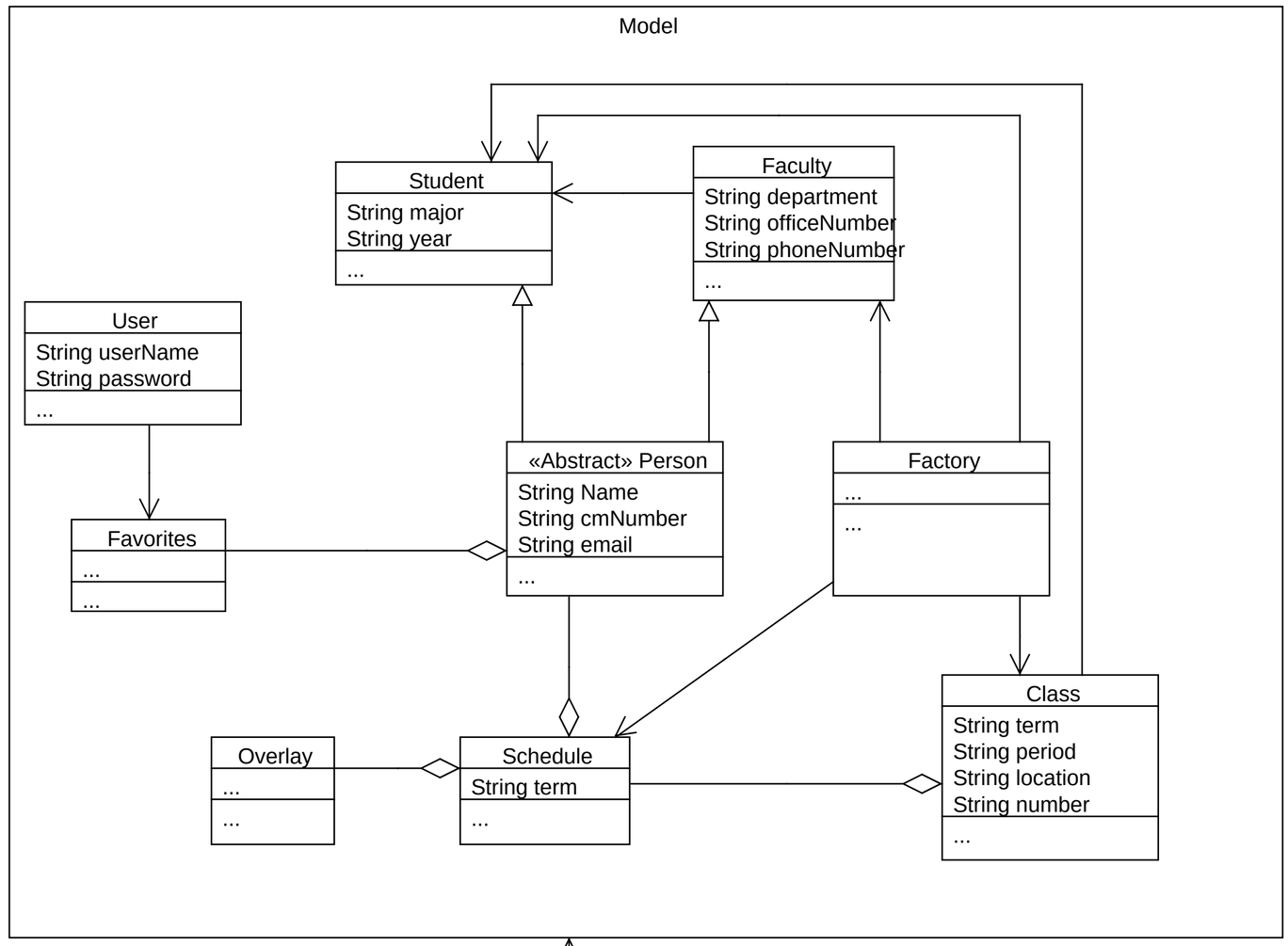
Due to iOS programming standards and constraints, we must adhere to Model, View, Controller Class architecture. Here we have diagrammed our View, which depends on the Controller, which then depends on the Model. As iOS deals with Views as XML entities instead of actual classes, they have been shown as simple boxes, and do not have any individual private or public variables or methods.

We are using the GRASP Principle of Simple Fabrication by "fabricating" factory classes which will fetch the website HTML for us and convert it into data for a Person(student / faculty).
In addition, the factory is a Protected Variation use, because if the website ever changes, we would only ever need to change the factory where all the scraping code is.
This also adds a layer of indirection, so the classes aren't directly talking to the web service.
This then makes the Factory a Creator Pattern, since the Factory will then have the information to create and use Faculty, Class, Student, and Schedule.

Low Coupling in our example is shown through the structure of the classes in Model. The Factory is the only class that will interface with web services returning, so if something changes, only one class is reliant on being changed to work properly. These classes will work well if they need to be reused, because if there were some different changes to the web service,such as if they no longer supported having faculty schedules, with our current class abstraction, we could still manage changes.
As far as High Cohesion, we constructed our classes so that they make the most possible sense, we have Faculty and Students, which bot hcomr from the Person class, and share some features of the Person class. We also have the classes constructed in the way that the code will perform. Such as the Overlay having many Schedules, and the User having a Favorites.
These are Examples of Polymorphism principles as well, having the Students and Faculty extend the Person class.



ViewControllers extend UIViewController

The SettingsViewController will be our Information expert, since we are assigning it the responsibilities of retrieving the password from the keychain, since this is where the user will store the password.
The Controller scheme fits the conglomeration of classes in the Controller group to the right, all of the classes will interface with the GUI and the Model depending on what the Information is.