Design Patterns

Recap

- The one constant in software development is **change**

- Think about your code/product/app to make it flexible, maintainable, and can cope with changes (extendable)

- Patterns rely on OO concepts & principles

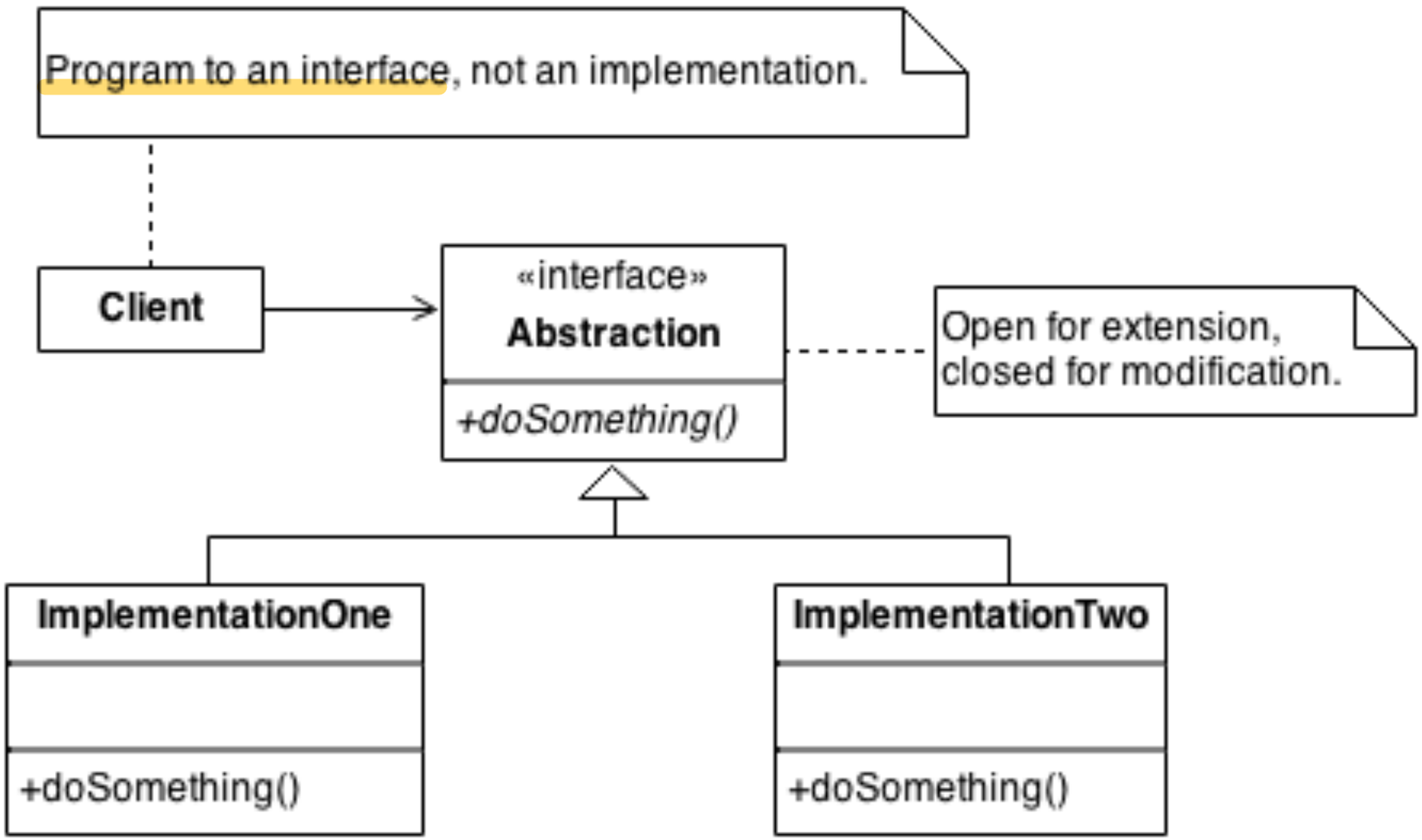
Principles

- encapsulated aspect of the code that can/will vary
- favored Has-A relationship over Is-A relationship
  - favor composition over inheritance
- creating systems using composition gives you a lot more flexibility.
- Not only does it let you encapsulate a family of algorithms into their own set of classes, but it also lets you change behavior at runtime

