CS 591 Final Project – Grading system

• Overview

This is the design document for the final project of our Object-Oriented Java CS591 course. It details the design, class structure, and the implementation details of the Grading System Final Project.

• Scope

The scope for the project is to apply the concepts learn through the entire semester to develop a grading system for the client, that is, our Professor. The objective is the provide a feasible and usable alternative to the current system, Microsoft Excel being used by the Professor and provide additional functionalities that are important to the client in a Java Application. While the client might desire a diverse functionality with a number of features, part of the scope was also to identify the requirements that could be accomplished within a given timeframe of 20 days. The project included an initialization and proposal phase, where the client provided valuable and necessary feedback and suggestions on how the system should be developed. The client was also informed about the proposed deliverables.

The current system being used is an MS Excel spreadsheet, with courses on different sheets and students being represented by different rows while each column represents a specific task or assignment. While this system has many benefits- most important being the flexibility to add almost anything in a cell, add and remove cells when are where needed, and having a fixed structure to it, there are also several downsides.

These include its difficulty in incorporating comments and its poor scalability. Whenever a new student or grade needs to be added, the cell ranges for its formula statements must also be changed. In other words, edits to the entire system are required for any changes. It is difficult to use a common template and involves tedious work in Excel programming. It also doesn't have Our proposed system is an alternative to this gradebook structure, as we have improved upon Excel without taking away the salient features in it that the Professor already liked. We have also incorporated several specialized features that our client – the Professor wanted in an ideal Grading System.

• Functionality

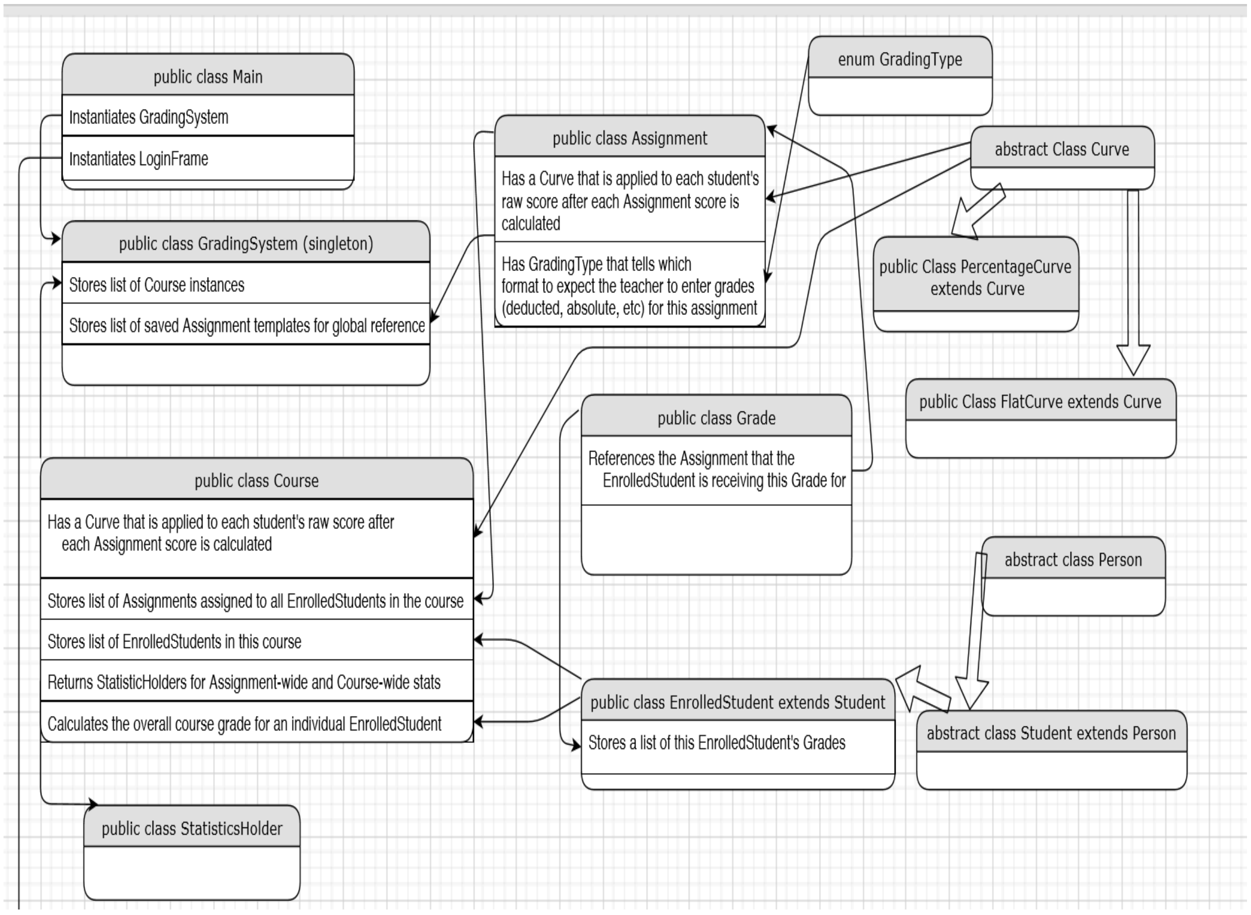
We expect our program to be able to preserve the flexibility and accessibility of excel while being intuitive and easy to use. The interface should be simple and self-explanatory enough that out Professor doesn't need to have to spend time learning how to use it. We want to enable the user to manage the course and its related information like assignments, enrolled students, curves and grades. We allow user to edit the assignment components of the course, and grade these assignments accordingly. We also want to add features like comments and easy calculation of statistics for grades which are specific to our application of grading system. Also the system will be able to do some auto-calculation to show the user some result of score and basic statistic of the course.

