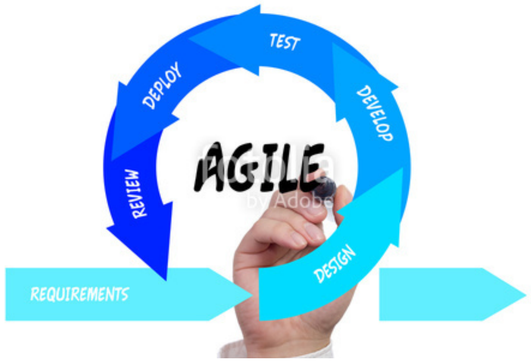


Classe Teoria Setmana 12: Disseny: Patrons de Disseny

Anna Puig

Enginyeria Informàtica

Facultat de Matemàtiques i Informàtica,

Universitat de Barcelona

Curs 2021/22

UNIVERSITAT DE BARCELONA

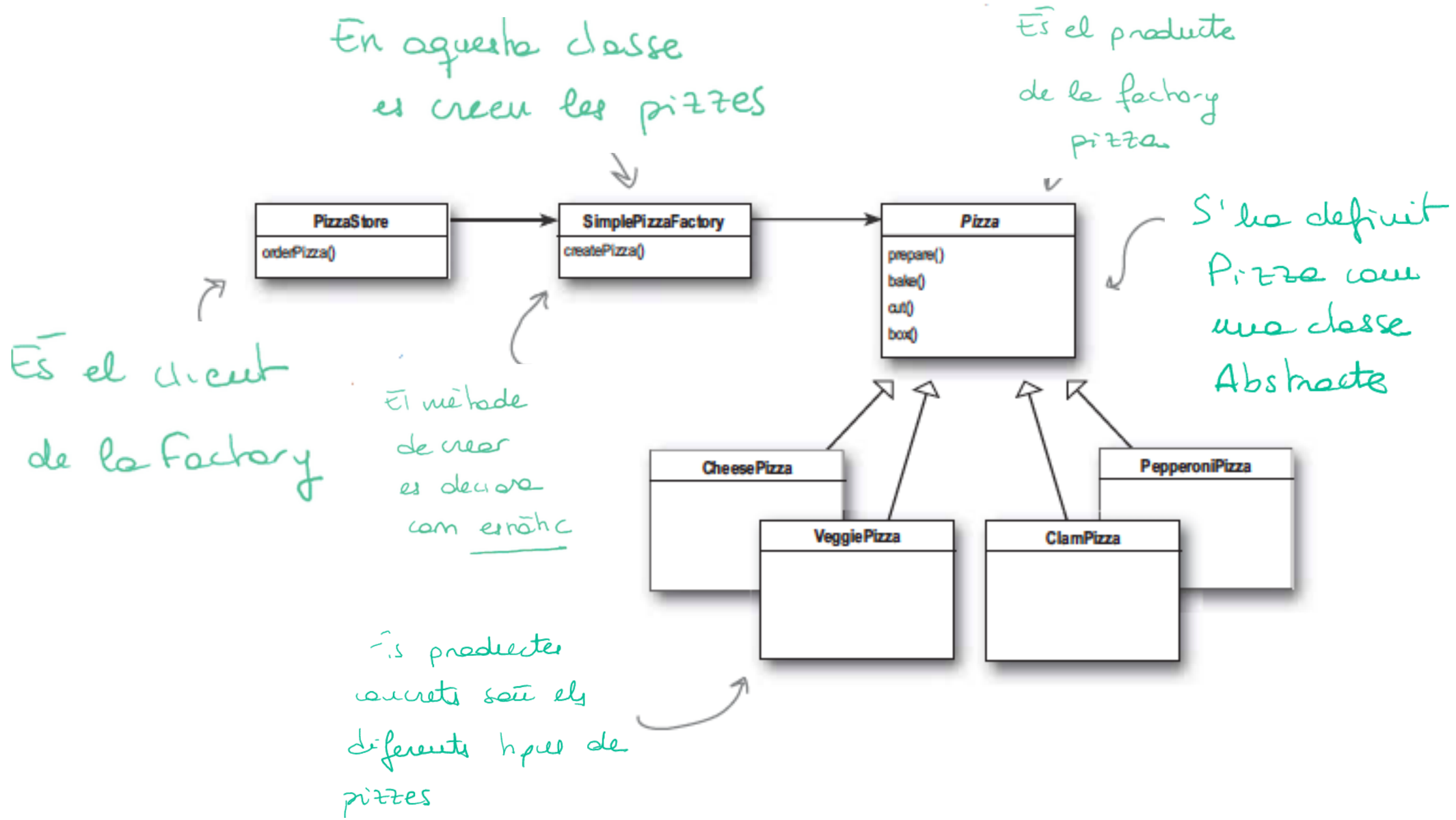
Temari

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

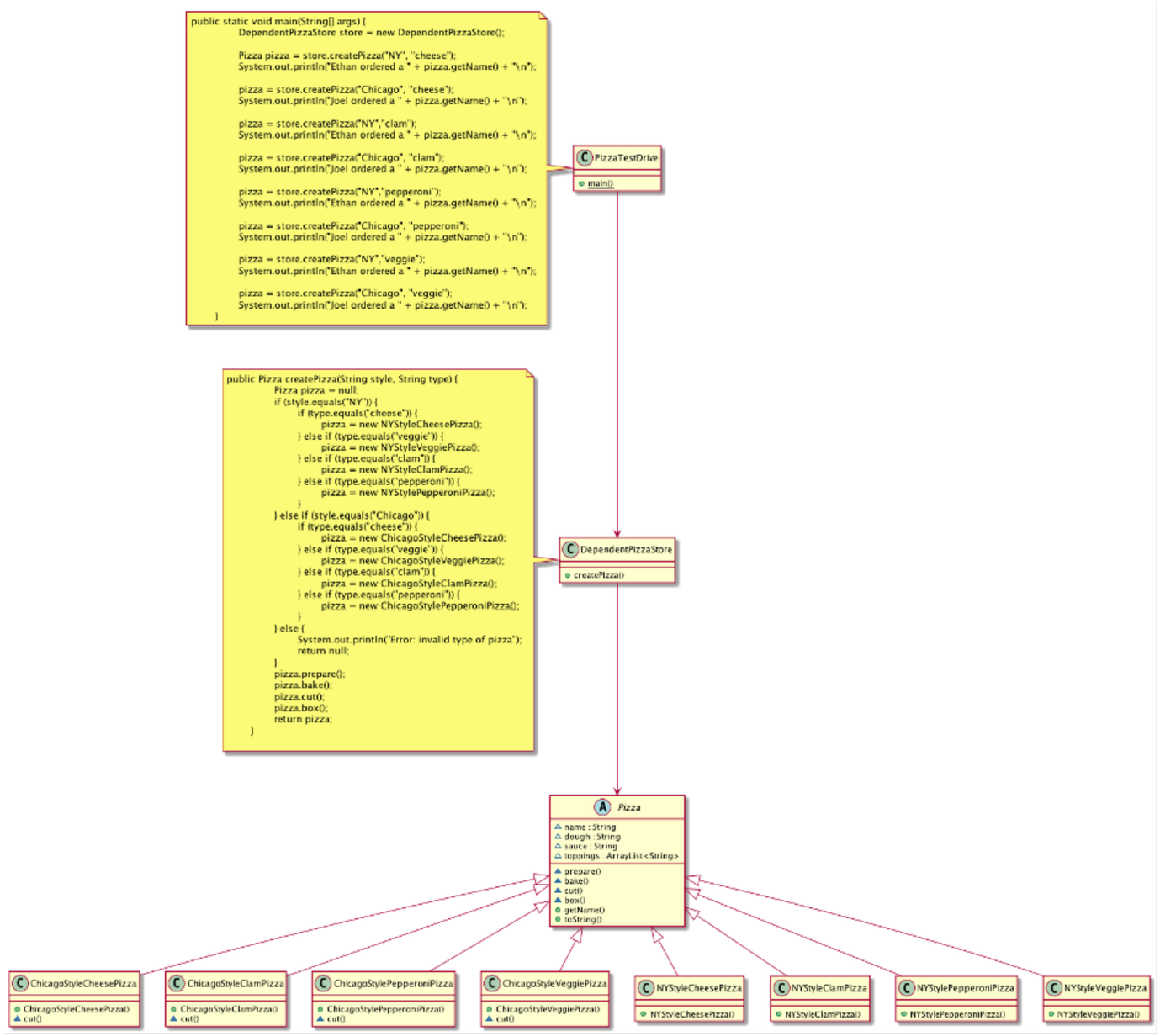
3.4. Patrons de disseny

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

