Tema 3: Disseny

Anna Puig

Enginyeria Informàtica

Facultat de Matemàtiques i Informàtica,

Universitat de Barcelona

Curs 2019/2020

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 Principis de Disseny: S.O.L.I.D.
- 3.3 Patrons arquitectònics
- 3.4 Patrons de disseny

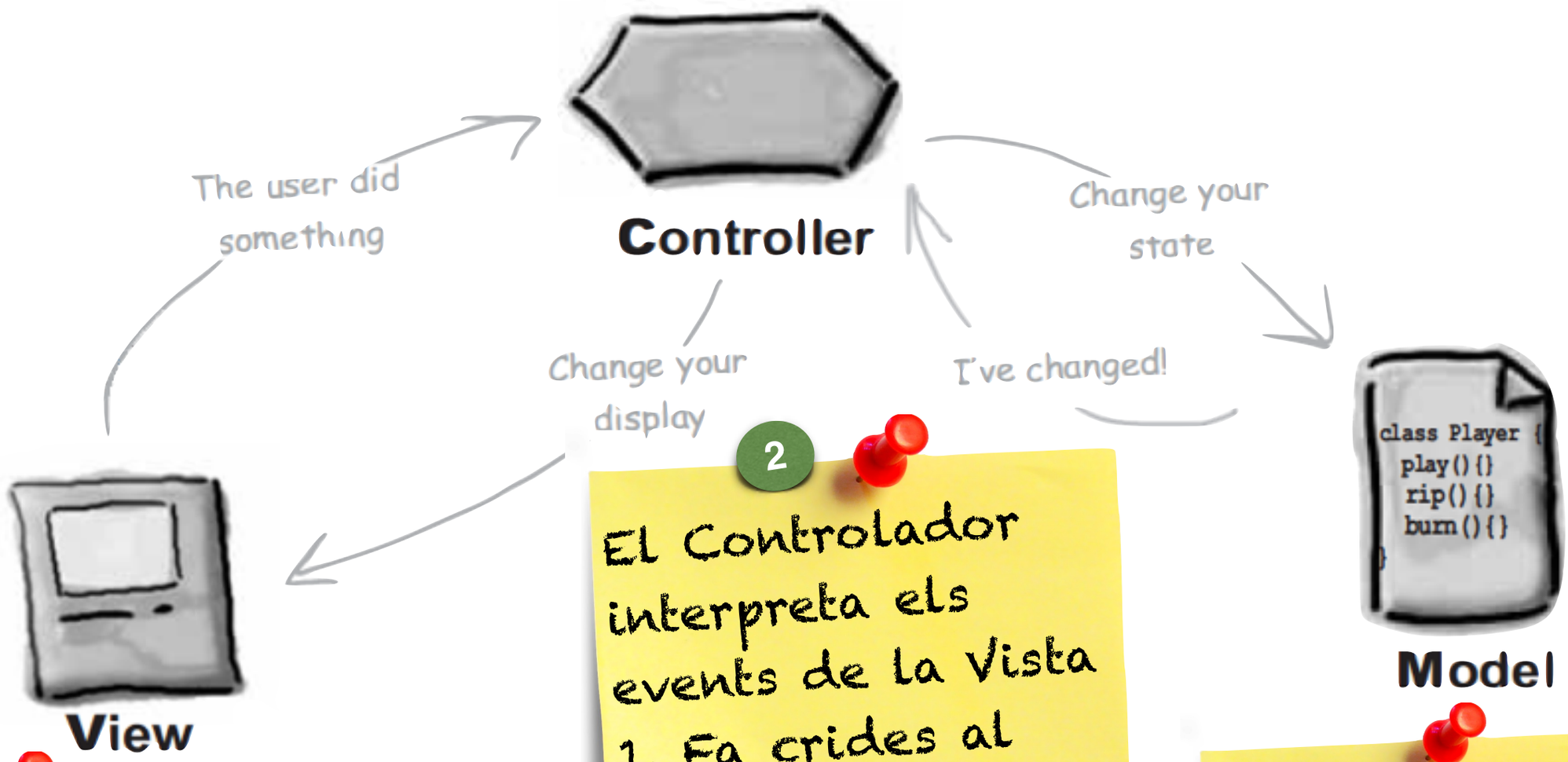
4 Del disseny a la implementació

5 Ús de frameworks de testing

3.4. 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

Model-Vista-Controlador
Aproximació senzilla (pràctica 2)

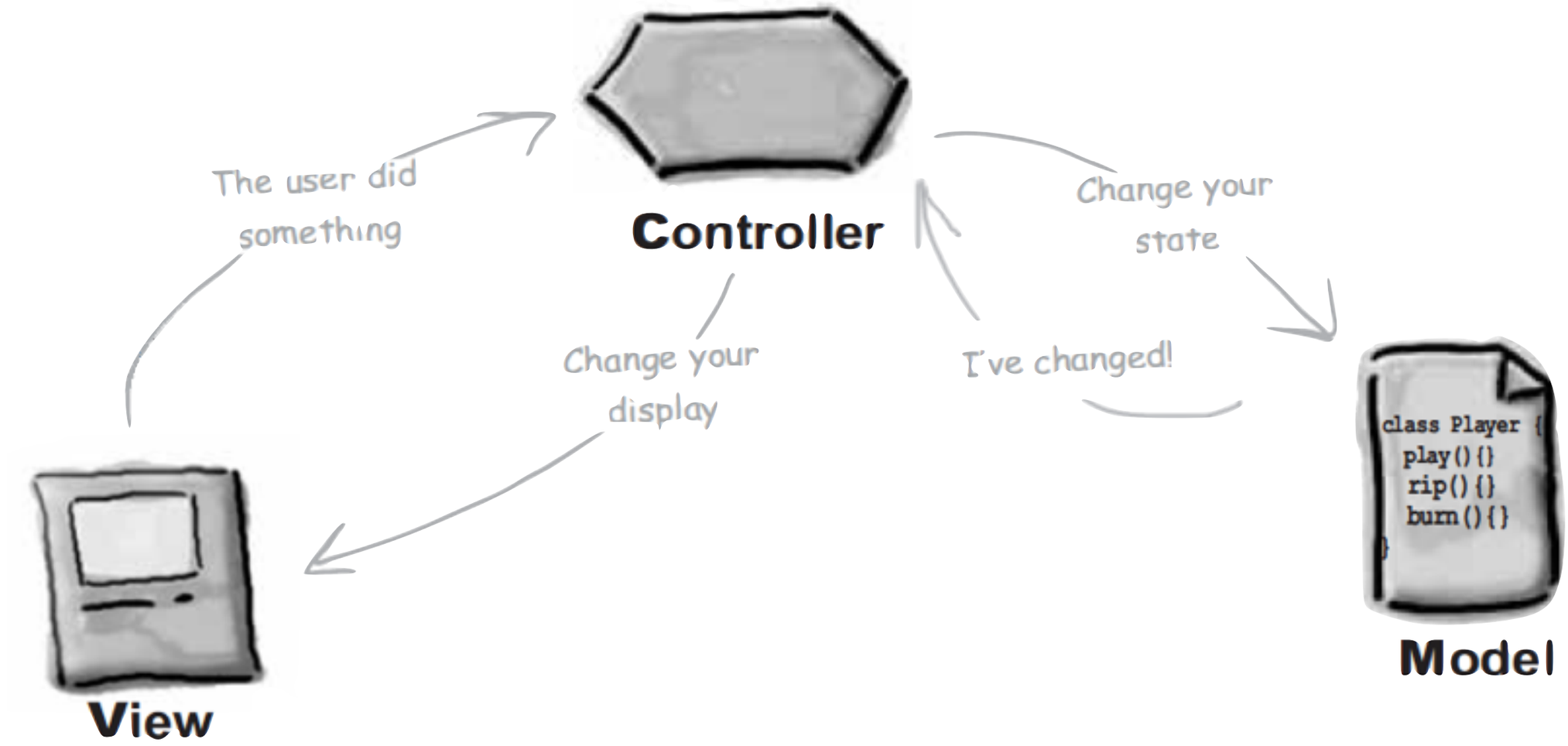


1. La Vista captura l'input de l'usuari i mostra l'estat del model

2. El Controlador interpreta els events de la Vista
   1. Fa crides al Model
   2. Fa crides a la Vista

3. El Model canvia el seu estat i ho notifica al Controlador

Model-Vista-Controlador
Aproximació senzilla (pràctica 2)



Problema:

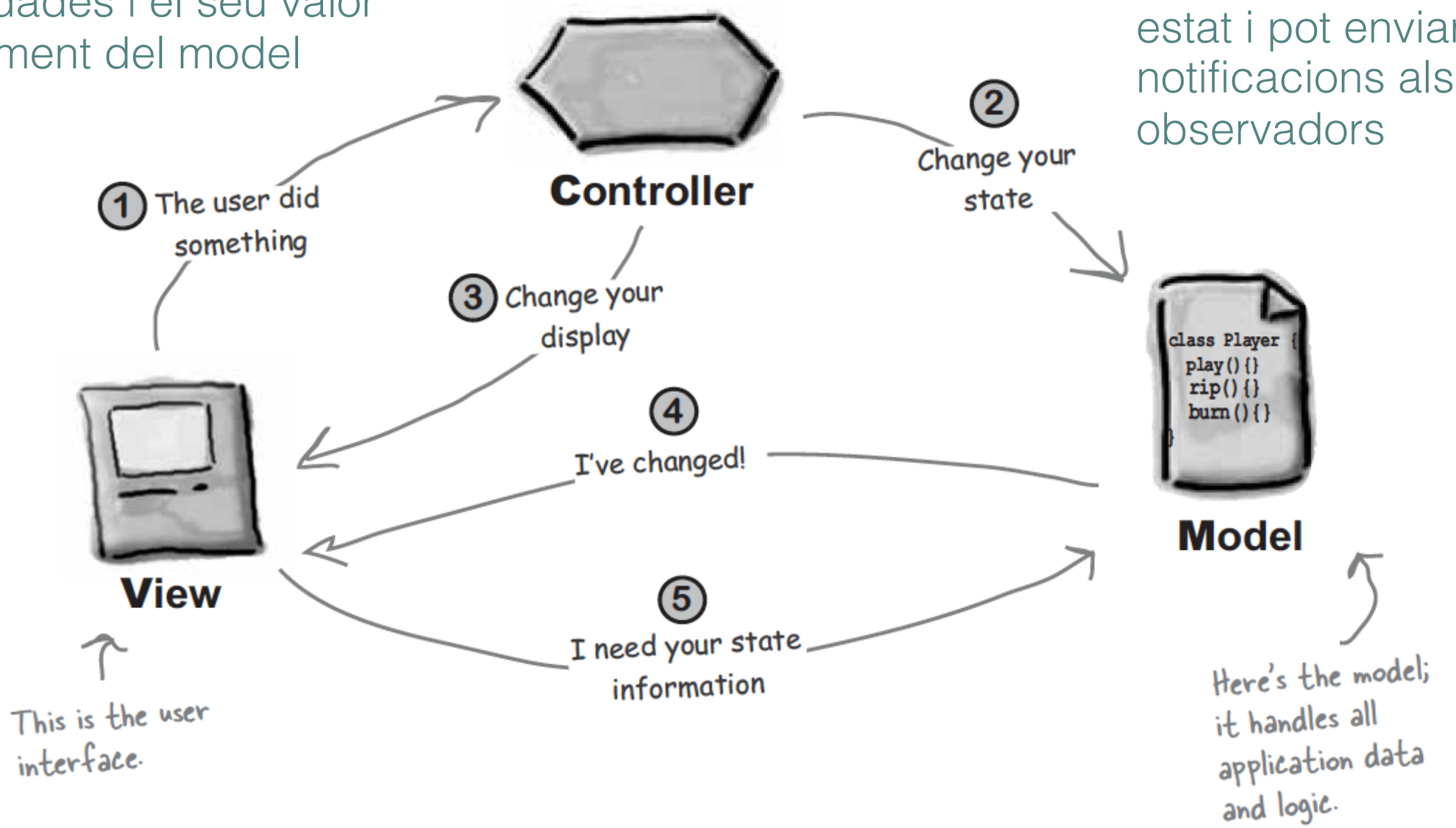
- Quan hi han actualitzacions independents en el model que no estan fetes des de la vista, com pot el model notificar aquests canvis a la Vista?
- El Controlador centralitza tota la comunicació produïnt retards a vegades innecessaris.

