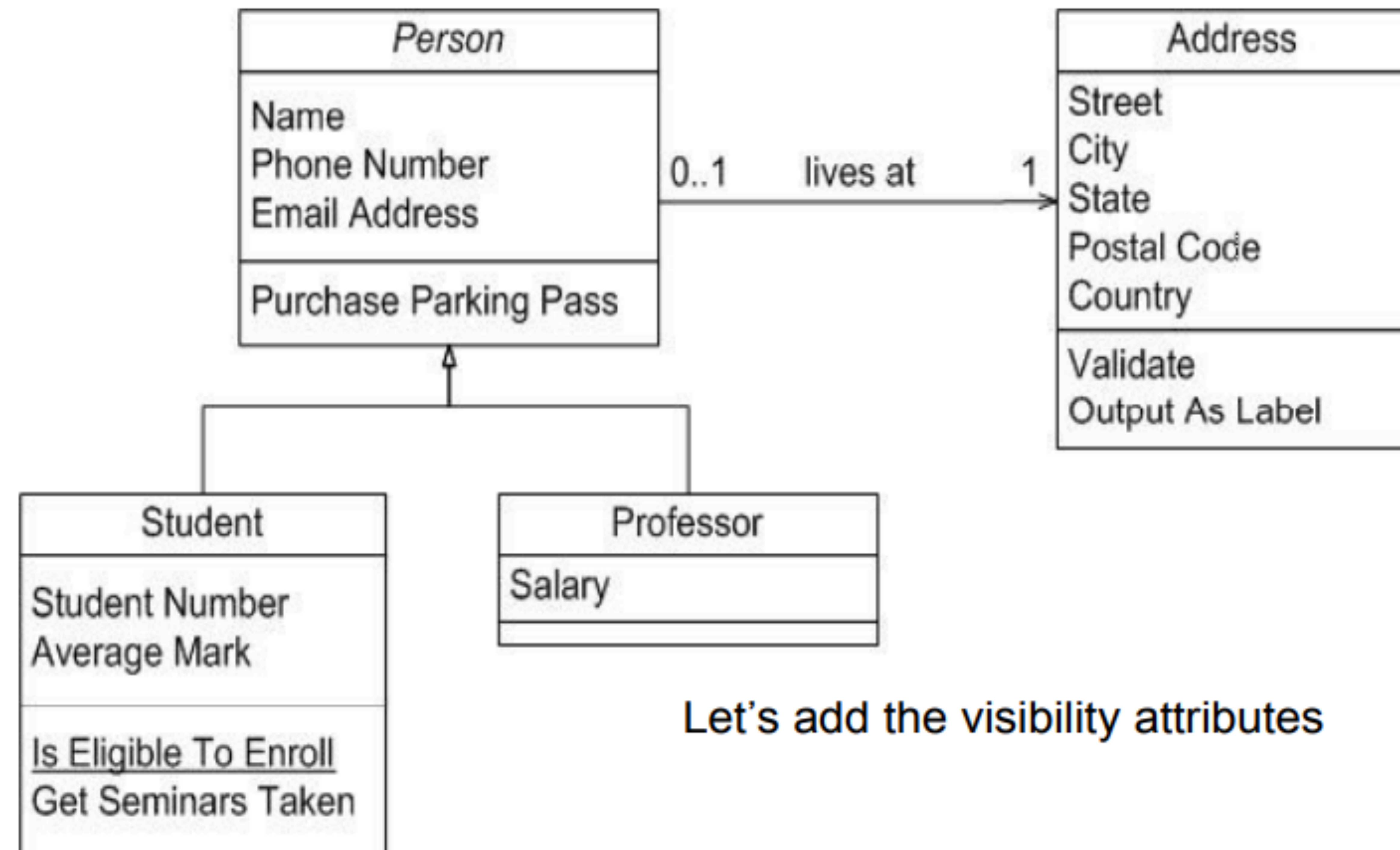
Information System Design

Lecture 3:

Dr. Moustafa Alzantot

UML class diagram exercise

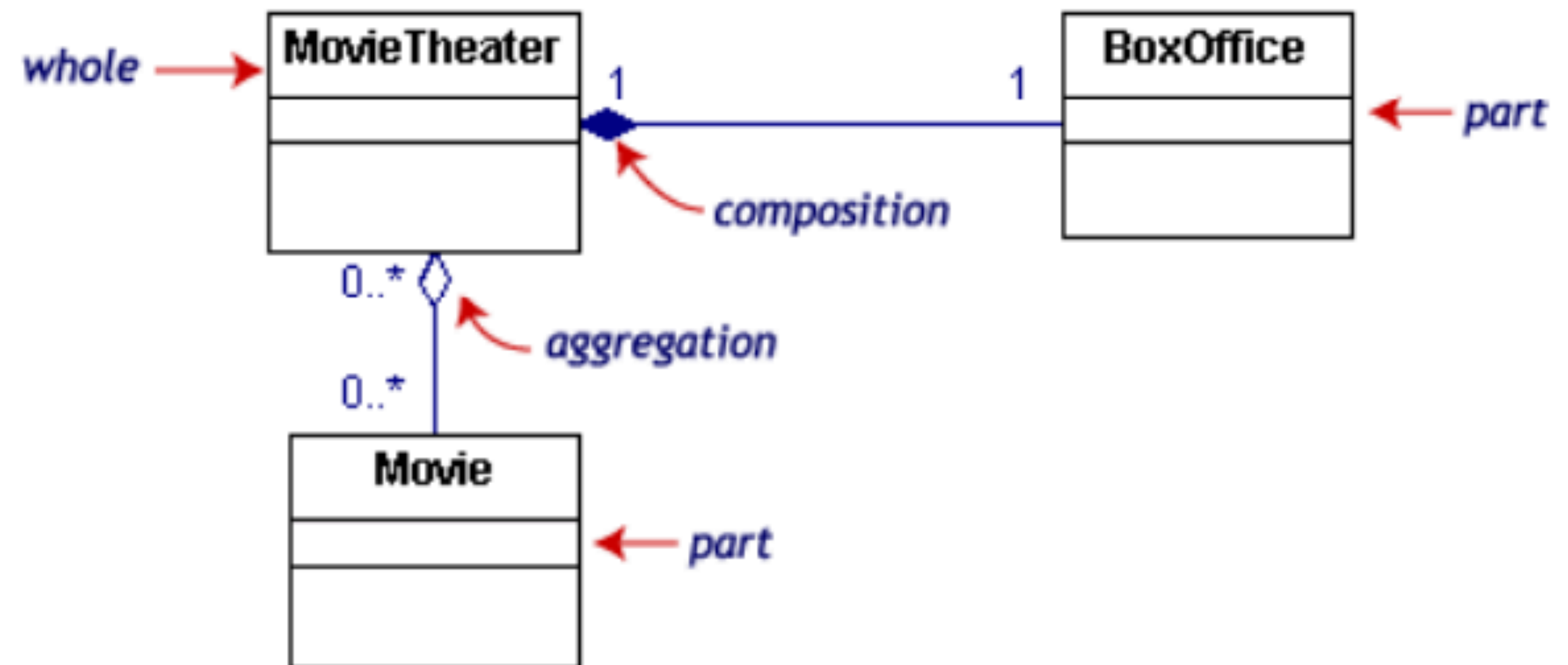
Can you read this ?



