

Méthodes de Conception

(L3 Info - SINFL5A1)

Jour 6

- Révisions sur MVC et Observer
- Révision sur Swing (interfaces graphiques en Java)
- MVC et Swing dans le cadre du TP et du devoir de CC : quelques pistes...

Révisions et compléments :

Le (méta) pattern MVC

Observer

Inversion de dépendance

Objectif

- Dissocier les trois "composantes". Ainsi :
- La conception du modèle n'est pas entachée par celles de la vue et du contrôleur
- On peut disposer de plusieurs vues (éventuellement différentes) sur un même modèle
- Réutilisabilité, en particulier du modèle

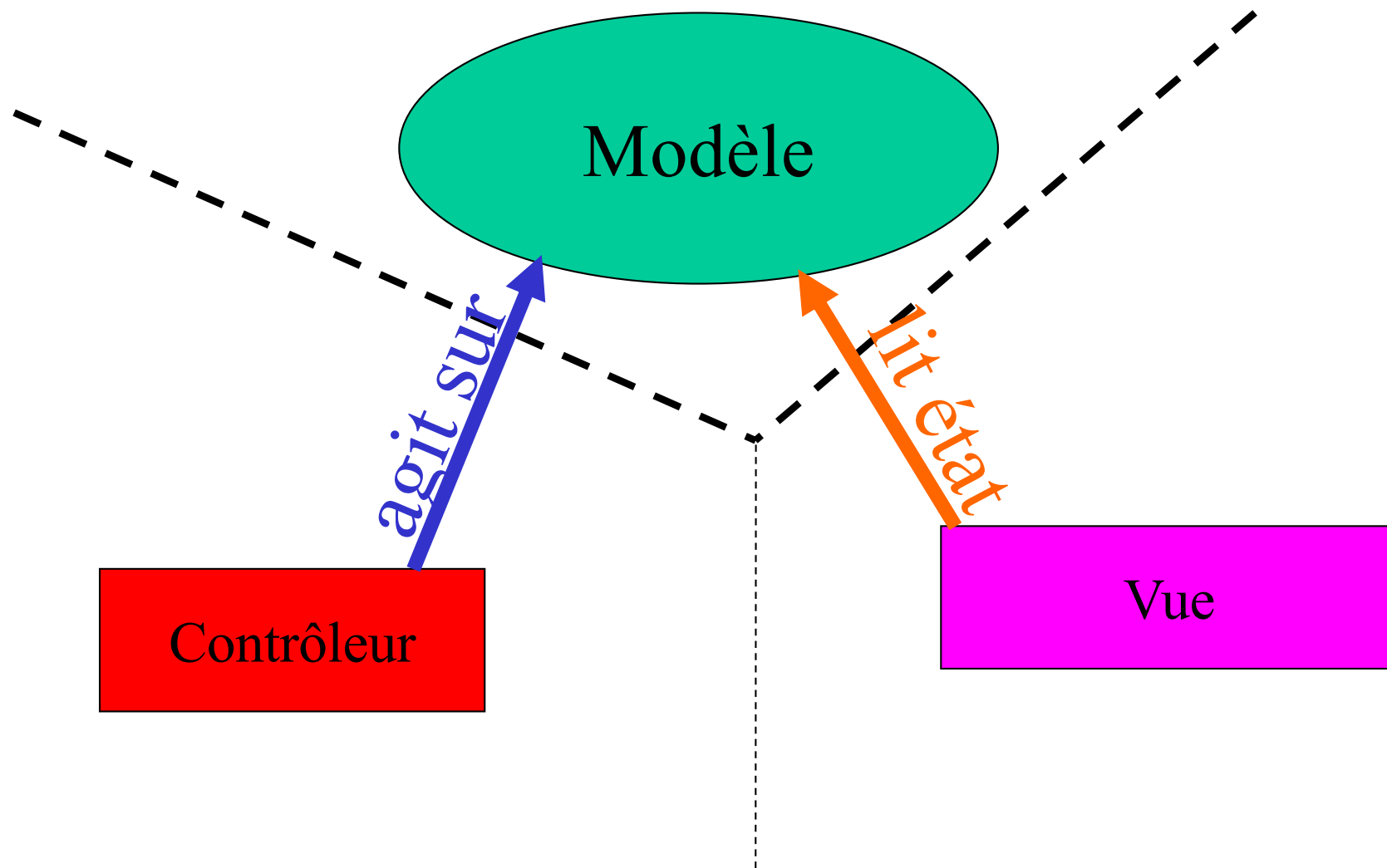
