

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

swing_cours2 — Modifiée

Nouveau Lecture Afficher Guides Thèmes Modèles Zone de texte Figures Tableaux Graphiques Commentaire Masquer Alpha Regrouper Dissocier Premier plus Dernier Inspecteur Données multimédias Couleurs

Gill Sans Normal 42

Structure

- 1 Cours 2 : Architecture MVC et composants Swing
+NFA035
+S. Rosmorduc
- 2 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 ?
- 3 Problème : visualisations multiples du même modèle
- 4 Études de quelques composants
- 5 JCombobox
- 6 JList
- 7 Composants textuels
- 8 Créer ses modèles
- 9 Un modèle pour JList
- 10 Adapter les sélections
- 11 Un exemple : listes communicantes
- 12

Le MVC

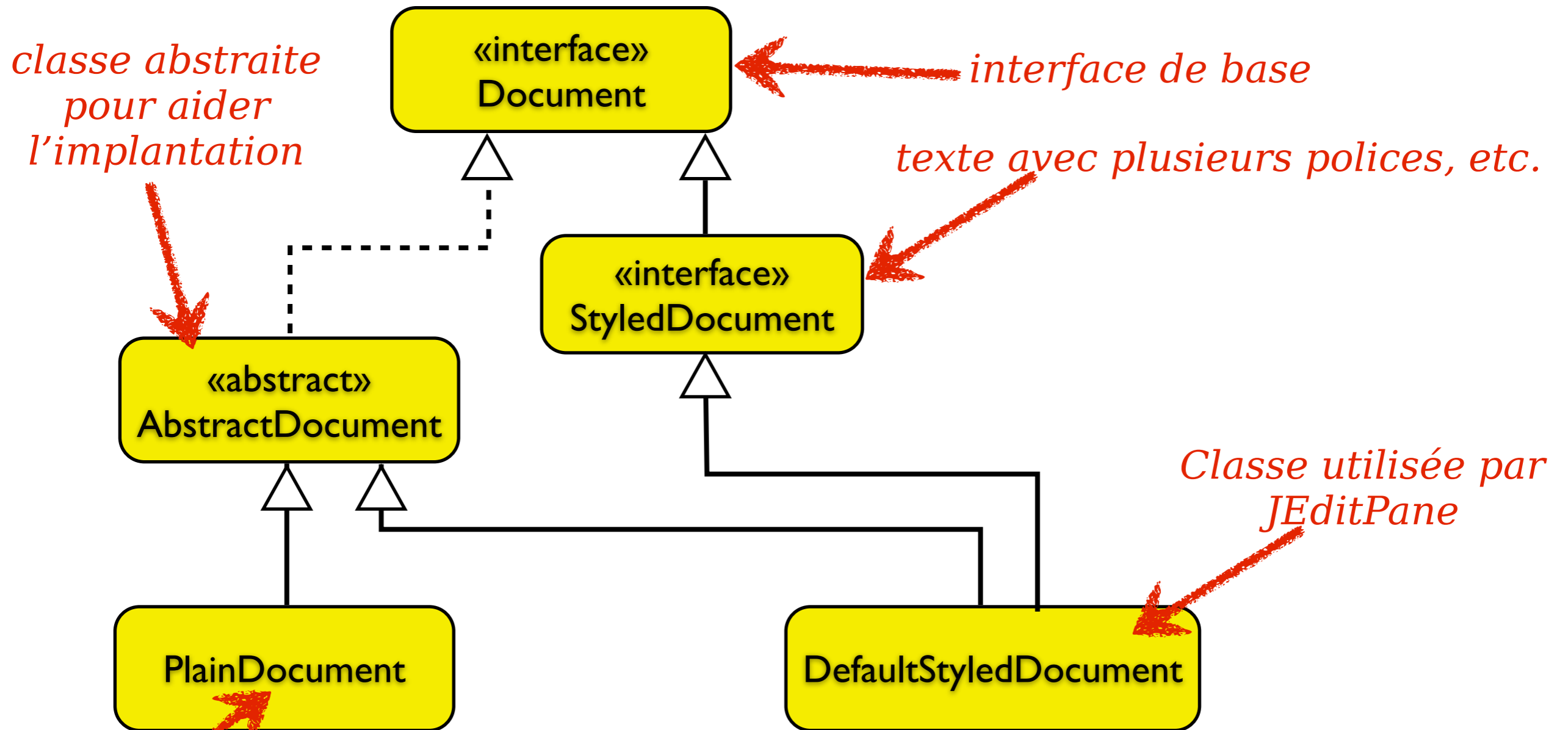
- Programmation en couches
- Le modèle : représentation des données indépendante de son interface utilisateur
- Problème : comment synchroniser modèle et interface graphique de manière simple, sans que le modèle ne «connaisse» son interface utilisateur ?

