Overview

This project's main objective is to show the majority of what I have learned in this course this semester, with the most important concept being object oriented programming. The main idea of this project was to create a very simplified version of a course management system like Canvas.

Project Summary

Project Title:
CourseM

Description:
This project's main objective is to show the majority of what I have learned in this course this semester, with the most important concept being object oriented programming. This project is a course management system that is like a very simplified Canvas. It has a user login with two different users, an instructor and a student. The instructor can have multiple courses where they can post quizzes and discussions. The student can join multiple courses using the courses' names, and take the posted quizzes, reply to the posted discussions, and rate their courses. This project is a command-line program and uses Java serialization to save and load data from files.

Intended Users:
The intended users of this program are course instructors and students taking their courses.

Use of Object-Oriented Programming Concepts:
This project is heavily reliant on object-oriented design, having 11 different classes. Here is how object-oriented programming concepts are utilized in this project:

- **Abstraction:** There are abstract classes of Account and Question

- **Encapsulation:** The data in all of the instantiated classes are accessed through getter and setter methods
- **Inheritance:** The Instructor and Student classes inherit from the Account class, and the different types of questions inherit from the Question class
- **Polymorphism:** Methods like printMenu() are implemented differently in classes that inherit from an abstract class
- **Composition:** An Instructor “owns” instances of Course classes
- **Aggregation:** A Student “uses” instances of Course classes

Use Case Analysis

This project has two use cases. Every use case will start off with a login or sign up screen. Then depending on what type of account you are, you will have the two different use cases.
There is an instructor who is able to:

1. View their courses
2. Create and delete courses
3. Create, delete, and view quizzes
4. Create delete, and view discussions
5. View their course ratings

There is a student who is able to:

1. View their courses
2. Join a course via course name
3. Take a quiz from a course
4. Reply to a discussion
5. Rate their courses

Data Design

The main classes that hold data in this project are the Account and Course classes. Most of the data that is being used in this program needs to be persistent. To achieve this, I utilized object serialization in Java and saved the objects that contain the data that needs to be persistent. There are two data files that load the Course and Accounts objects and their data. I used the ArrayList data structure to store classes like Account, Course, Quiz, Question, and Discussion. I also used the HashMap data structure to map the names of Courses and Accounts with their respective objects.

UI Design

This project is a command-line program. There are multi level menus.

The Instructor has an initial menu with these options:

1. View courses
2. Create a new course
3. Enter a course
4. Delete a course
5. Exit

When an Instructor enters a course, they have these options for their course:

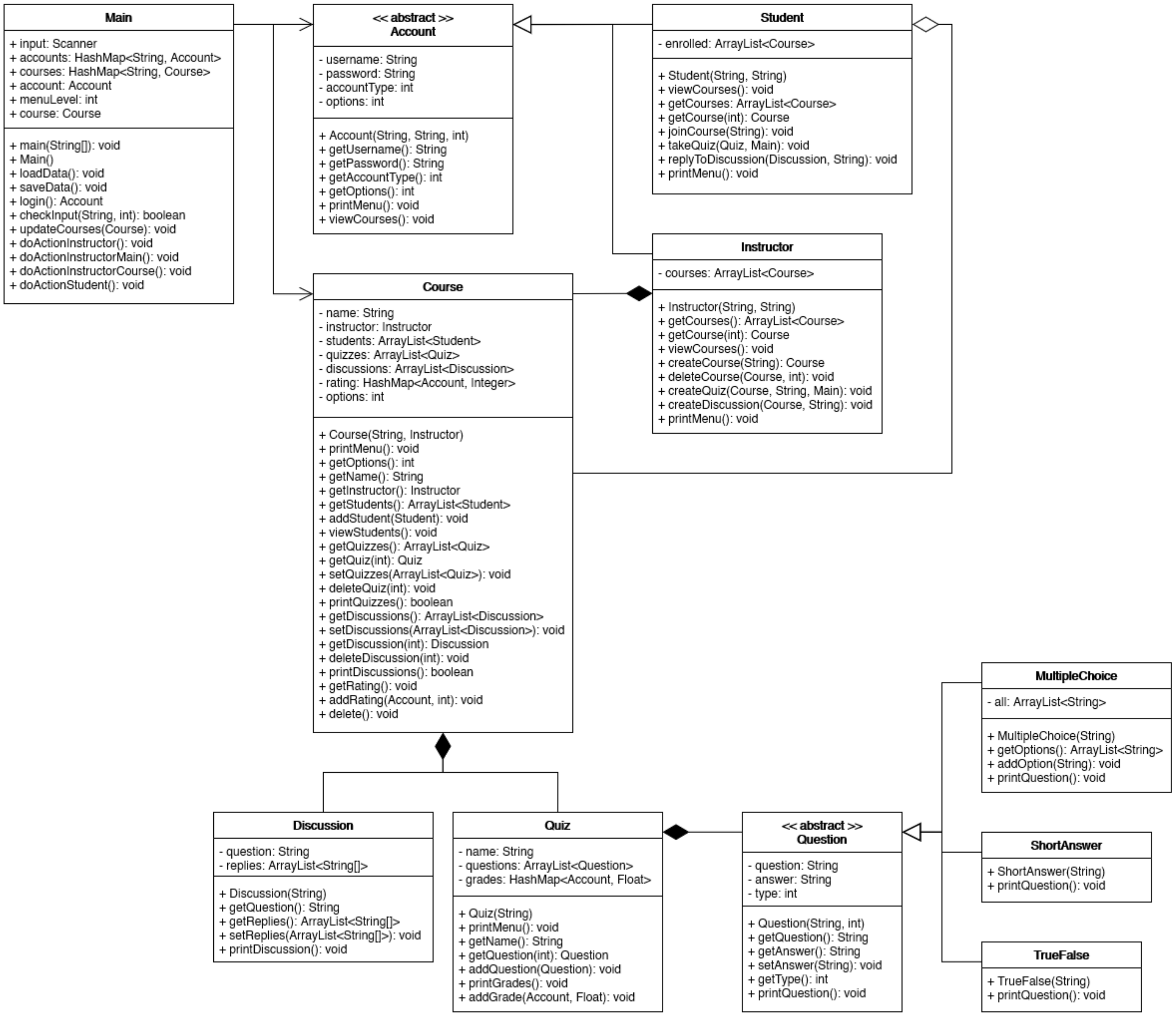
1. Create a discussion
2. View Discussions
3. Delete a discussion
4. Create a quiz
5. View quiz grades
6. Delete a quiz
7. View course rating
8. View students
9. Back