Let’s add the visibility attributes

UML class diagram exercise

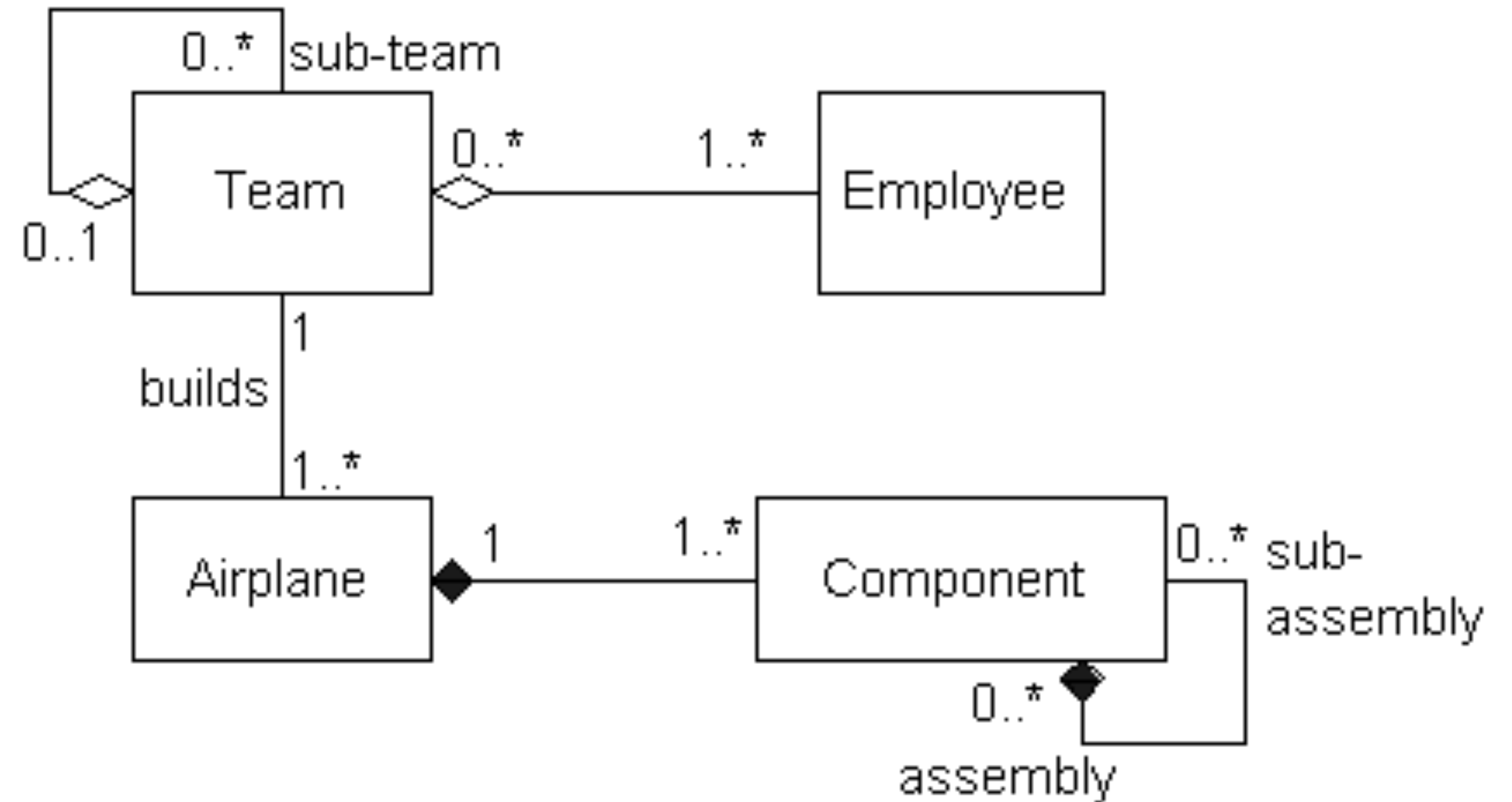
Can you read this ?



If the movie theatre goes away
so does the box office => composition
but movies may still exist => aggregation

