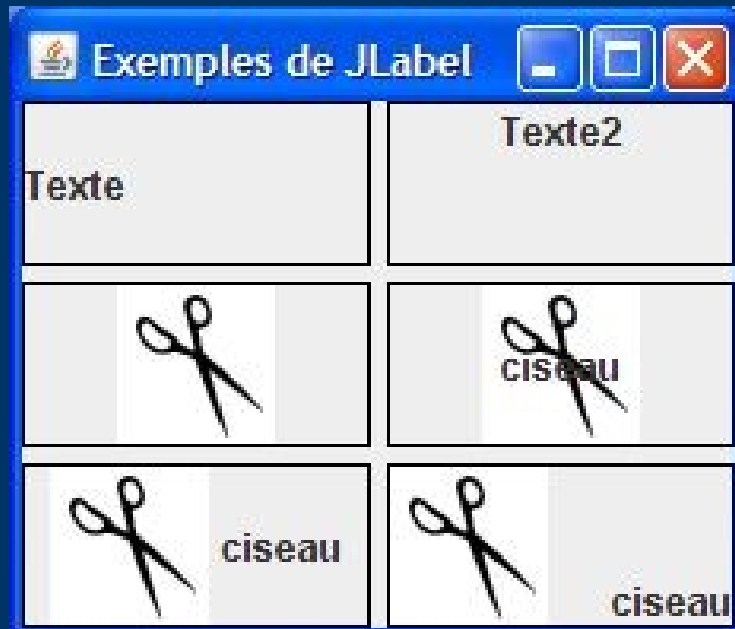


# *Détails de quelques composants*



# Les étiquettes (JLabel)



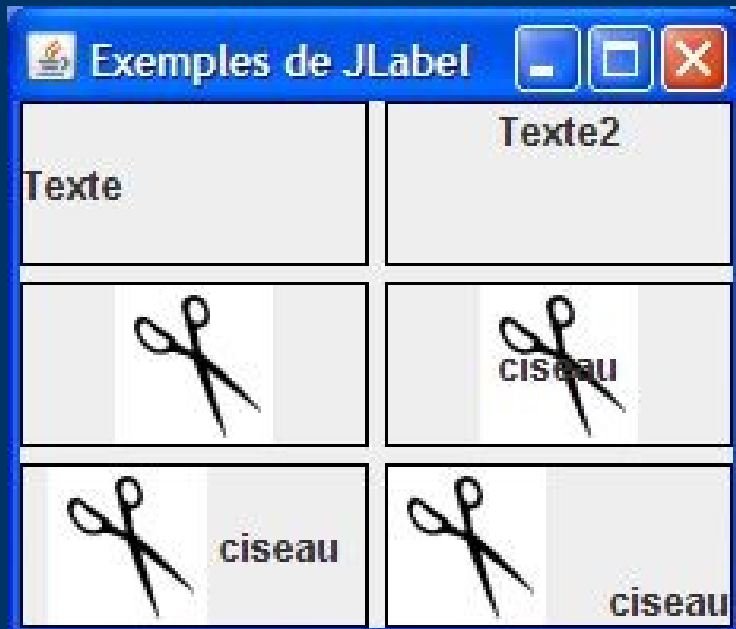
# Les étiquettes (*JLabel*)

- Rôle
    - Afficher un texte, une icône ou les deux
    - Interaction avec l'utilisateur nulle ou restreinte
    - Essentiellement informatif
  - Constructeurs
    - `JLabel()`
    - `JLabel(Icon image)`
    - `JLabel(String texte)`
    - `JLabel(Icon image, int alignementHoriz)`
    - `JLabel(String texte, int alignementHoriz)`
    - `JLabel(String texte, Icon image, int alignementHoriz)`
  - Alignement :
    - Position de l'icône, du texte, ou des deux
- 
-

# Positionnement des éléments

- Horizontalement
    - `public void setHorizontalAlignement(int h)`
      - LEFT, RIGHT
      - CENTER (défaut si icône seule)
      - LEADING (défaut si texte seul)
      - TREALING
  - Verticalement
    - `Public void setVerticalAlignement(int v)`
      - CENTER (défaut)
      - TOP, BOTTOM
  - Positionnement relatif du texte/icône
    - `setHorizontalTextPosition(int h)`
    - `SetVerticalTextPosition(int v)`
  - Espacement texte/icône
    - `setIconTextGap(int pixels)`
- 
-

# Positionnement des éléments



```
JLabel l1 = new JLabel("Texte");  
JLabel l2 = new JLabel(icone);  
JLabel l3 = new JLabel("ciseau", icone, JLabel.CENTER);
```

```
JLabel l4 = new JLabel("Texte2");  
l4.setHorizontalAlignment(JLabel.CENTER);  
l4.setVerticalAlignment(JLabel.TOP);
```

```
JLabel l5 = new JLabel("ciseau", icone, JLabel.CENTER);  
l5.setVerticalTextPosition(JLabel.CENTER);  
l5.setHorizontalTextPosition(JLabel.CENTER);
```

```
JLabel l6 = new JLabel("ciseau", icone, JLabel.CENTER);  
l6.setVerticalTextPosition(JLabel.BOTTOM);  
l6.setHorizontalTextPosition(JLabel.RIGHT);  
l6.setIconTextGap(20);
```

# Autres paramétrages

- void setDisabledIcon(Icon image)
  - void setDisplayedMnemonic(char car)
  - void setDisplayedMnemonic(int key)
  - void setText(String text)
    - Le texte d'un JLabel peut être en HTML
  - void setLabelFor(Component comp)
    - Le code mnémonique associé au label appellera comp.requestFocus()
  - Couleurs et polices
    - Méthodes setFont, setBackground, setForeground des super-classes
- 
-

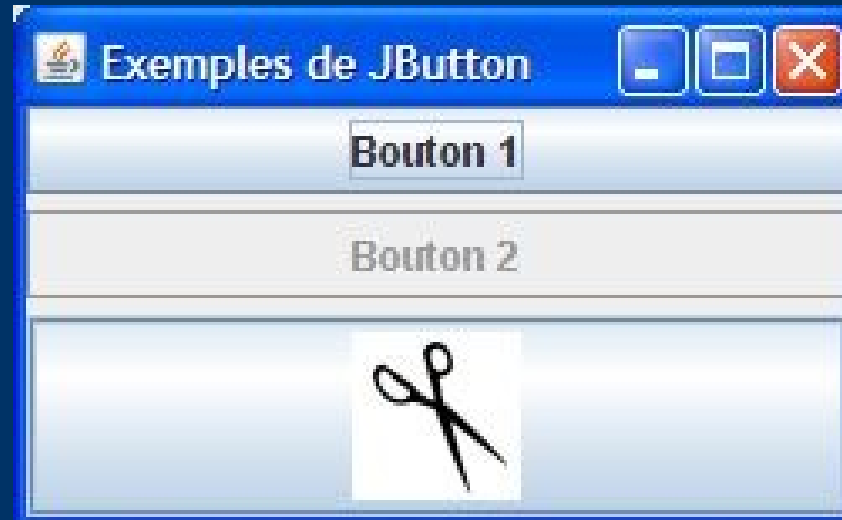
# Autres paramétrages : exemple



```
JLabel l1 = new JLabel("Texte");  
JLabel l2 = new JLabel("Texte2");  
  
l2.setForeground(Color.BLUE);  
l2.setBackground(Color.YELLOW);l2.setOpaque(true);  
l2.setFont(new Font("SansSerif",Font.PLAIN,16));
```

