INTERIM SEMESTER: 25-26

VITYARTHI PROJECT

Name: Utkarsh kulshrestha

Reg No: 22BCE10372

Course and code: Programming in java [CSE-2006]

School: SCOPE

Faculty: Dr. Vishal Singh Bhati

Introduction

The Student Course Registration System is a simple console-based Java application that helps manage basic academic tasks such as adding students, viewing courses, and registering students for available courses. The main aim of this project is to learn and implement Java OOP concepts, exception handling, and JDBC connectivity with a MySQL database.

The system allows data to be stored permanently inside a database so that even after closing the program, information is not lost. This project demonstrates how back-end logic and database work together in a real-world application.

Problem Statement

In many colleges, course registration is still done manually, which leads to missing data, repeated entries, and difficulty in maintaining proper records. Students may also face confusion about available courses and registration status.

There is a need for a simple digital system that allows:

- Storing student information
- Displaying available courses
- Registering students in a structured and safe way
- Maintaining records in a database

This project solves the above issues by implementing a small, efficient Java + MySQL–based registration system.

Functional Requirements

The system must provide the following core functionalities:

User Functions

1. **Add Student**
    a. Enter name and email
    b. Save student into the database
2. **View Courses**
    a. Display all available courses from the database
3. **Register Student for Course**
    a. Enter student ID
    b. Enter course ID

c. Save registration into the database

System Functions

4. **Database Connectivity**
    a. Connect to MySQL using JDBC
    b. Execute SQL operations
5. **Input Validation**
    a. Handle invalid IDs
    b. Handle wrong menu choices

Non-Functional Requirements

Performance Requirements

- System should fetch and display records within 1–2 seconds.
- Database operations should run efficiently for basic queries.

Usability Requirements

- Simple text-based menu
- Easy navigation using numeric choices

Reliability & Safety Requirements

- System should handle invalid input without crashing
- Database password is not exposed publicly

Portability

- Runs on Windows, macOS, or Linux with Java and MySQL installed

System Architecture

The system architecture describes how different parts of the Student Course Registration System interact with each other. It acts as a blueprint that explains the structure, components, and flow of data inside the application. The architecture is designed in a **three-layer model** so that each part of the system has a clear responsibility and the application becomes easier to understand, maintain, and update.

1. Presentation Layer (User Interface Layer)

This is the layer where the user interacts with the system.
In this project, the interface is a **simple console-based menu** that allows the user to:

- Add new students
- View available courses
- Register a student for a course
- Exit the program

Even though the interface is simple, it performs the important job of collecting inputs from the user and displaying outputs.

2. Application Layer (Logic Layer)

This layer contains the **core logic** of the system. It is responsible for:

- Processing user input
- Applying business rules
- Deciding which database operation to perform
- Managing classes like Student, Course, and RegistrationSystem

This layer uses **Object-Oriented Programming (OOP)** concepts such as classes, methods, and data encapsulation.
It acts as a bridge between the user interface and the database.

3. Data Layer (Database Layer)

This layer manages all the data of the system. It includes:

- MySQL database
- Tables for students, courses, and registrations
- SQL queries for inserting, retrieving, or updating data
- JDBC driver for connecting Java to MySQL

Database operations like **INSERT**, **SELECT**, and **UPDATE** are executed in this layer.
The DBConnection class is part of this layer because it handles:

