Simon Fraser University
School of Computing Science

Cmpt 275 Project

Date: 2015-08-01

Team name: Eleven ®

Project Deliverable #: 9

Project Deliverable Name: Final Version of Project

Phase(s) covered by this deliverable: Final Version of Class Diagram, Test Plan, Test Case Documents

---

Phase leader(s):

Team members and Student #:

Yinglun Qiao	301191776
Janice Sargent	301160485
Zhi Cheng	301184486
Susan Hamilton	301209641
Seong Jun Kim	301154246
Nari Shin	301178863
Fan Liu	301168674
Te Lun Chen	301200832

Roy Chan 301202770

Grade:

Table of Contents

Revision History

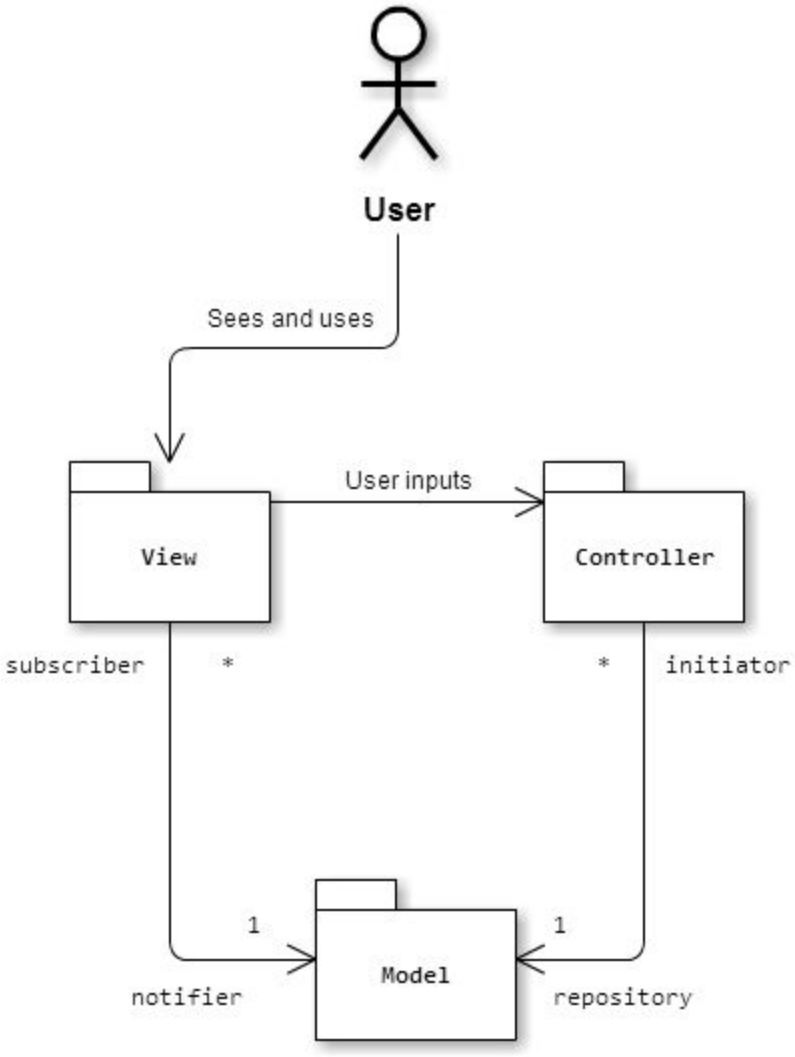
Revision	Status	Publication/ Revision Date	By:
1.0	Created skeleton for the document	*Wednesday June 24, 2015*	Fan Liu
1.1	Created the Refined Use Case 2	*Thursday June 25, 2015*	Yinglun Qiao
1.2	Modified another version of the Refined Use Case 2	*Thursday July 2, 2015*	Nari Shin
1.3	Created the Refined Use Case 1	*Thursday July 2, 2015*	Susan Hamilton
1.4	Created sequence diagram for Refined Use Case 1	*Thursday July 2, 2015*	Janice Sargent
1.5	Created High Level Design Diagram	*Friday July 3, 2015*	Andy Chen
1.6	Modified sequence diagram	*Friday July 3, 2015*	Janice Sargent
1.7	Created Refined Class Diagram	*Saturday July 4, 2015*	Roy Chan, Fan Liu, Josh Kim
1.8	Updated Refined Class Diagram	*Saturday July 4, 2015*	Oscar Cheng
1.8	Edited Refined Use Case 1 and 2	*Sunday July 5, 2015*	Nari Shin, Colin Qiao
1.9	Edited format and minor changes to Refined Class Diagram	*Sunday July 5, 2015*	Roy Chan
2.0	Added Collaboration Digram	*Sunday July 5, 2015*	Janice Sargent

2.1	Added Table Design	*Sunday July 5, 2015*	Oscar Cheng, Colin Qiao
3.0	Revision	*Sunday Aug. 2, 2015*	Fan Liu
3.1	Modified Class Diagram	*Wednesday Aug. 5, 2015*	Colin QIao, Andy Chen

Part I

High Level Design - Architecture Design

1.1 - Architecture Diagram



1.2 - Sub-system Description

1. **View**

   The View displays the Model and is notified whenever the Model is changed.