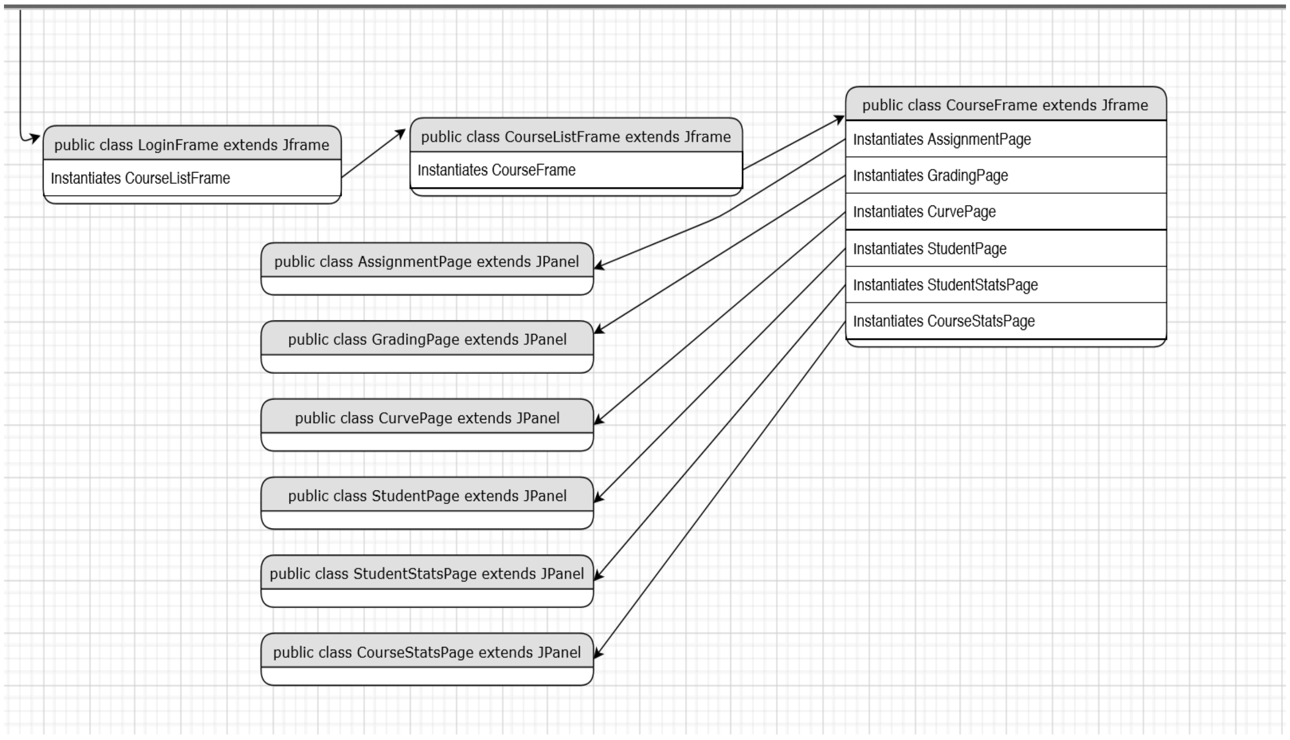
- **Goals**

We finished almost every requirements of our client like course editing, student editing, assignment editing, grade editing. We enable the use or curves, bonus point and the different section for a course, as well as score ranking, score auto calculation.

There is only one thing we didn't achieve in the limited time we are given, which is we didn't implement the A, A-, B+, etc grading according to the final score. But we will definitely put it into the future extension.

- **Object Diagram**


public class Main
Instantiates GradingSystem
Instantiates LoginFrame
enum GradingType
public class Assignment
Has a Curve that is applied to each student's raw score after each Assignment score is calculated
Has GradingType that tells which format to expect the teacher to enter grades (deducted, absolute, etc) for this assignment
abstract Class Curve
public class GradingSystem (singleton)
Stores list of Course instances
Stores list of saved Assignment templates for global reference
public Class PercentageCurve extends Curve
public Class FlatCurve extends Curve
public class Grade
References the Assignment that the EnrolledStudent is receiving this Grade for
public class Course
Has a Curve that is applied to each student's raw score after each Assignment score is calculated
Stores list of Assignments assigned to all EnrolledStudents in the course
Stores list of EnrolledStudents in this course
Returns StatisticHolders for Assignment-wide and Course-wide stats
Calculates the overall course grade for an individual EnrolledStudent
abstract class Person
public class EnrolledStudent extends Student
Stores a list of this EnrolledStudent's Grades
abstract class Student extends Person
public class StatisticsHolder



public class LoginFrame extends Jframe
Instantiates CourseListFrame
public class CourseListFrame extends Jframe
Instantiates CourseFrame
