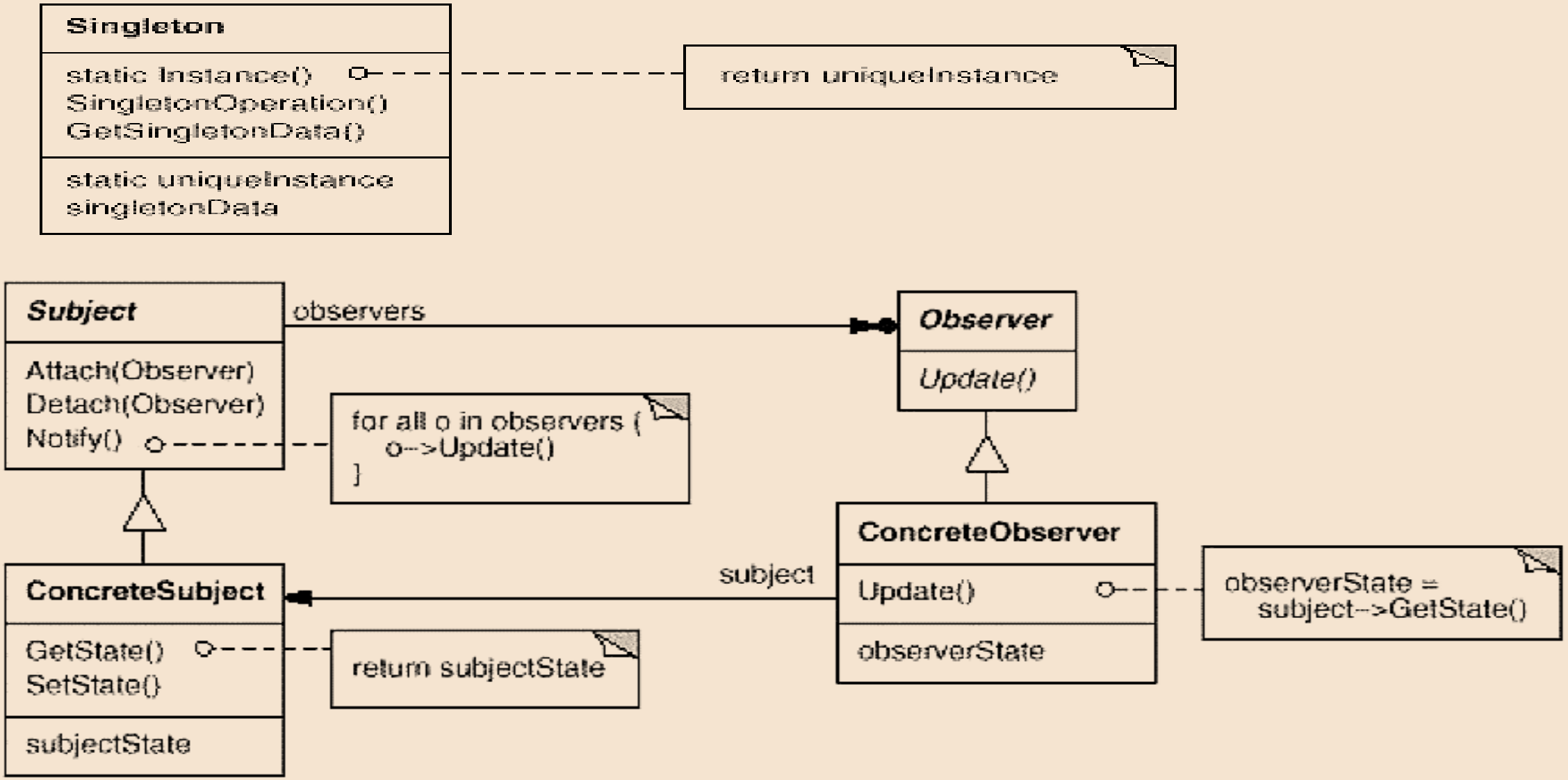
Object Oriented Analysis & Design
面向对象分析与设计

