# Observer

*Alebo*

*Don´t miss out when something interesting happens*

# Úloha: Meteorologická stanica (weather station)

Máme WeatherDataObject, ktorý vracia:

- Teplotu
- Vlhkosť vzduchu
- Tlak vzduchu

Úlohou je navrhnúť aplikáciu, ktorá bude informovať o počasí pomocou (zatiaľ) troch rôznych užívateľských rozhraní:

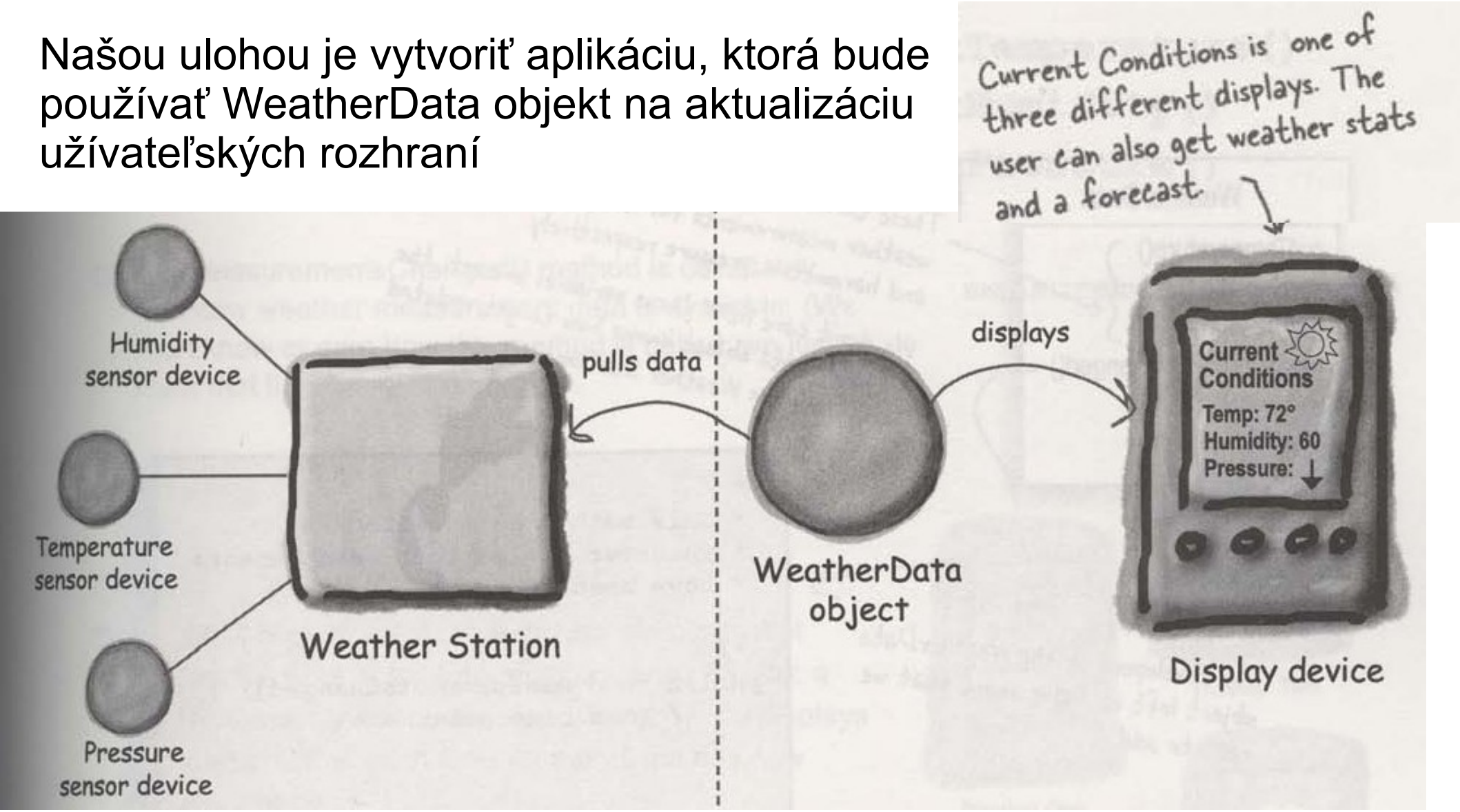
1. Aktuálne počasie
2. Štatistika o počasí za uplynulé obdobie
3. Predpoveď počasia

# Úloha: pokračovanie

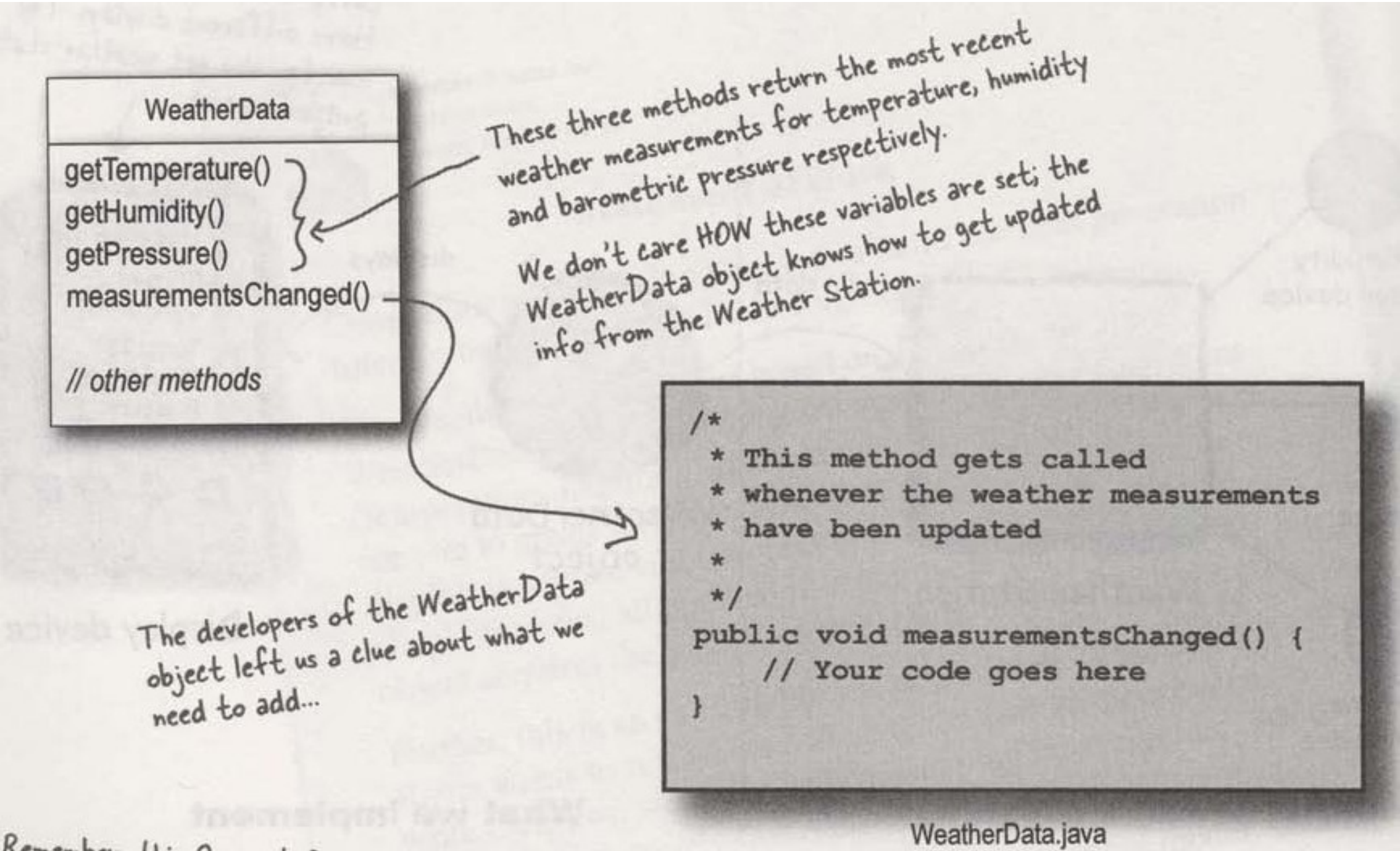
Navrhnite tiež API, ktoré by zákazníci mohli použiť na vytváranie vlastných užívateľských rozhraní