2. **Controller**

   The Controller manages sequence of interations with the user and sends messages to the Model.

3. **Model**

   The Model mainstains domain knowledge and the central data structure.

1.3 - Refined Use Cases

For Streamlined Grading System (SGS)

1.3.1 - Refined Use Case 1

A. Use Case Name: Add programming activity

Add a programming activity to a course.

B. Requirements Satisified:

- ❑ 3.1.1

C. Participating Actor(s):

1. Initiated by `Instructor`.

D. Preconditions:

1. Streamlined grading system (SGS) is initialized and ready to receive instructions. (Network and database is up).
2. The user has correctly logged in as an `Instructor`.
3. The course that the `Instructor` wants to add the activity to has already been created and `instructor` is assigned to the course.
4. After logging in, the `Instructor` chooses the specific course in which he or she wishes to add a programming activity into.

E. Flow of Events:

1. The `Instructor` selects the 'Create an Activity' option in the 'Manage Activity' screen.
2. `SGS` responds by presenting an interface associated with Create Activity.
3. On the Create Activity screen, the `Instructor` must specify the activity name, activity type as programming activity, activity language as the

chosen programming language, and a link(s) to the activity solution(s) in their appropriate fields.

4. The due date and whether it is a group or individual activity may or may not be specified on the Create Activity screen. The `Instructor` selects the “Next” button to continue and commit the entry.
5. `SGS` will update this information to the Database then display the interface for specifying programming tests.
6. On the Programming Test Specification screen, the `Instructor` can choose to specify the tests or the `Instructor` is able to add them later by choosing “Next”.
7. The `Instructor` specifies the number of programming tests to be run. For each test, the `Instructor` adds the path to one console input, one console output, multiple input files, and multiple output files. The `Instructor` selects “Next” to comitt this entry then proceed to creating a rubric for the activity.
8. `SGS` responds by displaying for the `Instructor` the interface for creating the rubric for the programming activity.
9. On the Rubric Creation screen, the `Instructor` then specifies the rubric name, and the expected outcomes and number of points associated with each outcome for the rubric and then selects “Next” to add a path to the directory containing the students’ submissions.
10. `SGS` responds by displaying for the `Instructor` the interface for specifying the path to the directory containing the students’ submissions.
11. The `Instructor` enters the path and chooses “Done”, or he or she can choose to leave the field blank and choose “Done” to finish creating the activity.

F. Postconditions:

1. All information provided by the `Instructor` are registered and saved into the `Database`.
2. `SGS` sends an acknowledgement back to the `Instructor` displaying a message informing them that the activity was successfully created.

3. SGS sends the Instructor back to the ‘Manage Activity’ interface and is ready for the next action that the Instructor may do.

G. Exceptional Flow of Events:

1. Exceptional Flow of Events # 1: *Modifying Information*

1.1. The Instructor wants to modify information entered in a previous screen.

1.2. The SGS saves all information stored temporarily into the Database until the Instructor enters create at the end and so the Instructor can choose the ‘back’ button and go the screen in which the Instructor wishes to modify the information entered and can then chooses ‘next’.

1.3. The Instructor can then continue as normal in the above flow of events.

2. Exceptional Flow of Events # 2: *Required Fields are not filled.*

2.1. The Instructor does not fill in all the required fields during step 3.

2.2. When the Instructor tries to select “Next” during step 4 to proceed from the Create Activity screen, the system will notify the user which field(s) they have not declared and that the user may not proceed to the next screen until doing so.

3. Exceptional Flow of Events # 3: *Activity Name Already Exists*

3.1. The Instructor enters an activity name which already exists in their course.

3.2. SGS responds by notifying the Instructor of the error and advises them to change the name or to modify the other activities name before creating this activity.

3.3. From this point the Instructor either changes the name or they push cancel.

1.3.2 - Refined Use Case 2

A. Use Case Name: Test Student's Code

Run one 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.

B. Requirements Satisified:

- ❑ **5.11, 5.111, 5.2, 5.21**

C. Participating Actor(s):

1. Initiated by `Marker` (`Instructor` or `TA`).

D. Preconditions:

1. Streamlined grading system (`SGS`) is initialized and ready to receive instructions. (Network and database is up).
2. The course has been created, and the `Marker` (`Instructor` or `TA`) are assigned to the course.
3. The programming activity has been created by the assigned `Instructor`.
4. The user is logged in correctly as `Marker` into the system
5. The previously specified test files are set up and ready to run in correct format and environment. (For this activity, only one test has been set up)
6. The solution of the programming test is correctly set up.

