

# Classe Teoria Setmana 12: Disseny: Patrons de Disseny

Anna Puig

Enginyeria Informàtica  
Facultat de Matemàtiques i Informàtica,  
Universitat de Barcelona  
Curs 2021/22

# 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 <b>Patrons de Disseny</b>

# 3.4. Patrons de disseny

Propòsit → Àmbit ↓	CREACIÓ	ESTRUCTURA	COMPORTAMENT
<b>CLASSE</b>	<ul style="list-style-type: none"> <li>• <b>Factory method</b></li> </ul>	<ul style="list-style-type: none"> <li>• class Adapter</li> </ul>	<ul style="list-style-type: none"> <li>• Interpreter</li> <li>• Template method</li> </ul>
<b>OBJECTE</b>	<ul style="list-style-type: none"> <li>• <b>Abstract Factory</b></li> <li>• Builder</li> <li>• Prototype</li> <li>• Singleton</li> <li>• Object pool</li> </ul>	<ul style="list-style-type: none"> <li>• Object Adapter</li> <li>• Bridge</li> <li>• Composite</li> <li>• Decorator</li> <li>• Facade</li> <li>• Flyweight</li> <li>• Proxy</li> </ul>	<ul style="list-style-type: none"> <li>• Chain of Responsibility</li> <li>• Command</li> <li>• Iterator</li> <li>• Mediator</li> <li>• Memento</li> <li>• Observer</li> <li>• State</li> <li>• Strategy</li> <li>• Visitor</li> </ul>

