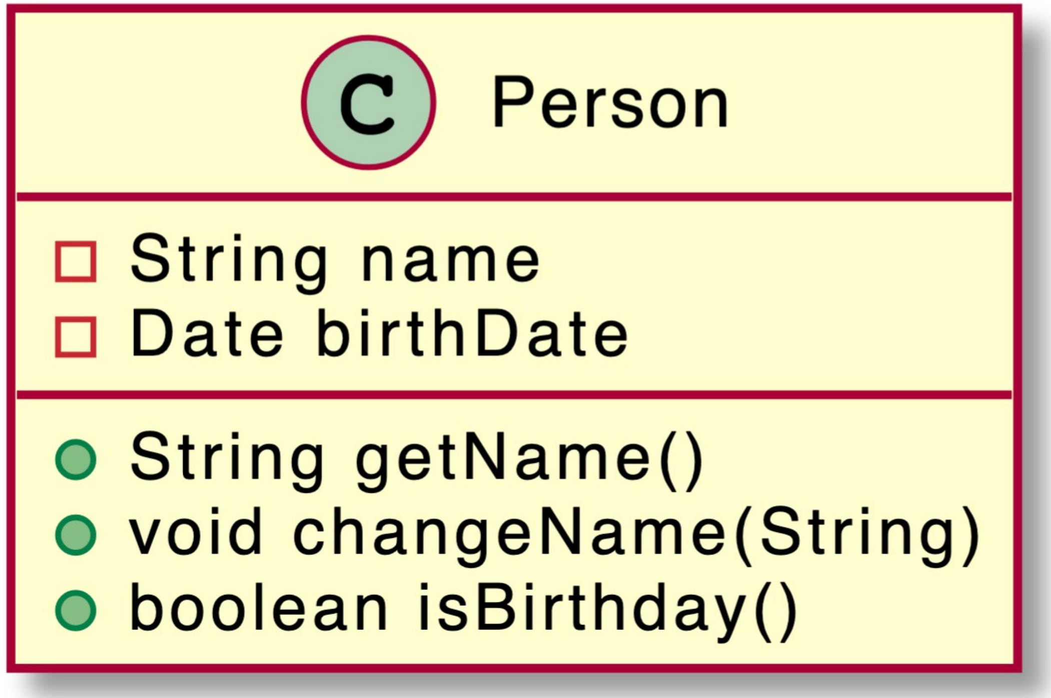
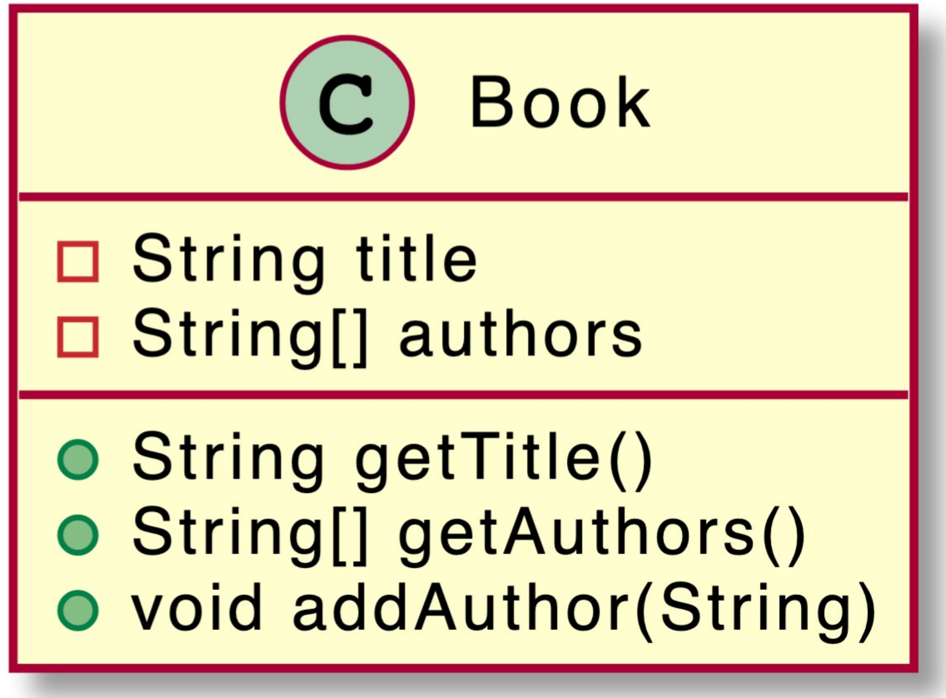
Classes Relationships

