Composants de Swing

- JavaBeans
- Actions
- Boutons
- Icônes
- Bordures
- Curseurs

Java Beans

Tout composant ***Swing*** est un JavaBean.

Un ***bean***

- est capable d'***introspection***
- possède et gère des ***propriétés***
- ***expose*** les propriétés
- communique par ***évènements*** les changements de propriétés
- réalise la ***persistance*** car sait se ***sérialiser***.

Propriétés

- Propriété
  - est ***exposée*** par **`setXX()`**, **`getXX()`**, **`isXX()`**
  - est ***simple***, ***liée*** (bound) ou ***contrainte***
- Propriété ***liée*** envoit un **`PropertyChangeEvent,`** chaque fois qu'elle change, aux *PropertyChangeListener*.
- Propriété ***contrainte*** envoit un **`PropertyChangeEvent`**
  - juste avant qu'elle ne change,
  - et d'autres composants peuvent s'opposer (véto) au changement (***VetoableChangeListener***).

ChangeEvent

- De plus, les composants *Swing* ont un événement “lèger”: **`ChangeEvent`**
- **`ChangeEvent`** n’a que la méthode **`getSource()`** (comme tout évènement).
- **`PropertyChangeEvent`** a les méthodes
  - **`getSource()`**
  - **`getOldValue()`**
  - **`getNewValue()`**
  - **`getPropertyName()`**

AbstractAction

- **`AbstractAction`** est une classe abstraite
  - elle implémente l'interface **`Action`**
  - **`Action`** étend **`ActionListener`**
  - la seule méthode à écrire est **`actionPerformed()`**
- Les conteneurs **`JMenu`**, **`JPopupMenu`** et **`JToolBar`** honorent les actions:
  - un même objet d'une classe implémentant **`AbstractAction`** peut être "ajouté" à plusieurs de ces conteneurs.
  - les diverses instances ***opèrent de concert***.
  - par exemple, un objet ajouté à un menu et à une barre d'outils est activé ou désactivé simultanément dans les deux.
- Les classes dérivées de **`AbstractAction`** sont utiles quand une ***même*** action peut être déclenchée de ***plusieurs*** manières.

Emploi d'AbstractAction

- Création d'une classe qui étend **`AbstractAction`**

```
class MonAction extends AbstractAction {
   public void actionPerformed( ActionEvent e )
   {
      ...
   }
}
```

- Utilisation comme **`ActionListener`**

```
Action monAction = new MonAction();
JButton b = new JButton("Hello");
b.addActionListener(monAction);
```

- Utilisation dans un menu *et* dans une barre d'outils

```
Action copyAction = new MonAction("Copy");
JMenu menu = new  JMenu("Edit");
JToolBar tools = new JToolBar();
JMenuItem copyItem = menu.add(copyAction);
JButton copyBtn = tools.add(copyAction);
```

Constructeurs et propriétés

- Avec une chaîne de caractères, et une icône

```
AbstractAction()
AbstractAction(String nom)
AbstractAction(String nom, Icon icône)
```

- Propriété : **`enabled`**
- Constantes

```
Action.DEFAULT  //
Action.NAME  // setText()
Action.SHORT_DESCRIPTION  // setToolTipText
Action.LONG_DESCRIPTION // aide contexte
Action.SMALL_ICON  // aussi setIcon()
```

- Gestion de propriétés:

```
void putValue(String clé, Object valeur)
Object getValue(String clé)
```