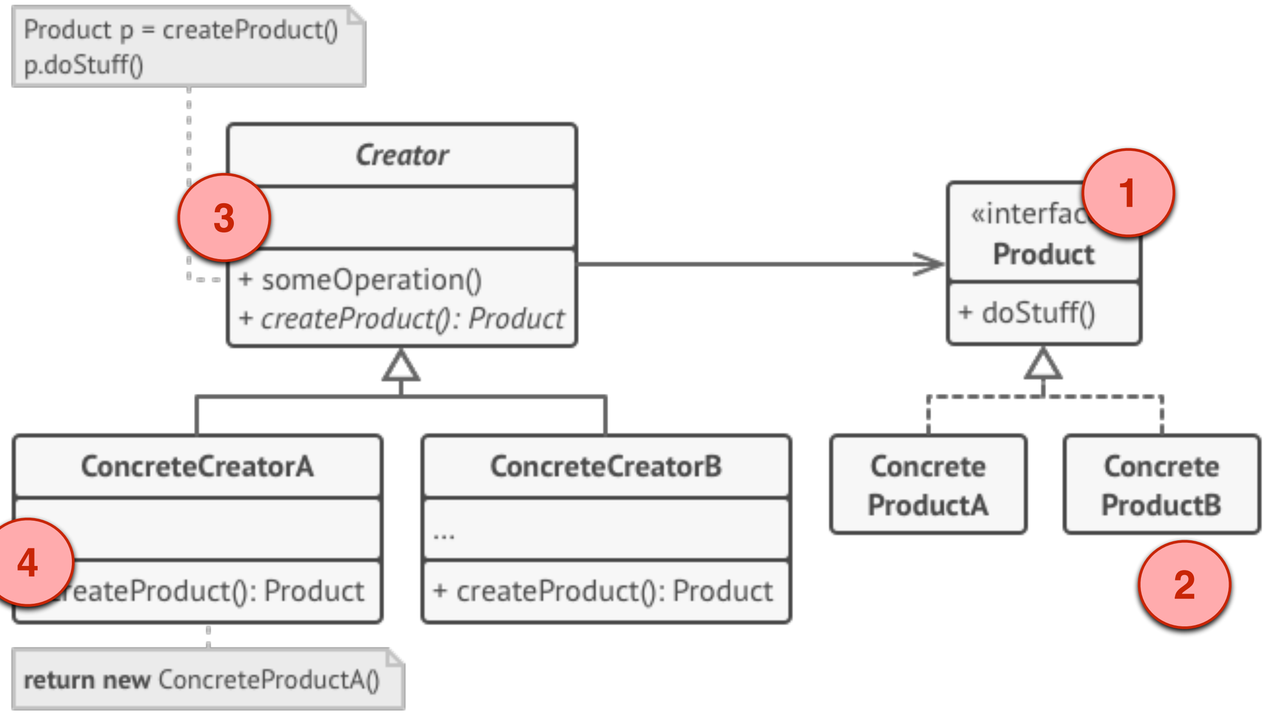
Exercici Pizza Store

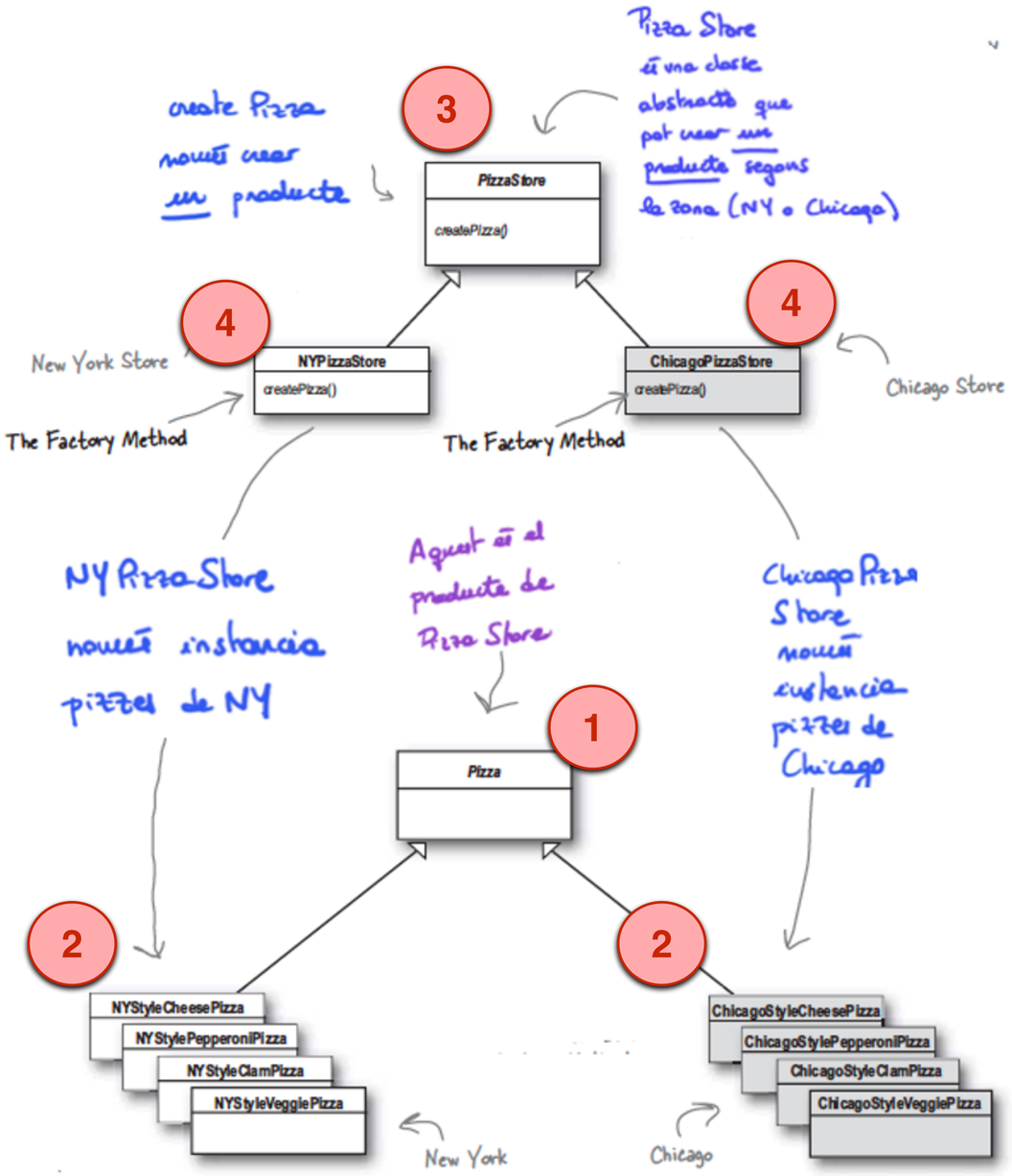


Exercici Pizza Store: Projecte franquícies



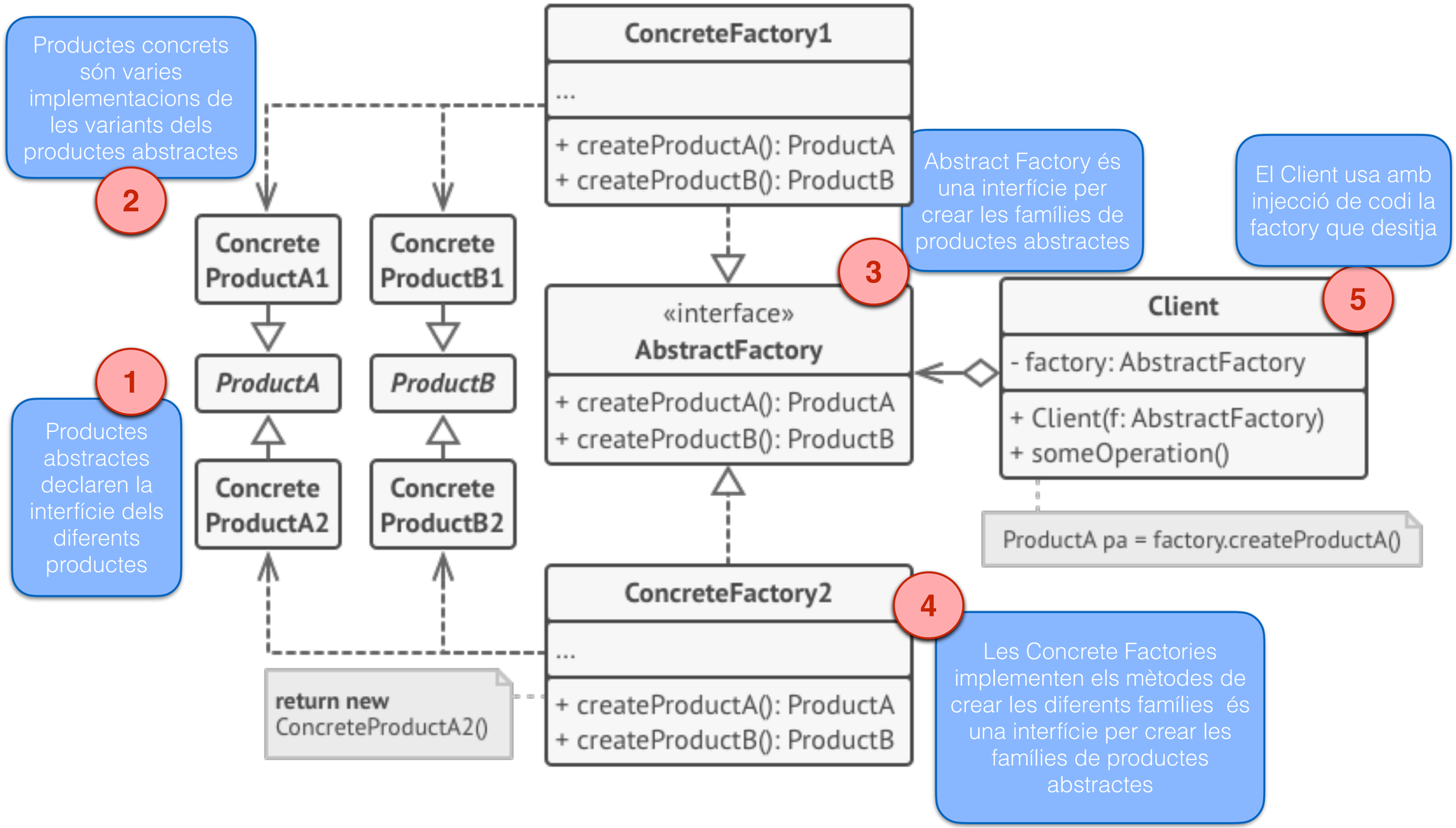
Exercici Pizza Store: Projecte Franquícies