Lecture_10 GOF设计模式（二）

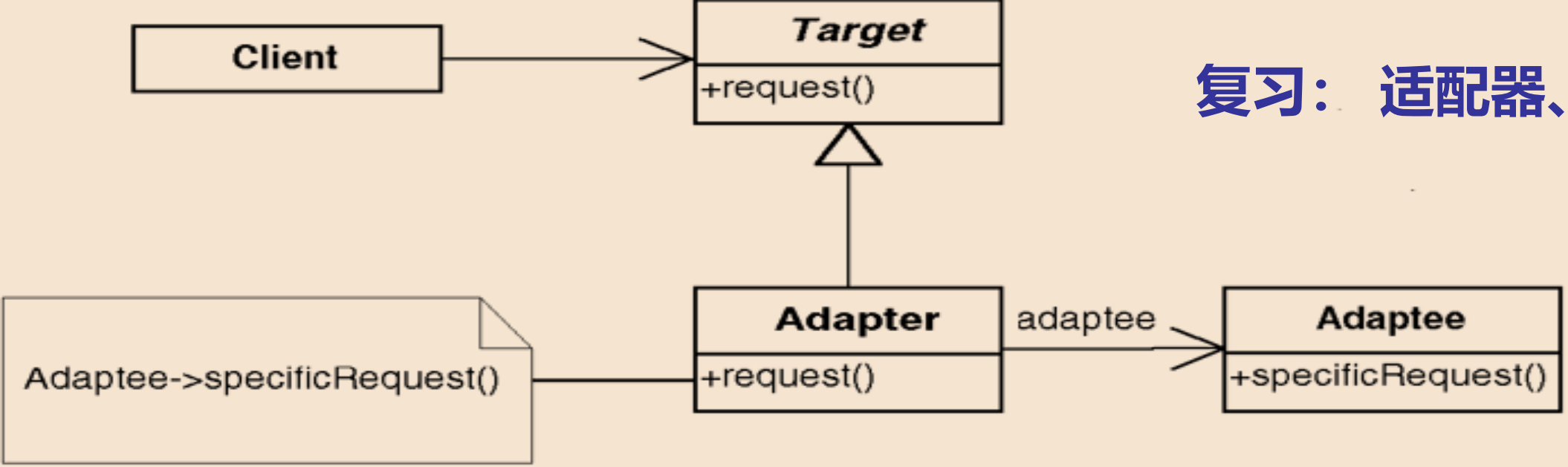
1）策略模式　　2）工厂

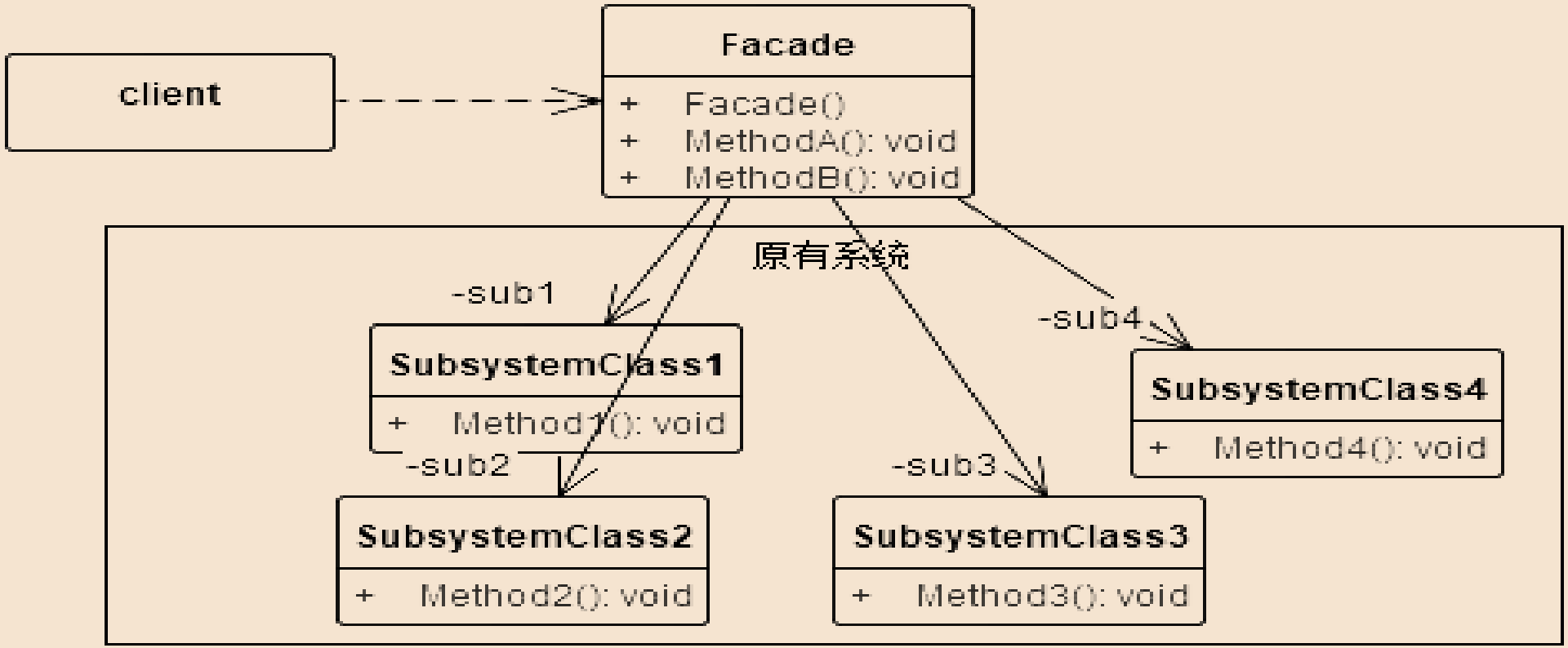
主讲: 姜宁康 博士

复习：单实例、观察者设计模式



复习： 适配器、外观


Client
Target
+request()
Adapter
+request()
adaptee
Adaptee
+specificRequest()
Adaptee->specificRequest()


client
Facade
+ Facade()
+ MethodA(): void
+ MethodB(): void
原有系统
-sub1
-sub2
-sub3
-sub4
SubsystemClass1
+ Method1(): void
SubsystemClass2
+ Method2(): void
SubsystemClass3
+ Method3(): void
SubsystemClass4
+ Method4(): void

复习：OO原则 Open-Closed Principle (OCP)

- "Software Systems change during their life time"
  - both better designs and poor designs have to face the changes
  - good designs are stable

