ASSOCIATION
CLASS ASSOCIATION [INHERITANCE]
OBJECT ASSOCIATION [COMPOSITION]

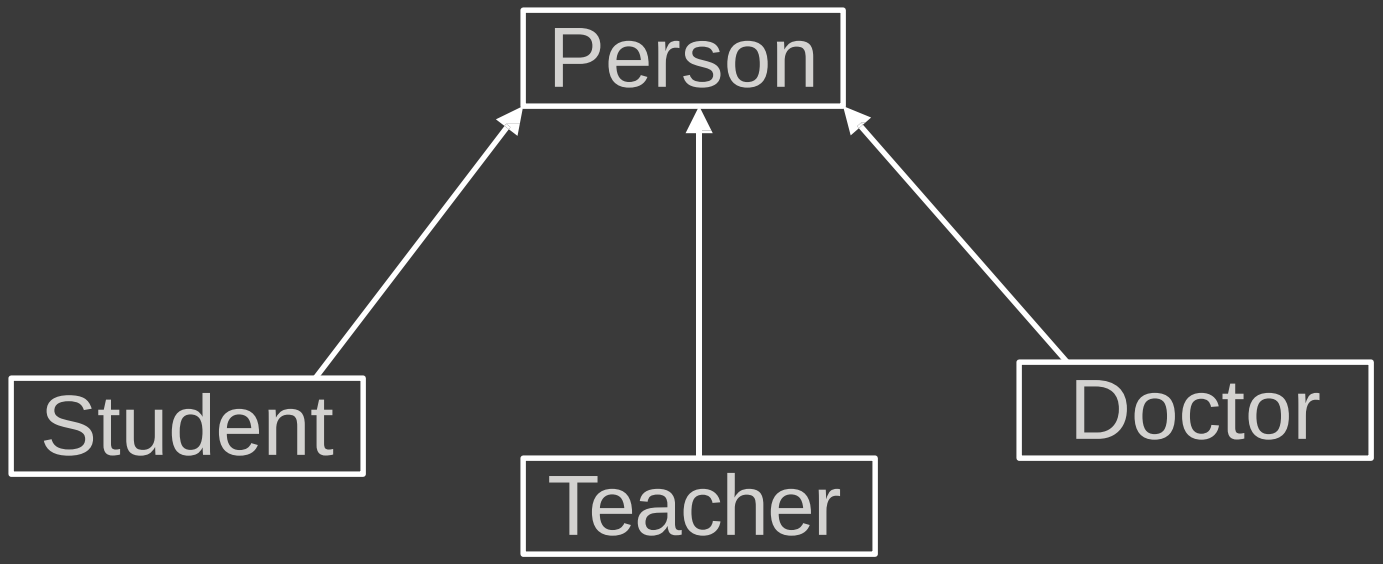
Inheritance

- A child inherits characteristics of its parents
- Besides inherited characteristics, a child may have its own unique characteristics

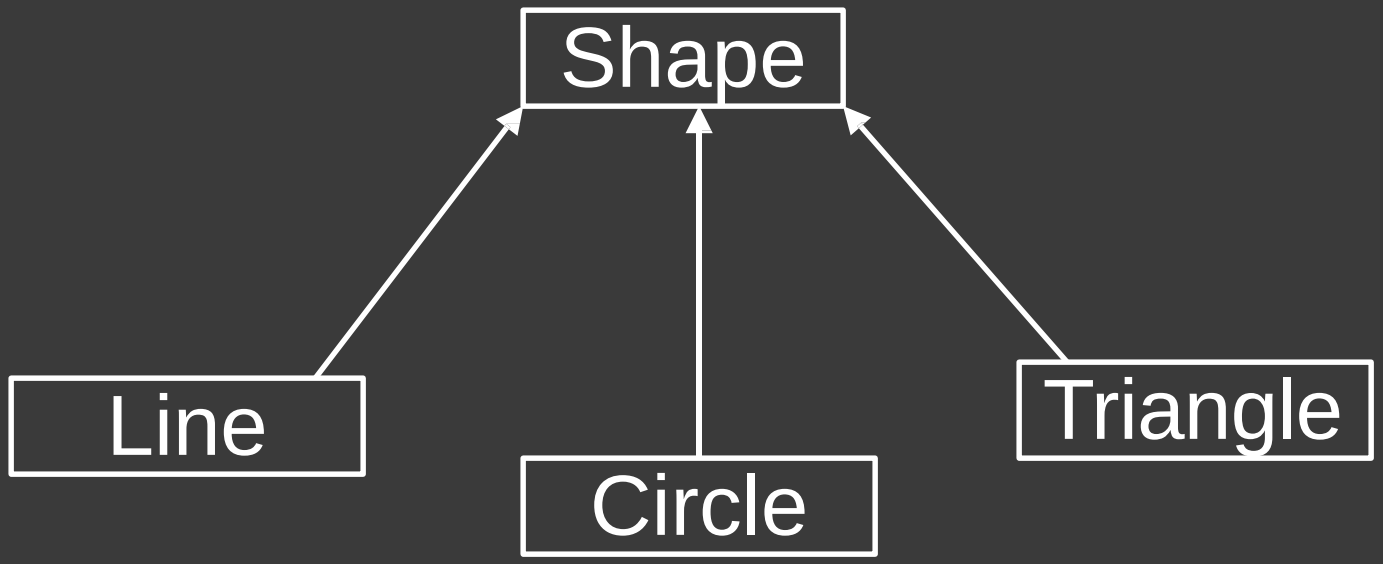
Inheritance in Classes

- If a class B inherits from class A then it contains all the characteristics (information structure and behaviour) of class A
- The parent class is called *base* class and the child class is called *derived* class
- Besides inherited characteristics, derived class may have its own unique characteristics

Example – Inheritance



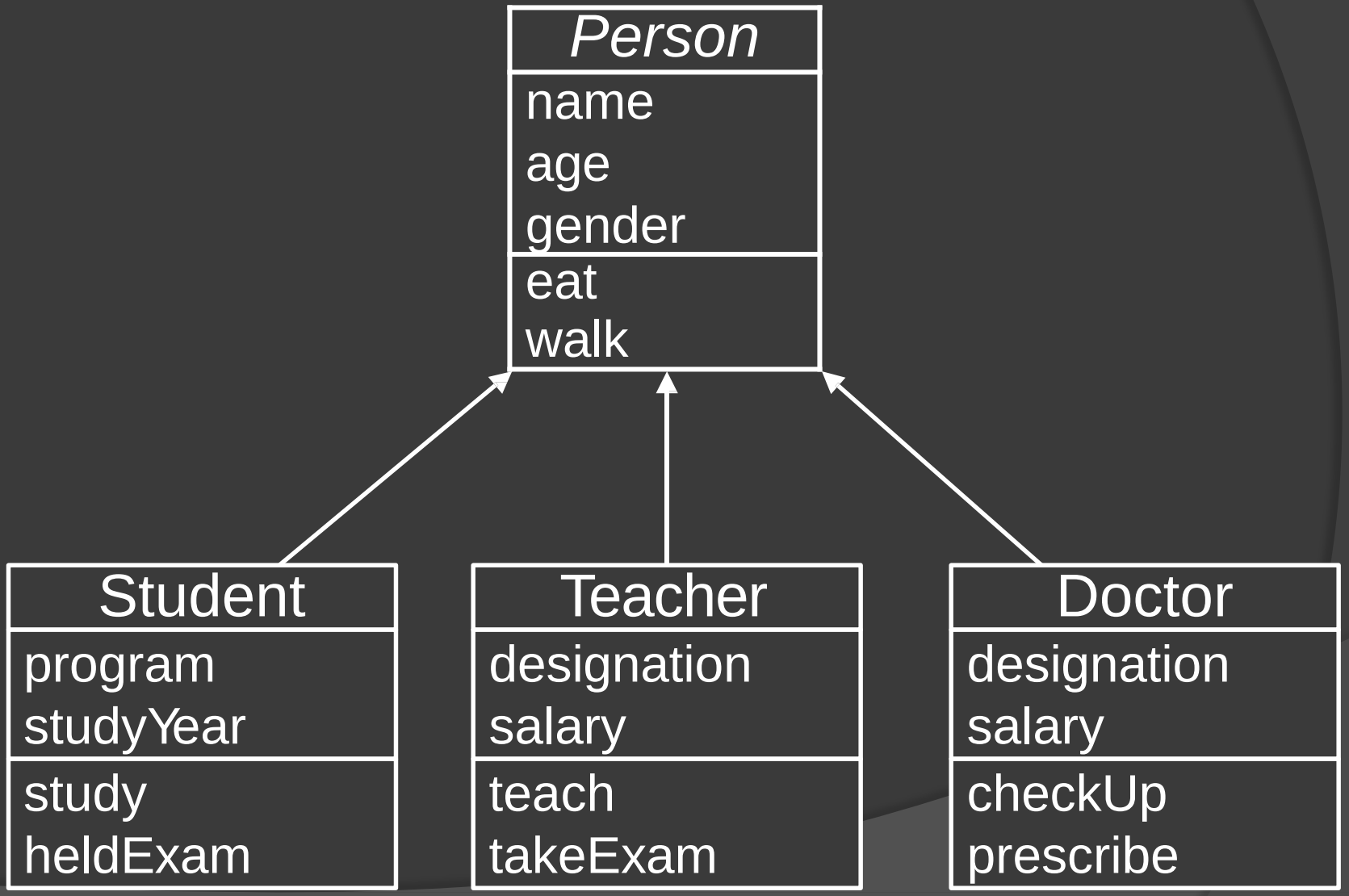
Example – Inheritance



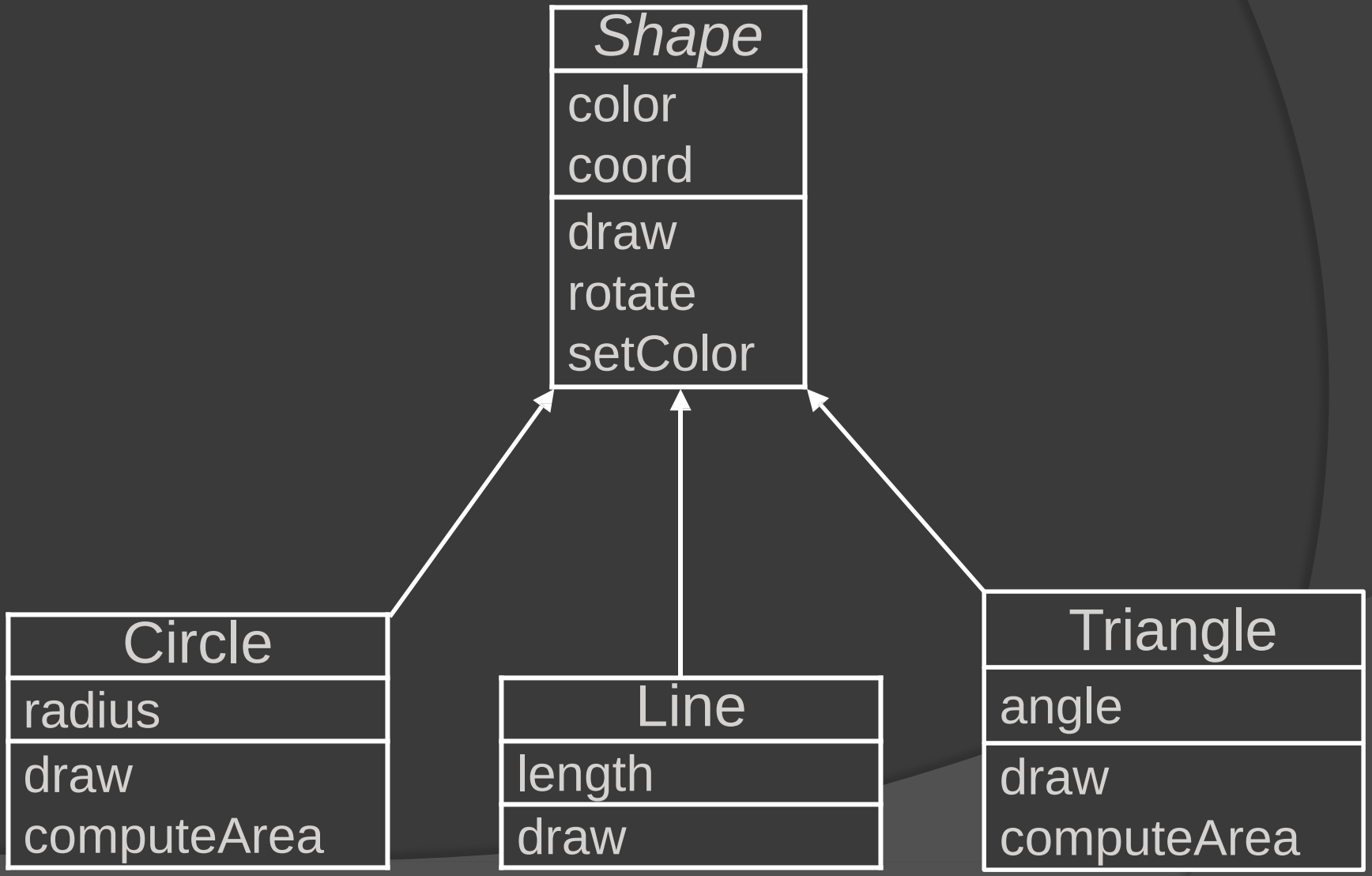
Inheritance – “IS A” or “IS A KIND OF” Relationship