Exemple

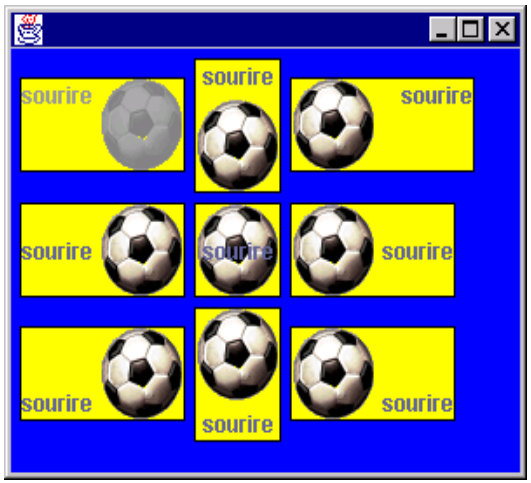

MyFrame.bat

```
public class MyFrame extends JFrame {
  private Action copyAction, cutAction, pasteAction;

  public MyFrame() {
    ...
    JMenuBar mbar = new JMenuBar();
    JToolBar toolbar = new JToolBar();
    JMenu edit = new JMenu("Edit");
    ...
    copyAction = new MonAction("Copy", new ImageIcon(”copy.gif"));

    JMenuItem copy = edit.add(copyAction);
    copy.setIcon(null);
      ...
    JButton bouton = toolbar.add(copyAction);
    bouton.setText(null);
    bouton.setToolTipText("copier");
    ...
    cutAction.setEnabled( false );
 }
```


```
class MonAction extends AbstractAction {
   String name;
   public MonAction(String name, Icon icon) {
      super( name, icon ); this.name = name;
   }
   public void actionPerformed(ActionEvent e) {
      System.out.println( name + " pressed" );
   }
}
```

Etiquettes

- Une étiquette peut contenir un texte, une icône, ou les deux.
- Leurs positions sont fixées par des constantes.
- Une étiquette peut être désactivée.



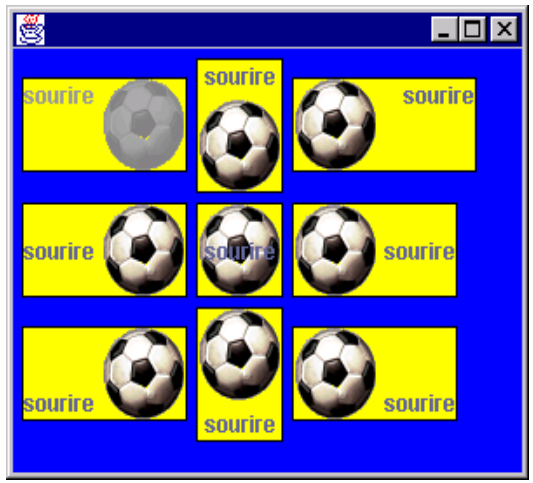
- Constructeurs

```
JLabel(String libellé, Icon image, int alignHorizontal)
```

  - les paramètres peuvent être absents
  - le dernier désigne le placement ***dans*** le conteneur
- Nombreuses fonctions utilitaires.

Etiquettes

```
public class ImageLabel {
  public static void main(String[] args) {
    JLabel[] e = new JLabel[9];
    e[0] = makeLabel(JLabel.TOP, JLabel.LEFT);
    e[1] = makeLabel(JLabel.TOP, JLabel.CENTER);
    e[2] = makeLabel(JLabel.TOP, JLabel.RIGHT);
    e[3] = makeLabel(JLabel.CENTER, JLabel.LEFT);
    e[4] = makeLabel(JLabel.CENTER, JLabel.CENTER);
    e[5] = makeLabel(JLabel.CENTER, JLabel.RIGHT);
    e[6] = makeLabel(JLabel.BOTTOM, JLabel.LEFT);
    e[7] = makeLabel(JLabel.BOTTOM, JLabel.CENTER);
    e[8] = makeLabel(JLabel.BOTTOM, JLabel.RIGHT);

    e[0].setEnabled(false);
    e[2].setIconTextGap(15);

    JFrame f = new JFra
    f.addWindowListener
    Container c = f.get
    c.setLayout(new Flo
   5));
    for (int i = 0; i <
      c.add(e[i]);
    f.setSize(300,250);
    f.setVisible(true);
    c.setBackground(Col
  }
```

