Cours 3 : Architecture MVC et composants

NFA035

S. Rosmorduc

Le MVC

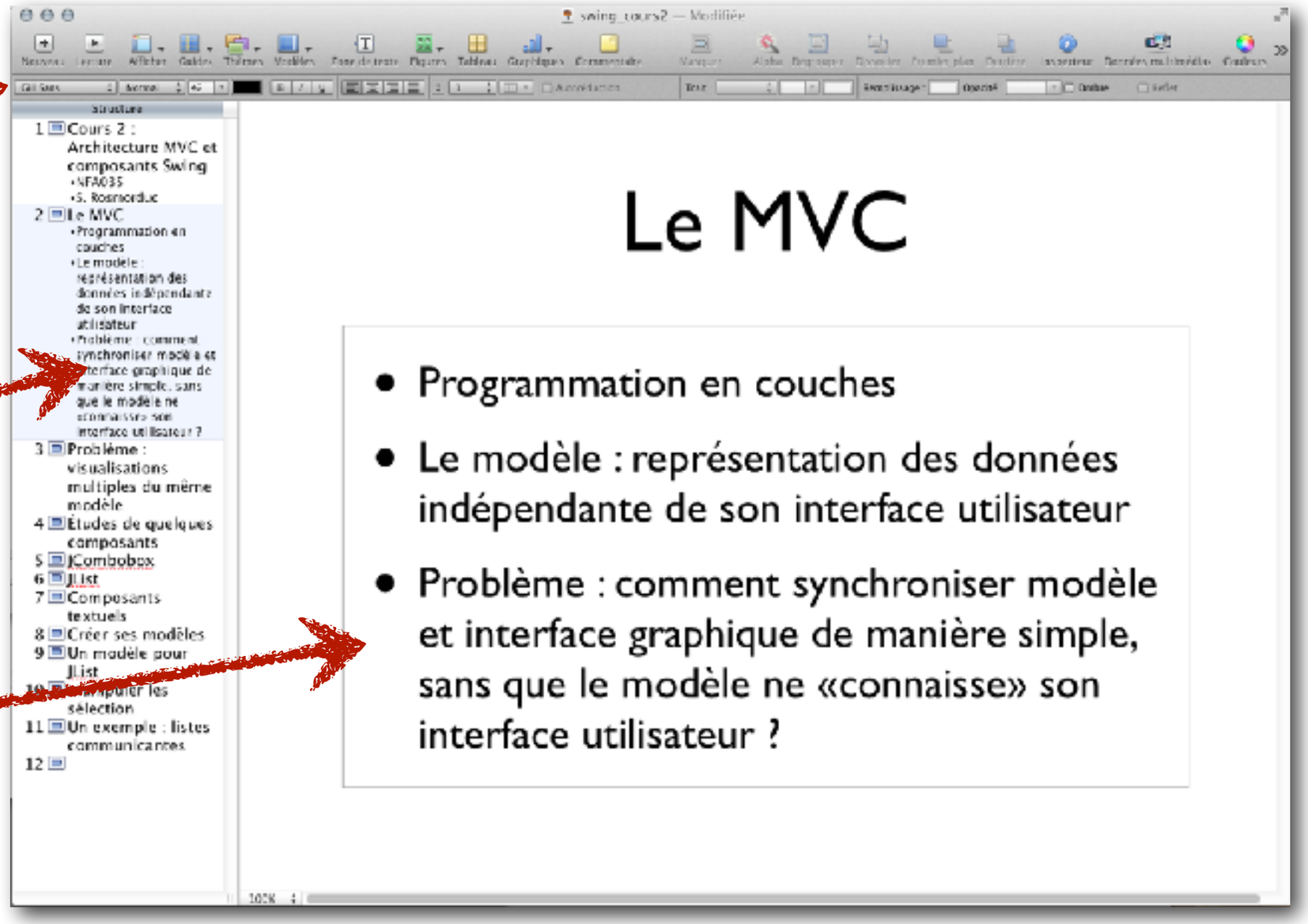
- Programmation en couches
- Le modèle : représentation des données *indépendante* de son interface utilisateur
- Problème : comment synchroniser modèle et interface graphique de manière simple, sans que le modèle ne «connaisse» son interface utilisateur ?