- Each derived class is a special kind of its base class

Example – “IS A” Relationship



Example – “IS A” Relationship



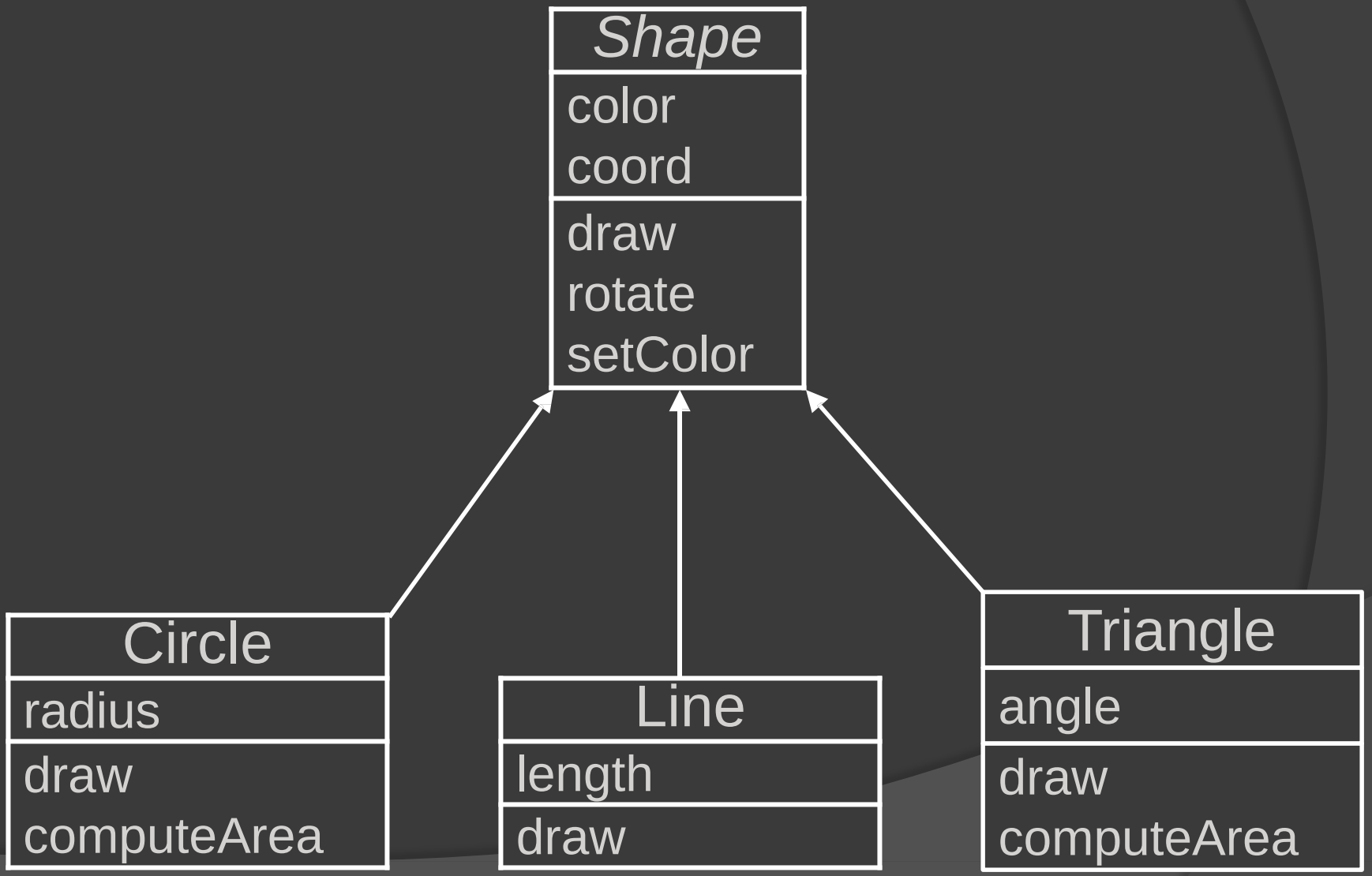
Inheritance – Advantages

- Reuse
- Less redundancy
- Increased maintainability

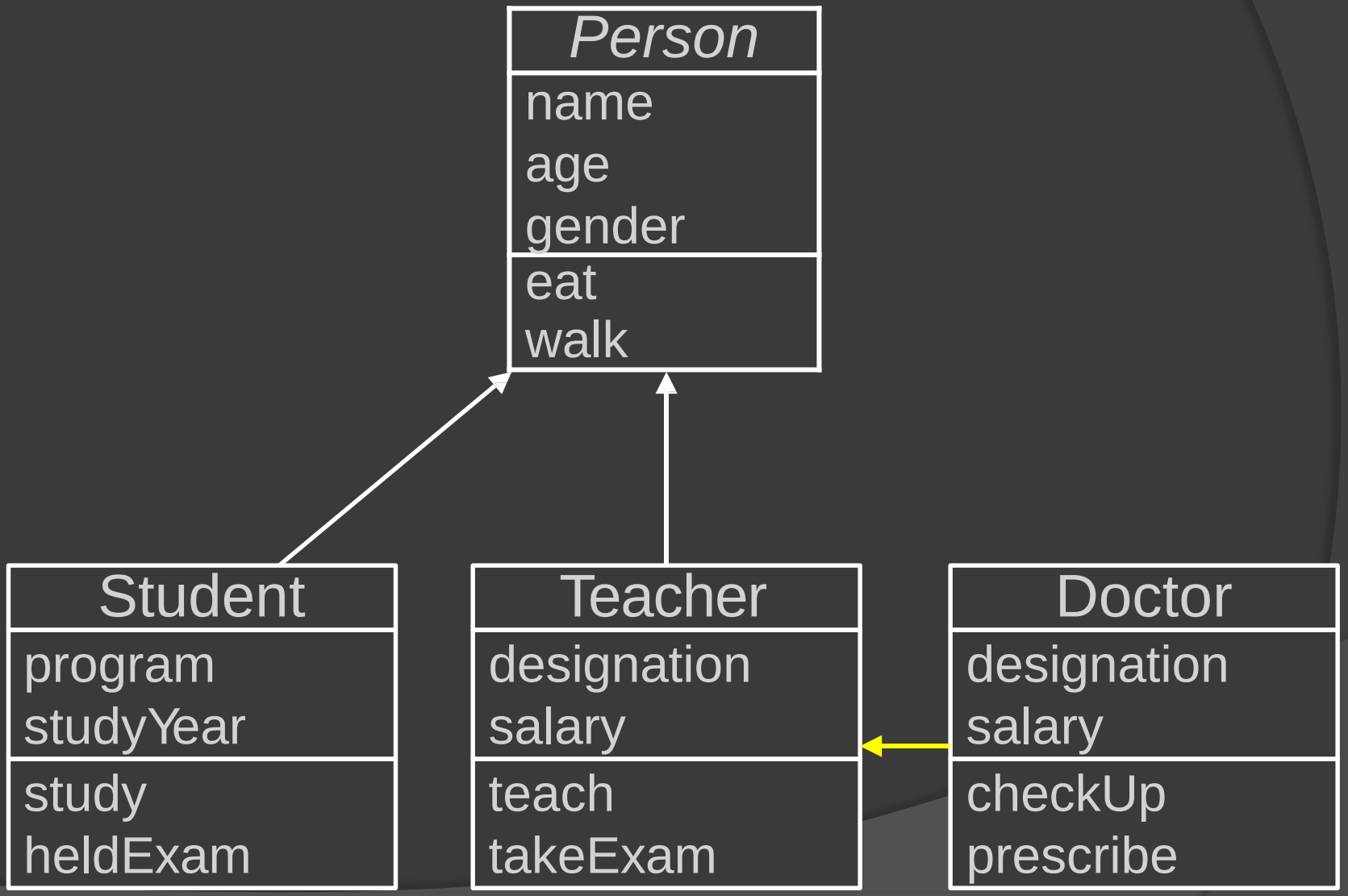
Reuse with Inheritance

- Main purpose of inheritance is reuse
- We can easily add new classes by inheriting from existing classes
  - Select an existing class closer to the desired functionality
  - Create a new class and inherit it from the selected class
  - Add to and/or modify the inherited functionality

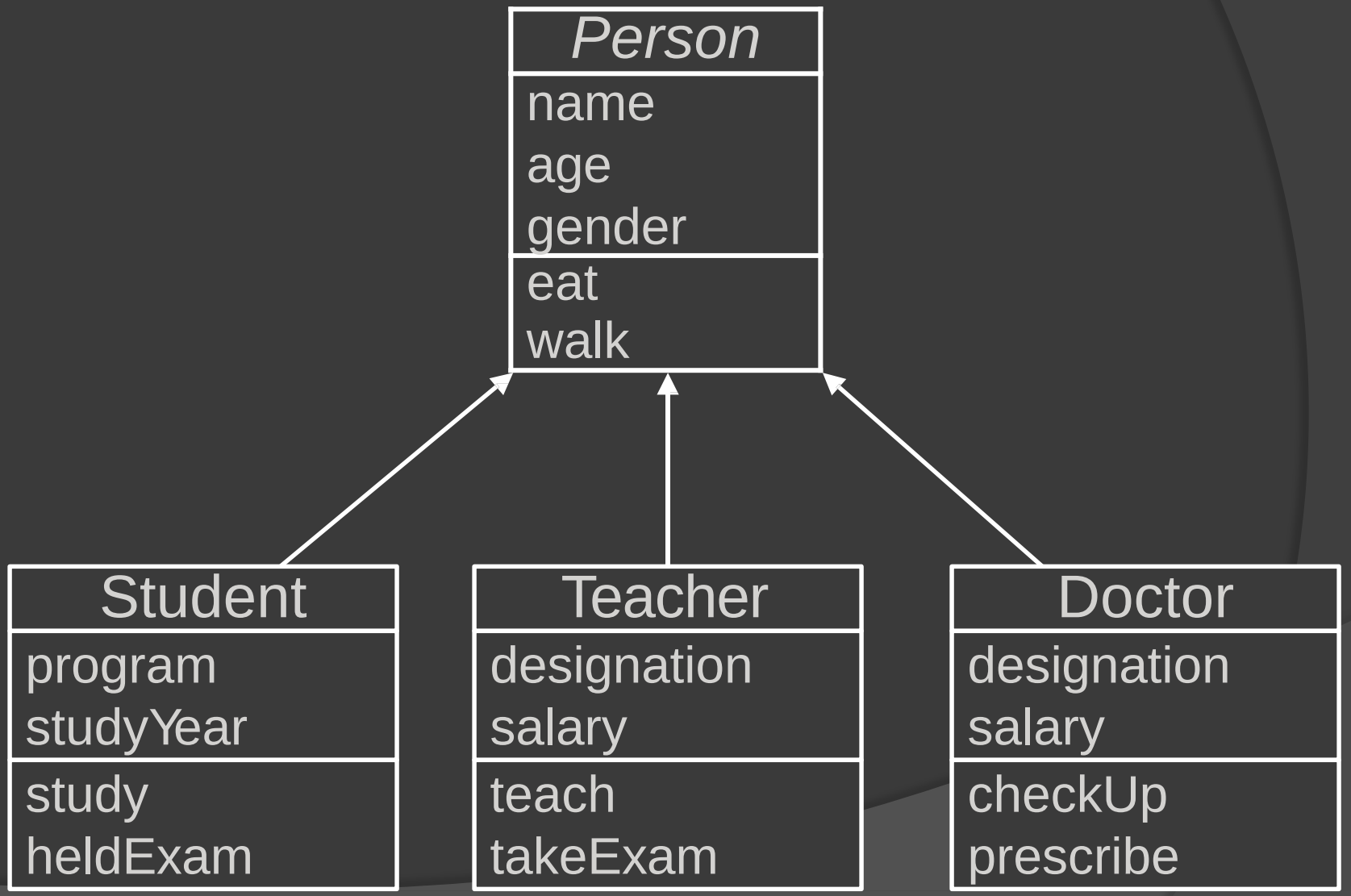
Example Reuse



Example Reuse



Example Reuse



Recap-Inheritance

- Derived class inherits all the characteristics of the base class
- Besides inherited characteristics, derived class may have its own unique characteristics
- Major benefit of inheritance is reuse

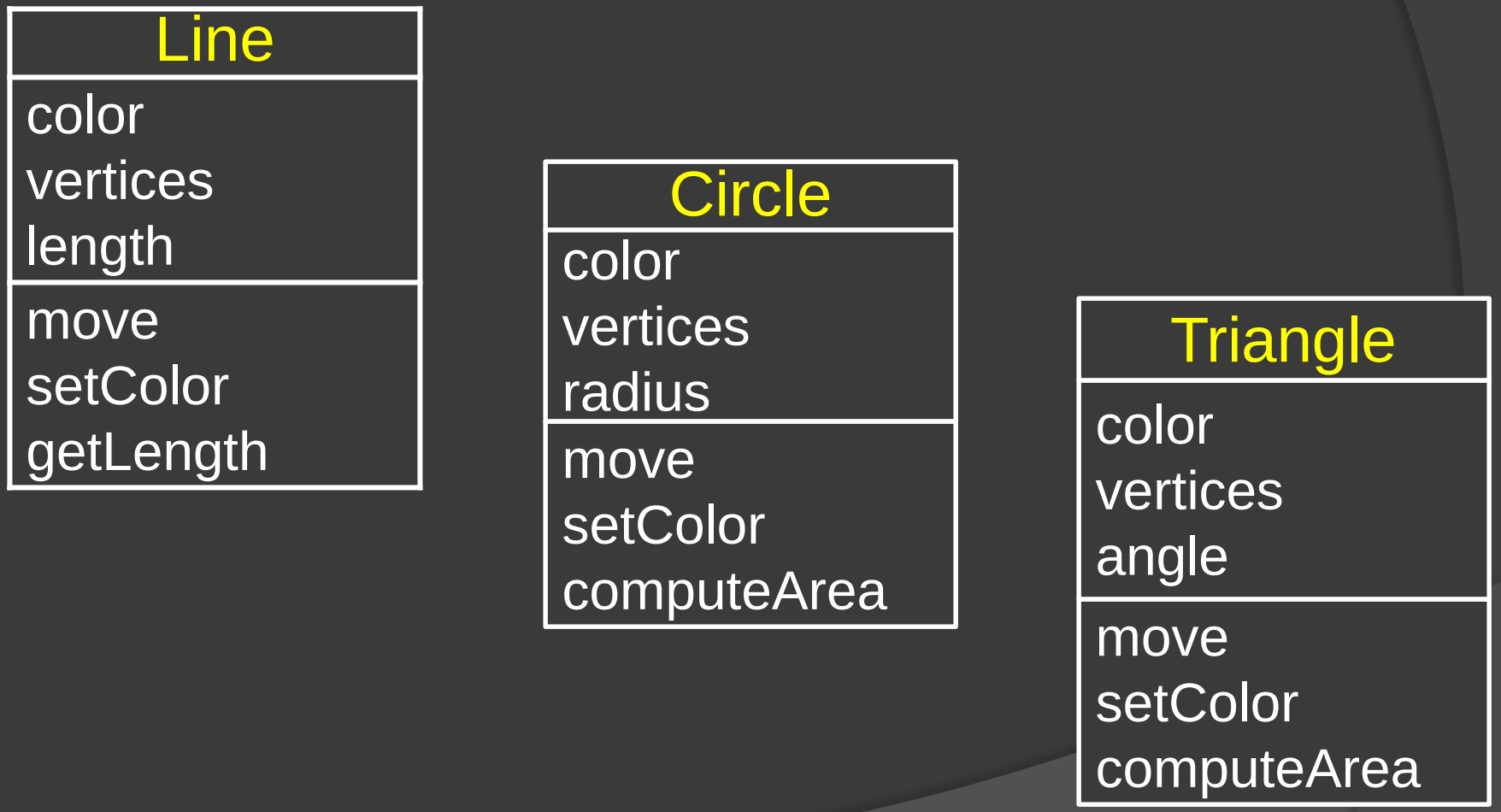
Concepts Related with Inheritance

- Generalization
- Subtyping (extension)
- Specialization (restriction)