# Les boutons

## Vue + Contrôleur(JButton)





# Les boutons

## Vue + Contrôleur(JButton)

- Rôle
  - Afficher un texte, une icône ou les deux
  - Le clic sur le bouton doit entraîner l'exécution d'une action
- Constructeurs
  - JButton()
  - JButton(String texte)
  - JButton(Icon image)
  - JButton(String texte, Icon image)
  - JButton(Action a)

Voir partie sur les actions

---

---

# *Les boutons modèle (ButtonModel)*

- Interface de javax.swing
  - Contient plusieurs propriétés représentant les états suivants :
    - Armé (boolean isArmed())
    - Pressé (boolean isPressed())
    - Sélectionné (boolean isSelected())
    - SousLaSouris (boolean isRollover())
    - Activé (boolean isEnabled())
  - Peut être membre d'un groupe de boutons
  - Peut avoir un code mnémonique
  - Peut gérer des écouteurs d>ActionEvent, d'ItemEvent et de ChangeEvent
- 
-

# *Les boutons*

## *Etats d'un bouton*

- Désactivé
    - Bouton grisé, aucun clic possible
  - Pressed
    - On a cliqué sur le bouton, mais pas encore relâché le bouton de la souris
  - Armed
    - On a cliqué sur le bouton, pas relâché le bouton de la souris, et le pointeur de la souris est sur le bouton
  - Rollover
    - La souris est sur le bouton
- 
-

# *Les boutons*

## *Mise en forme*

- Méthodes essentiellement définies dans `AbstractButton`, dont hérite `JButton`
  - Mêmes principes que pour les `JLabel` :
    - Alignements horizontal et vertical
    - Positionnement relatif texte/icône
  - Définition d'icônes différentes et du texte
    - `setDisabledIcon(Icon image)`, `setDisabledSelectedIcon(Icon image)`
    - `setIcon(Icon image)`, `setText(String texte)`
    - `setPressedIcon(Icon image)`
    - `setRolloverIcon(Icon image)`, `setRolloverSelectedIcon(Icon image)`
    - `setSelectedIcon(Icon image)`
  - Autres propriétés
    - `setMargin(Insets marges)`
    - `setBorderPainted(boolean b)`
- 
-

# Les boutons

## Exemples



```
ImageIcon icone = new ImageIcon("c:\\Documents and Settings\\Bruno\\Bureau\\ciseaux.gif");
```

```
JButton l1 = new JButton("Bouton 1");
```

```
JButton l2 = new JButton("Bouton 2");
```

```
JButton l3 = new JButton(icone);
```

```
l2.setEnabled(false);
```

# *Les boutons*

## *bouton par défaut*

- Principe

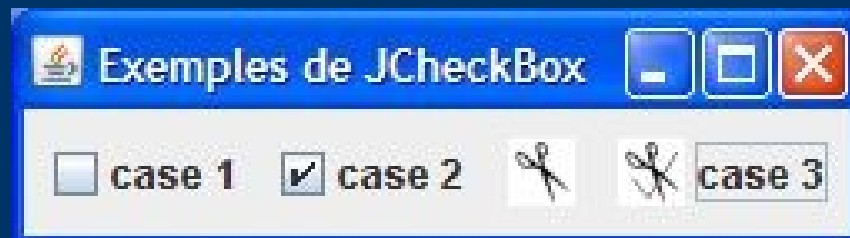
Bouton pré-sélectionné d'un conteneur C et qui générera un `ActionPerformed` lorsque C est activé et que l'on appuie sur Entrée

- Mise en œuvre

Appeler `setDefaultButton(JButton default)` sur le conteneur contenant le bouton



# Les cases à cocher



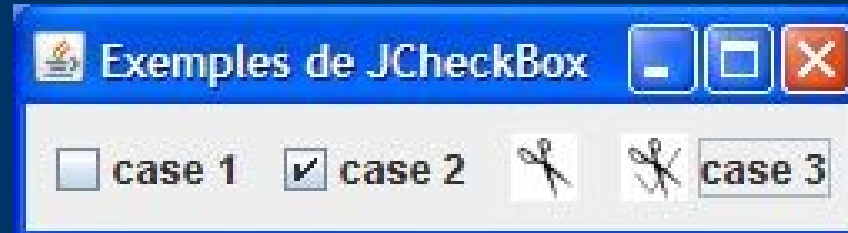
# Les cases à cocher

- Modèle : ButtonModel
  - Vue+Contrôleur : JCheckBox
  - Événement généré
    - itemStateChanged(ItemEvent ie)
    - (actionPerformed(ActionEvent ae))
  - Constructeurs
    - Les mêmes que pour un JButton
    - JCheckBox(Icon icone, boolean selected)
    - JCheckBox(String texte, boolean selected)
    - JCheckBox(String texte, Icon icone, boolean selected)
- 
-



# Les cases à cocher

## Exemples



```
ImageIcon iconeG = new ImageIcon("c:\\Documents and Settings\\Bruno\\Bureau\\ciseaux.gif");  
ImageIcon iconeSelectionneeG = new ImageIcon("c:\\Documents and  
Settings\\Bruno\\Bureau\\ciseauxS.gif");
```

```
ImageIcon icone = new ImageIcon(iconeG.getImage().getScaledInstance(20, 20,  
Image.SCALE_SMOOTH));  
ImageIcon iconeSelectionnee = new ImageIcon(iconeSelectionneeG.  
getImage().getScaledInstance(20, 20, Image.SCALE_SMOOTH));
```

```
JCheckBox l1 = new JCheckBox("case 1");  
JCheckBox l2 = new JCheckBox("case 2", true);  
JCheckBox l3 = new JCheckBox(icone);  
l3.setSelectedIcon(iconeSelectionnee);  
JCheckBox l4 = new JCheckBox("case 3", icone, true);  
l4.setSelectedIcon(iconeSelectionnee);
```

# Les boutons radio



# Les boutons radio

## Généralités

- Principe
    - Plusieurs boutons radios regroupés dans un groupe
    - Un seul bouton d'un groupe peut être sélectionné
  - Modèle
    - ButtonModel
  - Vue+Contrôleur
    - JRadioButton
    - Constructeurs : idem JCheckBox
  - Mise en oeuvre
    - Créer les différents boutons radio d'un groupe
    - Créer le groupe
    - Ajouter les boutons au groupe
    - La sélection d'un des boutons d'un groupe entraîne automatiquement la dé-sélection des autres boutons du groupe
- 
-

# Les boutons radio

## Classe `javax.swing.ButtonGroup`

- But
    - Dé-sélectionner automatiquement tous les autres boutons du groupe lorsque l'un d'eux est sélectionné
  - Constructeur
    - `ButtonGroup()`
  - Ajouter/Supprimer un bouton
    - `add(AbstractButton) / remove(AbstractButton)`
  - Gérer la sélection
    - `void clearSelection()`
    - `void setSelected(ButtonModel m, boolean b)`
  - Informations sur la sélection
    - `ButtonModel getSelection()`
    - `Boolean isSelected(ButtonModel m)`
  - Informations sur le groupe
    - `int getButtonCount()`
    - `Enumeration<AbstractButton> getElements()`
- 
-

# *Les boutons radio*

## *Événements générés*

- Types
    - `actionPerformed(ActionEvent ae)`
    - `itemStateChanged(ItemEvent ie)`
  - Nombres
    - 1 `actionPerformed` par clic
    - 1 ou 2 `itemStateChanged` (1 pour le bouton sélectionné, 1 pour l'éventuel bouton désélectionné)
- 
-

