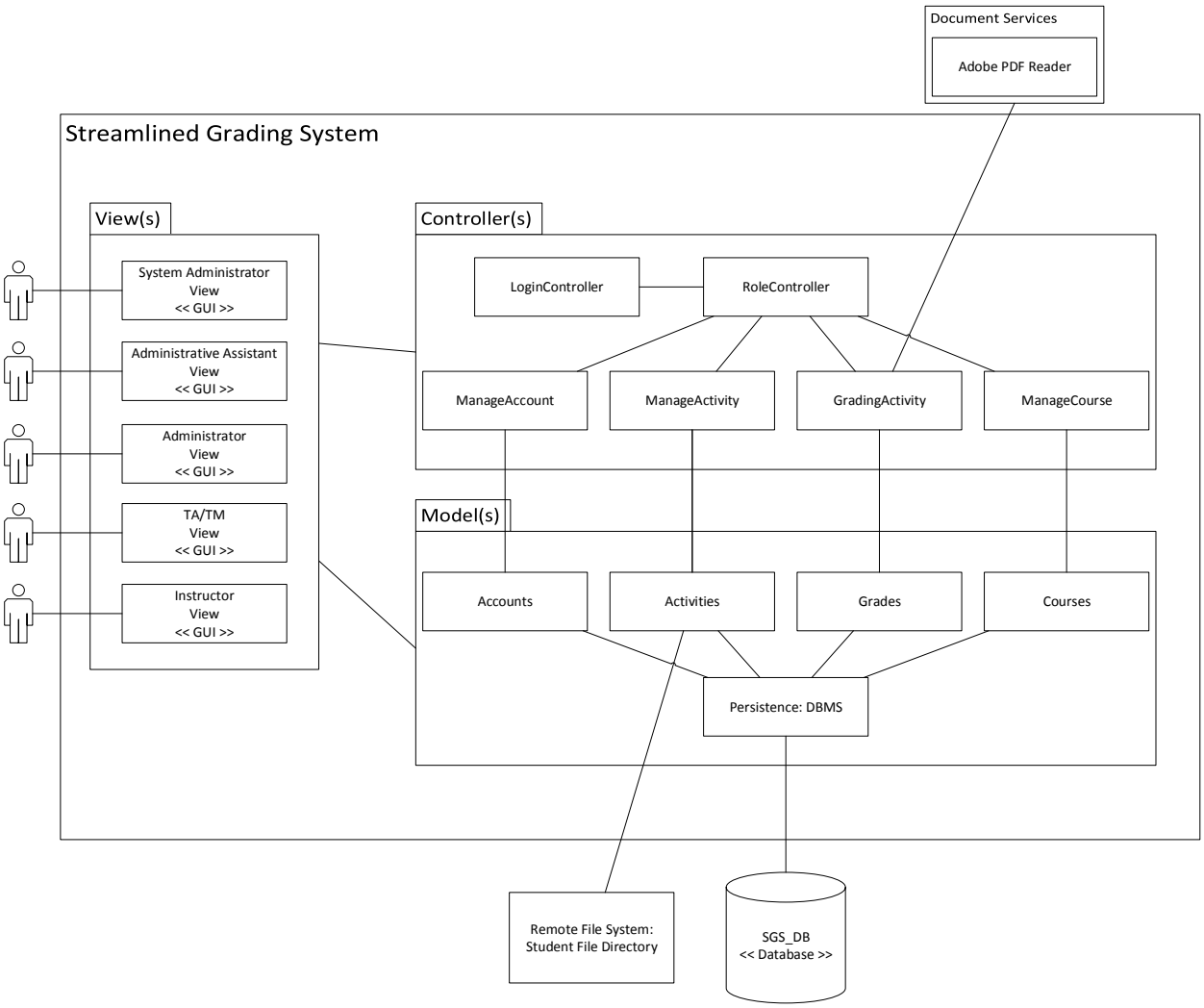
Software Design Document

Table of Contents

I. High Level Design – Architecture Design

a) Architecture Diagram



b) Sub-system Description

1. Model

i. Accounts
Stores user account data that is retrieved to the 'Manage Account' controller and displayed in the 'System Administrator <<GUI>>' view.

ii. Activities
Stores course activity data that is retrieved to the 'Manage Activity' controller and displayed in the 'Instructor <<GUI>>' view. It is also responsible for retrieving remote files from the student's file directory.

iii. Grades
Stores student grades that is retrieved to the 'Grading Activity' controller and displayed in the 'Instructor <<GUI>>', 'TA/TM <<GUI>>' and 'Administrator <<GUI>>' views.

iv. Courses
Stores course data that is retrieved to the 'Manage Course' controller and displayed in the 'Administrator Assistant <<GUI>>' view.

v. Persistence DBMS
Provides services for persistence models and communicates with the system's database.

2. View

i. System Administrator <<GUI>>
Displays <<Accounts>> model data, and sends user actions and inputs to <<Manage Account>> controller.

ii. Administrative Assistant <<GUI>>
Displays <<Courses>> model data, and sends user actions and inputs to <<Manage Course>> controller.

iii. Administrator <<GUI>>
Displays <<Grades>> model data, and sends user actions and inputs to <<Grading Activity>> controller.

iv. TA/TM <<GUI>>
Displays <<Grades>> model data, and sends user actions and inputs to <<Grading Activity>> controller.

v. Instructor <<GUI>>
Displays <<Activity>> and <<Grades>> model data, and sends user actions and inputs to <<Manage Activity>> and <<Grading Activity>> controllers.

