Improving Interfaces with Wrappers

Andrejs Doronins
TEST AUTOMATION ENGINEER

1 Creational

2 Behavioral

3 Structural

- Adapter
- Decorator
- Facade

What's the difference?

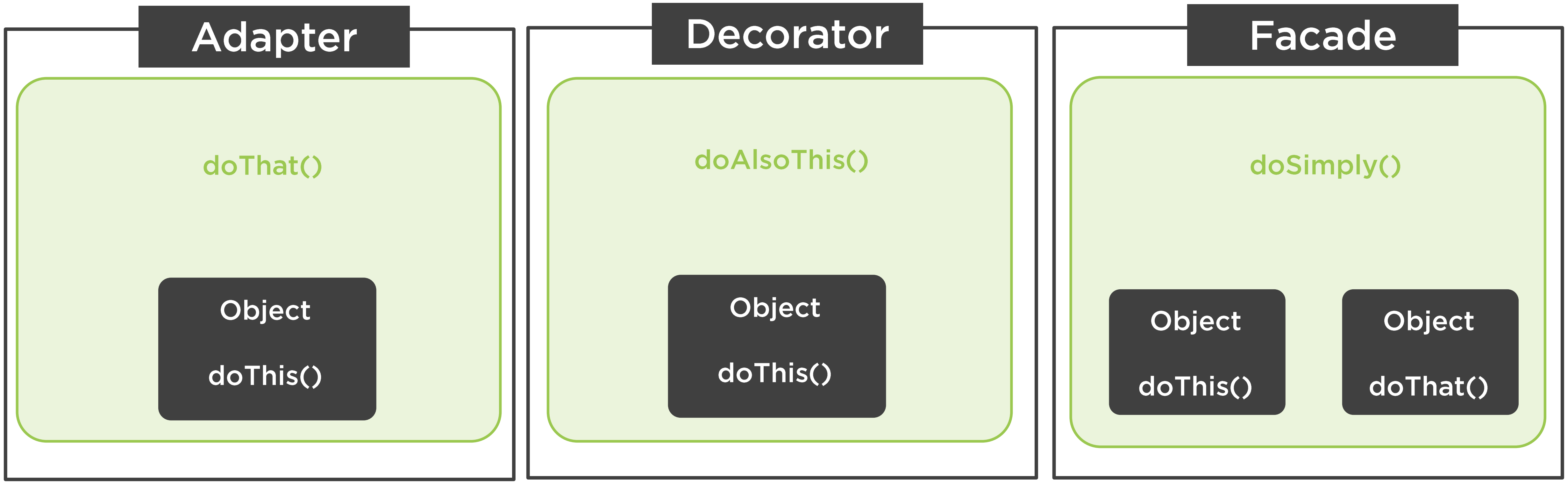
Overview

Understand the gist of each wrapper pattern

Apply the Adapter

