L'architecture MVC

Chapitre 6 du poly

Le modèle MVC : Model-View-Controller

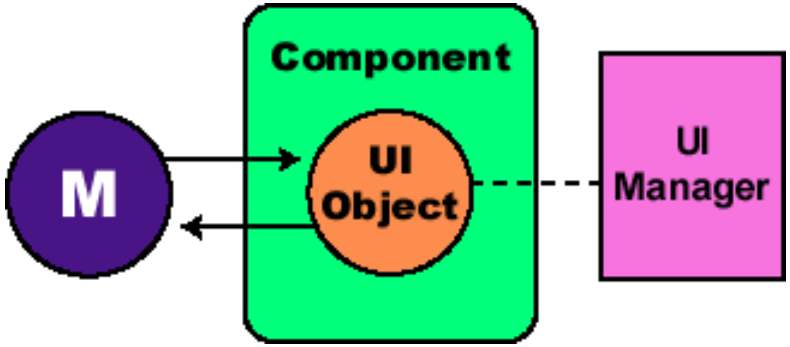
- Né en 1980 (créé par Xerox PARC pour le langage Smalltalk)
- Schéma de programmation qui prend en compte l'architecture d'un programme et classe les différents types d'objets selon 3 catégories :
  - **Les objets "view"** : représentation visuelle du "model"
  - **Les objets "model"** : données de l'application
  - **Les objets "controller"** : gèrent les interactions avec l'utilisateur, ils reçoivent les requêtes utilisateurs puis informent les objets "model"

Le modèle MVC

- But de MVC
  - Mieux structurer les applications
  - Représentation **multi-vues**
    - Un **modèle peut être associé à plusieurs vues**
    - **Synchronisation implicite**
- Remarques:
  - En pratique, V est fortement lié à C
  - Il existe de nombreuses variantes de MVC

MVC dans l'architecture Swing

- "Separable Model Architecture"
  - View et Controller regroupés dans UIComponent
  - Model reste séparé

