import javax.swing.\*;

import javax.swing.border.\*;

import javax.swing.table.\*;

import java.awt.\*;

import java.awt.event.\*;

import java.util.\*;

/\*\*

\* Student Management System GUI Application

\* This application provides a user-friendly interface for managing student records,

\* course enrollments, and grades.

\*/

public class StudentManagementSystem extends JFrame {

// Data models

private ArrayList<Student> students = new ArrayList<>();

private ArrayList<Course> courses = new ArrayList<>();

// Main GUI components

private JTabbedPane tabbedPane;

private JPanel studentPanel;

private JPanel coursePanel;

private JPanel enrollmentPanel;

private JPanel gradesPanel;

// Student management components

private JTable studentTable;

private DefaultTableModel studentTableModel;

private JButton addStudentButton;

private JButton updateStudentButton;

private JButton deleteStudentButton;

// Course management components

private JTable courseTable;

private DefaultTableModel courseTableModel;

private JButton addCourseButton;

// Enrollment management components

private JComboBox<String> studentComboBox;

private JComboBox<String> courseComboBox;

private JButton enrollButton;

private JTable enrollmentTable;

private DefaultTableModel enrollmentTableModel;

// Grade management components

private JComboBox<String> studentGradeComboBox;

private JTable studentCourseTable;

private DefaultTableModel studentCourseTableModel;

private JButton assignGradeButton;

/\*\*

\* Constructor to initialize the GUI

\*/

public StudentManagementSystem() {

// Set up the frame

super("Student Management System");

setSize(800, 600);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLocationRelativeTo(null);

// Initialize sample data

initializeSampleData();

// Set up the main components

setupMainComponents();

// Set up the tabbed pane

setupTabbedPane();

// Set up the student management panel

setupStudentPanel();

// Set up the course management panel

setupCoursePanel();

// Set up the enrollment panel

setupEnrollmentPanel();

// Set up the grades panel

setupGradesPanel();

// Add the tabbed pane to the frame

add(tabbedPane);

// Make the frame visible

setVisible(true);

}

/\*\*

\* Initialize sample data for testing

\*/

private void initializeSampleData() {

// Add sample students

students.add(new Student("S001", "John Doe", "Computer Science", "john@example.com"));

students.add(new Student("S002", "Jane Smith", "Mathematics", "jane@example.com"));

students.add(new Student("S003", "Bob Johnson", "Physics", "bob@example.com"));

// Add sample courses

courses.add(new Course("C001", "Introduction to Programming", "CS101", 3));

courses.add(new Course("C002", "Calculus I", "MATH101", 4));

courses.add(new Course("C003", "Physics I", "PHYS101", 4));

// Enroll students in courses

students.get(0).enrollCourse(courses.get(0));

students.get(0).enrollCourse(courses.get(1));

students.get(1).enrollCourse(courses.get(1));

students.get(2).enrollCourse(courses.get(2));

// Assign grades

students.get(0).setGrade(courses.get(0).getCourseCode(), "A");

students.get(0).setGrade(courses.get(1).getCourseCode(), "B+");

students.get(1).setGrade(courses.get(1).getCourseCode(), "A-");

students.get(2).setGrade(courses.get(2).getCourseCode(), "B");

}

/\*\*

\* Set up the main components

\*/

private void setupMainComponents() {

// Create the tabbed pane

tabbedPane = new JTabbedPane();

// Create the panels

studentPanel = new JPanel(new BorderLayout());

coursePanel = new JPanel(new BorderLayout());

enrollmentPanel = new JPanel(new BorderLayout());

gradesPanel = new JPanel(new BorderLayout());

}

/\*\*

\* Set up the tabbed pane

\*/

private void setupTabbedPane() {

// Add the panels to the tabbed pane

tabbedPane.addTab("Students", new ImageIcon(), studentPanel, "Manage Students");

tabbedPane.addTab("Courses", new ImageIcon(), coursePanel, "Manage Courses");

tabbedPane.addTab("Enrollment", new ImageIcon(), enrollmentPanel, "Manage Enrollment");

tabbedPane.addTab("Grades", new ImageIcon(), gradesPanel, "Manage Grades");