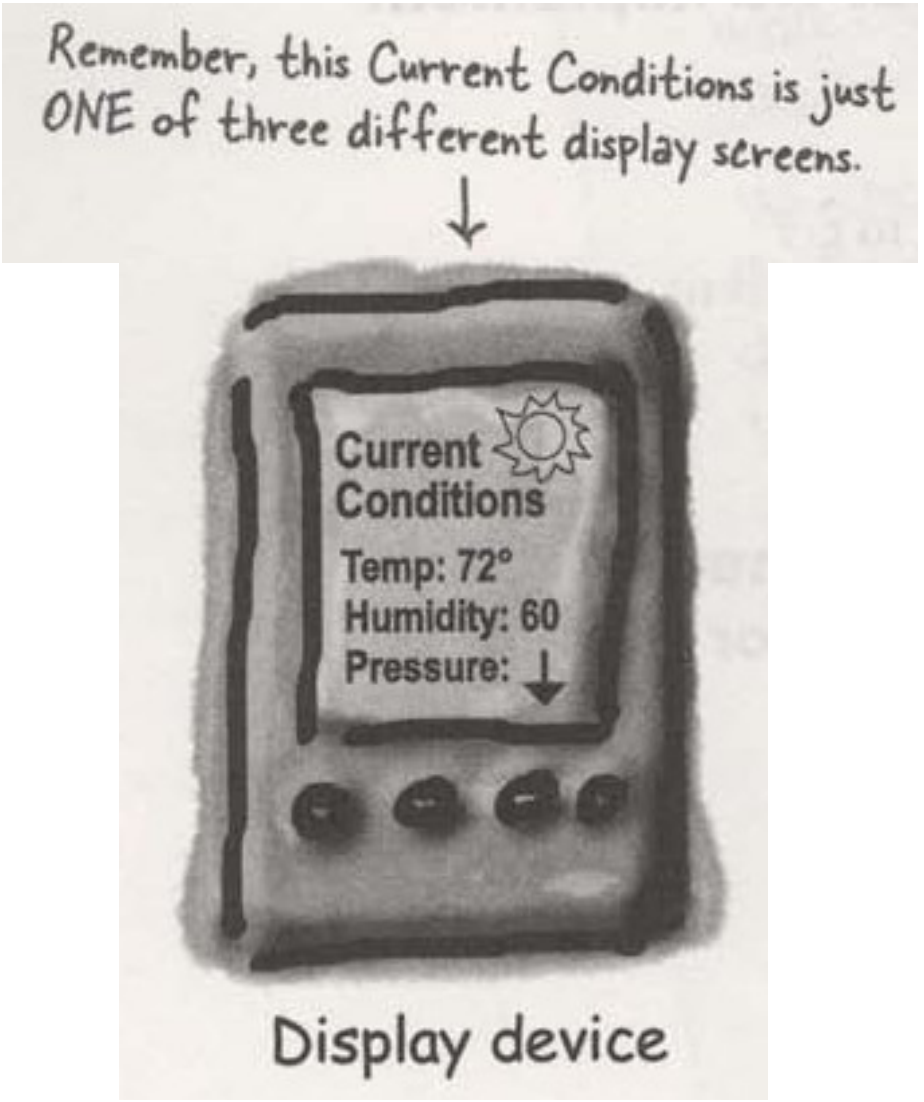
# Meteorologická stanica: prehľad

Našou ulohou je vytvoriť aplikáciu, ktorá bude používať WeatherData objekt na aktualizáciu užívateľských rozhraní

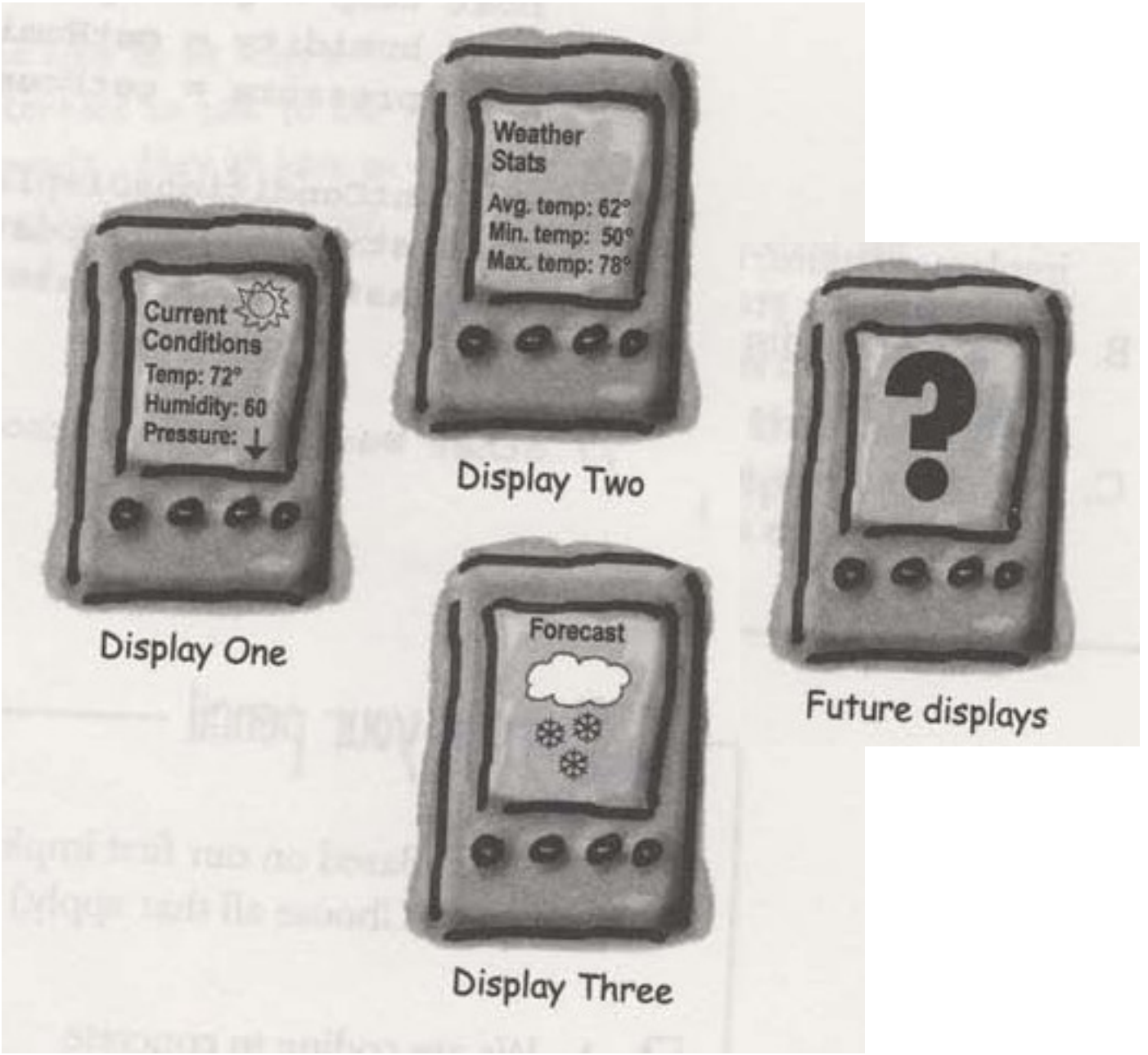


# WeatherData class






Úlohou je teda navrhnúť a naprogramovať metódu MeasurementsChanged()


Current
Conditions
Temp: 72°
Humidity: 60
Pressure:
Display One
Weather
Stats
Avg. temp: 62°
Min. temp: 50°
Max. temp: 78°
Display Two
Forecast
Display Three
Future displays

# Jedno z možných riešení:

```
public class WeatherData {

    // instance variable declarations

    public void measurementsChanged() {

        float temp = getTemperature();
        float humidity = getHumidity();
        float pressure = getPressure();

        currentConditionsDisplay.update(temp, humidity, pressure);
        statisticsDisplay.update(temp, humidity, pressure);
        forecastDisplay.update(temp, humidity, pressure);
    }

    // other WeatherData methods here
}
```

Grab the most recent measuremets by calling the WeatherData's getter methods (already implemented).

