IUT Lyon 1
l'excellence technologique

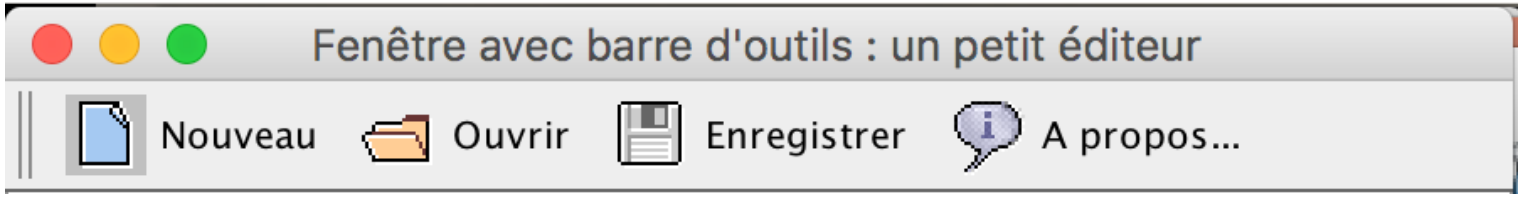
Programmation d'IHM- Cours 4
Barres d'outils
Architecture MVC, JList

V. DESLANDRES

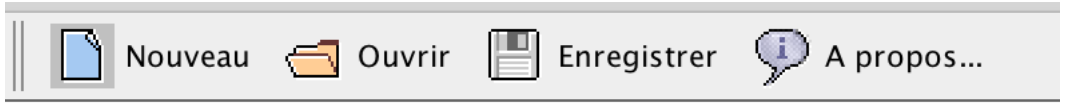
veronique.deslandres@univ-lyon1.fr

Sommaire

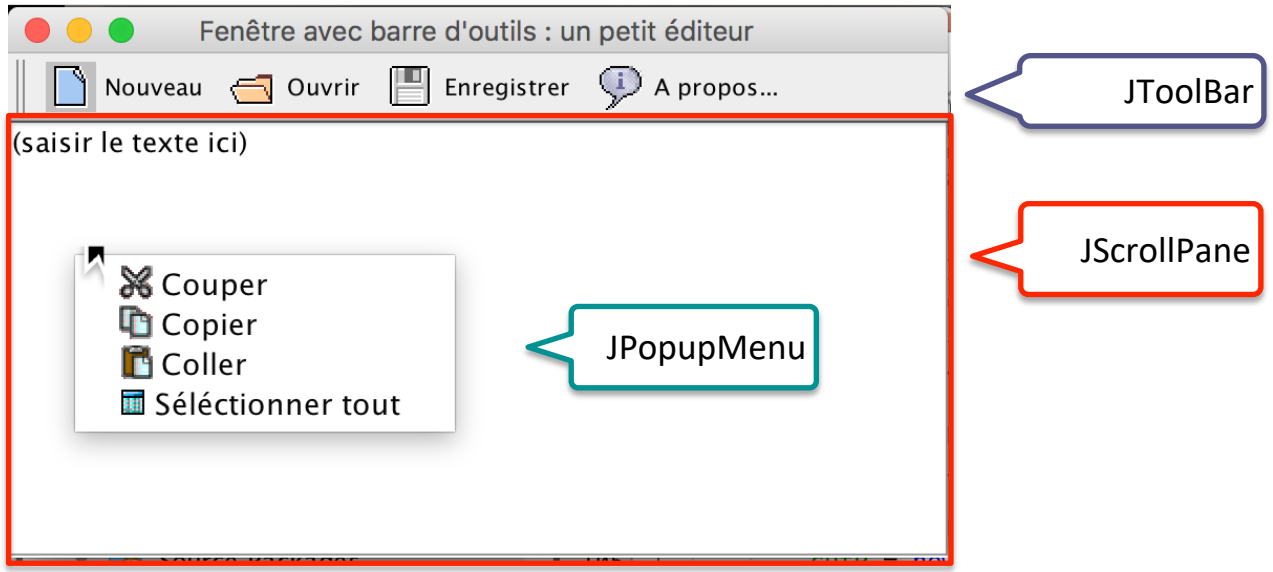
Les barres d'outils, menu déroulant et ScrollPane