}

/\*\*

\* Set up the student management panel

\*/

private void setupStudentPanel() {

// Create the table model

String[] columnNames = {"ID", "Name", "Major", "Email"};

studentTableModel = new DefaultTableModel(columnNames, 0) {

@Override

public boolean isCellEditable(int row, int column) {

return false;

}

};

// Add students to the table model

for (Student student : students) {

Object[] rowData = {student.getId(), student.getName(), student.getMajor(), student.getEmail()};

studentTableModel.addRow(rowData);

}

// Create the table

studentTable = new JTable(studentTableModel);

studentTable.setSelectionMode(ListSelectionModel.SINGLE\_SELECTION);

studentTable.getTableHeader().setReorderingAllowed(false);

// Create the scroll pane

JScrollPane scrollPane = new JScrollPane(studentTable);

// Create the button panel

JPanel buttonPanel = new JPanel();

addStudentButton = new JButton("Add Student");

updateStudentButton = new JButton("Update Student");

deleteStudentButton = new JButton("Delete Student");

// Add the buttons to the panel

buttonPanel.add(addStudentButton);

buttonPanel.add(updateStudentButton);

buttonPanel.add(deleteStudentButton);

// Add the scroll pane and button panel to the student panel

studentPanel.add(scrollPane, BorderLayout.CENTER);

studentPanel.add(buttonPanel, BorderLayout.SOUTH);

// Add action listeners to the buttons

addStudentButton.addActionListener(e -> addStudent());

updateStudentButton.addActionListener(e -> updateStudent());

deleteStudentButton.addActionListener(e -> deleteStudent());

}

/\*\*

\* Set up the course management panel

\*/

private void setupCoursePanel() {

// Create the table model

String[] columnNames = {"Course Code", "Course Name", "Course ID", "Credits"};

courseTableModel = new DefaultTableModel(columnNames, 0) {

@Override

public boolean isCellEditable(int row, int column) {

return false;

}

};

// Add courses to the table model

for (Course course : courses) {

Object[] rowData = {

course.getCourseCode(),

course.getCourseName(),

course.getCourseId(),

course.getCredits()

};

courseTableModel.addRow(rowData);

}

// Create the table

courseTable = new JTable(courseTableModel);

courseTable.setSelectionMode(ListSelectionModel.SINGLE\_SELECTION);

courseTable.getTableHeader().setReorderingAllowed(false);

// Create the scroll pane

JScrollPane scrollPane = new JScrollPane(courseTable);

// Create the button panel

JPanel buttonPanel = new JPanel();

addCourseButton = new JButton("Add Course");

// Add the button to the panel

buttonPanel.add(addCourseButton);

// Add the scroll pane and button panel to the course panel

coursePanel.add(scrollPane, BorderLayout.CENTER);

coursePanel.add(buttonPanel, BorderLayout.SOUTH);

// Add action listener to the button

addCourseButton.addActionListener(e -> addCourse());

}

/\*\*

\* Set up the enrollment panel

\*/

private void setupEnrollmentPanel() {

// Create the selection panel

JPanel selectionPanel = new JPanel(new GridLayout(3, 2, 10, 10));

selectionPanel.setBorder(BorderFactory.createTitledBorder("Enroll Student in Course"));

// Create the components

JLabel studentLabel = new JLabel("Select Student:");

studentComboBox = new JComboBox<>();

JLabel courseLabel = new JLabel("Select Course:");

courseComboBox = new JComboBox<>();

enrollButton = new JButton("Enroll");

// Add students to the combo box

for (Student student : students) {

studentComboBox.addItem(student.getId() + " - " + student.getName());

}

// Add courses to the combo box

for (Course course : courses) {

courseComboBox.addItem(course.getCourseCode() + " - " + course.getCourseName());