Diagramming with UML

Classes





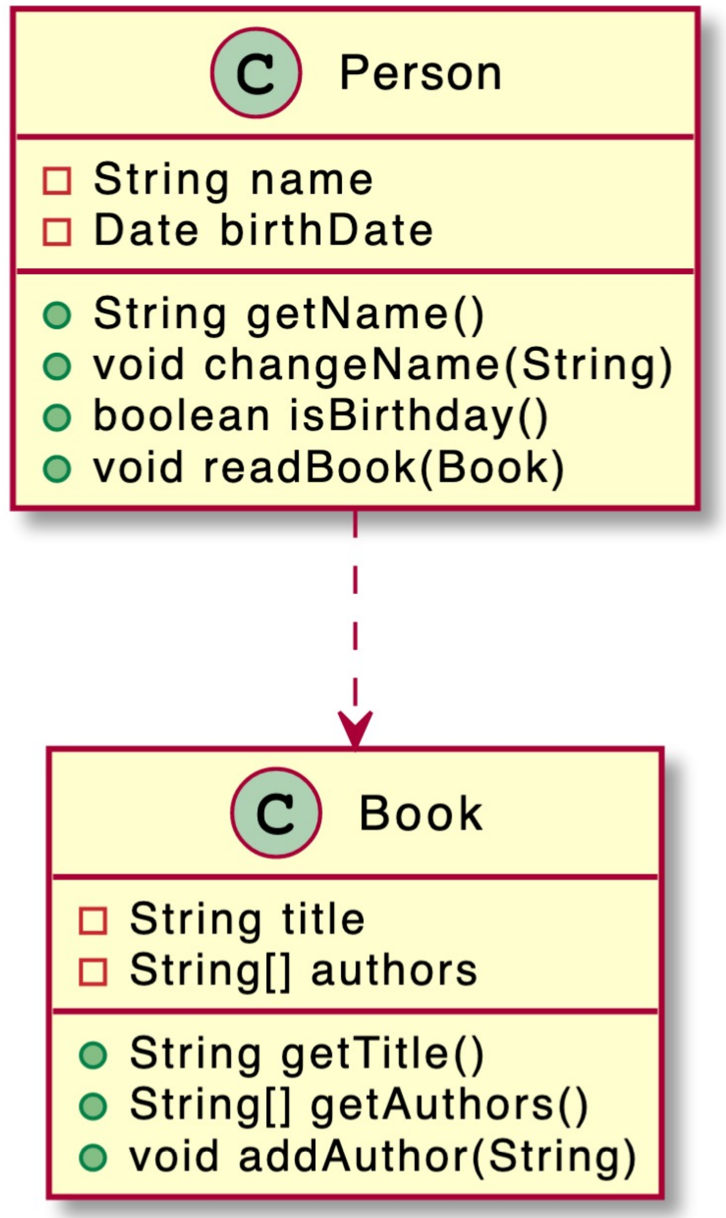
● Public Member (usually a “+” sign)

◆ Protected Member (usually a “#” sign)

□ Private Member (usually a “-” sign)

Dependency

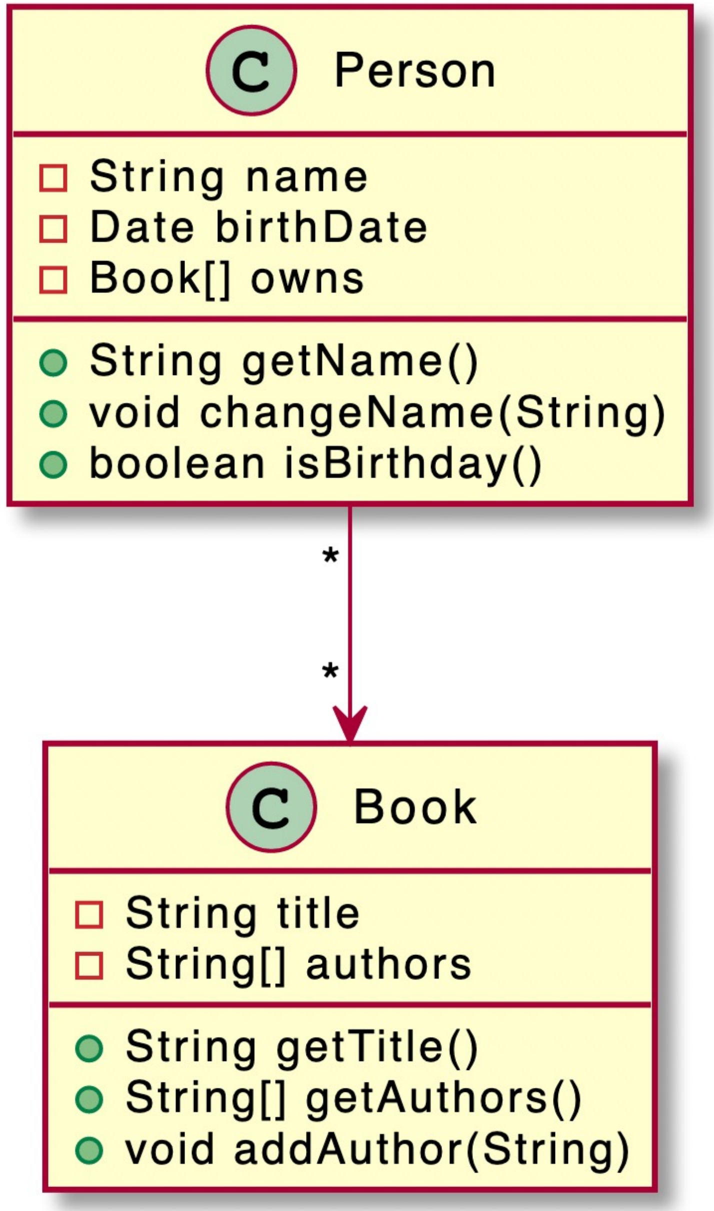
- An object of one class uses an object of another class in a method
- Usually as a parameter to a method
- The object is used but **is not** stored in the class