Barre d'outils

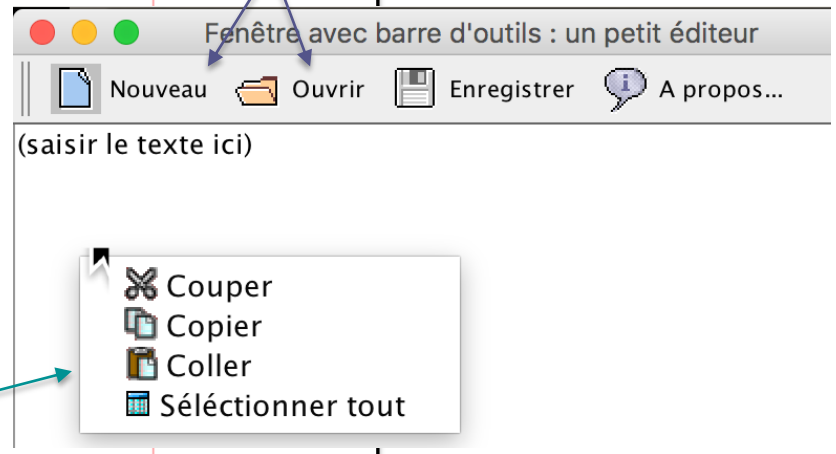


- Illustration sur un **petit éditeur de texte**
- La fenêtre est munie d'**ascenseurs** : `JScrollPane` est un conteneur disposant de barres de défilement, verticale et horizontale, uniquement quand c'est nécessaire
  - Ceci permet de visualiser des composants plus grands que l'espace dans lequel ils sont visualisés
- Un **menu déroulant** `JPopupMenu` apparaît quand on fait un clic droit



```java
import java.io.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class BarreOutil extends JFrame {
    private JTextArea zone_texte;
    private JPopupMenu popup;
    // les items du menu contextuel se terminent par la lettre "P"
    private JMenuItem cutP;
    private JMenuItem copyP;
    private JMenuItem pasteP;
    private JMenuItem allP;
    // les boutons requis pour la barre de menu
    private JButton nouv;
    private JButton ouvre;
    private JButton sauve;
    private JButton apropos;

    BarreOutil() {      // constructeur
        getContentPane().setLayout(new BorderLayout());

        zone_texte = new JTextArea("(saisir le texte ici)");
        popup = ajouterPopUp();
        zone_texte.setComponentPopupMenu(popup); // gère l'écoute automatique du popup
                                                 // pas besoin d'installer un écouteur associé
        JToolBar barreOutils = ajouter_barre_outils();
        JScrollPane scroll = new JScrollPane(zone_texte);

        getContentPane().add(barreOutils, BorderLayout.NORTH);
        getContentPane().add(scroll, BorderLayout.CENTER);
    } // fin du constructeur
```

(boutons sans bord)



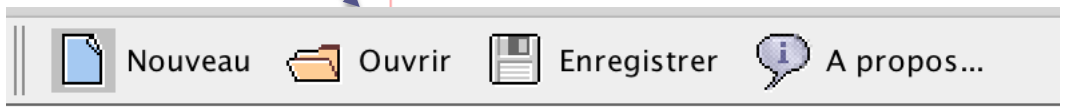
(méthode de JComponent)

