2020-2021 SPRING SEMESTER

CS319 - Object Oriented Software Engineering

Iteration 1 Peereview Design Report

Group Members

Abdul Razak Daher Khatib	21801340
Muhammad Salman Akhtar Soomro	21701446
Ege Moroğlu	21401240
Utku Gökçen	21703746
Yiğit Dinç	21704275

1. INTRODUCTION

1.1 PURPOSE OF THE SYSTEM

Peereview is a group mates review management system where students can perform several academic activities like enrolling in a course, creating/finding groups, and most importantly evaluate each other. The design is intuitive and simple, it is designed in a minimalistic way where users need the least amount of data to be entered, making it easy for both the instructors and students to familiarize themselves with the user interface while maintaining usability and reliability. With customizable management features, Peerview aims to help instructors and students in achieving the goal evaluating their group mates anonymously throughout the project time.

1.2 DESIGN GOALS

Design is a crucial factor for Peereview as it should be simple and easy to use. Following are the descriptions of the critical design goals.

1.2.1 Criteria

Usability

Peereview does not require any training or instructions before use as the design is very intuitive and minimalist with all the necessary features, thanks to its similarity with other commonly used LMS softwares and its simple user interface.

Performance

Peereview rigorously includes all the necessary features like course enrollment/ creation etc. With some tweaks like group chemistry and evaluation analysis graphs. These features are implemented in a simple way to keep the system fast and easy to load regardless of the device used.

Extendibility.

In LMS apps like Peereview, the room for updates and modification is most important as it is designed to maximise user-friendliness and usability. This is why the system was built using the notions of Object Oriented Programming, so that it is easy to extend and modify using various properties of OOP.

Reusability.

Our application is intended for a Bilkent University course, that's said the system would be written in a way which would allow seamless transition to other systems if needed.

1.2.2 Trade-Off:

Memory versus Performance : In order to fully implement the object-oriented system, there will be many objects in the system. Although this increases the memory of the application, the total runtime performance will increase as it will facilitate the reuse of the codes. But, Since the application is written for the web to be used by students, the system needs to be light so that it is easily loaded by all students without worrying about their connection speed. So the memory is preferred over performance.

Understandability versus Functionality : As it is mentioned in the requirement analysis, the whole system must provide the fundamental statistics to review others better. However, adding more numbers and graphs are not the correct answers that increase the understanding. Thus, there exist such challenges that, if too much unrelated information were displayed, the system could become complicated. Hence, the aim to achieve balance in terms of understandability versus functionality stays unsatisfied. To reach the goal of avoiding the functionality of the system being blurred by the excessive use of statistical tools, new ideas are decided to be added after they are compared with each other to test their eligibility and simplicity.

Cost vs. Protabilty

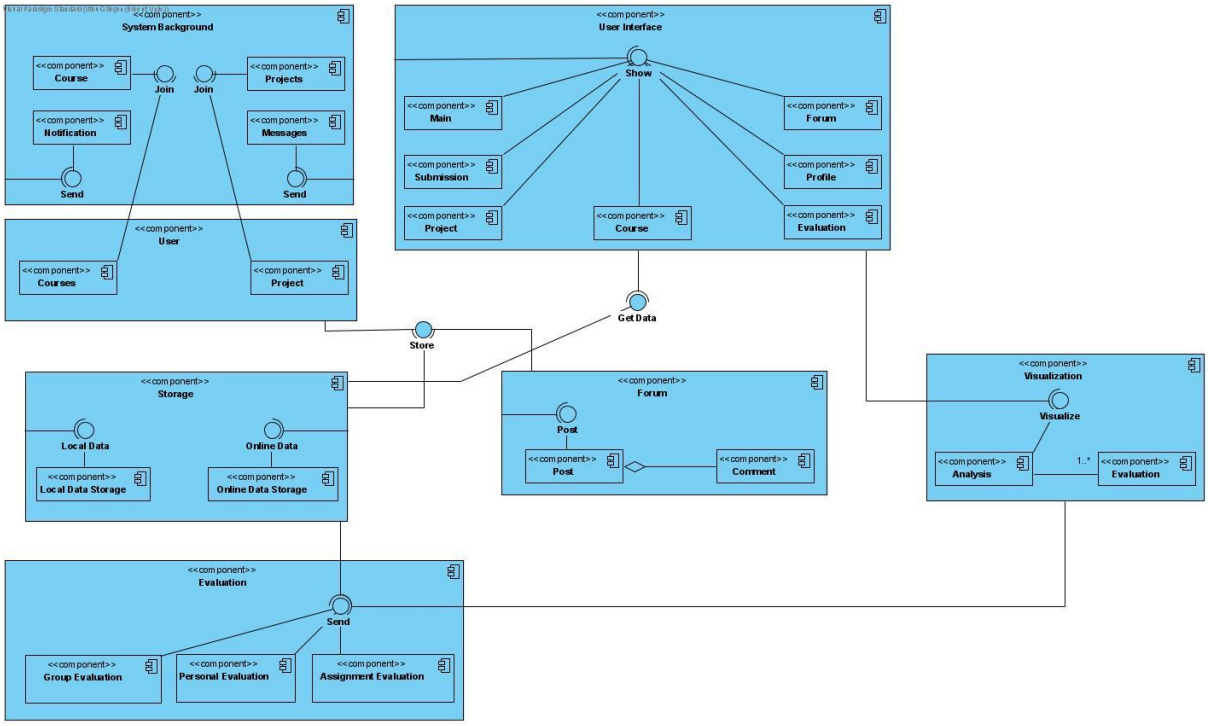
This system is built in a short time, so that it cannot be made completely compatible with mobile phones. However, modern web apps tools allow for a rather smooth experience on phones. Not perfect but the time is tight.

Functionality vs. Robustness

In order to make it light and easy to use we might need to give up on some feature that will result in a less robust product. However, robustness will not be completely neglected, but it will not be preferred over functionality either.

2. SYSTEM ARCHITECTURE

2.1 SUBSYSTEM DECOMPOSITION



https://ibb.co/q1kbwK4

In accordance with the design goals of our project, we divided the system into 7 different subsystems. By dividing the system into subsystems in this way, we aimed to increase the compatibility of the subsystems and minimize the coupling between them.

● **User Interface Subsystem** is the user-viewed part of the system and includes all of the interface components.

● **Forum Subsystem** is one of the subsystem components that allows users to share a post or make a comment to a post

● **Visualization Subsystem** receives the evaluations, analyzes them and creates graphs by visualizing these values. These graphs can be viewed on the User Interface.

● **Storage Subsystem** provides storage of both users' local and cloud data. It also sends data to the User Interface Subsystem to be displayed to the user in the application.

● **Evaluation Subsystem** sends group, personnel and assignment evaluations to the Storage Subsystem for storage and to the Visualization Subsystem for visualization.

● **System Background Subsystem** allows the system to send notifications to users, the users to send messages to other users and join in courses and projects.

● **User Subsystem** is a subsystem that contains users' information and sends this information to Storage Subsystem for storage.

2.2 HARDWARE / SOFTWARE MAPPING

Since the application is web-based, users do not need any hardware other than essential computer equipment which can connect to the internet to use the application. The keyboard is used for many operations in the application like entering username and password, evaluating, grading, and posting. Besides, a mouse is also required as it is used to click the buttons, images, and everything else in the application. Since the drag & drop system is used on the assignment submission page for students and the group creation page for instructors, a mouse is also needed there.

2.3 PERSISTENT DATA MANAGEMENT

In the application, all data will be stored 24/7 up to date in an online database environment. Users will be the main actors in the database system. Each user will have certain features that vary depending on what type of user they are. (Student, TA or instructor) These features will be stored in users' sub-tabs in different ways depending on the types of data. For example, the chemistry point of the group can be accessed by following the Courses -> Groups -> Chemistry Score tabs and this data is stored as a float. In addition, when users make any evaluation, share a post, change their profile, join a new course or group, the data in the storage will be updated instantly.

2.4 ACCESS CONTROL AND SECURITY

When users register to the application, their names, surnames, if they are students, student ids, email addresses, passwords and department information will be taken and this information will be stored in the online database. Users will not have access to this database, so none of the students, TAs, or instructors will be able to manipulate the application data externally and cause any information leakage. In addition, passwords stored in the database will be stored in encrypted form. Thus, even database administrators will not be able to view users' passwords.

2.5 BOUNDARY CONDITIONS

The Peerview application will be a web-based application and users will sign in with their Bilkent mails. If the user does not have an account in the application he/she will be able to sign up with their Bilkent mails. If the user tries to sign in or sign up with an email address other than the Bilkent email, the system will not accept the operation. Since the application will work on the web, so that users do not need to install anything.