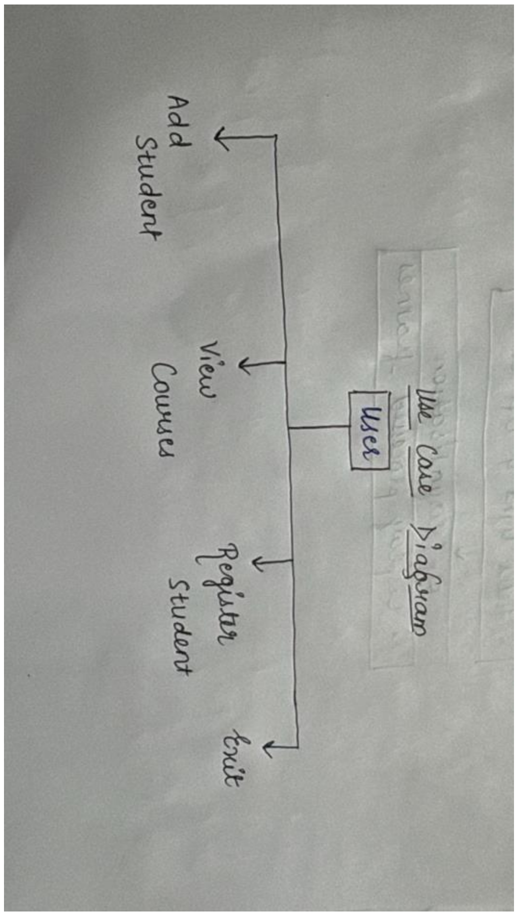
- Opening the connection
- Sending SQL commands
- Closing the connection

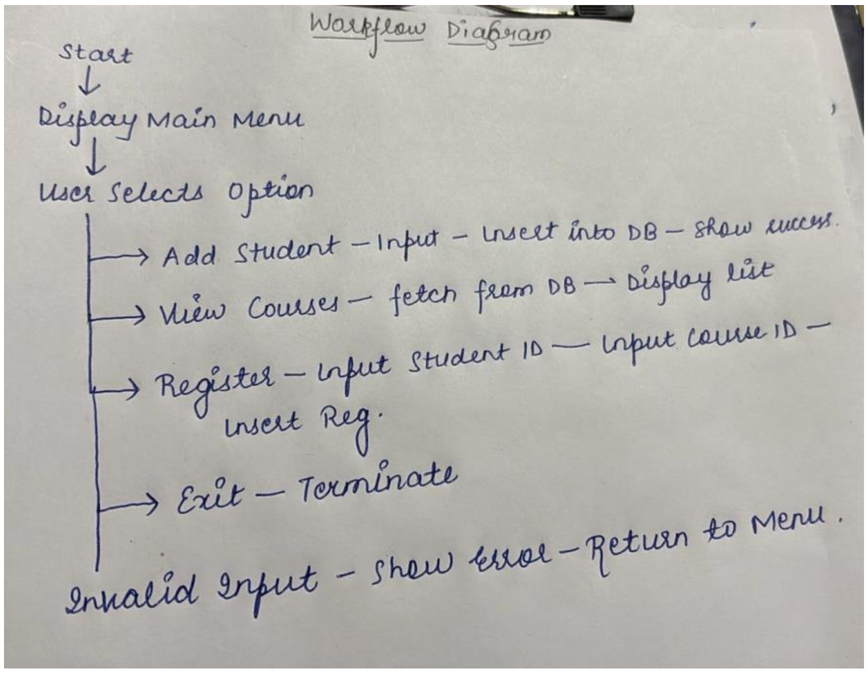
How the Layers Interact

The flow of the system happens in this order:

1. **The user selects an option** from the console menu.
2. The selection goes to the **Application Layer**, where logic decides what to do.
3. If the action requires data (like registering a student), the application sends a request to the **Database Layer**.
4. The database processes the request and returns results (e.g., course list).
5. The Application Layer receives the data and sends it back to the **Console Interface**.
6. The output is displayed to the user.

Design Diagrams


Use Case Diagram
User
Add Student
View Courses
Register Student
Exit


Workflow Diagram
Start
Display Main Menu
User Selects Option
Add Student – Input – Insert into DB – Show success.
View Courses – fetch from DB → Display list
Register – Input Student ID — Input Course ID – Insert Reg.
Exit – Terminate
Invalid Input – Show error – Return to Menu.

Design Decisions & Rationale

During the development of the Student Course Registration System, several design decisions were made to ensure the project remained simple, efficient, and easy to understand for academic purposes.

1. Console-Based Interface

A console (text-based) interface was chosen instead of a GUI because:

- It reduces complexity for beginners.
- It allows focus on core Java concepts.
- It runs on any system without additional setup.

2. Use of MySQL Database

MySQL was selected as the database because:

- It is widely used and stable.
- Free and easy to install.
- Works seamlessly with JDBC.

