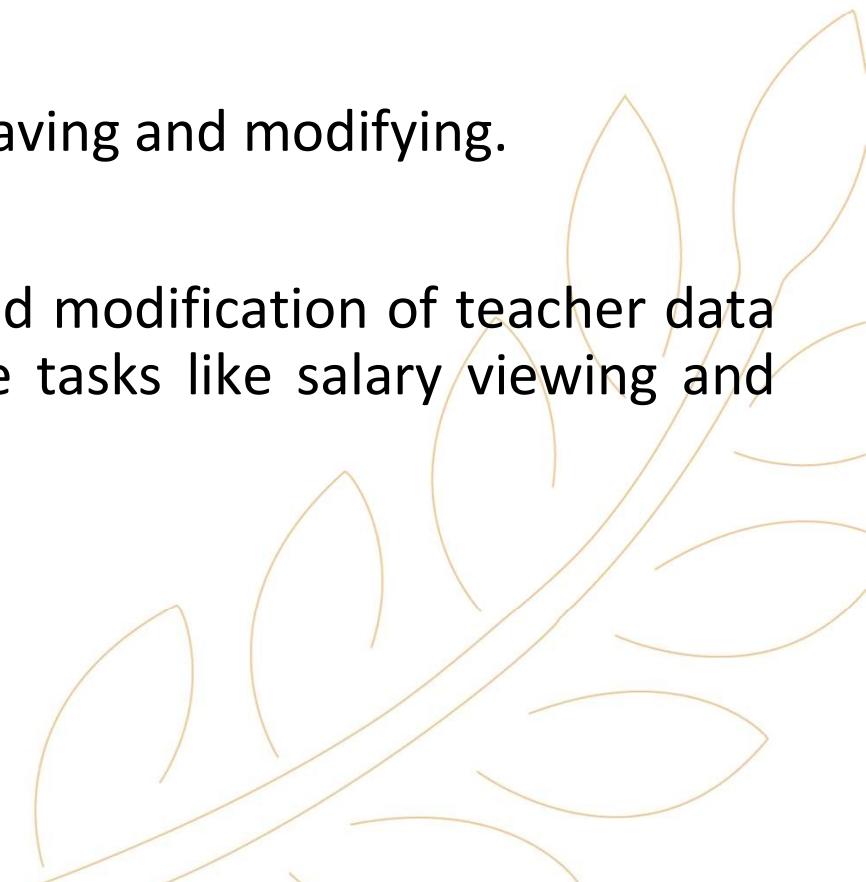
- **Role:** Defines an abstract superclass for record management.
- **Purpose:** Ensures consistency in the implementation of record management operations across all derived classes, promoting a standardized approach in the object-oriented design.



```
1/*
2 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4 */
5package schoolsystemgse122i;
6
7 /**
8 * 
9 * @author farah
10 */
11 public abstract class Person {
12     protected String name;
13
14     public abstract void addRecord();
15     public abstract void modifyRecord();
16     public abstract void searchRecord();
17     public abstract void deleteRecord();
18     public abstract String getName();
19 }
```

Search Results Notifications

6. Teacher



- **Role:** Manages teacher records, including saving and modifying.
- **Purpose:** Handles the storage, retrieval, and modification of teacher data in the database, supporting administrative tasks like salary viewing and record updates with precision.

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

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

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

    @Override
    public void addRecord() {
        String dayStr = JOptionPane.showInputDialog("Enter day of joining:");
        String month = JOptionPane.showInputDialog("Enter month of joining:");
        String yearStr = JOptionPane.showInputDialog("Enter year of joining:");
        String name = JOptionPane.showInputDialog("Enter teacher name:");
        String position = JOptionPane.showInputDialog("Enter position:");
        String salaryStr = JOptionPane.showInputDialog("Enter salary:");

        try {
            int day = Integer.parseInt(dayStr);
            int year = Integer.parseInt(yearStr);
            CustomDate date = new CustomDate(day, month, year);
            if (!date.chkDate()) throw new Exception("Invalid date!");

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

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

```
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 }
```