Software entities should be open for extension, but closed for modification (OCP)

two criteria:

Open for Extension - the behavior can be extended to meet new requirements
CLosed for Modification - the source code of the module is not allowed to change

OCP attacks software rigidity and fragility!
When one change causes a cascade of changes

1、策略模式... Strategy Pattern

- 在POS系统中，有时需要实行价格优惠，该如何处理?
  - 对普通客户或新客户报全价
  - 对老客户统一折扣5%
  - 对大客户统一折扣10%

- 注：课件来自Head-First OOAD课程资料

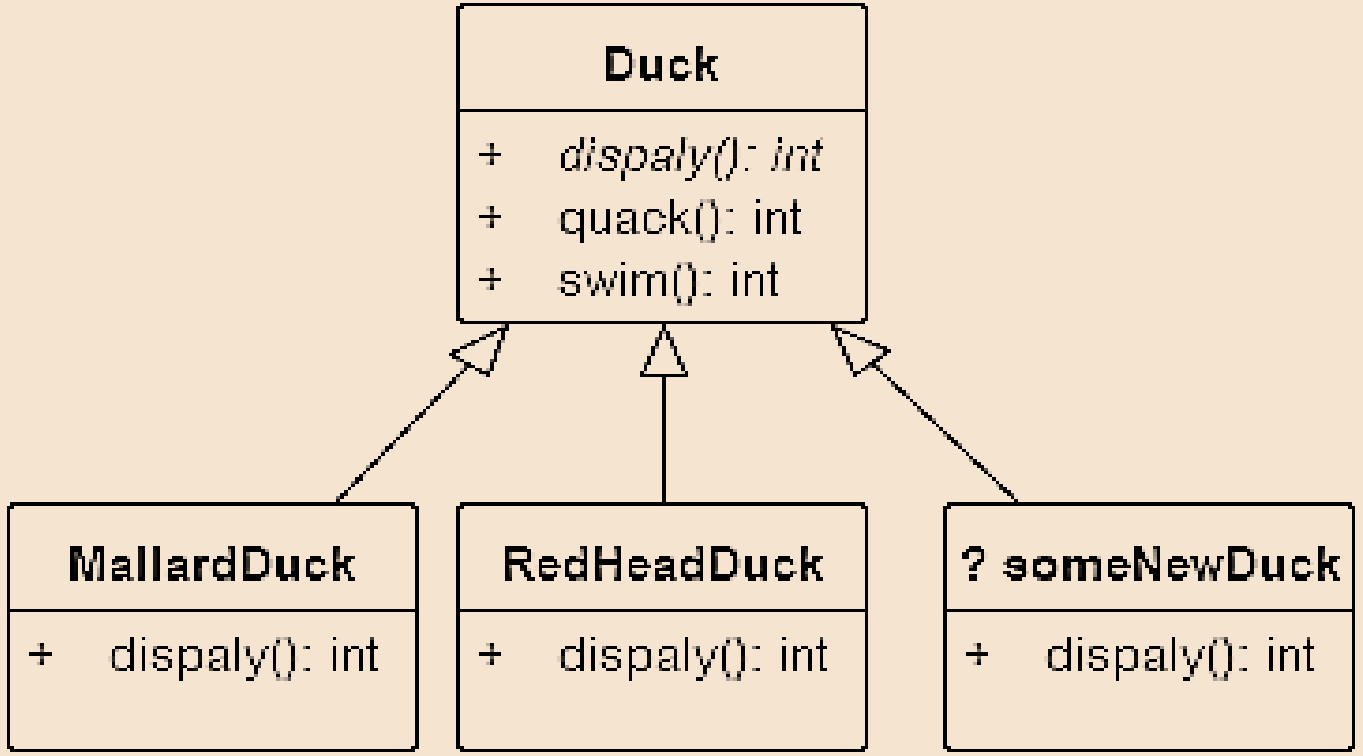
1.1 假设背景

- Joe是一个OO程序员，所在的公司正在开发一款仿真游戏软件“SimUDuck”，他的任务是完成游戏的重要功能 Joe works at a company that produces a simulation game called SimUDuck. He is an OO Programmer and his duty is to implement the necessary functionality for the game

- 游戏具备以下的需求规格说明 The game should have the following specifications:
  - 存在多种不同类型的鸭子 A variety of different ducks should be integrated into the game
  - 鸭子会游泳 The ducks should swim
  - 鸭子会叫“quack、quack” The duck should quack

1.2 初步的设计：简单

- **A First Design for the Duck Simulator Game**



All ducks quack() and swim(). The superclass takes care of the implementation 父类实施了共同的功能

The display()method is abstract since all the duck subtypes look different 因为不同的鸭子有不同的外形，子类覆盖

