Interfaces Homme - Machine

Chapitre 6:

Gestions d'événements (3ème partie)

Quelques composants spécialisés

SMI - S6
Année universitaire 2024-2025

Prof. M.D. RAHMANI
d.rahmani@um5r.ac.ma

Plan

- **La gestion des événements (3ème partie)**
  - `FocusListener`, recevoir ou perdre le focus
  - `TextListener`, modification du texte entré,
  - `DocumentListener`, permet le suivi de la saisie et de l'édition, en temps réel
  - `CaretListener`, réagit aux événements de type curseur dans un champ de type texte
- **Composants spécialisés :**
  - `JList`, liste
  - `JSlider`, une glissière
  - `JSpinner`, liste à défilement

L'interface `FocusListener` (1)

- Un composant détient le focus s'il peut recevoir les frappes du clavier.
  - ✓ Un **seul** composant peut détenir le focus à un moment donné .
  - ✓ Il est possible de passer le focus à un autre composant :
    - ✓ avec le **clavier**, en général avec ***une tabulation***,
    - ✓ avec la **souris**.
    - ✓ par **programmation** : méthodes, ***demande***, `requestFocus` et ***transfert***, `transferFocus`.
  - ✓ Cette interface d'écoute contient **2 méthodes** :
    - `public void focusGained(FocusEvent ev)` // **recevoir le focus**
    - `public void focusLost(FocusEvent ev)` // **perdre le focus**
  - ✓ **ID de l'événement `FocusEvent` : FOCUS_GAINED et FOCUS_LOST**
  - ✓ Il existe une classe abstraite d'adaptation: `FocusAdapter`

L'interface `FocusListener` (2)

Syntaxe :

```
public class Focus0 implements FocusListener {
    public Focus0() {
        monComposant.addFocusListener(ecouteur);
    }
    public void focusGained(FocusEvent ev) {
        // traitement 1 associé au gain du focus
    }
    public void focusLost(FocusEvent ev) {
        // traitement 2 associé à la perte du focus
    }
}
```

Perte de focus d'un champ :

Il est parfois intéressant de s'assurer qu'un champ a été correctement saisi :

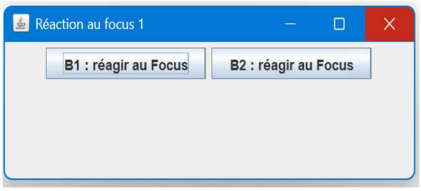
```
public void focusLost(FocusEvent ev) {
    if (! ev.isTemporary() && ! isCorrect(c.getText()))
        c.requestFocus();
}
```

✓ `isTemporary` retourne **vrai** si on est sûr de récupérer le focus (par exemple, parce qu'on a sélectionné une autre fenêtre).

L'interface FocusListener (3)

Exemple 1 : *Gain* et *Perte* du focus par **un composant**

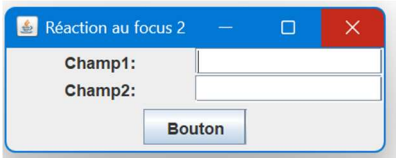
```
public class Focus1 extends JFrame implements FocusListener {
      JButton b1, b2;
      public Focus1() {
            setTitle("Réaction au focus 1");
            setLayout(new FlowLayout());
            b1 = new JButton("B1 : réagir au Focus");
            b1.addFocusListener(this);
            add(b1);
            b2 = new JButton("B2 : réagir au Focus");
            b2.addFocusListener(this);
            add(b2);
            setSize(300, 150);
            setDefaultCloseOperation(3);
            setVisible(true);
      }
      public void focusGained(FocusEvent ev) {
          if(ev.getSource() == b1){System.out.println("b1 a gagné le focus");}
          if(ev.getSource() == b2){System.out.println("b2 a gagné le focus");}
      }
      public void focusLost(FocusEvent ev) {
          if(ev.getSource() == b1){System.out.println("b1 a perdu le focus");}
          if(ev.getSource() == b2){System.out.println("b2 a perdu le focus");}
      }
      public static void main(String [] args) {
            new Focus1();
      }
}
```



L'interface FocusListener (4)