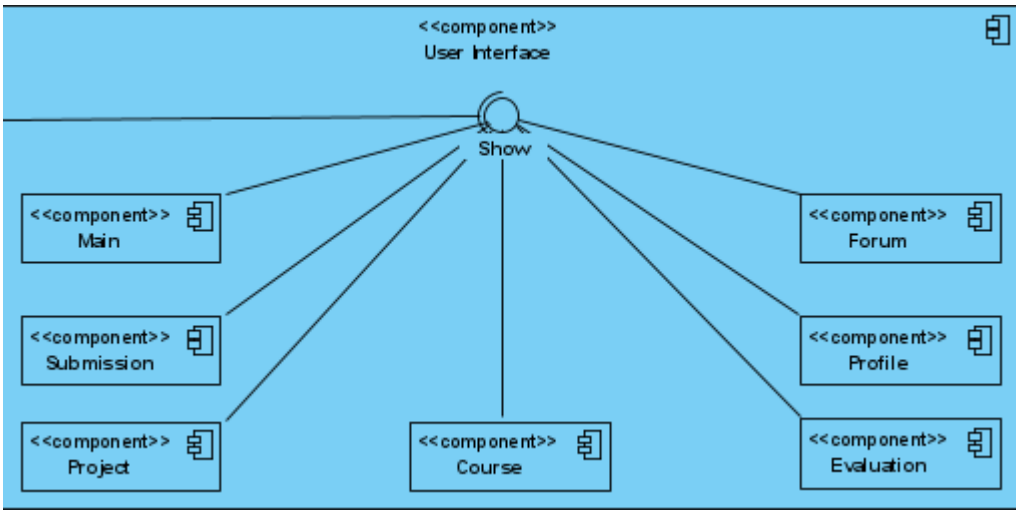
Once the users sign in the application, they will be able to access the pages provided for them depending on their roles in the system (student, grader, or instructor).They will be able to perform the actions provided for them. If the user is a student, he/she will be able to see their assignments or if the user is an instructor they will be able to assign projects and homeworks.

If the system crashes during any action, data will be lost if the user did not submit their work. Users need to submit their work to avoid data loss. If a student wants to submit a report and system crashes before submission, data will not be stored and the user will need to submit the work again.

If the user wants to log out, they can just close the tab or click on “log out”. Logging out will send the user to log in page.

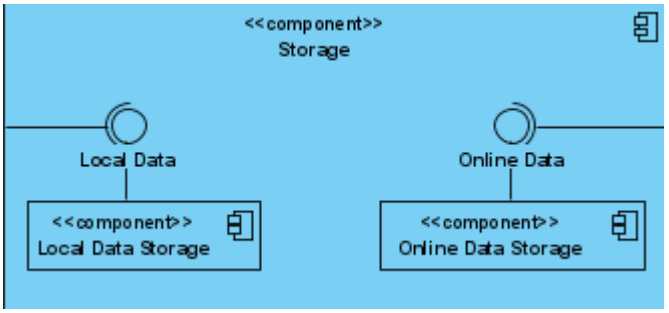
3. SUBSYSTEM SERVICES

3.1 USER INTERFACE SUBSYSTEM



User Interface Subsystem is responsible for providing the user interface for the application. It has seven components that enables users to execute the main functions of the application.

3.2 STORAGE SUBSYSTEM



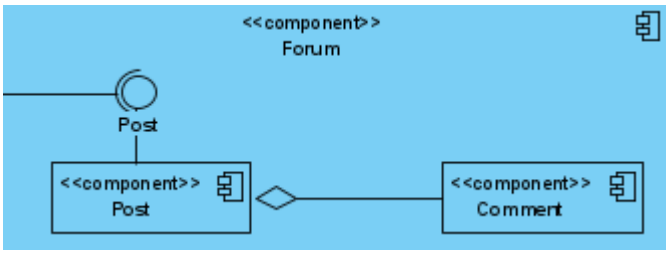
3.2.1 LOCAL DATA MANAGEMENT

This component will handle the local data such as messages, and submitted files.

3.2.2 CLOUD DATA MANAGEMENT

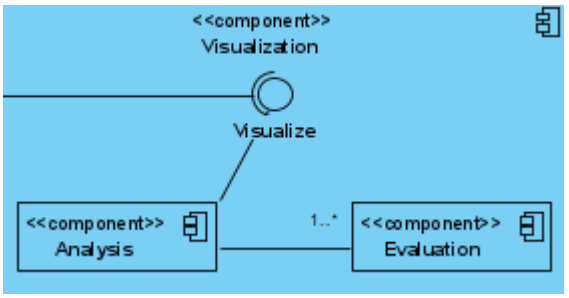
This is the main memory part of our application as all the data, like forums, submissions and courses would be stored in cloud

3.3 FORUM SUBSYSTEM



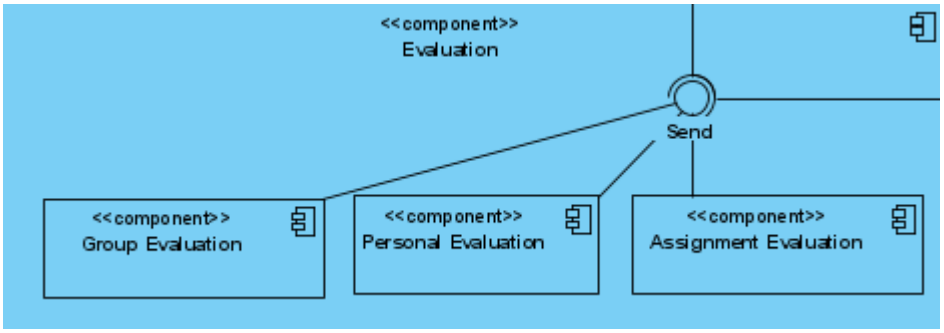
Forum subsystem represents the relation of the forum with other systems. It has two components which are Post and Comment.

3.4 VISUALIZATION SUBSYSTEM



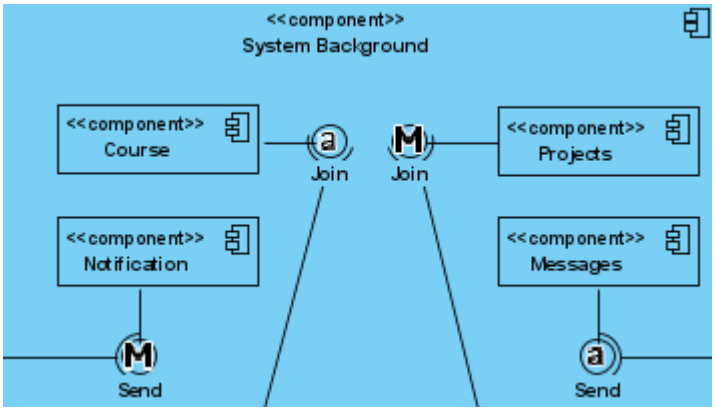
Visualization subsystem is used for analyzing and visualizing evaluations. It has two components which are Analysis and Evaluation.

3.5 EVALUATION SUBSYSTEM



Evaluation subsystem defines how the evaluation works. In Peereview, the evaluation system has group evaluation, personal evaluation (where students will evaluate themselves) and assignment evaluation.

3.6 SYSTEM BACKGROUND SUBSYSTEM



System background subsystem handles the essential pieces of the data in the system.

3.7 USER SUBSYSTEM

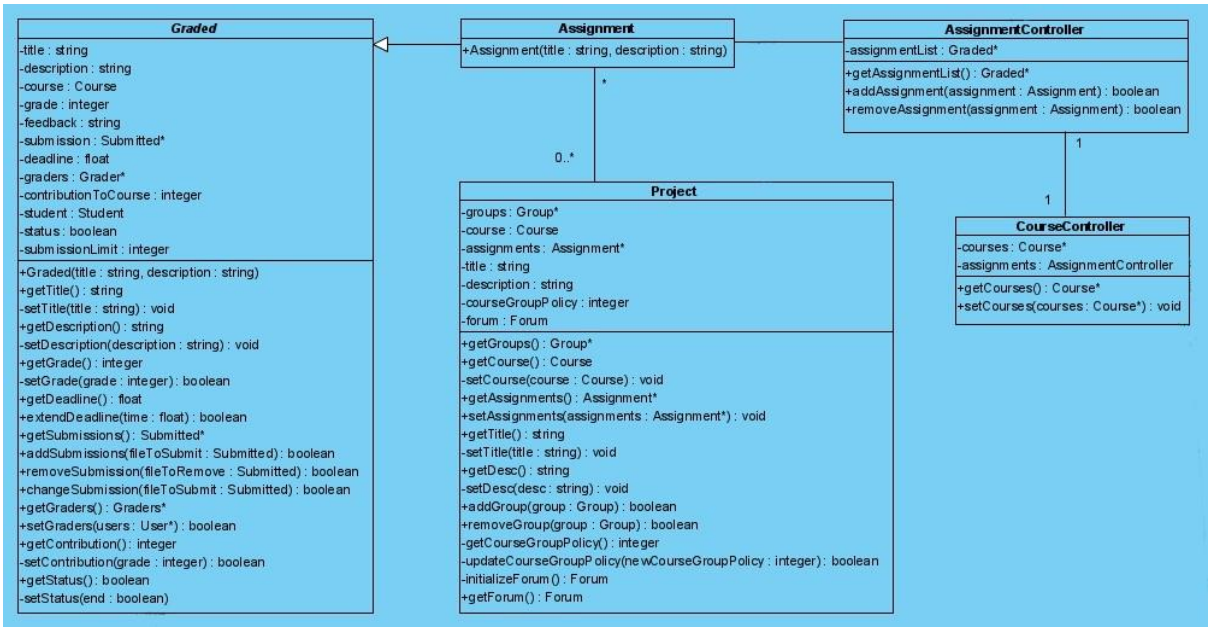
User subsystem defines the users and their main interactions with the system.

