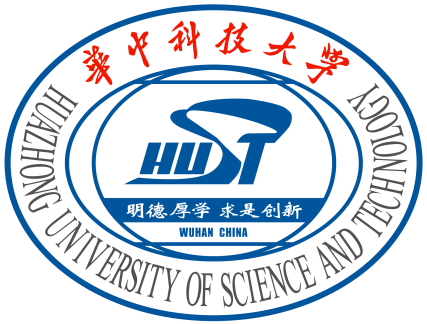
基于Java的面向对象程序设计

陈维亚
weiya_chen@hust.edu.cn

华中科技大学软件学院

第13讲：类的关系



目录

1. 1+3+1

- 类之间的3类基本关系

继承
Inheritance

关联
Association

聚合
Aggregation

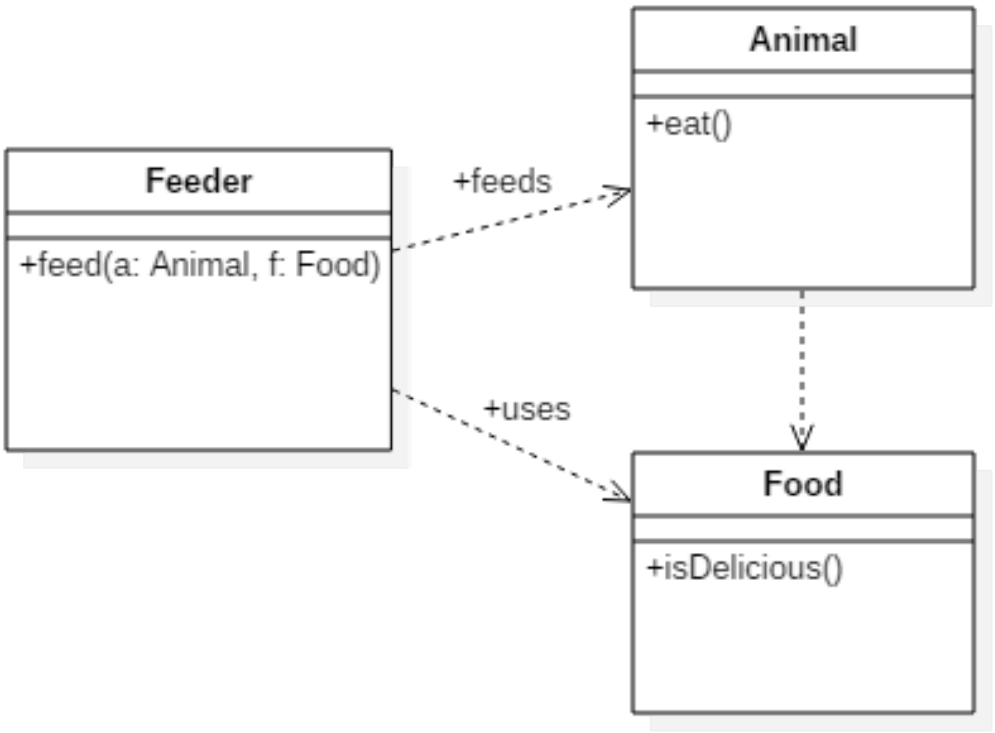
组合
Composition

依赖
Dependency

❑ 依赖 Dependency - 举例

```
public class Person {
    void drive(Vehicle v){
        …
    }

    void write(){
        Pen p = new Pen();
        p.write();
    }

    String getWeather(){
        return WeatherStation.getState();
    }
}
```