Exemple2 : Gain et Perte du focus par **plusieurs composants**

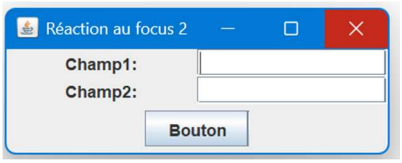
```
public class Focus2 extends JFrame implements FocusListener {
   JTextField champ1, champ2;
   JButton b;
   public  Focus2() {
       setTitle(" Réaction au focus 2");
       JLabel eChamp1 = new JLabel("Champ1:", JLabel.CENTER); // à gauche par défaut
       champ1 = new JTextField(15);
       champ1.addFocusListener(this);
       JLabel eChamp2 = new JLabel("Champ2:", JLabel.CENTER);
       champ2 = new JTextField(15);
       champ2.addFocusListener(this);
       JPanel p1 = new JPanel();
       p1.setLayout(new GridLayout(2,2));
       p1.add(eChamp1); p1.add(champ1);
       p1.add(eChamp2); p1.add(champ2);
       add("North",p1);
       b = new JButton("Bouton");
       b.addFocusListener(this);
       JPanel p2 = new JPanel();
       p2.add(b);
       add("South", p2);

       setDefaultCloseOperation(3);
       pack();
       setVisible(true);
   }
```



L'interface FocusListener (5)

Exemple2 (suite) : Gain et Perte du focus par plusieurs composants

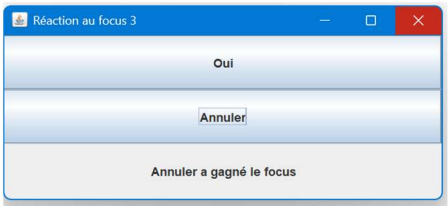
```
        public void focusGained(FocusEvent ev) {
          if(ev.getSource().equals(champ1)) // ou ev.getSource()==champ1
                System.out.println("Le champ de texte 1 a gagné le Focus du clavier.");
          else if(ev.getSource().equals(champ2))
                System.out.println("Le champ de texte 2 a gagné le Focus du clavier.");
          else if(ev.getSource().equals(b))
                System.out.println("Le bouton a gagné le focus");
        }
        public void focusLost(FocusEvent ev) {
          if(ev.getSource().equals(champ1))
                System.out.println("\t Le Champ de texte 1 a perdu le focus du clavier.");
          else if(ev.getSource().equals(champ2))
                System.out.println("\t Le Champ de texte 2 a perdu le focus du clavier.");
          else if(ev.getSource().equals(b))
                System.out.println("\t Le bouton a perdu le focus");
        }
        public static void main(String args[]) {
                new Focus2();
        }
}
```



L'interface FocusListener (6)

Exemple3 : Gain et Perte du focus par **plusieurs composants** et **écriture dans un autre composant**

```
public class Focus3 extends JFrame {
  JLabel etiquette;
  public Focus3() {
    setTitle(" Réaction au focus 3");
    setLayout(new GridLayout(3,0));
    JButton Ok = new JButton("Oui"); JButton Cancel = new JButton("Annuler");
    Ok.addFocusListener(new MonFocusListener());
    Cancel.addFocusListener(new MonFocusListener());
    add(Ok); add(Cancel);
    etiquette = new JLabel("", JLabel.CENTER);
    add(etiquette);
    setSize(450, 200);
    setVisible(true);
    setDefaultCloseOperation(3);
  }
  private class MonFocusListener implements FocusListener {
    public void focusGained(FocusEvent ev) {
      JButton bouton = (JButton) ev.getSource();
      etiquette.setText(bouton.getActionCommand()+" a gagné le focus");
    }
    public void focusLost(FocusEvent arg0) {}
  }
  public static void main(String[] args) {
    new Focus3();
  }
}
```



L'interface `TextListener` (1)

- ✓ Cette interface du package `java.awt.event` permet de réagir aux modifications de la zone de saisie ou du texte un caractère à la fois.
- ✓ L'événement généré par un objet de type `TextComponent` ou `Object` est `TextEvent`
- ✓ La méthode `addTextListener()` permet à un composant d'écouter un composant de texte.
- ✓ La méthode de cette interface à redéfinir `textValueChanged()` reçoit les événements.
- ✓ Cette interface écoute des composants de texte **AWT** uniquement !.