Association – Unidirectional

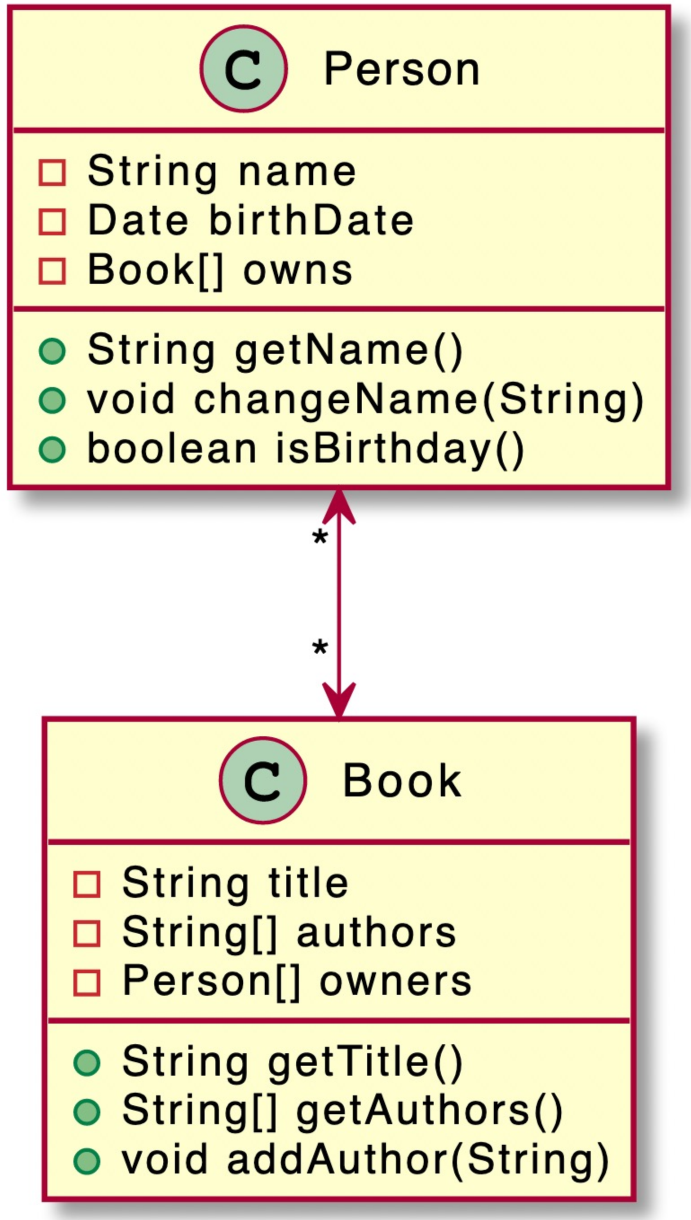
- An object of one class contains an object of another class as a data member

- The *’s are known as multiplicity.
  - a book can be owned by any number of people
  - a person can own any number of books



