

Classe 25.10.2021: Disseny: SOLID

Anna Puig

Enginyeria Informàtica

Facultat de Matemàtiques i Informàtica,

Universitat de Barcelona

Curs 2021/22

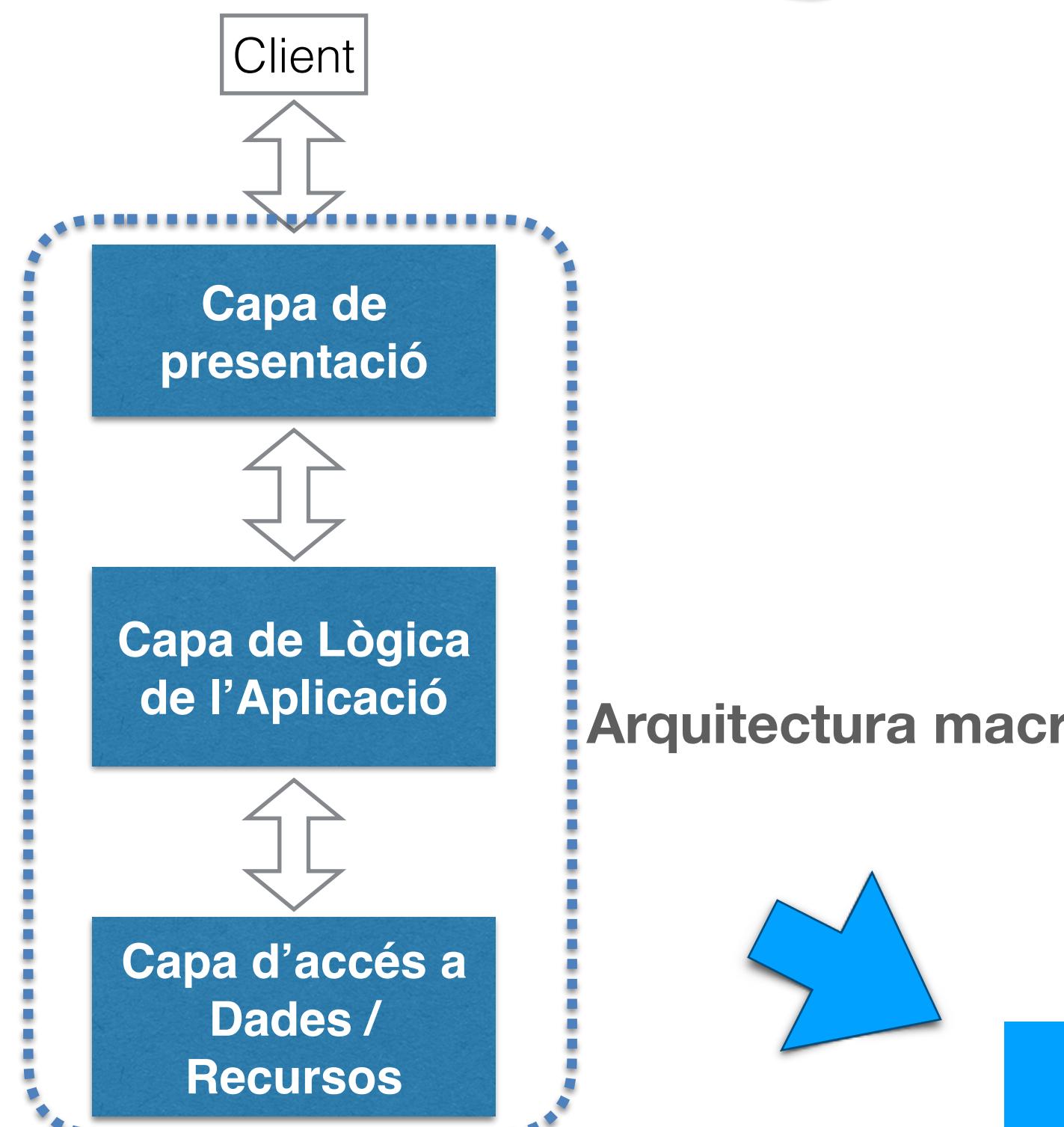
Sessió 25.10

Activitats

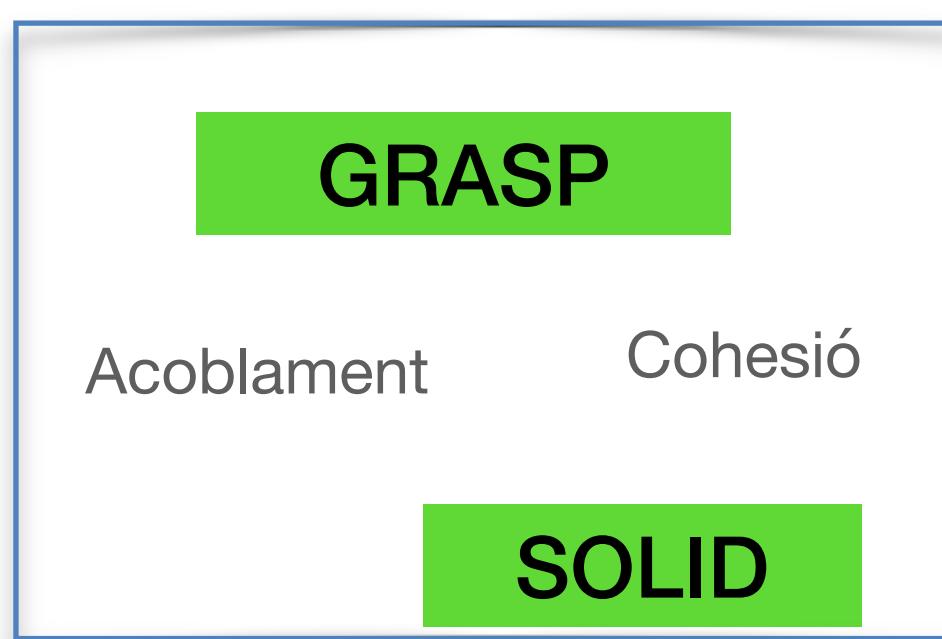
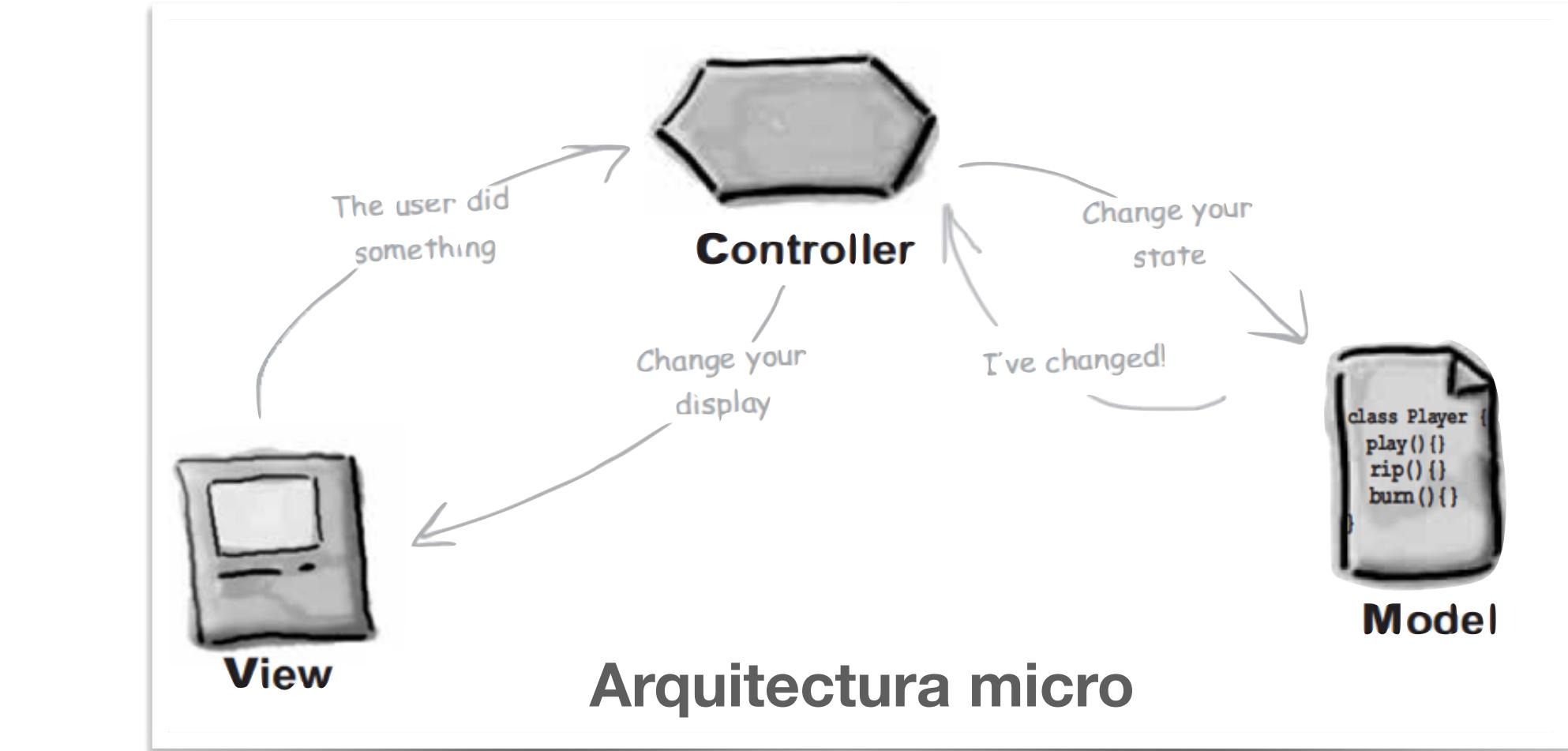
1. Breu explicació del CeXtrem-DAO
2. Exploració dels **projectes SOLID**