Strategy Pattern

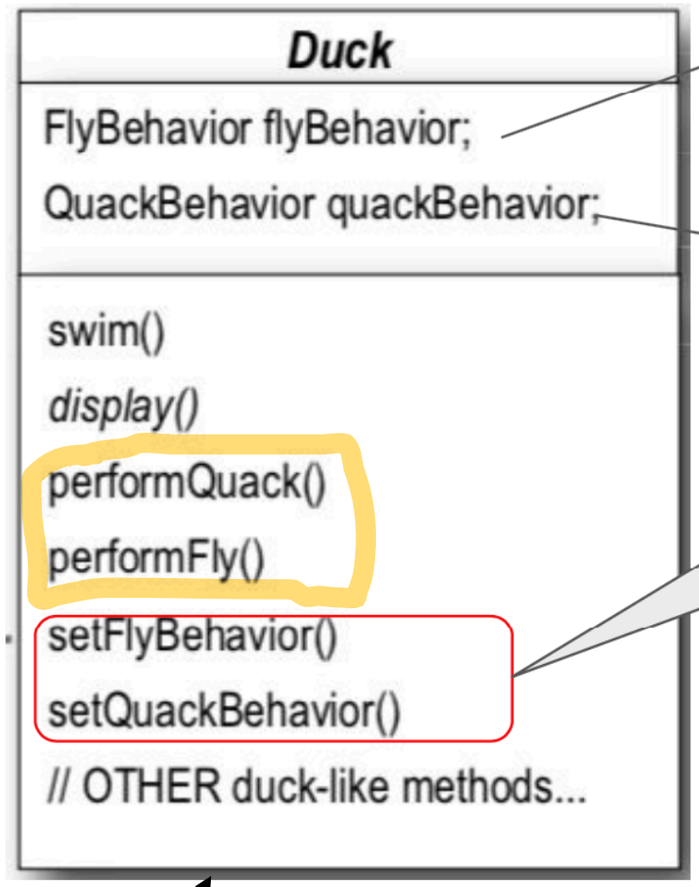
Defines a family of algorithms, encapsulate each one, and makes them interchangeable. Lets the algorithm vary independently from the client using it.

Class Diagram

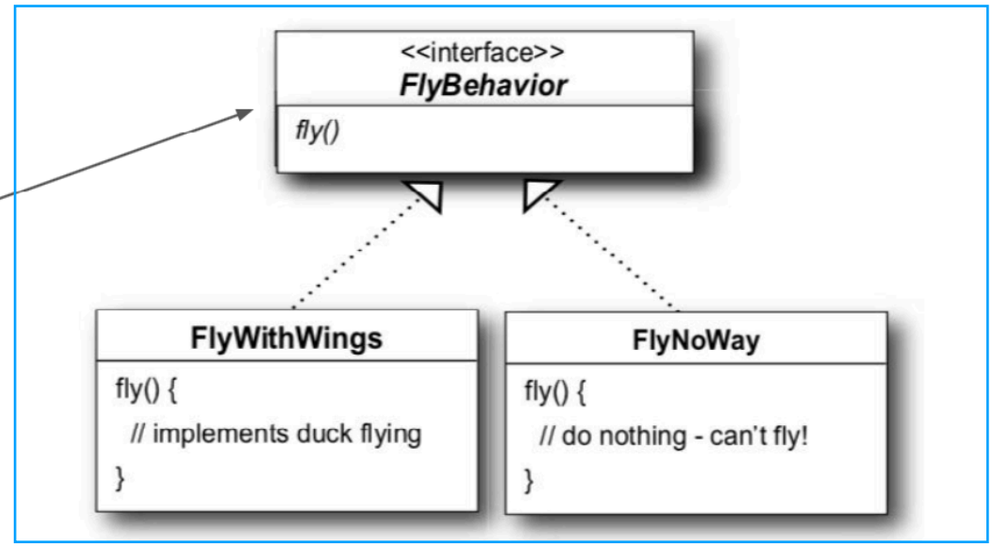


Does that look familiar to you?