Problème : visualisations multiples du même modèle

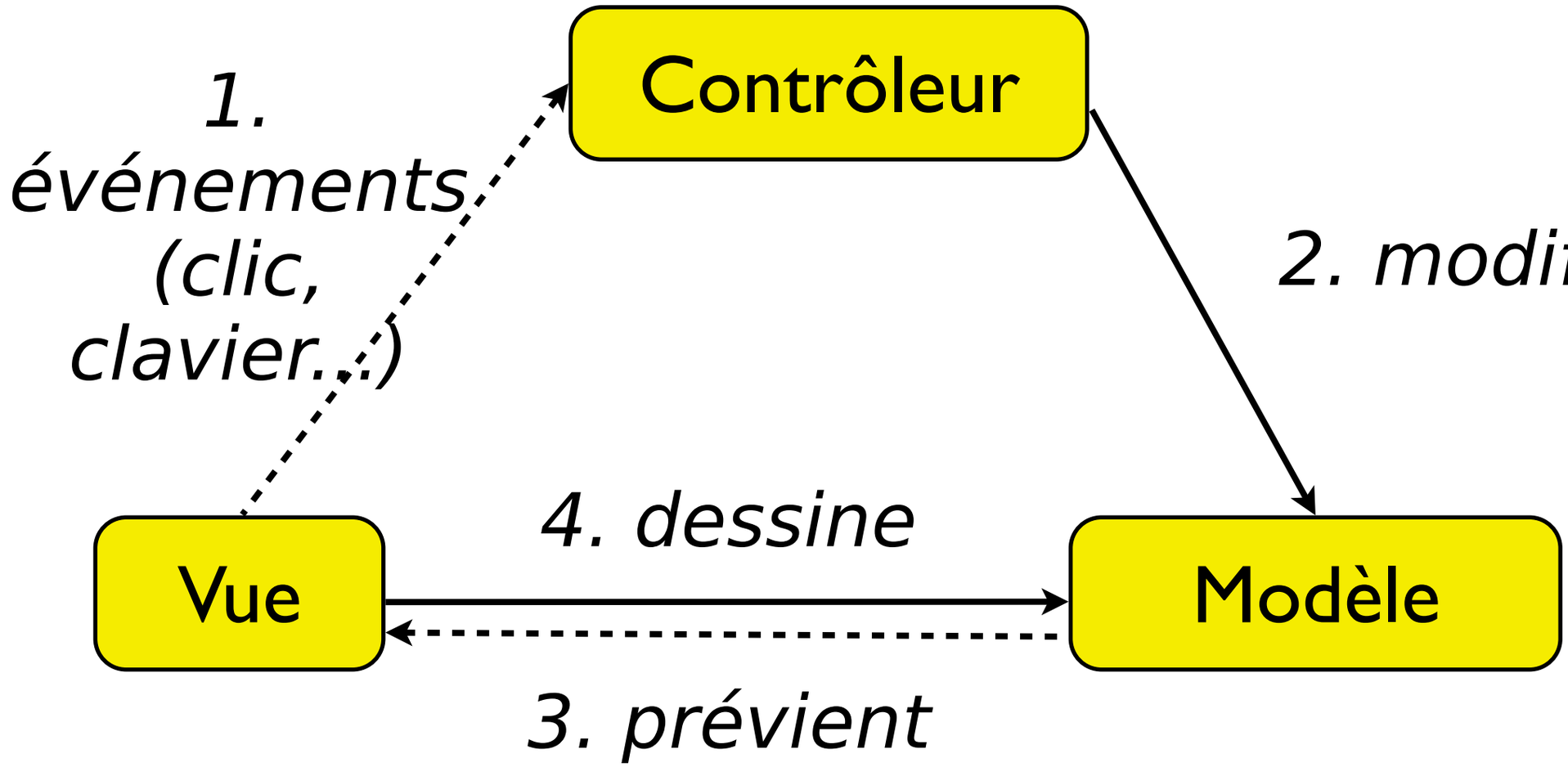
Style de l'élément sélectionné

Liste des pages et plan

Page courante



MVC



⟶ connaît le type

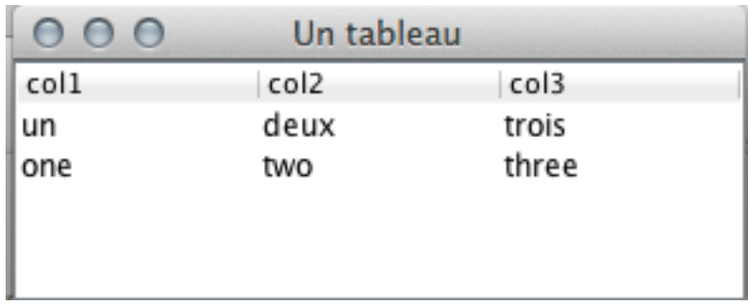
⇢ lien par une interface

(Il y existe des variantes)

Études de quelques composants

JTable

- Affiche un tableau à deux dimensions
- a priori, un nombre (relativement) fixe de colonnes, et un nombre de lignes variables
- utilise un TableModel




```
JFrame frame= new JFrame("Un tableau");
JTable table= new JTable(0, 3);
DefaultTableModel defaultModel= (DefaultTableModel)
                                                table.getModel();
defaultModel.setColumnIdentifiers(new String [] {
                                                col1", "col2", "col3"});
defaultModel.addRow(new String[] {"un", "deux", "trois"});
defaultModel.addRow(new String[] {"one", "two", "three"});
```

TableModel

- Interface qui décrit le contenu d'une table
- En pratique, on étend souvent AbstractTableModel (fournit certaines des méthodes)

TableModel en lecture seule

- trois informations nécessaires:
  - nombre de lignes
    - getRowCount()
  - nombre de colonnes
    - getColumnCount()
  - contenu d'une case
    - getValueAt(int rowIndex, int ColumnIndex)

Exemple : table de personnes

- Une personne:
  - un identifiant numérique (invariable)
  - un nom
  - un prénom

Modèle

```
public class PersonnesTableModelReadOnly extends AbstractTableModel {
    private List<Personne> personnes = new ArrayList<>();
    public PersonnesTableModelReadOnly(Collection<Personne> personnes ) {
        this.personnes= new ArrayList<>(personnes);
    }
   public int getRowCount() {
        return personnes.size();
    }
   public int getColumnCount() {
        return 3;
    }
   public Object getValueAt(int rowIndex, int columnIndex) {
        Personne p = personnes.get(rowIndex);
        switch (columnIndex) {
            case 0:
                return p.getId();
            case 1:
                return p.getNom();
            case 2:
                return p.getPrenom();
            default:
                return "";
        }
    }
}
```

Mise en place

```
public class DemoTablePersonnes {
    JFrame frame= new JFrame("Un tableau");
    JTable table= new JTable();
    public DemoTablePersonnes() {
        table.setModel(new PersonnesTableModelReadOnly(Personnes.getList()));
        frame.add(new JScrollPane(table));
        frame.setVisible(true);
        frame.pack();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
   public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> new DemoTablePersonnes());
    }
}
```

Tout TableModel

- Listeners
  - void addTableModelListener(TableModelListener l)
  - void removeTableModelListener(TableModelListener l)
- méta data
  - Class<?> getColumnClass(int columnIndex)
  - String getColumnName(int columnIndex)
- contenu
  - int getColumnCount()
  - int getRowCount()
  - Object getValueAt(int rowIndex, int columnIndex)
- modification
  - boolean isCellEditable(int rowIndex, int columnIndex)
  - void setValueAt(Object aValue, int rowIndex, int columnIndex)

Créer un modèle modifiable

- AbstractTableModel fournit des méthodes pour prévenir les listeners des modifications
  - void fireTableCellUpdated(int row, int column)

    contenu d'une case modifiée
  - void fireTableRowsDeleted(int firstRow, int lastRow)
  - void fireTableRowsInserted(int firstRow, int lastRow)
  - void fireTableRowsUpdated(int firstRow, int lastRow)

    *les coordonnées sont **inclusives**.*
  - void fireTableStructureChanged()

    structure modifiée (exemple: une colonne en plus)

permettre l'édition des champs

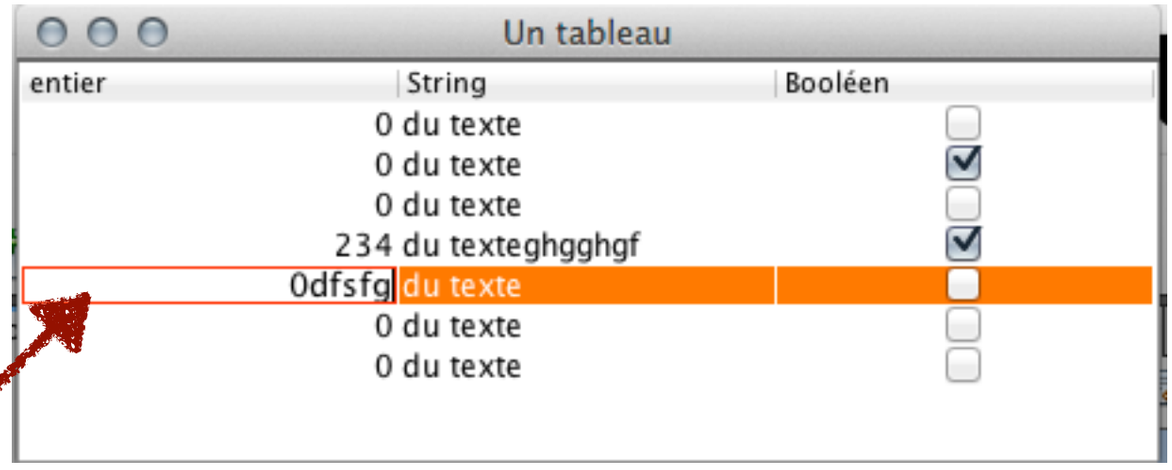
```
public class PersonnesTableModel extends AbstractTableModel {

    public boolean isCellEditable(int rowIndex, int columnIndex) {
         return columnIndex >= 1; // dans notre exemple, col. 0 non éditable
    }

    public void setValueAt(Object aValue, int rowIndex, int columnIndex) {
        Personne p = personnes.get(rowIndex);
        switch (columnIndex) {
            case 1:
                p.setNom((String)aValue);
                break;
            case 2:
                p.setPrenom((String)aValue);
                break;
        }
        fireTableCellUpdated(rowIndex, columnIndex);
    }
```