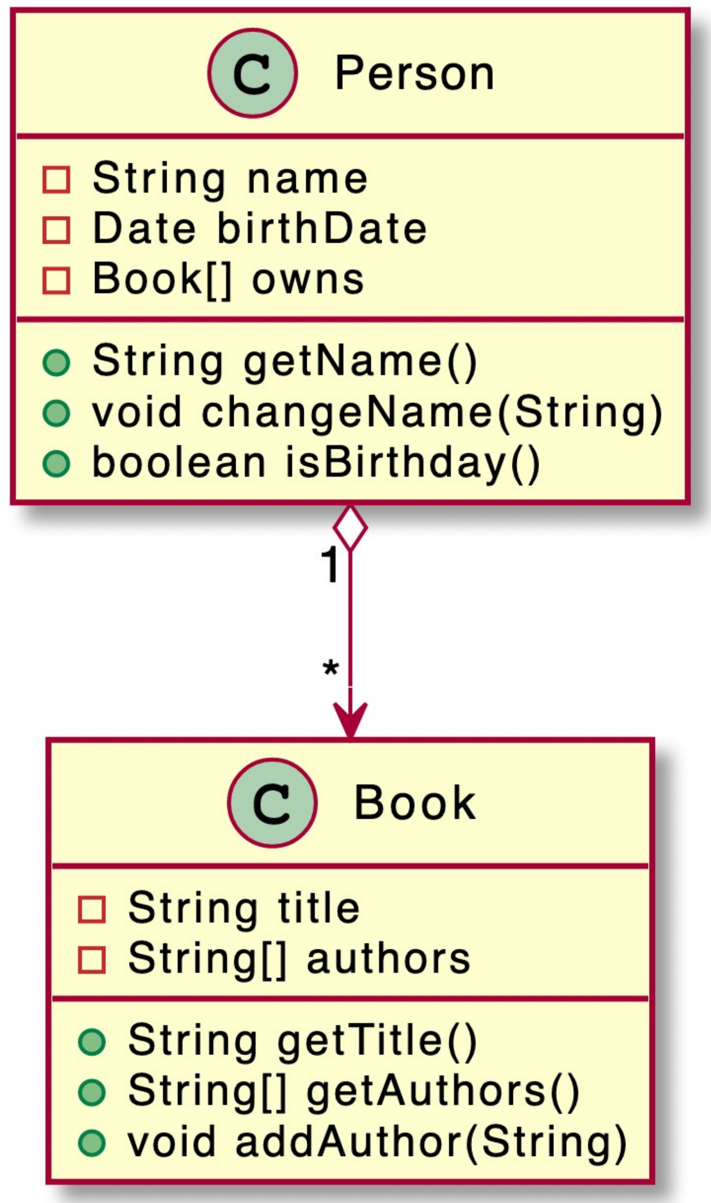
Association – Bidirectional

- Two classes each contain an object of the others type as a data member
- In this case a person lists the books they own while the book also lists people that own it
- Most often this is not an ideal relationship to have
  - Difficult to maintain. What happens when a person stops owning a book?



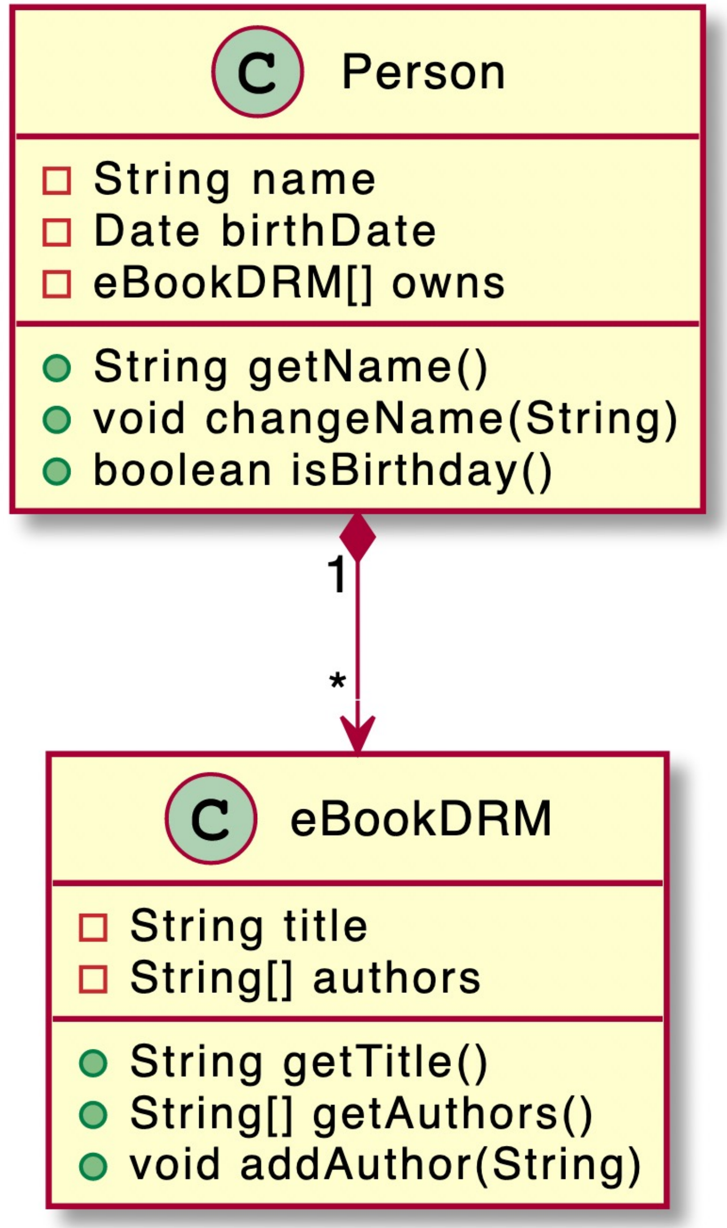
Aggregation

- A subset of association
- Implies that the life-time of Person does NOT determine the life-time of the Book object it holds
- A book may be “owned” by a person, but that same book might also belong to the library
  - If Person quits the library (the Person object is destroyed) the Book still exists for the library

