Object Oriented Programming (Lab)

Project Report: Examination or Quiz Management System

By: Qirrat Azam

Id: cs221007

Examination or Quiz Management System:

A summary on what is an examination or quiz management system purpose is, how it is build and works.

An examination or quiz management system is a software application or platform designed to facilitate the creation, administration, and evaluation of examinations or quizzes. It provides an efficient and organized way to manage the entire process, from creating the questions to grading the results.

Here are some key features commonly found in examination or quiz management systems:

Question Bank: The system allows educators or administrators to create and store a database of questions. These questions can be categorized and tagged for easy retrieval during exam or quiz creation.

Exam/Quiz Creation: Educators can create exams or quizzes by selecting questions from the question bank. They can define various parameters such as time limits, question randomization, and difficulty levels.

Scheduling: The system enables administrators to schedule exams or quizzes, specifying the date, time, and duration. It may also support multiple exam sessions for different groups of students.

Online Delivery: Many systems offer online delivery options, allowing students to access and take exams or quizzes remotely through a web-based interface. This eliminates the need for physical paper-based tests.

Security: To maintain the integrity of exams or quizzes, these systems often incorporate security measures such as password protection, access control, and prevention of cheating techniques like copying or external resource usage.

Grading and Evaluation: The system automates the grading process, calculating scores based on predefined answer keys or criteria. It can provide immediate feedback to students, allowing them to review their performance.

Analytics and Reporting: Examination management systems often generate reports and analytics, offering insights into student performance, class averages, and statistical analysis. This information can help educators identify areas of improvement and track overall progress.

Integration and Compatibility: Some systems integrate with learning management systems (LMS) or other educational software, enabling seamless data exchange and streamlining administrative tasks.

Overall, an examination or quiz management system simplifies the entire examination process, reduces manual effort, and provides a streamlined approach for both educators and students. It enhances efficiency, accuracy, and the overall experience of conducting and evaluating exams or quizzes.

OOP Concepts that are added to make an examination or quiz management system more robust and modular.

- **Encapsulation:** Encapsulation involves bundling data and methods into a single unit, known as a class. In the examination management system, encapsulation can be applied to encapsulate related functionality and data together. For example, you can have classes such as "Exam," "Question," or "Student" that encapsulate the relevant attributes and methods associated with those entities.
- **Inheritance:** Inheritance allows classes to inherit properties and behaviors from other classes, creating a hierarchy of classes. In the examination management system, you can create a base class, such as "Assessment," that contains common attributes and methods shared by different types of assessments (e.g., exams and quizzes). Then, specific assessment types can inherit from the base class and add their own unique features.
- **Polymorphism:** Polymorphism enables objects of different classes to be treated as objects of a common base class. In the examination management system, polymorphism can be utilized to handle different types of assessments uniformly. For example, you can have a method called "grade Assessment()" that can accept any type of assessment object (exam or quiz) and perform grading based on the specific implementation of the method in each class.
- **Abstraction:** Abstraction involves representing complex systems by focusing on essential features and hiding unnecessary details. In the examination management system, abstraction can be used to create abstract classes or interfaces that define common behaviors and attributes required for assessments. Concrete classes, such as "MultipleChoiceQuestion" or "EssayQuestion," can then implement these abstract classes or interfaces, providing specific implementations for their functionality.
- **Association and Composition:** Association represents a relationship between two or more classes, while composition represents a "has-a" relationship, where one class consists of or owns another class. In the examination management system, associations can be established between classes like "Student" and "Exam" to represent the relationship between students and the exams they take. Composition can be used to model complex entities, such as an "Exam" class composed of multiple "Question" objects.

By incorporating these OOP concepts, the examination or quiz management system can become more flexible, maintainable, and extensible. It allows for code reusability, promotes modular design, and enables future enhancements and modifications to the system with ease.

Here are some specific examples of how static data members are used in this Examination Management System:

QuizQuestions: QuizQuestion class a static member, can centralize the storage of quiz questions within the system. Instead of creating multiple instances of the QuizQuestion class for each quiz or exam, we can maintain a single instance accessible to all assessments.

