GÉNIE LOGICIEL ORIENTÉ OBJET (GLO-2004)
ANALYSE ET CONCEPTION DES SYSTÈMES ORIENTÉS OBJETS (IFT-2007)

Hiver 2016

Module 15 - Patrons de conception (partie 1)

Martin.Savoie@ift.ulaval.ca

Bachelier Génie logiciel, Chargé de cours,
département d'informatique et de génie logiciel

UNIVERSITÉ LAVAL

Questions sur le projet

- Visualisation des sports et stratégies?
- Mode d'édition image/image Vs temps réel
  - Le mode ne doit pas être fixe
- D'autres questions?
- Il ne faut pas hésiter du tout!

Examens

- Très content des moyennes
  - Même si trop d'examens en même temps
  - Est-ce encore le cas?
- Voir les copies?
  - Courriel + rendez-vous -> la semaine prochaine

Les démonstrations

- Pour IFT-2007
  - Mardi 29 novembre 19h00 à ...
  - Mercredi 30 novembre 19h00 à ...
- Pour GLO-2004
  - Jeudi 1er décembre de 16h30 à 21h30
- Cela va être sur mon ordinateur
  - Le jar executable doit être fonctionnel absolument
- Liste d'étape à suivre le soir même

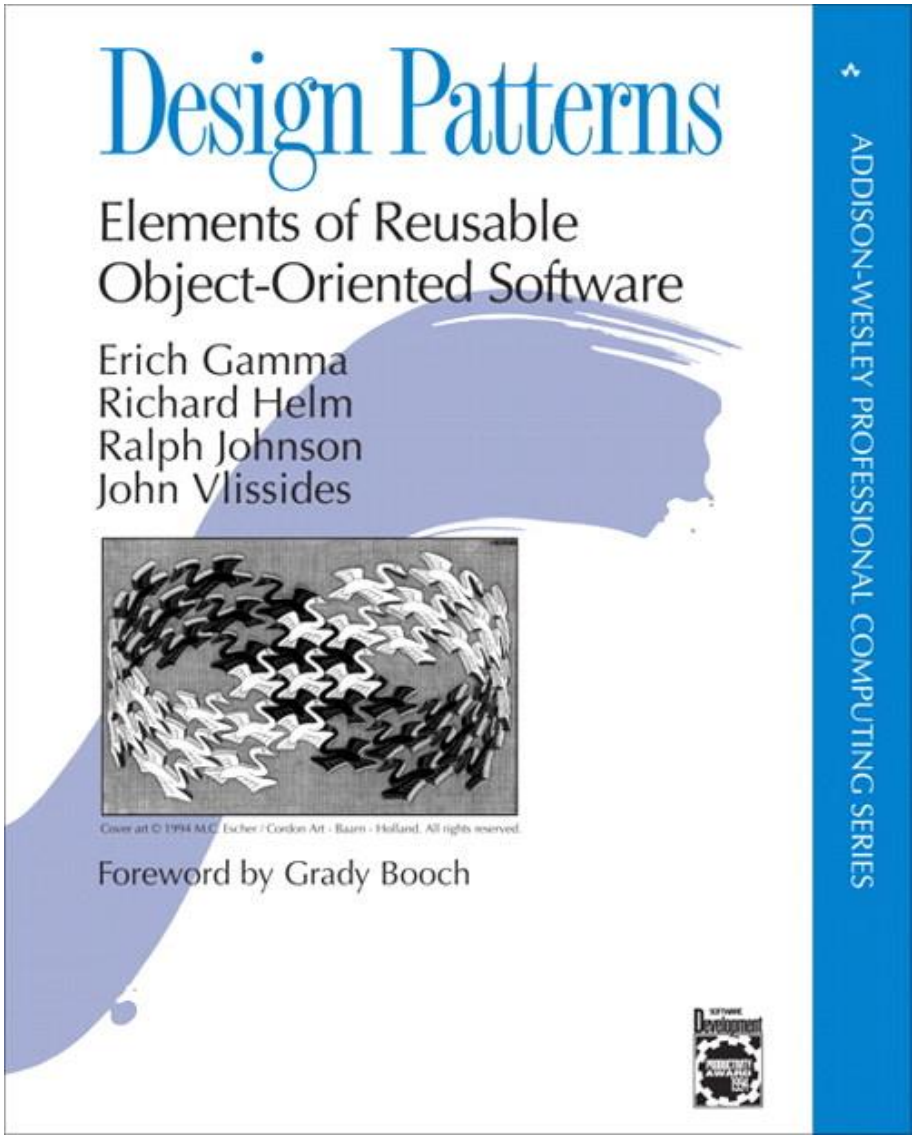
Qu’est-ce qui distingue un bon design d’un mauvais design?

Grands principes de conception orientée objet (GRASP) et Patrons de conception (GoF)

- **Grands principes de conception orientée-objet**
  - Larman les appelle les GRASP (General Responsibility Assignment Software Patterns)… mais ce ne sont pas vraiment des « patrons » (au sens classique du terme) à part « Contrôleur »

- **Patrons de conception / patterns**
  - Exemples de constructions/assemblages d'objets classiques que l'on retrouve fréquemment dans les design d'analystes expérimentés (constitue un répertoire d'idées)
  - Décrits dans le livre « Design patterns » de Gamma et al (« Gang of four »)

À la bibliothèque en version papier et électronique



Comment sont–ils utilisés en pratique?

- On planche sur un design et puis on se dit: « ah oui, j'ai déjà vu un exemple avec quelque chose de semblable »

Design patterns

- Nous verrons plusieurs patrons GoF qui sont également présentés dans le livre de Larman (chap. 26 et 26 en anglais; 23 et 30 en français)
- Vous verrez les autres patrons GoF dans le cours Architecture logicielle (GLO-4003 / IFT-6003) – (Ce cours est obligatoire dans certains programmes mais « optionnel » dans d'autres. En réalité c'est un incontournable)

Constat de Larman

- Les patrons GoF utilisent les GRASP
- Normal, puisque nous prétendons que ce sont des grands principes universels!

Singleton (GoF)

Nouveau!

- **Contexte/Problème:** une et une seule instance d'une classe est autorisée (un **singleton**). Les classes utilisatrices ont besoin d'un point d'accès global et unique à cette instance.

- **Solution:** Définir une méthode statique de la classe qui retourne le singleton.
  - Lors du premier appel à cette méthode statique, l'instance de la classe est créée, stockée dans un attribut statique, et retournée à l'appelant
  - Lors des appels subséquents à cette méthode, on retourne toujours le même objet stocké dans l'attribut statique
  - Ainsi, tous les utilisateurs de la classe travailleront avec la même instance.

Rappel - Vocabulaire important

- **Méthode statique** = méthode de classe
  - "Although most methods execute in response to method calls on specific objects, this is not always the case. Sometimes a method performs a task that does not depend on the contents of any object. Such a method applies to the class in which it is declared as a whole and is known as a static method or a class method" [Java for programmers, Deitel, section 6.3]
  - Ex: Math.sqrt(4)
- **Attribut statique** = attribut de classe

Attribut statique

```
class Patate
{
    public int nbPatates = 0;
    public Patate()
    {
        nbPatates++;
        System.out.println(nbPatates);
    }
}
```

Que verra-t-on d'afficher à l'écran si on fait 100 fois « new Patate() » ?

