Approche modèle vue contrôleur

Fabrice Rossi

Télécom ParisTech

Patron de conception

- *Design pattern*
- schéma général pour résoudre un problème classique :
  - s'appuie sur l'approche orientée objet : ensemble de classes et d'objets avec leurs rôles et relations
  - capitalisation d'expérience(s)
  - savoir faire
  - relativement informel
- principe issu de l'architecture (Christopher Alexander)
- à la mode depuis le livre « *Design Patterns – Elements of Reusable Object-Oriented Software* » du « Gang of Four » (E. Gamma, R. Helm, R. Johnson et J. Vlissides)

Schéma d'un patron

- problème à résoudre (objectif)
- motivations
- contextes d'application
- solution(s)

Exemple

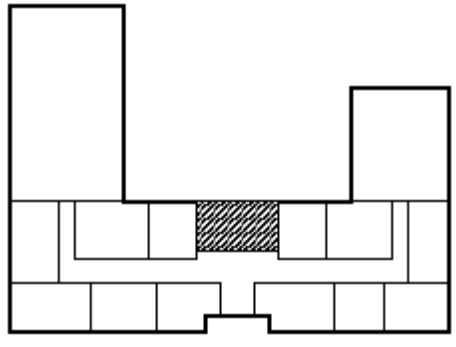
Cœur d'un bâtiment (Université de l'Oregon)

- objectif : favoriser les rencontres et les discussions
- motivation : favoriser l'échange d'idées, la formation d'un esprit de groupe, etc.
- contexte : lieu de travail

Exemple

Cœur d'un bâtiment (Université de l'Oregon)

- objectif : favoriser les rencontres et les discussions
- motivation : favoriser l'échange d'idées, la formation d'un esprit de groupe, etc.
- contexte : lieu de travail
- solution :
  - créer un cœur social au centre de gravité du bâtiment
  - s'assurer que le cœur est desservi par un chemin que tout le monde emprunte
  - équiper le cœur avec des canapés, une machine à café, etc.



Exemple

Noter un objet (Yahoo)

- objectif : permettre à un utilisateur de noter quelque chose
- motivation : retour sur un service, recommandation, etc.
- contexte : site web

Exemple

Noter un objet (Yahoo)

- objectif : permettre à un utilisateur de noter quelque chose
- motivation : retour sur un service, recommandation, etc.
- contexte : site web
- solution :
  - afficher une échelle d'objets cliquables
  - pas de note initiale
  - modification dynamique (*rollover*)
  - traduction textuelle
  - figé au clic

Interface humain-machine

- représentation visuelle interactive des données manipulées

Interface humain-machine

- représentation visuelle interactive des données manipulées
- données manipulées :
  - texte (éditeur de texte, client mail ou messagerie)
  - musique (gestion d'une collection musicale)
  - monde virtuel (jeu vidéo)

Interface humain-machine

- représentation visuelle interactive des données manipulées
- données manipulées :
  - texte (éditeur de texte, client mail ou messagerie)
  - musique (gestion d'une collection musicale)
  - monde virtuel (jeu vidéo)
- représentation visuelle :
  - texte brut ou aperçu avant impression
  - méta données (titre, compositeur)
  - vue à la première personne

Interface humain-machine

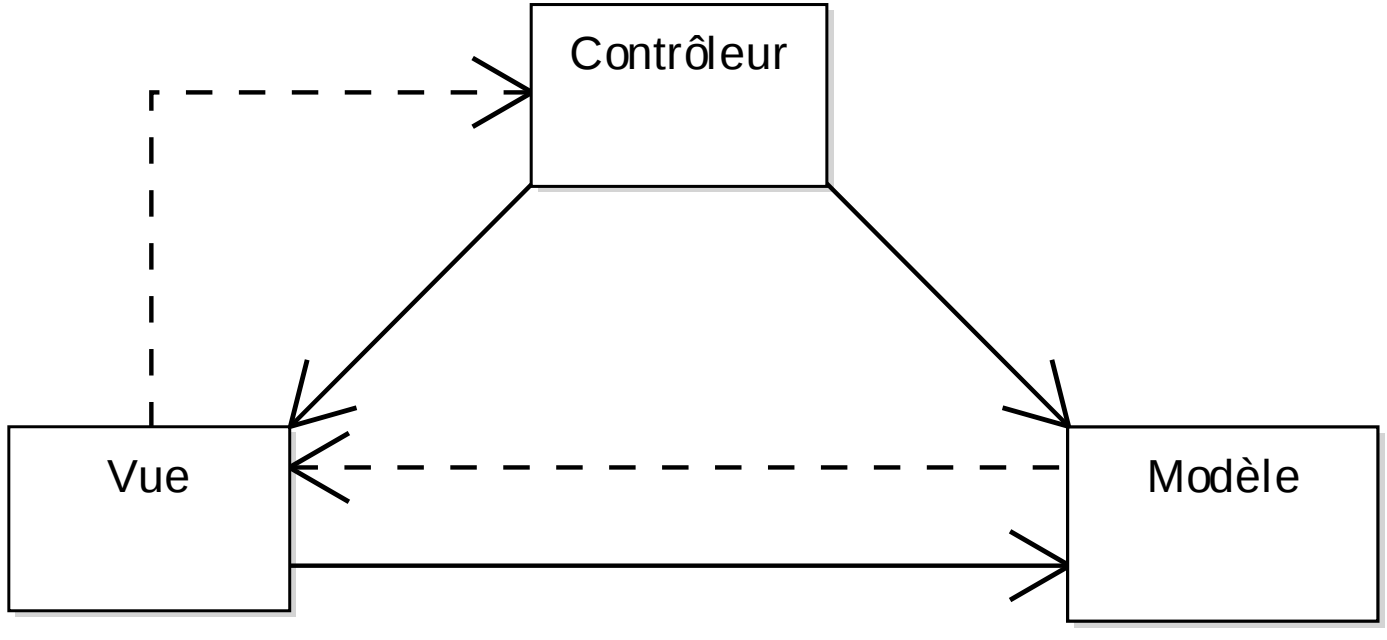
- représentation visuelle interactive des données manipulées
- données manipulées :
  - texte (éditeur de texte, client mail ou messagerie)
  - musique (gestion d'une collection musicale)
  - monde virtuel (jeu vidéo)
- représentation visuelle :
  - texte brut ou aperçu avant impression
  - méta données (titre, compositeur)
  - vue à la première personne
- interaction :
  - clavier
  - souris
  - contrôleur ad hoc (paddle, joystick, roue cliquable, etc.)

Modèle – Vue – Contrôleur

- patron inventé par Trygve Reenskaug en 1979
- objectif : séparer les trois aspects d'une IHM (données, représentation et interaction)
- motivations :
  - développement en équipe
  - représentations graphiques multiples pour un même objet
  - interfaces multiples
- solution : classes et objets séparés pour les trois aspects

Modèle – Vue – Contrôleur

- organisation :
  - données : modèle
  - représentation visuelle : vue
  - interaction : contrôleur



Modèle – Vue – Contrôleur

- Modèle :
  - représentation des données manipulées par le programme
  - texte, fichier mp3, terrain de jeu et joueurs, etc.
- Vue :
  - représentation(s) visuelle des données
  - vue 3D, tableau, texte, etc.
- Contrôleur :
  - gestion de l'interaction
  - effets de la souris, du clavier, etc.
- Dans tous les cas, une ou plusieurs classes et les objets associés

Exemple simple

- affichage graphique et interactif d'une note (patron Note)
- modèle :
  - représentation informatique d'une note
  - un objet d'une classe adaptée
- vues :
  - représentation(s) graphique(s) de la note
  - une classe par représentation
- contrôleur(s) :
  - contrôle du modèle (changer la valeur de l'entier) et/ou de la vue (changer la représentation de l'entier)
  - une classe par grande catégorie de contrôles

Modèle d'une note

Note

```
1  package fr.enst.cours.mvc2010;
2
3  public class Note {
4      private int value;
5      private boolean initialized;
6
7      public int getValue() {
8          return value;
9      }
10     public void setValue(int value) {
11         if (value < 0 || value > 5) {
12             throw new RuntimeException("Out of bounds: " + value);
13         }
14         this.value = value;
15         initialized = true;
16     }
17     public boolean isInitialized() {
18         return initialized;
19     }
20 }
```