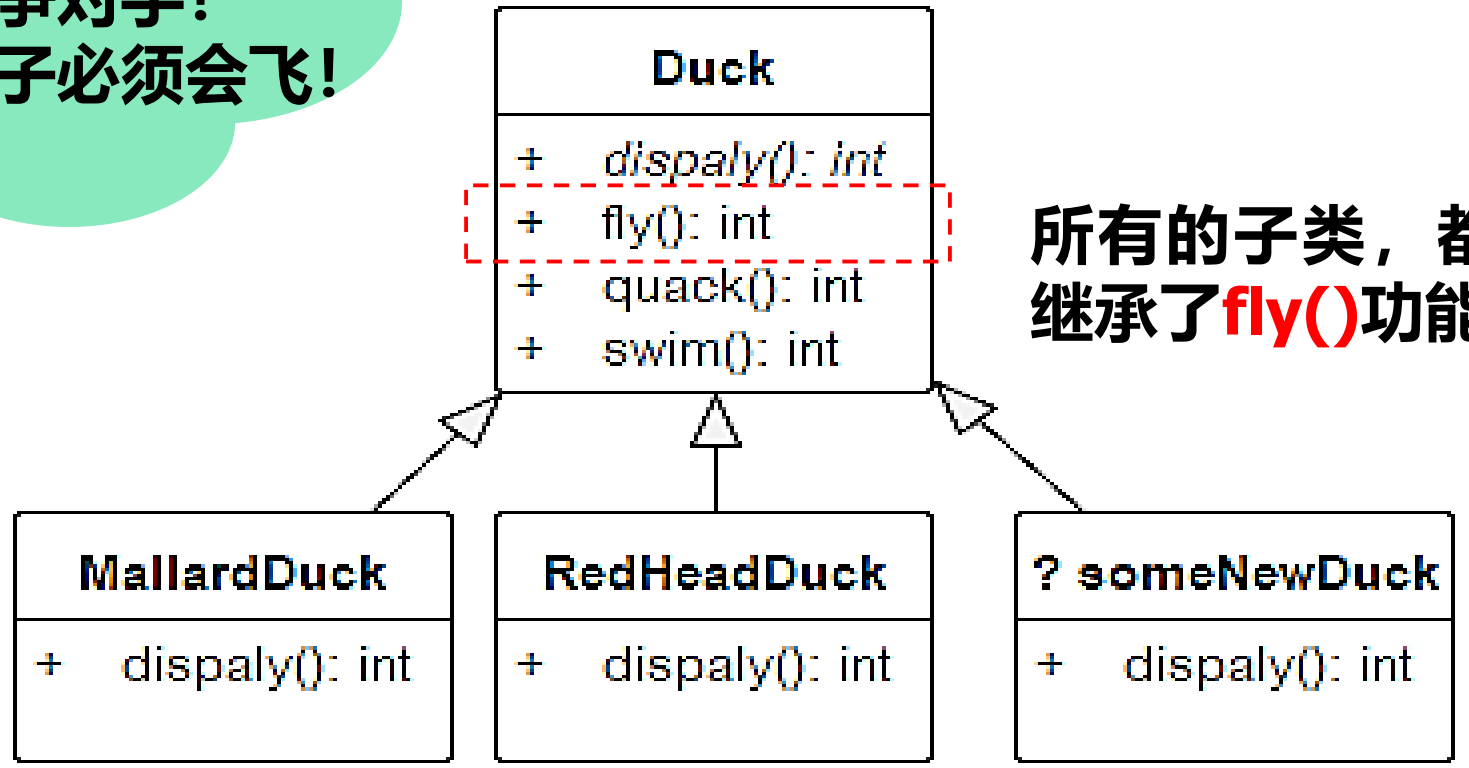
Each duck subtype is responsible for implementing its own display() behavior for how it looks on the screen

Lots of other types of ducks inherit from the Duck type

1.3 需要鸭子飞起来...