The screenshot shows a Java IDE interface with a green header bar. The main area displays a Java code editor containing three methods: `searchRecord()`, `deleteRecord()`, and `getName()`. The code uses JOptionPane dialogs for user input and message output, and it interacts with a `dbHandler` class to perform database operations. The code editor has a toolbar at the top with various icons for file operations, search, and navigation.

```
6 @Override
7 public void searchRecord() {
8     String name = JOptionPane.showInputDialog("Enter name to search:");
9     try {
10         var rs = dbHandler.searchTeacher(name);
11         if (rs.next()) {
12             String[] dateParts = rs.getString("date").split(" ");
13             CustomDate date = new CustomDate(Integer.parseInt(dateParts[0]), dateParts[1], Integer.parseInt(dateParts[2]));
14             JOptionPane.showMessageDialog(null, date + ", Name: " + rs.getString("name") + ", Position: " + rs.getString("position") + ", Salary: " + rs.getDouble("salary"));
15         } else {
16             JOptionPane.showMessageDialog(null, "No teacher found with Name: " + name);
17         }
18     } catch (Exception e) {
19         JOptionPane.showMessageDialog(null, "Error searching teacher: " + e.getMessage());
20     }
21 }
22
23 @Override
24 public void deleteRecord() {
25     String name = JOptionPane.showInputDialog("Enter name to delete:");
26     try {
27         dbHandler.deleteTeacher(name);
28         JOptionPane.showMessageDialog(null, "Teacher deleted successfully!");
29     } catch (Exception e) {
30         JOptionPane.showMessageDialog(null, "Error deleting teacher: " + e.getMessage());
31     }
32 }
33
34 @Override
35 public String getName() {
36     return name;
37 }
```

At the bottom of the screen, there is a status bar with the text "Activate Windows Go to Settings to activate Windows." and a system tray icon for "INS Windows".

```
String[] dateParts = rs.getString("date").split(" ");
CustomDate date = new CustomDate(Integer.parseInt(dateParts[0]), dateParts[1], Integer.parseInt(dateParts[2]));
JOptionPane.showMessageDialog(null, date + ", Name: " + rs.getString("name") + ", Position: " + rs.getString("position") + ", Salary: " + rs.getDouble("salary"));
} else {
    JOptionPane.showMessageDialog(null, "No teacher found with Name: " + name);
}
} catch (Exception e) {
    JOptionPane.showMessageDialog(null, "Error searching teacher: " + e.getMessage());
}

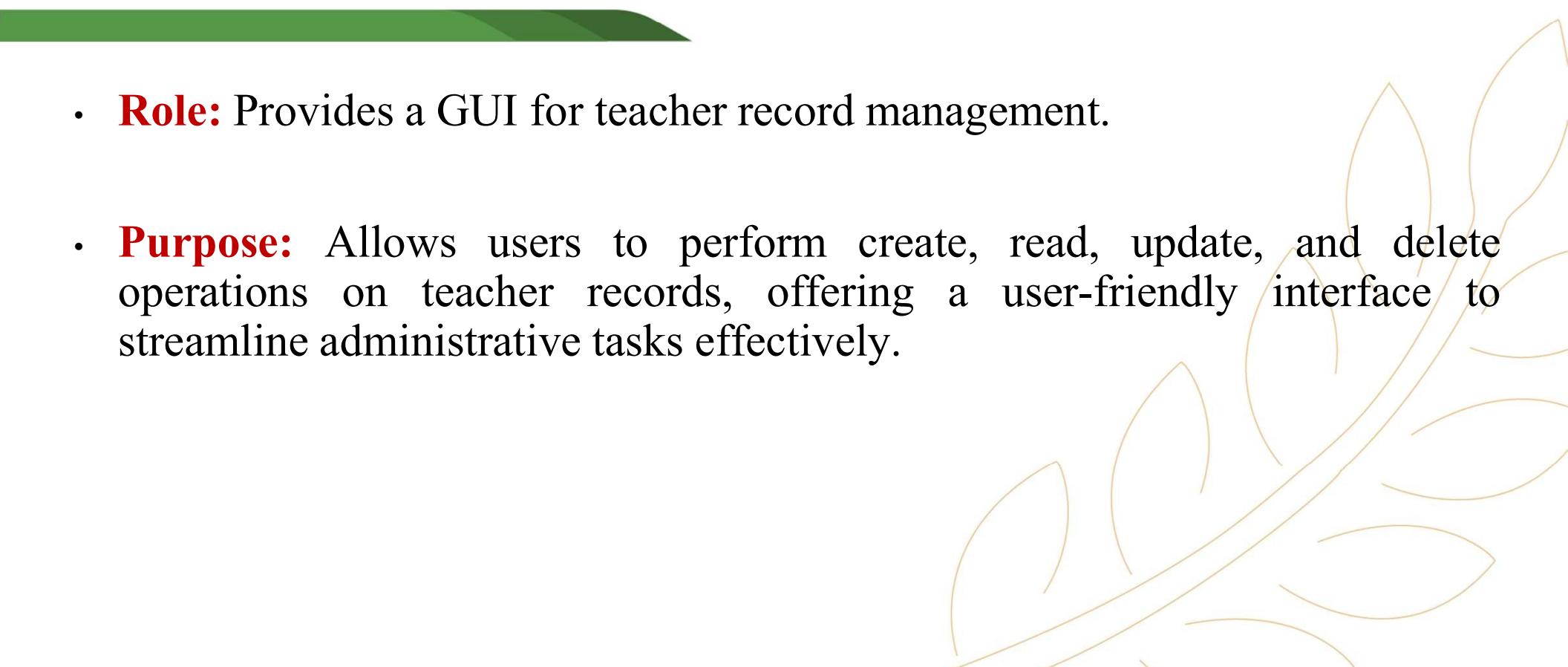
@Override
public void deleteRecord() {
    String name = JOptionPane.showInputDialog("Enter name to delete:");
    try {
        dBHandler.deleteTeacher(name);
        JOptionPane.showMessageDialog(null, "Teacher deleted successfully!");
    } catch (Exception e) {
        JOptionPane.showMessageDialog(null, "Error deleting teacher: " + e.getMessage());
    }
}

@Override
public String getName() {
    return name;
}

@Override
public String toString() {
    return dateOfJoining + ", Name: " + name + ", Position: " + position + ", Salary: " + salary;
}
```