4. LOW LEVEL DESIGN

4.1 CLASS INTERFACES

Full object design can be looked at here. Actual image can not fit into this paper. However, partial images might be used to achieve better description of what is designed.

Assignment-Project mechanism



4.1.1 System Class

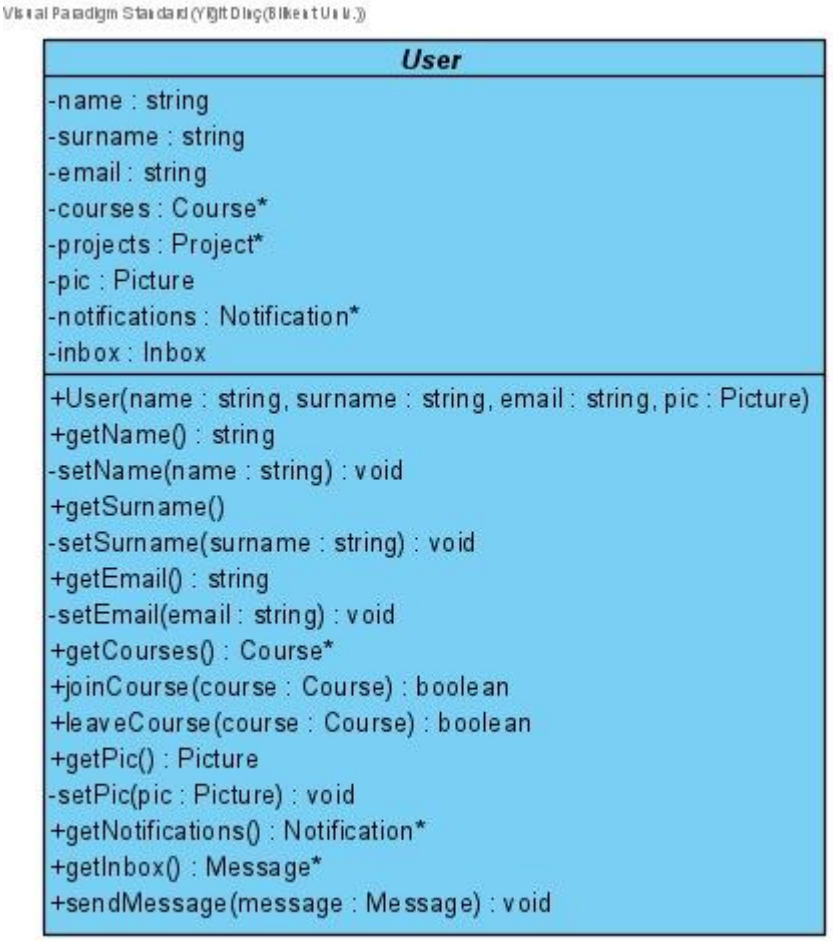
This class is the backbone of the whole system. Here save instances of controllers and all students, graders and instructors. We also deal with messages and notifications here.

Attributes:
Private CourseController courseControl: An instance of the controller for courses. Here we save an instance capable of doing the process we need on courses like adding or deleting courses.
Private UserController userControl: An instance of the controller for users. Here we save an instance capable of doing the process we need on users like adding or deleting them.
Private Student* students: A list of all students in the system.
Private Grader* graders: A list of all graders in the system.
Private Instructor* instructors: A list of all instructors in the system.
Private MessageController messageController: An instance of the controller for messages between users. Here we save an instance capable of doing the process we need on messages like sending or receiving messages.
Private NotificationController notificationController: An instance of the controller for notifications for users for various reasons. Here we save an instance capable of doing the process we need on notifications like sending them or receiving them.

Methods:
Public Course* getAllCourses(): Returns the list of all courses saved in the system.

Protected boolean addCourses (Course course): Here we add a course to the system, then return a boolean representing the success of the process.
Protected boolean removeCourse (Course course): Here we remove a course from the system, then return a boolean representing the success of the process.
Public Student* getAllStudents (): Return a list of all students in the system.
Protected boolean addStudent (Student student): Here we add a student to the system. It is protected not private because it can be provoked from another class.
Protected boolean removeStudent (Student student) : Here we remove a student to the system. It is protected not private because it can be provoked from another class.
Public Grader* getAllGraders (): Return the list of all graders in the system.
Protected boolean addGrader (Grader grader) : Here we add a grader to the system then return true if successful or false otherwise.
Protected boolean removeGrader (Grader grader) : Here we remove a grader to the system then return true if successful or false otherwise.
Private boolean checkExist (User anyUser) : Return true if a student exists; false otherwise.

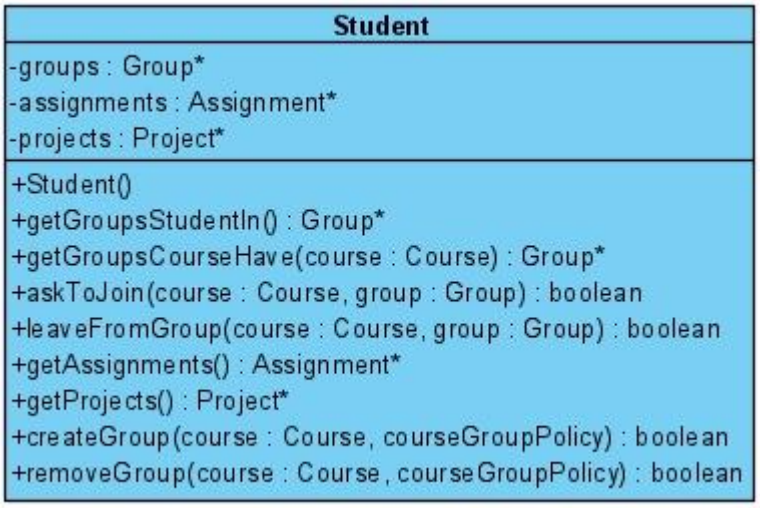
4.1.2 User Class



In this class we represent the user class, which is an abstract class that all user classes inherit. Here we have the most important data that are shared between all users, like name, email and courses. We also deal with notifications and inbox here.

Attributes:
Private String Name: Here we save the user's first name.
Private String surName: Here we save the user's last name.
Private String email: Here we save the user's email.
Private Course* courses: We save the course that a user is a part of. For all users, students, graders, and instructors, we have courses that they are a part of.
Private Picture pic: A personal picture of the user.
Private Notification* notifications: An array of notifications that a user has.
Private Inbox inbox: An inbox that contains all messages that a user has.

Methods:
Public User (String Name, String surName, String email, Picture pic): Here we instantiate a user using the most important and distinguishable features, like name, surname, and email.
Public String getName (): Returns the name of the user.
Public String getSurName (): Returns the surname of the user.
Public String getEmail (): Return the email of the user.
Private void setName (String name): Sets the name of the user.
Private void setSurName (String surName): Sets the surname of the user.
Private void setEmail (String email): Sets the user's email.
Public Course* getCourses (): return a list of the user's courses.
Public boolean joinCourse (Course course): Send a request to join a request, if successful return true, otherwise returns false.
Public boolean leaveCourse (Course course): Sends a request to the system to leave a course if successful returns true, otherwise returns false.
Public Picture getPic(): Returns the user's picture.
Private void setPic (Picture pic): Sets the user's personal picture.
Public Notification* getNotifications(): Returns a list of all of the user's notifications.
Public Message* getInbox(): Returns a list of all messages in the inbox for a user.
Public boolean sendMessage (Message message): Send a message to a user and then confirm if the other user received it by return true, or false otherwise.