❑ 依赖 Dependency - 举例

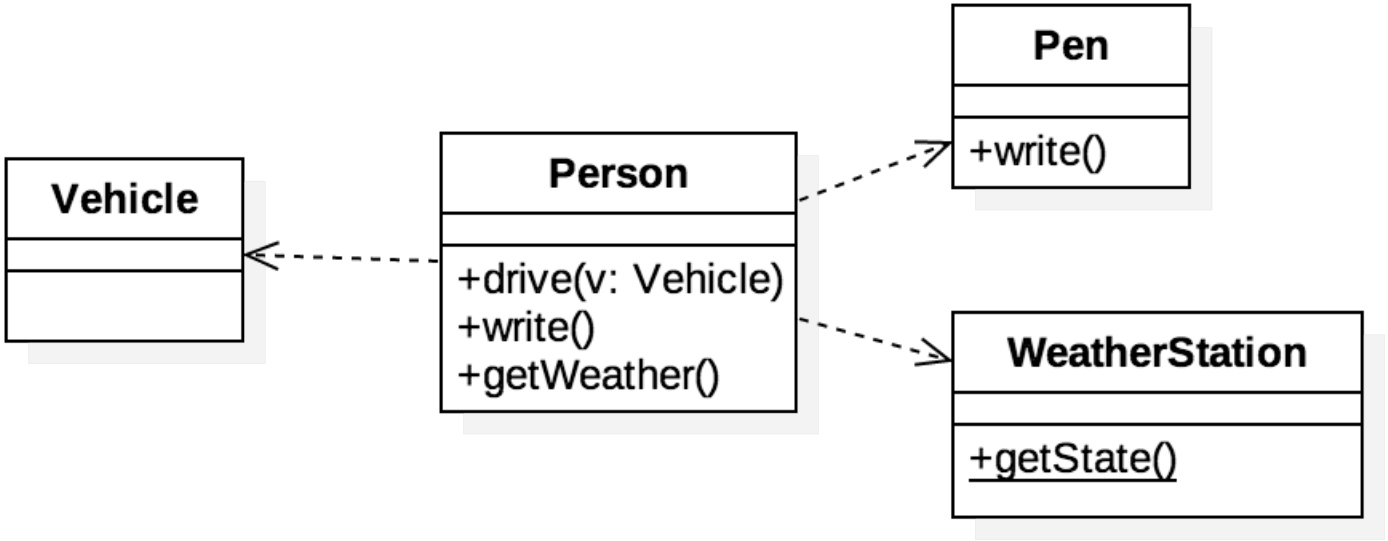


❑ 依赖 Dependency - 观察

Person类功能的实现，有赖于其他几个类Vehicle、Pen和WeatherStation。

其它几个类分别作为Person类方法内的局部变量、方法的形参，或者对静态方法的调用。

这几个类的变化会影响到Person类，但反之不会。



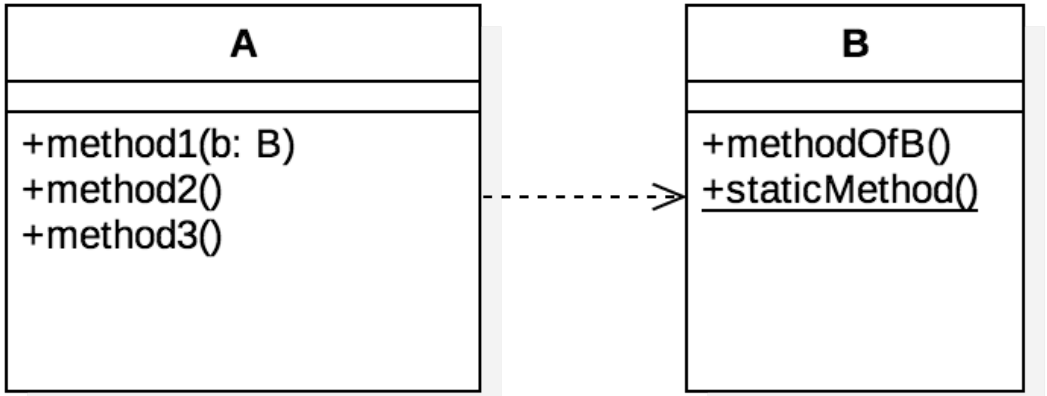
❑ 依赖 Dependency - 总结

简单的理解，就是一个类A中的方法使用到了另一个类B。

这种使用关系是具有偶然性的、临时性的、非常弱的，但是B类的变化会影响到A。

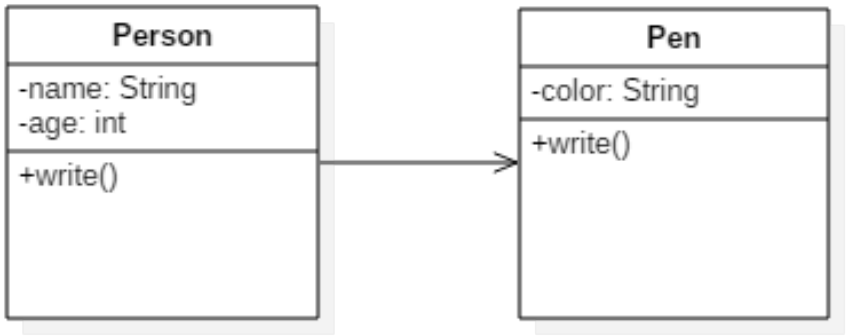
依赖是类之间关系最弱的一种形式。

```
public class A{
    void method1(B b){
        …
    }

    void method2(){
        B b = new B();
        b.methodOfB();
    }

    String method3(){
        return B.sMethod();
    }
}
```



❑ 关联 Association - 举例

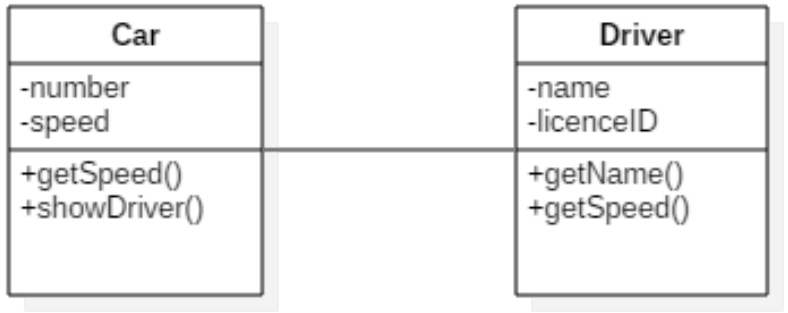
```
public class Person{
    private Pen pen;

    void write(){
        this.pen.write();
    }
}
```



❑ 关联 Association - 举例

```
public class Car{
    private Driver driver;

    public void showDriver(){

    print(driver.getName());
    }
}
```

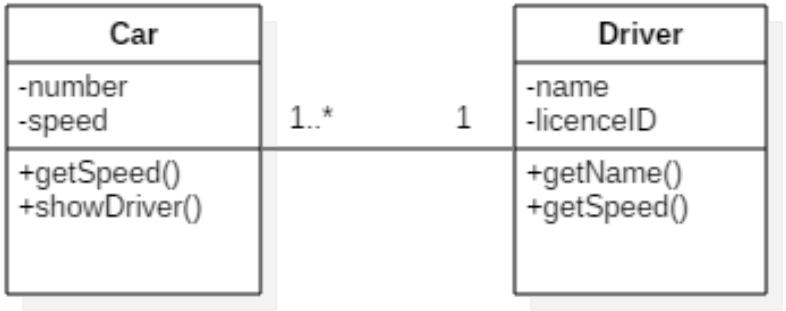
```
public class Driver{
    private Car car;

    void getSpeed(){
        this.car.getSpeed();
    }
}
```



