Patrons Observateur/MVC
programmation évènementielle

jean-michel Douin, douin au cnam point fr
version : 14 Septembre 2015

Notes de cours

Sommaire

- **Patron Observateur**
- ***Programmation évènementielle***
- **Patron MVC Modèle Vue Contrôleur**

Principale bibliographie utilisée

- [Grand00]
  - Patterns in Java le volume 1

    http://www.mindspring.com/~mgrand/

- [head First]
  - Head first : http://www.oreilly.com/catalog/hfdesignpat/#top

- [DP05]
  - L'extension « Design Pattern » de BlueJ : http://hamilton.bell.ac.uk/designpatterns/

- [divers]
  - Certains diagrammes UML : http://www.dofactory.com/Patterns/PatternProxy.aspx
  - informations générales http://www.edlin.org/cs/patterns.html

Patrons/Patterns pour le logiciel

- **Origine C. Alexander un architecte**
- **Abstraction dans la conception du logiciel**
  - **[GoF95] la bande des 4 : Gamma, Helm, Johnson et Vlissides**
    - **23 patrons/patterns**
- **Architectures logicielles**

Introduction : rappel

- **Classification habituelle**
  - **Créateurs**
    - **Abstract Factory, Builder, Factory Method, Prototyp,e Singleton**
  - **Structurels**
    - **Adapter, Bridge, Composite, Decorator, Facade, Flyweight, Proxy**
  - **Comportementaux**

    Chain of Responsability. Command, Interpreter, Iterator,

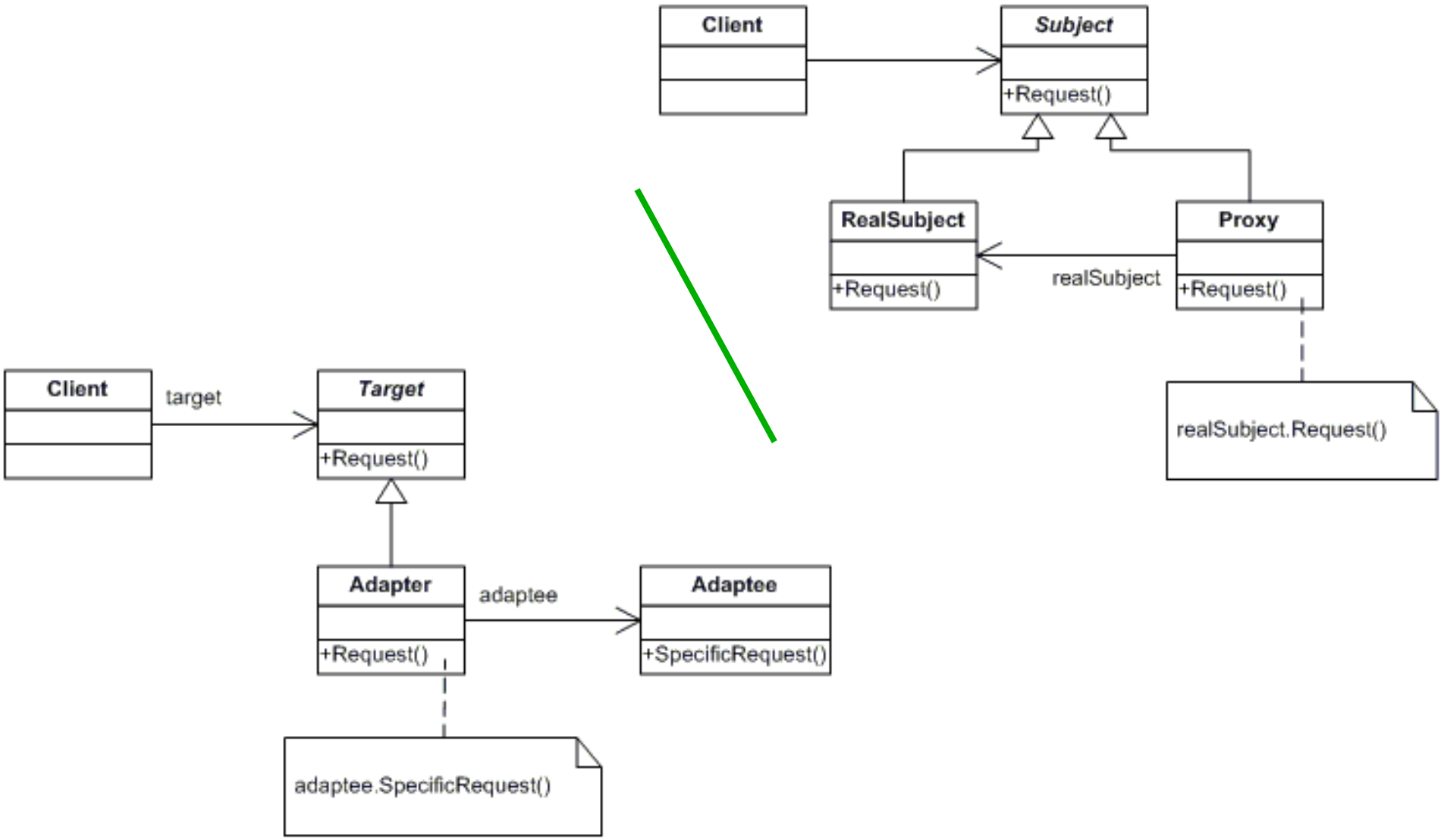
    Mediator, Memento, Observer, State,

    Strategy, Template Method, Visitor

Les patrons déjà vus …

- **Adapter**
  - **Adapte l'interface d'une classe conforme aux souhaits du client**
- **Proxy**
  - **Fournit un mandataire au client afin de contrôler/vérifier ses accès**

Adapter\Proxy



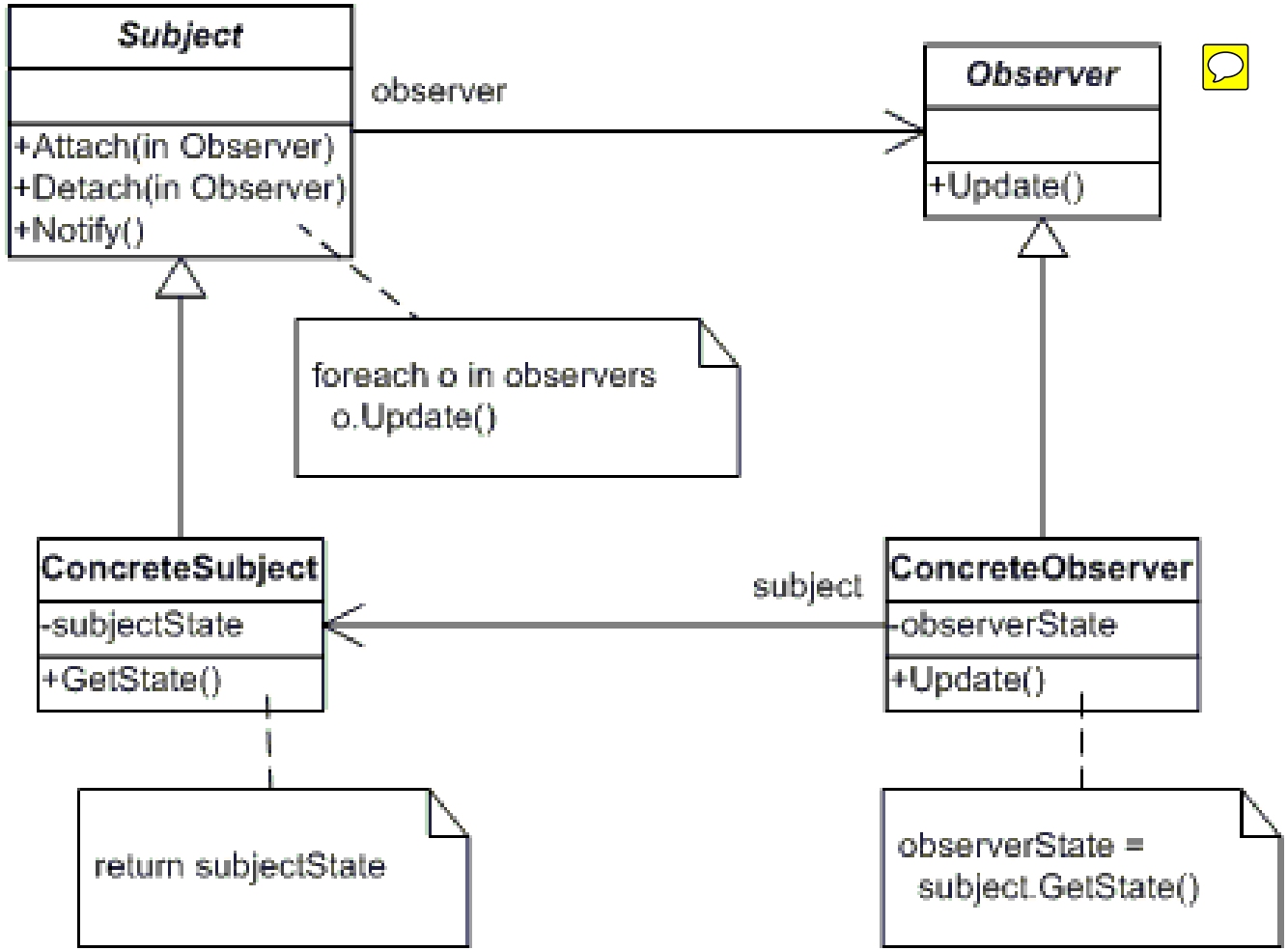
Patron Observer/observateur

- ***Notification aux observateurs inscrits***

d'un changement d'état d'une instance (Observé)

- **Un Observé**
  - **N'importe quelle instance qui est modifiée**
    - **i.e. un changement d'état comme par exemple la modification d'un attribut**
- **Les observateurs seront notifiés**
  - **A la modification de l'observé,**
  - **Synchrone, (sur la même machine virtuelle …)**
- **Un ou plusieurs Observés / un ou plusieurs observateurs**
- **Ajout et retrait dynamiques d'observateurs**

UML & le patron Observateur, l'original

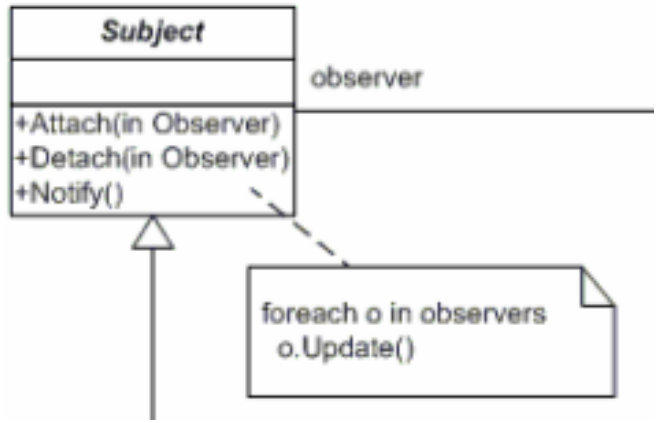


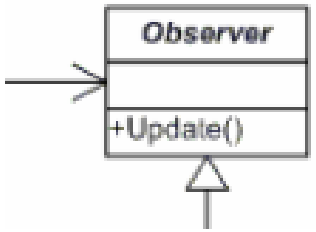
- **http://www.codeproject.com/gen/design/applyingpatterns/observer.gif**

Le patron observateur en Java

– **Lors d'un changement d'état : notification aux observateurs inscrits**

```
// les observés
public abstract class Subject{
  private List<Observer> observers …
  public  void attach(Observer o) …
  public void detach(Observer o) …

  public void notify(){
    for(Observer o : observers)
      o.update();
}
// les observateurs
public interface Observer{
  public void update();
}
```





ConcreteObservable

```
public class ConcreteSubject extends Subject{

  private … value;


  public setValue(int newValue
     this.value = newValue;
     notify()
```

ConcreteObserver

```
public class ConcreteObserver implements Observer{


   public void update(){

    // une notification a eu lieu ….
   }

}
```

Mais

Quel est l'observé initiateur ?
Quels sont les paramètres ?

