Software Design Patterns

Lecture 1

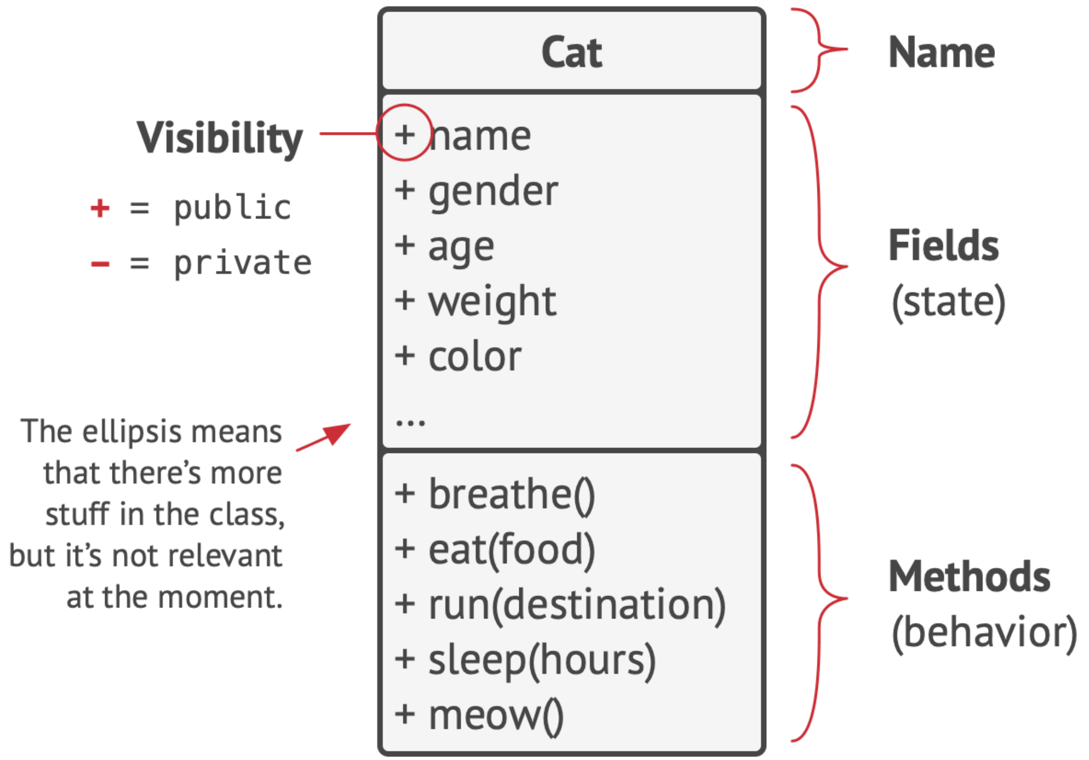
Object-Oriented Programming
UML Class Diagrams

Dr. Fan Hongfei

18 September 2025

Object-Oriented Programming

- A programming paradigm
- Classes and objects



Visibility

+ = public

– = private

The ellipsis means that there's more stuff in the class, but it's not relevant at the moment.