❑ 关联 Association - 举例

```
public class Car{
    private Driver driver;

    public void showDriver(){

    print(driver.getName());
    }
}
```

```
public class Driver{
    private List<Car> cars;

    Car getCarByNumber(int n){
        …
    }
}
```



❑ 关联 Association - 观察

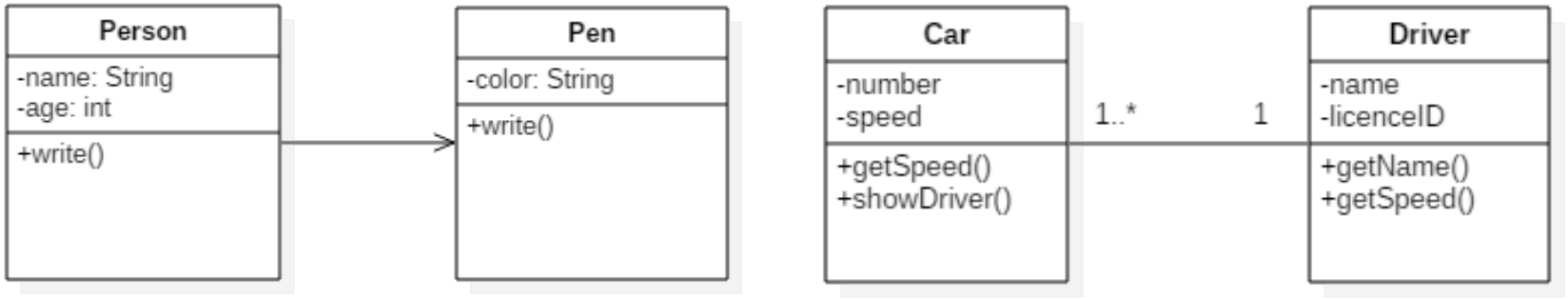
一个类作为另一个类的成员变量出现

关联可以是单向的(导航)，也可以是双向的

关联可以是1对1的，也可以是1对多的，也可以是多对多的

一个类使用了被关联类的属性和方法来实现自己的功能

被关联的类若发生改动，则使用它的类也会发生变化



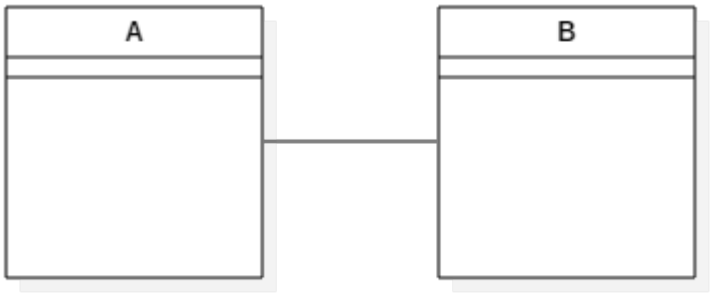
❑ 关联 Association - 总结

关联体现的是两个类之间比依赖更强的一种关系，一般是长期性的。

被关联类B以类属性的形式出现在关联类A中，就叫关联关系。

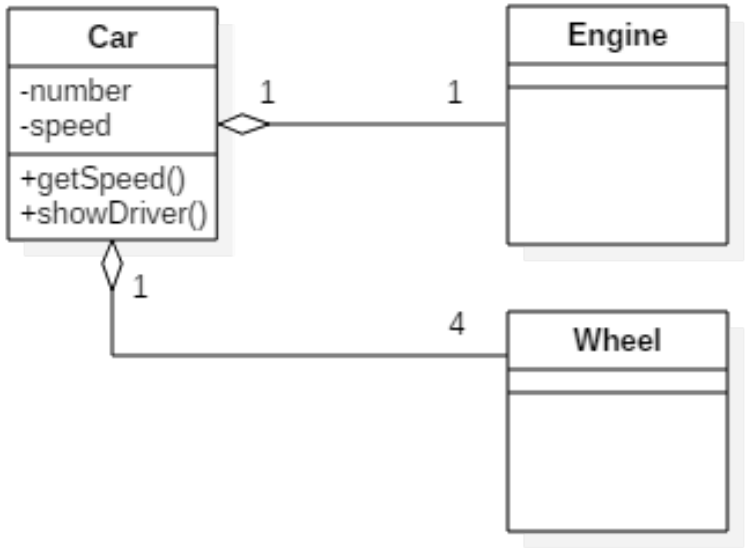
关联可以是单向（箭头指向被关联类）、双向的（无箭头）。

关联可以是1对1的，也可以是1对多（数组）的，也可以是多对多的，也可以自关联



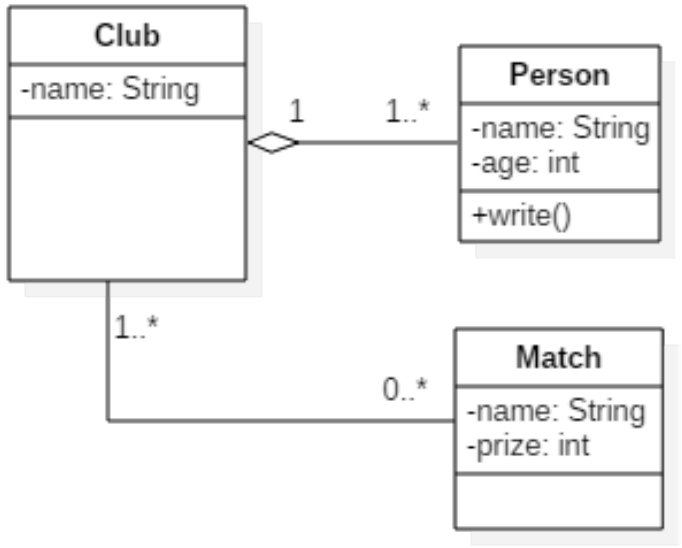
❑ 聚合 Aggregation - 举例

```
public class Engine{
    …
}

public class Wheel{
    …
}

public class Car{
    private Engine theEngine;
    private List<Wheel> wheels;
    …
}
```