Explore Decorator in-depth

Apply the Facade



Wraps and changes

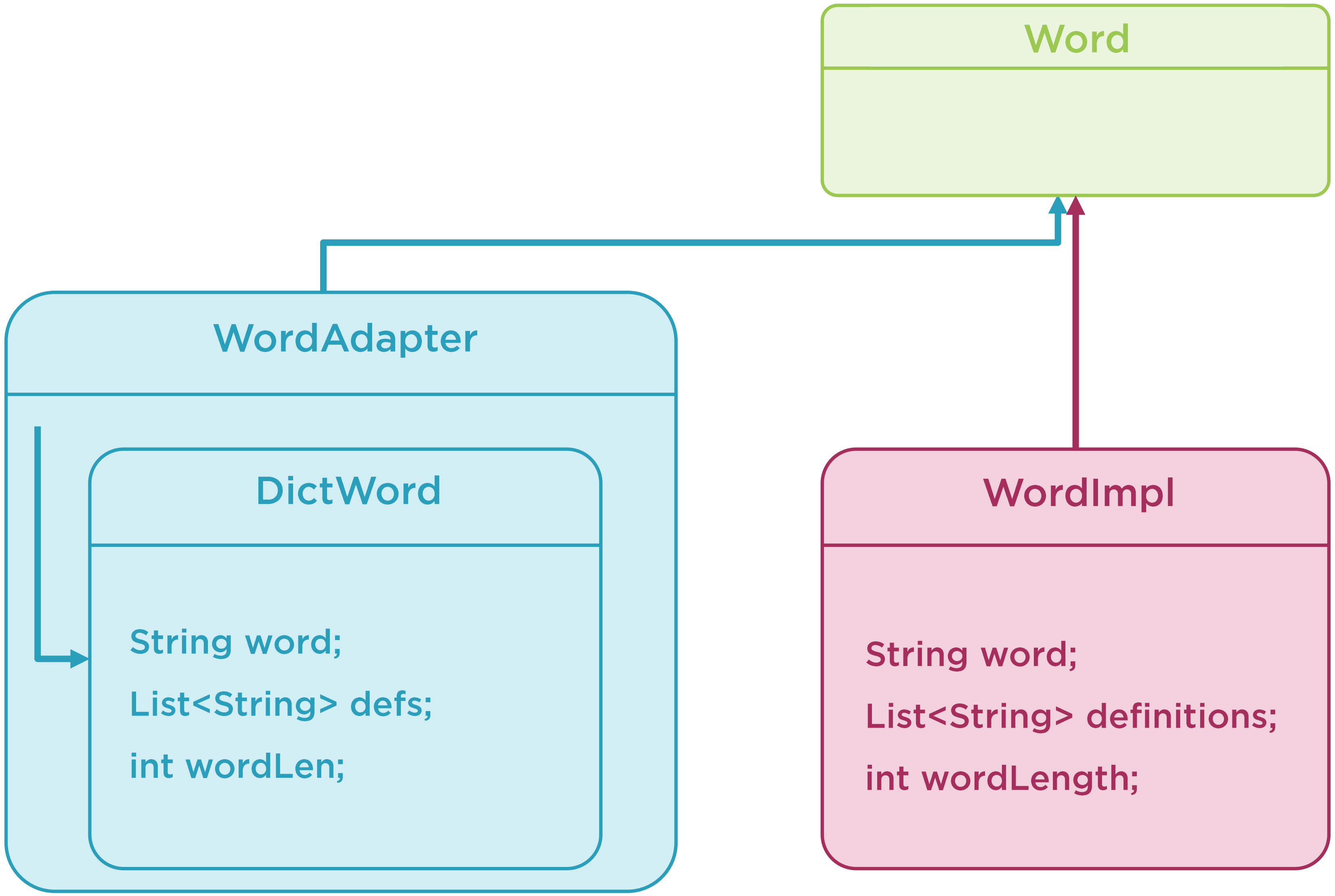
Wraps and adds

Wraps, unites and simplifies

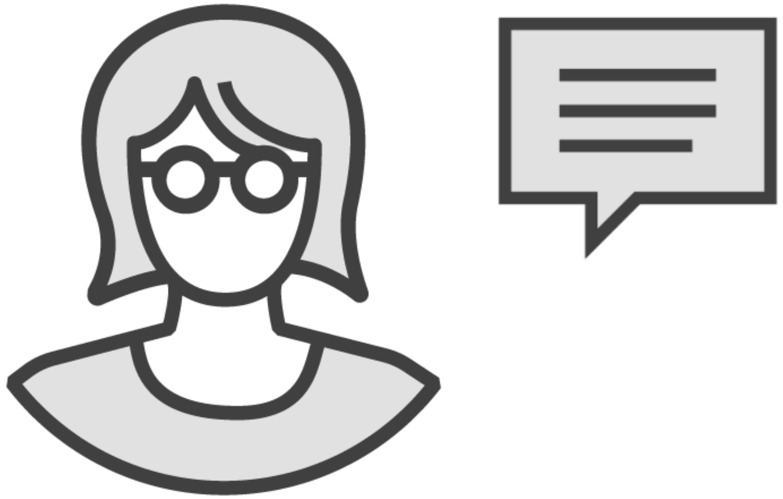
Adapter

Wraps an existing class and acts as a connector between two incompatible interfaces.