# Exercici Pizza Store

En aquesta classe  
es creen les pizzes

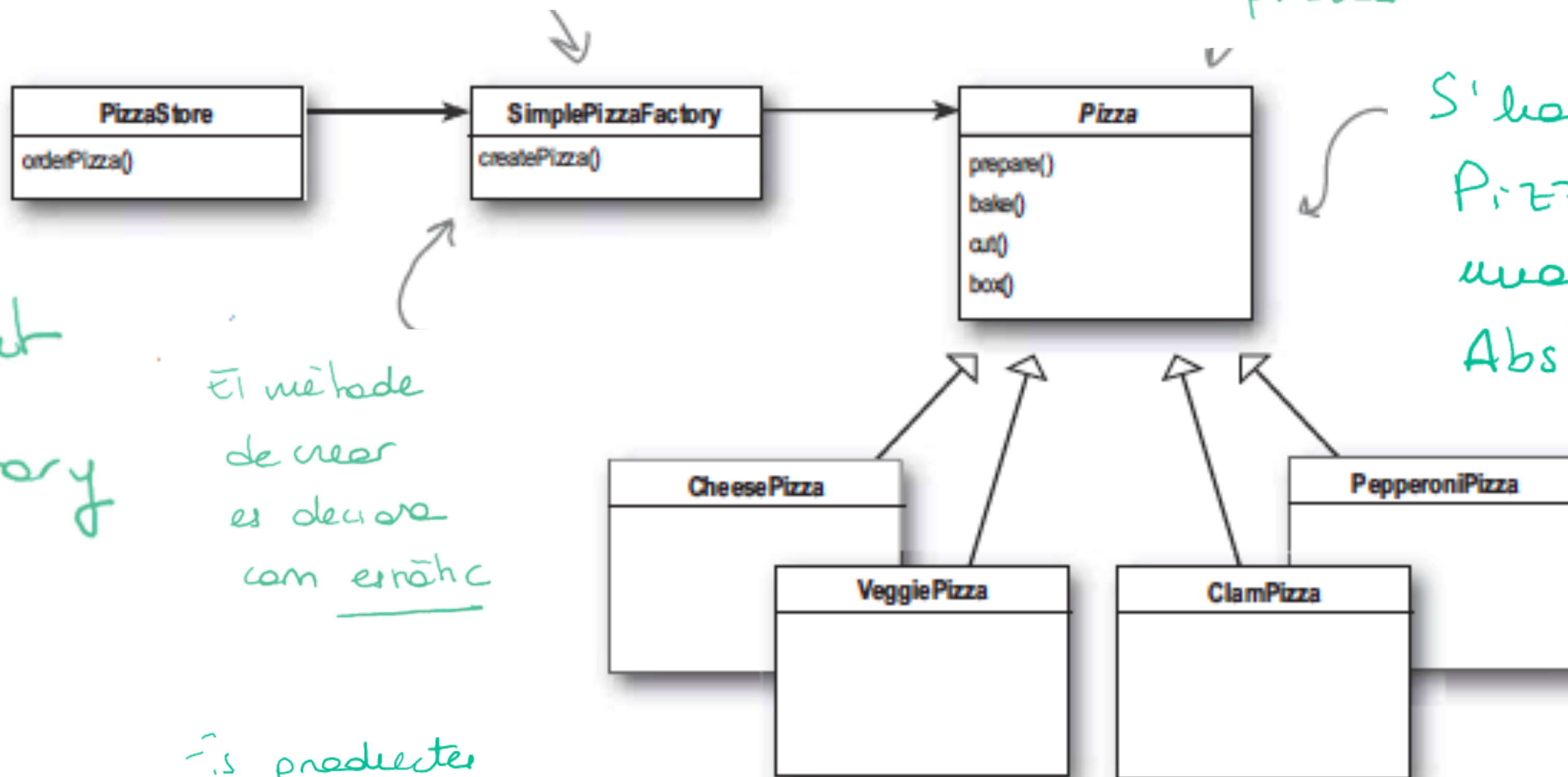
És el producte  
de la factory  
pizza

S'ha definit  
Pizza com  
una classe  
Abstracte

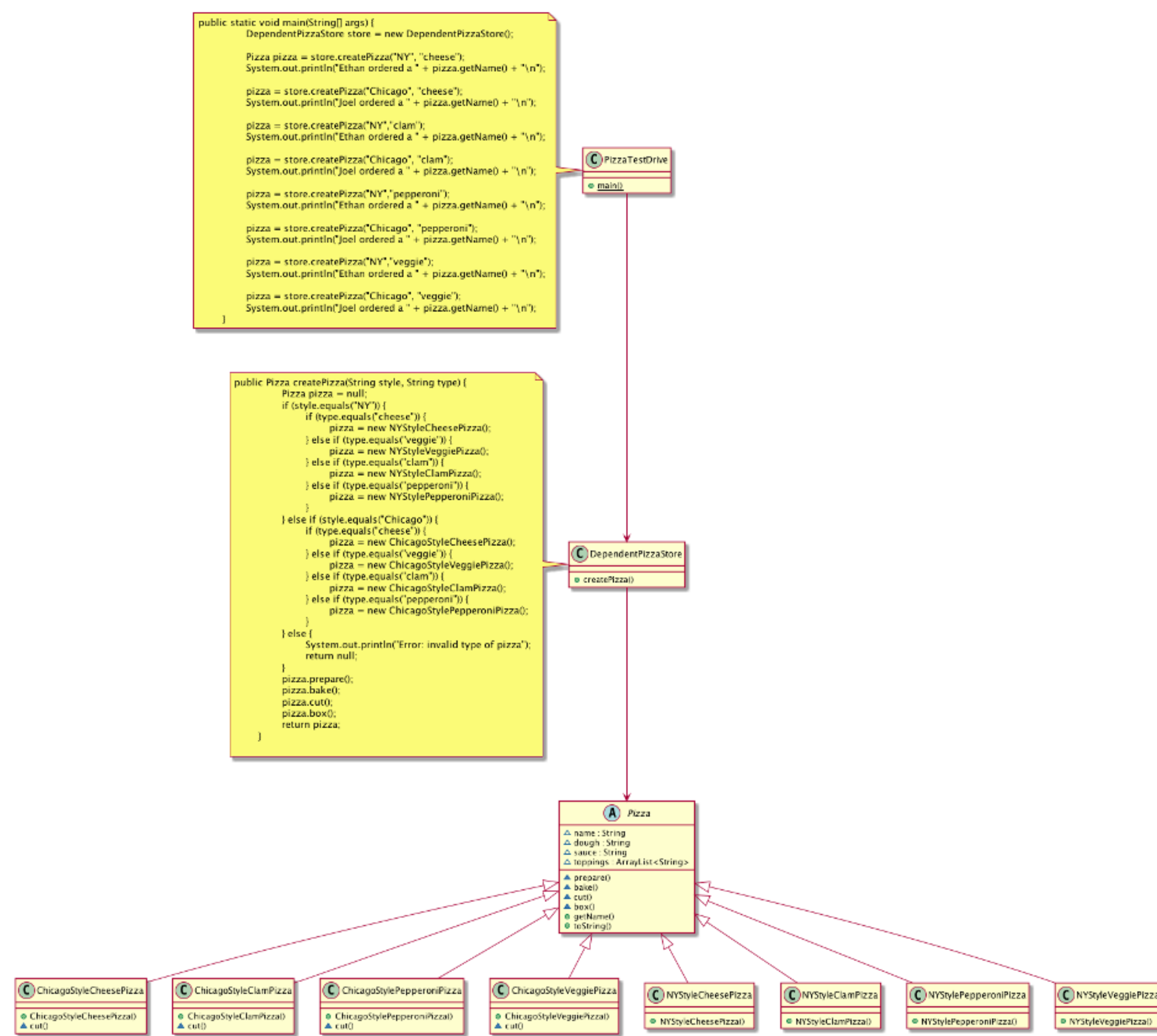
És el client  
de la Factory

El mètode  
de crear  
es declara  
com abstracte

És productes  
concrets són els  
