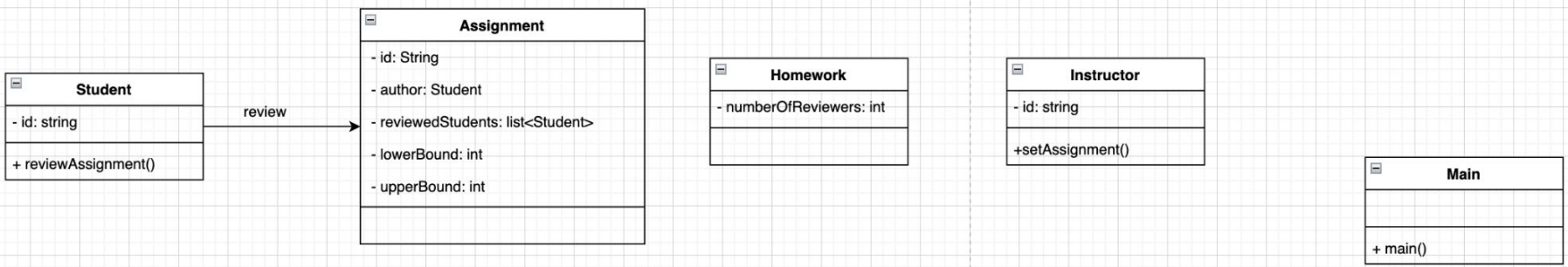
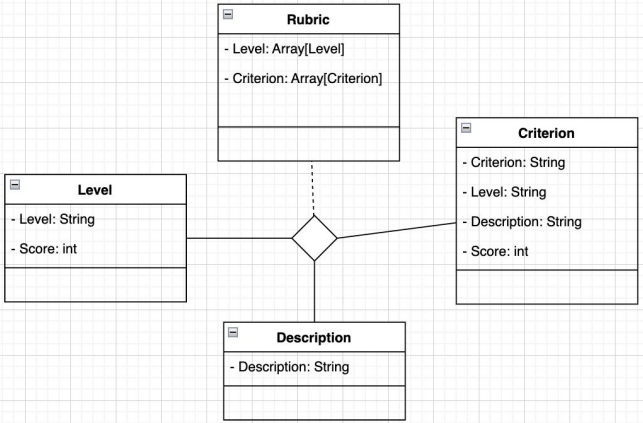
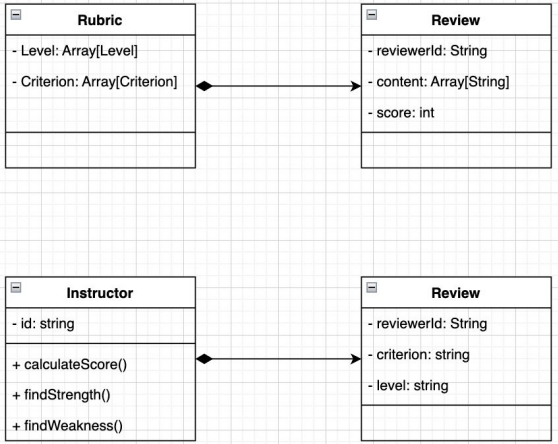
- We are developing a peer review system. Students can review and rank other students' homework. In general, each assignment is reviewed by 3-5 students but can be set by the instructor.



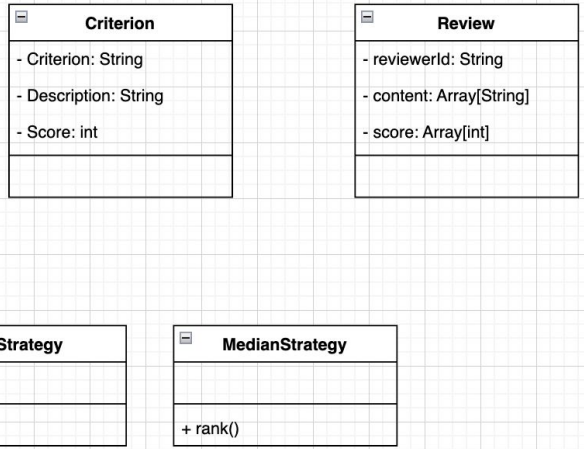
- To be more objectively, instructors are required to design a ranking criterion called rubric for each assignment. A rubric includes one or more dimensions on which performance is rated, definitions that illustrate the attribute being measured, and a rating scale for each dimension. Dimensions are generally referred to as criteria, the rating scale as levels, and definitions as descriptors. The following is a rubric example to evaluate the essay of describing the changes in one Portland community over the past 30 years.



