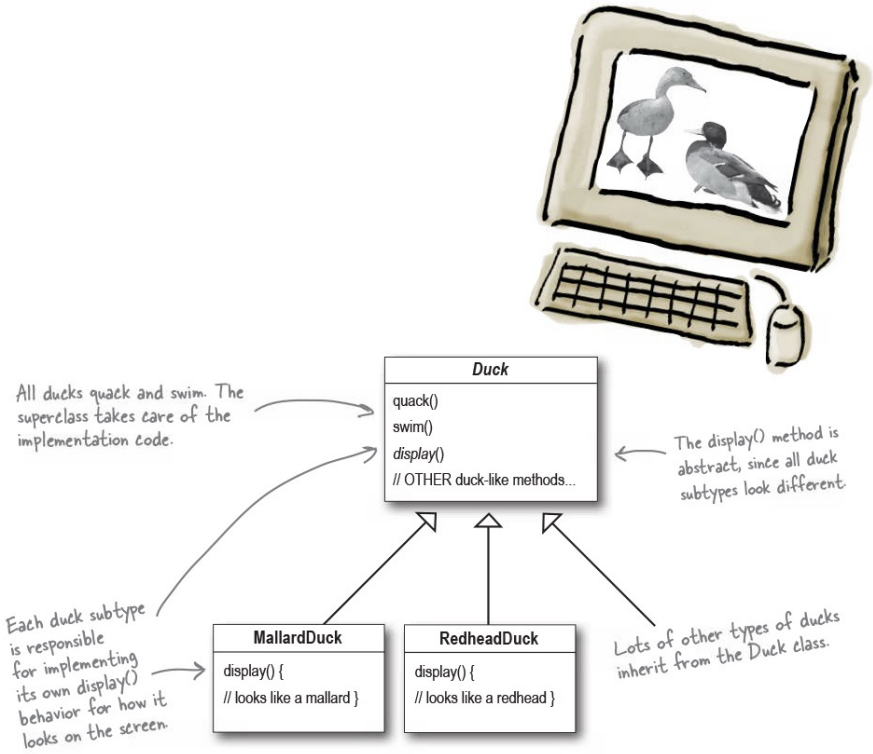
CS525 Advanced Software Development

Lesson 2 – The Strategy Pattern

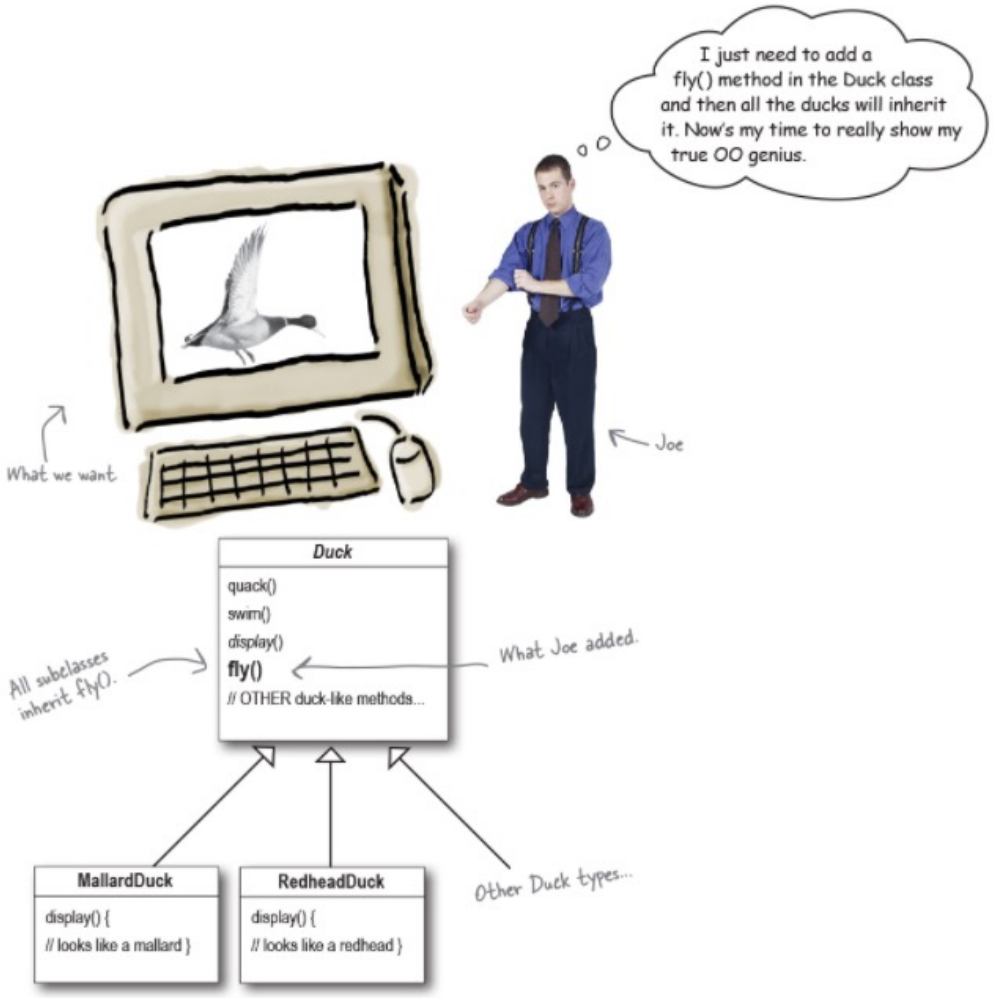
Design Patterns
Elements of Reusable Object-Oriented Software