diferents tipus de  
pizzes

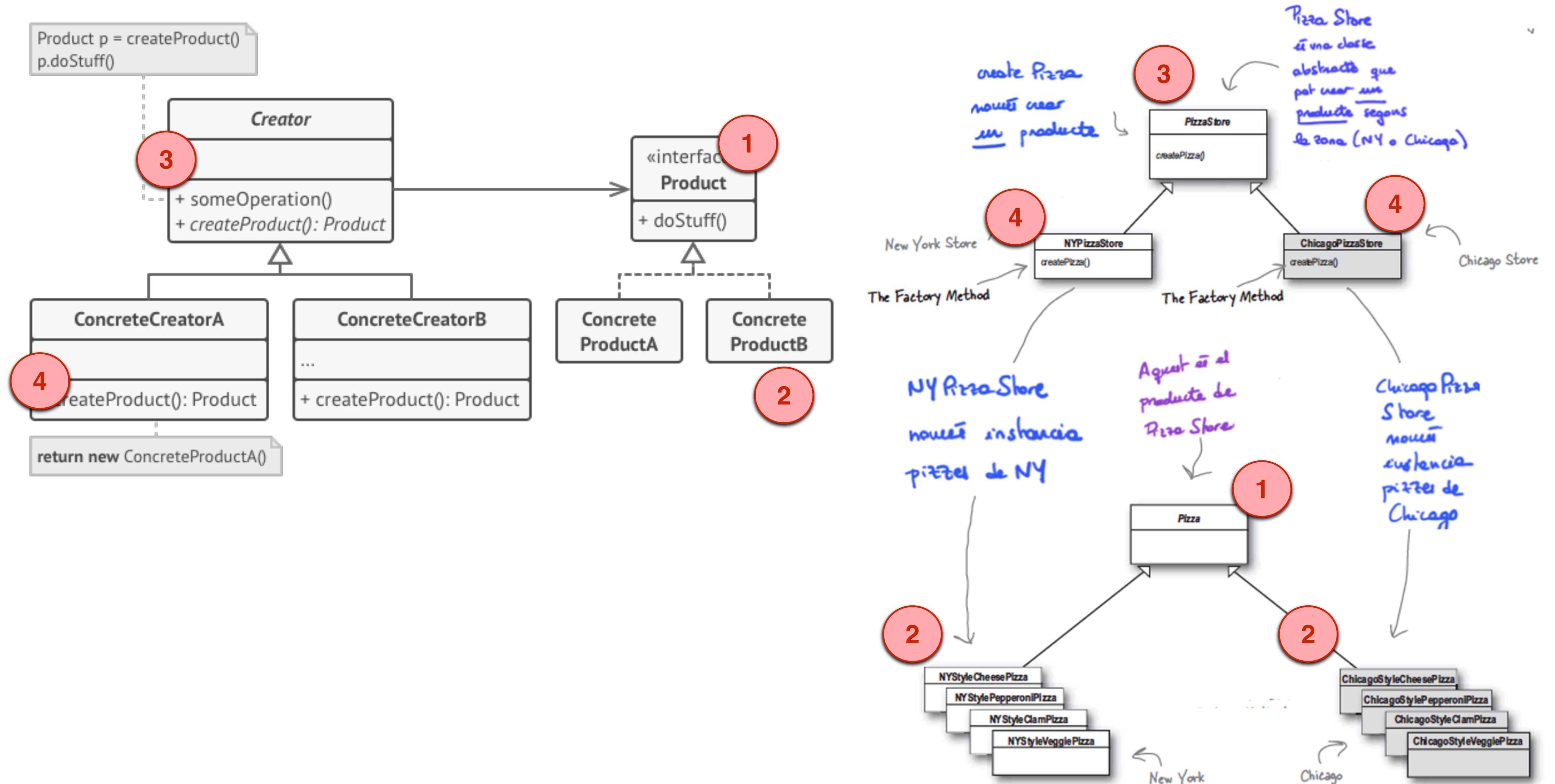


# Exercici Pizza Store: Projecte franquícies



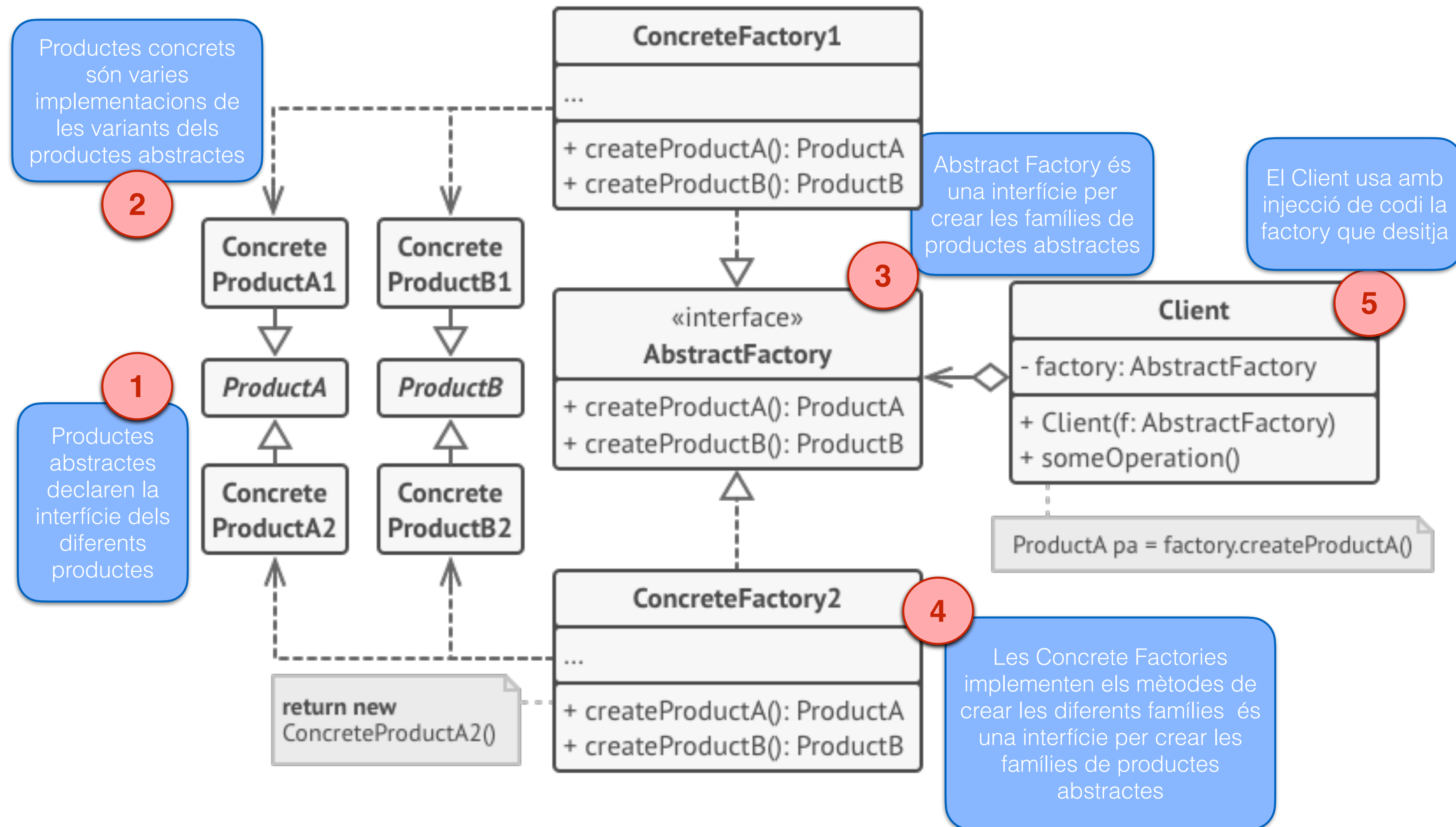


# Exercici Pizza Store: Projecte Franquícies



# Exercici Pizza Store: Projecte ingredients

## Patró Abstract Factory



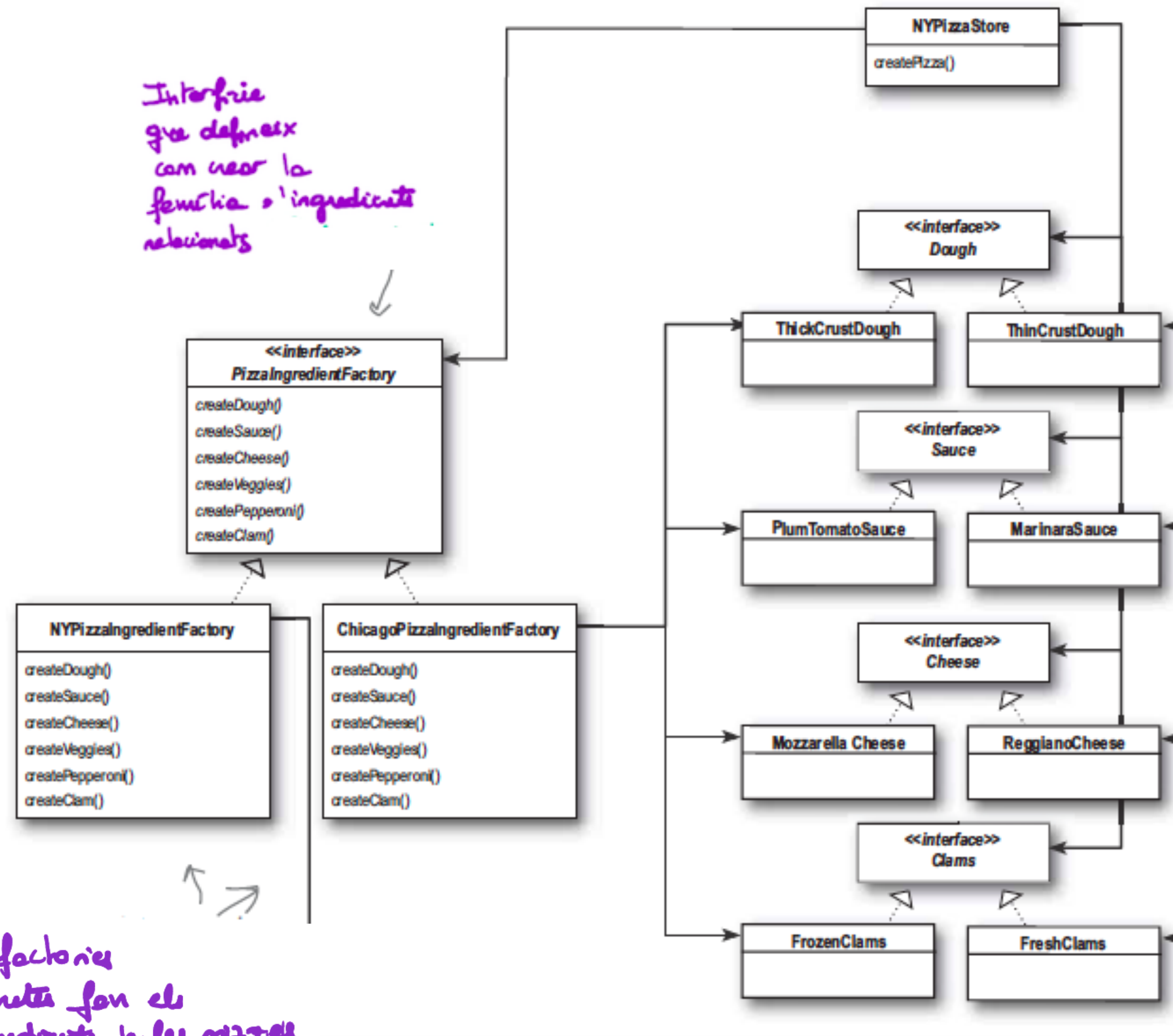
# Exercici Pizza Store: Projecte ingredients