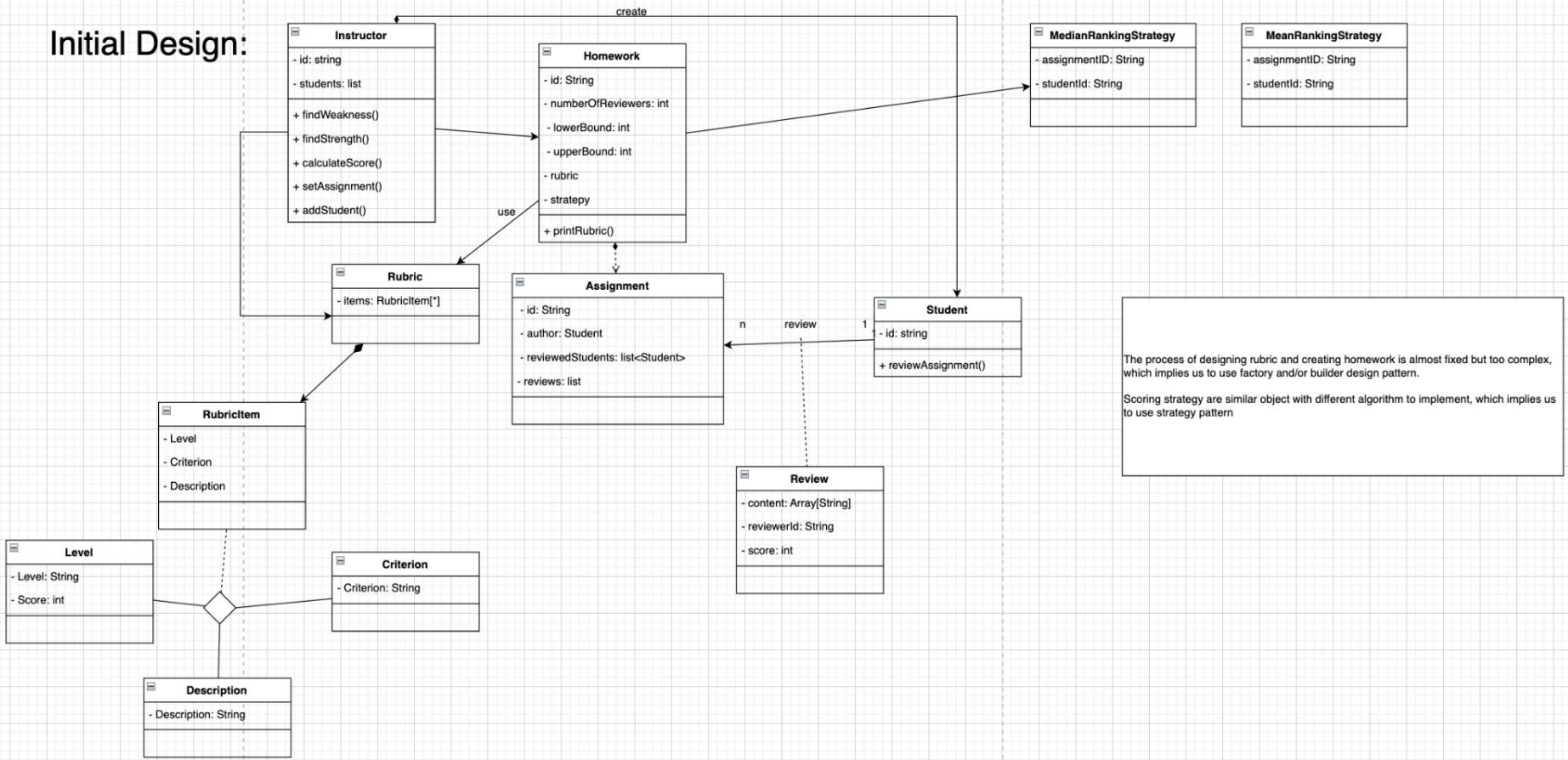
- After the peer review, the instructor can see the score of each student, and the average in each criterion. The score information can help the instructor understand the strength and weakness of each student.