Model-Vista-Controlador
Aproximació general (pràctica 3)

VISTA:
Dóna la presentació del model. La vista normalment mostra l'estat de les dades i el seu valor directament del model

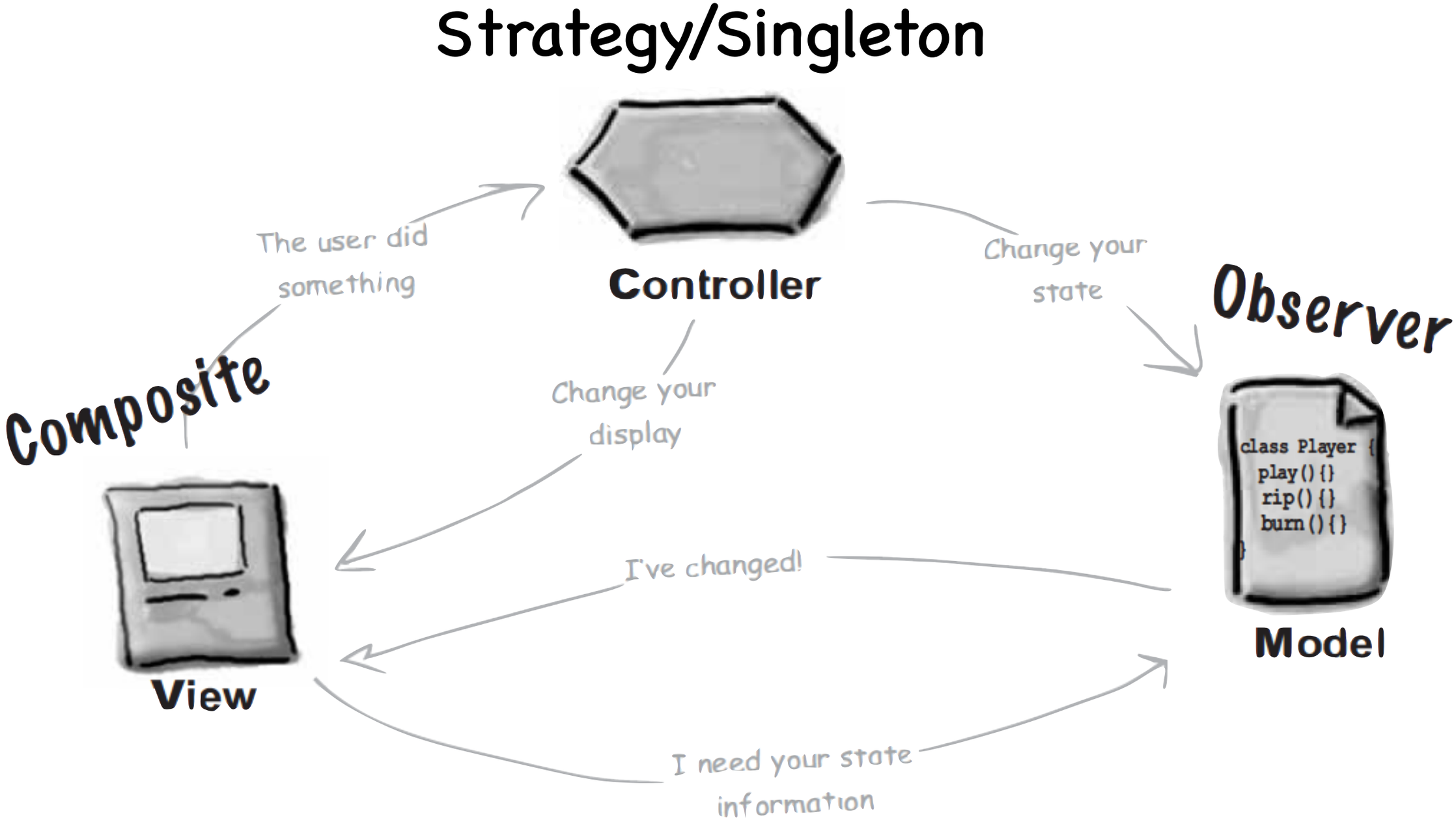
CONTROLADOR:
Agafa l'entrada de l'usuari i li dóna el què significa al model i actualitza la vista.

MODEL:
El model guarda totes les dates, l'estat i la lògica de l'aplicació. Dóna una interfície per manipular i donar el seu estat i pot enviar notificacions als observadors



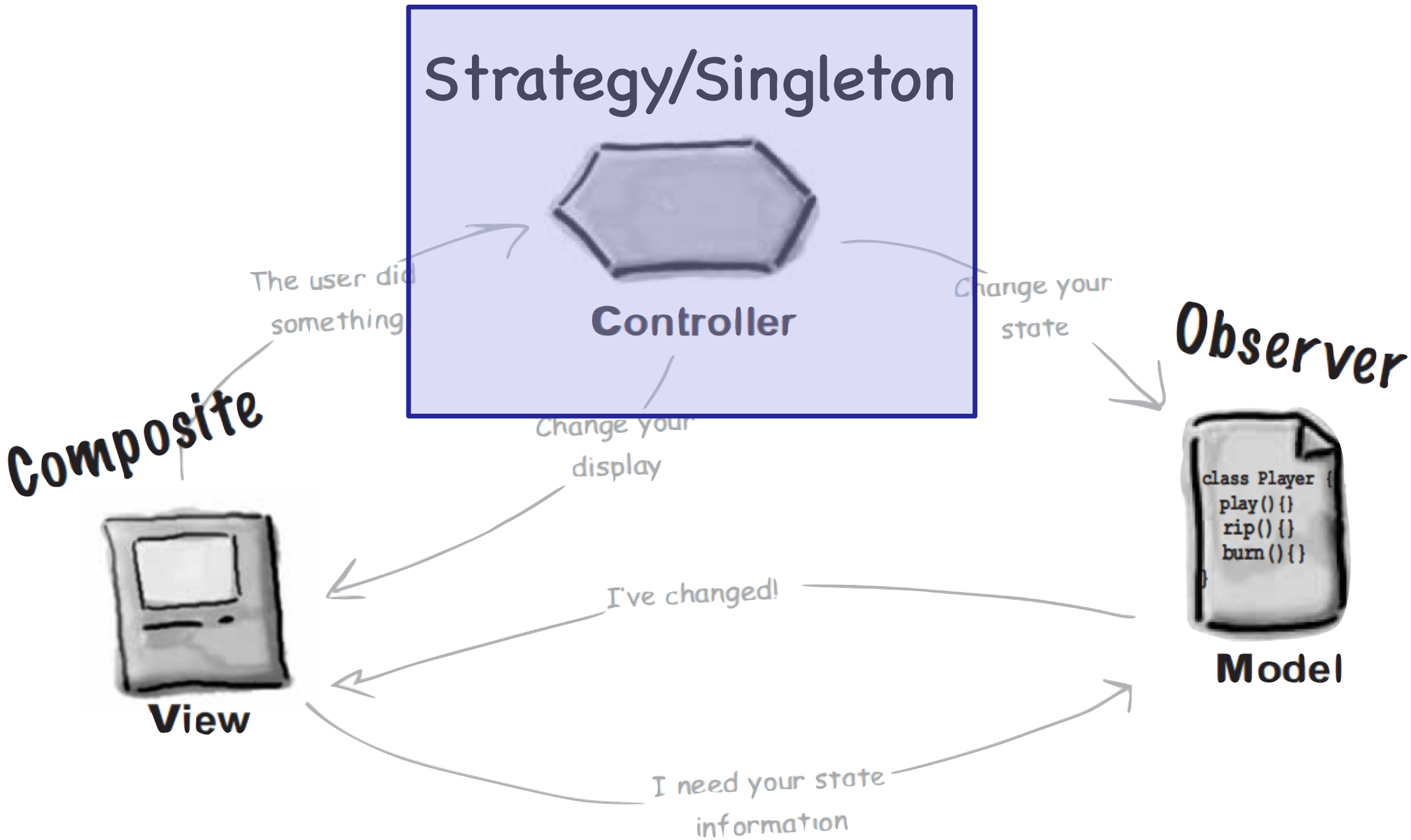
Model-Vista-Controlador

Aproximació general - Principals patrons



Model-Vista-Controlador

Aproximació general - Principals patrons



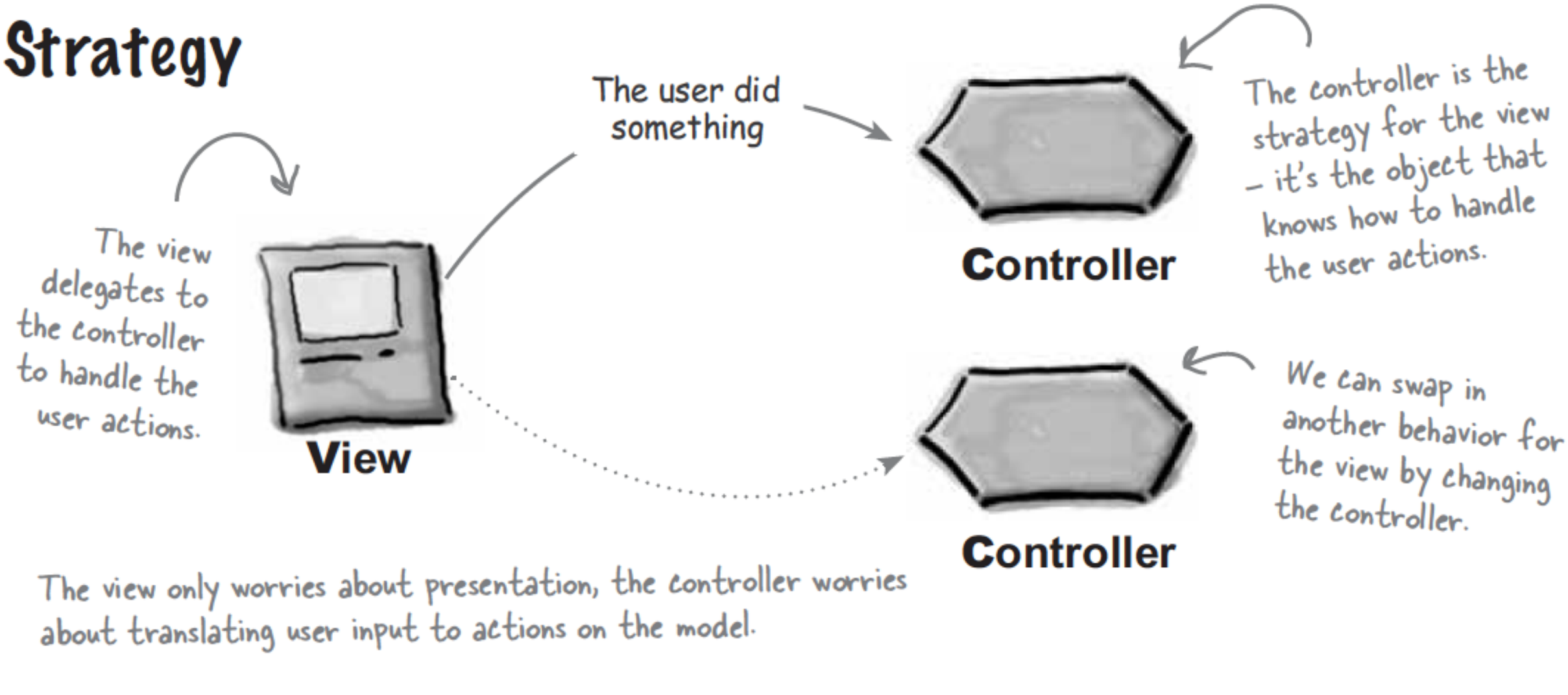
Controlador

Problemes	Possibles solucions
Un controlador modela un comportament concret de la Vista, que pot canviar en el temps	Usar Patró **Strategy** per a modelar els diferents comportaments de la Vista
Només es vol una instància de la classe Controlador	Patró **Singleton**
Hi ha un únic controller rebent tots els events del sistema i en són molts (baixa cohesió). Esdevé un oracle que tot ho sap o objecte "déu".	Utilitzar diferents **controladors** i diferents **Façanes** del model
Un controlador manté atributs i informació d'altres objectes o duplica la informació que es troba en altres llocs	Usar Patró **Expert** per delegar responsabilitats

Model-Vista-Controlador

Patrons en el Controlador:

