Tema 3: Disseny

Anna Puig

Enginyeria Informàtica

Facultat de Matemàtiques i Informàtica,

Universitat de Barcelona

Curs 2020/2021

UNIVERSITAT DE BARCELONA

Temari

1. Introducció al procés de desenvolupament del software
2. Anàlisi de requisits i especificació
3. Disseny
   - 3.1 Introducció
   - 3.2 Patrons arquitectònics
   - 3.3 Criteris de Disseny: G.R.A.S.P.
   - 3.4 Principis de Disseny: S.O.L.I.D.
   - 3.5 **Patrons de disseny**
4. Del disseny a la implementació
5. Ús de frameworks de testing

-

Classificació (GoF)

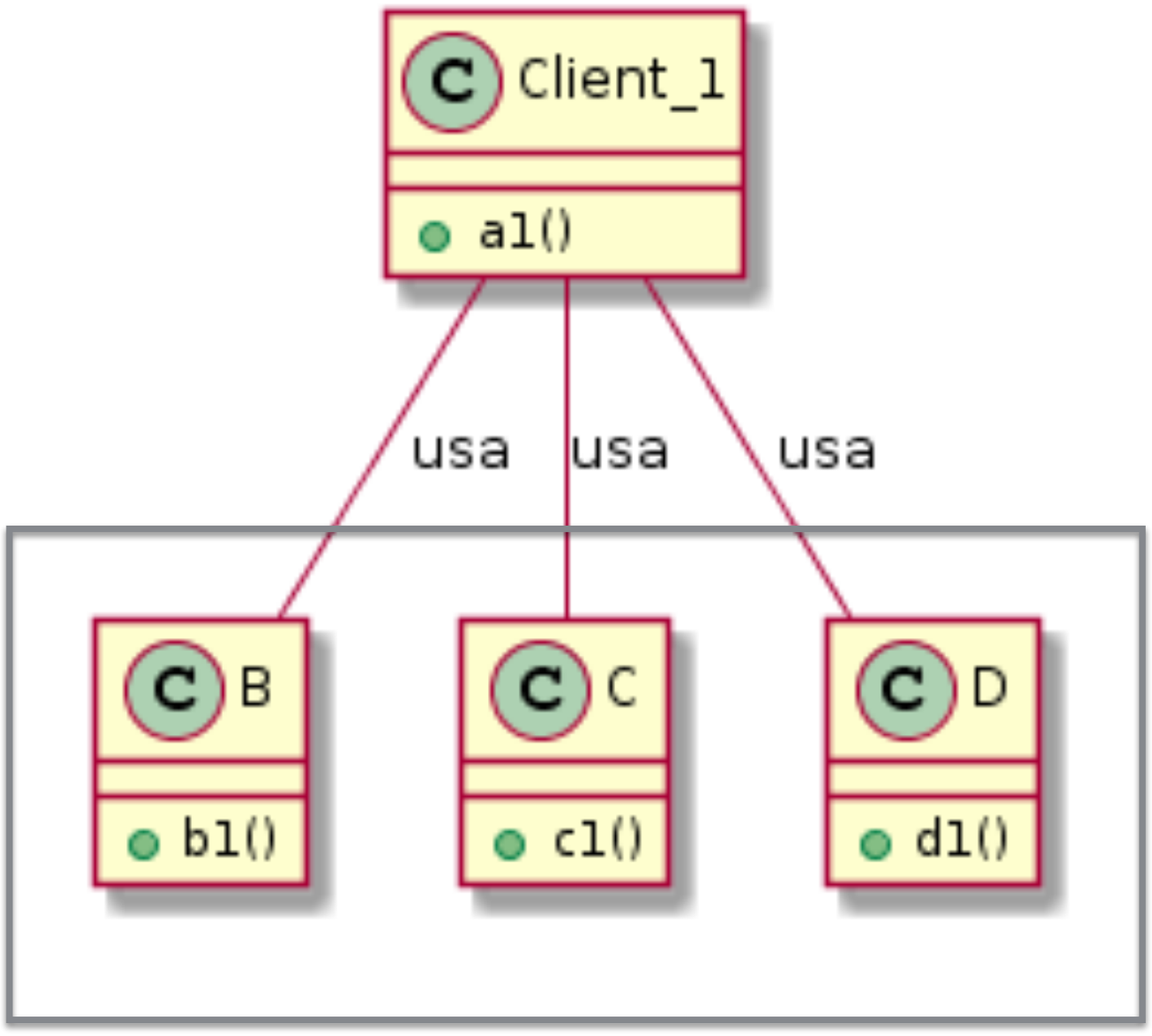
- Creació: Tracten la inicialització i configuració de classes i objectes
  - **Classes**: delega la creació dels objectes a les subclasses
  - **Objectes**: delega la creació d'objectes a altres objectes
- Estructura: Tracten la composició de classes i/o objectes
  - **Classes**: usen herència per composar classes
  - **Objectes**: descriuen maneres d'assemblar objectes
- Comportament: Descriuen situacions de control de flux i caracteritzen el mode en que interactuen i reparteixen responsabilitats les diferents classes o objectes
  - **Classes**: usen herència per descriure els algorismes i fluxos de control
  - **Objectes**: descriuen com un grup d'objectes cooperen per fer una tasca que un objecte la no podria fer sol