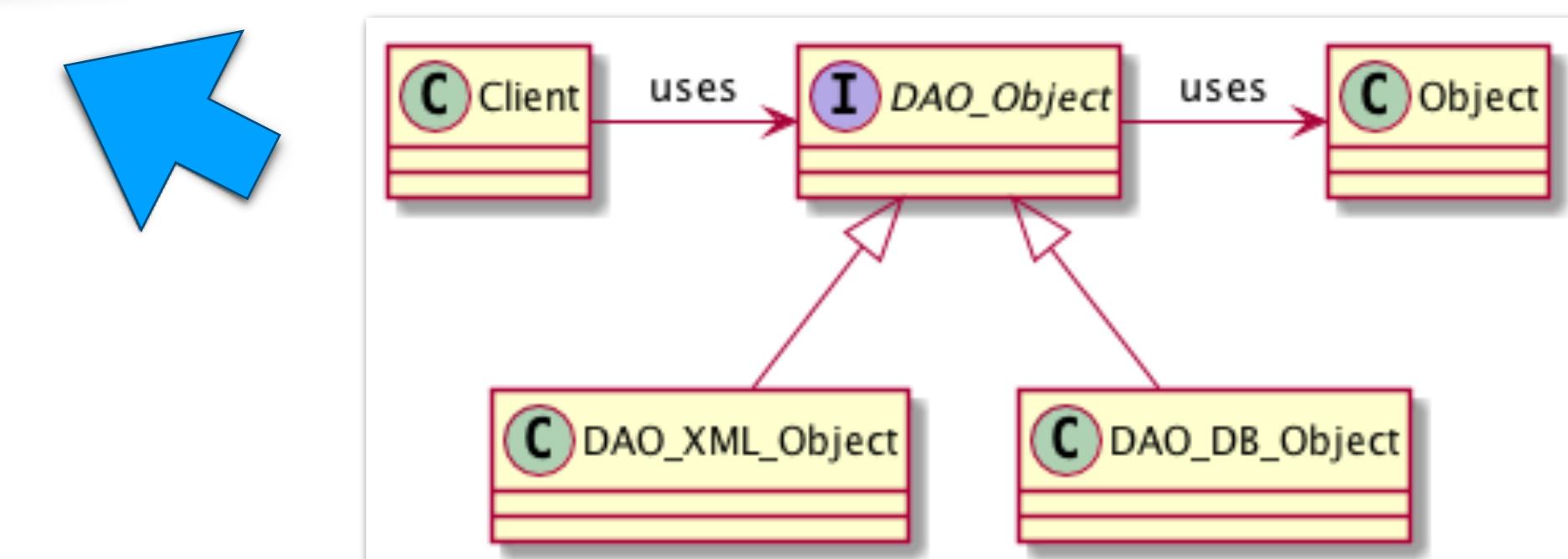
Visió general de la pràctica2



Disseny de l'aplicació



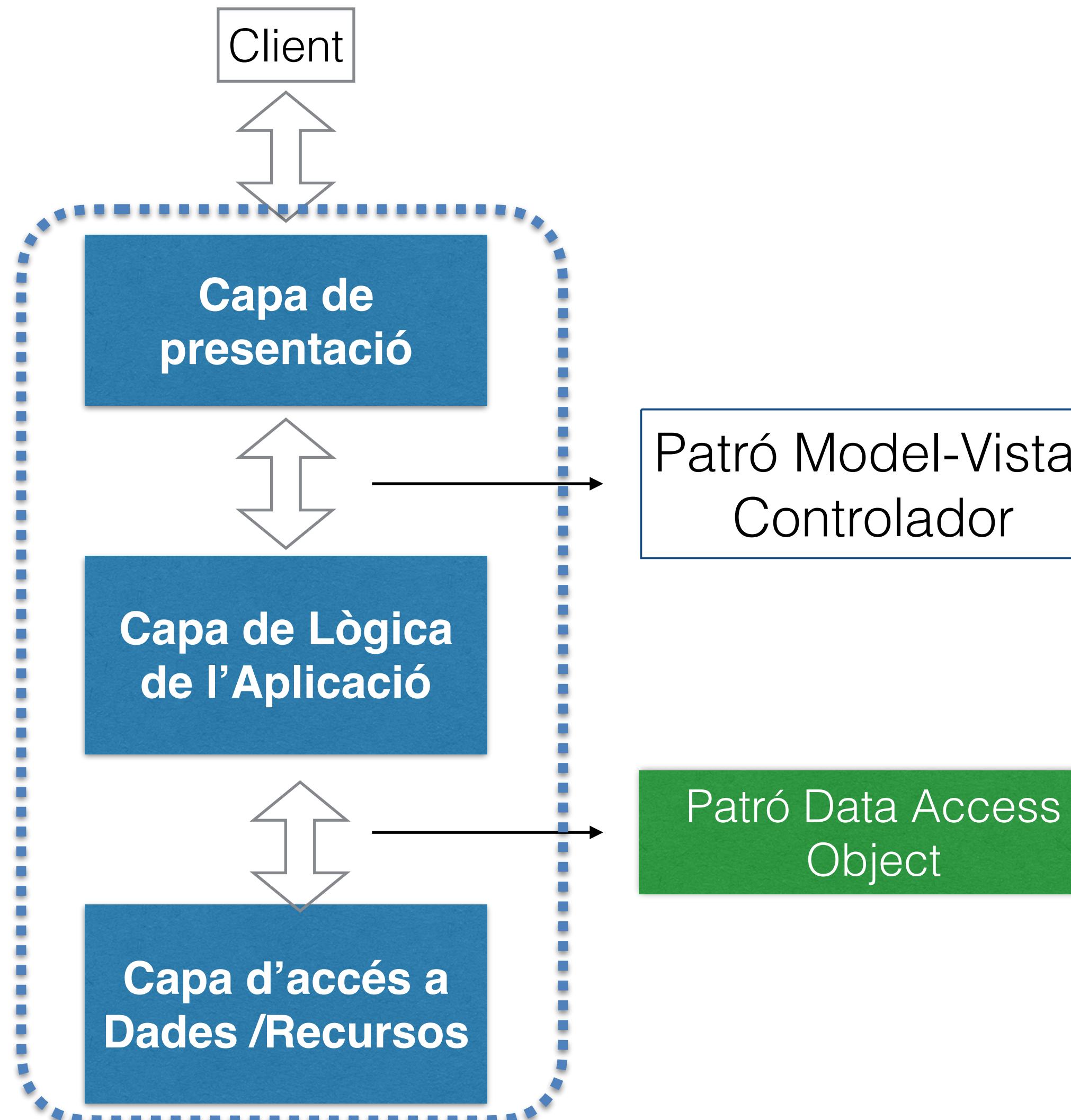
Criteris de disseny