Principe

- Modèle : état et comportement
- Contrôleur : des listeners (de clavier, de souris), et des composants graphiques générant des événements.
- Vue : différents éléments Swing.

Principe (2)

- Le Modèle doit tout ignorer de V et de C
- Le Contrôleur doit posséder une référence sur le Modèle, pour agir dessus (ex. modifier une donnée, lancer un comportement)
- La vue doit aussi posséder une référence sur le Modèle, mais uniquement en lecture d'information (jamais en modification)

un modèle indépendant



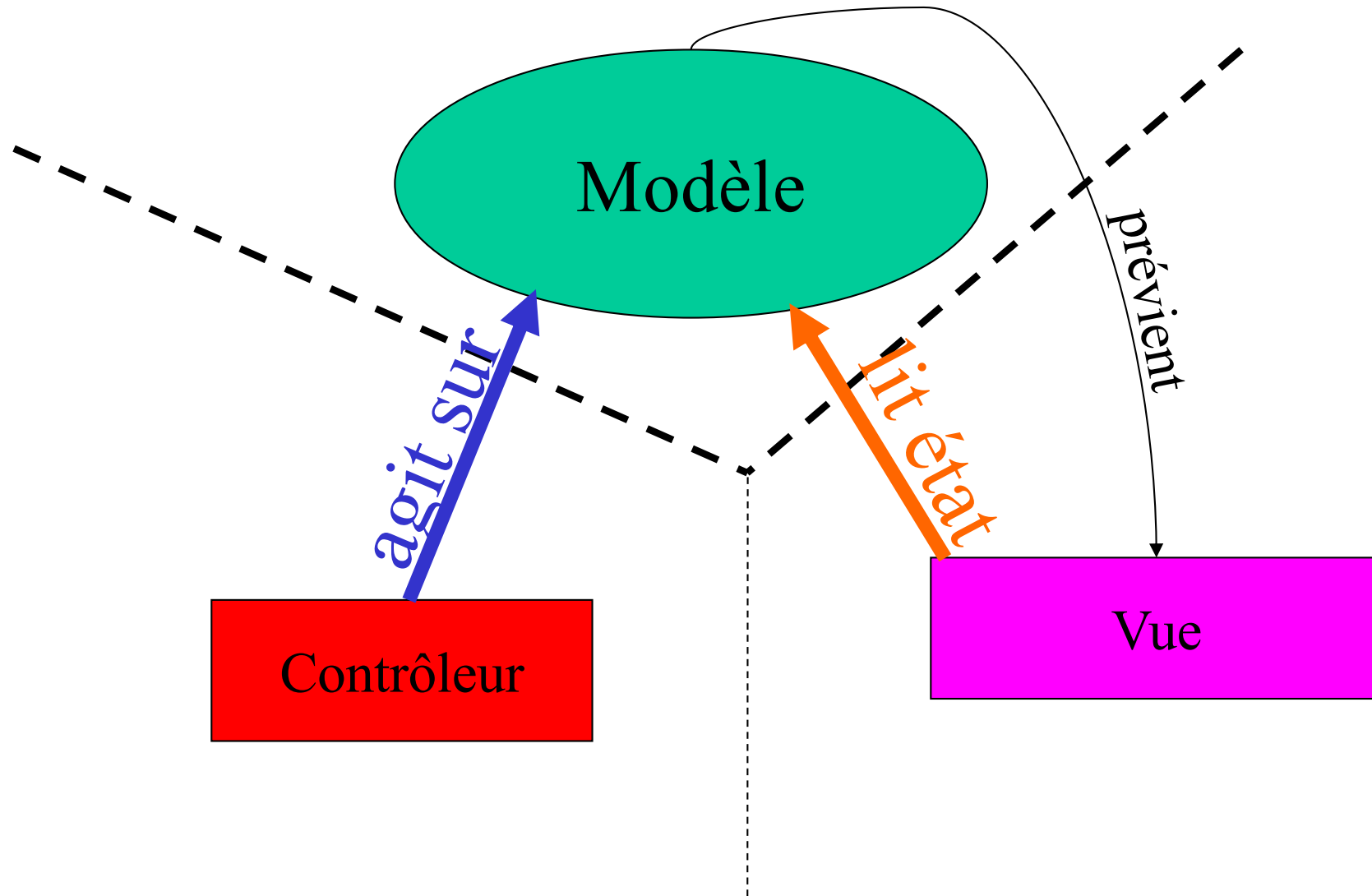
Indépendance du Modèle ?