Type et noms des colonnes

```
public class PersonnesTableModel extends AbstractTableModel {

    public Class<?> getColumnClass(int columnIndex) {
        Class<?>[] columnClasses = {
            Integer.class, String.class, String.class
        };
        return columnClasses[columnIndex];
    }

    public String getColumnName(int column) {
        String[] titres = {"id", "nom", "prénom"};
        return titres[column];
    }
```

Classe des colonnes et éditeur

- Fixer la classe de la colonne permet à JTable de choisir
  - un éditeur adapté pour en modifier le contenu
  - une visualisation adaptée



erreur de saisie détectée

Ajout/suppression de lignes

- Dans le modèle:

```
    public void ajouterPersonne(Personne p) {
        int ligne= personnes.size();
        personnes.add(p);
        fireTableRowsInserted(ligne, ligne);
    }

    public void supprimerPersonne(int ligne) {
        personnes.remove(ligne);
        fireTableRowsDeleted(ligne, ligne);
    }
```

Ajout/suppression de lignes

- Dans le programme principal

```
public class DemoTablePersonnes {
   private JButton addPersonneButton= new JButton("+");
   private JButton removePersonneButton= new JButton("-");
   private PersonnesTableModel model;
public DemoTablePersonnes() {
    // Bug fix:
    table.putClientProperty("terminateEditOnFocusLost", Boolean.TRUE);
    …
    addPersonneButton.addActionListener(e-> addPersonne());
    removePersonneButton.addActionListener(e-> enleverPersonne());
    … }
 public void addPersonne() {
   Personne p= new Personne(100, "(nom)", "(prenom)");
    model.ajouterPersonne(p);
 }
 public void enleverPersonne() {
    int ligne= table.getSelectedRow();
     if (ligne != -1)
         model.supprimerPersonne(ligne);
 }
```

(petit détail ennuyeux)

```
table.putClientProperty("terminateEditOnFocusLost", Boolean.TRUE);
```

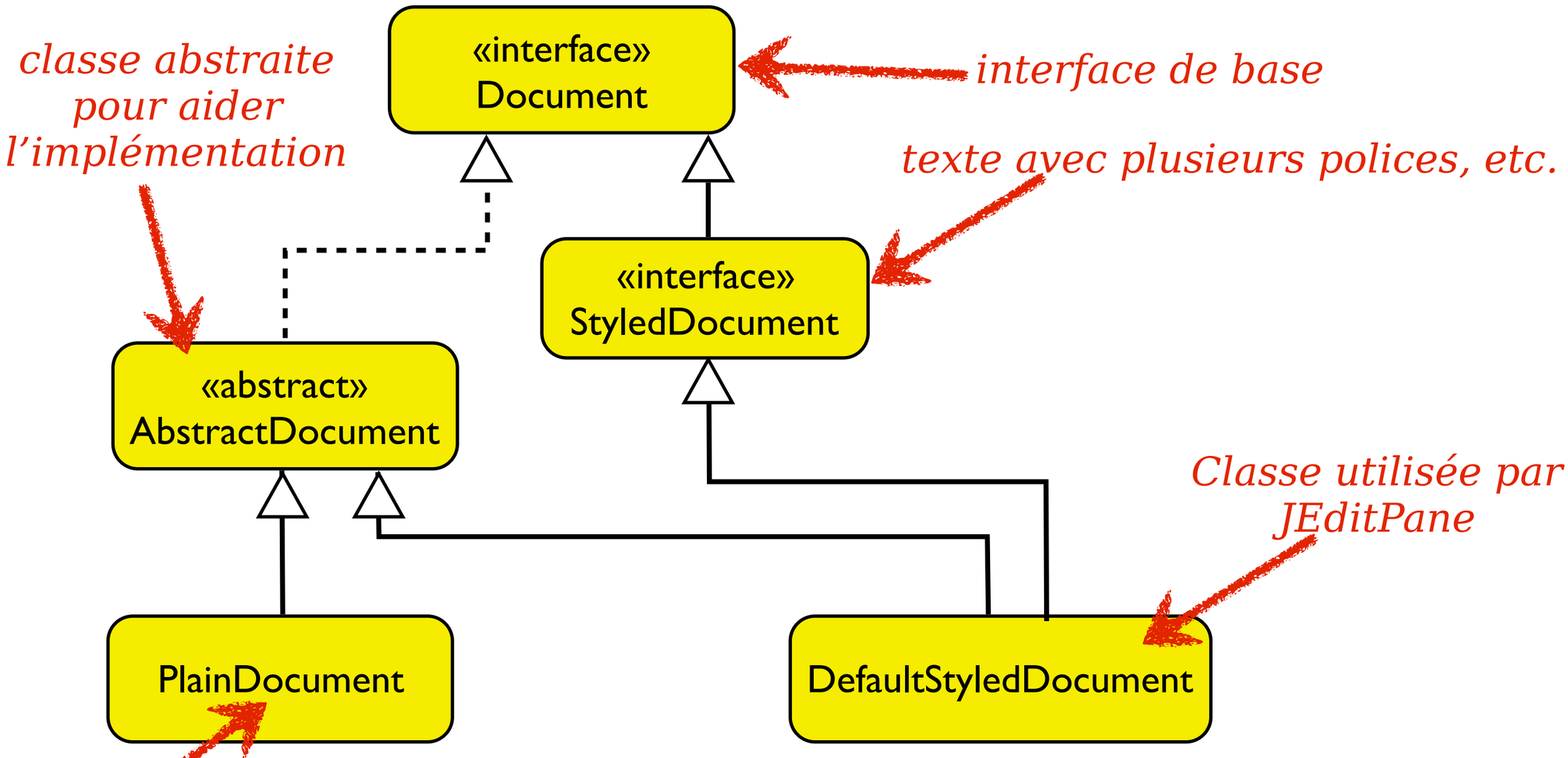
- la jtable tient à jour un éditeur qui est réutilisé pour les cases d'une colonne
- en cas de suppression d'une ligne, l'éditeur n'est pas prévenu par défaut (!)
- elle est déclenchée par la ligne ci-dessus
- voir

  http://bugs.java.com/bugdatabase/view_bug.do?bug_id=6349059
  http://bugs.java.com/bugdatabase/view_bug.do?bug_id=4709394
- Pour la petite histoire : la correction du bug n'est pas faite par défaut, par crainte de casser la compatibilité descendante.