Payman Salek, M.S.
March 2022

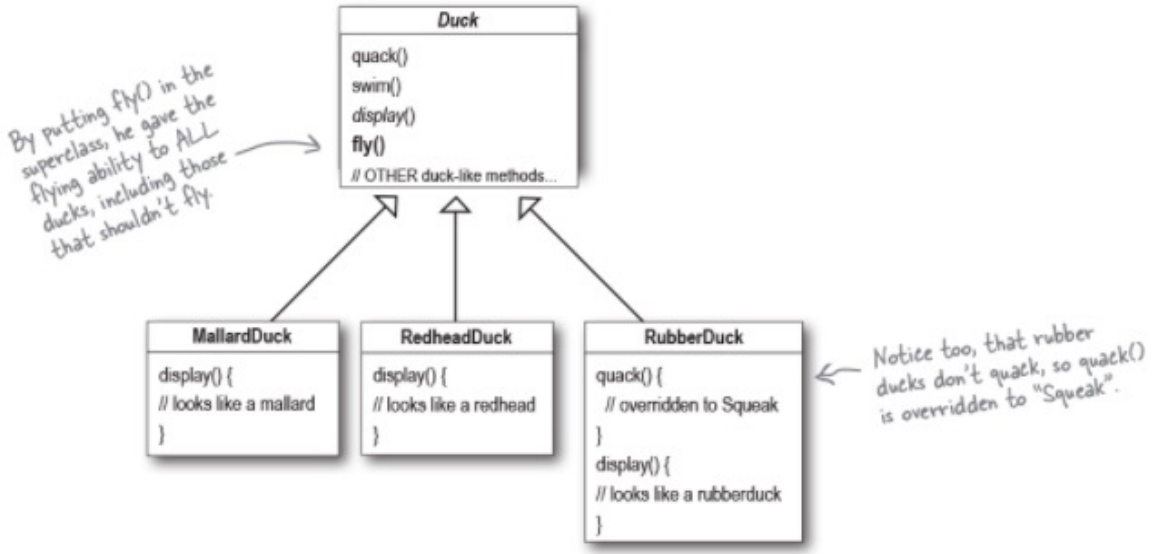
SimUDuck App



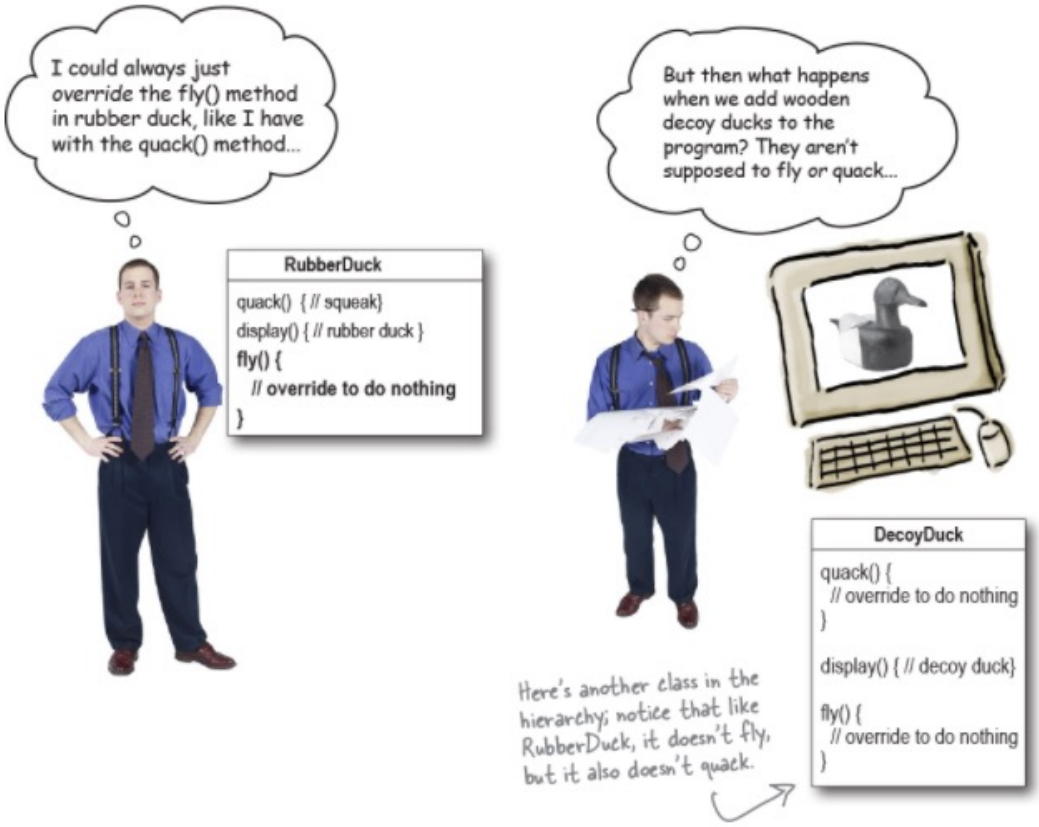
The only constant in SWE is "CHANGE"