Instructor_Menu: Instructors can use the Instructor menu to create new exams or quizzes. They can specify various parameters such as the title, duration, number of questions, and difficulty level. This functionality allows instructors to customize assessments according to their teaching goals and curriculum.

Students_Menu: The Students menu allows students to access their assigned exams or quizzes. It provides a list of available assessments, including the title, due date, and duration. Students can select and start the assessments from this menu, ensuring they have convenient access to their assigned tasks.

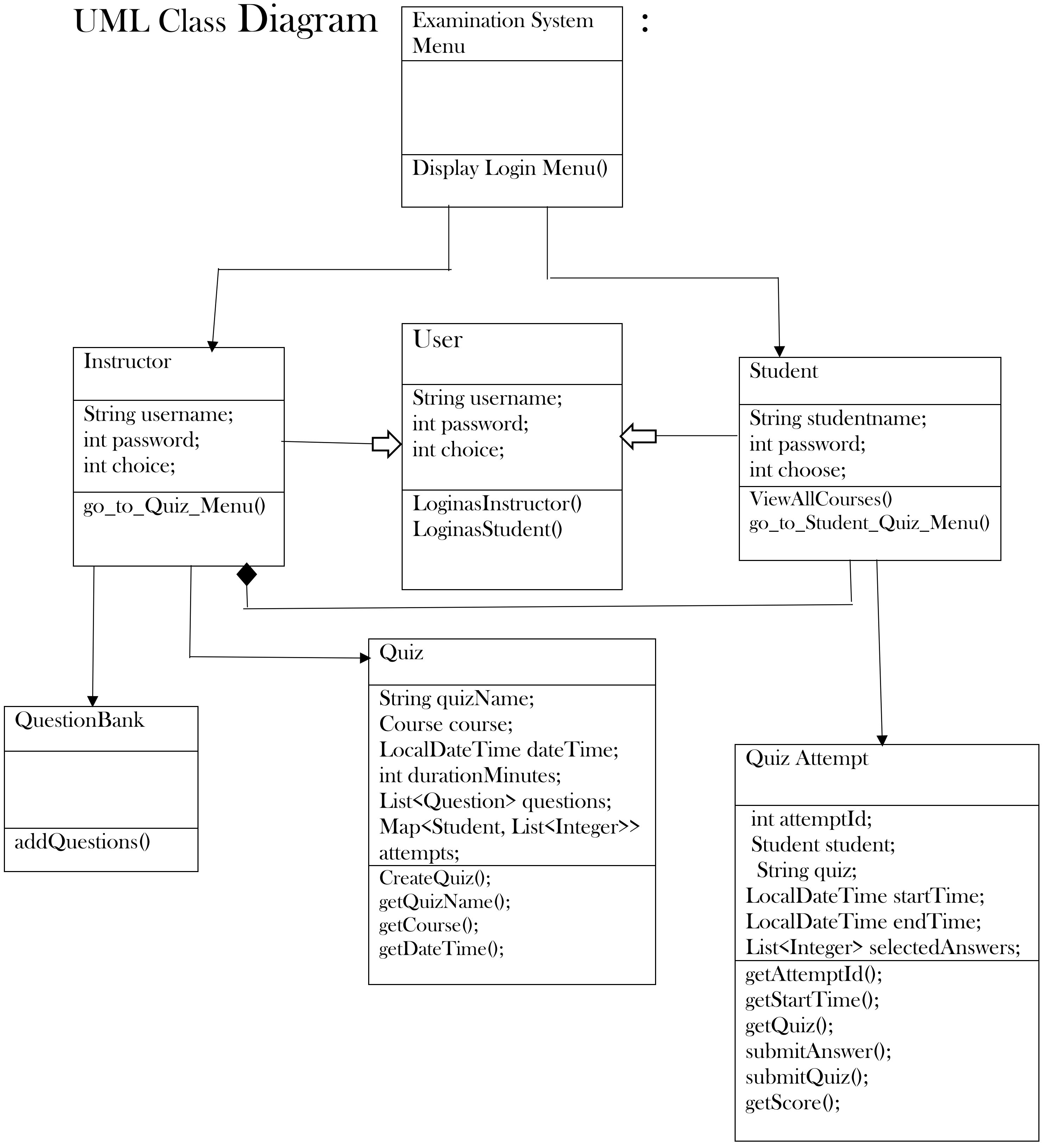
Filing: Filing enables the system to retrieve and access stored data when required. For example, when a student takes an exam, the system retrieves the relevant quiz questions from the file and presents them to the student. Similarly, when an instructor wants to view assessment results, the system retrieves the corresponding data from files and presents it to the instructor.