3. Controller

i. Login Controller

Responsible for log-in authentication process and password management.

ii. Role Controller

Responsible for directing users to an appropriate controller based on their selected role.

iii. Manage Account

Responsible for account creation, modification, and removal by sending commands to update the 'Accounts' model. It sends commands to 'System Administrator <<GUI>>' view to change the view's presentation of the model.

iv. Manage Activity

Responsible for activity creation, modification, and removal by sending commands to update the 'Activities' model. It sends commands to 'Instructor <<GUI>>' view to change the view's presentation of the model.

v. Grading Activity

Responsible for the process of grading an activity by communicating with the appropriate components within the 'Grading Activity' sub-system which are based on the type of the activity. It is also responsible for assigning grades to a student's work by sending commands to update the 'Grades' model. It acquires document service from an external package which handles PDF files for grading problem set activity. It sends commands to 'Instructor <<GUI>>', 'TA/TM <<GUI>>' and 'Administrator <<GUI>>' views to change the views' presentations of the models.

vi. Manage Course

Responsible for course creation, modification, and removal by sending commands to update the 'Courses' model. It sends commands to 'Administrative Assistant <<GUI>>' view to change the view's presentation of the model.

c) Refined Use Cases

1. Adding a programming activity to a course.

Use Case Name	Add a programming activity to a course (Requirement # 4.1.1, 4.1.2)
Primary Actor(s)	Instructor
Secondary Actor(s)	Database
Precondition(s)	♦ SGS is fully operational and is connected to the database upon executing SGS. ♦ User has signed in as instructor and their account is verified (permanent password has been delivered, replacing the temporary password). ♦ The main menu for the instructor has appeared after successfully signing in. ♦ The instructor has been assigned to the course (that exists) where the activity is to be created.
Main Flow of Events	♦ Use case begins when instructor selects 'Create an Activity' on the main menu after successfully signing in. ♦ The instructor is then redirected to a new window allowing the instructor to choose the desired activity type. ♦ Instructor selects 'Programming' as the type, the course that the activity is for, and clicks 'Next' to proceed to a new window designed for inputting information for a programming activity. ♦ The instructor enters the information for the programming activity: a. Required information at creation time: ▪ Name of programming activity (i.e. Programming assignment 1). ▪ Language of programming activity (C, C++, Java, or Python). ▪ The compiling environment for the activity (Visual studio, Eclipse, or Linux command line). ▪ Instructor creates programming test for programming activity. • Number of test cases must be specified • Paths to the input/output file(s) for each corresponding test case must be specified. For each test case, there will be: ○ Console input file

	o Console output file o Optional input file(s) o Optional output files(s) b. Information that can be entered later: ▪ Path to student's/group file directory ▪ The due date of the activity ▪ Rubric of programming activity The following must be specified: o Description for each of the grading criteria o Maximum grade for each grading criteria ▪ Bonus and Penalty The following must be specified: o 0 – 3 days of duration for the bonus / penalty o Bonus / penalty per day in percentage o A path to a CSV file which contains Student IDs and their Submission dates and times ♦ The instructor selects the 'Create' button upon completion of inputting data into the activity information fields. ♦ SGS performs an analysis of the submitted data for the activity to ensure input data is correct and satisfies requirements. ♦ SGS returns feedback in the form of returning to the main menu if no problems arise during the analysis. Else, a pop-up window will appear explaining the error found in the submitted activity data. ♦ Use case is completed upon successful return to the instructor's main menu.
Post Conditions	♦ Activity is created in the desired course and is reflected in the database. ♦ Instructor is returned to main menu upon successfully creating desired activity.
Exceptional Flow of Events	♦ Incomplete/invalid information has been entered upon selecting the 'Create' button (i.e. required information is not filled, activity name is the same as another existing activity in the course, path(s) does not exist, no test file is specified for a test case) a. System will display a pop-up window indicating which information is incomplete/invalid b. Instructor re-enters the incomplete/invalid information c. Instructor clicks 'Create' button once more and continues to instructor's main menu if successful in filling in all incomplete fields

	♦ Instructor exits the create activity window without selecting the 'Create' button a. System will display a pop-up window indicating that you have unsaved changes and ask if Instructor wishes to proceed or return to create the course b. Instructor will select 'Discard changes' or 'Continue editing' to proceed c. The SGS window will be updated to reflect the option the instructor has chosen and will return to instructor's main menu when activity has been created.

2. Run a previously specified test on a student's submitted code and display the results of that test and the instructor's solution ready to be compared.