Ajouter la zone texte au ScrollPane

Méthodes données ci-après pour définir la **fenêtre Popup** et la **barre d'outils** de l'éditeur

```java
//-----------------------------------------------------------------------------
// methode ajouter_barre_outils de construction de la barre d'outils
//-----------------------------------------------------------------------------
public JToolBar ajouter_barre_outils() {
    // définition d'une nouvelle barre
    JToolBar outil = new JToolBar();

    // définition des boutons avec les images :
    nouv = new JButton("Nouveau", new ImageIcon("src/gif/new.gif"));
    ouvre = new JButton("Ouvrir", new ImageIcon("src/gif/open.gif"));
    sauve = new JButton("Enregistrer", new ImageIcon("src/gif/save.gif"));
    apropos = new JButton("A propos...", new ImageIcon("src/gif/about.gif"));

    // propriétés des boutons de la barre d'outil : sans bordure
    nouv.setBorderPainted(false);
    ouvre.setBorderPainted(false);
    sauve.setBorderPainted(false);
    apropos.setBorderPainted(false);

    // enregistrement auprès de la classe écouteur pour la souris :
    nouv.addMouseListener(new PassageDeLaSouris());
    ouvre.addMouseListener(new PassageDeLaSouris());
    sauve.addMouseListener(new PassageDeLaSouris());
    apropos.addMouseListener(new PassageDeLaSouris());

    // enregistrement auprès de la classe écouteur pour les clics souris :
    nouv.addActionListener(new EcouteurMenu());
    ouvre.addActionListener(new EcouteurMenu());
    sauve.addActionListener(new EcouteurMenu());
    apropos.addActionListener(new EcouteurMenu());

    // on ajoute les boutons créés à la barre d'outils :
    outil.add(nouv);
    outil.add(ouvre);
    outil.add(sauve);
    outil.add(apropos);

    // on retourne la barre créée :
    return outil;
} // fin de la methode ajouter_barre_outils()
```

Créer les 4 boutons



Ajouter des écouteurs aux boutons

Ajouter les boutons à la barre d'outils

```java
//-------------------------------------------------------------
// creation d'un pop-up menu
//-------------------------------------------------------------
public JPopupMenu ajouterPopUp() {

    // on instancie le menu contextuel
    JPopupMenu pop = new JPopupMenu("Menu contextuel");

    // initialisation des items du menu contextuel
    cutP = new JMenuItem("Couper", new ImageIcon("cut.gif"));
    copyP = new JMenuItem("Copier", new ImageIcon("copy.gif"));
    pasteP = new JMenuItem("Coller", new ImageIcon("paste.gif"));
    allP = new JMenuItem("Séléctionner tout", new ImageIcon("datePicker.gif"));

    // enregistrement des items au près de l'écouteur approprié
    cutP.addActionListener(new EcouteurMenu());
    copyP.addActionListener(new EcouteurMenu());
    pasteP.addActionListener(new EcouteurMenu());
    allP.addActionListener(new EcouteurMenu());

    // ajout des items au menu contextuel
    pop.add(cutP);
    pop.add(copyP);
    pop.add(pasteP);
    pop.add(allP);

    // on retourne le nenu contextuel
    return pop;
}
```

Créer les menuItem

Associer les écouteurs aux items de menu

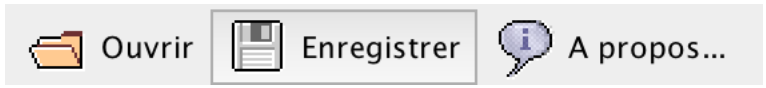
Ajouter les menuItem au menu Popup

```
//-----------------------------------------------------------------------
//   Les classes internes ecouteur
//-----------------------------------------------------------------------
class EcouteurMenu implements ActionListener {

    public void actionPerformed(ActionEvent e) {
        Object source = e.getSource();
        if (source == nouv) {
            zone_texte.setText(new String());  // on vide la zone de texte
            setTitle("Nouveau fichier"); // on modifie le titre de la fenêtre
        }
        if (source == ouvre) {
            String nomfic = ouvrir(); // ouvrir le fichier sélectionné
            if (nomfic != null) {
                zone_texte.setText(new String());
                setTitle(nomfic);
                afficher(nomfic); // afficher le contenu du fichier
            }
        }
        if (source == sauve) {
            String nomfic = sauver();
            ecrire(nomfic);
        }
        if (source == apropos) {
            aPropos();
        }

        if (source == cutP) {
            zone_texte.cut(); // methode de TextArea
        }
        if (source == copyP) {
            zone_texte.copy(); // idem
        }
        if (source == pasteP) {
            zone_texte.paste();
        }
        if (source == allP) {
            zone_texte.selectAll();
        }
    }
}
```