}

// Add the components to the selection panel

selectionPanel.add(studentLabel);

selectionPanel.add(studentComboBox);

selectionPanel.add(courseLabel);

selectionPanel.add(courseComboBox);

selectionPanel.add(new JLabel("")); // Spacer

selectionPanel.add(enrollButton);

// Create the table model

String[] columnNames = { "Student ID", "Student Name", "Course Code", "Course Name" };

enrollmentTableModel = new DefaultTableModel(columnNames, 0) {

@Override

public boolean isCellEditable(int row, int column) {

return false;

}

};

// Add enrollments to the table model

for (Student student : students) {

for (String courseCode : student.getEnrolledCourses()) {

Course course = getCourseByCode(courseCode);

if (course != null) {

Object[] rowData = {

student.getId(),

student.getName(),

course.getCourseCode(),

course.getCourseName()

};

enrollmentTableModel.addRow(rowData);

}

}

}

// Create the table

enrollmentTable = new JTable(enrollmentTableModel);

enrollmentTable.setSelectionMode(ListSelectionModel.SINGLE\_SELECTION);

enrollmentTable.getTableHeader().setReorderingAllowed(false);

// Create the scroll pane

JScrollPane scrollPane = new JScrollPane(enrollmentTable);

scrollPane.setBorder(BorderFactory.createTitledBorder("Current Enrollments"));

// \*\*✅ FIX: Create the button panel\*\*

JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT)); // This line fixes the issue

// Create a "De-Enroll" button

JButton deEnrollButton = new JButton("De-Enroll");

buttonPanel.add(deEnrollButton); // Now it works!

// Add action listener to the "De-Enroll" button

deEnrollButton.addActionListener(e -> deEnrollStudent());

// Add the components to the enrollment panel

enrollmentPanel.setLayout(new BorderLayout());

enrollmentPanel.add(selectionPanel, BorderLayout.NORTH);

enrollmentPanel.add(scrollPane, BorderLayout.CENTER);

enrollmentPanel.add(buttonPanel, BorderLayout.SOUTH); // Add button panel at the bottom

// Add action listener to the enroll button

enrollButton.addActionListener(e -> enrollStudent());

}

/\*\*

\* De-enroll a student from a course

\*/

private void deEnrollStudent() {

// Get the selected row in the enrollment table

int selectedRow = enrollmentTable.getSelectedRow();

// Check if a row is selected

if (selectedRow == -1) {

JOptionPane.showMessageDialog(this, "Please select an enrollment to de-enroll.", "Error",

JOptionPane.ERROR\_MESSAGE);

return;

}

// Get the student ID and course code from the selected row

String studentId = (String) enrollmentTableModel.getValueAt(selectedRow, 0);

String courseCode = (String) enrollmentTableModel.getValueAt(selectedRow, 2);

// Get the student

Student student = getStudentById(studentId);

if (student == null) {

JOptionPane.showMessageDialog(this, "Student not found.", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

// Confirm the de-enrollment

int result = JOptionPane.showConfirmDialog(this,

"Are you sure you want to de-enroll this student from the course?", "Confirm De-Enrollment",

JOptionPane.YES\_NO\_OPTION);

// Process the result

if (result == JOptionPane.YES\_OPTION) {

// Remove the course from the student's enrolled courses

student.getEnrolledCourses().remove(courseCode);

// Remove the grade for the course (if any)

student.setGrade(courseCode, null);

// Remove the row from the enrollment table

enrollmentTableModel.removeRow(selectedRow);

// Update the student course table if needed

if (studentGradeComboBox.getSelectedItem() != null

&& studentGradeComboBox.getSelectedItem().toString().startsWith(studentId)) {

updateStudentCourseTable();

}

// Show a success message

JOptionPane.showMessageDialog(this, "Student de-enrolled successfully.", "Success",

JOptionPane.INFORMATION\_MESSAGE);