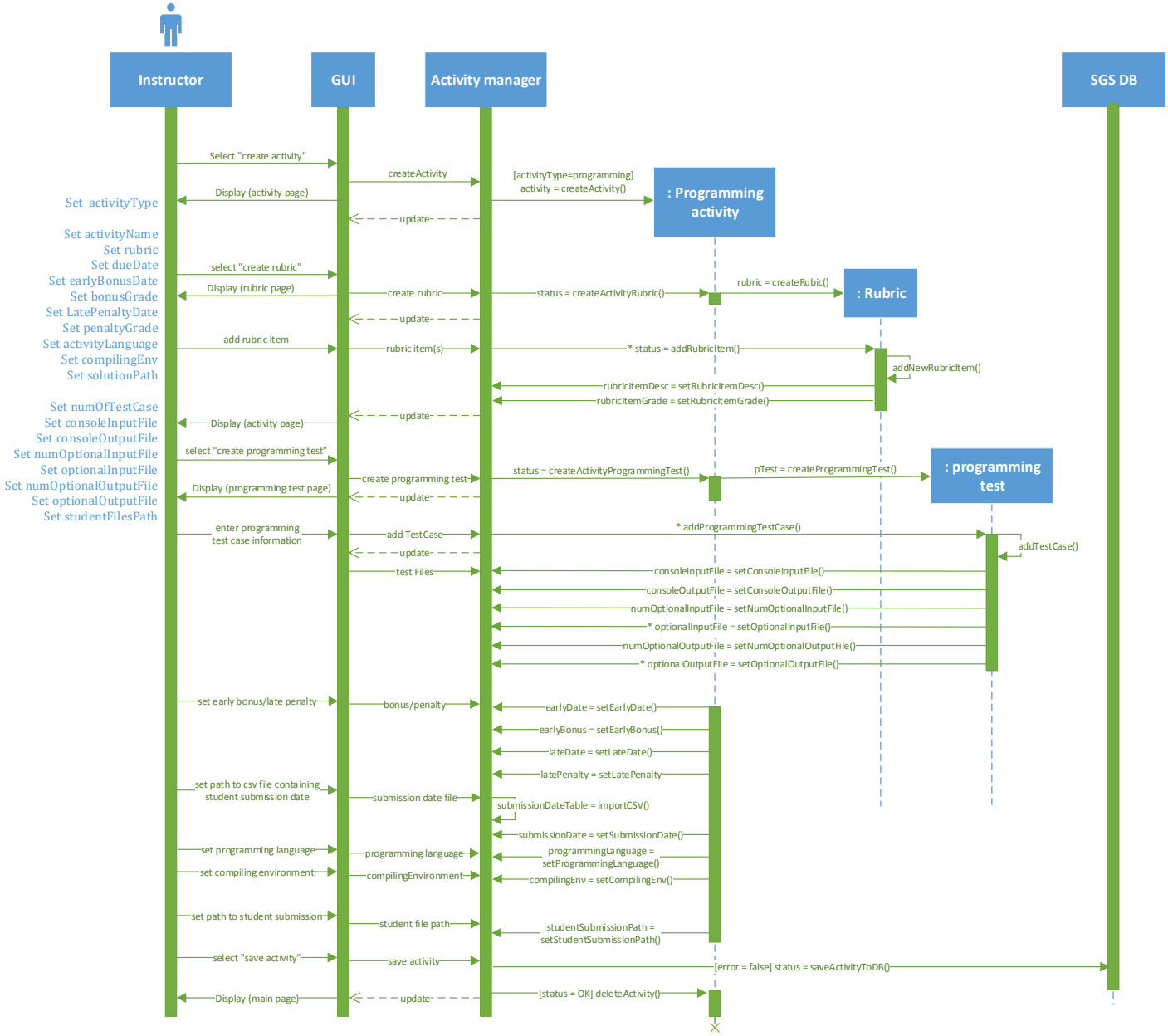
Use Case Name	Run one previously specified test on a students submitted code and display the results of that test and the instructor's solution ready to be compared (Requirement # 5.1)
Primary Actor(s)	Marker (TA(s)/TM(s), Instructor, Administrator)
Secondary Actor(s)	Database, Language-specific Compiler
Precondition(s)	♦ SGS is fully operational and is connected to the SGS database when executed from desktop. ♦ User has signed in with privileges of a 'Marker' and their account is verified (permanent password has been delivered, replacing the temporary password). ♦ The main menu for the 'Marker' has appeared after successfully signing in. (The main menu for each user that has 'Marker' privileges will vary, however each main menu will have at least the button to grade activities) ♦ The 'Marker' has been assigned to the course (that exists) where the activity is to be created. ♦ Previously specified test still exists and is available to be chosen when the Marker is selecting test input files to use for testing student submission.
Main Flow of Events	♦ Use case begins when marker selects 'Grade / Re-grade Activity' on the main menu after successfully signing in. ♦ The marker is then redirected to the 'Grade / Re-grade Activity' window allowing him/her to choose the course (where the programming activity exists). ♦ Upon selecting the course, the user can then select an activity from the list of activities belonging to the chosen course. ♦ The system will then display a new window with the list of students/groups for the selected activity (student number, marks, submission dates, and name of markers will appear as well). ♦ The marker then has to choose a student from the student table list to test their programming activity. ♦ The marker clicks 'Grade / Re-grade' button and is redirected to a new window called 'Grade Activity / Re-grade (Programming Activity)' which contains all necessary information and associated data input fields in order to test a student's specified code.

	♦ The marker now has to choose from the mandatory drop down list containing the test input file (the test file we choose will be the one that was previously specified). ♦ The window is updated to reflect the chosen test input file and the marker may now execute (test) the student submission and solution by clicking the 'Run test' button. ♦ SGS calls the compiler specific to the language and environment of the activity and the compiler output is displayed on the screen (for both the student submission and solution set). ♦ The marker may now visually compare the outputs of the student's code and specified test input. ♦ The marker will be returned to the 'Grade / Re-grade Activity' window when he/she clicks the 'Back' button when done comparing outputs. ♦ The use case ends when the marker presses the 'Main menu' button.
Post Conditions	♦ The marker is returned to their respective main menus upon finishing the task of running a previously specified test on a student's submission.
Exceptional Flow of Events	♦ Student has no submission under the specified programming activity. a. SGS will display a pop-up window indicating that the selected student has no submission file (which means we cannot run a test). b. Marker will click 'OK' to close the pop-up window after being indicated. c. Marker will select another student's submission to test and continue the main flow of events from here. ♦ Student's submitted source code fails to compile. a. A pop-up window will appear indicating that the student's submission failed to compile. b. Marker will click 'OK' to close the pop-up window after being indicated. c. Marker will press 'Back' and return to the previous screen to select another student submission to test and continue the main flow of events from here.