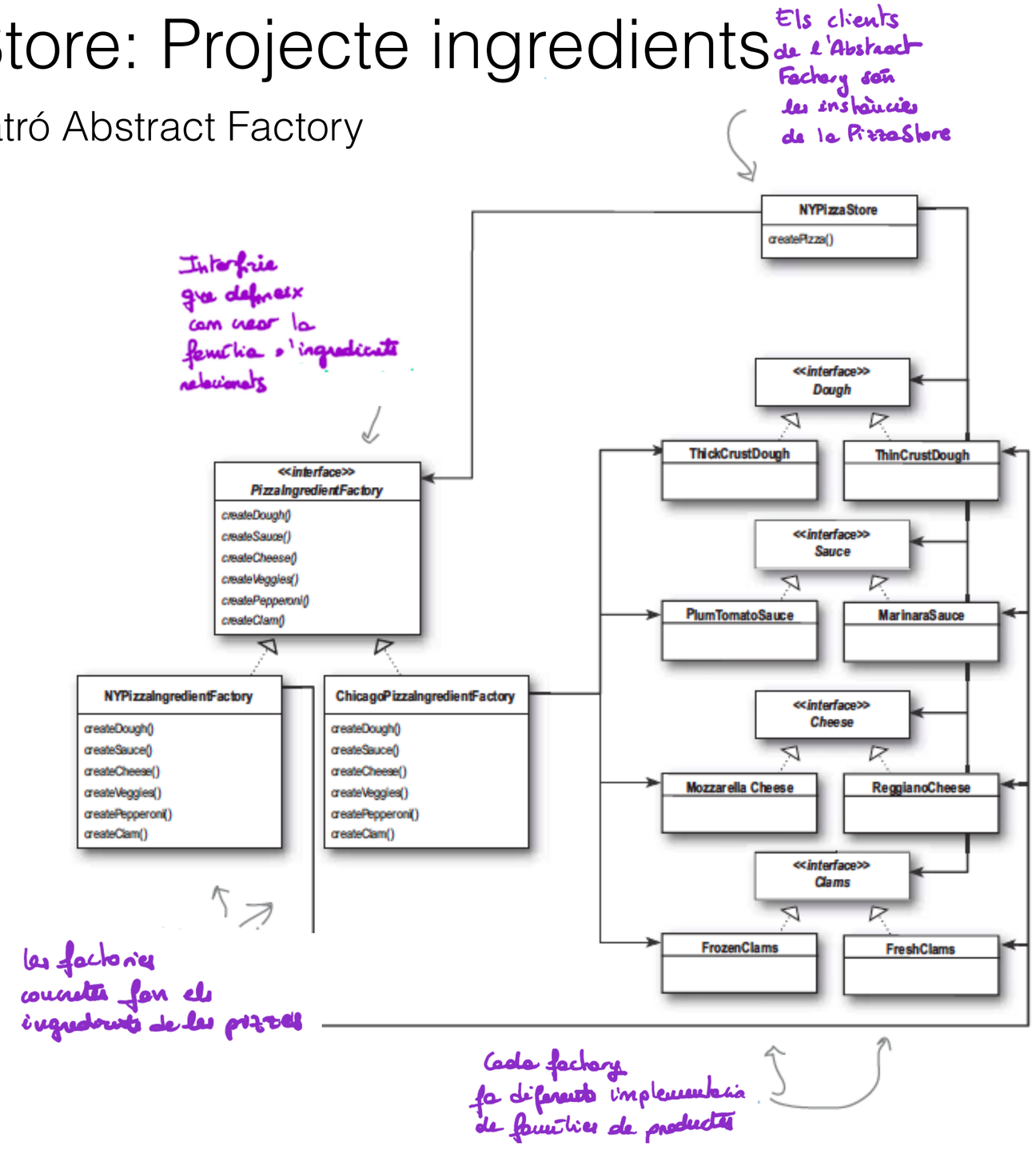
Exercici Pizza Store: Projecte ingredients