Attribut statique

```
class Patate
{
    public int nbPatates = 0;
    public Patate()
    {
        nbPatates++;
        System.out.println(nbPatates);
    }
}
```

Oui! Il y a un attribut nbPatates par objet de type Patate!

```
class Patate
{
    public static int nbPatates = 0;
    public Patate()
    {
        nbPatates++;
        System.out.println(nbPatates);
    }
}
```

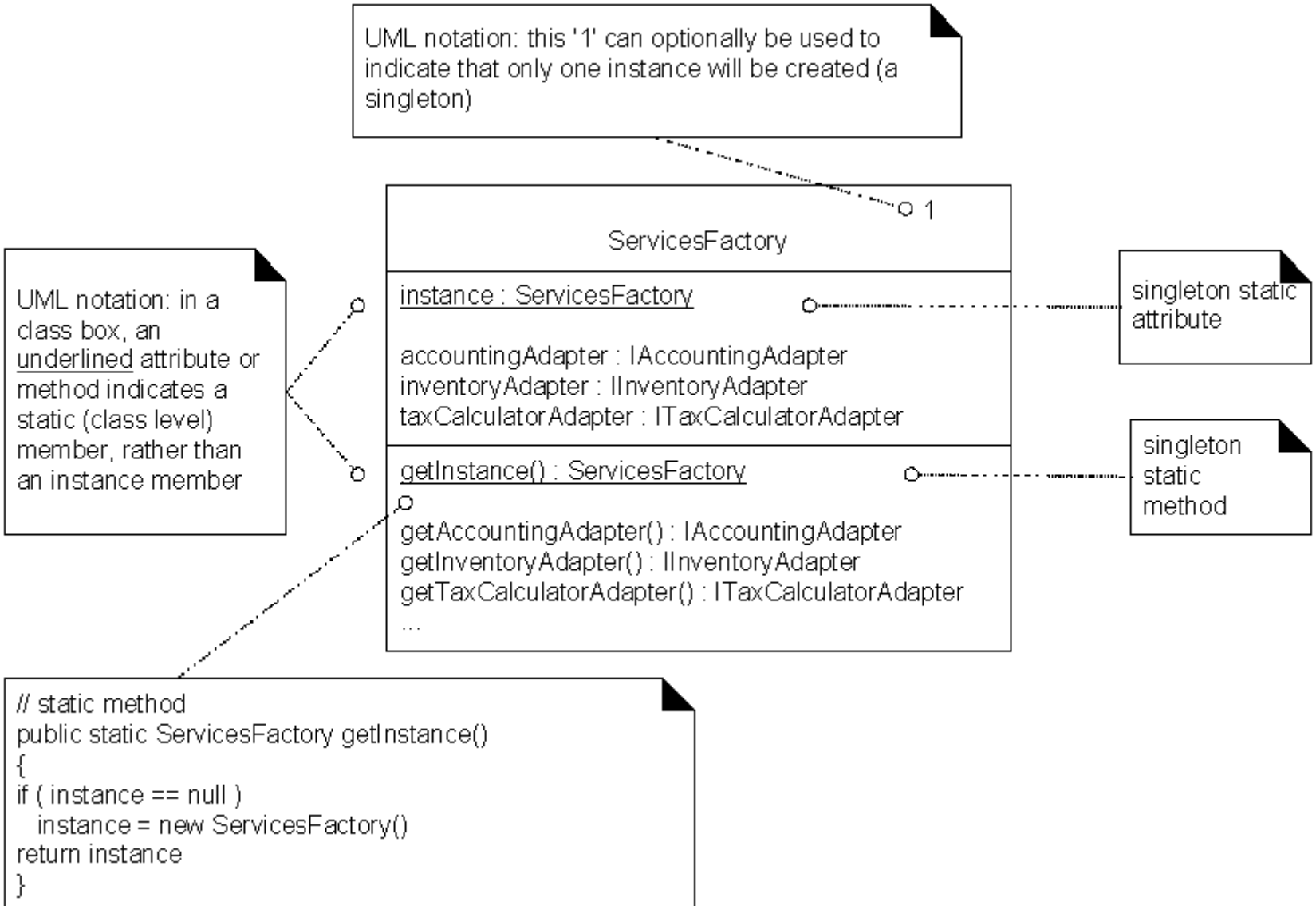
Solution:
Avec le mot clef static, nbPatates est partagé entre tous les objets patates...

Méthode statique

```
class Patate
{
    public static int nbPatates = 0;
    ...
    public static int getNbPatates()
    {
        return nbPatates;
    }
}
```

Les **méthodes de classe** (statique) **ne** définissent **pas** le comportement des instances (objets) de la classe.
Les **méthodes de classe** (statique) **définissent le comportement de la classe**.

Exemple Singleton - NexGen



Réflexion Singleton

- Ainsi donc, si j'utilise un Singleton, tout le monde partage la même instance (sans que j'aie eu à passer une référence sur l'objet à tous les utilisateurs).

- Si à un endroit j'appelle une méthode de mon singleton qui change un attribut de l'objet, tous les objets utilisateurs pourront constater que cet attribut a changé (car tout le monde utilise la même instance).

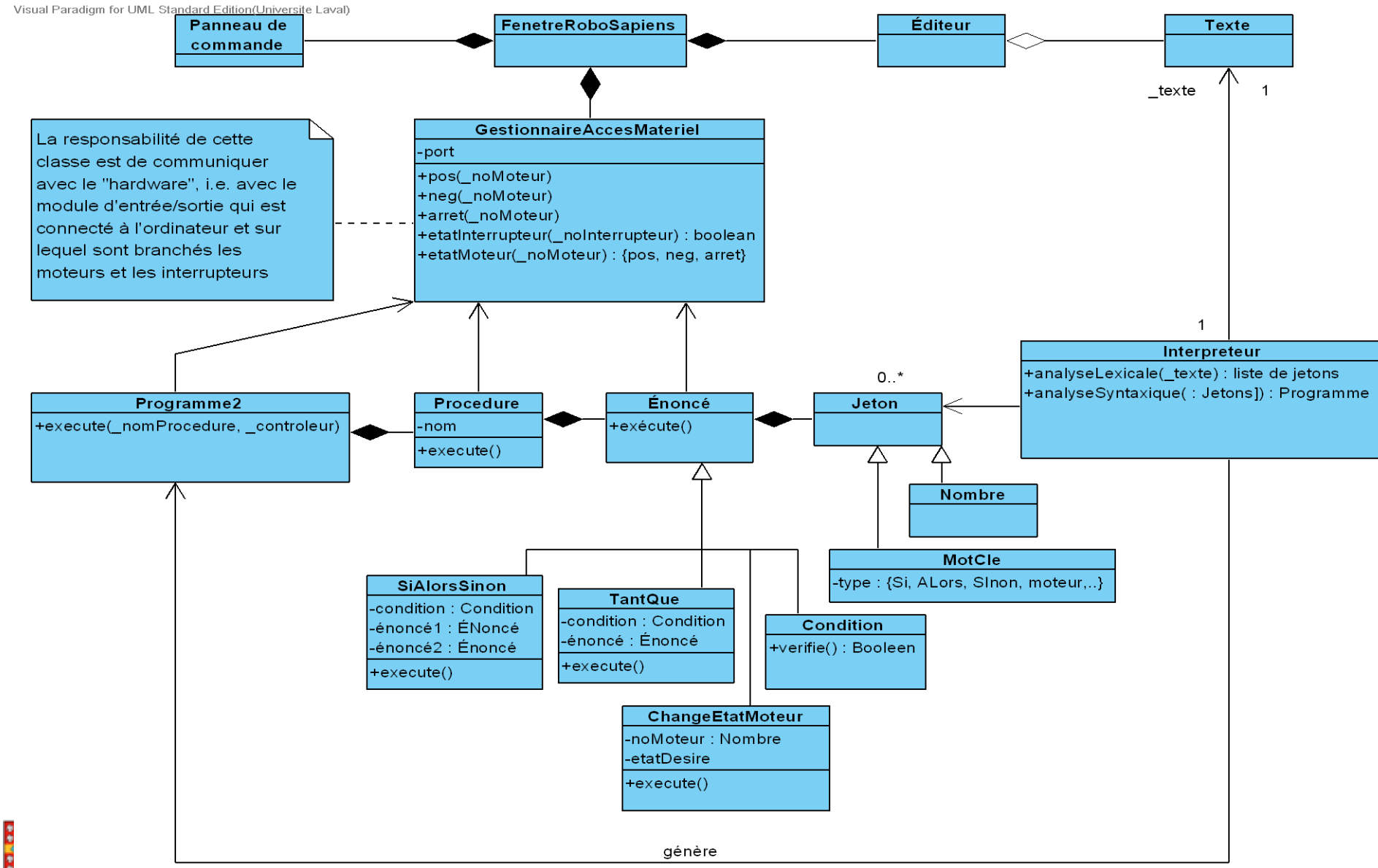
Réflexion Singleton

- Obtiendrais-je le même résultat si j'avais tout simplement défini tous les attributs et toutes les méthodes statiques?