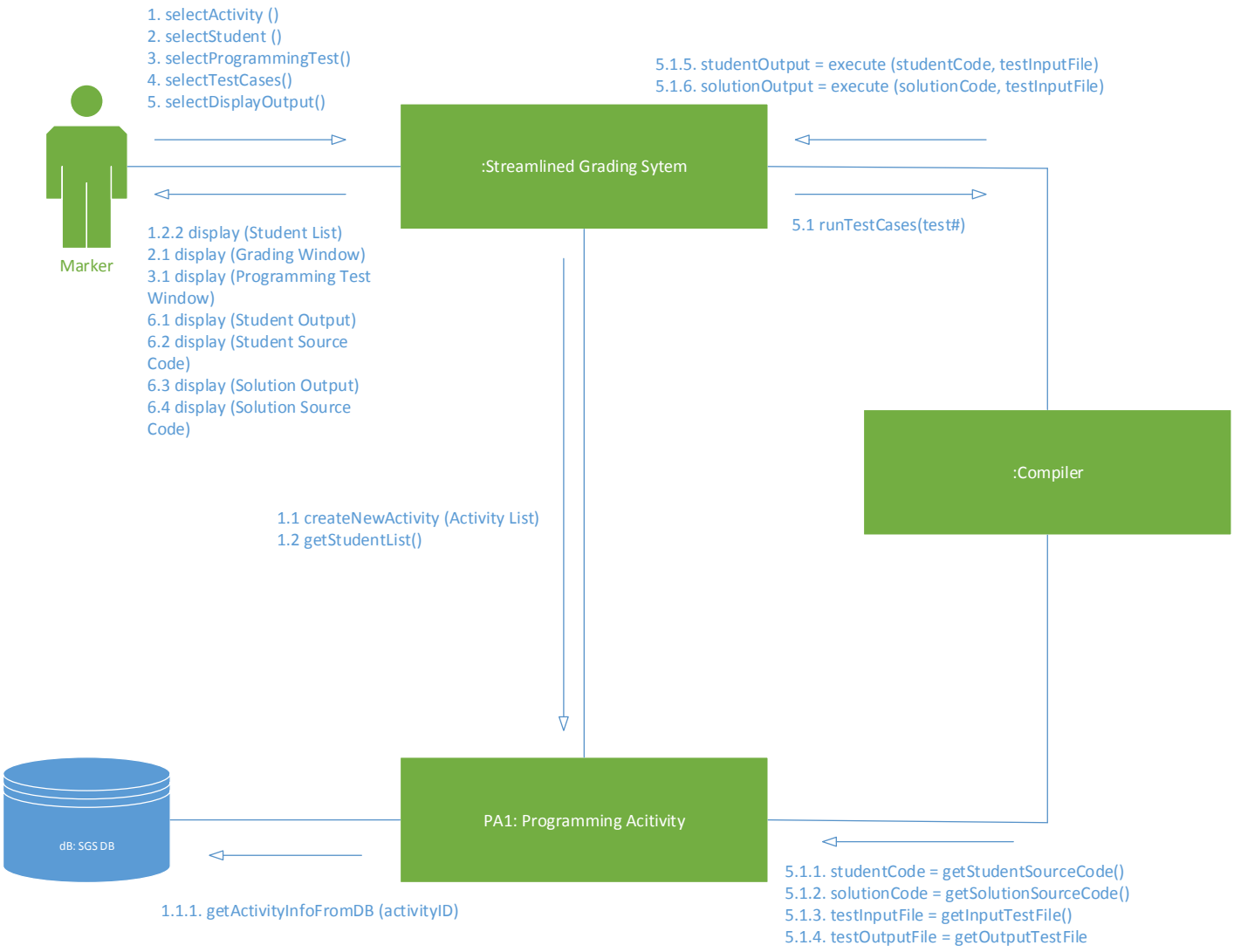
II. Low Level Design – Class Design

a) Interaction Diagrams

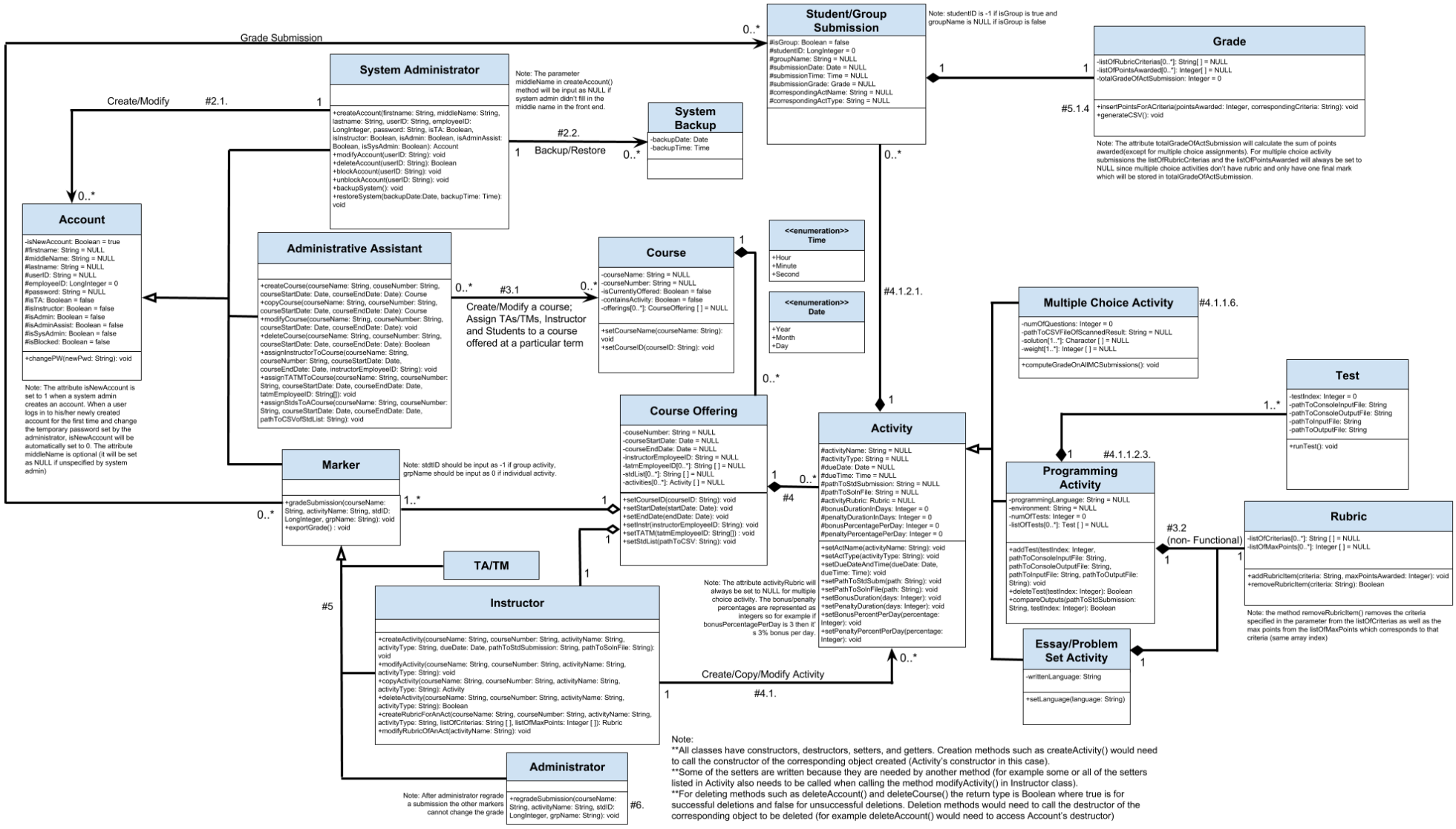
1. Sequence Diagram



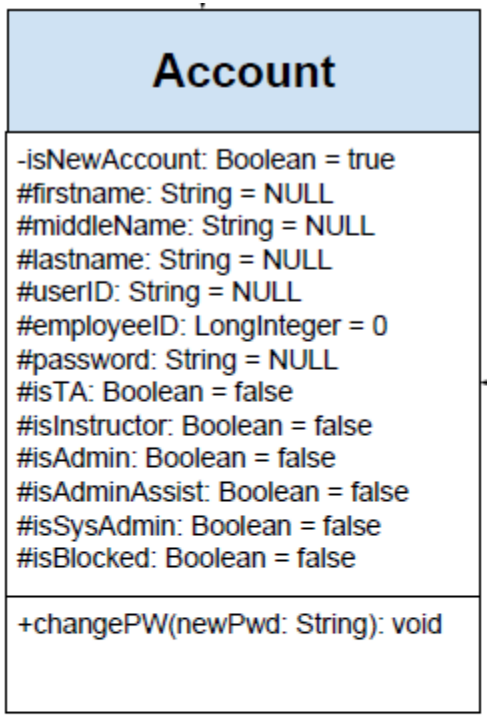
2. Collaboration Diagram



b) Detailed Class Diagram

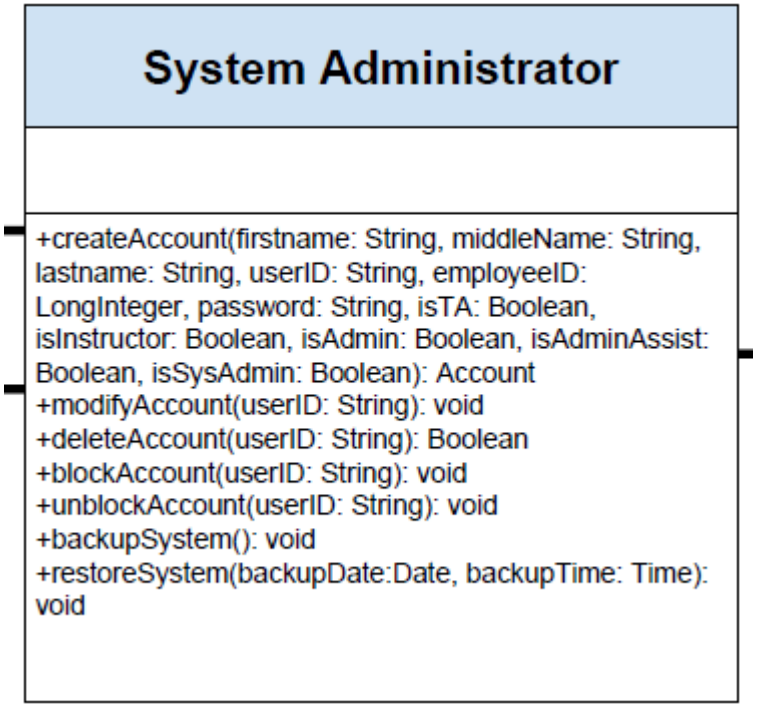


Account



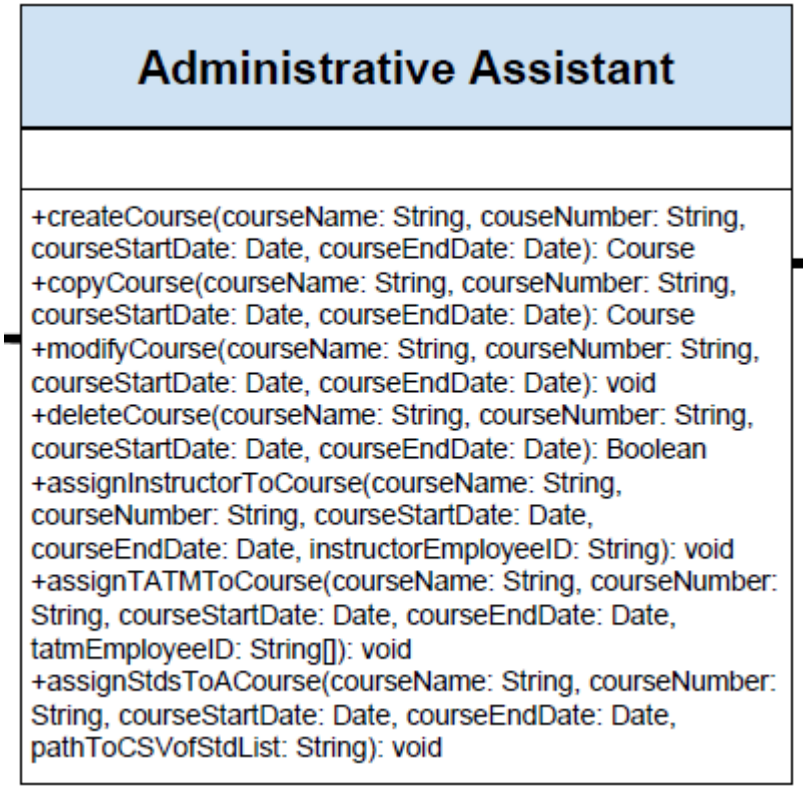
Note: The attribute isNewAccount is set to 1 when a system admin creates an account. When a user logs in to his/her newly created account for the first time and change the temporary password set by the administrator, isNewAccount will be automatically set to 0. The attribute middleName is optional (it will be set as NULL if unspecified by system admin)

System Administrator

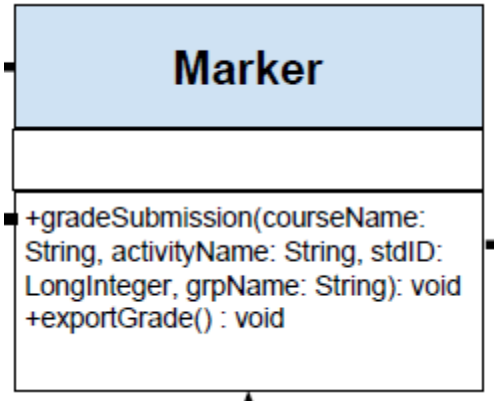


Note: The parameter middleName in createAccount() method will be input as NULL if system admin didn't fill in the middle name in the front end.