}

}

/\*\*

\* Set up the grades panel

\*/

private void setupGradesPanel() {

// Create the top panel

JPanel topPanel = new JPanel(new BorderLayout());

// Create the selection panel

JPanel selectionPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));

JLabel studentLabel = new JLabel("Select Student:");

studentGradeComboBox = new JComboBox<>();

// Add students to the combo box

for (Student student : students) {

studentGradeComboBox.addItem(student.getId() + " - " + student.getName());

}

// Add the components to the selection panel

selectionPanel.add(studentLabel);

selectionPanel.add(studentGradeComboBox);

// Create the table model

String[] columnNames = { "Course Code", "Course Name", "Grade" };

studentCourseTableModel = new DefaultTableModel(columnNames, 0) {

@Override

public boolean isCellEditable(int row, int column) {

return column == 2; // Only allow editing the grade column

}

};

// Create the table

studentCourseTable = new JTable(studentCourseTableModel);

studentCourseTable.setSelectionMode(ListSelectionModel.SINGLE\_SELECTION);

studentCourseTable.getTableHeader().setReorderingAllowed(false);

// Add a mouse listener to the table for editing grades

studentCourseTable.addMouseListener(new MouseAdapter() {

@Override

public void mouseClicked(MouseEvent e) {

int row = studentCourseTable.rowAtPoint(e.getPoint());

int col = studentCourseTable.columnAtPoint(e.getPoint());

// Check if the grade column (column 2) was clicked

// if (col == 2) {

// Get the selected student and course

String studentId = (String) studentGradeComboBox.getSelectedItem();

String courseCode = (String) studentCourseTableModel.getValueAt(row, 0);

// Open a modal dialog to edit the grade

editGradeModal(studentId, courseCode, row);

// }

}

});

// Create the scroll pane

JScrollPane scrollPane = new JScrollPane(studentCourseTable);

// Create the button panel

JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));

assignGradeButton = new JButton("Save Grades");

buttonPanel.add(assignGradeButton);

// Add the components to the top panel

topPanel.add(selectionPanel, BorderLayout.WEST);

topPanel.add(buttonPanel, BorderLayout.EAST);

// Add the components to the grades panel

gradesPanel.add(topPanel, BorderLayout.NORTH);

gradesPanel.add(scrollPane, BorderLayout.CENTER);

// Add action listeners

studentGradeComboBox.addActionListener(e -> updateStudentCourseTable());

assignGradeButton.addActionListener(e -> saveGrades());

// Initialize the student course table

if (studentGradeComboBox.getItemCount() > 0) {

updateStudentCourseTable();

}

}

/\*\*

\* Open a modal dialog to edit a grade

\*/

private void editGradeModal(String studentId, String courseCode, int row) {

// Create a panel for the input field

JPanel panel = new JPanel(new GridLayout(1, 2, 10, 10));

JLabel gradeLabel = new JLabel("Select Grade:");

// Define the available grades

String[] availableGrades = {"A", "B", "C", "D", "F"};

// Create a combo box with the available grades

JComboBox<String> gradeComboBox = new JComboBox<>(availableGrades);

// Pre-select the current grade

String currentGrade = (String) studentCourseTableModel.getValueAt(row, 2);

gradeComboBox.setSelectedItem(currentGrade);

// Add components to the panel

panel.add(gradeLabel);

panel.add(gradeComboBox);

// Show the dialog

int result = JOptionPane.showConfirmDialog(this, panel, "Edit Grade", JOptionPane.OK\_CANCEL\_OPTION);

// Process the result

if (result == JOptionPane.OK\_OPTION) {

// Get the selected grade

String newGrade = (String) gradeComboBox.getSelectedItem();

// Update the grade in the table and data model

studentCourseTableModel.setValueAt(newGrade, row, 2);

// Update the grade in the student's record

Student student = getStudentById(studentId.split(" - ")[0]);

if (student != null) {

student.setGrade(courseCode, newGrade);