Patró Abstract Factory

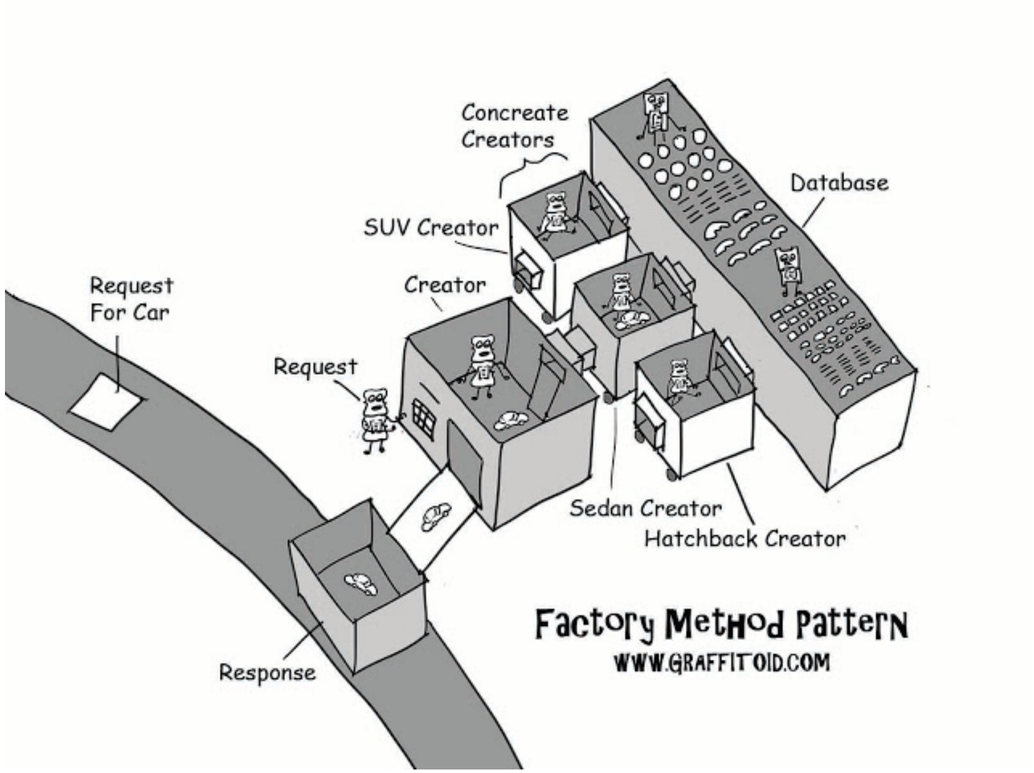


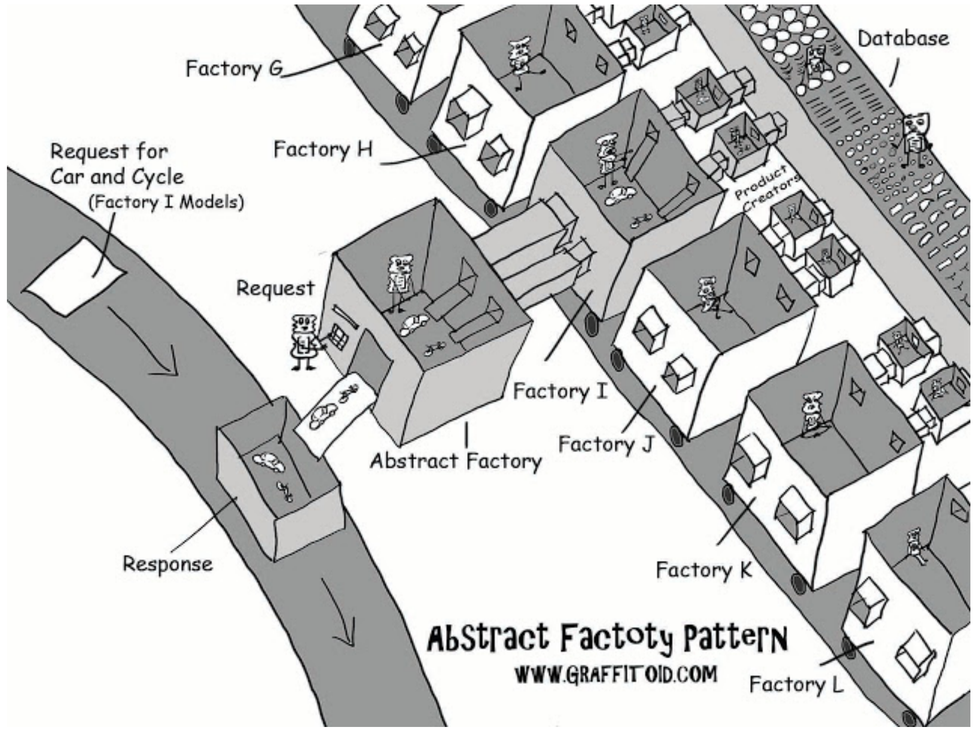
Exercici Pizza Store: Projecte ingredients

Patró Abstract Factory



3.5. Patrons de disseny





Factory pattern

Un únic producte

Abstract Factory

Múltiples productes relacionats entre si

Múltiples combinacions de productes???

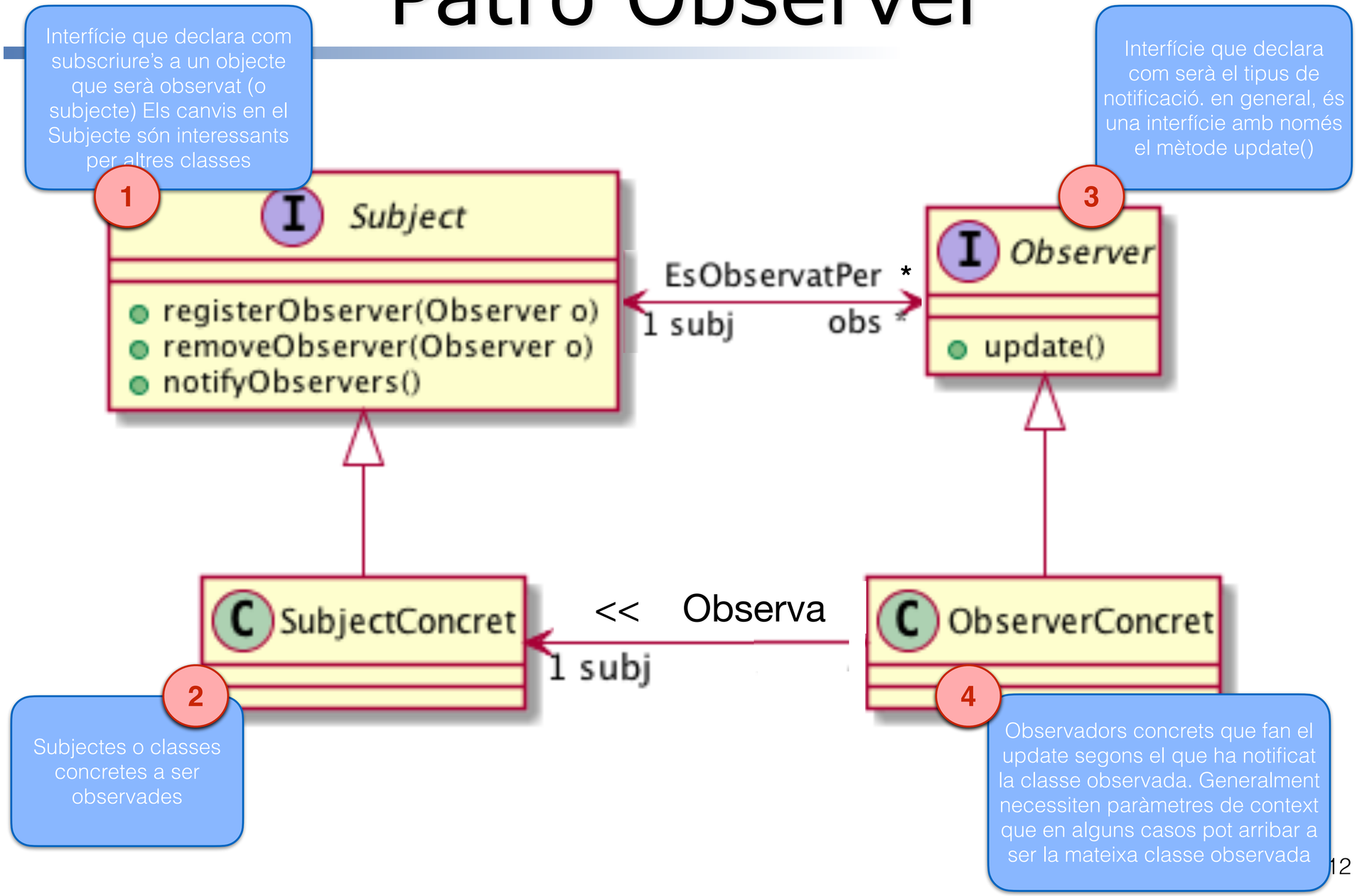
3.4. Patrons de dissen

Propòsit → **Àmbit ↓**	**CREACIÓ**	**ESTRUCTURA**	**COM**
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

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.