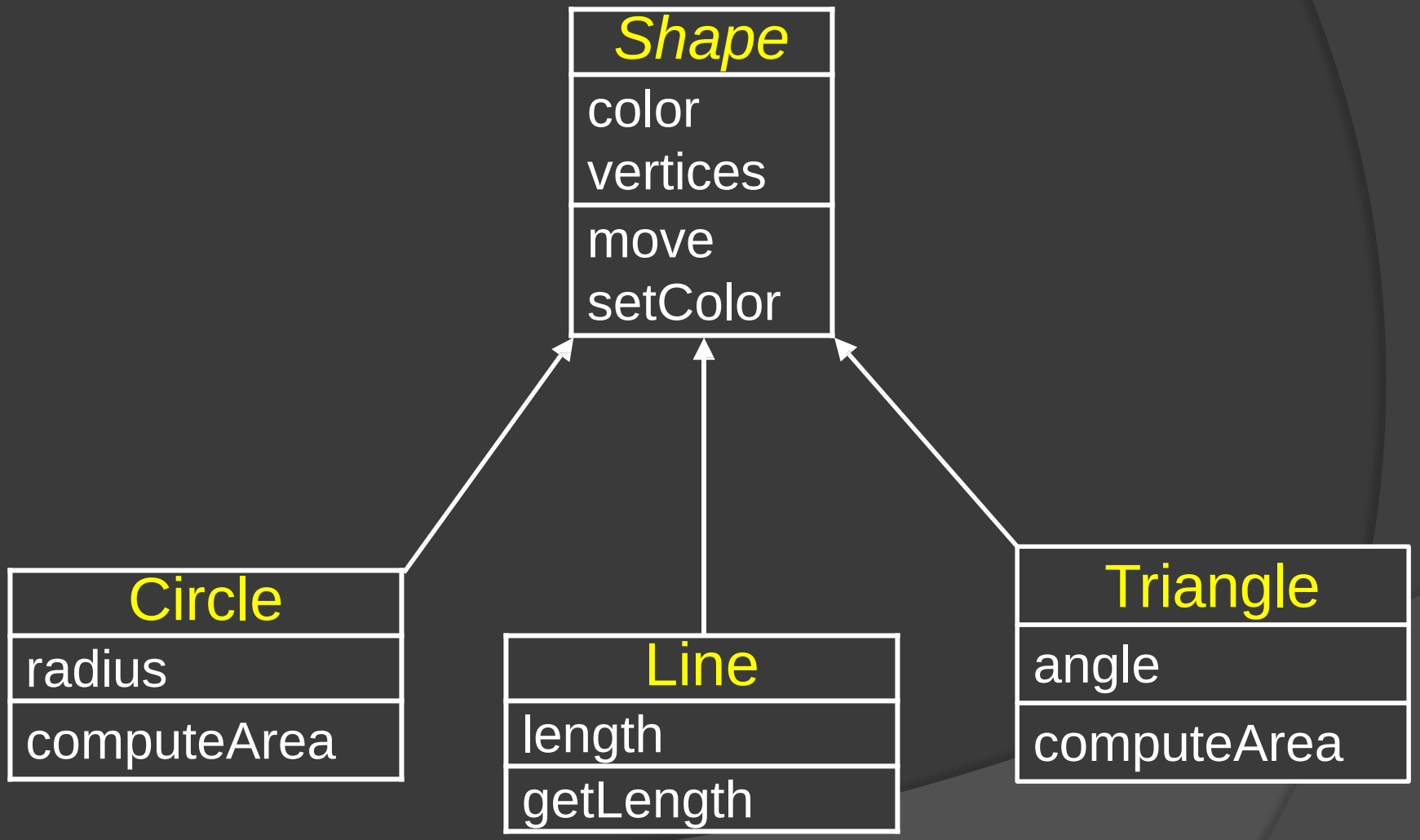
Generalization

- In OO models, some classes may have common characteristics
- We extract these features into a new class and inherit original classes from this new class
- This concept is known as Generalization

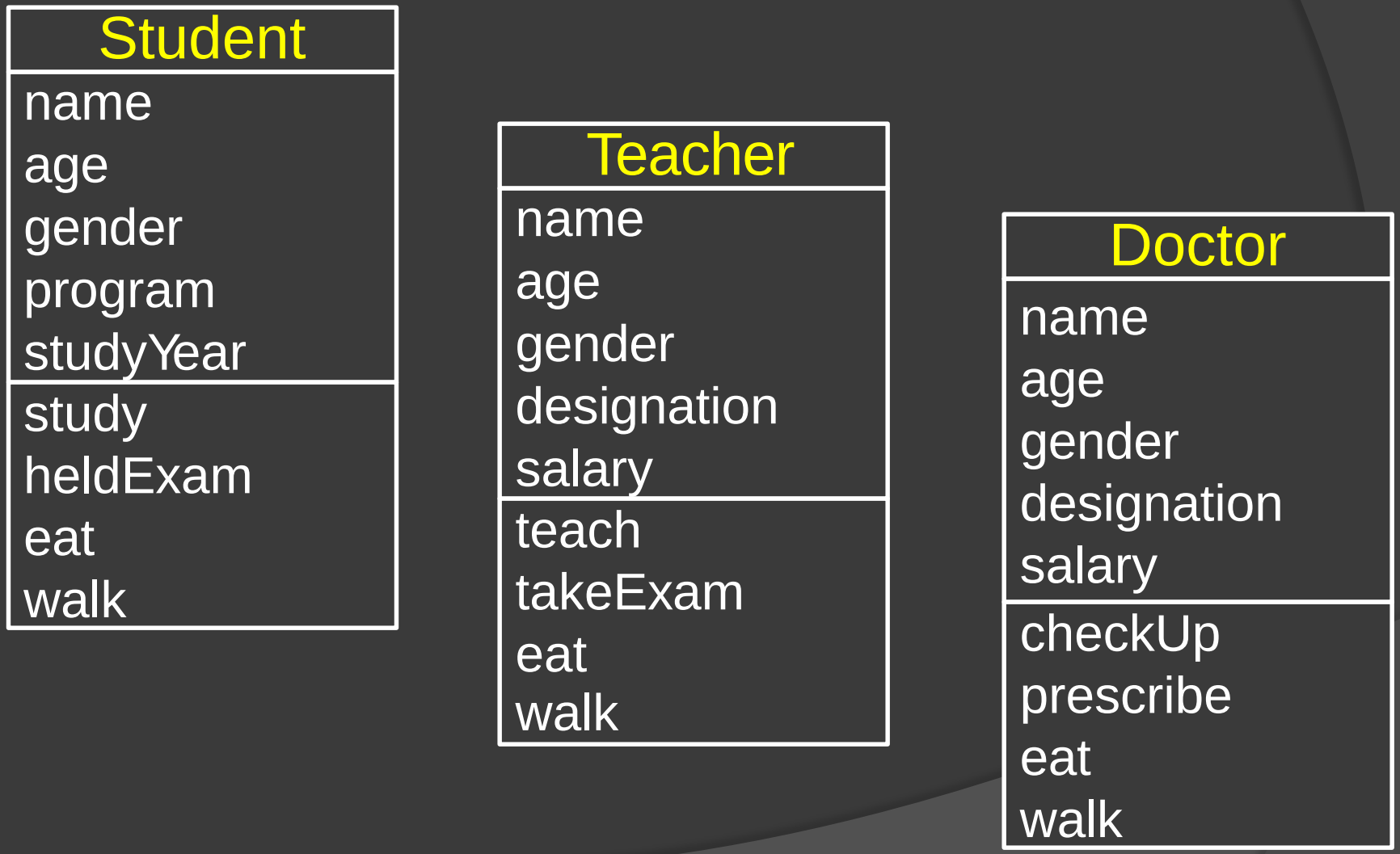
Example – Generalization



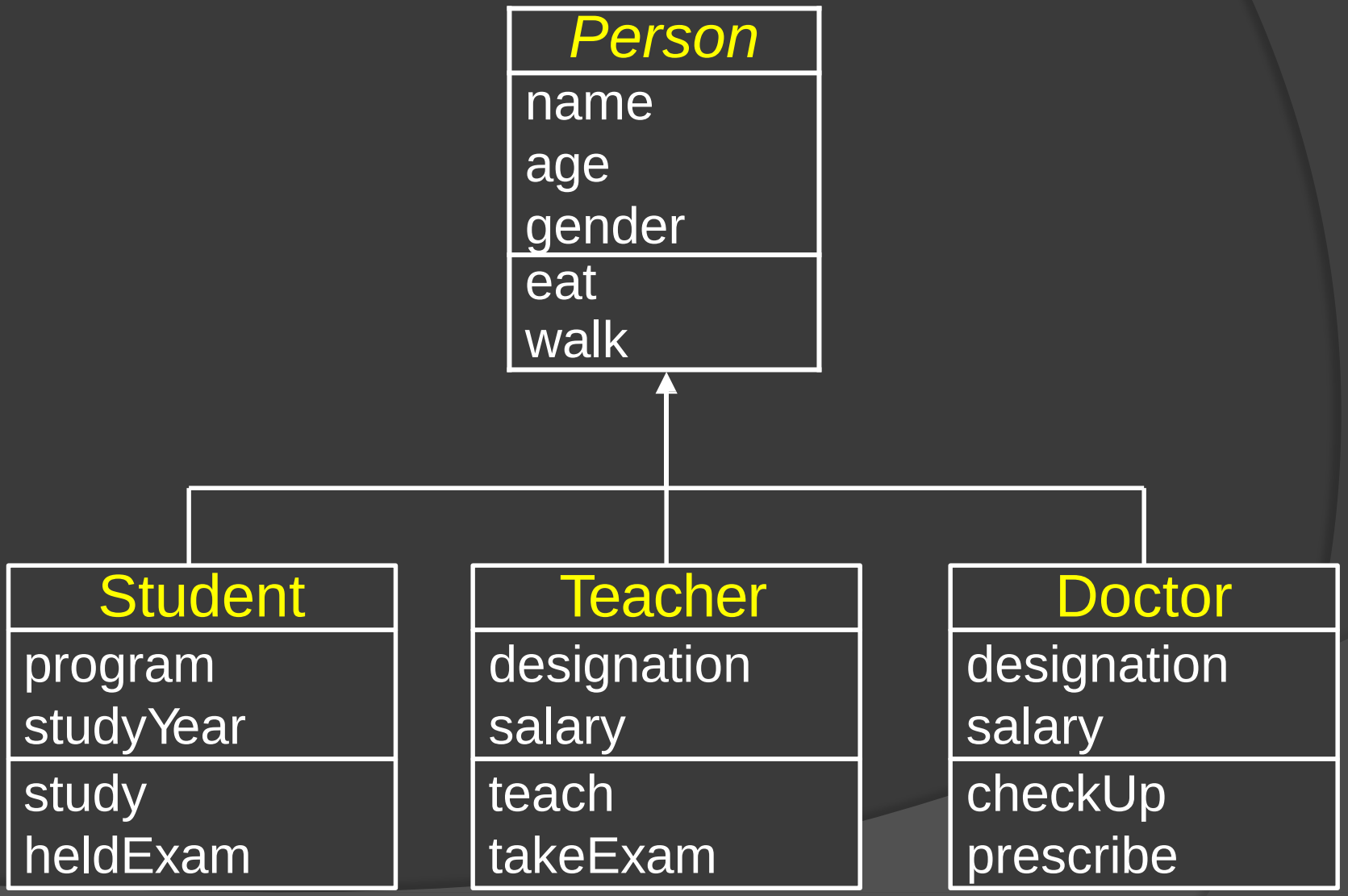
Example – Generalization



Example – Generalization



Example – Generalization



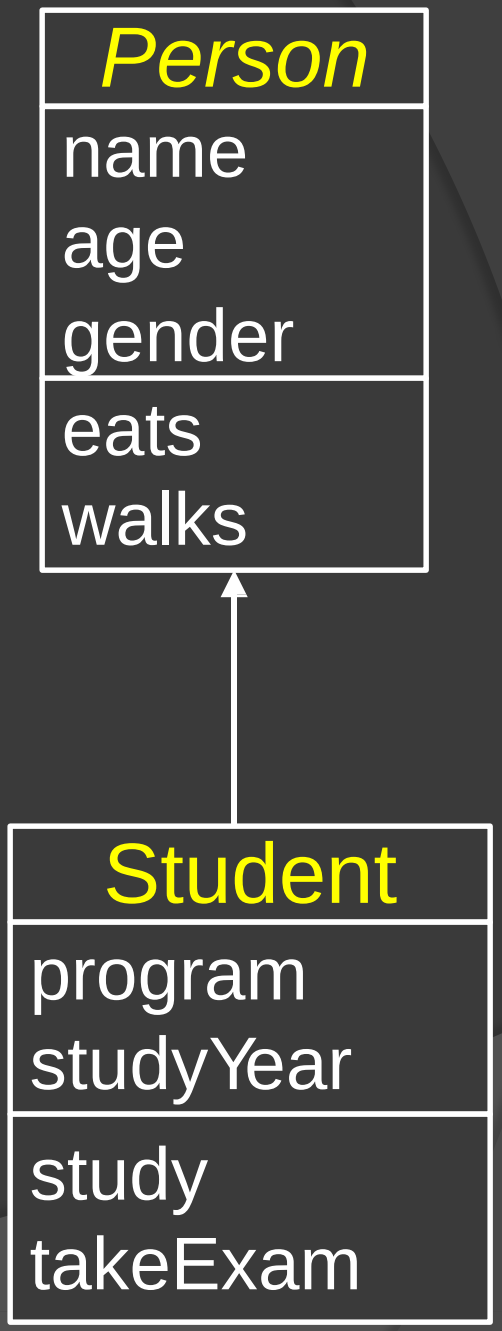
Sub-typing & Specialization

- We want to add a new class to an existing model
- Find an existing class that already implements some of the desired state and behaviour
- Inherit the new class from this class and add unique behaviour to the new class