}

// Show a success message

JOptionPane.showMessageDialog(this, "Grade updated successfully.", "Success",

JOptionPane.INFORMATION\_MESSAGE);

}

}

/\*\*

\* Add a new student

\*/

private void addStudent() {

// Create a panel for the input fields

JPanel panel = new JPanel(new GridLayout(4, 2, 10, 10));

// Create the input fields

JTextField idField = new JTextField();

JTextField nameField = new JTextField();

JTextField majorField = new JTextField();

JTextField emailField = new JTextField();

// Add the input fields to the panel

panel.add(new JLabel("ID:"));

panel.add(idField);

panel.add(new JLabel("Name:"));

panel.add(nameField);

panel.add(new JLabel("Major:"));

panel.add(majorField);

panel.add(new JLabel("Email:"));

panel.add(emailField);

// Show the dialog

int result = JOptionPane.showConfirmDialog(this, panel, "Add Student", JOptionPane.OK\_CANCEL\_OPTION);

// Process the result

if (result == JOptionPane.OK\_OPTION) {

// Get the input values

String id = idField.getText().trim();

String name = nameField.getText().trim();

String major = majorField.getText().trim();

String email = emailField.getText().trim();

// Validate the input

if (id.isEmpty() || name.isEmpty() || major.isEmpty() || email.isEmpty()) {

JOptionPane.showMessageDialog(this, "Please fill in all fields.", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

// Check if the ID already exists

for (Student student : students) {

if (student.getId().equals(id)) {

JOptionPane.showMessageDialog(this, "Student ID already exists.", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

}

// Create the student

Student student = new Student(id, name, major, email);

// Add the student to the list

students.add(student);

// Add the student to the table

Object[] rowData = {id, name, major, email};

studentTableModel.addRow(rowData);

// Add the student to the combo boxes

String item = id + " - " + name;

studentComboBox.addItem(item);

studentGradeComboBox.addItem(item);

// Show a success message

JOptionPane.showMessageDialog(this, "Student added successfully.", "Success", JOptionPane.INFORMATION\_MESSAGE);

}

}

/\*\*

\* Update an existing student

\*/

private void updateStudent() {

// Get the selected row

int selectedRow = studentTable.getSelectedRow();

// Check if a row is selected

if (selectedRow == -1) {

JOptionPane.showMessageDialog(this, "Please select a student to update.", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

// Get the selected student

String id = (String) studentTableModel.getValueAt(selectedRow, 0);

Student student = getStudentById(id);

// Create a panel for the input fields

JPanel panel = new JPanel(new GridLayout(4, 2, 10, 10));

// Create the input fields

JTextField idField = new JTextField(id);

idField.setEditable(false);

JTextField nameField = new JTextField(student.getName());

JTextField majorField = new JTextField(student.getMajor());

JTextField emailField = new JTextField(student.getEmail());

// Add the input fields to the panel

panel.add(new JLabel("ID:"));

panel.add(idField);

panel.add(new JLabel("Name:"));

panel.add(nameField);

panel.add(new JLabel("Major:"));

panel.add(majorField);

panel.add(new JLabel("Email:"));

panel.add(emailField);

// Show the dialog

int result = JOptionPane.showConfirmDialog(this, panel, "Update Student", JOptionPane.OK\_CANCEL\_OPTION);

// Process the result

if (result == JOptionPane.OK\_OPTION) {

// Get the input values

String name = nameField.getText().trim();

String major = majorField.getText().trim();

String email = emailField.getText().trim();

// Validate the input

if (name.isEmpty() || major.isEmpty() || email.isEmpty()) {

JOptionPane.showMessageDialog(this, "Please fill in all fields.", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

// Update the student

student.setName(name);

student.setMajor(major);

student.setEmail(email);

// Update the table

studentTableModel.setValueAt(name, selectedRow, 1);

studentTableModel.setValueAt(major, selectedRow, 2);

studentTableModel.setValueAt(email, selectedRow, 3);

