Object-oriented Modeling & Design

Assignment IV

Buğra Ekuklu, 150120016
Istanbul Technical University

A. Answers to the question (a)

a. Calculating GPA should not be done by `Student`.

From the behavioral aspect, *GPA* and *student* are different kinds. A *student* does not have to have a *GPA*, at all. For instance, a student might be enrolled to a course without a grading system. Hence, the contract cannot be provided by all possible concrete implementations of an abstract *student* implementation containing abstract method(s) for calculating GPA.

The solution for the problem could be implementing a grading calculation strategy, which is contained by a `Student` object. In overall, we prefer **composition** over **inheritance**, since the calculation of GPA should be seperated from the logic of `Student` class' itself.

b. If there is a GPA to be calculated, how to calculate it must be decided during creation of the `Student` object.

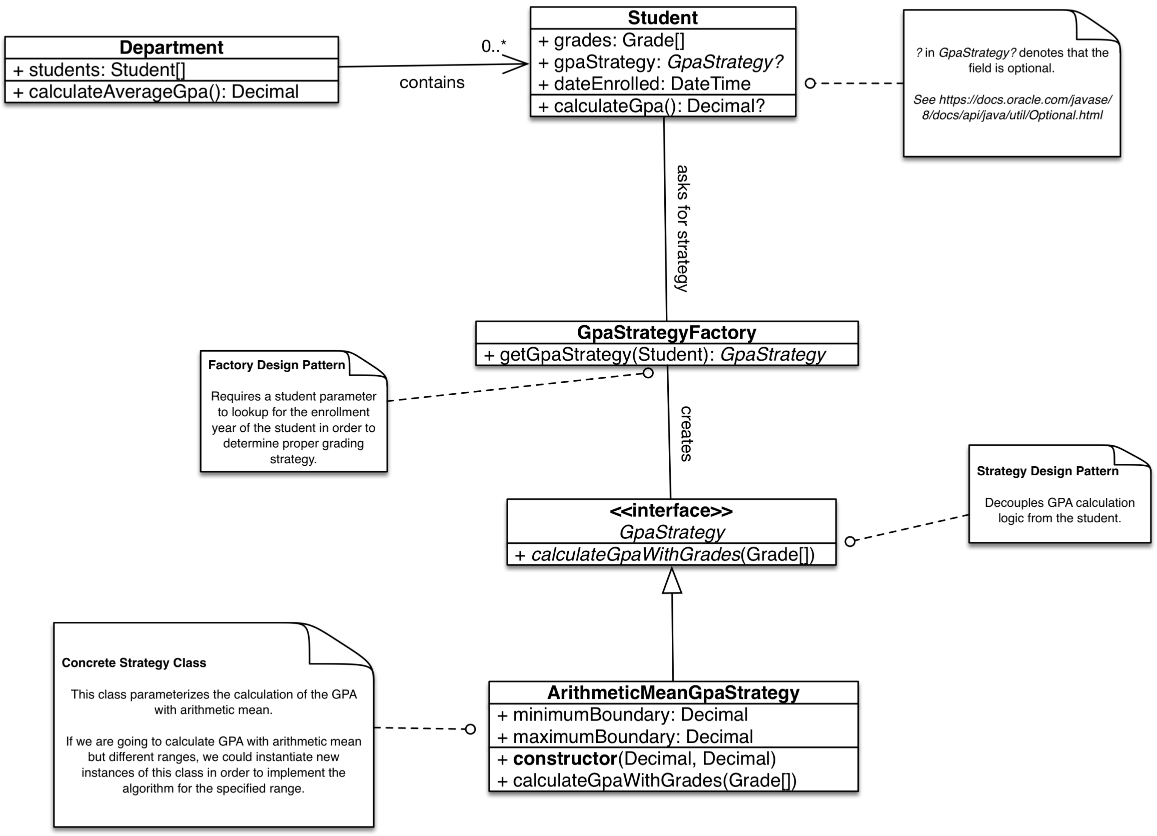
A factory class for creating `GpaStrategy` objects can be implemented. The new factory class can implement a factory method with a year parameter to provide objects to various `Students`. The system is highly cohesive, since the responsibility of creating strategy objects is now at factory class.

c. Instead of implementing new class for each GPA calculation method, we could implement a parametric class which makes a general implementation.

The `Grade` object may contain score and weight properties to supply enough information to the GPA calculator algorithm. Most of the time, GPAs will be calculated with a `Grade` object. One could implement such class, who is responsible for calculating GPA, with parameters of minimum and maximum boundaries of the grading system.

For example, instead of having classes for each GPA calculation method, we could parameterize the function.

B. Suggested UML class diagram


Department
+ students: Student[]
+ calculateAverageGpa(): Decimal
contains
0..*
Student
+ grades: Grade[]
+ gpaStrategy: *GpaStrategy?*
+ dateEnrolled: DateTime
+ calculateGpa(): Decimal?
asks for strategy
GpaStrategyFactory
+ getGpaStrategy(Student): *GpaStrategy*
creates
<<interface>>
GpaStrategy
+ *calculateGpaWithGrades*(Grade[])
ArithmeticMeanGpaStrategy
+ minimumBoundary: Decimal
+ maximumBoundary: Decimal
+ **constructor**(Decimal, Decimal)
+ calculateGpaWithGrades(Grade[])

? in *GpaStrategy?* denotes that the field is optional.

See https://docs.oracle.com/javase/8/docs/api/java/util/Optional.html

Factory Design Pattern

Requires a student parameter to lookup for the enrollment year of the student in order to determine proper grading strategy.

Strategy Design Pattern

Decouples GPA calculation logic from the student.

Concrete Strategy Class

This class parameterizes the calculation of the GPA with arithmetic mean.

If we are going to calculate GPA with arithmetic mean but different ranges, we could instantiate new instances of this class in order to implement the algorithm for the specified range.