- Si le modèle est construit en totale indépendance, comment la vue peut-elle savoir que son état a changé (il faut alors actualiser la vue) ?
- 1. On peut imaginer que le contrôleur, en même temps qu'il agit sur le modèle, le signale à la vue... Le modèle reste donc indépendant.
- 2. Le modèle peut aussi être conçu pour avoir des Listeners de ses modifications. Lorsque son état change, il prévient ses listeners, en lançant un événement (fire).

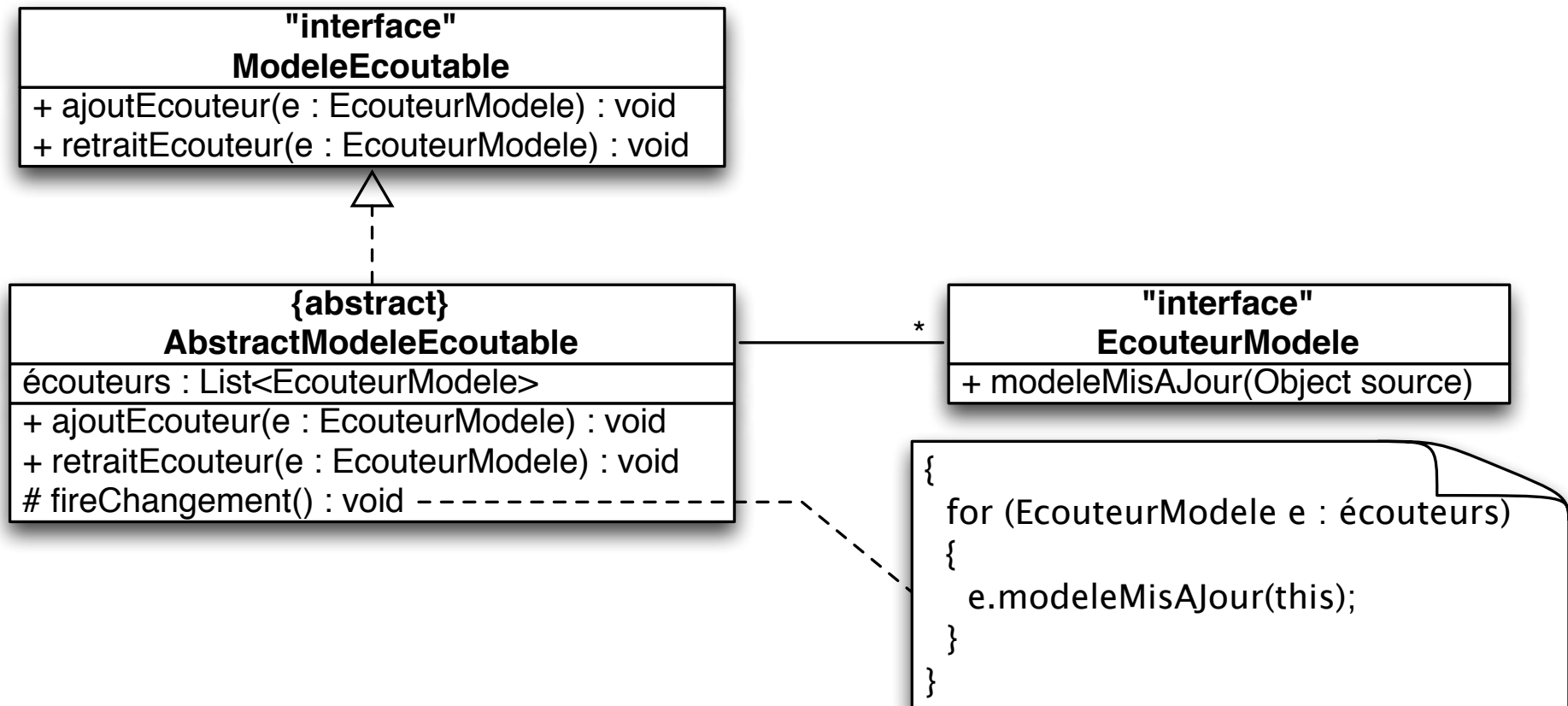
Quelle solution ?

