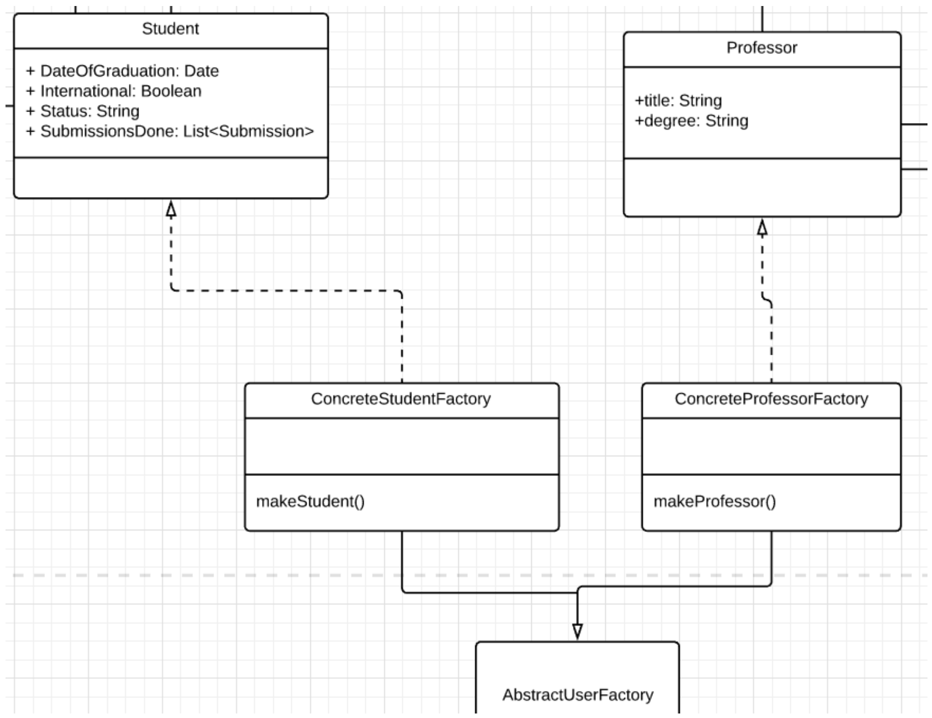
In the UML for our project we chose to use the following design patterns:

1. Abstract Factory Pattern

This is a creational design pattern that provides flexibility with respect to object creation by abstracting this logic from the client.

Usage

We decided to use this design pattern while creating user instances – student and professor.
The driver program will be exposed to ***AbstractUserFactory*** that provides methods:
makeProfessor()
makeUser()
The **ConcreteStudentFactory** and **ConcreteProfessorFactory** provide the implementation logic for creating these user instances.



Benefits:

- The idea was to make the system adaptable to changes in the future, i.e., if we want to replace existing users with specialized users (adding another factory would require only few modifications)
- The underlying concrete implementation and instance creation, composition and representation are invisible to the client/driver.

2. Singleton

Although we didn't explicitly mention this in the UML diagram, we would like to implement the AbstractUserFactory as a singleton. This class is limited to have a single instance but can also support multiple substitutable implementations.

Benefits:
This will help us ensure that there is just one instance of this factory which is globally accessible.

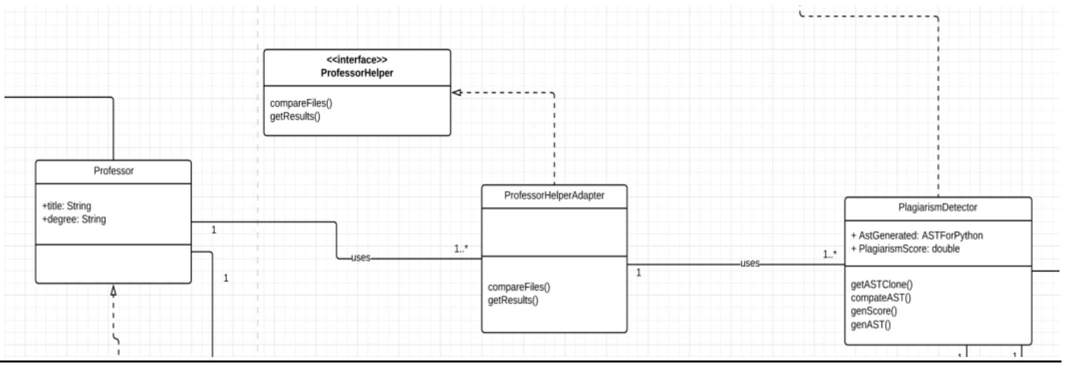
3. Adapter

An adapter class can be used when we want to use an existing class, but its interface does not exactly match what we need.

In our project, the **Professor** class needs to use the functionality of the **PlagiarismDetector** class which inturn implements the ASTcomparator interface. However, the class provides methods like:

getASTClone()
compareAST()
genScore()
genAST()

which are not relevant to what the professor is looking for. Hence, we decided to create a **ProfessorHelperAdapter** class that would call the PlagiarismDetector class methods within the compareFiles() and getResults() to achieve the desired results. The **ProfessorHelper** interface is exposed to the Professor class for use. We used the *object adapter pattern*.



Benefits:

1. The Professor wants an interface to compareFiles() and getResults() so rather than creating this from the scratch we used the methods of the PlagiarismDetector.

2. This logic restricts the Professor to access only those methods of PlagerismDetector that he requires through the ProfessorHelper interface. (Object Adapter Pattern).
3. In the future, a DeanHelperAdapter or a StudentHelperAdapter can be easily added for specialized functionalities.