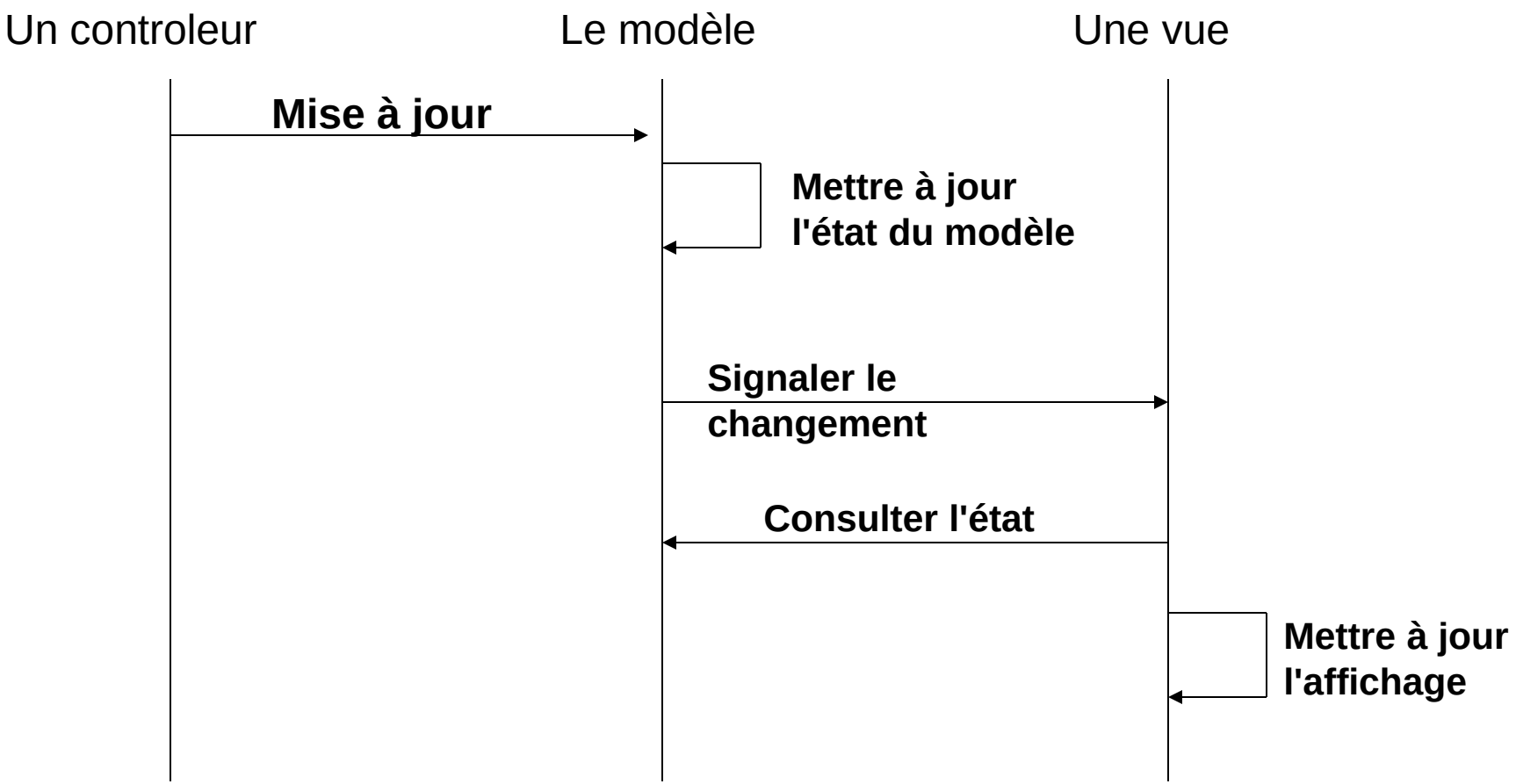


Ex: Les boutons (JButton) utilisent un objet de classe ButtonModel pour les données.

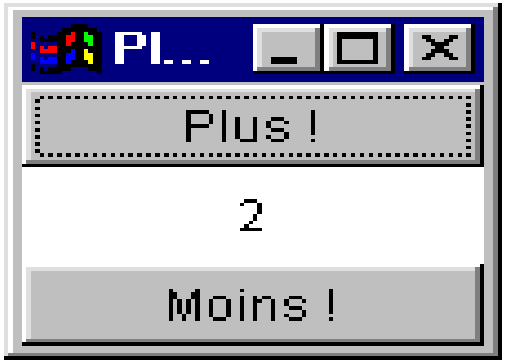
Communications entre composants

- Le contrôleur modifie le modèle en utilisant les méthodes du modèle
- À chaque fois qu'un modèle change d'état, il le signale à ses vues
- Quand une vue est informée d’une modification de l'état de son modèle, elle le consulte pour mettre à jour l'affichage

Principe du modèle MVC

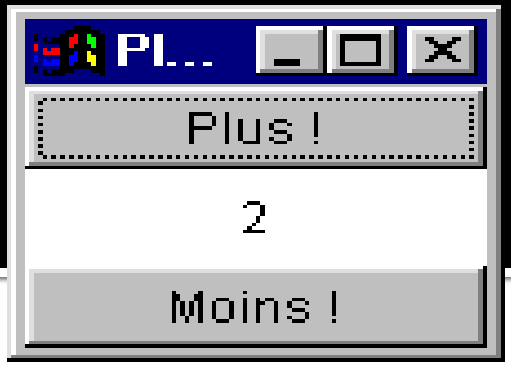


Exemple



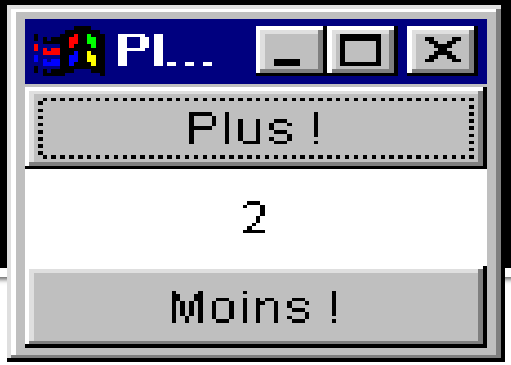
Créer l'interface ci-dessus selon le MVC

Exemple



Le modèle ?

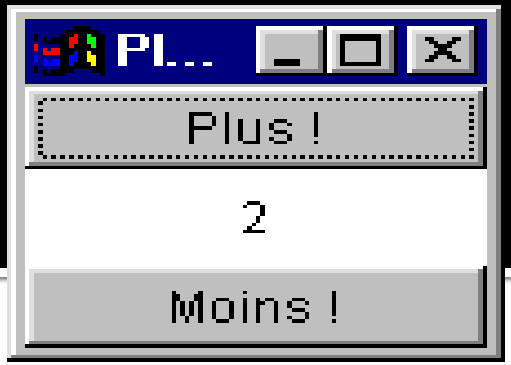
Exemple



Le modèle ?