Patró Observer

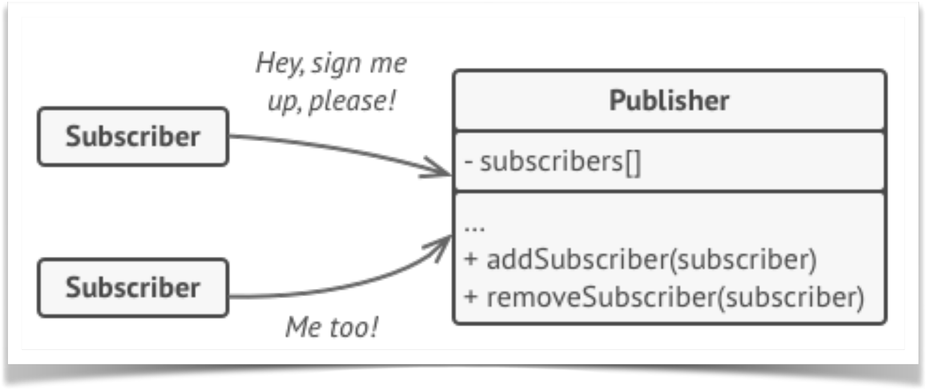


Patró Observer

Patró:

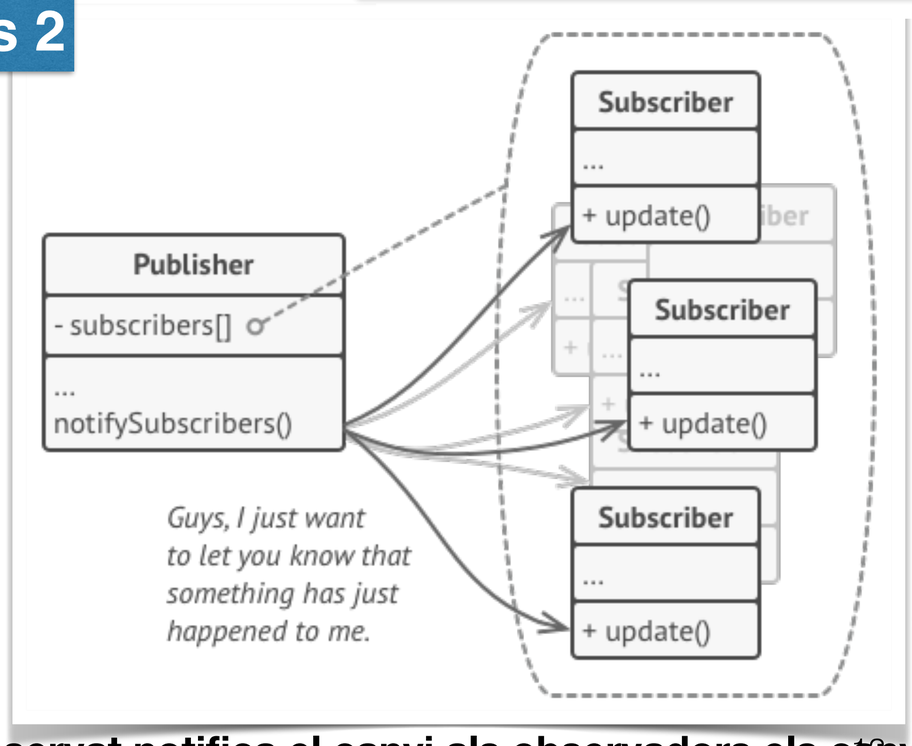
- **Observer/ Observable**

Pas 1



Observers es subscriuen a l'Observat (subject)

