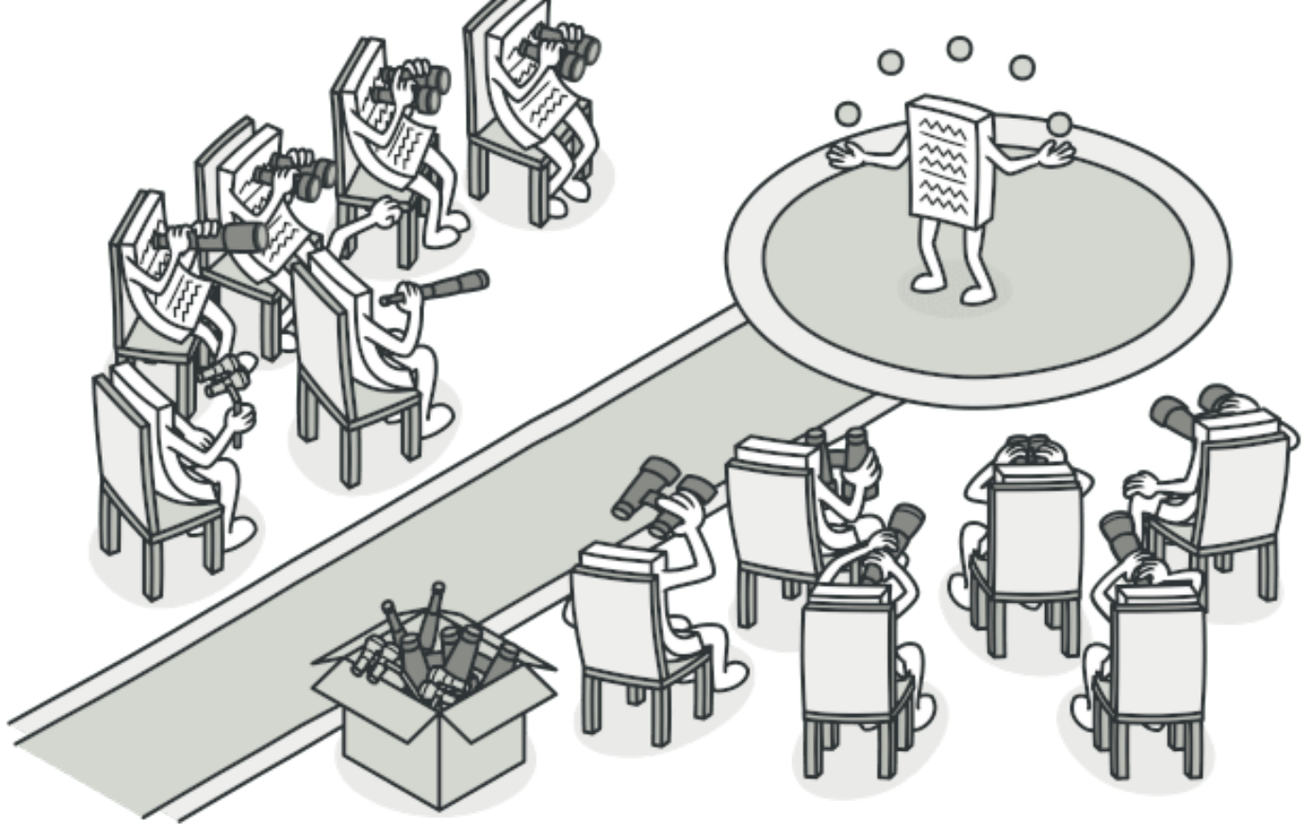


Tema 3: Disseny

Anna Puig

Enginyeria Informàtica

Facultat de Matemàtiques i Informàtica,

Universitat de Barcelona

Curs 2020/20201

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

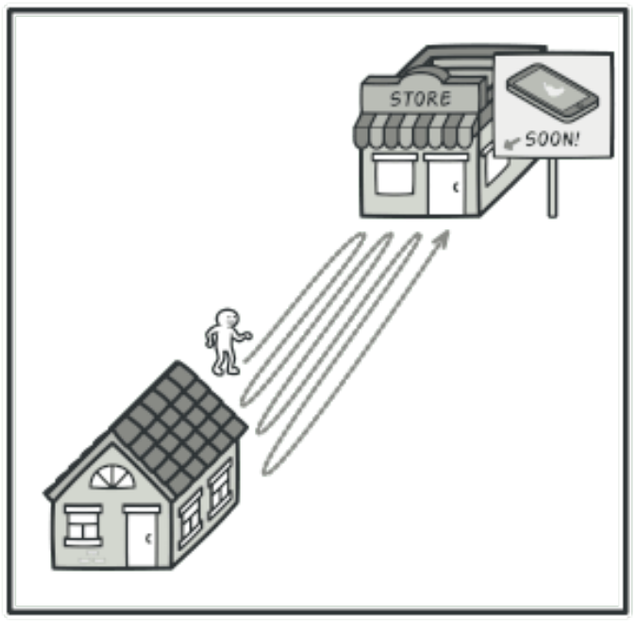
-

3.4. Patrons de disseny

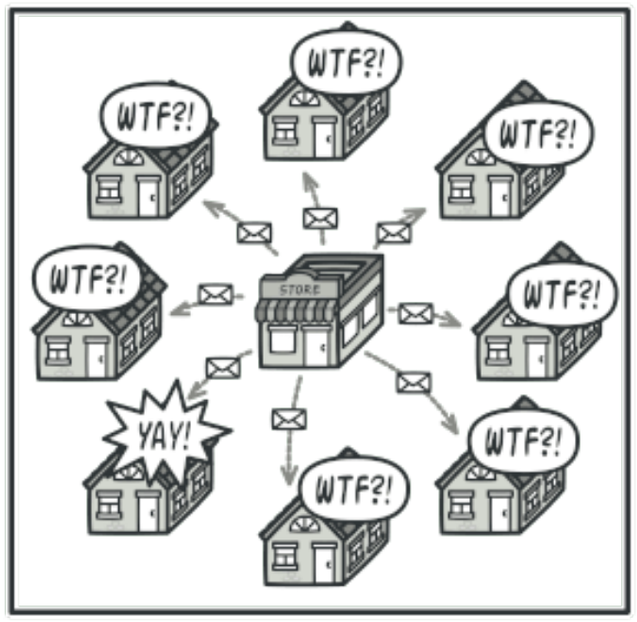
Propòsit → **Àmbit ↓**	**CREACIÓ**	**ESTRUCTURA**	**COMPORTAMENT**
CLASSE	• Factory method	• class Adapter	• Interpreter • Template method
OBJECTE	• Abstract Factory • Builder • Prototype • Singleton • Object pool	• Object Adapter • Bridge • Composite • Decorator • Facade • Flyweight • Proxy	• Chain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

Exemple patró Observer

- Tenim dos tipus d'objectes: Comprador i Botiga. Suposem que el Comprador està interessat en una marca concreta
- Com solucionem la notificació de les novetats que arriben a la Botiga?