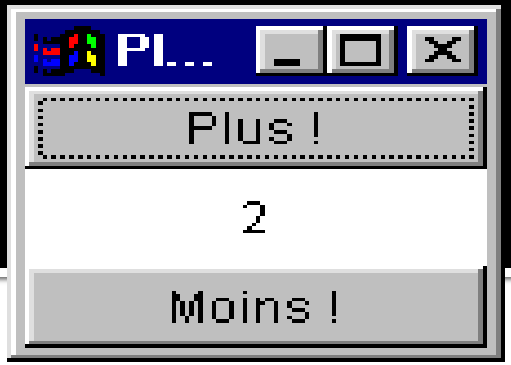
Model
- compteur : int
+ modifier(int incr) + getValue() : int

Exemple

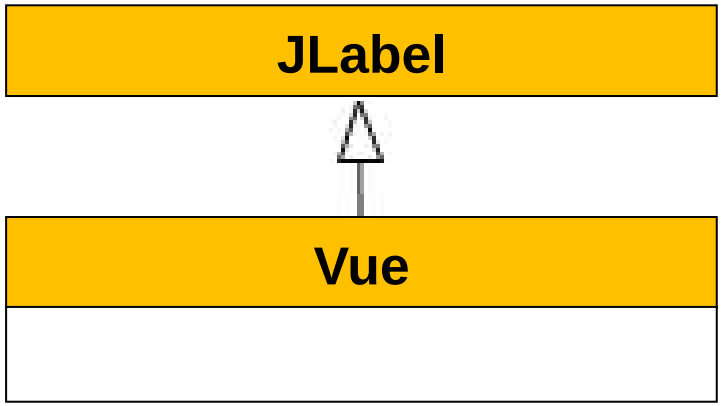


La vue?

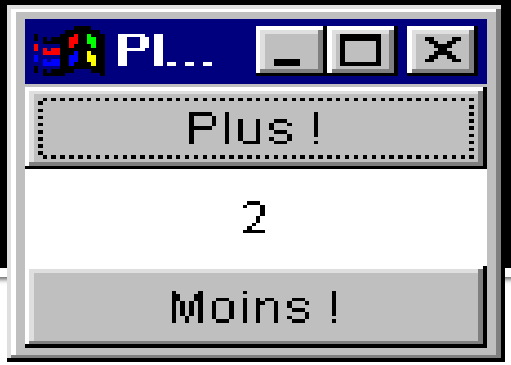
Exemple



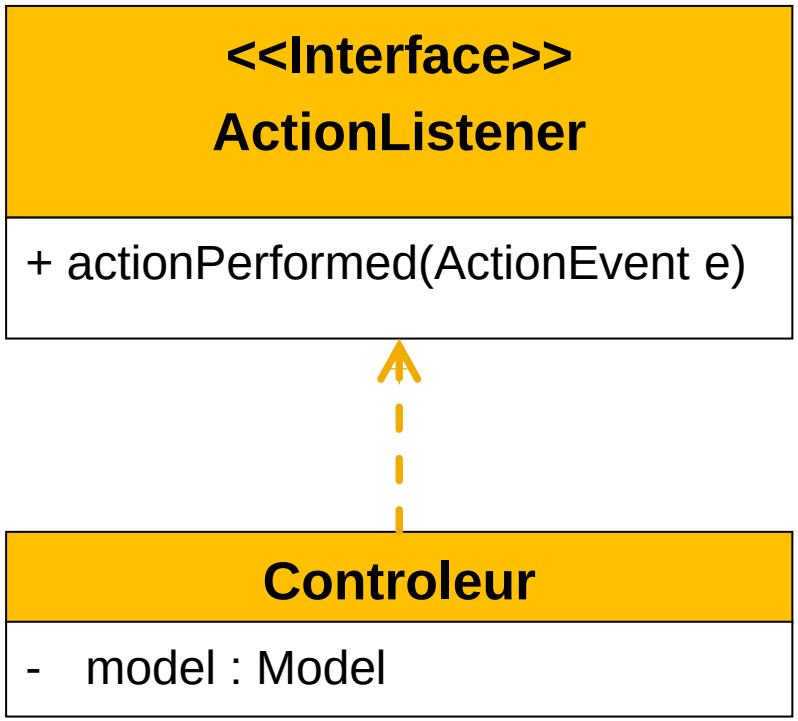
La vue?



