# **JavaFX Student Management System**

## **Overview**

This project is a JavaFX-based Student Management System that enables administrators to manage student records, enroll students in courses, and assign grades through a user-friendly graphical interface. The application follows object-oriented principles and features multi-window navigation and persistent data storage using .txt files.

## **File List**

1. **MainApp.java** – Launches the application and opens the main menu.
2. **StudentWindow.java** – Manages adding, updating, and viewing student records.
3. **CourseWindow.java** – Handles enrolling students in courses.
4. **GradeWindow.java** – Facilitates grade assignment to enrolled students.
5. **Student.java** – Data model for student information.
6. **Course.java** – Data model for course details.
7. **Grade.java** – Data model for grade records.
8. **StudentDataHandler.java** – Manages student data file operations.
9. **CourseDataHandler.java** – Manages course data file operations.
10. **GradeDataHandler.java** – Manages grade data file operations.

## **GUI Components**

Each window contains the following JavaFX components:

* **Labels**: Used to guide the user and display field names.
* **TextFields**: Allow input of student names, IDs, course names, and grades.
* **Buttons**: Trigger actions like Add, Update, View, Enroll, Assign, etc.
* **TableViews**: Present data dynamically for review and selection.
* **ComboBoxes**: Let users select existing students, courses, or grades from a list.

Layout managers like VBox, HBox, and GridPane are used to create logical and aesthetic arrangements.

## **Event Handling**

The application uses **event-driven programming** to ensure real-time interaction. Each button or input component is attached to an appropriate event handler that:

* Validates inputs
* Performs the requested operation (e.g., add student, assign grade)
* Updates the interface immediately with new information
* Shows confirmation or error dialogs for user feedback

## **Functional Features**

### **1. Student Management**

* Add new students using a form.
* Update existing student records.
* View all students in a TableView.

### **2. Course Enrollment**

* Select a course and enroll students.
* Shows list of available students and courses.

### **3. Grade Assignment**

* Select a student and course, then assign a grade.
* View current grades for enrolled courses.

## **Dynamic Interface Updates**

* Once an operation is performed (e.g., new student added), the corresponding TableView or list updates without restarting the app.
* All changes are reflected instantly in the GUI using observable lists and refresh methods.

## **Error Handling**

* Displays user-friendly error dialogs when:
  + Required fields are empty
  + Duplicate entries are found
  + File operations fail
* Uses try-catch blocks to manage exceptions and keep the application responsive.

## **Data Persistence**

The following .txt files are used:

* **students.txt** – Stores student ID and name.
* **courses.txt** – Stores course names.
* **grades.txt** – Stores student ID, course name, and grade.

Data is loaded from these files at runtime and updated immediately when changes are made.

## **Object-Oriented Design**

* **Encapsulation**: Each class hides internal logic and exposes public methods for interaction.
* **Inheritance**: Not explicitly required, but structure allows extension (e.g., Person superclass can be added).
* **Polymorphism**: Used for handling data writing/reading uniformly across data handlers.

## **Running the Program**

### **Requirements**

* Java 8 or higher with JavaFX support
* An IDE such as IntelliJ IDEA, Eclipse, or NetBeans

### **Steps**

1. Create a new JavaFX project in your IDE.
2. Create separate files for each Java class mentioned.
3. Copy the code from each Java file provided and paste it accordingly.
4. Run MainApp.java to launch the application.
5. Use the buttons in the main window to navigate to different management screens.

## **Design Rationale**

The application uses separate windows for modularity and clarity. This ensures that each responsibility (students, courses, grades) is handled in isolation, making the system more maintainable. Files are used for simplicity and portability over databases, and each class follows clean code and best practices (e.g., meaningful variable names, proper error checks, comments).