Het Facade Pattern
Het Principe van Kennisabstractie

Objectgeoriënteerde analyse en ontwerp

katholieke hogeschool
associatie KU Leuven
vives

Het Facade Pattern
Doel

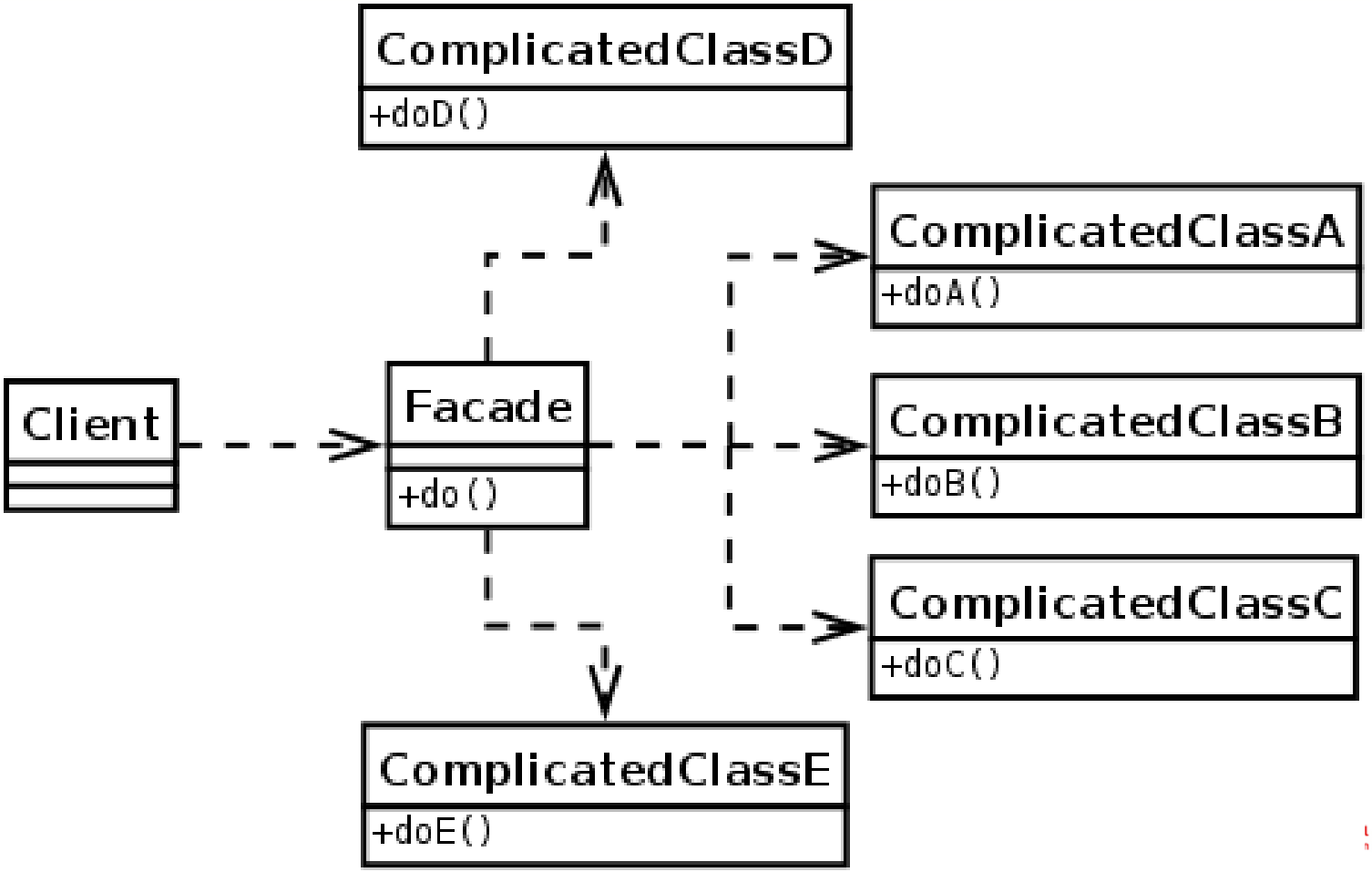
Het Facade Pattern zorgt voor **een vereenvoudigde interface** naar een verzameling interfaces in een subsysteem.

De vereenvoudigde interface

De subsystemen

katholieke hogeschool associatie KU Leuven vives

Het Facade Pattern
Structuur



Het Principe van Kennisabstractie

Praat alleen met je directe vrienden.

Hoe minder je weet, hoe beter.

(Andere naam: **Demeterprincipe**)

Een tegenvoorbeeld
een weerstation met daarin een thermometer

```
public float getTemperature() {
    return station.getThermometer().getTemperature();
}
```

Met hoeveel klassen is deze code gekoppeld?

Een tegenvoorbeeld

```
public float getTemperature() {
    return station.getThermometer().getTemperature();
}
```

Er is koppeling met 2 klassen: Station en Thermometer.

Het demeterprincipe zegt dat dat 1 koppeling te veel is.

katholieke hogeschool associatie KU Leuven vives

Principe van Kennisabstractie
Richtlijnen

Een methode Foo() in een object van klasse Bar mag enkel volgende andere methoden gebruiken:

- andere methodes van Bar
- methodes van componenten van Bar
- methodes van objecten die binnen Foo() geïnstantieerd zijn
- methodes van objecten uit de parameterlijst van Foo()

Dus NIET

methodes van objecten die de returnwaarde zijn van andere methoden

Richtlijnen
Voorbeeld

```
public class Car {
    Engine engine;
    public Car() {  // initialise }
    public void start(Key key) {
        Doors doors = new Doors();
        boolean authorized = key.turns();
        if (authorized) {
            engine.start();
            updateDashboard();
            doors.lock();
        }
    }
    public void updateDashboard() { // update dash }
}
```

uit parameterlijst

component van Car

andere methode van Car

geïnstantieerd binnen methode start

Principe van Kennisabstractie voor- en nadelen

Voordeel

- minder afhankelijkheden tussen objecten
  -> onderhoud wordt gemakkelijker

Nadelen

- meer wrapperklassen
  -> complexiteit neemt toe
  -> runtime performance vermindert
- erg streng principe dat tegen gewoontes ingaat
- streams in Java 8 doen net het tegenovergestelde