E. Flow of Events:

1. The `Marker` selects the course he or she is assigned to and chooses the "Manage Grades" option.
    2. `SGS` presents to the Marker an interface where the Marker can choose from the list of activities created.
3. On the Manage Grades screen, the `Marker` selects the programming activity that they would like to grade.
4. `Marker` chooses a student from the list of students who have submitted their works to the activity. Since the blind marking policy is used on behalf

of fairness reasons, the Marker will only be able to see the student ID associated with each student.

5. Marker selects the student's submission code file and chooses to compile the code.
    6. SGS responds by presenting an interface that shows the running process.
    7. SGS compiles the student's code, then runs the test that was previously specified by the Instructor.
    8. When SGS finishes running the test, it obtains the results of the student's code.
    9. SGS finishes running the test, and provides options for the Marker to choose between displaying the student's output, displaying the solution code and opening up the rubric with no more than one click away.
10. The Marker selects the "show output" option to view student's output.
    11. SGS presents a pop up window showing the student's output for each of the test.
12. The Marker selects the "show solution" option to view the sample solutions.
    13. SGS presents a pop up window showing the sample solutions provided by the Instructor.

F. Post Conditions:

1. If the use case is successfully initiated:
    1.1. The student's output is saved into the database.
    1.2. The Marker can open up the rubric associated with the activity and start grading by comparing the student's output and the solution.
2. The SGS returns to the initial student selection screen so that the Marker can start grading the next student.

G. Exceptional Flow of Events:

1. **Exceptional Flow of Events # 1:** *Student's code does not compile*

1.1. Student's code submission fails to compile in **Step 7** under **Flow of Events** (e.g. runtime error, syntax error) while running the specified test.

1.2. The `SGS` cannot proceed because results cannot be produced from the code.

1.3. `SGS` will cancel execution and return to the initial screen of the current student. Here the `Marker` can select a new student to grade or determine a default mark for the case where a student's code fails to compile.