Vue texte

VueNoteTexte

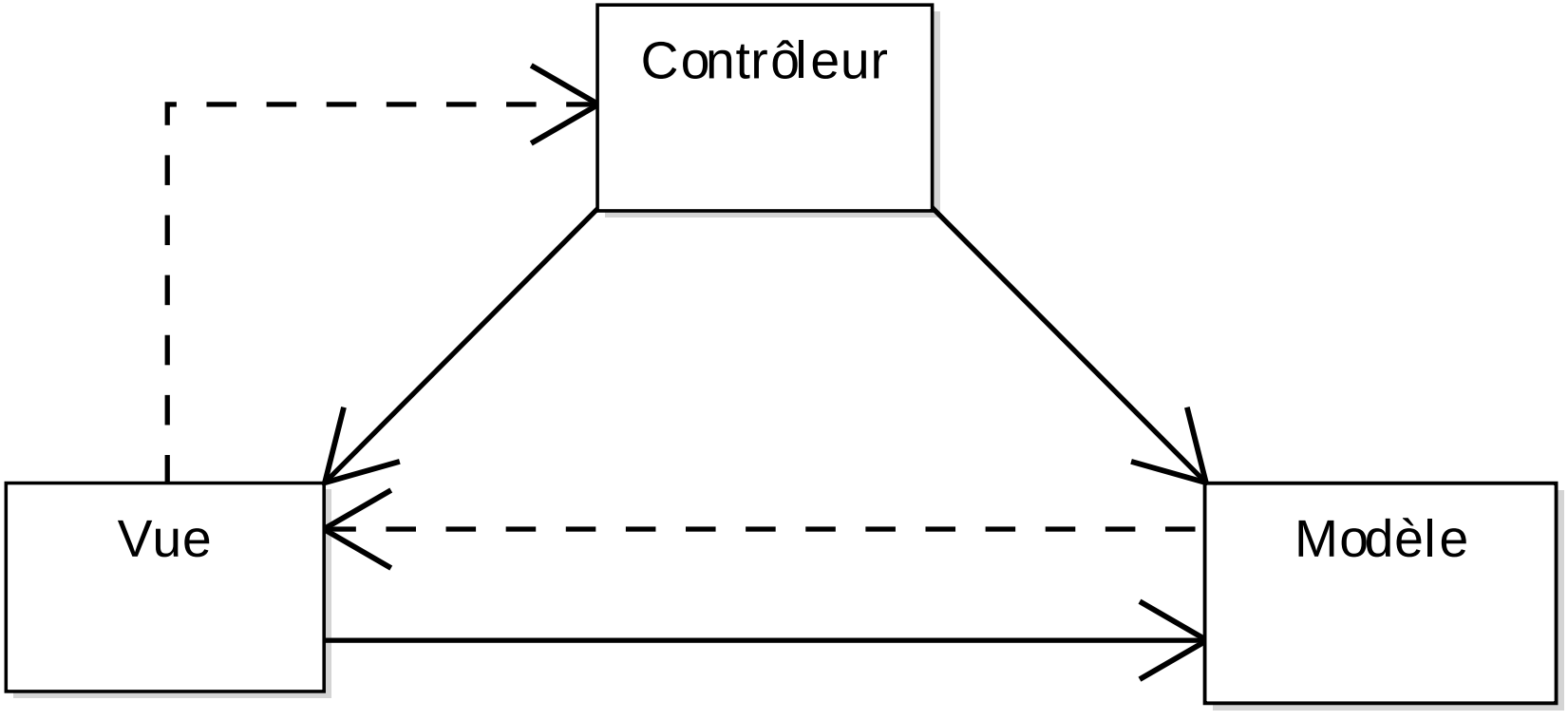
```
1  package fr.enst.cours.mvc2010;
2
3  import java.awt.Color;
4
5  import javax.swing.JLabel;
6
7  public class VueNoteTexte extends JLabel {
8      private Note note;
9
10     public VueNoteTexte(Note note) {
11         this.note = note;
12         setHorizontalTextPosition(CENTER);
13         setOpaque(true);
14         update();
15     }
16     public void update() {
17         if(note.isInitialized()) {
18             setForeground(Color.BLACK);
19             setText(String.valueOf(note.getValue()));
20         } else {
21             setForeground(Color.LIGHT_GRAY);
22             setText("aucune");
23         }
24     }
25 }
```

Programme principal

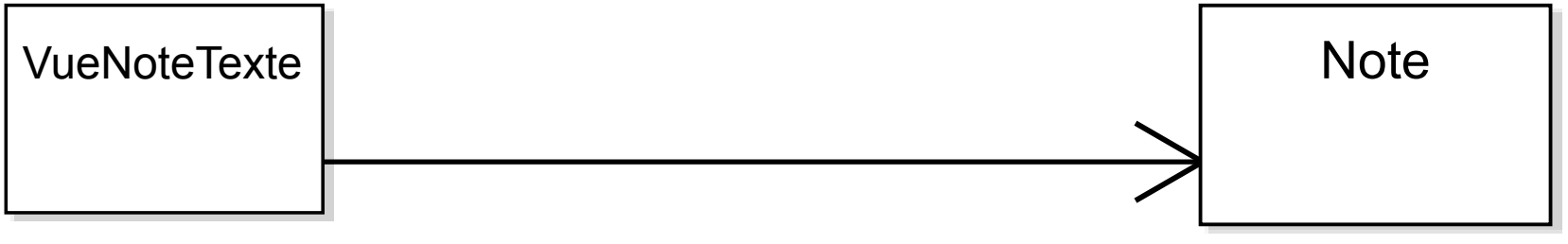
DemoV1

```
1  package fr.enst.cours.mvc2010;
2
3  import java.awt.FlowLayout;
4
5  import javax.swing.JFrame;
6  import javax.swing.JLabel;
7
8  public class DemoV1 {
9      public static void main(String[] args) {
10         Note note = new Note(); // le modèle
11         VueNoteTexte vue = new VueNoteTexte(note); // la vue
12         // la fenêtre principale
13         JFrame cadre = new JFrame("Une note");
14         cadre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
15         cadre.getContentPane().setLayout(new FlowLayout());
16         cadre.getContentPane().add(new JLabel("Note :"));
17         cadre.getContentPane().add(vue);
18         cadre.pack();
19         cadre.setSize(200, 50);
20         // affichage
21         cadre.setVisible(true);
22     }
23 }
```

Modèle – Vue – Contrôleur



Modèle – Vue – Contrôleur



Leçons

- Modèle :
  - une ou plusieurs classes relativement arbitraires pour l'instant
  - accès au « contenu » par des méthodes `set/get`
- Vue :
  - construite à partir des composants *Swing* de Java
  - liée au modèle (contient une référence vers le modèle)
  - un ou plusieurs objets :
    - contenant des instances de composants *Swing*
    - ou/et dont la classe hérite de celle d'un composant *Swing*
- le patron est souple : nombreuses variantes possibles

Interaction

- deux types d'effets :
  - effets sur une vue
  - effets sur le modèle
- pour l'exemple :
  - changer la valeur de l'entier
  - changer les caractéristiques de la représentation graphique
- en Java :
  - approche par évènement : les actions de l'utilisateur engendrent des évènements
  - les évènements sont transmis à des « écouteurs » (*listener*) : des objets particuliers qui ont pour mission de réagir aux évènements
  - le contrôleur est représenté par un ou plusieurs écouteurs

Interaction

- activation de la vue :
  - le passage de la souris « active » la vue
  - retour à l'état par défaut quand la souris quitte la vue
- évènement en Java :
  - évènement de type `MouseEvent`
  - géré par un `MouseListener`
  - méthodes `mouseEntered` et `mouseExited`
- un écouteur en Java :
  - doit implémenter une interface de `XxxListener`
  - par exemple en héritant de la classe `XxxAdapter` (méthodes vides)
  - doit s'enregistrer auprès d'un composant *Swing* pour recevoir les évènements associés à ce composant

