Design Document

University Grading System

Team Members:

Ruizhi Jiang - U17637349

Tiancheng Zhu - U24177199

Athanasios Filippidis - U95061883

Syahrial Dahler - U42782601

Overview

This is the design document of our implementation of a university grading system implemented using Java, Java Swing and SQL. Through this document, the reader will be able to learn more about the incentives that motivated us in the creation of this application, the main functionality decisions that we made and the software requirements of our virtual client, in which we had to adhere. The following sections are: Scope, where we describe the problem that this application is designed to solve, Functionality, where we describe the expected outcome of our application, Goals, where we discuss the goals that we set during the design phase of the application and on what level we achieved those, Object Diagram, where we present an abstract UML representation of our system and Object Justification, where list our core classes and we reason about them.

Scope

Our virtual client is a university professor. Until now, this professor was using Microsoft Office Excel in order to keep track of the grades of the students that are enrolled in the courses she is teaching. However this is not a very efficient way and even though Excel provides a very flexible interface for that use it is not optimal. Our application aims to provide that optimal grading system our client is looking for. Of course, in order to keep the client satisfied, one of our core motives was to only add functionality and better organized structures while deducting as little as possible flexibility (which was the main feature that our client liked in the previous grading system).

Functionality

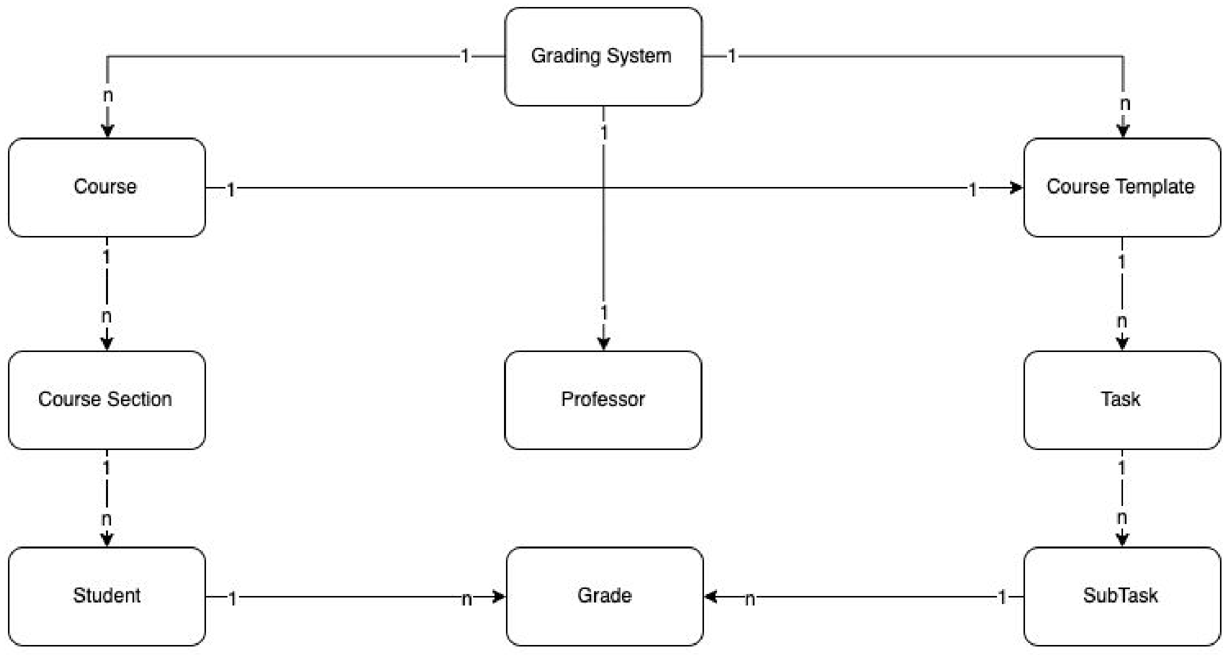
Our first priority is always to have a satisfied client. In order to achieve that in this case we had to make a list of the basic advantages of the currently used system and to ensure that we will be able to provide those in the new system as well. Moreover, we had to do the respective action for the disadvantages. The expected outcome of our work was a system that would combine those two lists of things that should and should not be included in the final product. After ensuring that those existed we brainstormed in order to add any more features that we all agreed that would be useful in such a product.