Create_Quiz: The createQuiz function would provide options for adding questions to the quiz. It could allow the instructor to select questions from a question bank, specify new questions directly, or import questions from external sources. The function would handle the logic of adding the selected questions to the quiz.

These are just a few examples of how static data members can be used in an examination or quiz management system.

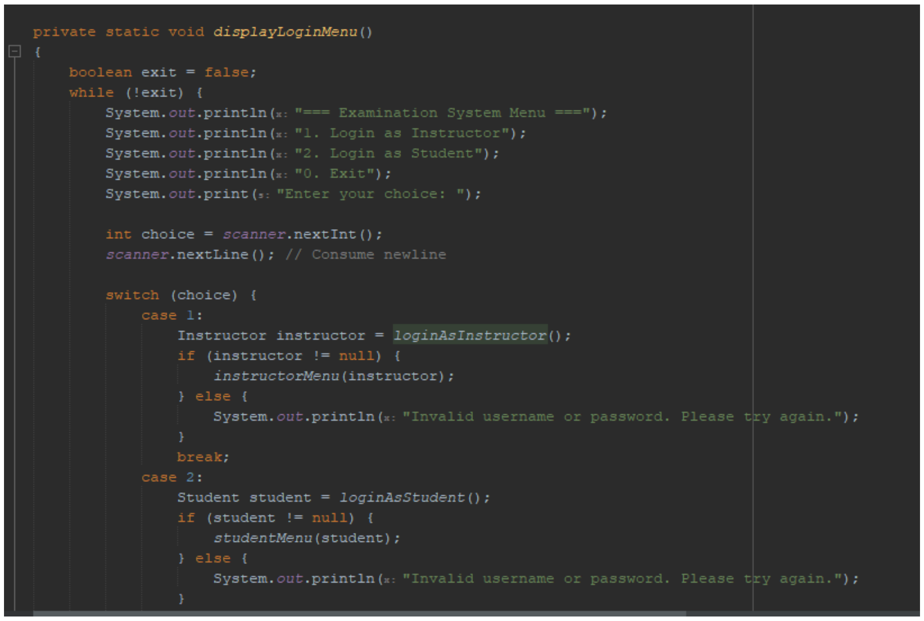
By utilizing static data members, we centralize certain functionalities, and ensure consistency across different components of the system.

UML Class Diagram :

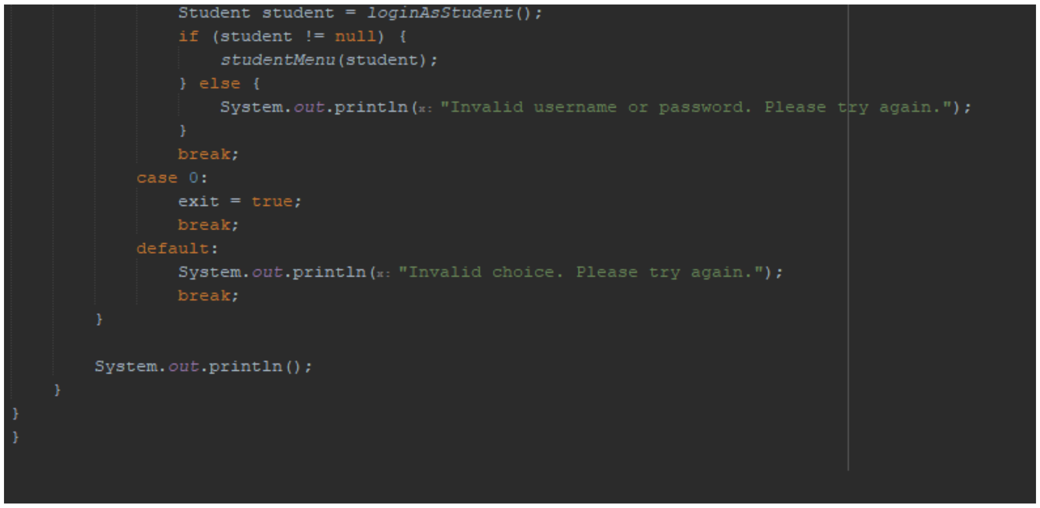


Code:

This is the main code, where when you execute this program it will first display displayLoginMenu(), where the user will be asked whether to login as a student or instructor/teacher.

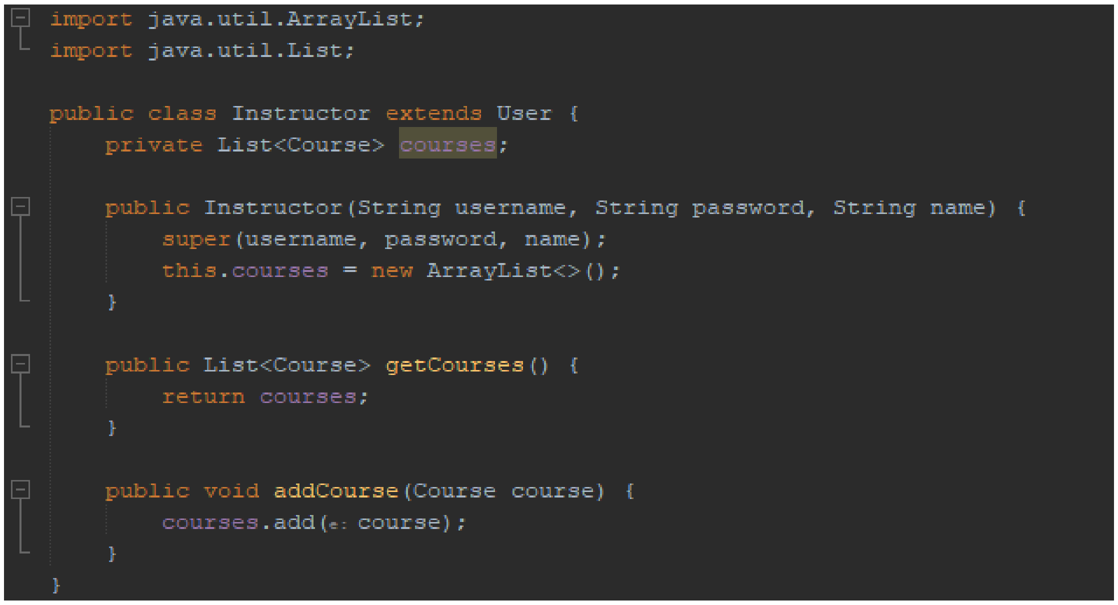


```java
private static void displayLoginMenu()
{
    boolean exit = false;
    while (!exit) {
        System.out.println(x: "=== Examination System Menu ===");
        System.out.println(x: "1. Login as Instructor");
        System.out.println(x: "2. Login as Student");
        System.out.println(x: "0. Exit");
        System.out.print(s: "Enter your choice: ");

        int choice = scanner.nextInt();
        scanner.nextLine(); // Consume newline

        switch (choice) {
            case 1:
                Instructor instructor = loginAsInstructor();
                if (instructor != null) {
                    instructorMenu(instructor);
                } else {
                    System.out.println(x: "Invalid username or password. Please try again.");
                }
                break;
            case 2:
                Student student = loginAsStudent();
                if (student != null) {
                    studentMenu(student);
                } else {
                    System.out.println(x: "Invalid username or password. Please try again.");
                }
```



