IUT Lyon 1
l'excellence technologique

Programmation d'IHM- Cours 4
Barres d'outils
Architecture MVC, JList

V. DESLANDRES

veronique.deslandres@univ-lyon1.fr

Sommaire

Architecture MVC
Modèle, Vue, Contrôle

Introduction

- Pour visualiser et manipuler un gros volume d'informations : composants spécifiques
- Les informations peuvent être présentées sous forme de **tableau**, de **liste**, **d'arbre** ou de **graphe**
- L'API Java Swing propose plusieurs composants pour visualiser les informations :
  - *JTable, JList, JTree, JGraph,…*

MVC: Principes de base

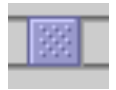
- Le modèle d'architecture MVC (Model View Controller) est à la base de nombreux systèmes de visualisation graphiques
- Principe de Base: **séparation des rôles**
  - Le ***modèle*** est l'élément principal du composant, il contient les **données**
  - Les ***vues*** du composant sont **des visualisations des données d**u modèle : une vue s'abonne à un modèle, et se met à jour quand les données du modèle évoluent
  - Le ***contrôleur*** assure la synchronisation entre modèle et vues (**traitement**)
- La Java Swing repose sur l'architecture M-VC
  - (càd que Vue et Contrôleur sont souvent dans le même composant graphique, séparés du Modèle)

MVC exemple (JSlider)

Quelles sont les données associées à un *slider* ?

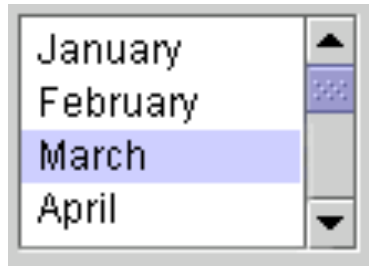
- *Modèle* :
  - valeur minimale = 0
  - valeur courante = 15
  - valeur maximale = 100

- *Vue* :

- *Contrôleur* :
  - Traiter les clics de souris sur les boutons terminaux 
  - Gérer les *drags* de souris sur l'ascenseur 

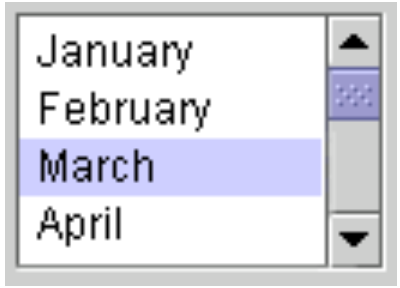
Modèles MVC des composants SWING

- Il existe en SWING des composants génériques pour les **modèles** des données
- `JList :`
  - `classe` **`ListModel`** `pour les données`
  - `classe` **`ListSelectionModel`** `pour gérer les sélections`
- `JTable :`
  - classe **`TableModel`** pour les données
  - classe **`TableColumnModel`** pour définir les colonnes
  - classe **`ListSelectionModel`** pour gérer les sélections



Host	User	Password	Last Modified
Biocca Games	Freddy	!#asf6Awwzb	Mar 16, 2006
zabble	ichabod	Tazb!34$fZ	Mar 6, 2006
Sun Developer	fraz@hotmail.com	AasW541!fbZ	Feb 22, 2006
Heirloom Seeds	shams@gmail.com	bkz[ADF78!	Jul 29, 2005
Pacific Zoo Shop	seal@hotmail.com	vbAf124%z	Feb 22, 2006