- Other requirements:
  - Each rubric has three levels: *excellent*, *competent* and *needs work*. The scores are 3, 2 and 1, respectively. However, the level may change in the future since different schools may have different strategies.
  - A student's score is ranked by averaging the scores from their reviewers. But the rule may be changed in the future - some instructors think using median is more reasonable.



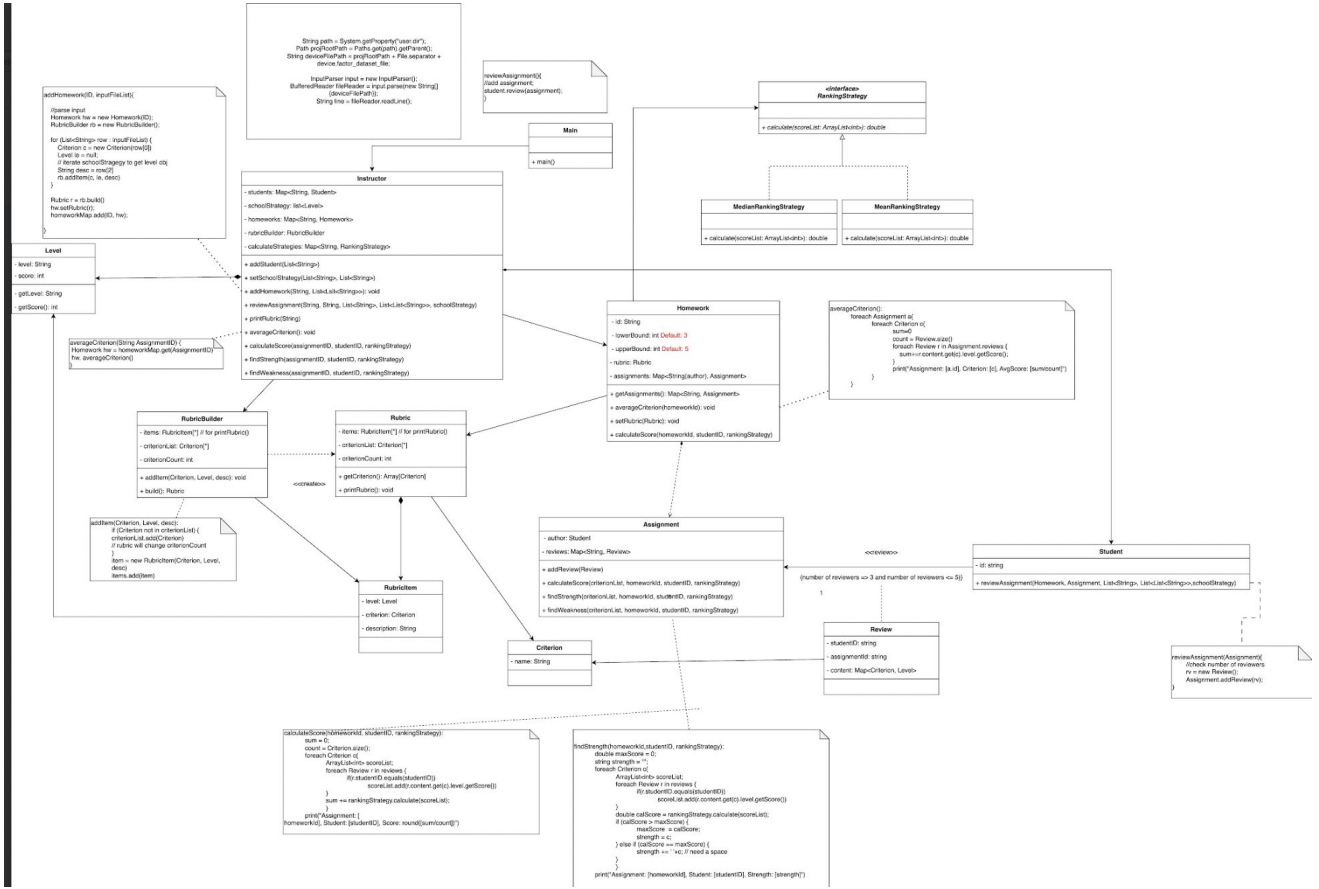
Initial Design:



The process of designing rubric and creating homework is almost fixed but too complex, which implies us to use factory and/or builder design pattern.

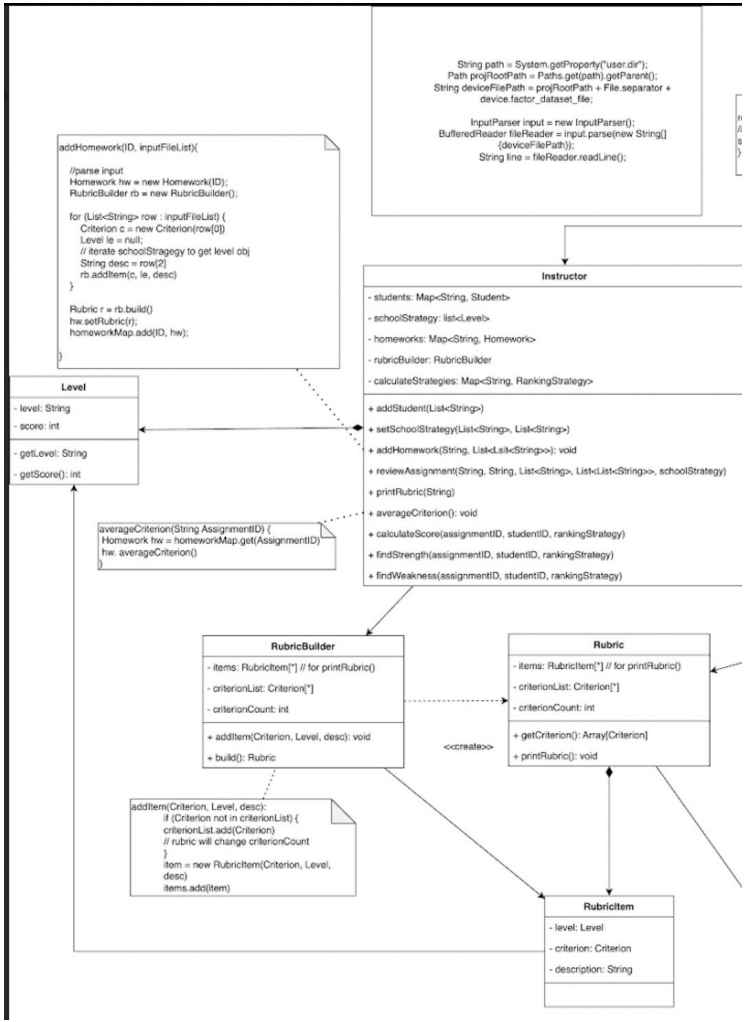
Scoring strategy are similar object with different algorithm to implement, which implies us to use strategy pattern

Refactor (overview)