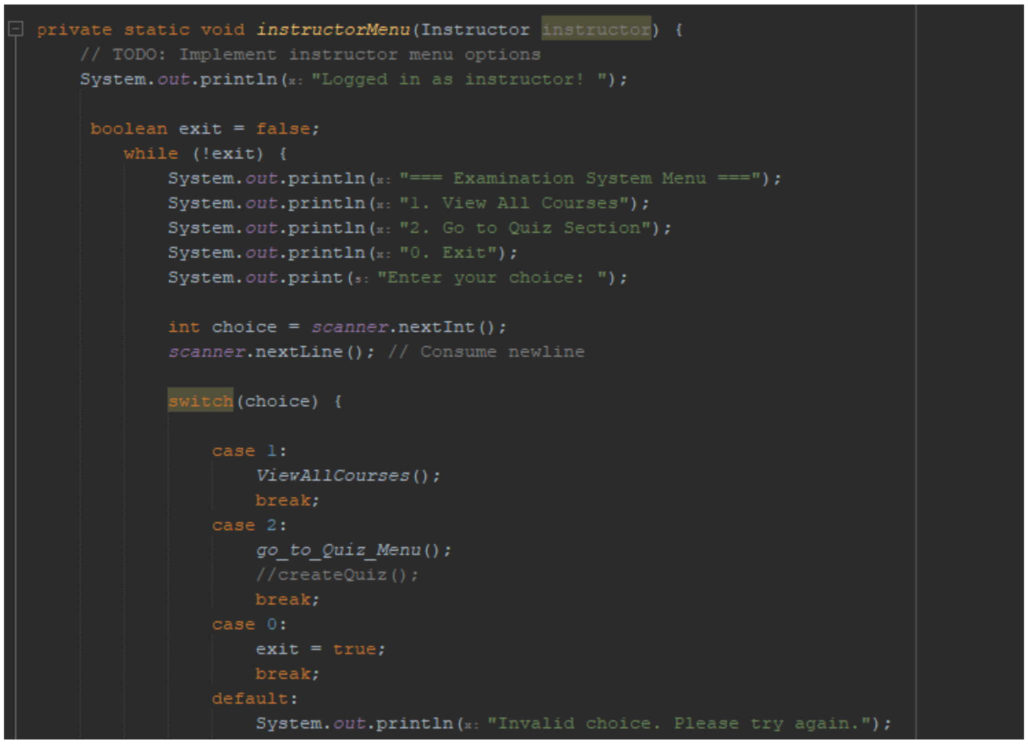
```java
                Student student = loginAsStudent();
                if (student != null) {
                    studentMenu(student);
                } else {
                    System.out.println(x: "Invalid username or password. Please try again.");
                }
                break;
            case 0:
                exit = true;
                break;
            default:
                System.out.println(x: "Invalid choice. Please try again.");
                break;
        }

        System.out.println();
    }
}
}
```

Here the instructor will be asked to enter name and password to get into system.



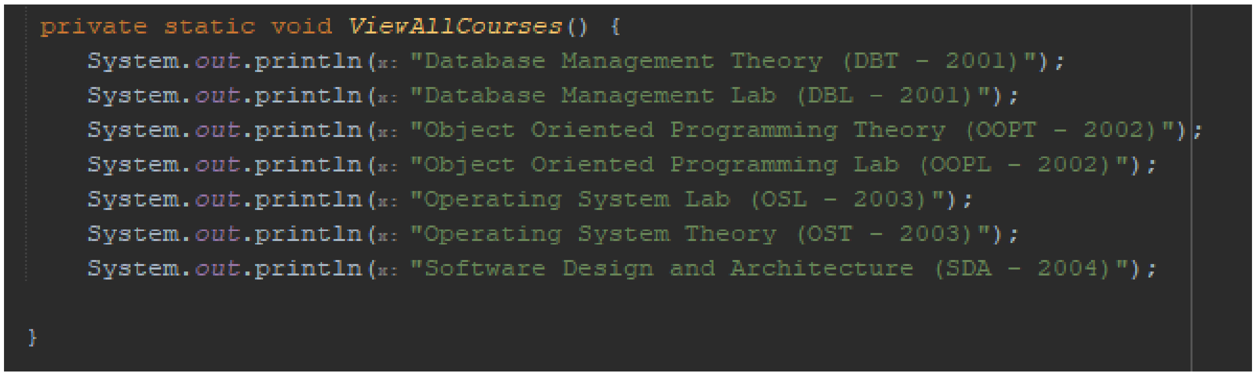
```java
import java.util.ArrayList;
import java.util.List;

public class Instructor extends User {
    private List<Course> courses;

    public Instructor(String username, String password, String name) {
        super(username, password, name);
        this.courses = new ArrayList<>();
    }

    public List<Course> getCourses() {
        return courses;
    }

    public void addCourse(Course course) {
        courses.add(e: course);
    }
}
```