Now update the displays...

Call each display element to update its display, passing it the most recent measurements.

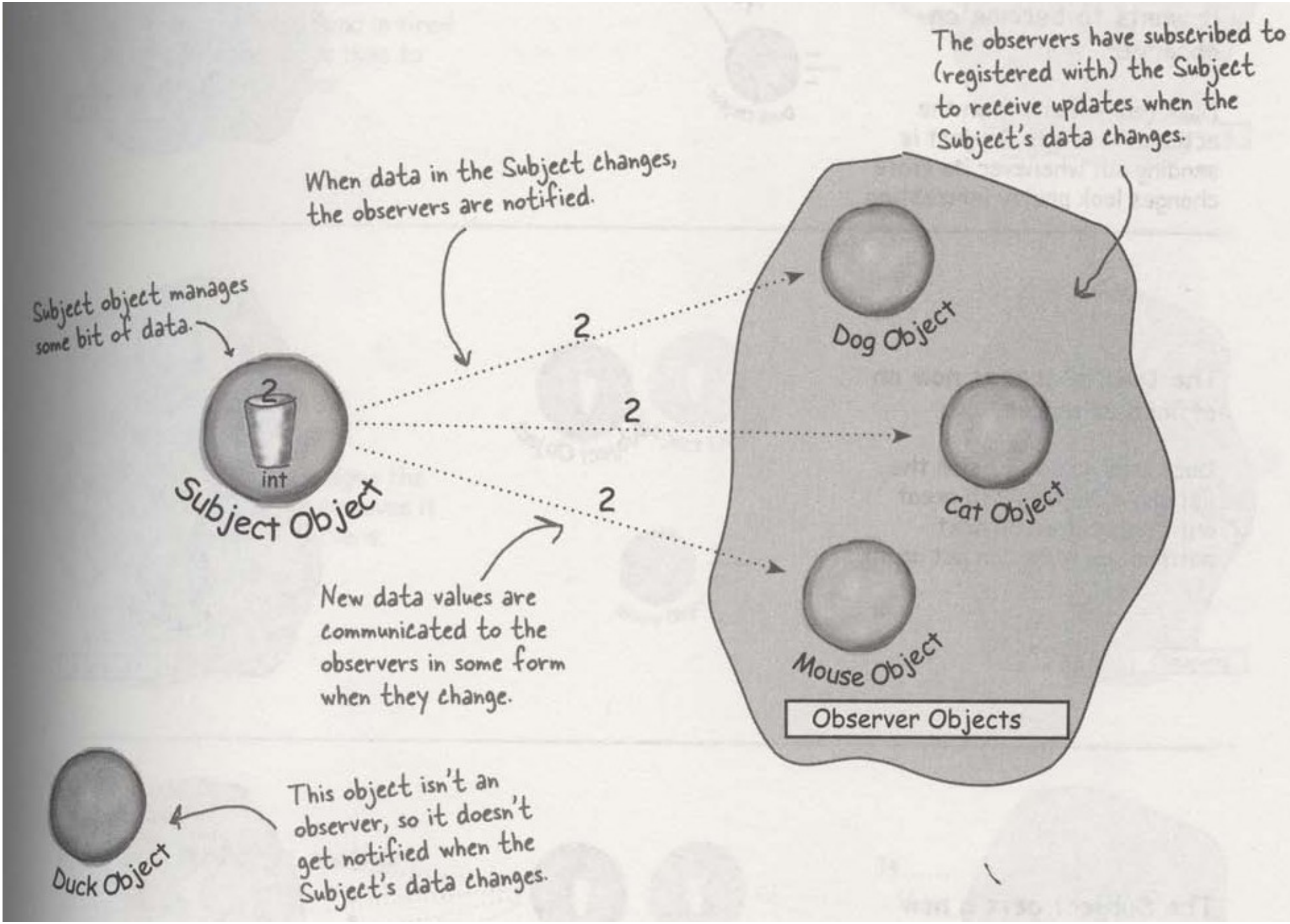
Čo si o tomto riešení myslíte Vy?