Réflexion Singleton

- Obtiendrais-je le même résultat si j'avais tout simplement défini tous les attributs et toutes les méthodes statiques?
  - Oui!
  - L'avantage du Singleton est qu'il permet de créer des descendants (voir l'héritage) au sein desquels je pourrais réécrire certaines méthodes.
  - Les méthodes statiques ne peuvent pas être redéfinies.

Candidat au singleton dans Robo sapiens?



Singleton

- Comprenez-vous bien son utilité?
- Quelles seraient trois alternatives à l'utilisation d'un Singleton?

Plus de détail sur le Singleton dans le livre de Larman et, au besoin, dans le livre Design patterns de Gamma (GoF).

Singleton

- Seriez-vous en mesure de coder une classe qui serait un Singleton (à l'examen, par exemple) ?
  - ... une bonne façon de vérifier est de coder un singleton... ☺
  - Ce que nous allons faire!

Inconvénients Singleton ?

Inconvénients Singleton ?

- Exemple d'arguments contre l'utilisation d'un singleton:
  - C'est comme une variable globale « cachée ».
  - On peut créer des dépendances au singleton partout ce qui va à l'encontre du faible couplage.
  - Difficile à tester
    - On peut difficilement substituer une autre classe à un singleton

- Le singleton est toutefois utile dans certains contextes!

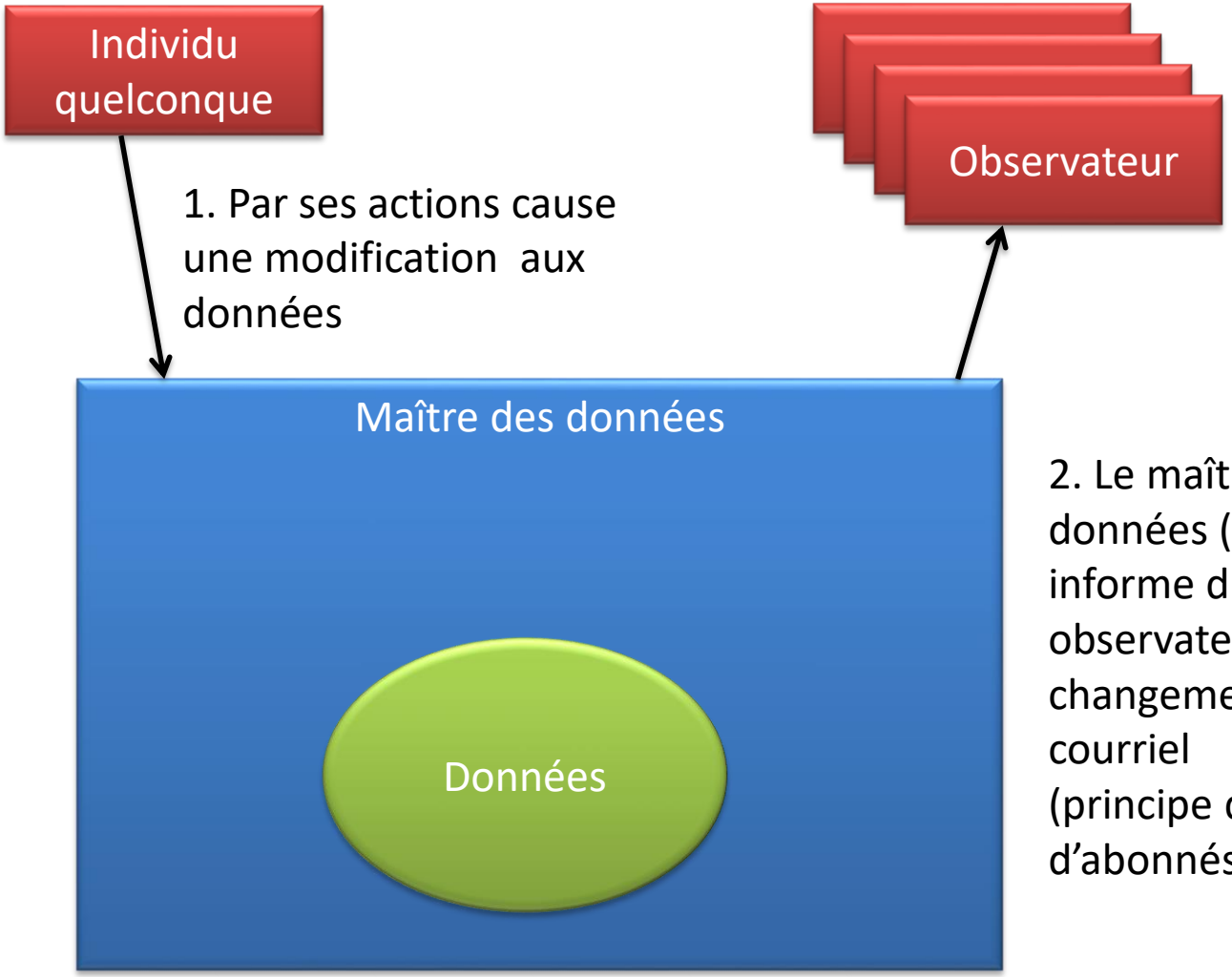
Et dans votre projet de session?

- Qui serait un bon candidat pour çâ?

Observer (GoF)

- **Problème**:
  - Différents objets (observateurs) désirent être informés lorsqu'une information change (ou lorsqu'un événement survient).
  - On veut cependant éviter que l'objet ayant causé le changement (diffuseur) appelle explicitement tous les observateurs (on veut éviter qu'il connaisse leurs classes)