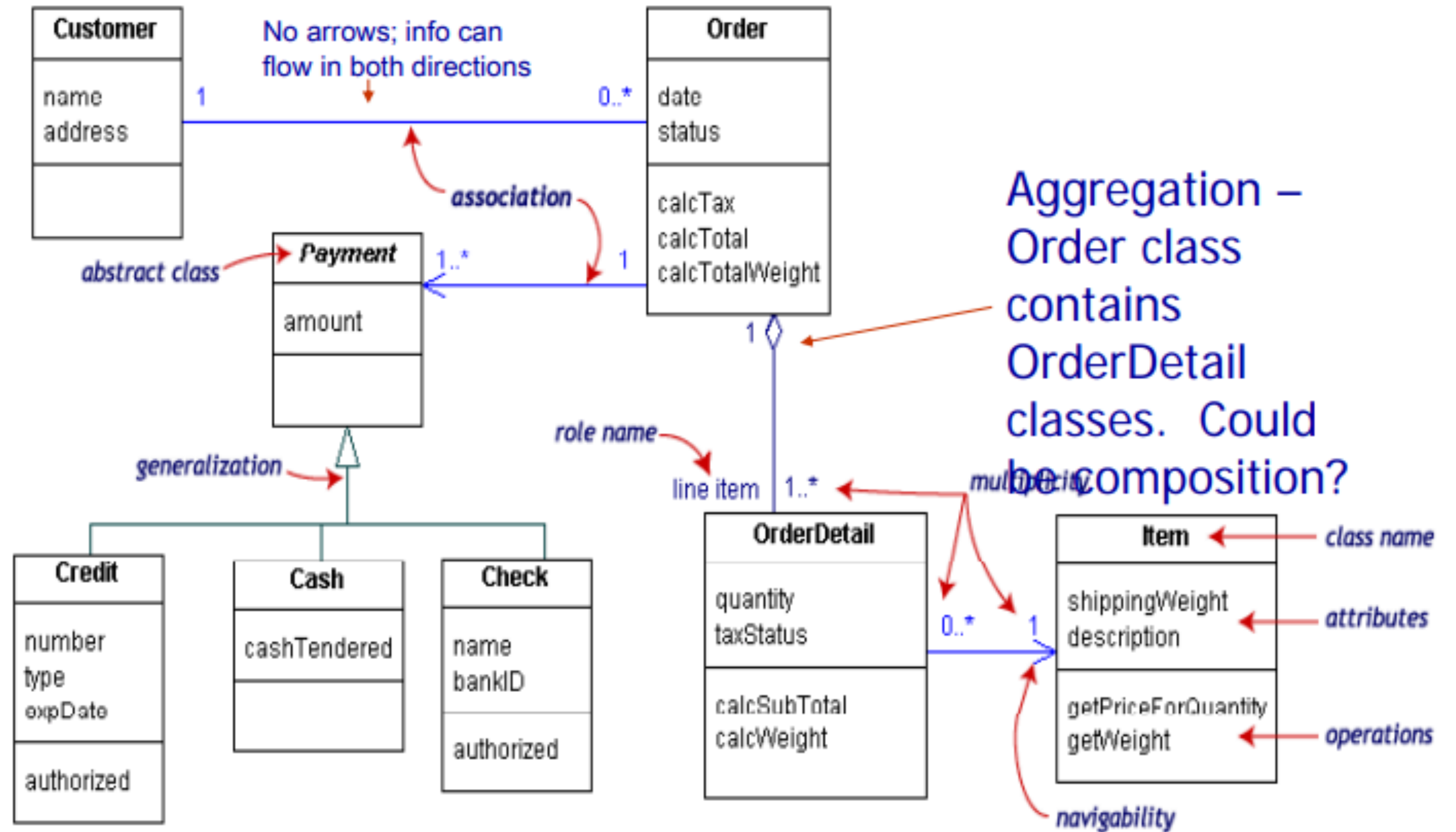
UML class diagram exercise

Can you read this ?



UML class diagram exercise

Can you read this ?



More on inheritance (generalization)

Some OO programming languages, like Java supports only inheritance from a single parent class.

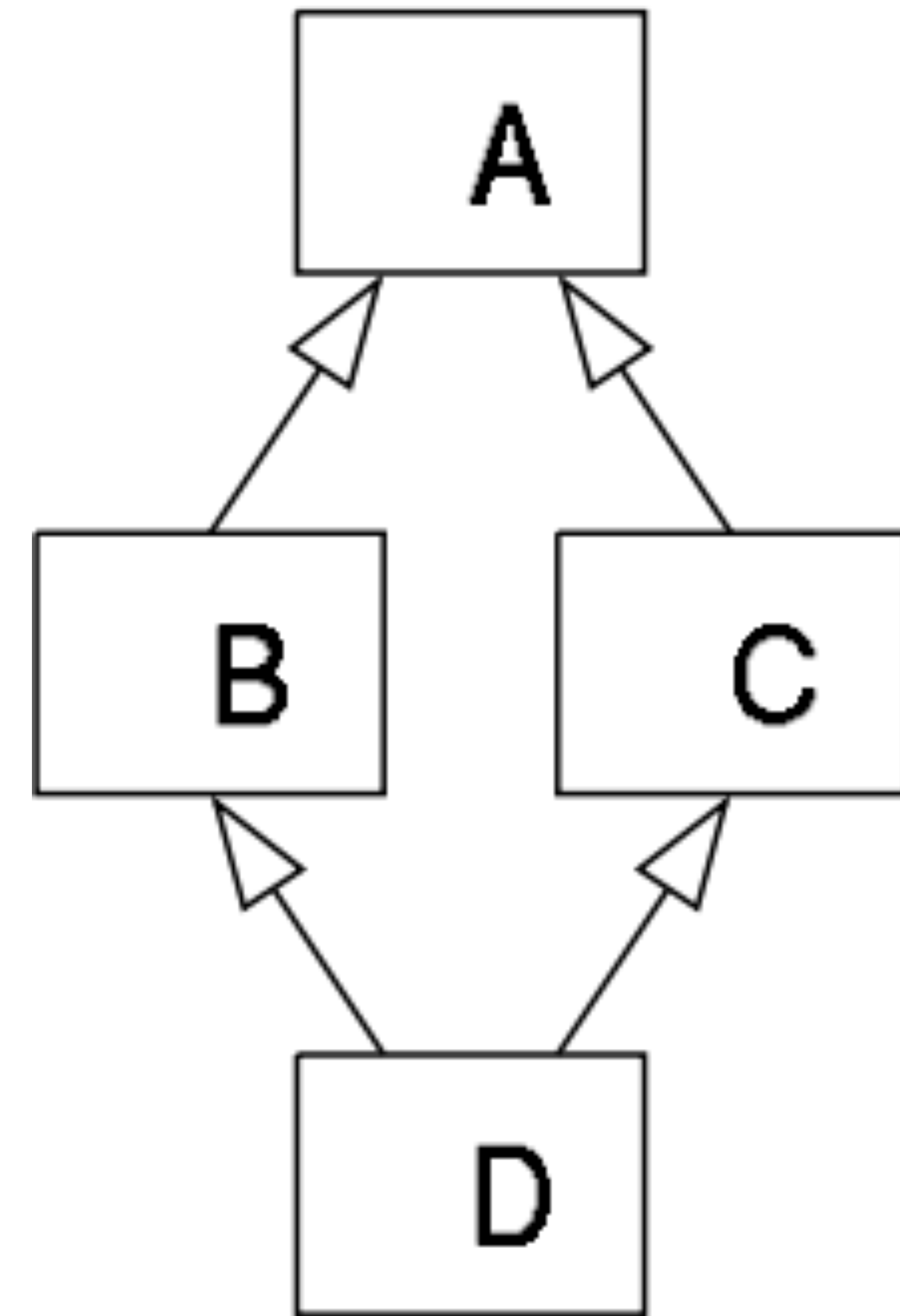
Other languages, such as C++ and Python, allow inheritance from multiple classes.

Inheritance from multiple parents can cause “***diamond problem***”.