Démonstration / discussion

Observer et ses paramètres …

- A chaque notification l'observé(able) est transmis

```
public interface Observer{
    public void update(Observable obs);
}
```

- A chaque notification l'observé et un paramètre sont transmis

```
public interface Observer{
    public void update(Observable obs, Object param);
}
```

Ces classes existent déjà… voir java.util

- **java.util.Observable** ***Subject***
- **java.util.Observer** ***Observer***

- **java.util.Observer** **&** **java.util.Observable**
  - **update**
  - **addObserver**
  - **removeObserver**
  - **notifyObservers**
  - **….**

java.util.Observer

```
public interface Observer{
   void update(Observable o, Object arg);
}
```

L'Observé est transmis en paramètre `Observable o`

accompagné éventuellement de paramètres `Object arg`

« update » est appelée à chaque notification

Observé/Observateurs

java.util

Class Observable

java.lang.Object
  └**java.util.Observable**

```
public class Observable
extends Object
```

This class represents an observable object, or "data" in the model-view paradigm.

java.util

Interface Observer

```
public interface Observer
```

A class can implement the `Observer` interface when it wants to be informed of changes in observable objects.

- **Prédéfinies *model-view paradigm …***

java.util.Observable

```
public class Observable{
  public void addObserver(Observer o) ;
  public void deleteObserver(Observer o) ;
  public void deleteObservers() ;
  public int countObservers() ;

  public void notifyObservers() ;
  public void notifyObservers(Object arg) ;

  public boolean hasChanged() ;
  protected  void setChanged() ;
  protected  void clearChanged() ;
}
```

Un Exemple : une liste et ses observateurs

- **Une liste est observée, à chaque modification de celle-ci, ajout, retrait, … les observateurs inscrits sont notifiés**

```
public class Liste<E> extends java.util.Observable{
  …
  public void ajouter(E e){
    … // modification effective de la liste
    setChanged();        // l'état de cette liste a changé
    notifyObservers(e); // les observateurs sont prévenus
}
```

Une liste ou n'importe quelle instance …

Un Exemple : un observateur de la liste

```
Liste<Integer> l = new Liste<Integer>();

l.addObserver( new Observer(){
      public void update(Observable o,Object arg){
         System.out.print( o + " a changé, " );
         System.out.println( arg + " vient d'être ajouté !");
      }
      });
```

C'est tout ! démonstration

Démonstration/discussion

EventListener / comme Observer

- **java.awt.event.EventListener**
  - Les écouteurs sont des observateurs

- **Convention syntaxique adoptée par les API JAVA**

  - XXXXXListener extends EventListener

    Ajout d'un écouteur sera addXXXXXListener

    exemple l'interface ActionListener → addActionListener

- **EventObject comme ActionEvent**

Observateur comme XXXXListener

java.util

Interface EventListener

All Known Subinterfaces:

Action, ActionListener, AdjustmentListener, AncestorListener, AWTEventListener, BeanContextMembershipListener, BeanContextServiceRevokedListener, BeanContextServices, BeanContextServicesListener, CaretListener, CellEditorListener, ChangeListener, ComponentListener, ConnectionEventListener, ContainerListener, ControllerEventListener, DocumentListener, DragGestureListener, DragSourceListener, DragSourceMotionListener, DropTargetListener, FlavorListener, FocusListener, HandshakeCompletedListener, HierarchyBoundsListener, HierarchyListener, HyperlinkListener, IIOReadProgressListener, IIOReadUpdateListener, IIOReadWarningListener, IIOWriteProgressListener, IIOWriteWarningListener, InputMethodListener, InternalFrameListener, ItemListener, KeyListener, LineListener, ListDataListener, ListSelectionListener, MenuDragMouseListener, MenuKeyListener, MenuListener, MetaEventListener, MouseInputListener, MouseListener, MouseMotionListener, MouseWheelListener, NamespaceChangeListener, NamingListener, NodeChangeListener, NotificationListener, ObjectChangeListener, PopupMenuListener, PreferenceChangeListener, PropertyChangeListener, RowSetListener, RowSorterListener, SSLSessionBindingListener, StatementEventListener, TableColumnModelListener, TableModelListener, TextListener, TreeExpansionListener, TreeModelListener, TreeSelectionListener, TreeWillExpandListener, UndoableEditListener, UnsolicitedNotificationListener, VetoableChangeListener, WindowFocusListener, WindowListener, WindowStateListener

- **Une grande famille !**

Une IHM et ses écouteurs

- **Chaque item est un sujet observable … avec ses écouteurs…**
  - **Pour un « Bouton », à chaque clic les écouteurs/observateurs sont prévenus**

```
public class Button extends Component{
  …
  public void addActionListener(ActionListener al){

  }
```

Un bouton prévient ses écouteurs …

Une instance de la classe java.awt.*Button* (observable)
notifie à
ses instances inscrites java.awt.event.Action*Listener* …(observer)

```
Button b = new Button("empiler");
b.addActionListener(unEcouteur);       // 1
b.addActionListener(unAutreEcouteur); // 2
b.addActionListener(
      new ActionListener(){            // 3 écouteurs
        public void actionPerformed(ActionEvent ae){
          System.out.println("clic !!! ");
        }
      });
```

Un écouteur comme Action Listener

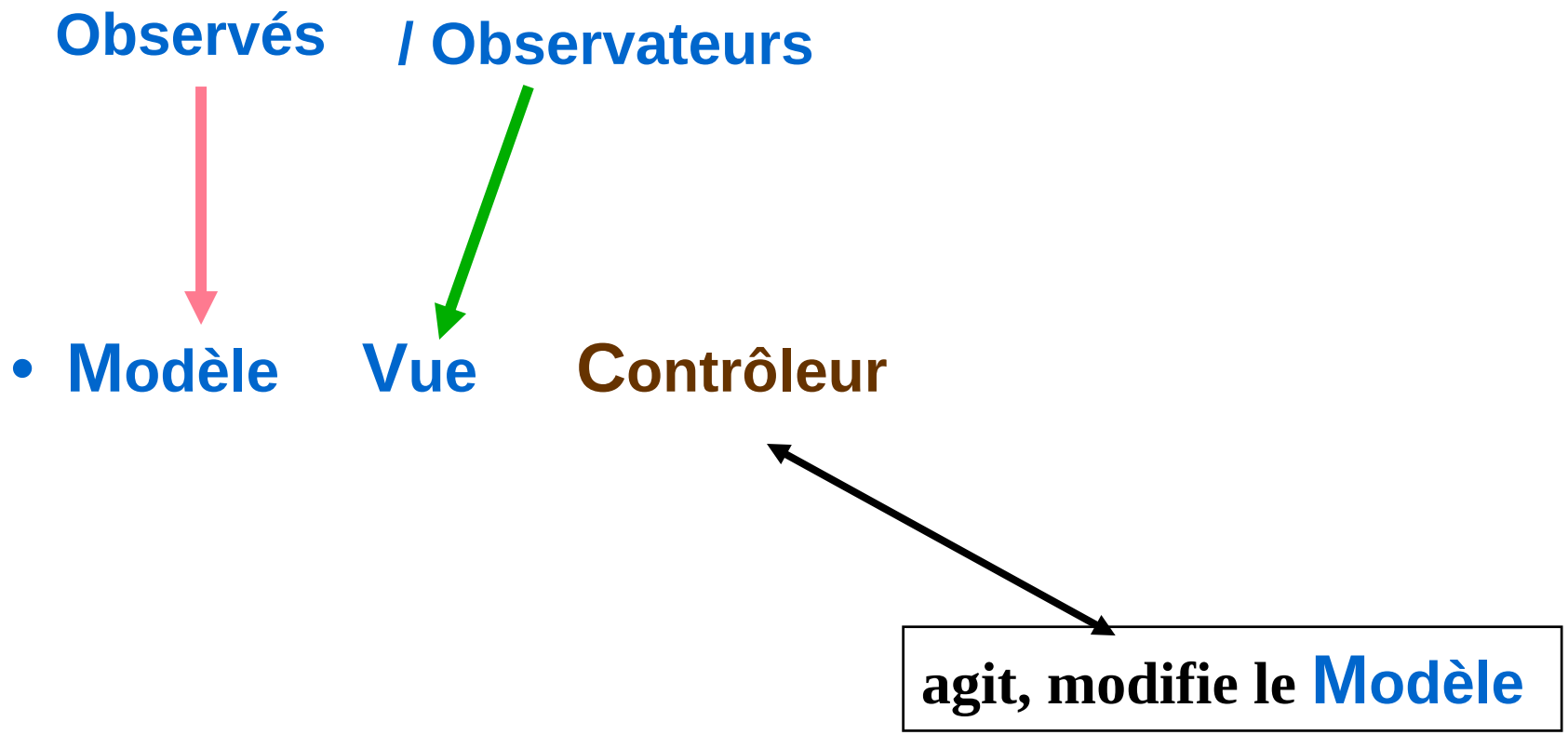
```
import java.util.event.ActionListener;
import java.util.event.ActionEvent;

public class EcouteurDeBouton
  implements ActionListener{

  public void actionPerformed(ActionEvent e){
   // traitement  à chaque action sur le bouton
  }

}

//c.f. page précédente
ActionListener unEcouteur = new EcouteurDeBouton();
b.addActionListener(unEcouteur);      // 1
```

Démonstration / Discussion

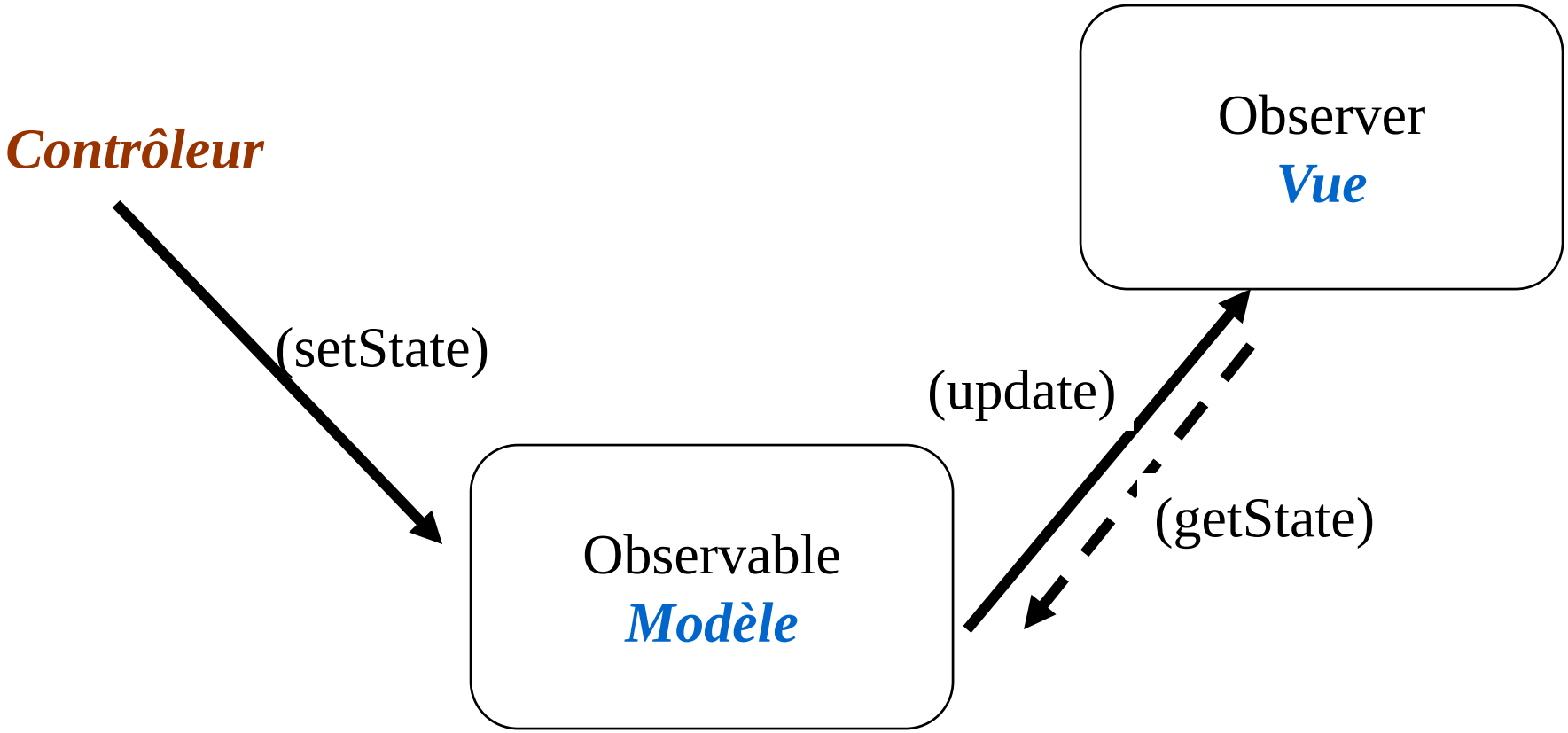
API Java, patron Observateur, un résumé

- **Ajout/retrait dynamiques des observateurs ou écouteurs**
- **L'observable se contente de notifier**
  - **Notification synchrone à tous les observateurs inscrits**
- **API prédéfinies java.util.Observer et java.util.Observable**

