Encore un autre problème…

- Construire un système permettant
  - D'accéder à des données fournies par une station météorologique
  - De les exploiter pour les afficher, ou faire des prévisions « en direct », ou piloter une centrale de domotique, etc.
- Contraintes et objectifs
  - La station météo mesure en continu les données météorologiques
    - Ces données doivent être fournies ou accessibles aux « clients » au fur et à mesure (instantanément) des évolutions
  - Le système doit être extensible
    - On doit pouvoir facilement ajouter des composants « clients » (calculs de statistiques, traceur de courbes, historiques d'observation…) sans modification de l'existant

Le modèle « Observateur » (1/6)

- Observateur
- Alias
  - Souscription-diffusion (*publish-subscribe*)
- Intention
  - Définit une relation un-à-plusieurs (1-N) entre des objets de telle sorte que lorsqu'un objet (le « sujet ») change d'état, tous ceux qui en dépendent (les « observateurs ») en soient notifiés et mis à jour « automatiquement »
  - Maintien d'une cohérence d'état entre objets
- Motivation
  - Ne pas introduire de couplage fort entre les classes sujet et observateur
  - Pouvoir attacher et détacher dynamiquement les observateurs
  - *Par exemple, pour afficher différentes représentations d'un jeu de données (des graphiques extraits d'un tableur par exemple)*

Le modèle « Observateur » (2/6)