Le composant JList



Caractéristiques de base

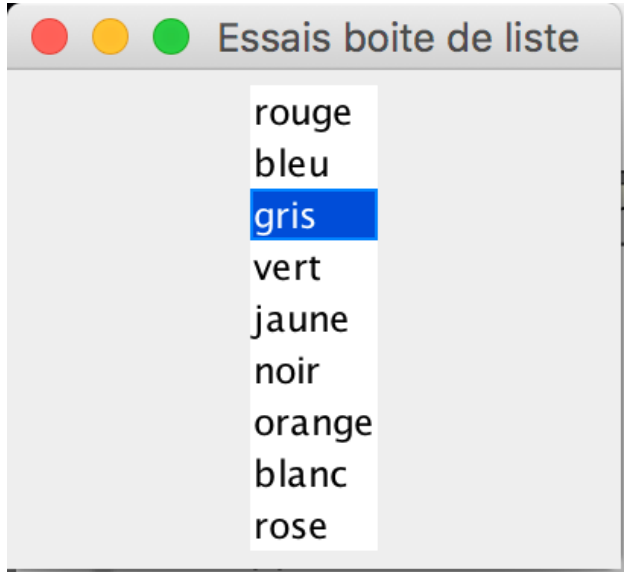
- Une **JList** est une présentation des données sous forme de liste
  - Affichage d'une liste d'items :
- 2 types de **JList**
  - Liste **statique** : sélectionner des éléments
  - Liste **dynamique** : la liste des items peut évoluer
- Modalité de sélection : **simple** ou **multiple**

Si une **seule valeur** doit être sélectionnée, **ComboBox** préférable

Initialisation d'une liste

- Constructeur simple (avec modèle implicite)

```
String[] couleurs = {"rouge", "bleu", "gris", "vert",
                     "jaune", "noir","orange","blanc","rose" } ;
JList liste = new JList(couleurs) ;
```



- Définir une pré-sélection d'un élément
  - Les indexes démarrent à 0
  - Exemple: sélection de l'élément de rang 2

```
liste.setSelectedIndex(2);
```

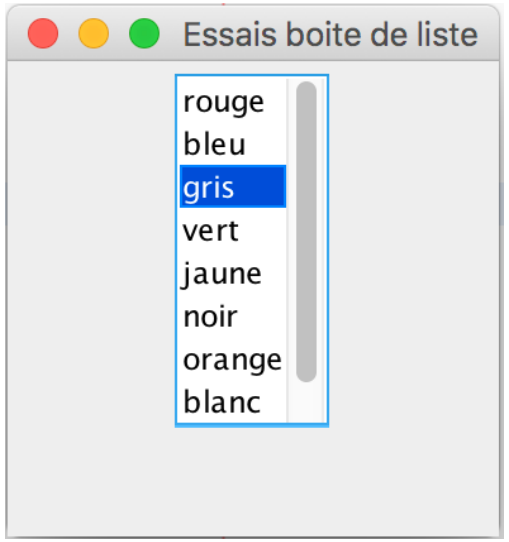
Affichage d'une liste

- Ajout d'une barre de défilement à une liste
  - Par défaut, la liste affichera **8 valeurs** avec une présentation verticale
  - La barre de défilement **n'apparaît pas** si la liste comporte moins de valeurs

```
JScrollPane jsp = new JScrollPane(liste);
getContentPane().add(jsp); // ajouter le jsp au content pane
```

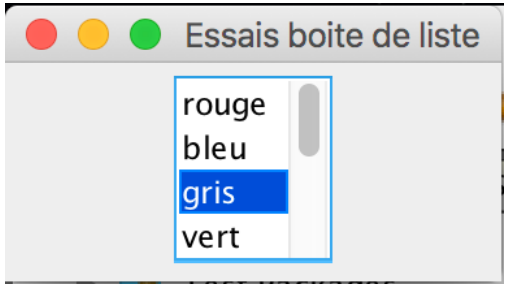
OU

```
JScrollPane jsp = new JScrollPane();
jsp.getViewport().setView(liste);
getContentPane().add(jsp); // ajouter le jsp au content pane
```



- Choisir le **nombre d'items** à afficher avec barre de défilement
  - Exemple: afficher seulement 4 valeurs à la fois

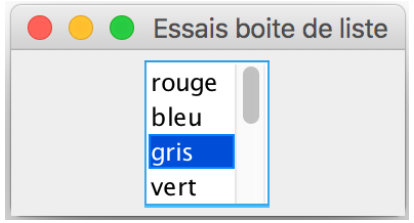
```
liste.setVisibleRowCount(4);
```



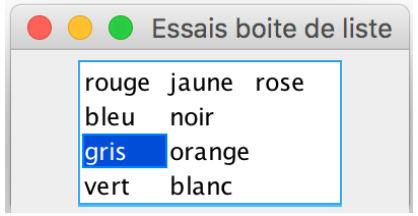
Mode d'affichage des items d'une liste

- Méthode: `liste.setLayoutOrientation(orientation);`
- 3 modes: VERTICAL VERTICAL_WRAP HORIZONTAL_WRAP

