

CSIT121 Lab

Revision exercises

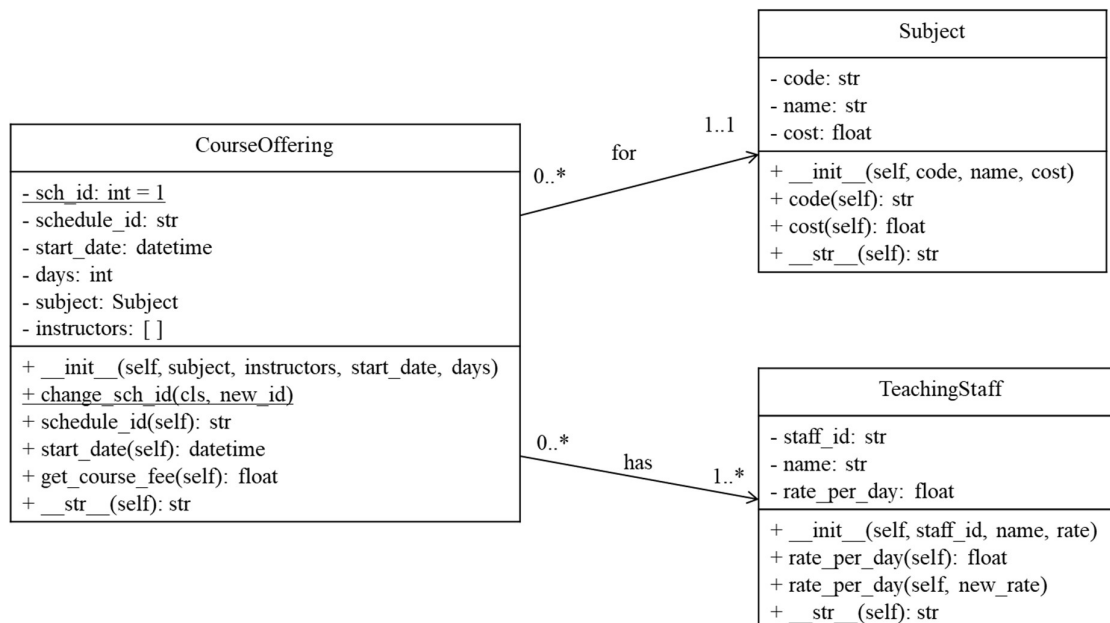
Objectives

- Revision of key OOP concepts.
- Practical applications of key OOP concepts.

Context:

Apart from students, UOW also provides training to the public, including both individuals and corporations. In this practice exercise, you will write python classes to support this business scenario.

1. The following classes model the subjects, teaching staff and course offering by UOW.



The Subject class has attributes code, name, and development cost. The TeachingStaff class models a teaching staff with staff_id, name, and how much the staff charges per day.

The CourseOffering class models the schedule of a course offered by UOW, for a subject, taught by a group of instructors, with a start datetime and the duration in days.

- a) Write the Subject class according to the class diagram given, with the 3 attributes, a constructor, getters (properties) for attribute code and cost. The `__str__()` method returns a string in the following format:

Subject Code: CSIT121

Name: Object Oriented Design & Programming

- b) Write the TeachingStaff class according to the class diagram given, with the 3 attributes, a constructor, getter (properties) for attributes staff_id, rate_per_day and setter method for rate_per_day. The `__str__()` method returns a string in the following format:

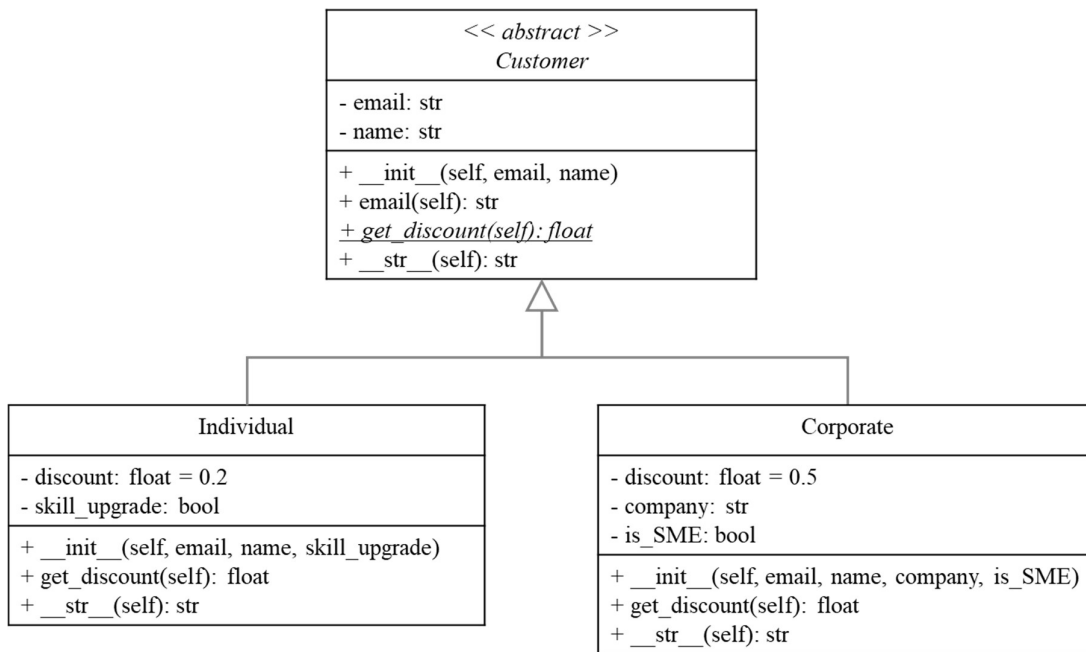
Staff ID: IT0101 Name: Jed Lee

c) Write the CourseOffering class with the following requirements:

- The schedule_id attribute is obtained by concatenating the subject code with the class variable _sch_id
 - o For example, if the subject code is CSIT121, then the schedule_id is CSIT121_1. If there is another course, then the schedule_id is CSIT114_2 etc.
 - o The class variable _sch_id should increment by 1 for every CourseOffering object created.
- The subject attribute is the Subject object.
- The instructors attribute is a list containing TeachingStaff objects.
- The start_date attribute is the starting datetime of the course offering. It is a datetime object.
- Attribute days is an integer value representing the duration in days.
- Besides the constructor, there are getter methods for schedule_id and start_date.
- The class method change_sch_id() modifies the class variable _sch_id to the new_id in the parameter.
- The get_course_fee() method computes and returns the course fee by taking the development cost of the subject, plus the duration in days multiply by the instructors' rate per day.
- The __str__() method returns a string in the following format:

```
Schedule ID: CSIT121_1      Start Date: 1/6/2024      Duration: 5 days
Subject Code: CSIT121      Name: Object Oriented Design & Programming
Staff ID: IT0101      Name: Jed Lee
Staff ID: IT0103      Name: James Wong
Basic Cost: $699.00
```

2. The following classes model customers who enrolled with UOW.



- a) Write the Customer class, which is an abstract class, with the following attributes and methods as shown in the class diagram:
- Two instance variables: email and name
 - The constructor initialises the email and name.
 - Define an email property that return the customer email.
 - An abstract `get_discount()` method which returns the discount the customer is entitled to. It has no implementation.
 - `__str__()` method which returns a string of the instance variables, in the following format:

Email: joelim@email.com Name: Joe Lim

- b) Write the Individual class, which is a subclass of Customer class:
- An additional instance variable: `skill_upgrade`. This value is True if the individual qualifies for skill upgrade.
 - One class variable, `discount = 0.2`
 - The `get_discount()` method returns the discount entitled. Customers who qualify for skill upgrade enjoys a discount of 20% as indicated in the class variable. Without the skill upgrade eligibility, there is no discount.
 - The `__str__()` method includes the email, name, and skill upgrade status, which should be in this format:

Email: joelim@email.com Name: Joe Lim Skill upgrade: Yes (or No)

- c) Write the Corporate class, which is also a subclass of Customer class:
- Two additional instance variables: `company` and `is_SME`

- One class variable, discount = 0.5
- The get_discount() method returns the discount entitled. All Corporate customers are enjoying 50% discount on course fees as indicated by the class variable, but SMEs get an additional 10% more (i.e. 60%).
- The __str__() method includes the email, name, company and is_SME status as follows:

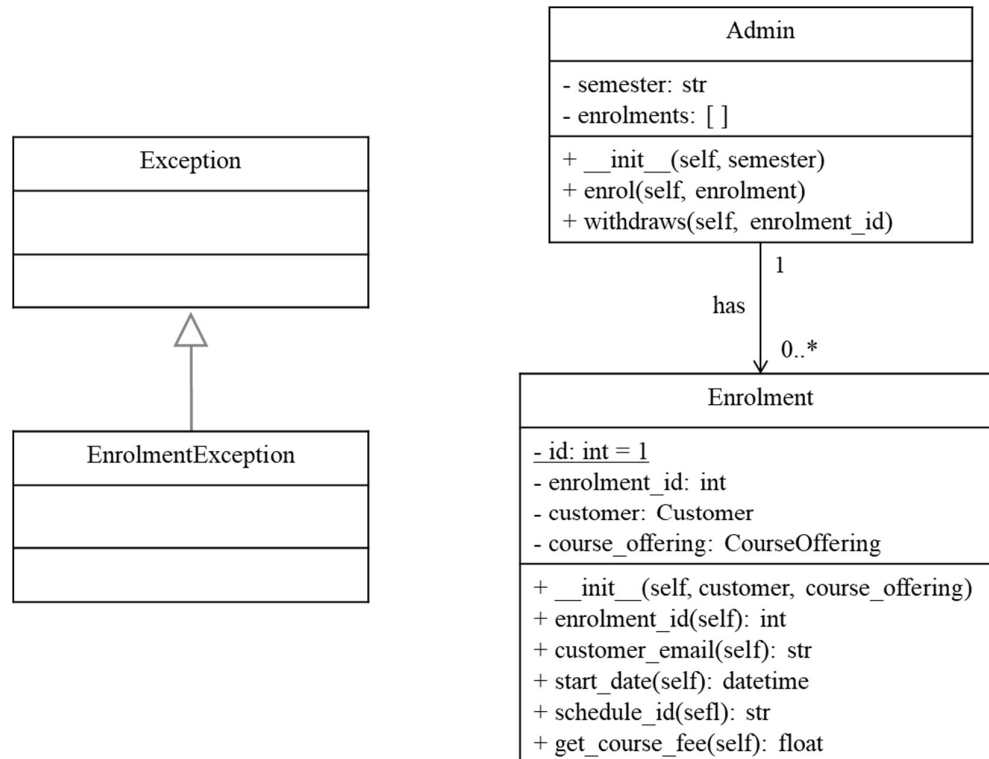
Email: as@email.com

Name: Ali Shan

Company: ANZ

SME: No

3. The class diagrams model an Enrolment that is tracked by the UOW Admin.



- Create the class **EnrolmentException**, which is a subclass of **Exception**.
- Write the **Enrolment** class, which captures a Customer enrolling for a **CourseOffering**.
 - The **Enrolment** class has enrolment id (starts from 1 from the class variable `id = 1`, and increments by 1), customer and `course_offering` as instance variables.
 - In the constructor, check the followings:
 - o If this enrolment (use current datetime) is later than the `course_offering`'s start date, raise an **EnrolmentException** with message "Course start date cannot be earlier than enrolment date".
 - o This enrolment must be at least 2 days before the course offering start date, failing which an **EnrolmentException** should be raised, with message "Can only enrol a course at least 2 days in advance".
 - Create getters/properties for the followings:
 - o enrolment id for this enrolment

- customer_email that returns the email of the enrolled customer
 - start_date that returns the course offering's start_date
 - schedule_id that returns the course offering's schedule_id
 - The get_course_fee() method computes and returns final course fee, which is the course fee from the CourseOffering, taking into consideration the discount the Customer is entitled to.
- c) Write the Admin class, which mimic the admission administration for a semester.
- The class has Two instance variables: semester and a list to store the enrolments.
 - It has method enrol(self, enrolment: Enrolment), that adds an Enrolment object to the list, provided that
 - the same Customer cannot enrol to the same course offering
 - one customer cannot enrol more than 2 course offerings in one semester
 - raise EnrolmentException with appropriate message if any of the above 2 conditions is violated
 - The method withdraws(self, enrolment_id: int) removes an enrolment from the list given the enrolment id.
 - If the enrolment id is not found, raise an EnrolmentException with the message "Enrolment not found!".
 - If the course has started, i.e. the start date of the course offering is on the same day as the withdrawal date or earlier, raise an EnrolmentException with appropriate message.
- d) Test your classes with the basic setup as follows:

```
def main():
    ad = Admin("2024-Apr")
    s1 = Subject("Prog101", "Python Programming", 2000)
    s2 = Subject("DB102", "All about Database", 1800)
    inst1 = TeachingStaff("IT101", "Jed Lee", 800)
    inst2 = TeachingStaff("IT103", "James Wong", 850)
    co1 = CourseOffering(s1, [inst1, inst2], datetime(2024,5,23), 3)
    co2 = CourseOffering(s2, [inst2], datetime(2024,5,22), 5)
    c1 = Individual("t1@email.com", "Test 1", False)
    c2 = Individual("t2@email.com", "Test 2", True)
    c3 = Corporate("t3@email.com", "Test 3", "ThatCompany", True)
    c4 = Corporate("t4@email.com", "Test 4", "ThisCompany", False)
    for co in [co1, co2]:
        for c in [c1, c2, c3, c4]:
            e = Enrolment(c, co)
            ad.enrol(e)
            print(e.get_course_fee())
```

Create more test scenarios such that your classes will raise exceptions. Ensure that all exceptions are properly handled.