After loging as an instructor, an instructor menu will be displayed, where you will have choices to select to view all courses and Go to Quiz Menu



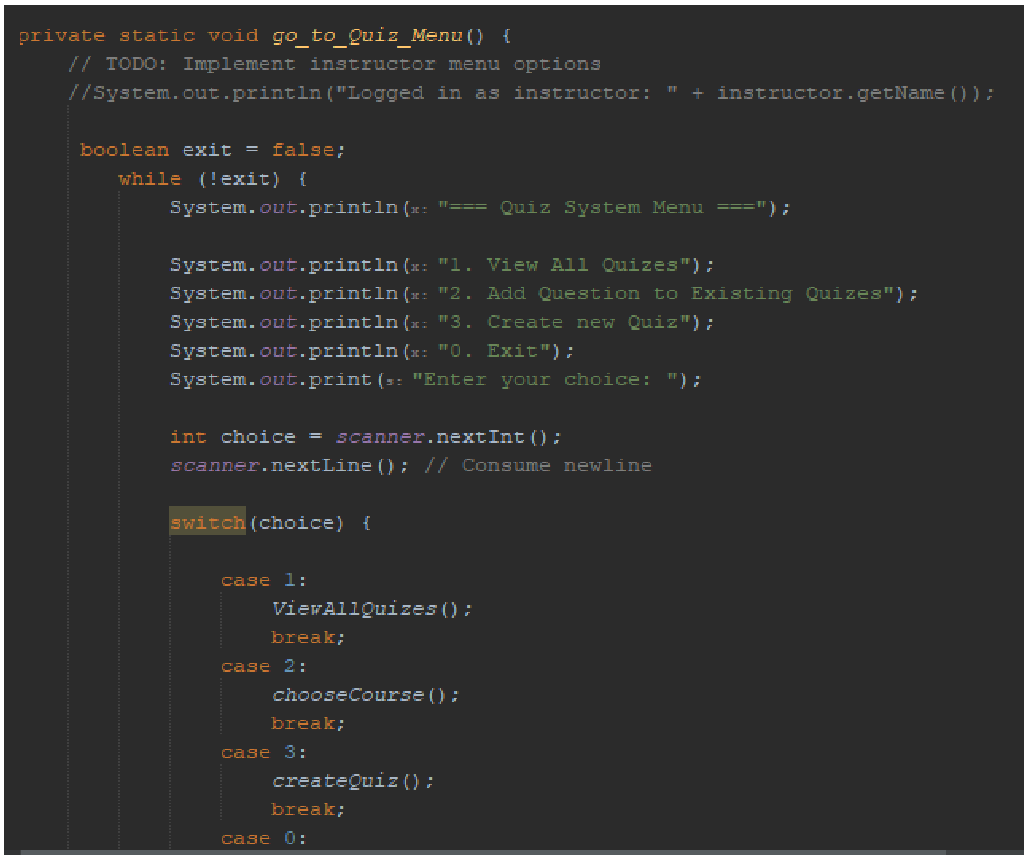
```java
private static void instructorMenu(Instructor instructor) {
    // TODO: Implement instructor menu options
    System.out.println(x: "Logged in as instructor! ");

     boolean exit = false;
        while (!exit) {
            System.out.println(x: "=== Examination System Menu ===");
            System.out.println(x: "1. View All Courses");
            System.out.println(x: "2. Go to Quiz Section");
            System.out.println(x: "0. Exit");
            System.out.print(s: "Enter your choice: ");

            int choice = scanner.nextInt();
            scanner.nextLine(); // Consume newline

            switch(choice) {

                case 1:
                    ViewAllCourses();
                    break;
                case 2:
                    go_to_Quiz_Menu();
                    //createQuiz();
                    break;
                case 0:
                    exit = true;
                    break;
                default:
                    System.out.println(x: "Invalid choice. Please try again.");
```