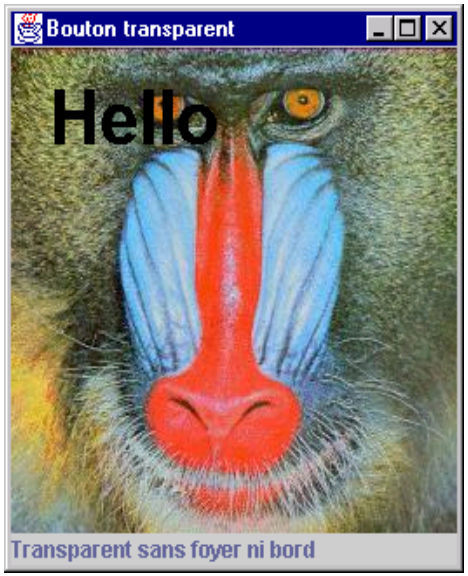



```
  protected static JLabel makeLabel(int vert, int horiz) {
    JLabel l = new JLabel("sourire", icon,
   SwingConstants.CENTER);
    l.setVerticalTextPosition(vert);
    l.setHorizontalTextPosition(horiz);
    l.setBorder(BorderFactory.createLineBorder(Color.black));
    l.setOpaque(true);
    l.setBackground(Color.yellow);
    return l;
  }
private static Icon icon = new ImageIcon("gifs/soccer.gif");
}
```

Boutons

TransparentButton.bat

- Un bouton délivre un **`ActionEvent`** quand on clique dessus, en plus des évènements de ses superclasses.
- Un bouton et une étiquette sont des ***conteneurs***.
- Dans le programme ci-contre,
  - le fond est une image dans un **`JLabel`**
  - le bouton “**`Hello`**” est ajouté à l’étiquette, vue comme conteneur
  - le bouton est rendu ***transparent***
  - quand on passe sur le bouton, le curseur se change en main.



Exemple de bouton

- l'image est chargée dans un **JLabel**
- l'étiquette est centrée dans le conteneur
- l'étiquette reçoit un Layout approprié et on y ajoute le bouton
- le bouton reçoit un **ActionListener**
- un message s'affiche dans l'étiquette du bas

```
public class TransparentButton extends JFrame implements ActionListener {
  JButton bouton = new JButton("Hello");
  static String[] msg = {"Opaque", "Transparent avec foyer",
       "Transparent sans foyer", "Transparent sans foyer ni bord",
       "Transparent sans fond"};
  int state = 0;
  JLabel message = new JLabel(msg[state]);
  TransparentButton() {
    super("Bouton transparent");
    Container contentPane = getContentPane();
    Icon image = new ImageIcon("gifs/mandrill.jpg");
    JLabel label = new JLabel(image);
    contentPane.add(label, BorderLayout.CENTER);
    contentPane.add(message, BorderLayout.SOUTH);
    label.setLayout(new FlowLayout(FlowLayout.LEFT));
    label.add(bouton);
    bouton.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
    bouton.setFont(new Font("Helvetica", Font.BOLD, 40));
    bouton.addActionListener(this);
  }
```

Exemple de bouton (fin)

```
 void actionPerformed(ActionEvent e) {
   switch (state) {
   case 0:bouton.setOpaque(false); break;
   case 1:bouton.setFocusPainted(false); break;
   case 2:bouton.setBorderPainted(false); break;
   case 3:bouton.setContentAreaFilled(false); break;
   case 4:
       bouton.setOpaque(true);
       bouton.setFocusPainted(true);
       bouton.setBorderPainted(true);
       bouton.setContentAreaFilled(true); break;
   }
   state = (state +1) % msg.length;
   message.setText(msg[state]);
   System.out.println(state + " Aille!");
 }

 public static void main(String[] args) {
   TransparentButton f = new TransparentButton();
   f.pack();
   f.setVisible(true);
   f.addWindowListener(new Fermeur());
 }
}
```

Icônes

- La classe **`ImageIcon`** implémente l'interface **`Icon`**
- Le chargement de l'image de l'icône se fait au moyen d'un **`MediaTracker`**, mais ceci est transparent à l'utilisateur.
- Constructeurs