Études de quelques composants

Composants textuels

- JTextField, JTextArea, JFormattedTextField, 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)



Classe utilisée par les éditeurs «simples» :
`JTextField`, `JTextArea`

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

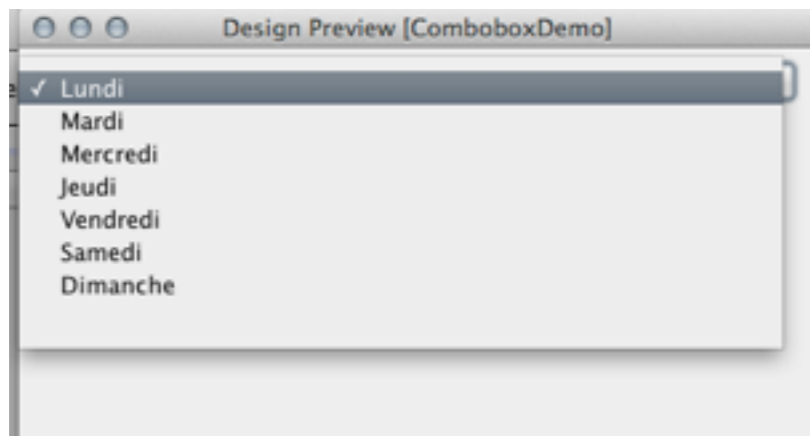
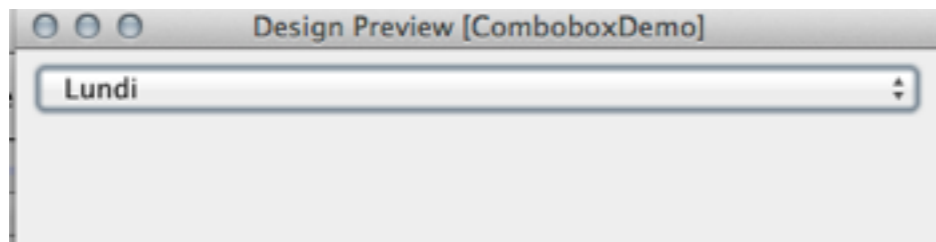
....