A view of all courses:



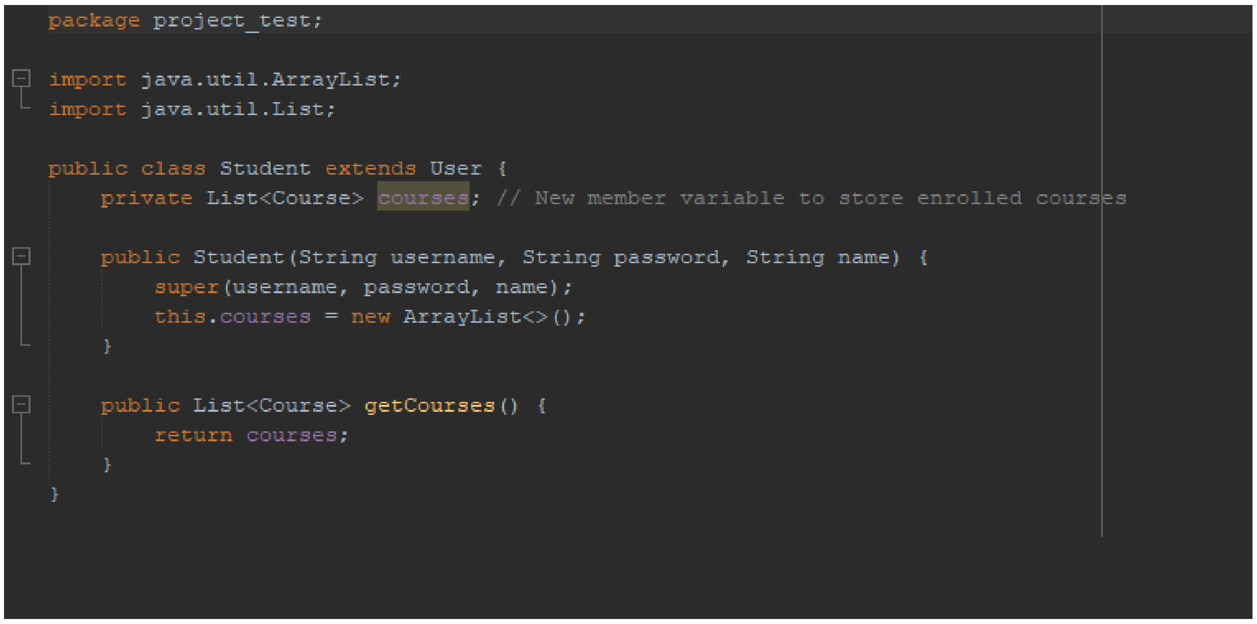
```
private static void ViewAllCourses() {
    System.out.println(x: "Database Management Theory (DBT - 2001)");
    System.out.println(x: "Database Management Lab (DBL - 2001)");
    System.out.println(x: "Object Oriented Programming Theory (OOPT - 2002)");
    System.out.println(x: "Object Oriented Programming Lab (OOPL - 2002)");
    System.out.println(x: "Operating System Lab (OSL - 2003)");
    System.out.println(x: "Operating System Theory (OST - 2003)");
    System.out.println(x: "Software Design and Architecture (SDA - 2004)");

}
```

Next Go to Quiz Menu , here you can create Quiz, choose Course, and view all Quizes.