Use this!


Word
WordAdapter
DictWord
String word;
List<String> defs;
int wordLen;
WordImpl
String word;
List<String> definitions;
int wordLength;

Can we just change the code to make it conform?



Yes
change the code, you don't need an Adapter

No
consider implementing an Adapter

Decorator

Attaches additional responsibilities to an object. In other words, it provides an enhanced interface to the original object.

Decorator Example

```
new BufferedReader(new FileReader(new File(“f.txt”)));

new BufferedReader(new File(“f.txt”)); // won’t compile
```

SaveFileApp.java

Requirements:

Save a file

Preformat it

Compress it

Encrypt it

Any mix of the above

```java
String data = /* ... */;
DataDecorator writer =
    new CompressionDecorator(
        new EncryptionDecorator(
            new FileData("file.txt")

writer.write(data);
```

Decorator Pattern Benefits and Drawbacks

Pros:

- Flexible and transparent
- Adheres to SRP (one functionality per class)

Cons:

- Looks ugly and complicated
- Sometimes inflexible with the order of decorating

Decorator replaced with functional composition

SaveFileApp.java

```java
Function<T,R> encrypt = /* ... */;
Function<T,R> compress = /* ... */;

encrypt.andThen(compress)
       .apply(new File("f.txt"));
```

ClunkyClass

method1(a,b,c,d)

method2(a,b,c)

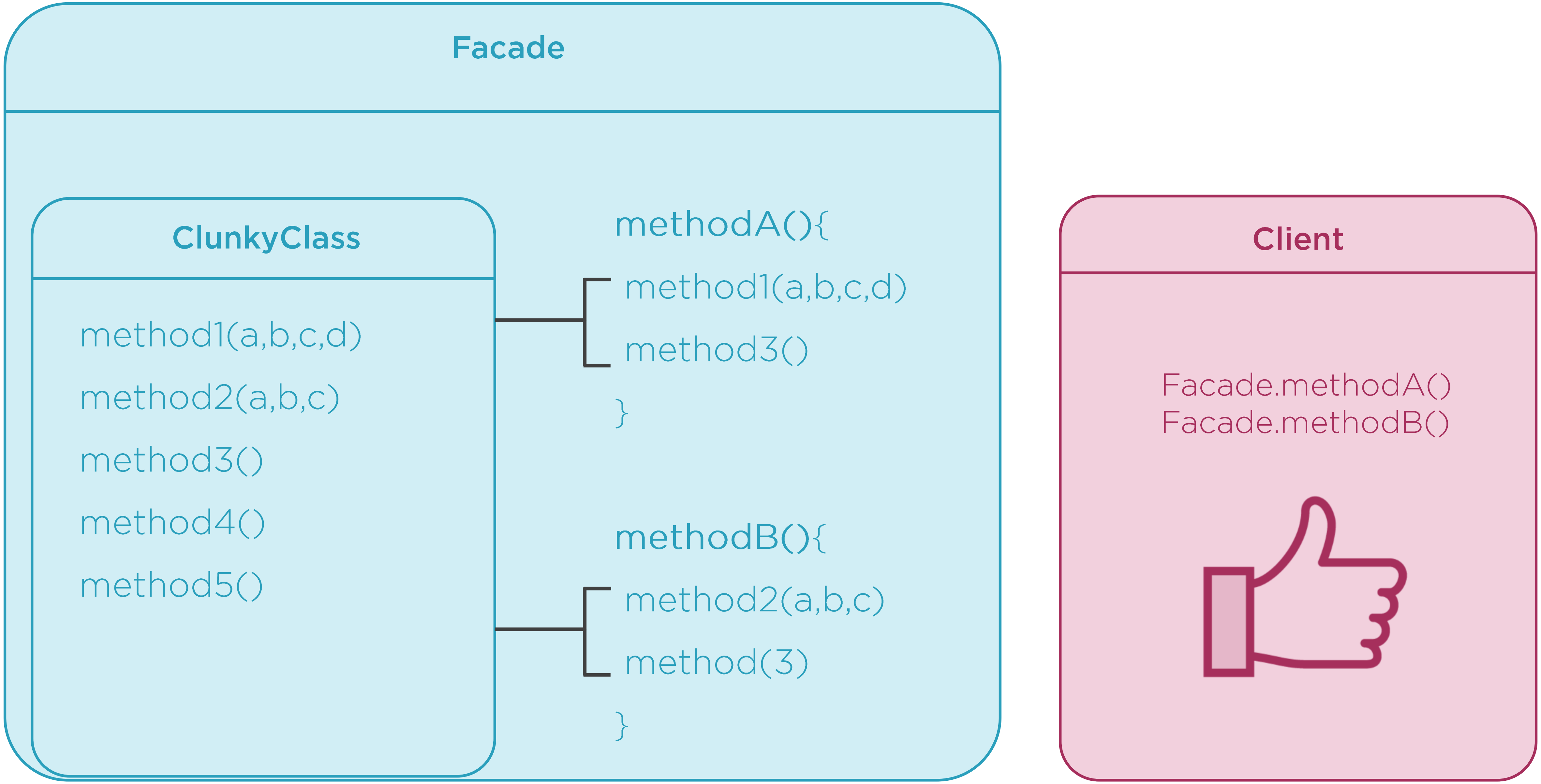
method3()