L'interface `TextListener` (2)

Exemple 1 : Copie du texte entré au clavier dans un champ de texte vers la console

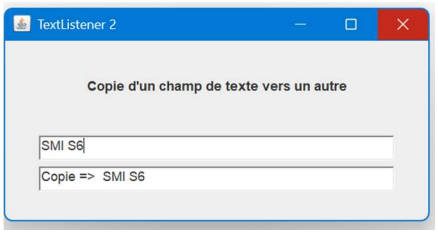
```
public class Texte1 extends JFrame {
   public Texte1() {
        setTitle("TextListener 1");
        TextField t = new TextField(40); // pas JTextField !
        add("North",t);
        t.addTextListener(new TextListener() {
           public void textValueChanged(TextEvent ev) {
                System.out.println("Le texte tapé: " + t.getText());
           }
/* ou         public void textValueChanged(TextEvent ev) {
                Object source = ev.getSource();
                System.out.println("Le texte tapé: "+((TextField)source).getText());
           }
*/
        });
        pack();
        setDefaultCloseOperation(3);
        setVisible(true);
   }
   public static void main(String args[]) {
        new Texte1();
   }
} // modification en temps réel , une mise à jour
```

Console: `Le texte tapé: SMI S6`

L'interface TextListener (3)

Exemple 2 : Copie du texte entré au clavier dans un champ de texte vers un autre

```
public class Texte2 extends JFrame {
   public Texte2(){
        setTitle("TextListener 2");
        setLayout(new GridLayout(0,1));
        JLabel eTitre = new JLabel("Copie d'un champ de texte vers un autre", JLabel.CENTER);
        TextField champ = new TextField(30); // cette taille impose le passage à la ligne du 2ème
        TextField copie = new TextField(30);
        JPanel panneau = new JPanel();
        panneau.add(champ);
        panneau.add(copie);
        add(eTitre);
        add(panneau);
        champ.addTextListener(new TextListener(){
           public void textValueChanged(TextEvent ev) {
                copie.setText("Copie => " + champ.getText());
           }
        });
        setSize(400,200);
        setDefaultCloseOperation(3);
        setVisible(true);
   }
   public static void main(String[] args){
        new Texte2();
   }
}
```



L'interface DocumentListener (1)

✓ Cette interface du package **javax.swing.event** permet le suivi de la saisie et de l'édition d'un texte **JTextField** ou **JTextArea**, en ***temps réel***,

✓ Un composant texte Swing utilise un Document pour stocker et éditer son texte.

✓ Les événements de Document se produisent lorsque le contenu d'un document change.

✓ Il faut attacher **DocumentListener** au document d'un composant texte, plutôt qu'au composant texte lui-même.

➢ Lorsque le texte a changé, l'une des méthodes de **DocumentListener** est appelée :

- **void insertUpdate(DocumentEvent ev);** *// caractères insérés*
- **void removeUpdate(DocumentEvent ev);** *// caractères supprimés*
- **void changedUpdate(DocumentEvent ev);** *// formatage ou changement de style*

✓ L'objet document associé à un composant s'obtient par la méthode **getDocument()**.

✓ Il n'existe pas de classe Adapter !, *l'écouteur de document doit implémenter les trois méthodes !*

✓ **La syntaxe :**

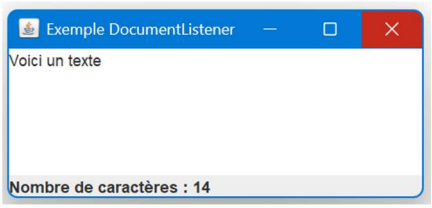
```
JTextField texte = new JTextField(20);
texte.getDocument().addDocumentListener(new MonDocumentListener());
```

L'interface `DocumentListener` (2)

✓ **Exemple:** créer une zone de texte et une étiquette.

Un `DocumentListener` est ajouté au document de la zone de texte met à jour l'étiquette chaque fois que le contenu du document change.

```
public class DocumentListener1 extends JFrame {
   public DocumentListener1() {
         setTitle("Exemple DocumentListener");
         JLabel et = new JLabel("Nombre de caractères : 0"); JTextArea ta = new JTextArea(6,15);
         ta.getDocument().addDocumentListener(new DocumentListener() {
            public void insertUpdate(DocumentEvent e) { nombreDeCaracteres();}
            public void removeUpdate(DocumentEvent e) { nombreDeCaracteres();}
            public void changedUpdate(DocumentEvent e) {
                  // nombreDeCaracteres();
            }
            private void nombreDeCaracteres() {
                  int compteur = ta.getDocument().getLength();
                  et.setText("Nombre de caractères : " + compteur);
            }
         });
         add("Center", ta); add("South", et);
         // la taille, la fermeture et la visibilité
   }
   public static void main(String[] args) {
         new DocumentListener1();
   }
}
```



L'interface `CaretListener` (1)

✓ Cette interface du package `javax.swing.event`, réagit aux événements de type curseur clignotant (*caret*) qui se produisent dans des composants textuels écoutés lorsque le curseur se déplace ou lorsque la sélection est modifiée.