Cat

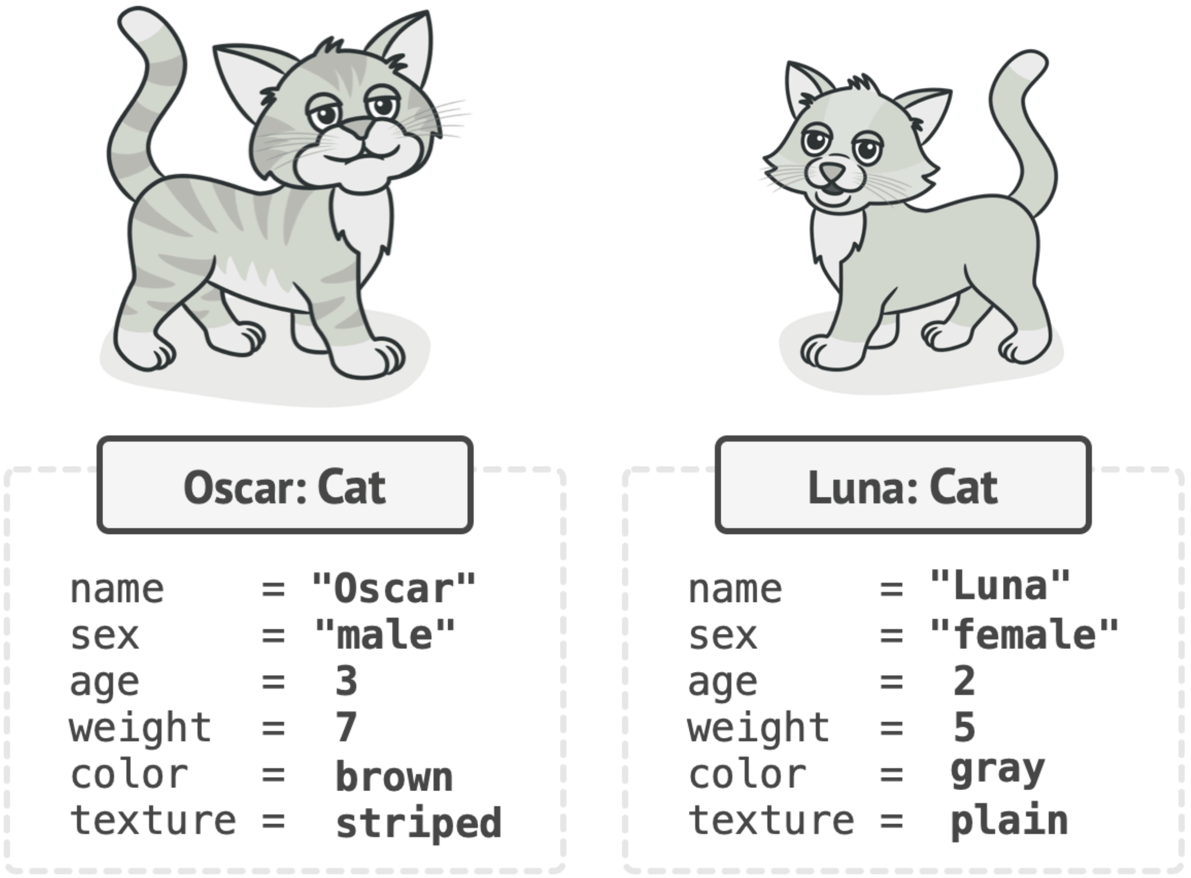
+ name
+ gender
+ age
+ weight
+ color
…

+ breathe()
+ eat(food)
+ run(destination)
+ sleep(hours)
+ meow()

Name

Fields (state)

Methods (behavior)



Oscar: Cat

```
name    = "Oscar"
sex     = "male"
age     =  3
weight  =  7
color   =  brown
texture =  striped
```