More on inheritance (generalization)

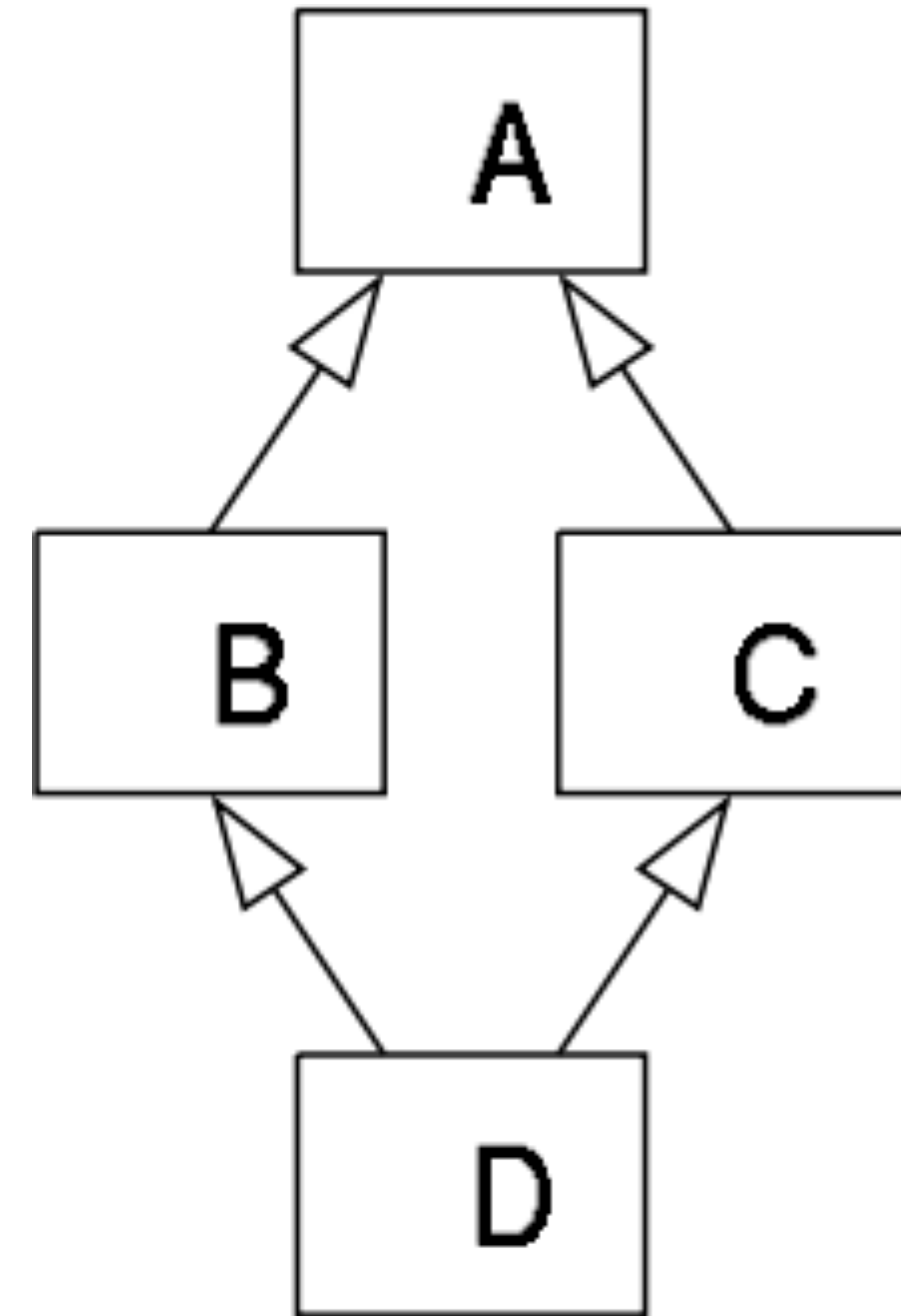
diamond problem: ambiguity that arises when

- two classes B and C inherit from A
- class D inherits from both B and C.
- If there is a method in A that both B and C have overrides, and D does not override it. Then which version of the method does D inherit ?



More on inheritance (generalization)

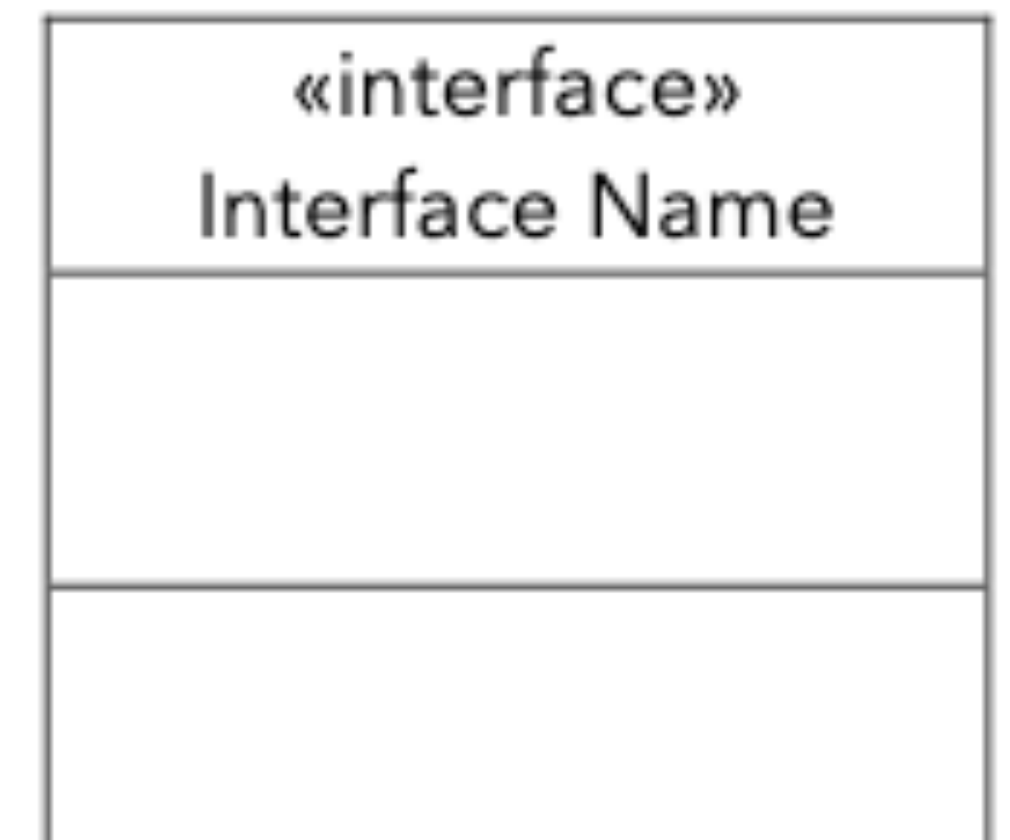
- Different programming languages have different ways to deal with ***diamond problem.***
- ***C++ :*** virtual inheritance* .
- ***Python:*** uses the list of classes to inherit from as ordered list.
- ***Java:*** *disallow multiple inheritance.*