public class CourseFrame extends Jframe
Instantiates AssignmentPage
Instantiates GradingPage
Instantiates CurvePage
Instantiates StudentPage
Instantiates StudentStatsPage
Instantiates CourseStatsPage
public class AssignmentPage extends JPanel
public class GradingPage extends JPanel
public class CurvePage extends JPanel
public class StudentPage extends JPanel
public class StudentStatsPage extends JPanel
public class CourseStatsPage extends JPanel

- ## Object Justification

  - ### Core Class

Main: The entrance for the Grading System. Main will create a singleton instance of class *GradingSystem* and create the very first UI for *GradingSystem* which is the login page.

GradingSystem: We use the singleton model for *GradingSystem* because there will only be a *GradingSystem* instance at one time. Also, it’s easy to access the singleton by using the static method *GradingSystem.getInstance().*

GradingSystem is responsible for the Account operation, the *LoginCheck()* and *signUp()* methods have to be within the *GradingSystem* class, it ensures the security and the data Confidentiality of the system.

GradingSystem class maintain the important data like all the information of courses and store all of them in the memory. When the instance is first created, it will interact with database and load the data into memory. In this way, if the database disconnects during the runtime of GradingSystem, the system won’t be affected, the data won’t be discarded.

GradingType: An Enum class which defines the type of grading. For now, we have two different grading type, one is *Absolute Grade* and another one is *Deduction Grade*.

Assignment: A class used to specify the assignment, it holds attributes like *GradingType*, *FullCredit*, *Weight*, *Curve* etc. It doesn't have other methods besides *get()* and *set()*. It's considered as a reference object when the users do the grading.