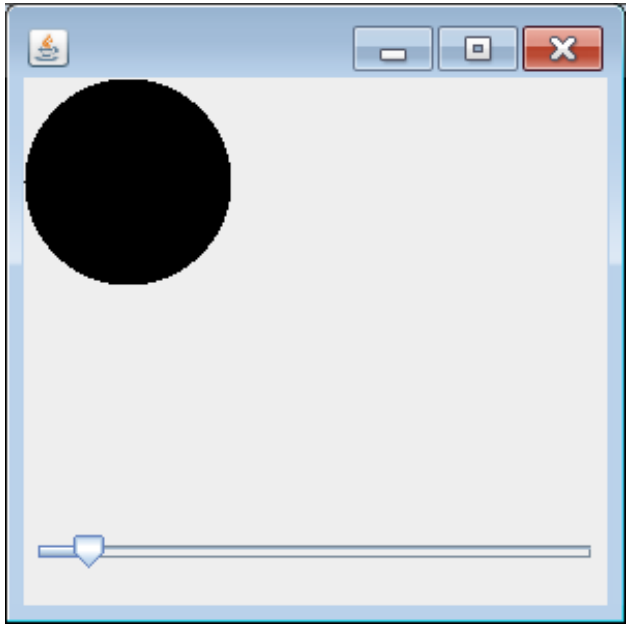
Exemple



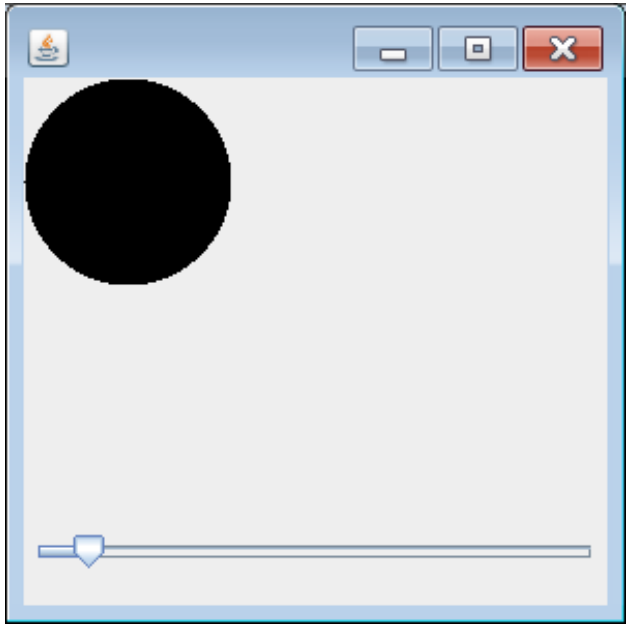
Le controleur ? Ecouteur sur les boutons



Exemple du poly, mis sur arche

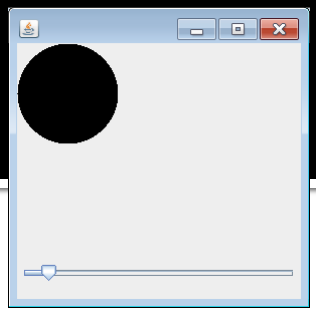


Exemple sur arche

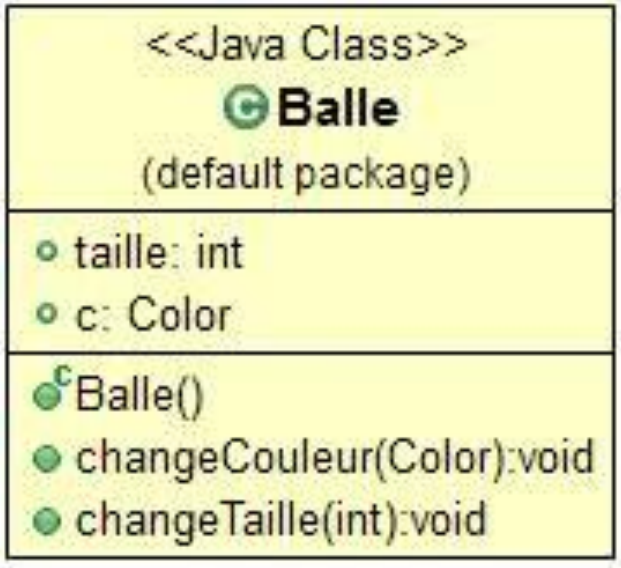


Modèle ?