# *Les boutons radio*

## *Conseils de présentation*

- ButtonGroup :
  - Structuration logique mais pas de structuration de présentation
  - Pas de sélection initiale
- Introduire la logique de groupe dans la présentation
  - Créer un JPanel pour les boutons
  - Mettre un cadre avec titre au JPanel
- Forcer ou non une sélection initiale

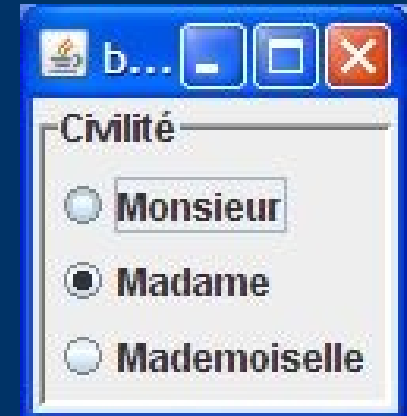


# Les boutons radio

## Exemple

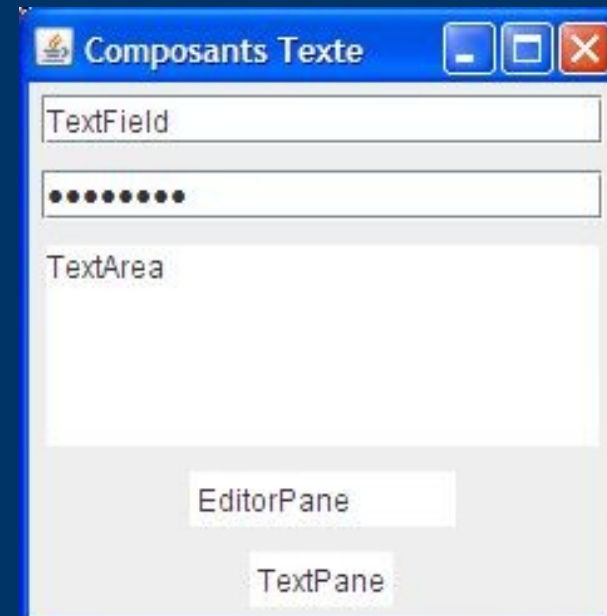
```
public class TestBoutonsRadio extends JFrame {
    public TestBoutonsRadio() {
        super("boutons radio"); Container cont = getContentPane();
        ButtonGroup group = new ButtonGroup();
        JPanel civilete = new JPanel();
        civilete.setBorder(new TitledBorder(new BevelBorder(BevelBorder.LOWERED) ,
        "Civilité"));
        civilete.setLayout(new GridLayout(3,1));
        JRadioButton monsieur = new JRadioButton("Monsieur");
        group.add(monsieur); civilete.add(monsieur);
        JRadioButton madame = new JRadioButton("Madame");
        group.add(madame); civilete.add(madame);
        JRadioButton mademoiselle = new JRadioButton("Mademoiselle");
        group.add(mademoiselle); civilete.add(mademoiselle);
        madame.setSelected(true);
        cont.add(civilete, BorderLayout.CENTER);
        pack();
        setVisible(true);
    }
}
```

```
public static void main(String[] args) {
    JFrame fenetre = new TestBoutonsRadio();
    fenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}}
```



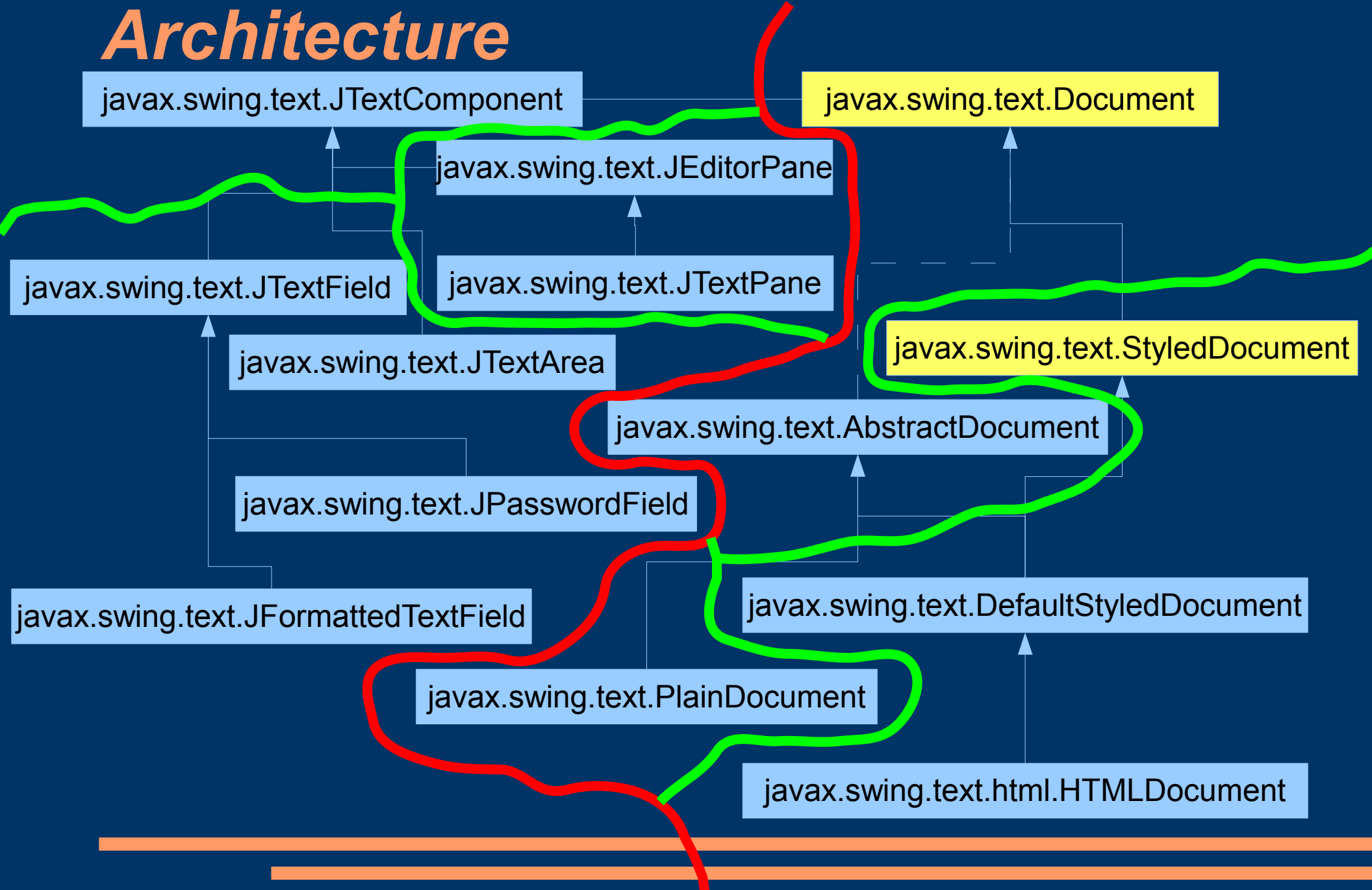
# Les composants Texte (texte éditable)

- JTextField  
Une ligne de texte sans mise en forme
- JPasswordField  
Une ligne de texte avec echo unique pour la saisie des mdp
- JFormattedTextField  
Une ligne de texte avec contrôle du format
- JTextArea  
Plusieurs lignes de texte sans mise en forme
- JEditorPane  
Plusieurs lignes de texte mis en forme
- JtextPane  
Texte mis en forme avec styles et structure de paragraphes