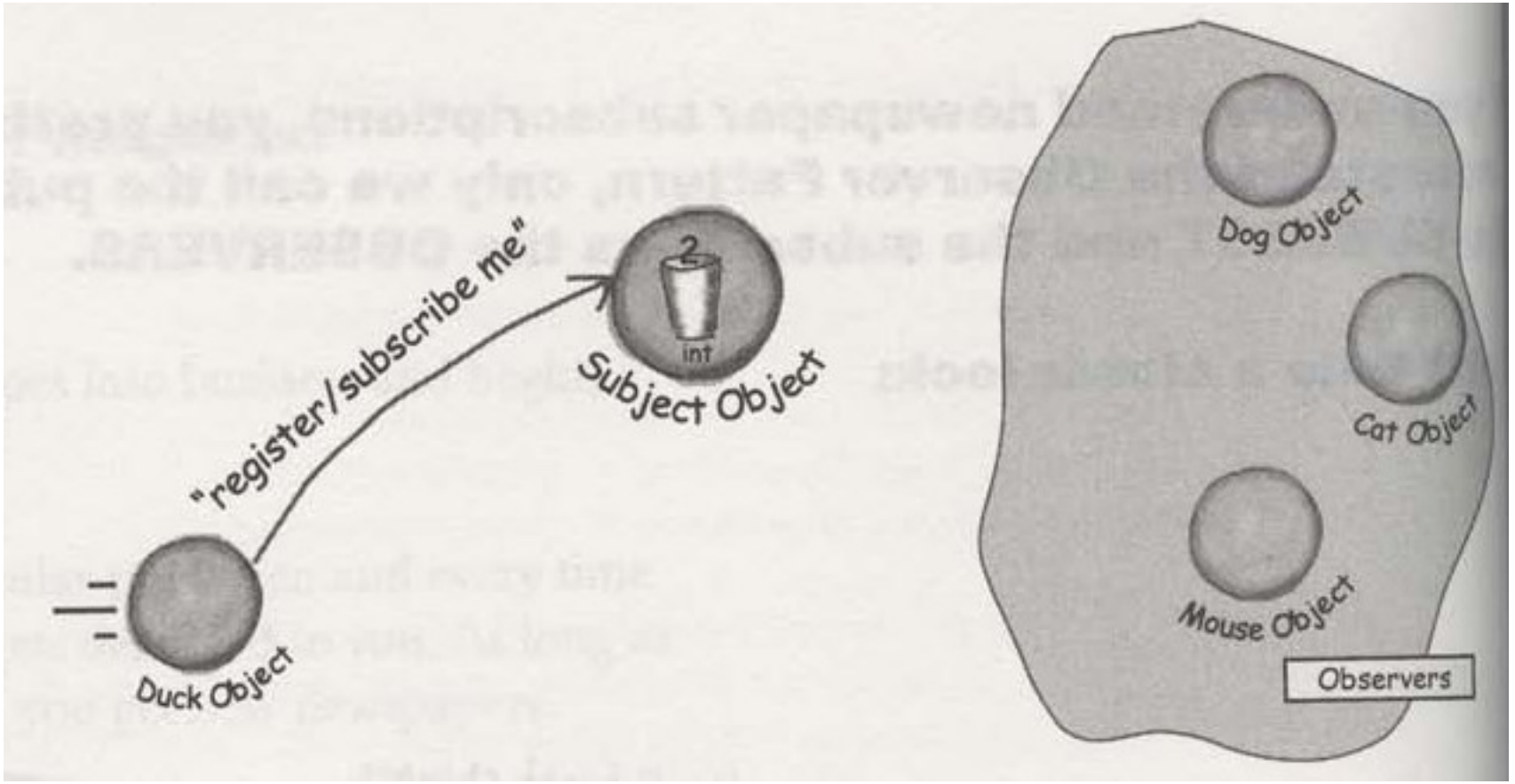
# Nevýhody predchádzajúceho riešenia

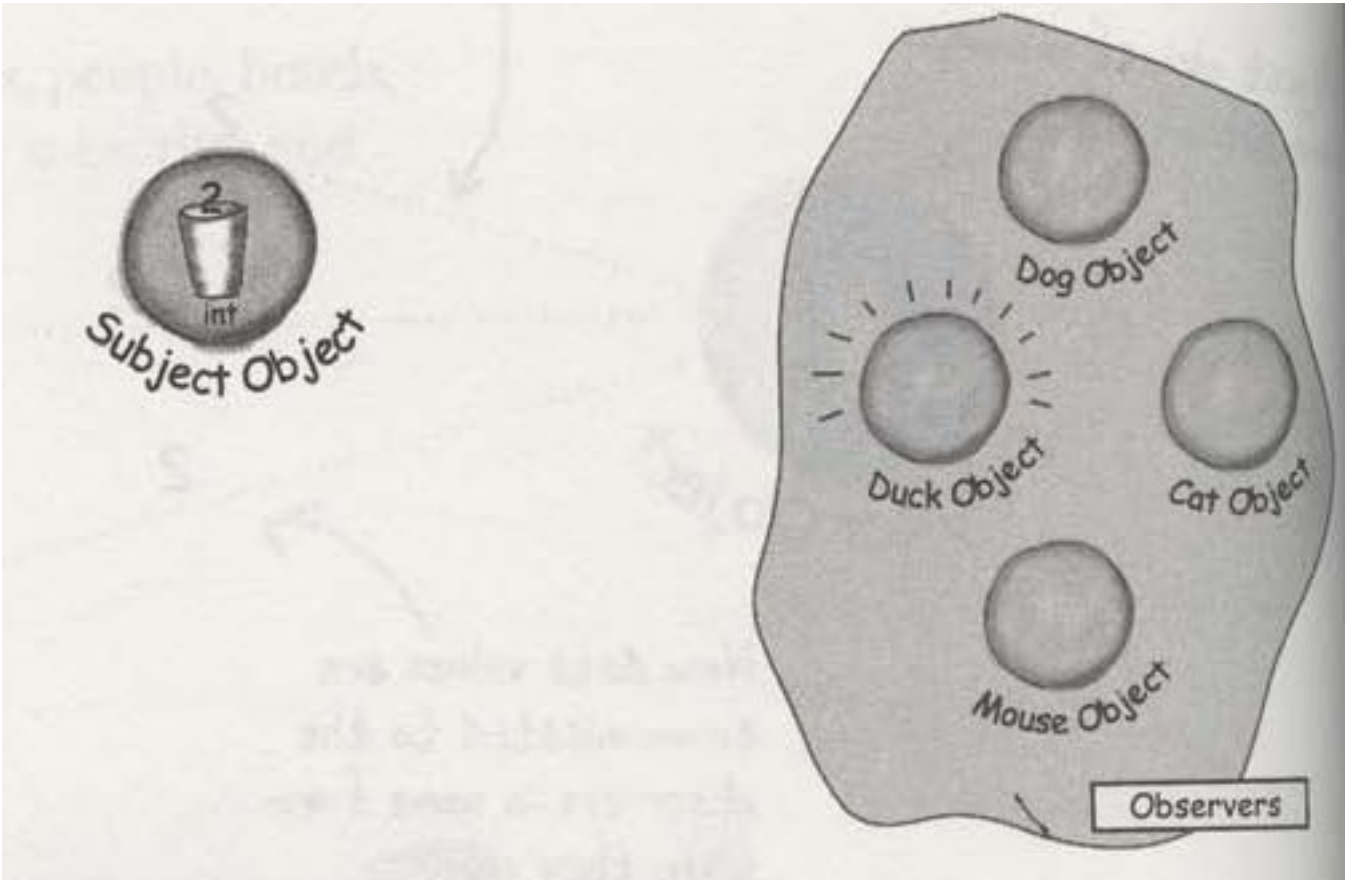
- Metóda MeasurementsChanged volá priamo metódy konkrétnych tried (zvýšené zaťaženie)
- Ak by sme chceli pridať nový display, museli by sme zasahovať do kódu MeasuremantsChanged (a znova ju kompilovať!)
- Nemáme možnosť pridať nový display počas behu aplikácie