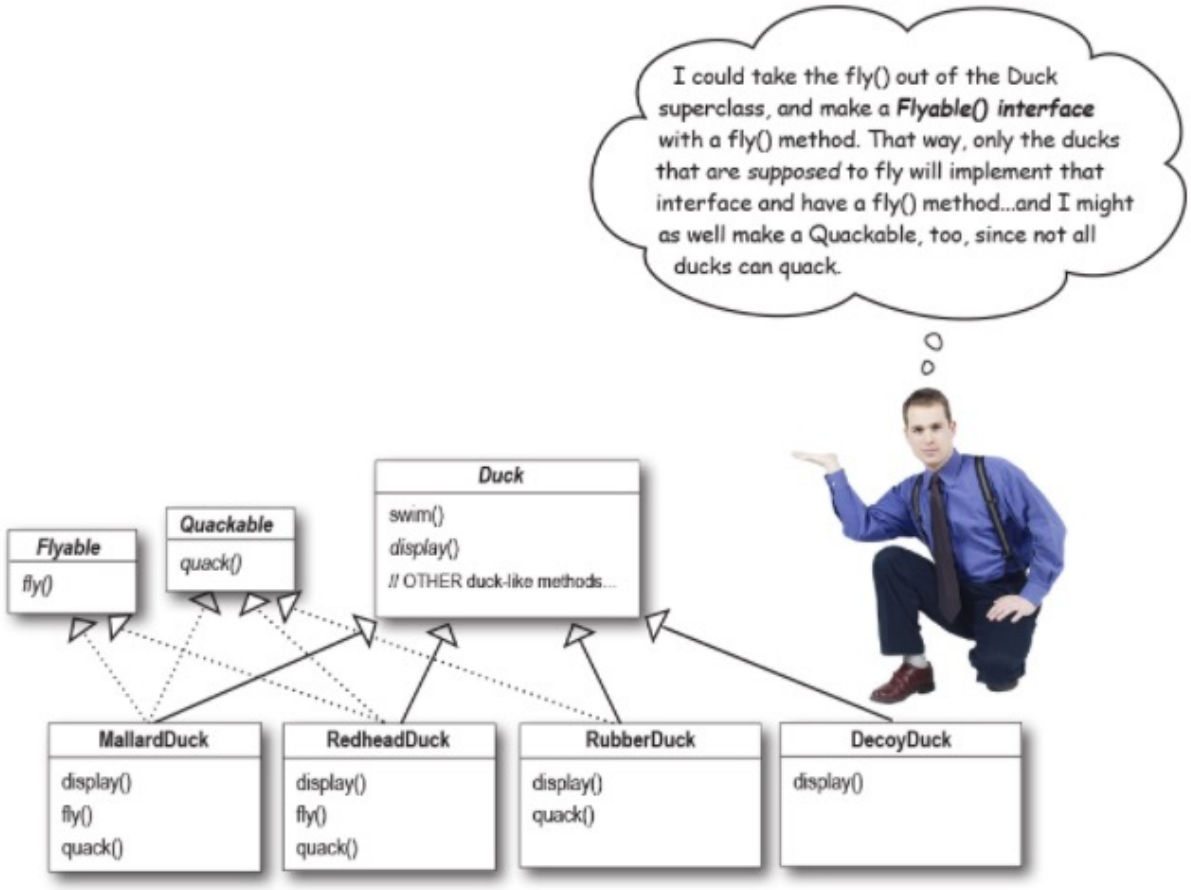
But the new change does not apply to all ducks!



Maybe there is a solution??