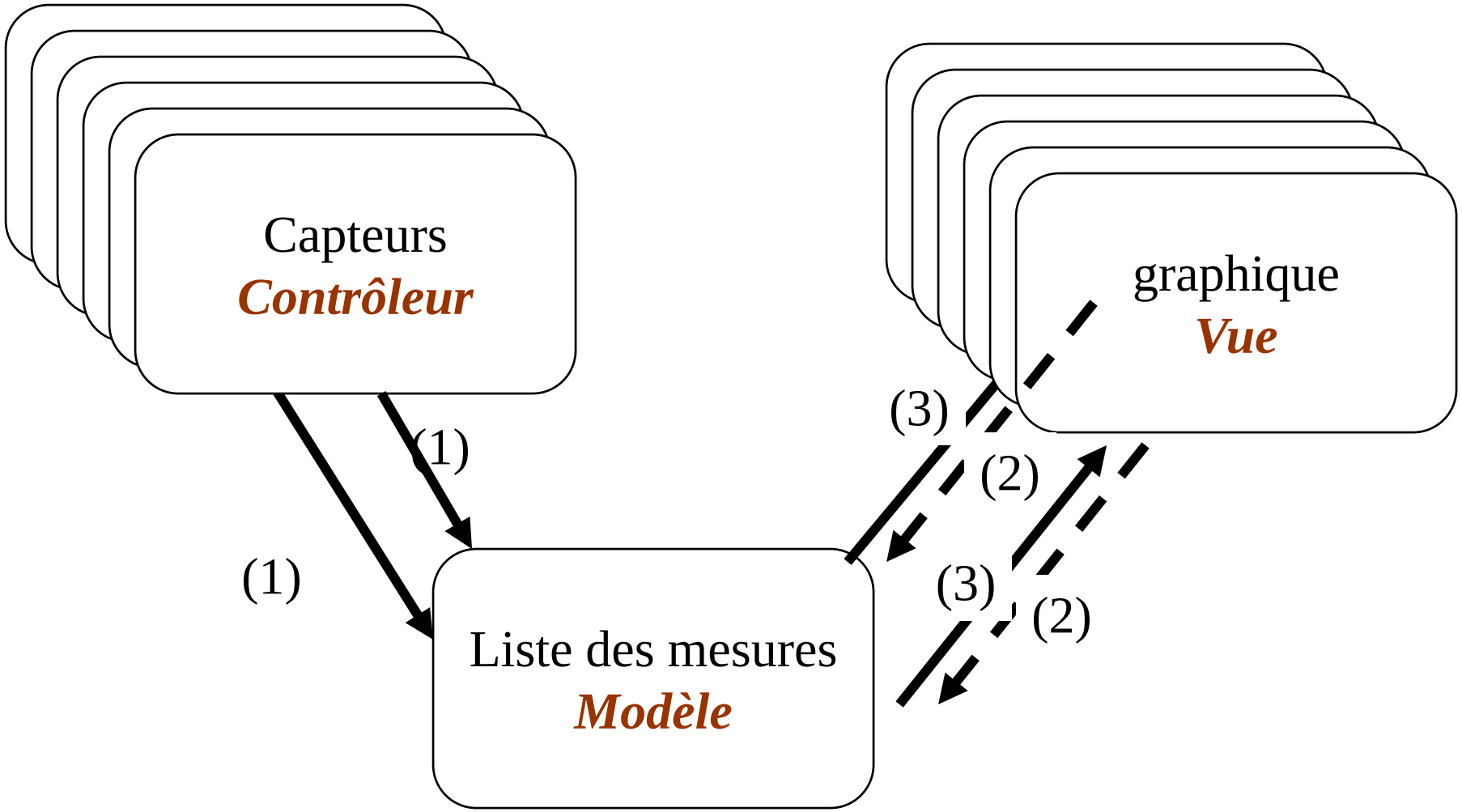
patrons Observer / MVC



MVC : Observer est inclus

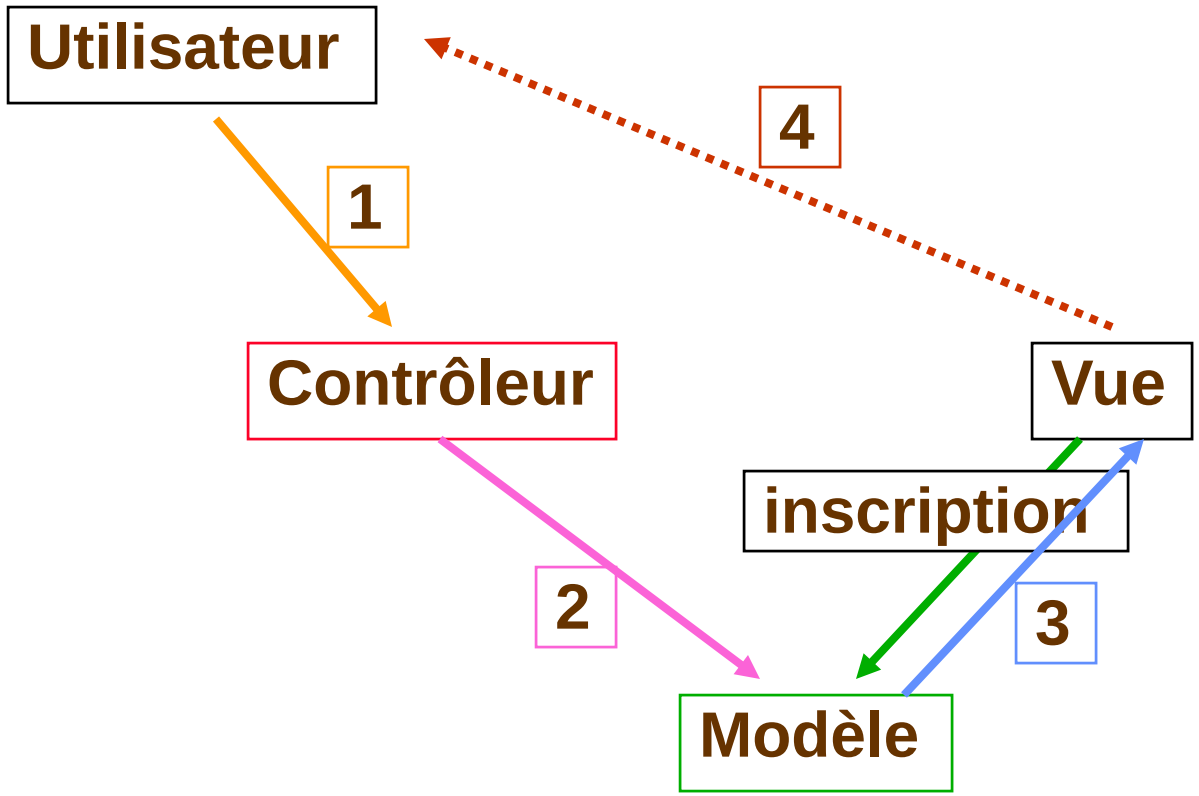


MVC Avantages

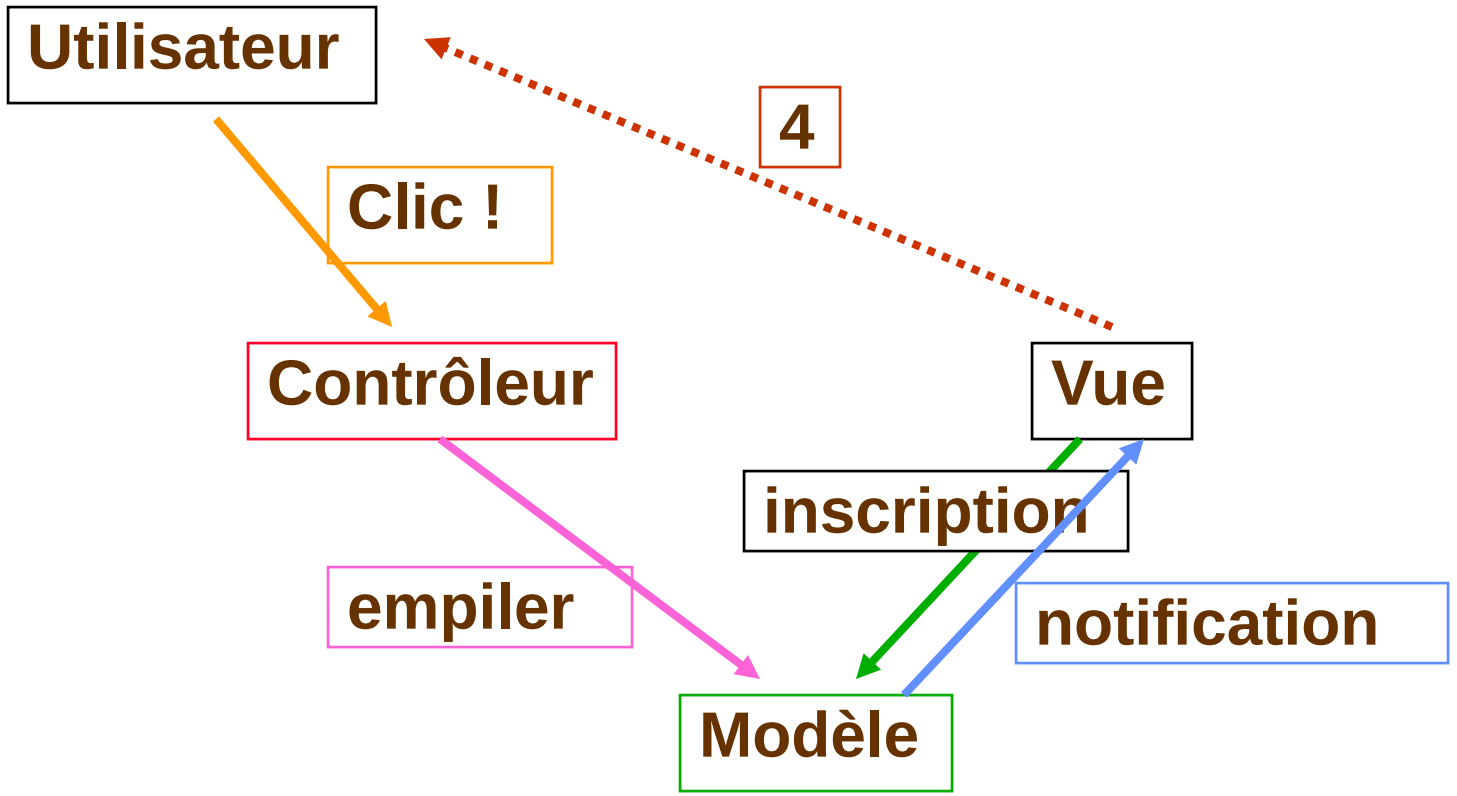


- **Souple ?**

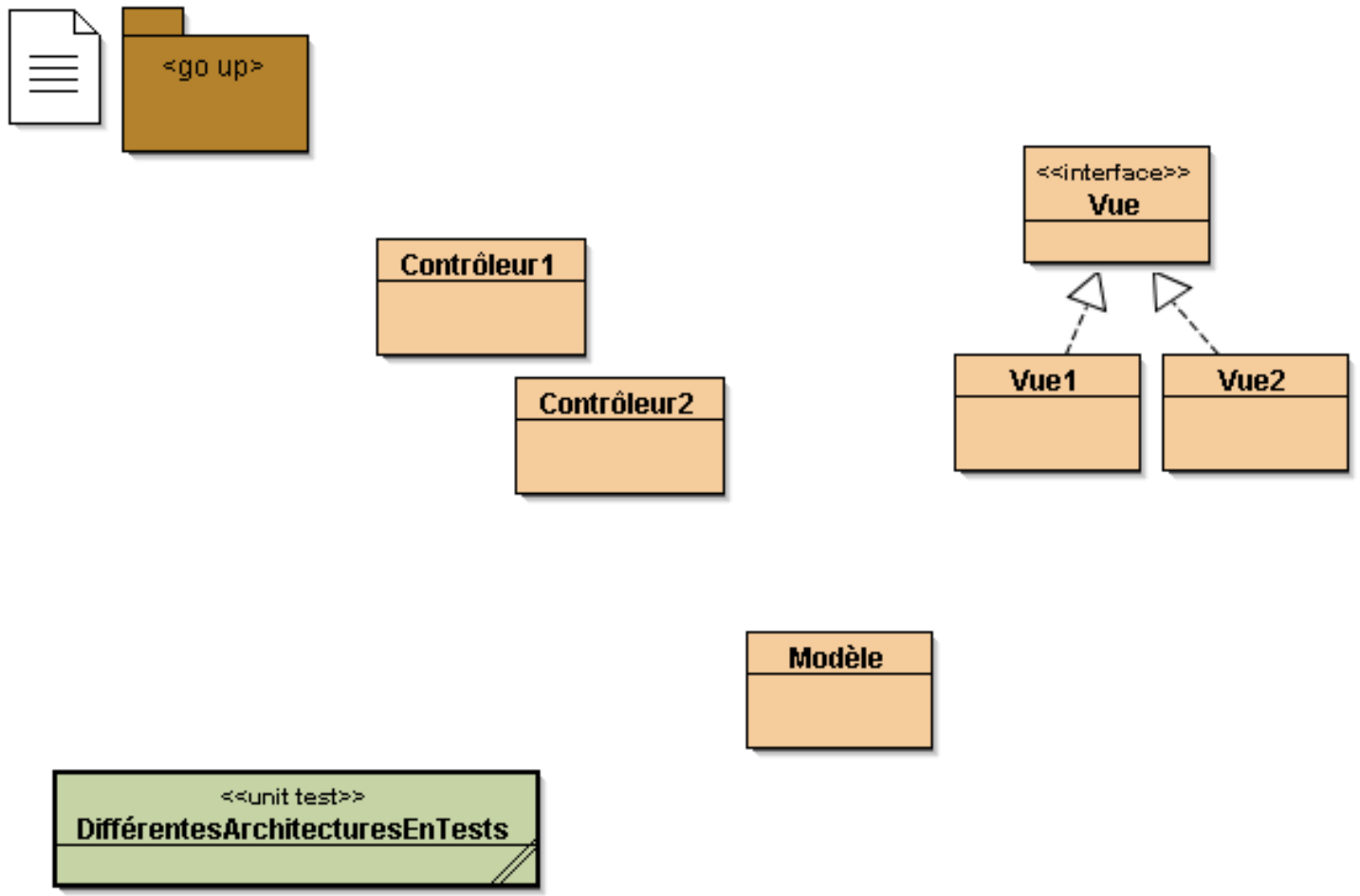
Un cycle MVC



Un cycle MVC, le bouton empiler



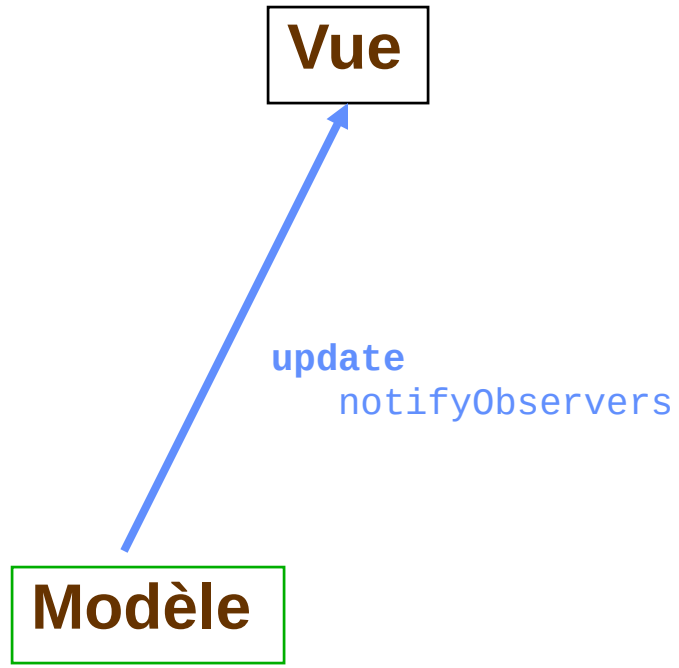
Démonstration / MVC en pratique



- **Un Modèle**
  - **Plusieurs Contrôleurs**
  - **Plusieurs Vues**

Démonstration : le Modèle i.e. un Entier

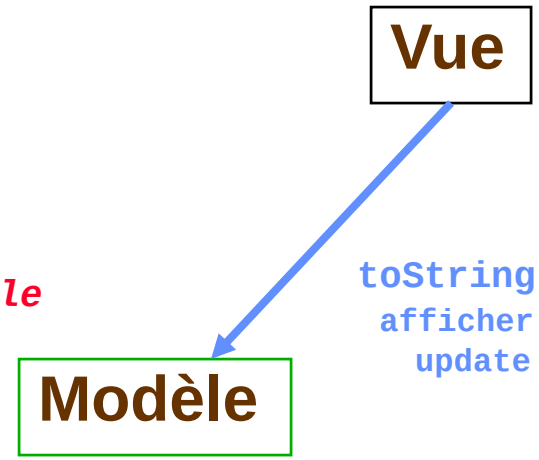
```
import java.util.Observable;

public class Modèle extends Observable{

  private int entier;

  public int getEntier(){
    return entier;
  }

  public String toString(){
    return "entier : " + entier;
  }

  public void setEntier(int entier){
    this.entier = entier;
    setChanged();
    notifyObservers(entier);
  }
}
```



Démonstration : une Vue

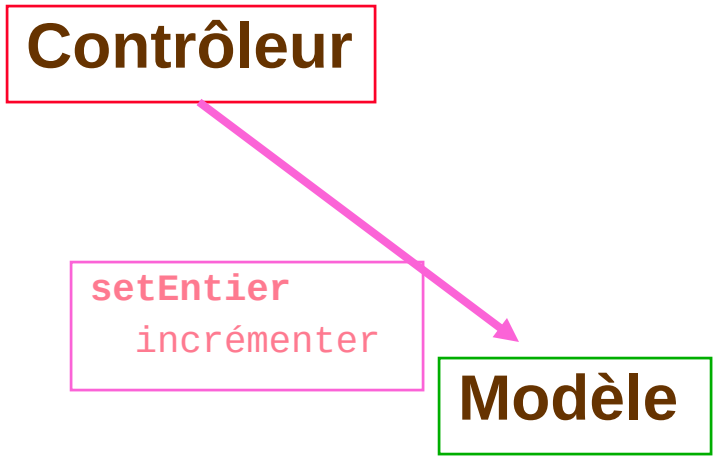
```
public interface Vue{
  public void afficher();
}
```

```
import java.util.Observable;
import java.util.Observer;