3. Use of JDBC for Connectivity

JDBC was used to connect Java with MySQL because:

- It is the standard database interface for Java.
- Easy to integrate in academic projects.
- Supports SQL operations directly.

4. Layered Architecture

The system follows a **three-layer architecture**:

- Presentation layer (console menu)
- Application logic layer (Java classes)
- Database layer (MySQL)

This separation ensures:

- Better organization of code
- Easier debugging and modification
- Clear flow of control

5. Object-Oriented Design

Classes like Student, Course, and DBConnection were created to:

- Organize code logically

- Demonstrate inheritance, encapsulation, and modularity
- Make the project more maintainable

Implementation Details

The system was implemented using Java, JDBC, and MySQL.

1. Java Classes Implemented

- **RegistrationSystem.java**
  Handles menu options, user input, and calls functions.
- **Student.java**
  Represents student data using OOP concepts.
- **Course.java**
  Stores course details retrieved from the database.
- **DBConnection.java**
  Connects Java to MySQL using JDBC.
  Stores database URL, username, and password.

2. Database Implementation

Three tables were created:

students(id, name, email)
courses(course_id, course_name)
registrations(reg_id, student_id, course_id)

SQL queries used:

- INSERT → Add student or registration
- SELECT → View courses or students
- JOIN operations for future expansion

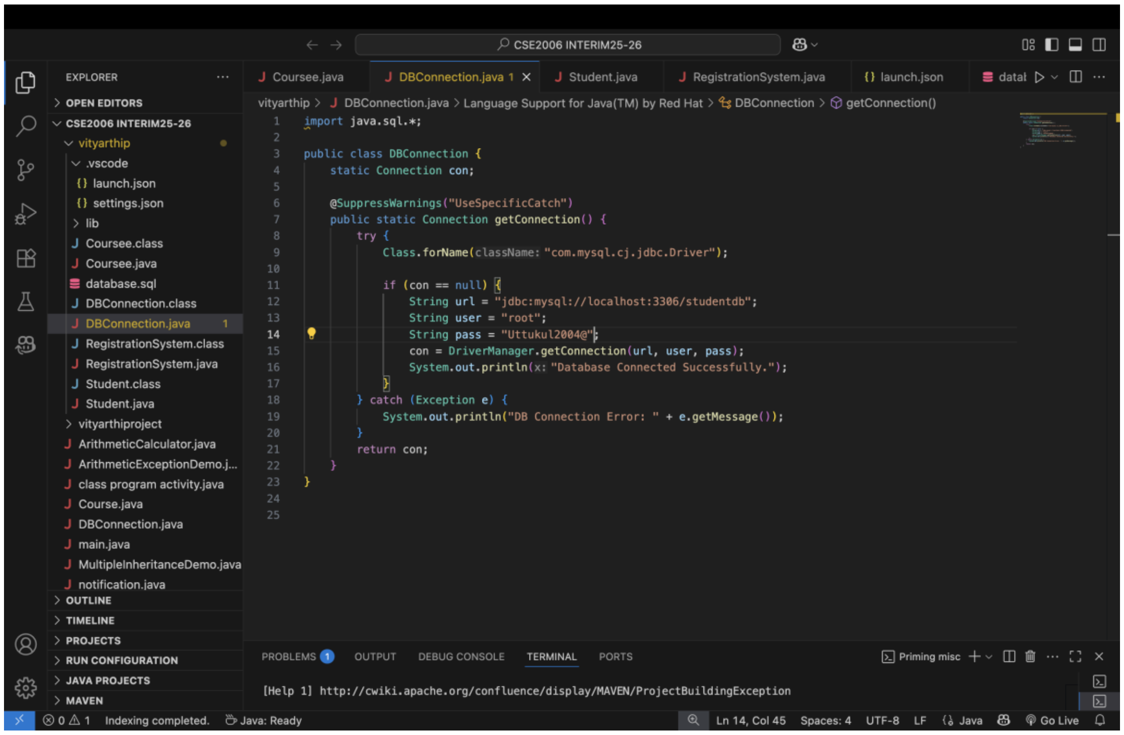
3. Program Flow

1. Program starts and displays menu
2. User selects an option
3. System performs the required operation
4. SQL commands interact with the database
5. Results are shown on screen