- **Strategy**
- Singleton



Model-Vista-Controlador

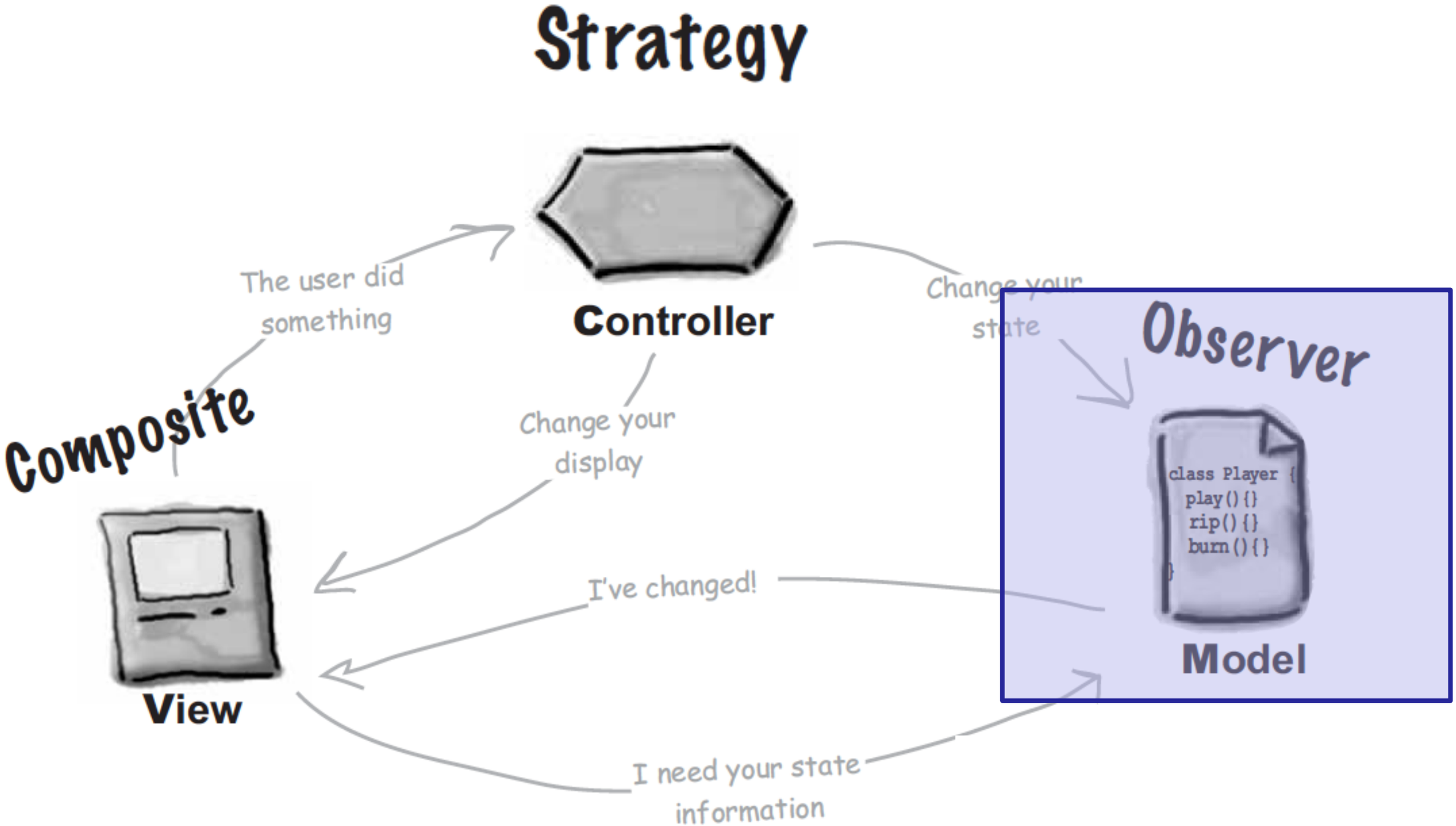
Patrons en el Controlador:

- Strategy
- **Singleton**

- El Controlador manega els estats de la Vista
- Cal assegurar que només hi ha una única sola instància del controlador i cal proporcionar un punt d'accés global a ella
- Interessa fer la instància de la classe només quan faci falta (*lazy instantiation*)

Model-Vista-Controlador

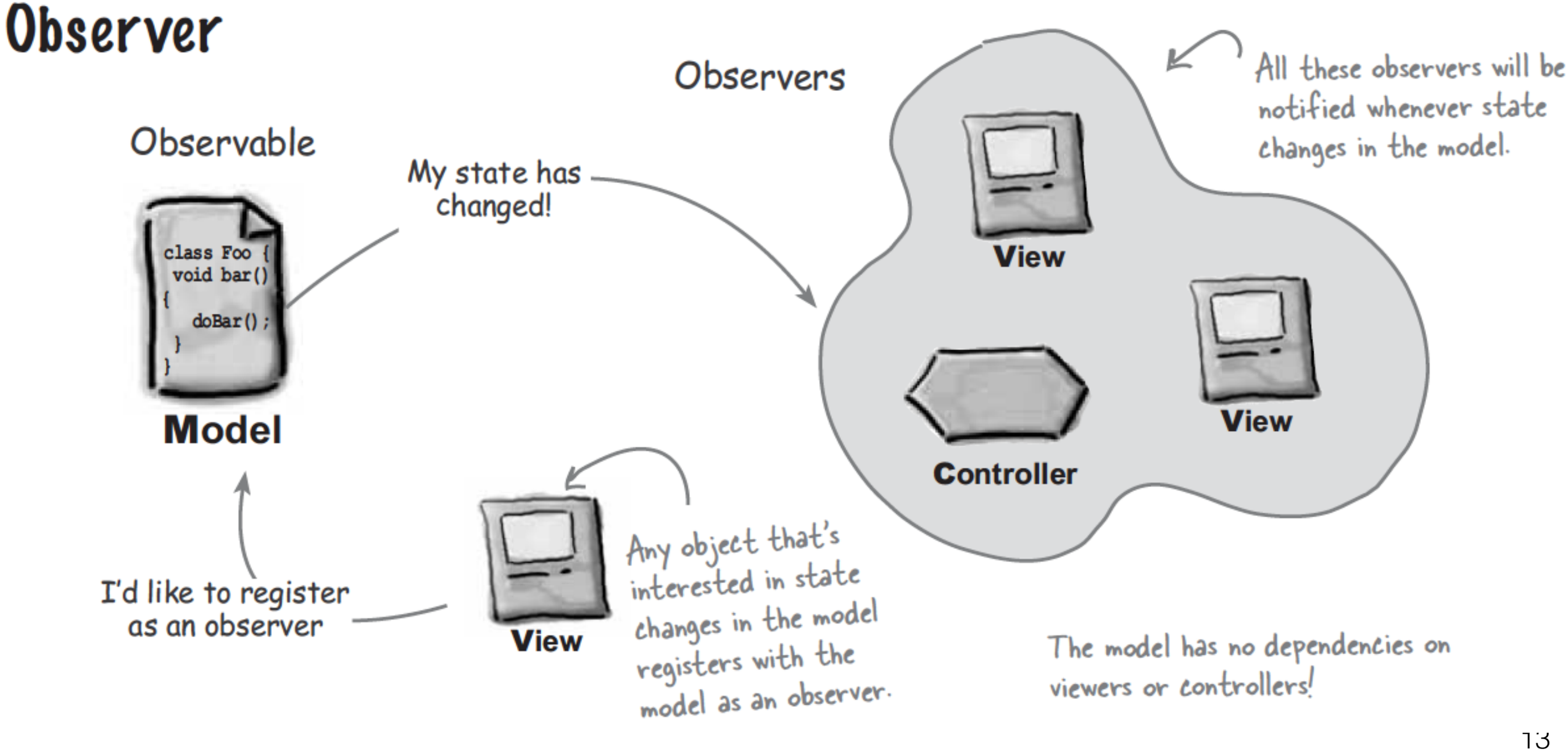
Aproximació general



Model-Vista-Controlador

Patró en el Model:

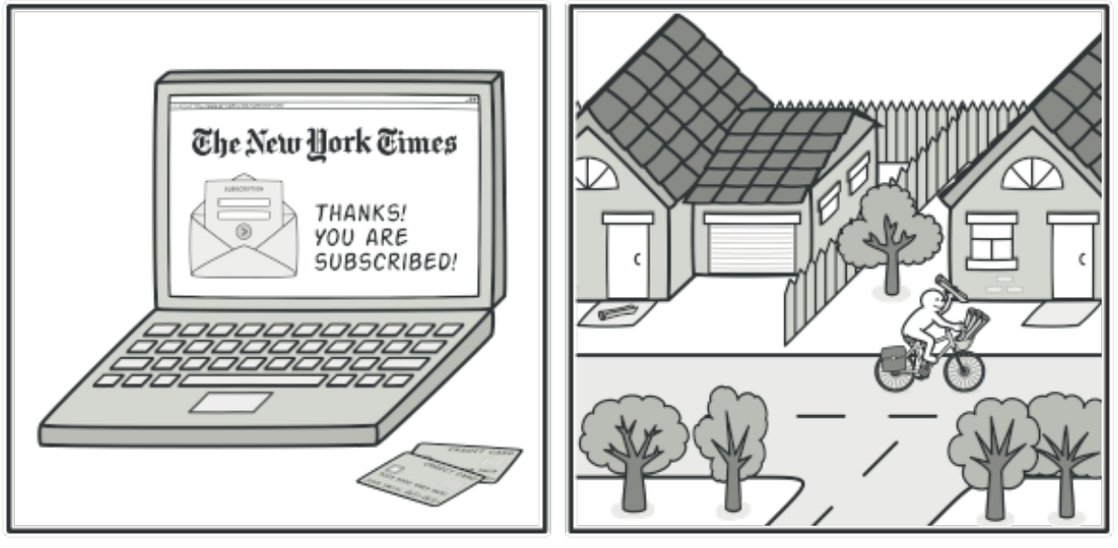
- **Observer**



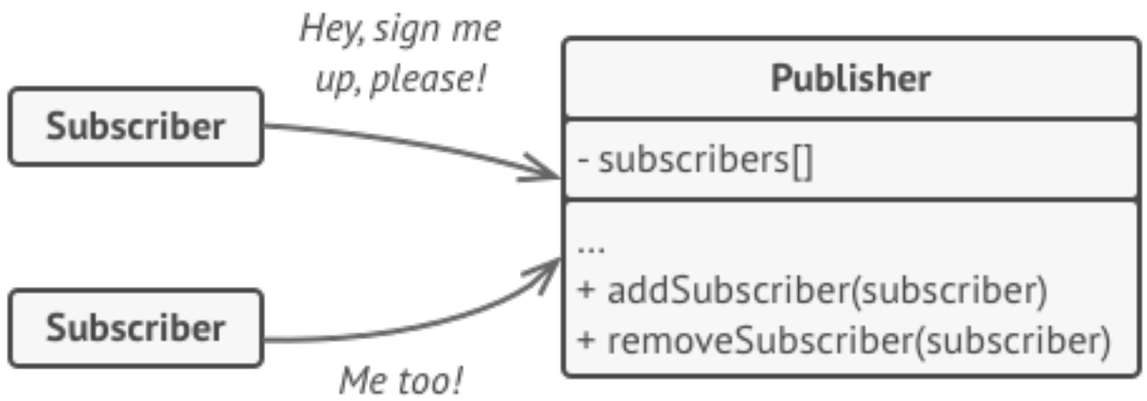
Model-Vista-Controlador

Patró en el Model:

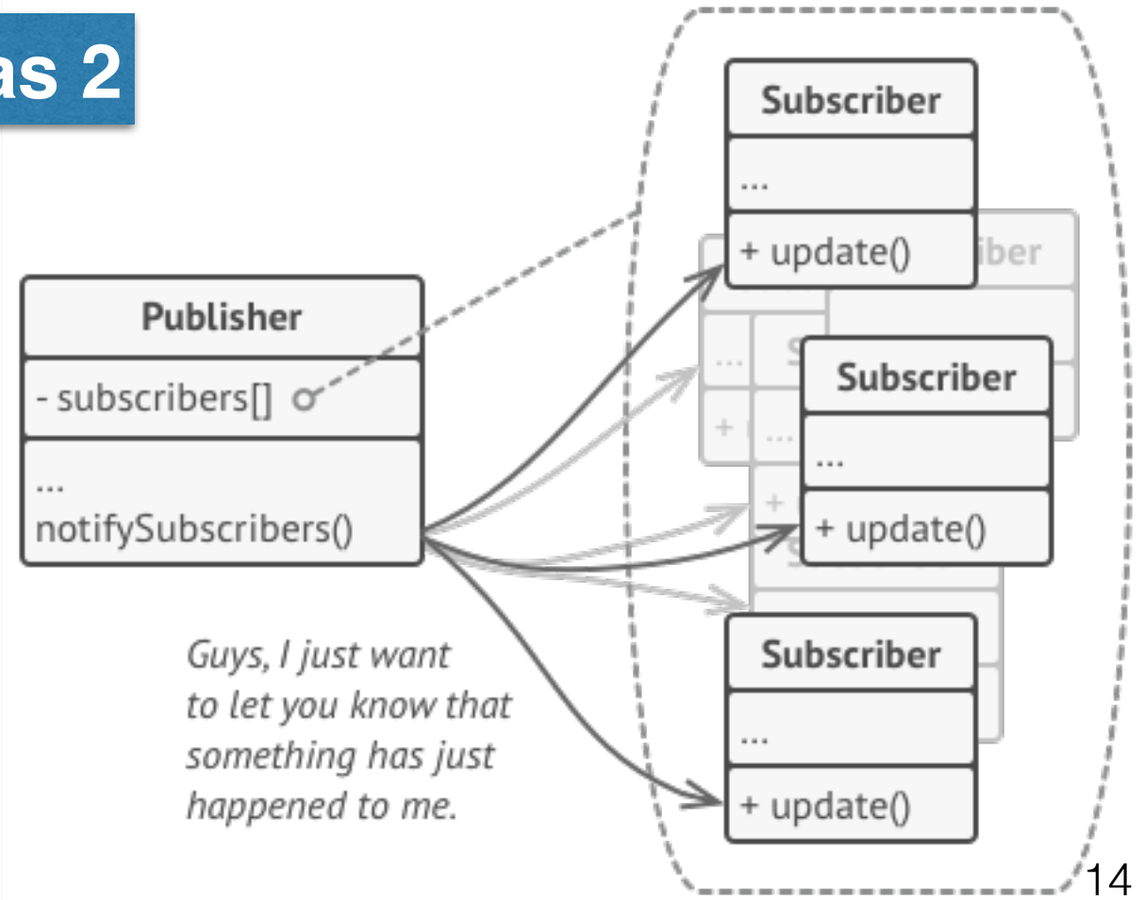
- **Observer/Observable**



Pas 1



Pas 2



Patró Observer

Nom del patró: **Observer**

Context:

Comportament i notificació de canvis en un objecte

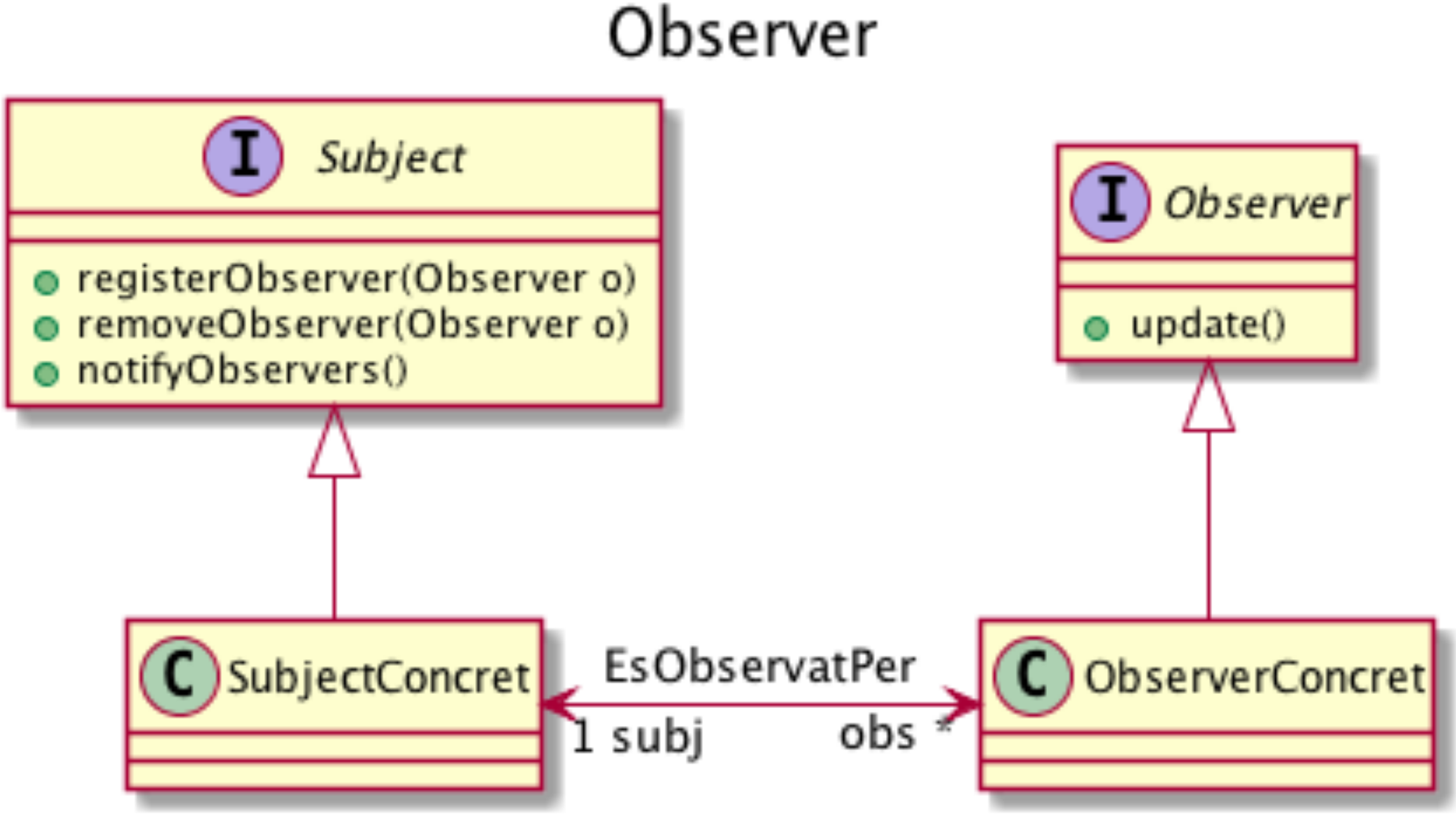
Problema:

- Un objecte vol saber els canvis que es produeixen en un altre objecte quan passen

Solució:

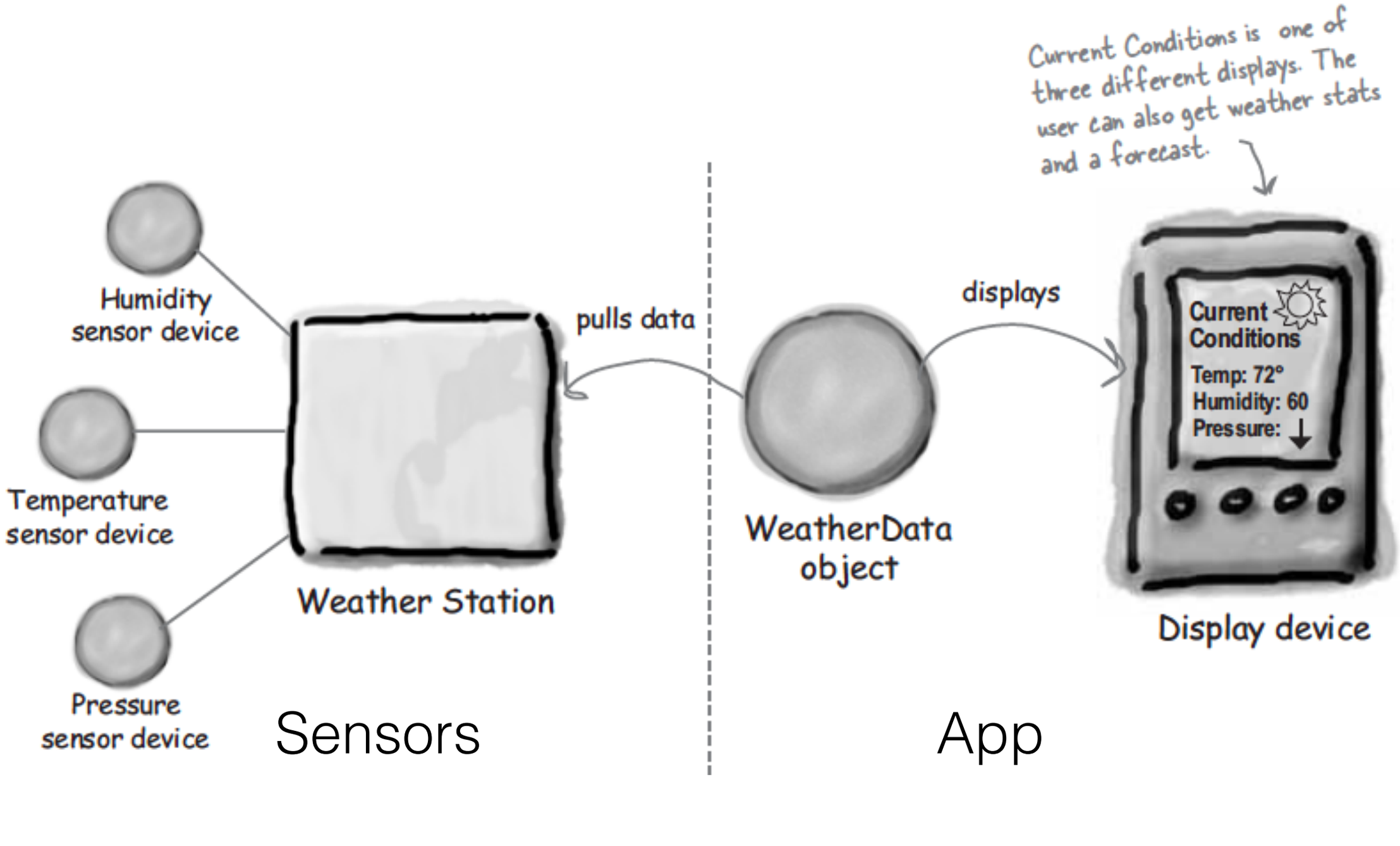
- L'objecte observat **A** (Observable) permet que altres objectes s'enregistrin per estar pendents dels seus canvis (Observadors)
- Quan es produeix un canvi a l'objecte **A**, **A** notifica a tots els objectes Observadors, els canvis que ha tingut
- També permet donar-se de baixa en l'enregistrament.