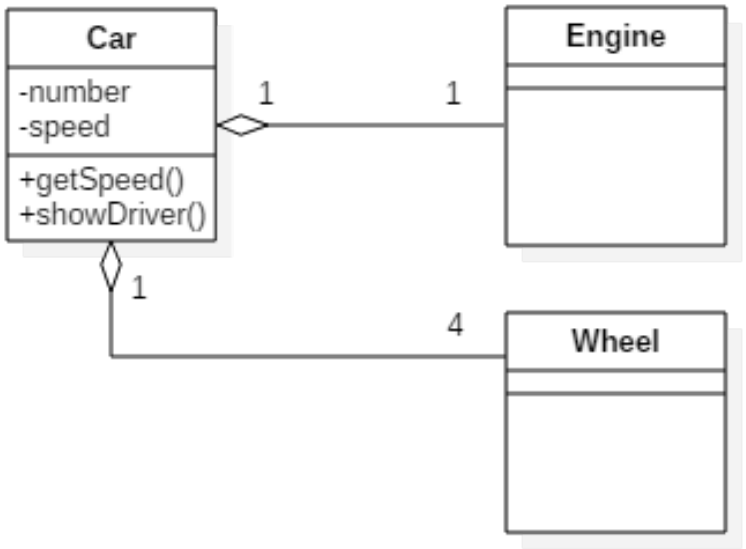
❑ 聚合 Aggregation - 举例

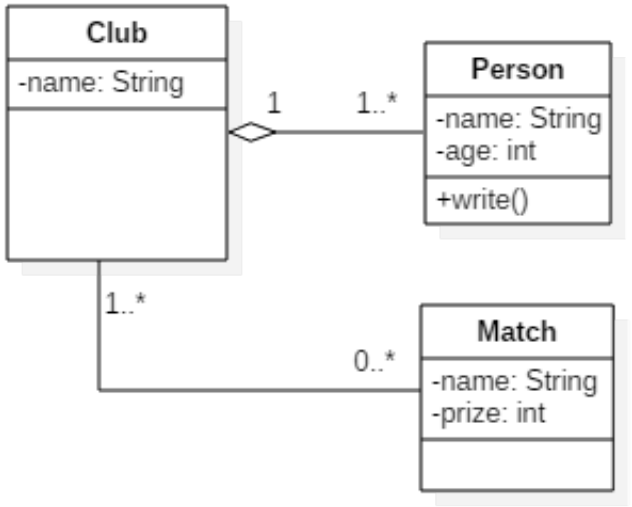
```
public class Match{
    private String name;
    private int prize;
}

public class Person{
    private String name;
    private int age;
}

public class Club{
    private List<Person> members;
    private List<Match> matches;
    …
}
```



❑ 聚合 Aggregation - 观察

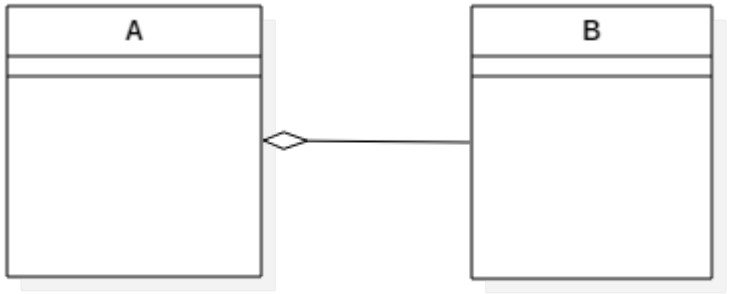
- 与关联关系一样，聚合关系也是通过将其它类作为成员变量实现的。
- 在聚合关系中，两个类是处在不同层次上的，一个代表整体，另一个代表部分。
- 虽然是整体和部分的关系，但是整体与部分之间是可分离的，他们可以具有各自的生命周期，部分可以属于多个整体对象，也可以被多个整体对象共享。





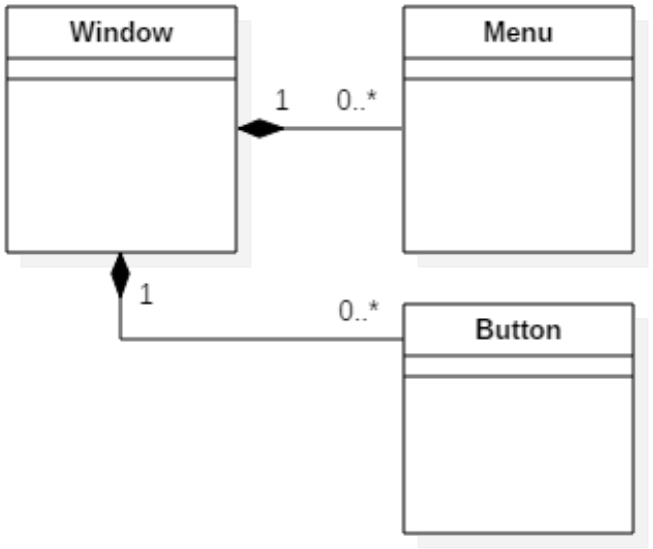
❑ 聚合 Aggregation - 总结

- 聚合是关联关系的一种特例，它体现的是整体与部分的拥有关系，即has-a的关系；
- 普通的关联关系中，A类和B类没有必然的联系，而聚合中，需要B类是A类的一部分，是一种”has-a“的关系，即 A has-a B;
- 但是，has 不是 must has，整体与部分之间是可分离的，他们可以具有各自的生命周期;



