Project Title :

School System (ANS)

Team Members :

1. Khaled Maged Adawi 2405428
2. Omar Hatem Helal 2405426
3. Mohamed Ahmad Ismail 2405443
4. Yosra Ayman Fayed 2405342
5. Merna Mohamed Abd EL Hamid 2405752

Instructor :

Dr. Ahmed Saleh

Teaching Assistant :

Eng. Fady

Project Overview

General introduction

This document aims to provide a comprehensive and detailed guide to the design and development of a school management system using object-oriented programming (OOP) with Java. The guide covers the four basic aspects required: object design (OOP), database design and management (SQLite and JDBC), user graphical interface design (Java Swing), and system design using the unified modelling language (UML). These documents were prepared based on specific requirements: the use of Java as a basic programming language, SQLite as a database, and Swing to build the graphical interface, focussing on applying OOP concepts at all stages and explaining them using UML.

The project was carried out using **the Montessori system** as a basic idea based on child development through self-learning and a stimulating environment. The Montessori curriculum is an educational system that aims to comprehensively develop the child through an organized and stimulating environment that allows him to self-learn and obtain practical experience. It is based on the child learning freely within clear limits, choosing activities that suit him or her, whereas the teacher's role is prescriptive rather than didactic. Classes are equipped with interactive educational tools that help the child develop his mental, sensory and linguistic skills. The curriculum encourages independence and is based on respect for individual differences among children.

(Database)

Introduction

This system is based on the SQLite database to efficiently store and retrieve school data. SQLite is a lightweight, embedded relational database that does not require a separate server and stores the entire database in a single file, making it suitable for office applications such as this project.

Tables (4)

Name	Type	Schema
Admin		CREATE TABLE "Admin" ("User" TEXT NOT NULL, "Password" TEXT)
User	TEXT	"User" TEXT NOT NULL
Password	TEXT	"Password" TEXT
Student		CREATE TABLE "Student" ("ID" INTEGER, "Name" TEXT NOT NULL, "DateOfBirth" TEXT, "Email" TEXT NOT NULL, "NationalID" INTEGER, "Address" TEXT, "Phone" TEXT NOT NULL, "Grade" NUMERIC NOT NULL, "Password" TEXT, "Arabic" NUMERIC, "English" NUMERIC, "Mathmatics" NUMERIC, "Science" NUMERIC, "SocialStudies" NUMERIC, "Total" NUMERIC, ArabicAbsence INT, EnglishAbsence INT, MathmaticsAbsence INT, ScienceAbsence INT, SocialStudiesAbsence INT, PRIMARY KEY("ID" AUTOINCREMENT))
ID	INTEGER	"ID" INTEGER
Name	TEXT	"Name" TEXT NOT NULL
DateOfBirth	TEXT	"DateOfBirth" TEXT
Email	TEXT	"Email" TEXT NOT NULL
NationalID	INTEGER	"NationalID" INTEGER
Address	TEXT	"Address" TEXT
Phone	TEXT	"Phone" TEXT NOT NULL
Grade	NUMERIC	"Grade" NUMERIC NOT NULL
Password	TEXT	"Password" TEXT
Arabic	NUMERIC	"Arabic" NUMERIC
English	NUMERIC	"English" NUMERIC
Mathmatics	NUMERIC	"Mathmatics" NUMERIC
Science	NUMERIC	"Science" NUMERIC
SocialStudies	NUMERIC	"SocialStudies" NUMERIC

Total	NUMERIC	"Total" NUMERIC
ArabicAbsence	INT	"ArabicAbsence" INT
EnglishAbsence	INT	"EnglishAbsence" INT
MathmaticsAbsence	INT	"MathmaticsAbsence" INT
ScienceAbsence	INT	"ScienceAbsence" INT
SocialStudiesAbsence	INT	"SocialStudiesAbsence" INT
Teacher		`CREATE TABLE "Teacher" ( "ID" INTEGER, "Name" INTEGER NOT NULL, "Age" INTEGER NOT NULL, "Subject" TEXT NOT NULL, "Salary" NUMERIC, "Password" INTEGER NOT NULL, PRIMARY KEY("ID" AUTOINCREMENT) )`
ID	INTEGER	"ID" INTEGER
Name	INTEGER	"Name" INTEGER NOT NULL
Age	INTEGER	"Age" INTEGER NOT NULL
Subject	TEXT	"Subject" TEXT NOT NULL
Salary	NUMERIC	"Salary" NUMERIC
Password	INTEGER	"Password" INTEGER NOT NULL
sqlite_sequence		`CREATE TABLE sqlite_sequence(name,seq)`
name		"name"
seq		"seq"