Patró Observer



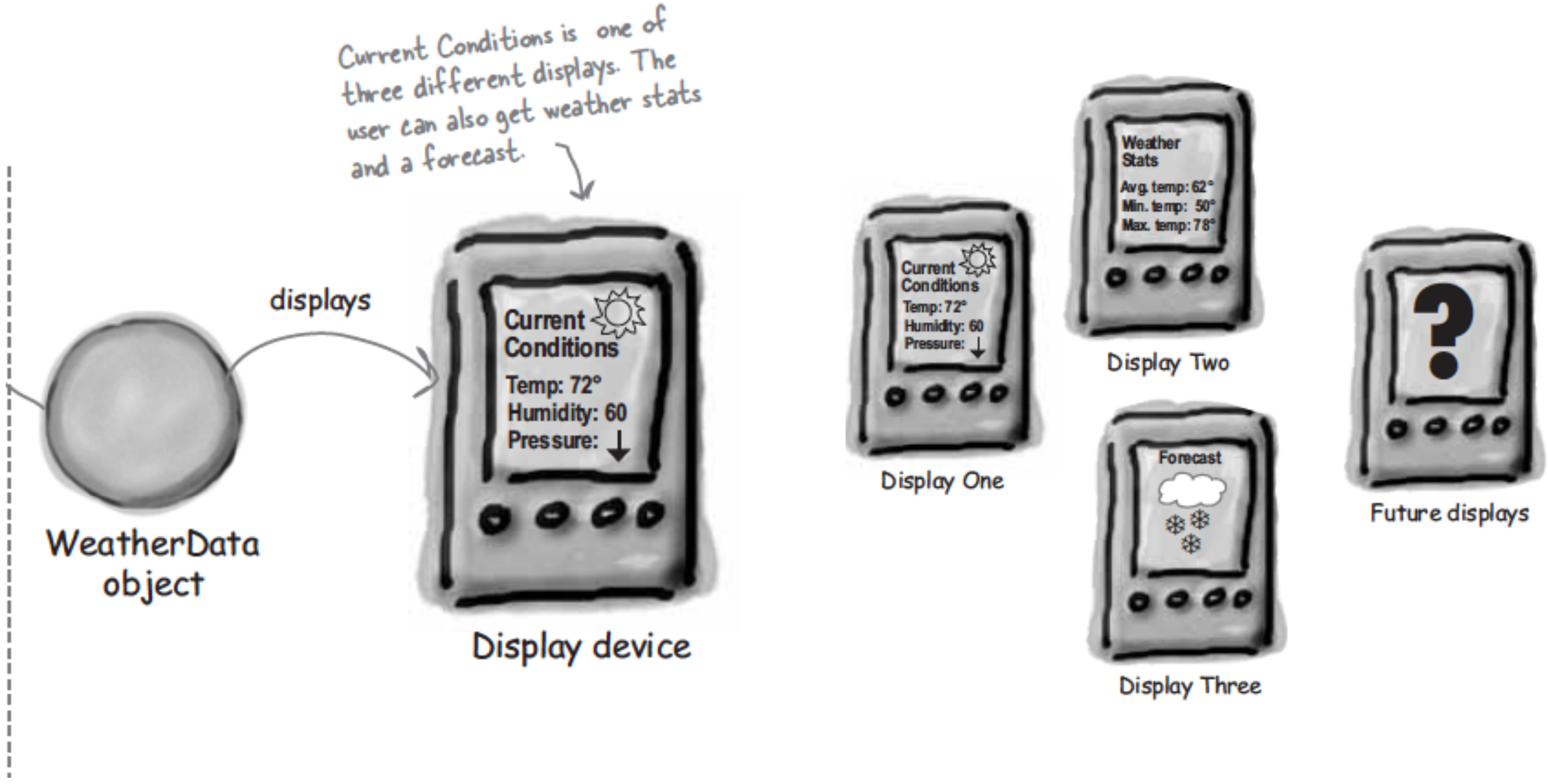
Model-Vista-Controlador

- **Observer**: Exemple de l'Estació meterològica



Model-Vista-Controlador

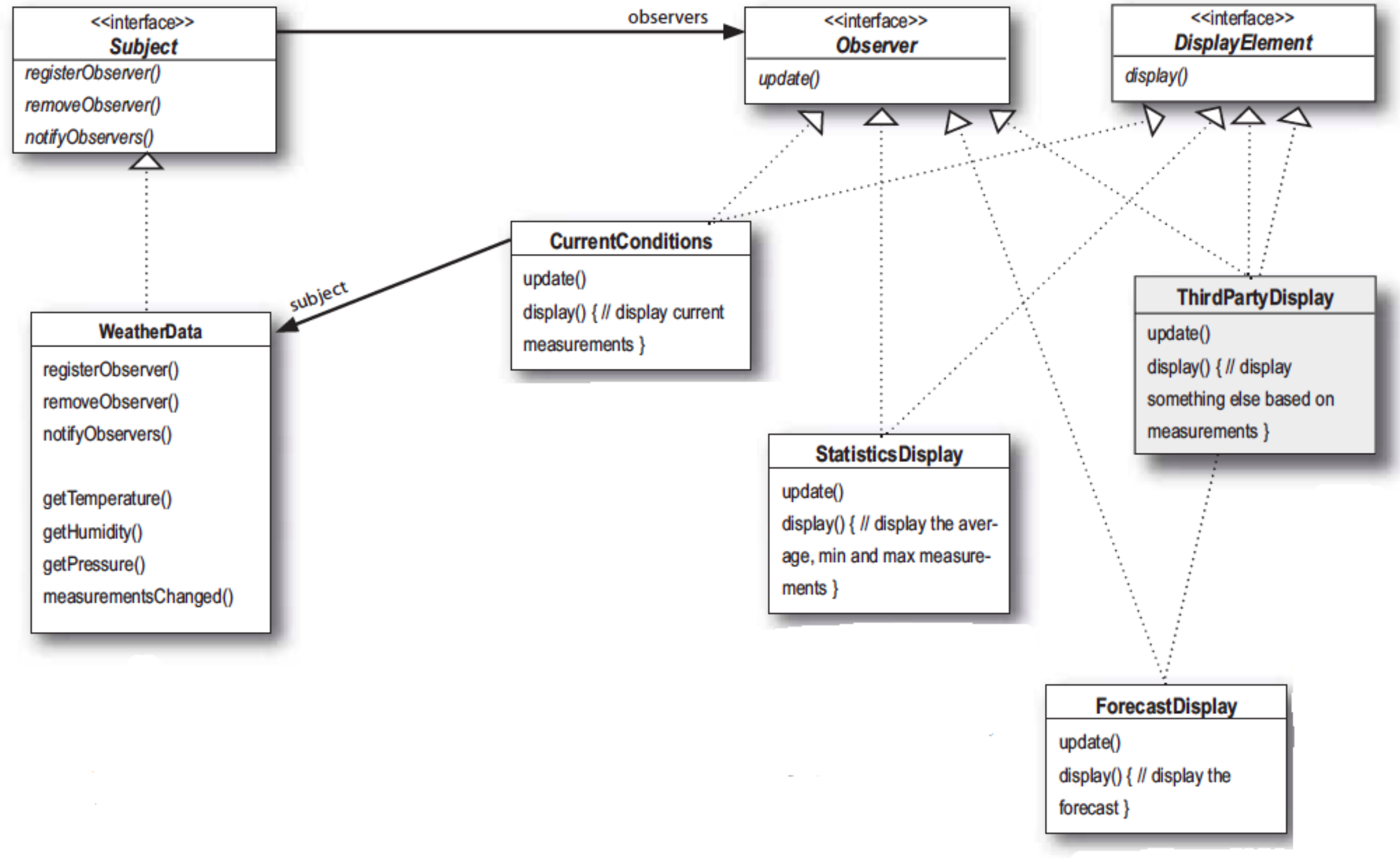
- **Observer**: Exemple de l'Estació meterològica



App

Model-Vista-Controlador

- **Observer**: Exemple de l'Estació meterològica



Model-Vista-Controlador

- **Observer**: Exemple de l'Estació meterològica

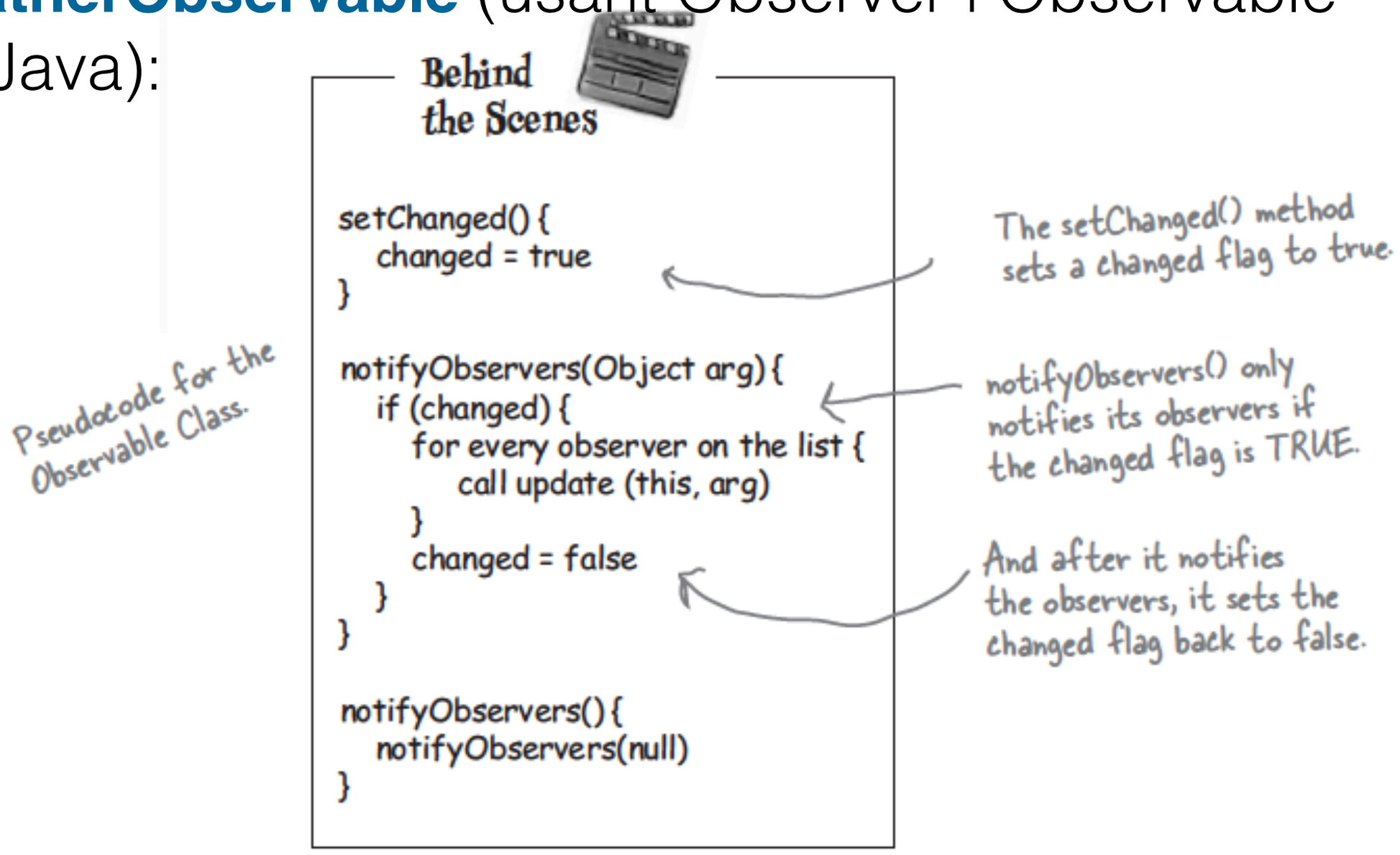
- **Exercici**:

1. Baixa el projecte Observer.zip del Campus Virtual
2. Analitza el codi del paquet **weather.** Vulnera algun principi?
3. Per què es guarda un atribut de tipus **Subject** en els observers?
4. Què en penses del mètode **update()**?
5. Analitza el codi del paquet **weatherObservable** (usa les classes Observable i Observer de Java)

Model-Vista-Controlador

Patró en el Model:

- **Observer**: Exemple de l'Estació meterològica
- **weatherObservable** (usant Observer i Observable de Java):



Patró Observer

Nom del patró: **Observer**

Context:

Comportament i notificació de canvis en un objecte

Pros:

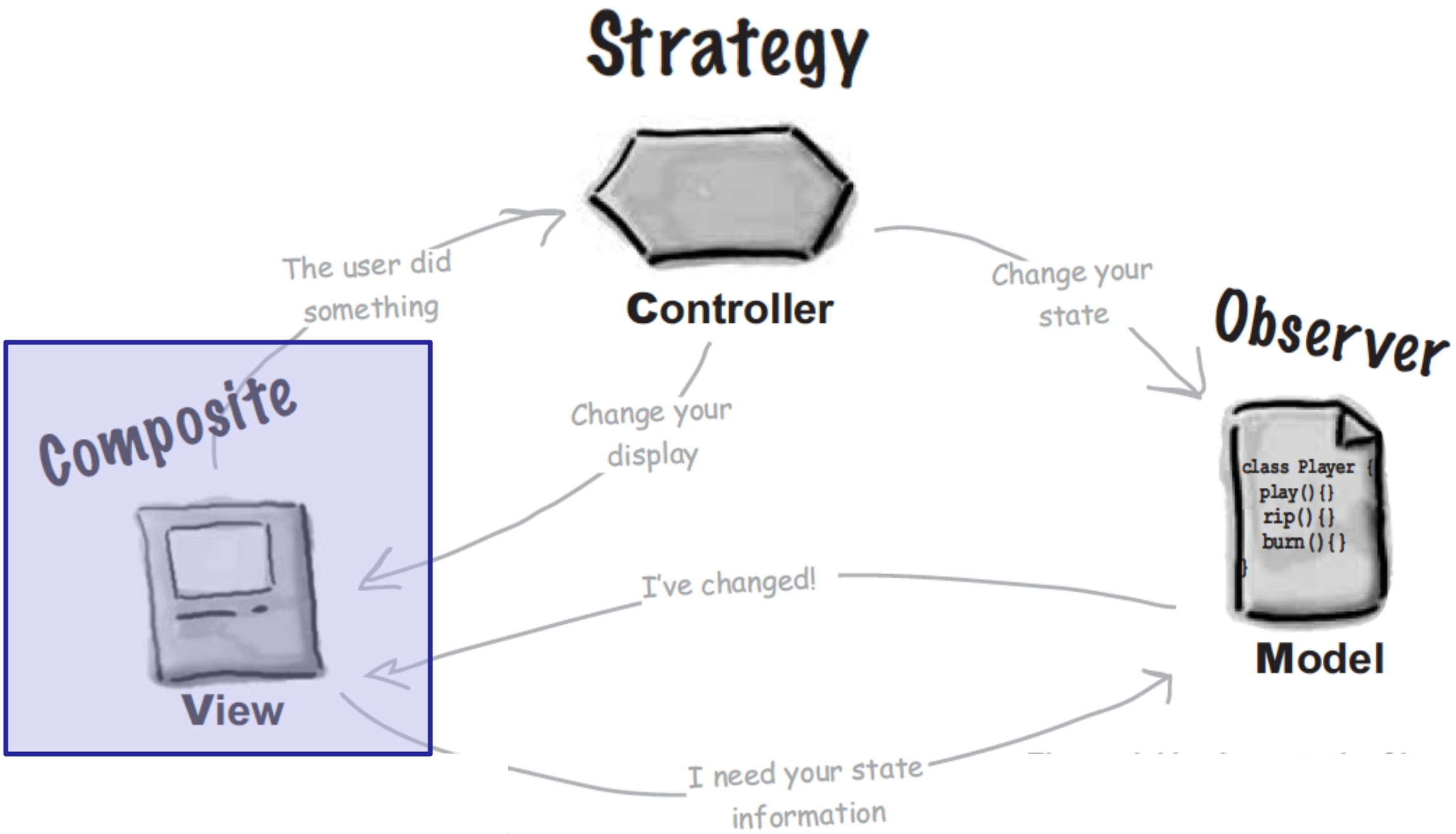
- Poc acoblament entre **Observer** i **Observat**
- Un mètode simple per notificar a tots els observers
- Existeix en Java java.util.Observable, java.util.Observer

Cons:

- Problemes de memòria a l'observat si els observers es mantenen sempre registrats (no criden a unregister)