Patró Abstract Factory

les factories  
converteixen els  
ingredients de les pizzes

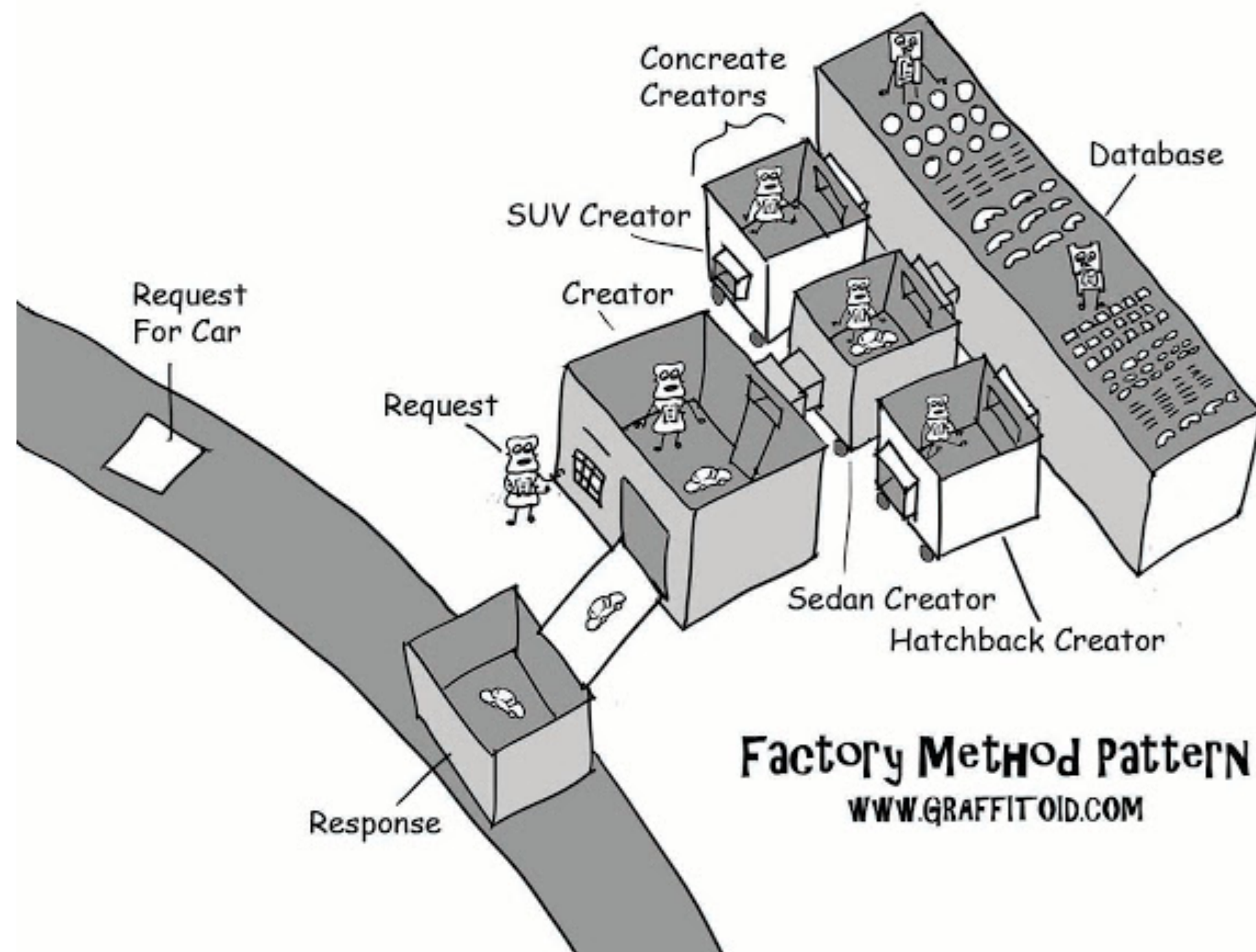
Cada factory  
fa diferents implementacions  
de funcions de productes