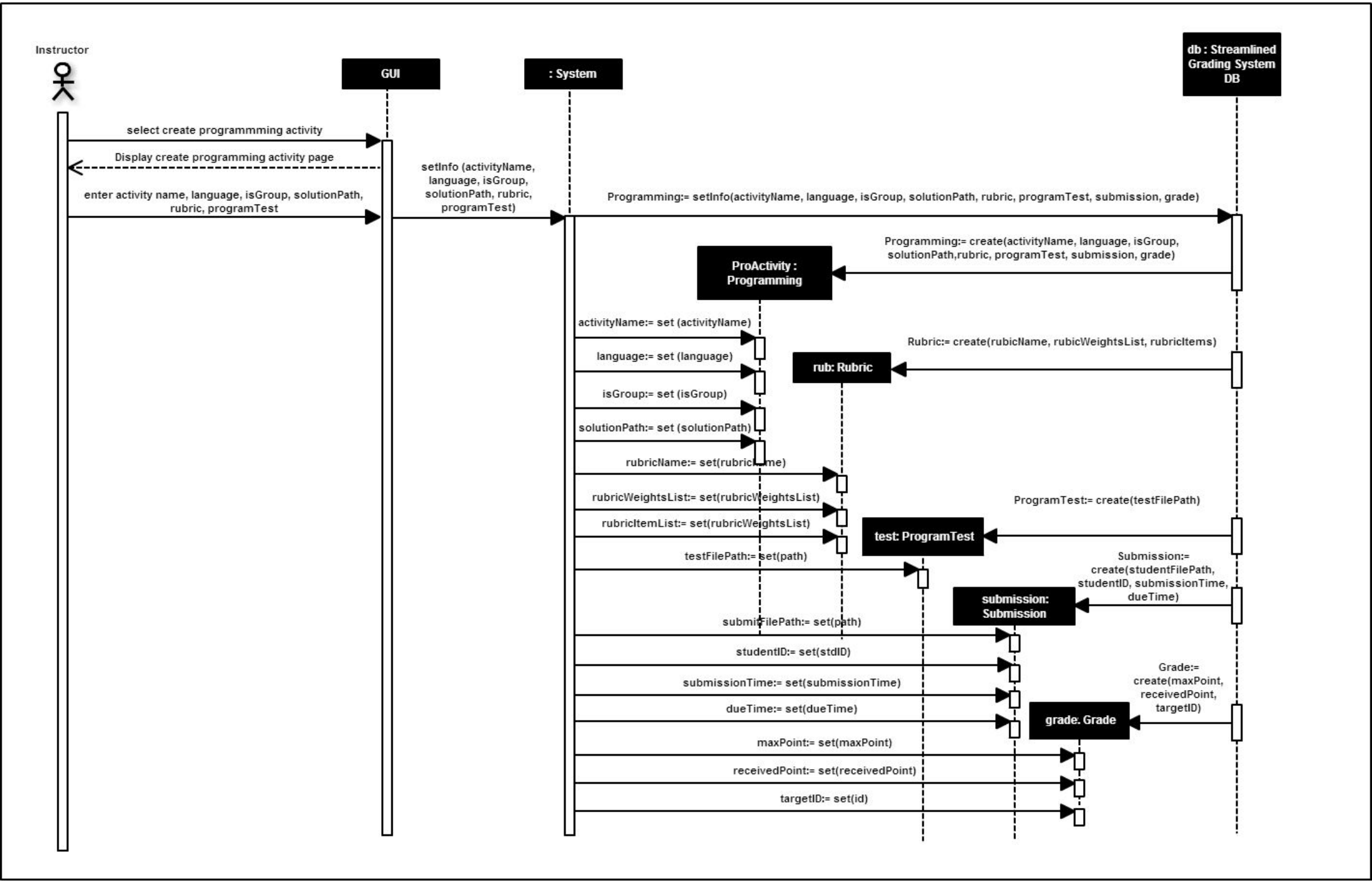
Part II

Low Level Design - Class Design

2.1 - Interaction Diagrams

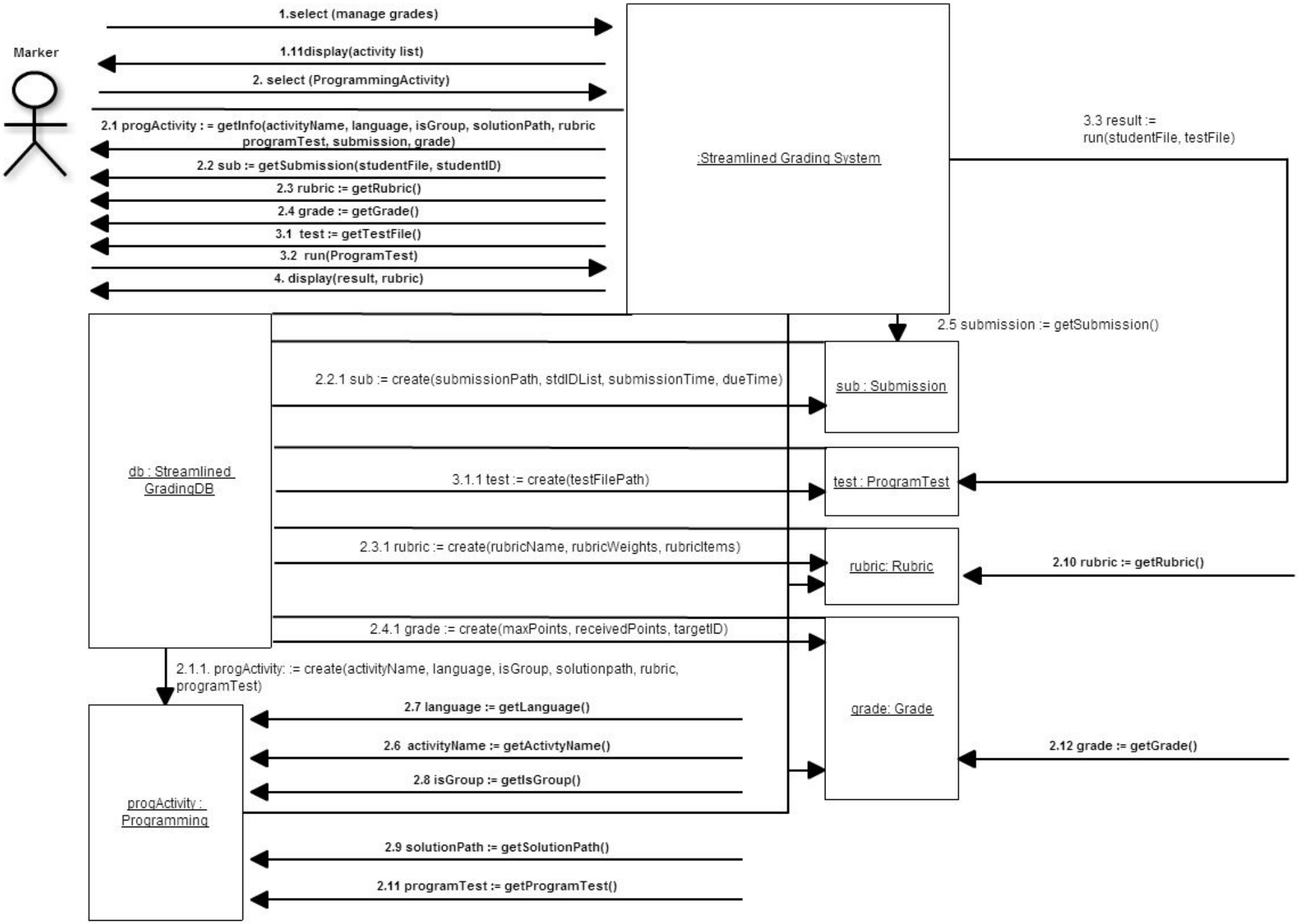
2.1.1 - sequence diagram (for refined use case 1)