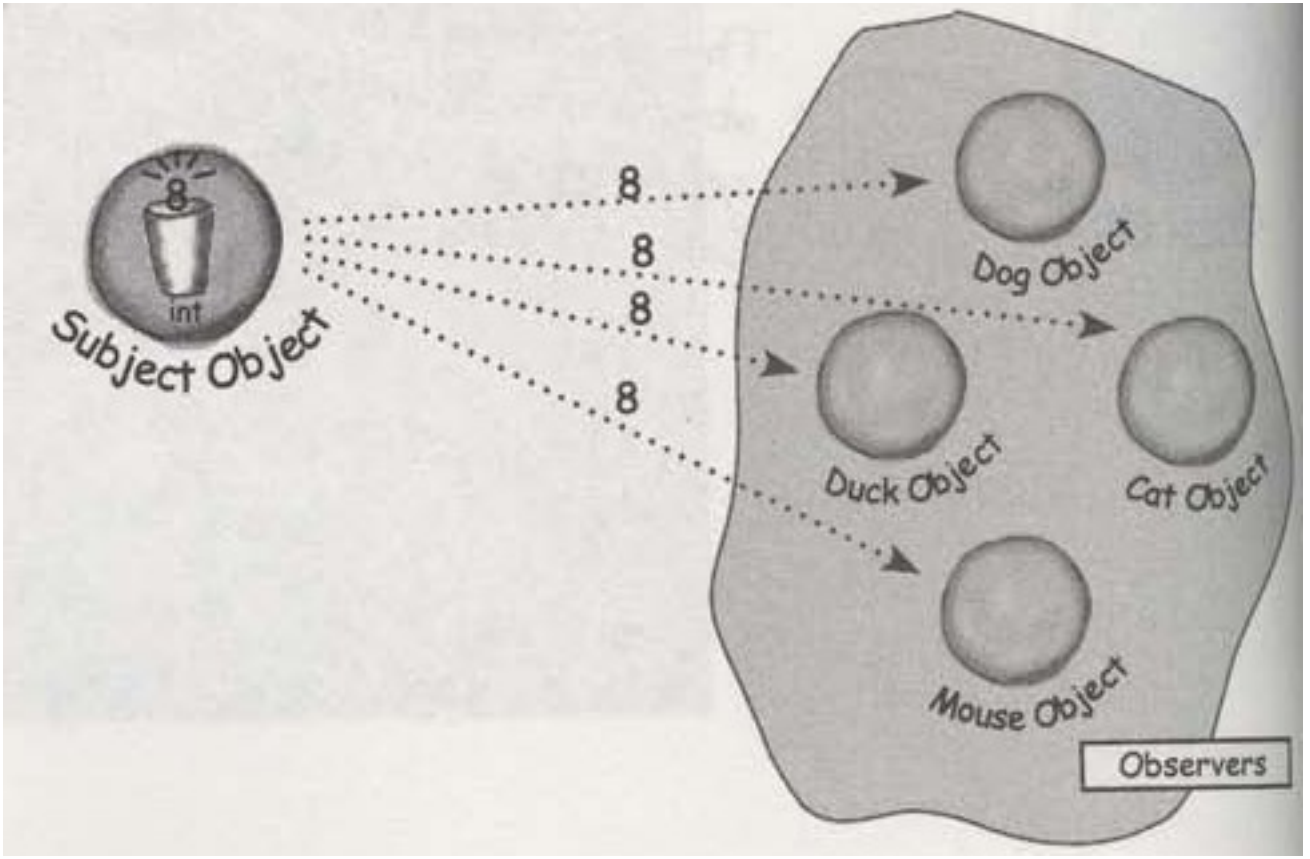
# Ako funguje Observer? Príklad

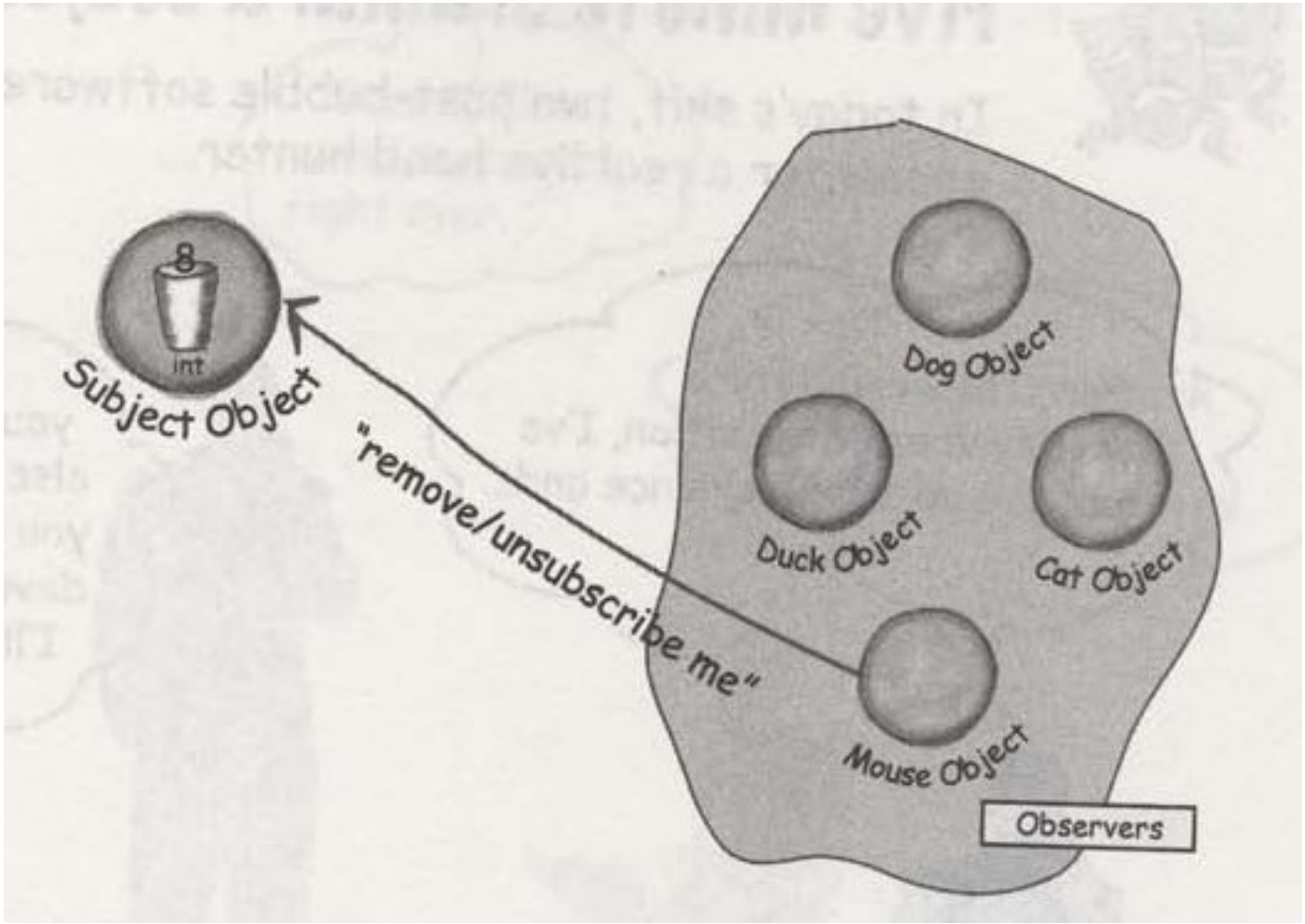
- Vydavateľ začne vydávať časopis
- Zákazník sa prihlási k odberu (zaplatí si predplatné)
- Vždy, keď vydavateľ vydá nové číslo, zašle ho automaticky všetkým predplatiteľom
- Keď zákazník nechce ďalej časopis odoberať, odhlási sa. Vydavateľ ho vyradí zo zoznamu predplatiteľov a prestane mu časopis zasielať
- Ostatní predplatitelia časopis dostávajú aj naďalej