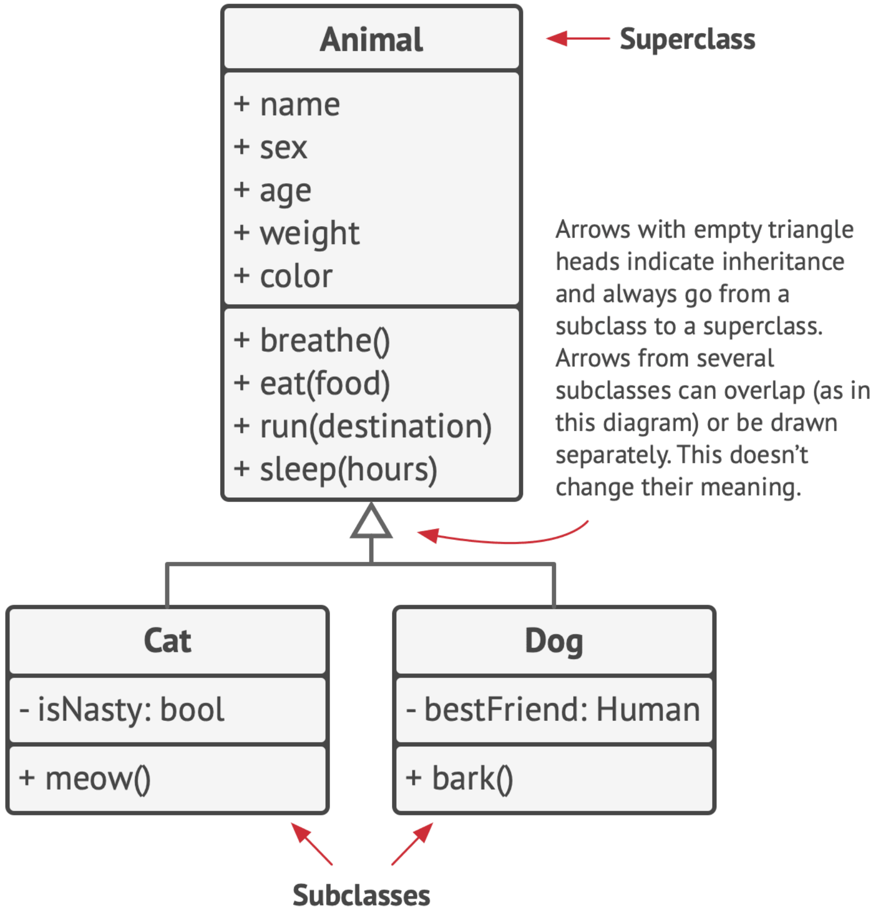
Luna: Cat

```
name    = "Luna"
sex     = "female"
age     =  2
weight  =  5
color   =  gray
texture =  plain
```

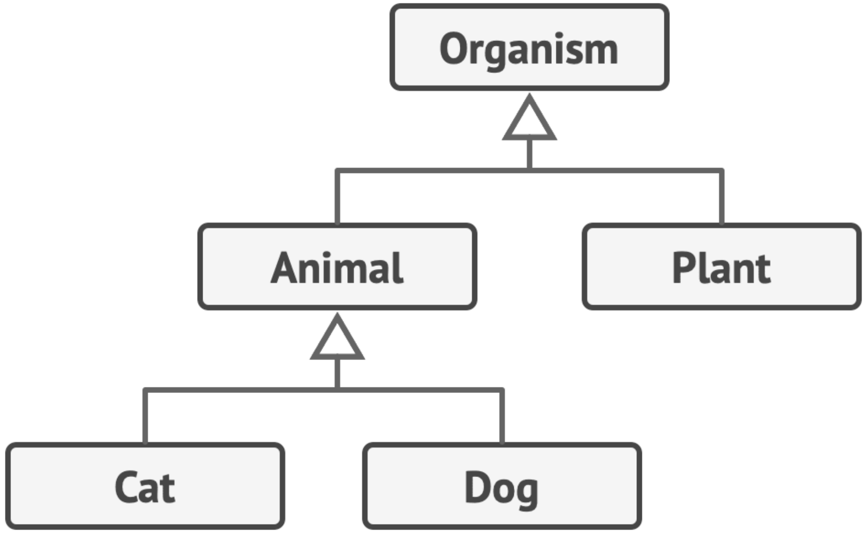
[Figures in this slide are extracted from "Dive Into Design Patterns" by Alexander Shvets]

Class Hierarchies

- Superclass and subclass



Arrows with empty triangle heads indicate inheritance and always go from a subclass to a superclass. Arrows from several subclasses can overlap (as in this diagram) or be drawn separately. This doesn't change their meaning.