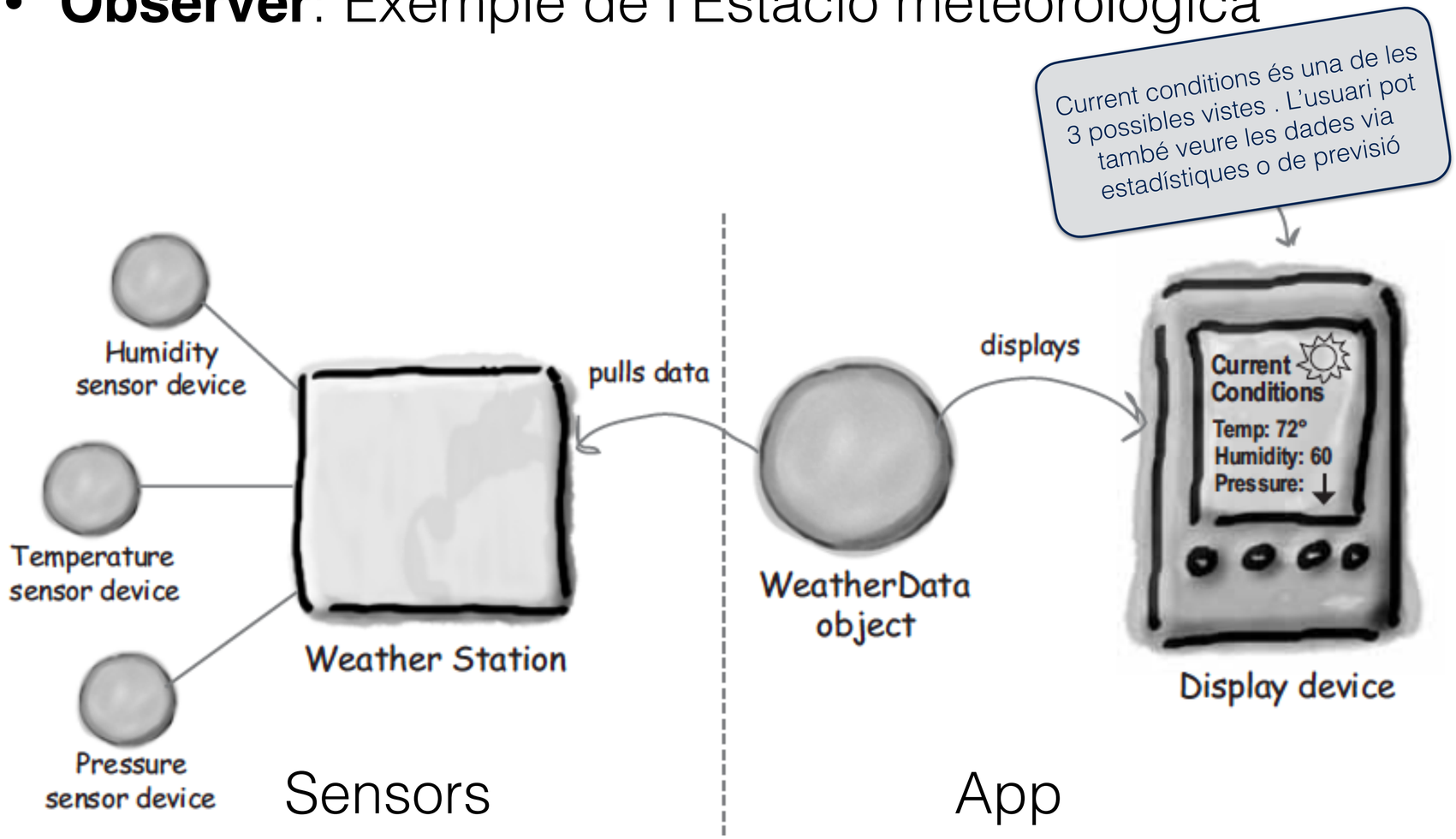
Pas 2



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

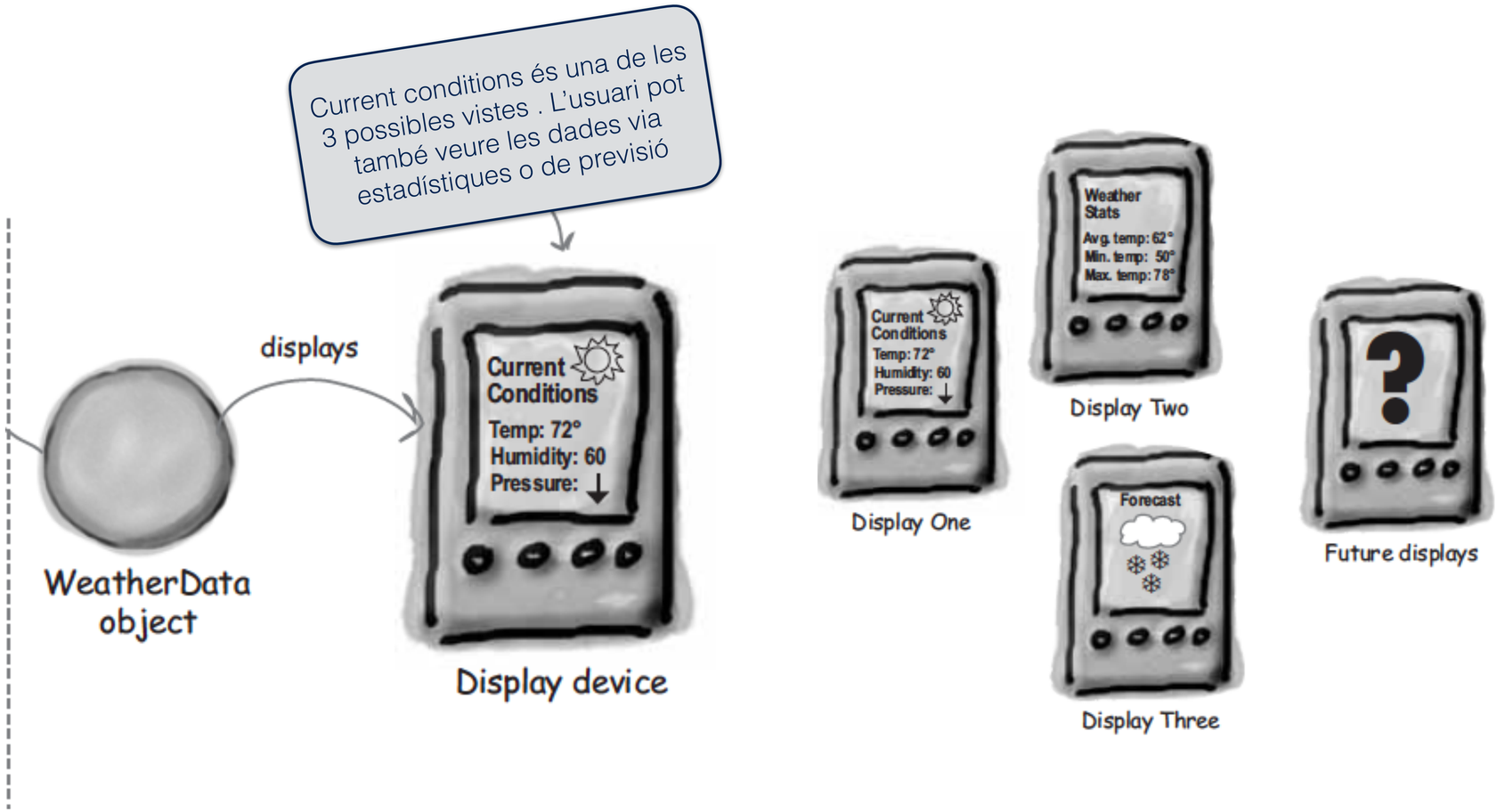
Patró Observer

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



Patró Observer

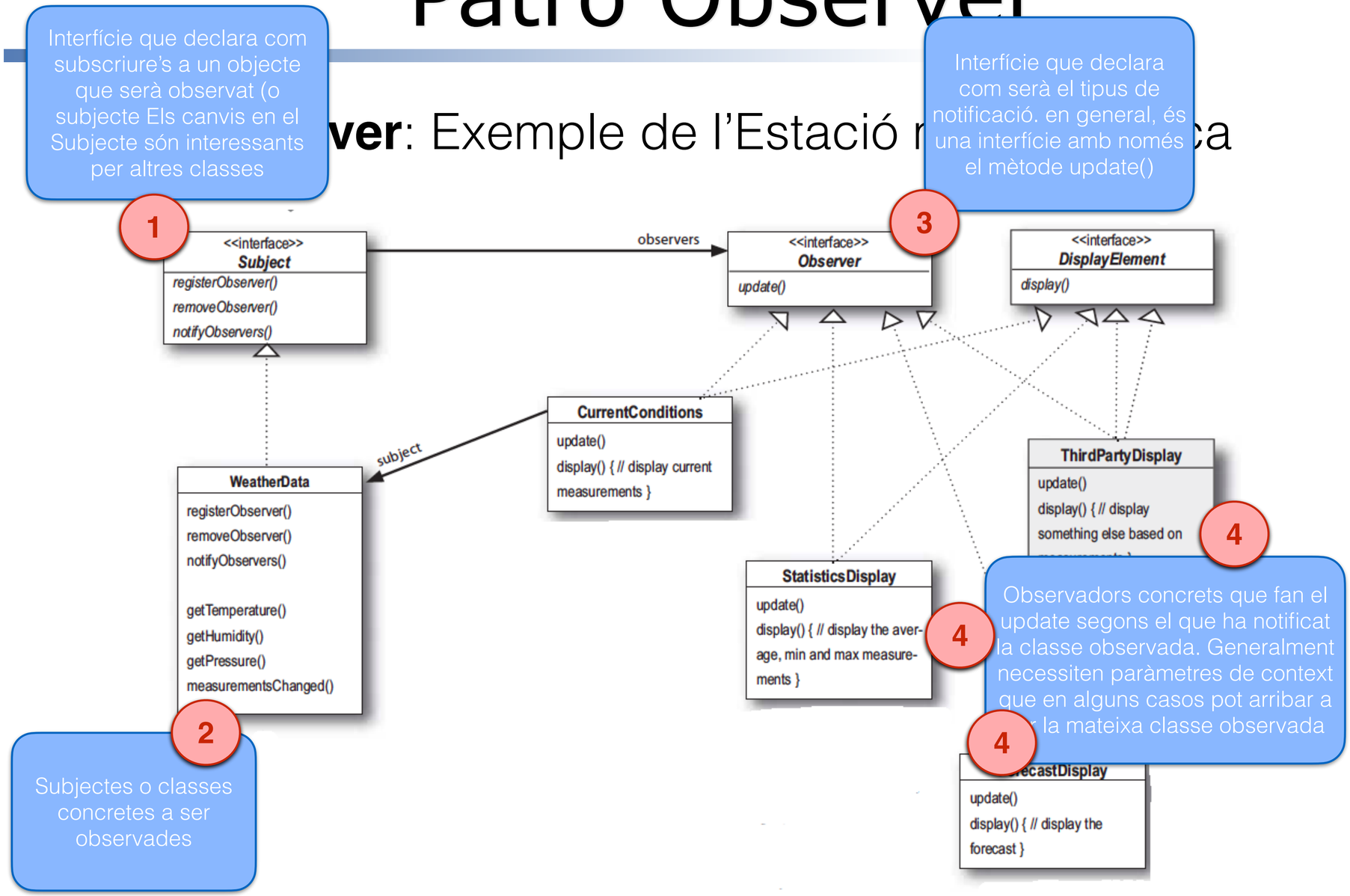
- **Observer**: Exemple de l'Estació meteorològica



