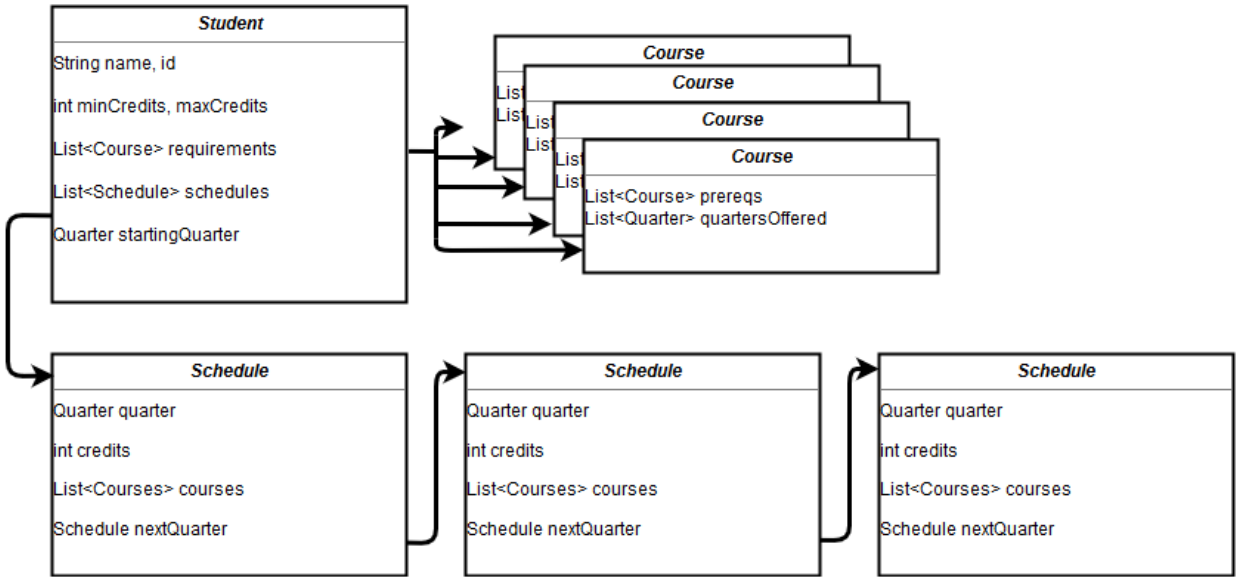
Data Types

A Student object will contain student attributes such as their name, student id, and starting quarter, and minimum and maximum quarterly credits. Their graduation requirements can be assigned to them using a list of Course objects.

Course represents a single graduation requirement, such as “English proficiency” or “Data Structures”. It contains a list of Course objects representing the prerequisite courses that can be used to fulfill that requirement. It can represent a single, individual course, such as “Discrete Math”, or multiple possible courses, such as “CS Elective” or “Breadth Requirement”. For multiple courses the course should be considered offered every quarter.

A generated student schedule will be represented by Schedule objects. Each schedule object will contain the quarter that the schedule represents, the number of credits taken, a list of courses, and a pointer to the schedule for the next quarter.



Algorithm

To make a schedule, the algorithm will have to generate a list of possible courses that the student can take each quarter. To do this, the Student object should take a list of all Course objects within their list of graduation requirements, and return a list of courses that meet the following requirements:

- Each course fulfills a graduation requirement.
- The student meets all prerequisites for each course.
- The course is offered during that quarter.
- The student is not currently taking the course, and has not already taken the course.

Because of the huge number of possibilities of course schedules, we must take as many measures as possible to trim calculations from the possibility tree. Because the algorithm should return a number of possible schedules, instead of just the one best schedule, the Student object should hold a list of Schedules of either a predetermined or user-inputted size, and then further Schedules should be compared against the worst (longest) schedule in the Student's list. For each iteration of the GenerateSchedule algorithm, the following scenarios should be checked before allowing a Schedule to be developed further:

- Check that the current schedule is not already longer than the longest schedule in the list.
- Check that the current schedule cannot be completed in less time than the longest schedule in the list. This can be accomplished by calculating:
  Credits remaining / Maximum credits per quarter = Minimum quarters remaining.

Pseudocode

```
/*
Example pseudocode of the schedule generating algorithm
Given a student and a blank schedule object, it recursively generates permutations of course schedules.
*/


GenerateSchedule( Student, current Schedule )
        if student has no remaining graduation requirements
                add current schedule to the list of the student's schedules
                return
        else if this schedule is worse than the schedules already generated
                return
        else
                make list of all possible courses for current quarter
                if list is empty
                        GenerateSchedule( Student, next Quarter )
                else
                        for each course C in possible courses
                                if course fits in students current schedule
                                        add to current schedules
                                        GenerateSchedule( Student, current Schedule )
                                else
                                        GenerateSchedule( Student, next Schedule )
        return
```

Example scenario tree

Given scenario: Student can take max 10 credits per quarter
Courses / GradReqs: [CS101] [CS110] [ENG101 – Only fall qtr] [ENG102 – Requires ENG101]
Legend: →denotes current quarter, P denotes the courses that the student is able to take that quarter