Course: A data structure for the course, it holds a list of assignments and a list of enrolled student as well as a curve object. It has two important methods called *SyncGradesWhenStudentUpdate*() and *SyncGradesWhenAssignmentUpdate*() which use to sync the grade instance when the enrolled student or assignment of this course has been updated. This class is like in the highest level of class hierarchy, it's assembled by multiple components of other class.

CourseTemplate: A class hold a list of assignments, it holds some necessary information of a course that can be used as the template of a course.

Curve: An abstract class, which represent a basic data structure for curve. It only holds attributes like *Amount* which represent the value of a curve. It also has two abstract methods like *getCurveString()* which output the type of curve and *ConvertRawToCurved()* which calculate the score after curved. Use a abstract super class of curve, we can make the future extension easier and reuse the code when add more types of curve.

FlatCurve: The class extends *Curve* and overrides the *getCurveString()* method and *ConvertRawToCurved()* method according to the definition of a flat curve.

PercentageCurve: The class extends *Curve* and overrides the *getCurveString()* method and *ConvertRawToCurved()* method according to the definition of a percentage curve.

Grade: The smallest and most basic unit of the system, it has attributes like *credit*, *assignment* and *comment.* The reason that it has an assignment object as a reference is that every grade has to match to a certain assignment instance. The *comments* field allows user to make comments to each score he grades.

Person: An abstract class, which represent a basic data structure for a person. It only holds attributes like *Name* and *ID*.

Student: The class extends *Person*, it gives a general idea of a student. For now it only has attributes like *Name* and *ID*. The reason we keep it is for future extension. We might need to add more attributes like *BirthDate* or *Sex* in the future.

EnrollStudent: It represent the student who registered a certain course. It holds attributes like *BonusPoint*, *comment*, *Section* and a list of *Grades*. Because each student who enrolled a course can be given a bonus point and comment for the whole course, he can also be assigned to a certain section of the course, and he certainly has a list of scores according to the assignments this course has.

- **Supplementary Class**

CustomizedTable: It extends *AbstractTableModel*. It overrides method *isCellEditable()*, to create a customized Table model, that allows the table in our system can be editable in different scenarios.

ReadExcel: This class has a method *getSheet()* which can parse .xls file from a certain path, enable the system to load student data from Excel.

SelectFile: This class has a method *select(),* which allow user to choose a file from operation system and pass the path of the selected file to our grading system.

SettingChangeListener: An listener interface which has one method called *updatePage().* Because we separate the setting of Curve, EnrolledStudent and Assignment in different frames. These Frames need to implement this listener interface, so that when the data in other frames change, this frame will change accordingly. For example, when we add new assignment in Assignment setting page and confirm it, the grading page will need to be updated and add more columns for the new assignment to be graded.

UUIDGenerator: We use this class to assign a unique ID for each instance we create, so that we can easily track these instance in memory as well as in database.

We need these supplementary classes to improve the convenience and efficiency of our system. Although it's not pivotal but it enable our system to have a better performance and usability.

- ## UI Frames:

The GradingSystem has a Login page that allows different users to sign in. After sign in, it shows a course list page that allow user to manage the information of different courses. After choosing a course, it created a integrated UI frames as CourseFrame. Every user interface and system component is assembled in this frame, and user can use menu bar to switch between pages. By doing this, we can reduce the numbers of pop-up windows in the system and give the user a clean and easy UI. Because it's very annoying for users if there are always a new window jumping out each time they click some button. All the UI frame I designed listed as follow:

LoginFrame: First UI when operate the GradingSystem, allow user to login.
CourseListFrame: The UI show all the current courses, user can edit some basic information of course in this page, as well as add/delete courses.
CourseFrame: An UI for a specific course, it's an intergraded frame that combines all the course/grade operation pages the user need for a course. It allows user to use menu bar to switch different pages.
GradingPage: An UI for the teacher to grade student.
AssignmentPage: An UI for the teacher to change the setting of assignment for this course.