public class Vue1 implements Vue, Observer{
  private Modèle modèle;

  public Vue1( Modèle modèle){  // inscription auprès du modèle
    this.modèle = modèle;
    modèle.addObserver(this);
  }

  public void afficher(){
    System.out.println(" Vue1 : le modèle a changé : " + modèle.toString());
  }

  public void update(Observable o, Object arg){   // notification
    if(o==modèle) afficher();
  }

}
```



Démonstration : un contrôleur

```
public class Contrôleur1{
  private Modèle modèle;

  public Contrôleur1(Modèle modèle){
    this.modèle = modèle;
  }

  public void incrémenter(){
    modèle.setEntier(modèle.getEntier() +1);
  }

}
```



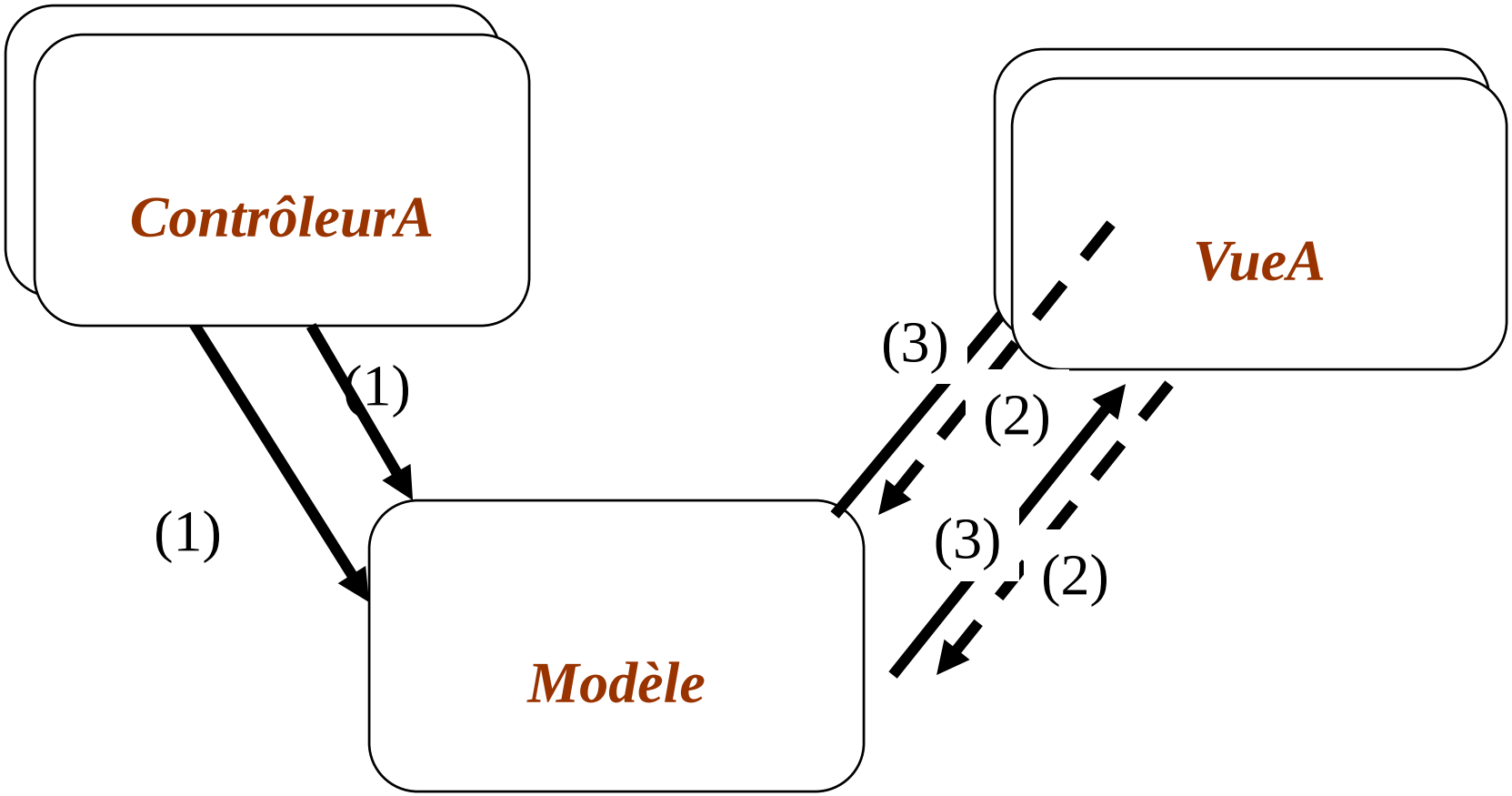
Un modèle, une vue, un contrôleur

```
    // Un Modèle
    Modèle modèle = new Modèle();

    // Ce modèle possède une vue
    Vue vue = new Vue1(modèle);

    // un Contrôleur ( exécute des méthodes du modèle)
    Contrôleur1 contrôleur = new Contrôleur1(modèle);

    contrôleur.incrémenter();
    contrôleur.incrémenter();
}
```