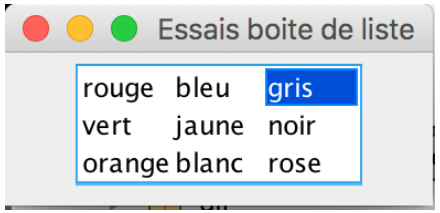
```
liste.setLayoutOrientation(JList.VERTICAL);
```
OU `liste.setLayoutOrientation(0);` (par défaut)




```
liste.setLayoutOrientation(JList.VERTICAL_WRAP);
```
OU `liste.setLayoutOrientation(1);`

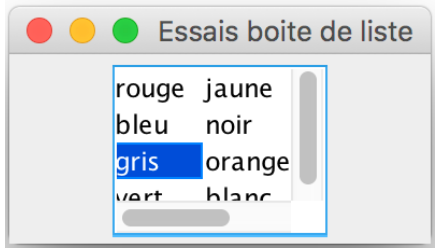



```
liste.setLayoutOrientation(JList.HORIZONTAL_WRAP);
```
OU `liste.setLayoutOrientation(2);`

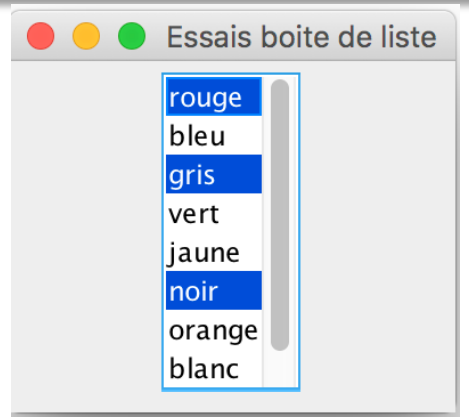


- Afficher avec une dimension et des barres de défilement

```
jsp.setPreferredSize(new Dimension(100, 80));
```



Modes de sélection des items d'une liste



- Méthode: `liste.setSelectionMode(mode);`
- 3 modes: SINGLE_SELECTION SINGLE_INTERVAL_SELECTION MULTIPLE_INTERVAL_SELECTION

```
liste.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
```
OU
```
liste.setSelectionMode(0);
```

```
liste.setSelectionMode(ListSelectionModel.SINGLE_INTERVAL_SELECTION);
```
OU
```
liste.setSelectionMode(1);
```

```
liste.setSelectionMode(ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);
```
OU
```
liste.setSelectionMode(2);
```
(par défaut)

Accès aux éléments du modèle

- Récupérer l'élément à la position i

Les indices commencent à 0

```
//récupérer le modèle de la liste
ListModel myModel=liste.getModel();
String premierelement=myModel.getElementAt(0).toString();
System.out.println("Le premier élément est: "+premierelement);
```

- Récupérer tous les éléments d'une liste par une boucle

```
ListModel myModel=liste.getModel();
int size = myModel.getSize();
for (int i = 0 ; i < size ; i++) {
  Object elem = myModel.getElementAt(i);
  System.out.println(elem);
}
```

Trouver par la **position** → modèle

Accès aux éléments de la liste

- Liste à sélection simple : récupérer l'élément sélectionné
  - Méthode: `Object valeur=liste.getSelectedValue();`

```
String ch = (String) liste.getSelectedValue();
System.out.println("Action Liste - La valeur sélectionnée: "+ch) ;
```

- Listes à sélection multiple : récupérer tous les éléments sélectionnés
  - Méthode: `List<type> valeurs = liste.getSelectedValuesList();`

```
System.out.println("Action Liste - Les valeurs selectionnees :");
List<String> valeurs = liste.getSelectedValuesList();
for (int i = 0; i<valeurs.size(); i++)
        System.out.println(valeurs.get(i)) ;
```

Trouver par la **sélection** → liste

Accès aux positions des items sélectionnés

- Liste à sélection simple : récupérer la position de la 1ère valeur sélectionnée par l'utilisateur
  - Méthode: **`public int getSelectedIndex();`**

```
int index = liste.getSelectedIndex();
System.out.println("Action Liste - Index de la valeur sélectionnée: "+index) ;
```