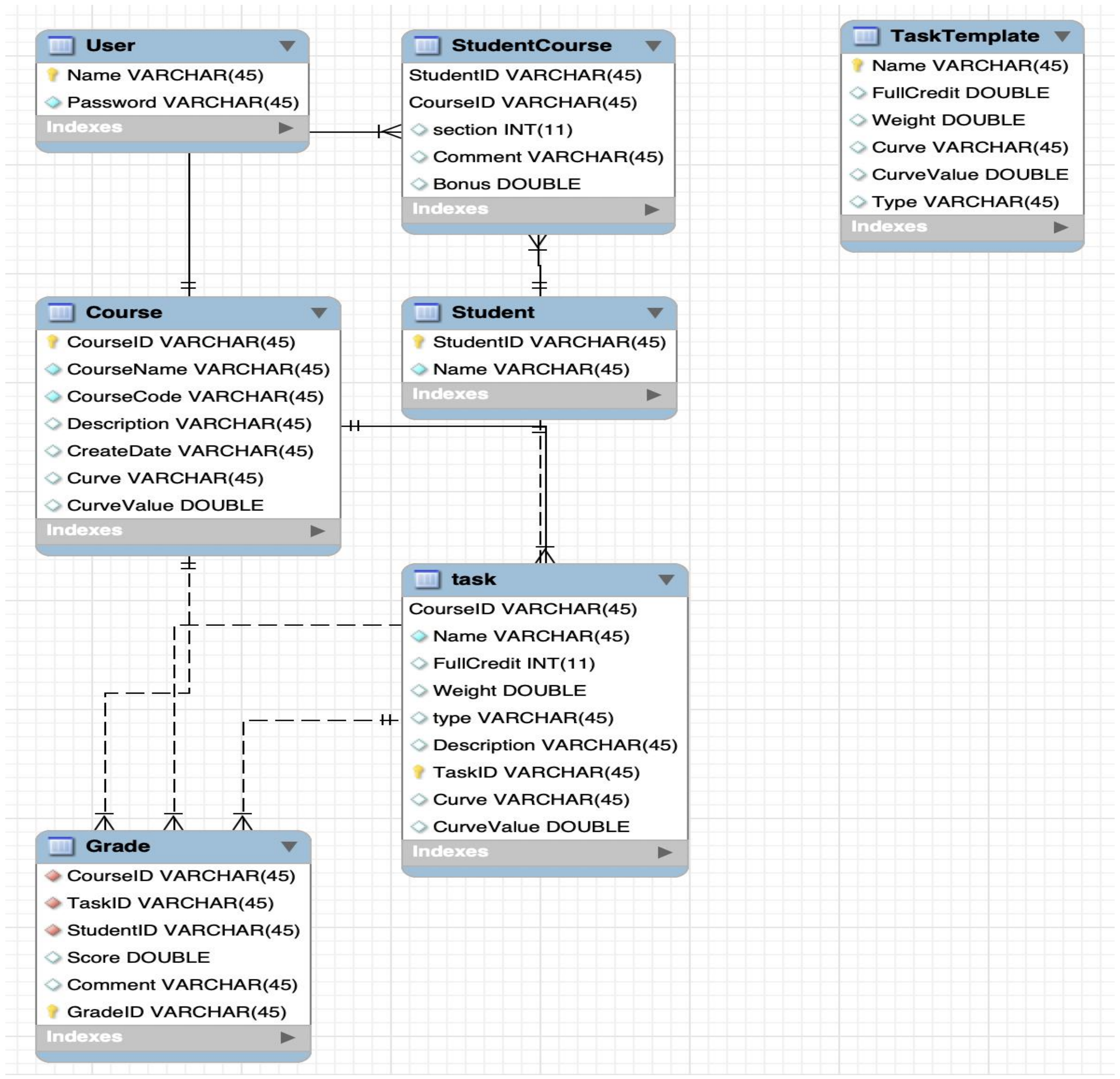
StudentPage: An UI for the teacher to register the enrolled student for this course.
CurvePage: An UI for the teacher to change the setting of curves for this course.
StudentStatsPage: An UI for teacher to view detailed statistics and edit comments for a specific student.
CourseStatsPage: An UI for teacher to view detailed statistics for this course.