Exemple

ActivationVue

```
1  package fr.enst.cours.mvc2010;
2
3  import java.awt.Color;
4  import java.awt.event.MouseAdapter;
5  import java.awt.event.MouseEvent;
6
7  public class ActivationVue extends MouseAdapter {
8      private VueNoteTexte vue;
9      private Color bg;
10
11     public ActivationVue(VueNoteTexte vue) {
12         this.vue = vue;
13         bg = vue.getBackground();
14         vue.addMouseListener(this);
15     }
16     @Override
17     public void mouseEntered(MouseEvent e) {
18         vue.setBackground(Color.YELLOW);
19         vue.repaint();
20     }
21     @Override
22     public void mouseExited(MouseEvent e) {
23         vue.setBackground(bg);
24     }
25 }
```

Programme principal

DemoV2

```
1  package fr.enst.cours.mvc2010;
2
3  import java.awt.FlowLayout;
4
5  import javax.swing.JFrame;
6  import javax.swing.JLabel;
7
8  public class DemoV2 {
9      public static void main(String[] args) {
10         Note note = new Note(); // le modèle
11         VueNoteTexte vue = new VueNoteTexte(note); // la vue
12         ActivationVue controleur = new ActivationVue(vue);
13         // la fenêtre principale
14         JFrame cadre = new JFrame("Une note");
15         cadre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
16         cadre.getContentPane().setLayout(new FlowLayout());
17         cadre.getContentPane().add(new JLabel("Note :"));
18         cadre.getContentPane().add(vue);
19         cadre.pack();
20         cadre.setSize(200, 50);
21         // affichage
22         cadre.setVisible(true);
23     }
24 }
```

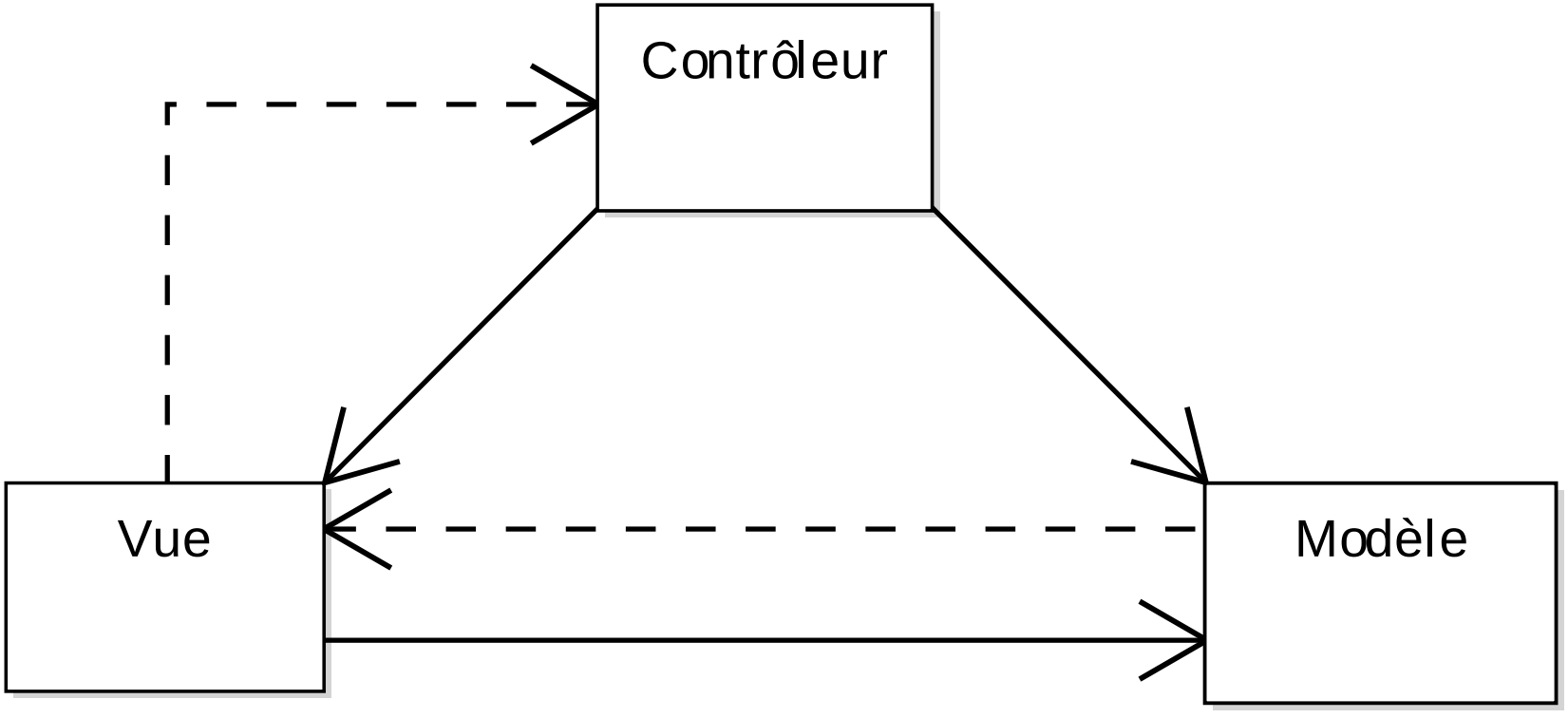
Mécanisme

- algorithme de transmission des évènements :
  1. action de l'utilisateur capturée par le système d'exploitation
  2. transmission de l'action à la fenêtre *focalisée*
  3. transmission par Java au composant *focalisé*
  4. appel des méthodes concernées de tous les écouteurs enregistrés auprès du composant
- mise à jour de la visualisation
  - automatique pour certaines actions : restauration de la fenêtre, changement de taille, etc.
  - à lancer par un appel à une méthode `repaint` dans la plupart des cas
  - ici automatique car on modifie les propriétés d'un composant standard (un `JLabel`)

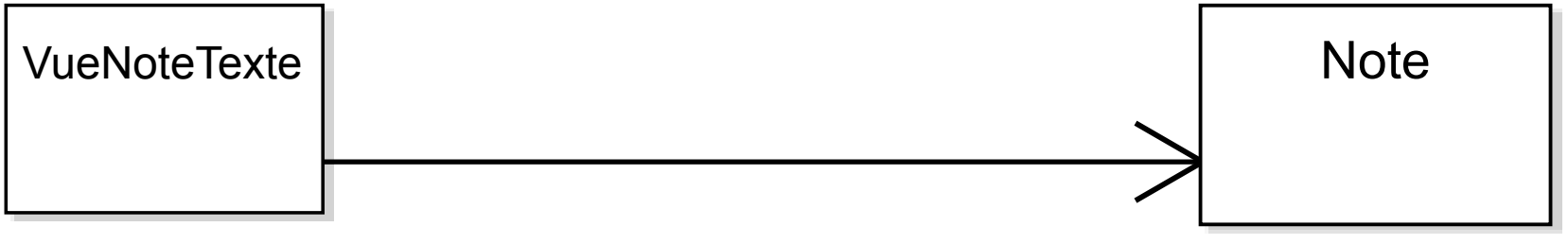
Récapitulation

- Modèle :
  - une instance de la classe `Note`
  - rien de spécifique à une interface graphique pour l'instant
- Vue :
  - une instance de la classe `VueNoteTexte` qui hérite de `JLabel` (d'un composant *Swing*)
  - doit connaître le modèle : référence
  - doit se mettre à jour si elle est modifiée
- Contrôleur :
  - une instance de la classe `ActivationVue` qui implémente `MouseListener`
  - doit connaître l'objet contrôlé : ici référence vers la vue
  - doit s'enregistrer auprès d'un ou plusieurs composants graphiques