Indices (0)

Name	Type	Schema

Views (0)

Name	Type	Schema

Triggers (1)

Name	Type	Schema
update_total		`CREATE TRIGGER update_total AFTER INSERT ON Student FOR EACH ROW BEGIN UPDATE Student SET total = COALESCE(NEW.Arabic, 0) + COALESCE(NEW.English, 0) + COALESCE(NEW.Mathmatics, 0) + COALESCE(NEW.Science, 0) + COALESCE(NEW.SocialStudies, 0) WHERE ID = NEW.ID; END`

Graphical interface (GUI) section using Swing

Introduction

The graphical interface (GUI) is an essential part of facilitating user interaction with the school management system. We will use the Java Swing library to build a desktop application. Swing provides a rich set of components such as windows (JFrames), buttons (JButtons), text fields (JTextFields), tables (JTables), and others, to create interactive interfaces. Based on the specific requirements and functions (login, student management, teachers, attendance, administrator control panel, top 10 students presentation), we will need to design several windows or panels:

1. Login Window (Login.java)

Fields:
Username input field
Password input field
“Log in” button
“Exit” button
Functionality:
On successful login, user data is verified from the User table in the database.
If the user is an Admin, open the Admin.java interface.
If the user is a Teacher, open the Dashboard.java interface.
On "Exit", the application closes.

2. Admin Control Panel (Admin.java)

Functions available to Admin:
Student Management (Student.java) – Add / Modify / Delete / Search students
Teacher Management (Teacher.java) – Add / Modify / Delete / Search teachers
Grade Management (ShowGrades.java) – View and modify student grades
Logout – Returns to the Login.java screen

3. Teacher Control Panel (Dashboard.java)

Functions available to Teacher:
View Students in Assigned Classes – Open Student.java in read-only or limited edit mode
Attendance Registration – Mark attendance (can be included inside Student.java or in a future class)
Enter Grades – Open ShowGrades.java
View Reports – Display student performance reports
Logout – Returns to the Login.java screen

4. Student Management (Student.java)

For Admin: Full access (Add / Edit / Delete / Search)
For Teacher: Limited access (View / Attendance / Grades entry)

5. Teacher Management (Teacher.java)

For Admin only: Add / Modify / Delete / Search teacher records

6. Grades Management (ShowGrades.java)

For Admin/Teacher: Enter and modify student grades
View student performance reports

7. Shared Data (Static.java)

Used to store and access global/static information such as:
Logged-in user information
Current class or subject
Database connection (optional)

OOP Code Analysis

➢ ANS

✓ Type:
Main Class (Application Entry Point)

✓ Purpose:
Acts as the main starting point for the Java application. Its primary function is to launch the graphical user interface, specifically the LogIn window, ensuring it runs on the correct thread (Event Dispatch Thread) for Swing applications.

✓ Key Fields:
♦

✓ Notable Methods:
♦ public static void main(String[] args): The standard entry point for a Java application. It uses SwingUtilities.invokeLater to safely create and display the LogIn GUI component on the Event Dispatch Thread.

➢ Person (Abstract Base Class)

```
public abstract class Person {
```

✓ Type:

Abstract Superclass

✓ Purpose:

Provides a base template for person-related entities within the system (like students and teachers), defining common attributes and behaviors.

✓ Key Fields:

♦ private int id
♦ private String name
♦ private int age
♦ private String email

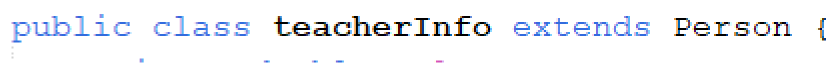
✓ Notable Methods:

♦ Person(): Default constructor.
♦ Person(int id, String name, int age, String email): Parameterized constructor.
♦ setId(int id), setName(String name), setAge(int age), setEmail(String email): Standard setters.
♦ getId(), getName(), getAge(), getEmail(): Standard getters.
♦ toString(): Returns a basic summary (ID, name, age, email).

✓ OOP Principles:

♦ Encapsulation: Private fields with public getters and setters provide controlled access.
♦ Abstraction: Defined as an abstract class, cannot be instantiated directly, serves as a blueprint.

➢ TeacherInfo (extends Person)



```java
public class teacherInfo extends Person {
```

✓ Type:
Concrete Subclass

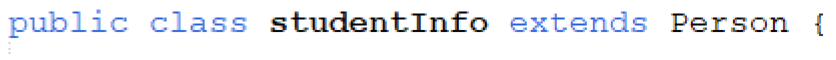
✓ Purpose:
Represents a teacher entity, inheriting common properties from Person and adding teacher-specific details.