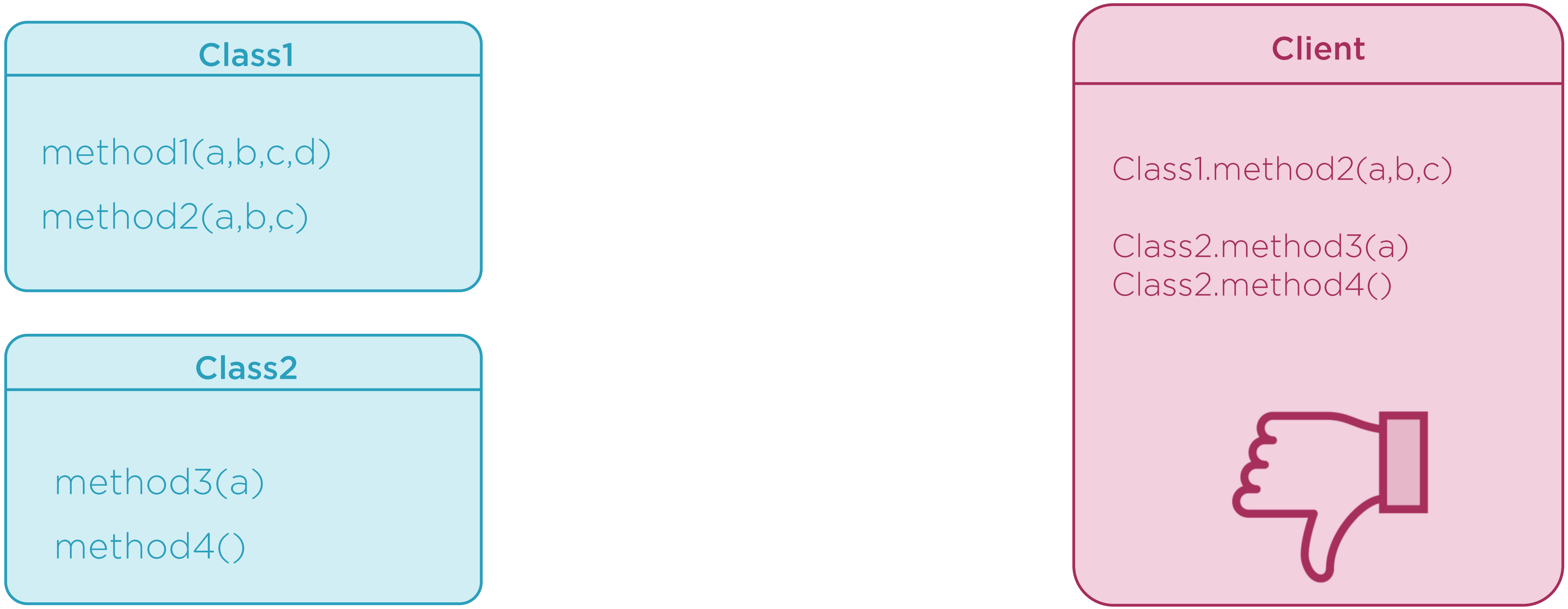
method4()