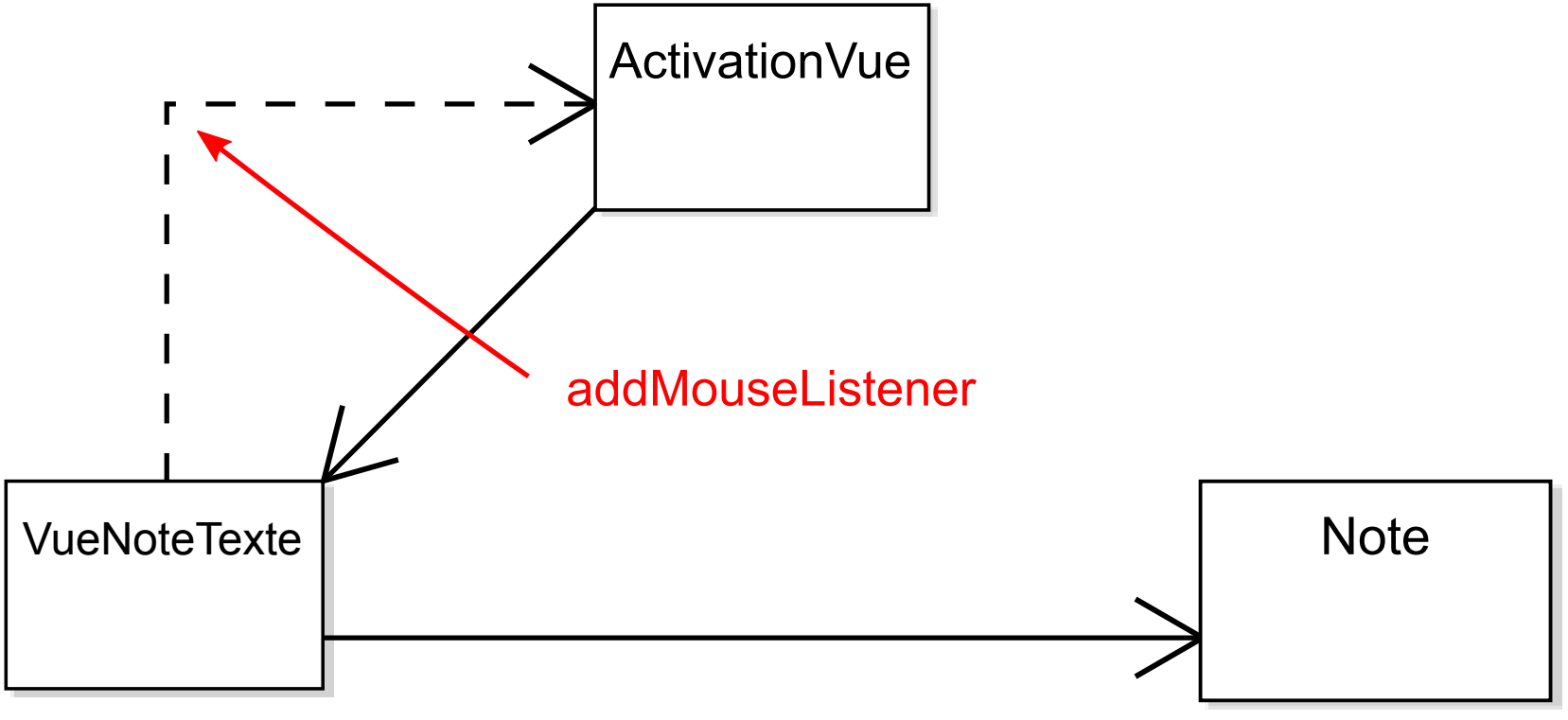
Modèle – Vue – Contrôleur



Modèle – Vue – Contrôleur



Modèle – Vue – Contrôleur



Contrôle du modèle

- exemple : changer la valeur de la note au clavier (touches + et -)
- évènement en Java :
  - évènement de type `KeyEvent`
  - géré par un `KeyListener`
  - méthode `keyTyped`
- on hérite ici de `KeyAdapter` pour ne pas s'occuper des autres méthodes de `KeyListener`
- attention au *focus* en Java :
  - un composant doit avoir le *focus* pour recevoir des évènements clavier
  - solution classique : un `MouseListener` qui demande l'obtention du *focus*

Contrôleur

ChangeNoteV1

```
1  package fr.enst.cours.mvc2010;
2
3  import java.awt.event.KeyAdapter;
4  import java.awt.event.KeyEvent;
5
6  public class ChangeNoteV1 extends KeyAdapter {
7      private Note note;
8      public ChangeNoteV1(Note note) {
9          this.note = note;
10     }
11     @Override
12     public void keyTyped(KeyEvent e) {
13         int val;
14         System.out.println(e.getKeyChar());
15         switch(e.getKeyChar()) {
16         case '+':
17             val = note.getValue();
18             if(val<5) {
19                 note.setValue(val+1);
20             }
21             break;
22         case '-':
23             val = note.getValue();
24             if(val>0) {
25                 note.setValue(val-1);
26             }
27             break;
28         }
29     }
30 }
```

RequestFocus

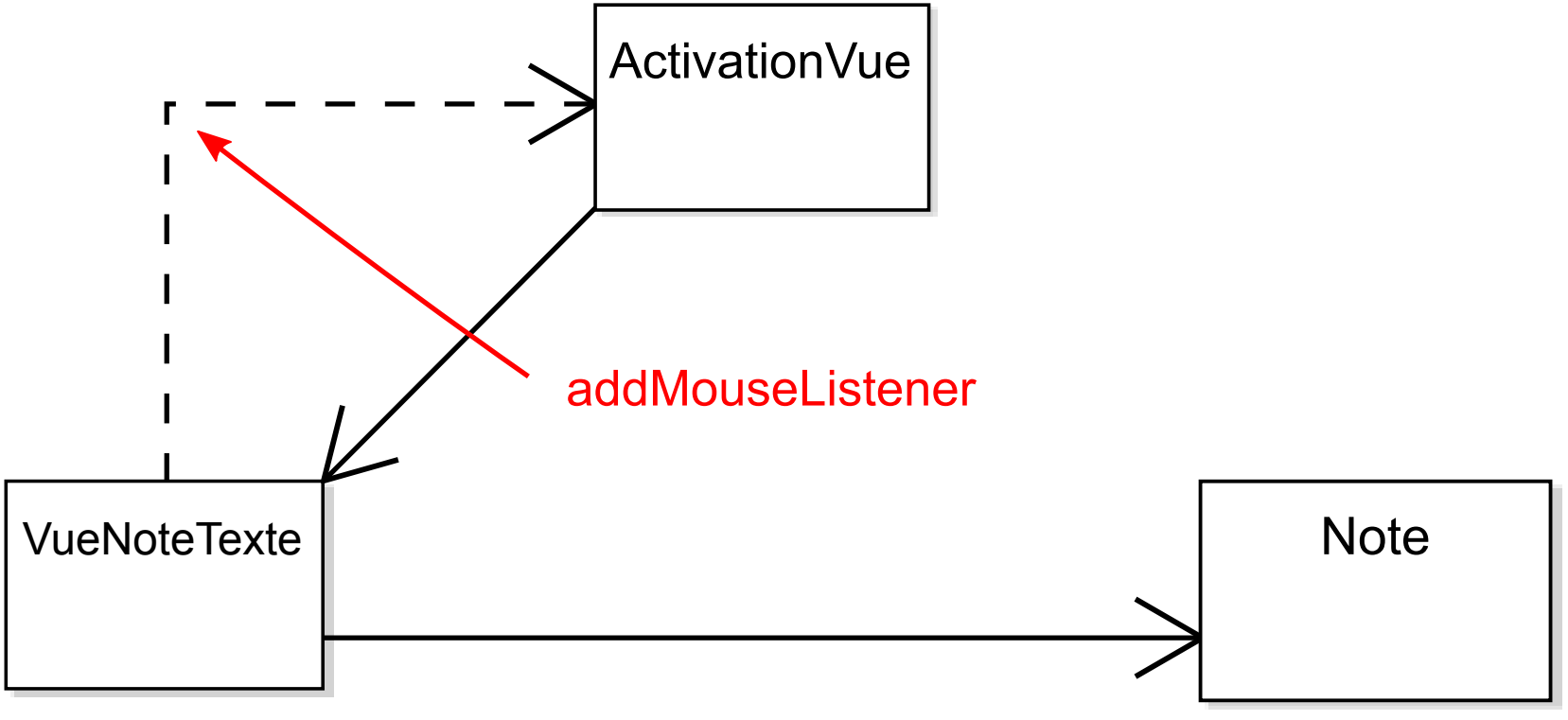
```
1  package fr.enst.cours.mvc2010;
2
3  import java.awt.event.MouseAdapter;
4  import java.awt.event.MouseEvent;
5
6  import javax.swing.JComponent;
7
8  public class RequestFocus extends MouseAdapter {
9      private JComponent comp;
10
11     public RequestFocus(JComponent comp) {
12         this.comp = comp;
13         comp.addMouseListener(this);
14     }
15
16     @Override
17     public void mouseEntered(MouseEvent e) {
18         comp.requestFocusInWindow();
19     }
20 }
```