Model-Vista-Controlador

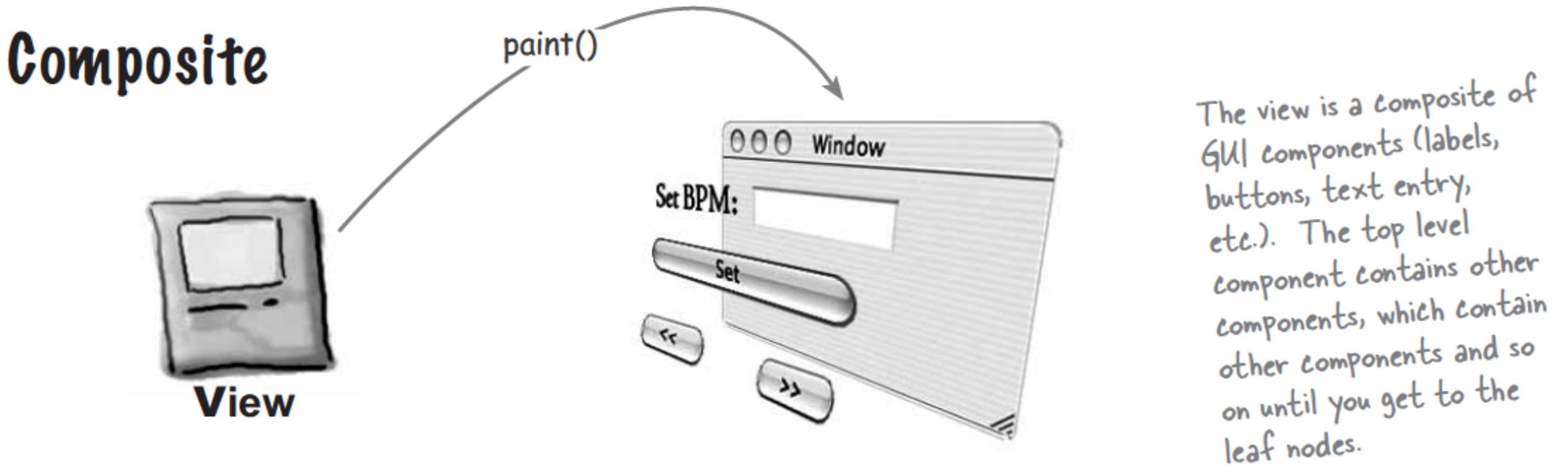
Aproximació general



Model-Vista-Controlador

Patró en la Vista:

- **Composite**



Patró Iterador

Nom del patró: **Iterador**

Context: Comportament

Problema:

- Aïllar el recorregut dels objectes que formen part d'un TOT

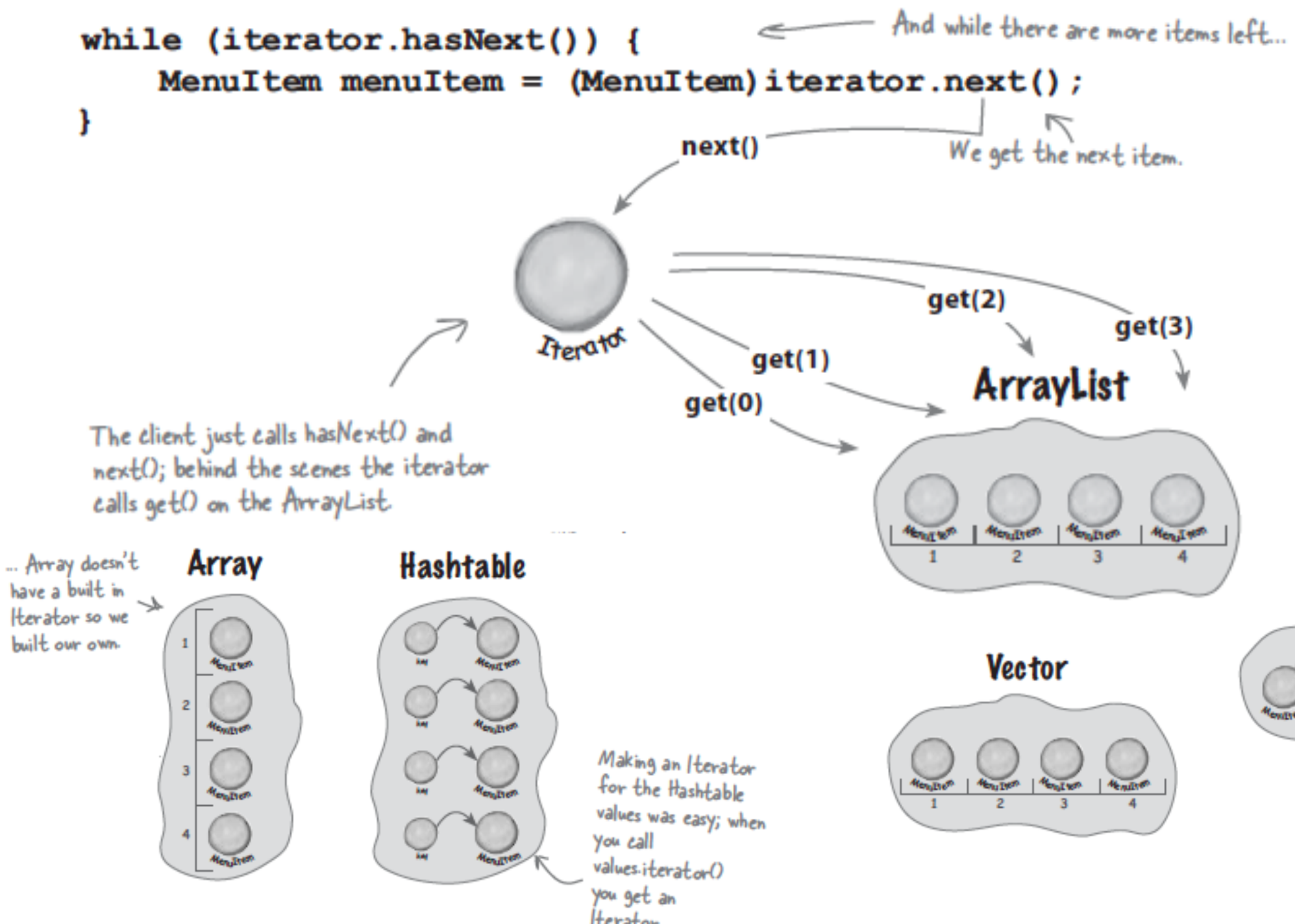
Solució:

- Defineix una interfície comuna per a col·leccions d'objectes per independitzar de l'estructura de dades que implementa la col·lecció.
- La interfície proveeix d'un conjunt de mètodes per a recórrer els elements d'una col·lecció.
- Java proporciona la interfície java.util.Iterator (amb els mètodes hasNext(), next() i remove())
- tExisteixen tipus a Java com Array o HashMap que no proporcionen la implementació d'Iterator i el programador ha de implementar els seus propis mètodes.

Patró Iterador

```
Iterator iterator

while (iterator.hasNext()) {
    MenuItem menuItem = (MenuItem)iterator.next();
}
```



Patró Composite

Nom del patró: **Composite**

Context: Creació d'objectes

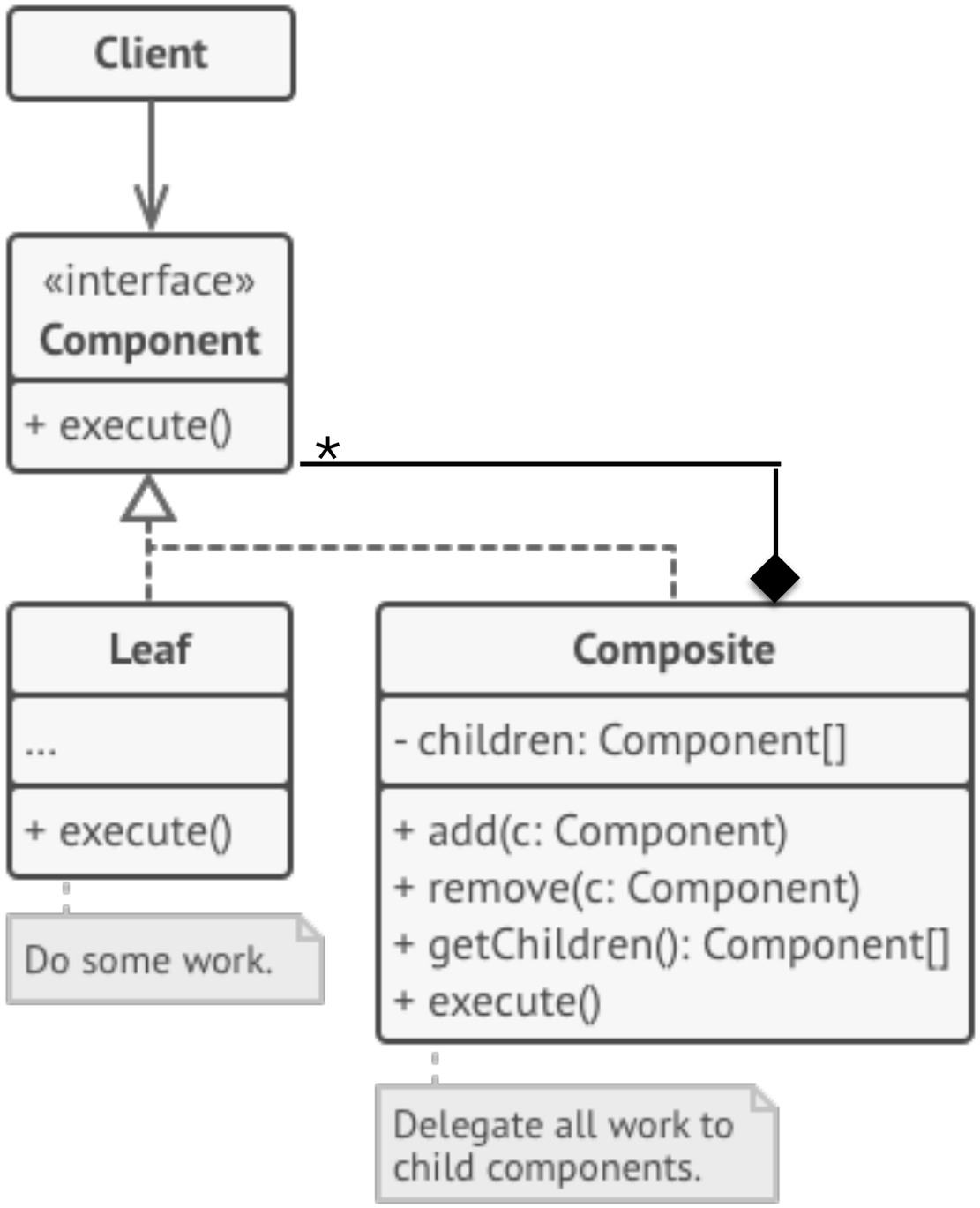
Problema:

- Composar objectes en jerarquies **TOT-PART** i permetre als clients tractar objectes simples i compostos de manera uniforme

Solució:

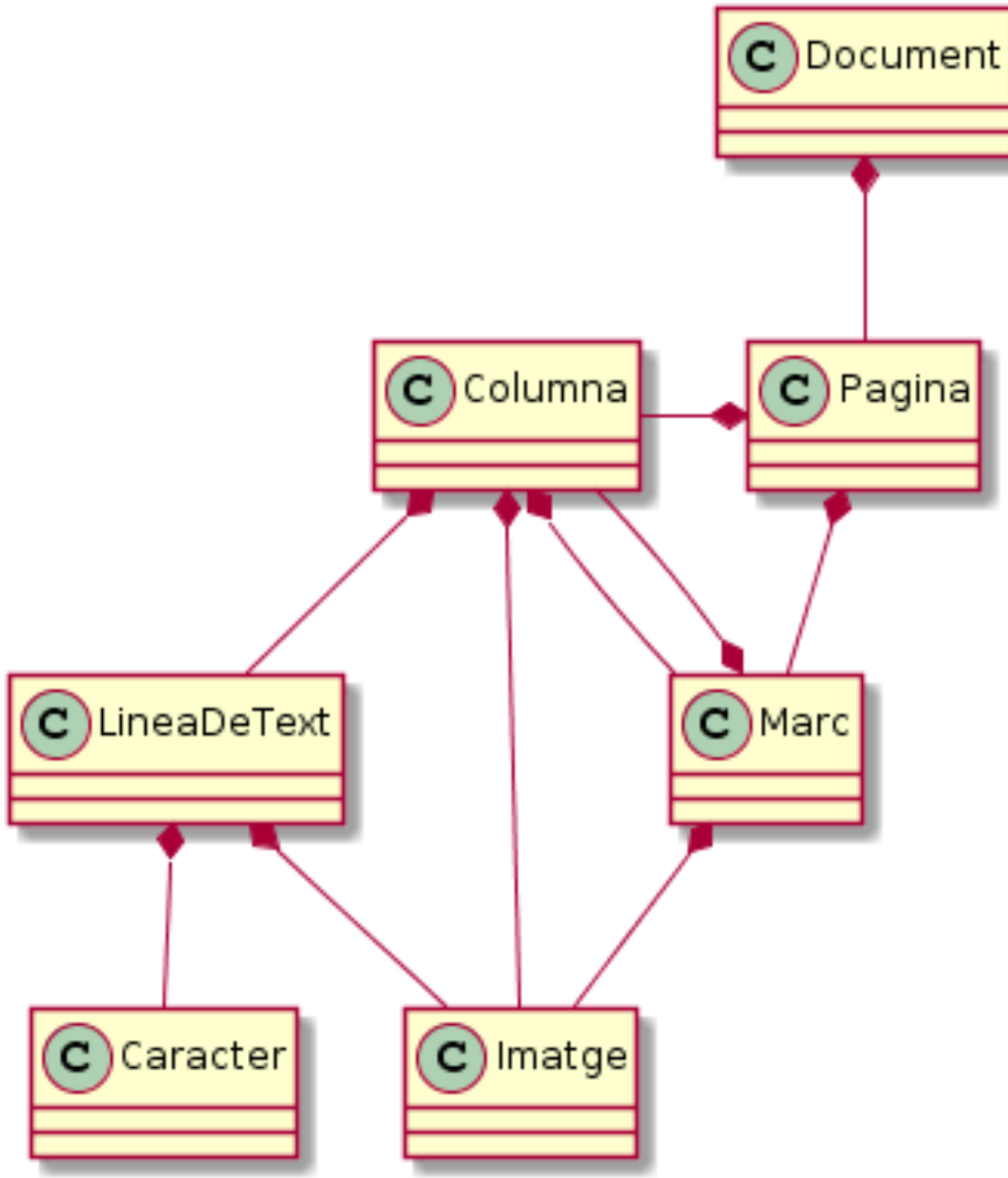
- Defineix jerarquies de classes que tenen objectes primitius i objectes compostos a la vegada que els compostos estan formats per objectes primitius o altres objectes compostos.
- Utilització de l'herència per modelar els diferents tipus d'objectes (simples i compostos)
- Utilització de la composició per modelar les relacions TOT-PART dels compostos