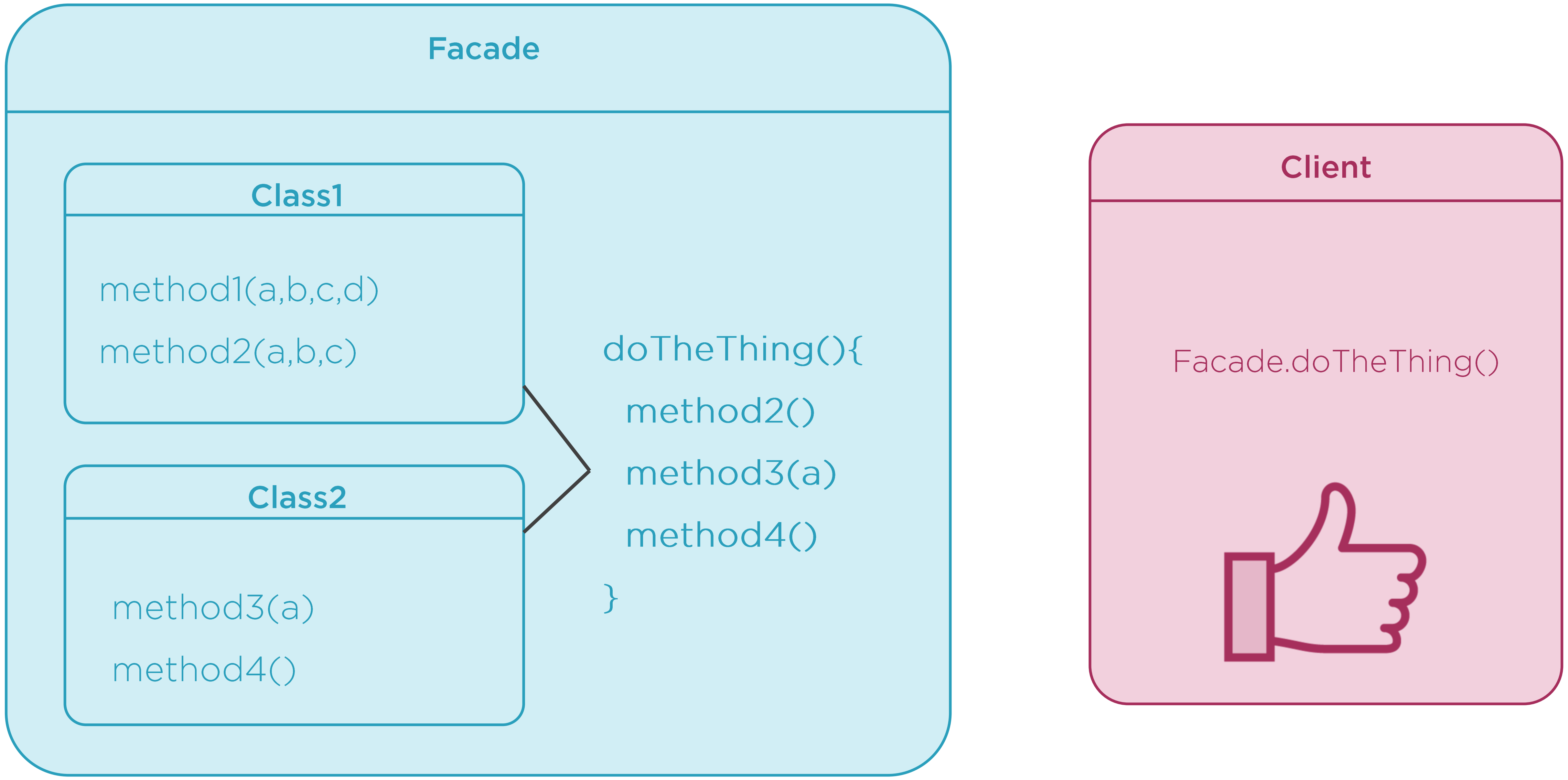
method5()