Programme principal

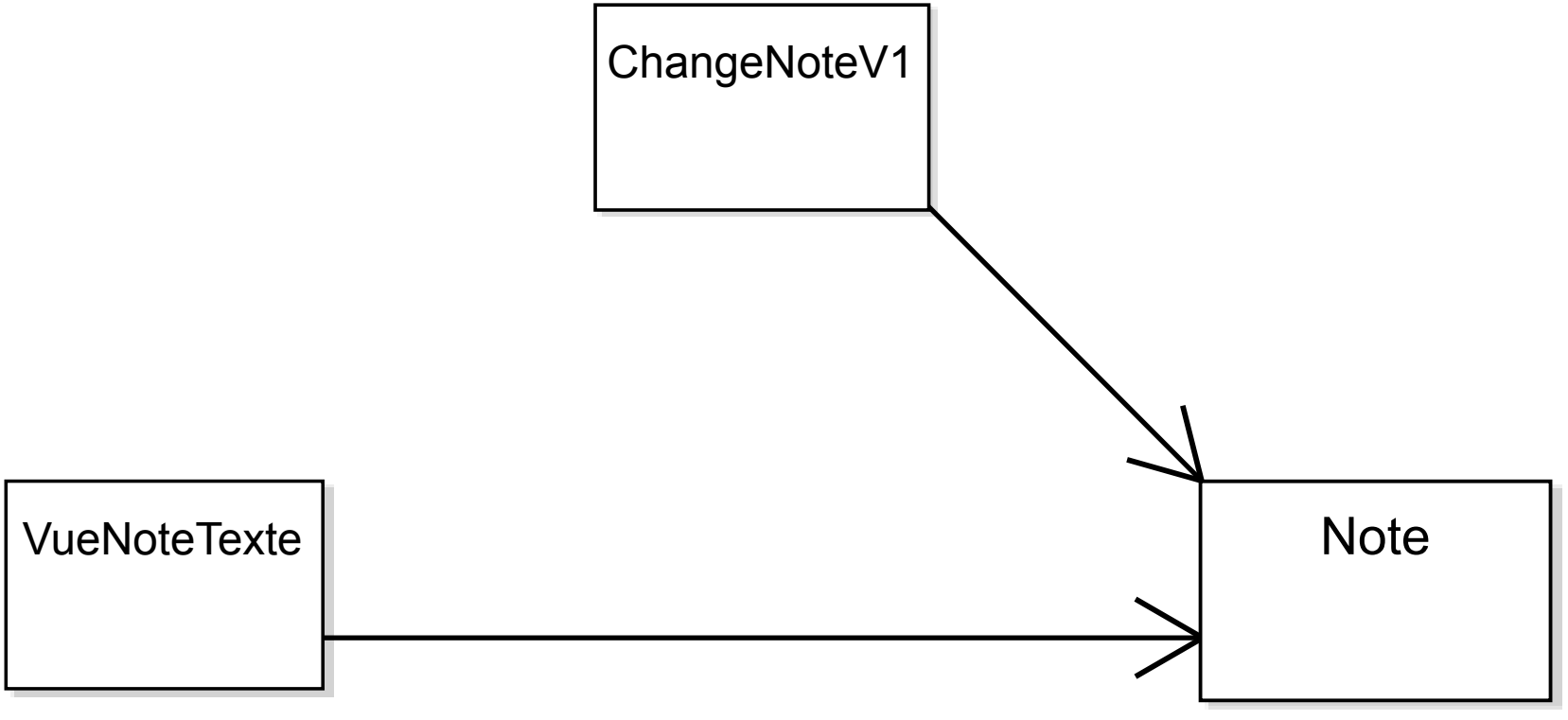
DemoV3

```
1  package fr.enst.cours.mvc2010;
2
3  import java.awt.FlowLayout;
4
5  import javax.swing.JFrame;
6  import javax.swing.JLabel;
7
8  public class DemoV3 {
9      public static void main(String[] args) {
10         Note note = new Note(); // le modèle
11         VueNoteTexte vue = new VueNoteTexte(note); // la vue
12         ActivationVue controleur = new ActivationVue(vue);
13         ChangeNoteV1 clavier = new ChangeNoteV1(note);
14         vue.addKeyListener(clavier);
15         RequestFocus rf = new RequestFocus(vue);
16         // la fenêtre principale
17         JFrame cadre = new JFrame("Une note");
18         cadre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
19         cadre.getContentPane().setLayout(new FlowLayout());
20         cadre.getContentPane().add(new JLabel("Note :"));
21         cadre.getContentPane().add(vue);
22         cadre.pack();
23         cadre.setSize(200, 50);
24         // affichage
25         cadre.setVisible(true);
26     }
27 }
```

Modèle – Vue – Contrôleur



Modèle – Vue – Contrôleur



Analyse du problème

- le programme précédent ne fonctionne pas : l'affichage ne change pas
- mais on peut vérifier que la note change (par ajour d'un affichage dans `setValue` ou avec un débugueur)
- source du problème : la vue n'est pas mise à jour !

Analyse du problème

- le programme précédent ne fonctionne pas : l'affichage ne change pas
- mais on peut vérifier que la note change (par ajour d'un affichage dans `setValue` ou avec un débugueur)
- source du problème : la vue n'est pas mise à jour !
- solution : indiquer à la vue que le modèle est modifié
- approche basique :
  - à éviter !
  - couplage fort : le contrôleur modifie le modèle puis prévient la vue

Nouveau contrôleur

ChangeNoteV2

```
1  package fr.enst.cours.mvc2010;
2
3  import java.awt.event.KeyAdapter;
4  import java.awt.event.KeyEvent;
5
6  public class ChangeNoteV2 extends KeyAdapter {
7      private Note note;
8      private VueNoteTexte vue;
9      public ChangeNoteV2(Note note,VueNoteTexte vue) {
10         this.note = note;
11         this.vue = vue;
12         vue.addKeyListener(this);
13     }
14     @Override
15     public void keyTyped(KeyEvent e) {
16         int val;
17         System.out.println(e.getKeyChar());
18         switch(e.getKeyChar()) {
19         case '+':
20             val = note.getValue();
21             if(val<5) {
22                 note.setValue(val+1);
23                 vue.update();
24             }
25             break;
```