- ❏ Please see the attachment on the next page


Instructor
GUI
: System
db : Streamlined Grading System DB
select create programmming activity
Display create programming activity page
enter activity name, language, isGroup, solutionPath, rubric, programTest
setInfo (activityName, language, isGroup, solutionPath, rubric, programTest)
Programming:= setInfo(activityName, language, isGroup, solutionPath, rubric, programTest, submission, grade)
Programming:= create(activityName, language, isGroup, solutionPath,rubric, programTest, submission, grade)
ProActivity : Programming
activityName:= set (activityName)
language:= set (language)
isGroup:= set (isGroup)
solutionPath:= set (solutionPath)
rubricName:= set(rubricName)
Rubric:= create(rubicName, rubicWeightsList, rubricItems)
rub: Rubric
rubricWeightsList:= set(rubricWeightsList)
rubricItemList:= set(rubricWeightsList)
ProgramTest:= create(testFilePath)
test: ProgramTest
testFilePath:= set(path)
Submission:= create(studentFilePath, studentID, submissionTime, dueTime)
submission: Submission
submitFilePath:= set(path)
studentID:= set(stdID)
submissionTime:= set(submissionTime)
dueTime:= set(dueTime)
Grade:= create(maxPoint, receivedPoint, targetID)
grade. Grade
maxPoint:= set(maxPoint)
receivedPoint:= set(receivedPoint)
targetID:= set(id)

2.1.2 - collaboration diagram (for refined use case 2)

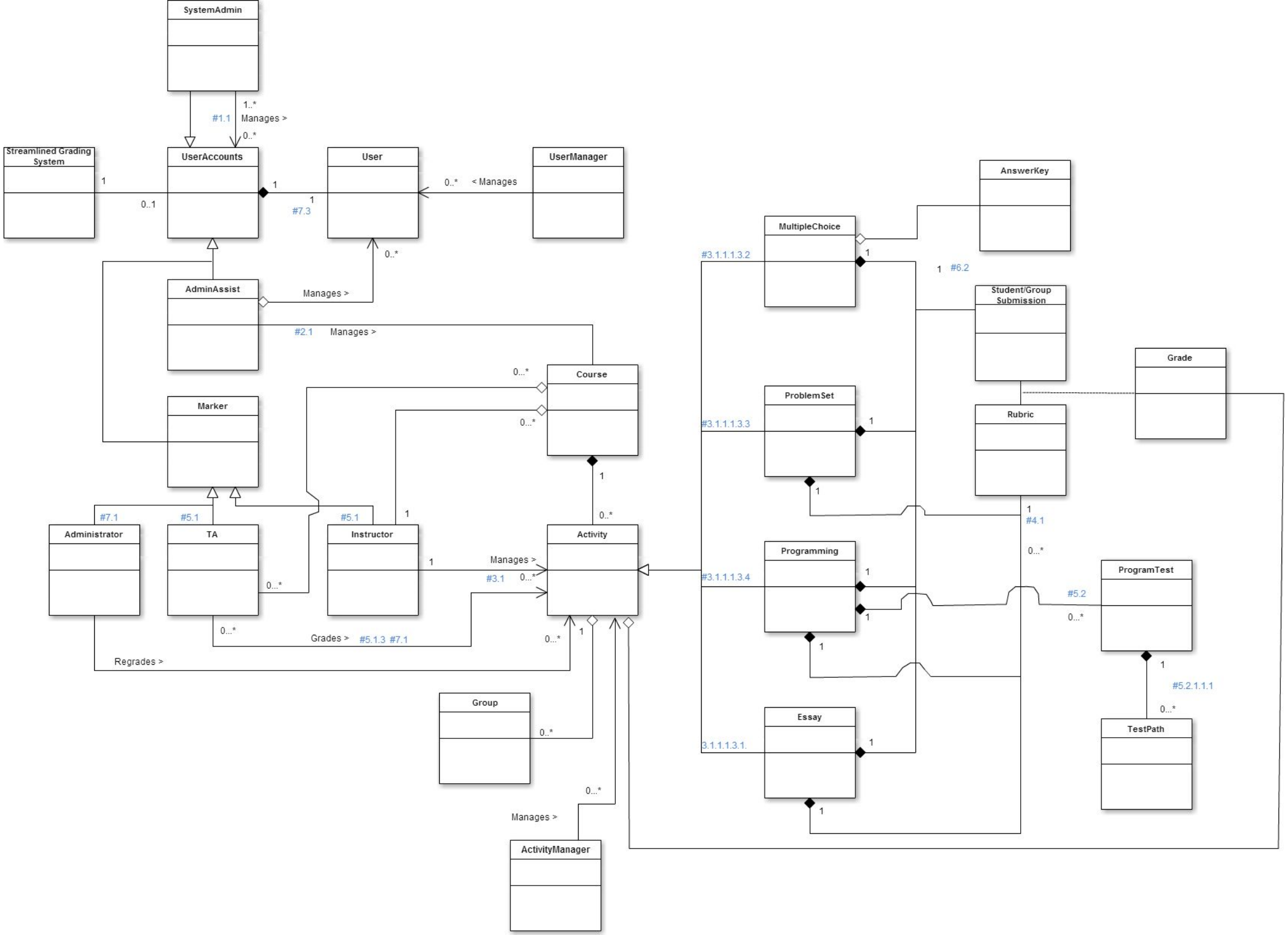
❏ Please see attachment on the next page