Composants textuels

- JTextField, JTextArea, JFormatedTextField, JPasswordField, JEditorPane
- leur modèle implémente l'interface Document
  - méthodes getDocument(), setDocument()
- On peut attacher au document un DocumentListener pour être au courant des modifications dans le document

Document (modèle textuel)



En pratique...

- On peut déjà faire des choses sans créer son propre Document:
- on récupère le document sur le JTextField avec
  - Document doc= textField.getDocument()
  - et on lui attache un documentListener.

DocumentListener

```
public interface DocumentListener {
    /**
     * prévient d'une insertion de texte.
     */
    void insertUpdate(DocumentEvent e);
    /**
     * prévient d'une suppression de texte.
     */
    void removeUpdate(DocumentEvent e)
    /**
     * prévient d'une modification des styles (utilisé uniquement par JEditPane).
     */
    void changedUpdate(DocumentEvent e);
}
```

Exemple

La classe demo a deux champs textes, dont l'un sert à saisir un mot de passe («secret»). On veut afficher «correct» dès que celui-ci est bon.

```
class MonDocListener implements DocumentListener {
    private Demo demo;
    public MonDocListener(Demo demo) {
        this.demo = demo;
    }
    public void insertUpdate(DocumentEvent e) {
        verifierMotDePasse();
    }
    public void removeUpdate(DocumentEvent e) {
        verifierMotDePasse();
    }
    public void changedUpdate(DocumentEvent e) {
        // Jamais appelée
    }
    private void verifierMotDePasse() {
        if ("secret".equals(demo.getField().getText())) {
            demo.getAffichageField().setText("correct");
        } else {
            demo.getAffichageField().setText("incorrect");
        }
    }
}
```

Exemple... mise en place

....
Demo demo= new Demo();
demo.getField().**getDocument().addDocumentListener**(
    new MonDocListener(demo));

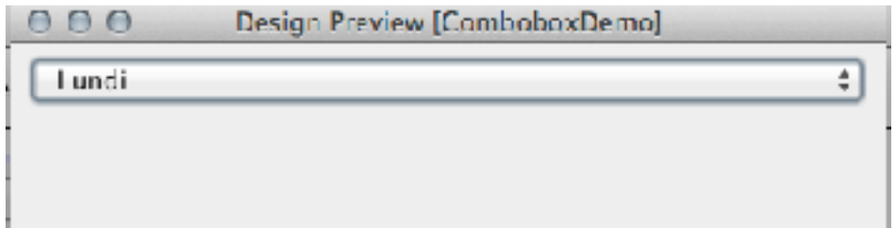
- Utile par exemple pour déclencher des recherches quand un champ texte est rempli

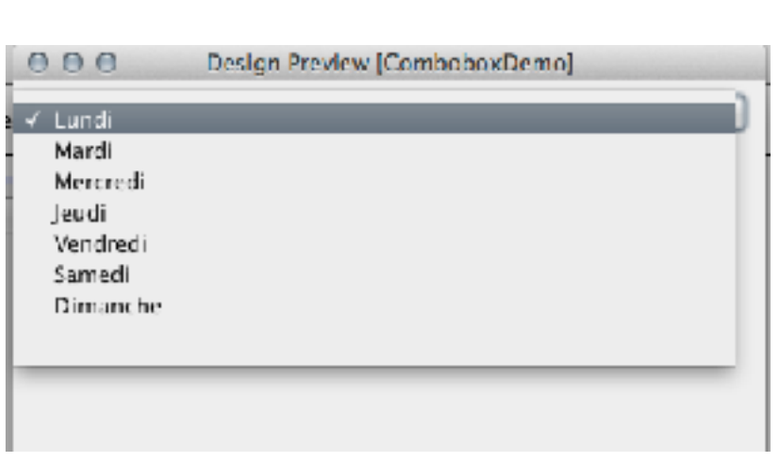
Annexe

- La fin de ces transparents n'est pas au programme de *l'examen*. Ceci dit, les composants qui y sont décrits fonctionnent de manière similaire à JTable. Si vous avez besoin de les utiliser, vous devriez pouvoir vous y retrouver.