✓ L'interface `CaretListener` ne possède qu'une méthode, il n'existe donc pas de classe Adapter associée.

✓ `CaretListener` permet d'écouter un composant textuel héritant de la classe `JTextComponent` à l'aide de la méthode `addCaretListener()`.

✓ La méthode associée à cette interface est :

`void caretUpdate(CaretEvent)`

✓ L'argument est un objet `CaretEvent` qui fournit des informations sur le changement du curseur.

✓ Pour obtenir la référence du composant textuel ayant émis l'événement, on utilise la méthode `getSource()`.

L'interface `CaretListener` (2)

✓ **Exemple:** créer une zone de texte écoutée par un `CaretListener`. La méthode `caretUpdate()` est appelée chaque fois que le caret se déplace ou que la sélection de texte change. La méthode affiche la position actuelle du caret et la sélection de texte.

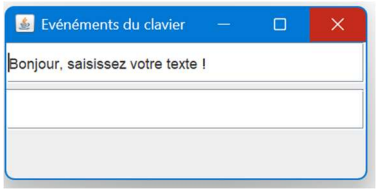
```
public class Caret1 extends JFrame {
   public Caret1() {
     setTitle("CaretListener 1");
     JTextArea textArea = new JTextArea();
     textArea.addCaretListener(new CaretListener() {
       public void caretUpdate(CaretEvent ev) {
         int dot = ev.getDot(); // la position
         int mark = ev.getMark(); // le texte sélectionné du début à la fin
         System.out.println("Position du caret : " + dot);
         System.out.println("Zone sélectionnée : "+(mark == dot ? "" : mark + "-" + dot));
       }
     });
     add(textArea);
  // la taille, la fermeture et la visibilité
   }
   public static void main(String[] args) {
        new Caret1();
   }
}
```

```
Console:
   Position du caret : 1
   Zone sélectionnée :
   Position du caret : 26
   Zone sélectionnée : 19-26
```


Exemple de saisie et de copie d'un texte

Voici un exemple de prise en compte de la saisie d'un texte en temps réel écouté par 3 types d'écouteurs liés aux interfaces:

- ✓ `KeyListener`,
- ✓ `DocumentListener`,
- ✓ `CaretListener`



La fenêtre de notre interface graphique est constituée de **2 champs de texte** de type `JTextField` et d'une **étiquette** de type `JLabel`.

Il s'agit d'insérer et de supprimer des caractères dans le **1er champ** et suivre la répercussion en temps réel sur le **2ème champ** et l'**étiquette**

Exemple avec KeyListener


```
public class PKey extends JFrame {
     private JTextField saisie, cible1;
     private JLabel cible2;
     public PKey() {
        setTitle("Evénéments du clavier");
        setLayout(new GridLayout(3,1,10, 5));
        saisie = new JTextField("Bonjour, saisissez votre texte !");
        cible1 = new JTextField(15); cible2 = new JLabel();
        add(saisie); add(cible1); add(cible2);
        saisie.addKeyListener(new KeyAdapter() {
             public void keyReleased(KeyEvent ev) {
          //   public void keyTyped(KeyEvent ev) {
                  if (ev.getKeyCode()==KeyEvent.VK_ENTER) {
                       saisie.setText("A bientôt !");
                  }
                  cible1.setText(saisie.getText());
                  cible2.setText(saisie.getText());
              }
        });
        setSize(300, 150); setDefaultCloseOperation(3); setVisible(true);
     }
     public static void main(String[] args) {
        new PKey();
     }
}
```


Exemple avec DocumentListener

```
public class PDocument extends JFrame {
     private JTextField saisie, cible1;
     private JLabel cible2 = new JLabel();
     public PDocument() {
          setTitle("Evénéments du clavier"); setLayout(new GridLayout(3,1,10, 5));
          saisie = new JTextField("Bonjour, saisissez votre texte !");
          cible1 = new JTextField(15); cible2 = new JLabel();
          add(saisie); add(cible1); add(cible2);
          saisie.getDocument().addDocumentListener( new DocumentListener() {// Avec DocumentListener
               public void insertUpdate(DocumentEvent ev) {
                    cible1.setText(saisie.getText()); cible2.setText(saisie.getText());
               }
               public void removeUpdate(DocumentEvent ev) {
                    cible1.setText(saisie.getText()); cible2.setText(saisie.getText());
               }
               public void changedUpdate(DocumentEvent ev) {}
          });
          saisie.addKeyListener(new KeyAdapter() {       // Pour quitter
               public void keyReleased(KeyEvent ev) {
                    if (ev.getKeyCode()==KeyEvent.VK_ENTER) {
                         saisie.setText("A bientôt !");
                    }
               }
          });
          setSize(300, 150); setDefaultCloseOperation(3);
     }
     public static void main(String[] args) {
          new PDocument().setVisible(true);
     }
}
```