Client

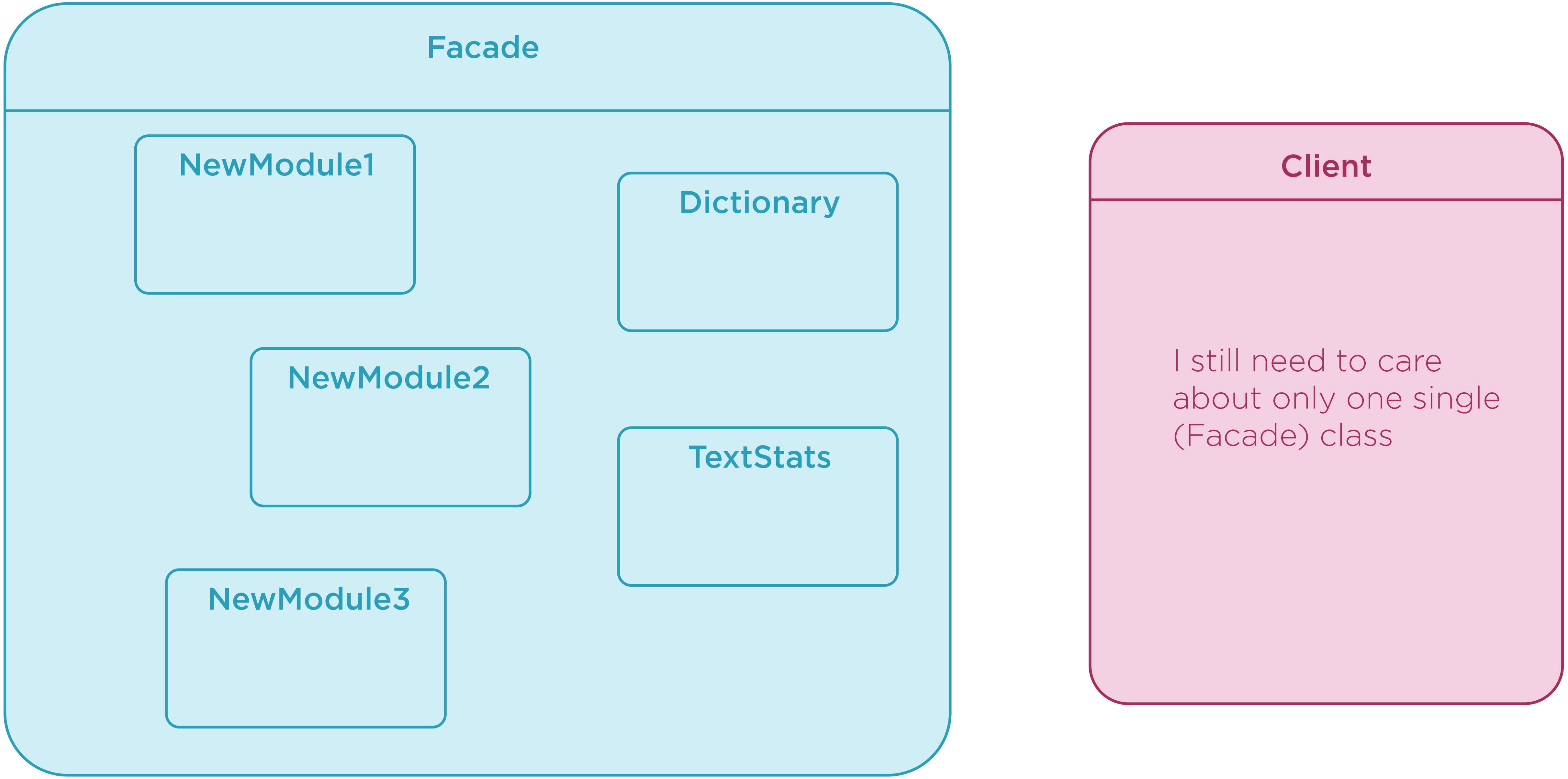
ClunkyClass.method3()
ClunkyClass.method5()
ClunkyClass.method2(a,b,c)