Patró Composite



- **`Component`**: declara una classe abstracta per la composició d'objectes
- **`ElementSimple (Leaf)`** representa els objectes de la composició que no tenen fills i implementa les seves operacions
- **`ElementCompost (Composite)`**: implementa les operacions per als components amb fills i emmagatzema els fills
- **`Client`**: utilitza objectes de la composició mitjanánt la interfície de **`Component`**

Patró Composite



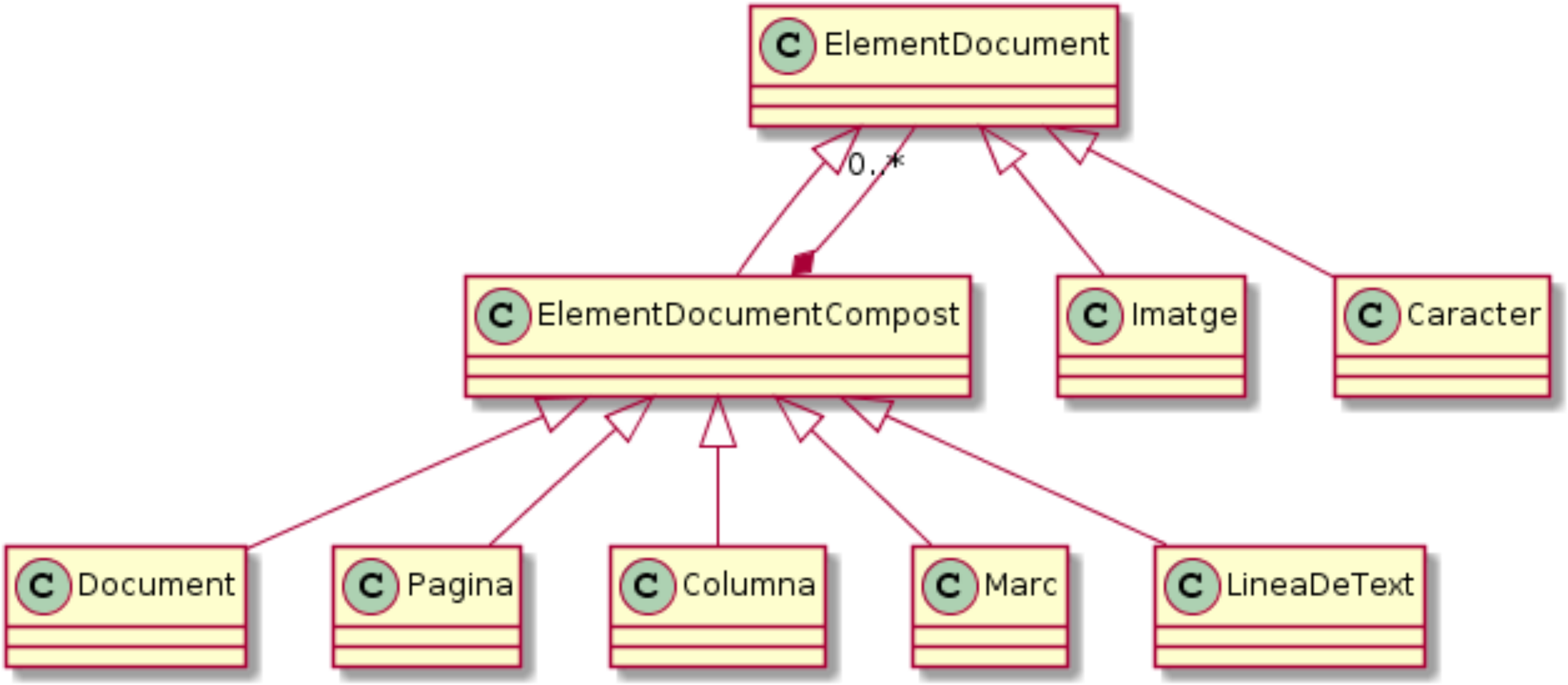
Exemple:

- Un document està format per vàries pàgines, les quals estan formades per columnes que contenen línies de text, formades per caràcters

- Les columnes i pàgines poden contenir marcs. Els marcs poden contenir columnes

- Les columnes, marcs i línies de text poden contenir imatges

Patró Composite



- Un document està format per vàries pàgines, les quals estan formades per columnes que contenen línies de text, formades per caràcters
- Les columnes i pàgines poden contenir marcs. Els marcs poden contenir columnes
- Les columnes, marcs i línies de text poden contenir imatges

Patró Composite

Nom del patró: Composite

Context: Creació d'objectes

Pros:

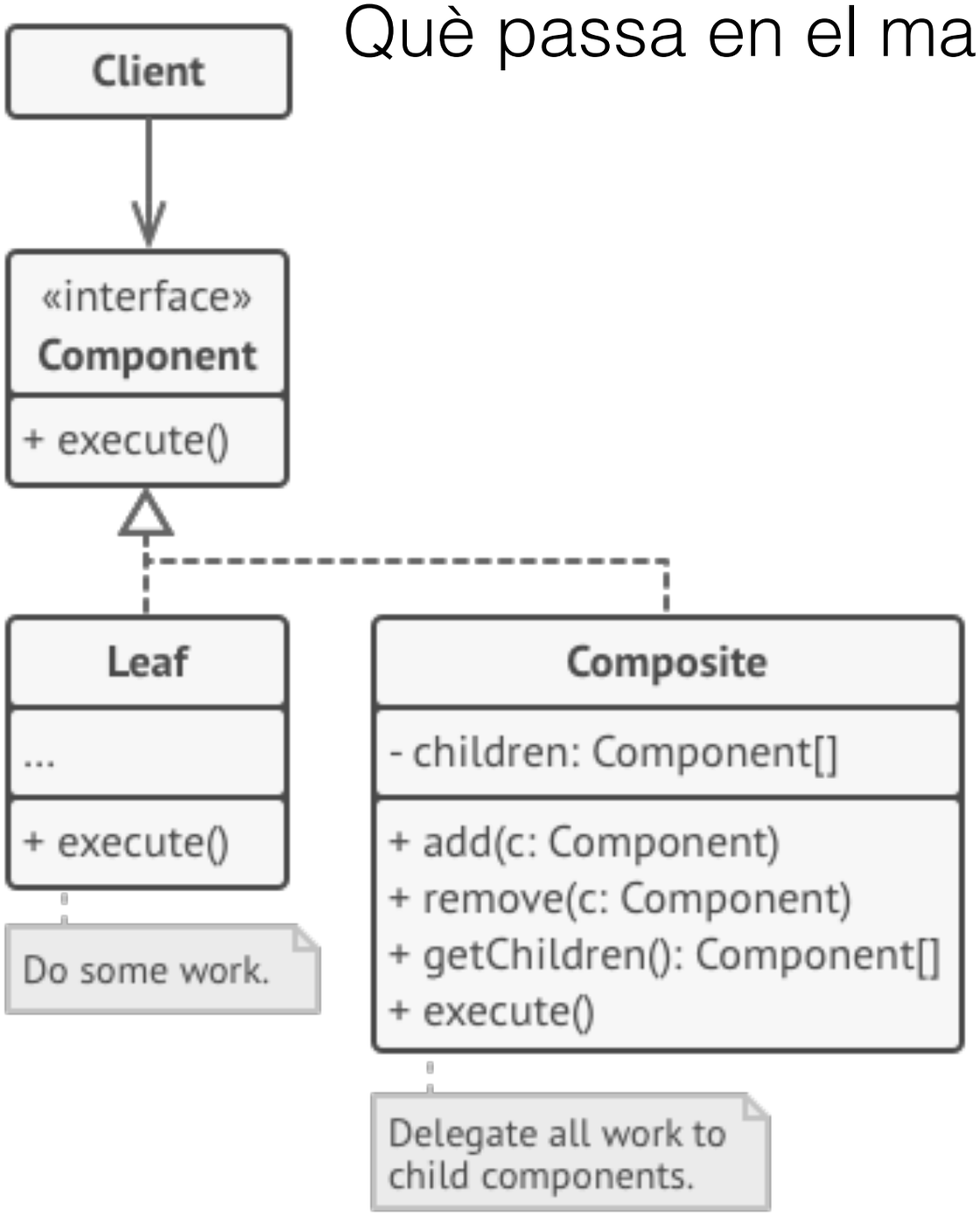
- Permetre el tractament uniforme d'objectes simples i complexes, així com les seves composicions recursives
- Simplifica el codi dels clients, que només usen una interfície
- Facilita afegir nous components sense afectar als clients
- Exemples en Java: JFrame, JPanel, JComponents

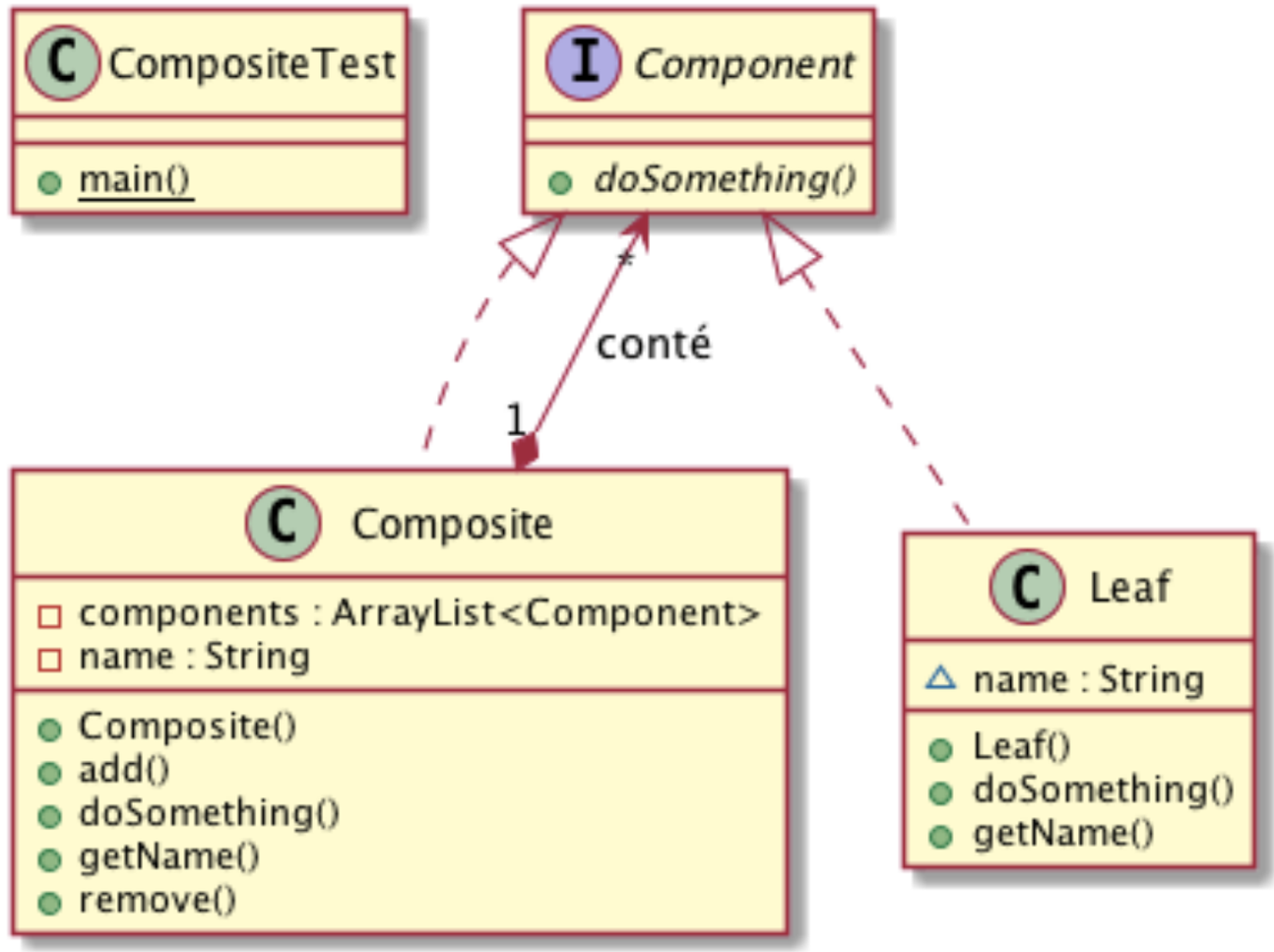
Cons:

- És difícil restringir els tipus dels fills
- Les operacions de gestió dels fills en els objectes compostos s'han de codificar les comprovacions en temps d'execució.