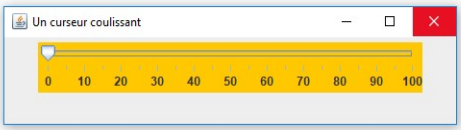
Exemple avec `CaretListener`

```
public class PCaret extends JFrame {
     JTextField saisie, cible1;
     JLabel cible2;
     public PCaret() {
         setTitle("Evénéments du clavier"); setLayout(new GridLayout(3,1,10, 5));
         saisie = new JTextField("Bonjour, saisissez votre texte !");
         cible1 = new JTextField(15); cible2 = new JLabel();
         add(saisie); add(cible1); add(cible2);
         // Avec CaretListener
         saisie.addCaretListener(new CaretListener() {
                  public void caretUpdate(CaretEvent ev) {
                     cible1.setText(saisie.getText());
                     cible2.setText(saisie.getText());
                  }
         });
         // Pour quitter
         saisie.addKeyListener(new KeyAdapter() {
                  public void keyReleased(KeyEvent ev) {
                     if (ev.getKeyCode()==KeyEvent.VK_ENTER) {
                           saisie.setText("A bientôt !");
                     }
                  }
         });
         setSize(300, 150); setDefaultCloseOperation(3);
     }
     public static void main(String[] args) {
         new PCaret().setVisible(true);
     }
}
```

Plan

- **Composants spécialisés :**
  - **`JSlider`,** une glissière
  - **`JSpinner`,** liste à défilement
  - **`JList`**, liste

Les glissières : `JSlider` (1)



Les glissières : `Jslider`

Un objet de type **`JSlider`** est un ***curseur coulissant*** ou une ***glissière*** que manipule l'utilisateur pour choisir une valeur dans une gamme de valeurs entières.

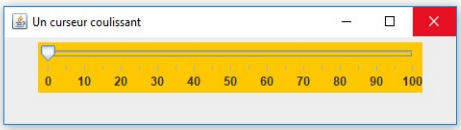
- **Création** :

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

- Les valeurs extrêmes par défaut sont 0, 100, et 50 pour la valeur initiale.
- Il est possible de la positionner *horizontalement* ou *verticalement*.
- Une glissière verticale est construite avec :

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


Les glissières : `JSlider` (2)



- La position du curseur est fixée avec **`setValue`** et lue avec **`getValue`**,
- L'orientation peut être changée en utilisant la méthode **`setOrientation`**
- Les **`JSlider`** peuvent être configurés pour couvrir une étendue à l'aide de **`setMinimum`** et **`setMaximum`**
- Les **traits de graduation** peuvent être rendus **visibles** avec la méthode **`setPaintTicks`**.
- Il est possible de tracer des **repères**, deux types sont proposés, les **repères majeurs** et les **repères mineurs**; les intervalles de chaque type sont définis avec **`setMinorTickSpacing`** et **`setMajorTickSpacing`**.
- On peut configurer les bornes et la position du curseur :

Exemple:

```
JSlider sCouleur = new JSlider(JSlider.HORIZONTAL,0,255,127);
```

Les glissières : `JSlider` (3)

Personnaliser l'apparence d'un curseur coulissant

```
void setValue(int n)
int getValue()  // fixe ou retourne la valeur courante du curseur.
void setMinimum(int minimum)
int getMinimum()
void setMaximum(int maximum)
int getMaximum()  // fixe ou retourne la borne min ou max du curseur.
void setOrientation(int orientation)
int getOrientation()  // fixe ou retourne l'orientation (JSlider.HORIZONTAL
                      // ou JSlider.VERTICAL) du curseur.
void setInverted(boolean b)
boolean getInverted()  // fixe ou retourne si les bornes min et max du curseur
                       // sont inversées.
void setMajorTickSpacing(int n)
int getMajorTickSpacing()  // fixe ou retourne l'espace entre les graduations
                           // principales.
```