Administrative Assistant

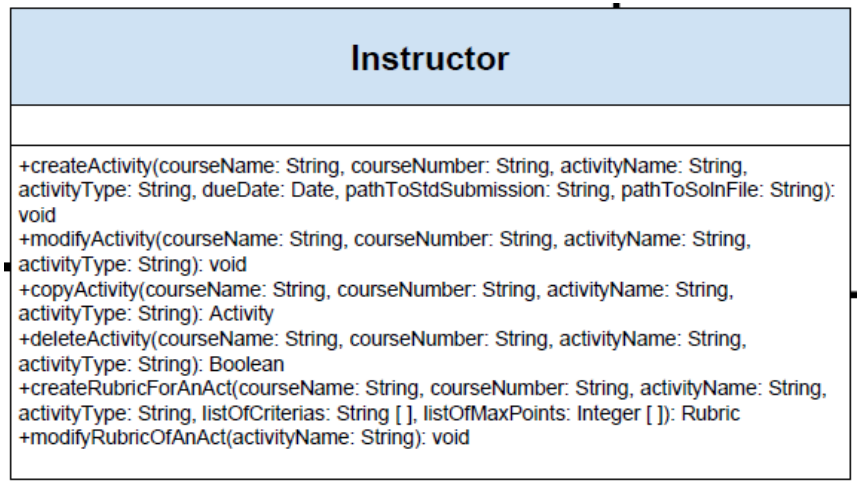


Marker

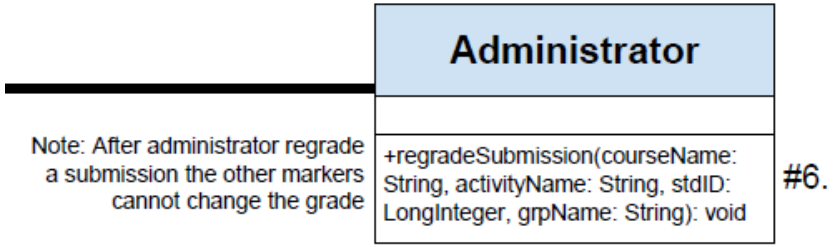


Note: stdtID should be input as -1 if group activity, grpName should be input as 0 if individual activity.