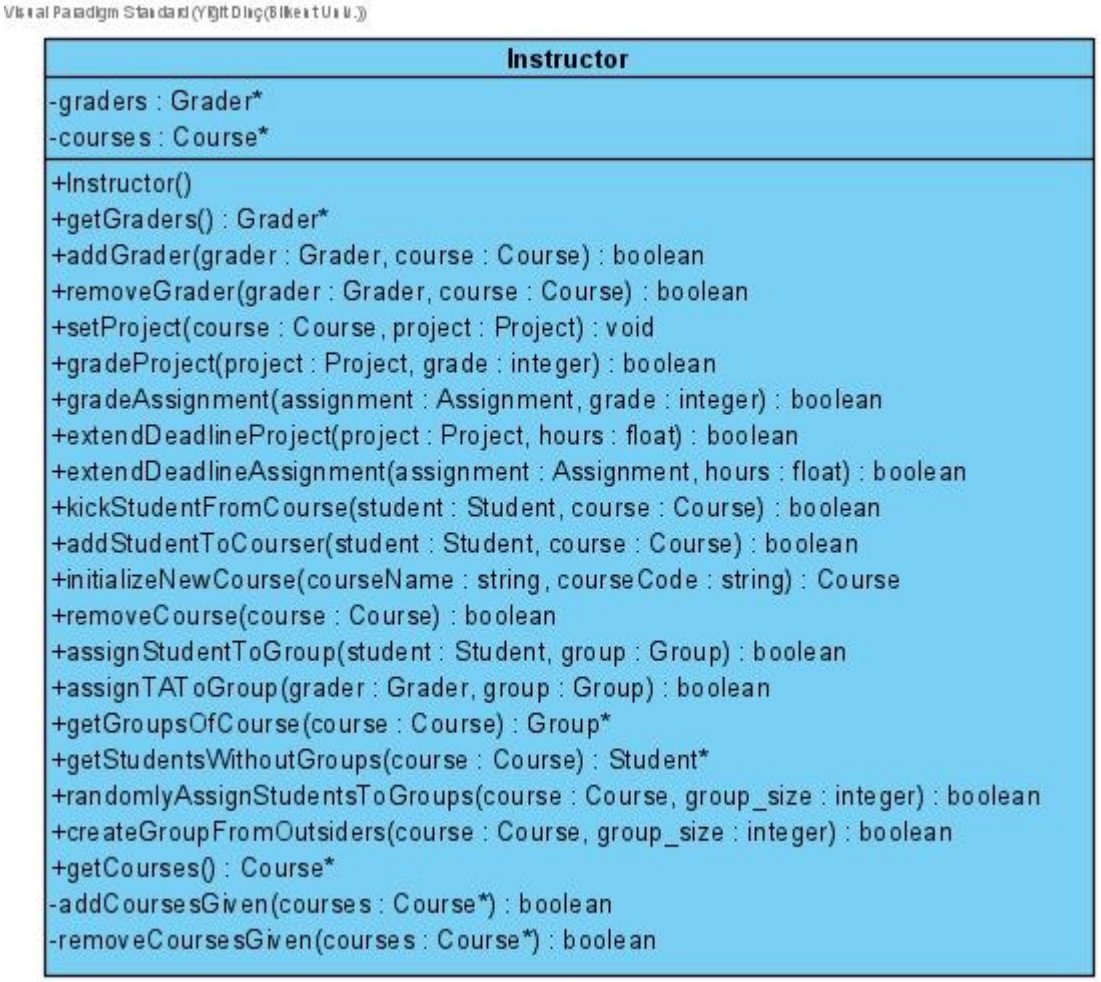
4.1.3 Student Class

This class represents the student, it implements the user class, the main user of the system. In this class we have the info of the student, like the name, email, and email, and the courses, groups, and projects that she chose to enrol in.

Attributes:
Private Group* groups: This attribute represents the array of the groups the student is a member of.
Private Assignment assignments: The attribute represents the array of assignments that the student has to submit.
Private Project* projects: This attribute represents the projects that the student is enrolled in.

Methods:
Public Student (String name, String ID, String email) : This is a constructor used to initialize the object of a student using the most important information about her.
Public Group* getGroupsStudentIn() : Returns all groups that the student is a member of.
Public Group* getGroupsCourseHave() :
Public boolean askToJoin (Course course, Group group) : Here we use this when a student sends a request to join a group when there is a place available during the groups' formation stage, returns true in case of success of sending the request.
Public boolean leaveFromGroup(Course course, Group group) : Here we use this when a student sends a request to leave a group returns true in case of success of leaving the group.
Public Assignment* getAssignments() : Returns all assignments that a student has to submit.
Public Project* getProjects() : Return projects that the user is enrolled in.
Public boolean createGroup(Course course) : Here the user sends a request to the system to create a group, if the course policy allows that the group is created, otherwise it is rejected. A boolean is returned accordingly.

Public boolean removeGroup(Course course): Here the user sends a request to the system to remove her group, if the course policy allows that the group is removed, otherwise it is rejected. A boolean is returned accordingly.



4.1.4 Instructor Class

This class represents an instructor, it implements the user class. In this class we represent an instructor with her main info and the methods related to her authority.

Attributes:
Private Grader* graders: This represents the graders that the instructor appointed. Each Grader is assigned to a course but they are also stored according to the instructor so we can limit their access according to the instructor's access and in order to keep track of things.

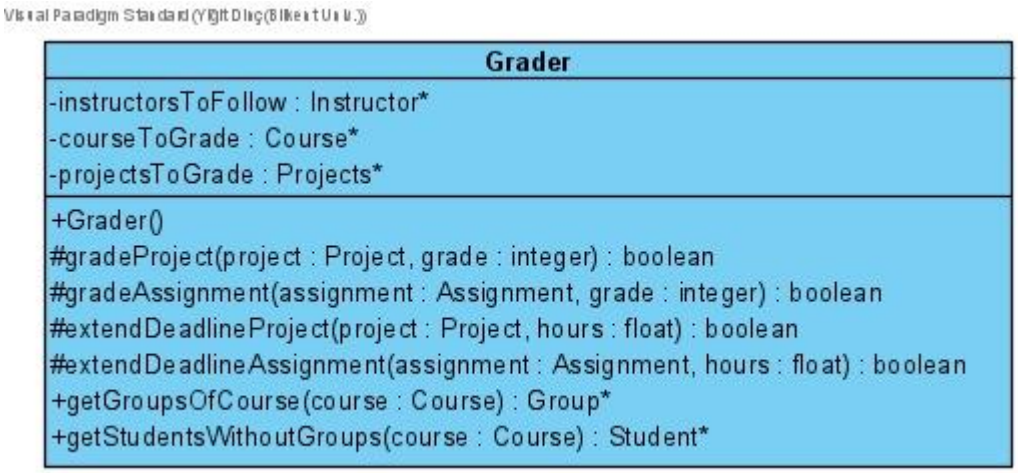
Methods:
Public Instructor (String name, String surName, String email, Picture pic) : This is a constructor where we create an instructor and assign the essential values to it.
Public Grader* getGraders () : Returns a list of all graders that were assigned by the instructor.

Public boolean addGrader (Grader grader, Course course) : In this method we assign a grader to an instructor for reasons explained in the graders part, returns whether succeeded or not in a boolean.
Public boolean removeGrader(Grader grader, Course course) : In this method we remove a grader from the instructor she was assigned to and from the course, then return whether the process was successful or not as a boolean.
Public void setProject (Course course, Project project) : Assigns a project to a course.
Public boolean gradeProject (Project project, int grade) : Sends a grade to be assigned in a project instance and then the boolean return variable is used to return whether it was a success to assign the grade or not.
P**ublic boolean gradeAssignment (Assignment assignment, int grade) :** Here the instructor class sends a request to grade an assignment, then if it was completed the return variable is true, otherwise it is false.
Public boolean extendDeadlineProject (Project project, float hours) : Here the instructor extends the deadline for the project, then returns a boolean to confirm that the process was completed.
Public boolean extendDeadlineAssignement (Assignment assignment, float hours) : Here the instructor extends the deadline for the assignment, then returns a boolean to confirm that the process was completed.
Public boolean removeStudentFromCourse () : Here the instructor can remove a student from a course, the boolean represents the success of the process.
Public boolean addStudentToCourse () : Here the instructor can add a student to a course, if the authority was given to the instructors to do so, the boolean represents the success of the process.
Public Course initializeNewCourse (String name, String code) : Here we set the name and the code for a course once it is created and then return it.
Public boolean removeCourse (Course course) : Here the instructor's class sends a request to remove a course from the system, then if successful returns a confirmation as a boolean.
Public boolean assignStudentToGroup (Student student, Group group) : Here the instructor assigns a student to a group, this option depends on the course, could be rejected if chosen not to be available or if the student is already in a group or if there was a problem. Accordingly the system returns a boolean variable to confirm the success of the process.
Public boolean assignGraderToGroup (Grader grader, Group group) : Here the instructor assigns a grader to a group then a boolean is returned to inform the system regarding success or failure of the process.
Public Group* getGroupsOfCourse (Course course) : returns a list of groups in a course.
Public Student* getStudentsWithoutGroups (Course course) : This returns a list of students without a group in a course, this can be used to know how to assign the remaining students to groups.
Public boolean randomlyAssignStudentsToGroups (Course course, int group_size) : Here we have the option to assign all students to random groups of group_size size, then return boolean to show the case of the success of the process.
Public boolean createGroupsFromOutsiders (Course course, int group_size) : Here we have the option to assign all remaining students to random groups of group_size size, then return boolean to show the case of the success of the process.
Public Courses* getCourses (): Returns a list of courses that an instructor is a member of.