Les glissières : `JSlider` (4)

```
void setMinorTickSpacing(int n)
int getMinorTickSpacing()
// fixe ou retourne l'espace entre les graduations secondaires.
void setPaintTicks(boolean b)
boolean getPaintTicks()
// fixe ou retourne si les graduations doivent être visibles.
void setPaintLabels(boolean b)
boolean getPaintLabels()
// fixe ou retourne si les étiquettes des graduations doivent être visibles.
```

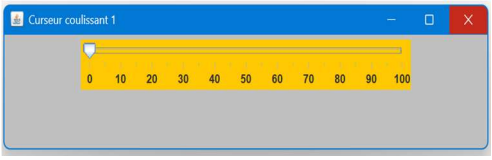
- **Gérer les déplacements du curseur :**

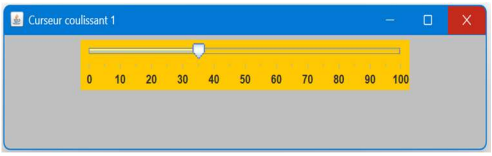
```
void  addChangeListener(ChangeListener)
// associe un écouteur déplacements du curseur.
boolean  getValueIsAdjusting()
// retourne true si l'utilisateur a terminé de déplacer le curseur.
```

Les glissières : `JSlider` (5)

Exemple 1 : Création d'une glissière

```
public class CurseurCoulissant1 extends JFrame {
   public CurseurCoulissant1() {
        setTitle("Curseur coulissant 1");

        JPanel panneau = new JPanel();
        panneau.setBackground(Color.lightGray);
        JSlider glissiere = new JSlider();
        glissiere.setOrientation(JSlider.HORIZONTAL);
        glissiere.setMaximum(100);
        glissiere.setValue(0); // valeur de départ du curseur
        glissiere.setMajorTickSpacing(10); // 10 est la valeur séparant les chiffres
        glissiere.setMinorTickSpacing(5); // 5 est la valeur séparant les tirets
        glissiere.setPaintTicks(true); // rendre visible les tirets
        glissiere.setPaintLabels(true); // rendre visible les nombres
        glissiere.setPreferredSize(new Dimension(400,50));
        glissiere.setBackground(Color.orange);
        panneau.add(glissiere);
        add(panneau);
        setBounds(20, 20, 600, 150);
        setDefaultCloseOperation(3);
   }

   public static void main(String[] args) {
        new CurseurCoulissant1().setVisible(true);
   } // On peut déplacer le curseur avec le déplacement de la souris et le bouton gauche appuyé
}
```





Les glissières : `JSlider` (6)

- Nous pouvons associer à **`JSlider`** l'écouteur d'événement **`ChangeListener`** du paquage **`javax.swing.event`**,
- Pour écouter une glissière, on utilise la méthode **`addChangeListener`**
- Les événements associés de type, **`ChangeEvent`** peuvent être générés par: **`AbstractButton`**, **`JProgressBar`**, **`JSlider`**, …
- La méthode **`getValue()`** donne la valeur associée au curseur.
- Cette interface dispose d'une seule méthode à redéfinir :

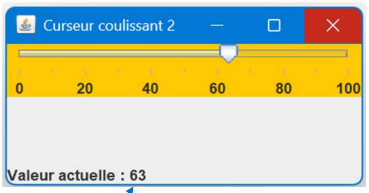
```
stateChanged(ChangeEvent ev);
```

Exemple :

```
public void stateChanged(ChangeEvent ev){
        (JSlider)ev.getSource()).getValue();
}
```

Les glissières : JSlider (7)

Exemple 2 : Curseur coulissant avec écouteur




```
public class CurseurCoulissant2 extends JFrame{
  private JLabel etiquette = new JLabel("Valeur actuelle : 30");

  public CurseurCoulissant2(){
    setTitle("Curseur coulissant");
    JSlider glissiere = new JSlider();
    glissiere.setMaximum(100); glissiere.setMinimum(0);
    glissiere.setValue(30); // valeur de départ du curseur
    glissiere.setPaintTicks(true); // rendre visible les tirets
    glissiere.setPaintLabels(true); // rendre visible les nombres
    glissiere.setMinorTickSpacing(10); // 10 est la valeur séparant les tirets
    glissiere.setMajorTickSpacing(20); // 20 est la valeur séparant les chiffres
    glissiere.setBackground(Color.orange);
    glissiere.addChangeListener(new ChangeListener(){
       public void stateChanged(ChangeEvent ev){
          etiquette.setText("Valeur actuelle : "+ glissiere.getValue());
  //        etiquette.setText("Valeur actuelle : "+((JSlider)ev.getSource()).getValue());
       }
    });
    add("North", glissiere); add("South", etiquette);
    setSize(300, 150);
    setDefaultCloseOperation(3);
  }

  public static void main(String[] args){
    new CurseurCoulissant2().setVisible(true);
  }
}
```