Traitement des actions sur les boutons de la barre d'outils

Traitement des actions sur les items de la fenêtre Popup

```
//-------------------------------------------------------------------------------------
// classe interne écouteur pour encadrer les boutons lorsque l  on passe dessus avec la souris
//-------------------------------------------------------------------------------------

class PassageDeLaSouris extends MouseAdapter {
    public void mouseEntered(MouseEvent e) { // lorsque la souris "entre" sur un des composants écoutés

        if (e.getSource() instanceof JButton) { // le MouseEvent peut provenir d'autres composants
            ((JButton) e.getSource()).setBorderPainted(true); // on fait apparaitre le bord du bouton
        }
    }

    public void mouseExited(MouseEvent e) { // lorsque la souris 'quitte' le composant
        if (e.getSource() instanceof JButton) {
            ((JButton) e.getSource()).setBorderPainted(false);
        }
    }
} // de classe interne PassageDeLaSouris
```



```
//-------------------------------------------------------------------------------------
//  Les méthode utilitaires
//-------------------------------------------------------------------------------------

public String ouvrir() {   // choix du fichier en lecture
    String nomFic = new String("");
    try {
        // chargement de fichier
        FileDialog fd = new FileDialog(this, "Sélectionnez votre fichier...", FileDialog.LOAD);
        fd.setVisible(true);
        nomFic = ((fd.getDirectory()).concat(fd.getFile()));
    } catch (NullPointerException e) {
        System.out.println("Erreur ouverture dossier !");
    }
    return nomFic;
} // fin de ouvrir()
//-------------------------------------------------------------------------------------
// méthode de chargement de la zone de texte depuis le fichier sélectionné en lecture
// lecture en flux caracteres

public void afficher(String nom) {
    try {
        FileReader fichier = new FileReader(nom);
        LineNumberReader lecteur = new LineNumberReader(fichier);
        String ligne = new String("");
        setTitle(nom);
        do {
            ligne = lecteur.readLine();
            zone_texte.append(ligne);
            zone_texte.append("\n\r");
        } while (ligne != null);
        fichier.close();
    } catch (FileNotFoundException e) {
    } catch (IOException e) {
    }
} // fin de afficher()
```

```
//-------------------------------------------------------------------------

public void ecrire(String nom) {    // Ecriture mode caractere dans le fichier
    try {
        FileWriter fic = new FileWriter(nom);
        BufferedWriter buff = new BufferedWriter(fic);
        buff.write(zone_texte.getText());
        buff.close();
        fic.close();
    } catch (IOException e) {
    }

} // de ecrire()
//-------------------------------------------------------------------------

public String sauver() {            // choix du fichier en ecriture
    String nomFic = new String("");
    try {
        FileDialog fd = new FileDialog(this, "Sélectionnez votre  fichier...", FileDialog.SAVE);
        fd.setVisible(true);
        nomFic = ((fd.getDirectory()).concat(fd.getFile()));
    } catch (NullPointerException e) {
    }

    return nomFic;
} // de sauver()
```

```
    //-----------------------------------------------------------------------
    public void aPropos() {  // Boite d'information
        JOptionPane.showMessageDialog(this, "VDe Corp. 2020", "A propos", JOptionPane.INFORMATION_MESSAGE);
    }
    //-----------------------------------------------------------------------

} // fin de la classe BarreOutil




class TestBarreOutil {

    public static void main(String[] args) {
        //Création de la fenêtre
        BarreOutil frame = new BarreOutil();

        //Affichage de la fenêtre
        frame.setVisible(true);
    }
}
```

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

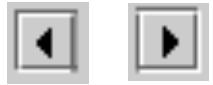
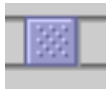
- Le modèle d'architecture MVC (Model View Controller) est à la base de nombreux systèmes de visualisation graphiques
- Principe de Base: **séparation des rôles**
  - Le ***modèle*** est l'élément principal du composant, il contient les **données**
  - Les ***vues*** du composant sont **des visualisations 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* ?