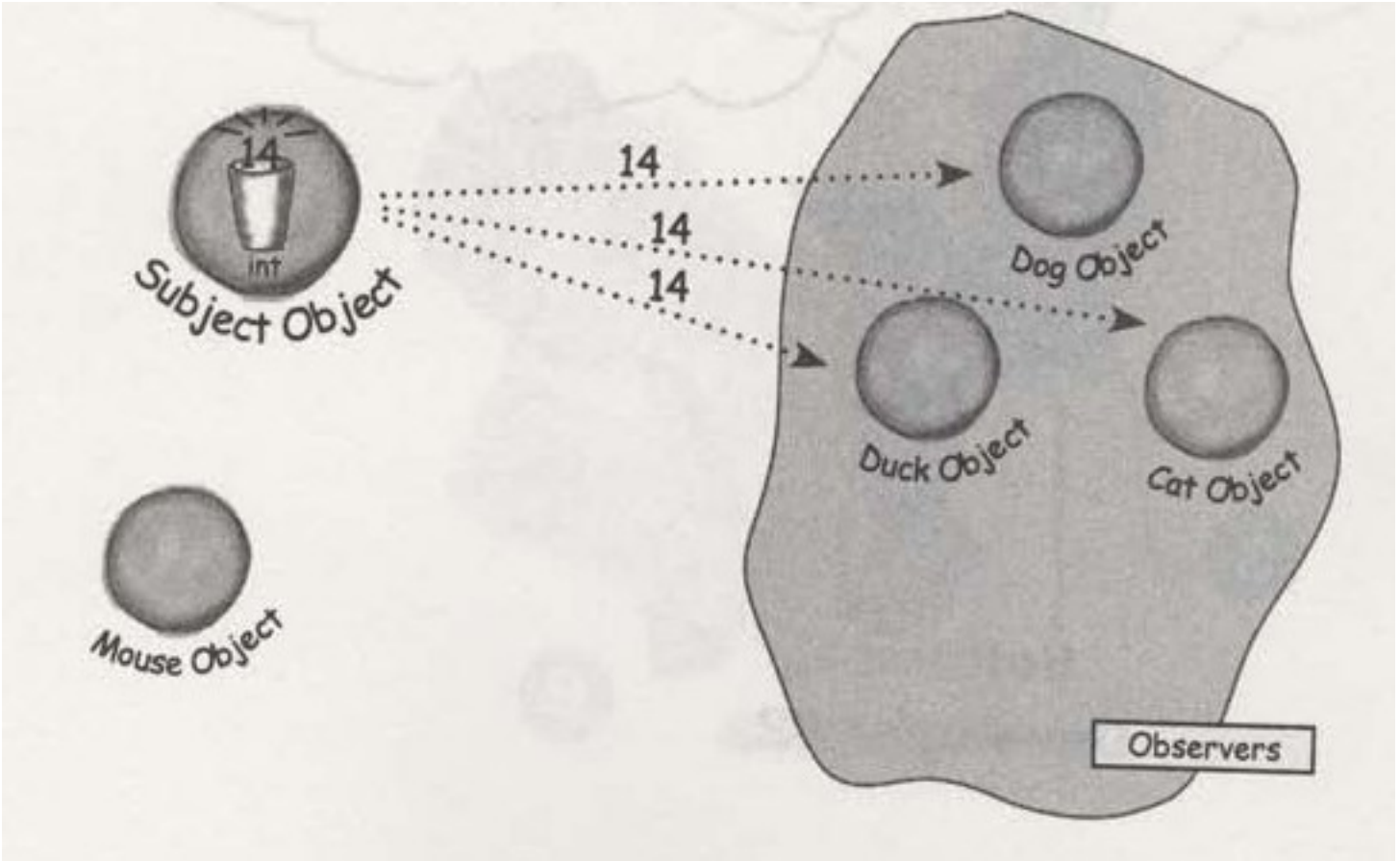

The observers have subscribed to (registered with) the Subject to receive updates when the Subject's data changes.
When data in the Subject changes, the observers are notified.
Subject object manages some bit of data.
2
int
Subject Object
2
2
2
Dog Object
Cat Object
Mouse Object
Observer Objects
New data values are communicated to the observers in some form when they change.
Duck Object
This object isn't an observer, so it doesn't get notified when the Subject's data changes.


"register/subscribe me"
2
int
Subject Object
Duck Object
Dog Object
Cat Object
Mouse Object
Observers


2
int
Subject Object
Dog Object
Duck Object
Cat Object
Mouse Object
Observers

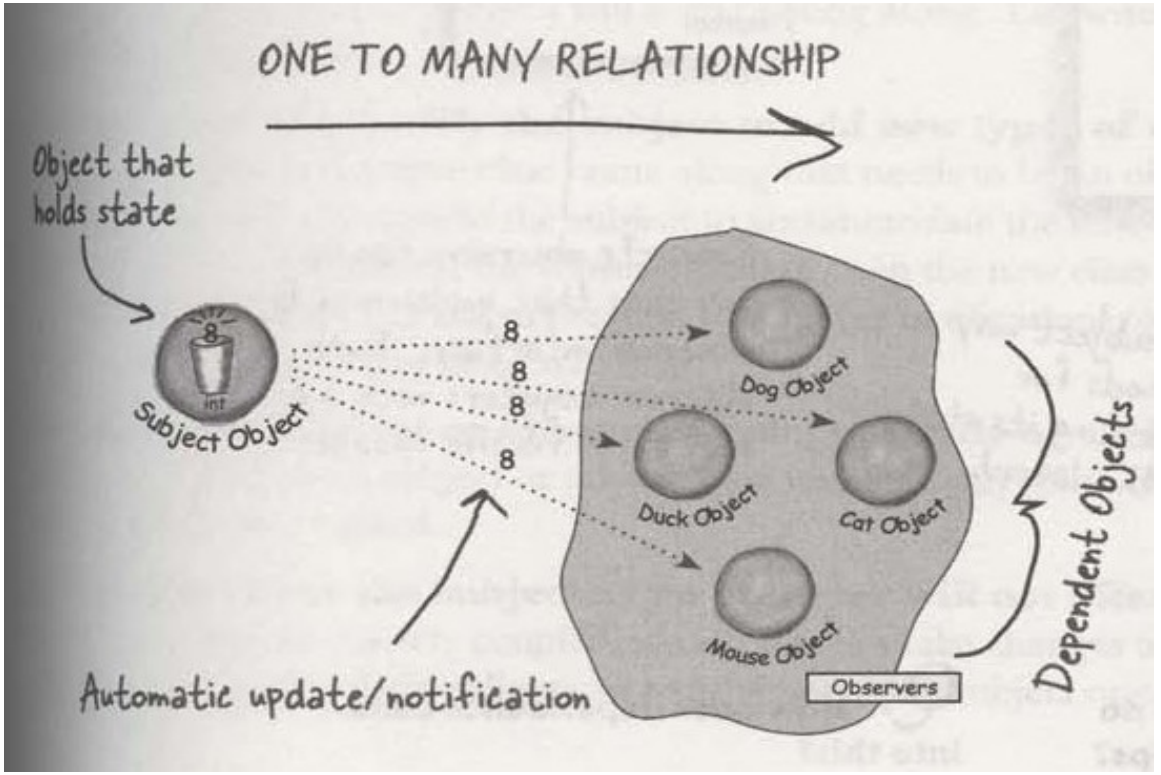

8
int
Subject Object
8
8
8
8
Dog Object
Duck Object
Cat Object
Mouse Object
Observers


8
int
Subject Object
"remove/unsubscribe me"
Dog Object
Duck Object
Cat Object
Mouse Object
Observers