Joe, 公司董事会要求，
我们的产品必须战胜竞争对手！
从现在开始，我们的鸭子必须会飞！

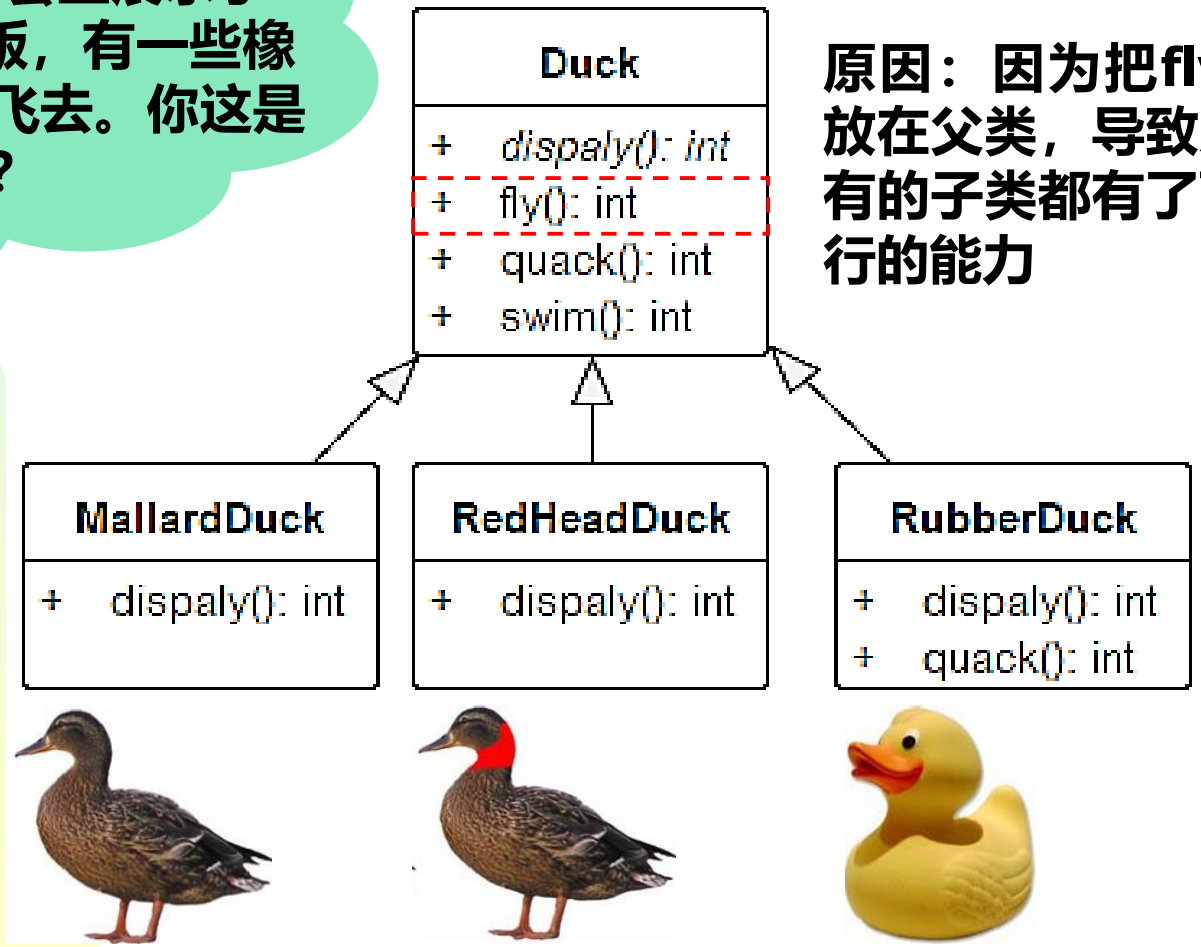
新方案



所有的子类，都继承了fly()功能

1.4 出了一点小问题：橡皮鸭子也飞起来了..

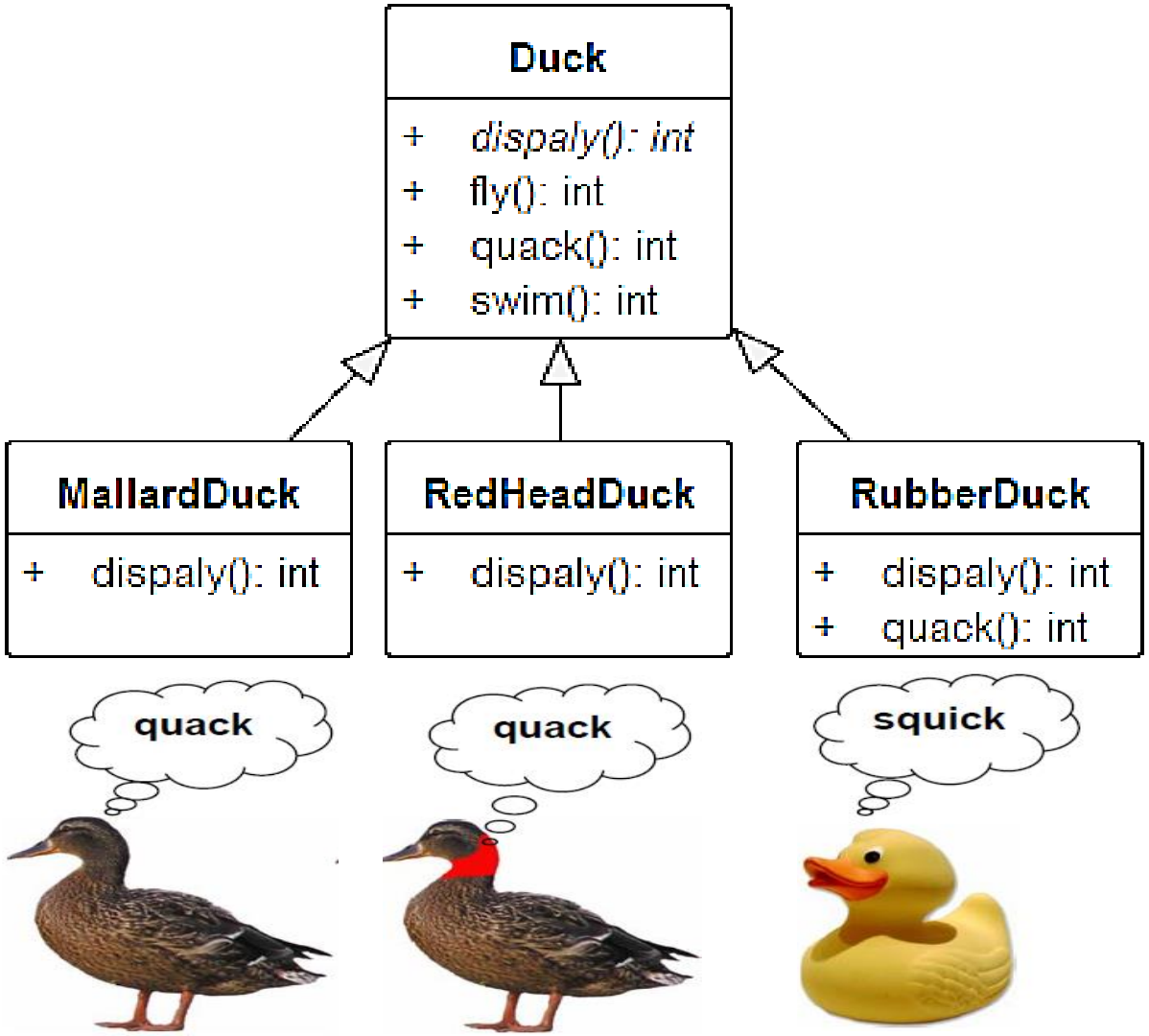
Joe, 我正在开董事会。会上展示了我们游戏软件的demo版，有一些橡皮鸭子也在屏幕上飞来飞去。你这是开玩笑还是真这么做的?