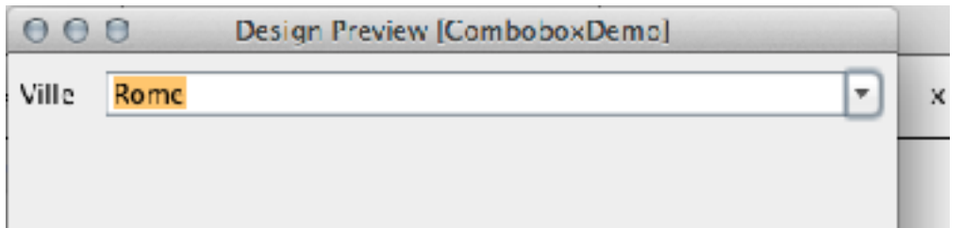
JCombobox

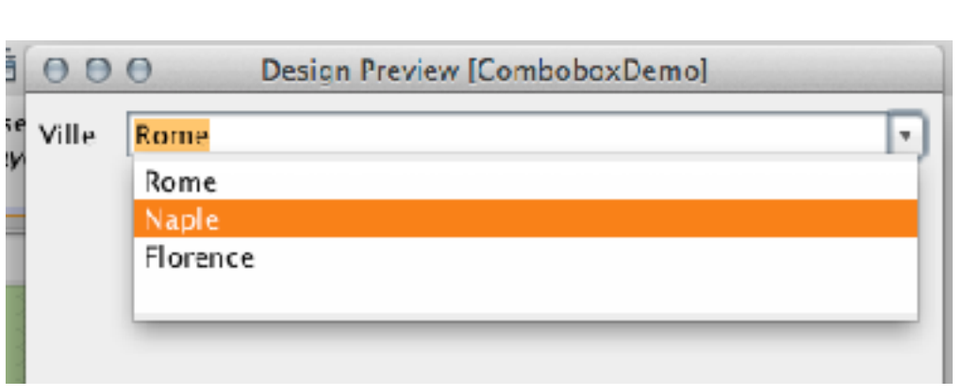
non éditable

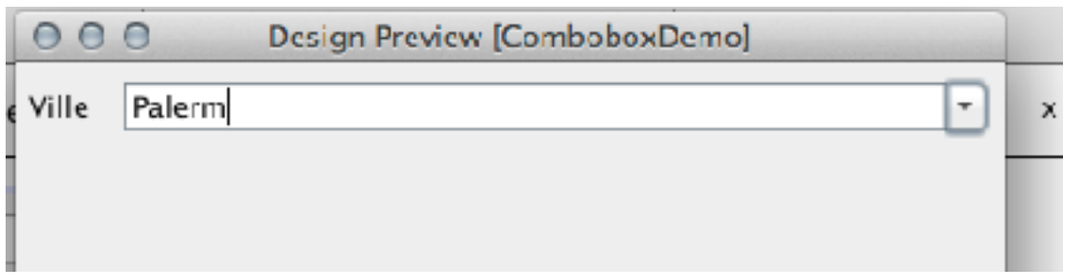




éditable







On peut presque tout changer

JCombobox : modèle

- Interface ComboBoxModel (type générique depuis jdk 1.7! avant : **Object** au lieu de T)
- Implantation par défaut : DefaultComboboxModel
- *Les entrées du modèle sont par défaut visualisées en utilisant toString().*

JCombobox (Constructeurs)

- JComboBox() : construit une combobox avec un DefaultComboboxModel
- JComboBox(ComboBoxModel)
- JComboBox(Object[]) : construit une combobox avec un DefaultComboboxModel qui contient les objets du tableau.
- JComboBox(Vector) JComboBox(Object[]) : construit une combobox avec un DefaultComboboxModel qui contient les objets du Vector (équivalent ArrayList).

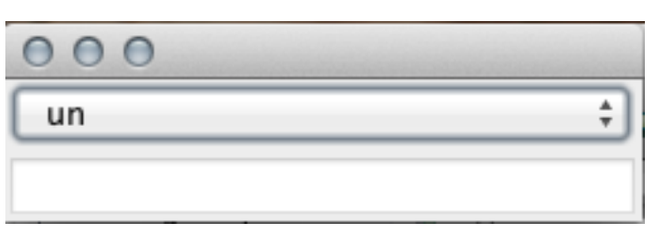
JCombobox (méthodes utiles)

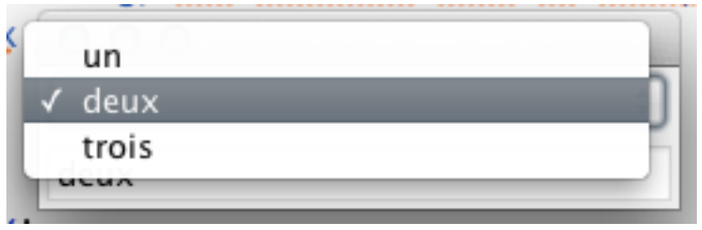
- addActionListener(ActionListener) : l'action listener est appelé après une édition, ou après une sélection. Alternative : il existe aussi une interface ItemListener (plus complexe, voir tutoriel Swing)
- getItemCount() , getItemAt(int) : permet de lister les entrées de liste (mais on peut utiliser le modèle)
- getSelectedItem() / setSelectedItem(Object) : renvoie ou fixe l'item sélectionné ; getSelectedIndex() renvoie sa position dans la liste.
- getModel()/setModel(ComboboxModel) : gère le modèle.
- setEditable(boolean) : l'utilisateur peut-il saisir de nouvelles entrées ?

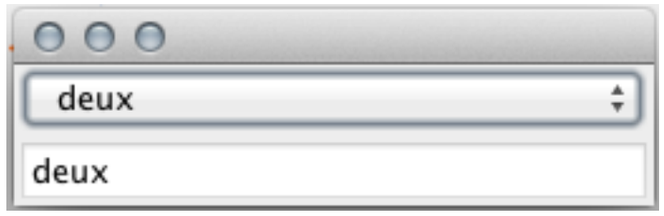
Exemple

```
public class SimpleCBDemo {
  private JComboBox comboBox;
  private JTextField textField;

  public SimpleCBDemo() {
   String[] tab= {"un", "deux", "trois"};
   comboBox= new JComboBox(tab);
   textField= new JTextField(20);
   comboBox.addActionListener(new RecopierActionListener(this));
   mettreEnPage();
  }

  public void recopier() {
   String texte= (String) comboBox.getSelectedItem();
   textField.setText(texte);
  }
  ....
}

class RecopierActionListener implements ActionListener {
  SimpleCBDemo simpleCBDemo;
  public RecopierActionListener(SimpleCBDemo simpleCBDemo) {
   this.simpleCBDemo = simpleCBDemo;
  }
  public void actionPerformed(ActionEvent arg0) {
   simpleCBDemo.recopier();
  }}
```

Exemple







DefaultComboBoxModel

- Représente l'état de la combobox : liste des choix possibles, et item sélectionné
- méthodes
  - getSelectedItem() ; setSelectedItem(Object): gestion de la sélection
- getSize() ; getElementAt(int) : liste des items
- addElement(Object); insertElementAt(int i) ; removeElementAt(int i); removeElement(Object); removeAllElements() : modification de la liste des items
- lors de grosses modifications, on remplace souvent l'intégralité du modèle plutôt que de modifier son contenu.

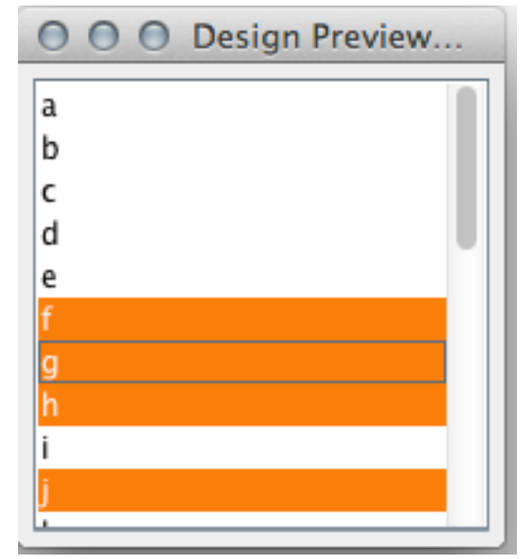
Manipulation du modèle à travers la Combobox

- Plus simple, mais passe par l'interface graphique
- La classe JCombobox fournit les méthodes:
  - addItem(Object item)
  - insertItemAt(Object item, int pos)
  - removeItem(Object item)
  - removeItemAt(int pos)
  - removeAllItems()

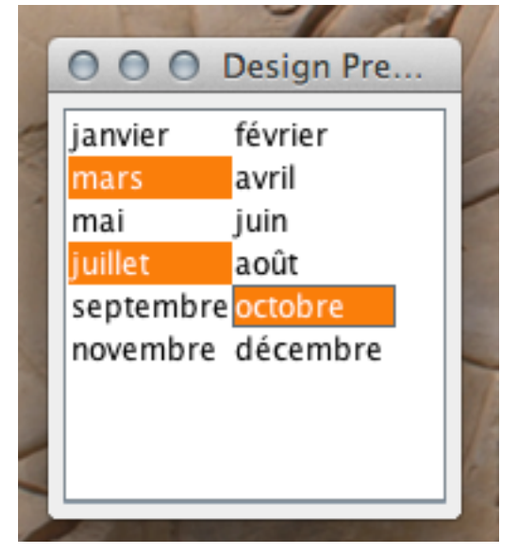
Pour aller plus loin...