Explication intuitive



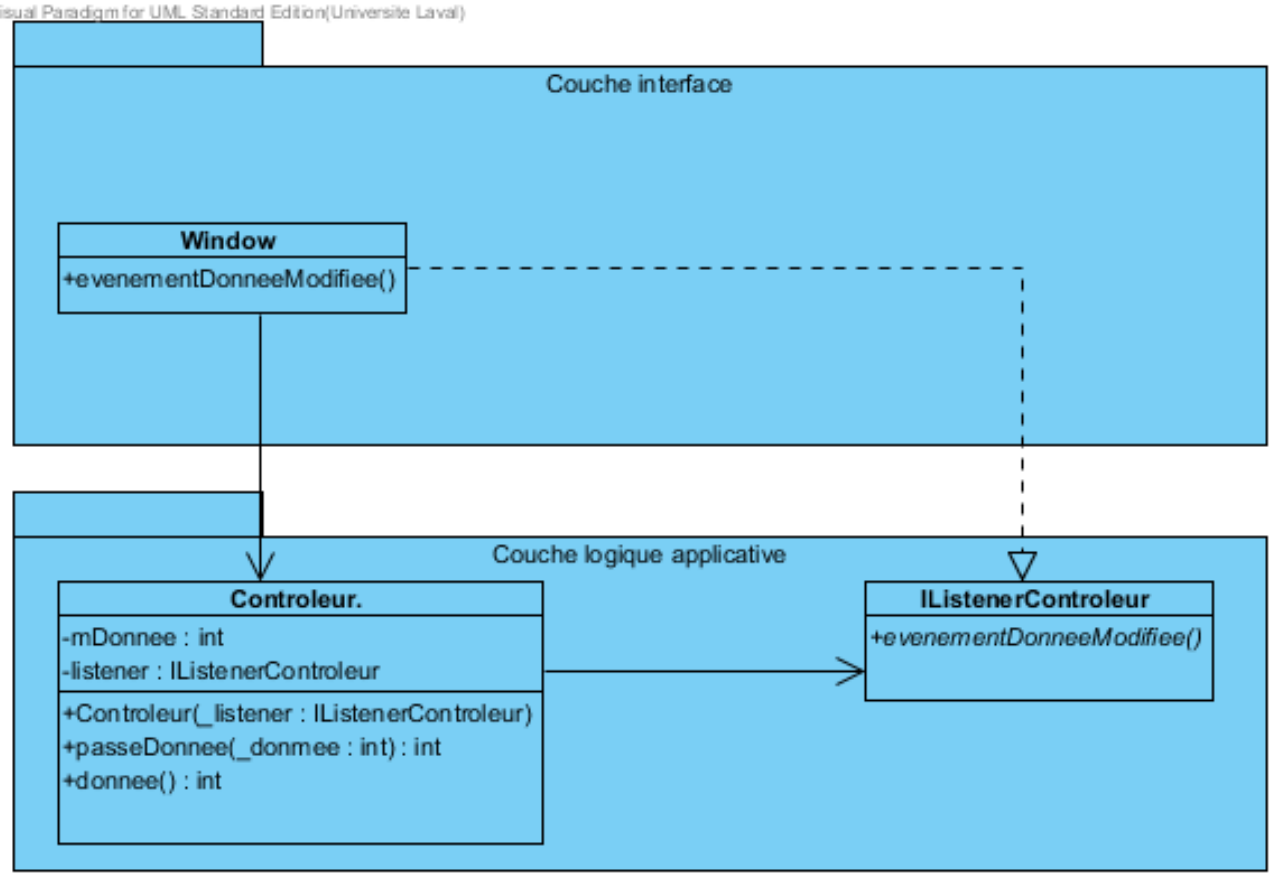
Observer (GoF)

- **Problème**:
  - Différents objets (observateurs) désirent être informés lorsqu'une information change (ou lorsqu'un événement survient).
  - On veut cependant éviter que l'objet ayant détecté le changement (diffuseur) appelle explicitement tous les observateurs (on veut éviter qu'il connaisse leurs classes)
- **Solution**:
  - Définir une « interface » appelée Observateur.
  - Tous les objets désirant être mis au courant des changements implémentent cette interface.
  - Le diffuseur maintient une liste d'observateurs qu'il informe le moment venu.

Usage classique

- Dans une architecture en couche, cela permet aux objets d'une couche inférieure d'informer des objets d'une couche supérieure sans connaître les classes de ces derniers (i.e. sans dépendance, sans couplage).

Observer (cas simplifié avec un seul observateur)



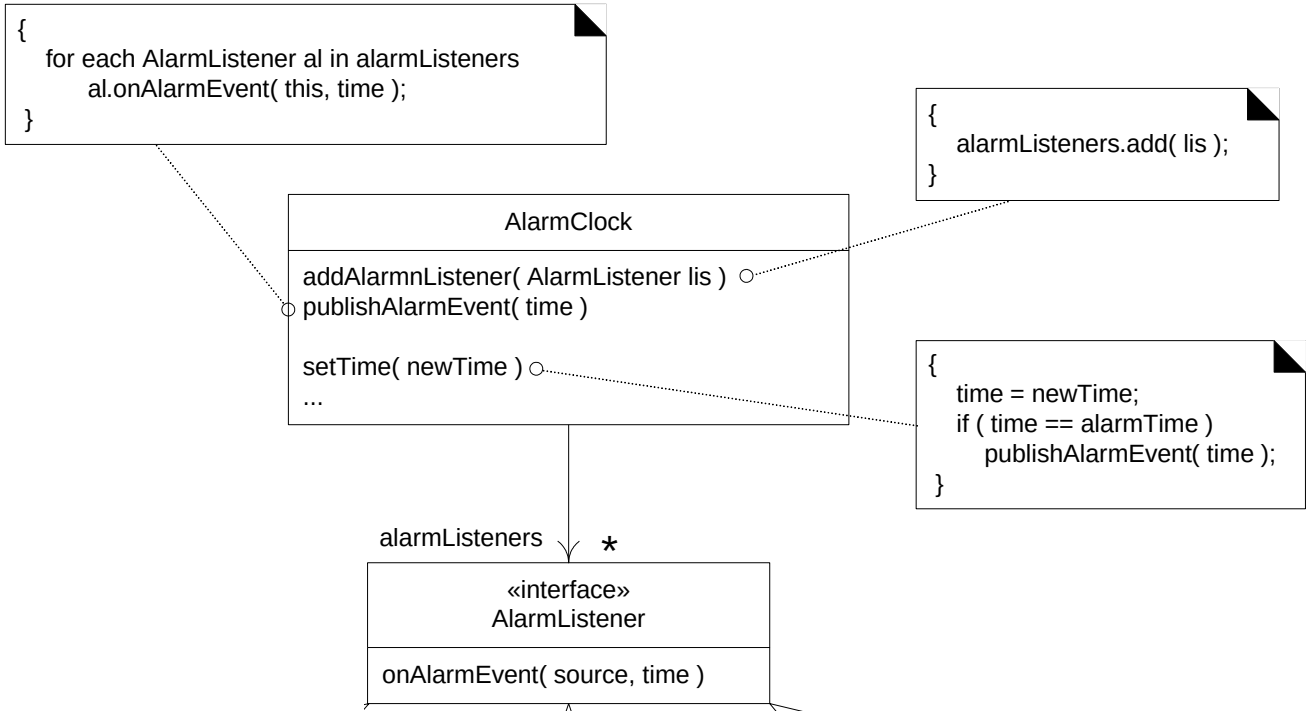
L'appel à passeDonnee déclanche un appel à evenementDonneeModifiee()

Dans cet exemple il y aun seul Listener qui celui qui a fait l'appel à passeDonnee()