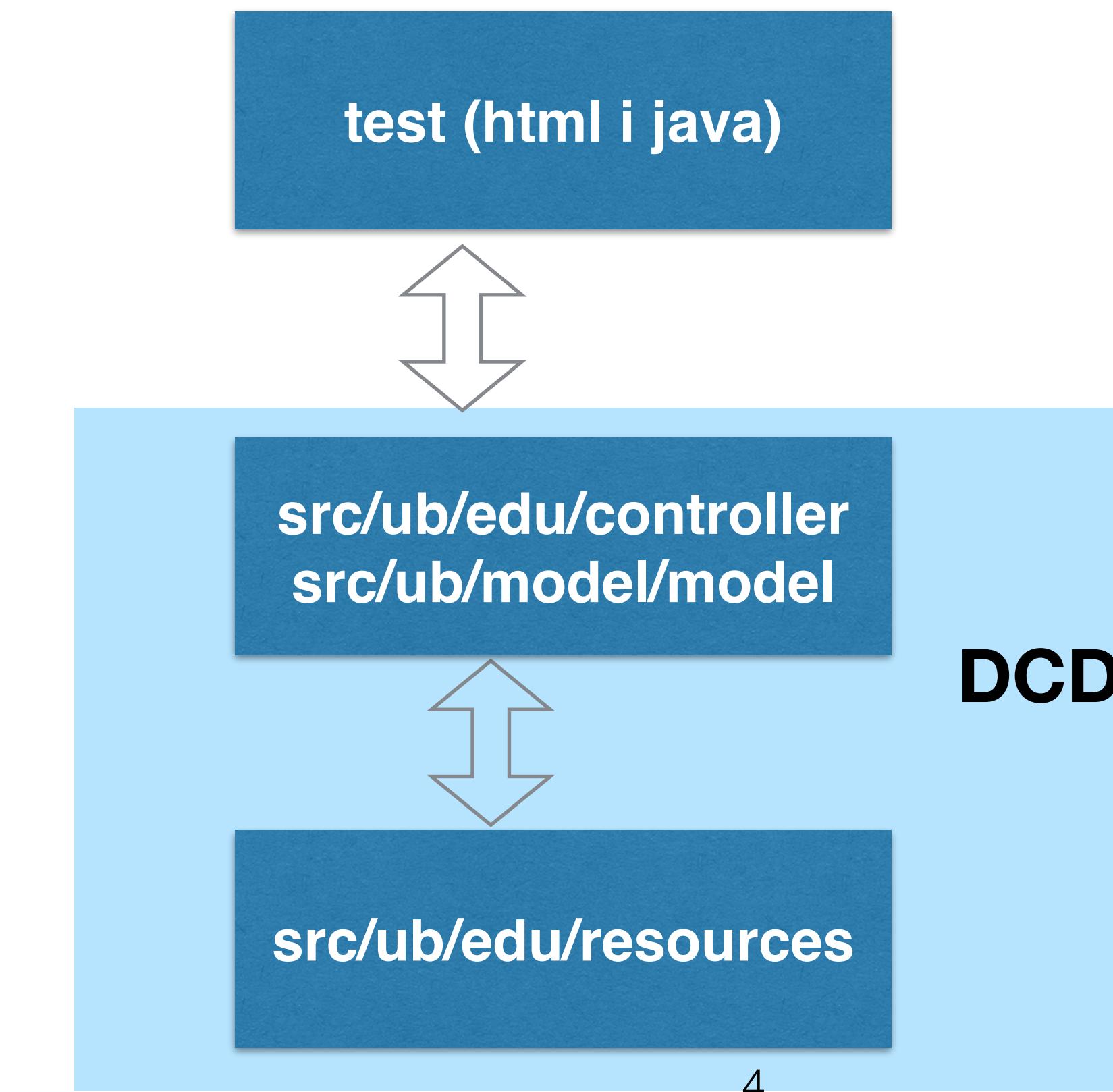
Patrons de disseny

Projecte de la pràctica 2

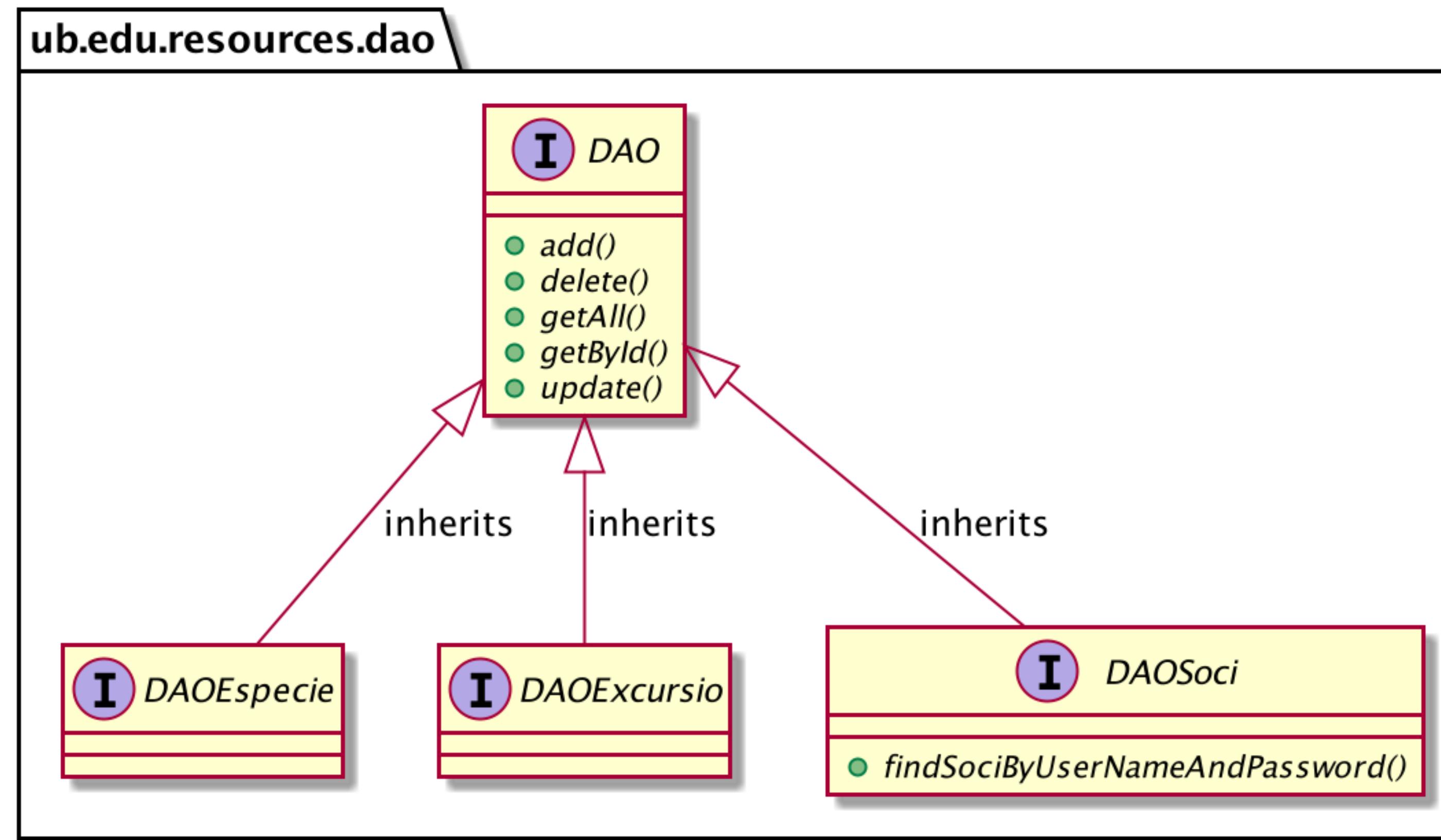
Patró per capes:



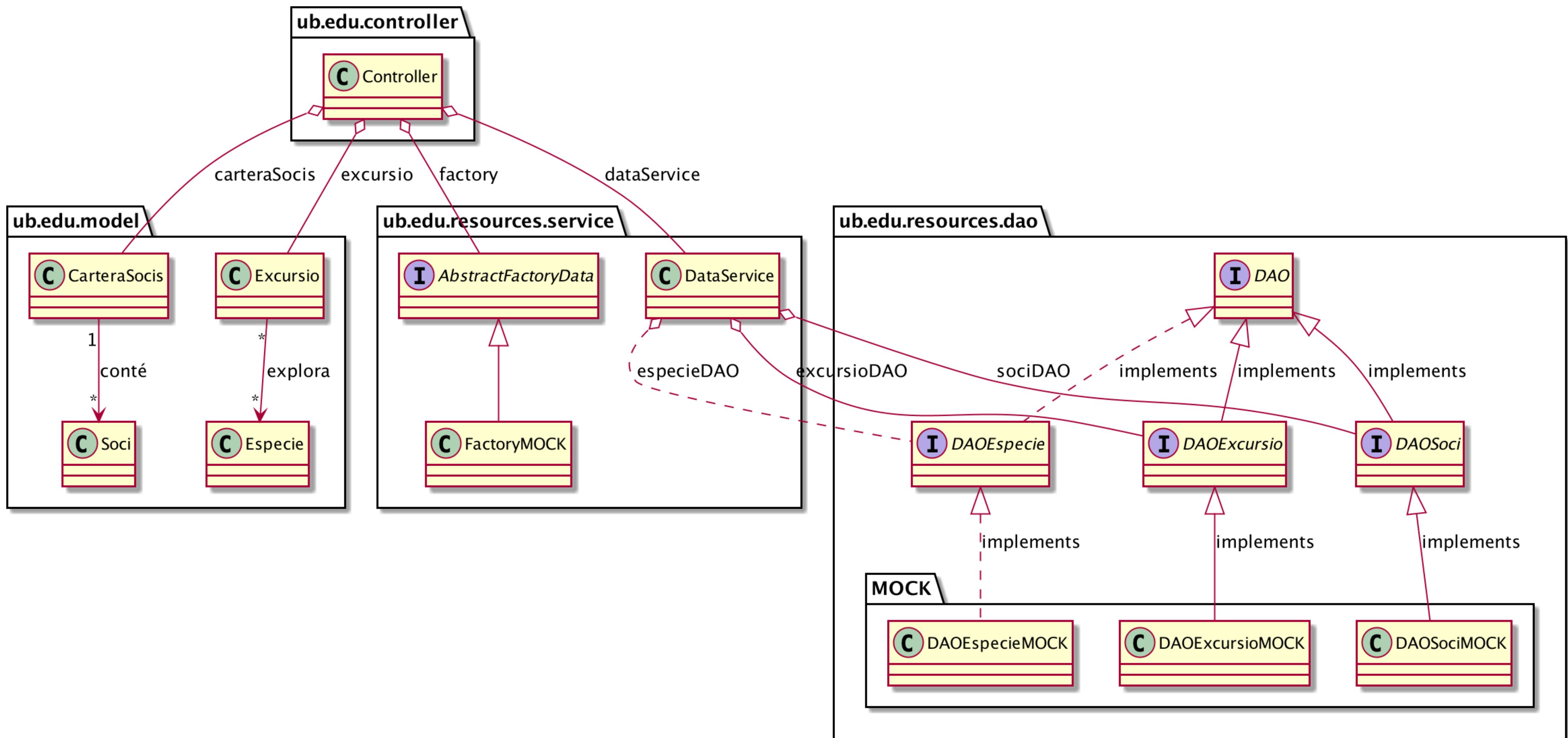
Pràctica 2



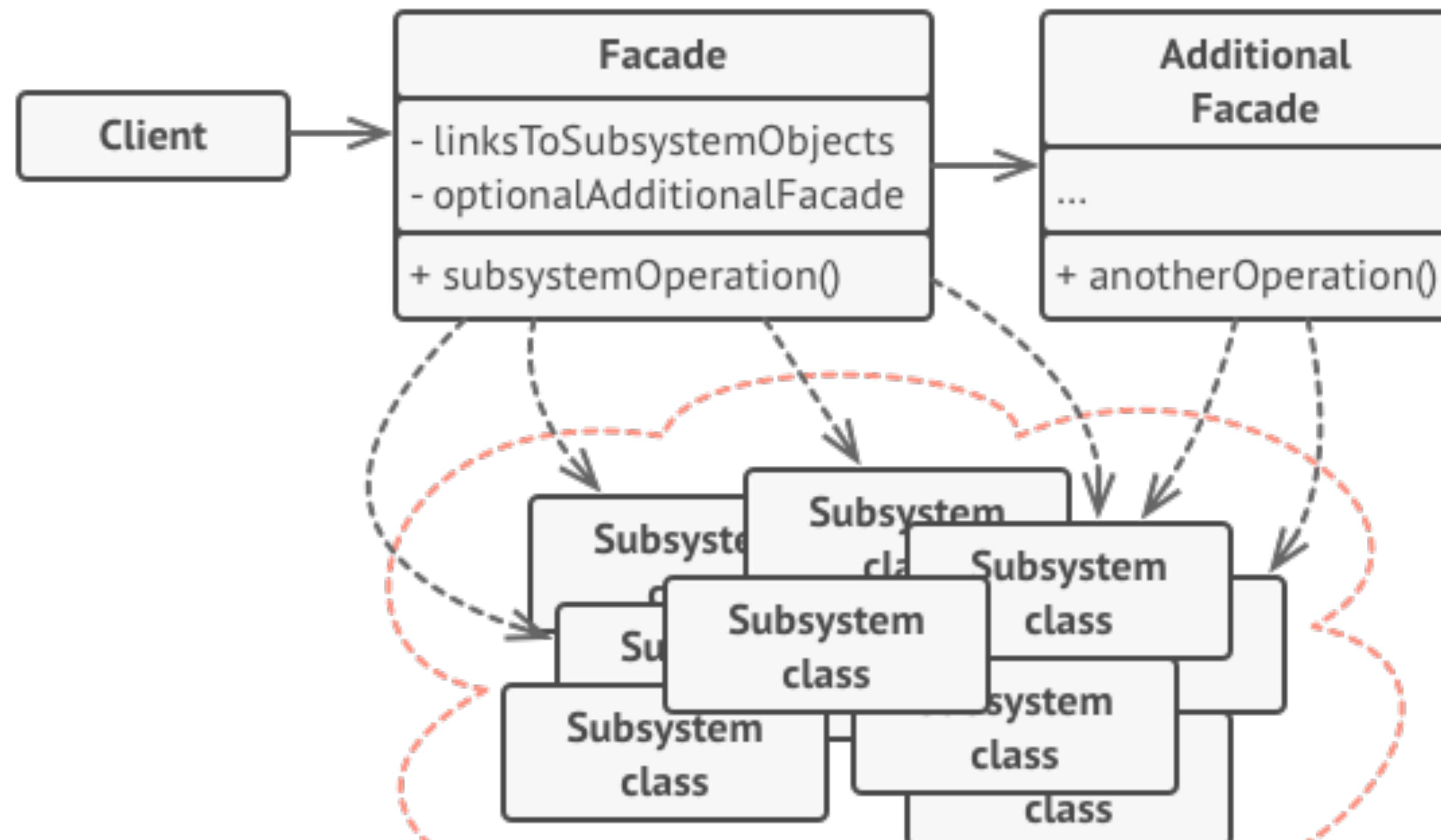
Capa de persistència: DAO



Arquitectura general



3.5.1. Patrons de disseny: Patró Façana (Facade)



Pràctica 2

- **Parts:**

1. Refactoritzar seguint els criteris GRASP
 1. Respostes al document de l'enunciat
 2. Projecte
2. Noves prestacions en persistència:
 1. Respostes a l'enunciat
 2. [OPT] Connexió amb la capa de persistència utilitzant el patró DAO