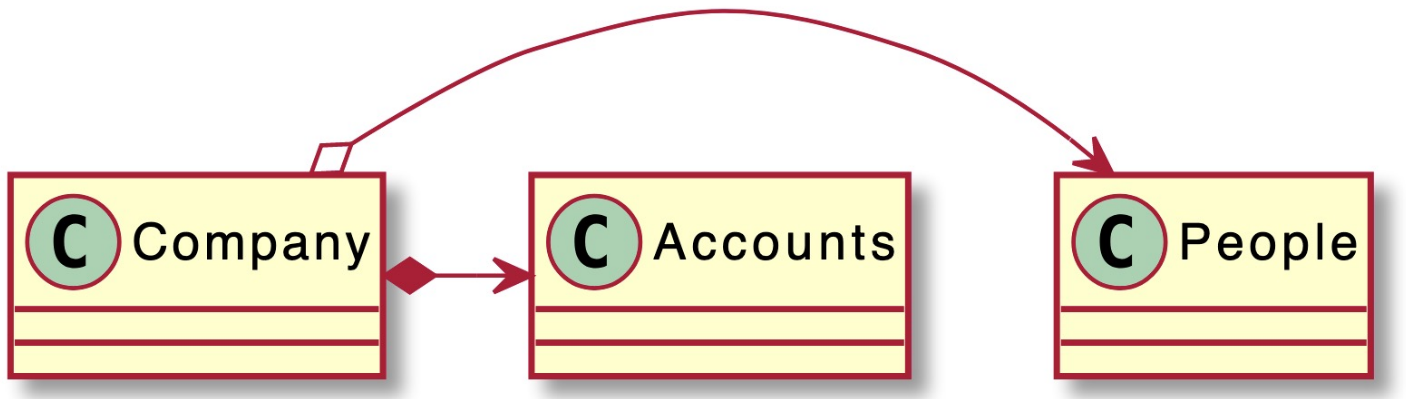


Composition

- Part of aggregation
- Implies that the life-time of Person determines the life-time of the eBook object it holds
- When the person object is destroyed, that specific DRMed eBook is also destroyed as it is specific to the person who bought it