原因：因为把fly()放在父类，导致所有的子类都有了飞行的能力

Joe：知道了，这是设计上的一个小问题，马上改好。我倒是觉得这样也蛮“酷”的!

1.5 继承机制起作用了

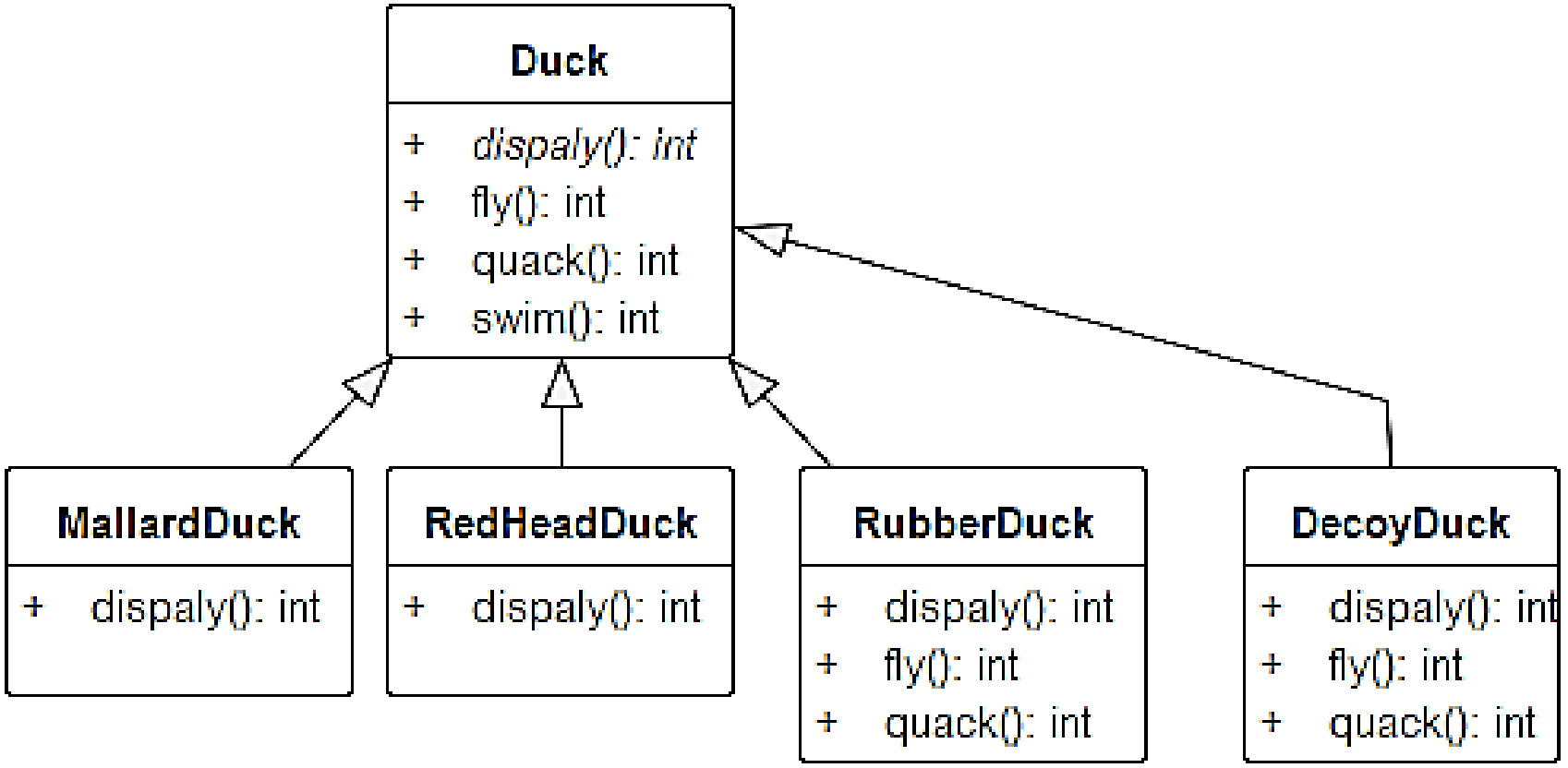



```
void Duck::quack(){
    cout << ”quack, quack” << endl;
}

void RubberDuck::quack(){
   cout << ”squick, squick” << endl;
}
```