(1) Comprador va preguntant



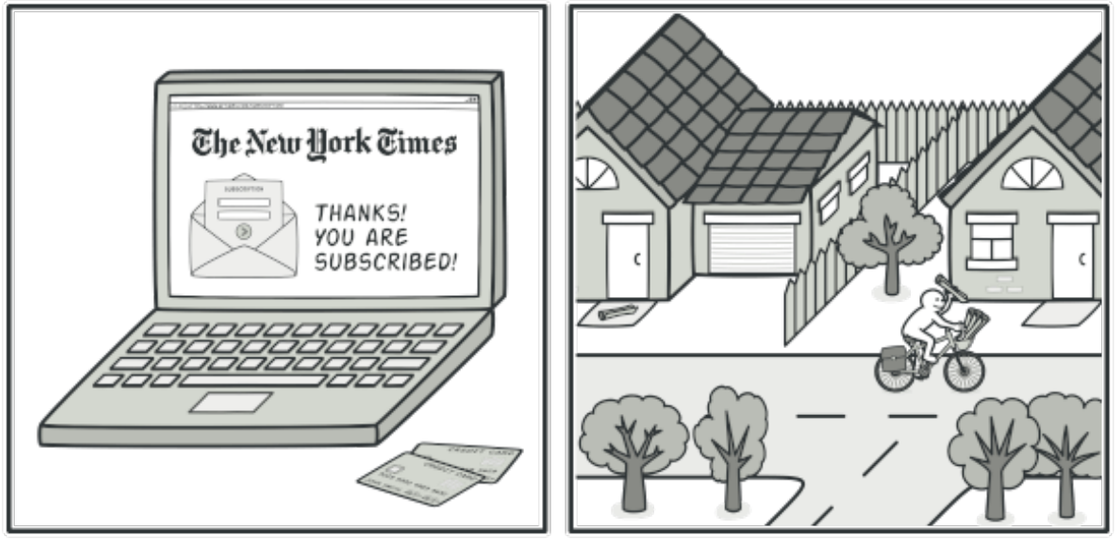
(2) Enviament massiu

(3) Subscripció

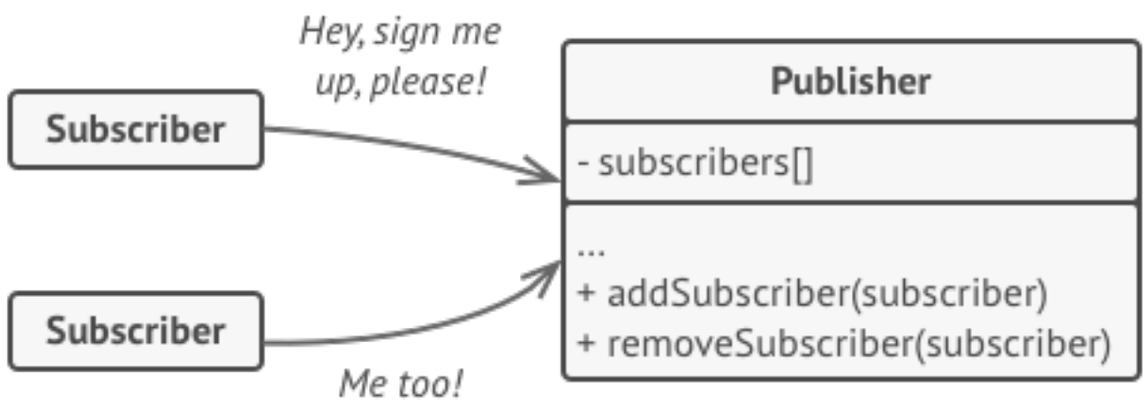
Patró Observer

Patró:

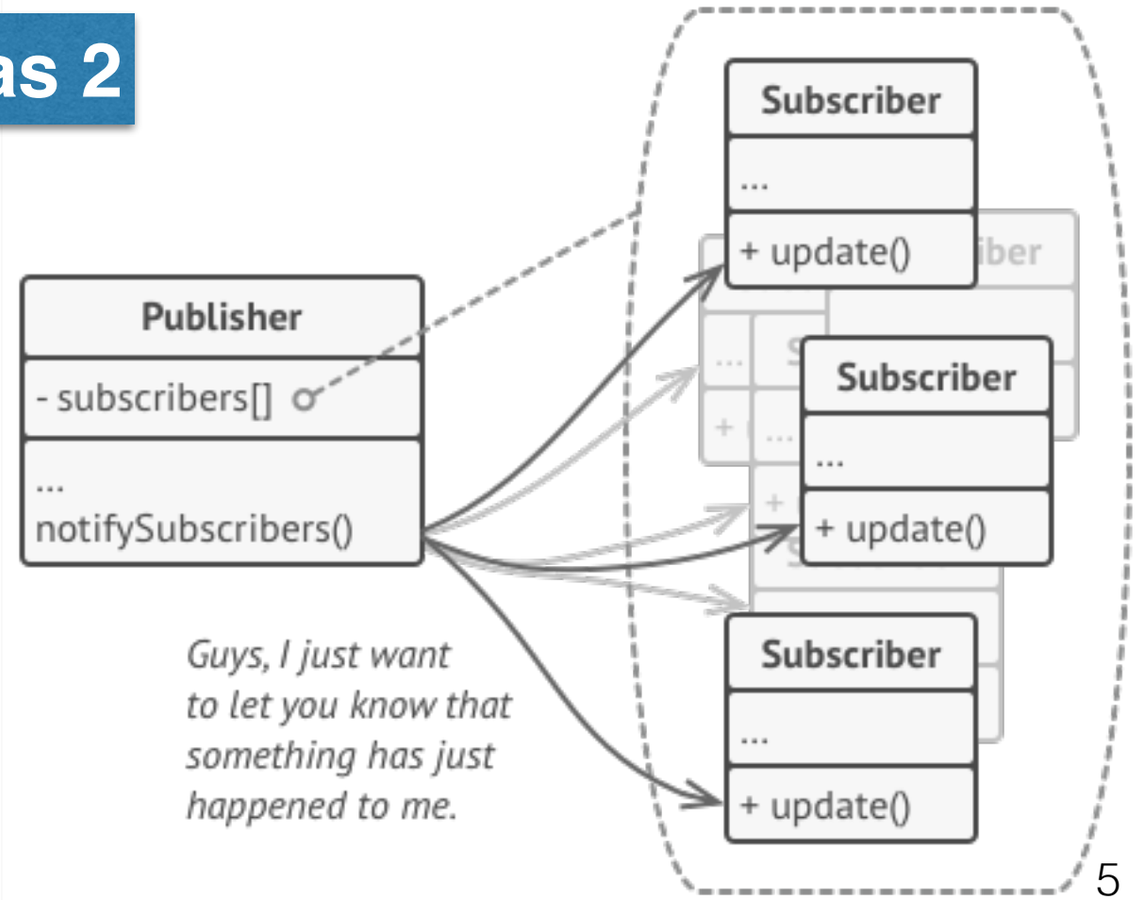
- **Observer/ Observable**