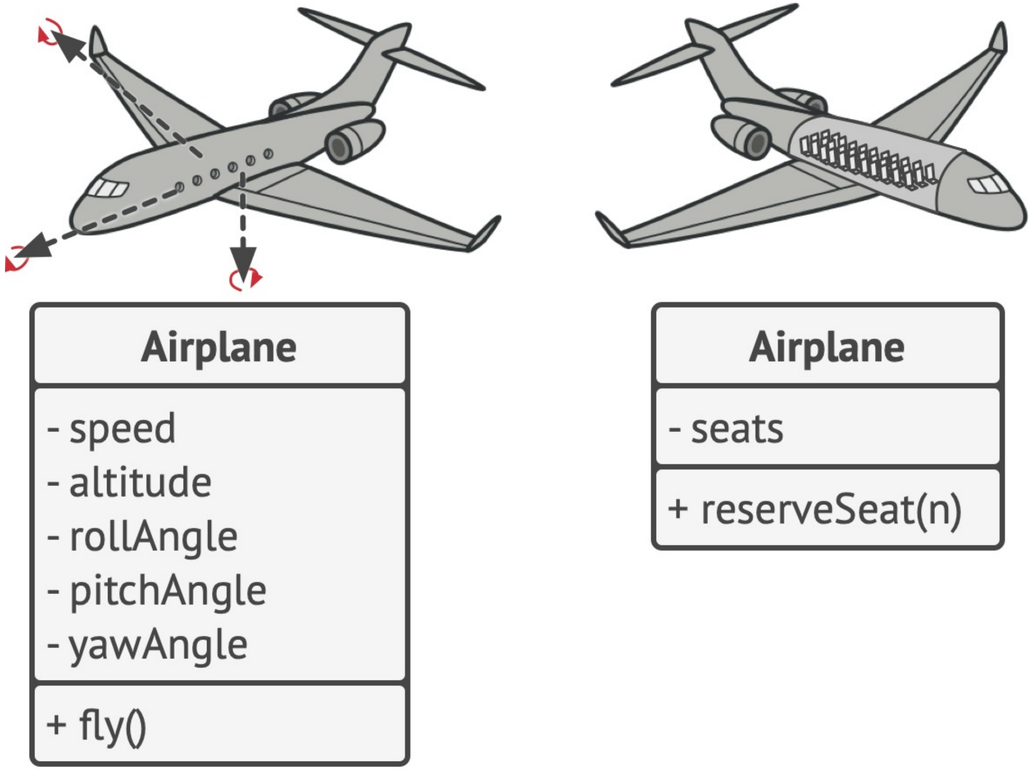


[Figures in this slide are extracted from "Dive Into Design Patterns" by Alexander Shvets]

Pillars of Object-Oriented Programming

1) Abstraction

- Modelling attributes and behaviors of real objects, in specific contexts



Airplane

- speed
- altitude
- rollAngle
- pitchAngle
- yawAngle

+ fly()