Els clients  
de l'Abstract  
Factory són  
les instàncies  
de la PizzaStore



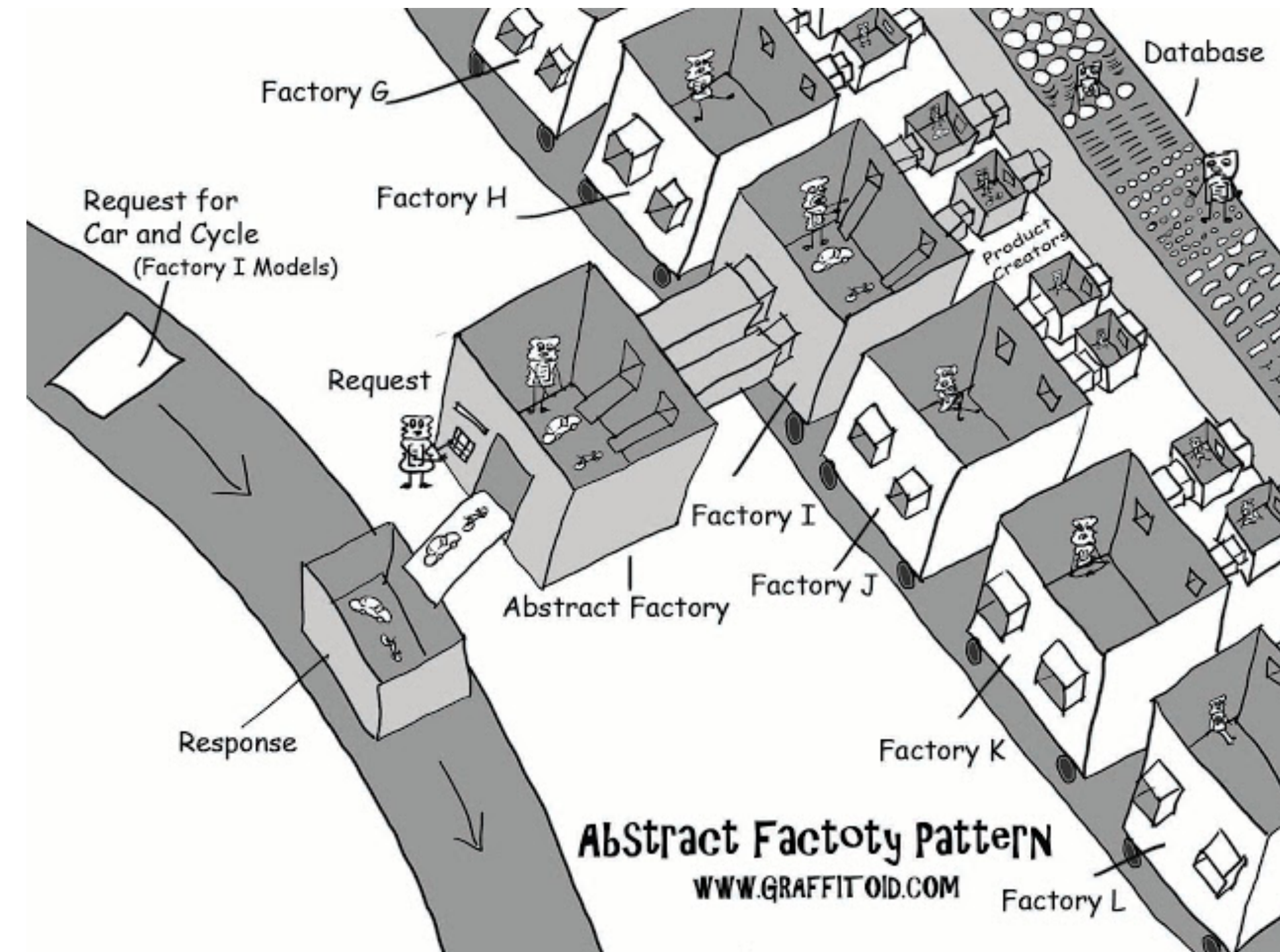


## 3.5. Patrons de disseny



### Factory pattern

Un únic producte



### Abstract Factory

Múltiples productes relacionats entre si

Múltiples combinacions de productes???



# 3.4. Patrons de disseny



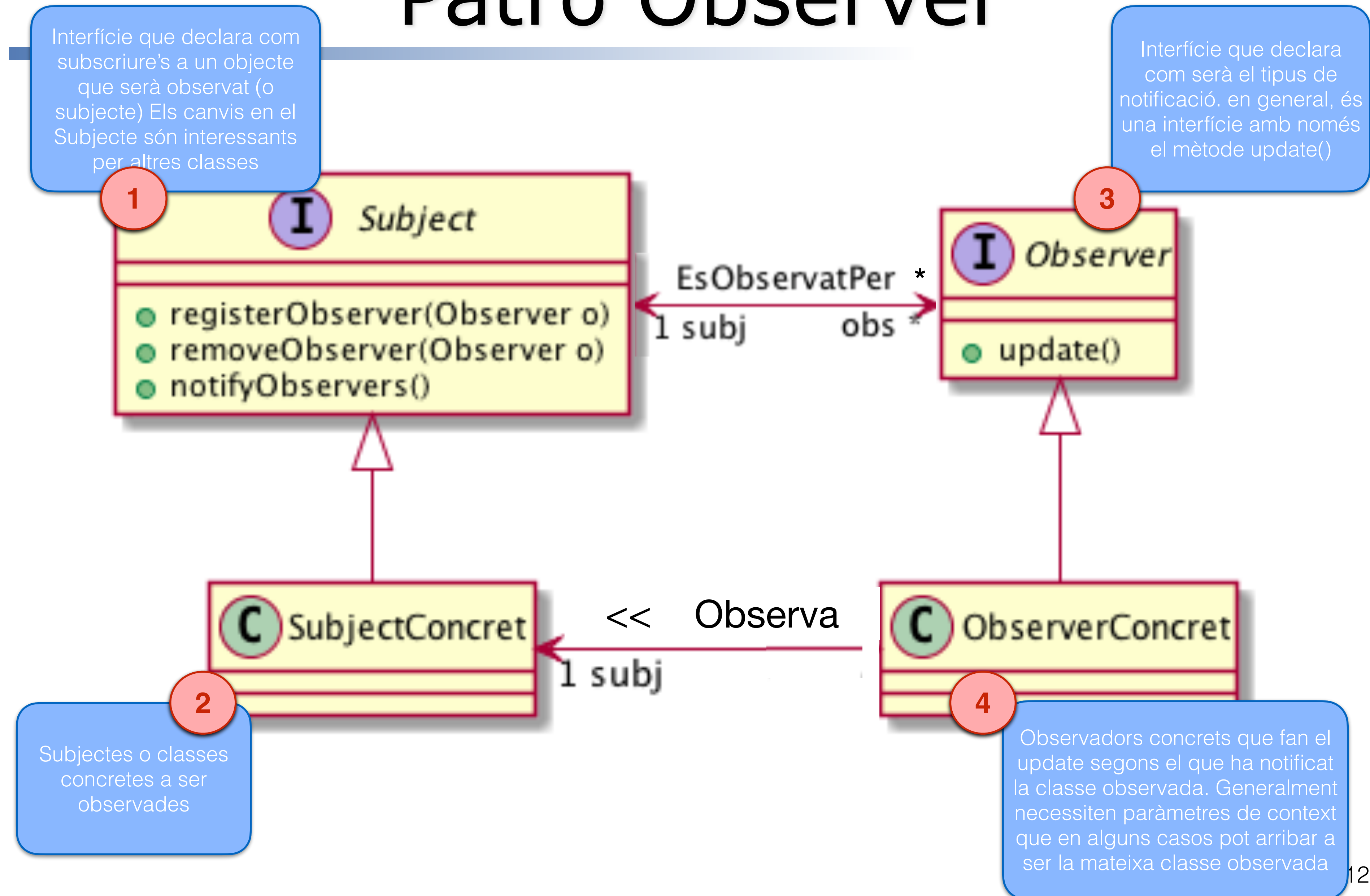
Propòsit → Àmbit ↓	CREACIÓ	ESTRUCTURA	COM
CLASSE	<ul style="list-style-type: none"><li>• Factory method</li></ul>	<ul style="list-style-type: none"><li>• class Adapter</li></ul>	<ul style="list-style-type: none"><li>• Interpreter</li><li>• Template method</li></ul>
OBJECTE	<ul style="list-style-type: none"><li>• Abstract Factory</li><li>• Builder</li><li>• Prototype</li><li>• Singleton</li><li>• Object pool</li></ul>	<ul style="list-style-type: none"><li>• Object Adapter</li><li>• Bridge</li><li>• Composite</li><li>• Decorator</li><li>• Facade</li><li>• Flyweight</li><li>• Proxy</li></ul>	<ul style="list-style-type: none"><li>• Chain of Responsibility</li><li>• Command</li><li>• Iterator</li><li>• Mediator</li><li>• Memento</li><li>• <b>Observer</b></li><li>• State</li><li>• Strategy</li><li>• Visitor</li></ul>

# Observer

---

- **Observer** – Patró de disseny de comportament que permet definir un mecanisme de subscripció per notificar diversos objectes sobre qualsevol esdeveniment que passi a l'objecte que estan observant.