Private boolean addCoursesGiven (Course* courses) : add courses passed as a list in the parameter and then return whether that was done or not

Private boolean removeCoursesGiven (Course* courses): removes the courses passed as a list in the parameter and then returns whether that was done or not.

4.1.5 Grader



This class represents a grader, or a teaching assistant, it implements the user class. Here we have the important info about a grader. And saves the function that a grader can do.

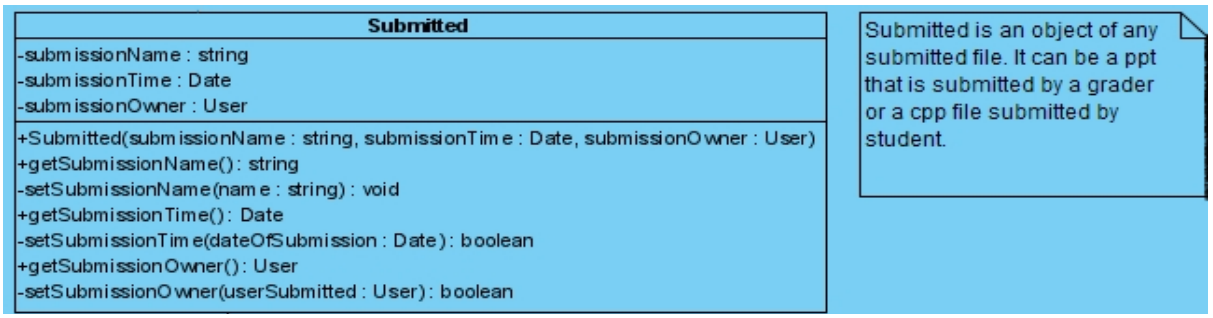
Attributes:
Private Instructor* instructorToFollow: here have the list of instructors that a grader is added to. The reason for adding this attribute is similar to the reason graders are added to the instructor class, to keep track of things and to limit authority based on the authority of the instructor.
Private Course* coursesToGrade: Here we have the list of all grades that a grader can grade.
Private Project* projectsToGrade: This attribute represents the projects that a grader can access and grade.

Methods:
Public Grader (String name, String surName, String email, Picture pic): A constructor that creates the instance of a grader and assigns its essential attributes.
Protected boolean gradeProject (Project project, int grade): Here a grader grades a project using the parameter passed and then the function returns a boolean to confirm the status of process; success or failure for some reasons.
Protected boolean gradeAssignment (Assignment assignment, int grade): Here a grader grades an assignment using the parameter passed and then the function returns a boolean to confirm the status of process; success or failure for some reasons.
Protected boolean extendDeadlineProject (Project project, float hours) : Here a request to the system to extend a deadline of project, then return true in case of success, or false if the user does not have authority or if it is not allowed.

Protected boolean extendDeadlineAssignment (Assignment assignment, float hours) : Here a request to the system to extend a deadline of an assignment, then return true in case of success, or false if the user does not have authority or if it is not allowed.
Public Group* getGroupsOfCourse (Course course): Returns the list of groups in a specific project.
Public Student* getStudentsWithoutGroups (Course course): Returns the list of students who are not in a group yet, this can be used to assign them later to groups.

4.1.6 Submitted Class

Lower part of the model above, is a realization of the Graded interface.

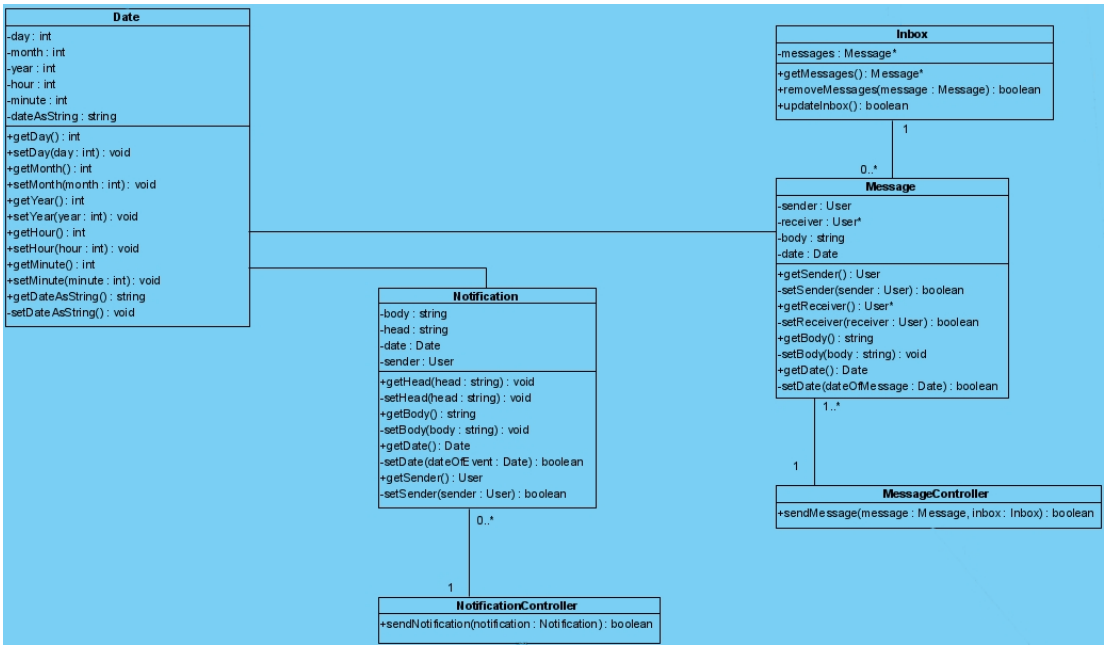


This class represents a submitted assignment, it has the necessary features of such an object like time and submitter (owner).

Attributes:
Private String submissionName: Here we have the name of the submission.
Private String submissionOwner: Here we save the name of the submitter user.
Private Date submissionTime: Here we save the time of submission.

Methods:
Public Submitted (String name, Date time, String owner) : A constructor that initializes the submitted object with the essential info.
Public String getSubmissionName () : Returns the name of the submission.
Public Date getSubmissionTime () : Returns the time of the submission.
Public String getSubmissionOwner () : Returns the name of the owner of the submission.
Private void setSubmissionName (String name) : Sets the name of the submission.
Private void setSubmissionTime (Date time) : Sets the time of the submission.
Private void setSubmissionOwner (String owner) : Sets the name of the owner of the submission.

Date relationship with notification and message. Better resolution version is provided at the given link



4.1.7 Date Class:

This class allows keeping the date in notifications and messages.

Attributes:
Private int day: The day of a message or a notification.
Private int month: The month of a message or a notification.
Private int year: The year of a message or a notification.
Private int hour: The hour of a message or a notification.
Private int minute: The minute of a message or a notification.
Private String dateAsString: The date of a message or a notification.

Methods:
Public int getDay(): Returns the day of a message or a notification.
Public void setDay(int day): Sets the day of a message or a notification.

Public int getMonth(): Returns the month of a message or a notification.
Public void setMonth(int day): Sets the day of a message or a notification.
Public int getYear(): Returns the day of a message or a notification.
Public void setYear(int day): Sets the year of a message or a notification.
Public int getHour(): Returns the day of a message or a notification.
Public void setHour(int day): Sets the hour of a message or a notification.
Public int getMinute(): Returns the minute of a message or a notification.
Public void setMinute(int day): Sets the day of a message or a notification.
Private void setDateAsString(String date): Sets the date of a message or a notification as a string.

4.1.8 Notification Class

This class represents a notification and includes the information about the notification.

Attributes:
Private String body: This attribute stores the body of the notification.
Private String head: This attribute stores the head of the notification.
Private Date date: This attribute stores the date of the notification.
Private User sender: This attribute stores the sender of the notification.

Methods:
Public String getHead(): Returns the head of the notification.
Private void setHead(String head): Sets the head of the notification.
Public String getBody(): Returns the body of the notification.
Private void setBody(String body): Sets the body of the notification.
Public Date getDate(): Returns the date of the notification.
Private void setDate(Date dateOfEvent): Sets the date of the notification.
Public User getSender(): Returns the sender of the notification.
Private void setSender(User sender): Sets the sender of the notification.