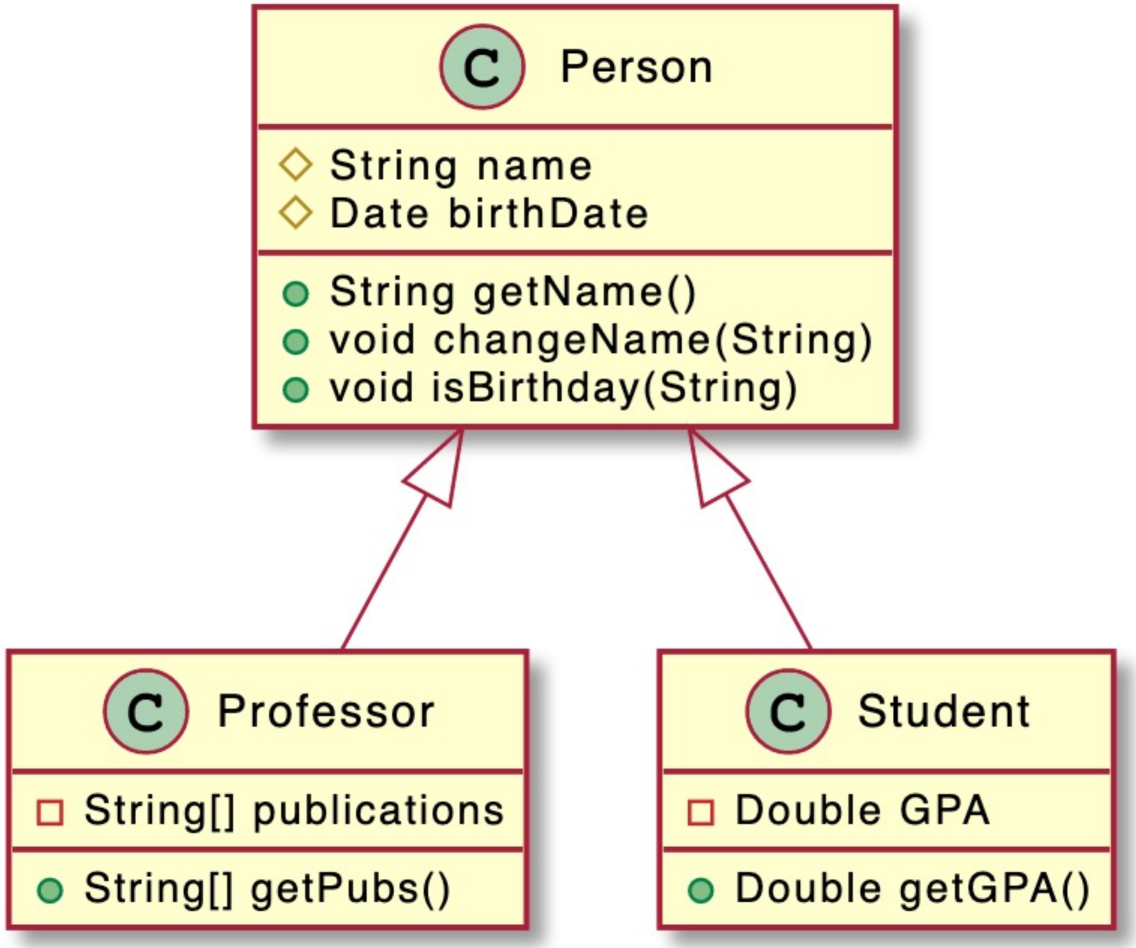
Aggregation vs Composition



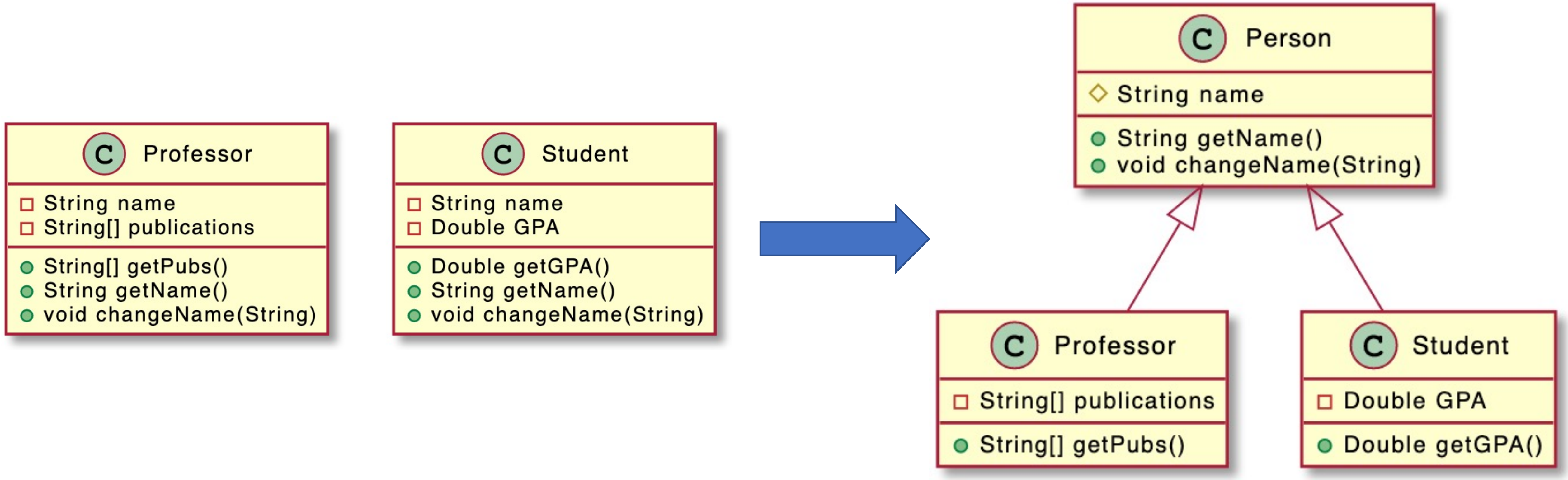
- A Company is an **aggregation** of People
- A Company is a **composition** of Accounts
- If the Company closes, the Accounts cease to exist, but the People still do.

Inheritance

- Arrows point from the derived (child) class to the base (parent) class
- An “is a” relationship
- A Professor is a Person, and a Student is a Person
- All derived classes have copies of the Person object
- Can only access protected or public members from the base class (C++)