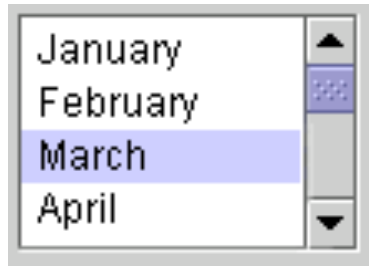
- *Modèle* :
  - valeur minimale = 0
  - valeur courante = 15
  - valeur maximale = 100

- *Vue* :

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

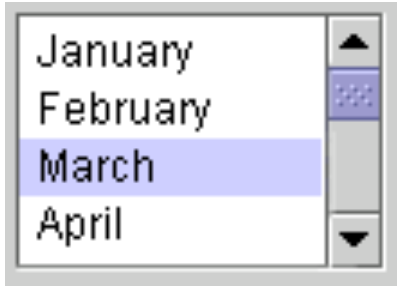
Modèles MVC des composants SWING

- 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

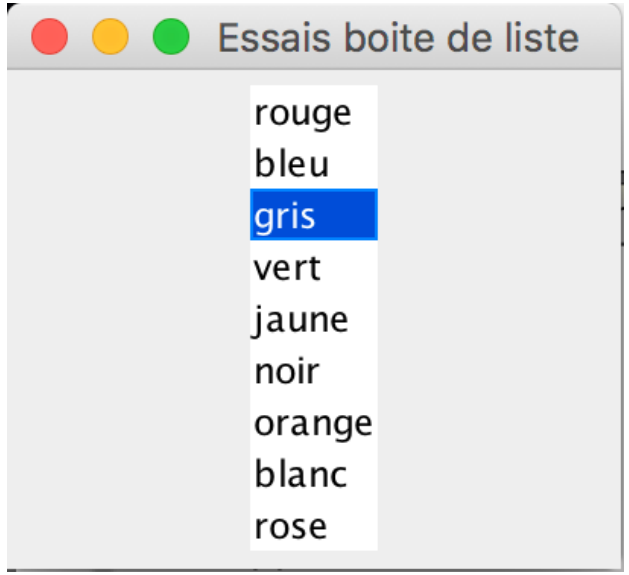
- Une **JList** est une présentation des données sous forme de liste
  - Affichage d'une liste d'items :
- 2 types de **JList**
  - Liste **statique** : sélectionner des éléments
  - Liste **dynamique** : la liste des items peut évoluer
- Modalité de sélection : **simple** ou **multiple**

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);
```

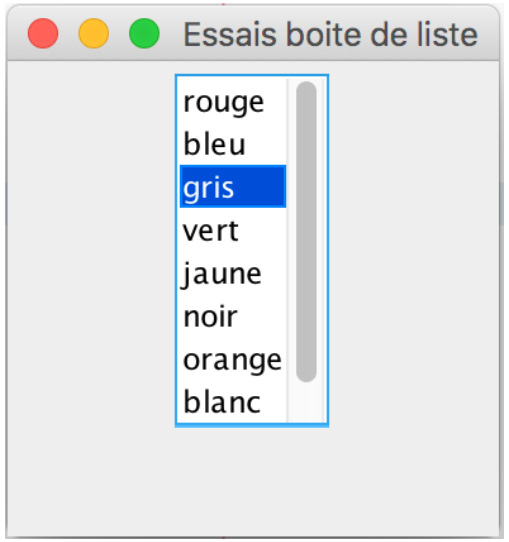
Affichage d'une liste

- 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
```