Activate Windows
Go to Settings to activate Windows.

7. TeacherSectionGUI



A decorative element consisting of a solid green horizontal bar at the top, followed by a stylized gold leaf graphic on the right side.

- **Role:** Provides a GUI for teacher record management.
- **Purpose:** Allows users to perform create, read, update, and delete operations on teacher records, offering a user-friendly interface to streamline administrative tasks effectively.

```
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

```
    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);  
}
```

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();
```

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) {
134 }
```

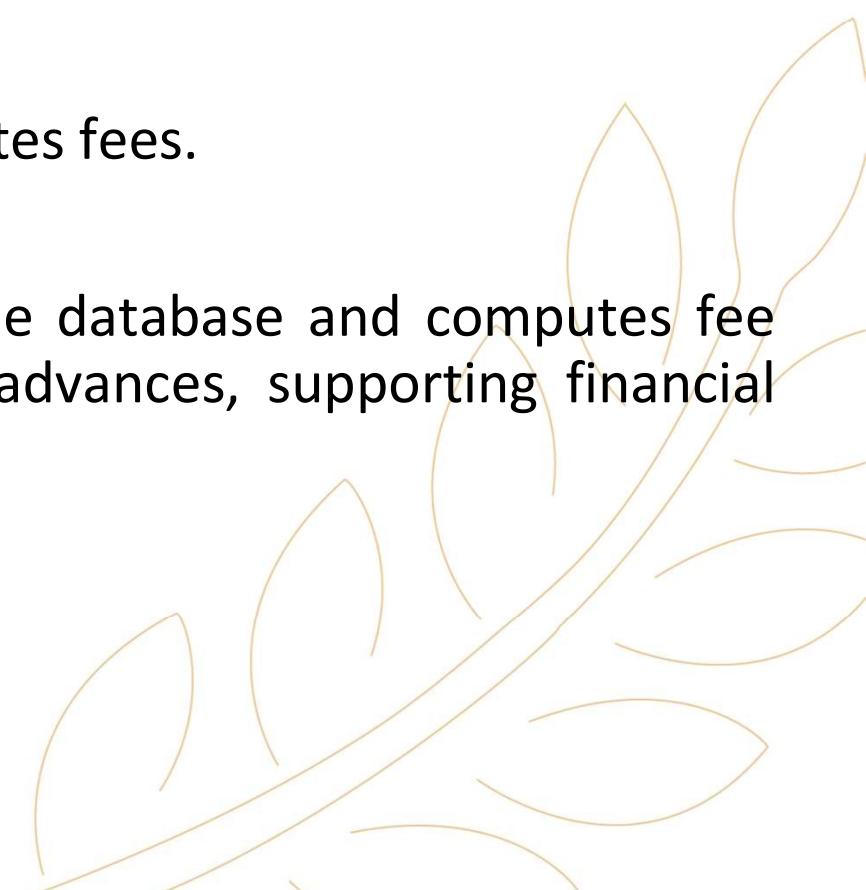
The screenshot shows a Java code editor with the following code:

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

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

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

8. Student



- **Role:** Manages student records and calculates fees.
- **Purpose:** Tracks student information in the database and computes fee details such as due amounts, fines, and advances, supporting financial oversight within the educational system.