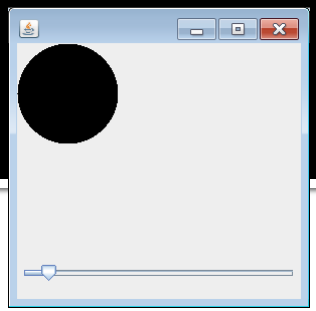
Exemple sur arche

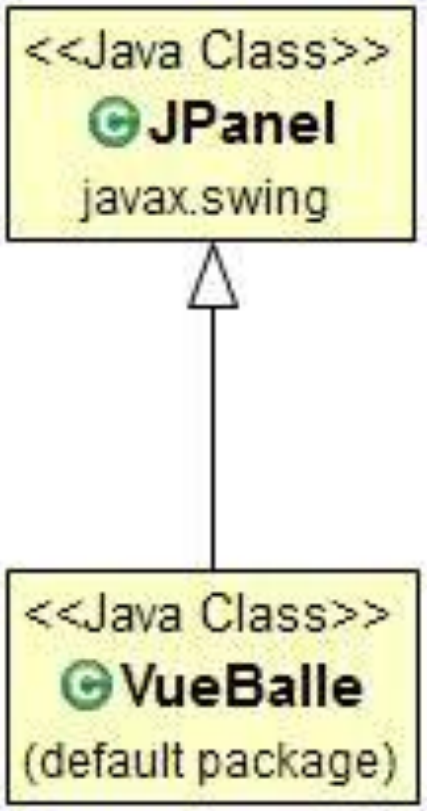


Modèle ?



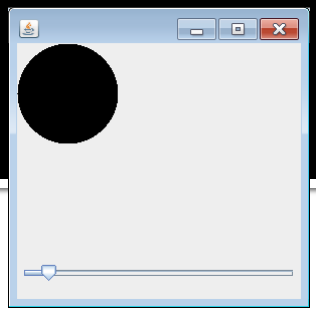
Exemple sur arche

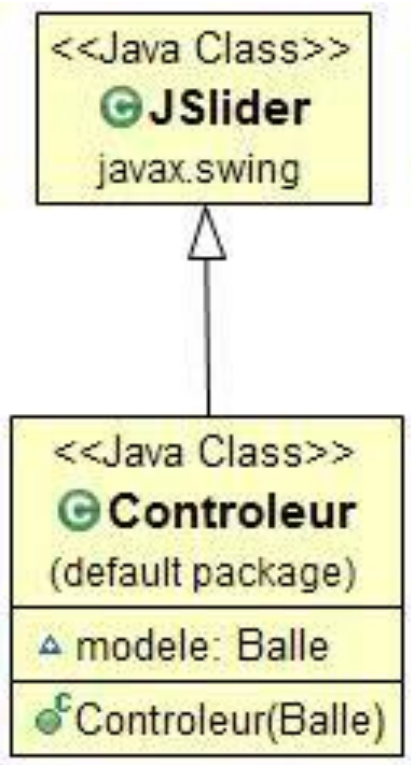




Vue graphique?

Exemple sur arche





Contrôleur ?

Comment mettre en place les interactions MVC ?

- L'utilisateur modifie la taille avec le controleur (JSlider)
- Le controleur demande la modification au modèle (appel d'une méthode de Balle)
- Le modèle se modifie et informe la vue de cette modification
- La vue consulte les changements du modèle et se met à jour