- La solution 1 a le mérite de laisser le modèle totalement indépendant. Mais le contrôleur outrepassa son rôle. Mais surtout, l'état du modèle peut changer pour une raison autre que l'action du contrôleur !
- La solution 2 est donc préférable : c'est le modèle qui sait quand il change, quelle que soit l'origine de ce changement. De plus, le modèle ignore toujours tout du contrôleur et de la vue : il possède juste des "listeners" de ses éventuels changements.

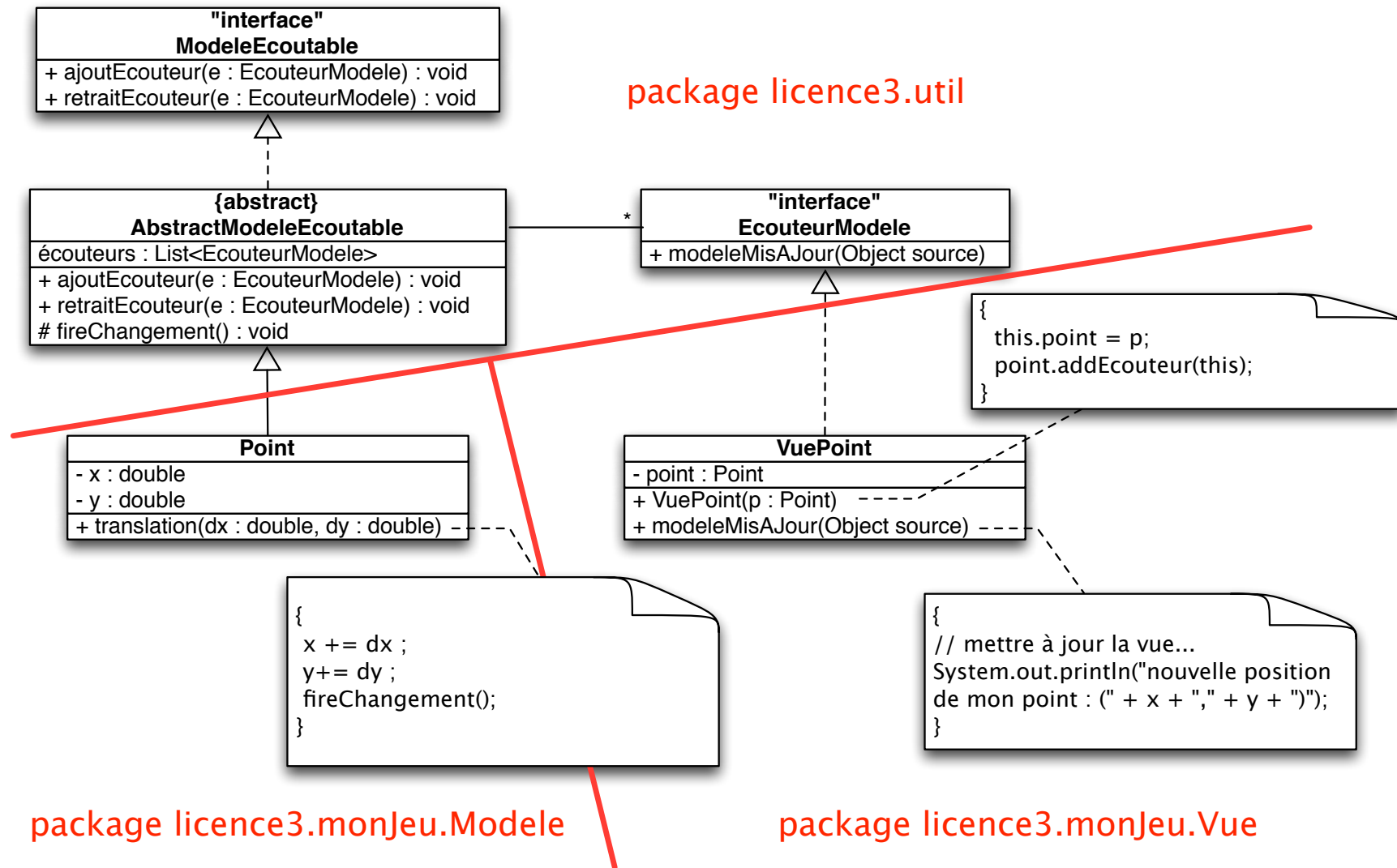
Un modèle écouté.



Implémentons Observer...



Application à MVC



L'inversion de dépendance

- Le code de la classe Point (modèle) ne dépend pas de celui de la vue, et le code de la classe VuePoint dépend de celui du modèle. $V \rightarrow M$
- En revanche, au niveau des instances, un point est capable de prévenir une vuePoint. $M \rightarrow V$
- Grâce à l'utilisation d'une interface EcouteurModèle conçue du côté du modèle, on procède à un **découplage** et on parvient à **inverser la dépendance** entre le code (la vue dépend du modèle) et les instances (le modèle prévient la vue).

Une subtilité...

L'interface `ModeleEcoutable` est utile lorsque l'on veut définir un nouveau type d'objets (écoutables) via une interface plutôt que par une classe. Dans ce cas là, l'interface de ce nouveau type hérite de `ModeleEcoutable`, et ses implémentations peuvent hériter (si elles n'héritent pas déjà d'une autre classe) de la classe `AbstractModeleEcoutable` (pour ne pas avoir à réécrire la partie événementielle...)

MVC ou M-VC

- Quand on le peut : MVC (exemple : calculatrice classique)
- Souvent : M-VC, car il n'est pas toujours aisé voire possible de distinguer le contrôleur de la vue (exemple : calculatrice autorisant la modification directe de la valeur affichée sur son écran JTextField).

MVC et Swing

- Les composants Swing sont bâtis selon le principe.
 - La séparation n'est pas totalement explicite : une certaine **classe** de composant propose une "Vue-Contrôleur" sur un certain **modèle**.
 - La vue peut être paramétrée par l'utilisation de "renderers"
 - Ex :
 - **JTable** (vue-contrôleur), **TableModel** (le modèle).
 - **JTree** (vue-contrôleur), **TreeModel** (le modèle).
- getModel() → référence vers le modèle

Exemple d'une calculatrice