3.5. Patrons de disseny

Propòsit → **Àmbit ↓**	**CREACIÓ**	**ESTRUCTURA**	**COMPORTAMENT**
CLASSE	• Factory method	• class Adapter	• Interpreter • Template method
OBJECTE	• Abstract Factory • Builder • Prototype • Singleton • Object pool	• Object Adapter • Bridge • Composite • Decorator • **Facade** • Flyweight • Proxy	• Chain of Responsability • Command • Iterator • Mediator • Memento • Observer • State • Strategy • Visitor

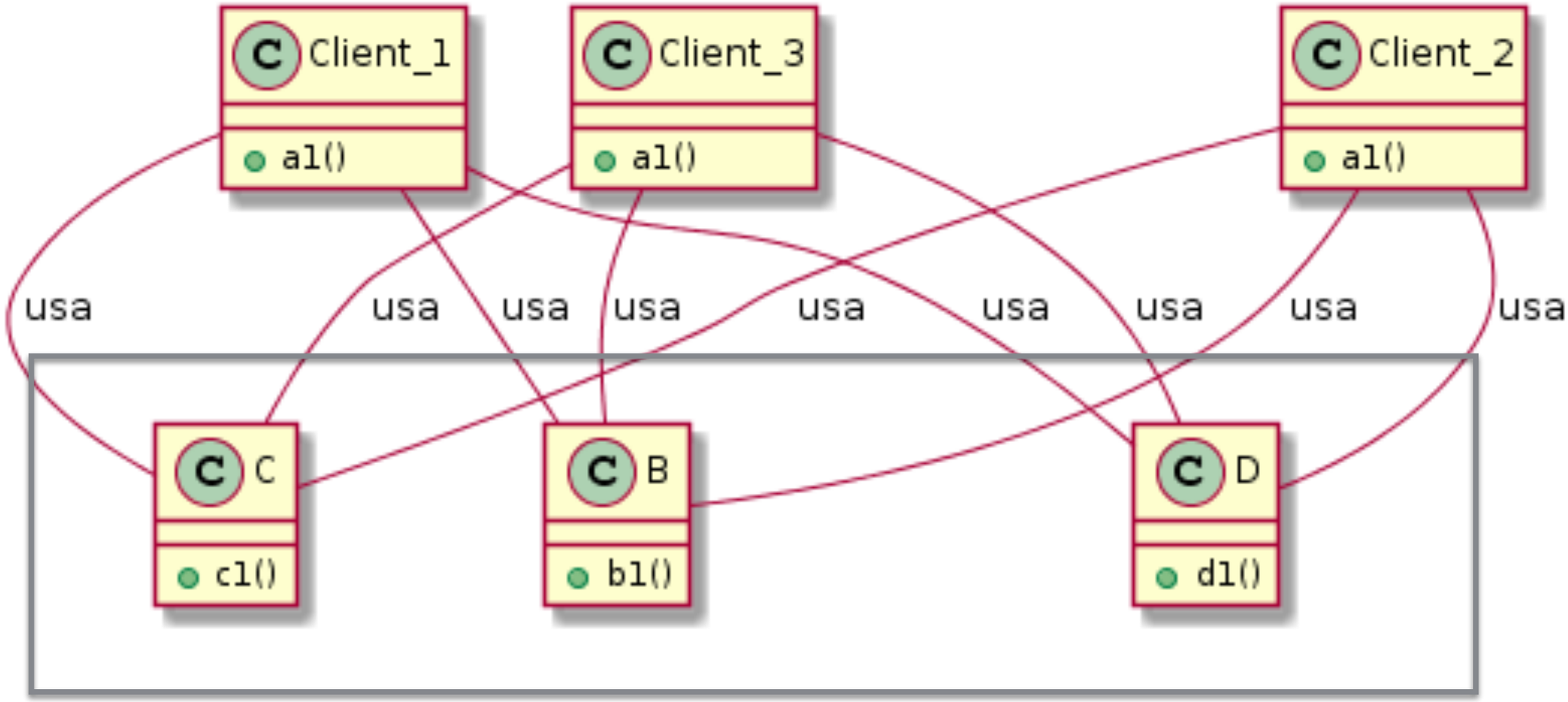
3.5.1. Patrons de disseny: Patró Façana (Facade)