Marker
1.select (manage grades)
1.11display(activity list)
2. select (ProgrammingActivity)
2.1 progActivity : = getInfo(activityName, language, isGroup, solutionPath, rubric programTest, submission, grade)
2.2 sub := getSubmission(studentFile, studentID)
2.3 rubric := getRubric()
2.4 grade := getGrade()
3.1 test := getTestFile()
3.2 run(ProgramTest)
4. display(result, rubric)
:Streamlined Grading System
3.3 result := run(studentFile, testFile)
2.5 submission := getSubmission()
db : Streamlined GradingDB
2.2.1 sub := create(submissionPath, stdIDList, submissionTime, dueTime)
sub : Submission
3.1.1 test := create(testFilePath)
test : ProgramTest
2.3.1 rubric := create(rubricName, rubricWeights, rubricItems)
rubric: Rubric
2.10 rubric := getRubric()
2.4.1 grade := create(maxPoints, receivedPoints, targetID)
grade: Grade
2.1.1. progActivity: := create(activityName, language, isGroup, solutionpath, rubric, programTest)
2.7 language := getLanguage()
2.6 activityName := getActivtyName()
2.8 isGroup := getIsGroup()
2.12 grade := getGrade()
progActivity : Programming
2.9 solutionPath := getSolutionPath()
2.11 programTest := getProgramTest()

2.2 - Class Diagram
(final and most detailed)

2.2.1 - Diagram

- ❏ Please see the attachment on the next page


SystemAdmin
1..*
#1.1
Manages >
0..*
Streamlined Grading System
1
UserAccounts
1
User
0..*
< Manages
UserManager
0..1
1
#7.3
0..*
AdminAssist
Manages >
#2.1
Manages >
0...*
Course
Marker
0...*
1
#7.1
#5.1
#5.1
1
Administrator
TA
Instructor
Activity
0...*
Manages >
#3.1
0...*
1
0...*
Grades >
#5.1.3 #7.1
0...*
1
Regrades >
Group
0..*
0...*
Manages >
ActivityManager
AnswerKey
MultipleChoice
#3.1.1.1.3.2
1
1
#6.2
Student/Group Submission
Grade
ProblemSet
#3.1.1.1.3.3
1
1
Rubric
1
#4.1
0...*
Programming
#3.1.1.1.3.4
1
1
1
ProgramTest
#5.2
0...*
1
#5.2.1.1.1
0...*
Essay
3.1.1.1.3.1.
1
1
TestPath

2.2.2 - Attributes and methods

- [visibility]: + means public, - means public, # means protected

1. **Grade Class**
   - 1.1. Attributes:
     - 1.1.1. maxPoints : float
     - 1.1.2. receivedPoints : float
     - 1.1.3. targetID: int
   - 1.2. Methods:
     - 1.2.1. + grade() : void
     - 1.2.2. + enterGrade() : void
     - 1.2.3. + setTargetID(int id) : void
     - 1.2.4. + getTargetID() : int
     - 1.2.5. + getGradeInFloat() : float
     - 1.2.6. + getGradeInPercentage() : float

2. **Activity Class**
   - 2.1. Attributes:
     - 2.1.1. - activityName : String
     - 2.1.2. - activityLanguage : String
     - 2.1.3. - solutionPath : String
     - 2.1.4. - submission : Submisssion
     - 2.1.5. - isGroup : boolean
     - 2.1.6. - rubric : Rubric
     - 2.1.7. - grades : ArrayList<Grade>
   - 2.2. Methods:
     - 2.2.1. + setActivityName(name : String) : void
     - 2.2.2. + setActivityLanguage(language : String) : void
     - 2.2.3. + setSolutionPath(path : String) : void
     - 2.2.4. + setSubmission(submission : Submission) : void
     - 2.2.5. + setGroup(isGroup : boolean) : void
     - 2.2.6. + setRubric(rubric : Rubric) : void