How about an interface??



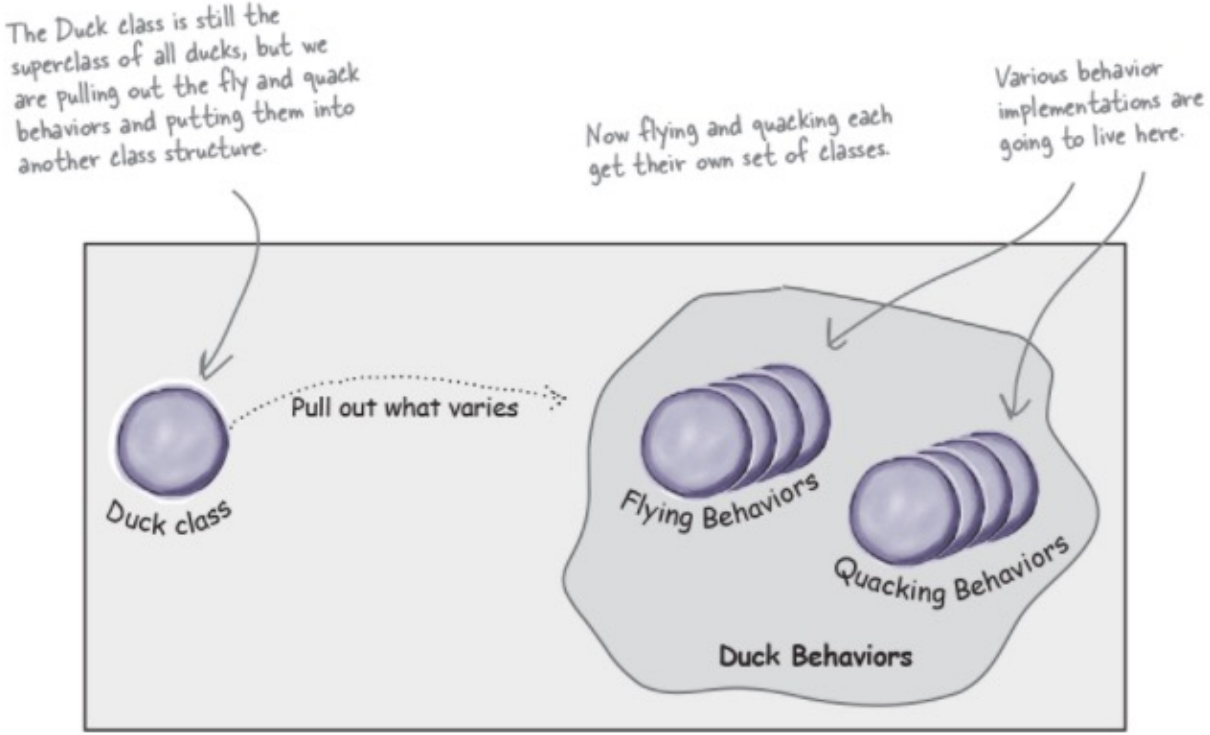
Houston we have a problem!