```
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) + "\nFine: " + fine + "\nAdvance: " + advance);
    }

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

Activate Windows
Go to Settings to activate Windows.

Search Results Notifications



1:1

INS Win

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

public String getName() {
    return name;
}

public String getStudentClass() {
    return studentClass;
}

public int getRollNo() {
    return rollNo;
}

public double getFeePaid() {
    return feePaid;
}

public CustomDate getDateOfAdmission() {
    return dateOfAdmission;
}

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

Activate Windows
Go to Settings to activate Windows.

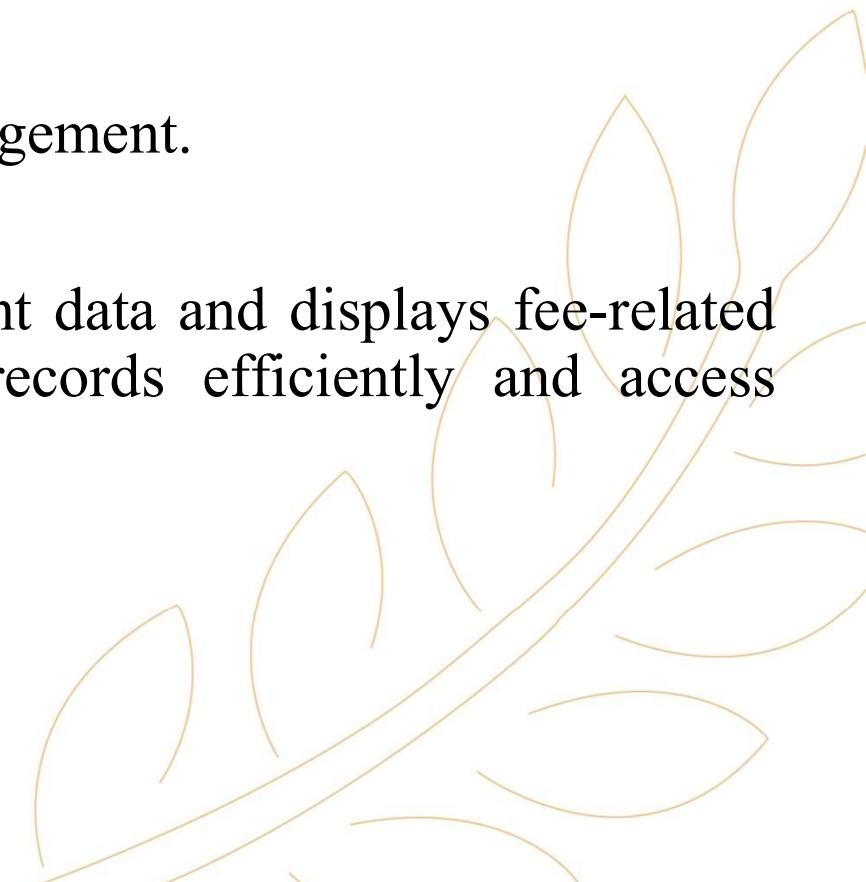
Search Results Notifications



1:1

INS Win

9. StudentSectionGUI



- **Role:** Offers a GUI for student record management.
- **Purpose:** Facilitates the handling of student data and displays fee-related information, enabling users to manage records efficiently and access financial details with ease.

```
3 import javax.swing.*;
4 import java.awt.*;
5
6 public class StudentSectionGUI extends JFrame {
7     private JTextField dayField, monthField, yearField, nameField, classField, rollField, feeField;
8     private JButton saveButton, feeButton, listButton, modifyButton, searchButton, deleteButton;
9
10    public StudentSectionGUI() {
11        setTitle("Student Section");
12        setSize(500, 500);
13        setLayout(new GridLayout(12, 2, 10, 10));
14
15        add(new JLabel("Day of Admission:"));
16        dayField = new JTextField();
17        add(dayField);
18
19        add(new JLabel("Month of Admission:"));
20        monthField = new JTextField();
21        add(monthField);
22
23        add(new JLabel("Year of Admission:"));
24        yearField = new JTextField();
25        add(yearField);
26
27        add(new JLabel("Name:"));
28        nameField = new JTextField();
29        add(nameField);
30    }

```

Search Results Notifications

```
31         add(new JLabel("Class:"));
32         classField = new JTextField();
33         add(classField);
34
35         add(new JLabel("Roll Number:"));
36         rollField = new JTextField();
37         add(rollField);
38
39         add(new JLabel("Fee Paid:"));
40         feeField = new JTextField("0.0");
41         add(feeField);
42
43         saveButton = new JButton("Save");
44         saveButton.addActionListener(e -> saveStudent());
45         add(saveButton);
46
47         feeButton = new JButton("View Fee");
48         feeButton.addActionListener(e -> viewFee());
49         add(feeButton);
50
51         listButton = new JButton("List Students");
52         listButton.addActionListener(e -> displayRecordList());
53         add(listButton);
54
55         modifyButton = new JButton("Modify");
56         modifyButton.addActionListener(e -> modifyRecord());
57         add(modifyButton);
58

```

Search Results Notifications

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

```
88     var dbHandler = new DatabaseHandler();
89     dbHandler.saveStudent(name, studentClass, rollNo, feePaid, date.toString());
90     JOptionPane.showMessageDialog(this, "Student saved: " + date + " " + name);
91     dispose();
92 } catch (Exception e) {
93     JOptionPane.showMessageDialog(this, "Error: " + e.getMessage());
94 }
95 }

96
97 private void viewFee() {
98     try {
99         int rollNo = Integer.parseInt(rollField.getText().trim());
100        double feePaid = Double.parseDouble(feeField.getText().trim());
101        int day = Integer.parseInt(dayField.getText().trim());
102        String month = monthField.getText().trim();
103        int year = Integer.parseInt(yearField.getText().trim());
104        CustomDate date = new CustomDate(day, month, year);