Problema:



- La classe Client_1 ha de conèixer quina és la classe que exactament li proporciona el servei:
  - b1() es de B, c1() es de C, d1() es de D, etc.
- Alt acoblament !!

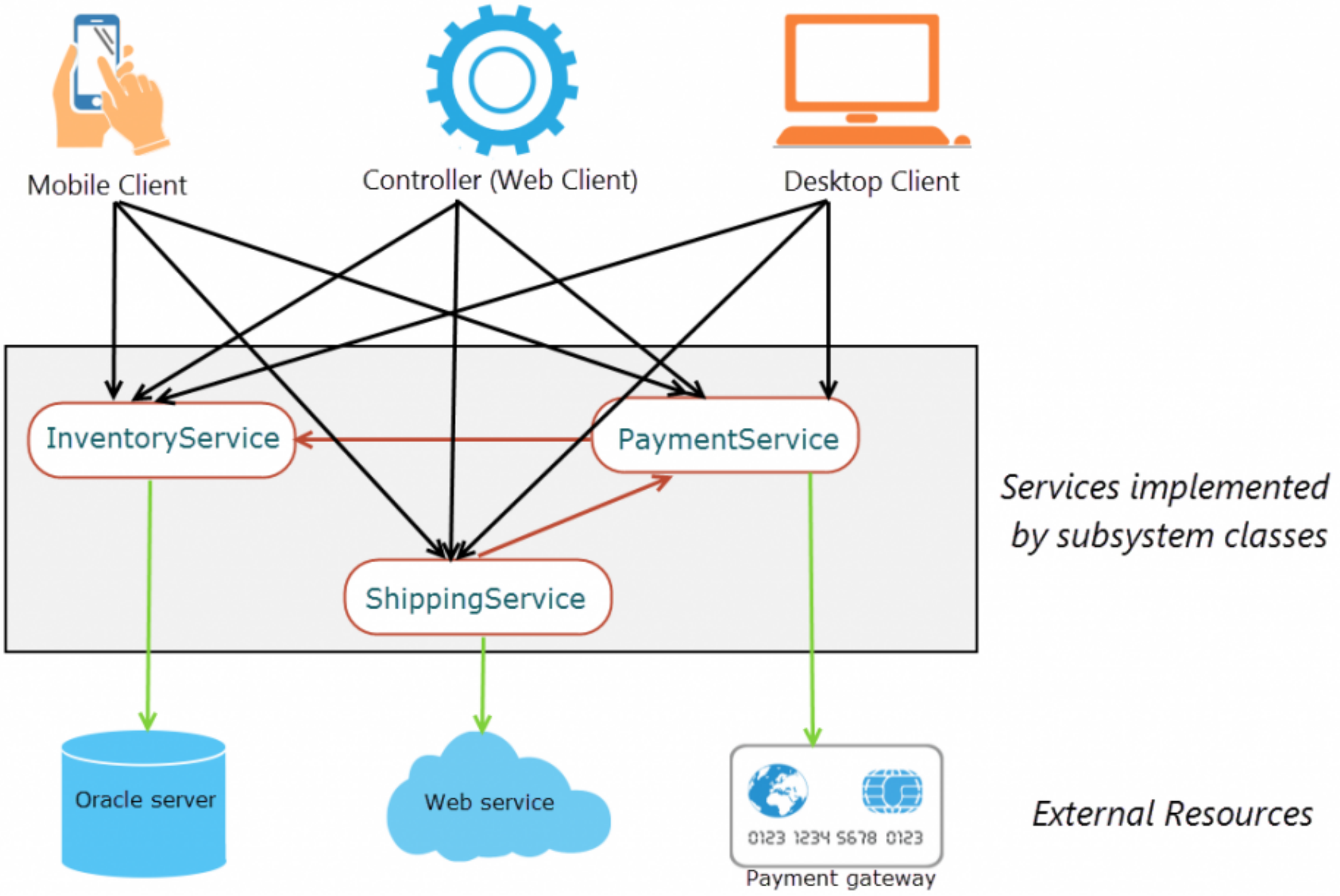
3.5.1. Patrons de disseny: Patró Façana (Facade)