# Patrón Observer





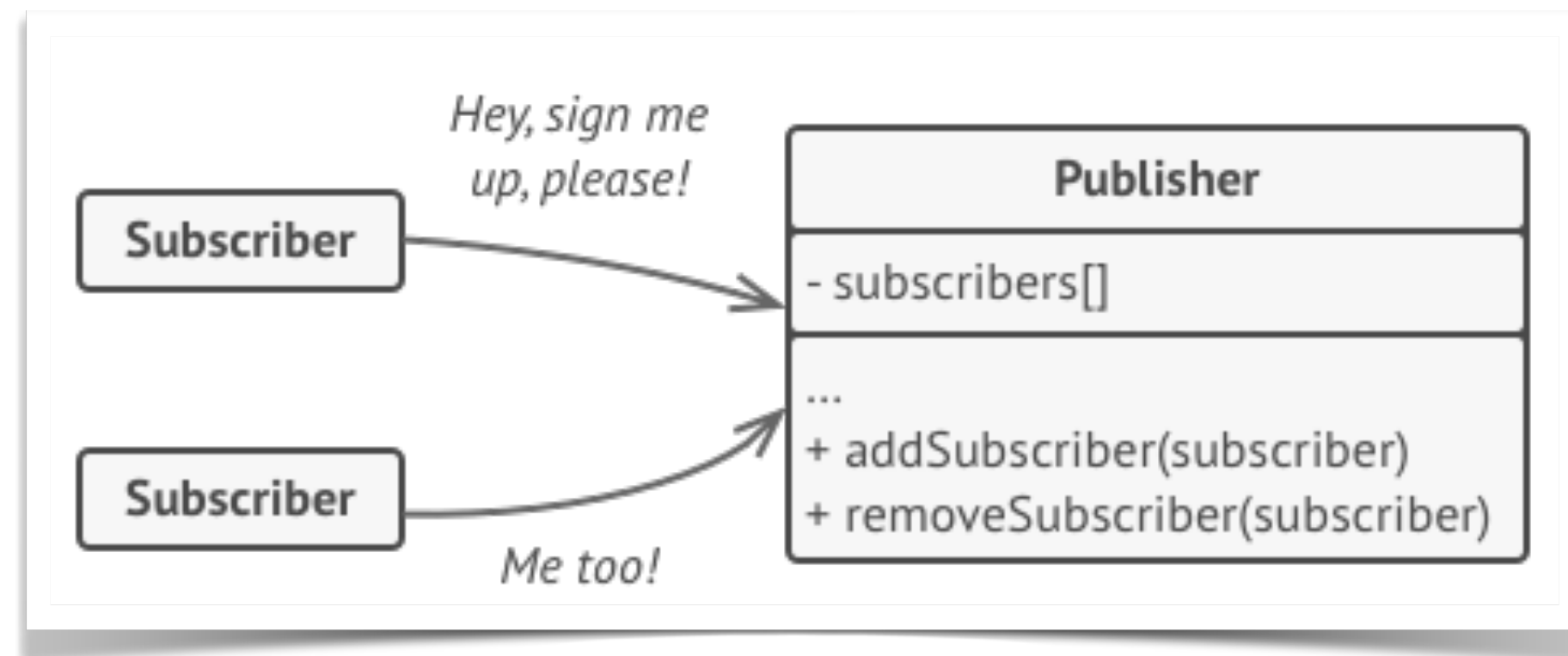
# Patró Observer

**Patró:**

- **Observer/  
Observable**

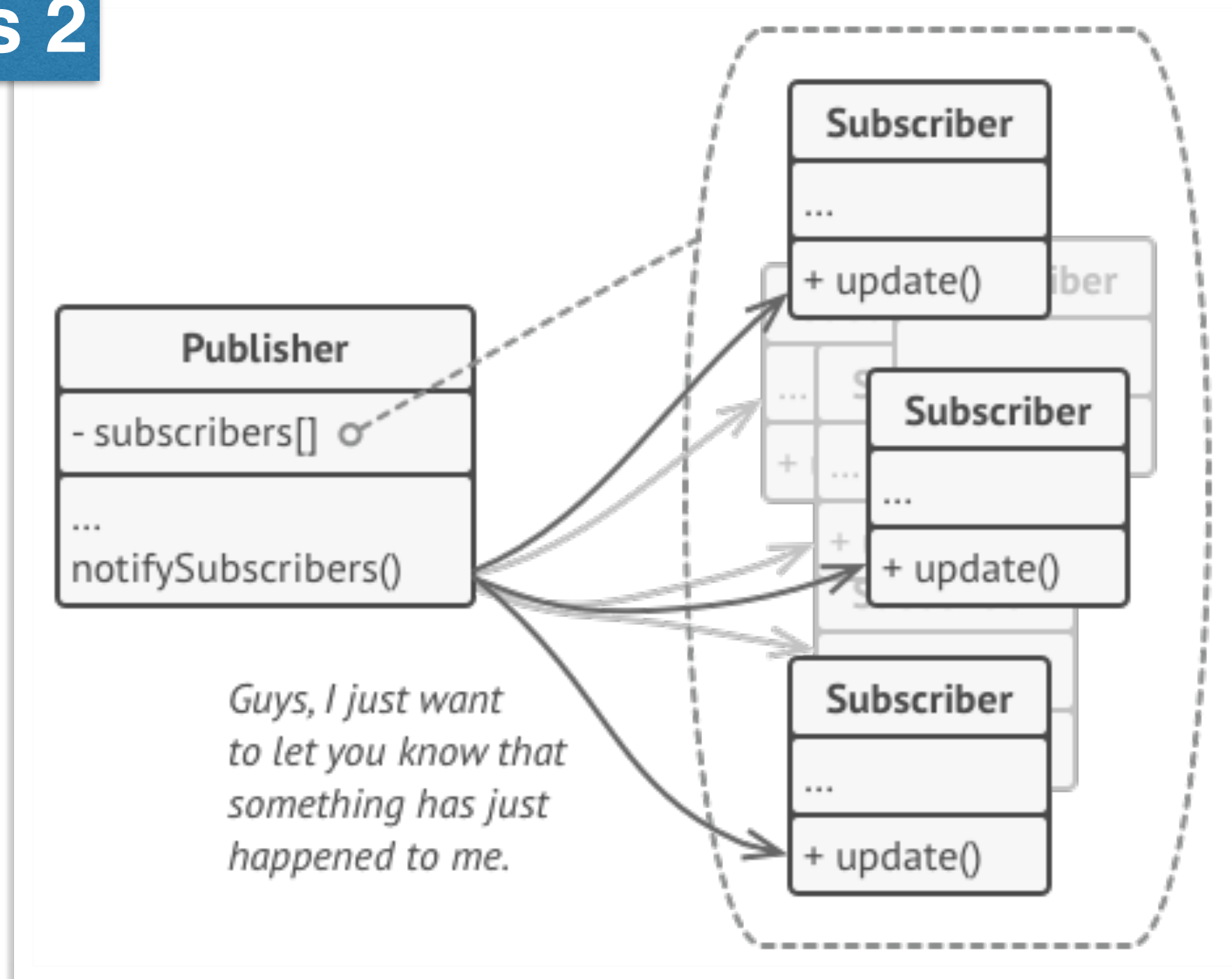


**Pas 1**



**Observers es subscriuen a l'Observat (subject)**

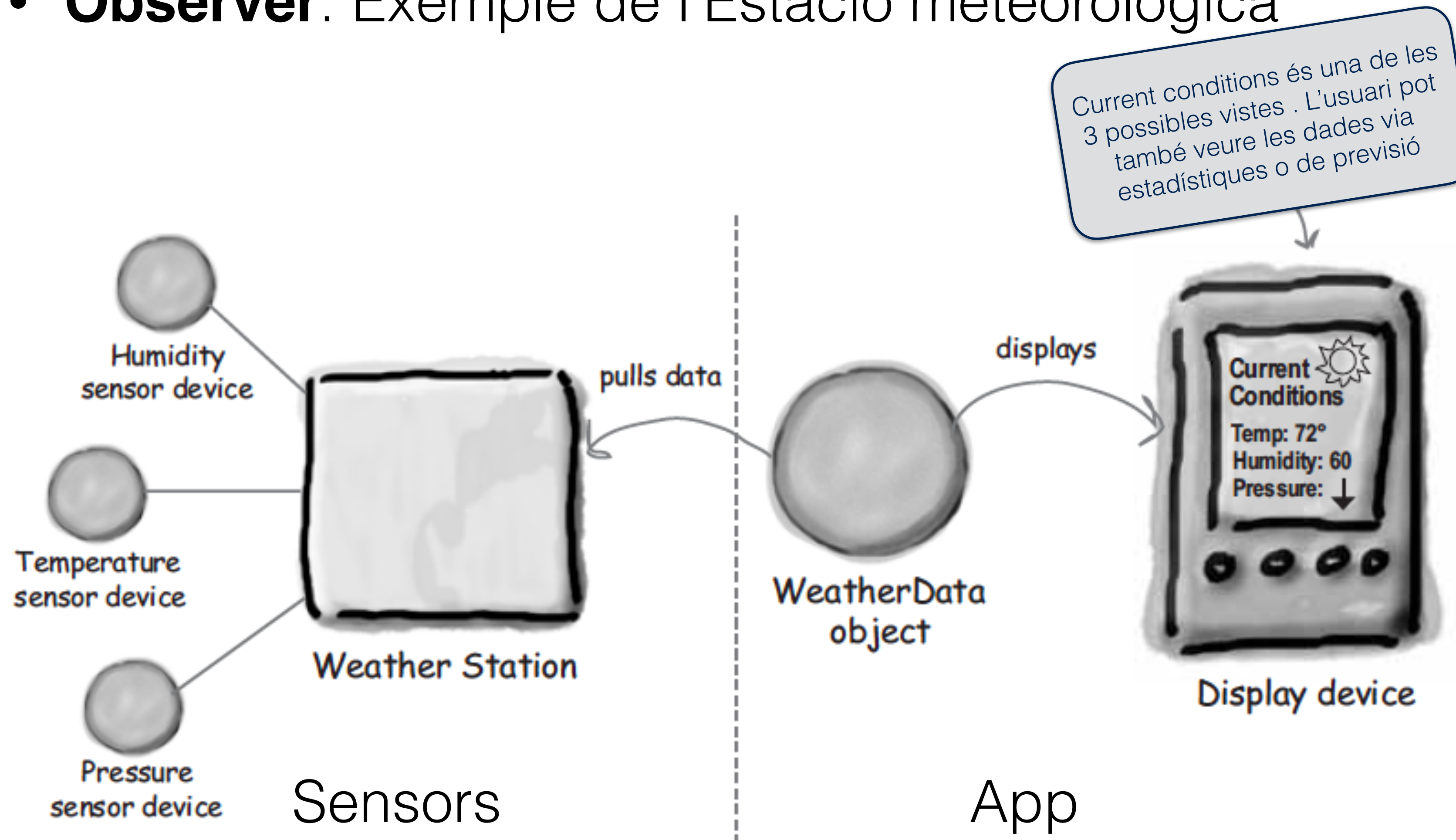
**Pas 2**



**L'observat notifica el canvi als observadors els canvis (push)**

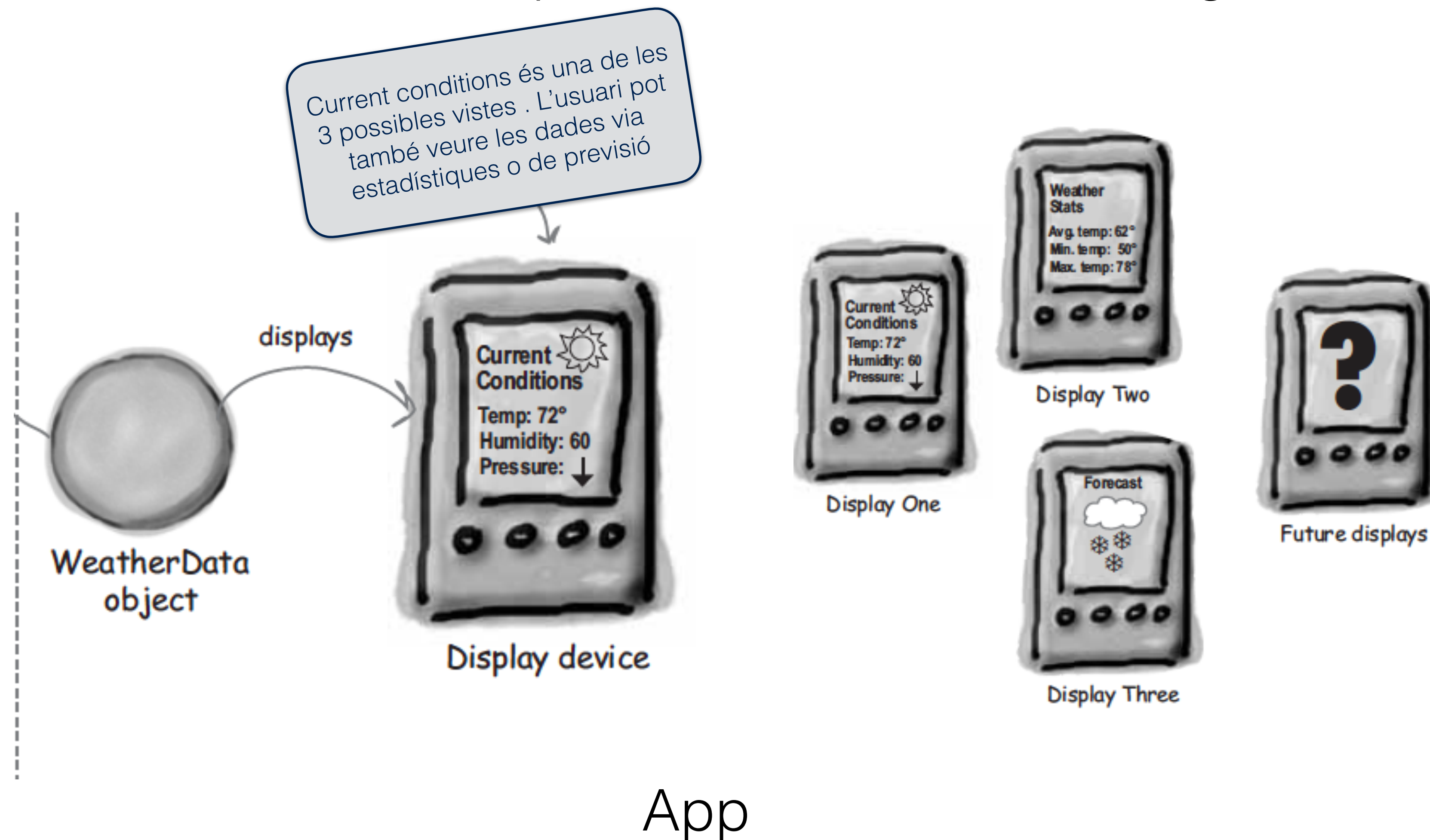
# Patró Observer

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