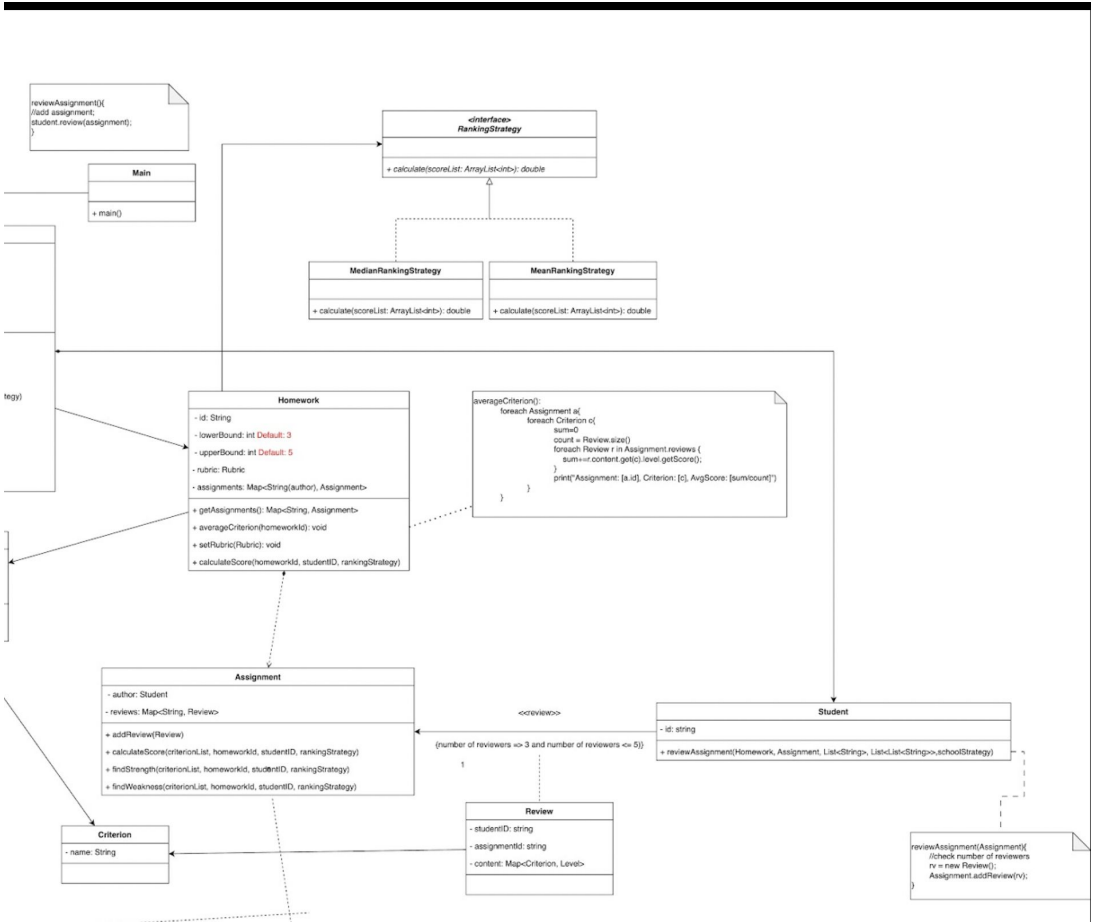


Sorry Professor and TAs, because our diagram is too big, when we export it, it will be not tha clear, so we decompose it to some parts to show them more clearly, with this overview in advance.