PROBLEMA

A més a més, poden haver-hi moltes classes client

3.5.1. Patrons de disseny: Patró Façana (Facade)



Quin principi SOLID vulnera?

3.5.1. Patrons de disseny: Patró Façana (Facade)

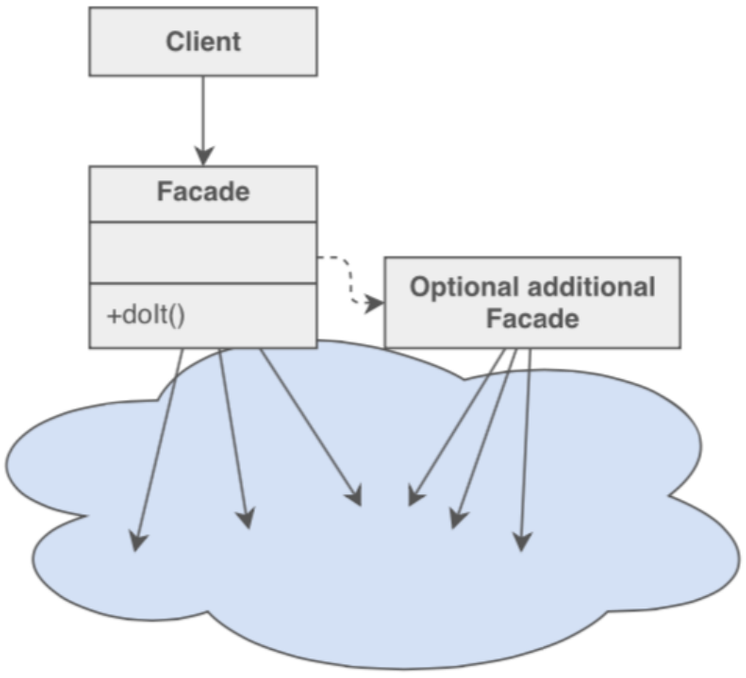
Nom del patró: **Façana** (*Facade*)

Context:

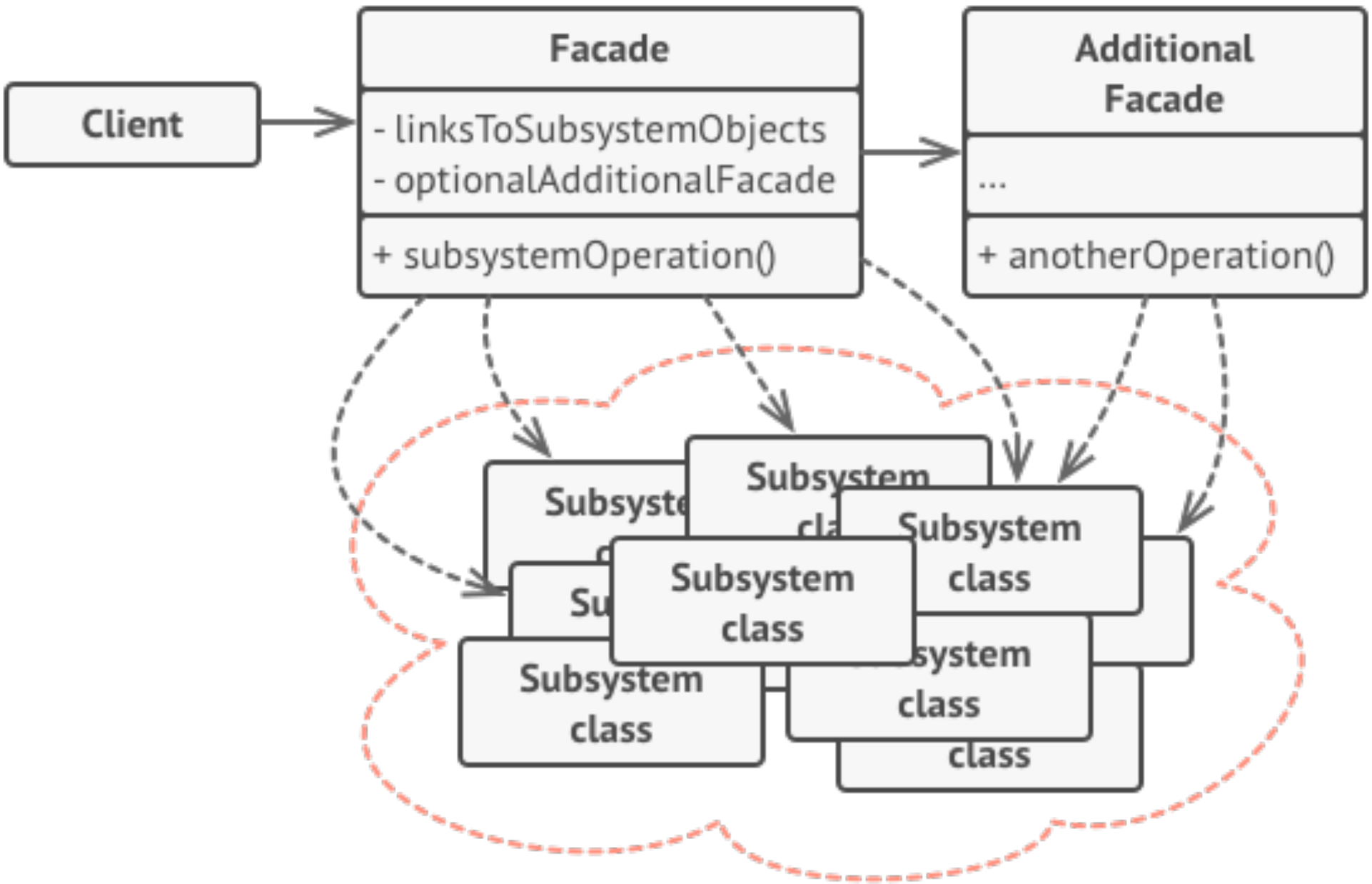
Subsistema amb moltes utilitats diferents

Solució:

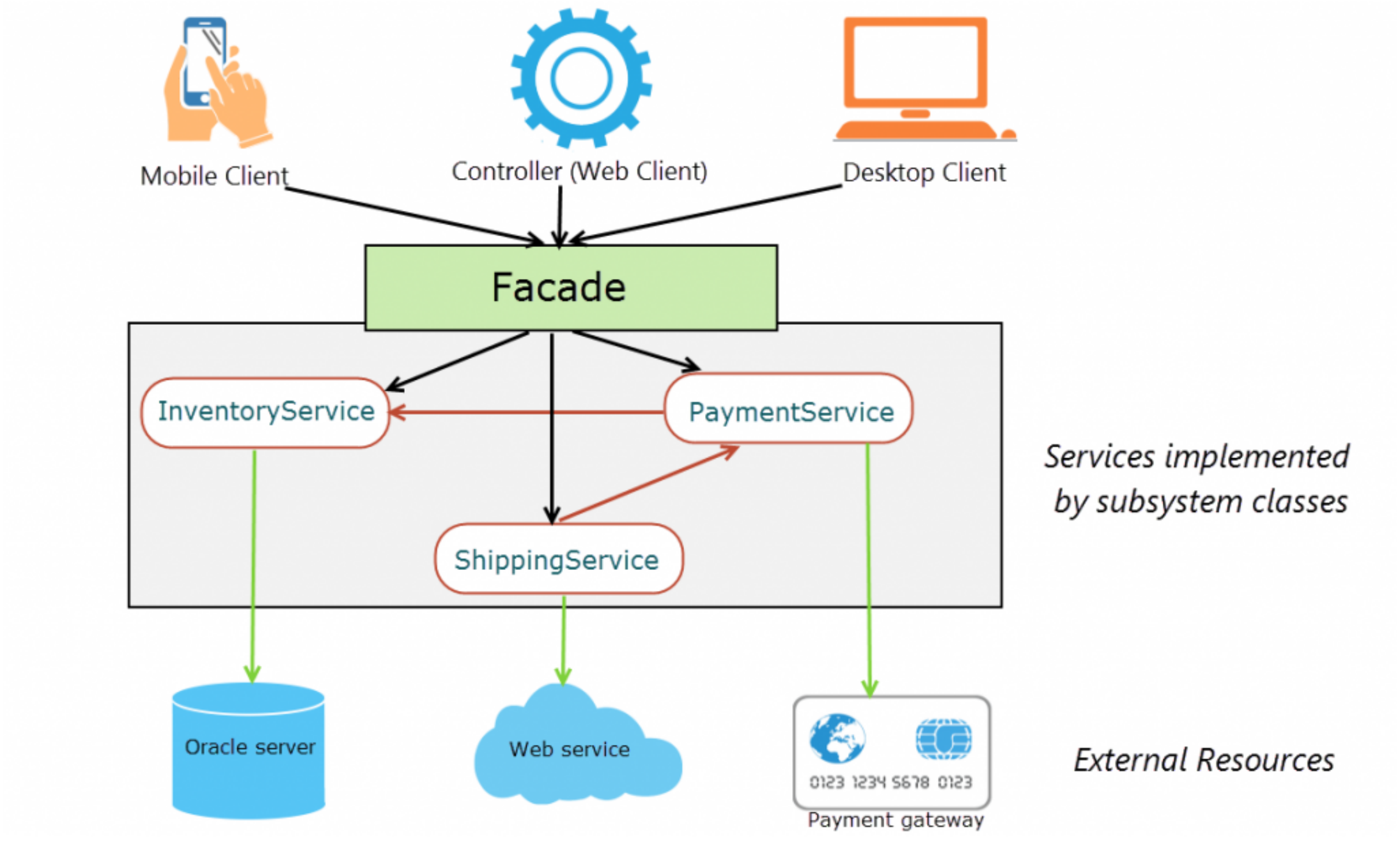
Proporciona una interfície unificada a un conjunt d'utilitats d'un subsistema complex. Redueix la corba d'aprenentatge necessària per aprofitar amb èxit el subsistema.



