Introduction to Java Programming

CSCS290

Fall 2025

FORMAN CHRISTIAN COLLEGE
(A CHARTERED UNIVERSITY)

Project

Abdul Hanan Khan
271048594

Department of Computer Science
Forman Christian College University

Project Objective

OOP , Database and CRUD

Abstract

Project Summary: The Smart University Management System is a Java based console application designed to digitize administrative and academic operations. It replaces manual record keeping with a centralized database system, allowing efficient management of students, faculty, courses, and finances.

Major Modules:

1. **Registration Module:** Manages CRUD operations for Students, Faculty, and Staff.
2. **Academic Module:** Handles Course creation, Prerequisite mapping, and Student Enrollment.
3. **Grading Module:** Facilitates grade assignment and GPA calculation.
4. **Library Module:** Tracks book inventory, issuing, and return transactions.
5. **Finance Module:** Automates tuition fee calculation and salary processing.
6. **Admin Module:** Generates system wide reports and manages departments.

Tools Used:

- **Language:** Java (JDK 21)
- **Database:** MySQL (via XAMPP)
- **Connectivity:** JDBC (Java Database Connectivity)
- **IDE:** IntelliJ IDEA

Key Features:

- Object Oriented Design (Inheritance, Polymorphism, Encapsulation, Abstraction, Interfaces, Composition, Aggregation, Associations).
- Data Persistence using MySQL.
- Modular Architecture (DAO Pattern) separating logic from data access.
- Error handling for database constraints and user input.

System Overview

Overall System Flow: The application launches a main menu acting as the central hub. Users navigate to specific sub modules (Registration, Finance). Each module accepts user input via the console, processes the request using Java objects (Models), and communicates with the MySQL database through Data Access Objects (DAOs) to fetch or store data.

Why a Database is Used: A relational database (MySQL) is required to ensure data persistence. Without a database, all records (students, grades, transactions) would be lost once the program terminates. The database also ensures data integrity through constraints (Foreign Keys) and allows for complex retrieval operations like generating transcripts via JOIN queries.

How OOP and Modularity Help:

- **Modularity:** The project uses a package based structure (core, database, modules). This separates the User Interface (Console Menus) from the Business Logic (DAOs), making the code easier to debug and maintain.
- **OOP Structure:** The **DAO Pattern** prevents the database code from breaking the OOP structure. The core classes (Student, Course) remain pure Java objects containing only data and behavior, while separate DAO classes handle the SQL logic. Inheritance is used in the Person class, allowing Student and Faculty to share common attributes like name and email without code duplication.

Code Snippets with Explanations

A. JDBC Connection Code

```
public class DatabaseConnection {
   private static final String URL =
"jdbc:mysql://localhost:3306/SmartUniversityDB";
   private static final String USER = "root";
   private static final String PASSWORD = "Wtf6767$";

   public static Connection getConnection() {
      Connection con = null;
      try {
         Class.forName("com.mysql.cj.jdbc.Driver");
         con = DriverManager.getConnection(URL, USER, PASSWORD);
      } catch (ClassNotFoundException | SQLException e) {
         System.out.println("Connection Failed: " + e.getMessage());
      }
      return con;
   }
}
```

Explanation: Class.forName() loads the JDBC driver, allowing the Java application to recognize the MySQL database. DriverManager.getConnection() uses the database URL, username, and password to establish an active connection session, which is returned to the calling method for executing queries.

B. OOP & Polymorphism Example

Java

```
// Parent Class
public abstract class Person {
   protected String name;
   public Person(String name) { this.name = name; }
   public abstract void displayInfo();
}
```

```
// Child Class
public class Student extends Person {
    private String program;
    public Student(String name, String program) {
        super(name);
        this.program = program;
    }
    @Override
    public void displayInfo() {
        System.out.println("Student: " + name + " | Program: " + program);
    }
}
```

Explanation: This snippet demonstrates **Polymorphism** and **Inheritance**. The Student class extends the abstract Person class, inheriting the name field. It overrides the displayInfo() method to provide a specific implementation for students. This allows the system to treat different entities (Students, Faculty) uniformly while executing their specific behaviors.