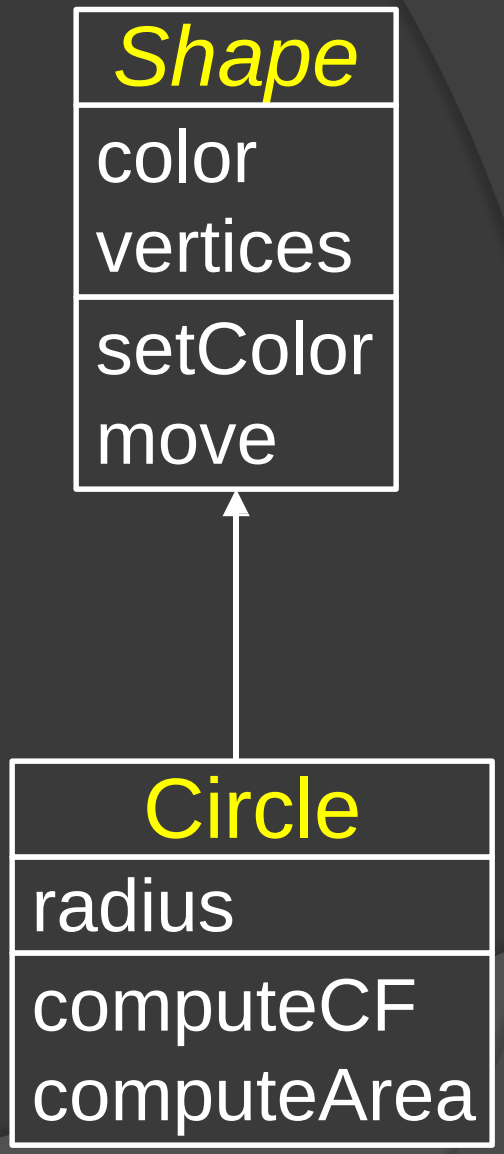
Sub-typing (Extension)

- Sub-typing means that derived class is behaviourally compatible with the base class
- Behaviourally compatible means that base class can be replaced by the derived class

Example – Sub-typing (Extension)



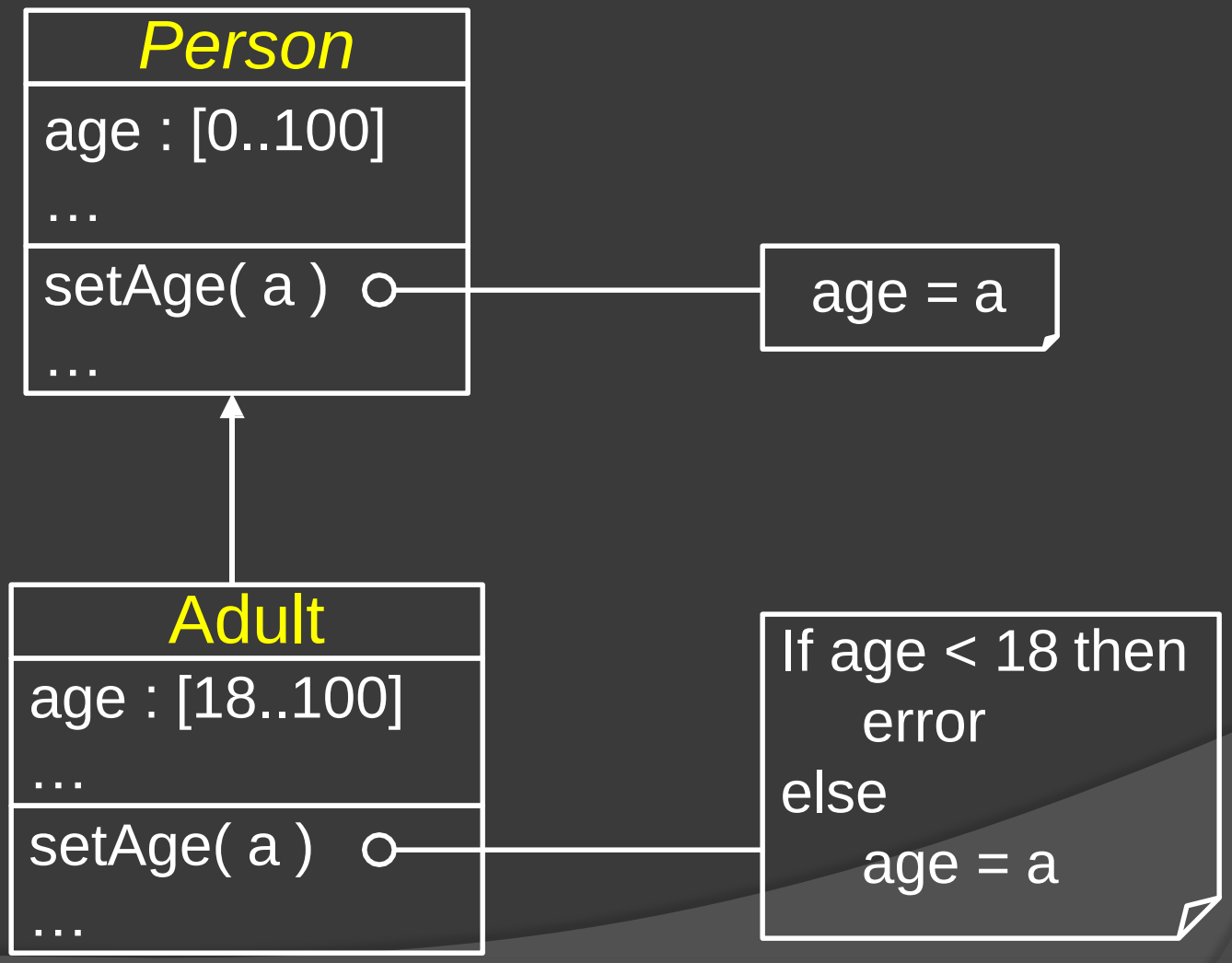
Example – Sub-typing (Extension)



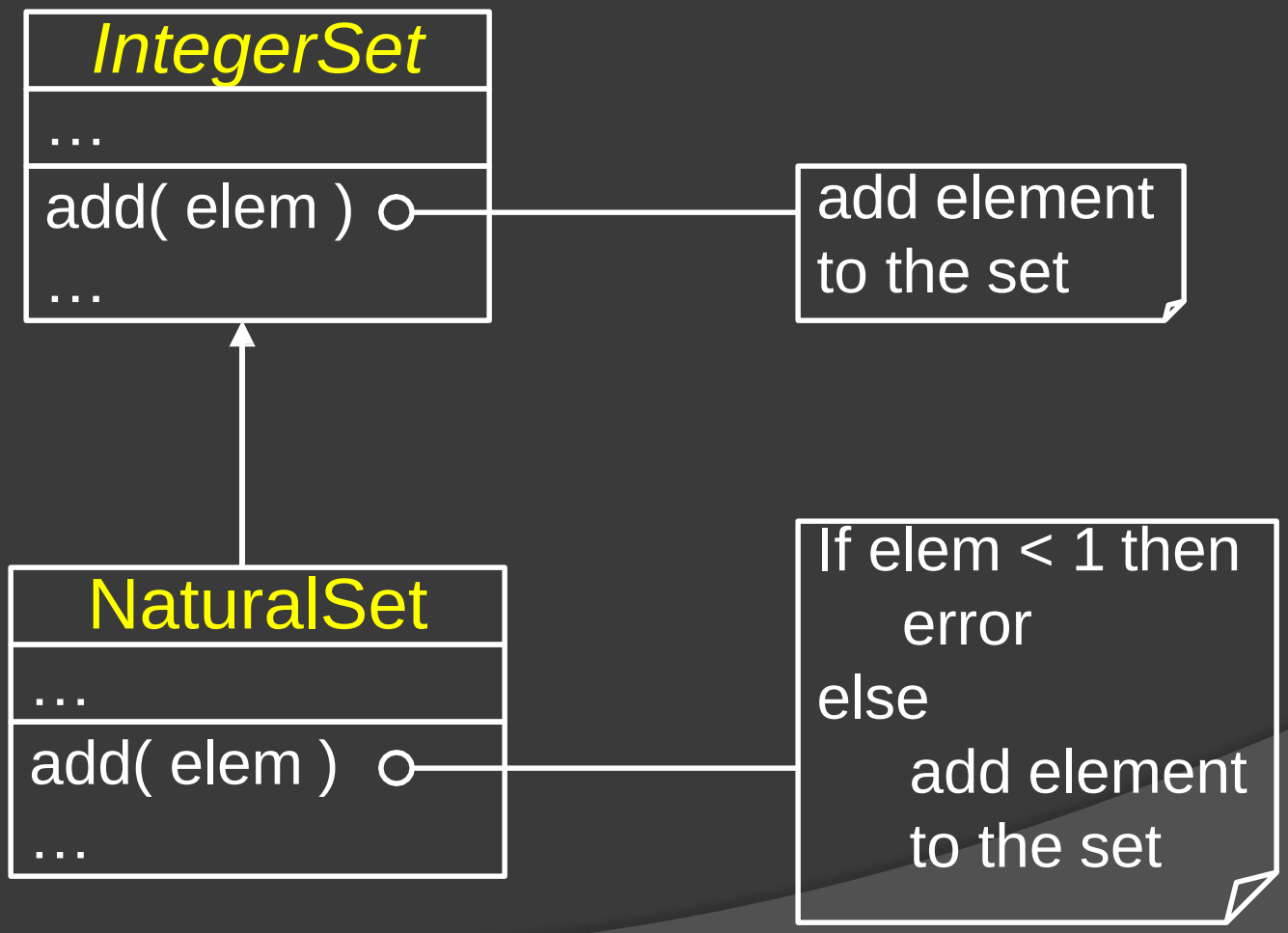
Specialization (Restriction)

- Specialization means that derived class is behaviourally incompatible with the base class
- Behaviourally incompatible means that base class can’t always be replaced by the derived class

Example – Specialization (Restriction)



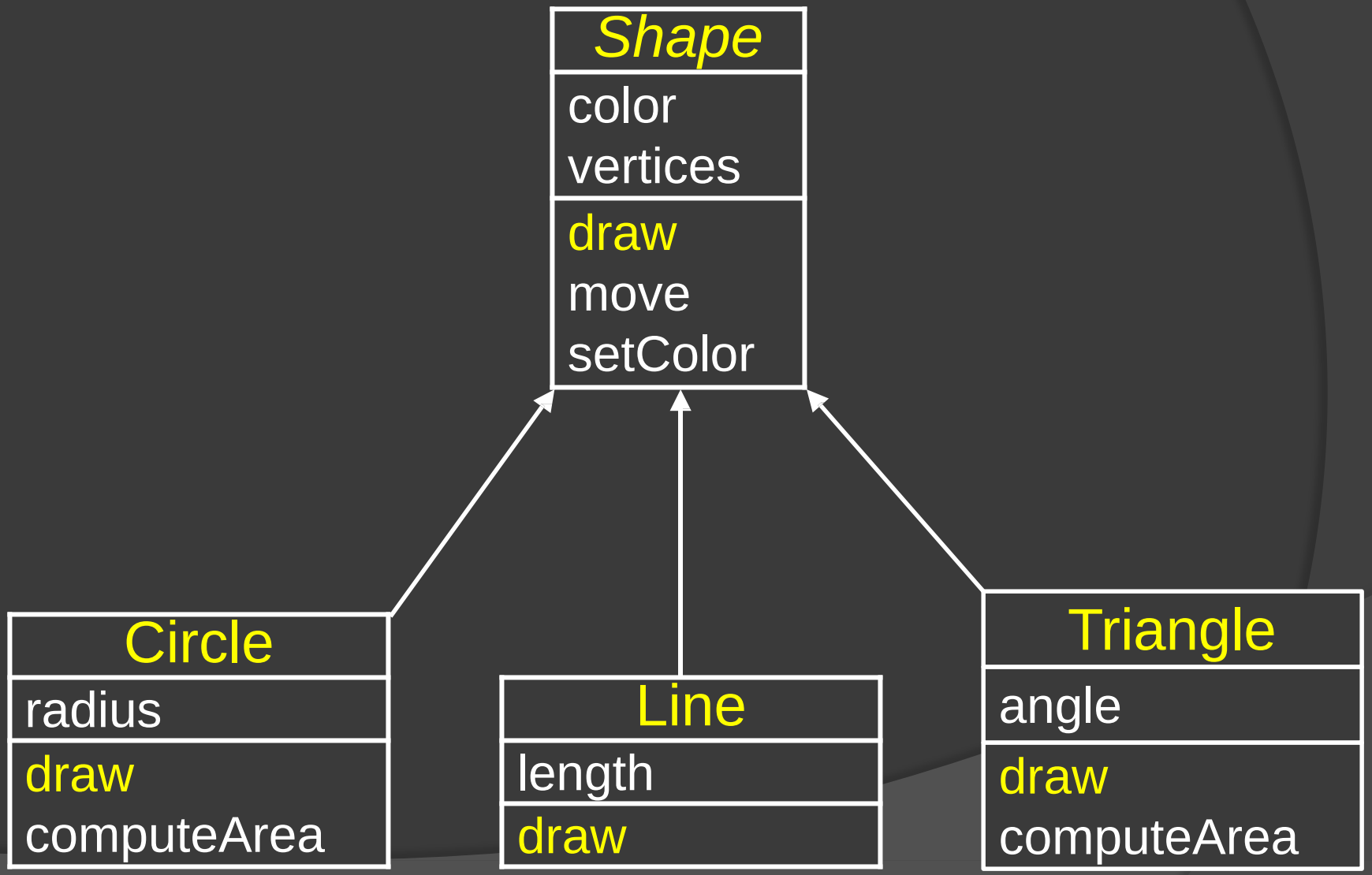
Example – Specialization (Restriction)