Nouveau contrôleur

ChangeNoteV2

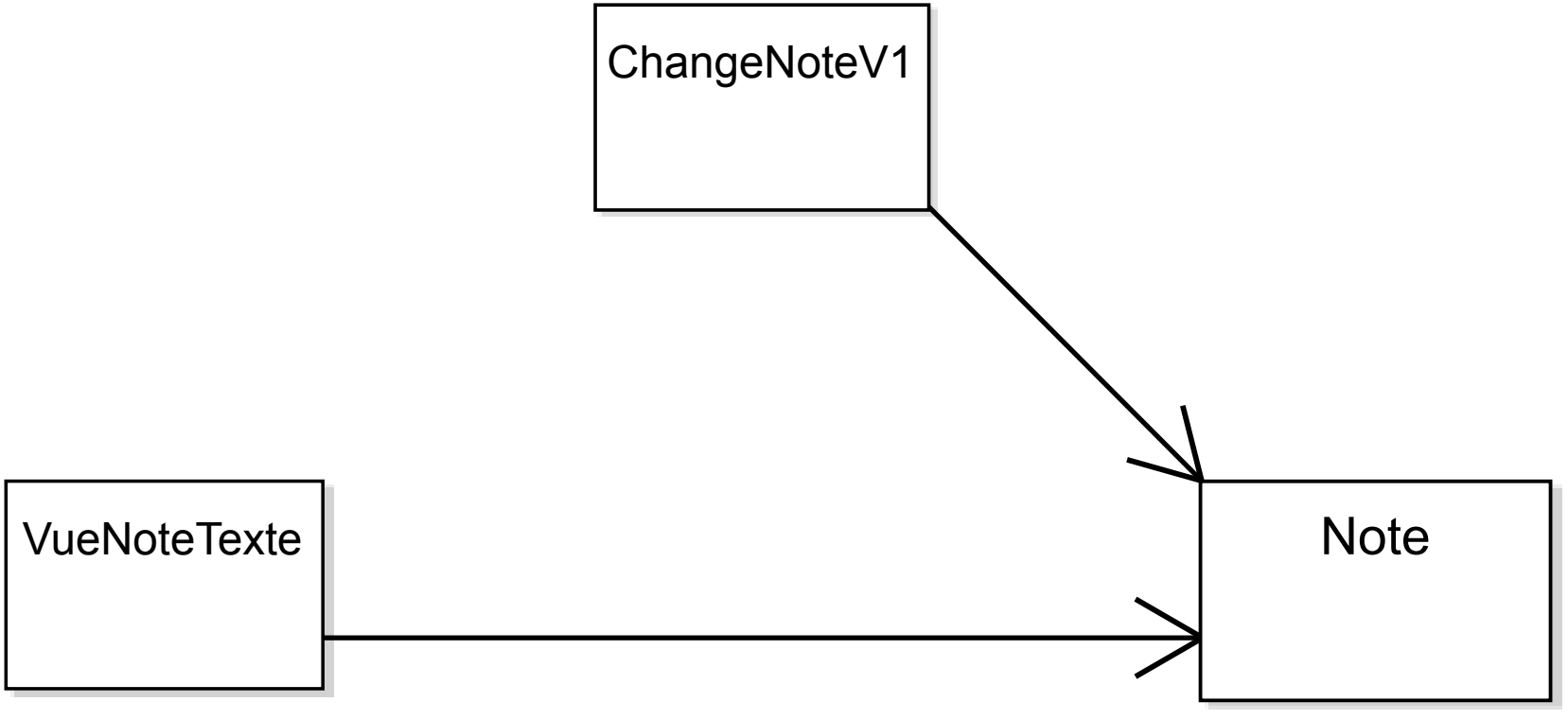
```
26          case '-':
27              val = note.getValue();
28              if(val>0) {
29                  note.setValue(val-1);
30                  vue.update();
31              }
32              break;
33          }
34      }
35  }
```

Programme principal

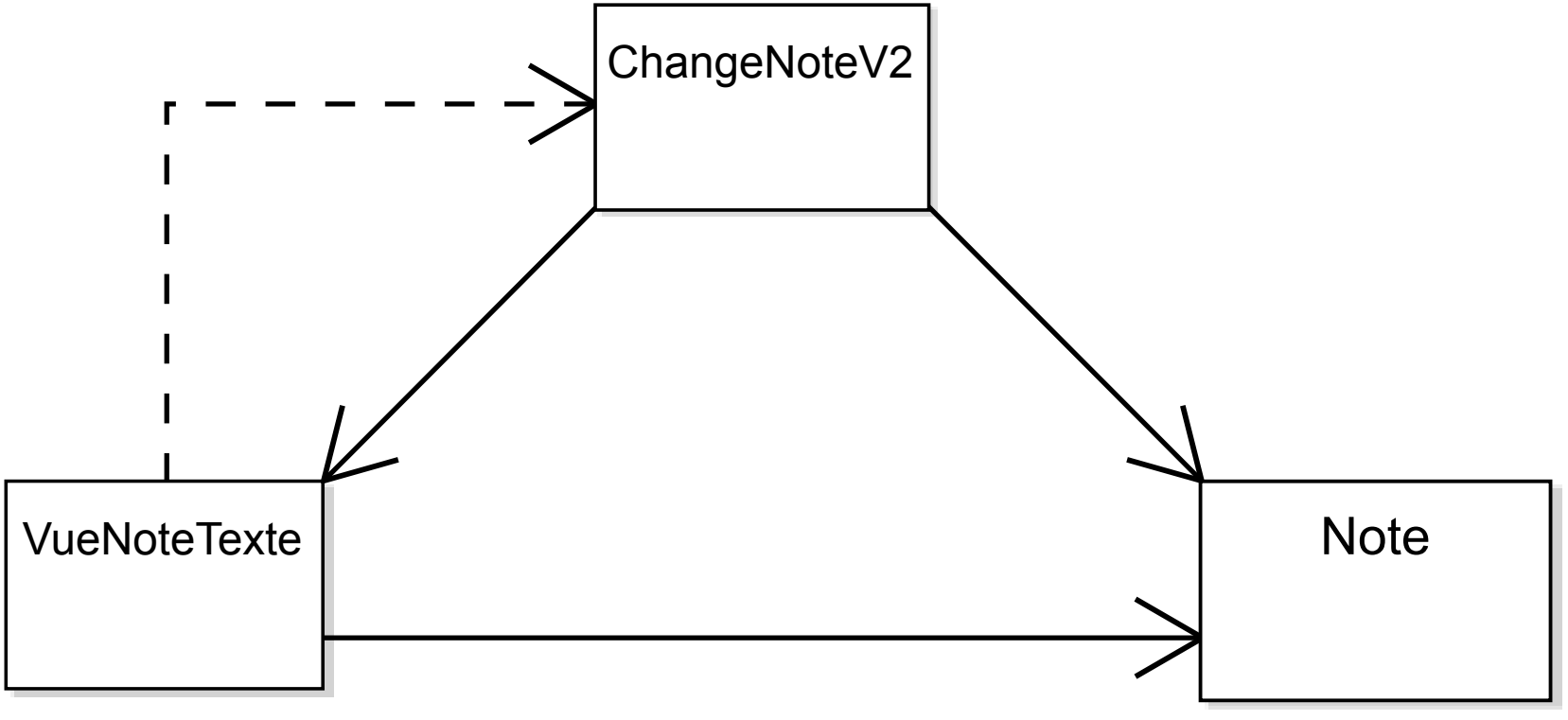
DemoV4

```
1  package fr.enst.cours.mvc2010;
2
3  import java.awt.FlowLayout;
4
5  import javax.swing.JFrame;
6  import javax.swing.JLabel;
7
8  public class DemoV4 {
9      public static void main(String[] args) {
10         Note note = new Note(); // le modèle
11         VueNoteTexte vue = new VueNoteTexte(note); // la vue
12         ChangeNoteV2 clavier = new ChangeNoteV2(note,vue);
13         RequestFocus rf = new RequestFocus(vue);
14         // la fenêtre principale
15         JFrame cadre = new JFrame("Une note");
16         cadre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
17         cadre.getContentPane().setLayout(new FlowLayout());
18         cadre.getContentPane().add(new JLabel("Note :"));
19         cadre.getContentPane().add(vue);
20         cadre.pack();
21         cadre.setSize(200, 50);
22         // affichage
23         cadre.setVisible(true);
24     }
25 }
```

Modèle – Vue – Contrôleur



Modèle – Vue – Contrôleur



Critiques

- couplage fort : le contrôleur du modèle doit connaître l'existence (et le fonctionnement) de la vue
- duplication de code : appel de la méthode de mise à jour de la vue dans toutes les méthodes de gestion d'évènements
- généralisation à plusieurs vues lourde : il faut que le contrôleur connaisse toutes les vues
- origine du problème :
  - le couplage fort
  - ce n'est pas au contrôleur de prendre en charge les conséquences de la modification du modèle
  - solution : déléguer cette tâche au modèle

Patron observateur/observable

- Cadre d'application :
  - quand des objets ont besoin d'être prévenus du changement d'état d'un autre objet
  - relation un vers plusieurs : un sujet et des observateurs
  - notion de publication/souscription : les observateurs s'abonnent au sujet qui publie des informations
- Organisation :
  - un objet *observable*, le *sujet* :
    - propose une méthode d'abonnement pour les observateurs
    - s'engage à appeler une méthode spécifique des observateurs quand il veut publier une information
  - des objets *observateurs* :
    - s'abonnent auprès du sujet
    - possèdent une méthode qui sera appelée en cas de publication d'information par le sujet