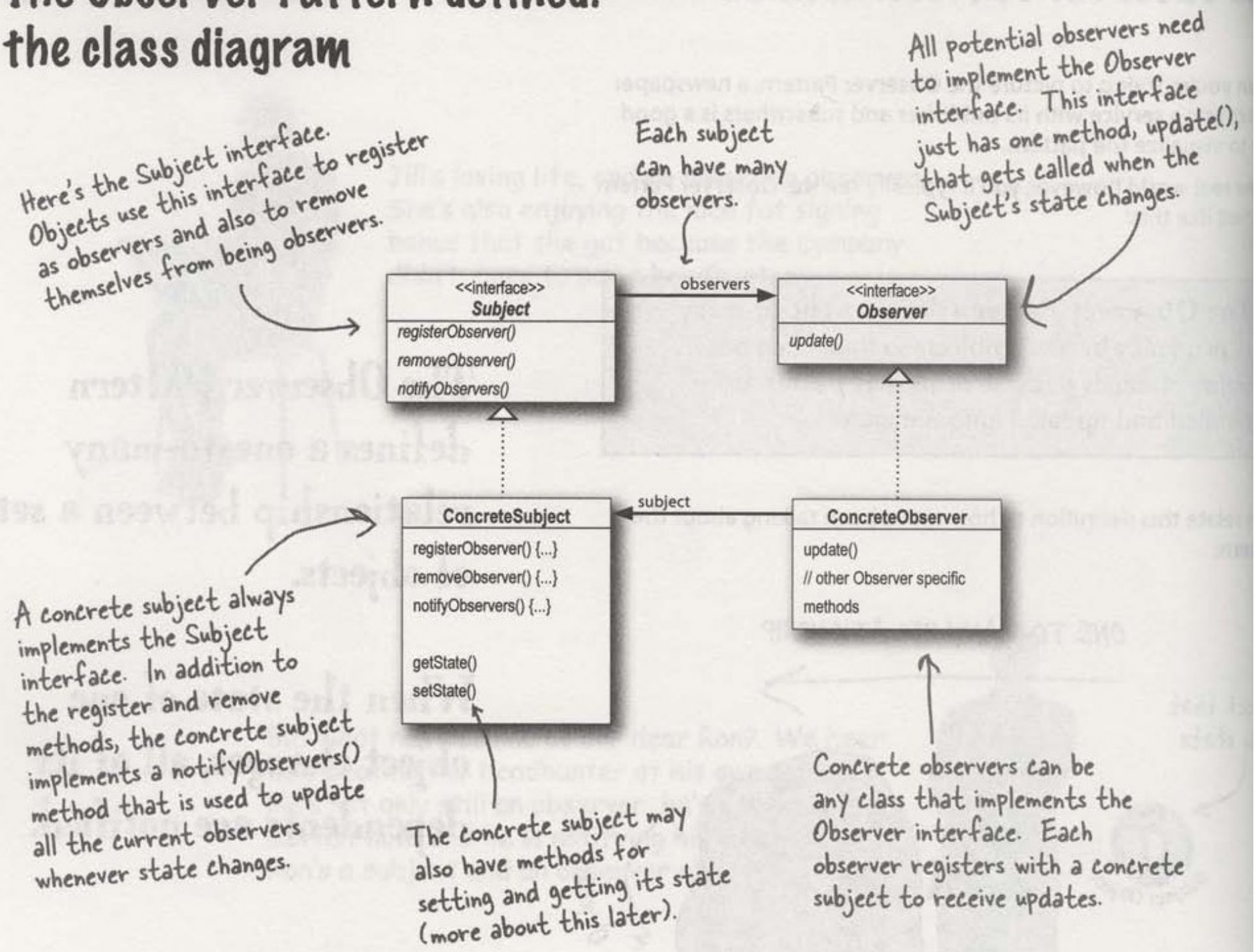

14
int
Subject Object
14
14
14
Dog Object
Duck Object
Cat Object
Mouse Object
Observers

# Vzor Observer: Definícia

Vzor Observer definuje závislosť s kardinalitou 1:N medzi objektmi. Pre túto závislosť platí, že ak objekt na strane 1 zmení svoj stav, sú o tom informované všetky objekty na strane N.



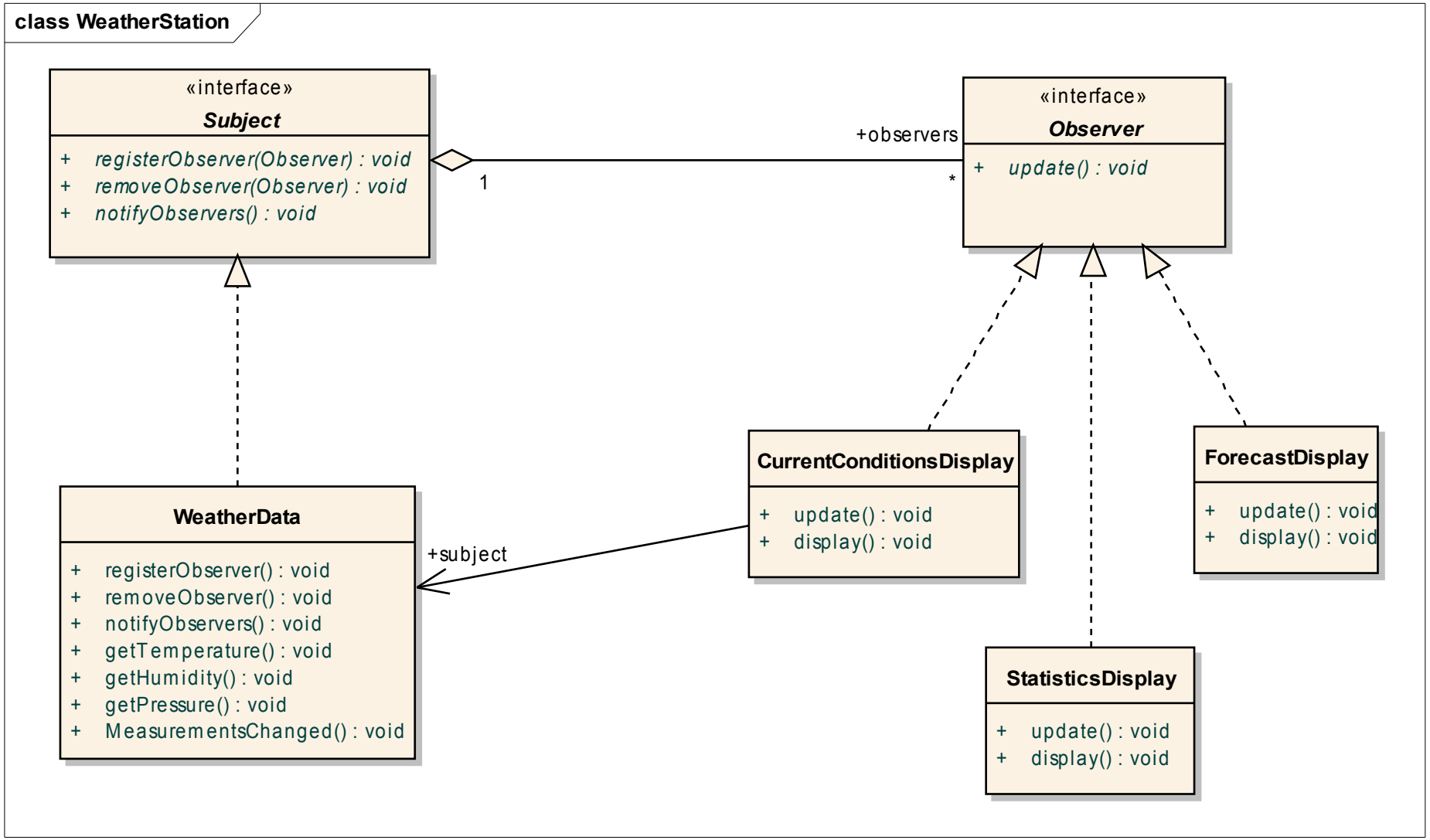
# The Observer Pattern defined: the class diagram



# Observer ako implementácia princípu Loose coupling

- Jediné, čo subject vie o observeroch je to, že implementujú nejaké konkrétne rozhranie
- Kedykoľvek máme možnosť pridať ďalšieho observera (bez zásahu do kódu subjectu)
- Subjekty a observery môžeme znovu použiť (reuse) nezávisle od seba (nízke zaťaženie)
- Zmeny subjektu alebo observera zostávajú izolované (nemajú dôsledky pre iné objekty)
- Rozšírenie aplikácie o ďalšieho observera je veľmi jednoduché