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

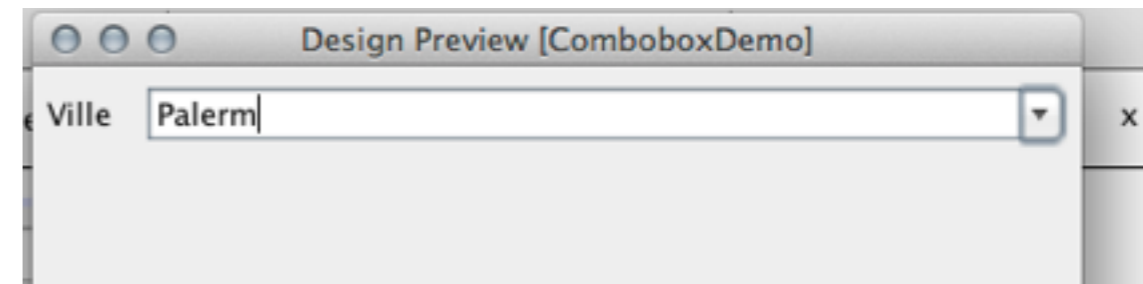
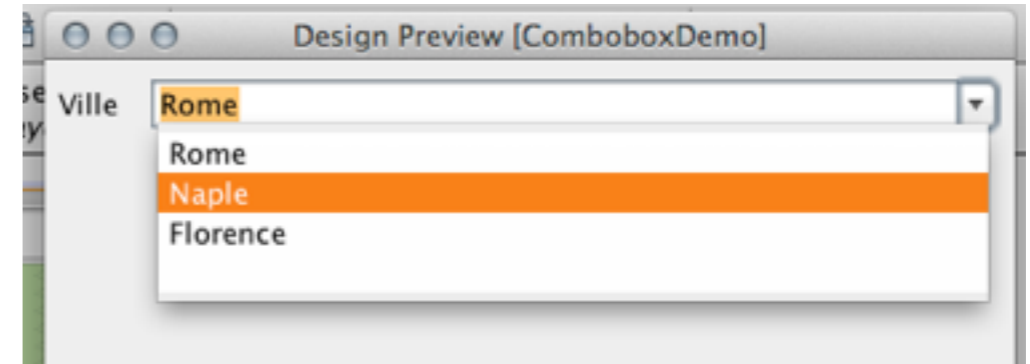
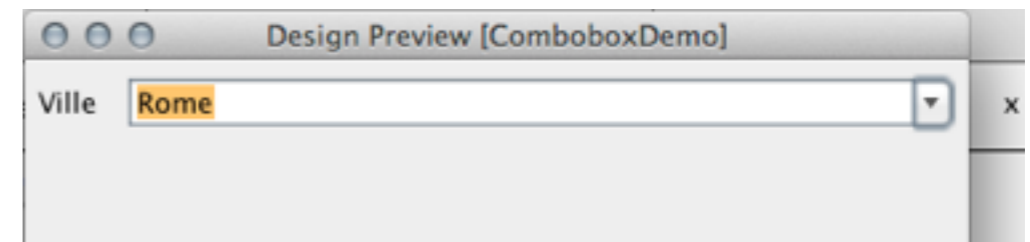
- Utile par exemple pour déclencher des recherches quand un champ texte est rempli

JComboBox

non éditable



éritable



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`.
- `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 ; `getSelectedItemIndex()` renvoie sa position dans la liste.
- `getModel()/setModel(ComboBoxModel)` : gère le modèle.
- `setEditable(boolean)` : l'utilisateur peut-il saisir de nouvelles entrées ?