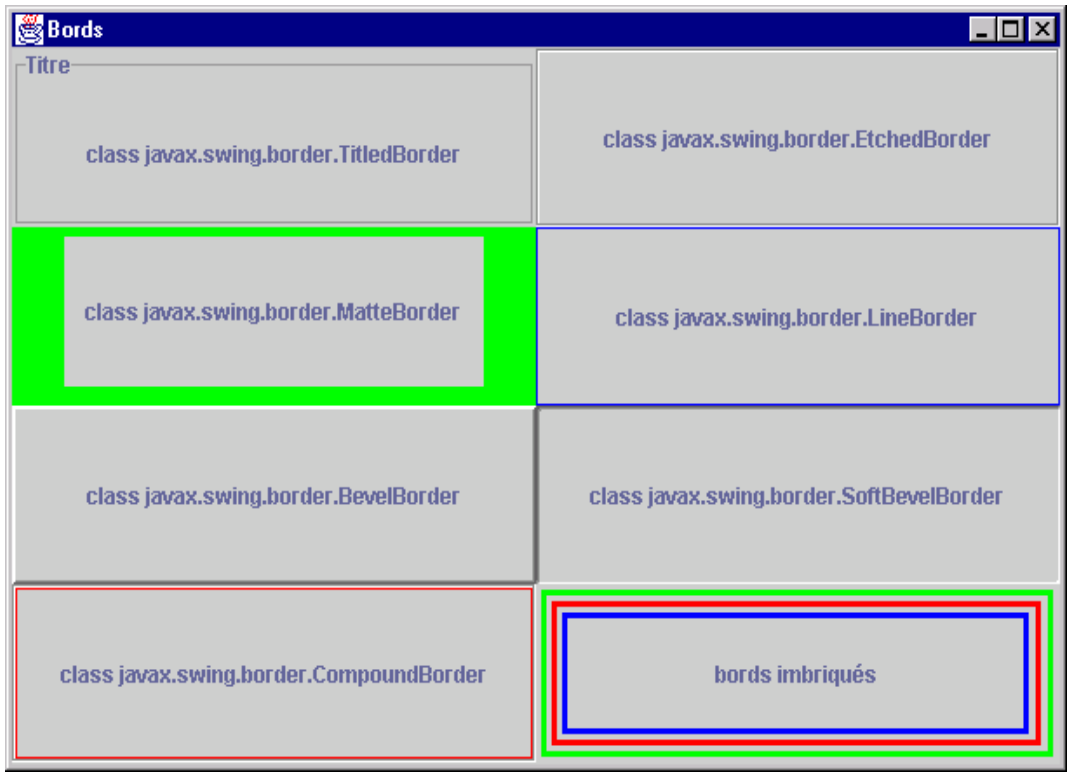
```
ImageIcon(String nomfichier)
ImageIcon(Image image)
ImageIcon(URL url)
```

- Methodes

```
int getIconHeight()
int getIconWidth()
Image getImage()
```

Bordures

- Peut être associé à tout composant dérivant de **`JComponent`**
- Figurent dans **`javax.swing.border`**
- Tout bord étend **`AbstractBorder`** qui implémente Border
- La classe **`BorderFactory`** a des méthodes statiques de création qui de plus essaient d'utiliser des bords déjà construits. Elle est dans **`javax.swing`**



Les divers bords


Border.bat

```
public class Bords extends JPanel {
  static JPanel panelBord(Border b) {...}
  public Bords(){
    setLayout(new GridLayout(4,2));
    add(panelBord(new TitledBorder("Titre")));
    add(panelBord(new EtchedBorder()));
    add(panelBord(new MatteBorder(5,30,10,30,Color.green)));
    add(panelBord(new LineBorder(Color.blue)));
    add(panelBord(new BevelBorder(BevelBorder.RAISED)));
    add(panelBord(new
   SoftBevelBorder(BevelBorder.LOWERED)));
    add(panelBord(new CompoundBorder(new EtchedBorder(),
        new LineBorder(Color.red))));
    ...
  }
```