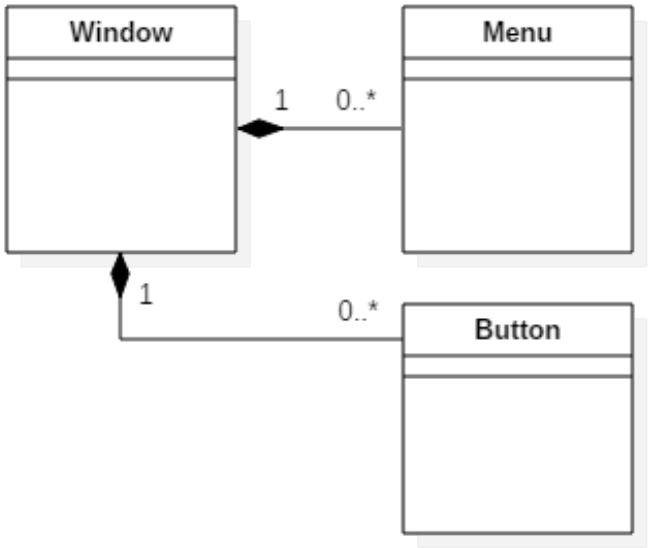
❑ 组合 Composition - 举例

```
public class Menu{
    private String name;
    private List<String> options;
}

public class Button{
    private String label;
}

public class Window{
    private List<Menu> menu;
    private List<Button> matches;
    …
}
```



❑ 组合 Composition - 观察

- 与关联关系一样，组合关系也是通过将其它类作为成员变量实现的。
- 在组合关系中，两个类是有着不同层级的关系的，一个代表整体，另一个代表部分。
- 整体与部分之间是不可分离的，整体的生命周期结束也就意味着部分的生命周期结束 - 同生共死

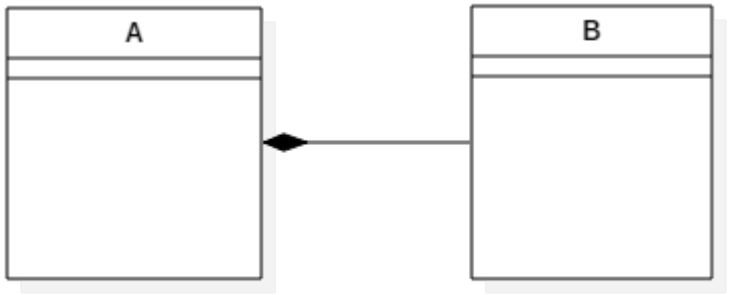


❑ 组合 Composition - 总结

组合也是关联关系的一种特例，他体现的是一种contains-a的关系，这种关系比聚合更强，也称为强聚合。

组合同样体现整体与部分间的关系，但此时整体与部分是不可分的，整体的生命周期结束也就意味着部分的生命周期结束。

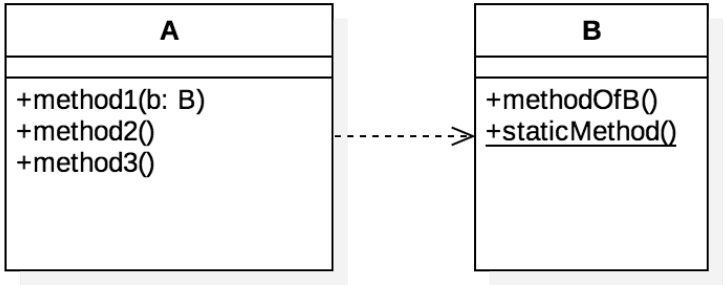
组合要求普通的聚合关系中代表整体的对象负责代表部分对象的生命周期，因此，组合关系是不能共享的。换言之，代表部分的对象在每一个时刻只能与一个对象形成组合关系，由后者负责其生命周期。