Airplane

- seats

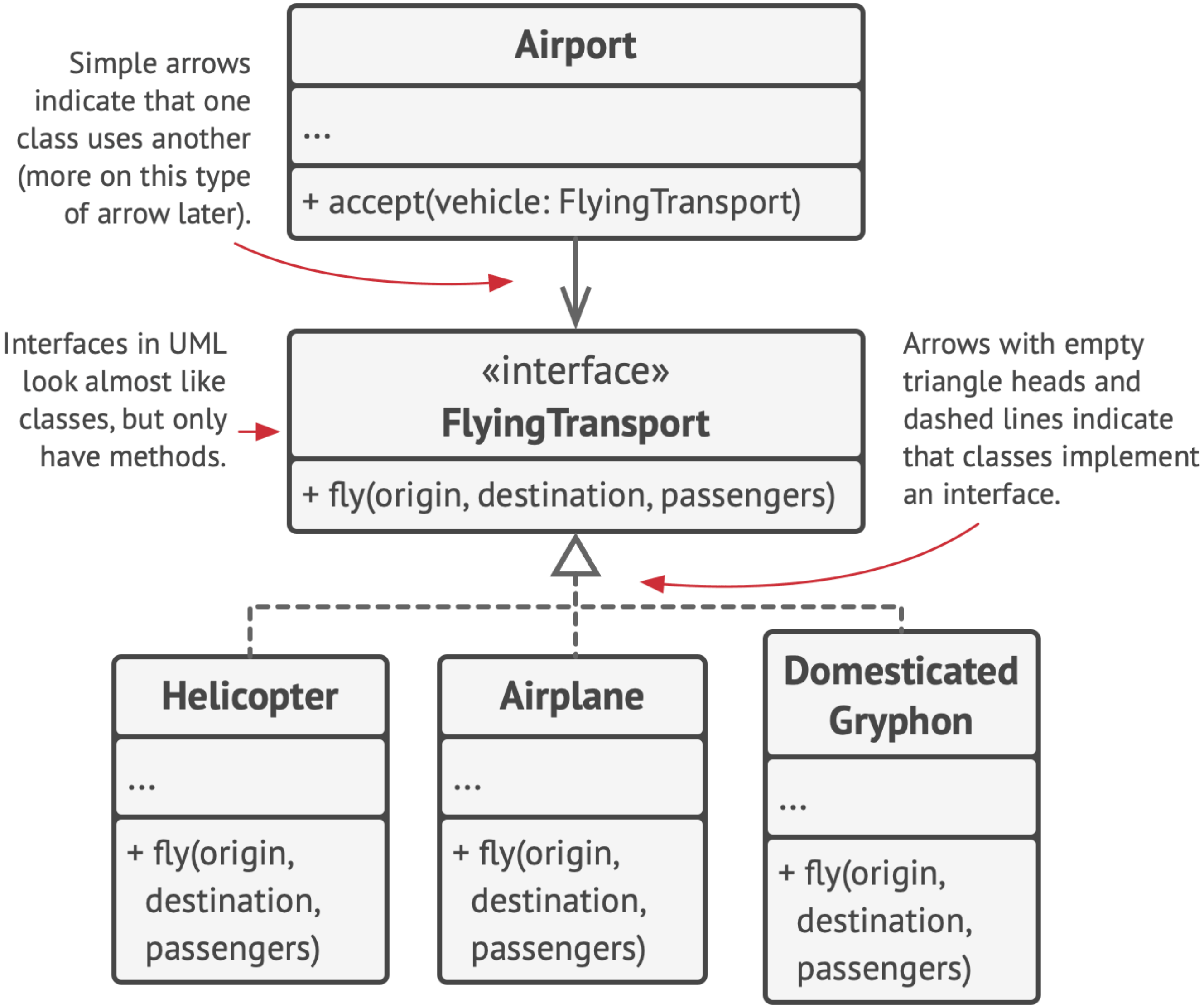
+ reserveSeat(n)

[Figures in this slide are extracted from "Dive Into Design Patterns" by Alexander Shvets]

Pillars of Object-Oriented Programming (cont.)

2) Encapsulation

- Hiding parts of an object's states and behaviors from others, and exposing a limited set of interfaces
- public, private, and protected
- Interfaces and abstract classes



Airport
...
+ accept(vehicle: FlyingTransport)
«interface»
FlyingTransport
+ fly(origin, destination, passengers)
Helicopter
...
+ fly(origin, destination, passengers)
Airplane
...
+ fly(origin, destination, passengers)
Domesticated Gryphon
...
+ fly(origin, destination, passengers)

Simple arrows indicate that one class uses another (more on this type of arrow later).

Interfaces in UML look almost like classes, but only have methods.