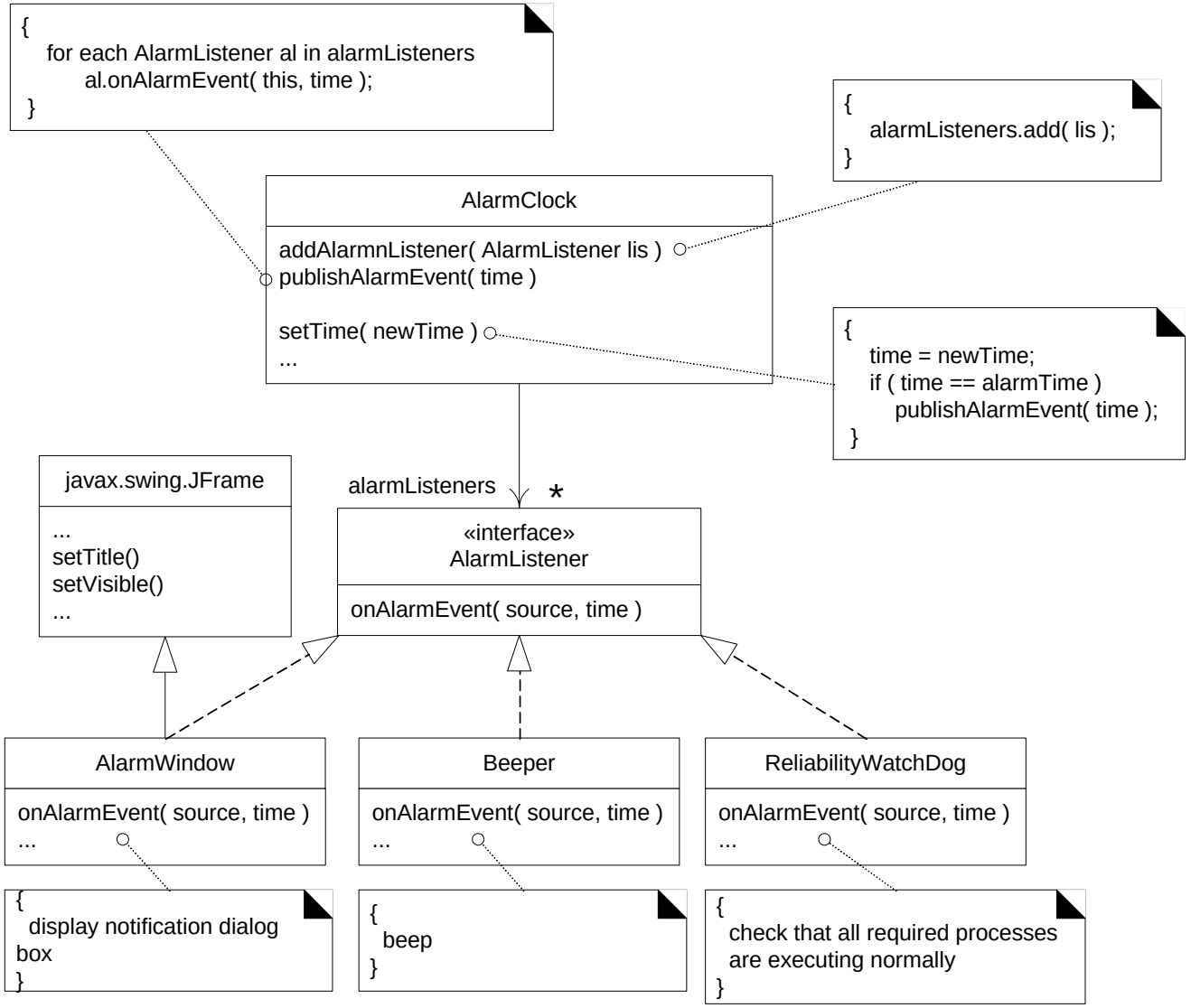
L'utilisation du patron Observer est donc plus ou moins utile.

Dans quel contexte serait-ce plus utile?