5. SQL Queries Section

Table Creation Query

SQL

```
CREATE TABLE enrollment (
    enrollment_id INT AUTO_INCREMENT PRIMARY KEY,
    student_id VARCHAR(20),
    course_code VARCHAR(20),
    grade_point DOUBLE,
    FOREIGN KEY (student_id) REFERENCES students(id),
    FOREIGN KEY (course_code) REFERENCES courses(code)
);
```

Insert Query

SQL

```
INSERT INTO students (id, name, age, email, program)
VALUES ('S001', 'Ali Khan', 21, 'ali@test.com', 'CS');
```

Update Query

SQL

```
UPDATE enrollment
SET grade_point = 3.5
WHERE student_id = 'S001' AND course_code = 'CS101';
```

Delete Query

SQL

```
DELETE FROM courses WHERE code = 'CS101';
```

Select Query

SQL

```
SELECT * FROM library_books WHERE available_copies > 0;
```

JOIN Query (System Report)

SQL

```
SELECT s.name, c.title, e.grade_point
FROM enrollment e
JOIN students s ON e.student_id = s.id
JOIN courses c ON e.course_code = c.code
WHERE s.id = 'S001';
```

6. Testing Scenarios

The following scenarios were executed to validate system functionality.

1. Student Enrollment

- **Action:** Validated existence of Student (S001) and Course (CS101). Selected "Enroll" option.
- **Expected Result:** Database record inserted into enrollment table.
- **Actual Result:** " Student enrolled successfully."

2. Faculty Teaching Courses

- **Action:** Added Faculty (F001) with Department Code "CS".
- **Expected Result:** Faculty record linked to Department.
- **Actual Result:** " Faculty added "

3. Grade Update

- **Action:** Admin selected "Grading Module", entered Student ID S001 and Course CS101, and input Grade 3.5.
- **Expected Result:** grade_point column in enrollment table updated from NULL to 3.5.
- **Actual Result:** " Grade updated successfully."

4. Book Issuing and Returning

- **Action:** Selected "Issue Book" for ISBN B999 to Student S001.
- **Expected Result:** available_copies decreased by 1; new record in library_transactions.
- **Actual Result:** " Book Issued."
- **Action:** Returned the same book.
- **Expected Result:** available_copies increased by 1; return_date timestamp updated.
- **Actual Result:** " Book Returned."