- * **Virtual Inheritance**: https://en.wikipedia.org/wiki/Virtual_inheritance

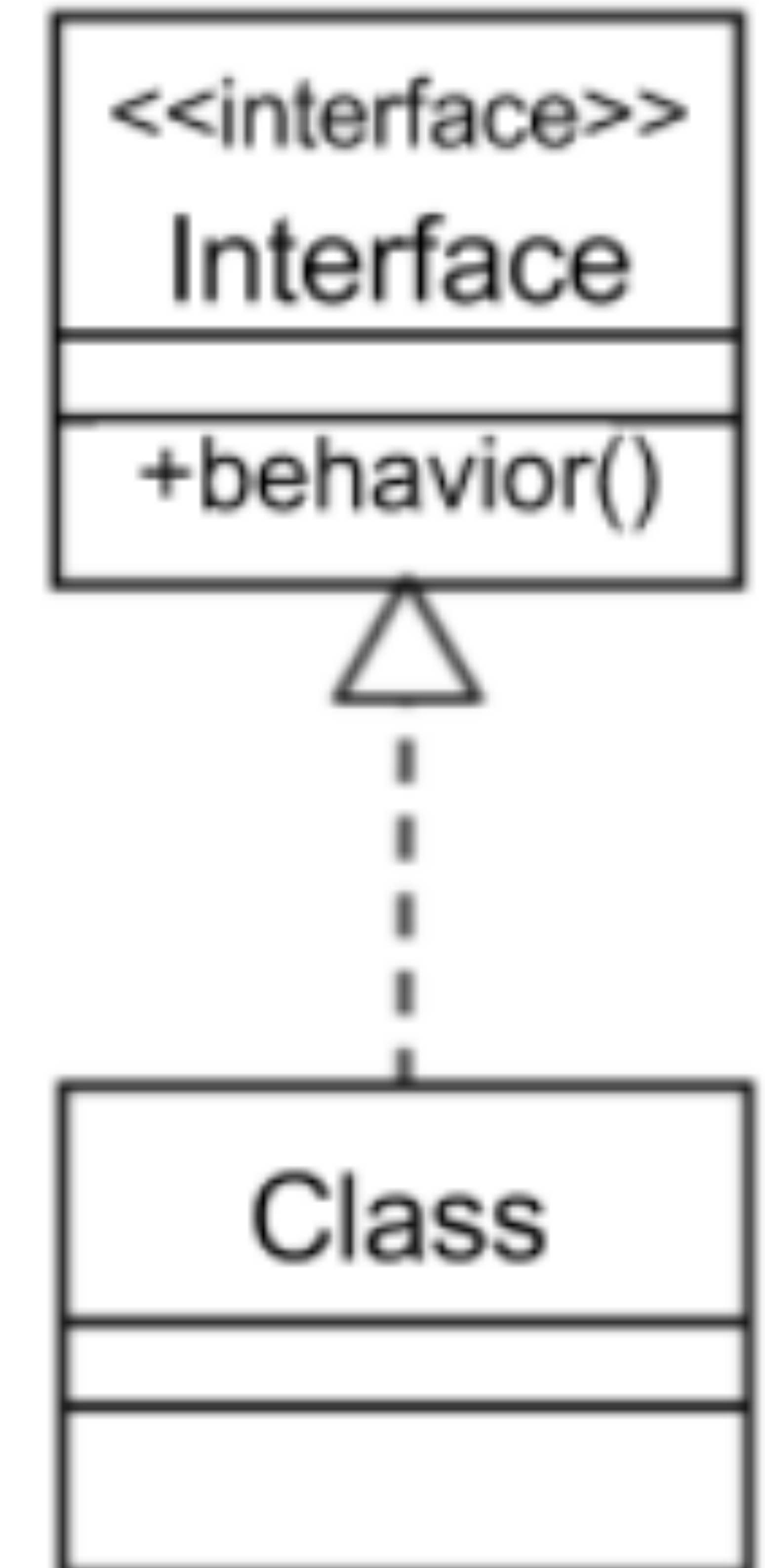
Interfaces

- ***Interface (in Java):*** a type that declares only method signatures with no attributes, constructors or method bodies.
- Interface specify the expected behaviors via method signatures but doesn't provide any implementation details.
- Interfaces are represented in UML class diagram in similar ways to class diagrams but adding the **<<interface>>** to the top of interface name.



Interfaces

- A class that implements an interface must provide implementation for that interface methods.
- An interface is like a contract to be fulfilled by the implementing class.
- Interaction between an interface and a class that implements it is represented as a dot-arrow where the interfaces touches the head of the arrow.



Interfaces

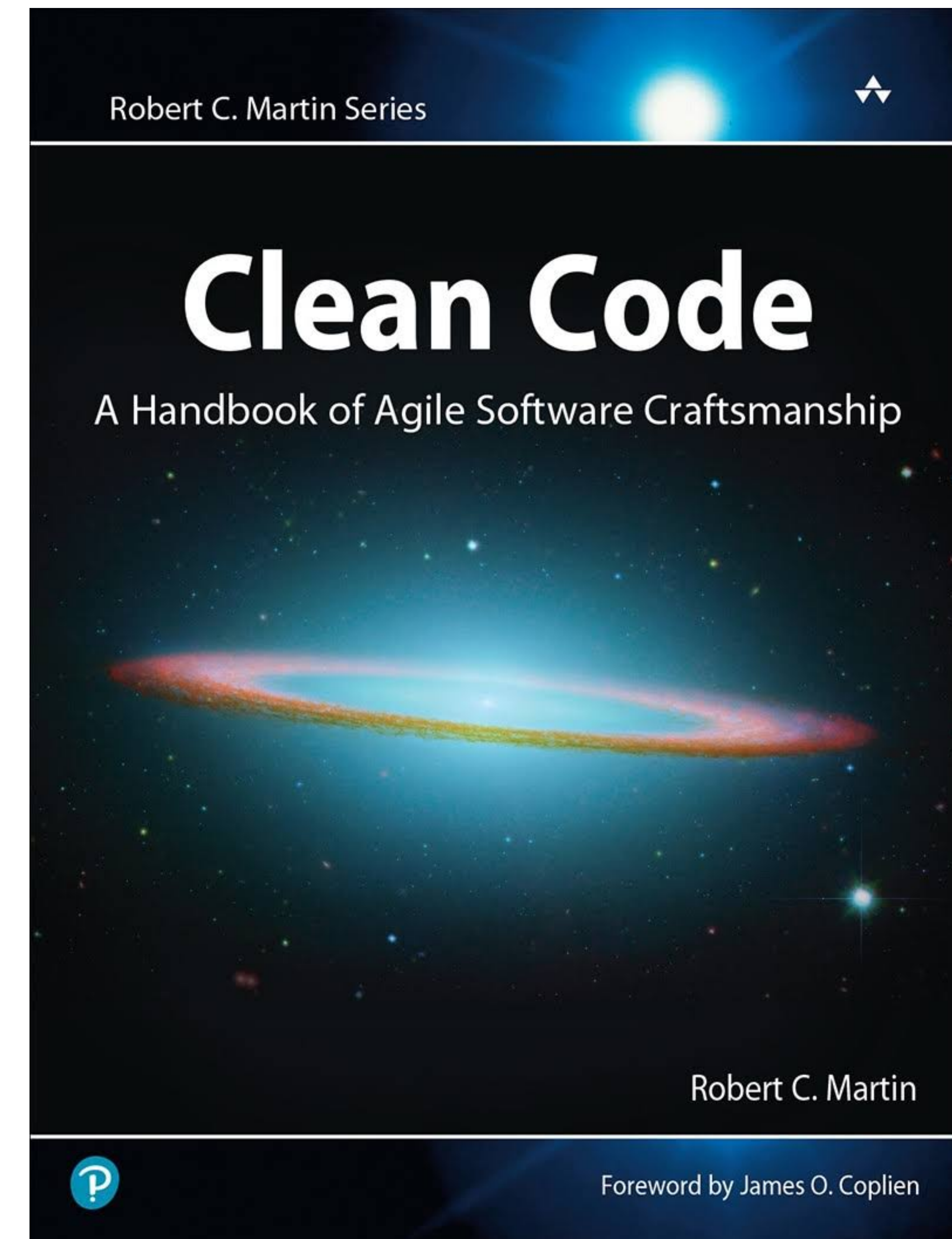
- Java interfaces (same way as abstract classes in C++) cannot be instantiated.
- A class that implements as many interfaces as needed.
- An overlapping method signature is not a problem, because interface doesn’t provide an implementation and the implementing the class will have to implement it itself.