Goals

Some of the main goals that were set during the analysis process and presented in our initial presentation are the following. Firstly, our client must be able to create a course structure consisting of tasks (homework, exams, assignments etc.) and those tasks will consist of subtasks (homework 1, homework 2, midterm, final etc.). We decided that the client must have absolute freedom in adding or removing as many as possible instances of those entities. Secondly, we decided that our client is highly possible to want to re-use those structures for courses that she will teach in the future and will be strongly based on previously taught courses. For that reason, we decided that there should exist some way of creating blueprints of courses that will be reusable and we prioritized that as well. One more main feature that we set as a prioritized goal was the grades. A professor has to be able to set the grades of a student in many different ways (by setting the points deducted, by setting the absolute score achieved, by setting the percentage score etc.) and has to be able to see the grade of a student in many different representations. Our whole design is structured around those main values which all live under the goal of providing maximum flexibility and usability.

Object Diagram

At this point we had to decide between presenting an in-detail diagram that could possibly create an overflow of information to the reader or a more abstract diagram that shows the basic components-entities of our system. We decided the latter. One main component that is missing from it is the Import Excel class and we decided to not include it because while it is a core component it is more a function than an entity.



Object Justification

Our implementation is strictly following the MVC (Model-View-Controller) architecture. The views and the controllers have as a target to make the front-end functional. The core of our implementation is our back-end so this is where we will focus in this part. The main components/classes of our backend are the following:

- **ProfessorsTool**: While we know that at this point of time were assigned to create only one professor tool we believe that in the future this might change and we may need to add more tools like a messaging application between the students and the professor or a platform that the students will use to register for courses. Having this in mind, this class is an abstract class which will, in the future, encapsulate all the common properties those tools may share.
- **GradingSystem**: This is the actual class that encapsulates all the functionality and properties of the system we implemented. The main properties of this class in our opinion are the actual courses that are taught from the logged in professor and the templates of previous courses that this professor has taught before.
- **Course**: Each course has some identifying values (as name, year, semester), a list of course sections that perform as groups of students in the case that a course has multiple sections (like morning, evening sections), a list of tasks (like Homeworks, Exams etc.) and a course template which serves as the skeleton of this course.
- **CourseTemplate**: Someone could think as the course template being the blueprint of a course. This class is the idea that we came up with in order to be able to preserve the structure of a course without having to keep its actual data. So for example, if course A is at some moment deleted, the structure that a professor has created for this course will survive and will be reusable in the future. What this means is that if course A had only one task which was the Exams task and this task had only one subtask which was the Final Exam and the professor wants to create a new course B that will have again only one final exam she can just use the template of course A. After that, there will be no need for her to enter any other more specific details, the whole structure of the new course will be ready for use (the reader can imagine how quickly the usability of this feature scales up when we add a more complex grading structure).
- **Task**: Task is representing a group of actual tasks. One may think of it as a generic category. For example, a professor may decide that the grading of a course will be based on the Assignments and the Exams. So those two will be the two tasks of this course. As expected, each one of those has a specific weight in the calculation of the final grade.

- **SubTask**: A subtask, as expected, is a sub category of the above class. So for example, a professor may want to have two different exams during a semester. In that case, she would add a Midterm and a Final subtask under the Task Exam category. Both of those have all the necessary information a professor may need while grading, their weight for example. Moreover, every subtask has a list of grades, one for each student course.
- **Grade:** A grade is a structure that connects every student to the specific score that they achieved in one subtask. This score is saved as an absolute points scored score, so for example in a subtask that has as a maximum available score of 130 a possible score of student is 126 and this is the value that will be saved. Then, whenever we need to show that grade, depending on the context, we translate that to either a percentage or a letter grade.

There are a couple more classes that could be listed here and are used in our implementation but we believe that it is better for the reader to focus on the core functionality of our project and not get lost in the details. If however someone is interested in more details we made sure that the class and variable naming is very descriptive and we also have added comments in our code to ensure that it easy to understand for everyone.