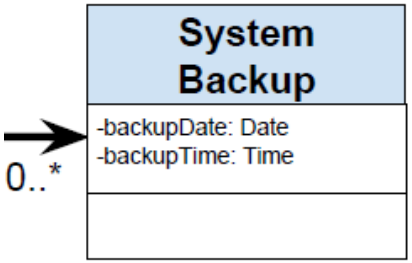
Instructor



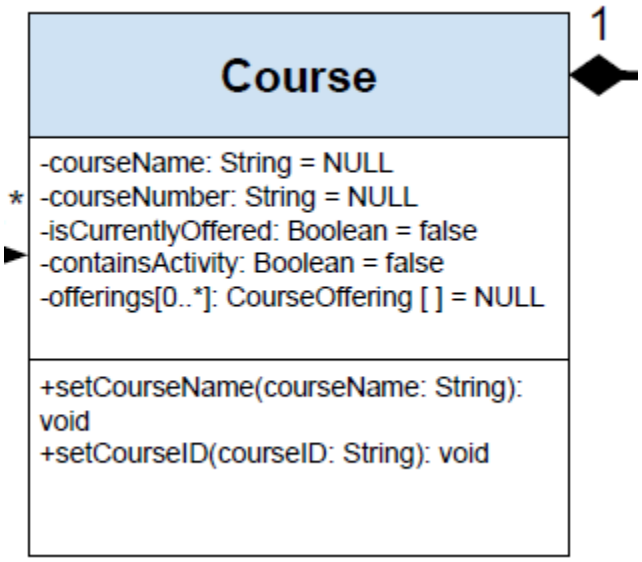
Administrator



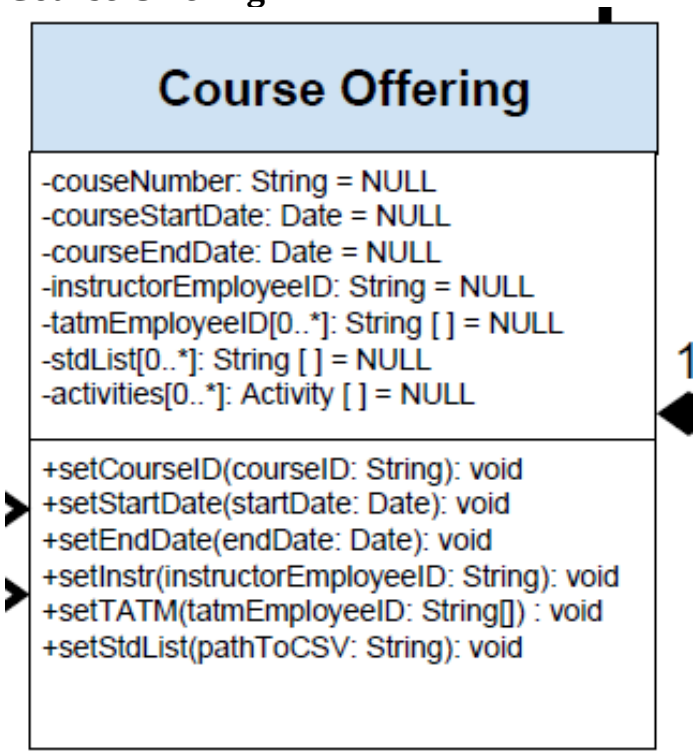
System Backup



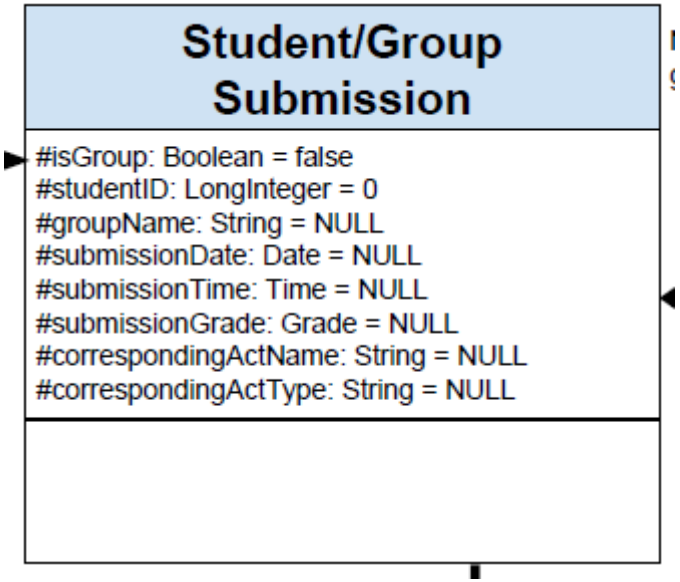
Course



Course Offering

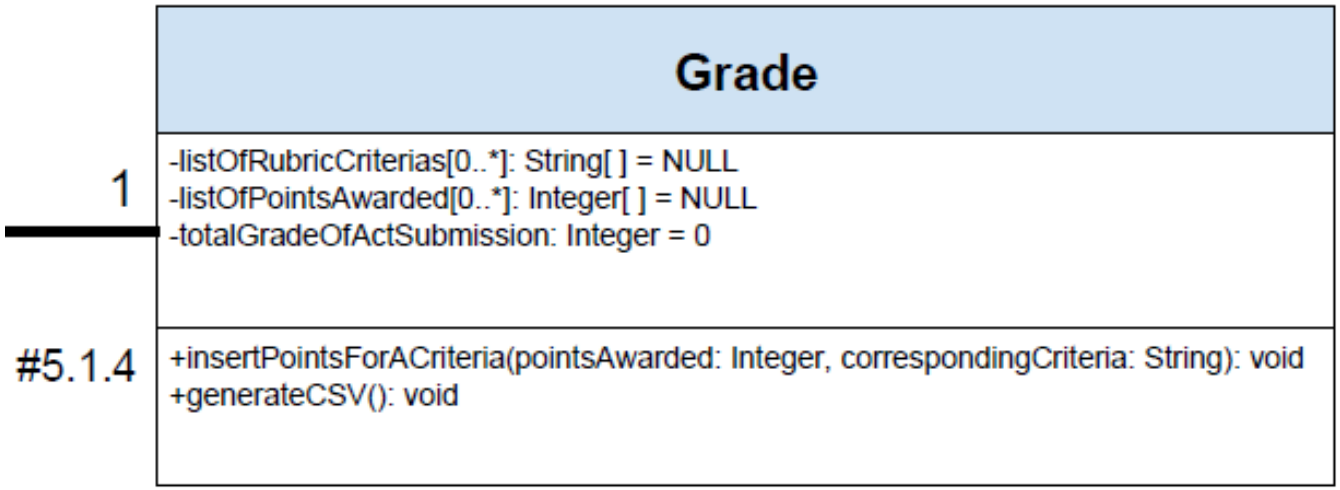


Student / Group Submission



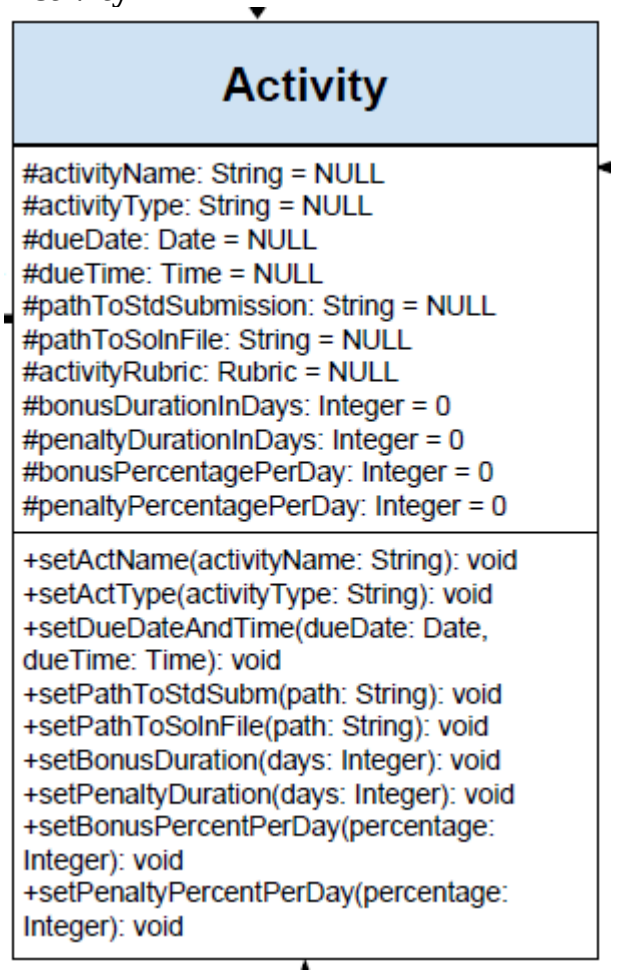
Note: studentID is -1 if isGroup is true and groupName is NULL if isGroup is false

Grade



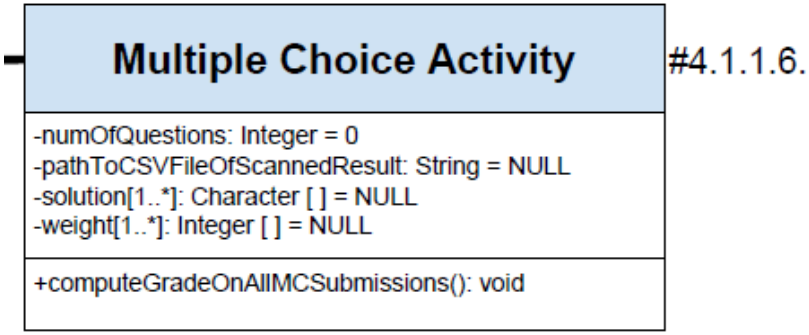
Note: The attribute totalGradeOfActSubmission will calculate the sum of points awarded(except for multiple choice assignments). For multiple choice activity submissions the listOfRubricCriterias and the listOfPointsAwarded will always be set to NULL since multiple choice activities don't have rubric and only have one final mark which will be stored in totalGradeOfActSubmission.

Activity

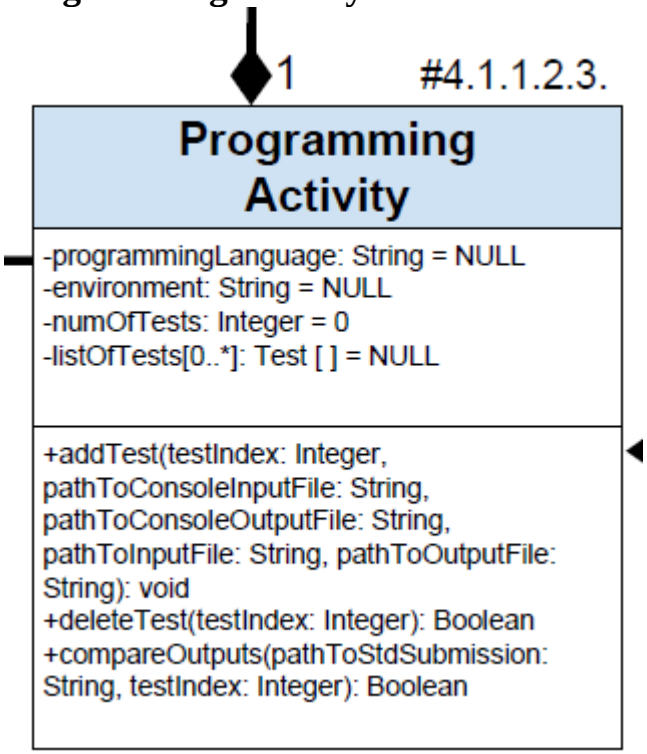


Note: The attribute activityRubric will always be set to NULL for multiple choice activity. The bonus/penalty percentages are represented as integers so for example if bonusPercentagePerDay is 3 then it's 3% bonus per day.