105        if (!date.chkDate()) throw new Exception("Invalid date!");
106
107        Student student = new Student("", "", rollNo, feePaid, date);
108        student.viewFee();
109    } catch (Exception e) {
110        JOptionPane.showMessageDialog(this, "Error: " + e.getMessage());
111    }
112 }

113
114 private void displayRecordList() {
115     try {
116         var dbHandler = new DatabaseHandler();
117         var rs = dbHandler.getStudents();
```

```
18
19     StringBuilder display = new StringBuilder("Student Records:\n");
20     boolean hasRecords = false;
21
22     while (rs.next()) {
23         hasRecords = true;
24         String[] dateParts = rs.getString("date").split(" ");
25         CustomDate date = new CustomDate(Integer.parseInt(dateParts[0]), dateParts[1], Integer.parseInt(dateParts[2]));
26         display.append(date)
27             .append(", Name: ").append(rs.getString("name"))
28             .append(", Class: ").append(rs.getString("studentClass"))
29             .append(", Roll No: ").append(rs.getInt("rollNo"))
30             .append(", Fee Paid: ").append(rs.getDouble("feePaid"))
31             .append("\n");
32     }
33
34     if (!hasRecords) display.append("No records found.");
35     JOptionPane.showMessageDialog(this, display.toString());
36 } catch (Exception e) {
37     JOptionPane.showMessageDialog(this, "Error: " + e.getMessage());
38 }
39
40 private void modifyRecord() {
41     try {
42         int rollNo = Integer.parseInt(JOptionPane.showInputDialog(this, "Enter roll number to modify:"));
43         int day = Integer.parseInt(JOptionPane.showInputDialog(this, "Enter new day of admission:"));
44         String month = JOptionPane.showInputDialog(this, "Enter new month of admission:");
45         int year = Integer.parseInt(JOptionPane.showInputDialog(this, "Enter new year of admission:"));
46         CustomDate date = new CustomDate(day, month, year);
47         if (!date.chkDate()) throw new Exception("Invalid date!");
48     }
49 }
```

```
199     String name = JOptionPane.showInputDialog(this, "Enter new name:");
200     String studentClass = JOptionPane.showInputDialog(this, "Enter new class:");
201     double feePaid = Double.parseDouble(JOptionPane.showInputDialog(this, "Enter new fee paid:"));
202     var dbHandler = new DatabaseHandler();
203     dbHandler.updateStudent(rollNo, name, studentClass, feePaid, date.toString());
204     JOptionPane.showMessageDialog(this, "Record modified: " + date + " " + name);
205 } catch (Exception e) {
206     JOptionPane.showMessageDialog(this, "Error: " + e.getMessage());
207 }
208
209 private void searchRecord() {
210     try {
211         int rollNo = Integer.parseInt(JOptionPane.showInputDialog(this, "Enter roll number to search:"));
212         var dbHandler = new DatabaseHandler();
213         var rs = dbHandler.searchStudent(rollNo);
214         if (rs.next()) {
215             String[] dateParts = rs.getString("date").split(" ");
216             CustomDate date = new CustomDate(Integer.parseInt(dateParts[0]), dateParts[1], Integer.parseInt(dateParts[2]));
217             JOptionPane.showMessageDialog(this, date + ", Name: " + rs.getString("name") + ", Class: " + rs.getString("studentClass") + ", Roll No: " + rs.getInt("rollNo"));
218         } else {
219             JOptionPane.showMessageDialog(this, "No student found with Roll No: " + rollNo);
220         }
221     } catch (Exception e) {
222     JOptionPane.showMessageDialog(this, "Error: " + e.getMessage());
223 }
224
225 private void deleteRecord() {
226     try {
227         int rollNo = Integer.parseInt(JOptionPane.showInputDialog(this, "Enter roll number to delete:"));
228         var dbHandler = new DatabaseHandler();
229     }
230 }
```