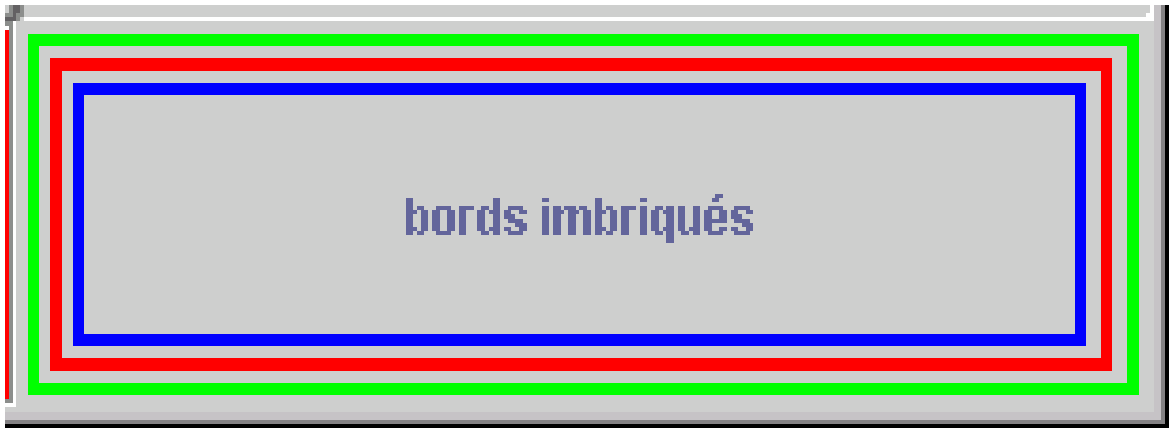
- Les classes de bords ont des noms différents.
- Les paramètres dépendent des classes
- On peut imbriquer un bord dans un autre en créant un bord composé.

Les noms des classes

- Par *introspection*, on obtient le nom de la classe

```
static JPanel panelBord(Border b) {
   JPanel p = new JPanel();
   p.setLayout(new BorderLayout());
   String nom = b.getClass().toString();
   p.add(new JLabel(nom,JLabel.CENTER), BorderLayout.CENTER);
   p.setBorder(b);
   return p;
}
```

Un bord compliqué




```
JPanel p = new JPanel();
p.setLayout(new BorderLayout());
p.add(new JLabel("bords imbriqués",JLabel.CENTER), BorderLayout.CENTER);
Border emptyBorder = new EmptyBorder(3,3,3,4);
Border b = new CompoundBorder(emptyBorder, new LineBorder(Color.blue,3) );
b = new CompoundBorder(new LineBorder(Color.red,3),b);
b = new CompoundBorder(emptyBorder, b);
b = new CompoundBorder(new LineBorder(Color.green,3), b);
b = new CompoundBorder(emptyBorder, b);
p.setBorder(b);
```

Curseur

- Un *curseur* (*slider*) permet d'entrer une valeur entre deux bornes selon uen échelle linéaire.
- On construit un curseur par

```
JSlider curseur = new JSlider(min, max, valeurInitiale);
```

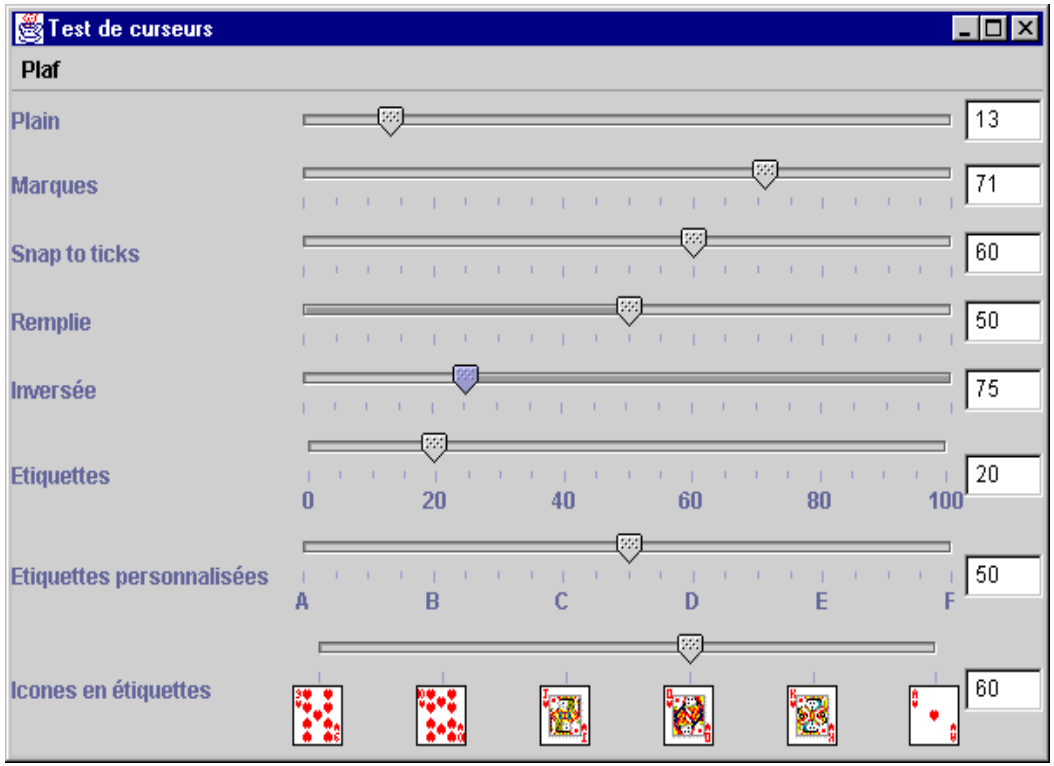
- si les valeurs extrêmes ne sont pas données, elles prennent les valeurs par défaut 0, 100, et 50 pour la valeur initiale.
- Un curseur vertical est construit avec

```
JSlider curseur = new JSlider(JSlider.VERTICAL,
  min, max, valeurInitiale);
```