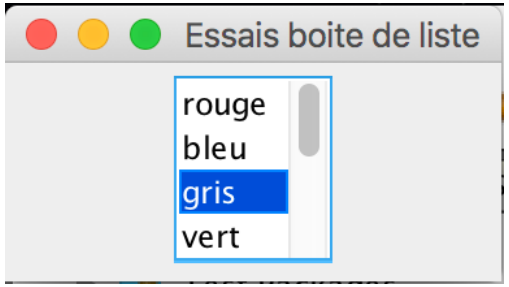
OU

```
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

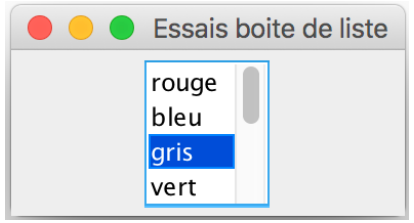
```
liste.setVisibleRowCount(4);
```



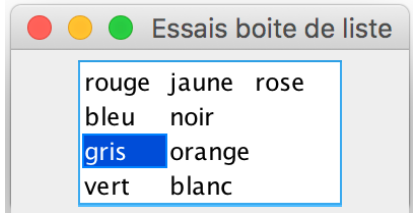
Mode d'affichage des items d'une liste

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

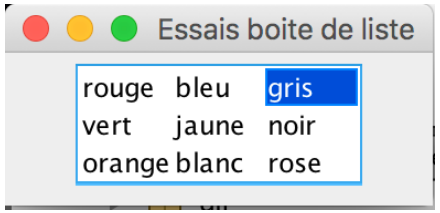
```
liste.setLayoutOrientation(JList.VERTICAL);
```
OU `liste.setLayoutOrientation(0);` (par défaut)




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

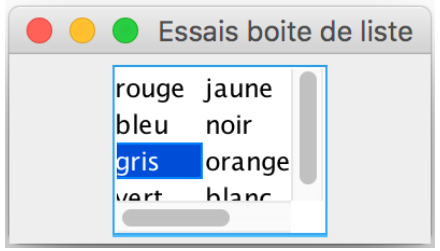



```
liste.setLayoutOrientation(JList.HORIZONTAL_WRAP);
```
OU `liste.setLayoutOrientation(2);`

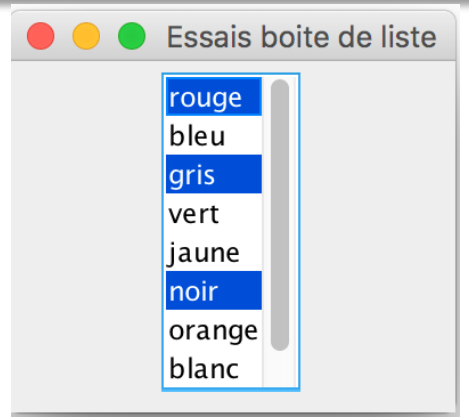


- Afficher avec une dimension et des barres de défilement

```
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

- (Une liste ne génère pas d'événement de type `ActionEvent`)
- Les événements générés par une liste sont des **événements de sélection**
  - de type : `ListSelectionEvent`
- 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();
```

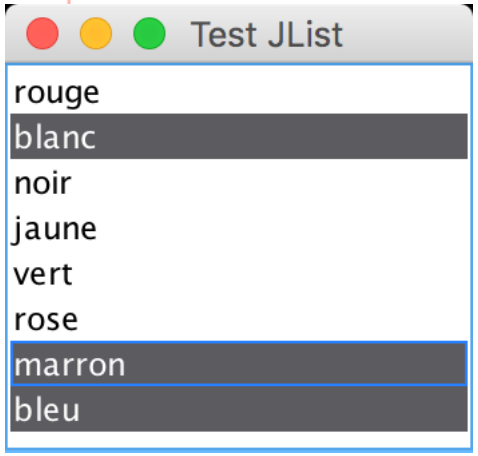
```
public void valueChanged (ListSelectionEvent e) {
    if ( !e.getValueIsAdjusting()) {
        // accès aux informations sélectionnées et traitement
    }
}
```