# Patró Observer

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

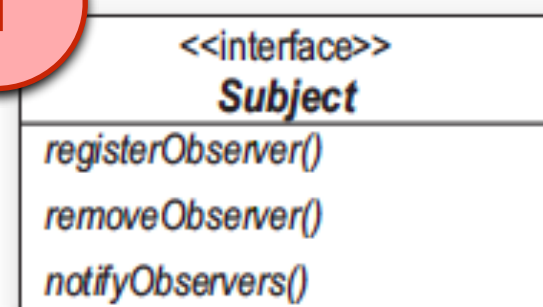




# Patró Observer

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



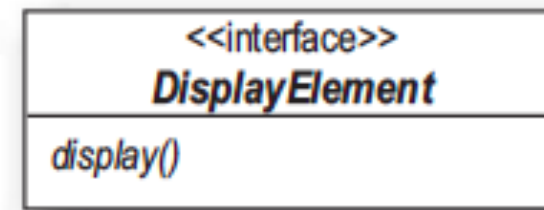
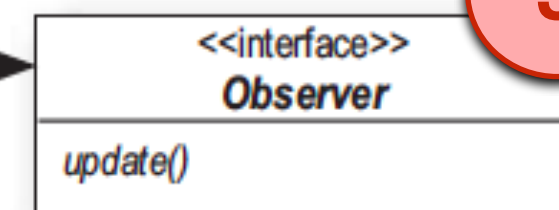
2

Subjectes o classes concretes a ser observades

ver: Exemple de l'Estació m

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

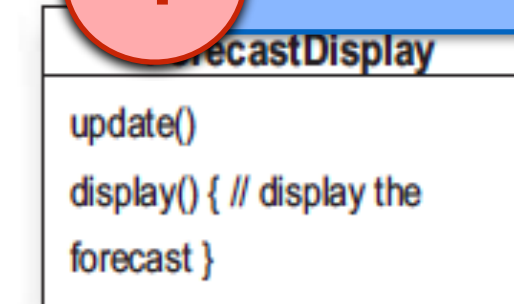
3