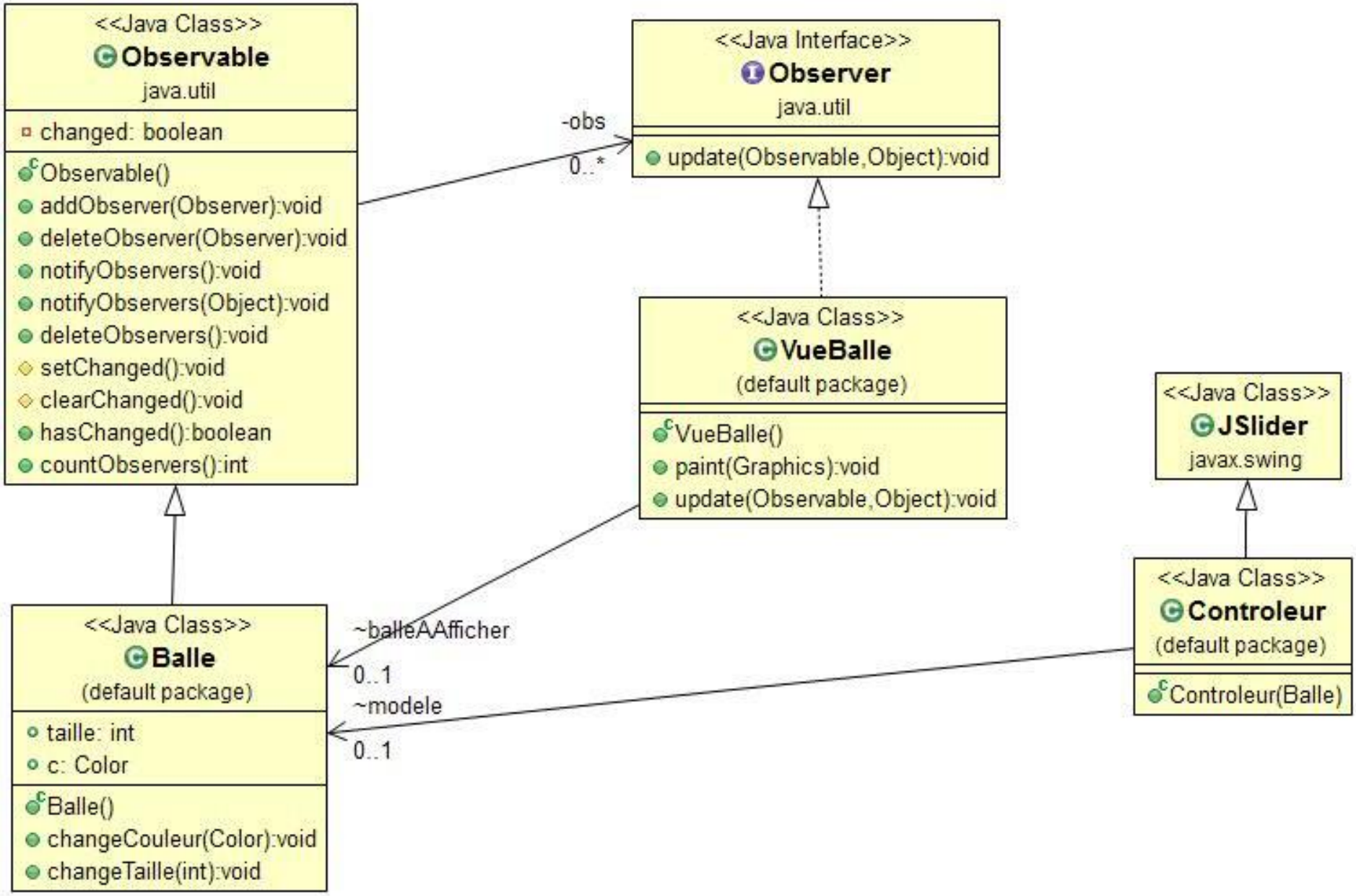
Comment mettre en place les interactions MVC ?

- L'utilisateur modifie la taille avec le controleur (JSlider)
- Le controleur demande la modification au modèle (appel d'une méthode de Balle)
- Le modèle se modifie et informe la vue de cette modification
- La vue consulte les changements du modèle et se met à jour

→ Utilisation de la classe Observable et de l'interface Observer sde Java

Observer/Observable

- Reposant sur le patron de conception Observateur (sera détaillé en S3)
- Le modèle hérite de la classe Observable
- Les vues implémentent l'interface Observer (« ils observent le modèle »)


<<Java Class>>
Observable
java.util
changed: boolean
Observable()
addObserver(Observer):void
deleteObserver(Observer):void
notifyObservers():void
notifyObservers(Object):void
deleteObservers():void
setChanged():void
clearChanged():void
hasChanged():boolean
countObservers():int
-obs
0..*
<<Java Interface>>
Observer
java.util
update(Observable,Object):void
<<Java Class>>
VueBalle
(default package)
VueBalle()
paint(Graphics):void
update(Observable,Object):void
<<Java Class>>
JSlider
javax.swing
<<Java Class>>
Balle
(default package)
taille: int
c: Color
Balle()
changeCouleur(Color):void
changeTaille(int):void
~balleAAfficher
0..1
~modele
0..1
<<Java Class>>
Controleur
(default package)
Controleur(Balle)

Classe Observable

- Méthodes de la classe Observable *(java.util.Observable)*
  - addObserver(Observer) ajoute un objet d'Observer à la liste des objets Observer de l'instance
  - countObservers() retourne le nombre d'objets Observer
  - deleteObservers(Observer) enlève un objet d'Observer de la liste des objets Observer de l'instance
  - setChanged() indique qu'un changement s'est produit sur l'instance de l'objet
  - notifyObservers() ou notifyObservers(Object) indique aux objets Observer qu'un changement a eu lieu après que la méthode setChanged() ait été utilisée