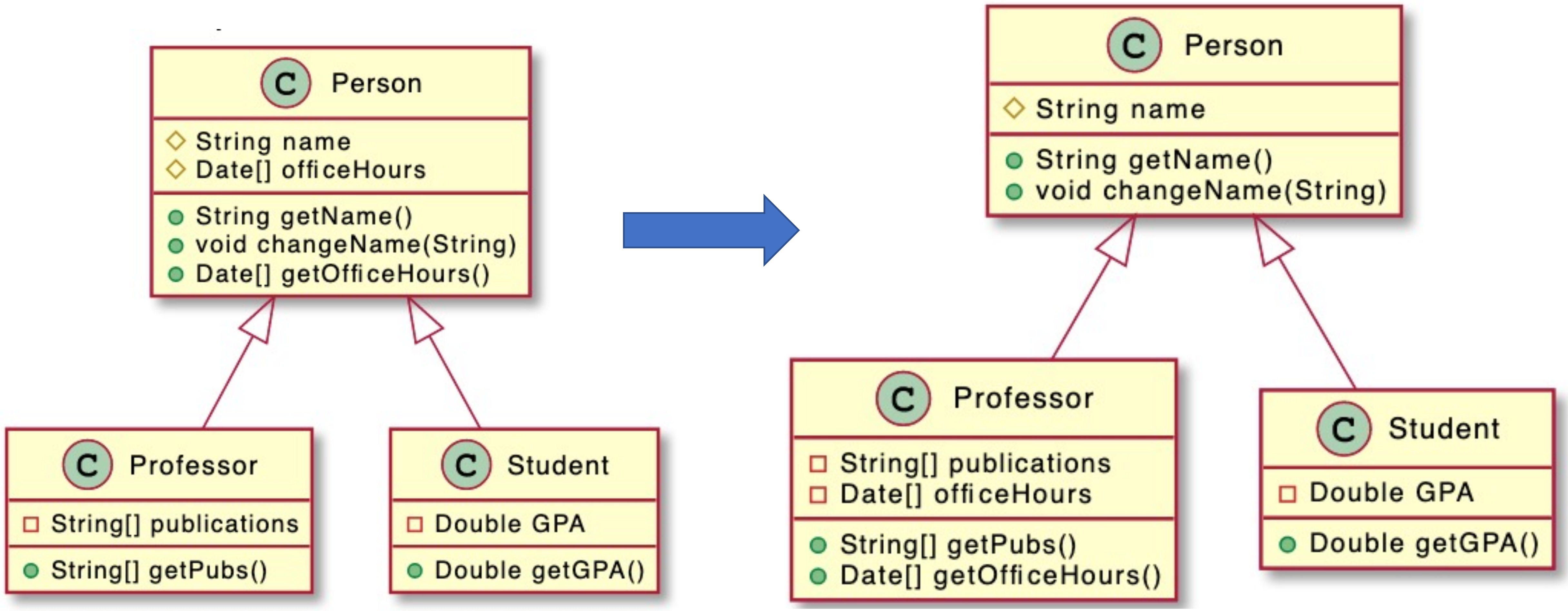


Generalization



Moving members to a more generic class

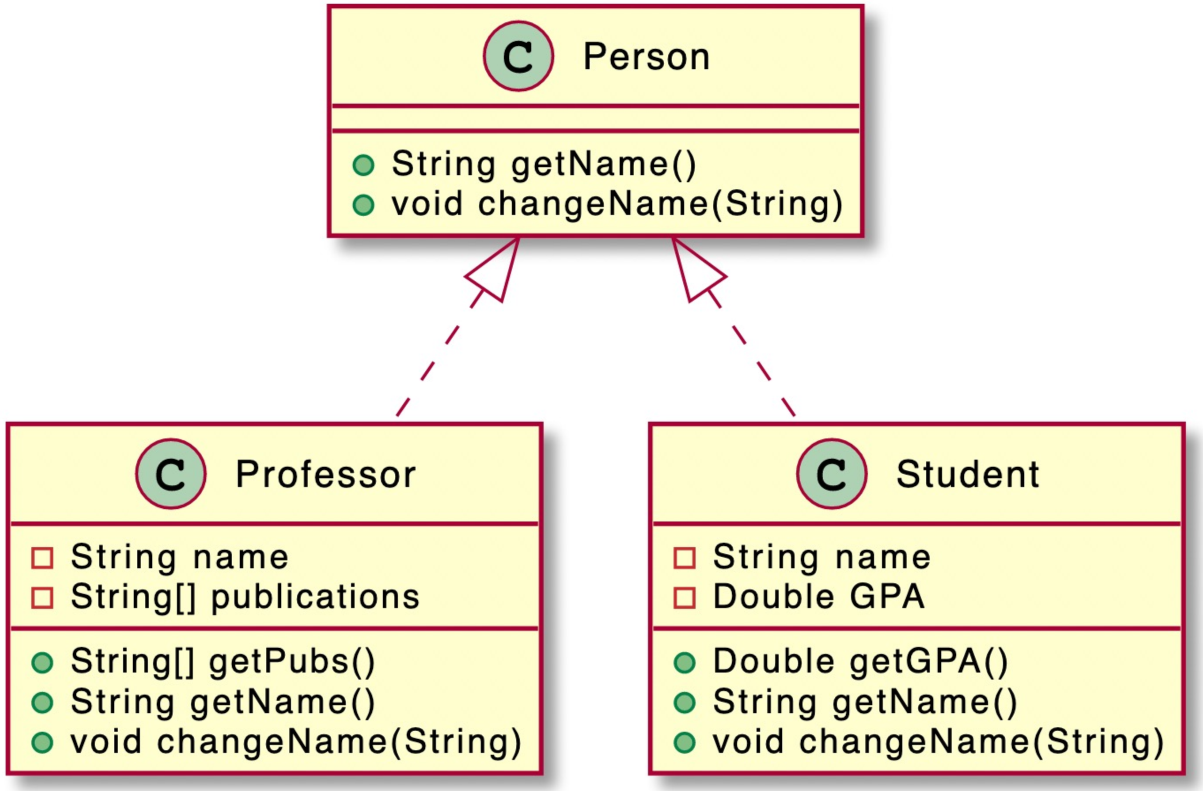
Specialization



Moving members from a base to a derived class

Realization

- We inherit from an interface
  - In python we call this an **abstract base class**
- An interface it not not used directly, but instead serves as a blueprint for similar classes
- When we inherit from an interface, we then are required to implement the functions for our base class to ensure behaviors are present in derived classes