App

Patró Observer

ver: Exemple de l'Estació m ca



Patró Observer

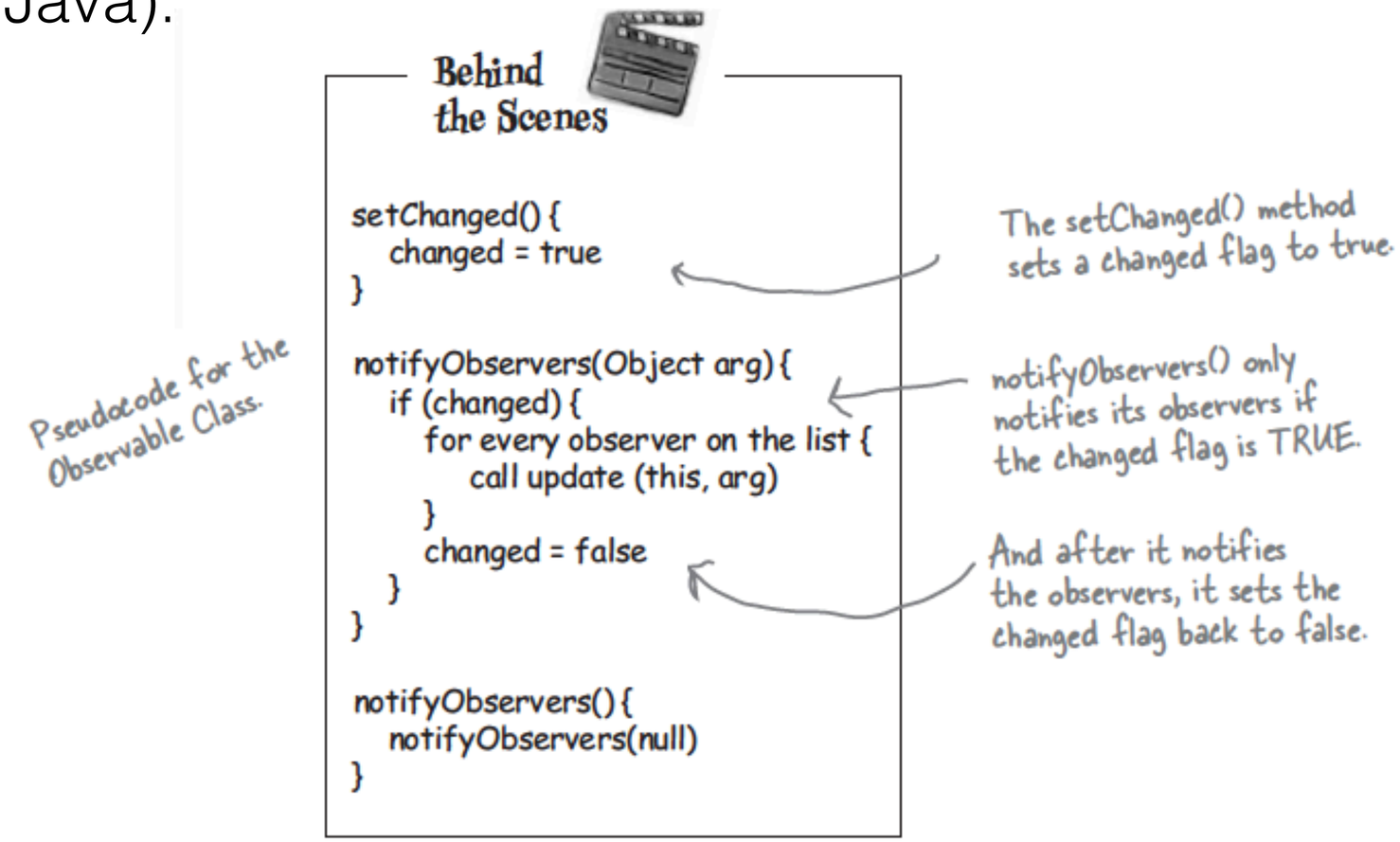
- **Observer**: Exemple de l'Estació meteorològica

- **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 Responsability? 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**