# Les composants Texte Architecture



# Les composants Texte

## *JTextComponent* : quelques méthodes

- **Void copy()/void cut()/void paste()**
    - Transfert de la sélection vers le presse-papiers système et inversement
  - **Caret getCaret() / int getCaretPosition() / String getSelectedText() / int getSelectionEnd() / int getSelectionStart()**
    - Infos sur le curseur/la sélection (c.f. Transparent Caret)
  - **setCaret(Caret c) / setCaretColor(Color c) / setCaretPosition(int i)**
    - Contrôle du curseur
  - **setMargin(Insets marges)**
    - Marges entre le texte et le cadre
  - **Void setNavigationFilter(NavigationFilter filtre)**
    - Pour contrôler le déplacement du curseur
  - **Void setEditable(boolean b)**
    - Texte éditable ou non
  - **String getText() / void setText(String s)**
    - Texte du composant
  - **Void replaceSelection(String s)**
    - Remplacement de la sélection par un autre texte
- 
-

# Les composants texte

## L'interface Caret

- Rôle : gère le curseur et la sélection
    - Deux positions :
      - dot
      - mark
    - Si pas de sélection : dot=mark
    - Si sélection :
      - mark = premier point marqué de la sélection,
      - dot = point mobile de la sélection
  - Quelques méthodes
    - int getDot() / int getMark()
    - boolean isVisible()/isSelectionVisible()
    - void setDot(int pos)  
Positionne dot et mark à pos
    - void moveDot(int pos)  
Positionne dot à pos sans changer mark
- 
-

# *Les composants Texte*

## *JTextField : Constructeurs*

- JTextField()
- JTextField(int largeur)
  - Largeur en caractères, utilisée pour calculer la preferredSize
- JTextField(String txtInitial)
- JTextField(String txtInitial, int largeur)
- JTextField(Document doc, String txtInitial, int largeur)
  - Permet de forcer le modèle utilisé. Si à null, un modèle par défaut sera créé



# *Les composants Texte*

## *JTextField : quelques méthodes*

- void scrollRectToVisible(Rectangle rec)
    - Pour afficher une partie donnée
  - void setFont(Font f)
    - Pour définir la police
  - void setHorizontalAlignment(int alignement)
    - Définir l'alignement du texte
  - void setScrollOffset(int nbPixels)
    - Définir le décalage lors d'un défilement
- 
-

# Les composants texte

## *JTextField* : gestion de <<Entrée>>

- Si des `actionListeners` sont enregistrés
  - Génération d'un `actionPerformed(ActionEvent ae)`
  - Les écouteurs sont prévenus
  - L'appui sur <<Entrée>> est consommé
- Sinon
  - L'appui sur <<Entrée>> n'est pas consommé
  - Événement passé au composant Parent
    - Bouton par défaut peut être validé

# Les composants texte

## *JTextField* : réaction "en direct" à la saisie

Deux types d'écouteurs

- CaretListener
  - 1 méthode : `caretUpdate(CaretEvent)`
  - Méthodes de `CaretEvent` : `int getDot()` et `int getMark()`
- DocumentListener
  - 3 méthodes :
    - `changedUpdate(DocumentEvent e)`
      - Changement d'un attribut/ensemble d'attributs du document
    - `insertUpdate(DocumentEvent e)`
      - Insertion dans le document
    - `removeUpdate(DocumentEvent e)`
      - Suppression dans le document

# *Les composants texte*

## *JTextArea : Constructeurs*

- `JTextArea()`
  - `JTextArea(int lignes, int colonnes)`
  - `JTextArea(String texte)`
  - `JTextArea(String texte, int lignes, int colonnes)`
  - `JTextArea(Document doc, String texte, int lignes, int colonnes)`
  - `JTextArea(Document doc)`
- 
-



# *Les composants texte*

## *JTextArea : retour à la ligne*

- void setLineWrap(boolean b)
  - Si vrai, retour automatique
  - Si faux, retour uniquement sur <<Entrée>>
- Void setWrapStyleWord(boolean b)
  - Actif uniquement si LineWrap à vrai
  - Action
    - Si vrai, retour entre les mots
    - Si faux, retour n'importe où



# *Les composants texte*

## *JTextArea : défilement*

- Ne gère pas directement le défilement
- Implante Scrollable
  - Si défilement souhaité, incorporer le JTextArea dans un JScrollPane (voir transparent)



# *Les composants texte*

## *JTextArea : gestion du contenu*

- Void append(String s)
  - Int getColumns() / int getRows()
  - Int getLineCount()
  - Void insert(String, int pos)
  - Void replaceRange(String, int start, int end)
  - Int getTabSize() / void setTabSize(int taille)
    - Espaces à générer par <TAB>
- 
-