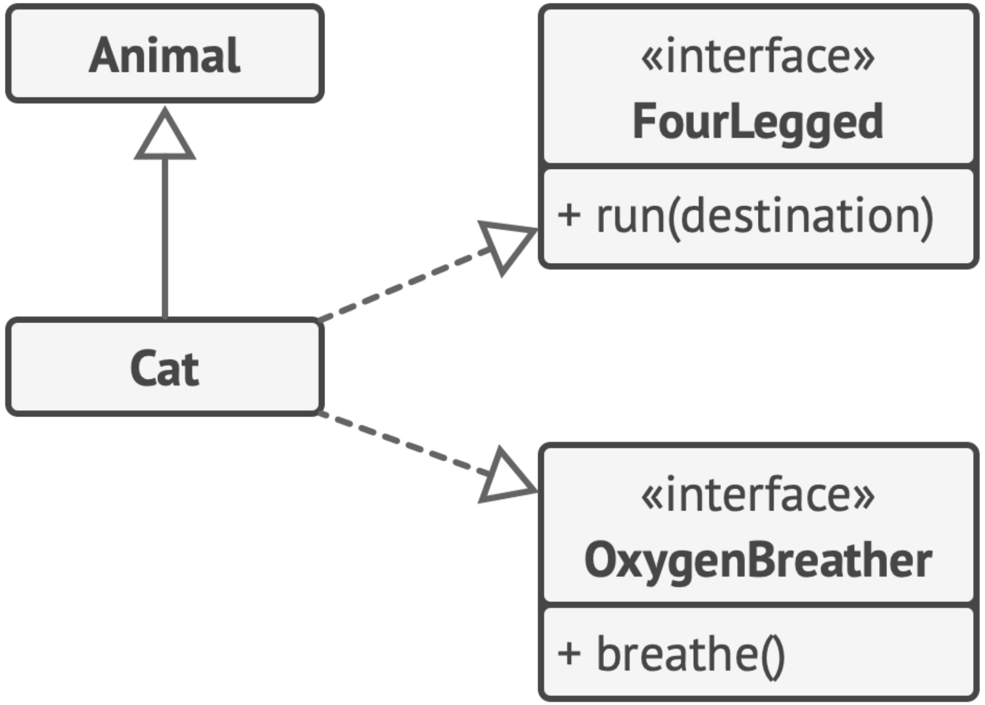
Arrows with empty triangle heads and dashed lines indicate that classes implement an interface.

[Figures in this slide are extracted from "Dive Into Design Patterns" by Alexander Shvets]

Pillars of Object-Oriented Programming (cont.)

3) Inheritance

- Main benefit: code reuse



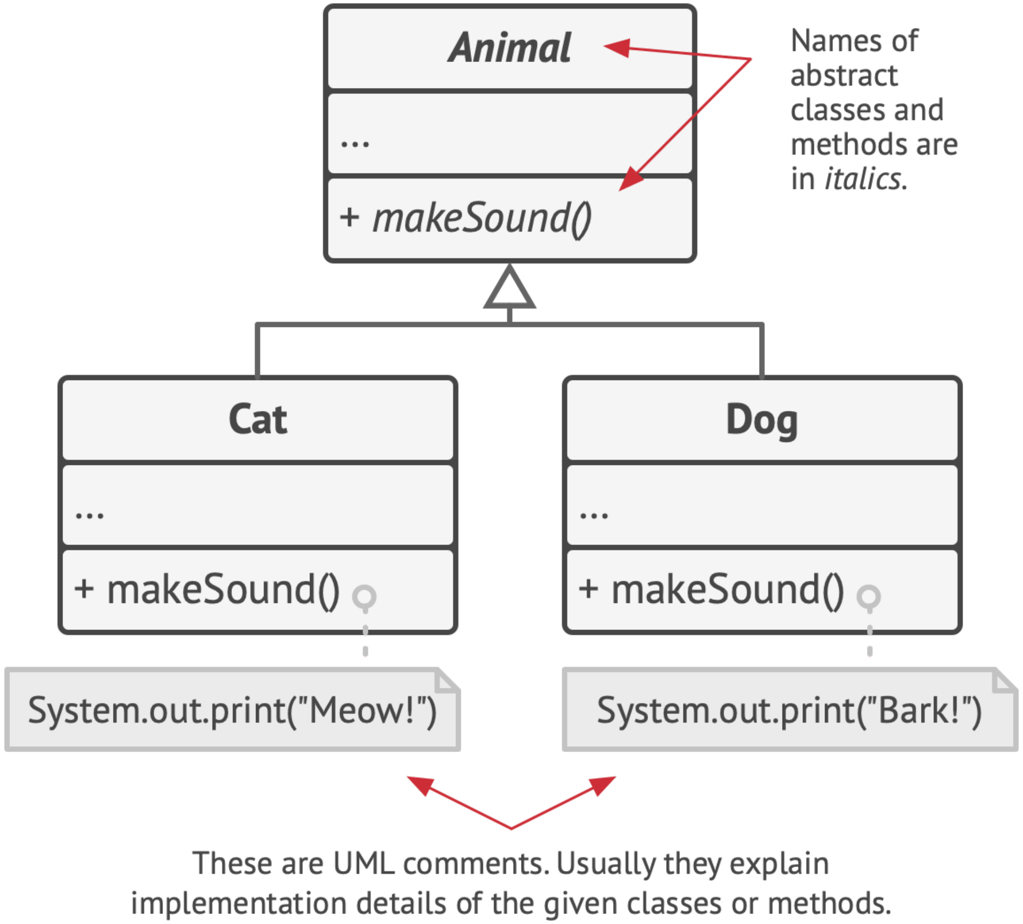
[Figures in this slide are extracted from “Dive Into Design Patterns” by Alexander Shvets]

