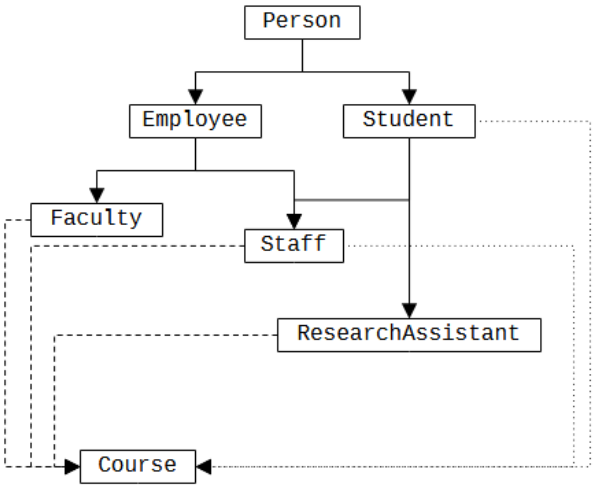
CSDS 297/397 Special Topics: C/C++ Programming Fall 2024

Assignment 3: Classes, Inheritance & Polymorphism

Instructor: Thomas Shkurti Due: Wednesday, December 4 before 11:59 PM

This assignment involves creating a small set of objects to model a university department:



Please download `HW3.zip`. It will include a `main.cpp`. This file creates objects and makes some `print()` calls at the end that are intended to display the structure of the department. When you open it, there will be errors because the classes it calls are missing. You will create classes and source/header files to fill in those errors.

Your submission should only include the `.h` and `.cpp` files discussed below. You are free to modify the `main.cpp` file to test additional functionality. We will use our own copy of `main.cpp` to grade.

Question 1

Create the base class Person (`person.h` and `person.cpp`). A Person should contain only an integer ID and a string for a name. You should have both a full constructor (i.e. `Person(int ID, string name)`) and an empty constructor (i.e. `Person()`) which initializes the member values to some sort of defaults that a real Person could never have (like an ID of 0 and a name that is an empty string).

You should also implement getter and setter methods for each field, as well as a `print()` method that outputs the name and then the ID.

Question 2

Create the classes Student and Employee (`student.h` / `student.cpp`, and `employee.h` / `employee.cpp`). Both Students and Employees should inherit from Person.

A Student should not have any of the Employee fields or methods, but should have

- An integer graduation year
- A major (a string).
- A GPA (a float that will normally take on a range between 0.0 and 4.0)

You should have both a full constructor (i.e. `Student(int ID, string name, int grad_year, string major, float GPA)`) and an empty constructor (i.e. `Student()`) which initializes the member values to some sort of defaults that a real Student could never have (like a negative GPA and a major that is an empty string). You should also implement getter and setter methods for each field in the Student, as well as a `print()` method that outputs the graduation year, the major, and then the GPA, as well as the name and ID.

An Employee should not have any of the Student fields or methods, but should have

- A float Salary
- A Department the Employee works for. This should be an enum with the values `Department::CS`, `Department::EE`, `Department::MATH`, and `Department::BME`, and `Department::None`.

You should have both a full constructor (i.e. `Employee(double salary, string name, int ID, Department dep)`) and an empty constructor (i.e. `Employee()`) which initializes the member values to some sort of defaults that a real Employee could never have (like a zero salary and a department of None). You should also implement getter and setter methods for each field in the Employee class, as well as a `print()` method that outputs a string describing the department and then the salary, as well as the person's name and ID.

Question 3

Create two types of Employee: Faculty and Staff (`faculty.h` / `faculty.cpp`, and `staff.h` / `staff.cpp`).

Staff have none of the Faculty fields or methods, but have

- A Title string different from the Faculty title ("Administrative Assistant", "Student Affairs Rep", etc.),

You should have both a full constructor (i.e. `Staff(int ID, string name, double salary, Department dep, string title)`) and an empty constructor (i.e. `Staff()`) which initializes the member values to some sort of defaults that a real Staff could never have (like a title that is an empty string). You should also implement getter and setter methods for each field, as well as a `print()` method that outputs the Title, salary, department, then the name and then the ID.

Faculty have none of the Staff fields or methods, but have

- A Title string different from the Staff title ("Associate Professor", "Assistant Professor", "Senior Professor" and so on)
- a Research Focus string ("Bioinformatics", "Complexity Theory", "Robotics", and so on)

You should have both a full constructor (i.e. `Faculty(int ID, string name, double salary, Department dep, string title, string research)`) and an empty constructor (i.e. `Faculty()`) which initializes the member values to some sort of defaults that a real Faculty could never have (like empty strings for the title and research focus). You should also implement getter and setter methods for each field, as well as a `print()` method that outputs the Title, then the Research Focus, then the Salary and a string describing the Department, then the name and then the ID.

Question 4

Create a ResearchAssistant class (`research_assistant.h` and `research_assistant.cpp`). A ResearchAssistant has the properties of both a Student and a Staff, since either profession can take that role. The ResearchAssistant itself also has a Research Area field.

It should thus have two constructors:

```
ResearchAssistant(
   string area, string title, Department dep, double salary, int id, string name
)
```

creates a ResearchAssistant that is like a Staff member, and

```
ResearchAssistant(
   string area, string major, double gpa, int grad_year, int id, string name
)
```

creates a ResearchAssistant that is like a Student. The fields not applicable to the ResearchAssistant in question should be initialized automatically to default values no real person in that role could have (see the previous Question). You do not need to create an empty constructor `ResearchAssistant()`.

The ResearchAssistant should have getters and setters for all its fields, and it should also have a `print()` function. This function should behave differently depending on whether the ResearchAssistant has defaults in its Student or Staff fields.

- If it has defaults in the Staff fields, the ResearchAssistant is actually a `Student` (because it was given only *Student* information), and should print its Research Area, graduation year, major, GPA, name, and ID number.
- If it has defaults in the `Student` fields, the ResearchAssistant is actually a `Staff` member (because it was given only *Staff* information), and should print its Research Area, a string describing its Department, Title, Salary, name, and ID number.
- If it has defaults in both fields, or defaults in *no* fields, it should print an error message.
- Setters should print an error message if setting a `Student` field on a `Staff` member or vice versa, and should *not* alter the field.

Question 5

Create a `Course` class. `Course`s don't inherit from any of the previous classes, but do interact with them internally.

A `Course` has a Name (a string), a Department enum, and an integer Number.

A `Course` should *also* have a Teacher, which can be one of three classes: `Faculty`, `Research Assistant`, or `Staff`. Any `Faculty` or `Research Assistant` can teach a `Course`, but an ordinary `Staff` member can only teach a `Course`, if their Title string is equal to "Instructor".

A `Course` can have an arbitrary number of students. Either actual `Student`s can take the course, or `Staff`, but a person cannot take a course that they are currently teaching or teach a course that they are currently taking.

You should have both a full constructor (i.e. `Course(int number, string name, Department dep)`) and an empty constructor (i.e. `Course()`) which initializes the member values to some sort of defaults that a real `Course` could never have (like an empty string for the name). You do not need to be able to construct `Courses` with students and teachers in place, but you *do* need to create getters and setters that can view and add them. The setters should leave the element (student or teacher) as it was previously, if the new element does not satisfy the rules above. Getters should return an empty vector of `Students` or a blank `Staff` object if called on a `Course` that does not have a teacher assigned or does not have any students.

You will also need to create a `printTeacher()` and `printStudents()` method. This should print out the *full information* for the teacher, and for each student, respectively (that is, if the teacher is `Faculty`, it should print their research interest, etc.).