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

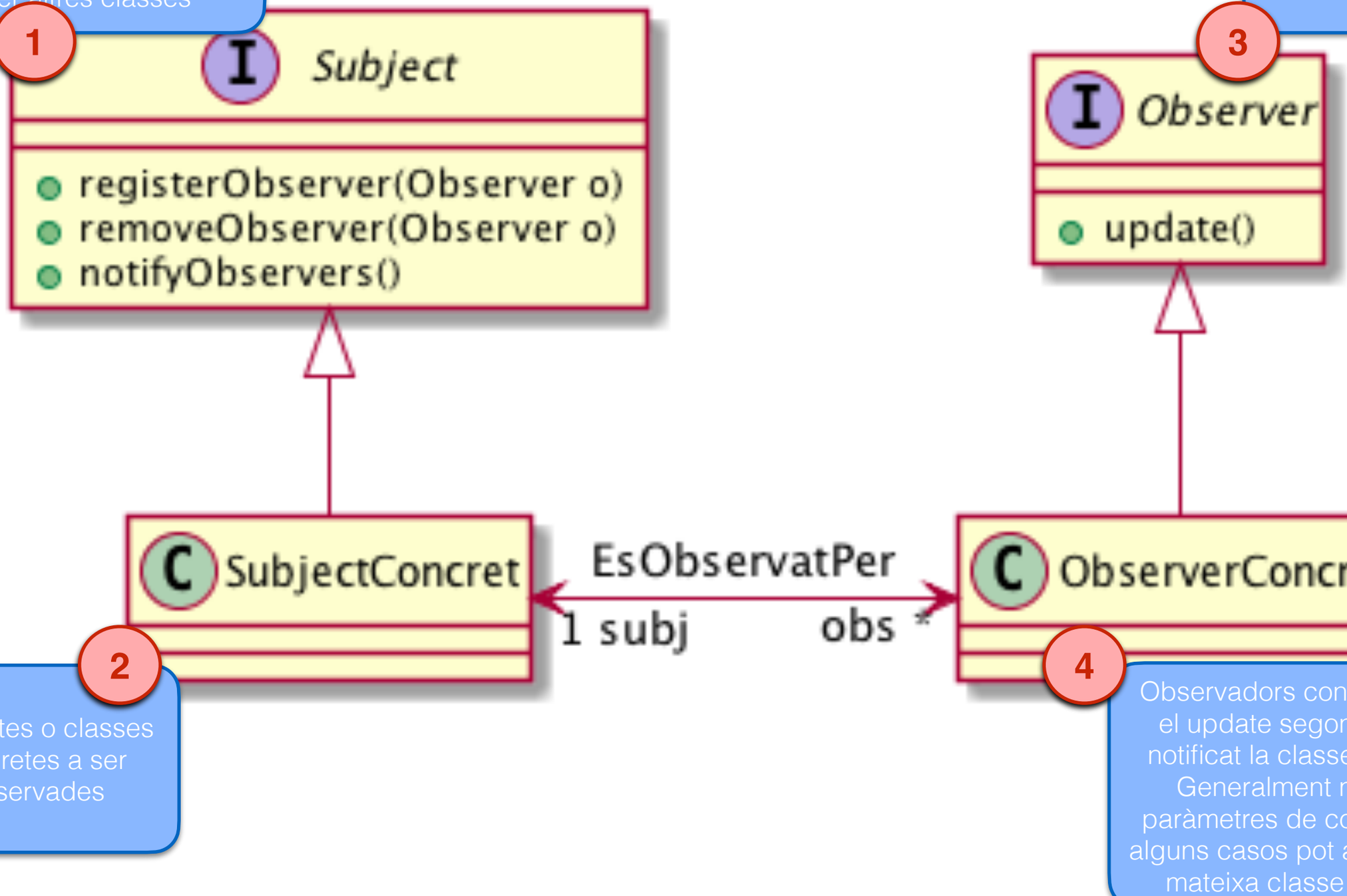
Solució:

- 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

Interfície que declara com subscriure's a un objecte que serà observat (o subjecte) Els canvis en el Subjecte són interessants per altres classes

1



Interfície que declara com serà el tipus de notificació. en general, és una interfície amb només el mètode update()

3

2

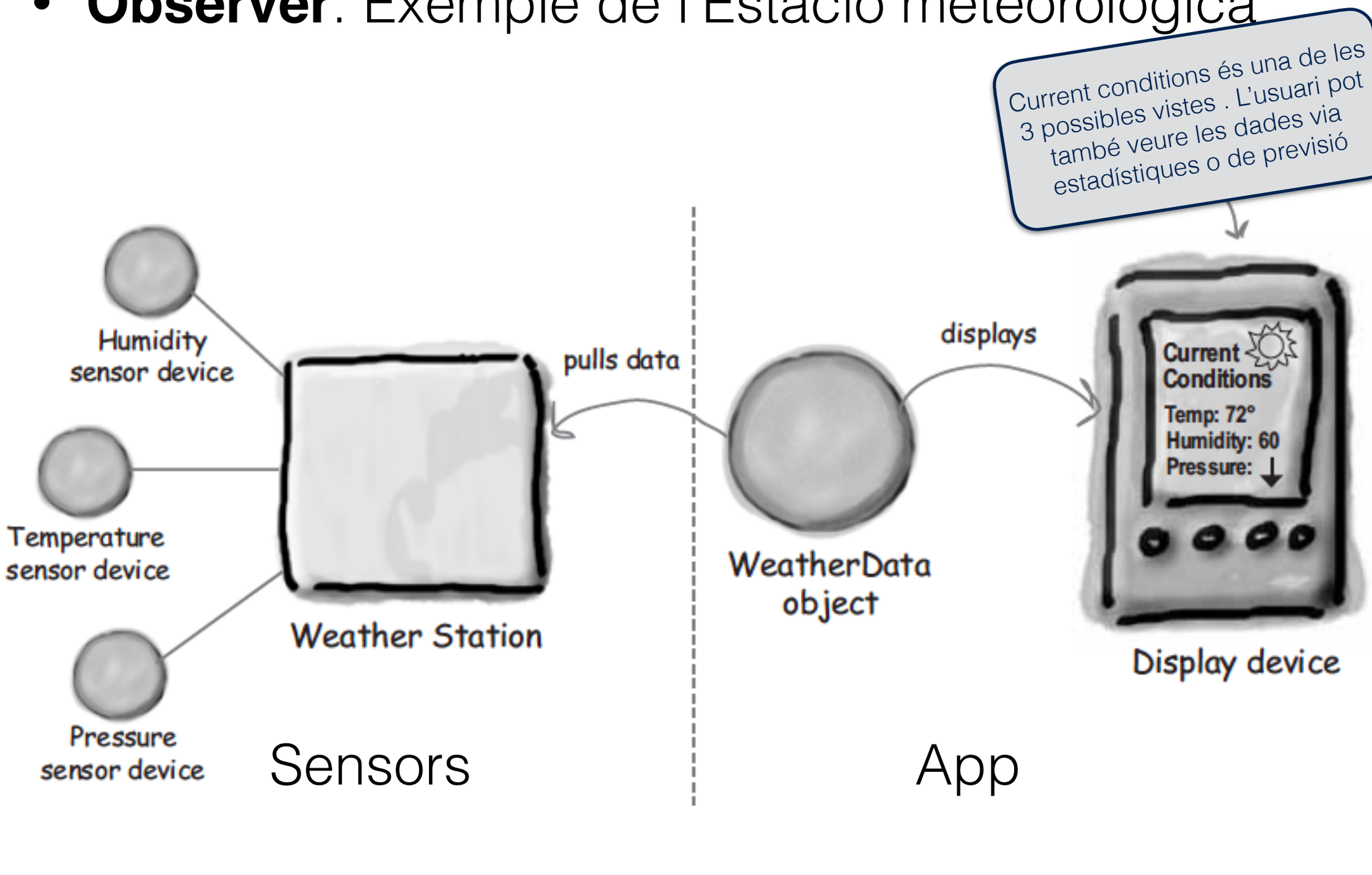
Subjectes o classes concretes a ser observades

4

Observadors concrets que fan el update segons el que ha notificat la classe observada. Generalment necessiten paràmetres de context que en alguns casos pot arribar a ser la mateixa classe observada