Liste à défilement: JSpinner (1)

Le composant **JSpinner** permet à l'utilisateur de sélectionner une information parmi une séquence ordonnée de valeurs à l'aide de **2 boutons** permettant de faire **défiler les valeurs possibles**.

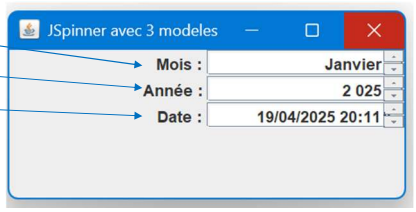
Fonctionnellement, le composant **JSpinner** est proche d'un composant **JComboBox** mais **seule la valeur courante est visible**.

Le composant **JSpinner** est un ***conteneur*** composé de **3 composants** :

- ✓ **Un champ** appelé éditeur dans lequel seront affichées ou éditées les valeurs.
- ✓ **Deux boutons** permettant de faire défiler les valeurs possibles.

La librairie Swing offre **3 modèles** par défaut :

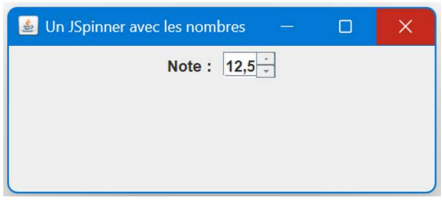
- ✓ **SpinnerListModel**
- ✓ **SpinnerNumberModel**
- ✓ **SpinnerDateModel**



Liste à défilement : JSpinner (2)

Exemple : Création d'un modèle **JSpinner** pour les nombres

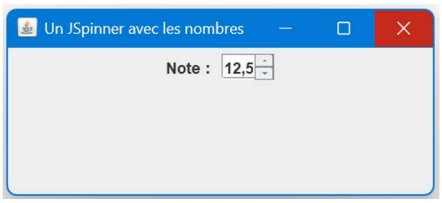
```
float init = 12.5f; // valeur initiale
float min = 0.0f; // valeur minimale
float max = 20.0f; // valeur maximale
float pas = 0.5; // incrément

SpinnerNumberModel snm = new SpinnerNumberModel(init,
                                                min,
                                                max,
                                                pas);

// Création du composant JSpinner
JSpinner note = new JSpinner(snm);

// Ajout de l'étiquette et du composant JSpinner
add(new JLabel("Note : ", JLabel.RIGHT));
add(note);
```



Liste à défilement : JSpinner (3)

Exemple 1 : Création d'un modèle **JSpinner** pour les **nombres**

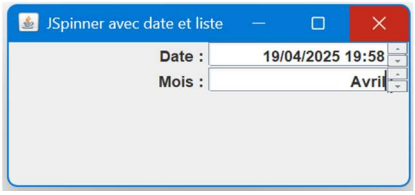
```
public class SpinnerNombre extends JFrame {
    public SpinnerNombre() {
        setTitle("Un JSpinner avec les nombres");
        setLayout(new FlowLayout());
        float init = 12.5f; // valeur initiale
        float min = 0.0f; // valeur minimale
        float max = 20.0f; // valeur maximale
        float pas = 0.5f; // incrément
        SpinnerNumberModel snm = new SpinnerNumberModel(init, min, max, pas);
        // Création du composant JSpinner
        JSpinner note = new JSpinner(snm);
        // Ajout de l'étiquette et du composant JSpinner
        add(new JLabel("Note : ", JLabel.RIGHT));
        add(note);
        setSize(350, 150);
        setDefaultCloseOperation(3);
    }
    public static void main(String[] argv) {
        new SpinnerNombre().setVisible(true);
    }
}
```



Liste à défilement : JSpinner (4)

