Patient & Doctor

Learning objectives

- Model a real world interaction using OOP
- Practice encapsulation

Introduction

We will be using simple OOP to model interaction between a doctor and their patient. There are many ways to meet the requirements of the assignment, so think about the problem and how it would best be modeled. Consider sketching out the interactions and properties on a whiteboard.

Tools

Create a new command-line Objective C project.

Use `NSLog` to help you get visibility into and understand the flow of messages as these two classes communicate with each other.

Instructions

Task 1 - Setup & General Information

The doctor should be able to ask the patient for some general information about them (age and name for example). The patient should be able to ask the doctor their name and specialization.

Implement two classes `Doctor` and `Patient`. Think about how instances of doctors and patients should be constructed. What general information for each of them need to be initialized, based on the instructions above?

Create custom initializers for both the `Doctor` and `Patient` class. Use the custom initializer to signal to other coders using your classes how they are to be used.

Task 2 - Visiting a Doctor

A patient should be able to visit a doctor. Tip: This is an action that the patient is executing and requires an instance of a doctor to do so.

The doctor instance should be able to accept a patient. In order for the doctor to accept the patient, the patient must have a valid health card. When the

patient visits the doctor the doctor will ask them if they have a health card. If the answer is no, then the doctor cannot accept the patient. If they answer is yes, then the patient is accepted. The doctor should keep track of all their accepted patients. Think about which collection type is best suited to this task.

Task 3 - Treating a Patient

A patient can request medication from the doctor. The patient will do this by calling `requestMedication` on the doctor instance. Only patients previously accepted by that doctor can ask for prescriptions. In order for the doctor to create and return a prescription to the patient, the doctor needs to know what symptoms the patient is currently experiencing. This will affect what is prescribed.

Suggestion: While you could keep things simple and not define any other classes to accomplish this, you may want to consider what other objects are involved in this interaction.

Task 4 - Prescriptions are Shared

Even though a particular doctor writes prescriptions for their patients, a patient's doctor should be able to access any prescription that is ever written for that patient. While it's true that our health care system doesn't presently

work this way, it should! Anyway, once the doctor writes the prescription, before handing it the patient, the doctor should add it the collection of prescriptions written by any doctor. What type of collection is best suited to this problem?

Tip: Since any other doctor can access these prescriptions, think about where this collection of prescriptions will be stored. As well, think about who should be able to read/change this information. In other words, should an instance of a patient be able to read/write this information? Ideally not!

Task 5 - Modularity

Clean encapsulation of a problem makes your classes modular. That means they can be easily swapped out for other classes that solve the same problem in a different way. Your coffee maker doesn't care what type of container is placed under it, as long as it meets the requirements (i.e. large enough to hold all the coffee, and short enough to fit under the spout).

With that in mind, find another student who has reached this step and trade `Doctor` and `Patient` classes with them. Without modifying the classes, just your `main.m`, how much effort is needed to adapt their classes to your interaction. Try to avoid reading the `.m` files and instead use the header files as "documentation" of how the objects work.