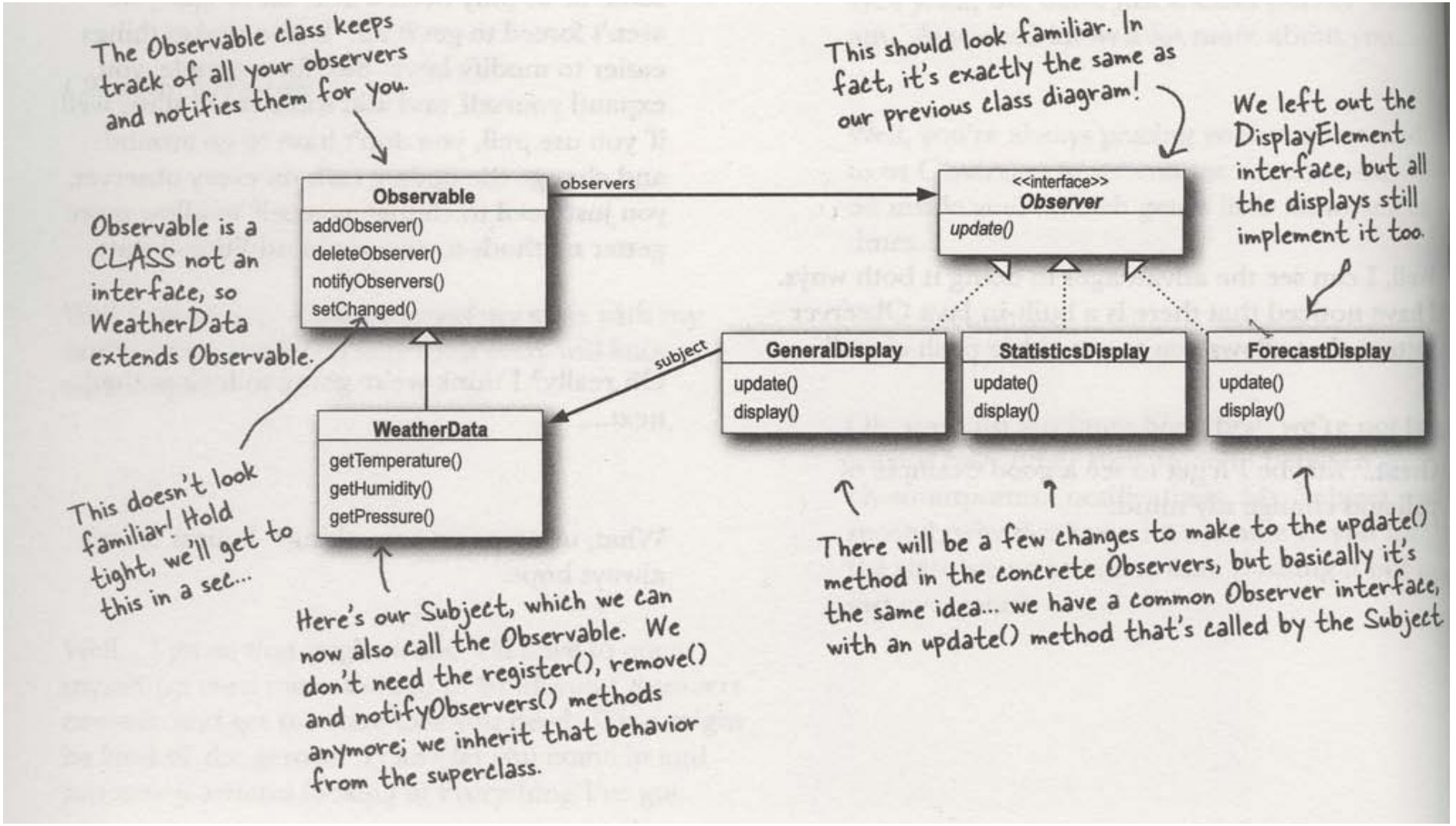
# Návrh WeatherStation



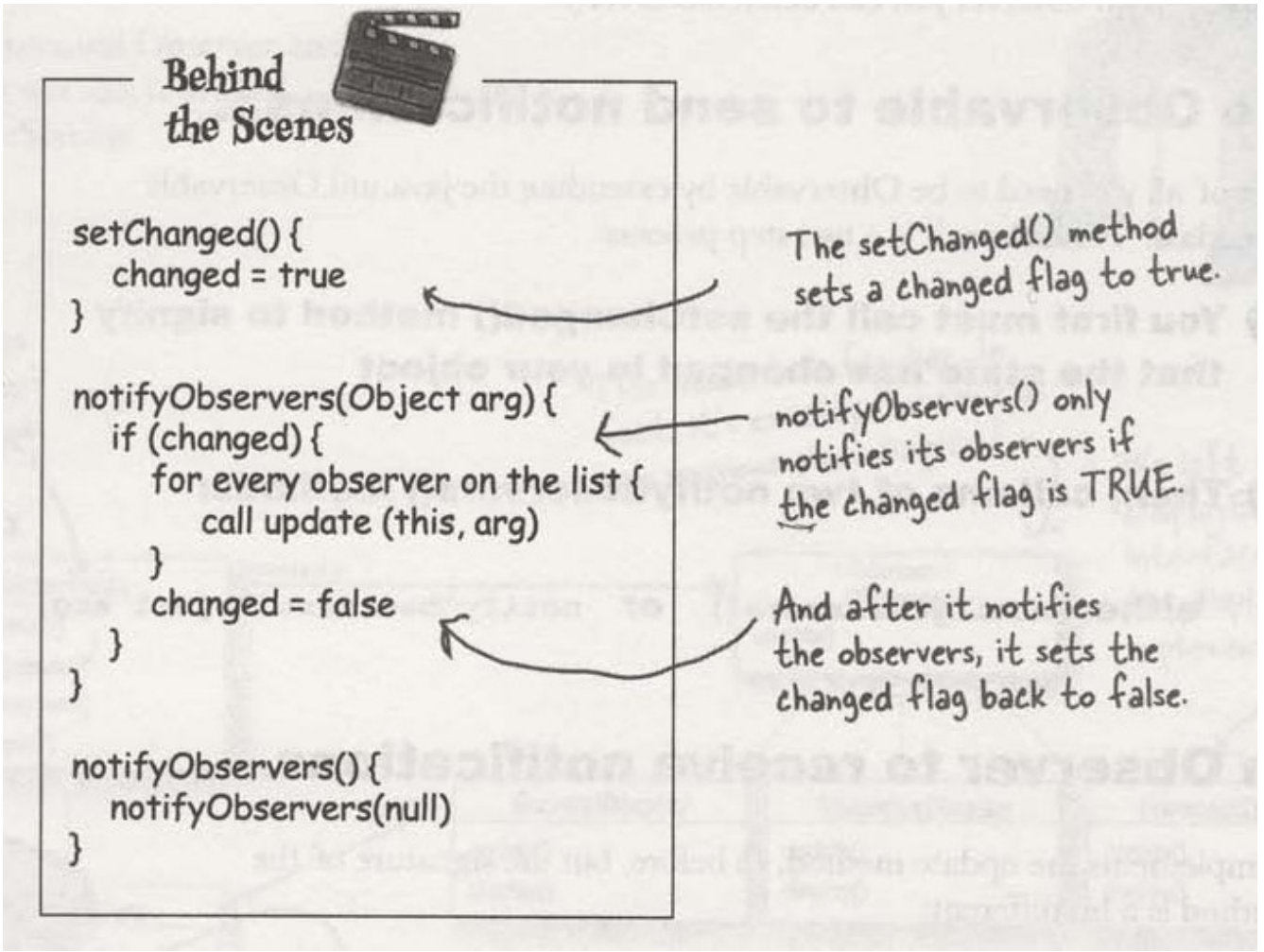
# Použité princípy návrhu

- Loose coupling
- Oddelenie kódu podliehajúceho zmenám od statického kódu

# Vstavaná podpora vzoru Observer v prostredí Java



# SetChanged


Behind the Scenes
setChanged() {
changed = true
}
notifyObservers(Object arg) {
if (changed) {
for every observer on the list {
call update (this, arg)
}
changed = false
}
}
notifyObservers() {
notifyObservers(null)
}
The setChanged() method sets a changed flag to true.
notifyObservers() only notifies its observers if the changed flag is TRUE.
And after it notifies the observers, it sets the changed flag back to false.

# Príklad knižnica

- Prípad užitia: zasielanie noviniek
- Knihovník môže do systému zadať novinky
- Novinky sa zobrazia:
  - užívateľom, ktorí sú práve v knižnici a listujú v katalógu knižnice (sú on-line)
  - užívateľom, ktorí sa prihlásili k odoberaniu noviniek prostredníctvom emailu