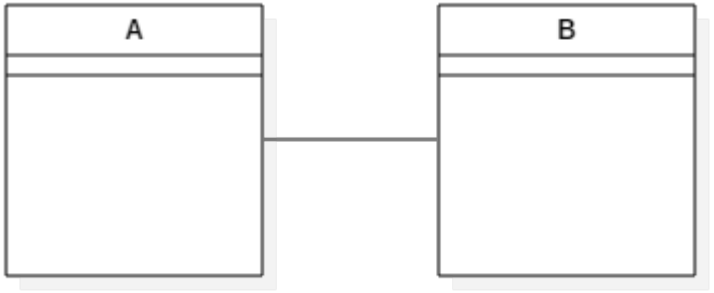
2. 关系的比较

❑ 关系汇总

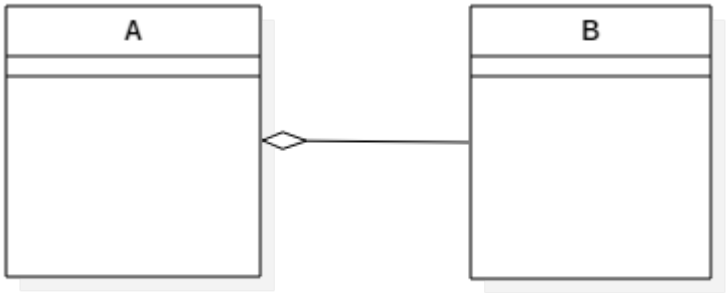
依赖 uses a



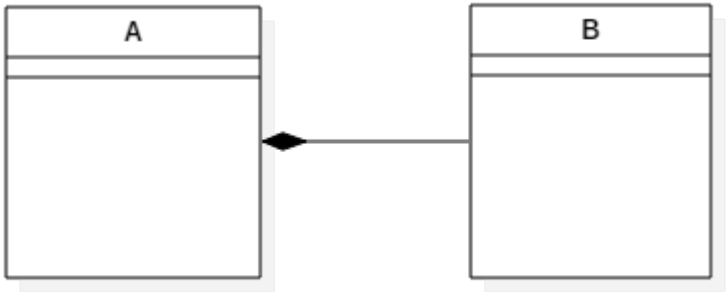
关联 uses a



聚合 has a

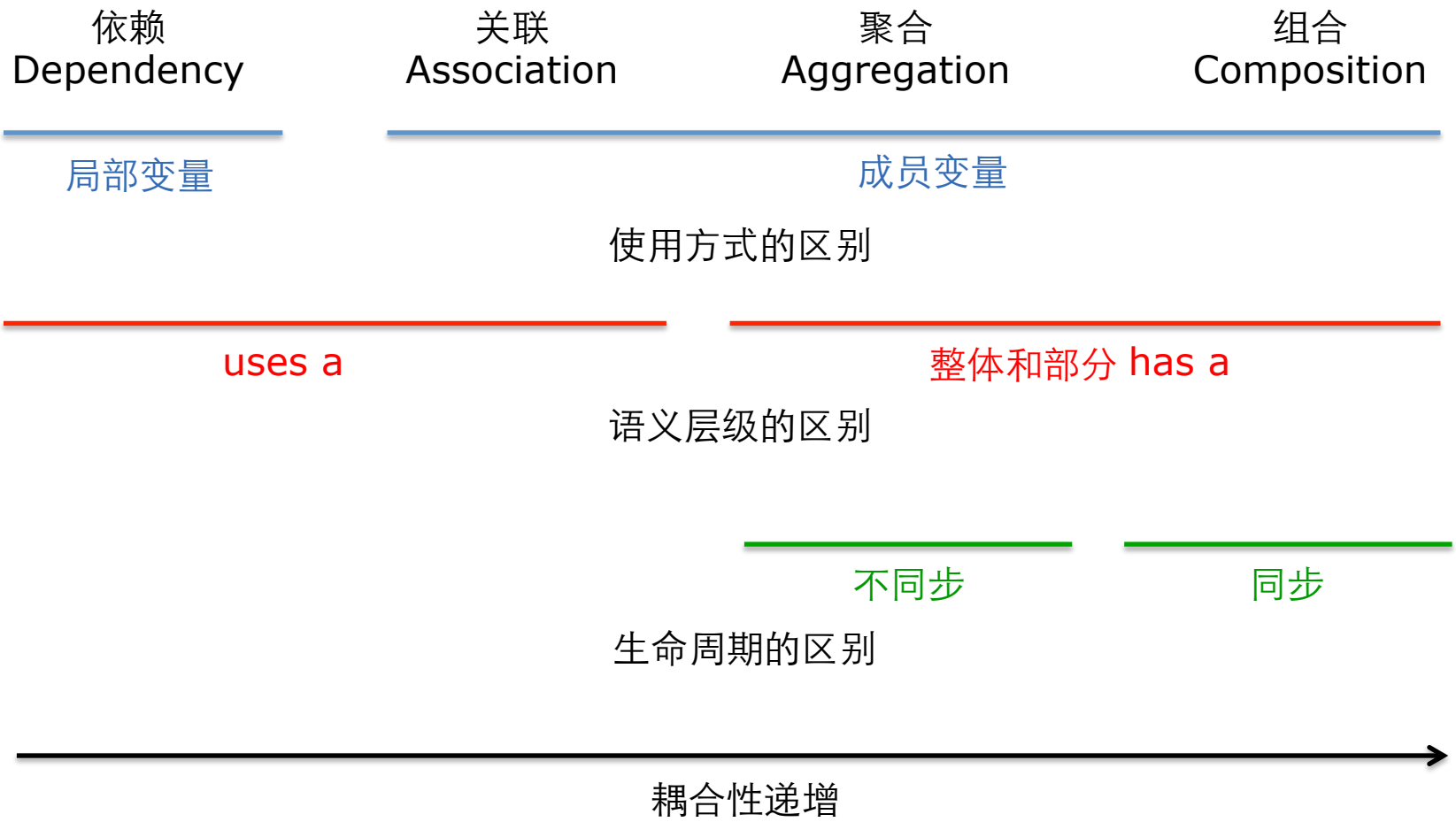


组合 contains a



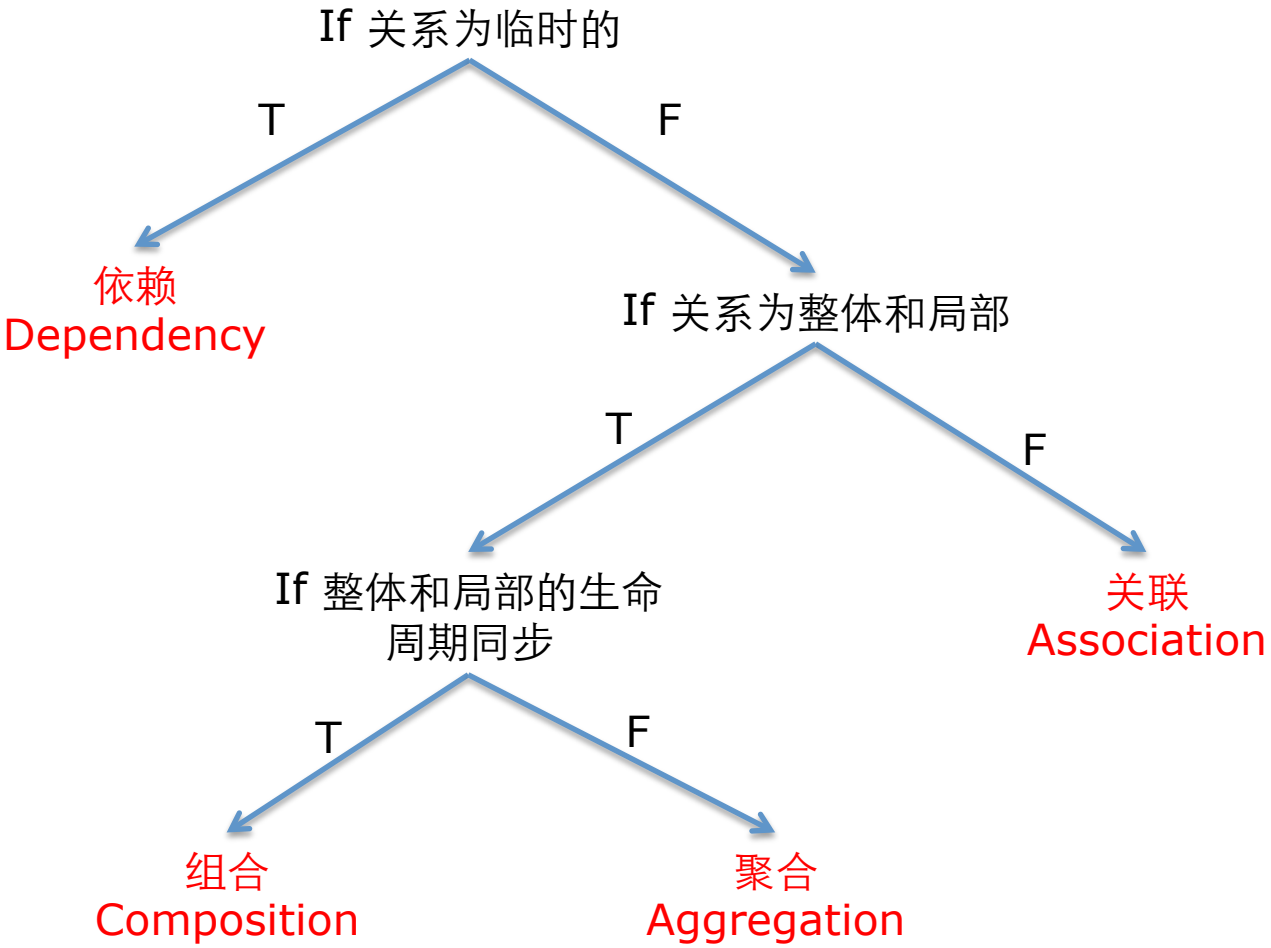
2. 关系的比较

❑ 关系汇总


依赖
Dependency
关联
Association
聚合
Aggregation
组合
Composition
局部变量
成员变量
使用方式的区别
uses a
整体和部分 has a
语义层级的区别
不同步
同步
生命周期的区别
耦合性递增

2. 关系的比较

- 关系判断


If 关系为临时的
T
F
依赖
Dependency
If 关系为整体和局部
T
F
If 整体和局部的生命
周期同步
关联
Association
T
F
组合
Composition
聚合
Aggregation

练习1

判断下列语句中的出现的概念分别属于哪种关系?

- 我有一所大房子，有很大的落地窗户;
- 朋友的朋友，我们最后的关联;
- 我的征途是星辰大海;
- 谁把我自行车轮子弄走了?
- 绝望之中我发现了一辆吉普车，毫不犹豫地就跳了上去。