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

```
History | Back | Forward | Home | Stop | Refresh | Search | Find | Replace | Cut | Copy | Paste | Delete | Select All | View | Help | Exit |
```

```
57     }
150 }
155
160 private void searchRecord() {
161     try {
162         int rollNo = Integer.parseInt(JOptionPane.showInputDialog(this, "Enter roll number to search:"));
163         var dbHandler = new DatabaseHandler();
164         var rs = dbHandler.searchStudent(rollNo);
165         if (rs.next()) {
166             String[] dateParts = rs.getString("date").split(" ");
167             CustomDate date = new CustomDate(Integer.parseInt(dateParts[0]), dateParts[1], Integer.parseInt(dateParts[2]));
168             JOptionPane.showMessageDialog(this, date + ", Name: " + rs.getString("name") + ", Class: " + rs.getString("studentClass") + ", Roll No: " + rs.getInt("rollNo"));
169         } else {
170             JOptionPane.showMessageDialog(this, "No student found with Roll No: " + rollNo);
171         }
172     } catch (Exception e) {
173         JOptionPane.showMessageDialog(this, "Error: " + e.getMessage());
174     }
175 }

177 private void deleteRecord() {
178     try {
179         int rollNo = Integer.parseInt(JOptionPane.showInputDialog(this, "Enter roll number to delete:"));
180         var dbHandler = new DatabaseHandler();
181         dbHandler.deleteStudent(rollNo);
182         JOptionPane.showMessageDialog(this, "Record deleted.");
183     } catch (Exception e) {
184         JOptionPane.showMessageDialog(this, "Error: " + e.getMessage());
185     }
186 }
```



Thank You!

THE FIRST BRITISH HIGHER EDUCATION IN EGYPT

26th July Mehwar Road Intersection with Wahat Road, 6th of October City, Egypt
Tel: 00238371113 Postal code: 12451 Email: info@msa.edu.eg
Hotline: 16672 Website: www.msa.edu.eg