Lots of code duplication!!!

Design Principle

Identify the aspects of your application that vary and separate them from what stays the same.

Separate changing from non-changing

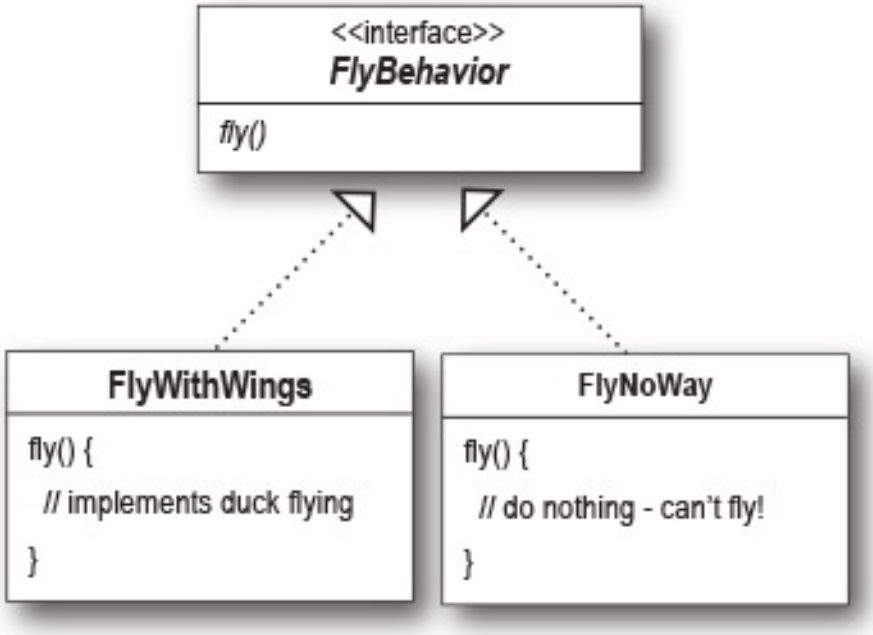


Design Principle

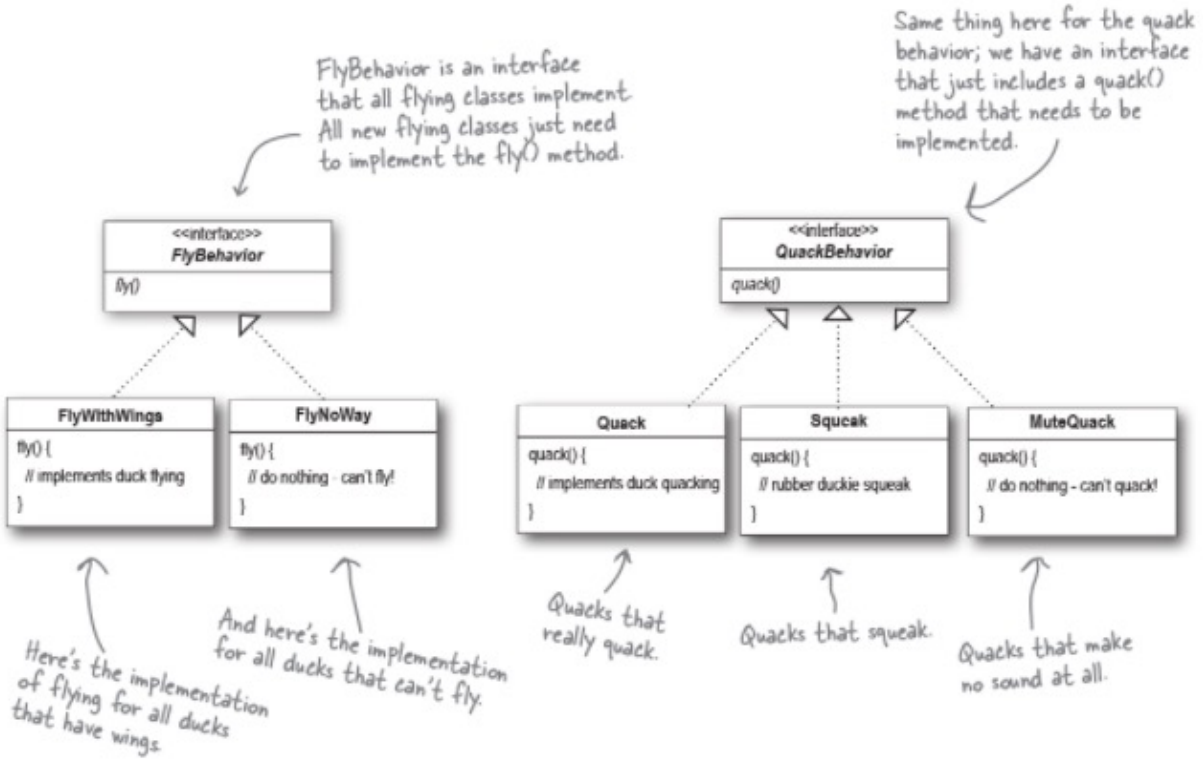
Program to interface, not an implementation.

Note: “Program to an interface” really means “Program to a supertype.”

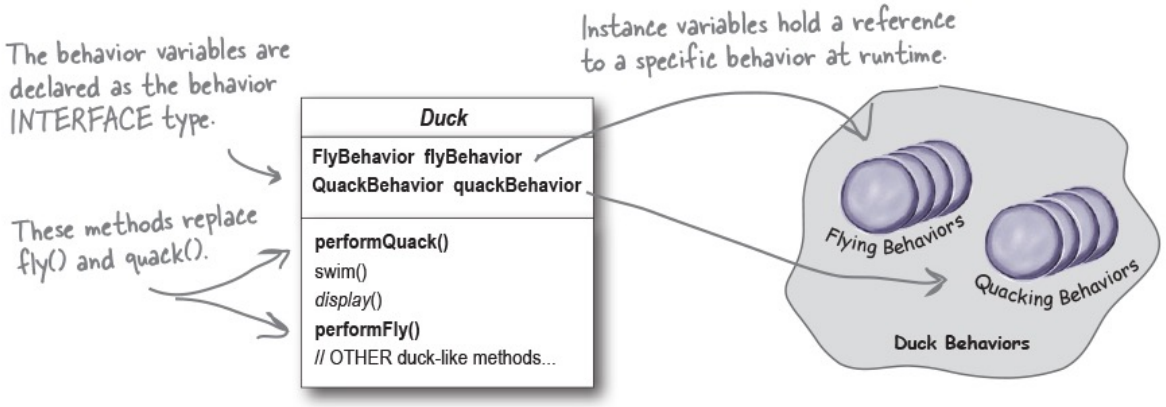
Program to interface



Extend to all changing aspects



Integrating the Duck behaviors



Practice

Can you write code for the previous slide?

```
public abstract class Duck {
    QuackBehavior quackBehavior;
    // more

    public void performQuack() {
      quackBehavior.quack();
    }
}
```

Each Duck has a reference to something that implements the QuackBehavior interface.

Rather than handling the quack behavior itself, the Duck object delegates that behavior to the object referenced by quackBehavior.