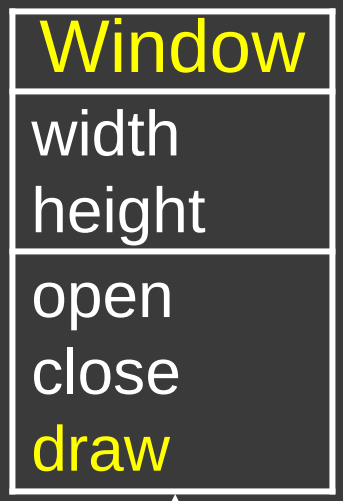
Overriding

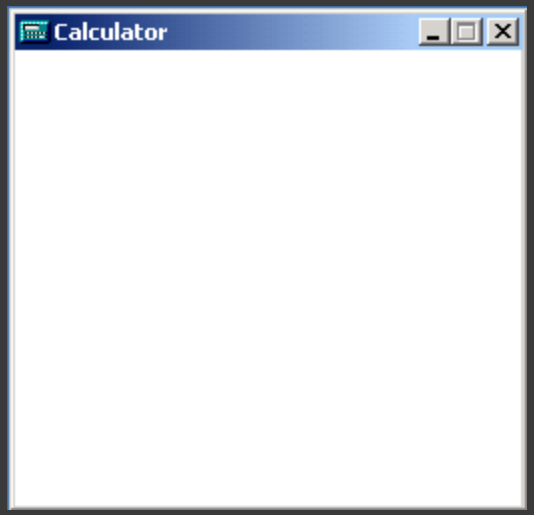
- A class may need to override the default behavior provided by its base class
- Reasons for overriding
  - Provide behavior specific to a derived class
  - Extend the default behavior
  - Restrict the default behavior
  - Improve performance

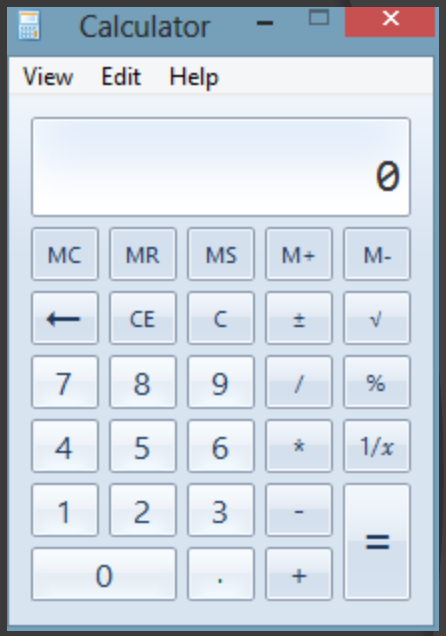
Example – Specific Behaviour

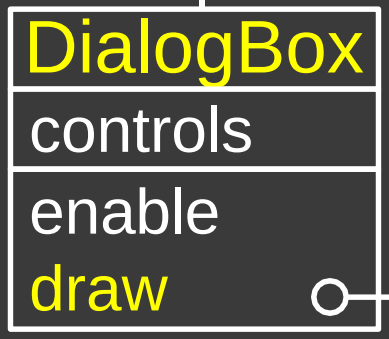


Example – Extension



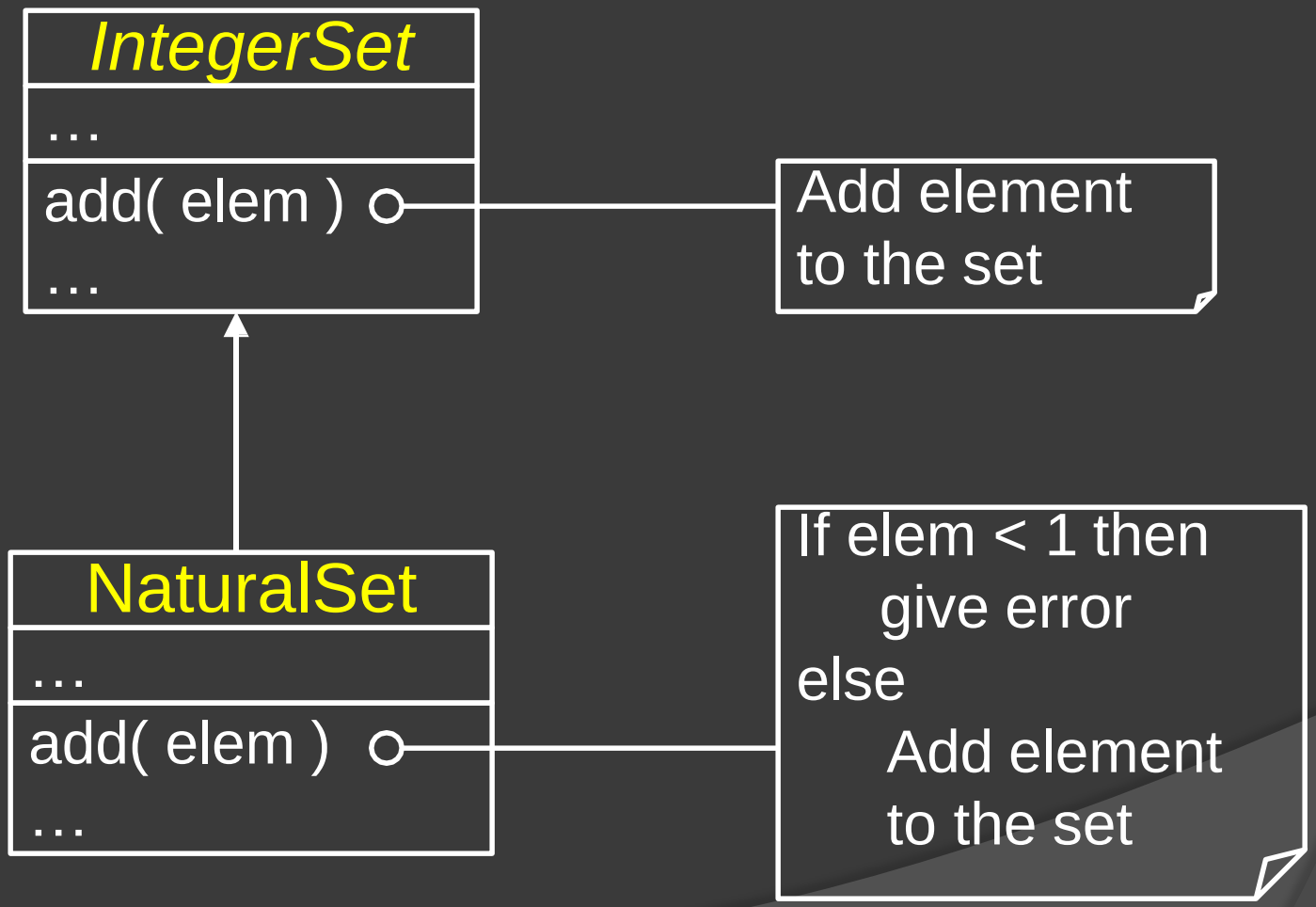






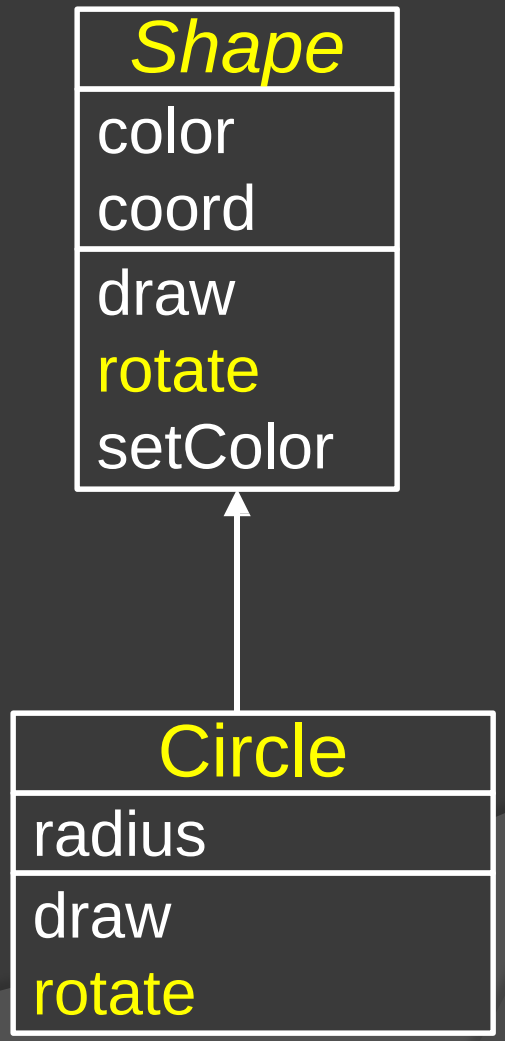
1Invoke Window's draw
2draw the dialog box

Example – Restriction



Example – Improve Performance

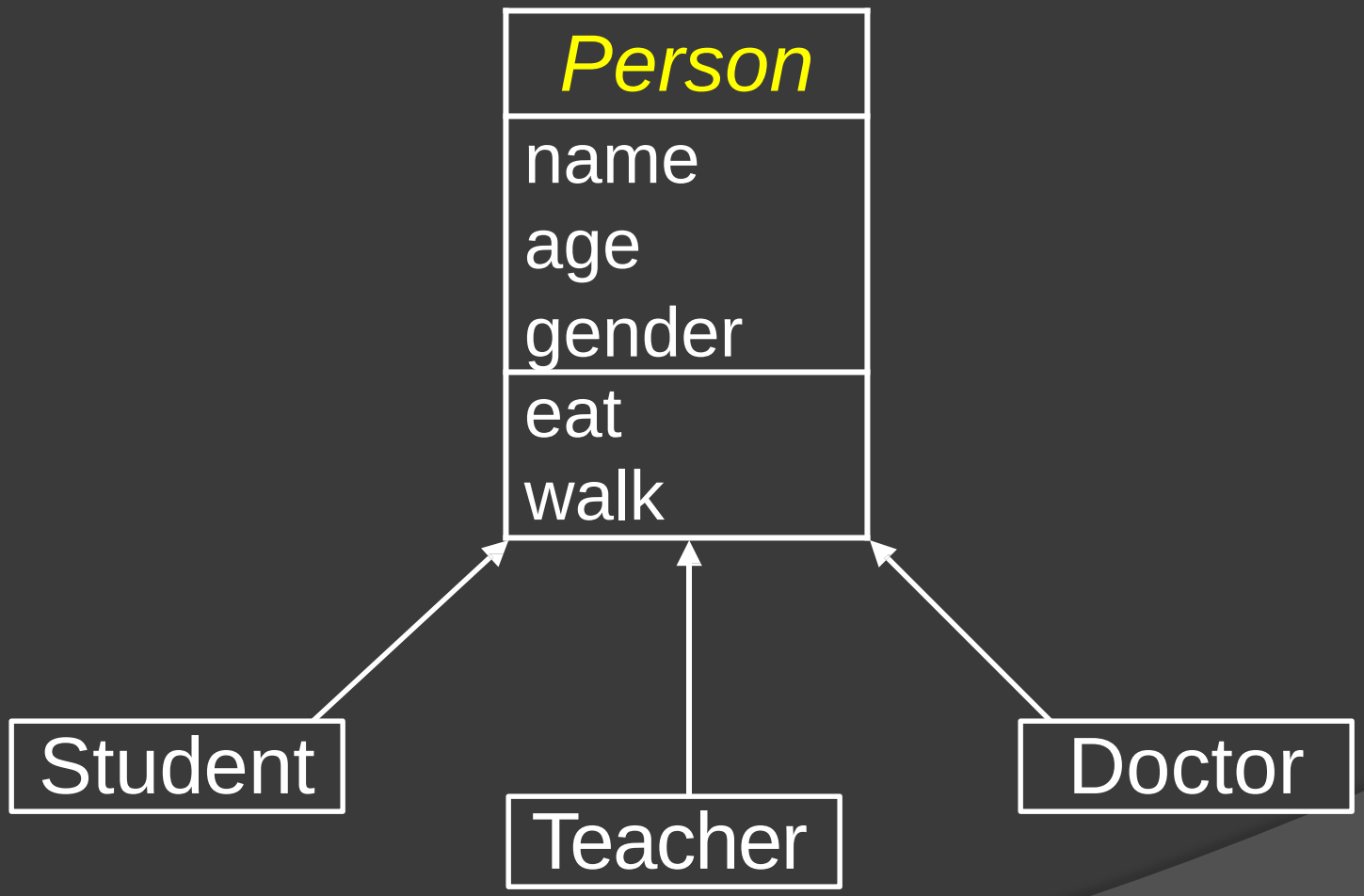
- Class Circle overrides *rotate* operation of class Shape with a Null operation.