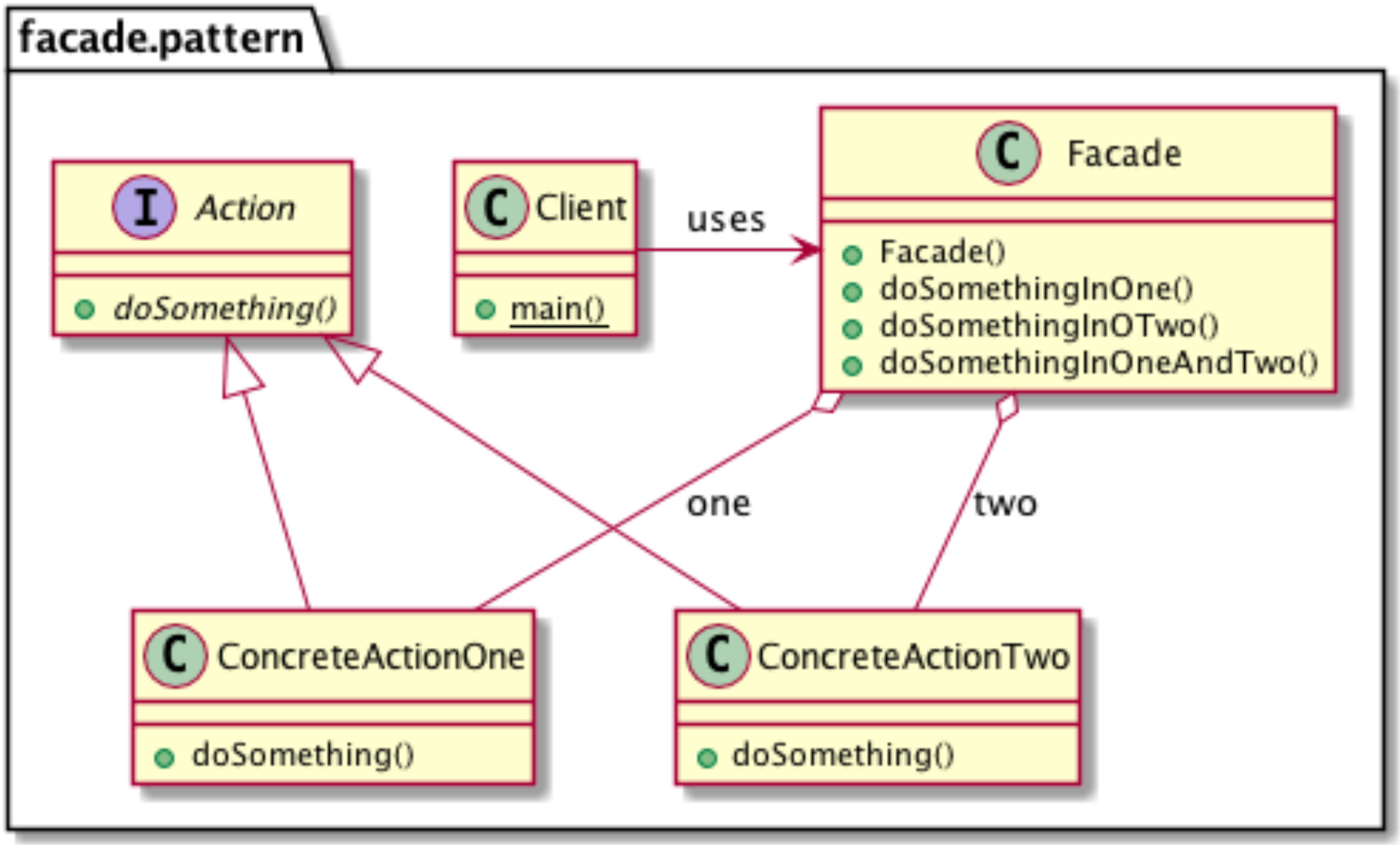
3.5.1. Patrons de disseny: Patró Façana (Facade)



3.5.1. Patrons de disseny: Patró Façana (Facade)



3.5.1. Patrons de disseny: Patró Façana (Facade)



3.5.1. Patrons de disseny: Patró Façana (Facade)

```
public class Facade {

    private ConcreteActionOne one;
    private ConcreteActionTwo two;

    public Facade() {
        System.out.println("This is the FACADE pattern...");
        this.one = new ConcreteActionOne();
        this.two = new ConcreteActionTwo();
    }

    public void doSomethingInOne() {
        System.out.println("Calling doSomething in action ONE:");
        one.doSomething();
    }

    public void doSomethingInOTwo() {
        System.out.println("Calling doSomething in action TWO:");
        two.doSomething();
    }

    public void doSomethingInOneAndTwo() {
        System.out.println("Calling doSomething in action ONE and TWO:");
        one.doSomething();
        two.doSomething();
    }
```

3.5.1. Patrons de disseny: Patró Façana (Facade)

Conseqüència

- Simplifica l'accés a un conjunt de classes proporcionant una única classe que tots utilitzen per comunicar-se amb el conjunt de classes

Pros:

- Les aplicacions client no necessiten conèixer les classes que hi ha darrera la classe FACADE
- Es poden canviar les classes "ocultes" sense necessitat de canviar els clients. Només s'ha de fer els canvis necessaris a FACADE
- Es minimitzen les comunicacions i dependències entre subsistemes

Cons:

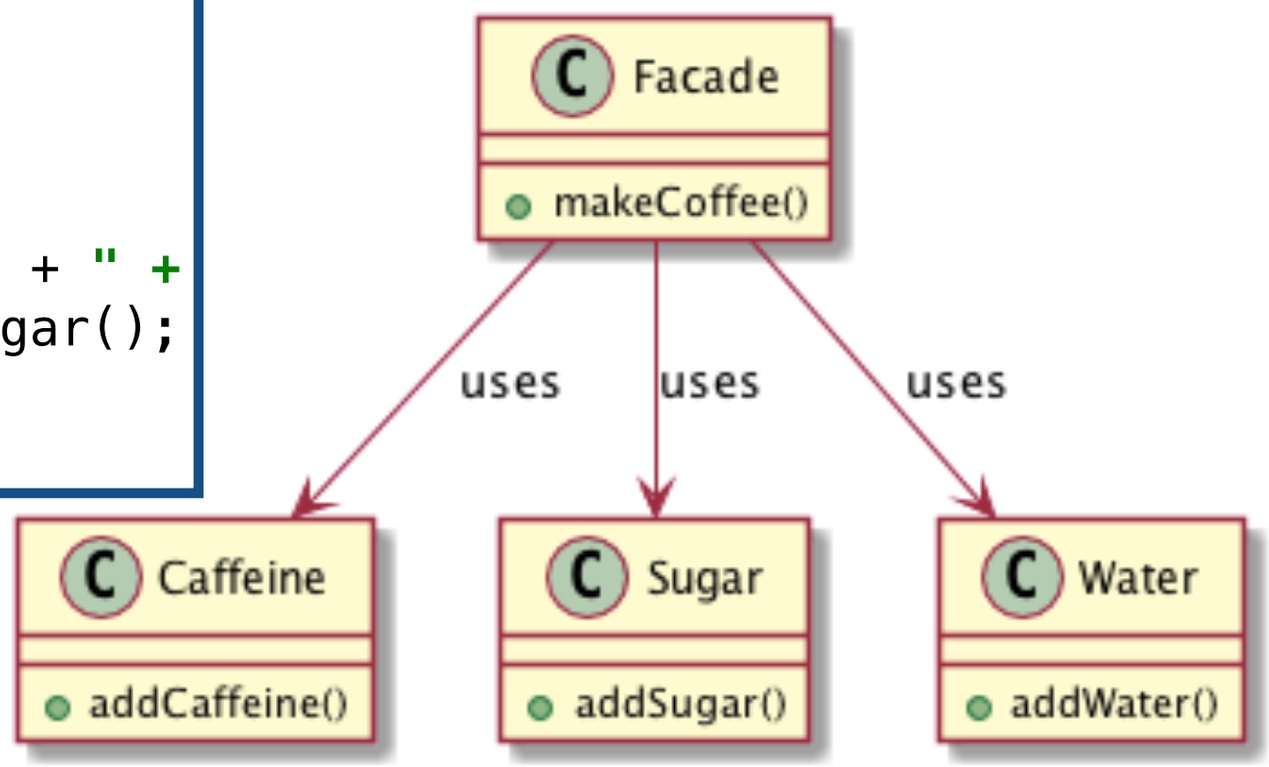
- Dóna funcionalitats més limitades que les que realment té el subsistema
- Pot esdevenir un objecte Deu

3.5.1. Patrons de disseny: Patró Façana (Facade)

```
class Facade {

  public String makeCoffee() {
    Sugar s = new Sugar();
    Caffeine c = new Caffeine();
    Water w = new Water();

    return "Coffee = " + w.addWater() + " +
" + c.addCaffeine() + " + " + s.addSugar();
  }
}
```



3.5.1. Patrons de disseny: Patró Façana (Facade)

- Reducció de l'acoblament client-subsistema
  - Es pot reduir l'acoblament fent que la **façana** sigui una classe abstracta amb subclasses concretes per les diferents implementacions del subsistema. Els clients es comuniquen amb el subsistema utilitzant la interfície de la classe façana abstracta (patró **Adapter**)
  - Una altra possibilitat és configurar l'objecte façana amb diferents objectes abstractes del subsistema. Per personalitzar la façana només cal canviar un o varis objectes del subsistema
  - El patró **Abstract Factory** es pot utilitzar junt amb la **Facade** per crear objectes del subsistema de manera independent
- Classes del subsistema privades i públiques
  - En Java es pot usar els paquets per determinar les classes que són visibles fora o que no seran visibles.

Exemple: https://springframework.guru/gang-of-four-design-patterns/facade-pattern/