- Dans le JCombobox, par exemple :
  - on peut créer son propre modèle (par exemple lié à une base de données)
  - on peut remplacer le système de rendu des éléments (Renderer) : ListCellRenderer
  - on peut remplacer l'éditeur (l'objet chargé de prendre en charge la saisie d'une nouvelle entrée) : ComboBoxEditor
- pour la classe JTable, l'écriture de modèles est souvent très utile

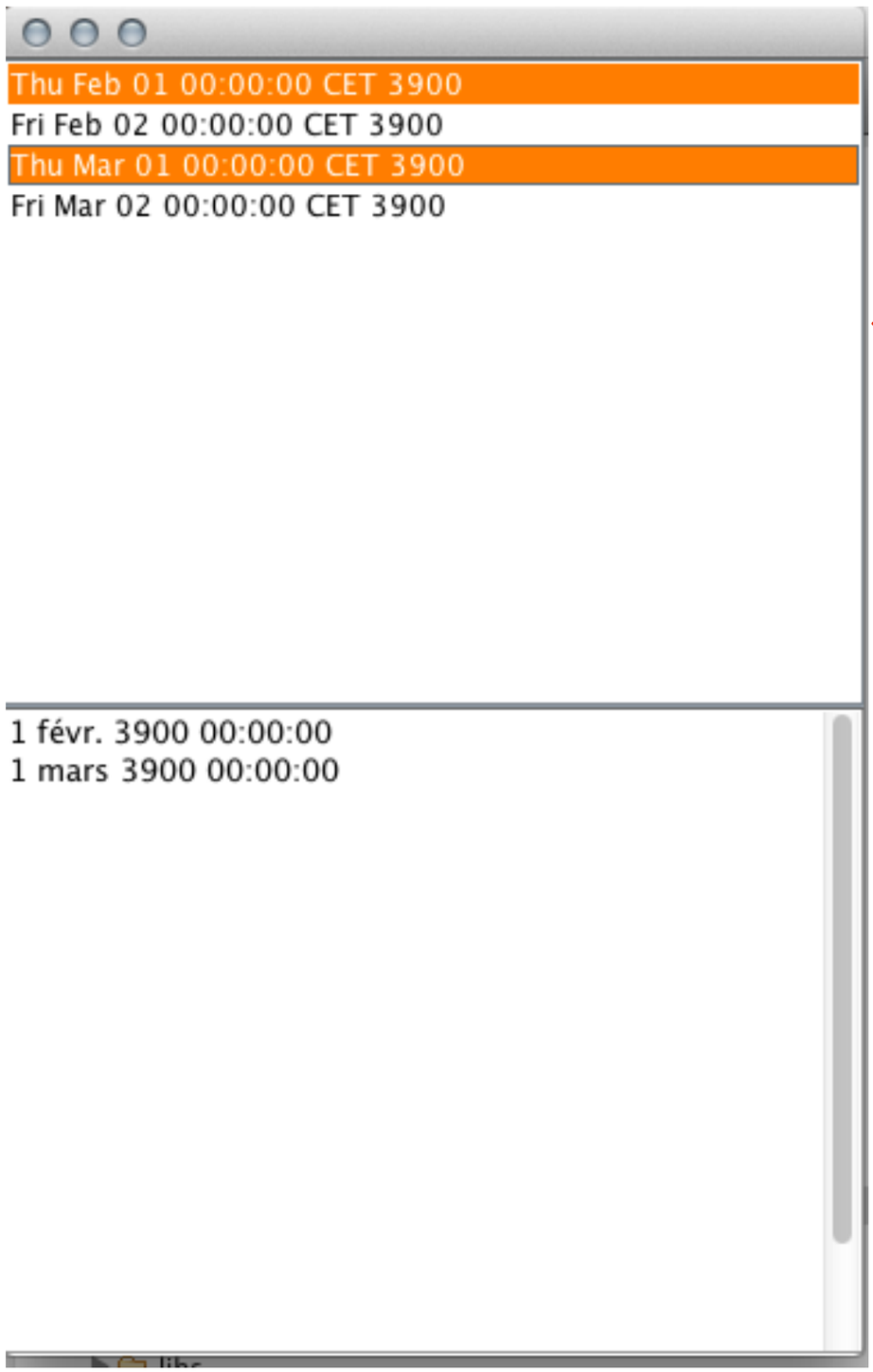
JList



- Une liste dans laquelle on peut faire un ou plusieurs choix.
- Souvent placée dans un JScrollPane pour bénéficier des ascenseurs.
- modèle : interface ListModel, implantée par DefaultListModel
- plusieurs dispositions possibles