4.1.9 Message Class

This class defines the message which is sent by users. It holds the sender and the receiver of the message.

Attribute:
Private User sender: This attribute stores the sender of the message.
Private User* receiver: This attribute stores the receiver of the message.
Private String body: This attribute stores the content (body) of the message.
Private Date date: This attribute stores when the message is sent.

Methods:
Public User getSender(): Returns the sender of the message.
Private boolean setSender(User sender): Sets the sender of the message.
Public User* getReceiver(): Returns the receiver of the message
Private boolean setReceiver(User receiver): Sets the receiver of the message
Public String getBody(): Returns the body of the message.
Private void setBody(String body): Sets the body of the message.
Public Date getDate(): Returns the date of the message.
Private boolean setDate(Date dateOfMessage): Sets the sender of the message.

4.1.10 Inbox Class

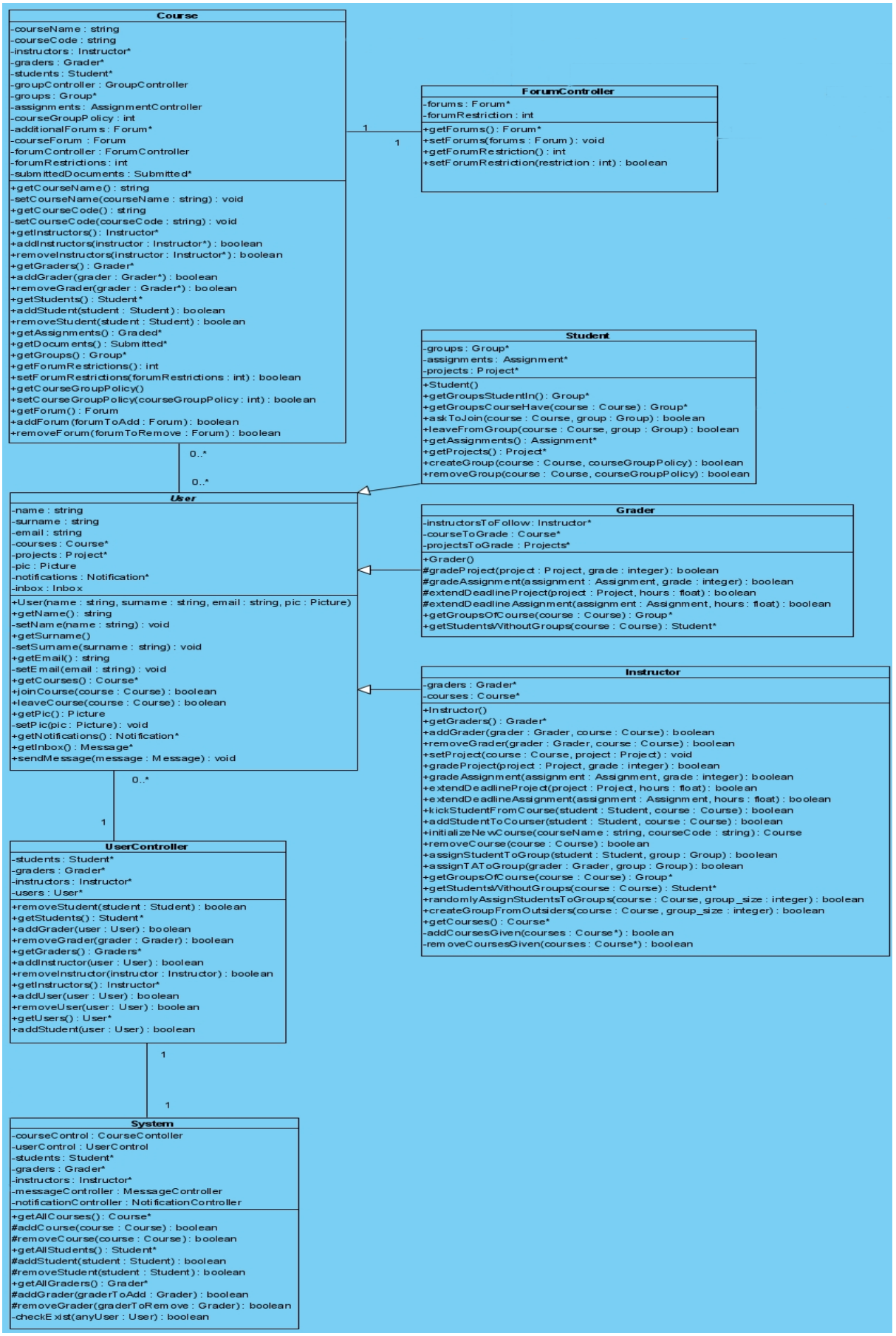
This class manages the messages of users.

Attributes:
Private Message* messages: Contains the link to all the messages of the user.

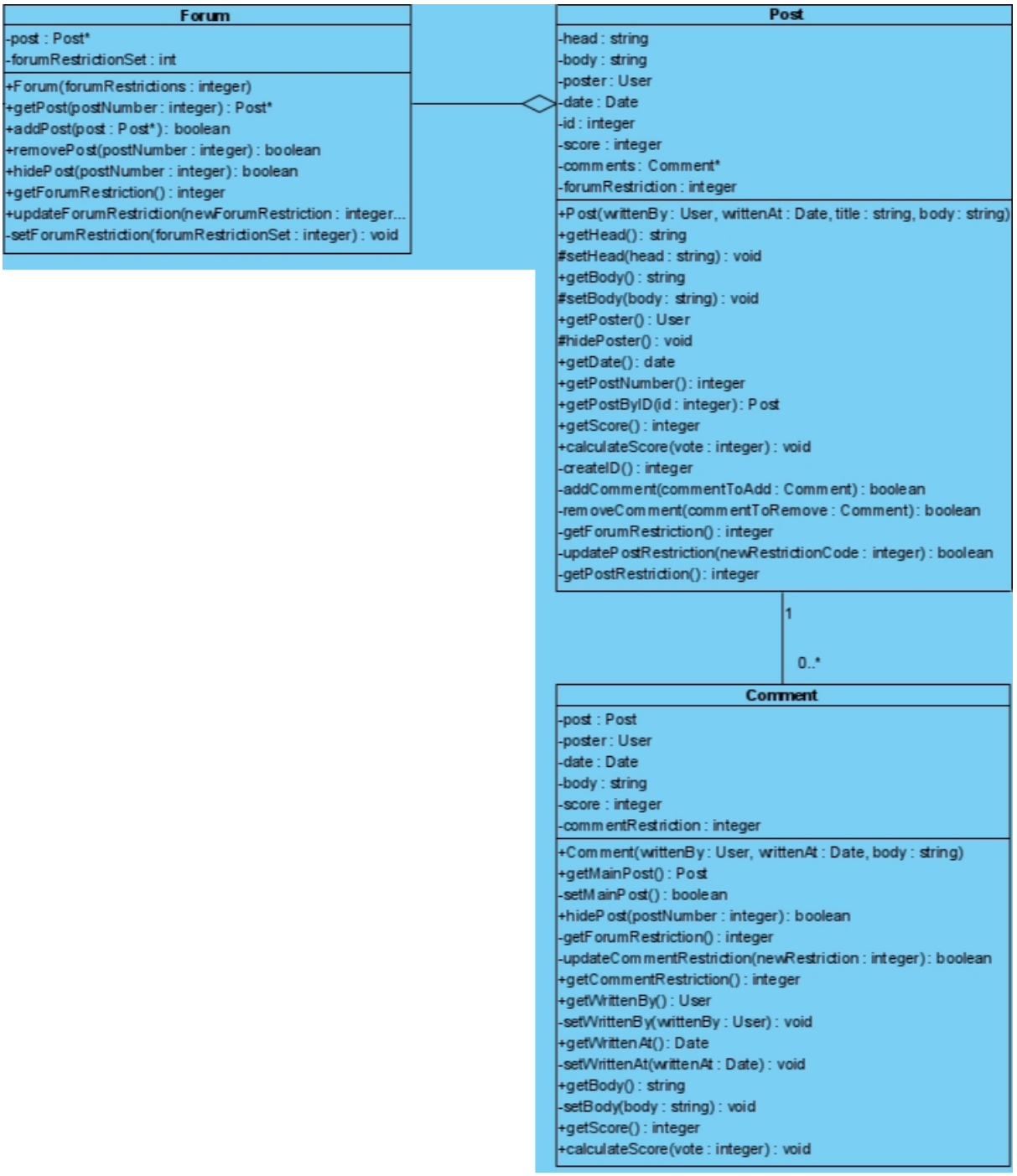
Methods:
Public Message* getMessages(): Returns the message pointer to the user's messages.