Test d'acceptació	Criteris GRASP	Breu explicació de les classes canviades

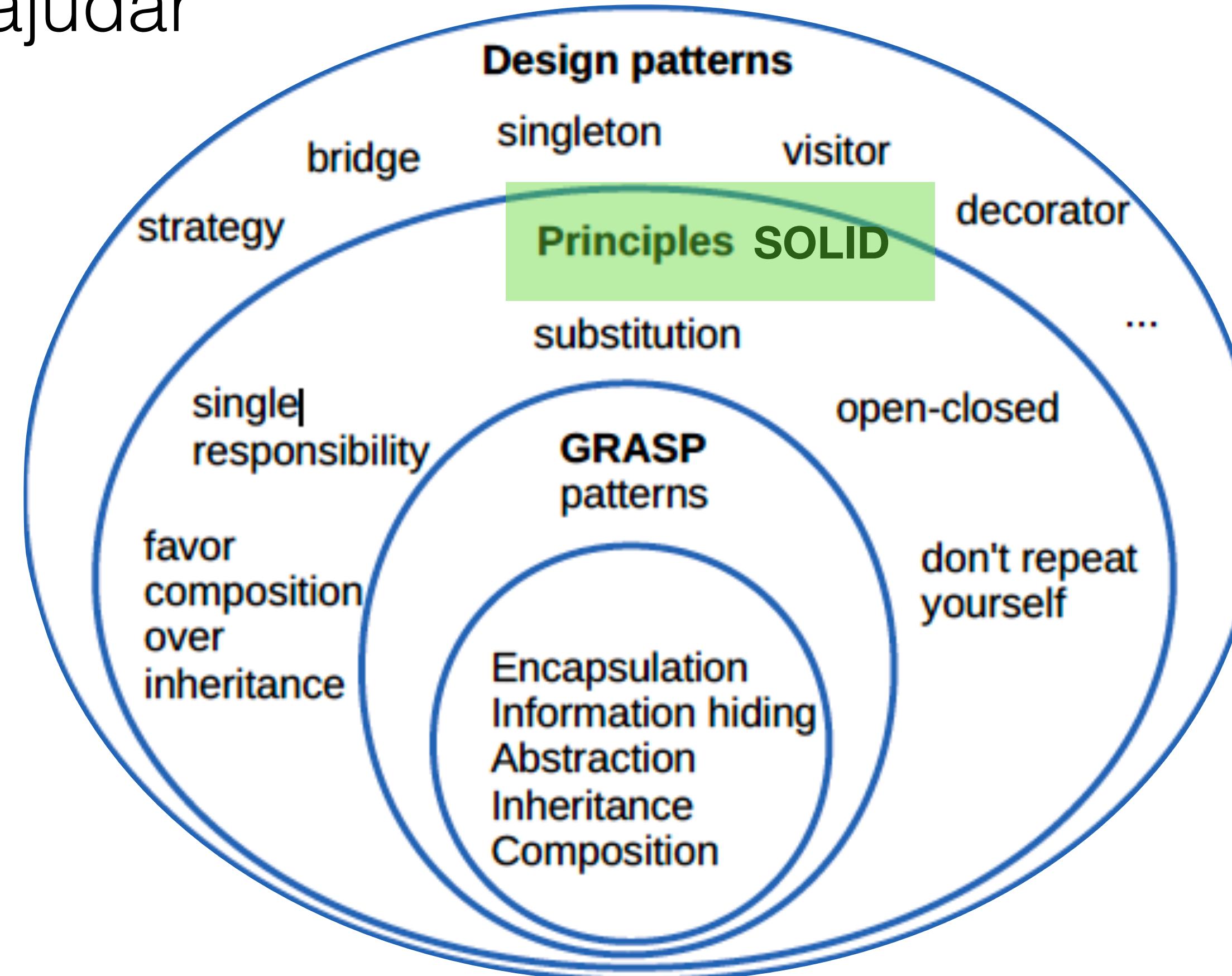
Patró GRASP usat a la capa de persistència	Breu Justificació
Per exemple, Cohesió alta a les classes DAO-MOCK	Les classes DAO-MOCK internament només tenen la responsabilitat de gestionar les dades com si fossin una base de dades i cap altra

Temari

1	Introducció al procés de desenvolupament del software
2	Anàlisi de requisits i especificació
3	Disseny
4	Del disseny a la implementació
5	Ús de frameworks de testing
	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

Com disseny en OO?

- No hi ha una metodologia que doni el millor disseny però hi han principis, heurístiques i patrons que poden ajudar



3.4. Principis de Disseny: S.O.L.I.D.

Principis de disseny [Robert C. Martin 98]:

- **S**: Single Responsibility Principle
- **O**: Open-Close Principle
- **L**: Liskov Substitution Principle
- **I**: Interface Segregation Principle
- **D**: Dependency Inversion Principle

"One class should have one and only one responsibility"

"Software components should be open for extension, but closed for modification"

"Derived types must be completely substitutable for their base types"

"Clients should not be forced to implement unnecessary methods which they will not use"

"Depend on abstractions, not on concretions"



[Articles de suport de cada principi](#)

3.4. Principis de Disseny: S.O.L.I.D.

Single Responsibility Principle:

"One class should have one and only one responsibility"

De Marco 79 and Page-Jones 88



- Una classe ha de tenir **una i només una responsabilitat**
- Responsabilitat s'entén com a raó que fa canviar la classe
- Una classe hauria de tenir només una raó per a ser canviada
- Això permet tenir **ALTA** cohesió i evitar classes **fràgils**

3.4. Principis de Disseny: S.O.L.I.D.

Open Closed Principle:

"Software components should be open for extension, but closed for modification"

Meyer, 88



- Els mòduls (classes, funcions, operacions, etc.) haurien de ser:
 - **Oberts** per extensió: per satisfer nous requisits
 - **Tancats** per modificació: L'extensió no implica canvis en el codi del mòdul. No s'ha de tocar la versió **executable** del mòdul.
- El comportament dels mòduls que satisfan aquest principi es canvia afegint nou codi, i no pas canviant codi existent.
- L'ús correcte del **polimorfisme** afavoreix aquest principi

3.4. Principis de Disseny: S.O.L.I.D.

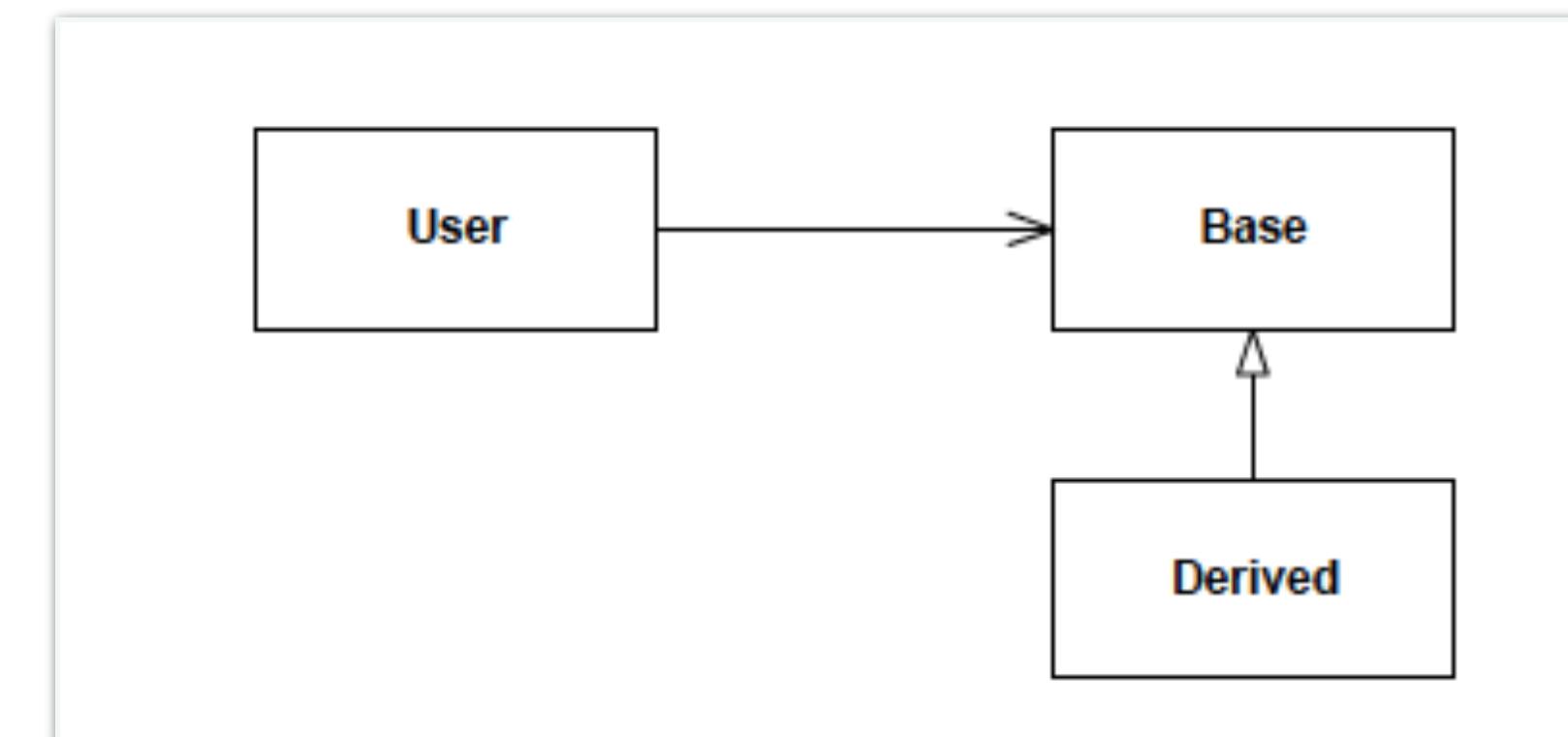
Liskov Substitution Principle:

"Derived types must be completely substitutable for their base types"

Barbra Liskov 1988



- Donada una entitat **Base** amb un cert mètode i altres subentitats **Derivades** que implementen el mètode original, el comportament d'un objecte **o1** que era de tipus Base no ha de variar si s'usa **o2** de tipus Derivat



3.4. Principis de Disseny: S.O.L.I.D.

Interface Segregation Principle:

"Clients should not be forced to implement unnecessary methods which they will not use"

Gamma et al., 1995



- No s'ha d'obligar a les classes a dependre de classes o mètodes que no han d'usar (deriva del SRP)
- Tot i que hi hagin classes molt grans que inclouen molts mètodes (interfícies no cohesionades), els clients (altres classes) només haurien de conèixer classes abstractes que tinguin interfícies cohesionades.
- Això permet tenir **ALTA** cohesió i evitar classes **fràgils**