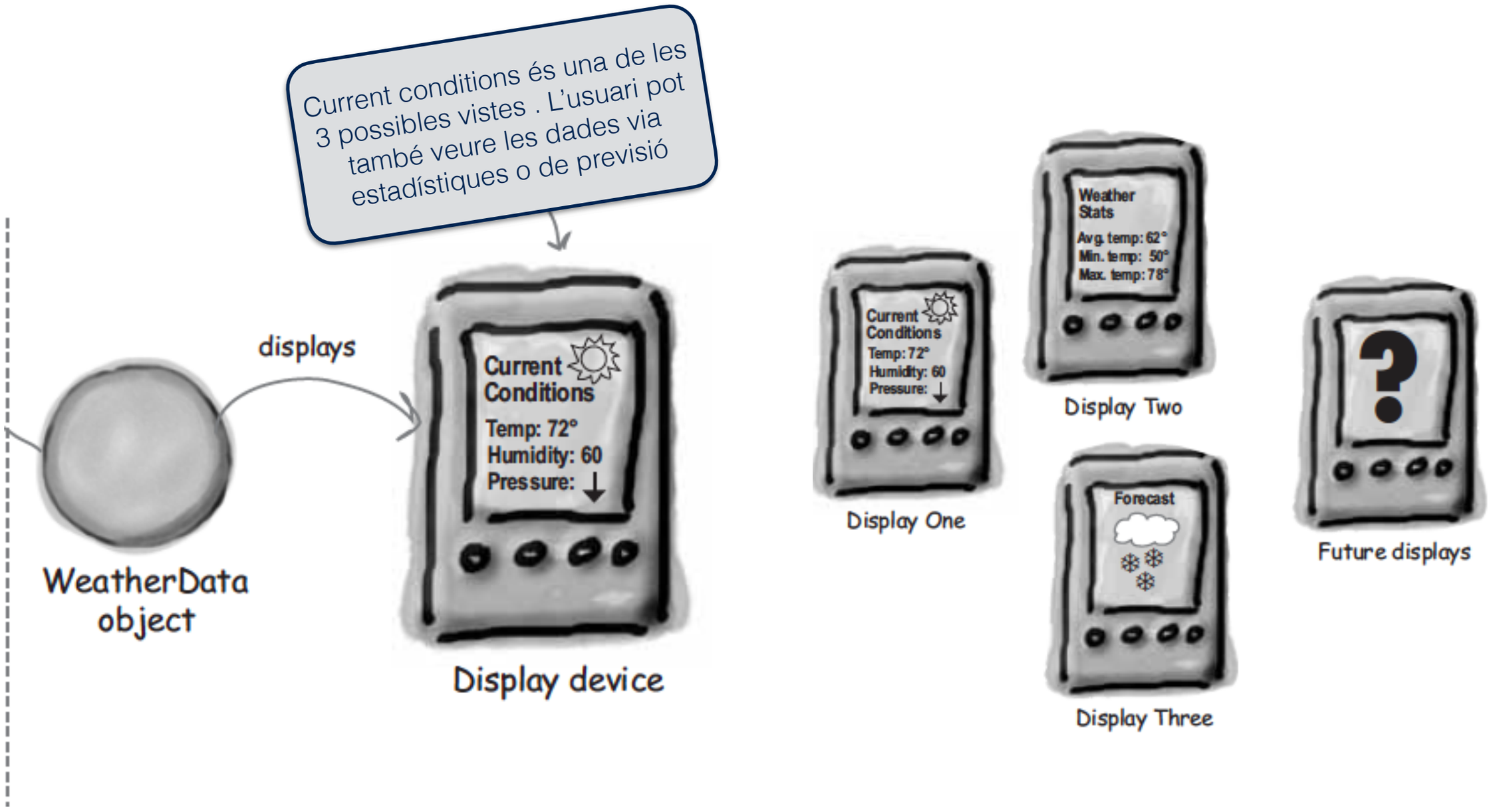
Patró Observer

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



Patró Observer

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



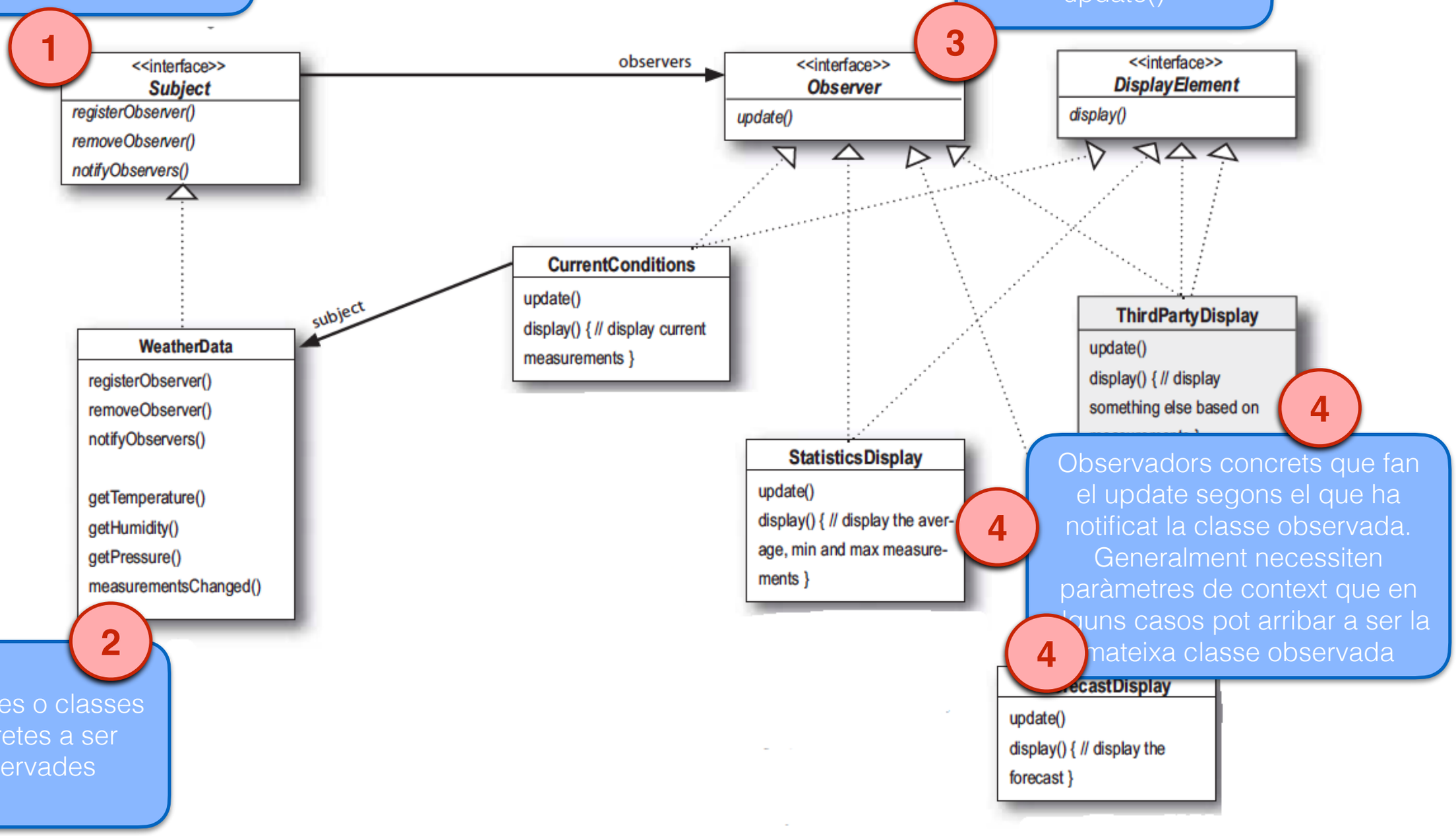
App

Patró Observer

erver: Exemple de l'Estació gica

Interfície que declara com subscriure's a un objecte que serà observat (o subjecte Els canvis en el Subjecte són interessants per altres classes

Interfície que declara com serà el tipus de notificació. en general, és una interfície amb només el mètode update()



Observadors concrets que fan el update segons el que ha notificat la classe observada. Generalment necessiten paràmetres de context que en alguns casos pot arribar a ser la mateixa classe observada