SOLID design principles

SOLID is an acronym for five design principles :

- Single Responsibility Principle (SRP)
- Open-Closed Principle (OCP)
- Liskov Substitution Principle (LSP)
- Interface Segregation Principle (ISP)
- Dependency Inversion Principle (DIP)

- These principles were introduced by Robert Martin in his articles and book “Clean Code”.



Single Responsibility Principle

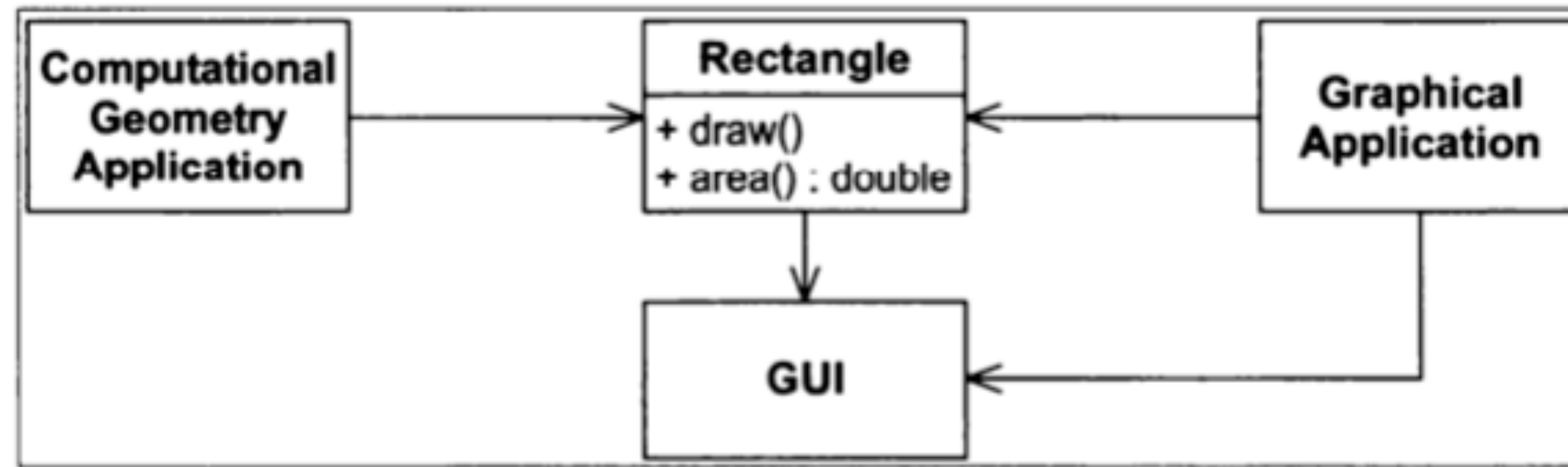
Single Responsibility Principle states :

A class should have only one reason to change.

- If the class has more than one responsibility, then there will more than one reason for it change when requirements change.
- When responsibilities become coupled, a change to one responsibility may impair the ability of the class to perform others.

Single Responsibility Principle

Example



Source : Agile Software Development, Principles, Patterns and Practices

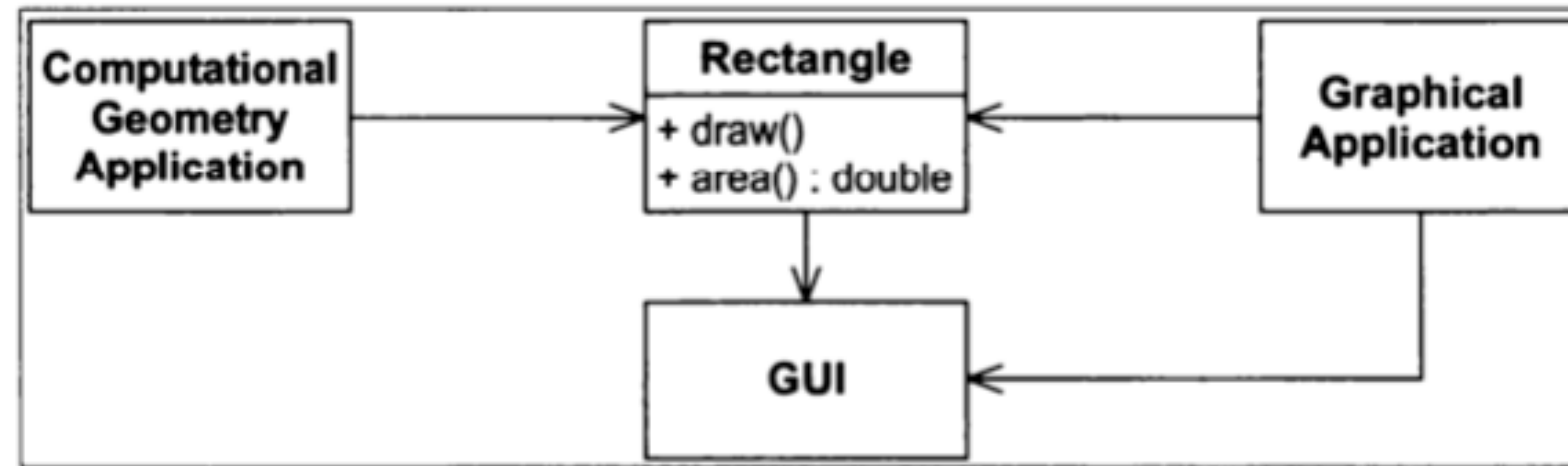
`Rectangle` class has two responsibilities.