Classe Observable

- Précisions sur les appels de setChanged et notifyObservers
  - L'appel de setChanged fait passer un attribut boolean *changed* à true
  - A l'appel de notifyObservers, 2 possibilités :
    - Si *changed* est à true,
      - l'objet Observable appelle la méthode **update** sur chaque objet Observer
      - Puis *changed* est mis à false
    - Si *changed* est à false, la méthode update n'est pas appelée sur les objets Observer

Conclusion : nécessité de coupler les appels à setChanged et notifyObservers pour que les modifications soient répercutées au niveau des Observers

Interface Observer *(java.util.Observer)*

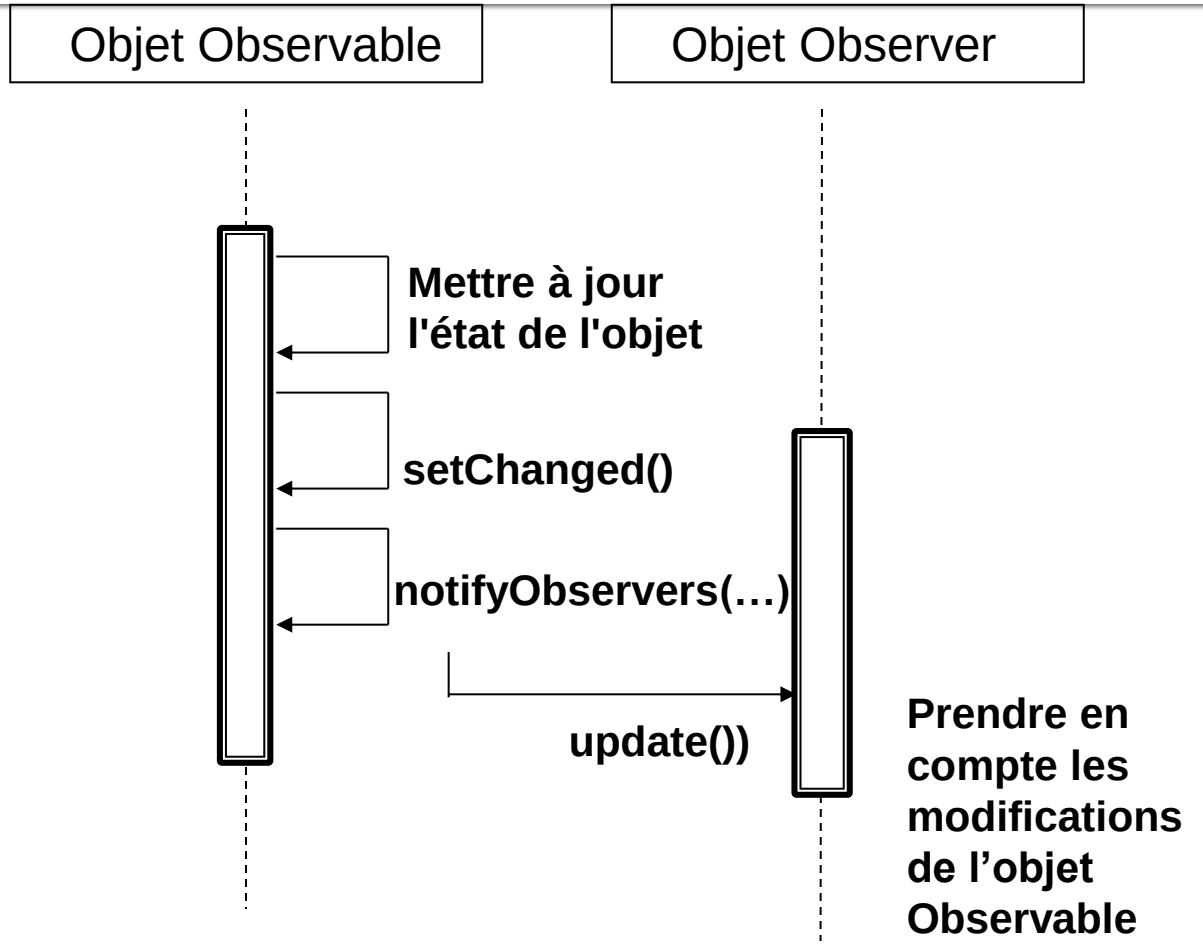
- Contient l'unique signature de la méthode **update** appelée quand l'instance appartient à la liste des objets Observer de **o** :

  public abstract void update (Observable o, Object arg)