3.4. Principis de Disseny: S.O.L.I.D.

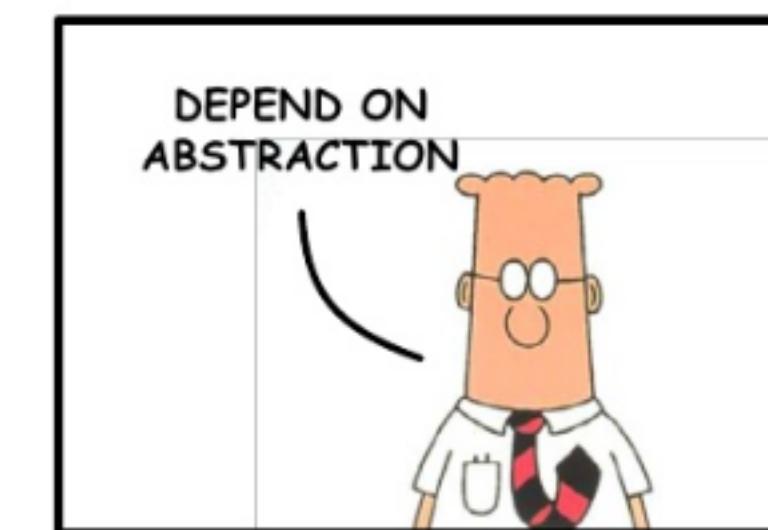
Dependency Inversion Principle:

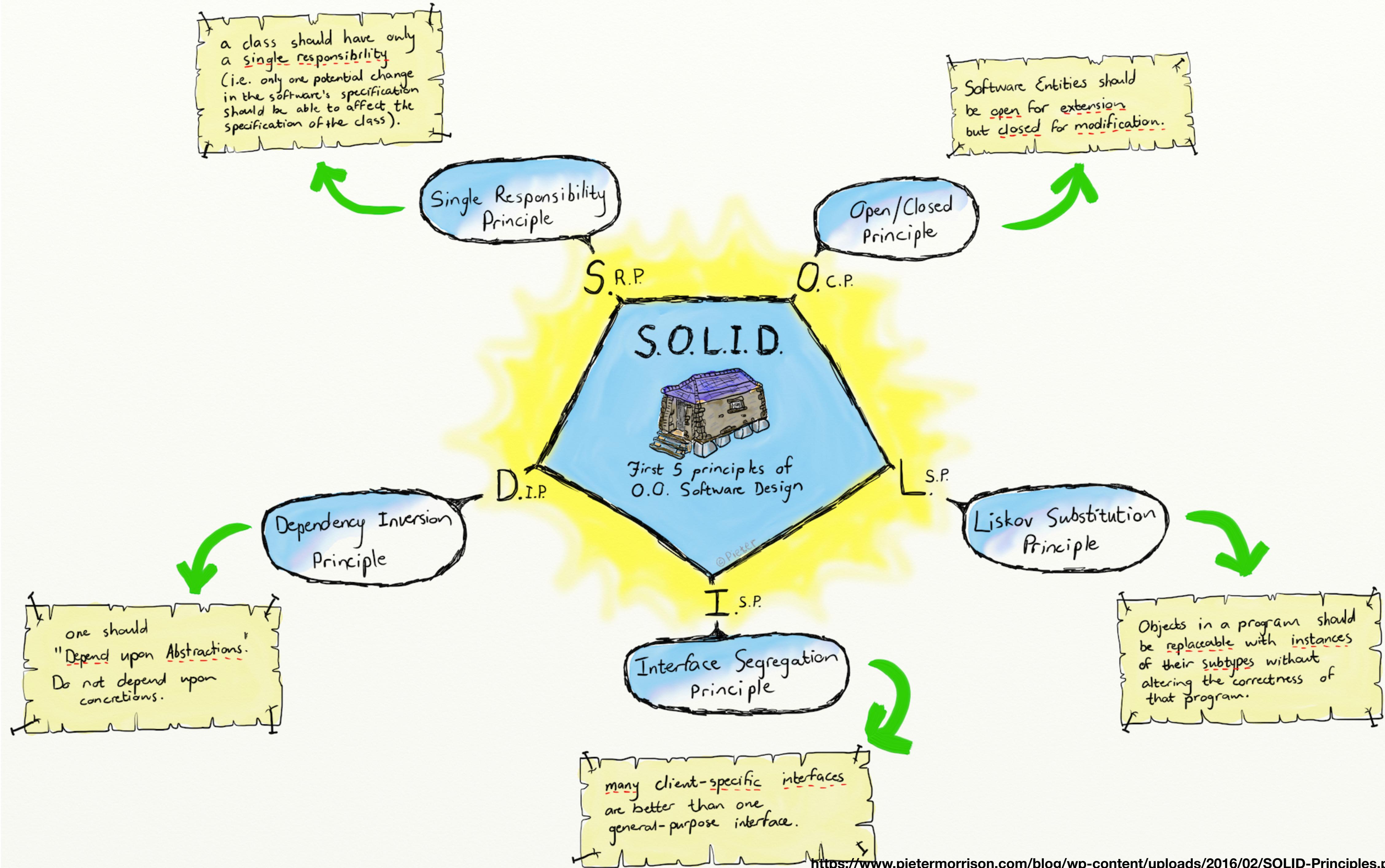
"Depend on abstractions, not on concretions"

Martin, 1997



- Les classes d'alt nivell no han de canviar per que canvien les classes més senzilles o de baix nivell. Les dues haurien de dependre d'abstraccions.
- Les abstraccions no han de dependre de detalls tecnològics, són els detalls que haurien de dependre de les abstraccions.
- Aplicar aquest principi dóna **BAIX** acoblament