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 la mateixa classe observada

4





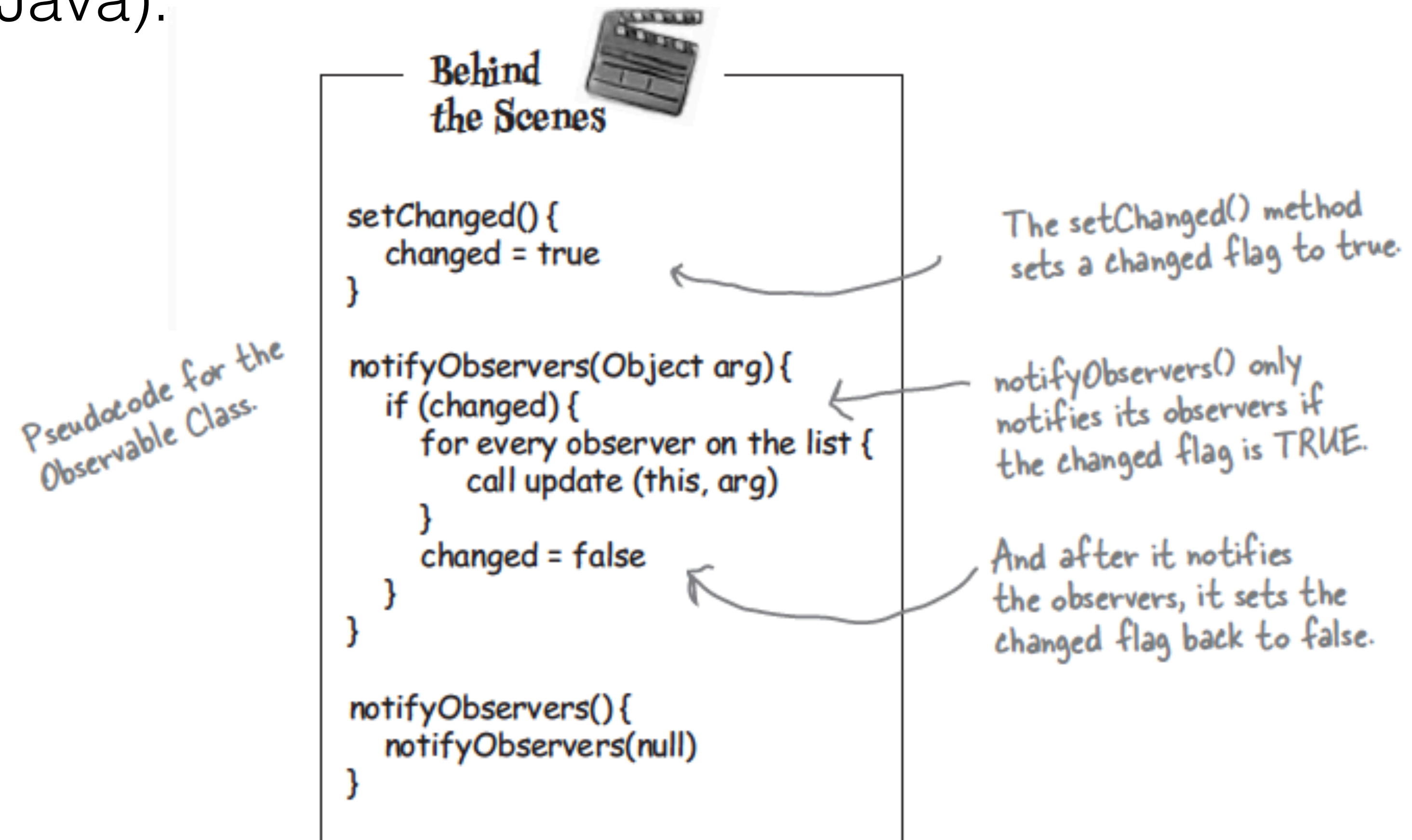
# Patró Observer

---

- **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. Open Closed? Single Responsibility? Dependency?
  6. Analitza el codi del paquet **weatherObservable** (usa les classes **Observable** i **Observer** de Java)

# Patró Observer

- **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

### Observer