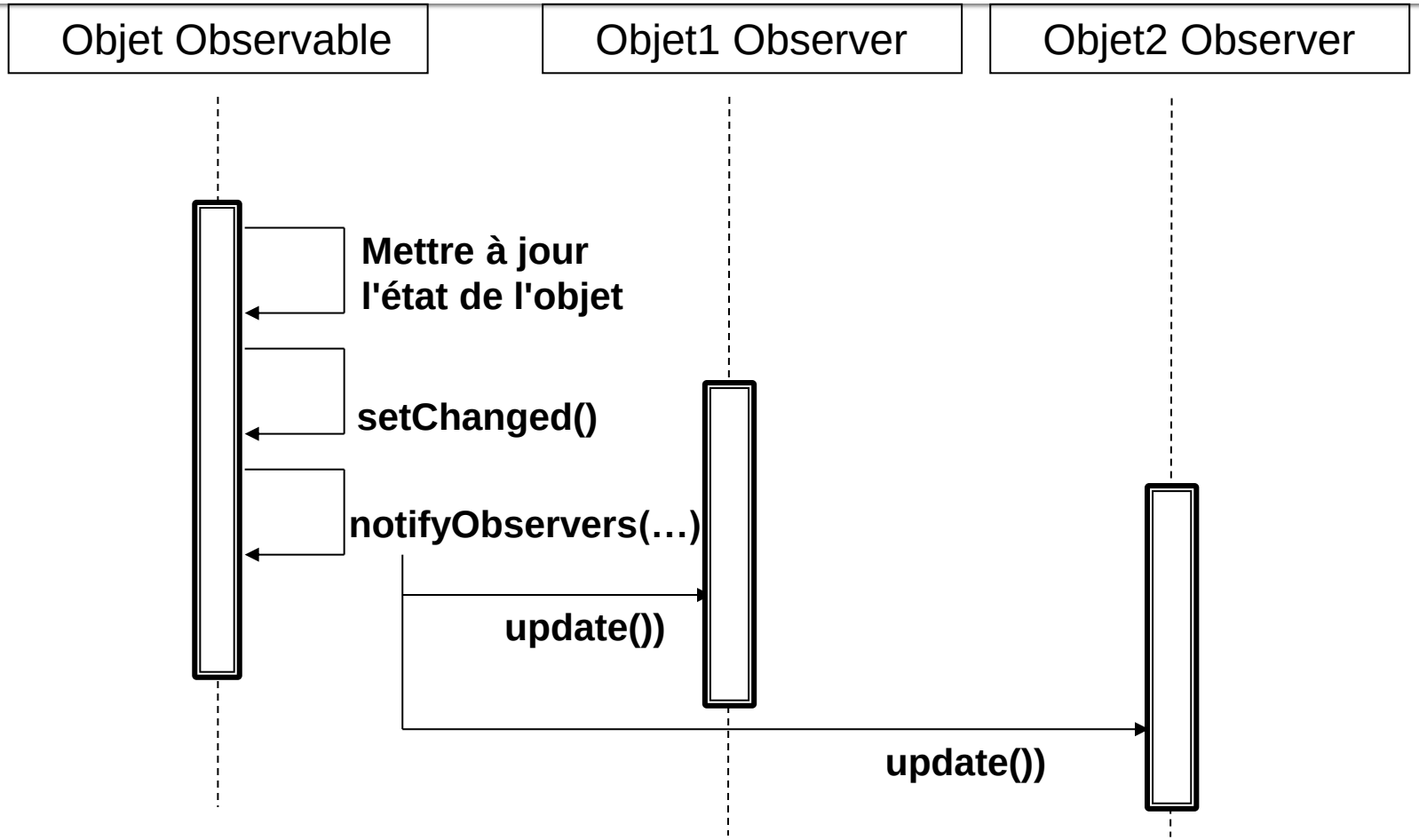
  - arg permet de passer une information de l'Observable à ses objets Observer associés : soit null, soit le paramètre d'appel de notifyObservers()

  Remarque : l'objet Observable est donc connu par l'objet Observer grâce au paramètre o de la méthode update

Observer/Observable



Observer/Observable



A faire aujourd'hui

- A partir du code de l'exemple MVC sur arche :
  - Faire le **diagramme des classes** avec le plugin ObjectAid d'Eclipse
  - Ajouter **une classe VueTexte** correspondant à une vue textuelle (P 58), pour cela des getters devront être ajoutés à la classe Balle. Modifier la classe Principale en conséquence.
  - Modifier l'interface et ajouter **une liste de choix** proposant plusieurs couleurs (rouge, vert, bleu, noir) au nord de l'interface permettant de modifier la couleur, une classe **ControleurCou**l devra être crée

- Faire le TP9