Facade
ClunkyClass
method1(a,b,c,d)
method2(a,b,c)
method3()
method4()
method5()
methodA(){
method1(a,b,c,d)
method3()
}
methodB(){
method2(a,b,c)
method(3)
}
Client
Facade.methodA()
Facade.methodB()


Class1
method1(a,b,c,d)
method2(a,b,c)
Class2
method3(a)
method4()
Client
Class1.method2(a,b,c)
Class2.method3(a)
Class2.method4()



```
doTheThing(){
  method2()
  method3(a)
  method4()
}
```


Facade
NewModule1
Dictionary
NewModule2
TextStats
NewModule3
Client
I still need to care about only one single (Facade) class

Summary

Structural patterns allow for cleaner code structure

Adapter wraps and changes an interface of a class

Decorator wraps and adds functionality

Facade wraps and encapsulates one or more classes or modules

OO design pattern may be sometimes replaced with a different programming style

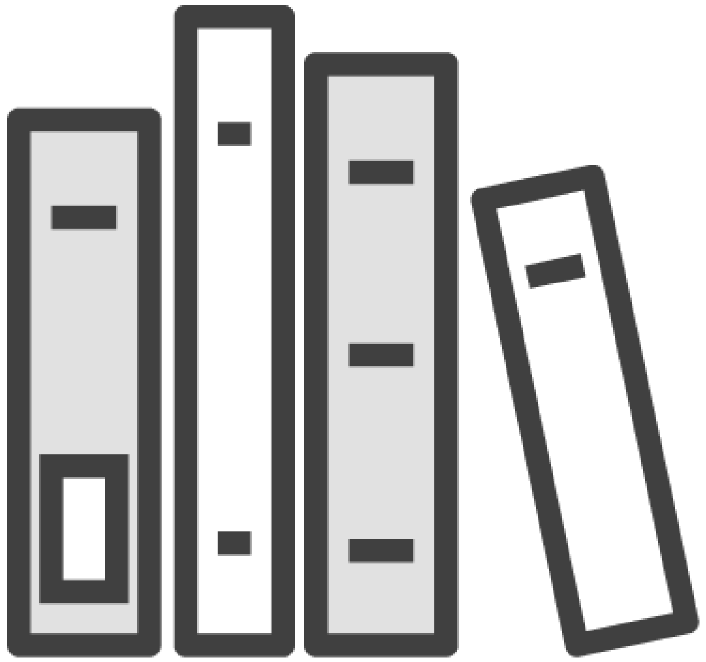
Course Summary

Developed a dictionary and text application

Gradually refactored as complexity increased:

- Applied various creational techniques (factories)
- Applied non-pattern solutions
- Explored functional alternatives

Further Material



Path: Java Coding Practices

Path: Design Patterns in Java

Course: Making Your Java Code More Object-oriented

Course: Implementing Design Patterns with Java 8 Lambdas

Rating

Thank you!

(Happy coding)