- Listes à sélection multiple : récupérer les positions de toutes les valeurs sélectionnées
  - Méthode: **`public int[] getSelectedIndices();`**

```
System.out.println("Action Liste - Les indexes des valeurs selectionnees :");
int[] indexes = liste.getSelectedIndices();
for (int i = 0; i<indexes.length; i++)
        System.out.println(indexes[i]) ;
```

Evénements générés

Les événements générés par une liste sont des **événements de sélection**

- de type : `ListSelectionEvent`
- (pas d'événement de type `ActionEvent`)

- Implémentation de l'interface: `ListSelectionListener`

- L'interface ne comporte qu'une seule méthode :

```
public void valueChanged(ListSelectionEvent e)
```

Méthodes de `ListSelectionEvent`

- `Object getSource()`: objet source de l'événement (héritée de *EventObject*)
- `int getFirstIndex()`: index du 1er item dont la valeur de sélection a changé
- `int getLastIndex()`: index du dernier item dont la valeur de sélection a changé
- boolean `getValueIsAdjusting()` →

Spécificité des événements générés par une JList

- `ListSelectionEvent` est généré :
  - Lors de l'appui sur le bouton de la souris
  - Lors du relâchement du bouton

- Les traitements **pourraient ainsi être générés 2 fois**
- Pour pallier cette redondance, il existe la méthode :

```
public boolean getValueIsAdjusting();
```

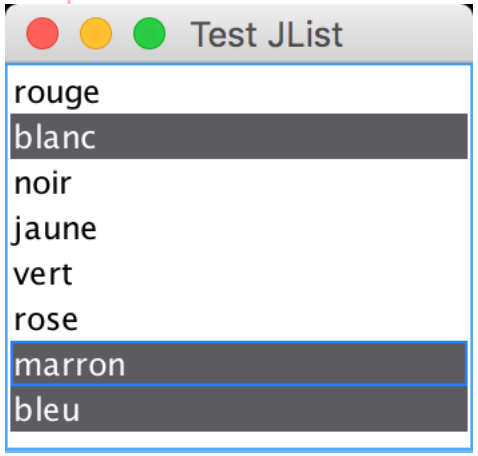
```
public void valueChanged (ListSelectionEvent e) {
    if ( !e.getValueIsAdjusting()) {
        // accès aux informations sélectionnées et traitement
    }
}
```

```
public class JListStatiqueTest extends JFrame implements ListSelectionListener {

    private JList liste ;
    static final String[] couleurs = {"rouge", "blanc", "noir", "jaune", "vert",
        "rose", "marron", "bleu"};


    public JListStatiqueTest(String t) {
        super (t);

        // definition d'une liste
        liste = new JList(couleurs);
        liste.setSelectionMode(ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);
        this.getContentPane().setLayout(new BorderLayout());

        liste.addListSelectionListener(this);
        JScrollPane panneau = new JScrollPane(liste);
        liste.setSelectedIndex(1);

        this.getContentPane().add(panneau, BorderLayout.CENTER);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
```

La fenêtre est son propre écouteur

Exemple de JList (statique) **avec gestion des événements**

Ajouter un écouteur à la liste



Exemple JList statique (suite)

```
@Override
public void valueChanged(ListSelectionEvent e) {

    if (e.getValueIsAdjusting())
        System.out.println("=> "+e.getSource());
    else {
        // affiche les items sélectionnés
    System.out.println("**** affichage des éléments sélectionnés :");
    List<String> valeurs;
        valeurs = liste.getSelectedValuesList();
    for (int i = 0; i < valeurs.size(); i++)
        System.out.println(valeurs.get(i));
    }


}
```

```
public static void main (String[] args) {

    JListTest f1 = new JListTest("Test JList");
    f1.setSize(200, 300);
    f1.setVisible(true);
}
```

```
run:
**** affichage des éléments sélectionnés :
blanc
=> javax.swing.JList[,
0,0,171x142,alignmentX=0.0,alignmentY=0.0,border=,flags=50332008,maximumSize=,minimumSize=
,preferredSize=,fixedCellHeight=-1,fixedCellWidth=-1,horizontalScrollIncrement=-1,selectio
nBackground=com.apple.laf.AquaImageFactory$SystemColorProxy[r=92,g=92,b=96],selectionForeg
round=com.apple.laf.AquaImageFactory$SystemColorProxy[r=255,g=255,b=255],visibleRowCount=8
,layoutOrientation=0]
**** affichage des éléments sélectionnés :
blanc
bleu
```