Exemple



une liste de dates

affichage des dates de la sélection

Exemple

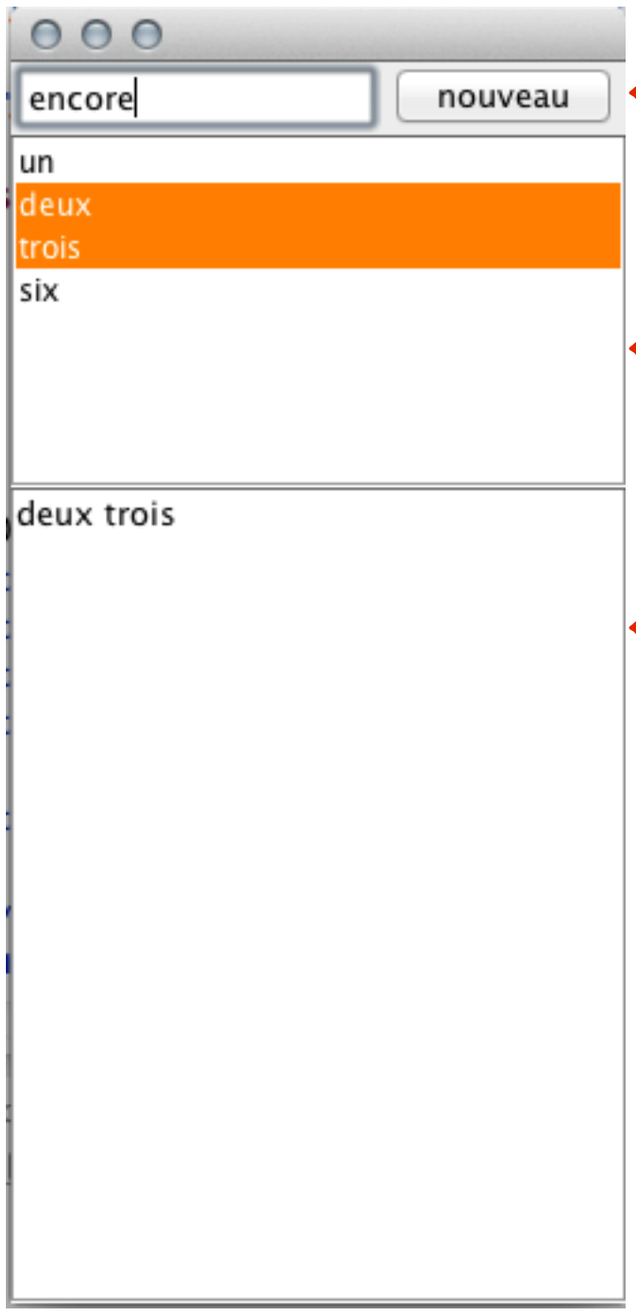
```
public class SimpleListDemo {
  private JList listComponent;
  private JTextArea textField;
  public SimpleListDemo() {
   // On fait une liste de dates, juste pour
   // montrer qu'on n'est paslimité aux Strings...
   Date[] tab= {new Date(2000,1,1), new Date(2000,1,2),
         new Date(2000,2,1),new Date(2000,2,2)};
   listComponent= new JList(tab);
   textField= new JTextArea(20,10);
      listComponent.addListSelectionListener(
           new MonSelectionListener(this));
   mettreEnPage();
  }

  public void recopier() {
   String texte = "";
   for (Object o : listComponent.getSelectedValues()) {
      Date d= (Date) o;
      texte= texte + d.toLocaleString()+ "\n";
   }
   textField.setText(texte);
  }
...
}
```

```
class MonSelectionListener implements ListSelectionListener {

  SimpleListDemo simpleListDemo;

  public MonSelectionListener(SimpleListDemo simpleListDemo) {
   this.simpleListDemo = simpleListDemo;
  }

  @Override
  public void valueChanged(ListSelectionEvent e) {
   simpleListDemo.recopier();
  }

}
```

Utilisation du modèle

- Par exemple pour modifier les données affichées par la liste
- On peut souvent utiliser le modèle par défaut proposé

Exemple



Ajout de nouveaux éléments à la liste

la liste...

Affichage de la sélection

```
public class DefaultListModelDemo {
  private JTextField nouvelleEntreeField;
  private JButton ajouterButton;
  private JList listComponent;
  private JTextArea affichageField;
  private DefaultListModel listModel;

  public DefaultListModelDemo() {
   listModel = new DefaultListModel();
   listModel.addElement("un");
   listModel.addElement("deux");
   listModel.addElement("trois");
   listComponent = new JList(listModel);

   nouvelleEntreeField = new JTextField(10);
   ajouterButton = new JButton("nouveau");
   affichageField = new JTextArea(20, 20);
   affichageField.setEditable(false);
   ajouterButton.addActionListener(new AjouterListener(this));
   listComponent.addListSelectionListener(
        new MonSelectionListener(this));
   mettreEnPage();
  }

  public void ajouteEntree() {....
```

```
public class DefaultListModelDemo {
  ....

  public void ajouteEntree() {
   String texte = nouvelleEntreeField.getText();
   listModel.addElement(texte);
  }

  public void recopier() {
   String texte = "";
   for (Object o : listComponent.getSelectedValues()) {
      String s = (String) o;
      texte += s + " ";
   }
   affichageField.setText(texte);
  }
}
```

```
class AjouterListener implements ActionListener {
  DefaultListModelDemo simpleListDemo;

  public AjouterListener(DefaultListModelDemo simpleListDemo) {
   this.simpleListDemo = simpleListDemo;
  }
  public void actionPerformed(ActionEvent e) {
   simpleListDemo.ajouteEntree();
  }
}
```

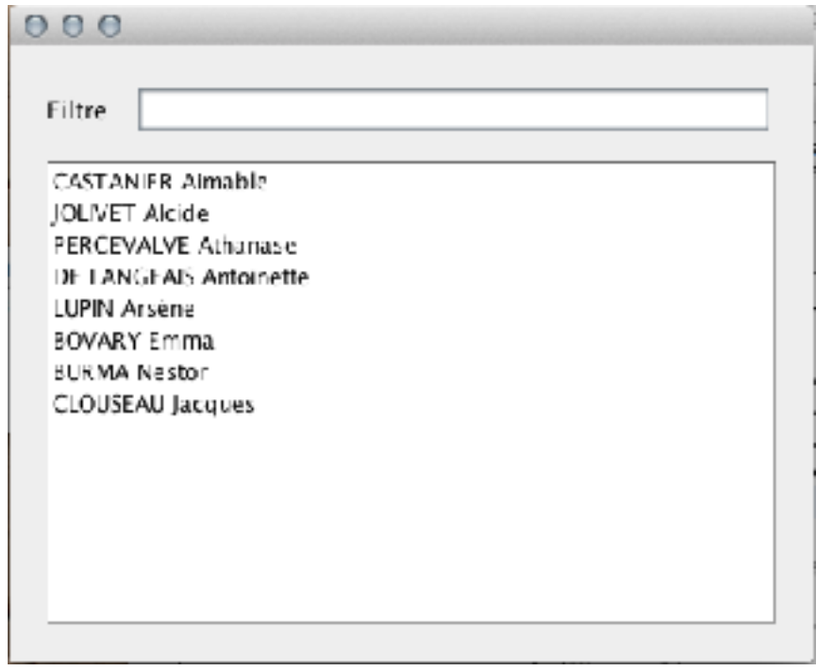
```
class MonSelectionListener implements ListSelectionListener {

  DefaultListModelDemo simpleListDemo;

  public MonSelectionListener(DefaultListModelDemo simpleListDemo) {
   this.simpleListDemo = simpleListDemo;
  }

  public void valueChanged(ListSelectionEvent e) {
   simpleListDemo.recopier();
  }
}
```