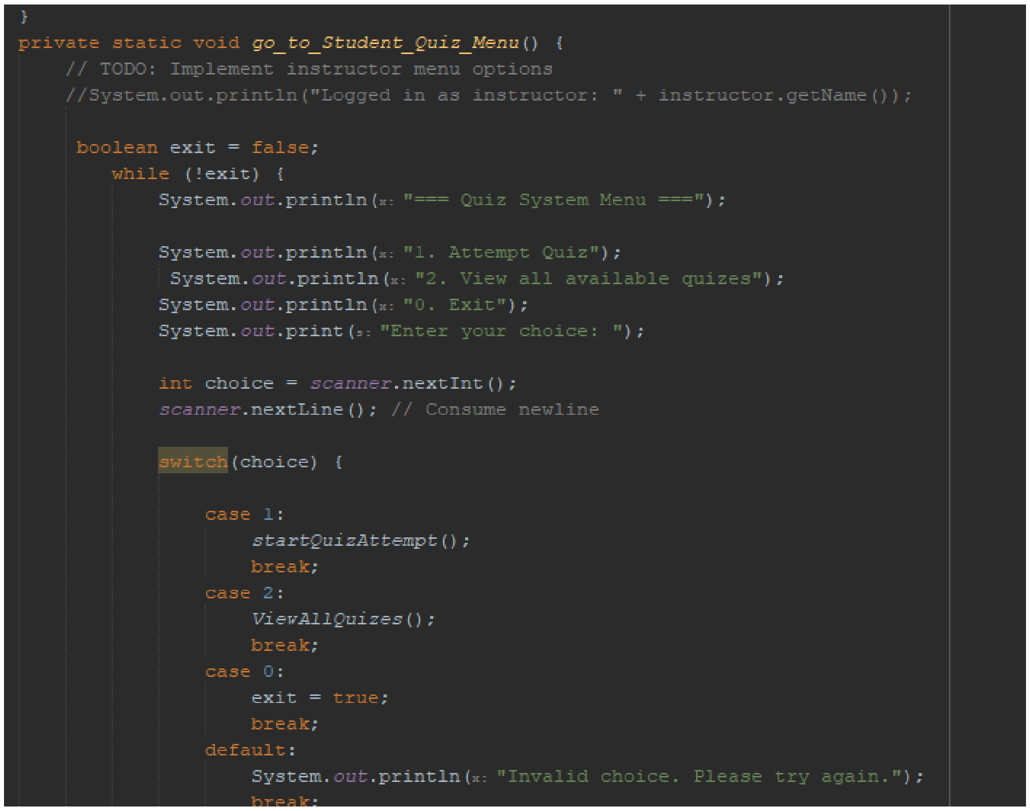
```
private static void go_to_Quiz_Menu() {
    // TODO: Implement instructor menu options
    //System.out.println("Logged in as instructor: " + instructor.getName());

     boolean exit = false;
        while (!exit) {
            System.out.println(x: "=== Quiz System Menu ===");

            System.out.println(x: "1. View All Quizes");
            System.out.println(x: "2. Add Question to Existing Quizes");
            System.out.println(x: "3. Create new Quiz");
            System.out.println(x: "0. Exit");
            System.out.print(s: "Enter your choice: ");

            int choice = scanner.nextInt();
            scanner.nextLine(); // Consume newline

            switch(choice) {

                case 1:
                    ViewAllQuizes();
                    break;
                case 2:
                    chooseCourse();
                    break;
                case 3:
                    createQuiz();
                    break;
                case 0:
```

In this part of code, student can login.



```
package project_test;

import java.util.ArrayList;
import java.util.List;

public class Student extends User {
    private List<Course> courses; // New member variable to store enrolled courses

    public Student(String username, String password, String name) {
        super(username, password, name);
        this.courses = new ArrayList<>();
    }

    public List<Course> getCourses() {
        return courses;
    }
}
```

Now this part of program will display a Student Menu, in which he can see number of quizzes made by teacher, and an attempt option and by attempting the student will also get the obtain result.



```
}
private static void go_to_Student_Quiz_Menu() {
    // TODO: Implement instructor menu options
    //System.out.println("Logged in as instructor: " + instructor.getName());

     boolean exit = false;
        while (!exit) {
            System.out.println(x: "=== Quiz System Menu ===");

            System.out.println(x: "1. Attempt Quiz");
             System.out.println(x: "2. View all available quizes");
            System.out.println(x: "0. Exit");
            System.out.print(s: "Enter your choice: ");

            int choice = scanner.nextInt();
            scanner.nextLine(); // Consume newline

            switch(choice) {

                case 1:
                    startQuizAttempt();
                    break;
                case 2:
                    ViewAllQuizes();
                    break;
                case 0:
                    exit = true;
                    break;
                default:
                    System.out.println(x: "Invalid choice. Please try again.");
                    break;
```