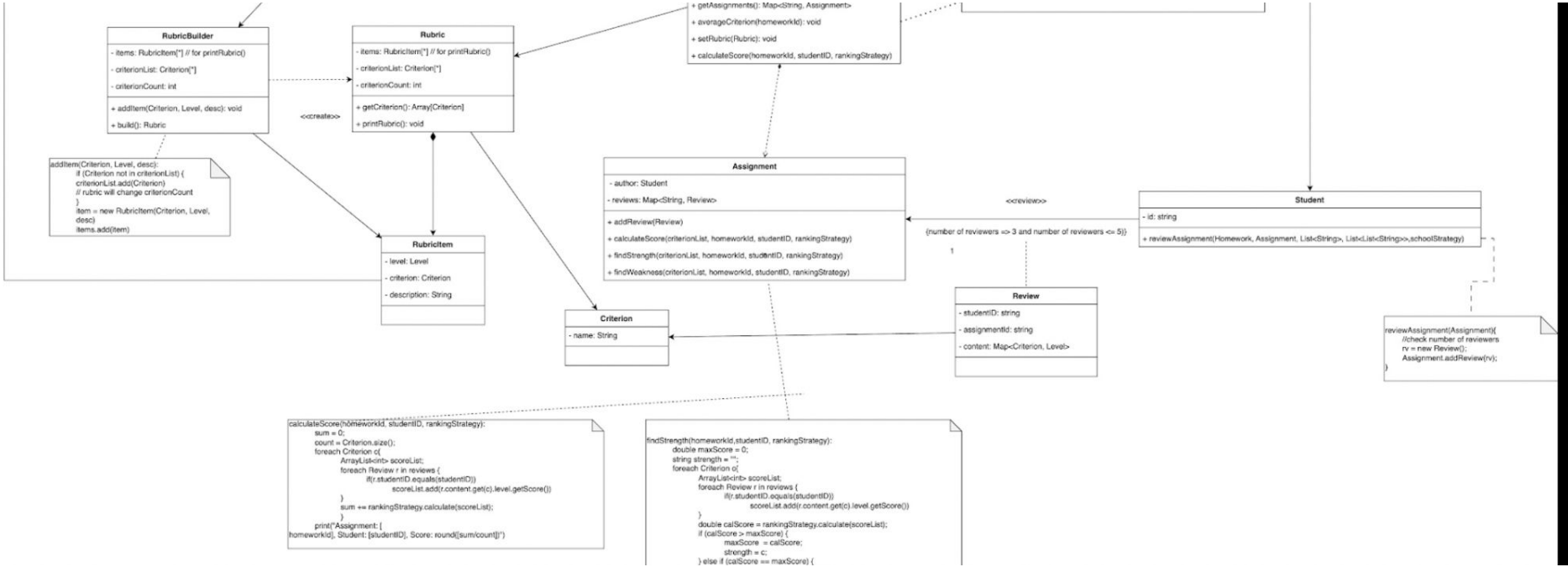
Upper left part



Upper right part



Lower part


RubricBuilder
- items: RubricItem[*] // for printRubric()
- criterionList: Criterion[*]
- criterionCount: int
+ addItem(Criterion, Level, desc): void
+ build(): Rubric
Rubric
- items: RubricItem[*] // for printRubric()
- criterionList: Criterion[*]
- criterionCount: int
+ getCriterion(): Array[Criterion]
+ printRubric(): void
<<create>>
+ getAssignments(): Map<String, Assignment>
+ averageCriterion(homeworkId): void
+ setRubric(Rubric): void
+ calculateScore(homeworkId, studentID, rankingStrategy)
addItem(Criterion, Level, desc):
if (Criterion not in criterionList) {
criterionList.add(Criterion)
// rubric will change criterionCount
}
item = new RubricItem(Criterion, Level, desc)
items.add(item)
RubricItem
- level: Level
- criterion: Criterion
- description: String
Assignment
- author: Student
- reviews: Map<String, Review>
+ addReview(Review)
+ calculateScore(criterionList, homeworkId, studentID, rankingStrategy)
+ findStrength(criterionList, homeworkId, studentID, rankingStrategy)
+ findWeakness(criterionList, homeworkId, studentID, rankingStrategy)
<<review>>
{number of reviewers => 3 and number of reviewers <= 5}
1
Student
- id: string
+ reviewAssignment(Homework, Assignment, List<String>, List<List<String>>,schoolStrategy)
Criterion
- name: String
Review
- studentID: string
- assignmentId: string
- content: Map<Criterion, Level>
reviewAssignment(Assignment){
//check number of reviewers
rv = new Review();
Assignment.addReview(rv);
}
calculateScore(homeworkId, studentID, rankingStrategy):
sum = 0;
count = Criterion.size();
foreach Criterion c{
ArrayList<int> scoreList;
foreach Review r in reviews {
if(r.studentID.equals(studentID))
scoreList.add(r.content.get(c).level.getScore())
}
sum += rankingStrategy.calculate(scoreList);
}
print("Assignment: [homeworkId], Student: [studentID], Score: round([sum/count])")
findStrength(homeworkId,studentID, rankingStrategy):
double maxScore = 0;
string strength = "";
foreach Criterion c{
ArrayList<int> scoreList;
foreach Review r in reviews {
if(r.studentID.equals(studentID))
scoreList.add(r.content.get(c).level.getScore())
}
double calScore = rankingStrategy.calculate(scoreList);
if (calScore > maxScore) {
maxScore = calScore;
strength = c;
} else if (calScore == maxScore) {