Exemple

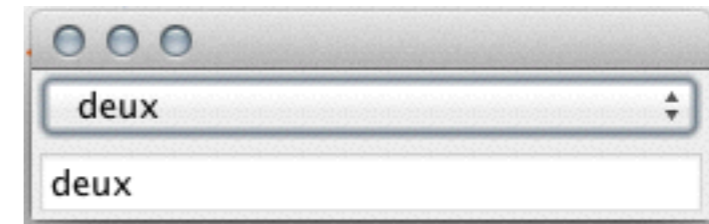
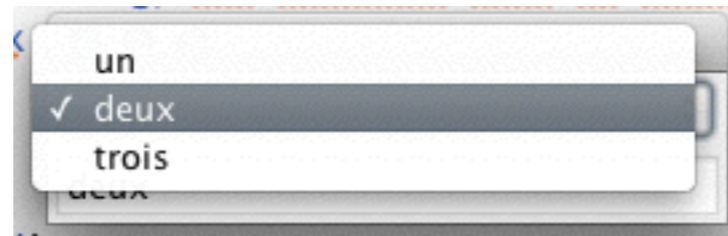
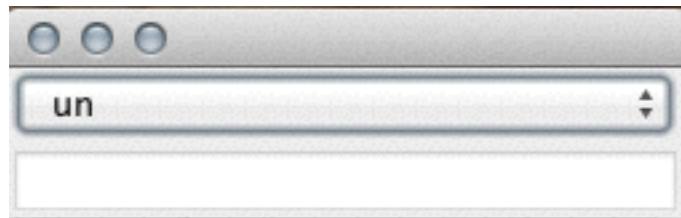
```
public class SimpleCBDemo {
    private JComboBox comboBox;
    private JTextField textField;

    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

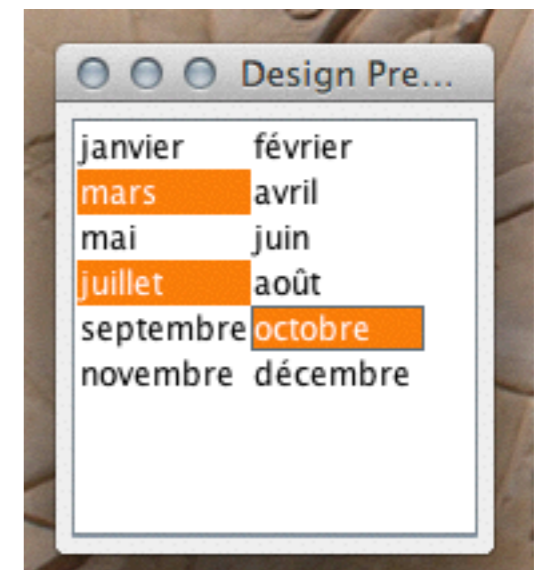
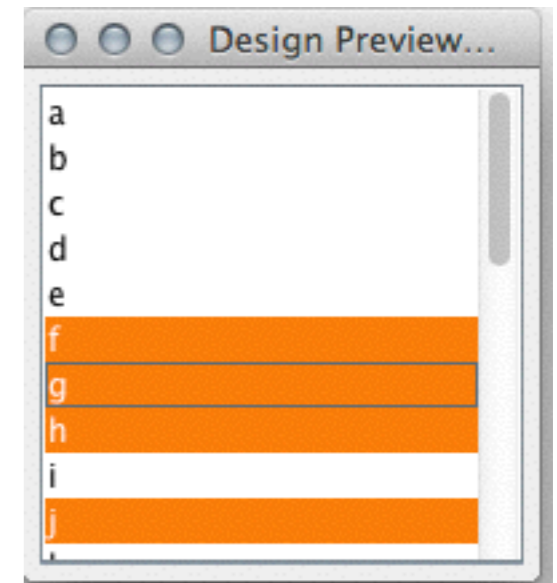


DefaultComboBoxModel

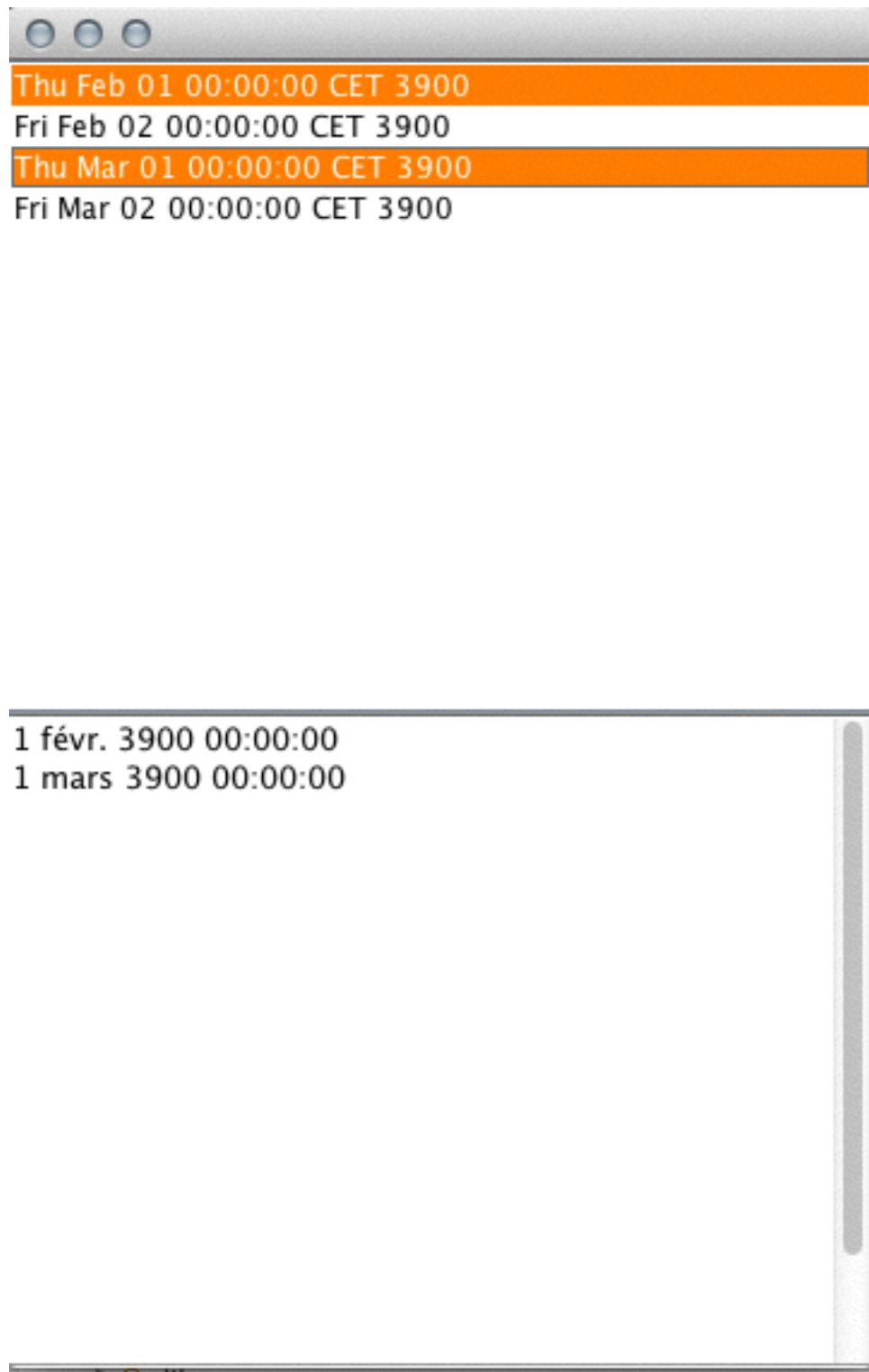