Abstract Classes

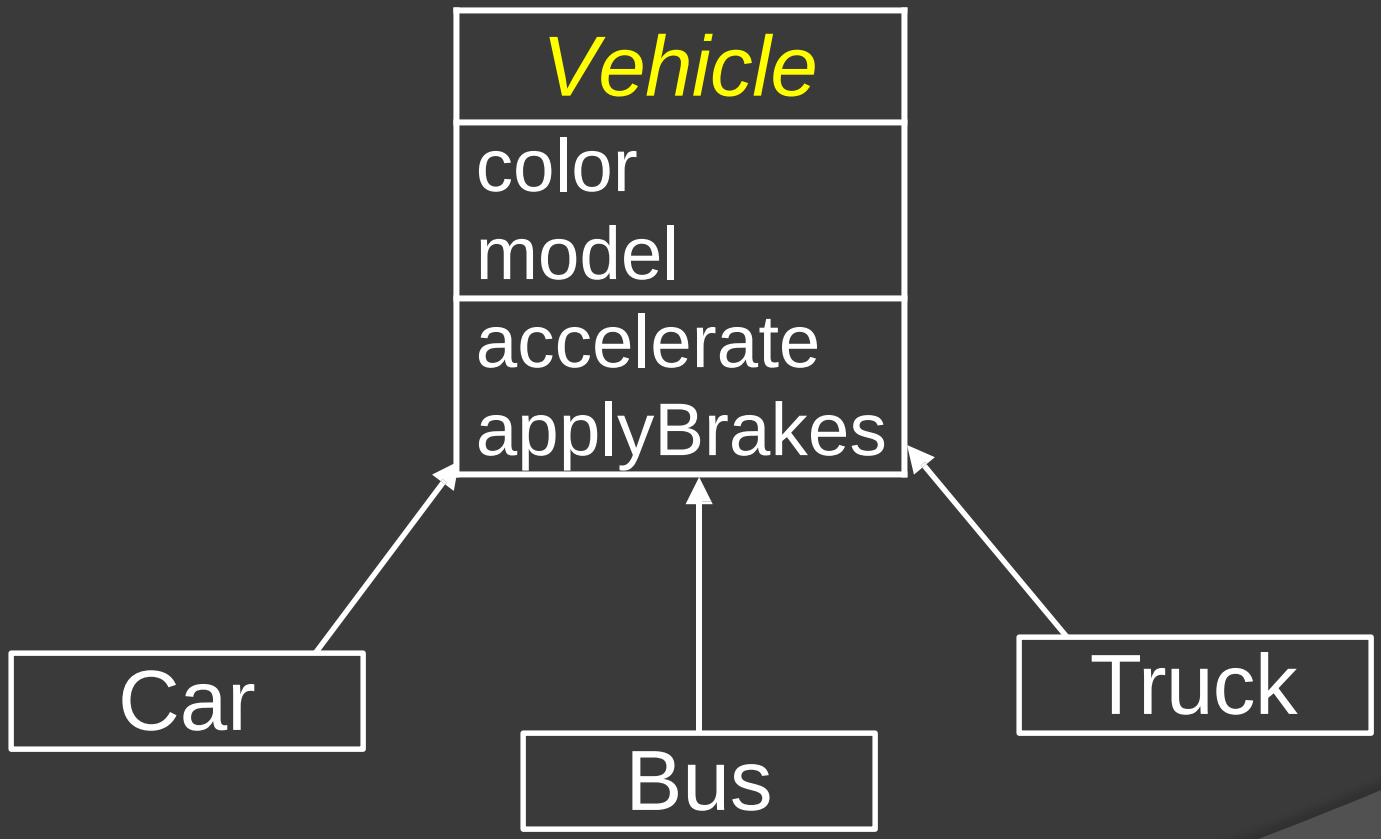
- An abstract class implements an abstract concept
- Main purpose is to be inherited by other classes
- Can’t be instantiated
- Promotes reuse

Example – Abstract Classes



- Here, Person is an abstract class

Example – Abstract Classes

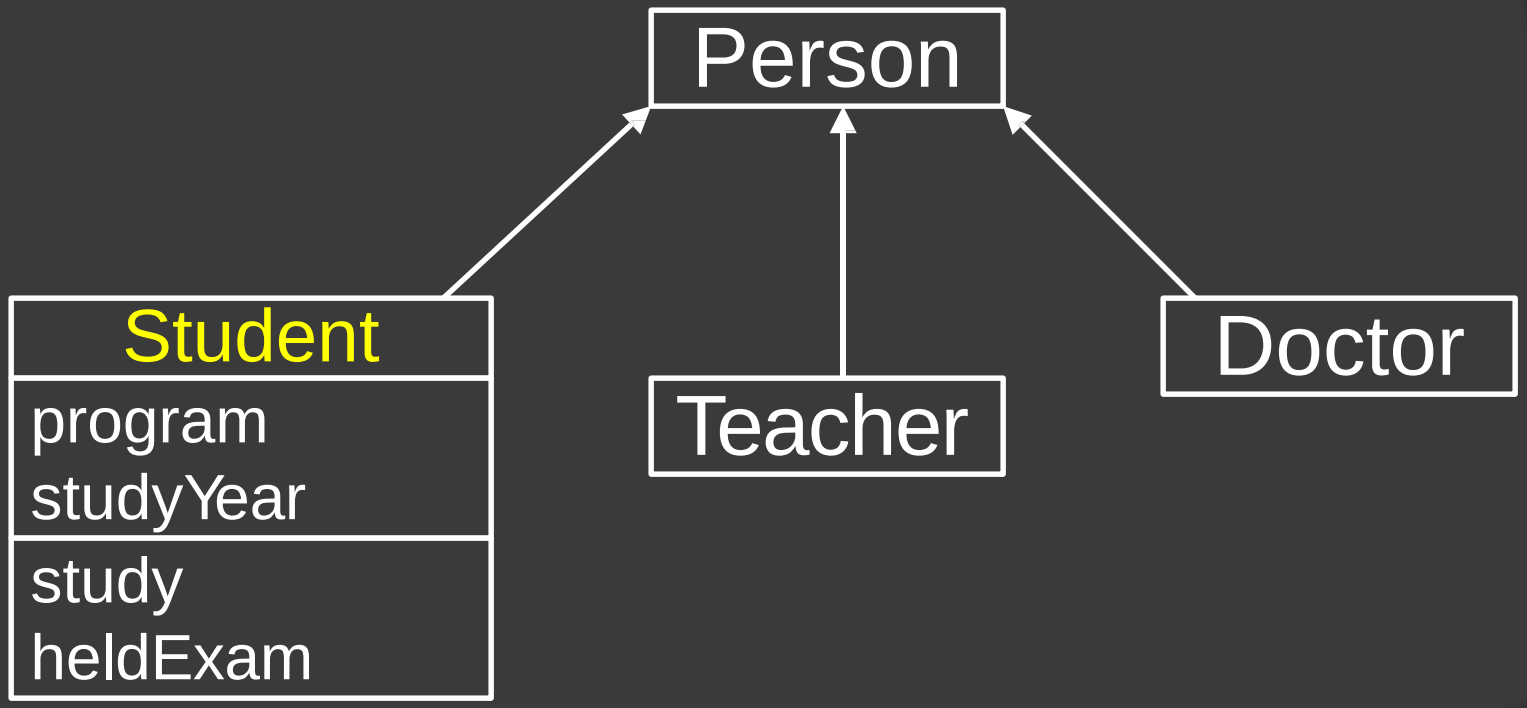


- Here, Vehicle is an abstract class

Concrete Classes

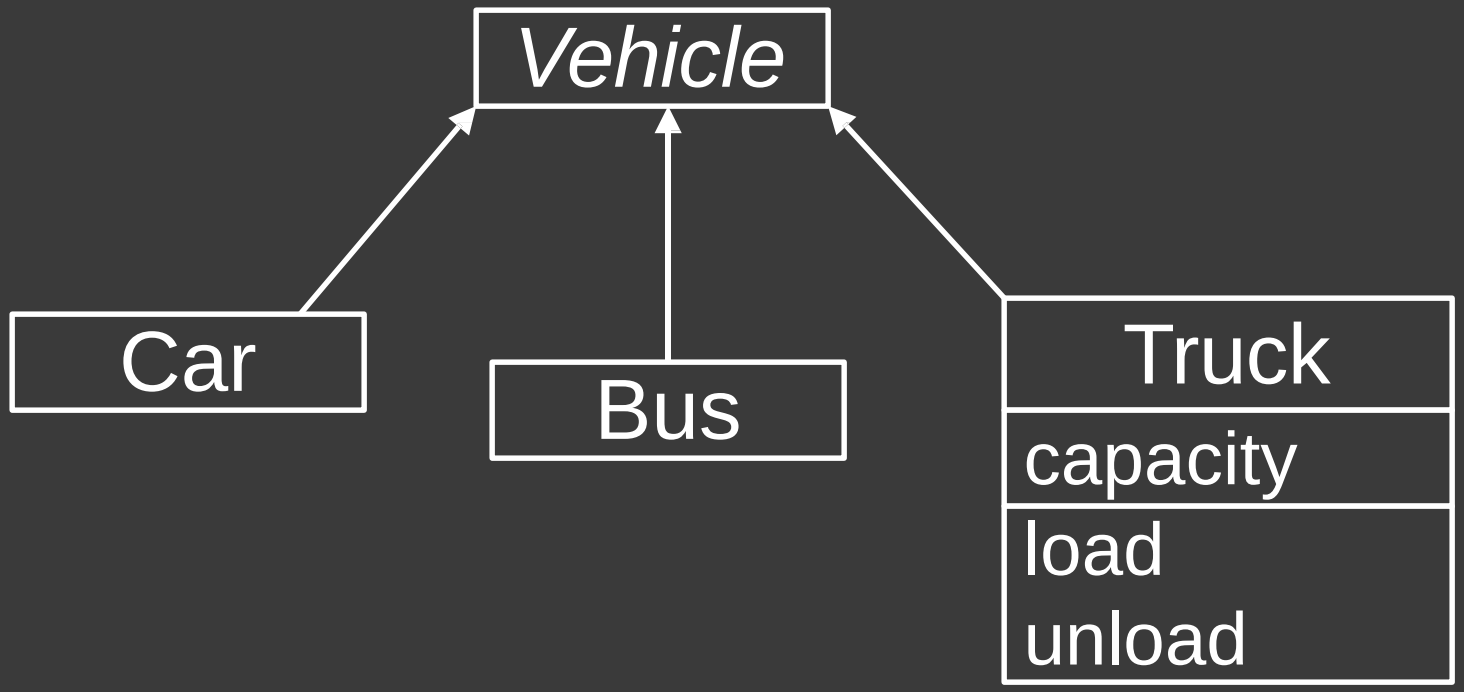
- A concrete class implements a concrete concept
- Main purpose is to be instantiated
- Provides implementation details specific to the domain context

Example – Concrete Classes



- Here, Student, Teacher and Doctor are concrete classes

Example – Concrete Classes



- Here, Car, Bus and Truck are concrete classes

Multiple Inheritance

[Not Supported by Java or C#]