Exercici

Veure la carpeta patterBasic del projecte del campus:
Què passa en el main?

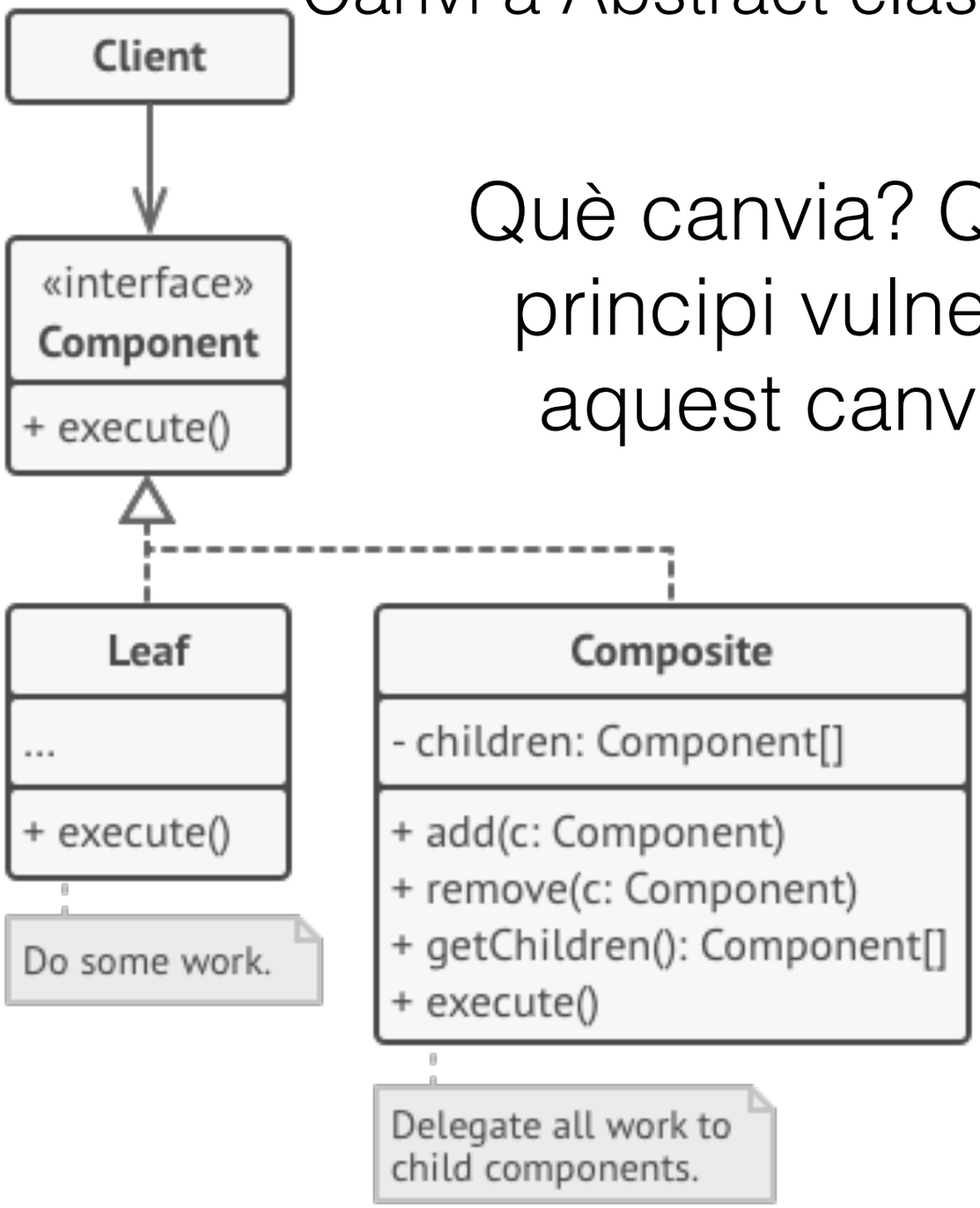


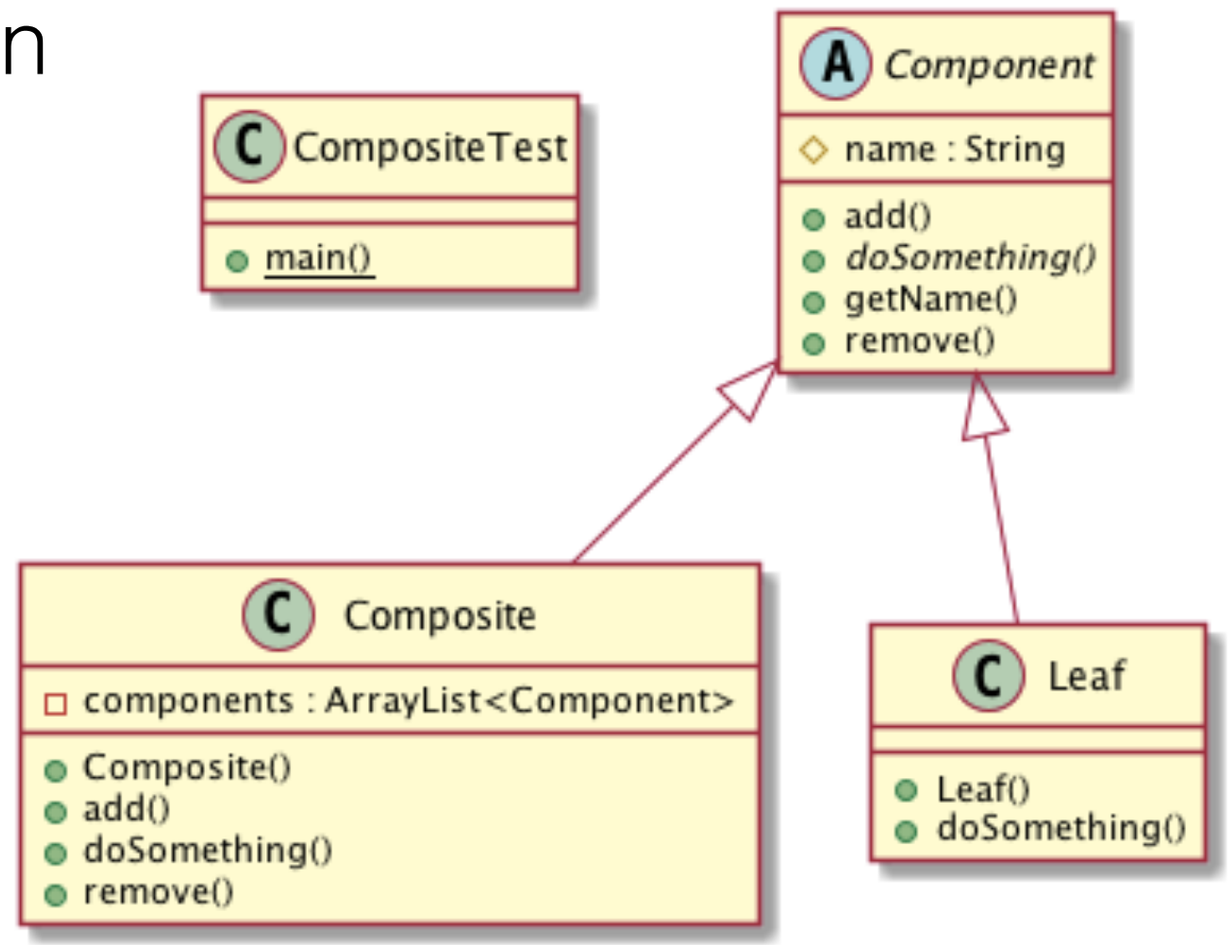


Exercici

Veure la carpeta pattern del projecte del campus:
Canvi a Abstract class enlloc d'interfície

Què canvia? Quin principi vulnera aquest canvi?

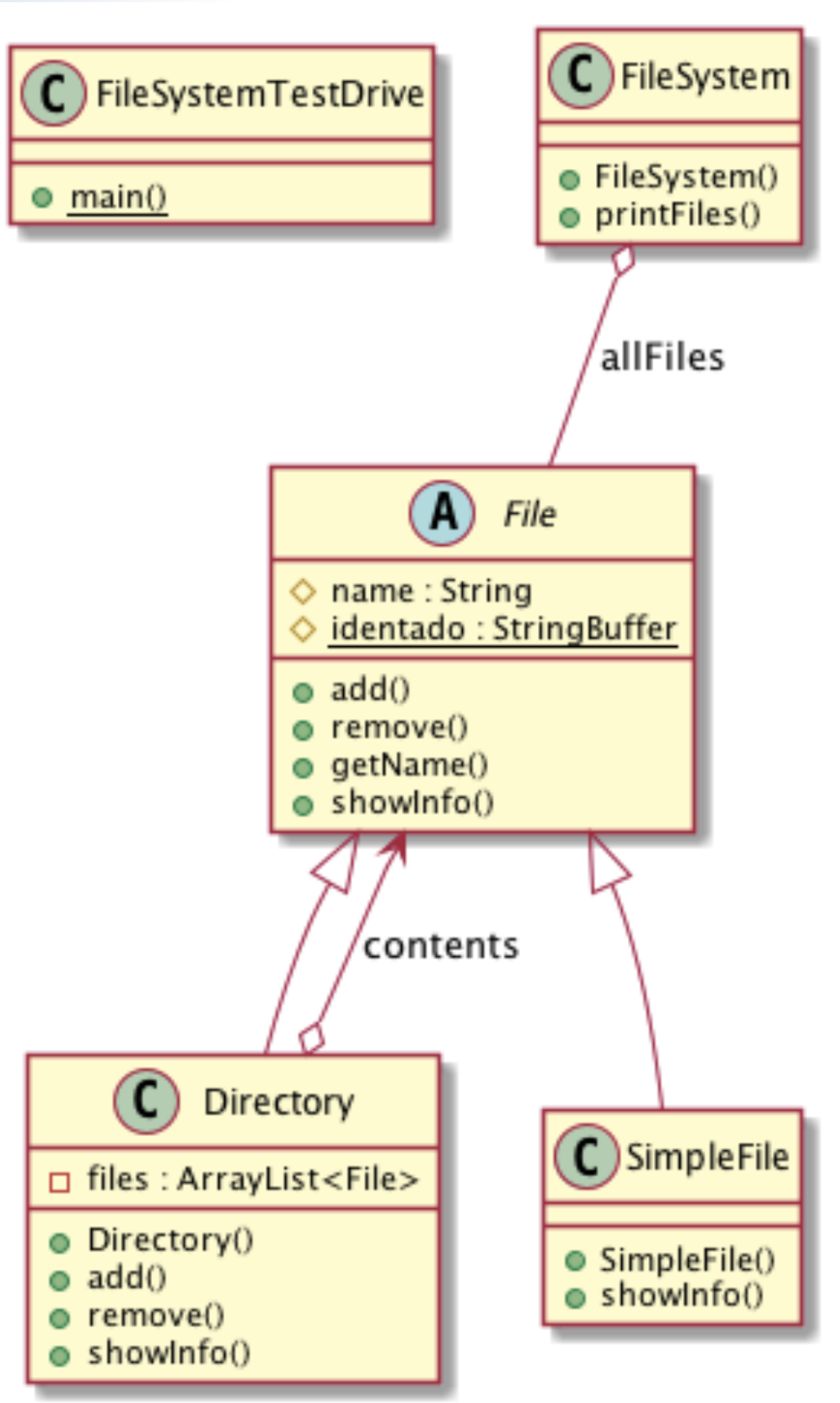




Exercicis

1. En el projecte Composite del campus, explora l'aplicació del patró Composite que modela un sistema de Fitxers:

- quins principis vulnera?
- identifica quina classe és la Component del patró
- identifica quina classe és el Composite del patró
- identifica quina classe és la Leaf del patró



Exercicis

2. En el mateix Projecte Composite: Completa l'exercici que modela el llistat d'un menú d'un restaurant. Es vol obtenir el llistat següent:

```
* Menus
        * Comidas
            * Plato Fuerte
                # Crispy Chicken,100.89
            * Postres
                # Apple Pie,15.59
                # Cheesecake,19.99
    * Cenas
            # Hotdogs,6.05
            # Spaghetti (v),30.89
```