Un modèle, deux vues, deux contrôleurs

```
// Un Modèle
Modèle modèle = new Modèle();

// deux vues
Vue vueA = new Vue1(modèle);
Vue vueB = new Vue1(modèle);

// 2 Contrôleurs
Contrôleur1 contrôleurA = new Contrôleur1(modèle);
Contrôleur1 contrôleurB = new Contrôleur1(modèle);

contrôleurA.incrémenter();
contrôleurB.incrémenter();
```

Discussion



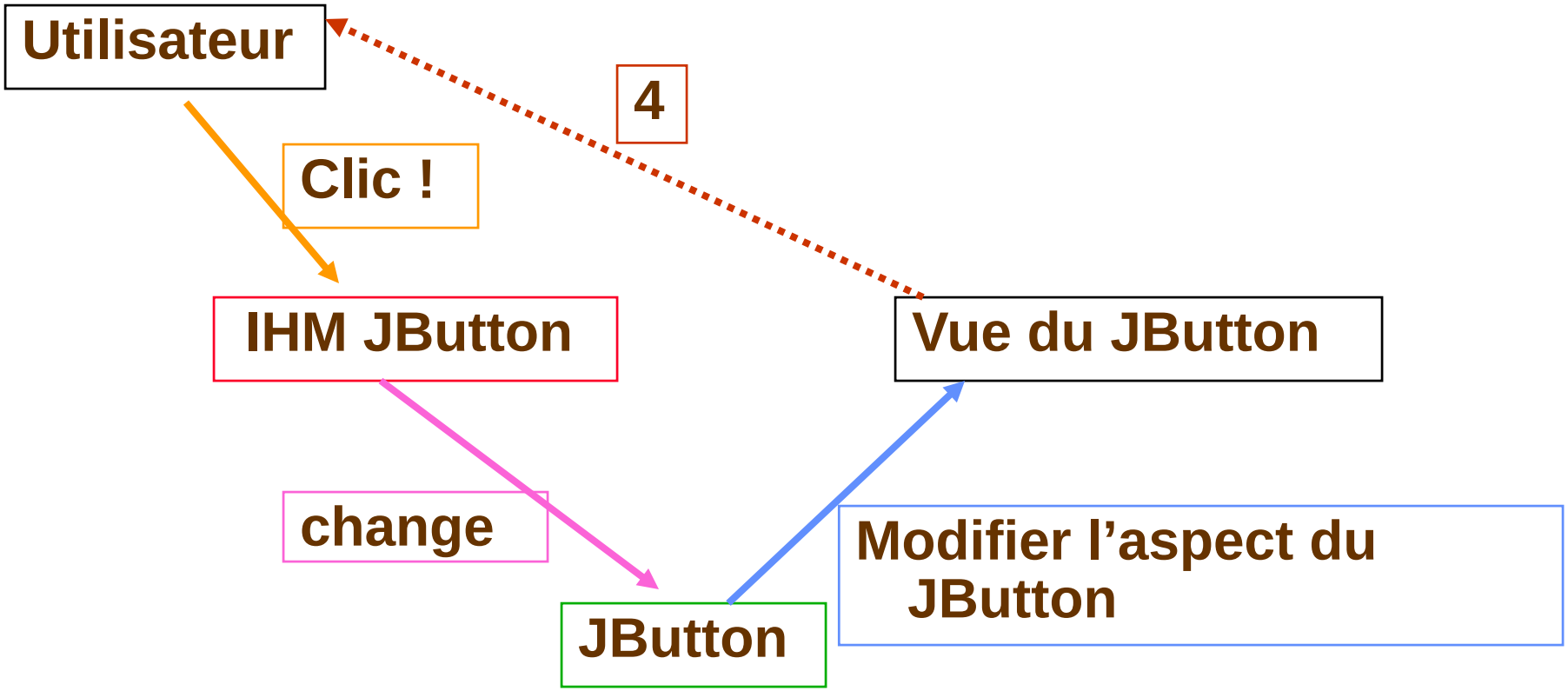
AWT / Button, discussion

- **Un « Button » (le contrôleur) contient un MVC**
  - **À part entière**

- **Text, TextField, Label, … « sont » des Vues**
- **Button, Liste, … « sont » des contrôleurs**

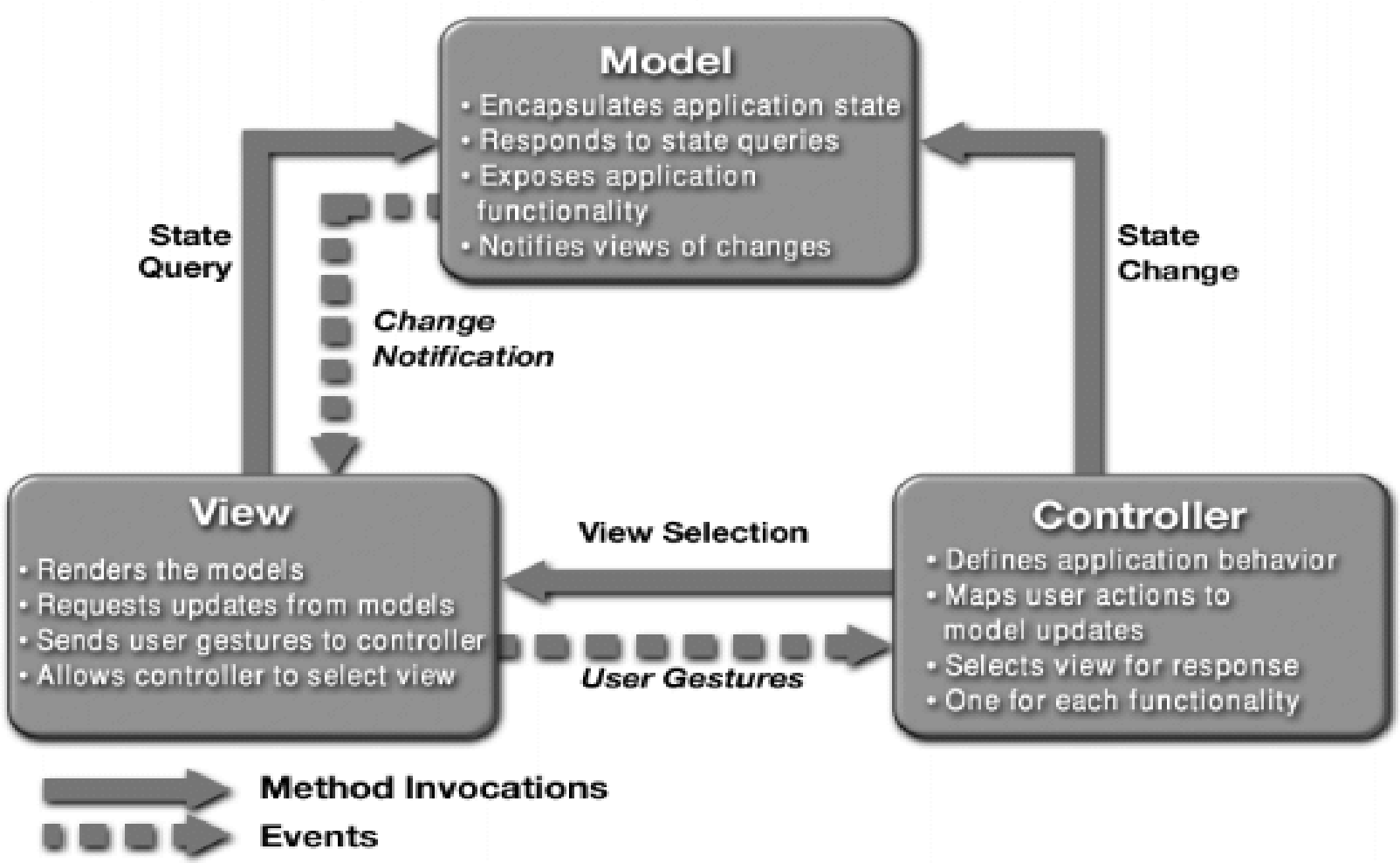
- **Une IHM (JApplet,…) contient la Vue et le Contrôle**
  - **Alors le compromis architecture/lisibilité est à rechercher**

Un JButton comme MVC



- Au niveau applicatif appel de tous les observateurs inscrits
  - actionPerformed(ActionEvent ae), interface ActionListener