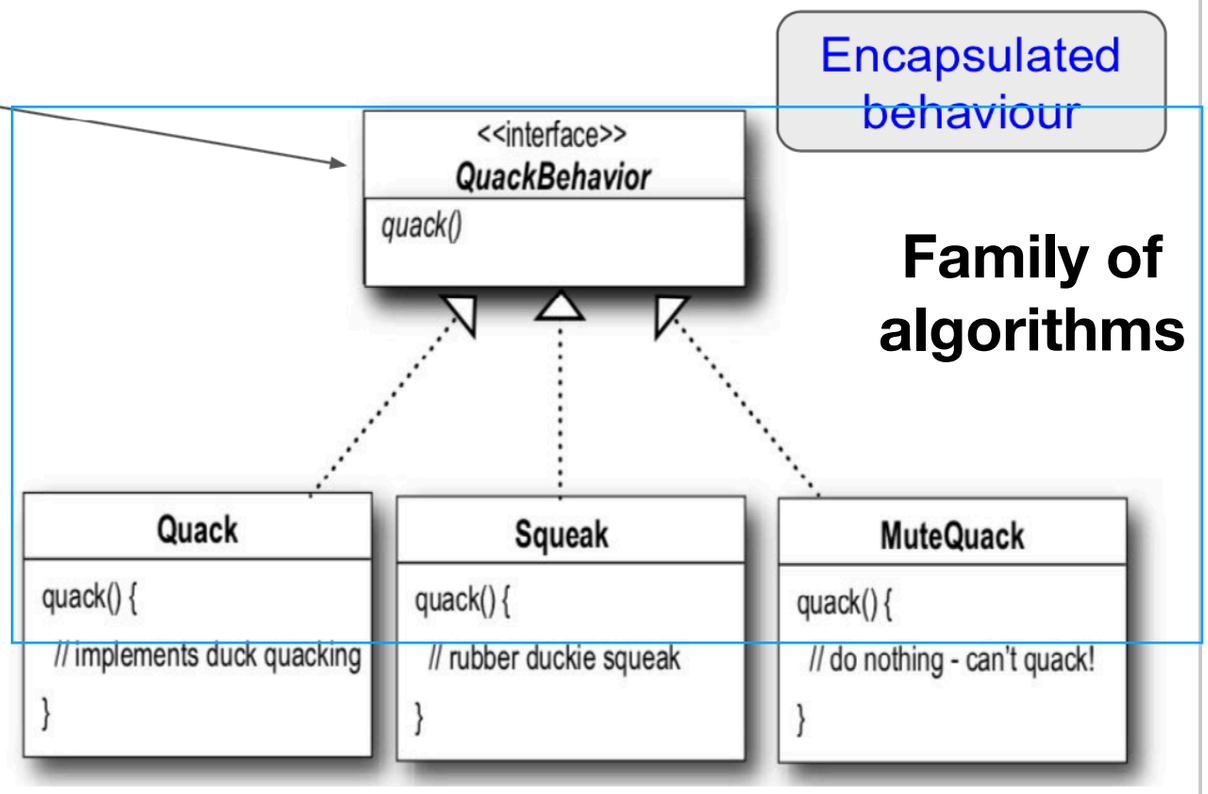
Abstract Class makes use of encapsulated family of algorithms



Behaviour can be set at runtime



Encapsulated behaviour



Family of algorithms

MallardDuck

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

Concrete class

Test the Code

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

Call the inherited method

```
public class MiniDuckSimulator {
public static void main(String[] args) {
Duck mallard = new MallardDuck();
mallard.performQuack();
mallard.performFly();   }
 }
```

Delegate to object concrete quack behavior

```
public class Quack implements QuackBehavior
{   public void quack() {
System.out.println(“Quack”);
 }
 }
```

We can still do one more modification to improve the code

- Set the behavior dynamically at runtime
- What will happen if we want to modify the behavior at runtime?

- Do it by yourself (see next slide)

Tutorial

- Download the code from lms
- Create new java projects, use the classes you downloaded from lms
- Create a simulator class which create an instance from the Mallard Duck class, show its behavior
- Add new Duck type call it ModelDuck; this type can't fly and can quack
- Modify the code to show how at runtime we can change the fly behavior of the ModelDuck which can fly as the rocket
  - You need to update the Duck class
  - Add new fly behavior: fly as rocket
  - In the simulator show how to do the changes