Solution

Solution

```
public class MallardDuck extends Duck {

    public MallardDuck() {
        quackBehavior = new Quack();
        flyBehavior = new FlyWithWings();
    }

    public void display() {
        System.out.println("I'm a real Mallard duck");
    }
}
```

A MallardDuck uses the Quack class to handle its quack, so when performQuack() is called, the responsibility for the quack is delegated to the Quack object and we get a real quack.

And it uses FlyWithWings as its FlyBehavior type.

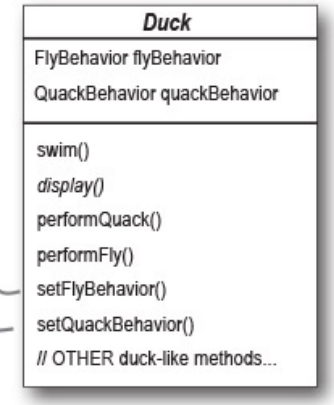
Remember, MallardDuck inherits the quackBehavior and flyBehavior instance variables from class Duck.

```
public void setFlyBehavior(FlyBehavior fb) {
    flyBehavior = fb;
}

public void setQuackBehavior(QuackBehavior qb) {
    quackBehavior = qb;
}
```

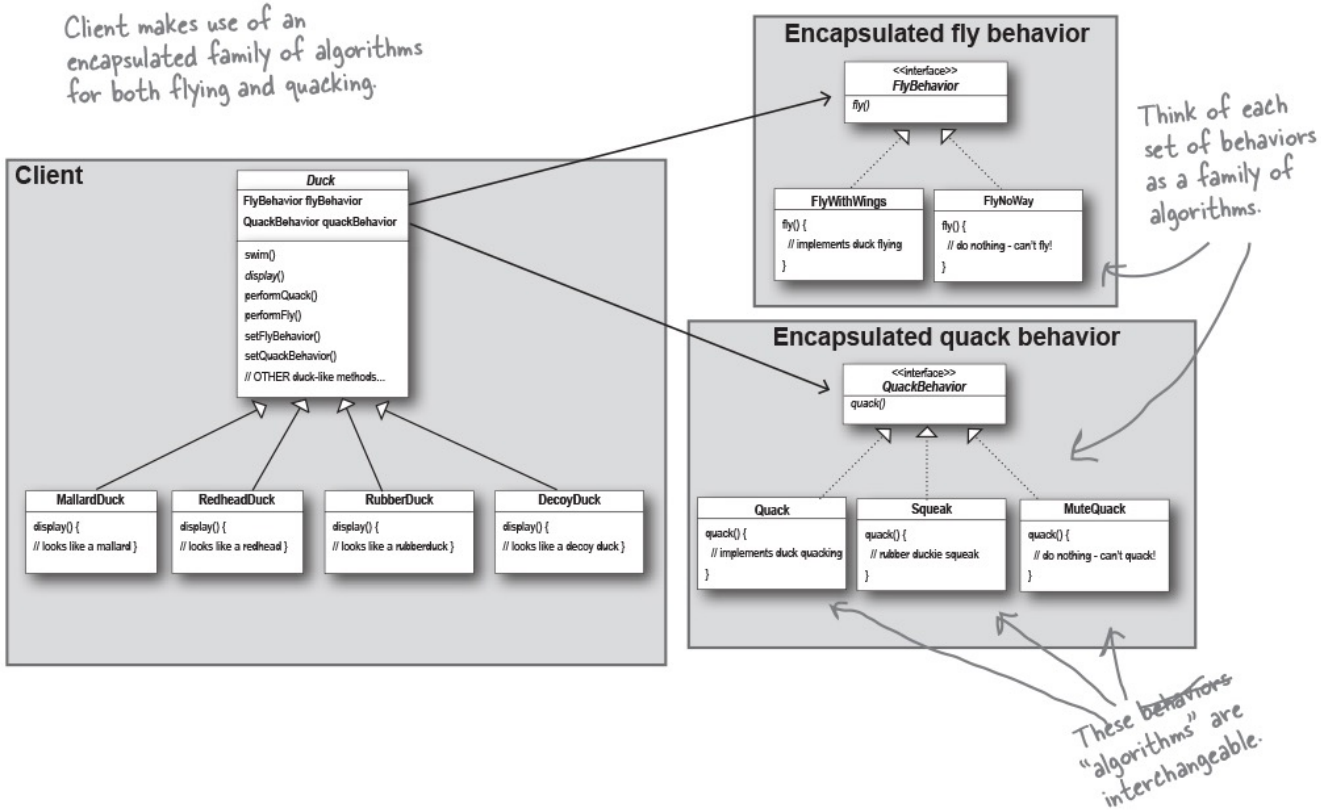
We can call these methods anytime we want to change the behavior of a duck on the fly.