# Les composants texte

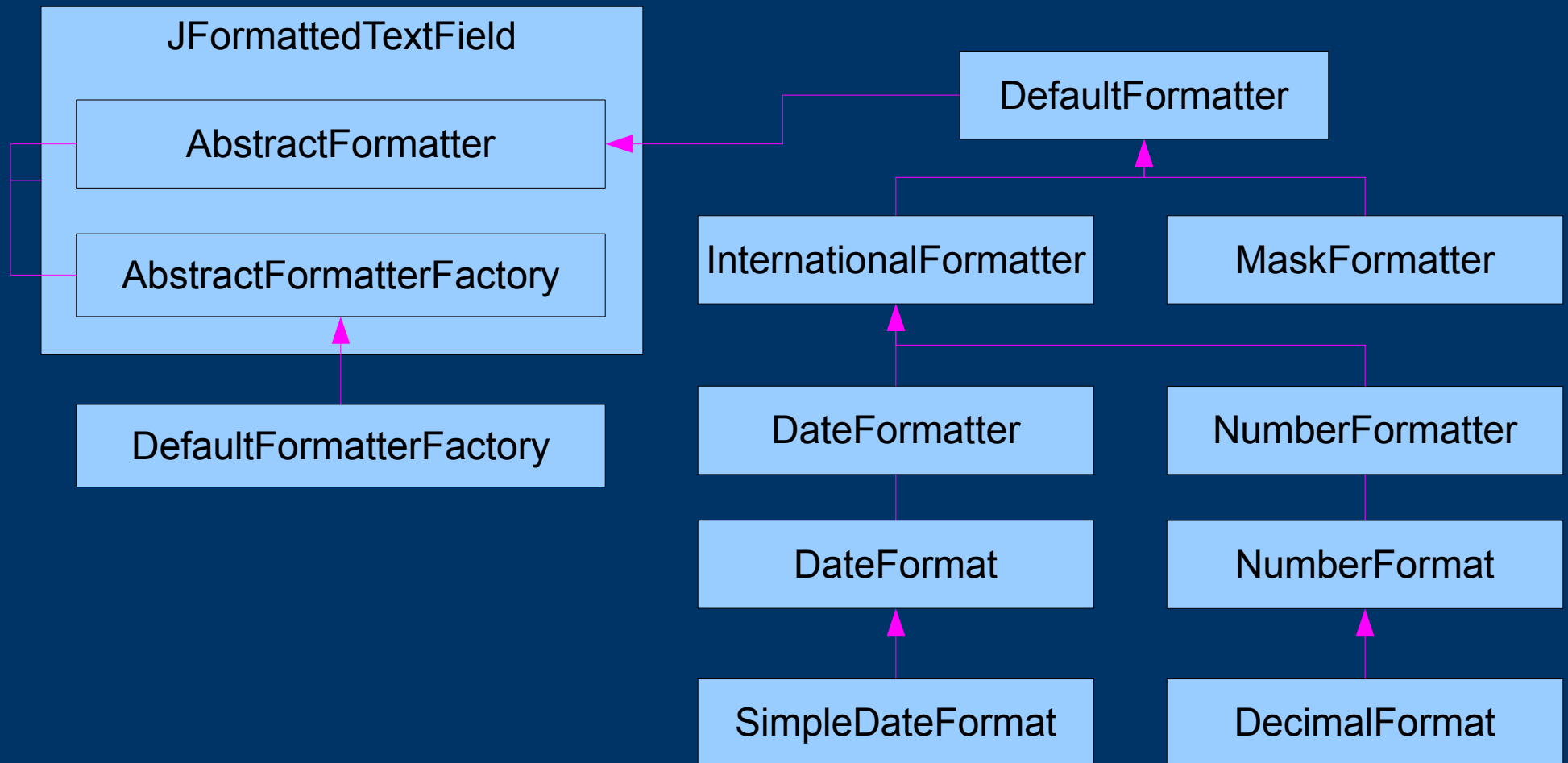
## *JPasswordField*

- Rôle
    - Saisir un mot de passe avec un caractère d'écho unique cachant ainsi la saisie
  - Structure
    - Sous-classe de JTextField
  - Constructeurs
    - JPasswordField(), JPasswordField(int columns)
    - JPasswordField(String text, int columns)
    - JPasswordField(Document doc, String txt, int col)
  - Quelques méthodes
    - char[] getPassword()
    - void setEchoChar(char c)
- 
-

# Les composants texte

## JFormattedTextField

Permet de spécifier des champs texte avec validation de la saisie. Plusieurs types de formateurs prédéfinis existent



# Les composants texte

## *JFormattedTextField*

### Comportement sur perte de focus

#### – Types possibles

- Retour à la valeur précédente
- Prise en compte de la valeur actuelle, même si non conforme au format
- Prise en compte de la valeur actuelle si conforme, sinon, retour à la valeur précédente (comportement par défaut)
- Ne rien faire (valeur affichée reste affichée, mais valeur du composant pas mise à jour)

#### – Valeurs possibles

REVERT, COMMIT, COMMIT\_OR\_REVERT, PERSIST

#### – Spécification

SetFocusLostBehaviour (int type)

---

---

# *JFormattedTextField*

## *DateFormatter*

- Rôle
  - N'autoriser que des saisies de dates
- Spécification du format de la date
  - Constructeur : `DateFormatter(DateFormat df)`
  - Méthode : `setFormat(DateFormat df)`
- Objets `DateFormat` :
  - Soit `static DateFormat(int type)`
  - Soit `new SimpleDateFormat(String pattern)`



# JFormattedTextField

## DateFormatter : exemple

```
public class TestFormattedTextField extends JFrame {
    private JFormattedTextField champDate; private JLabel valide;

    public TestFormattedTextField() {
        super("Test champs formats"); JPanel panneau = new JPanel(); panneau.setLayout(new BorderLayout());
        setContentPane(panneau);
        champDate = new JFormattedTextField(new DateFormatter(DateFormat.getDateInstance(DateFormat.SHORT)));
        champDate.setColumns(12);
        champDate.setFocusLostBehavior(JFormattedTextField.COMMIT);
        panneau.add(champDate, BorderLayout.NORTH);
        JButton test = new JButton(new Valideur()); panneau.add(test, BorderLayout.CENTER);
        valide = new JLabel("A tester"); panneau.add(valide, BorderLayout.SOUTH); pack(); setVisible(true);
    }
    class Valideur extends AbstractAction {
        public Valideur() { super("test");}
        public void actionPerformed(ActionEvent e) {
            AbstractFormatter formatteur = champDate.getFormatter();
            try {Object date = formatteur.stringToValue(champDate.getText()); valide.setText("valide");}
            catch (ParseException pe) {valide.setText("invalide");}
        }
    }
    public static void main(String[] args) { JFrame fenetre = new TestFormattedTextField();
        fenetre.setDefaultCloseOperation(EXIT_ON_CLOSE);}
}
```

Ou : champDate = new JFormattedTextField(new DateFormatter(new SimpleDateFormat("dd/MM/yyyy")));

---

---



# *JFormattedTextField* *NumberFormatter*

- Rôle
  - Valider la saisie de nombre
- Spécification du format des nombres
  - Utilisation de la classe `DecimalFormat`
- Exemples
  - `New DecimalFormat("0.###");`
  - `New DecimalFormat("0,000");`
  - `New DecimalFormat("0.00;(0.00)");`

0 : chiffre obligatoire pour l'affichage  
# : chiffre facultatif pour l'affichage

Séparateur des milliers local (nb de car entre "," et virgule (ou fin) spécifie les chiffres séparés

Représentation des nombres négatifs après le "-"

Séparateur décimal local



# *JFormattedTextField*

## *MaskFormatter*

- Rôle

Formatage général d'une chaîne de caractères

- Constructeur

MaskFormatter(String format)

- Exemple

```
try {  
    champFiltre = new JFormattedTextField(new MaskFormatter("0#-##-##-##-##"));  
} catch (ParseException ex) {
```

```
    Logger.getLogger(TestFormattedTextFieldMask.class.getName()).log(Level.SEVERE,  
    null, ex);
```

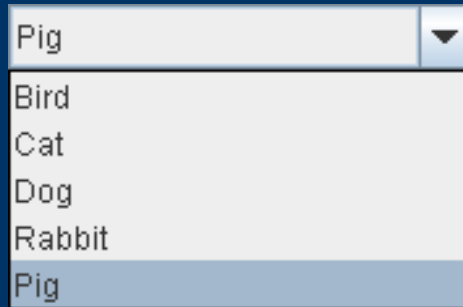
```
    }  
    champFiltre.setColumns(12);  
    champFiltre.setFocusLostBehavior(JFormattedTextField.COMMIT);  
    champFiltre.setText("00-00-00-00-00");
```