- Database

We design seven tables in our database: Course, Grade, Student, StudentCourse, Task, TaskTemplate, User.

In order to improve the runtime speed of the system, we want to reduce the interaction with database as much as possible. But to prevent the data loss when system crashes, we want to save the data in the runtime memory to database when it's necessary. We found a balance point between the use of runtime memory and database by designing the mechanism as follow:
When run the application, it will interact with database and load all the data of courses into memory. When the use finish some kind of operations like edit the assignment, student, curves and grades of the course, he has to click 'confirm' or 'save' button to make a confirmation about the change of information, so that it can notify the database interaction. The reason for doing this is to reduce the times of interaction with the database to speed up the system's response time.

ER graph as follow:



- ## Pros and cons of the feature design

  - ### Course Access

Application: Able to view all of your courses in one consolidated view . Includes a description of the course in addition to just its name.

Excel: Have to manually search into your filesystem to open up a new file for each course. Teacher has a responsibility to maintain an organized filesystem. Can only see file name when selecting the course.

  - ### Course Creation

Application: Able to quickly add and delete courses from the same consolidated list that you select courses from. Every new course starts with sections to keep track of assignments, grades, curves, and students. Able to import a previous course's assignment structure template

Excel: Have to create a new Excel file from scratch and manually name each of the columns, enter in grade calculation formulas, etc. To use settings/calculations from a previous course, you need to duplicate the excel file, delete the old students' grades, etc.

- **Course Creation**

Application: Able to delete courses from the same consolidated menu that you are able to select and view courses.

Excel: Have to manually go into your filesystem and delete the course file.

- **Uploading grades**

Application: Similar grid style as excel, but has preset columns for each assignment, bonus points, etc. Automatically locks cells that should not be editable. Ability to hover over assignment columns and view total points/grading style. Teacher is able to enter in grades in a grading style that makes sense to them for that particular assignment without having to worry about how it is calculated under the hood.

Excel: Have to name columns manually. Have to manually program the data validations, lock cells, etc so that the teacher doesn't make a mistake. The displayed value for each grade cell is the actual value that will be passed into the grade calculation, so it would take extra work to implement a deducted grade input format. Most importantly, teacher has to have a basic knowledge of math to input the current formulas and calculate the final grade, curves, etc.

- **Viewing particular students**

Application: Able to view just one particular student's grades with overall comments for that one student

Excel: Manually hide every other student's row and add a column for overall student Comments.

- **Enrolling Students**

Application: Can manually enter in new students with their name and ID, or import from an existing excel spreadsheet

Excel: Manually create new rows and enter student information into each row.

- **Adding and deleting assignments**

Application: Exact same layout and functionality as adding/deleting courses. Automatically checks that all assignment weights add up to 100%. Each students' grades will be automatically when an assignment is deleted, added, or has its weight changed. (New assignment → new grade starting at 0 for that student) (Delete assignment → total grade calculation updates accurately)

Excel: Have to manually add/delete columns and specify a formula for assignment weight validation. Adding/deleting columns might mess up the previous formula you had for calculating grades, so now the formula needs to be updated.

- **Specifying curves for courses and assignments**

Application: User interface that allows you to view all possible curves at the same time. Calculates curves automatically. Operates consistently no matter how many grades an assignment has or how many students are in the course

Excel: Have to manually enter in formulas. Formulas need to be adjusted if new rows/columns are added for students, assignments, or individual grades

- **Course statistics**

Basically the same pros and cons of the other sections (excel requires you to enter in the formulas, while our application doesn't). One important difference is that as you change the curve formulas in excel, you can see the overall grades change immediately after, since everything is on one view (the excel grid). In our app, you would have to switch back and forth between views if you want to keep readjusting the curve.

- **Overall layout**

Application: Isolated views for entering grades, viewing particular students, assignment/curve/student settings, course stats, etc. Can help to comprehend the information but might be inconvenient to have to switch back and forth

Excel: Everything exists on the same grid, which can be hard to read/overwhelming. Have to manually hide/show columns. But could be convenient to be able to view everything at once.