2.2.7. + deleteRubric() : void
2.2.8. + getActivityName() : String
2.2.9. + getActivityLanguate() : String
2.2.10. + getSolutionPath() : String
2.2.11. + getSubmission() : Submissison
2.2.12. + getGrades() : ArrayList<Grade>
2.2.13. + isGroup() : boolean
2.2.14. + getRubric() : Rubric

3. MultipleChoice Class extends **Activity Class**

3.1. Attributes:

3.1.1. - answerKeys : ArrayList<AnswerKey>

3.2. Methods:

3.2.1. + setAnswerKeys() : void
3.2.2. + getAnswerKeys() : ArrayList<AnswerKey>

4. AnswerKey Class

4.1. Attributes:

4.1.1. - questionNumber : int
4.1.2. - key : String
4.1.3. - points : float

4.2. Methods:

4.2.1. + setQuestionNumber(questionNumber : int) : void
4.2.2. + setKey(key : String) : void
4.2.3. + setPoints(points : float) : void
4.2.4. + getQuestionNumber() : int
4.2.5. + getKey() : String
4.2.6. + getPoints() : float

5. ProblemSet Class extends **Activity Class**

5.1. Attributes:

5.1.1. - problemSet : String

5.2. Methods:

5.2.1. + setProblemSet(String ProblemSet) : void

5.2.2. + getProblemSet() : String

6. ProgrammingActivity Class extends **Activity Class**

6.1. Attributes:

6.1.1. - programmingTests : ArrayList<ProgramTest>

7. ProgrammingTest Class

7.1. Attributes:

7.1.1. - testFilePath : String

7.2. Methods:

7.2.1. + setTestFilePath(String path) : void

7.2.2. + getTestFilePath(): String

7.2.3. + isCompiled(String programName, String[] path) : boolean

7.2.4. + runTest(String programName, String[] path) : String

8. Essay Class extends **Activity Class**

8.1. Attributes:

8.1.1. - description : String

8.2. Methods:

8.2.1. + setDescription(String description) : void

8.2.2. + getDescription() : String

9. Rubric Class

9.1. Attributes:

9.1.1. - rubricName : String

9.1.2. - rubricWeights : ArrayList<float>

9.1.3. - rubricItems : ArrayList<String>

9.2. Methods:

9.2.1. + void setRubricName(String name)

9.2.2. + addRubricWeights(weights : float) : void

9.2.3. + addRubricItem(item : String) void

9.2.4. + modifiyRubricWeights(index : int, weights : float) : void

9.2.5. + modifyRubricItem(index : int, item : String) : void

9.2.6. + getRubricName() : String

9.2.7. + getRubricWeights() : ArrayList<float>

9.2.8. + getRubricItems() : ArrayList<String>

10. Submission Class

10.1. Attributes:

10.1.1. - submissionPath : String

10.1.2. - studentIDs : ArrayList<Ingeter>

10.1.3. - submissionTime : Date

10.1.4. - DueTime : Date

10.2. Methods:

10.2.1. + setSubmissionPath(submissionPath : String) : void

10.2.2. + setStudentIDs(studentIDs : ArrayList<Interget>) : void

10.2.3. + setDueTime(Date : DueTime) : void

10.2.4. + getSubmissionPath() : String

10.2.5. + getStudentIDs() : ArrayList<Interget>

10.2.6. + getSubmissionTime() : Date

11. Course

11.1. Attributes:

11.1.1. - activities : ArrayList<Activity>

11.1.2. - instructors : ArrayList<Instructor>

11.1.3. - tas : ArrayList<TA>

11.1.4. - courseNumber : int

11.1.5. - courseName : String

11.1.6. - startDate : Date

11.1.7. - endDate : Date

11.1.8. - studentsName : ArrayList<String>

11.1.9. - studentsID : ArrayList<Long>

11.2. Methods:

11.2.1. + setCourseName() : void

11.2.2. + setCourseNumber() : void
11.2.3. + setStartDate() : void
11.2.4. + setEndDate() : void
11.2.5. + setInstructor(instructorName : String,instructorID : String) : void
11.2.6. + setTA() : void
11.2.7. + createActivity() : void
11.2.8. + deleteActivity() : void
11.2.9. + getActivity(name : String) : Activity
11.2.10. + getAllActivities() : ArrayList<Activity>
11.2.11. + getStudentsNames() : ArrayList<String>
11.2.12. + getStudentsIDs() : ArrayList<Long>