4. Compilation & Execution

```
javac -cp ".:lib/mysql-connector-j-9.5.0.jar" *.java
java -cp ".:lib/mysql-connector-j-9.5.0.jar" RegistrationSystem
```

Screenshots / Results



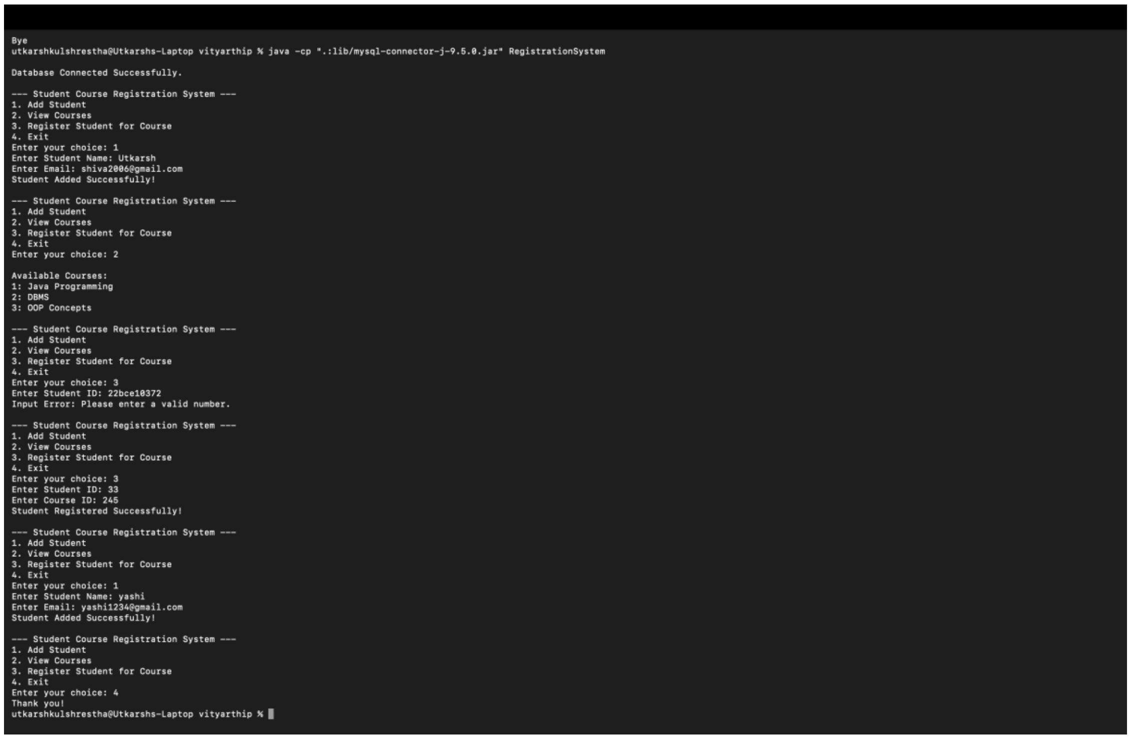
```java
import java.sql.*;

public class DBConnection {
    static Connection con;

    @SuppressWarnings("UseSpecificCatch")
    public static Connection getConnection() {
        try {
            Class.forName(className: "com.mysql.cj.jdbc.Driver");

            if (con == null) {
                String url = "jdbc:mysql://localhost:3306/studentdb";
                String user = "root";
                String pass = "Uttukul2004@";
                con = DriverManager.getConnection(url, user, pass);
                System.out.println(x: "Database Connected Successfully.");
            }
        } catch (Exception e) {
            System.out.println("DB Connection Error: " + e.getMessage());
        }
        return con;
    }
}
```

```
[Help 1] http://cwiki.apache.org/confluence/display/MAVEN/ProjectBuildingException
```



```
Bye
utkarshkulshrestha@Utkarshs-Laptop vityarthip % java -cp ".:lib/mysql-connector-j-9.5.0.jar" RegistrationSystem

Database Connected Successfully.

--- Student Course Registration System ---
1. Add Student
2. View Courses
3. Register Student for Course
4. Exit
Enter your choice: 1
Enter Student Name: Utkarsh
Enter Email: shiva2006@gmail.com
Student Added Successfully!

--- Student Course Registration System ---
1. Add Student
2. View Courses
3. Register Student for Course
4. Exit
Enter your choice: 2

Available Courses:
1: Java Programming
2: DBMS
3: OOP Concepts

--- Student Course Registration System ---
1. Add Student
2. View Courses
3. Register Student for Course
4. Exit
Enter your choice: 3
Enter Student ID: 22bce10372
Input Error: Please enter a valid number.

--- Student Course Registration System ---
1. Add Student
2. View Courses
3. Register Student for Course
4. Exit
Enter your choice: 3
Enter Student ID: 33
Enter Course ID: 245
Student Registered Successfully!

--- Student Course Registration System ---
1. Add Student
2. View Courses
3. Register Student for Course
4. Exit
Enter your choice: 1
Enter Student Name: yashi
Enter Email: yashi1234@gmail.com
Student Added Successfully!

--- Student Course Registration System ---
1. Add Student
2. View Courses
3. Register Student for Course
4. Exit
Enter your choice: 4
Thank you!
utkarshkulshrestha@Utkarshs-Laptop vityarthip % 
```