Pillars of Object-Oriented Programming (cont.)

4) Polymorphism

– Performing an action in many forms

– A mechanism for detecting the real class of an object and call its implementation

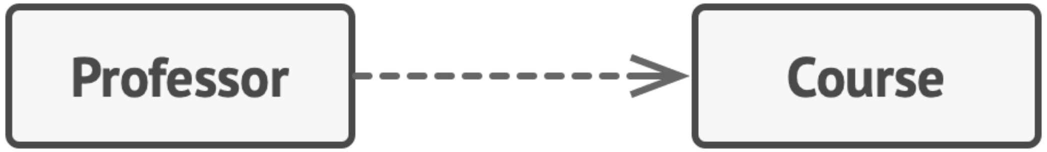


Animal
...
+ *makeSound()*
Names of abstract classes and methods are in *italics*.
Cat
...
+ makeSound()
Dog
...
+ makeSound()
System.out.print("Meow!")
System.out.print("Bark!")
These are UML comments. Usually they explain implementation details of the given classes or methods.

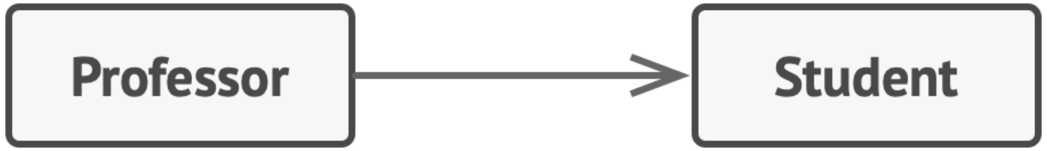
[Figures in this slide are extracted from "Dive Into Design Patterns" by Alexander Shvets]