✓ Key Fields:
♦ (Inherited from Person: id, name, age, email)
♦ private double salary
♦ private String subject
♦ private String password (Note: Security risk - store hashed password)

✓ Notable Methods:
♦ teacherInfo(): Default constructor.
♦ teacherInfo(int id, String name, int age, String email): Constructor initializing only inherited fields via super().
♦ teacherInfo(int id, String name, int age, double salary, String subject, String email): Constructor initializing inherited and some specific fields (Note: password is not initialized here).
♦ Getters and Setters for salary, subject, password.
♦ toString(): Overridden, correctly includes super.toString() and teacher-specific fields.

✓ OOP Principles:
♦ Inheritance: Extends the Person class.
♦ Encapsulation: Own fields are private with public getters/setters.
♦ Polymorphism: Overrides the toString() method from Person.

➢ StudentInfo (extends Person)



```
public class studentInfo extends Person {
```

✓ Type:

Concrete Subclass

✓ Purpose:

Represents a student entity, inheriting common properties from Person and adding student-specific details.

✓ Key Fields:

♦ (Inherited from Person: id, name, age, email)
♦ private String dateOfBirth
♦ private String nationalID
♦ private String Address (Note: Should be address)
♦ private String Phone (Note: Should be phone)
♦ private Double Grade (Note: Should be grade, consider double vs Double)
♦ private String password (Note: Security risk - store hashed password)

✓ Notable Methods:

♦ studentInfo(): Default constructor.
♦ studentInfo(int id, String name, int age, String email): Constructor initializing only inherited fields via super().
♦ Getters and Setters for all specific fields (dateOfBirth, nationalID, Address, Phone, Grade, password).
♦ toString(): Overridden, but only shows student-specific fields (Recommendation: include super.toString()).

✓ OOP Principles:

♦ Inheritance: Extends the Person class.
♦ Encapsulation: Own fields are private with public getters/setters.

♦ Polymorphism: Overrides the toString() method from Person (though implementation could be improved).

➢ Course

✓ Type:

Regular Class (Represents an entity with relationships)

✓ Purpose:

Models a course, managing its details, enrolled students, assigned teacher, and schedule.

✓ Key Fields:

♦ private int id
♦ private String name
♦ private String description
♦ private List<Student> enrolledStudents (Note: Uses Student type, while studentInfo class exists)
♦ private Teacher teacher (Note: Uses Teacher type, while teacherInfo class exists)
♦ private Schedule schedule

✓ Notable Methods:

♦ Course(int id, String name, String description): Constructor.
♦ getId(), getName(), getDescription(), getEnrolledStudents(): Getters.
♦ enrollStudent(Student student): Adds a student to the list.
♦ removeStudent(Student student): Removes a student from the list.
♦ assignTeacher(Teacher teacher): Setter for the teacher.
♦ setSchedule(Schedule schedule): Setter for the schedule.
♦ toString(): Returns course name and description.

✓ OOP Principles:

♦ Encapsulation: Fields are private, access controlled via getters and specific methods (e.g., enrollStudent).

♦ Composition/Aggregation: Holds references to Schedule and Teacher objects (Composition/Association) and a list of Student objects (Aggregation).

➢ Schedule

✓ Type:
Regular Class (Simple Data Object)

✓ Purpose:
Represents the schedule details for a course (day, start time, end time).

✓ Key Fields:
♦ private String day
♦ private String startTime
♦ private String endTime

✓ Notable Methods:
♦ Schedule(String day, String startTime, String endTime): Constructor initializing all fields.
♦ getDay(), getStartTime(), getEndTime(): Getters only (suggests immutability).
♦ toString(): Returns a formatted schedule string.

✓ OOP Principles:
♦ Encapsulation: Private fields with public getters.

➢ Database Connection

✓ Type:
Utility Class (with static methods)

✓ Purpose:
Handles database interactions (connection, adding students/teachers, retrieving data) for the application, specifically with an SQLite database.

✓ Key Fields:
♦ private static final String URL: Database connection string.
♦ private static Connection connection: Static connection object (Note: problematic design).

✓ Notable Methods:
♦ getConnection(): Static method to get/create a database connection.
♦ addStudent(...): Static method to insert student data into the DB.
♦ addTeacher(...): Static method to insert teacher data into the DB.
♦ getStudentSubjects(String studentID): Static method to retrieve subject/grade data for a student (Returns Swing DefaultTableModel).
♦ loadTop10(): Static method to retrieve top 10 students (Returns Swing DefaultTableModel).
♦ loadGrades(): Empty method stub.
♦ searchStudent(Integer studentID): Empty method stub.