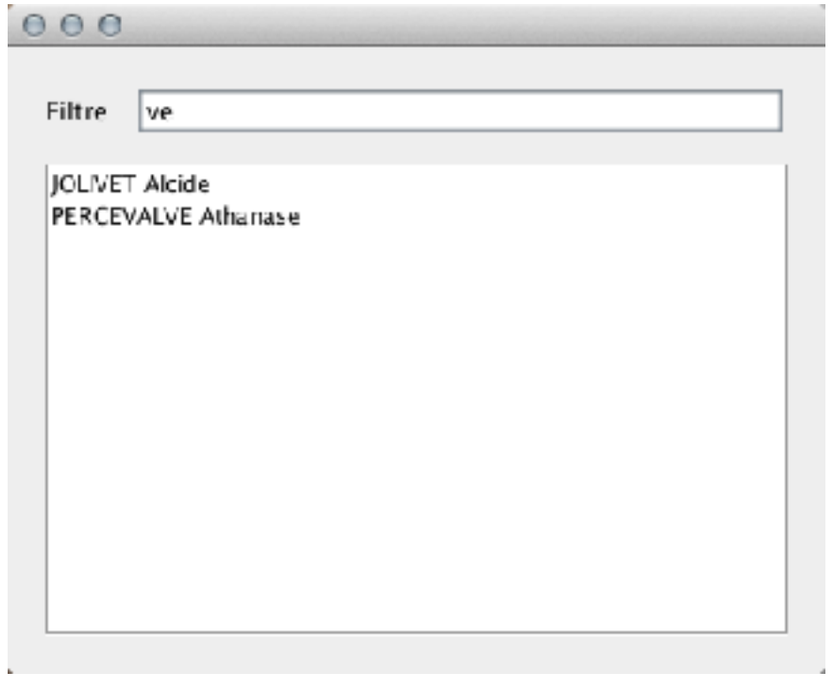
Créer ses modèles

- Utile quand les modèles de base ne suffisent pas
- par exemple: on veut, sans les recopier, récupérer les données d'une ArrayList ou d'une base de données
- On dispose généralement d'une classe de base, qui limite le nombre de méthodes qu'on devra écrire:
  - AbstractListModel, AbstractDocument...

Un modèle pour JList

- Exemple: on veut un modèle de JList qui soit
  - basé sur une collection d'éléments
  - qui soit doté d'un filtre, qui permette de n'afficher que certains éléments.
- Pour cela on va étendre AbstractListModel
- remarque: les JList sont sans doute l'exemple le moins utile, mais en revanche, c'est le plus simple.





Étendre AbstractListModel

- deux méthodes à écrire:
  - public int getSize() : taille de la liste
    - public T getElementAt(int i) : élément numéro i
- *Note : si vous utilisez java <= 1.6, remplacez T par «Object».*

```
public class FilterModele extends AbstractListModel<Personne> {
    private List<Personne> filtrees; // liste après filtre
    private Collection<Personne> base; // la source
    private String filtre = ""; // le filtre...

    public FilterModele(Collection<Personne> base) {
        this.base = base;
        this.filtrees = new ArrayList<Personne>(base);
    }
    @Override
    public int getSize() {
        return filtrees.size();
    }
    @Override
    public Personne getElementAt(int index) {
        return filtrees.get(index);
    }
```


...

Modification des données du modèle

- Quand on change le filtre dans notre exemple, le contenu affiché de la liste va changer
- Le modèle doit donc prévenir sa ou ses vues qu'il a changé
- la classe AbstractListModel fournit plusieurs méthodes...

Pattern observateur et ListModel

- void addListDataListener(ListDataListener l)
  - appelée automatiquement par le JList quand on lui donne un modèle
- void **fireContentsChanged**(Object source, int i0, int i1)
  - les valeurs entre i0 et i1, inclus ont changé. Pas d'ajout ou de suppression !!
- void **fireIntervalAdded**(Object source, int i0, int i1)
  - on a ajouté des valeurs entre les indices i0 et i1, inclus
- void **fireIntervalRemoved**(Object source, int i0, int i1)
  - on a enlevé des valeurs entre les indices i0 et i1, inclus.
- Ces trois méthodes sont **appelées par le modèle lui même** quand il est modifié, pour prévenir les listeners des modifications

```
public class FilterModele extends AbstractListModel<Personne> {
    private List<Personne> filtrees;
    private Collection<Personne> base;
    private String filtre = "";
...
    public void setFiltre(String filtre) {
        this.filtre = filtre.toLowerCase(); // On va ignorer majuscules/minuscules.
        int ancienneLongueur = this.filtrees.size();
        if (ancienneLongueur > 0) { // On efface l'ancienne liste, si besoin.
            this.filtrees.clear();
            // Prévient que les éléments entre les indices
            // 0 et ancienneLongueur -1, inclus, ont été enlevés.
            fireIntervalRemoved(this, 0, ancienneLongueur - 1);
        }
        for (Personne p : base) { // Filtrage à proprement parler.
            if (p.getNom().toLowerCase().contains(filtre)) {
                this.filtrees.add(p);
            }
        }
        // On prévient les observateurs de la modification...
        if (this.filtrees.size() > 0) {
            fireIntervalAdded(this, 0, this.filtrees.size() - 1);
        }
}
```

Mise en place...

```
public class DemoFiltreApplication {

    FiltreFrame frame = new FiltreFrame();
    FilterModele filterModele;

    public DemoFiltreApplication() {
        remplirModele(); // crée et remplit filtreModele
        frame.getPersonneListe().setModel(filterModele); // On le met en place
       // La frappe dans le champ texte va modifier la valeur du filtre directement
        frame.getFiltreChamp().getDocument().addDocumentListener(new DocumentListener() {
            public void insertUpdate(DocumentEvent e) {
                filterModele.setFiltre(frame.getFiltreChamp().getText());
            }
            public void removeUpdate(DocumentEvent e) {
                filterModele.setFiltre(frame.getFiltreChamp().getText());
            }
            public void changedUpdate(DocumentEvent e) {
            }
        });
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
}
```

manipuler les sélection

- jlist.getSelectedValues() : renvoie les objets sélectionnés
- jlist.clearSelection() : efface la sélection
- jlist.setSelectionMode(mode) : fixe le mode de sélection; SINGLE_SELECTION, SINGLE_INTERVAL_SELECTION, ou MULTIPLE_INTERVAL_SELECTION
- jlist. addListSelectionListener(sel) : ajoute un ListSelectionListener qui recevra les événements concernant la sélection

ListSelectionListener

- Très simple : une seule méthode
  - void valueChanged(ListSelectionEvent e)
- Qu'est-ce que ListSelectionEvent
  - une source (la JList généralement)
  - firstIndex et lastIndex: les limites de l'espace modifié (**inclusives**)
  - isAdjusting : un booléen qui permet de savoir si la modification est effectuée dans une opération atomique (false) ou complexe (true). Par exemple, avec un «drag», on aura isAdjusting à true.
- On peut dans tous les cas regarder la valeur actuelle de la sélection.

Pour aller plus loin...

- Dans le JCombobox, par exemple :
  - on peut remplacer le système de rendu des éléments (Renderer) : ListCellRenderer
  - on peut remplacer l'éditeur (l'objet chargé de prendre en charge la saisie d'une nouvelle entrée) : ComboBoxEditor
  - pour la classe JTable, l'écriture de modèles est souvent très utile