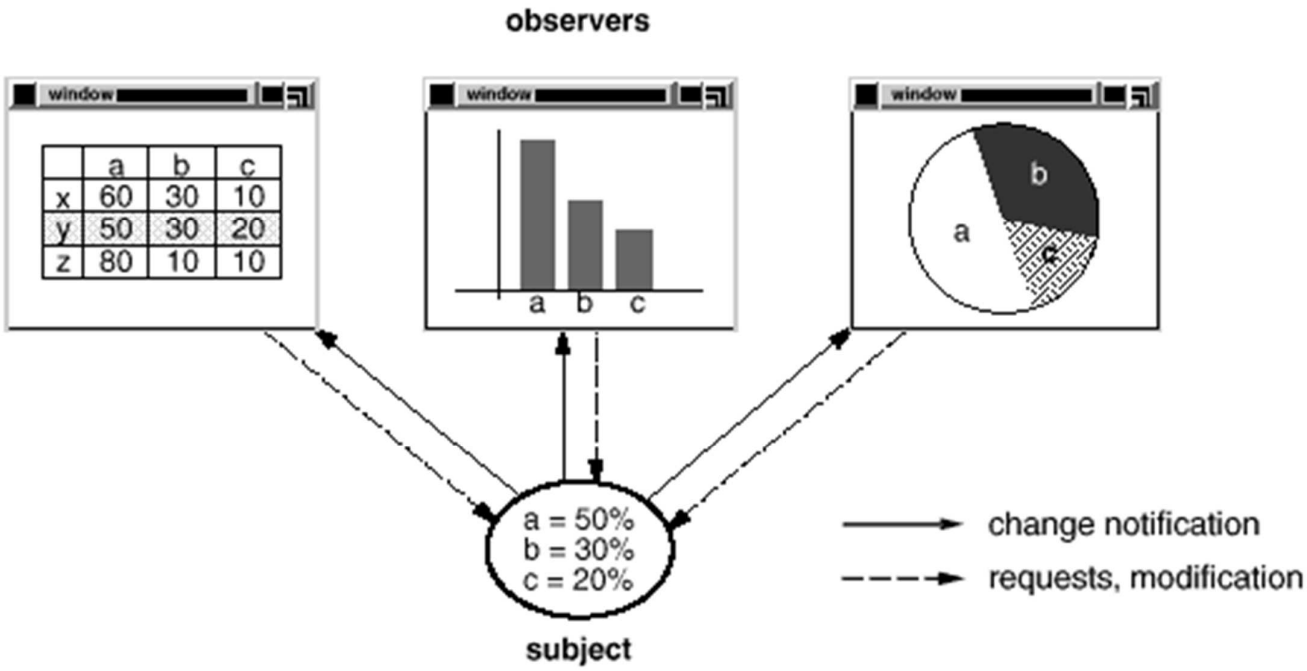


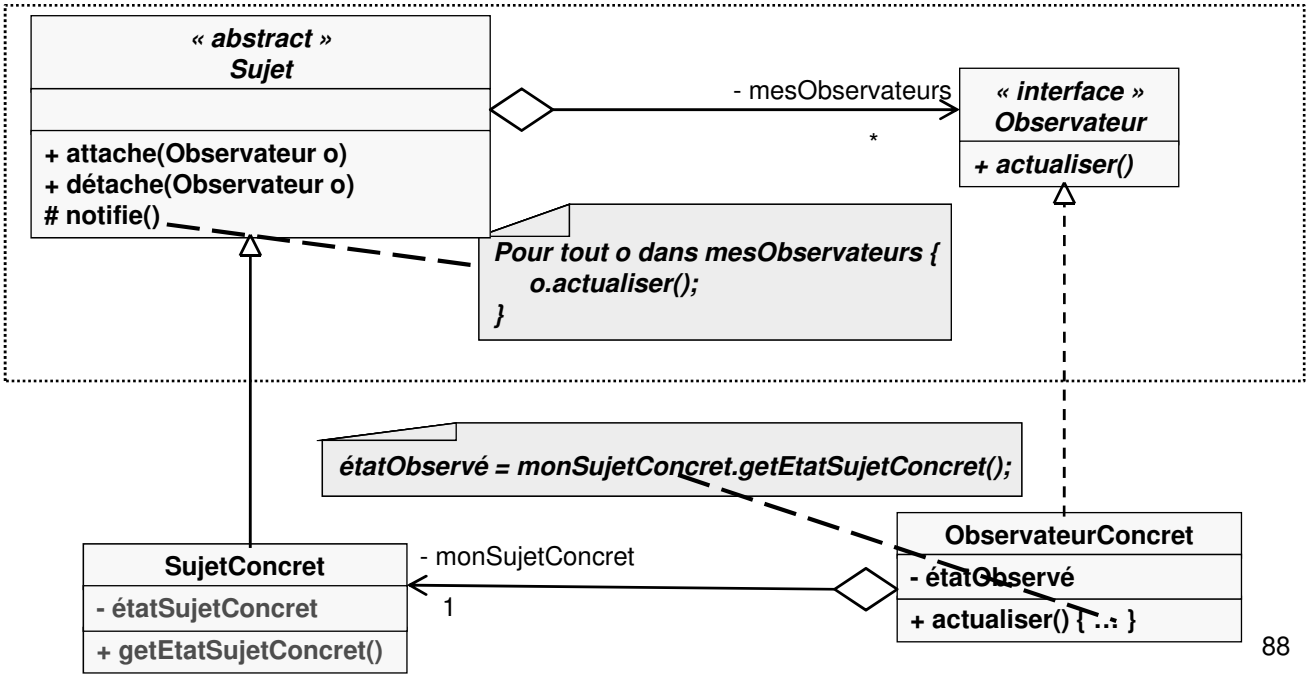
Figure extraite de
« Design Patterns, Elements of Reusable Object-Oriented Software »,
E. Gamma, R. Helm, R. Johnson & J. Vlissides, 1995

Le modèle « Observateur » (3/6)

- Participants
  - *Sujet* : classe abstraite en association avec *Observateur*
    - Offre une interface pour attacher et détacher les observateurs
    - Implémente la notification (protocole de diffusion)
    - Peut aussi être une interface ou une classe concrète
  - *Observateur* : interface qui spécifie la réception de la notification
  - *SujetConcret* : mémorise l'état et envoie la notification
    - Offre une méthode d'acquisition d'état aux observateurs (mode *pull*)
    - Un objet *SujetConcret* a la référence de ses *ObservateurConcrets*
  - *ObservateurConcret* : gère la référence au sujet concret et, éventuellement, mémorise l'état en cohérence avec le sujet
    - Sollicite le sujet pour acquérir l'état (en mode *pull*)

Le modèle « Observateur » (4/6)

- Structure (mode « pull »)



Le modèle « Observateur » (5/6)

- Conséquences
  - On peut modifier sujets et observateurs indépendamment
    - Pas de lien de la classe SujetConcret vers la classe ObservateurConcret
    - On peut ajouter de nouveaux observateurs sans avoir à modifier le sujet
      - Initialement, on a identifié que les observateurs pouvaient varier
  - Communication possible en mode *push*
    - Mais interface de notification spécifique (côté observateur)
  - Un observateur pourrait observer plusieurs sujets (relation N-1 possible)
  - D'autres modèles sont possibles en termes de synchronisation (*i.e.* évènementiel) et d'interaction

Le modèle « Observateur » (6/6)

- Implémentation
  - Il existe une implémentation native en Java
    - Classe java.util.Observable et interface java.util.Observer
    - Deprecated in Java 9 (=> *Listeners*)
- Utilisations remarquables
  - Dans la mise en œuvre des IHM
    - En particulier dans le modèle MVC
- Modèles apparentés
  - Dans certains cas, Observateur peut supporter la mise en œuvre de Médiateur