- Représente l'état de la combobox : liste des choix possibles, et item sélectionné
- méthodes
 - `getSelectedItem()` ; `setSelectedItem(Object)`: gestion de la sélection
- `getSize()` ; `getElementAt(int)` : liste des items
- `addElement(Object)`; `insertElementAt(int i)` ; `removeElementAt(int i)`; `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.

JList

- Une liste dans laquelle on peut faire un ou plusieurs choix.
- Souvent placée dans un JScrollPane pour bénéficier des ascenseurs.
- modèle : interface ListModel, implémentée 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 pas limité 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;
    }

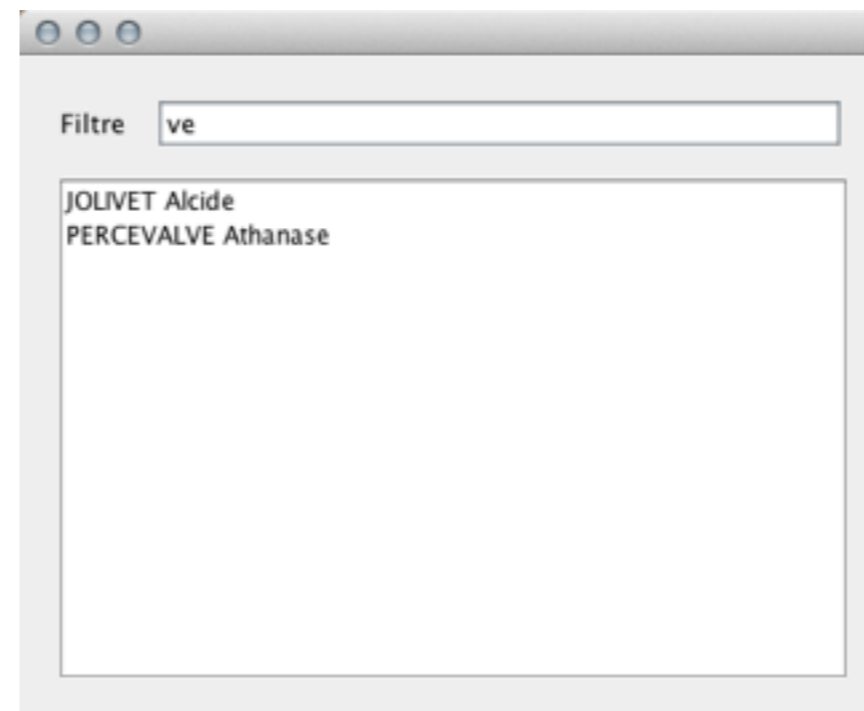
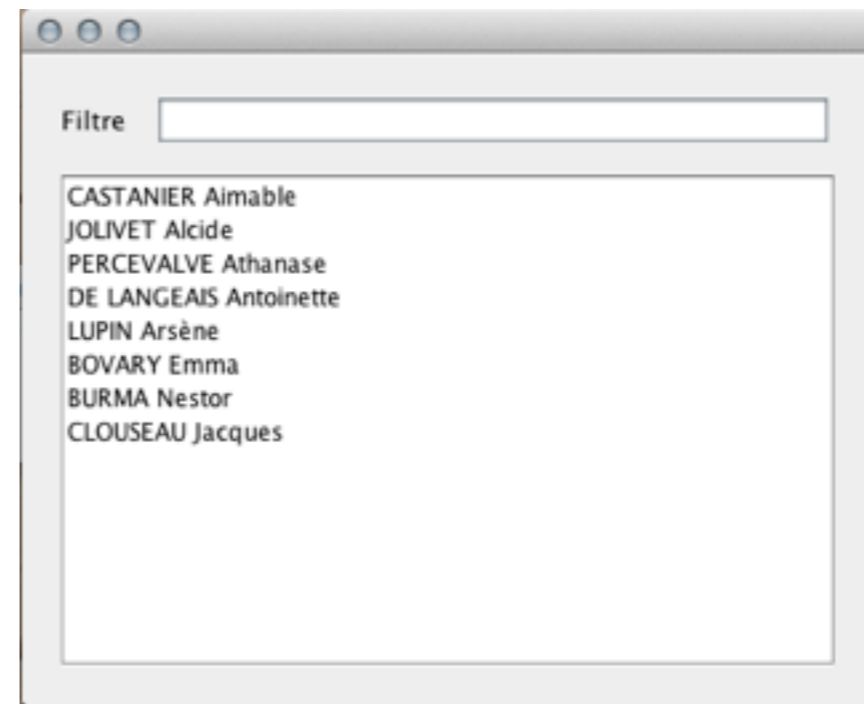
    public void valueChanged(ListSelectionEvent e) {
        simpleListDemo.recopier();
    }
}
```

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 \leq 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