Editor note: gratuitous pun – fix



Setting the behavior dynamically

Final Solution



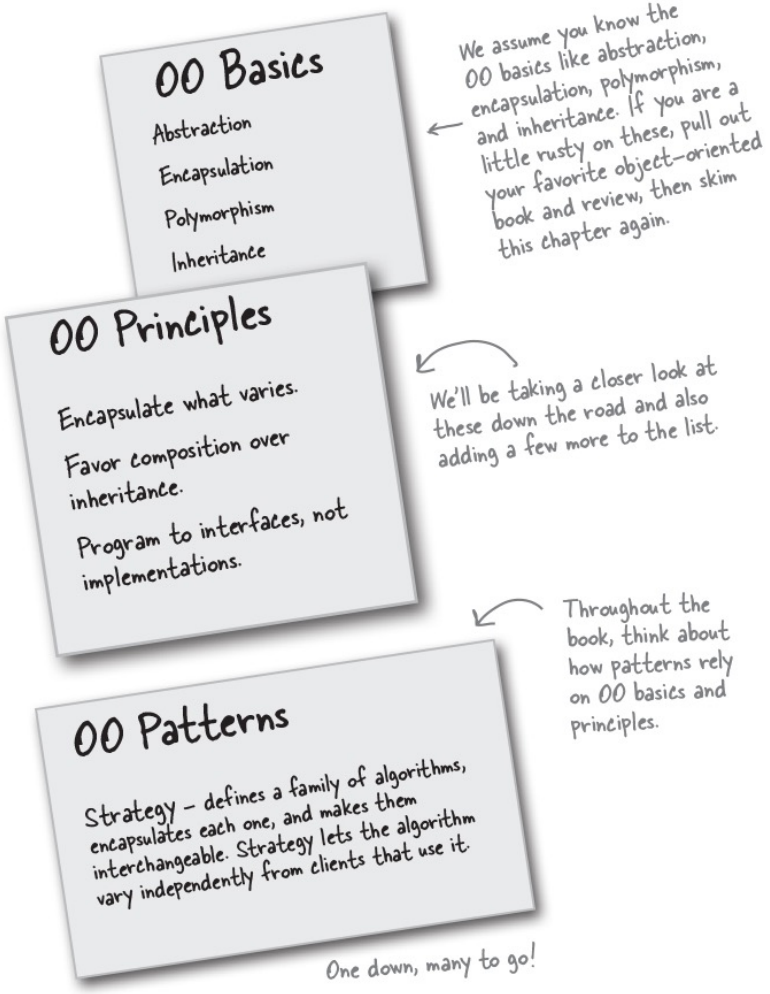
Design Principle

Favor composition over inheritance

The Strategy Pattern

Defines a family of algorithms, encapsulates each one, and makes them interchangeable. Strategy lets the algorithm vary independently from clients that use it.

Summary


OO Basics
Abstraction
Encapsulation
Polymorphism
Inheritance
We assume you know the OO basics like abstraction, encapsulation, polymorphism, and inheritance. If you are a little rusty on these, pull out your favorite object-oriented book and review, then skim this chapter again.
OO Principles
Encapsulate what varies.
Favor composition over inheritance.
Program to interfaces, not implementations.
We'll be taking a closer look at these down the road and also adding a few more to the list.
OO Patterns
Strategy – defines a family of algorithms, encapsulates each one, and makes them interchangeable. Strategy lets the algorithm vary independently from clients that use it.
Throughout the book, think about how patterns rely on OO basics and principles.
One down, many to go!