---

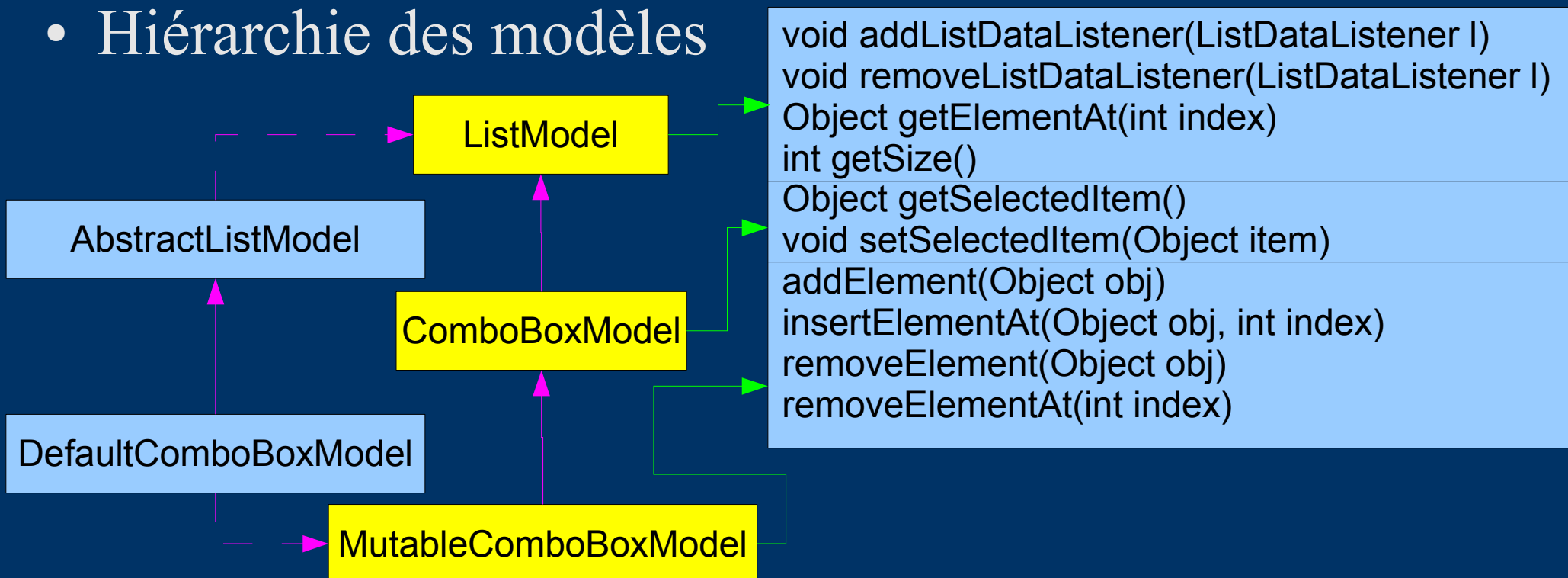
---

# JComboBox

- Aperçu :



- Hiérarchie des modèles



# JComboBox

- Remarques sur la structure
  - La liste d'une JComboBox est a priori "mutable"
  - Un seul élément peut être sélectionné
  - L'élément sélectionné n'est pas forcément membre de la liste
- Constructeurs
  - JComboBox(ComboBoxModel modèle)
  - JComboBox(Object[] éléments)
  - JComboBox(Vector<?> éléments)

# *JcomboBox*

## *quelques méthodes essentielles*

- void addItem(Object item)  
Ajoute un item à la fin
  - void insertItemAt(object obj, int index)  
Insère un article dans la liste
  - void setMaximumRowCount(int nb)  
Spécifie le nombre de lignes affichées
  - void setEditable(boolean b)  
Spécifier si le champ de saisie est éditable
  - void setSelectedIndex(int index),  
void setSelectedItem(Object obj)  
Spécifient l'item sélectionné
- 
-

# *JcomboBox*

## *quelques méthodes avancées*

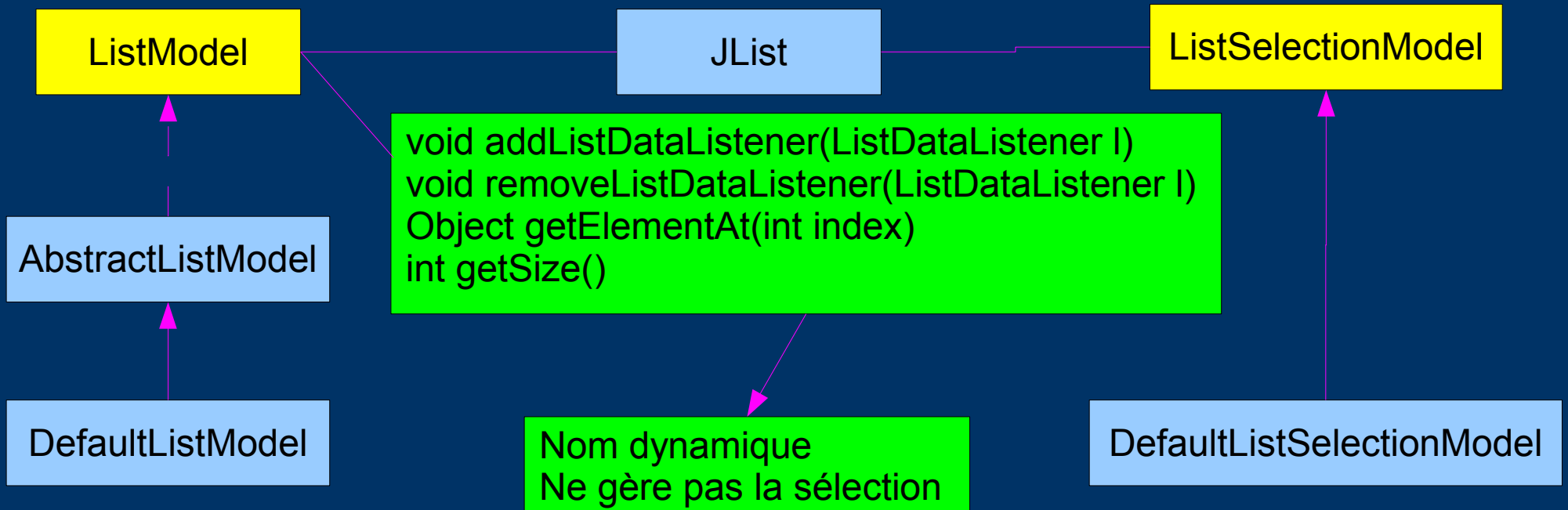
- `void setRenderer(ListCellRenderer l)`  
Spécifie l'objet responsable de l'affichage des différents éléments dans le menu PopUp
  - `void setEditor(ComboBoxEditor editeur)`  
Spécifie l'objet responsable de l'édition du composant sélectionné de la liste (par défaut, un `BasicComboBoxEditor` correspondant à un `JtextField`)
  - `void setKeySelectionManager(JComboBox.KeySelectionManager k)`  
Spécifie comment convertir une frappe clavier en la sélection d'un item
  - Interface `JcomboBox.KeySelectionManager`  
Une seule méthode : `int selectionForKey(char touche, ComboBoxModel m)`
- 
-

# JList

- Aperçu



- Structure



# *JList*

## *Le modèle de sélection (1)*

- Mode de sélection :
  - 3 modes de sélection :
    - SINGLE\_SELECTION
    - SINGLE\_INTERVAL\_SELECTION
    - MULTIPLE\_INTERVAL\_SELECTION
  - Méthodes
    - void setSelectionMode(int selectionMode)
    - int getSelectionMode()
- Gestion de la sélection (l'ordre des indices n'a pas d'importance)
  - void clearSelection()
  - void setSelectionInterval(int index0, int index1)
  - void addSelectionInterval(int index0, int index1)
  - void removeSelectionInterval(int index0, int index1)
  - boolean isSelectedIndex(int index)
  - boolean isEmptySelection()
  - int getMaxSelectionIndex()/int getMinSelectionIndex()  
indices max et min de la sélection (-1 si pas de sélection)
- 





# *JList*

## *Le modèle de sélection (2)*

- Prise en compte d'un changement dynamique de la liste
    - void insertIndexInterval(int index, int length, boolean before)
    - void removeIndexInterval(int index0, int index1)
  - Débuts et fin des dernières sélections
    - int getAnchorSelectionIndex()
    - int getLeadSelectionIndex()
    - void setAnchorSelectionIndex(int index)
    - void setLeadSelectionIndex(int index)

derniers index0 (anchor) et index1 (lead) d'une sélection effectuée avec setSelection, addSelection, removeSelection
  - Gestion des écouteurs de sélection
    - void addListSelectionListener(ListSelectionListener x)
    - void removeListSelectionListener(ListSelectionListener x)
  - Spécification d'une sélection temporaire
    - boolean getValueIsAdjusting()
    - void setValueIsAdjusting(boolean valueIsAdjusting)
- 
-