- Computes the area of rectangle
- Renders a rectangle on the screen

- It is being used by two applications `ComputationalGeometryApplication` and `GraphicalApplication`

Single Responsibility Principle

Example

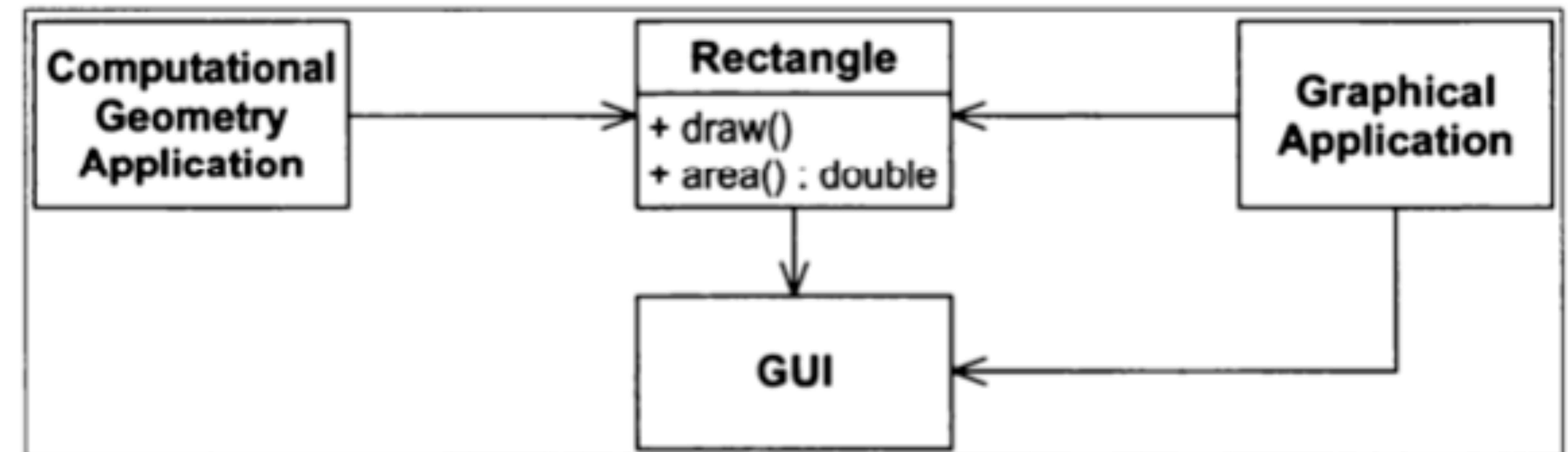


Why is this a violation of the SRP ?

Single Responsibility Principle

This design violates SRP in many ways

- We must include `GUI` in `ComputationalGeometryApplication` which doesn't need it.
- A change on how rendering happens in the screen, will require re-building, re-deploying and re-testing `ComputationalGeometryApplication`.



Source : Agile Software Development, Principles, Patterns and Practices

Single Responsibility Principle

- A better design is to separate responsibilities
  - Computational parts of `Rectangle` moves into a separate class `GeometricRectangle` and then `ComputationalGeometryApplication` need to depend only on this class.

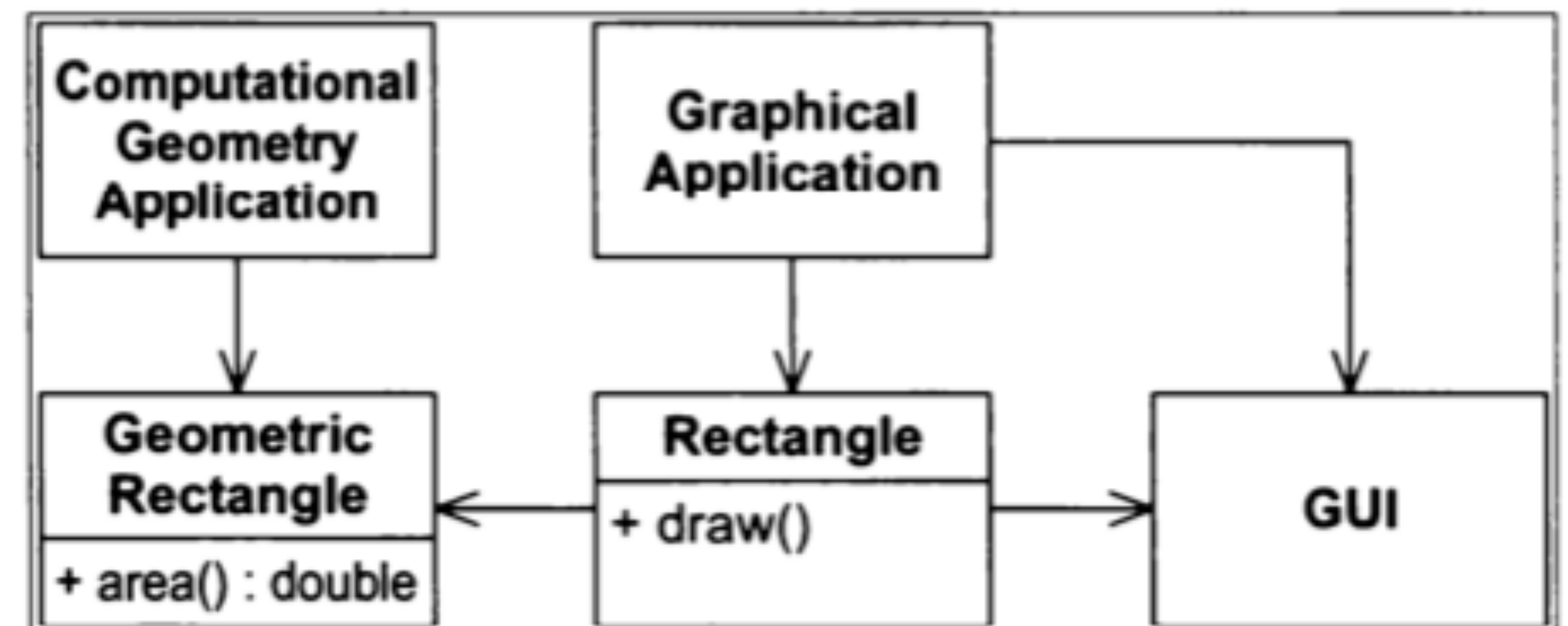


Figure 8-2 Separated Responsibilities

Source : Agile Software Development, Principles, Patterns and Practices

Single Responsibility Principle

- A common violation of SRP, Employee class has both business logic and persistence control.
  - Business logic changes too often the persistence control, do we need to rebuild and retest the persistence part every time we change it ?
  - What if we want to change way data is stored ?