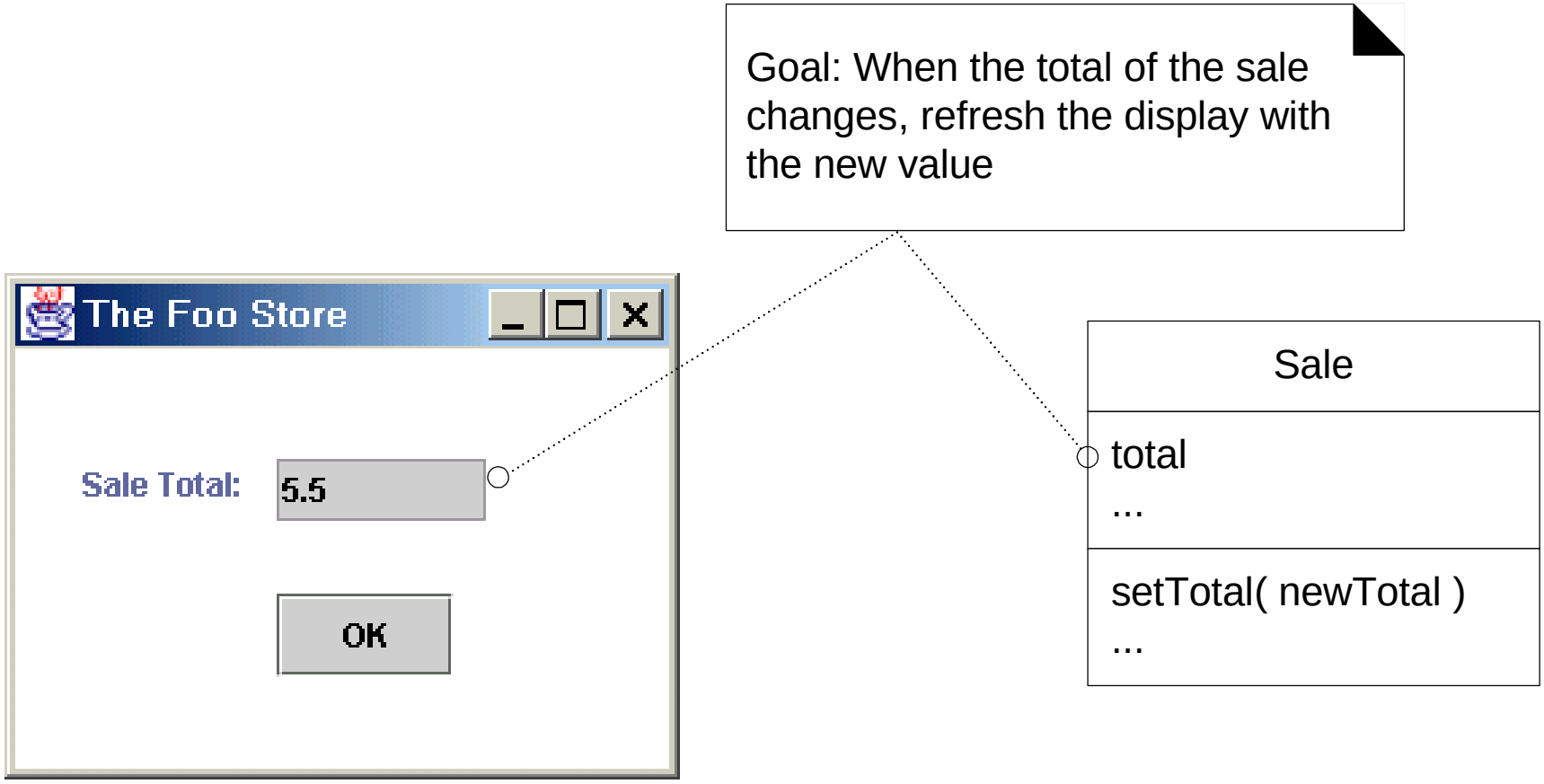
Réveil matin



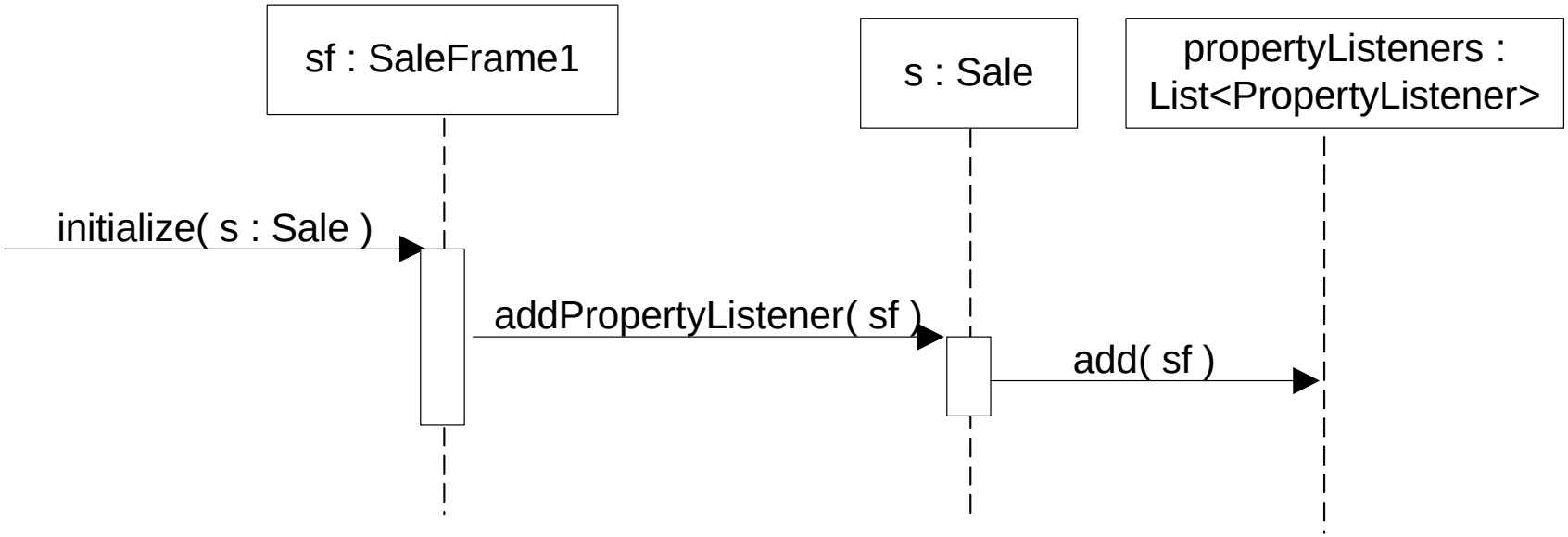
Réveil matin



Observer - NexGen

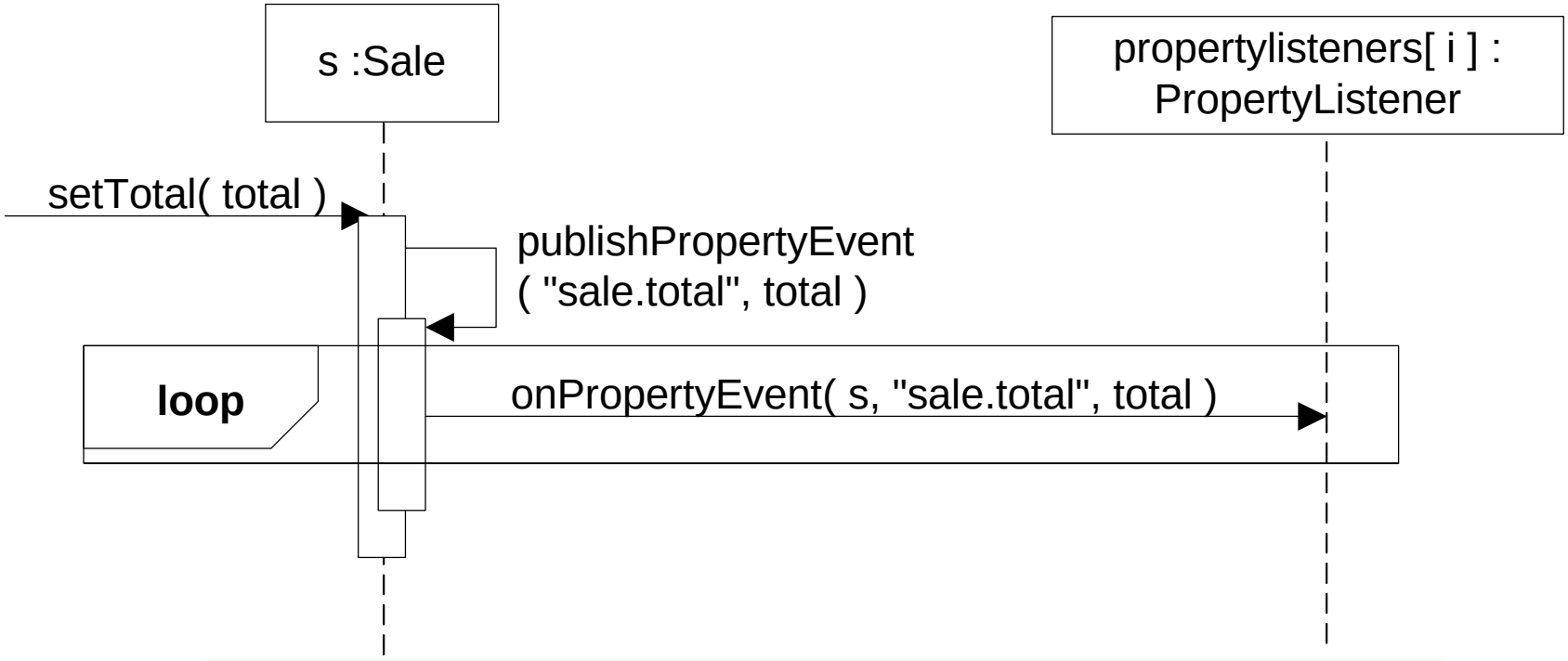


Observer - NexGen



L'objet SaleFrame1 s'inscrit à la « liste de diffusion » de Sale

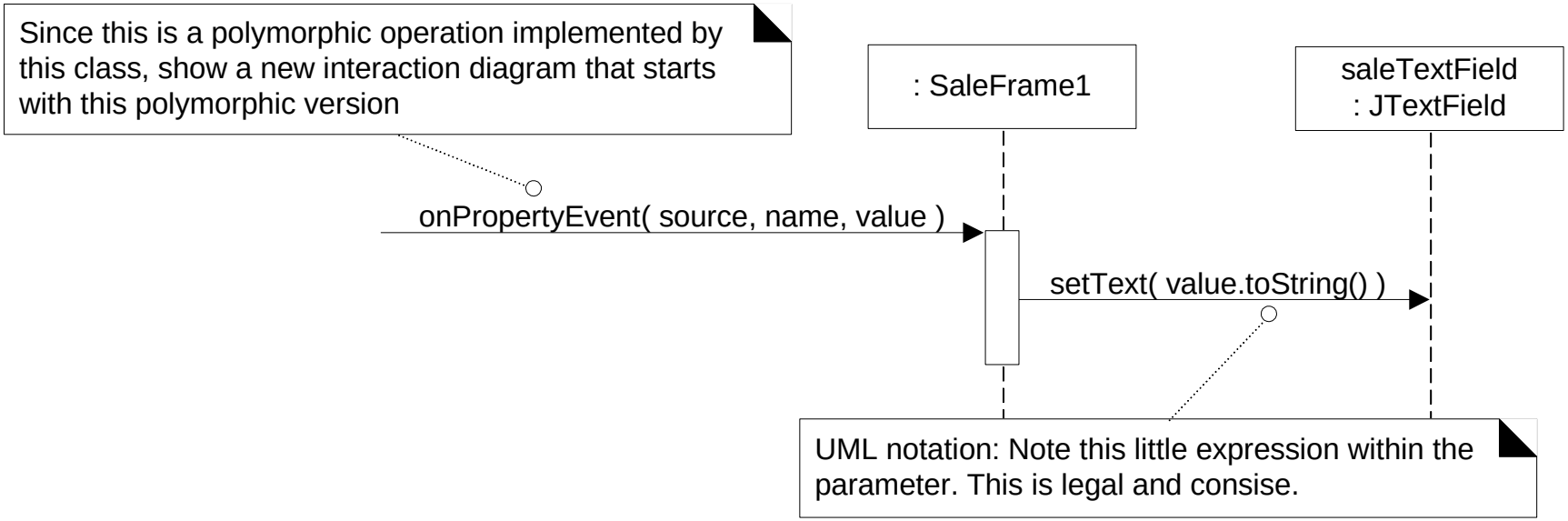
Observer - NexGen



Quand le total de l'objet Sale est modifié, il doit avertir les objets qui sont dans sa liste de diffusion.