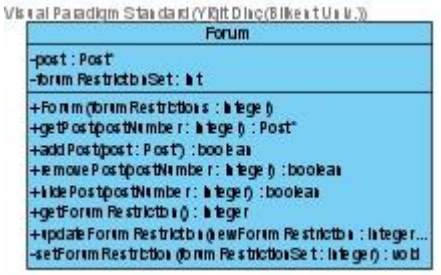
Public boolean removeMessages(): Removes the current message and returns true if successful, false otherwise.
Public boolean updateInbox(): When a user signs in it automatically updates the inbox. That is every mail system has some sort of built-in function to get the messages that are sent newly. Normally they just stay at the database and when the update function is called the new sent mails are retrieved from the database.

Course-forum-user model with User Student Grader Instructor realization.



Forum Post Comment Aggregation





4.1.11 Forum Class

This class defines the forum of the application. It has the post attribute as the main attribute.

Attributes:
Private Post* post: This is the main attribute of the class and allows users to post on the forum.
Private int forumRestrictionSet: This attribute allows instructors to restrict actions of the other users.

Methods:
Public forum (int forumRestriction): This method creates the forum with the restrictions by the instructor.
Public Post* getPost(int postNumber): This method returns the desired post.
Public boolean addPost(Post* post): Adds a new post on the forum.
Public boolean removePost(int postNumber): Removes the desired post from the forum.
Public boolean hidePost(int postNumber): Hides the desired post on the forum.
Public int getForumRestriction(): Returns if the forum is restricted or not.
Public int void updateForumRestriction(int newForumRestriction): Allows updating the forum restriction.
Private void setForumRestriction(int forumRestrictionSet): Allows setting the restriction of the forum at the beginning.

4.1.12 Post Class

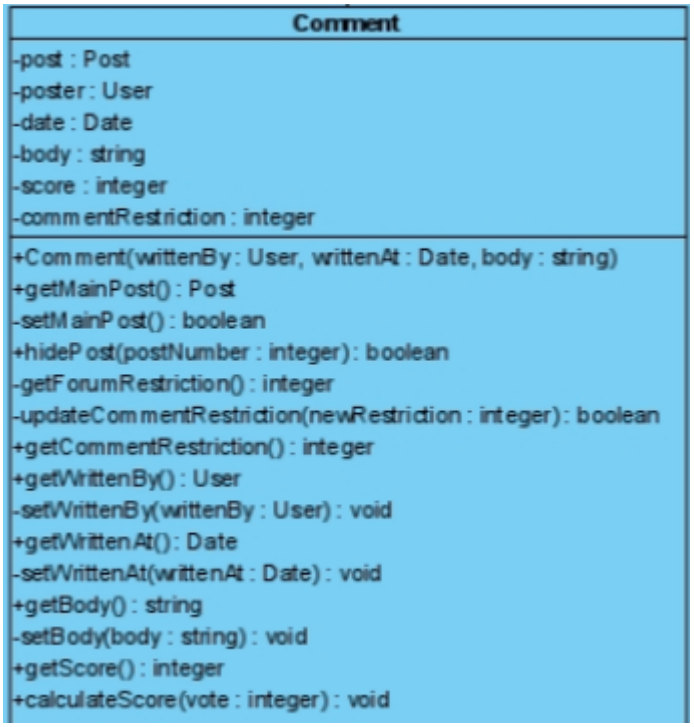
This class defines the content of the posts. Each post has their head as the title and the body which is the text.

Attributes:
Private String head: Head or title of the post
Private String body: Actual content of the forum post which is a text.
Private String poster: The attribute that holds the sender of the specific post.
Private Date date: The date defines when the forum post is posted.
Private int id: Specific id of the forum post.
Private int score: In this application users can rate the posts by upvote or downvote. Score holds votes for the post.
Private Comment* comments: This attribute stores the comments of the post if there are any.
Private int forumRestriction: Stores the restriction of the post if there are any.

Methods:
Public void post(User writtenBy, Date writtenAt, String title, String body): This method allows user to post anything on the forum and gets the user who writes the post, date of the post, title (head) of the post and actual content (body) of the post.
Public String getHead(): Returns the title of the post.
Protected void setHead(String head): Allows users to set the title of the post.
Public String getBody(): Returns the actual content (body) of the post.
Protected void setBody(String body): Allows users to write the body of the post.
Public User getPoster(): Returns the user of the poster.
Protected void hidePost(): Allows the posters to hide their posts.
Public Date getDate(): Returns the date of the post.
Public int getPostNumber(): Returns the post's number.

Public Post getPostById(int id): Allows to search the specific post by its id.
Public int getScore(): Returns the score of the post (upvotes or downvotes).
Public void calculateScore(int vote): Calculates the score of the post (1 is upvote and -1 is downvote).
Private int createID(): Creates an id for the post.
Private boolean addComment(Comment commentToAdd): Allows users to add comments for the specific post.
Private boolean removeComment(): Allows users to remove their comments on a post.
Private boolean updatePostRestriction(int newRestrictionCode): Allows users to update the restriction on the post.
Private int getPostRestriction(): Returns the restriction on the post if there are any.

4.1.13 Comment Class



This class defines the comments on posts

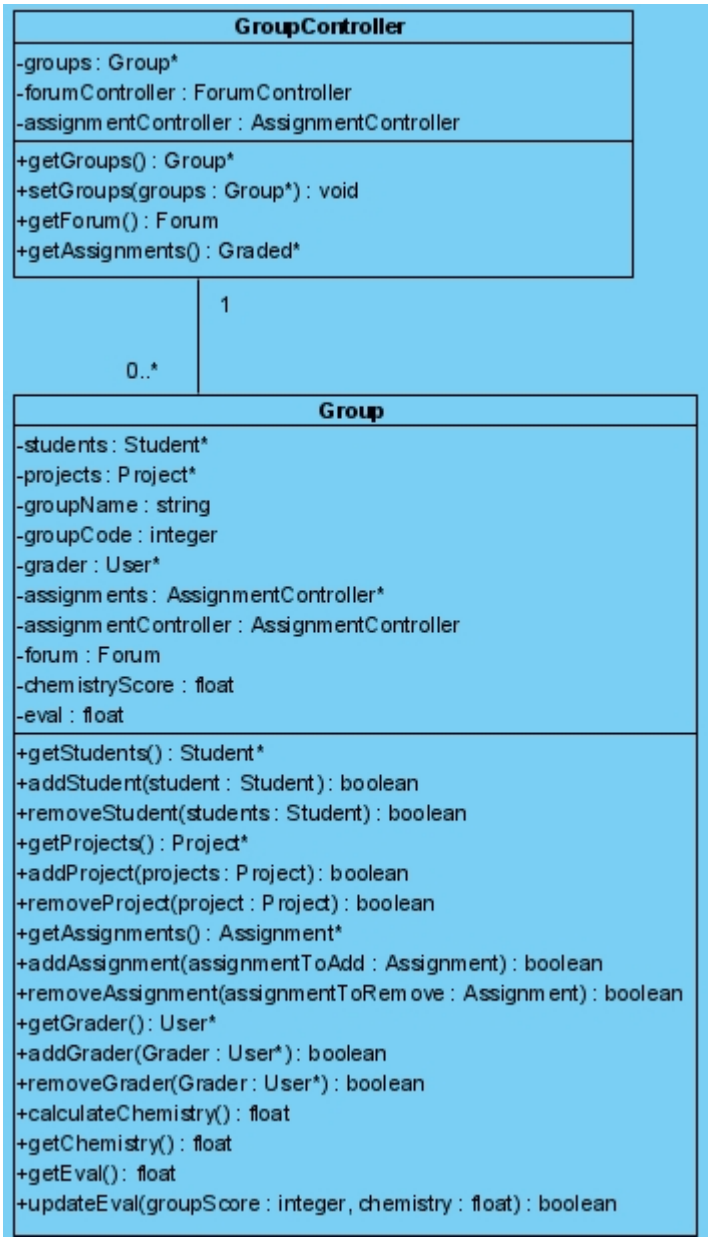
Attributes:
Private Post: The post which will be commented upon.
Private User Poster: The user who creates the post.
Private Date: The date of the creation of the post.
Private String Body: The actual content of the comment
Private int Score: Score is the comment's score. Like in Reddit, it is the total number of upvotes-downvotes.
Private integer commentRestriction: Comment restriction is again set by the Instructor when setting up the course. It is a number that works like a code, that is used to let students comment or not. For example, if the restriction is set at 0 it means students are not given any chance to comment on anything, if it is set at 1 then this may indicate they can only

comment on anything that is written by a Student, if it is 2 than they can comment on anything that is written by any Student or Grader.