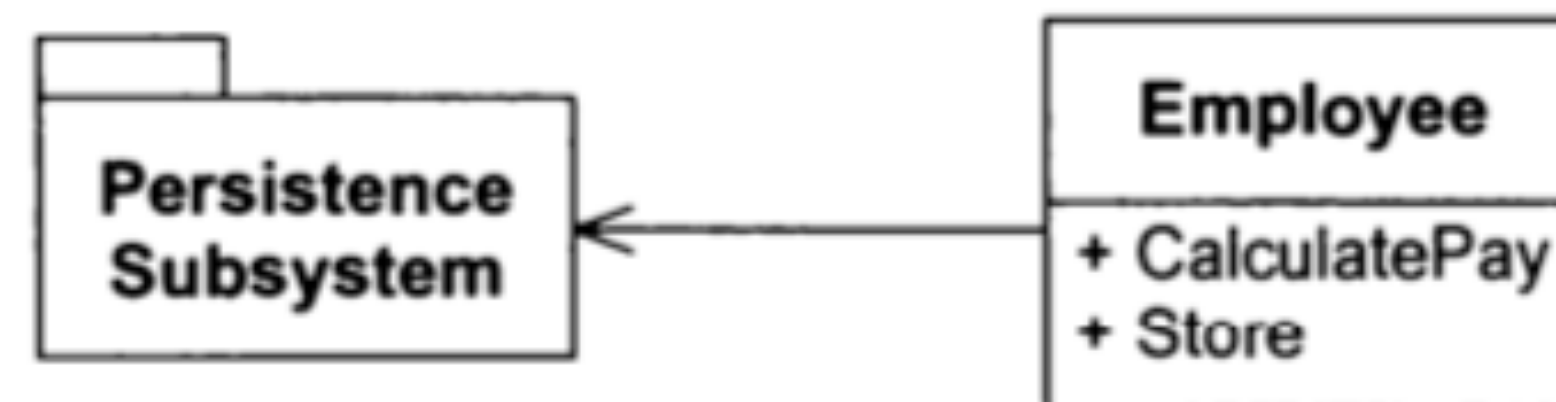


Figure 8-4 Coupled Persistence

Source : Agile Software Development, Principles, Patterns and Practices

Open-Closed Principle

- The Open-Closed Principle

  Software entities should be open for extension, but closed for modification.

Modules that satisfy (OCP) principle are :

Open for extension: this means their behavior can be extended. If the requirements of the application change, we can extend the module with new behaviors to satisfy the requirements change.

Closed for modification: extending the behavior of the module doesn't result in changes to source or binary of the module. The binary executable version (e.g. DLL or java JAR) remains unchanged.

Open-Closed Principle

How can a module be both open for extension and closed for modification at the same time ?

Use abstractions and Polymorphism. Abstractions are abstract base classes and that could be extended by an unbounded group of possible behaviors through derivative classes.

A module that relies on abstract class is closed for modification because the abstract class remains unchanged. Yet the behavior can be extended by creating a new derivative of the abstraction.

Open-Closed Principle

Figure 9-1 Client is not open and closed

Source : Agile Software Development, Principles, Patterns and Practices

Both `Client` and `Server` are concrete classes.

The `Client` uses `Server` class, if we wish to change a different server object, the `Client` class must be changed.

Open-Closed Principle

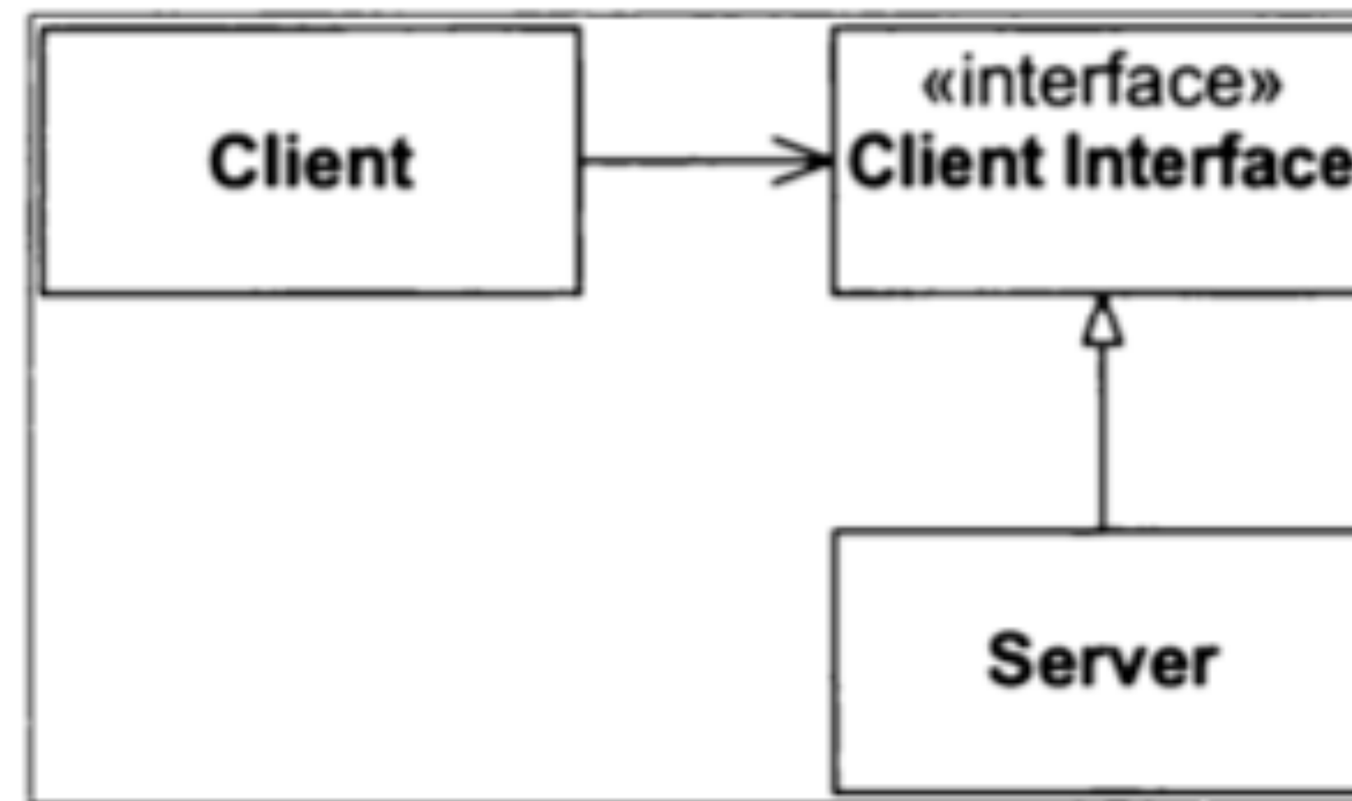


Figure 9-2 STRATEGY pattern: Client is both open and closed

Source : Agile Software Development, Principles, Patterns and Practices

`Client` needs some work to get done, it can describe it in terms of abstract interface “`ClientInterface`”.

Sub-types of `ClientInterface` can implement the interface in any manner the choose.