# *JList*

## *Événement de modification de sélection*

- Interface `ListSelectionListener`
    - 1 seule méthode : `void valueChanged(ListSelectionEvent e)`
  - Classe `ListSelectionEvent`
    - Constructeurs :
      - `ListSelectionEvent(Object source, int firstIndex, int lastIndex, boolean isAdjusting)`
    - Méthodes
      - `int getFirstIndex()`
      - `int getLastIndex()`
      - `boolean getValueIsAdjusting()`
      - `String toString()`
- 
-

# *JList*

## *Constructeurs et Modes*

- Constructeurs
    - JList(), JList(ListModel modèle)
    - JList(Object[] données), JList(Vector<?> données)
  - Mode de sélection
    - setSelectionMode(int mode)
  - Mode d'affichage
    - setLayoutOrientation(int mode) :
      - VERTICAL
      - HORIZONTAL\_WRAP
      - VERTICAL\_WRAP
  - Méthodes
    - Méthodes des modèles sous-jacents
    - Autre méthodes (c.f. doc)
- 
-

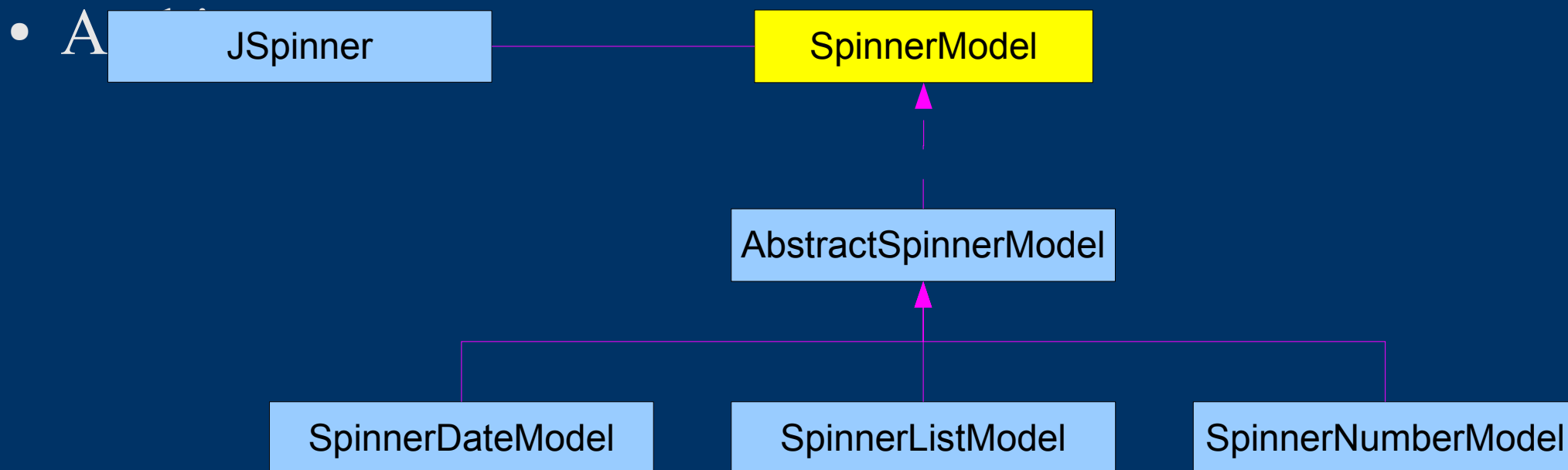
# JSpinner

- Aperçu

- But



- Sélection d'une seule valeur séquentiellement dans une liste ordonnée (pouvant être infinie) via un champ d'une seule ligne



# *JSpinner*

## *Modèle SpinnerModel*

- Valeur courante
    - setValue(Object value)
    - Object getValue()
  - Navigation
    - Object getNextValue()
    - Object getPreviousValue()
  - Ecouteurs
    - addChangeListener(ChangeListener l)
    - removeChangeListener(ChangeListener l)
  - Classe AbstractSpinnerModel
    - Implante la gestion de la liste d'écouteurs
- 
-

# *JSpinner*

## *Classe SpinnerListModel*

- Constructeurs
  - SpinnerListModel()
  - SpinnerListModel(List<?> objets)
  - SpinnerListModel(Object[] objets)
- Méthodes
  - List<?> getList()
  - Void setList(List<?> objets)



```
String [] valeurs = {"veau", "vache", "cochon", "couvee"};  
SpinnerModel modele = new SpinnerListModel(valeurs);  
  
JSpinner spinner = new JSpinner(modele);
```

# JSpinner

## Classe SpinnerNumberModel

- Constructeurs

- SpinnerNumberModel()
- SpinnerNumberModel(double value, double min, double max, double incr)
- SpinnerNumberModel(int value, int min, int max, int incr)
- SpinnerNumberModel(Number value, Comparable min, Comparable max, Number incr)



- Méthodes

- Comparable getMaximum(), Comparable getMinimum()
- Number getNumber(), Number getStepSize(), void setStepSize(Number n)
- void setMinimum(Comparable c), void setMaximum(Comparable c)

```
SpinnerModel modele = new SpinnerNumberModel(20,0,100,2);  
JSpinner spinner = new JSpinner(modele);
```

```
SpinnerModel modele2 = new SpinnerNumberModel(0.5,0.,1.0,0.05);  
JSpinner spinner2 = new JSpinner(modele2);
```

# *Jspinner*

## *Classe SpinnerDateModel*

- Constructeurs
  - SpinnerDateModel()
  - SpinnerDateModel(Date value, Comparable start, Comparable end, int calendarField)
- Valeurs min et max
  - Des dates
  - Null si pas de borne
- Incrément
  - Un entier correspondant à un champ de calendrier
  - Exemples : Calendar.YEAR, Calendar.MONTH, Calendar.DAY\_OF\_MONTH, Calendar.SECOND, ...





# *JSpinner pour une date*

## *Exemple*



```
Calendar debut, courant, fin;  
courant = Calendar.getInstance();  
debut = (Calendar) courant.clone(); fin= (Calendar) courant.clone();  
debut.set(courant.get(Calendar.YEAR), 0, 1);  
fin.set(courant.get(Calendar.YEAR), 11, 1);  
  
SpinnerModel modele = new SpinnerDateModel(  
    courant.getTime(),debut.getTime(),fin.getTime(),Calendar.DAY_OF_MONTH);  
JSpinner spinner = new JSpinner(modele);  
  
spinner.setEditor(new JSpinner.DateEditor(spinner, "dd/MM/yyyy"));
```

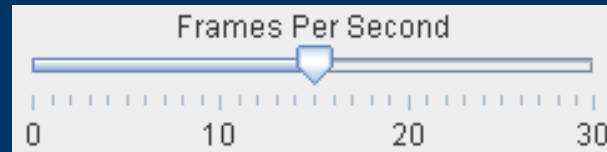
# *Jspinner*

## *Essentiel de la classe*

- Constructeurs
    - Jspinner()
    - Jspinner(SpinnerModel m)
  - Editeurs : différents éditeurs en fonction du modèle
    - SpinnerListModel → JSpinner.ListEditor
    - SpinnerNumberModel → JSpinner.NumberEditor
    - SpinnerDateModel → JSpinner.DateEditor
    - Autre → JSpinner.DefaultEditor
  - Méthodes essentielles
    - void addChangeListener(ChangeListener cl)
    - void removeChangeListener(ChangeListener cl)
- 
-