More Relations Between Objects

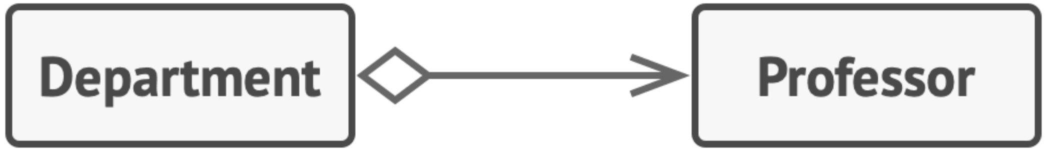
- **Dependency**



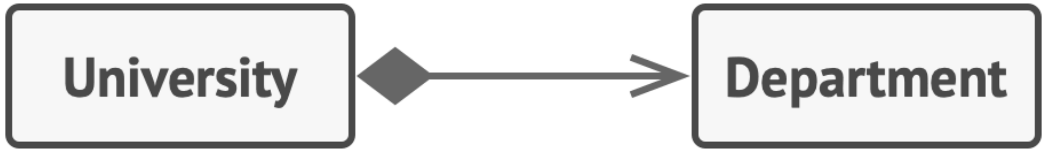
- **Association**

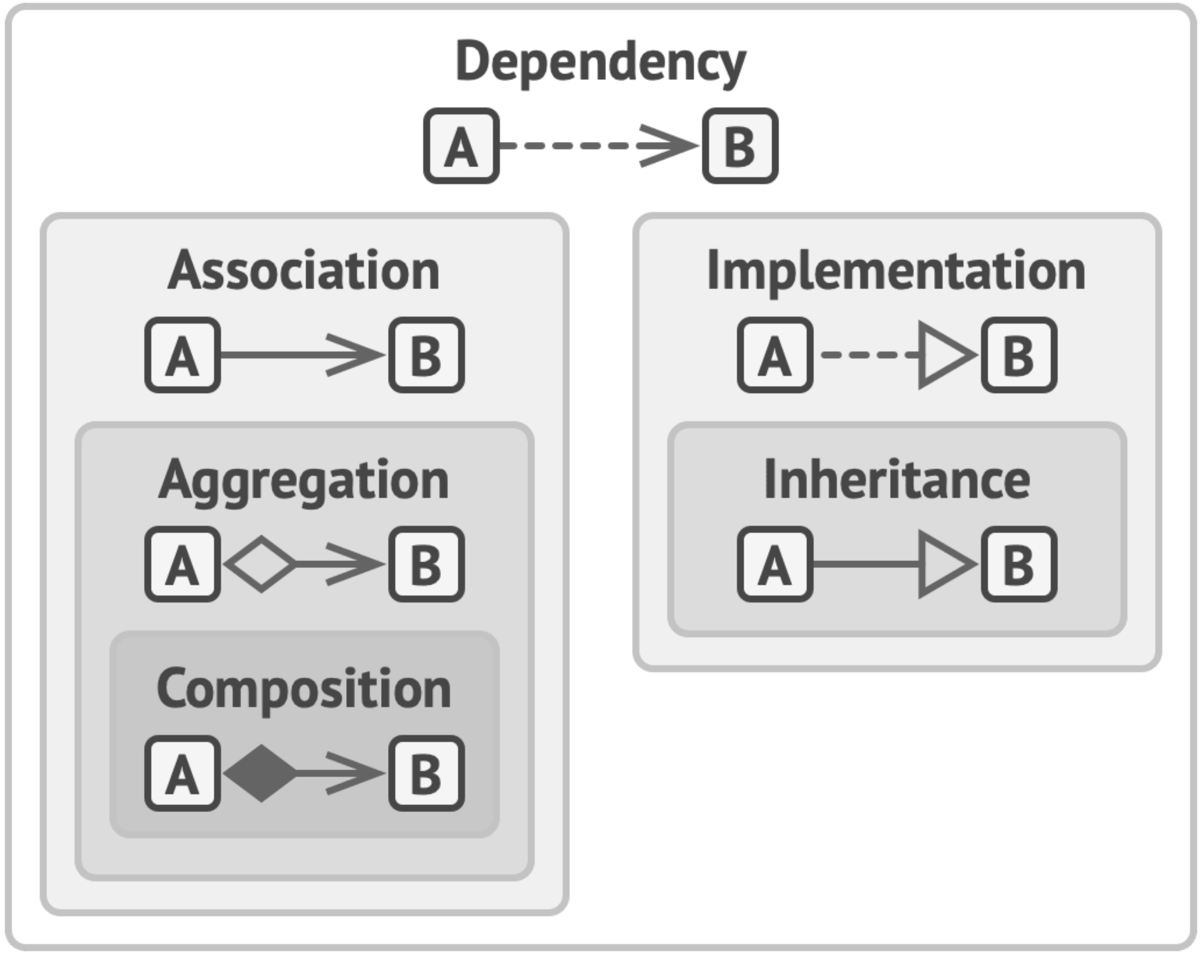


- **Aggregation**



- **Composition**





[Figures in this slide are extracted from "Dive Into Design Patterns" by Alexander Shvets]