Curseurs

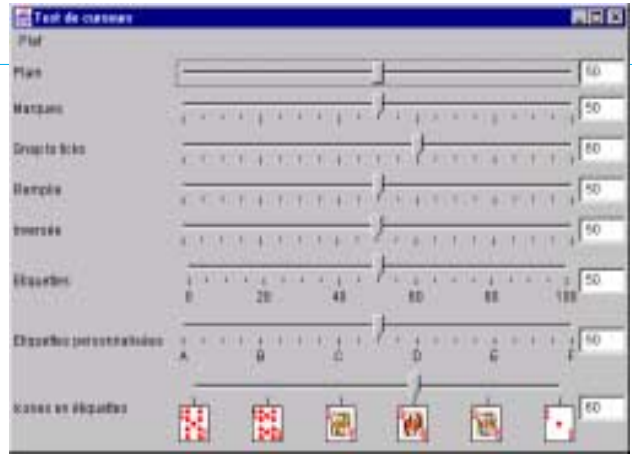

CurseurTest.bat

- A chaque changement de valeur, un curseur engendre un **`ChangeEvent`**
- On écoute ces évènements en enrégistrant un **`ChangeListener`**.
- Sa seule méthode est **`stateChanged()`.**




```
public void stateChanged(ChangeEvent ev) {
  JSlider curseur = (JSlider)ev.getSource();
  int valeur = curseur.getValue();
...
```

Curseurs : décorations



- Marques d'espacement:

```
curseur.setMajorTickSpacing(20);
curseur.setMinorTickSpacing(5);
```

- Affichage de ces marques:

```
curseur.setPaintTicks(true);
```

- Arrêt sur graduation:

```
curseur.setSnapToTicks(true);
```

- Libellés des graduations:

```
curseur.setPaintLabels(true);
```

- Inversion de direction:

```
curseur.setInverted(true);
```

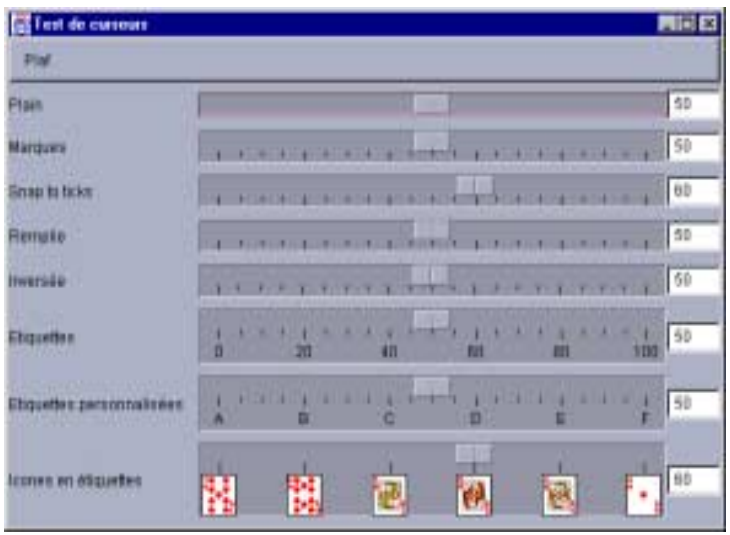
Curseurs: étiquettes personnalisées

- Les étiquettes sont des **`JLabel`**, rangées dans une table de hachage avec les valeurs de marques correspondantes, transformées en objets:

```
labelTable = new Hashtable();
labelTable.put(new Integer(0), new JLabel(new ImageIcon("9h.gif")));
labelTable.put(new Integer(20), new JLabel(new
  ImageIcon("10h.gif")));
labelTable.put(new Integer(40), new JLabel(new ImageIcon("jh.gif")));
```