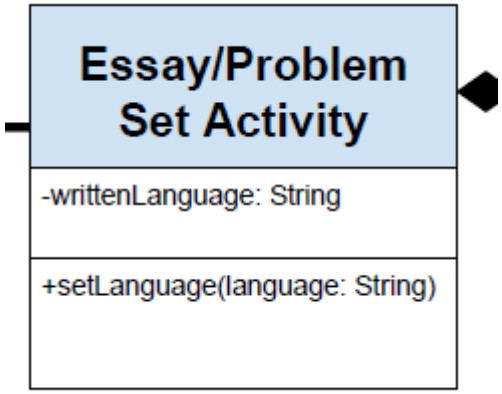
Multiple Choice Activity



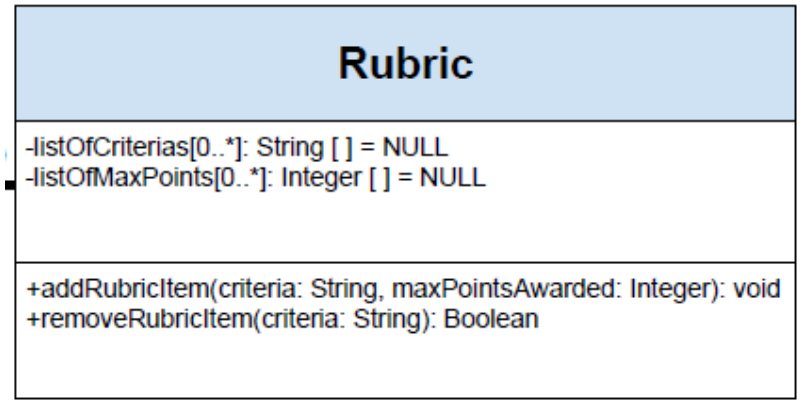
Programming Activity



Essay / Problem Set Activity

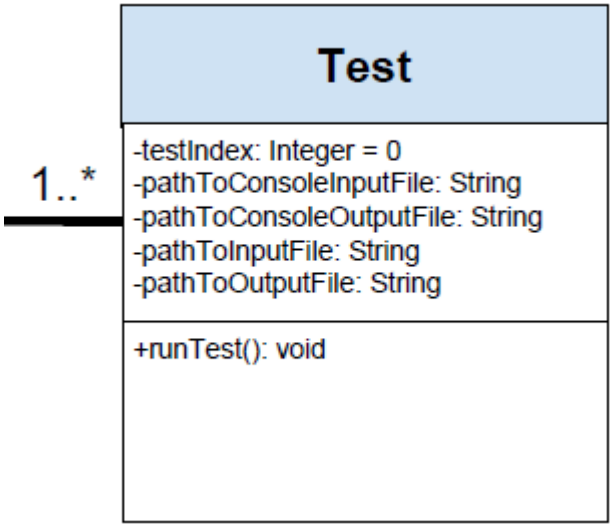


Rubric



Note: the method removeRubricItem() removes the criteria specified in the parameter from the listOfCriterias as well as the max points from the listOfMaxPoints which corresponds to that criteria (same array index)

Test



Note:
**All classes have constructors, destructors, setters, and getters. Creation methods such as createActivity() would need to call the constructor of the corresponding object created (Activity's constructor in this case).
**Some of the setters are written because they are needed by another method (for example some or all of the setters listed in Activity also needs to be called when calling the method modifyActivity() in Instructor class).
**For deleting methods such as deleteAccount() and deleteCourse() the return type is Boolean where true is for successful deletions and false for unsuccessful deletions. Deletion methods would need to call the destructor of the corresponding object to be deleted (for example deleteAccount() would need to access Account's destructor)

III. Data Model

Account Table

EmployeeID	UserID	fname	midname	lname	password	isTA	isInstructor	isAdmin	isAdminAssist	isSysAdmin	isNewAC	isBlocked

Course Table

CourseID	CourseName	CourseNumber

Course Offering Table

CourseID [FK]	**StartDate**	**EndDate**	InstructorEmpID [FK]	hasActivity

Student Table

CourseID [FK]	**StudentID**	fname	lname

Course TA/TM Table

CourseID [FK]	**TAempID**	**startDate**	**endDate**

Course Student Table

CourseID [FK]	**StudentID [FK]**	**startDate**	**endDate**

Activity Table

CourseID [FK]	**ActivityName**	dueDateTime	type	submissionPath	solutionPath	bonusDuration	penaltyDuration

bonusPercentage	penaltyPercentage

Programming Activity Table

CourseID [FK]	**ActivityName [FK]**	programmingLanguage	numOfTest	environment

Programming Test Table

CourseID [FK]	**ActivityName [FK]**	TestID	pathToConsoleInput	pathToConsoleOutput	pathToInput	pathToOutput

Multiple Choice Activity Table

CourseID [FK]	**ActivityName [FK]**	pathToScannedSolution

Multiple Choice Answer Table

CourseID [FK]	**ActivityName [FK]**	QuestionNumber	AnswerKey	Weight

Rubric Table

CourseID [FK]	**ActivityName [FK]**	**CriteriaID**	CriteriaDesription	MaxCriteriaGrade

Student Rubric Grade Table

CourseID [FK]	**ActivityName [FK]**	**StudentID [FK]**	**CriteriaID [FK]**	Grade

Student Submission Table

CourseID [FK]	**ActivityName [FK]**	**StudentID [FK]**	SubmissionDateTime

Group Submission Table

CourseID [FK]	**ActivityName [FK]**	**GroupID [FK]**	SubmissionDateTime