同样地， 我们也可以覆盖实现橡皮鸭的fly()功能

```
void Duck::fly(){
     // fly implementation
}

void RubberDuck::fly(){
   // do nothing
}
```

1.6 再增加一只鸭子：诱导鸭



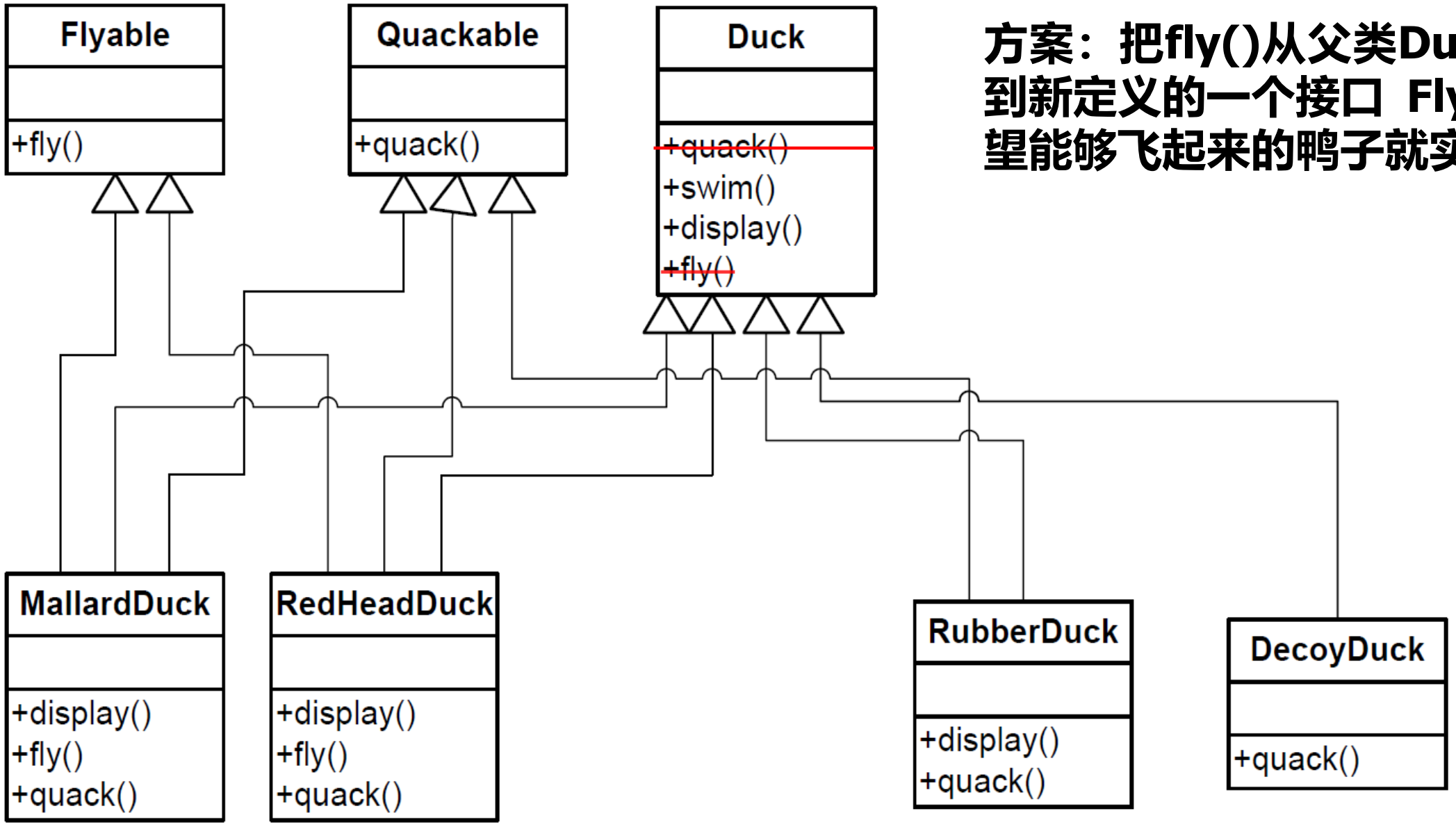

```
void DecoyDuck::quack(){
    // do nothing;
}

void DecoyDuck::fly(){
    // do nothing
}
```

需要修改的函数总量 = 每增加一个特殊功能*2 * 每增加一类鸭子*2，
好辛苦啊！好容易出哦！

1.7 设计原则

- 软件行业“公理”
  - 软件项目中，唯一不变的事情就是“变化”
- 解决方法
  - 拥抱变化，让“变化”成为你的设计的一部分
  - 标识变化点，把它们与系统其余部分隔离开来
    - 比较：GRASP原则： Protected Variations 隔离变化
- “SimUDucK”游戏的需求总在变化，刚才的方案有点问题！
- 解决方法
  - 把变化的部分，封装起来