2

Subjectes o classes concretes a ser observades

Patró Observer

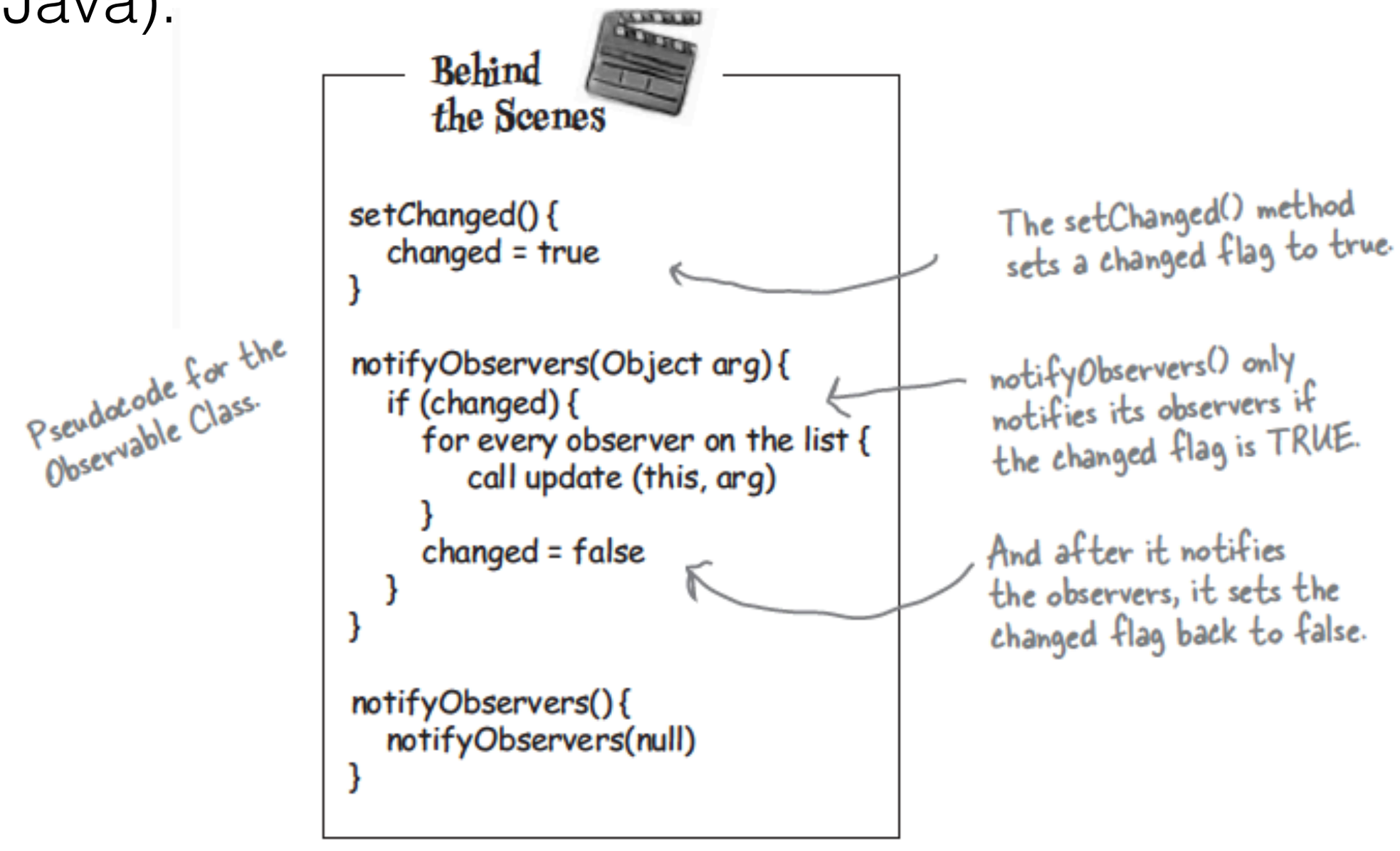
- **Observer**: Exemple de l'Estació meteorològica

- **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

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



Patró Observer

Nom del patró: **Observer**

Context:

Utilitzeu-lo quan canvis en l'estat d'un objecte requereix fer canvis en un altre objecte. Els objectes que canviaran no es coneixen a priori o poden canviar dinàmicament

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)
- Cal tenir en compte que els observadors poden cridar-se de forma no ordenada.
- Cal veure que en la implementació Java no es pot fer herència múltiple, ja que Observable és una classe i no una interfície

Patró Observer

Nom del patró: **Observer**

Context: Comportament

On s'usa en la realitat?

- A l'API de Swing:
  - Cada widget es pot veure com l'aplicació d'un patró Observer. Per exemple:
    - Un JButton és el Subject
    - Cada ActionListener és un Observer
    - Cada actionPerformed és el update()
  - Si s'usen expressions lambda, també és un observer?

```
button.addActionListener(event->
      System.out.println("Don't do it, you might regret it!"));
}
```

Aplicació al MVC

Patró en el Model:

- **Observer**