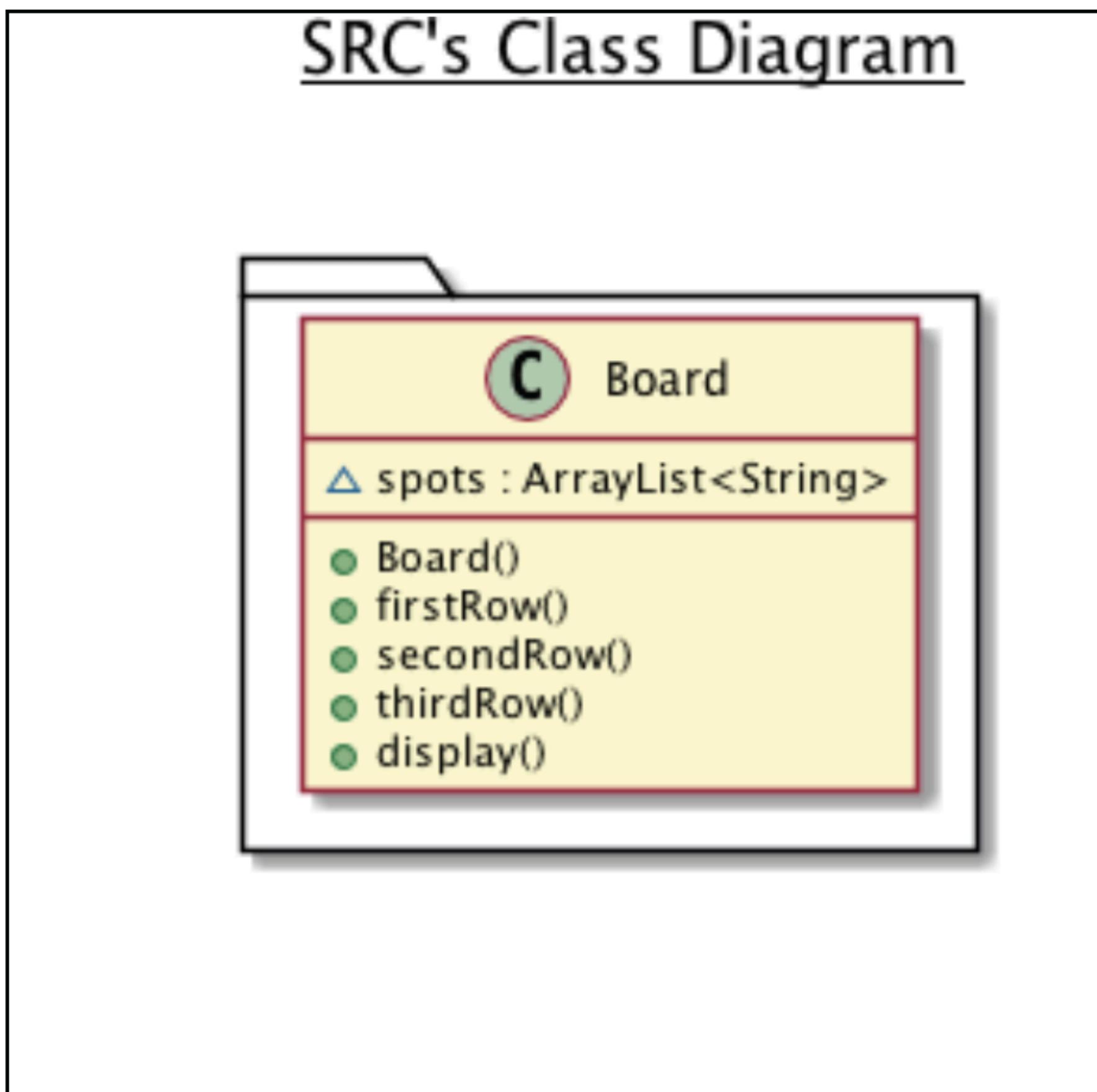




Projectes a explorar

- Repassa els principis via aquesta guia: <https://devexperto.com/principios-solid/>
- En el Campus, a la secció de Problemes, teniu 5 projectes per detectar vulneracions de principis SOLID i proposar solucions que permetin no vulnerar-los. Trobaràs l'enunciat <https://campusvirtual.ub.edu/mod/resource/view.php?id=2668592>
- Baixa els projectes iobre'ls dins del IntelliJ. Genera el seu diagrama de classes amb el Skecht it! i explora quins principis vulneren.

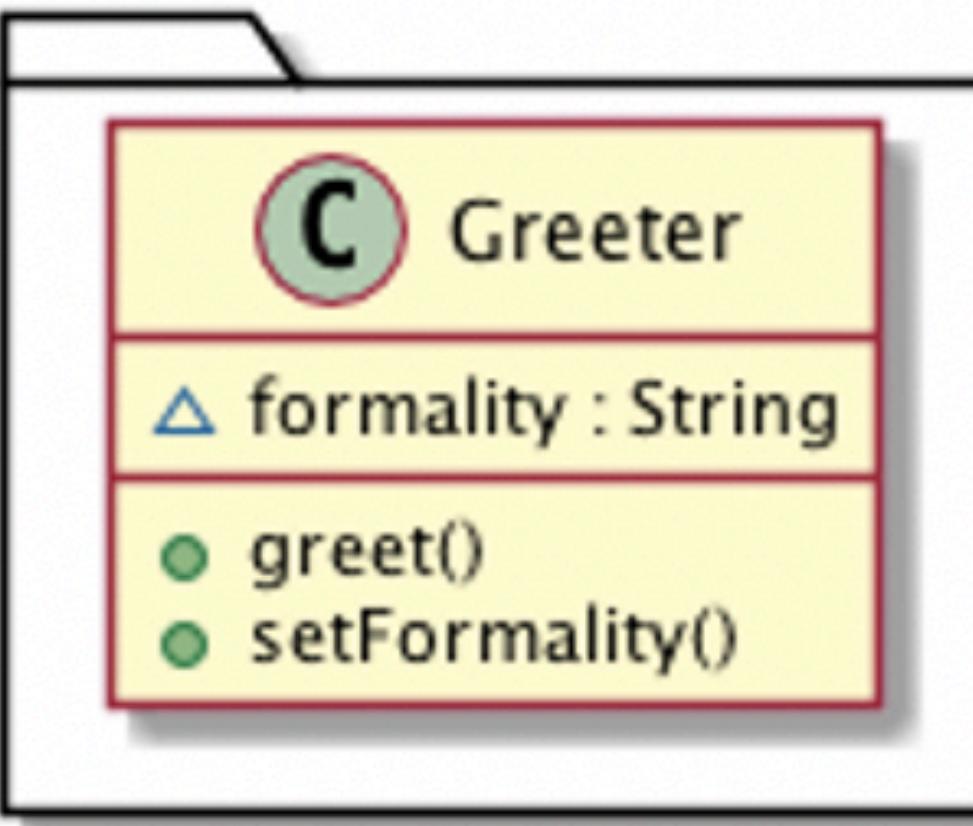
2. Exploració dels projectes: Projecte 1



Què fa aquesta classe?

Fa el taulell i el visualitza?

2. Exploració dels projectes: Projecte 2



A UML class diagram showing a class named 'Greeter'. It has a private attribute 'formality' of type String, and two public methods: 'greet()' and 'setFormality()'. The class is represented by a yellow rounded rectangle with a red border.

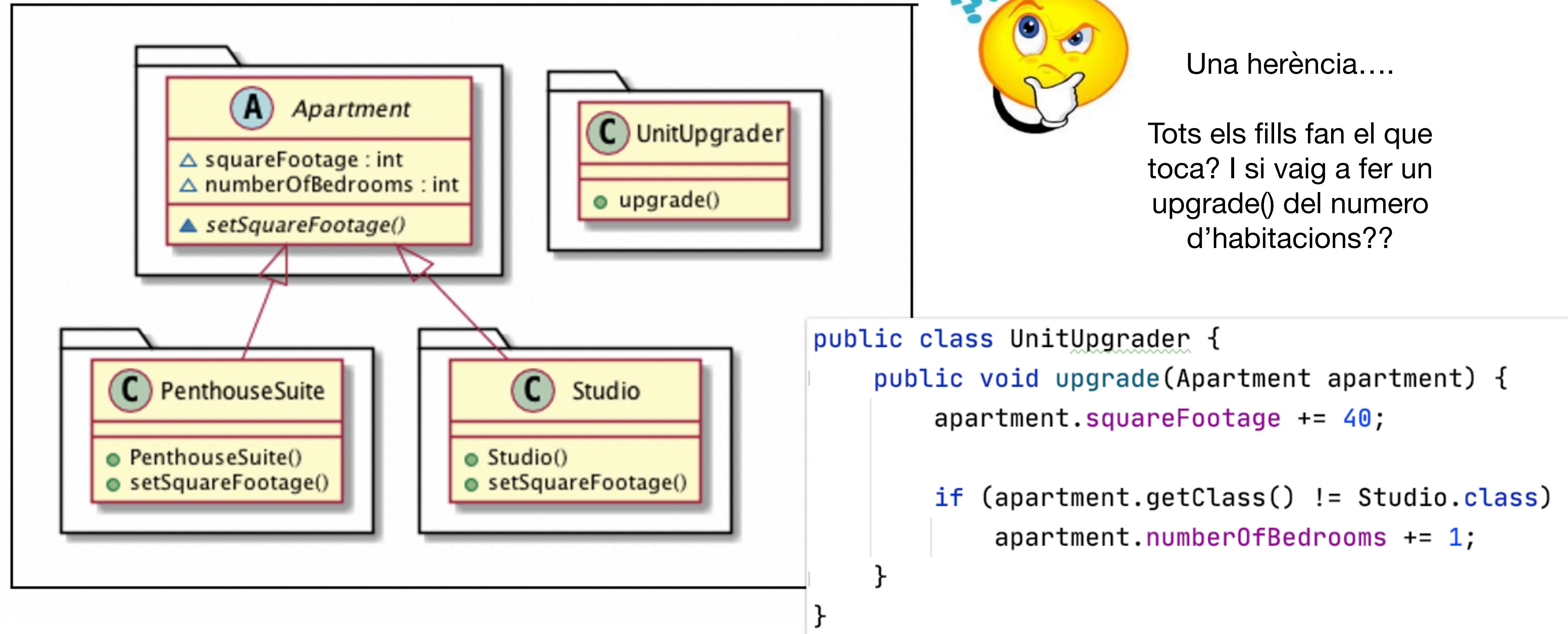


A cartoon emoji of a yellow circular face with a thinking expression, featuring a hand on its chin and question marks above its head.

Què fa aquesta classe?