// Update the combo boxes

String item = id + " - " + name;

updateComboBoxItem(studentComboBox, selectedRow, item);

updateComboBoxItem(studentGradeComboBox, selectedRow, item);

// Update the enrollment table

for (int i = 0; i < enrollmentTableModel.getRowCount(); i++) {

if (enrollmentTableModel.getValueAt(i, 0).equals(id)) {

enrollmentTableModel.setValueAt(name, i, 1);

}

}

// Show a success message

JOptionPane.showMessageDialog(this, "Student updated successfully.", "Success", JOptionPane.INFORMATION\_MESSAGE);

}

}

/\*\*

\* Delete a student

\*/

private void deleteStudent() {

// Get the selected row

int selectedRow = studentTable.getSelectedRow();

// Check if a row is selected

if (selectedRow == -1) {

JOptionPane.showMessageDialog(this, "Please select a student to delete.", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

// Get the selected student

String id = (String) studentTableModel.getValueAt(selectedRow, 0);

Student student = getStudentById(id);

// Confirm the deletion

int result = JOptionPane.showConfirmDialog(this, "Are you sure you want to delete this student?", "Confirm Deletion", JOptionPane.YES\_NO\_OPTION);

// Process the result

if (result == JOptionPane.YES\_OPTION) {

// Remove the student from the list

students.remove(student);

// Remove the student from the table

studentTableModel.removeRow(selectedRow);

// Remove the student from the combo boxes

studentComboBox.removeItemAt(selectedRow);

studentGradeComboBox.removeItemAt(selectedRow);

// Remove the student's enrollments from the enrollment table

for (int i = enrollmentTableModel.getRowCount() - 1; i >= 0; i--) {

if (enrollmentTableModel.getValueAt(i, 0).equals(id)) {

enrollmentTableModel.removeRow(i);

}

}

// Show a success message

JOptionPane.showMessageDialog(this, "Student deleted successfully.", "Success", JOptionPane.INFORMATION\_MESSAGE);

}

}

/\*\*

\* Add a new course

\*/

private void addCourse() {

// Create a panel for the input fields

JPanel panel = new JPanel(new GridLayout(4, 2, 10, 10));

// Create the input fields

JTextField codeField = new JTextField();

JTextField nameField = new JTextField();

JTextField idField = new JTextField();

JTextField creditsField = new JTextField();

// Add the input fields to the panel

panel.add(new JLabel("Course Code:"));

panel.add(codeField);

panel.add(new JLabel("Course Name:"));

panel.add(nameField);

panel.add(new JLabel("Course ID:"));

panel.add(idField);

panel.add(new JLabel("Credits:"));

panel.add(creditsField);

// Show the dialog

int result = JOptionPane.showConfirmDialog(this, panel, "Add Course", JOptionPane.OK\_CANCEL\_OPTION);

// Process the result

if (result == JOptionPane.OK\_OPTION) {

// Get the input values

String code = codeField.getText().trim();

String name = nameField.getText().trim();

String id = idField.getText().trim();

String creditsStr = creditsField.getText().trim();

// Validate the input

if (code.isEmpty() || name.isEmpty() || id.isEmpty() || creditsStr.isEmpty()) {

JOptionPane.showMessageDialog(this, "Please fill in all fields.", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

// Check if the course code already exists

for (Course course : courses) {

if (course.getCourseCode().equals(code)) {

JOptionPane.showMessageDialog(this, "Course code already exists.", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

}

// Parse the credits

int credits;

try {

credits = Integer.parseInt(creditsStr);

} catch (NumberFormatException e) {

JOptionPane.showMessageDialog(this, "Credits must be a number.", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

// Create the course

Course course = new Course(code, name, id, credits);

// Add the course to the list

courses.add(course);

// Add the course to the table

Object[] rowData = {code, name, id, credits};

courseTableModel.addRow(rowData);

// Add the course to the combo box

courseComboBox.addItem(code + " - " + name);