MVC doc de Sun



- **http://java.sun.com/blueprints/patterns/MVC-detailed.html**

MVC un autre schéma …

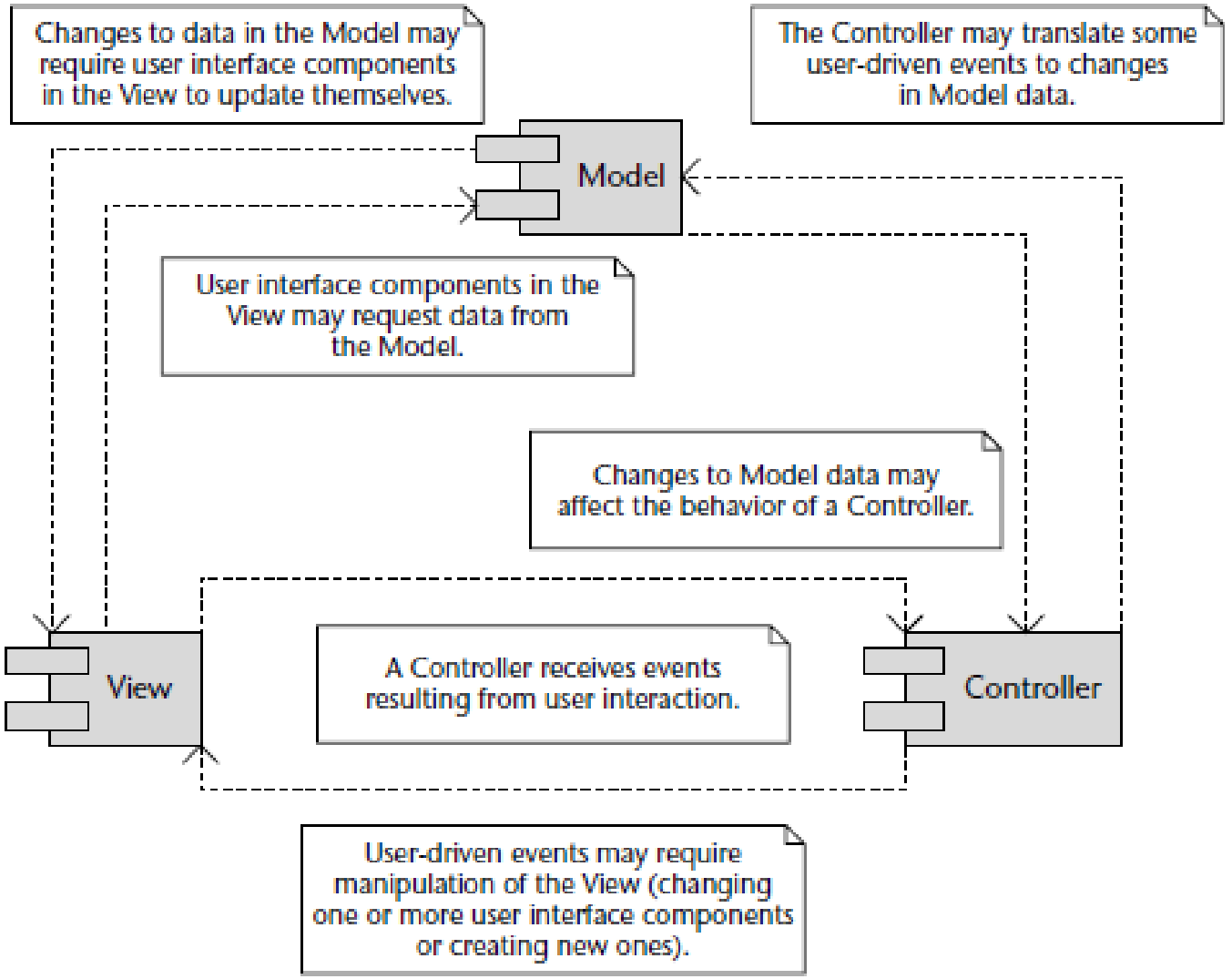
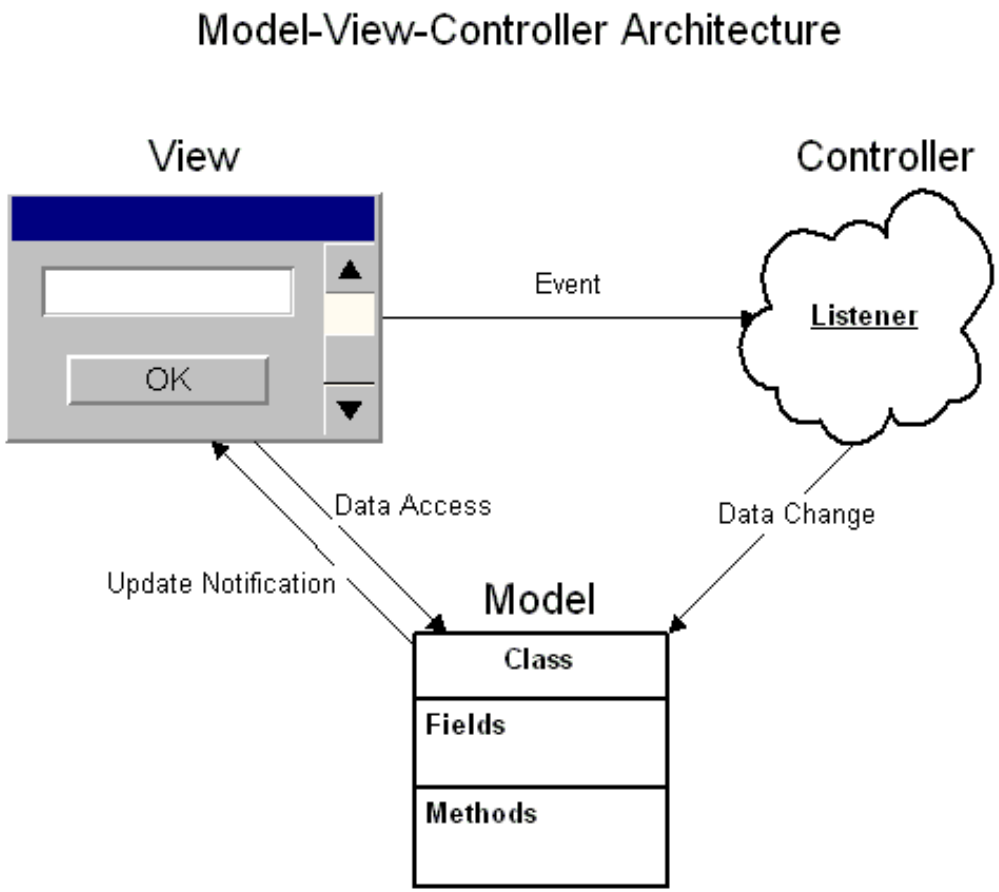


Figure 1.1 MVC Pattern structure.

src : Mastering JSF page 6

IHM et MVC assez répandu …



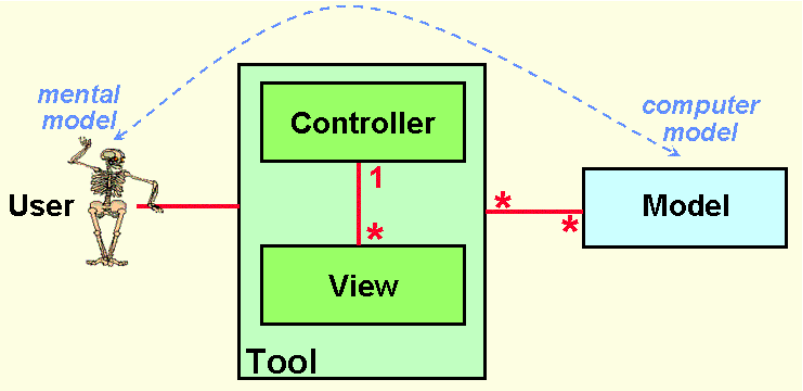
- **Discussion**
  - **Evolution, maintenance, à la recherche du couplage faible**
  - **Exemple**
    - **peut-on changer d'IHM ?, peut-elle être supprimée ?**
    - **peut-on placer le modèle sur une autre machine ? …**

Conclusion

- **MVC**
  - **Incontournable**
  - **Couplage faible induit**
  - **Intégration du patron Observateur**

Modèle Vue Contrôleur (MVC) est une méthode de conception pour le développement d'applications logicielles qui sépare le modèle de données, l'interface utilisateur et la logique de contrôle. Cette méthode a été mise au point en 1979 par Trygve Reenskaug, qui travaillait alors sur Smalltalk dans les laboratoires de recherche Xerox PARC[1].

–Extrait de http://fr.wikipedia.org/wiki/MVC



Donner l'illusion à l'utilisateur de manipuler les données du modèle

•**L'original en 1979 http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html**

Annexe

- **Un exemple au complet**
  - **Une démonstration**
  - **http://jfod.cnam.fr/cours2_mvc_swing/**
- **Un Bean ?**
- **Discussion possible autour d'un TP possible**

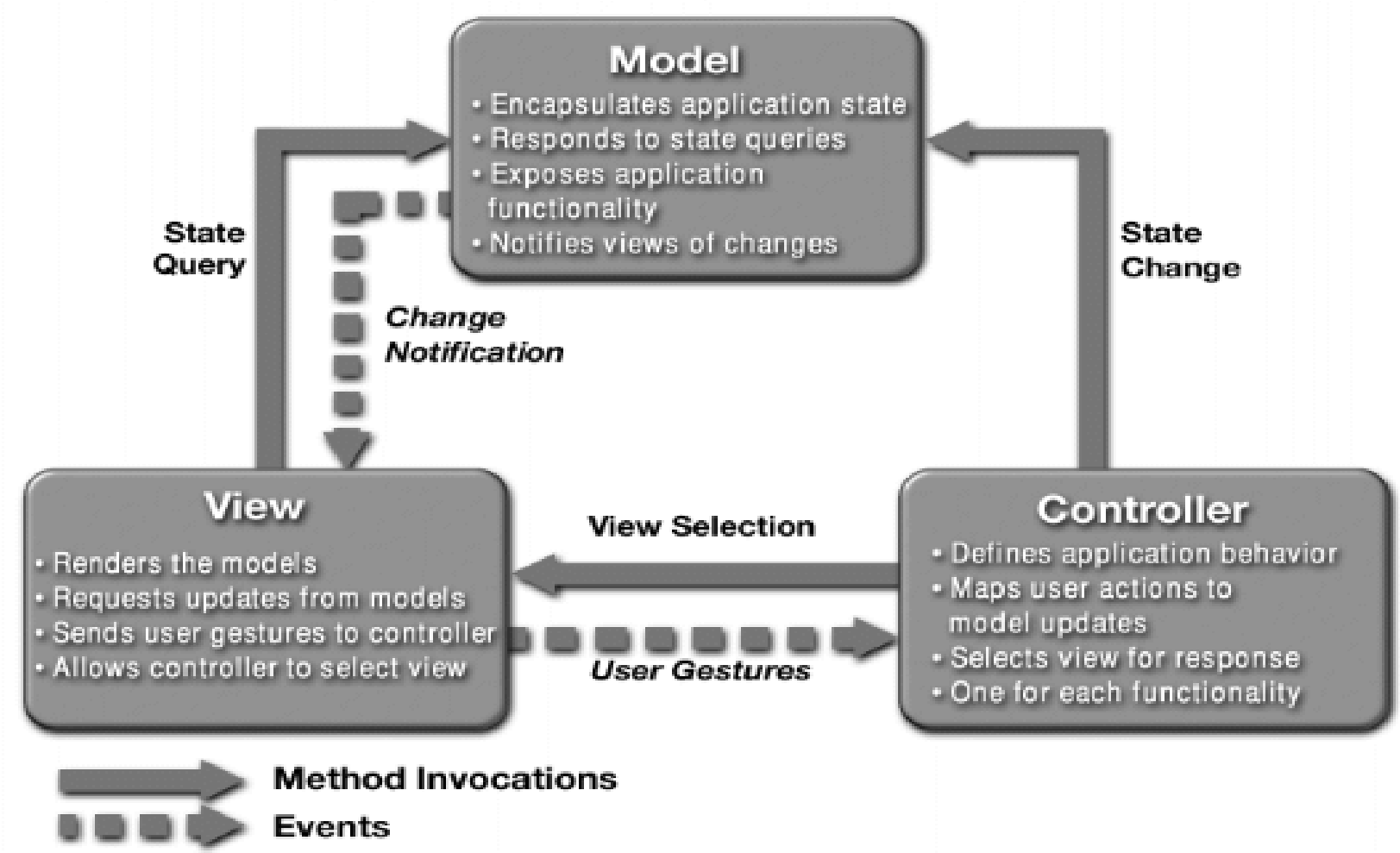
Interactivité, MVC

- **L'interface graphique**
  - **La vue**
- **Interactions avec l'utilisateur**
  - **Le contrôle**
- **Les données de l'application**
  - **Le modèle**
- **A la recherche**

d'une adéquation des données de l'application et des vues

  - **Paradigme MVC**
    - **Modèle Vue Contrôleur**

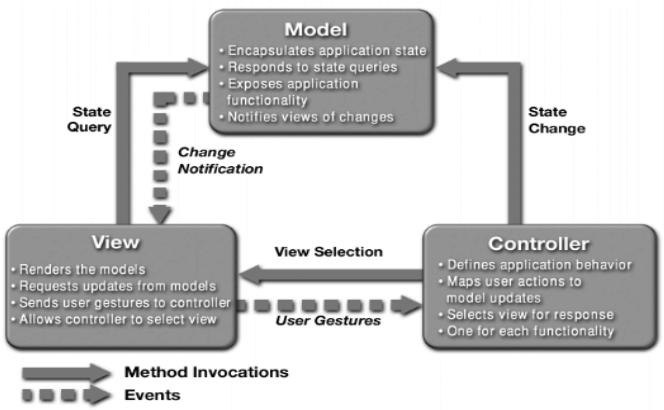
MVC, doc de Sun

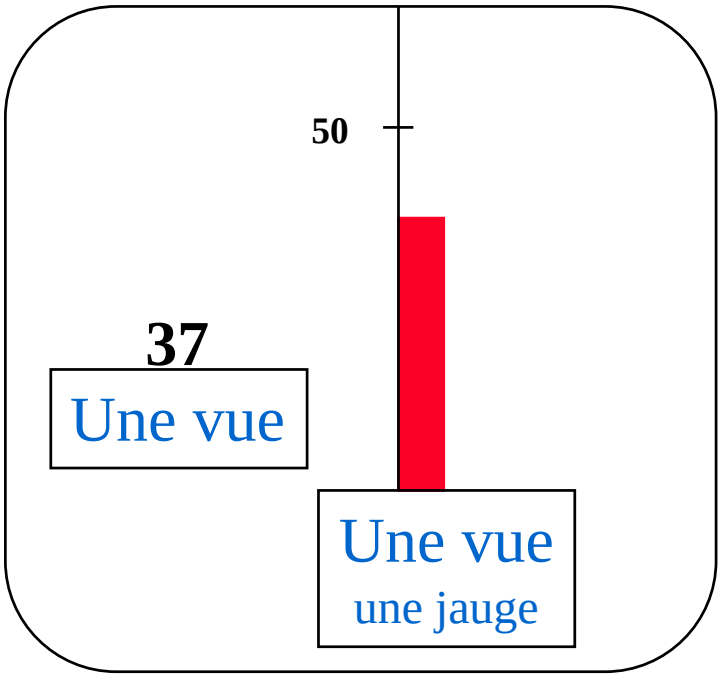


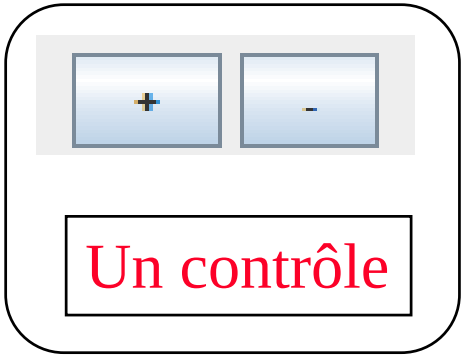
- **Un exemple …**

MVC un exemple : un nombre !