Methods:
Public void Comment(user writtenBy, date writtenAt, string body): The constructor for the comment which takes the user (poster), the current date and the content of the comment and then creates it.
Public post getMainPost(): Returns the post which is commented upon
Public boolean hidePost(integer postNumber): Hides the associated post and returns true if successful, false otherwise
Public int getCommentRestriction(): Returns the maximum number of available comments for post
Public User getWrittenBy(): Returns the user who wrote the comment
Public Date getWrittenAt(): Returns the date when the comment was written
Public String getBody(): Returns the content of the comment
Public int getScore(): Returns the current score of the comment
Public void calculateScore(integer vote): Calculates the score of the comment based on the votes.

Private boolean setMainPost(): Links the comment with the main post and returns true if successful, false otherwise
Private int getForumRestriction(): returns the restriction number for the post.
Private boolean updateCommentRestriction(integer newRestriction): Updates the restriction number for the comments and returns true if successful, false otherwise.
Private void setWrittenBy(User writtenBy): Sets the user as the author of the comment.
Private void setWrittenAt(Date writtenAt): Sets the date as the date of creation of the comment.
Private void setBody(String body): Sets the string as the content of the comment.

4.1.14 Group Class



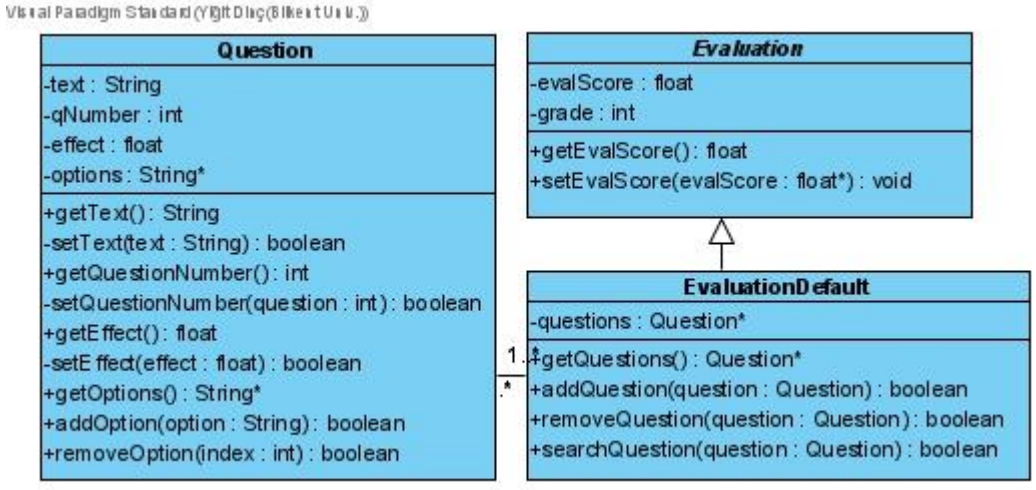
This class represents the group of a course.

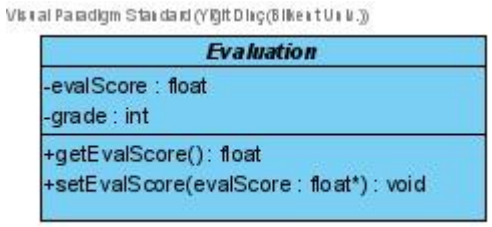
Attributes:
Private Student* students: Represents the array of students in a group.
Private Project* projects: Represents the array of projects a group has.
Private String groupName: Represents the name of the group.
Private int groupCode: Represents the group code.
Private User* grader: Represents the grader of the group.
Private AssignmentController* assignments: Assignments are called by using the AssignmentController's methods to provide a secure system. That is, only touch between system and assignments is connected through AssignmentController objects.
Private AssignmentController* assignment Controller: Assignment controller is the controller in the system for the assignments. It is hoped that it will control all the processes

about any assignment in the system. So by making a place for an AssignmentController we can be sure about the assignment created correctly. That is, if an assignment created mistakenly that refers to an assignment is added to the database directly without any control.
Private Forum forum: Represents the forum of a group.
Private float chemistryScore: Represents the chemistry score of a group.
Private float eval: Represents the evaluation of a group.

Methods:
Public Student* getStudents(): Returns all students in a group.
Public boolean addStudent(Student student): Adds a new student to a group.
Public boolean removeStudent(Student student): Removes a student from a group.
Public Assignment* getAssignments(): Returns all assignments that a group has.
Public boolean addAssignment(Assignment assignmentToAdd): Adds assignment to group's assignment list
Public boolean removeAssignment(Assignment assignmentToRemove): Removes an assignment from a group's assignments list.
Public User* getGrader(): Returns the grader of a group.
Public boolean addGrader(User* Grader): Adds a grader to a group.
Public boolean removeGrader(User* Grader): Removes a grader from the group.
Public float calculateChemistry(): Calculates the chemistry of a group.
Public float getChemistry(): Returns the chemistry of a group.
Public float getEval(): Returns the evaluation of a group.
Public boolean updateEval(int groupScore, float chemistry): Updates the evaluation of a group.

Question EvaluationDefault diagram with Evaluation Generalization

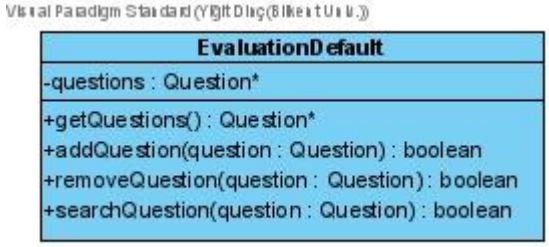




4.2.15 Evaluation

Attributes
Private float evalScore: It is the score calculated for the evaluation.
Private int grade: It is the grade of the calculated evalScore.

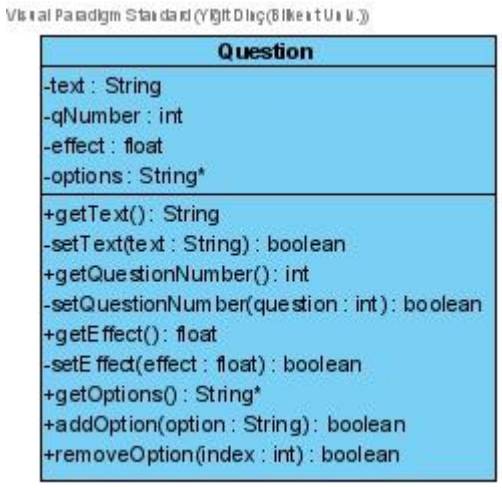
Methods
Public float getEvalScore() : It is the function that returns the evalScore.
Public void setEvalScore(float* evalScore) : It takes a float type parameter to process. For the given evalScore parameters it calculates the evaluation score.



4.1.16 EvaluationDefault

Attributes
Private Question* questions: It is a list of questions that are given to the person who takes the evaluation form.

Methods
Public Question* getQuestions() : It is a method to get the questions in the evaluation form.
Public boolean addQuestion(Question question) : It takes a parameter, a question, and adds it to the evaluation form. If the process is correct then returns true, else false.
Public boolean removeQuestion(Question question) : It takes a parameter, a question, and removes it from the evaluation form. If the process is correct returns true, else false.
Public Question searchQuestion(Question question) : It takes a question parameter, searches it in the Evaluation form if it exists in the form returns true, else false.



4.1.17 Question

Attributes
Private String text: It is the question body.
Private int qNumber: It is the question number.
Private float effect: It is how much this question affects the overall evaluation form score.
Private String* options: It is the options for that specific question.

Methods
Public String getText() : returns the question body.
Private boolean setText(String text) : sets the question body to the given text.
Public int getQuestionNumber() : returns the question number.
Private boolean setQuestionNumber(int number) : sets the question number.
Public float getEffect() : returns how much this question affects the grade.
Private boolean setEffect(float effect) : sets the question effect to the given float. If the process is correct returns true, else false.
Public String* getOptions() : returns the options of the question.
Public boolean addOption(String option) : adds given text to the option field. If the process is correct, returns true else false.
Public boolean removeOption(int index) : removes the option at given index. If the process is correct returns true, else false.

4.2 PACKAGES

4.2.1 NodaTime

NodaTime is a date and time API acting as an alternative to the built-in DateTime/DateTimeOffset etc. types in .NET.

4.2.2 NodaTime.Testing

Provides extra types which may be useful when testing code which uses Noda Time, such as a fake programmable implementation of IClock and a time zone which has a fixed transition. These types are also used to test Noda Time itself.

4.2.3 NodaTime.Serialization.JsonNet

Provides serialization support between Noda Time and Json.NET.

4.2.4 System.Data.SqlClient

Provides the data provider for SQL Server. These classes provide access to versions of SQL Server and encapsulate database-specific protocols, including tabular data stream.

5. Glossary and References

1. https://www.nuget.org/packages/NodaTime
2. https://www.nuget.org/packages/NodaTime.Testing
3. https://www.nuget.org/packages/NodaTime.Serialization.JsonNet
4. https://www.nuget.org/packages/Microsoft.Data.SqlClient/