Remarquez ici que le système est implémenté pour pouvoir gérer/annoncer plusieurs types d'événements distincts

Observer - NexGen

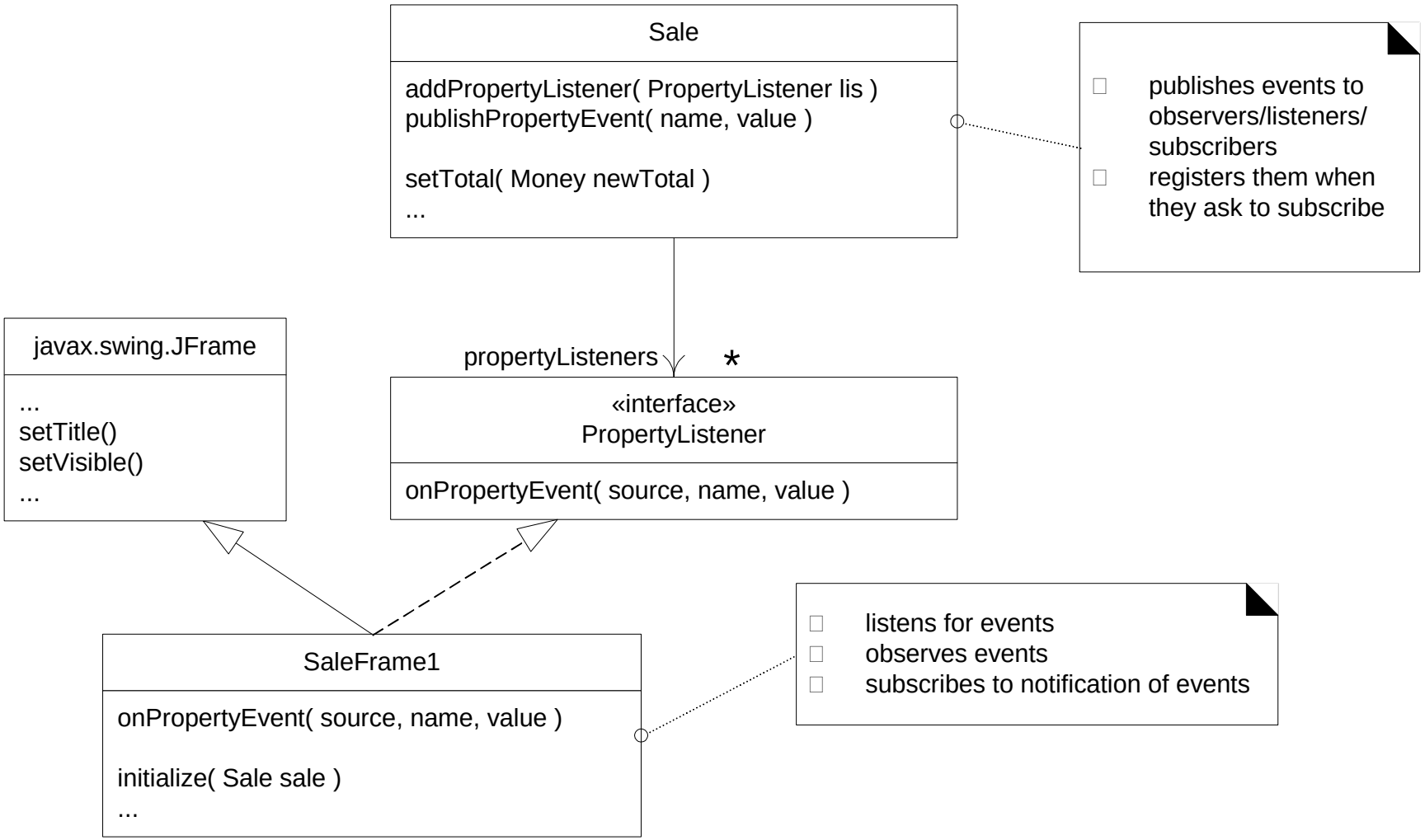


Réception du message par SaleFrame1

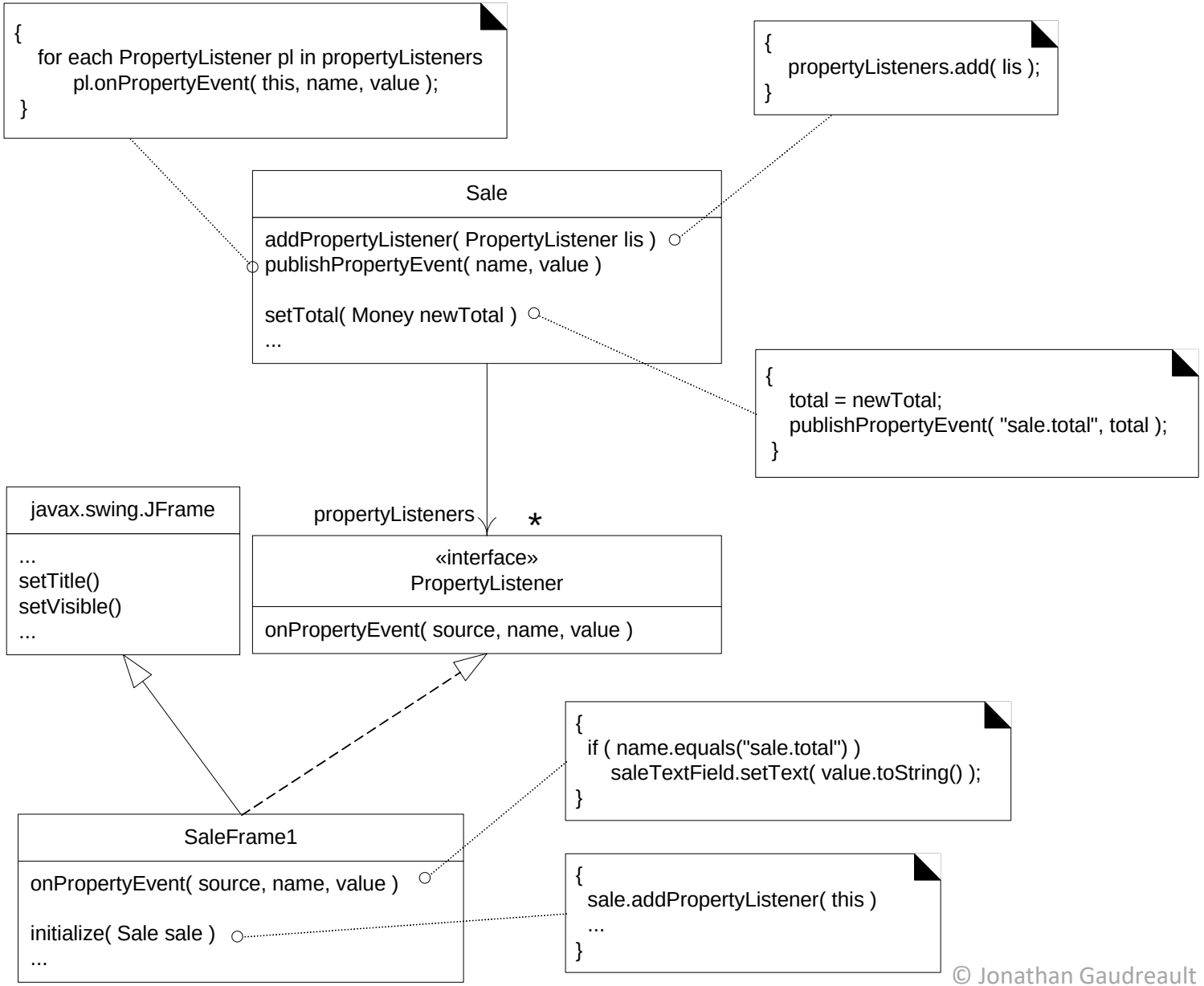
Mais c’est scandaleux!?