Application à MVC

- application immédiate :
  - le modèle est le *sujet observable*
  - les vues sont les *observateurs*
  - découplage :
    - le contrôleur modifie le modèle
    - le modèle prévient les vues des modifications
    - autant de vues qu'on le souhaite
- très utilisé en `Java` dans *Swing* :
  - couple modèle/vue, par ex. `JList` et `ListModel`
  - le modèle est toujours observable et la vue observe le modèle
- aide à la mise en place (en `Java`) :
  - interface `Observer` pour les observateurs
  - classe `Observable` pour le sujet

Outils Java standard

- interface `Observer` (observateur)
- classe `Observable` (sujet)
- mécanisme légèrement plus général :
  - l'unique méthode `Observer` est `update(Observable,Object)`
  - on spécifie donc le sujet concerné et un éventuel paramètre
  - `Observable` inclus une gestion des modifications :
    - on indique par `setChanged` que l'objet est modifié
    - `notifyObservers` publie les modifications
    - découplage possible : plusieurs modifications et une notification régulière

Nouveau modèle

NoteObservable

```
1  package fr.enst.cours.mvc2010;
2
3  import java.util.Observable;
4
5  public class NoteObservable extends Observable {
6      private int value;
7      private boolean initialized;
8
9      public int getValue() {
10         return value;
11     }
12     public boolean setValue(int value) {
13         if (value < 0 || value > 5 || value == this.value) {
14             return false;
15         } else {
16             this.value = value;
17             initialized = true;
18             setChanged();
19             notifyObservers();
20             return true;
21         }
22     }
23     public boolean isInitialized() {
24         return initialized;
25     }
26 }
```

Nouvelle vue

VueNoteTexteObserver

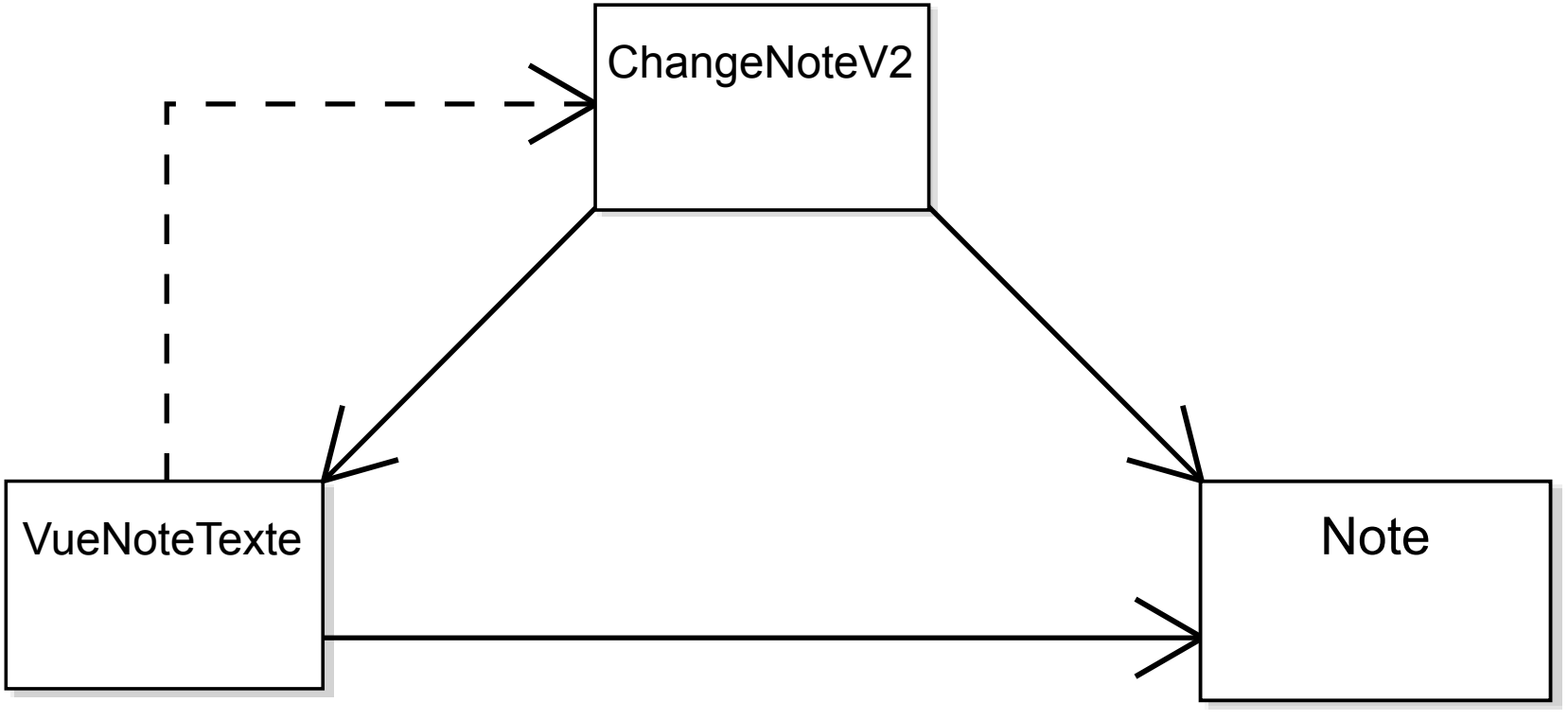
```
1  package fr.enst.cours.mvc2010;
2
3  import java.awt.Color;
4  import java.util.Observable;
5  import java.util.Observer;
6
7  import javax.swing.JLabel;
8
9  public class VueNoteTexteObserver extends JLabel implements Observer {
10     private NoteObservable note;
11
12     public VueNoteTexteObserver(NoteObservable note) {
13         this.note = note;
14         note.addObserver(this);
15         setHorizontalTextPosition(CENTER);
16         setOpaque(true);
17         update();
18     }
19     public void update() {
20         if(note.isInitialized()) {
21             setForeground(Color.BLACK);
22             setText(String.valueOf(note.getValue()));
23         } else {
24             setForeground(Color.LIGHT_GRAY);
25             setText("aucune");
26         }
27     }
28     @Override
29     public void update(Observable source, Object value) {
30         update();
31     }
32 }
```