- We may want to reuse characteristics of more than one parent class

Example – Multiple Inheritance

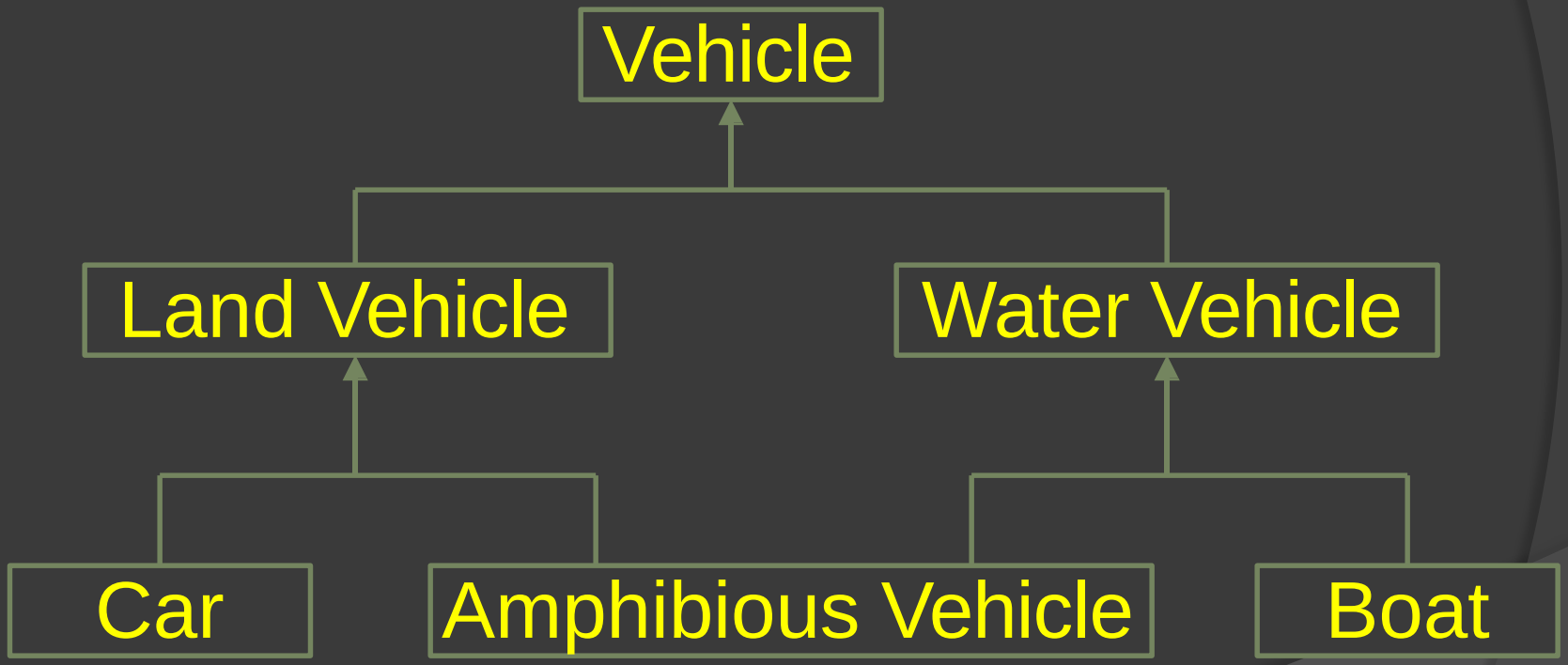
Mermaid

Example – Multiple Inheritance

Example – Multiple Inheritance

Amphibious Vehicle

Example – Multiple Inheritance



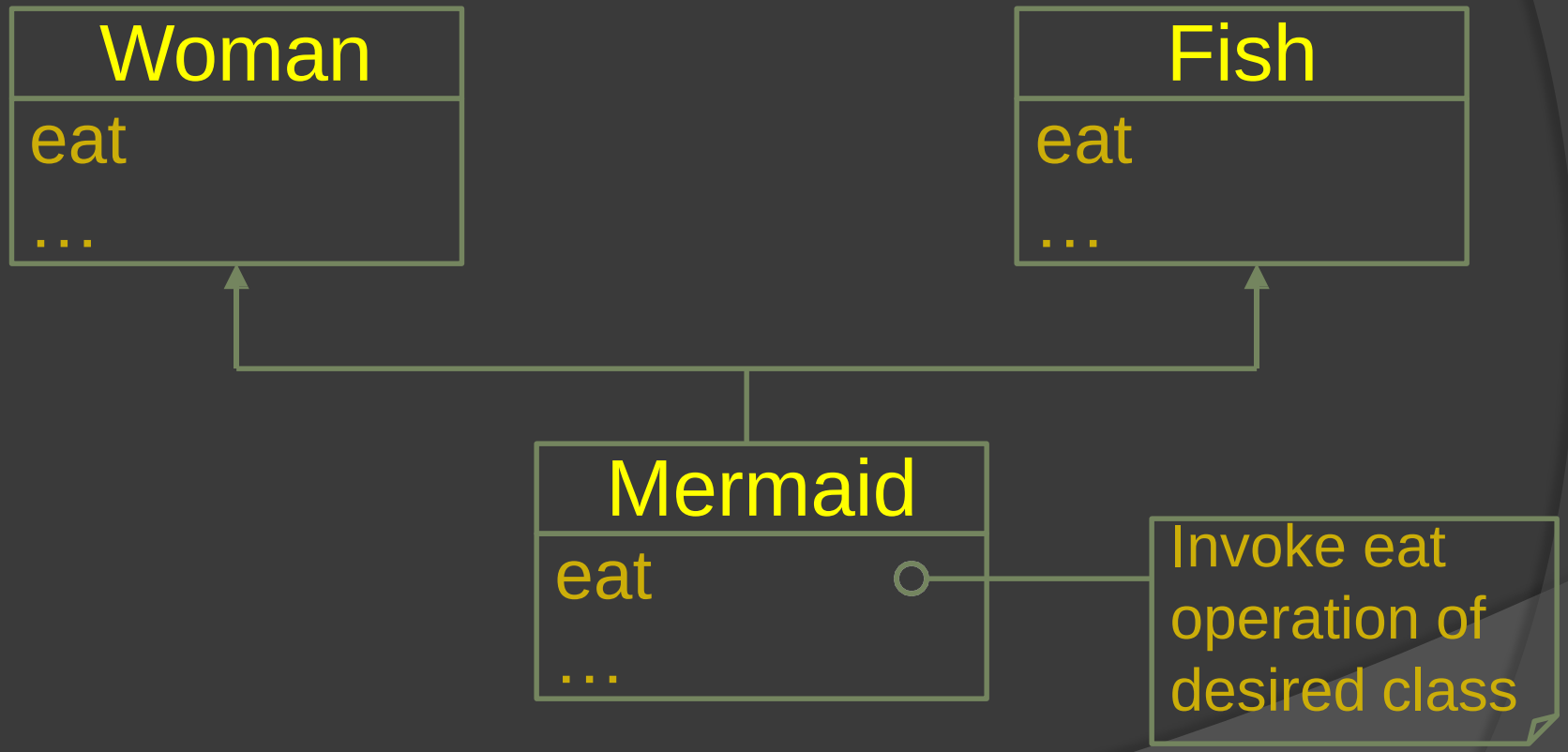
Problems with Multiple Inheritance

- Increased complexity
- Reduced understanding
- Duplicate features

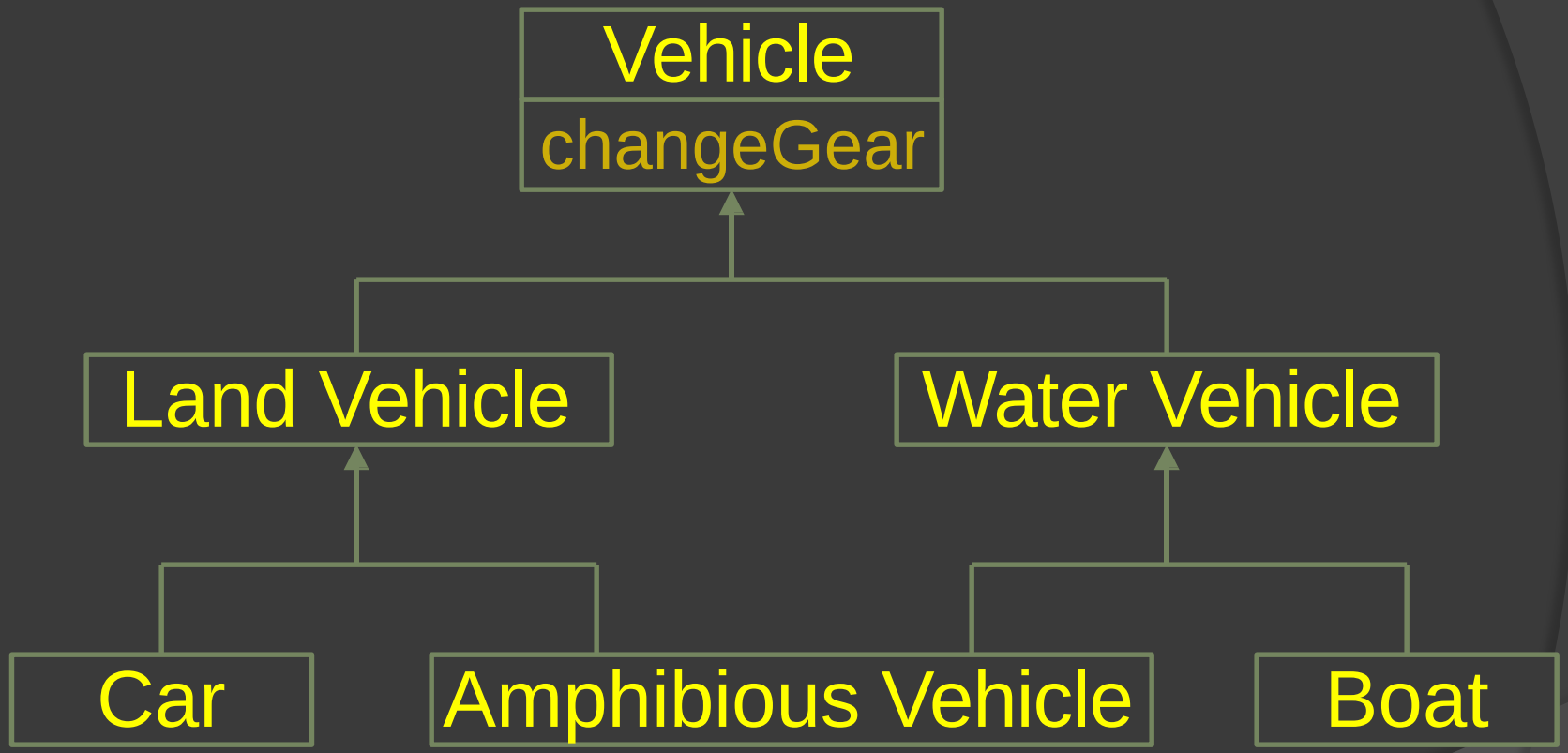
Problem – Duplicate Features

- Which *eat* operation *Mermaid* inherits?

Solution – Override the Common Feature



Problem – Duplicate Features (Diamond Problem)



- Which *changeGear* operation Amphibious Vehicle inherits?

Solution to Diamond Problem

- Some languages disallow diamond hierarchy
- Others provide mechanism to ignore characteristics from one side

Association

- Objects in an object model interact with each other
- Usually an object provides services to several other objects
- An object keeps associations with other objects to delegate tasks

Kinds of Association

- Class Association
  - Inheritance
- Object Association
  - Simple Association
  - Composition
  - Aggregation

Simple Association

- Is the weakest link between objects
- Is a reference by which one object can interact with some other object
- Is simply called as “association”

Kinds of Simple Association

- w.r.t navigation
  - One-way Association
  - Two-way Association
- w.r.t number of objects
  - Binary Association
  - Ternary Association
  - N-ary Association

One-way Association

- We can navigate along a single direction only
- Denoted by an arrow towards the server object

Example – Association

- Ali lives in a House

Example – Association

- Ali drives his Car

Two-way Association

- We can navigate in both directions
- Denoted by a line between the associated objects

Example – Two-way Association

- Employee works for company
- Company employs employees

Example – Two-way Association

- Yasir is a friend of Ali
- Ali is a friend of Yasir

Binary Association

- Associates objects of exactly two classes
- Denoted by a line, or an arrow between the associated objects

Example – Binary Association

- Association “works-for” associates objects of exactly two classes

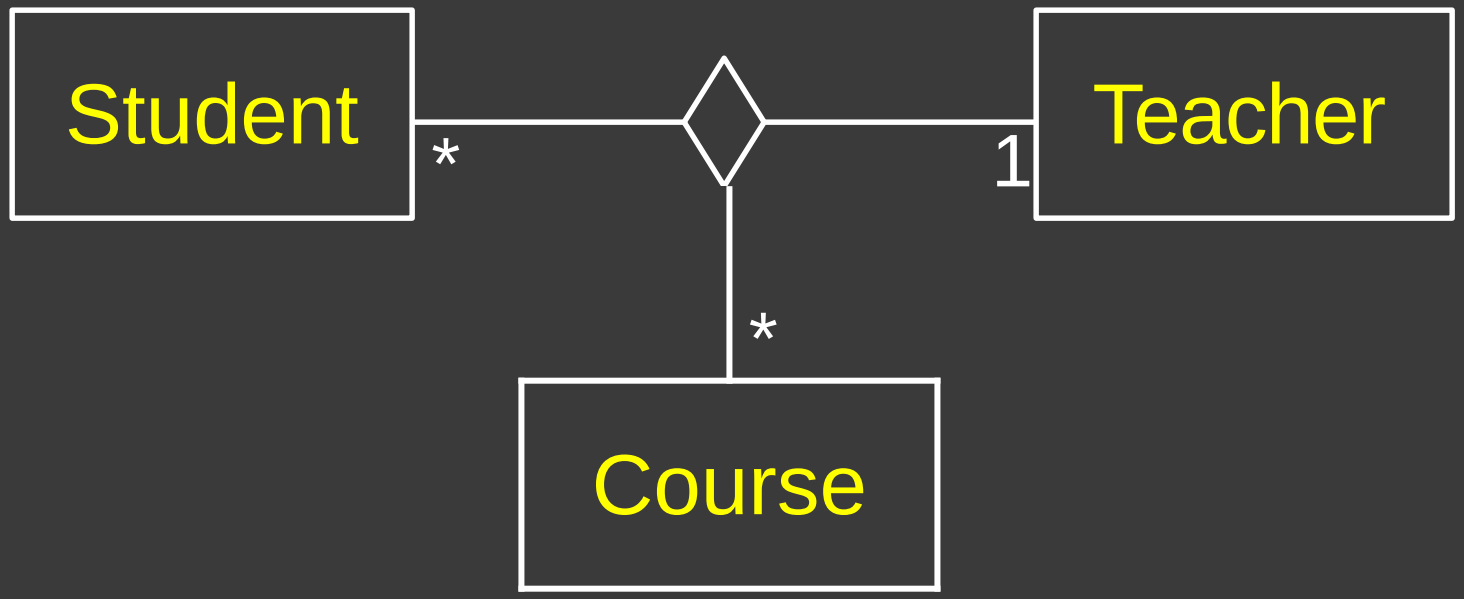
Example – Binary Association

- Association “drives” associates objects of exactly two classes

Ternary Association

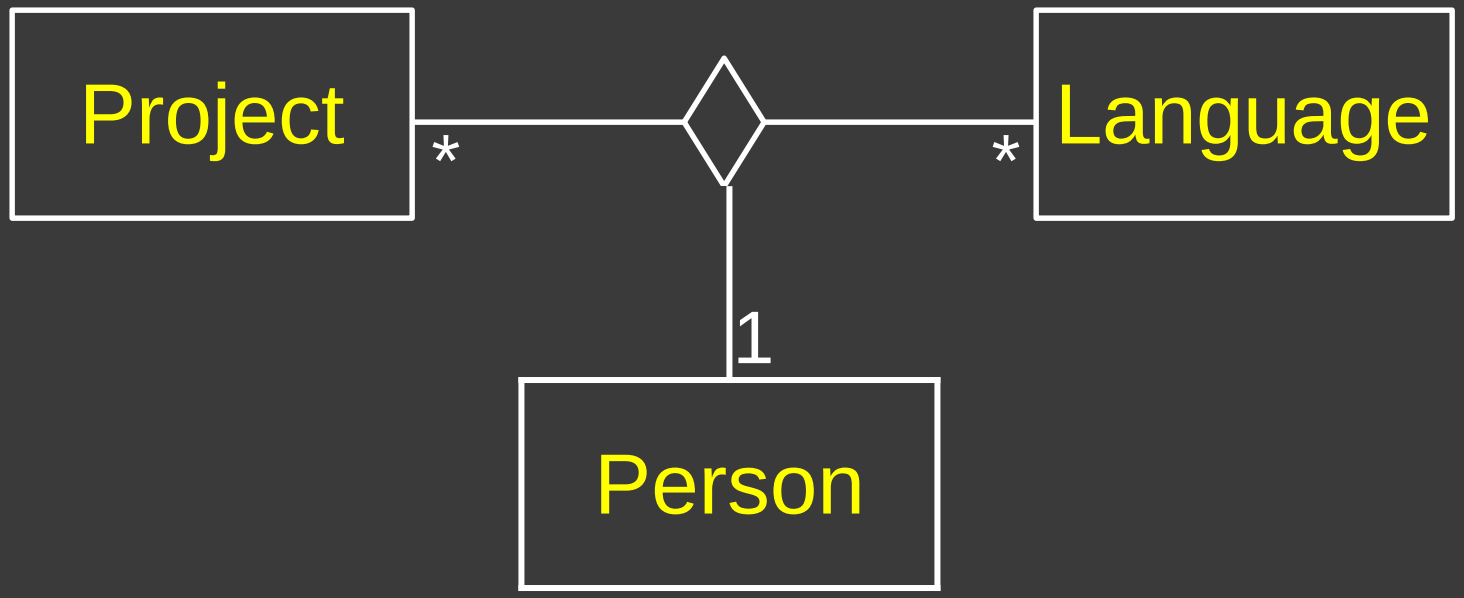
- Associates objects of exactly three classes
- Denoted by a diamond with lines connected to associated objects