// Show a success message

JOptionPane.showMessageDialog(this, "Course added successfully.", "Success", JOptionPane.INFORMATION\_MESSAGE);

}

}

/\*\*

\* Enroll a student in a course

\*/

private void enrollStudent() {

// Get the selected student

int studentIndex = studentComboBox.getSelectedIndex();

if (studentIndex == -1) {

JOptionPane.showMessageDialog(this, "Please select a student.", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

// Get the selected course

int courseIndex = courseComboBox.getSelectedIndex();

if (courseIndex == -1) {

JOptionPane.showMessageDialog(this, "Please select a course.", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

// Get the student and course

Student student = students.get(studentIndex);

Course course = courses.get(courseIndex);

// Check if the student is already enrolled in the course

if (student.getEnrolledCourses().contains(course.getCourseCode())) {

JOptionPane.showMessageDialog(this, "Student is already enrolled in this course.", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

// Enroll the student in the course

student.enrollCourse(course);

// Add the enrollment to the table

Object[] rowData = {student.getId(), student.getName(), course.getCourseCode(), course.getCourseName()};

enrollmentTableModel.addRow(rowData);

// Update the student course table if needed

if (studentGradeComboBox.getSelectedIndex() == studentIndex) {

updateStudentCourseTable();

}

// Show a success message

JOptionPane.showMessageDialog(this, "Student enrolled successfully.", "Success", JOptionPane.INFORMATION\_MESSAGE);

}

/\*\*

\* Update the student course table

\*/

private void updateStudentCourseTable() {

// Clear the table

studentCourseTableModel.setRowCount(0);

// Get the selected student

int selectedIndex = studentGradeComboBox.getSelectedIndex();

if (selectedIndex == -1) {

return;

}

// Get the student

Student student = students.get(selectedIndex);

// Add the student's courses to the table

for (String courseCode : student.getEnrolledCourses()) {

Course course = getCourseByCode(courseCode);

if (course != null) {

String grade = student.getGrade(courseCode);

Object[] rowData = {courseCode, course.getCourseName(), grade != null ? grade : ""};

studentCourseTableModel.addRow(rowData);

}

}

}

/\*\*

\* Save the grades for the selected student

\*/

private void saveGrades() {

// Get the selected student

int selectedIndex = studentGradeComboBox.getSelectedIndex();

if (selectedIndex == -1) {

JOptionPane.showMessageDialog(this, "Please select a student.", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

// Get the student

Student student = students.get(selectedIndex);

// Save the grades

for (int i = 0; i < studentCourseTableModel.getRowCount(); i++) {

String courseCode = (String) studentCourseTableModel.getValueAt(i, 0);

String grade = (String) studentCourseTableModel.getValueAt(i, 2);

student.setGrade(courseCode, grade);

}

// Show a success message

JOptionPane.showMessageDialog(this, "Grades saved successfully.", "Success", JOptionPane.INFORMATION\_MESSAGE);

}

/\*\*

\* Get a student by ID

\*/

private Student getStudentById(String id) {

for (Student student : students) {

if (student.getId().equals(id)) {

return student;

}

}

return null;

}

/\*\*

\* Get a course by code

\*/

private Course getCourseByCode(String code) {

for (Course course : courses) {

if (course.getCourseCode().equals(code)) {

return course;

}

}

return null;

}

/\*\*

\* Update an item in a combo box

\*/

private void updateComboBoxItem(JComboBox<String> comboBox, int index, String item) {

comboBox.removeItemAt(index);

comboBox.insertItemAt(item, index);

comboBox.setSelectedIndex(index);

}

/\*\*

\* Main method to start the application

\*/

public static void main(String[] args) {

// Use the system look and feel

try {

UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());

} catch (Exception e) {

e.printStackTrace();

}

// Create the application

SwingUtilities.invokeLater(() -> new StudentManagementSystem());

}

}

/\*\*

\* Student class to represent a student

\*/

class Student {

private String id;

private String name;

private String major;

private String email;

private ArrayList<String> enrolledCourses;

private HashMap<String, String> grades;

