TEAM 1: 1-Direction

1. *Patrick Mc Grath*
2. *Joshua Nwabuzor*
3. *Tyler Pham*
4. *Daniel Belonio*
5. *Antoine Merino*

Entity Methods & Attributes

https://github.com/StormHooper/1-Direction

1. **Course**
    a. Attributes
        i. *Capacity* (*int)*
            1. Stores the maximum amount of spots within the course
        ii. *Students* (*list[Student])*
            1. Stores a list of current students(student objects)
        iii. *Waitlist* (*list[Student])*
            1. Store a list of students(student objects) who wish to take the course
        iv. *Course_ID (int)*
            1. Stores the course's ID number
        v. *Criteria (list[Criteria[List]])*
            1. Stores a list of criteria a course may fit
    b. Methods
        i. *notify() -> None*
            1. For student (Student object) in Waitlist, update the student's "Notifications" list and for all students in
        ii. *Description()(args -> output type*
            1. Prints out the provided description of the course object.
        iii. *is_full() -> bool*
            1. If the amount of students in the class is greater than or equal to the capacity of the class, return true
        iv. *add_student(student: Student) -> None*
            1. If student not already in students attribute, appends student(Student Object), to course.students attribute
        v. *remove_student(student: Student) -> None*
            1. Removes a student(Student Object), to course.students attribute.
        vi. *add_wait(student: Student) -> None*

1. If a student is not already in the waitlist and not in class, add student to waitlist attribute.

vii. *remove_wait(student: Student) -> Student*

1. Removes a Student(student Object) from the course.waitlist attribute.
2. If not class.is_full(), may be implemented such that course.add_student(remove_wait(waitlist[0]))

2. Student

a. Attributes

i. *student_name* (*type : str)*

1. Stores student name

ii. *courses* (*type: List[String])*

1. Stores list of courses the student is or has taken.

iii. *notifications* (*List[List[String]])*

1. Store a table of notifications (strings) with timestamps

b. Methods

i. *add_course(course: Course) -> None*

1. If not course.is_full(), appends a course object to courses attribute and does course.add_student(self)
2. If course.is_full(), prompts users whether they would like to enter a waitlist. If true, course.

ii. *remove_course(course: Course) -> None*

1. If course in courses attribute, remove course from courses and course.remove_student(self)

iii. *add_criteria(criteria: Criteria) -> None*

1. criteria.add_student(self)

iv. *remove_criteria(criteria: Criteria) -> None*

1. criteria.remove_student(self)

3. Admin

a. Attributes

i. *admin_name* (*str)*

1. Stores the name of the admin user.

ii. *admin_id* (*int)*

1. Stores the ID of the admin user.

iii. *permissions* (*list)*

1. Stores a list of permissions as bool values.

b. Methods

i. *edit_student_info (student) -> None*

1. Can edit student information such as their major, name, and ID.

ii. *create_course (id, name) -> Course*
1. Can create a course

iii. *delete_course (course) -> output type*
1. Can delete a course which will remove it from the list of courses and remove any students in the course.

4. Interface

a. Attributes

i. *students* (*list[Student])*
1. Stores a list of current students(student objects)

ii. *admins* (*list[Admin])*
1. Stores a list of admins.

iii. *criteria(list[Criteria])*
1. Stores a list of the criteria that can be used to search for classes.

iv. *courses(list[Courses])*
1. Stores of list of currently available courses.

b. Methods

i. get *student_list*()
1. Returns the current students(student objects) within the students attribute.

ii. *get_course_list()*
1. Returns the current courses(course objects) within the course attribute.

iii. *get_student(int)*
1. Returns the student at the associated ID number.

iv. *get_admin(int)*
1. Returns the admin at the associated ID number.

v. *authenticate(role -> str, email -> str, password -> str)*
1. Depending on the role,email and password of the user will authenticate what they can or can not do or see.

vi. *get_user()*
1. Returns a specific user/email depending on the role, and if the user does not exist, will return an error.

vii. *login(email -> str, password -> str, choice -> int)*
1. Logs in the user, and making use of both get_user() and authenticate(), verifies if the user is within the system. And if so gain access to the home page.

5. **Criteria**
    a. Attributes
        i. name (*str*)
            1. Stores the name of the criteria
        ii. students (list[student])
            1. Holds the list of students
    b. Methods
        i. add_student(student)
            1. Adds a student and checks if they are already on the student list of a course
        ii. remove_student(student)
            1. Removes a student from the student list of a course