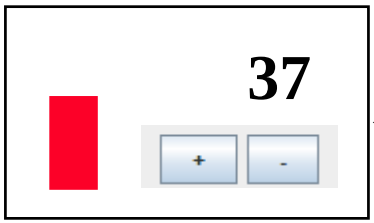
le modèle : un nombre





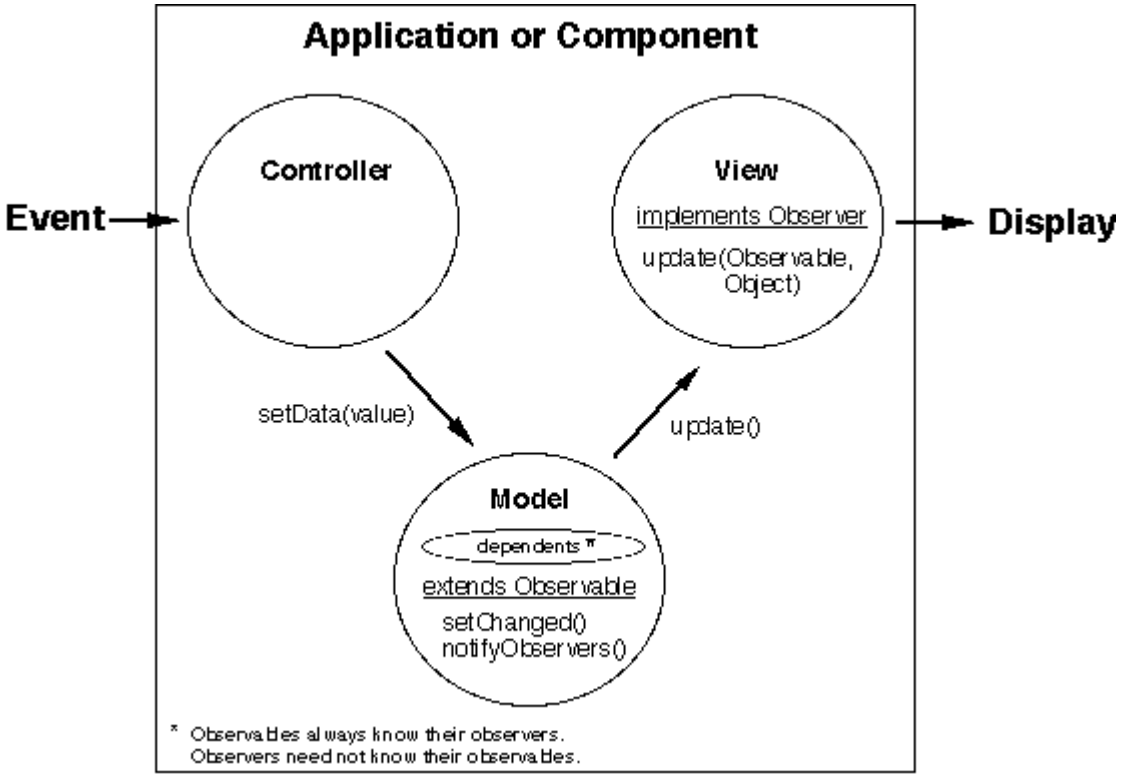


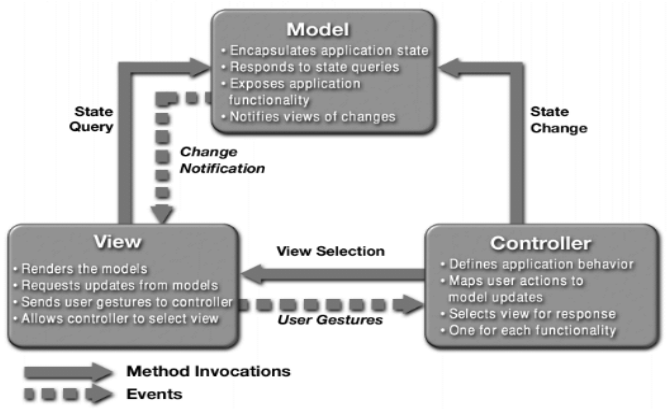
- **Déploiement ?**
  - **IHM ?**
  - **Contrôle ?**
  - **Modèle ?**



Une JFrame

Déploiement ? Choix de classes

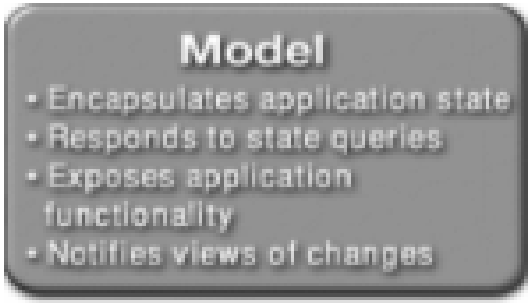




- **Discussion …**
  - **Le modèle :** **la classe Nombre**
  - **La Vue :** **Une JFrame, une jauge, un affichage, des boutons**
  - **Le Contrôle :** **Réactions aux actions de l'utilisateur …**

Un déploiement possible, démonstration

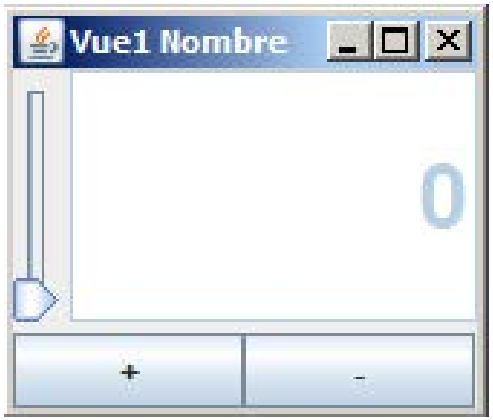
- **La classe Nombre est Observable**
  - **Elle hérite de java.util.Observable**



- **Les Vues sont des observateurs**
  - **Elles implémentent java.util.Observer,**
  - **Peuvent être des IHM, des « Container » swing,**
  - **Sans interface : des vues sans être vues ?…**
    - **Par exemple : un journal des évènements**
  - **…**

- **Les Contrôles gèrent les actions de l'utilisateur**
  - **Elles implémentent les XXXXListener des composants swing**
  - **Une classe par action ?**

Exemple un nombre, une vue, un contrôle



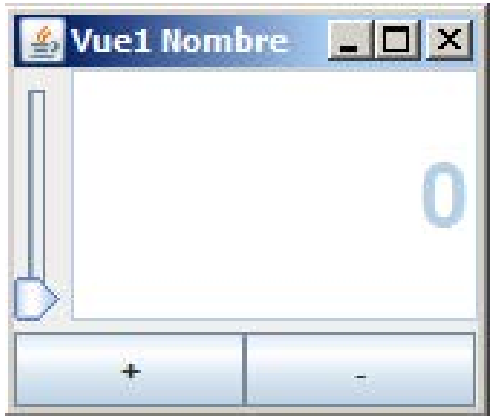

```
public class MainNombreMVC{

  public static void main(String[] args){
     Nombre nombre = new Nombre(0,10); // le modèle

     Vue1 vue1 = new Vue1(nombre);

     ControleVue1 controle1 = new ControleVue1(nombre, vue1);

  }
}
```

Le modèle

```
public class Nombre extends java.util.Observable{
  public final int VALEUR_MIN;
  public final int VALEUR_MAX;
  private int valeur;

  public Nombre(int min, int max){
   this.VALEUR_MIN = this.valeur = min;
   this.VALEUR_MAX = max;
  }

  public void inc(){
    if(valeur < VALEUR_MAX){
      this.valeur++;
      setChanged();
      notifyObservers();
    }
  }

  public void dec(){ // idem -- …
  … getValeur() + setValeur(…) + toString()
```

La Vue est une JFrame et implémente Observer

```
public class Vue1 extends JFrame implements Observer{

  private JButton     plus;
  private JButton     moins;
  private JTextField  valeur;
  private JSlider     jauge;

  public Vue1(Nombre nombre) {
    super("Vue1 Nombre");
     nombre.addObserver(this);
    // Mise en place des composants graphiques
```



en quelques lignes …,
un arbre instance du composite

```
    pack();
    setVisible(true);
  }
```

La Vue implémente java.util.Observer

```
@Override
  public void update(Observable obs, Object arg){
  if(obs instanceof Nombre){
      Nombre n = (Nombre)obs;
      this.valeur.setText(n.toString());
      this.jauge.setValue(n.getValeur());
    }
  }


// accesseurs
  public JButton getPlus(){return plus;}
  public JButton getMoins(){return moins;}

}
```

Le Contrôle implémente les XXXXListener

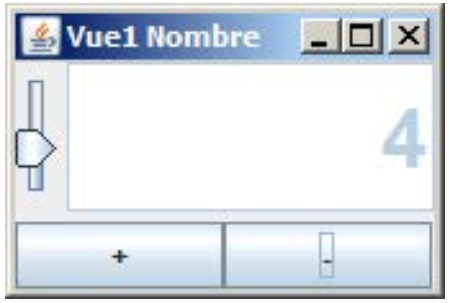
```
public class ControleVue1{
  private Nombre nombre;

  public ControleVue1(Nombre nombre, Vue1 vue){
    this.nombre = nombre;
    vue.getMoins().addActionListener(
      new ActionListener(){
        public void actionPerformed(ActionEvent ae){
          ControleVue1.this.nombre.dec();
        }
      });

    vue.getPlus().addActionListener(
      new ActionListener(){
        public void actionPerformed(ActionEvent ae){
          ControleVue1.this.nombre.inc();
        }
      });

  }
}
```

adéquation actions de l'utilisateur / opérations sur le modèle

Démonstration

- **Démonstration suite**
- **MVC pourquoi faire ?**
- **Couplage faible induit**
  - **Maintenance**

Une nouvelle vue, un nouveau contrôle, critiques




```
public class MainNombreMVC{

  public static void main(String[] args){
    Nombre nombre = new Nombre(0,10); // le modèle

    Vue1 vue1 = new Vue1(nombre);
    ControleVue1 controle1 = new ControleVue1(nombre, vue1);

    Vue2 vue2 = new Vue2(nombre);
    ControleVue2 controle2 = new ControleVue2(nombre, vue2);

  }
}
```

l'utilisateur peut maintenant modifier la valeur …

ajout d'un observateur auprès d'un JTextField
soit jTextField.addActionListener…

La classe Vue2, est un observateur

```
public class Vue2 extends JFrame implements Observer{

  private JTextField  valeur;
  private JSlider     jauge;

  public Vue2(Nombre nombre) {
    super("Vue2 Nombre");
    nombre.addObserver(this);
    Container content = getContentPane();
```

en quelques lignes …

```
  }

  public void update(Observable obs, Object arg){
    if(obs instanceof Nombre){  // instanceof prévention …
      Nombre n = (Nombre)obs;
      this.valeur.setText(n.toString());
      this.jauge.setValue(n.getValeur());
    }
  }
  public JTextField getValeur(){return valeur;}}
```

Le Contrôle2 de la vue 2, implements XXXXListener

```
public class ControleVue2{
  private Nombre nombre;
  private Vue2   vue2;

  public ControleVue2(Nombre nombre, Vue2 v){
    this.nombre = nombre;
    this.vue2 = v;

    vue2.getValeur().addActionListener(
      new ActionListener(){
        public void actionPerformed(ActionEvent ae){
          try{
            int valeur = Integer.parseInt(vue2.getValeur().getText());
            ControleVue2.this.nombre.setValeur(valeur);
          }catch(NumberFormatException nfe){
          }

        }
      });
  }
}
```

Une nouvelle vue engendre ici une nouvelle classe de contrôle, critiques ? Faire autrement ? mieux ?