# JSlider

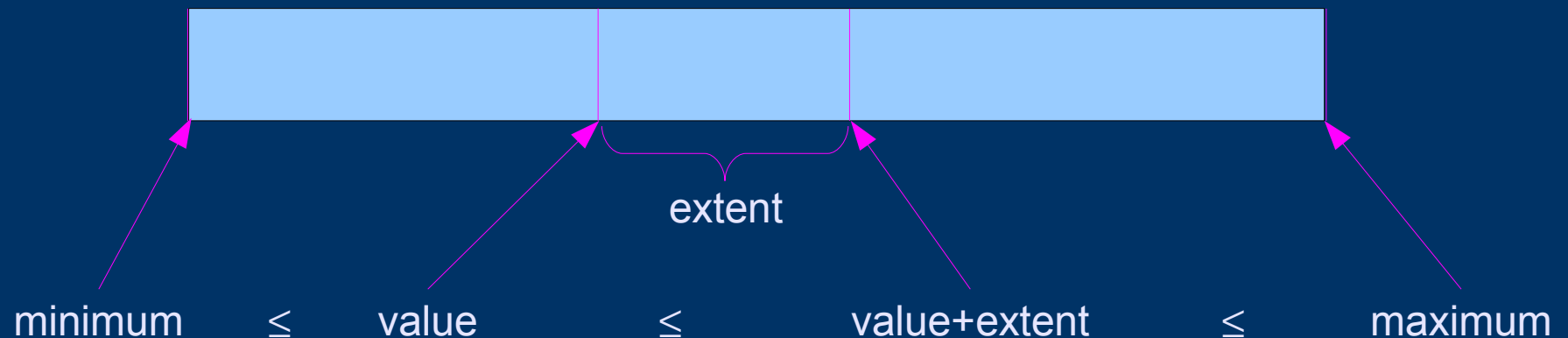
- Aperçu



- Rôle

- Sélectionner graphiquement une valeur dans un intervalle borné
- Des graduations mineures et majeures peuvent être définies ainsi que des étiquettes

- Modèle sous-jacent : BoundedRangeModel



# *JSlider*

## *Construction*

- Constructeurs
  - JSlider()
  - JSlider(BoundedRangeModel brm)
  - JSlider(int orientation)
  - JSlider(int min, int max)
  - JSlider(int min, int max, int value)
  - JSlider(int orientation, int min, int max, int value)
- Orientation
  - SwingConstants.HORIZONTAL
  - SwingConstants.VERTICAL



# *JSlider*

## *Présentation*

- Graduations principales
    - Void setMajorTickSpacing (int intervalle)
  - Graduations secondaires
    - Void setMinorTickSpacing (int intervalle)
    - Passer 0 si graduations non désirées
  - Labels
    - Méthode void setLabelTable(Dictionary table) où table est un liste de couples (Integer position, JComponent label)
    - Construction automatique après un setMajorTickSpacing
    - Construction facilitée via createStandardLabels (int inc)
  - Options de dessin
    - Piste : setPaintTrack (vrai par défaut)
    - Graduations : setPaintTicks (faux par défaut)
    - Partie colorée : gauche-bas/droite-haut : setInverted (faux par défaut)
    - Forcer l'alignement sur une graduation : setSnapToTicks (faux par défaut)
    - Dessin des étiquettes : setPaintLabels (faux par défaut)
- 
-

# JSlider

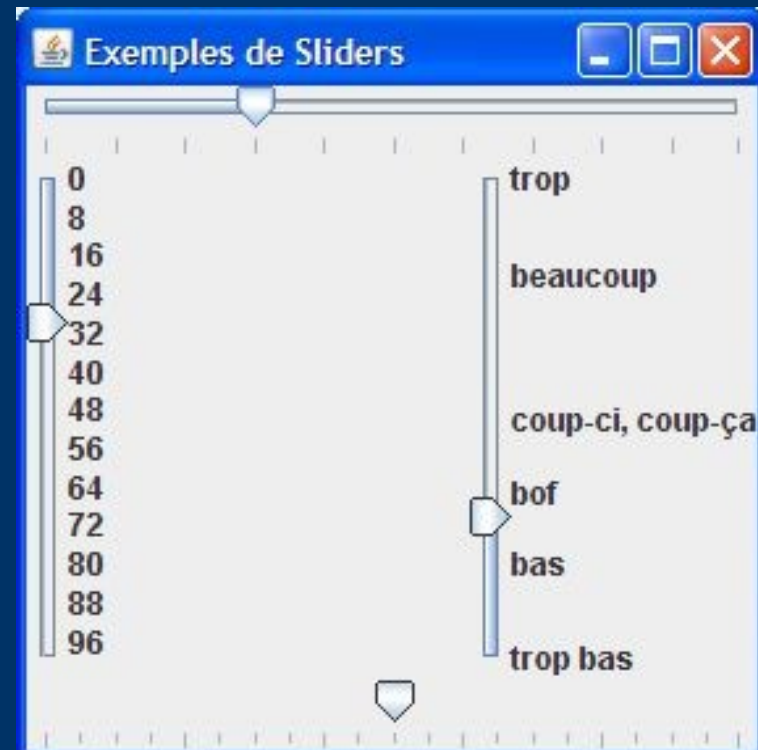
## Exemples

```
JSlider slider1 = new JSlider(JSlider.HORIZONTAL, 0, 100, 30);  
slider1.setMajorTickSpacing(10);  
slider1.setMinorTickSpacing(0);  
slider1.setPaintTicks(true);
```

```
JSlider slider2 = new JSlider(JSlider.HORIZONTAL, 0, 100, 50);  
slider2.setMajorTickSpacing(20);  
slider2.setMinorTickSpacing(5);  
slider2.setPaintTicks(true);  
slider2.setPaintTrack(false);
```

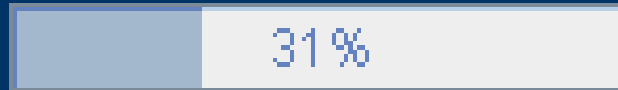
```
JSlider slider3 = new JSlider(JSlider.VERTICAL, 0, 100, 30);  
slider3.setInverted(true);  
slider3.setPaintLabels(true);  
slider3.setLabelTable(slider3.createStandardLabels(8));
```

```
JSlider slider4 = new JSlider(JSlider.VERTICAL, 0, 100, 30);  
Dictionary<Integer,JLabel> d = new Hashtable<Integer,JLabel>() {};  
d.put(0,new JLabel("trop bas"));  
d.put(20,new JLabel("bas"));  
d.put(35,new JLabel("bof"));  
d.put(50,new JLabel("coup-ci, coup-ça"));  
d.put(80,new JLabel("beaucoup"));  
d.put(100, new JLabel("trop"));  
slider4.setLabelTable(d);  
slider4.setPaintLabels(true);
```



# *JProgressBar*

- Aperçu



- Rôle

- Représenter un état d'avancement pour un processus de longueur déterminée ou non

- Modèle sous-jacent : `BoundedRangeModel`

- Présentation

- `void setStringPainted` (faux par défaut) : texte affiché dans la zone de progression
- `void setString(String s)` : texte à afficher si à null (défaut), affiche le pourcentage que représente la valeur courante
- `void setIndeterminate` (faux par défaut) : pour afficher un ascenseur faisant un va-et-vient plutôt qu'une valeur de progrès atteinte)