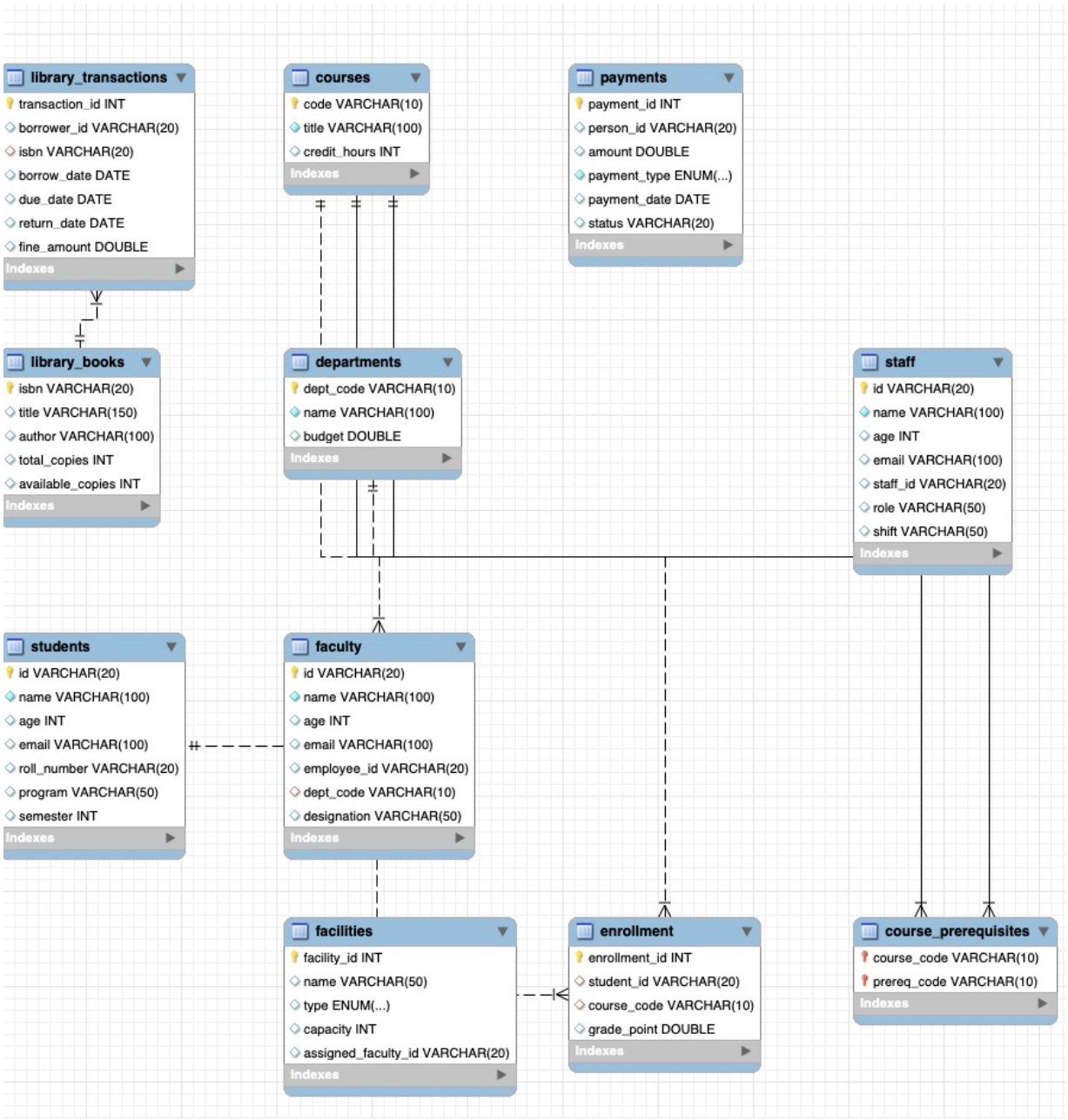
5. Fees Submission

- **Action:** Finance Module calculated tuition based on credit hours (3 credits * $100).
- **Expected Result:** System prompts for $300 payment. Upon confirmation, record inserted into payments.
- **Actual Result:** " Tuition payment recorded."