Example – Ternary Association



- Objects of exactly three classes are associated

Example – Ternary Association



- Objects of exactly three classes are associated

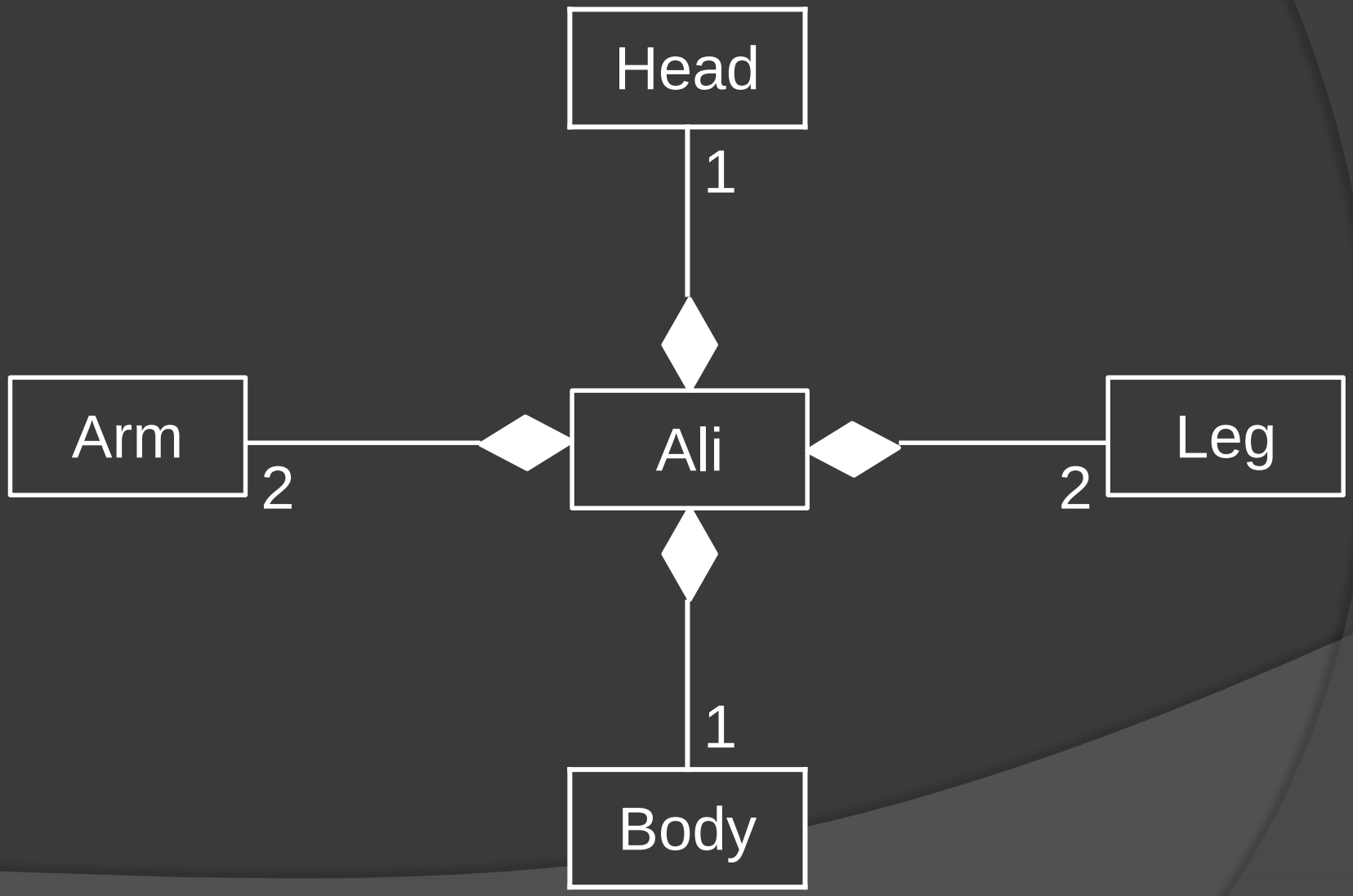
N-ary Association

- An association between 3 or more classes
- Practical examples are very rare

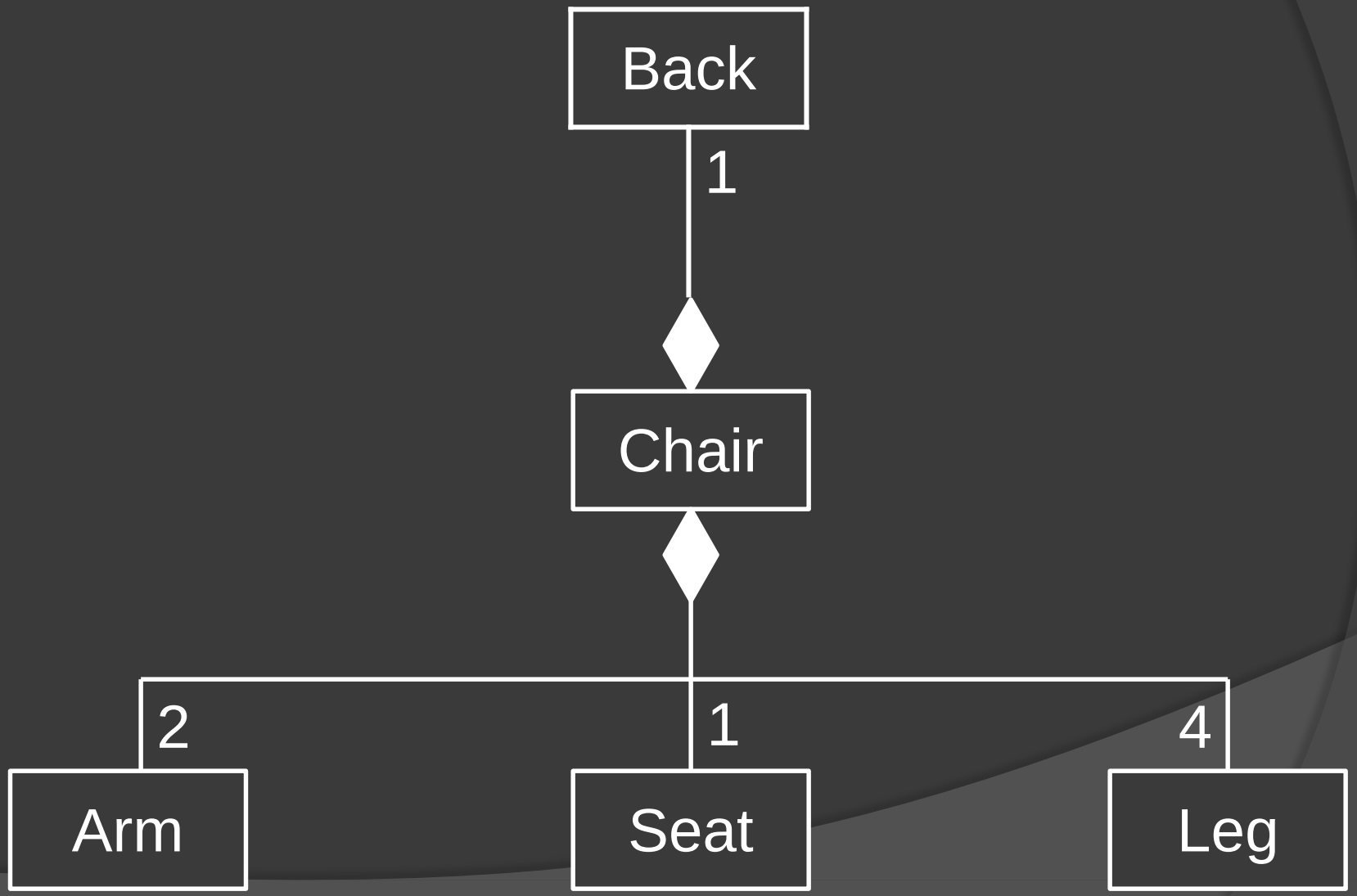
Composition

- An object may be composed of other smaller objects
- The relationship between the “part” objects and the “whole” object is known as Composition
- Composition is represented by a line with a filled-diamond head towards the composer object

Example – Composition of Ali



Example – Composition of Chair



Composition is Stronger

- Composition is a stronger relationship, because
  - Composed object becomes a part of the composer
  - Composed object can’t exist independently

Example – Composition is Stronger

- Ali is made up of different body parts
- They can't exist independent of Ali

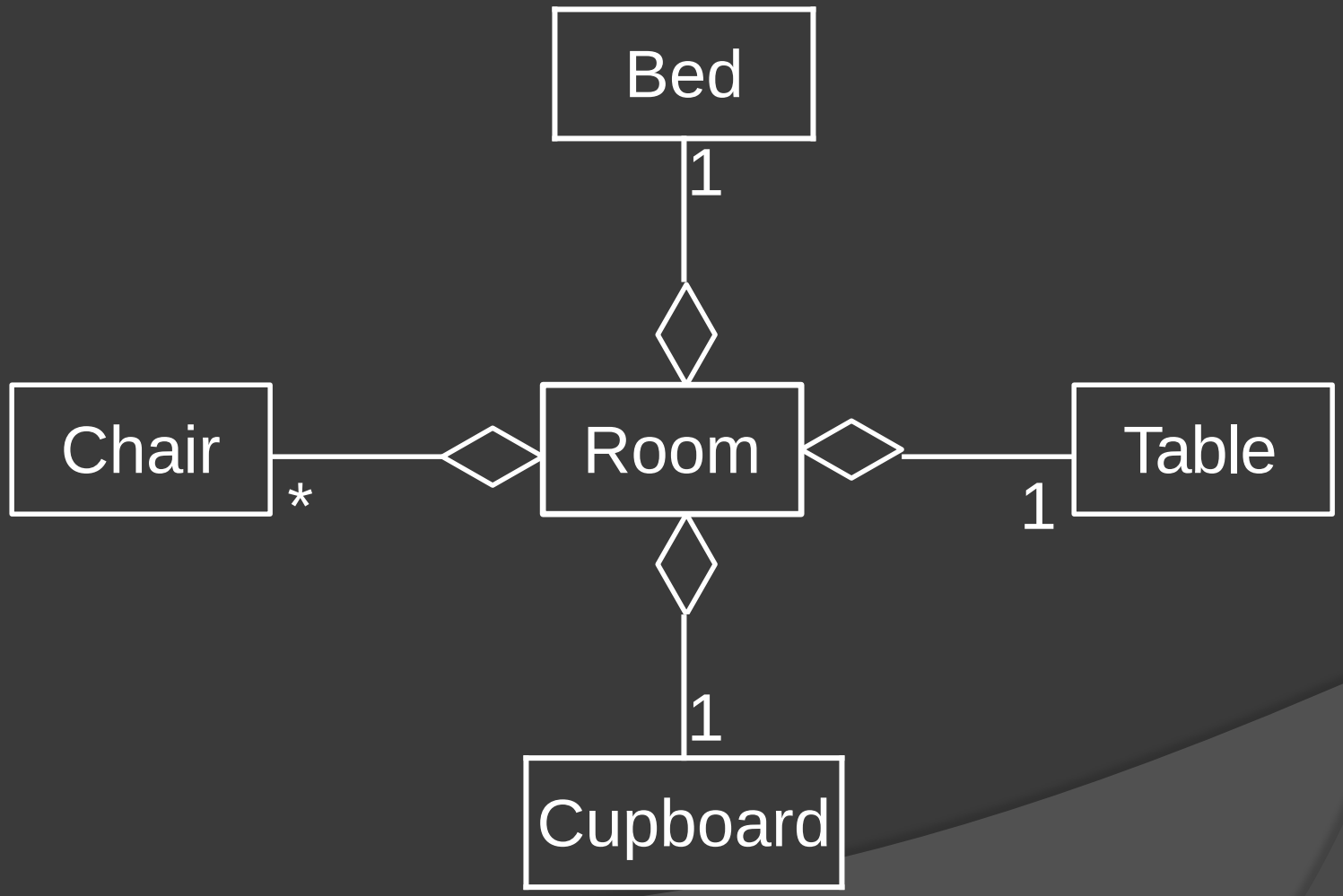
Example – Composition is Stronger

- Chair's body is made up of different parts
- They can't exist independently

Aggregation

- An object may contain a collection (aggregate) of other objects
- The relationship between the container and the contained object is called aggregation
- Aggregation is represented by a line with unfilled-diamond head towards the container

Example – Aggregation



Example – Aggregation

Aggregation is Weaker

- Aggregation is weaker relationship, because
  - Aggregate object is not a part of the container
  - Aggregate object can exist independently

Example – Aggregation is Weaker

- Furniture is not an intrinsic part of room
- Furniture can be shifted to another room, and so can exist independent of a particular room

Example – Aggregation is Weaker

- A plant is not an intrinsic part of a garden
- It can be planted in some other garden, and so can exist independent of a particular garden