- L’objet « Sale » connaît ainsi la classe de la couche interface utilisateur « SaleFrame » ?
- Mais non!!

L'objet "Sale" gère une liste de "PropertyListeners" définie au niveau de la couche métier



Récapitulation



Et dans Robo sapiens?

- Événement envoyé par le Gestionnaire des accès matériels pour avertir l'interface utilisateur qu'un des interrupteurs ou des moteurs vient de changer d'état

Observer

- Seriez-vous en mesure de coder un observer (à l'examen, par exemple) ?
  - ... une bonne façon de vérifier est de coder un observer... ☺
  - Ce que nous allons faire!

Et dans votre projet de session?

- Ou cela serait utile?
  - Indice: Wiki du cours Atelier 7 ☺

Et dans votre projet de session?

- Pas réellement nécessaire de l'utiliser pour que votre couche interface soit informée d'un changement dans la couche du domaine.
- Pourquoi?

Et dans votre projet de session?

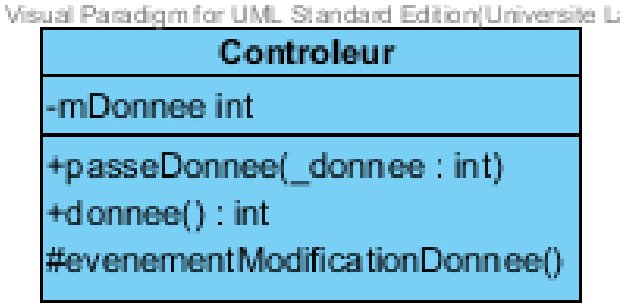
- Pas réellement nécessaire de l'utiliser pour que votre couche interface soit informée d'un changement dans l'horaire
- Pourquoi?
- Tous les changements sont commandés par la couche interface, il est donc beaucoup plus simple de procéder comme suit:
  - À chaque fois que la couche interface comprends que l'utilisateur veut un changement dans le plan, elle passe la requête au contrôleur
  - Le contrôleur fait faire les changements dans le plan
  - L'interface-utilisateur fait ensuite un réaffichage du plan

Et dans votre projet de session?

- Autre alternative:
  - Lorsqu'on demande au contrôleur une modification, le retour de fonction indique (via un booléen) si un changement a eu lieu

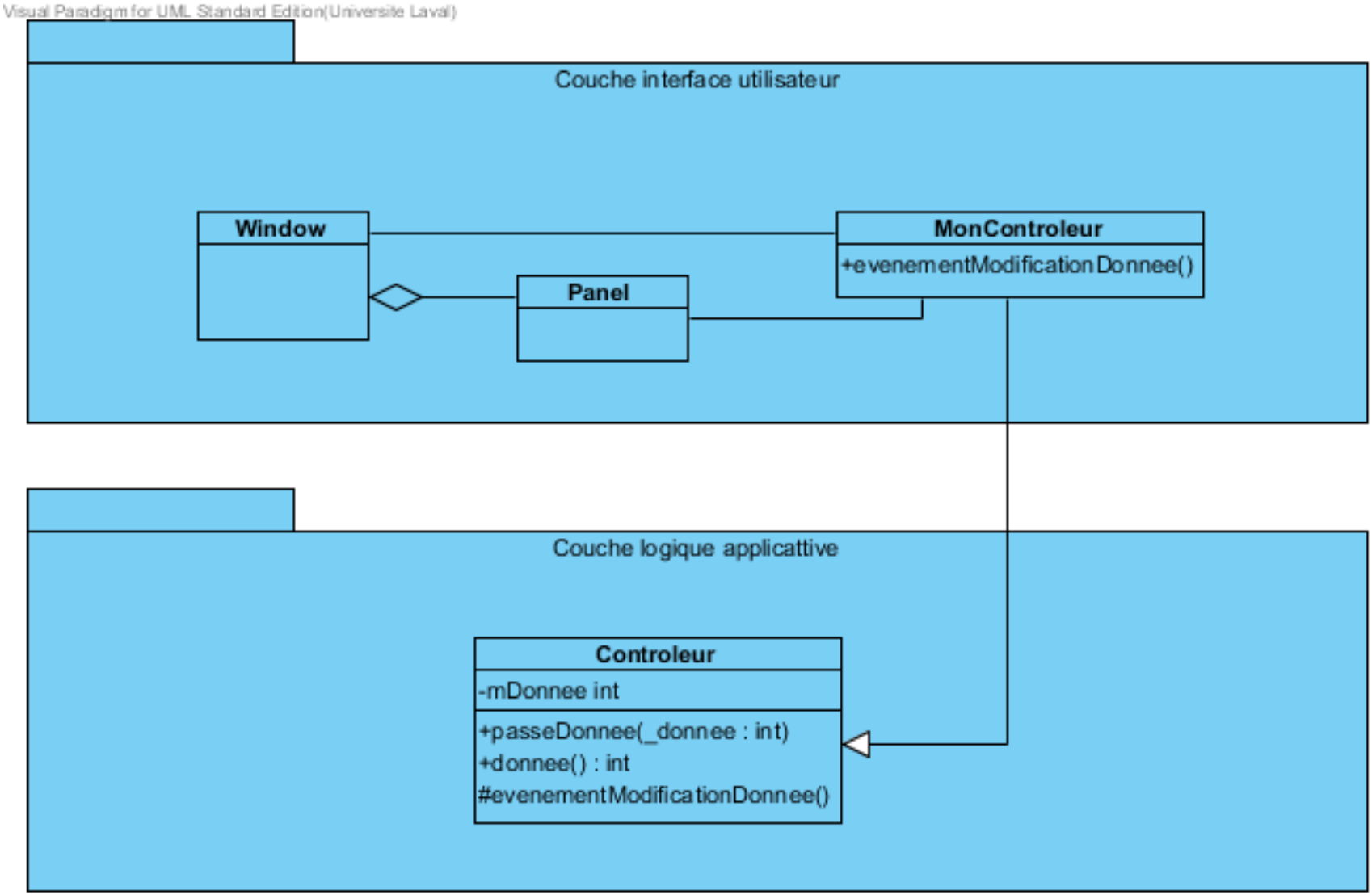
Alternative au patron Observer (pour les langages où le concept *d'interface* n'existe pas)

- Dans la couche contenant la logique applicative:
  - L'objet qui gère la donnée (ex. Contrôleur) appelle une de ses méthodes après chaque modification à la donnée (ex: **evenementModificationDonnee**) :



  - Au niveau de la couche interface-utilisateur, on définira un descendant de cette classe qui redéfinit la méthode **evenementModificationDonnee**();

Alternative au patron Observer
(pour les langages où le concept d'interface n'existe pas)



À faire pour ce module

- Lecture des chapitres
  - Version anglaise: 25, 26 (en mettant l'accent sur les patrons étudiés dans le cours)
  - Version française: 22, 23 (en mettant l'accent sur les patrons étudiés dans le cours)
- Comprendre les patrons présentés
- Comprendre la relation avec les grands principes de design orienté objet (GRASP)