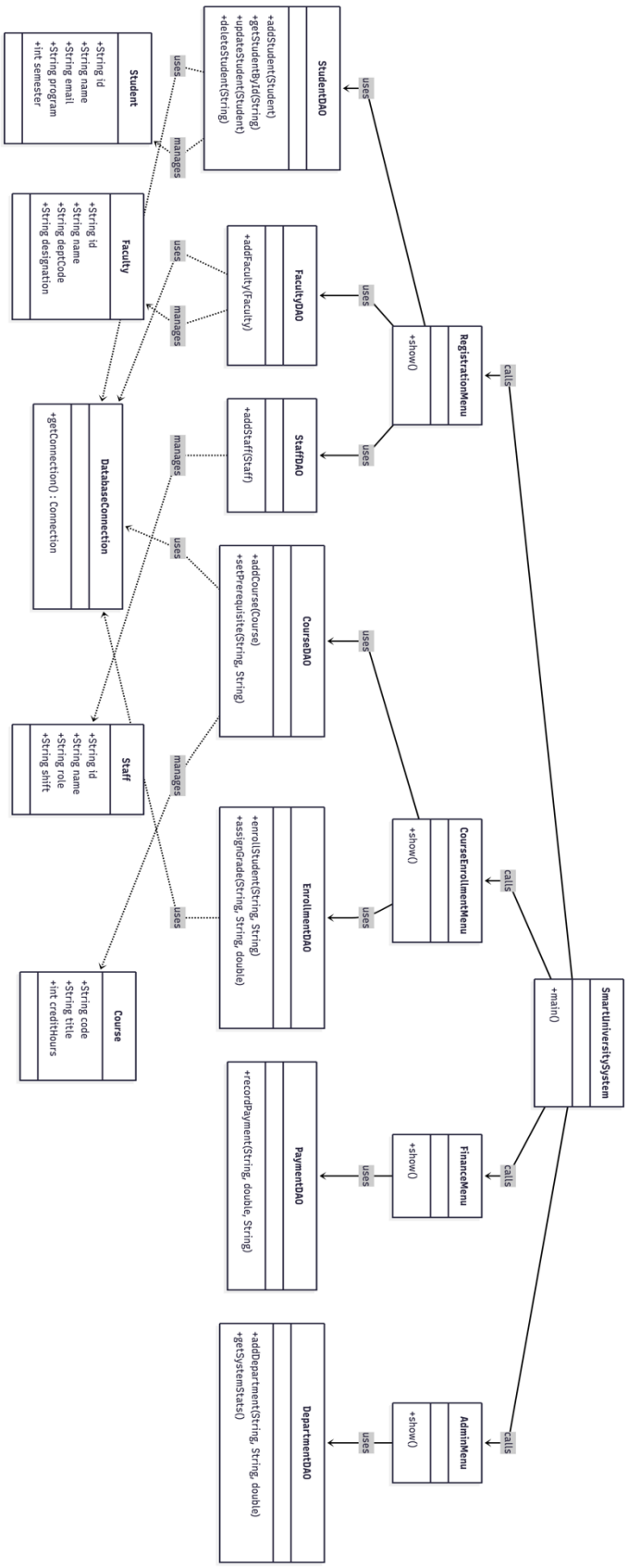
6. Searching and Listing Records

- **Action:** Used "Search Student" with query "Ali".
- **Expected Result:** Displays all students containing "Ali" in ID or Name.
- **Actual Result:** List of matching Student objects displayed in console.

7. ERD


library_transactions
transaction_id INT
borrower_id VARCHAR(20)
isbn VARCHAR(20)
borrow_date DATE
due_date DATE
return_date DATE
fine_amount DOUBLE
Indexes
courses
code VARCHAR(10)
title VARCHAR(100)
credit_hours INT
Indexes
payments
payment_id INT
person_id VARCHAR(20)
amount DOUBLE
payment_type ENUM(...)
payment_date DATE
status VARCHAR(20)
Indexes
library_books
isbn VARCHAR(20)
title VARCHAR(150)
author VARCHAR(100)
total_copies INT
available_copies INT
Indexes
departments
dept_code VARCHAR(10)
name VARCHAR(100)
budget DOUBLE
Indexes
staff
id VARCHAR(20)
name VARCHAR(100)
age INT
email VARCHAR(100)
staff_id VARCHAR(20)
role VARCHAR(50)
shift VARCHAR(50)
Indexes
students
id VARCHAR(20)
name VARCHAR(100)
age INT
email VARCHAR(100)
roll_number VARCHAR(20)
program VARCHAR(50)
semester INT
Indexes
faculty
id VARCHAR(20)
name VARCHAR(100)
age INT
email VARCHAR(100)
employee_id VARCHAR(20)
dept_code VARCHAR(10)
designation VARCHAR(50)
Indexes
facilities
facility_id INT
name VARCHAR(50)
type ENUM(...)
capacity INT
assigned_faculty_id VARCHAR(20)
Indexes
enrollment
enrollment_id INT
student_id VARCHAR(20)
course_code VARCHAR(10)
grade_point DOUBLE
Indexes
course_prerequisites
course_code VARCHAR(10)
prereq_code VARCHAR(10)
Indexes

8. Class Diagram



9. Conclusion

What was Learned: Developing this project provided hands on experience in integrating Object Oriented Programming with a relational database. Key learning outcomes include mastering the JDBC workflow, implementing the Data Access Object (DAO) pattern to maintain code modularity, and handling complex SQL relationships (Foreign Keys) within a Java application.

Challenges: The primary challenge was managing database relationships, specifically preventing foreign key constraint violations when deleting records (deleting a student who has active enrollments). Another challenge was properly mapping SQL ResultSet data back into Java Objects while ensuring null safety. These were resolved by implementing robust error handling and check constraints before deletion.