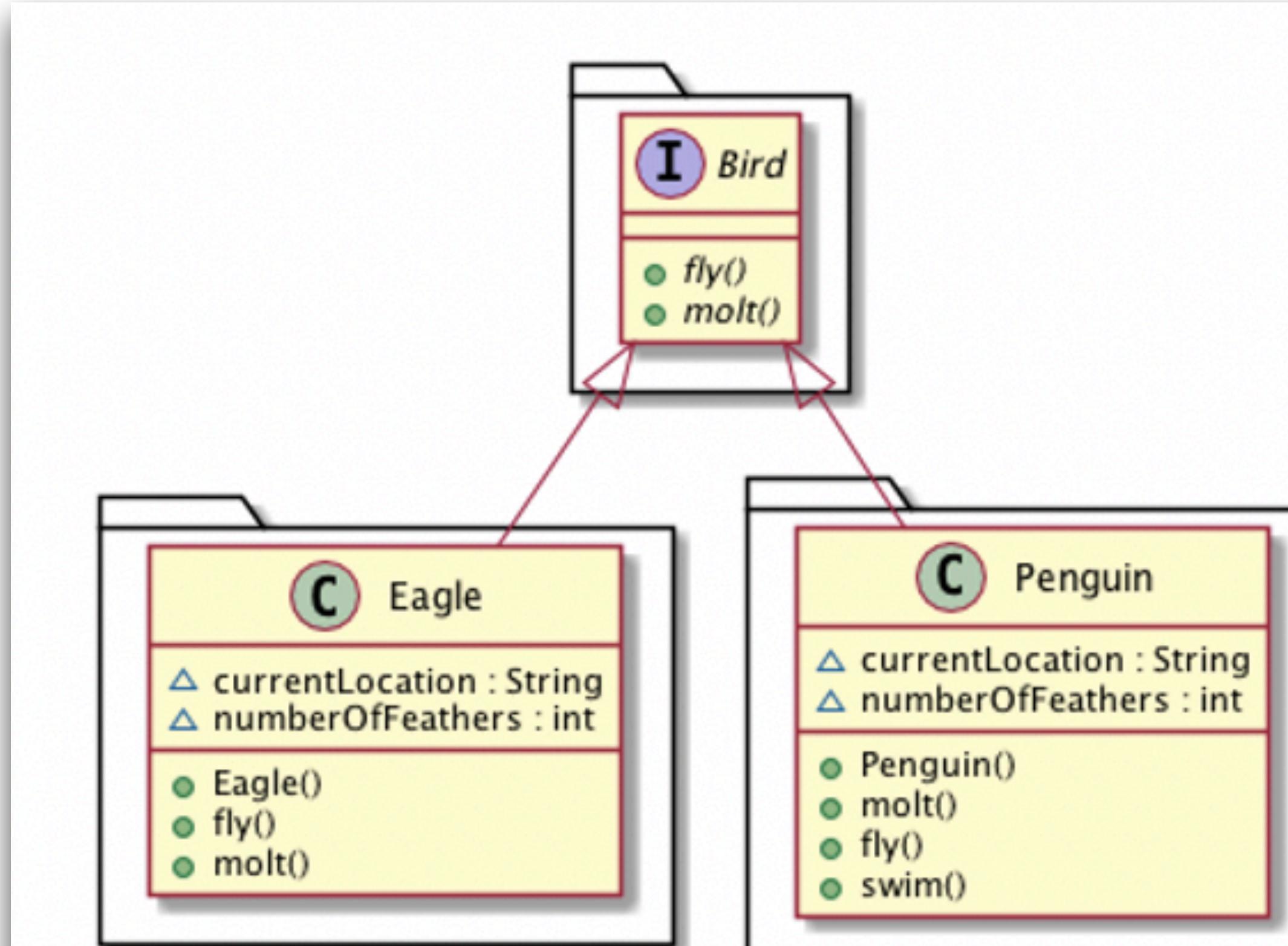
```
public class Greeter {  
    String formality;  
  
    public String greet() {  
        if (this.formality == "formal") {  
            return "Good evening, sir.";  
        }  
        else if (this.formality == "casual") {  
            return "Sup bro?";  
        }  
        else if (this.formality == "intimate") {  
            return "Hello Darling!";  
        }  
        else {  
            return "Hello.";  
        }  
    }  
  
    public void setFormality(String formality) { this.formality = formality; }  
}
```

2. Exploració dels projectes: Projecte 3



2. Exploració dels projectes:

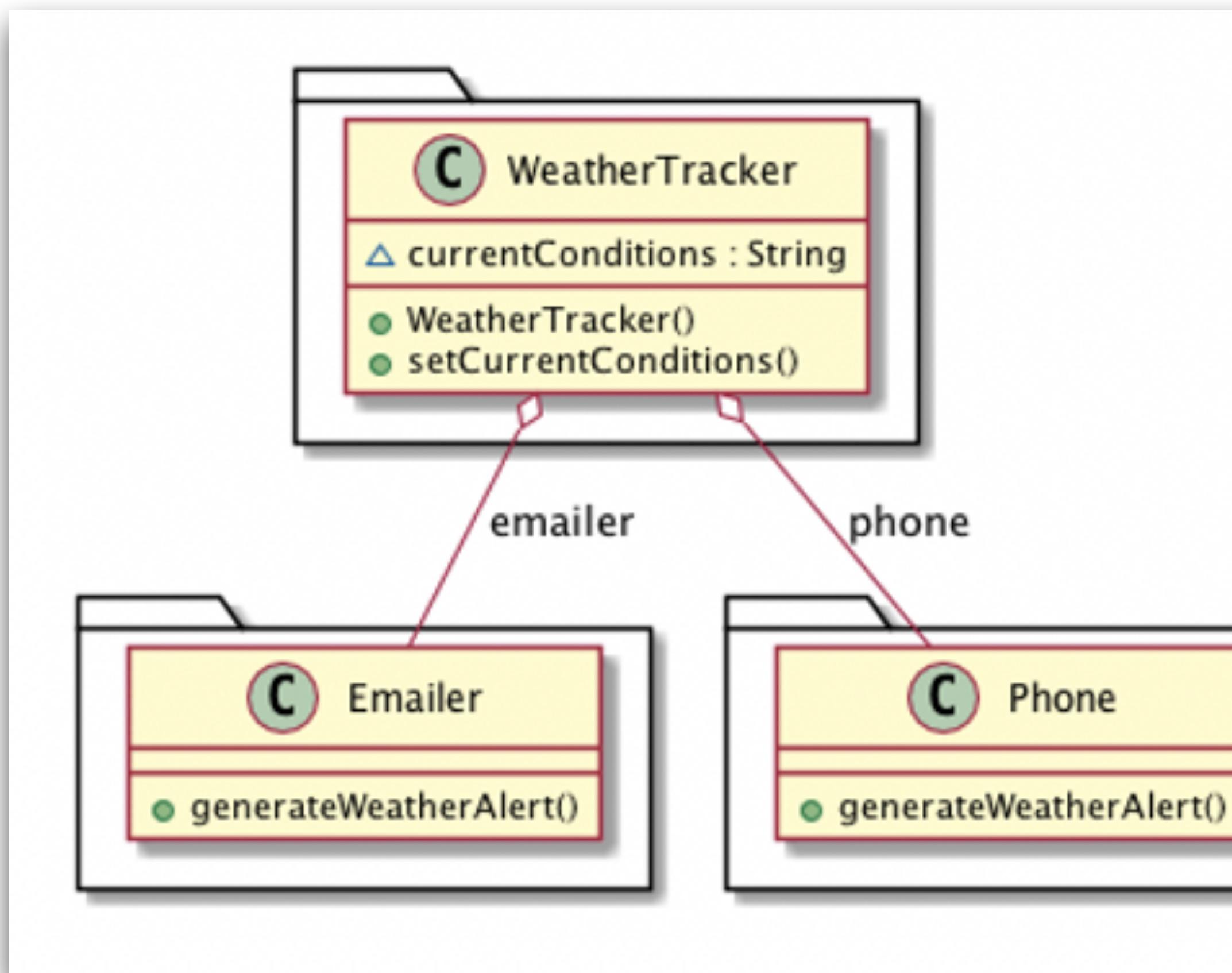
Projecte 4



Interfície ?
Està sobrecarregada?
Com són les seves
implementacions?

```
public class Penguin implements Bird {  
    String currentLocation;  
    int numberOfFeathers;  
  
    public Penguin(int initialFeatherCount) { this.numberOfFeathers = initialFeatherCount; }  
  
    @Override  
    public void molt() { this.numberOfFeathers -= 1; }  
  
    @Override  
    public void fly() { throw new UnsupportedOperationException(); }  
  
    public void swim() { this.currentLocation = "in the water"; }  
}
```

2. Exploració dels projectes: Projecte 5



WeatherTracker... dóna missatges per mail o per telèfon segons el temps que fa....

```
public class WeatherTracker {  
    String currentConditions;  
    Phone phone;  
    Emailer emailer;  
  
    public WeatherTracker() {  
        phone = new Phone();  
        emailer = new Emailer();  
    }  
  
    public void setCurrentConditions(String weatherDescription) {  
        this.currentConditions = weatherDescription;  
        if (weatherDescription == "rainy") {  
            String alert = phone.generateWeatherAlert(weatherDescription);  
            System.out.print(alert);  
        }  
        if (weatherDescription == "sunny") {  
            String alert = emailer.generateWeatherAlert(weatherDescription);  
            System.out.print(alert);  
        }  
    }  
}
```