- La table est enregistrée dans le curseur

```
curseur.setLabelTable(labelTable);
```

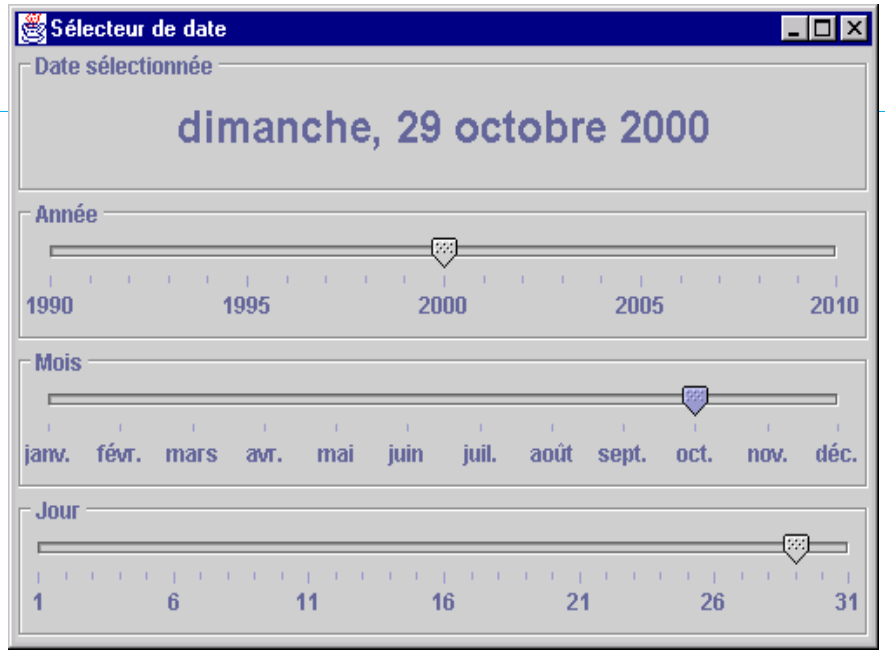


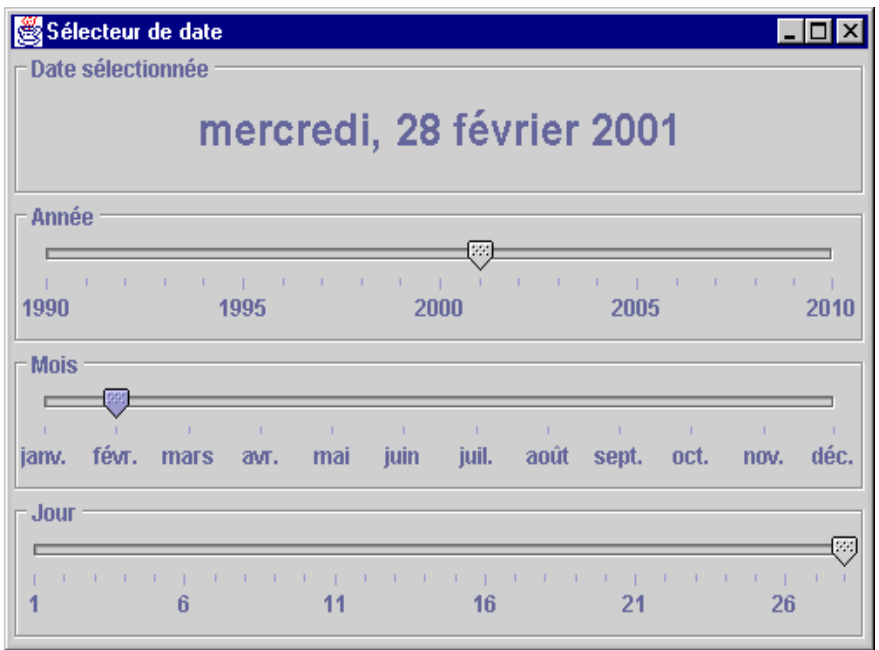
Exemple

CurseurDate.bat

- Trois curseurs permettent de sélectionner et d'afficher une date.