Testing Approach

The system was tested using different methods to ensure correctness.

1. Functional Testing

Each option in the menu was tested:

- Add student → verified in database
- View courses → checked list correctness
- Register student → validated registration entries

2. Input Validation Testing

- Entering wrong student ID
- Entering non-numeric menu choices

- Leaving fields empty

3. Database Testing

Executed SQL queries manually:

```
SELECT * FROM students;
SELECT * FROM registrations;
```

to validate correct data entry.

4. Integration Testing

Verified the complete flow:
Add student → View courses → Register → Check DB

5. Exception Handling Tests

Tried:

- Wrong database password
- Missing JAR file
- Incorrect SQL

All exceptions were caught without crashing.

Challenges Faced

1. JDBC Driver Errors

Initially, the driver was not detected because the .jar file was placed incorrectly.
Solution: Added correct path to classpath.

2. MySQL Connection Issues

Faced errors like “Access denied” & “Unknown database”.
Solution: Corrected MySQL root password & created the required database manually.

3. Input Handling

Program crashed when users entered non-numeric values.
Solution: Added exception handling using try-catch.

4. Directory Structure Problems

Java source files in different folders caused compilation errors.
Solution: Organized project into a clean folder structure.

5. Classpath on macOS

macOS required a different classpath format (.:lib/...).
Solution: Used correct syntax for UNIX-based systems.

Learnings & Key Takeaways

- Understanding Java OOP concepts (classes, objects, methods).
- Hands-on experience with JDBC and SQL queries.
- Knowledge of how Java interacts with external databases.
- Practice with exception handling and debugging.

- Learned how to structure a real-world console application.
- Understood the importance of proper folder organization.
- Gained experience working with GitHub & project documentation.

Future Enhancements

Possible future upgrades:

1. GUI Version

Add a graphical interface using:

- JavaFX
- Swing
- Web-based UI

2. Login System

Different logins for:

- Admin
- Students

3. Validation Improvements

Check email format, duplicate entries, etc.

4. More Features

- Add/remove courses
- Update student details
- View student’s registered courses

5. Deployment

Convert project into:

- A desktop application
- A web application with Servlets/JSP

6. Reports & Analytics

Create reports like:

- Number of students per course
- Registration trends

References

We can include the following safe references:

1. Oracle Java Documentation: https://docs.oracle.com/javase/

2. MySQL Documentation: https://dev.mysql.com/doc/

3. JDBC Tutorial: https://docs.oracle.com/javase/tutorial/jdbc/

4. StackOverflow discussions for Java & MySQL issues

5. ChatGPT (Used for explanation & learning)