Listes dynamiques (modifiables)

- Quand on crée la liste en lui envoyant un vecteur d'objets, Java crée implicitement un **`DefaultListModel`** mais il est **non modifiable**
  - On ne peut ni **ajouter**, ni **supprimer** les items de la liste

- Quand on veut pouvoir modifier les items de la liste :
  - Il faut créer le modèle **explicitement** avec **`DefaultListModel`**

Listes dynamiques

- Création du modèle:

```
DefaultListModel monModele = new DefaultListModel();
```

- Création de la liste:

```
JList liste = new JList(monModele);
```

- Ajout d'un élément **à la fin** de la liste :

```
monModele.addElement(element);
```

- Ajout d'un élément **à la position i** :

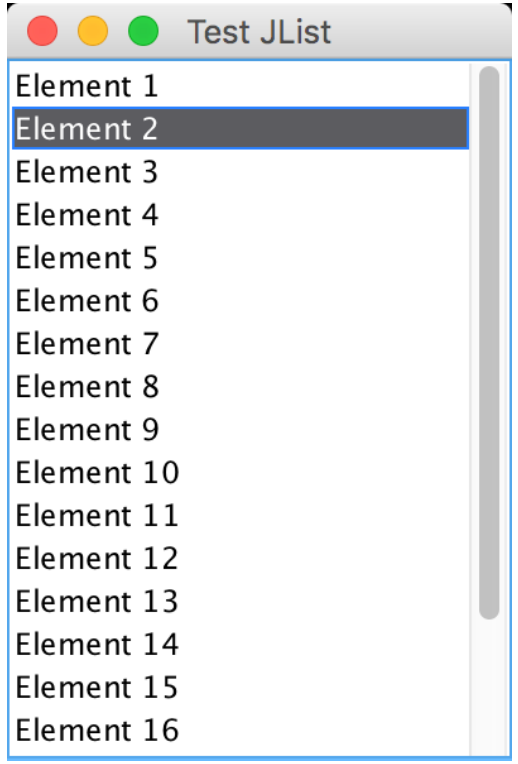
```
monModele.add(i,element);
```

- Supprimer un élément :

```
monModele.removeElement(element);
```

- Supprimer l'élément à la position i:

```
monModele.remove(i);
```




```
lm = new DefaultListModel();
// definition d'une liste
for (int i=0; i<20; i++) {
        lm.addElement("Element "+(i+1));
}
liste = new JList(lm);
```

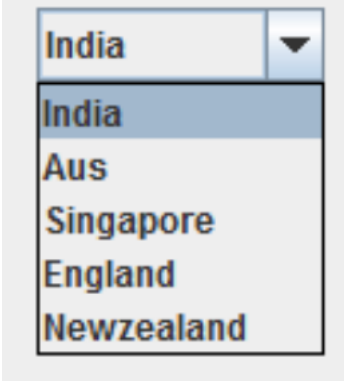
Exemple avec création de modèle

```
static JList liste;
static DefaultListModel monModele;
TestJListModel(){
    //Utilisation d'un modèle des données (par défaut)
    monModele = new DefaultListModel();
    //Construction de la liste
    monModele.addElement("rouge");
    monModele.addElement("gris");
    monModele.addElement("bleu");
    monModele.addElement("bleu");

    liste = new JList(monModele);

    Container contenu = getContentPane();
    contenu.setLayout(new FlowLayout());

    JScrollPane jsp = new JScrollPane(liste);
    contenu.add(jsp);
}
```

```
//Ajout d'un élément à une position donnée
monModele.add(1,"jaune");
//Ajout d'un élément à la fin de la liste
monModele.addElement("rose");
//Suppression d'un élément d'un index donné
monModele.remove(0);
//Supprimer l'élément sélectionné
int index = liste.getSelectedIndex();
monModele.remove(index);
//Suppression d'un item donné
//Supprimer la 1ère occurrence de « bleu »  (boolean)
//Retourne vrai si « bleu » était un item de la liste, faux sinon
monModele.removeElement("bleu");
//Pour supprimer toutes les occurrences :
boolean  suppr = false;
do
    suppr = monModele.removeElement("bleu");
while (suppr);
```