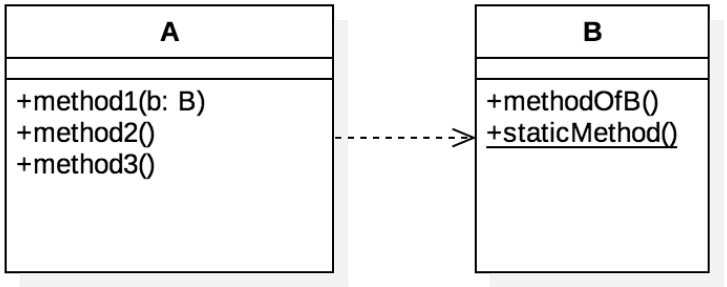
2. 关系的比较

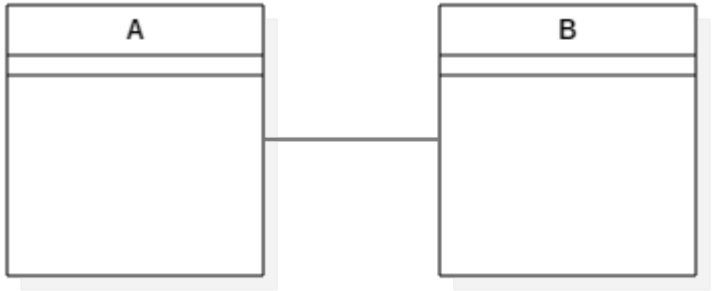
❑ 关联与依赖

使用频率：关联 > 依赖 > 继承

类关系的强度： 关联 > 依赖

时间角度： 持久 / 临时





2. 关系的比较

❑ 组合与继承 - 扩展新的功能

继承

优点:
- 子类能自动继承父类的接口

- 创建子类的对象时，无须手动创建父类的对象

缺点:
-破坏封装，子类与父类之间紧密耦合

- 支持扩展，但是往往以增加系统结构的复杂度为代价

- 不支持动态继承。在运行时，子类无法选择不同的父类

- 子类不能改变父类的接口

组合

优点:
- 不破坏封装，整体类与局部类之间松耦合，彼此相对独立

- 具有较好的可扩展性

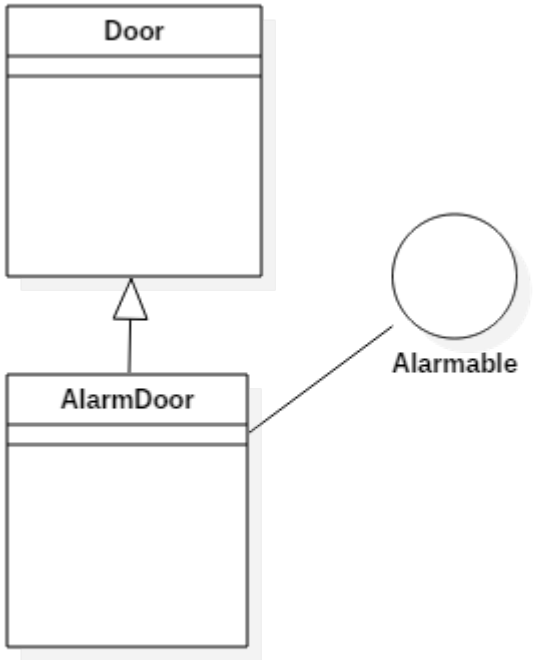
- 支持动态组合，在运行时整体对象可以选择不同类型的局部对象

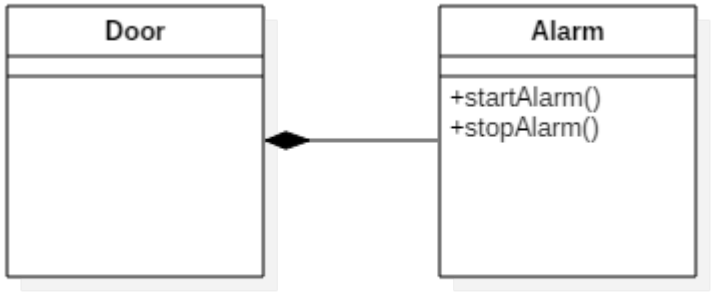
- 整体类可以对局部类进行包装，封装局部类的接口，提供新的接口

缺点:
- 整体类不能自动获得和局部类同样的接口

- 创建整体类的对象时，需要创建所有局部类的对象

2. 关系的比较





面向对象中有一个重要原则『**多用组合、少用继承**』或者说『**组合优于继承**』。

练习2

请为下面的问题域建立相应的模型：

A. 自行车分为山地越野型和城市公路型，以及一般型；

B. 自行车有轮子、刹车、铃铛；

C. 自行车上可以有二维码和小广告；

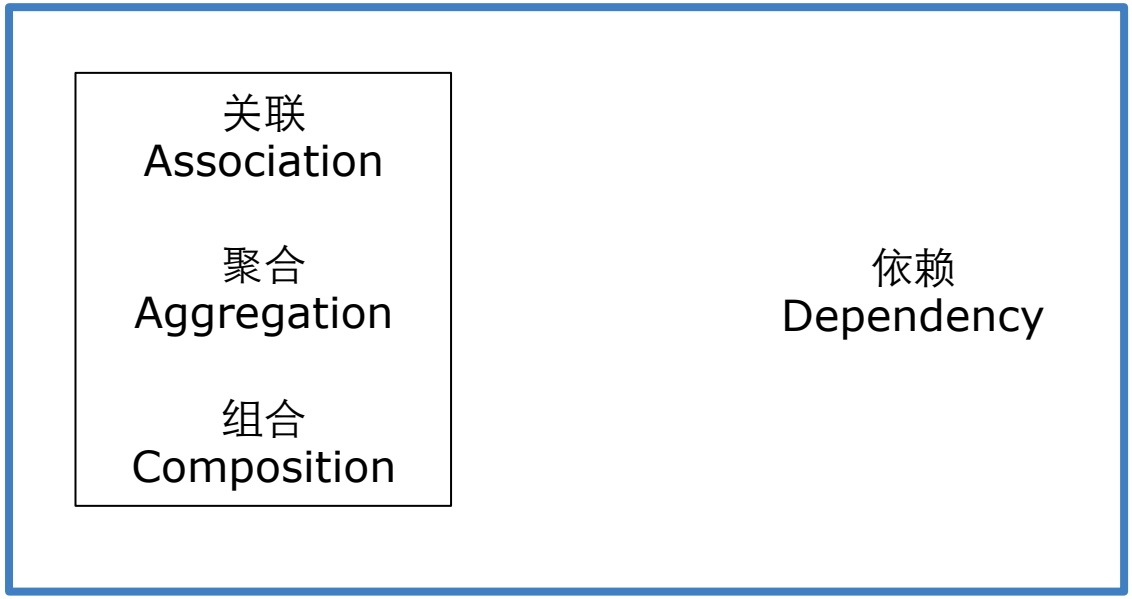
D. 打气筒Pump可以给车轮子打气；

E. 年满12岁的公民都可以骑小黄车；

3. 总结

类的关系

- 1+3+1
- 关系的比较与选择



下节预告

UML 第一部分