Programme principal

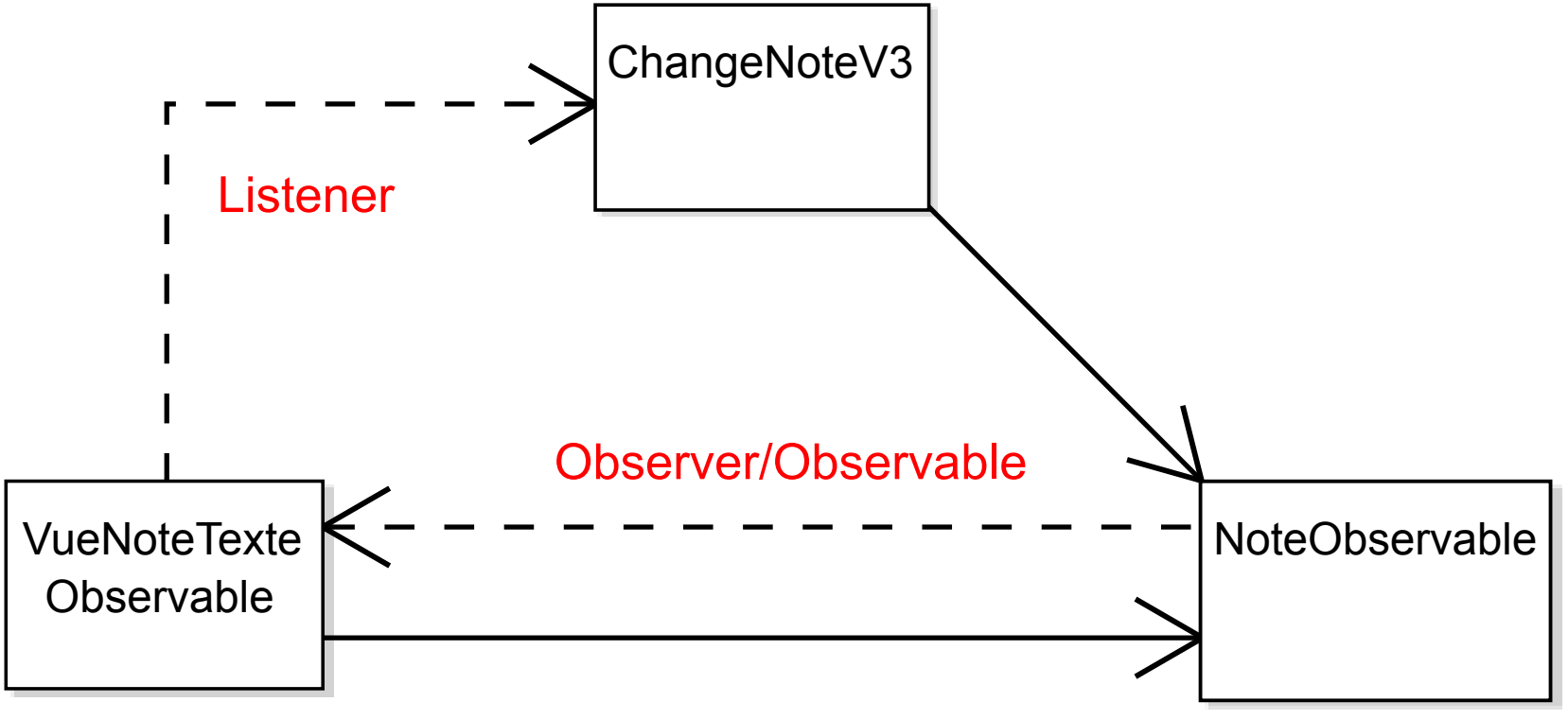
DemoV5

```
1  package fr.enst.cours.mvc2010;
2
3  import java.awt.FlowLayout;
4
5  import javax.swing.JFrame;
6  import javax.swing.JLabel;
7
8  public class DemoV5 {
9      public static void main(String[] args) {
10         NoteObservable note = new NoteObservable(); // le modèle
11         VueNoteTexteObserver vue = new VueNoteTexteObserver(note); // la vue
12         ChangeNoteV3 clavier = new ChangeNoteV3(note);
13         vue.addKeyListener(clavier);
14         RequestFocus rf = new RequestFocus(vue);
15         // la fenêtre principale
16         JFrame cadre = new JFrame("Une note");
17         cadre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
18         cadre.getContentPane().setLayout(new FlowLayout());
19         cadre.getContentPane().add(new JLabel("Note :"));
20         cadre.getContentPane().add(vue);
21         cadre.pack();
22         cadre.setSize(200, 50);
23         // affichage
24         cadre.setVisible(true);
25     }
26 }
```

Modèle – Vue – Contrôleur



Modèle – Vue – Contrôleur



Récapitulation

- modèle :
  - un ou plusieurs objets
  - en général les classes héritent de `Observable`
  - on peut regrouper les éléments du modèle dans une seule classe `Observable`
- vue :
  - un objet instance d'une classe composant graphique ou héritant d'une telle classe
  - doit connaître le modèle : référence
  - doit se mettre à jour si elle est modifiée : implémente l'interface `Observable` et s'enregistre auprès des objets qui constituent le modèle
- contrôleur :
  - un objet instance d'une classe implémentant une interface `XxxListener`
  - doit connaître l'objet contrôlé : référence vers la vue ou vers le modèle
  - doit s'enregistrer auprès d'un ou plusieurs composants graphiques

Exemple final

- mouvement d'une unité graphique
- modèle : une unité
- vue : version graphique de l'unité
- contrôleur : souris

Modèle

Unit

```
1  package fr.enst.cours.mvc2010;
2
3  import java.util.Observable;
4
5  public class Unit extends Observable {
6      private int X;
7      private int Y;
8      private boolean moving;
9      public int getX() {
10         return X;
11     }
12     public int getY() {
13         return Y;
14     }
15     public void moveTo(int X,int Y) {
16         this.X=X;
17         this.Y=Y;
18         setChanged();
19         notifyObservers();
20     }
21     public boolean isMoving() {
22         return moving;
23     }
24     public void setMoving(boolean moving) {
25         this.moving = moving;
26     }
27 }
```

Version graphique

GraphicalUnit

```
1  package fr.enst.cours.mvc2010;
2
3  import java.awt.Graphics;
4  import java.awt.image.BufferedImage;
5
6  public class GraphicalUnit extends Unit {
7      private BufferedImage picture;
8
9      public GraphicalUnit(BufferedImage picture) {
10         this.picture = picture;
11     }
12     public void paint(Graphics g) {
13         g.drawImage(picture, getX()-picture.getWidth()/2,
14                             getY()-picture.getHeight()/2,
15                             null);
16     }
17     public BufferedImage getPicture() {
18         return picture;
19     }
20 }
```

RichUnitView

```
1  package fr.enst.cours.mvc2010;
2
3  import java.awt.Graphics;
4  import java.util.Observable;
5  import java.util.Observer;
6
7  import javax.swing.JPanel;
8
9  public class RichUnitView extends JPanel implements Observer {
10     private GraphicalUnit myUnit;
11
12     public RichUnitView(GraphicalUnit myUnit) {
13         this.myUnit = myUnit;
14         myUnit.addObserver(this);
15     }
16
17     @Override
18     public void update(Observable o, Object arg) {
19         repaint();
20     }
21
22     @Override
23     protected void paintComponent(Graphics g) {
24         super.paintComponent(g);
25         myUnit.paint(g);
26     }
27
28 }
```

Contrôleur

UnitMouseController

```
1  package fr.enst.cours.mvc2010;
2
3  import java.awt.event.MouseAdapter;
4  import java.awt.event.MouseEvent;
5
6  public class UnitMouseController extends MouseAdapter {
7      private Unit myUnit;
8
9      public UnitMouseController(Unit myUnit) {
10         this.myUnit = myUnit;
11     }
12
13     @Override
14     public void mouseClicked(MouseEvent e) {
15         myUnit.moveTo(e.getX(), e.getY());
16     }
17
18 }
```

Programme principal

DemoRichUnit

```
1  package fr.enst.cours.mvc2010;
2
3  import java.awt.image.BufferedImage;
4  import java.io.File;
5  import java.io.IOException;
6
7  import javax.imageio.ImageIO;
8  import javax.swing.JFrame;
9
10 public class DemoRichUnit {
11     public static void main(String[] args) throws IOException {
12         BufferedImage img = ImageIO.read(new File("cavalier-ranged-2.png"));
13         GraphicalUnit unit = new GraphicalUnit(img);
14         unit.moveTo(100, 100);
15         RichUnitView vue = new RichUnitView(unit);
16         UnitMouseController controller = new UnitMouseController(unit);
17         vue.addMouseListener(controller);
18         RequestFocus rf = new RequestFocus(vue);
19         // la fenêtre principale
20         JFrame cadre = new JFrame("Unité");
21         cadre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
22         cadre.getContentPane().add(vue);
23         cadre.pack();
24         cadre.setSize(400, 400);
25         // affichage
26         cadre.setVisible(true);
27     }
28 }
```

Principes

- Modèle :
  - une classe sans aspect graphique et observable
  - une classe graphique riche héritant de la classe de base
  - variante recommandée : utilisation plutôt qu'héritage
- Vue :
  - classe héritant d'un `JPanel` (technique classique pour les graphismes personnalisés)
  - observateur
- Contrôleur :
  - classe `Listener`
  - connaît le modèle (en version graphique)
  - s'enregistre sur la vue principale

Principes

- Modèle :
  - une classe sans aspect graphique et observable
  - une classe graphique riche héritant de la classe de base
  - variante recommandée : utilisation plutôt qu'héritage
- Vue :
  - classe héritant d'un `JPanel` (technique classique pour les graphismes personnalisés)
  - observateur
- Contrôleur :
  - classe `Listener`
  - connaît le modèle (en version graphique)
  - s'enregistre sur la vue principale
- Extensions :
  - plusieurs vues
  - animation

Contrôle automatique

- le contrôleur n'est pas nécessairement l'utilisateur
- animation :
  - modification périodique du modèle (ou de la vue)
  - affichage du nouvel état
- contrôleur d'animation : réalise les modifications de façon automatique
- support dans *Swing* :
  - le `Timer`
  - permet de lancer une action dans le futur ou à intervalle régulier dans le temps
  - le `Timer` envoie des `ActionEvent`