Pour aller plus loin…

ComboBox vs. List, modifier le bord, imposer une taille d'affichage



JComboBox

JList

Caractéristiques de la JList

- Avec une **JList**, l'utilisateur peut choisir entre une sélection unique ou **multiple** des éléments de la liste :
  - **C'est la principale différence avec la ComboBox (une seule sélection possible)**
- Une JList ayant moins de 8 éléments n'a nativement pas de barre de défilement. Il faut passer par un **JScrollPane** si on le souhaite ;

- La méthode **getSelectedIndex()** renvoie l'index du 1er élément sélectionné **ou -1 si aucun élément n'est sélectionné** et la méthode **getSelectedIndexes()** renvoie un tableau avec l'index de chaque élément sélectionné. Le tableau est vide si aucun élément n'est sélectionné ;
- La méthode **getSelectedValue()** renvoie le 1er élément sélectionné ou *null* si aucun élément n'est sélectionné ;

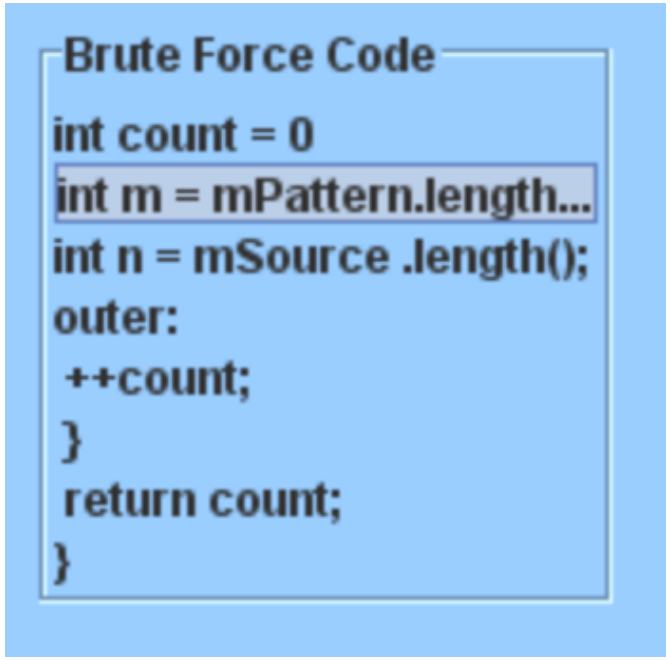
- Une classe **DefaultListModel** fournit une implémentation simple d'un modèle de liste, qui peut être utilisé pour gérer dynamiquement les éléments affichés par une JList.

Modifier le bord de la JList

Il est possible de modifier le bord d'une JList avec :

```
Border bo = BorderFactory.createEtchedBorder();
maList.setBorder(BorderFactory.createTitledBorder(bo, "Le
titre"));
```

Par exemple ici, on utilise une liste avec un bord, pour afficher des lignes de code *sélectionnable* :



Taille d'affichage de la JList

- Nativement, le composant **JList** parcourt tous les éléments pour choisir *la taille à afficher* de la liste
  - Ça peut être pénalisant en cas de longue liste…
- On peut **définir soi-même** la taille d'affichage de la liste avec la méthode `setPrototypeCellValue()` :

```
JList<String> bigDataList =
                        new JList<String>(bigData);
/* On donne ici la cellule qui a la taille la plus grande. La
MVJ l'utilise pour calculer la valeur des propriétés
fixedCellWidth et fixedCellHeight de la liste */

bigDataList.setPrototypeCellValue("Index 1234567890");
```

Autres Références

- **JList** tutorial
- http://fr.slideshare.net/martyhall/java-7-programming-tutorial-advanced-swing-and-mvc-custom-data-models-and-cell-renderers

22 slides, visible le 12 mai 2020