/\*\*

\* Constructor to initialize a student

\*/

public Student(String id, String name, String major, String email) {

this.id = id;

this.name = name;

this.major = major;

this.email = email;

this.enrolledCourses = new ArrayList<>();

this.grades = new HashMap<>();

}

/\*\*

\* Get the student's ID

\*/

public String getId() {

return id;

}

/\*\*

\* Get the student's name

\*/

public String getName() {

return name;

}

/\*\*

\* Set the student's name

\*/

public void setName(String name) {

this.name = name;

}

/\*\*

\* Get the student's major

\*/

public String getMajor() {

return major;

}

/\*\*

\* Set the student's major

\*/

public void setMajor(String major) {

this.major = major;

}

/\*\*

\* Get the student's email

\*/

public String getEmail() {

return email;

}

/\*\*

\* Set the student's email

\*/

public void setEmail(String email) {

this.email = email;

}

/\*\*

\* Get the student's enrolled courses

\*/

public ArrayList<String> getEnrolledCourses() {

return enrolledCourses;

}

/\*\*

\* Enroll the student in a course

\*/

public void enrollCourse(Course course) {

enrolledCourses.add(course.getCourseCode());

}

/\*\*

\* Get the student's grade for a course

\*/

public String getGrade(String courseCode) {

return grades.get(courseCode);

}

/\*\*

\* Set the student's grade for a course

\*/

public void setGrade(String courseCode, String grade) {

grades.put(courseCode, grade);

}

}

/\*\*

\* Course class to represent a course

\*/

class Course {

private String courseCode;

private String courseName;

private String courseId;

private int credits;

/\*\*

\* Constructor to initialize a course

\*/

public Course(String courseCode, String courseName, String courseId, int credits) {

this.courseCode = courseCode;

this.courseName = courseName;

this.courseId = courseId;

this.credits = credits;

}

/\*\*

\* Get the course code

\*/

public String getCourseCode() {

return courseCode;

}

/\*\*

\* Get the course name

\*/

public String getCourseName() {

return courseName;

}

/\*\*

\* Get the course ID

\*/

public String getCourseId() {

return courseId;

}

/\*\*

\* Get the course credits

\*/

public int getCredits() {

return credits;

}

}

# **Student Management System - Documentation**

## **1. GUI Components**

The system uses **Java Swing** for a user-friendly interface. Key components:

* **JFrame** – Main window.
* **JPanel** – Organizes sections (enrollment, grades).
* **JComboBox** – Dropdowns for student, course, and grade selection.
* **JButton** – Triggers actions like enrolling or saving grades.
* **JTable** – Displays student enrollments and grades.
* **JOptionPane** – Handles alerts and input dialogs.

## **2. Event Handlers**

* **Student Selection for Grading:** Updates course table when a student is chosen.
* **Editing Grades:** Opens a modal for grade selection and updates records.
* **Saving Grades:** Updates student records when "Save Grades" is clicked.
* **Enrolling Students:** Adds a student to a course dynamically.
* **De-Enrolling Students:** Removes a student from a course and updates the table.

## **3. Running the Program**

### **Prerequisites:**

* Install **Java JDK 8+**
* Use **VS Code** or any Java IDE

### **Steps to Run:**

1. **Compile:** javac StudentManagementSystem.java
2. **Run:** java StudentManagementSystem

1. **Usage:**
   * Add students via **"Add Student"**.
   * Enroll students using **"Enroll"**.
   * Assign grades through **grade dropdown** in the table.
   * Remove enrollments via **"De-Enroll"**.

## **4. Design Choices**

* **Dropdowns & Tables:** Simplify data selection and display.
* **Real-Time Updates:** Prevents manual refresh.
* **Error Handling:** Prevents invalid inputs with alerts.

This **Student Management System** ensures efficiency with real-time updates and error handling, making administration easy. 🚀

# ScreenShots