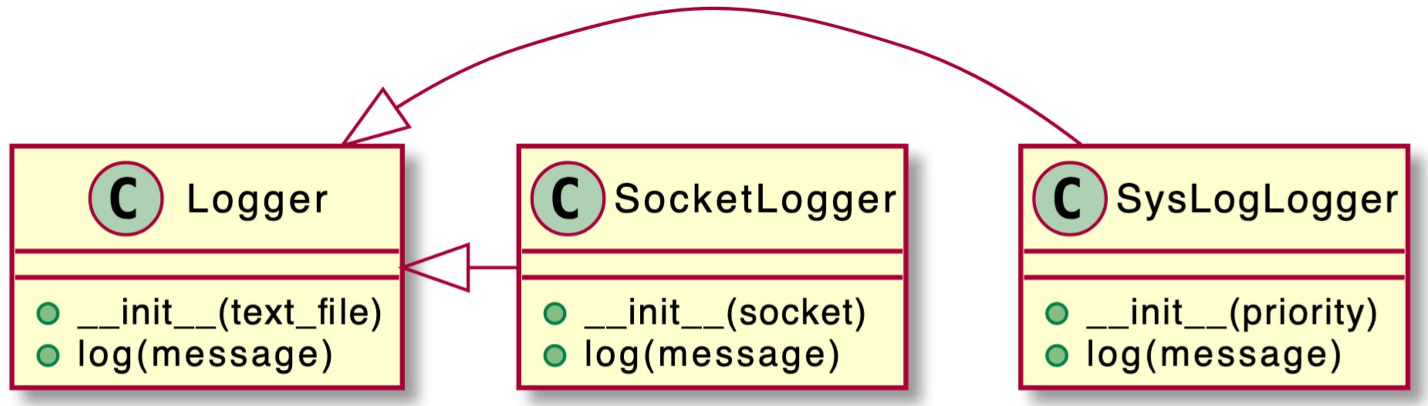


Favor Composition over Inheritance

- Inheritance and composition both serve an important role in OOP
- However, when we use inheritance, we are tightly coupling the base (parent) and derived (child) classes together
- If used improperly, inheritance can result in complicated class hierarchies or a “sub-class” explosion
- An object that is composed of other objects to represent specialized behavior can be used to mitigate this issue (Design Patterns)

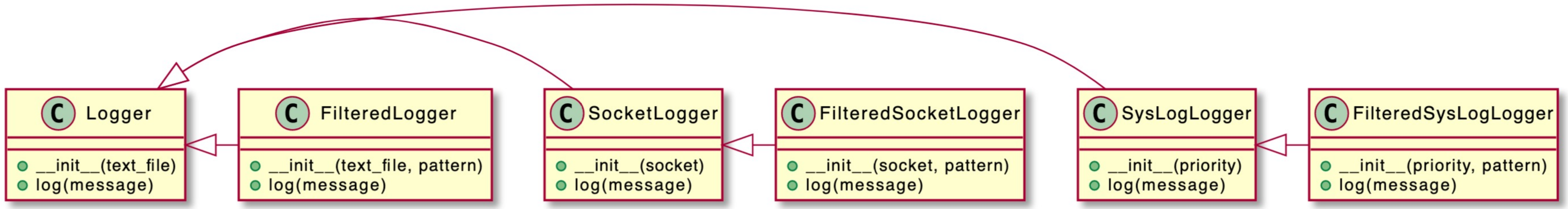
The Sub-Class Explosion

- Let’s assume we have some logging classes where each one logs to a different source
- We get a request to add a logger that can filter only very important error messages



The Sub-Class Explosion

- Let’s assume we have some logging classes where each one logs to a different source
- We get a request to add a logger that can filter only very important error messages
- Now we need a variation of each class to support filtering
- This becomes unmanageable very quickly



C Logger
__init__(text_file)
log(message)
C FilteredLogger
__init__(text_file, pattern)
log(message)
C SocketLogger
__init__(socket)
log(message)
C FilteredSocketLogger
__init__(socket, pattern)
log(message)
C SysLogLogger
__init__(priority)
log(message)
C FilteredSysLogLogger
__init__(priority, pattern)
log(message)