```
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** :

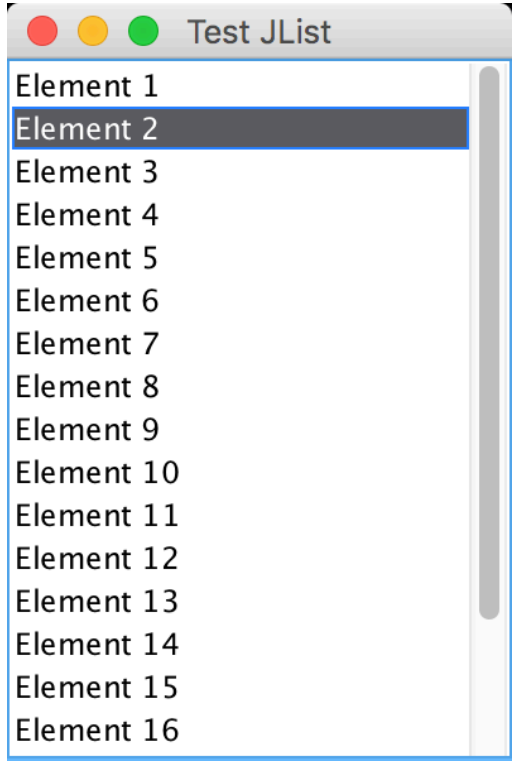
```
monModele.add(i,element);
```

- 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);
```

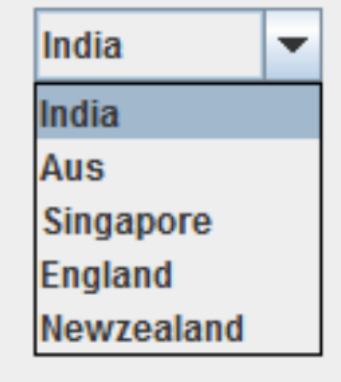
Exemple avec création de modèle

```
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 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 ;

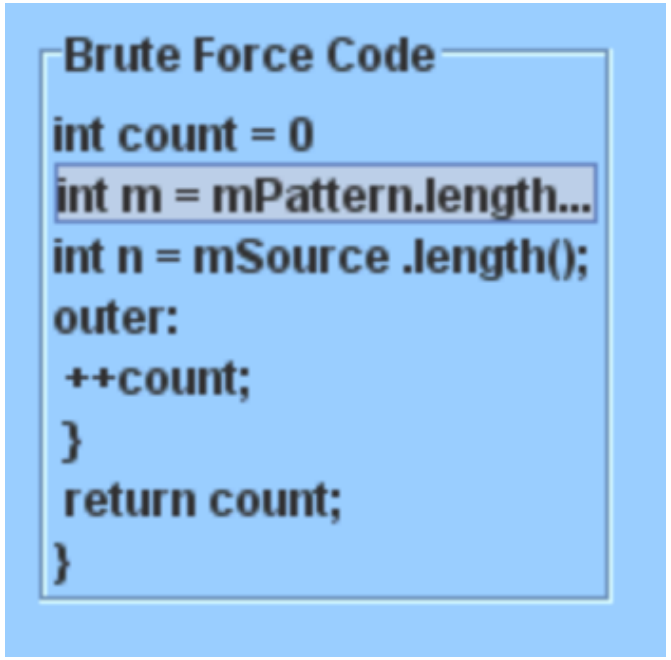
- Une classe **DefaultListModel** fournit une implémentation simple d'un modèle de liste, qui peut être utilisé pour gérer les éléments affichés par une JList.

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"));
```

Pour par exemple, afficher des lignes de code sélectionnable :



Taille d'affichage de la JList

- Nativement, le composant **JList** parcourt tous les élé-ments 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