The Student has an initial menu with these options:

1. View courses
2. Join a course

3. Reply to a discussion
4. Take a quiz
5. Rate a course
6. Exit

UML


Main
+ input: Scanner
+ accounts: HashMap<String, Account>
+ courses: HashMap<String, Course>
+ account: Account
+ menuLevel: int
+ course: Course
+ main(String[]): void
+ Main()
+ loadData(): void
+ saveData(): void
+ login(): Account
+ checkInput(String, int): boolean
+ updateCourses(Course): void
+ doActionInstructor(): void
+ doActionInstructorMain(): void
+ doActionInstructorCourse(): void
+ doActionStudent(): void
<< abstract >>
Account
- username: String
- password: String
- accountType: int
- options: int
+ Account(String, String, int)
+ getUsername(): String
+ getPassword(): String
+ getAccountType(): int
+ getOptions(): int
+ printMenu(): void
+ viewCourses(): void
Student
- enrolled: ArrayList<Course>
+ Student(String, String)
+ viewCourses(): void
+ getCourses: ArrayList<Course>
+ getCourse(int): Course
+ joinCourse(String): void
+ takeQuiz(Quiz, Main): void
+ replyToDiscussion(Discussion, String): void
+ printMenu(): void
Instructor
- courses: ArrayList<Course>
+ Instructor(String, String)
+ getCourses(): ArrayList<Course>
+ getCourse(int): Course
+ viewCourses(): void
+ createCourse(String): Course
+ deleteCourse(Course, int): void
+ createQuiz(Course, String, Main): void
+ createDiscussion(Course, String): void
+ printMenu(): void
Course
- name: String
- instructor: Instructor
- students: ArrayList<Student>
- quizzes: ArrayList<Quiz>
- discussions: ArrayList<Discussion>
- rating: HashMap<Account, Integer>
- options: int
+ Course(String, Instructor)
+ printMenu(): void
+ getOptions(): int
+ getName(): String
+ getInstructor(): Instructor
+ getStudents(): ArrayList<Student>
+ addStudent(Student): void
+ viewStudents(): void
+ getQuizzes(): ArrayList<Quiz>
+ getQuiz(int): Quiz
+ setQuizzes(ArrayList<Quiz>): void
+ deleteQuiz(int): void
+ printQuizzes(): boolean
+ getDiscussions(): ArrayList<Discussion>
+ setDiscussions(ArrayList<Discussion>): void
+ getDiscussion(int): Discussion
+ deleteDiscussion(int): void
+ printDiscussions(): boolean
+ getRating(): void
+ addRating(Account, int): void
+ delete(): void
Discussion
- question: String
- replies: ArrayList<String[]>
+ Discussion(String)
+ getQuestion(): String
+ getReplies(): ArrayList<String[]>
+ setReplies(ArrayList<String[]>): void
+ printDiscussion(): void
Quiz
- name: String
- questions: ArrayList<Question>
- grades: HashMap<Account, Float>
+ Quiz(String)
+ printMenu(): void
+ getName(): String
+ getQuestion(int): Question
+ addQuestion(Question): void
+ printGrades(): void
+ addGrade(Account, Float): void
<< abstract >>
Question
- question: String
- answer: String
- type: int
+ Question(String, int)
+ getQuestion(): String
+ getAnswer(): String
+ setAnswer(String): void
+ getType(): int
+ printQuestion(): void
MultipleChoice
- all: ArrayList<String>
+ MultipleChoice(String)
+ getOptions(): ArrayList<String>
+ addOption(String): void
+ printQuestion(): void
ShortAnswer
+ ShortAnswer(String)
+ printQuestion(): void
TrueFalse
+ TrueFalse(String)
+ printQuestion(): void