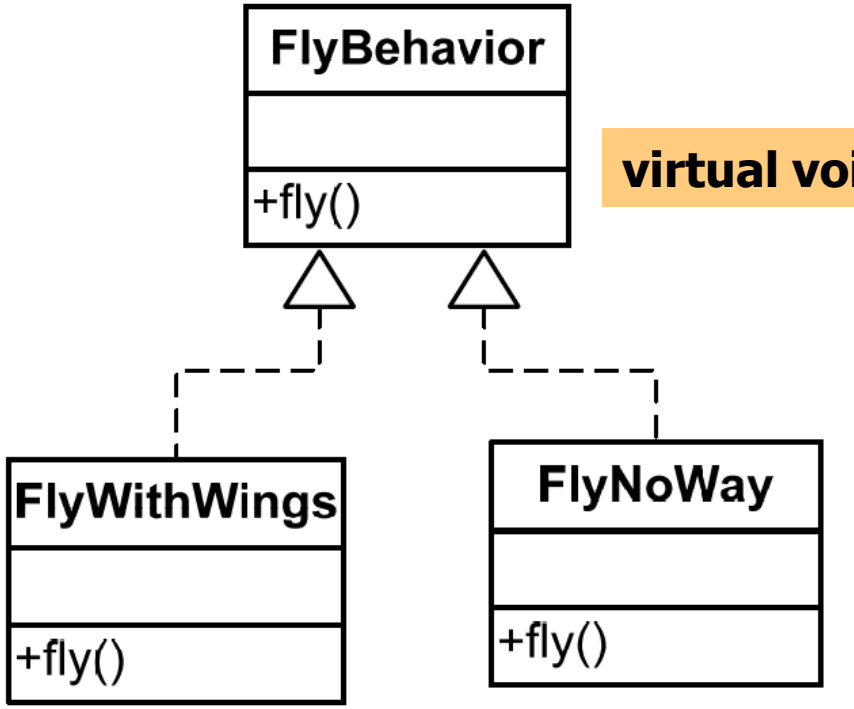
1.8 尝试利用接口的威力



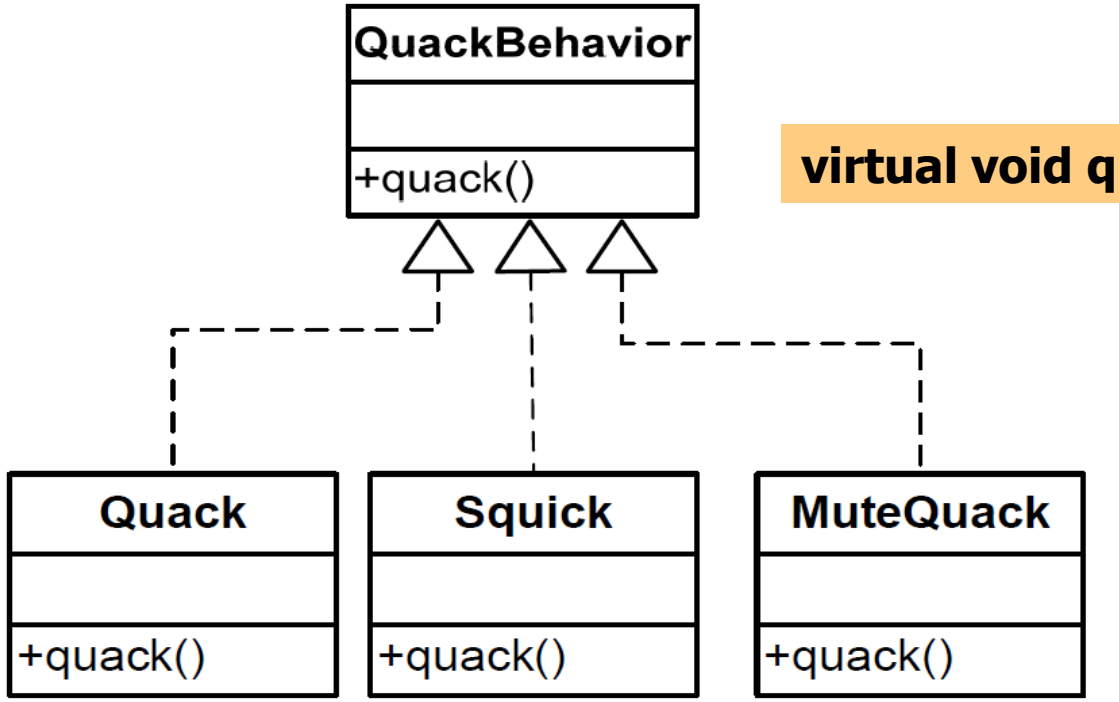
方案：把fly()从父类Duck中剥离，放到新定义的一个接口 Flyable，每个希望能够飞起来的鸭子就实现这个接口

1.9 Duck游戏哪些是会变化的?

- Fly() 和 quack() 行为在变化，所以，为每一种行为创建新类



virtual void fly()=0;



virtual void quack()=0;

```
void FlyWithWings::fly(){
   cout << "I'm flying!" << endl;
};
```

```
void FlyNoWay::fly(){
   cout << "I can't fly." << endl;
};
```

```
void Quack::quack(){
cout << "Quack" << endl;
};
```

```
void Squeak::quack(){
   cout << "Squeak" << endl;
};
```

```
void MuteQuack::quack(){
    cout << "....." << endl;
};
```

复习：面向对象设计原则

- 1、把变化的部分，封装起来

- 2、面向接口设计（编程），而不是面向实现设计（编程）
  Program to an interface，not to an implementation

请同学们思考一下，你会如何修改Duck游戏的设计方案?

- 本讲结束