12. User Class

12.1. Attributes:

12.1.1. - firstName : String
12.1.2. - lastName : String
12.1.3. - employeeID : long
12.1.4. - role : String

12.2. Methods:

12.2.1. + setFirstName(firstName : String) : void
12.2.2. + setLastName(lastName : String) : void
12.2.3. + setEmployeeID(Employee ID : long): void
12.2.4. + setRole(role : String) : void
12.2.5. + getFirstName() : String
12.2.6. + getLastName() : String
12.2.7. + getEmployeeID() : long
12.2.8. + getRole() : String

13. UserAccount Class

13.1. Attributes:

13.1.1. - UserID : long
13.1.2. - password : String
13.1.3. - user : User

13.2. Methods:

13.2.1. + setUserID(UserID : long) : void

13.2.2. + setEmployeeID(Employee ID : long) : void

13.2.3. + setTemporaryPassword(password : String) : void

13.2.4. + resetTemporaryPassword(password : String) : void

13.2.5. + resetPassword(password : String) : void

13.2.6. + setFirstName(firstName : String) : void

13.2.7. + setLastName(lastName : String) : void

13.2.8. + setRole(role : String) : void

13.2.9. + getUserID() : long

13.2.10. + getEmployeeID() : long

13.2.11. + getPassword() : String

13.2.12. + getFirstName() : String

13.2.13. + getLastName() : String

13.2.14. + getRole() : String

14. Marker Class extends **UserAccount Class**

14.1. Attributes:

14.1.1. - activities : ArrayList<Activity>

14.2. Methods:

14.2.1. + gradeActivity(activityName : String) : void

15. Instructor Class extends **Marker Class**

15.1. Methods:

15.1.1. + createActivity(courseName : String, courseNumber : int) : void

15.1.2. + deleteActivity(courseName : String, courseNumber : int, activityName : String) : void

15.1.3. + copyActivity(activityName) : void

16. TA Class extends **Marker Class**

16.1. Attributes:

16.1.1. - taID : int

16.2. Methods:

16.2.1. + setTAID(int id)

16.2.2. + getTAID()

17. Administrator Class extends **Marker Class**

17.1. Methods:

17.1.1. + regradeActivity(courseName : String, courseNumber : int, activityName : String) : void

18. AdministrativeAssist Class extends **UserAccount Class**

18.1. Attributes:

18.1.1. - courses : ArrayList<Course>

18.2. Methods:

18.2.1. + createCourse() : void

18.2.2. + getCourse(courseName : String, courseNumber : int) : Course

18.2.3. + deleteCourse(courseName : String, courseNumber : int) : void

18.2.4. + copyCouse(courseName : String, courseNumber : int) : void

19. SystemAdmin Class extends **UserAccount Class**

19.1. Attributes:

19.1.1. - accounts : ArrayList<UserAccount>

19.2. Methods:

19.2.1. + createAccount() : void

19.2.2. + getAccount(userID : long) : UserAccount

19.2.3. + deleteAccount(userID : long) : void

19.2.4. + blockAccount(userID : long) : void

Part III

Data Persistence

3.1 - Table Design

1. grade(maxPoints, receivedPoints, targetID, rubric)

Grade Table			
maxPoints	receivedPoints	targetID	rubric

2. course(couseName, courseNumber, startDate, endDate, activityName, instructors, TAs, submissions, studentsNames, StudentIDs)

Course Table									
courseName	courseNumber	startDate	endDate	activityName	instructors	TAs	submissions	studnetNames	studentsIDs

3. answerKey(questionNumber, key, points)

AnswerKey Table		
questionNumber	key	points

4. multipleChoiceActivity(activityName, questionNumber, key, points)

Multiple Choice Activity Table			
activityName	questionNumber	key	points

5. programmingActivity(activityName, programmingTest, activityLanguage)

programming Activity table		
activityName	programmingTest	activityLanguage

6. problemsetActivity(activityName, description, problem sets)

problemsetActivity table		
activityName	description	problemSet

7. essayActivity(activityName, description)

Essay activity table	
activityName	description

8. submission(studentID, submissionPath, submissionDate, dueTime)

Submission Table			
studentID	submissionPath	submissionTime	dueTime

9. userAccount(employeeID, userFirstName, userLastName, password, role)

	User Account			
employeeID	userFirstName	userlastName	password	role

10. user(firstName, lastName, employeeID, role)

User Table			
firstName	lastName	employeeID	role

11. activity(activityName, activityLanguage, solutionPath, submission, isGroup, rubric, grades)

Activity Table						
activityName	activityLanguage	solutionPath	submission	isGroup	rubric	grades

12. rubric(rubricName, rubricPoints, rubricItems)

Rubric Table		
rurbicName	rurbicPoints	rubricItems