Deux nombres … 4 vues, 4 Contrôles

```
public class MainNombreMVC{

  public static void main(String[] args){
    Nombre nombre = new Nombre(0,10); // le modèle
    Vue1 vue1 = new Vue1(nombre);
    ControleVue1 controle1 = new ControleVue1(nombre, vue1);
    Vue2 vue2 = new Vue2(nombre);
    ControleVue2 controle2 = new ControleVue2(nombre, vue2);
    Vue1 vue11 = new Vue1(nombre);
    ControleVue1 controle11 = new ControleVue1(nombre, vue11);

    Nombre nombreBis = new Nombre(0,100);
    Vue1 vueBis = new Vue1(nombreBis);
    ControleVue1 controle = new ControleVue1(nombreBis, vueBis);

  }
}
```

Démonstration …

Un dernier exemple : deux nombres … 8 vues, 25 Contrôles, non merci

Discussions

Modèle et Bean

- **Un JavaBean ?**

- **Observé/Observateur de Bean …**
  - **Vue**

Un bean ?

- **Un Bean est avant tout une classe …**
  - ***Un bean est un (extends) POJO, une classe quelconque***

Avec

- **Le respect de certaines conventions**
  - **implements Serializable**
  - **Un constructeur par défaut**
  - **Un *getter* et/ou un *setter* pour chaque variable d'instance**
  - **firePropertyChange au sein du setter**
  - **Simple n'est-ce pas ?**
- **La suite**
  - **Nombre devient Bean**
  - **Usage de l'introspection**

Discussion sur l'implémentation du Modèle, un Bean ?

- **public class Nombre …**

- **Le Modèle est une classe**
  - **Cette classe hérite de java.util.Observable**

- **Ce modèle peut devenir un composant logiciel**
  - **Un JavaBean**
  - **Peu de différences à part quelques règles d'écritures**
    - **Ces règles permettront une utilisation de ce composant par des outils d'introspection,**
    - **Vers une meilleure réutilisation ?**
    - **Vers une industrie du composant logiciels ?**
      - **EJB ? Enterprise JavaBeans**

        » Un nouveau langage au dessus de Java?

Observable/Observer *à la mode* JavaBeans

- **Paquetage java.beans**
- **Champs d'instance**
  - **int valeur comme property**
- **Observer comme PropertyChangeListener**
  - **public void update(Observable obs, Object o)**

  comme
  - **public void propertyChange(PropertyChangeEvent evt)**
- **Observable comme PropertyChangeSupport**
  - **void notifyObservers (Object o)**

  comme
  - **void firePropertyChange (String property,Object oldValue,Object newValue)**

La classe Nombre devient Bean : NombreBean

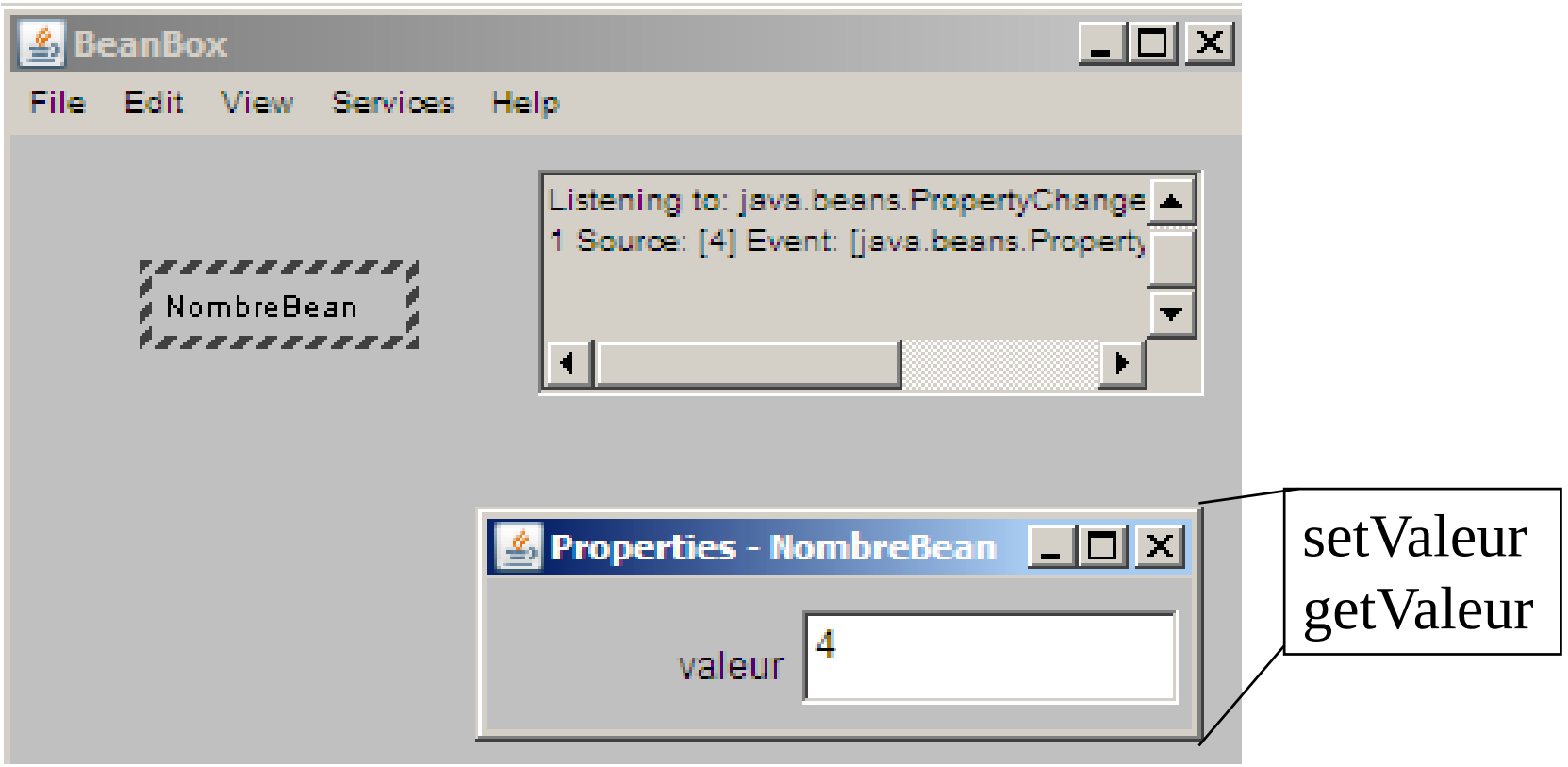
```
public class NombreBean implements Serializable{

  public final int VALEUR_MIN;
  public final int VALEUR_MAX;
  private int valeur;
  private PropertyChangeSupport propertySupport;

  public NombreBean(){
   this.VALEUR_MIN = this.valeur = 0;
   this.VALEUR_MAX = 10;
   this.propertySupport = new PropertyChangeSupport(this);
  }

  public void inc(){
    if(valeur < VALEUR_MAX){
      int old = valeur;
      this.valeur++;
      propertySupport.firePropertyChange("valeur",old,valeur);
    }
  }

  public void addPropertyChangeListener(PropertyChangeListener l) {
    propertySupport.addPropertyChangeListener(l);
  }
     // public int getValeur()  et setValeur()
```

La Vue est à l'écoute de son Bean

```
public class Vue1Bean extends JFrame implements PropertyChangeListener{

  private JButton     plus;
  private JButton     moins;
  private JTextField  valeur;
  private JSlider     jauge;

  public Vue1Bean(NombreBean nombre) {
    super("Vue1 Nombre");
    nombre.addPropertyChangeListener(this);

   // Le dessin ici
  }

  @Override
  public void propertyChange(PropertyChangeEvent evt){
    assert evt.getPropertyName().equals("valeur");
    this.valeur.setText(evt.getNewValue().toString());
    this.jauge.setValue((Integer)evt.getNewValue());
  }
```

Les contrôles ne changent pas, le MVC reste en l'état

NombreBean intègre la BeanBox



- **Ajout par l'outil BeanBox d'un listener (EventMonitor) à chaque changement de valeur de *valeur* …**

Usage de l'introspection

Choix d'architecture

- **Un TP :**

Application : un TP

Développez une application de type calculette à pile

L'évaluation d'une expression arithmétique est réalisée à l'aide d'une pile.

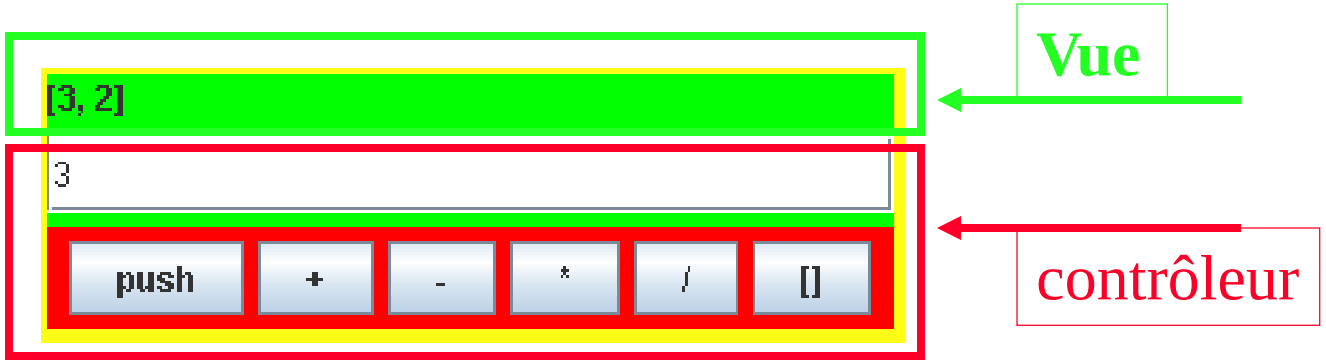
Par exemple l'expression 3 + 2 engendre la séquence :

empiler(3); empiler(2); empiler(depiler() + depiler()); *// simple non ?*

Une Interface graphique :

Quel découpage ? Modèle, Vue et Contrôleur ?

Proposition avant discussion



- **MVC proposé :**
  - **Le** Contrôleur est un JPanel,
    - Transforme les actions sur les boutons ou l'entrée d'une opérande en opérations sur le Modèle
  - ou bien Le Modèle est une calculette qui utilise une pile
    - Est un « Observable »
  - **La** Vue est un JPanel,
    - Observateur du Modèle, la vue affiche l'état du Modèle à chaque notification

Proposition après discussion

Vue

- **MVC proposé :**
  - **Le** Contrôleur est une classe qui implémente tous les listeners
    - Transforme les actions sur les boutons ou l'entrée d'une opérande en opérations sur le Modèle
  - ou bien Le Modèle est une calculette qui utilise une pile
    - Est un « Observable »
  - **La** Vue est un JPanel, une JApplet, Une Frame … ce que l'on voit
    - Observateur du Modèle, la vue affiche l'état du Modèle à chaque notification