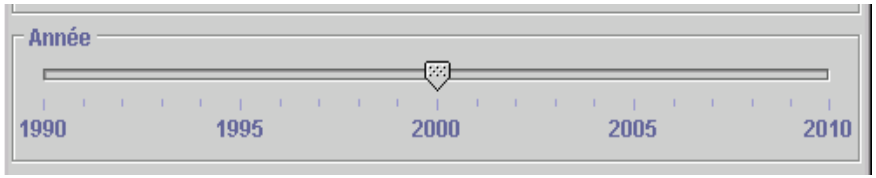
- Le curseur des jours ajuste son maximum en fonction du nombre de jours dans le mois.





Curseur des années

- De 1990 à 2010
- initialisé à l'année courante
- encadré et titré




```
Calendar calendar = new GregorianCalendar();
Date date = new Date();
calendar.setTime(date);
```

```
curAnnee = new JSlider(1990, 2010, calendar.get(Calendar.YEAR));
curAnnee.setPaintLabels(true);
curAnnee.setMajorTickSpacing(5);
curAnnee.setMinorTickSpacing(1);
curAnnee.setPaintTicks(true);
curAnnee.addChangeListener(lst);

p = new JPanel(new BorderLayout());
p.setBorder(new TitledBorder(new EtchedBorder(), " Année "));
p.add(curAnnee, BorderLayout.CENTER);
```

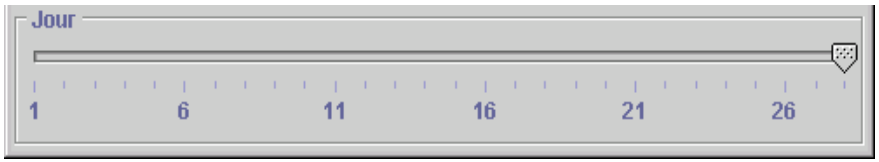
Curseur des mois

- les étiquettes sont des abréviations des noms des mois
- elles sont rangées dans une table de hachage


```
curMois = new JSlider( 0, 11, calendar.get(Calendar.MONTH));
String[] months = (new DateFormatSymbols()).getShortMonths();
Hashtable labels = new Hashtable(12);
for (int k = 0; k < 12; k++)
   labels.put(new Integer(k), new JLabel(months[k]));
curMois.setLabelTable(labels);
curMois.setPaintLabels(true);
curMois.setMinorTickSpacing(1);
curMois.setPaintTicks(true);
curMois.addChangeListener(lst);
```

Curseur des jours

- Initialisé au nombre de jours du mois courant
- Numérotation à partir de 1




```
int maxDays = calendar.getActualMaximum(Calendar.DAY_OF_MONTH);
curJour = new JSlider(1, maxDays, calendar.get(Calendar.DAY_OF_MONTH));
curJour.setPaintLabels(true);
curJour.setMajorTickSpacing(5);
curJour.setMinorTickSpacing(1);
curJour.setPaintTicks(true);
curJour.addChangeListener(lst);
```

Ajuster la date

- En réponse à un **ChangeEvent** de l'un des trois curseurs
  - recalculer la date
  - ajuster l'échelle des jours dans le curseur des jours
  - afficher la date dans la zone de date.

```
lst = new DateListener();
class DateListener implements ChangeListener {
   public void stateChanged(ChangeEvent e) {
       showDate();
   }
}
```

```
public void showDate() {
  calendar.set(curAnnee.getValue(), curMois.getValue(), 1);
  int maxDays = calendar.getActualMaximum(Calendar.DAY_OF_MONTH);

  if (curJour.getMaximum() != maxDays)
    curJour.setMaximum(maxDays);

  calendar.set(curAnnee.getValue(), curMois.getValue(),
    curJour.getValue());
  Date date = calendar.getTime();
  zoneDate.setText(dateFormat.format(date));
}
```