Exemple 2 : Création d'un modèle **JSpinner** pour **Date** et **List**

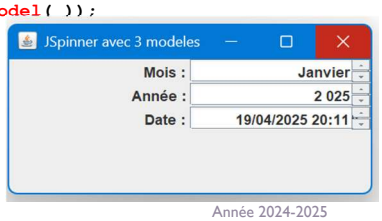
```
public class SpinnerDateEtListe extends JFrame {
    public SpinnerDateEtListe() {
        setTitle("JSpinner avec date et liste");
        JPanel p = new JPanel(); p.setLayout(new GridLayout(0,2));

       // date
        JSpinner spinner1 = new JSpinner(new SpinnerDateModel( ));
        p.add(new JLabel ("Date : ", JLabel.RIGHT));
        p.add(spinner1);
       // Liste
        String[] mois = {
         "Janvier", "février", "Mars", "Avril", "Mai", "Juin", "Juillet",
         "Aout", "Septembre", "Octobre", "Novembre", "Décembre"};
        JSpinner spinner2 = new JSpinner(new SpinnerListModel(mois));
        p.add(new JLabel("Mois : ", JLabel.RIGHT));
        p.add(spinner2);
        add("North", p);
        setSize(300, 150); setVisible(true);
        setDefaultCloseOperation(3);
   }
   public static void main(String[] argv) {
        new SpinnerDateEtListe();
   }
}
```



Liste à défilement : JSpinner (4)

Exemple 3 : Création d'un modèle **JSpinner** pour **Date** et **List et Nombre**

```
public class SpinnerDateEtListe extends JFrame {
    public SpinnerDateEtListe() {
        setTitle("JSpinner avec 3 modeles");
        JPanel p = new JPanel(); p.setLayout(new GridLayout(0,2));
         // Liste
        String[] mois = {
          "Janvier", "février", "Mars", "Avril", "Mai", "Juin", "Juillet",
          "Aout", "Septembre", "Octobre", "Novembre", "Décembre"};
        JSpinner spinner1 = new JSpinner(new SpinnerListModel(mois));
        p.add(new JLabel("Mois : ", JLabel.RIGHT));
        p.add(spinner1);
        // Nombre
        int init = 2025, min = 2000, max = 2050, pas = 1;
         // Nombre
        JSpinner spinner2=new JSpinner(new SpinnerNumberModel(init,min,max,pas));
        p.add(new JLabel("Année : ", JLabel.RIGHT));
        p.add(spinner2);
         // Date
        JSpinner spinner3 = new JSpinner(new SpinnerDateModel( ));
        p.add(new JLabel ("Date : ", JLabel.RIGHT));
        p.add(spinner3);
        add("North", p);
        setSize(300, 150); setDefaultCloseOperation(3);
   }
   public static void main(String[] argv) {
         new SpinnerDateEtListe(); setVisible(true);
   }
}
```



Les listes `JList` (1)

- C'est une liste déroulante prédéfinie dans laquelle l'utilisateur peut choisir un ou plusieurs éléments (*items*).
- On crée un objet de type `JList` en fournissant un **tableau de chaînes de caractères** à son constructeur,
  - `String[] choix = {"choix1", "choix2", "choix3", "choix4"};`
  - `Jlist <String>  liste = new Jlist <String>(choix);`
- Aucune valeur par défaut n'est sélectionnée.
- Pour choisir un élément d'un rang donné, on utilise la méthode:
  - `liste.setSelected(2);` // sélection préalable de l'élément de rang 2
  - `System.out.println(liste.getSelectedValue());` // Affichera choix3
- On peut ajouter une barre de défilement,
  - `JScrollPane bDefilement = new JScrollPane(liste);`

Les listes `JList` (2)

- Pour connaître la position d'un élément dans une liste :

```
int getSelectedIndex() // position d'un élément
int[] getSelectedIndices[] // tableau des positions de tous les éléments
```

  - Pour choisir le type de sélection (un ou plusieurs):

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

  - Il y'a 3 types de listes caractérisés par les paramètres :
    - `SINGLE_SELECTION` // sélection d'un seul élément
    - `SINGLE_INTERVAL_SELECTION` // une seule plage d'éléments
    - `MULTIPLE_INTERVAL_SELECTION` /* sélection d'un nombre quelconque de plages d'éléments */
- Pour sélectionner **une plage d'éléments**, l'utilisateur doit cliquer sur le 1er élément, appuyer sur la touche **MAJ**, tout en la maintenant enfoncée, cliquer sur le dernier élément de la plage.
- Pour sélectionner **plusieurs plages d'éléments**, on procède de la même manière avec en plus un appui sur **CTRL**.

Les listes `JList` (3)

❑ **Exemple 1** : création d'une liste

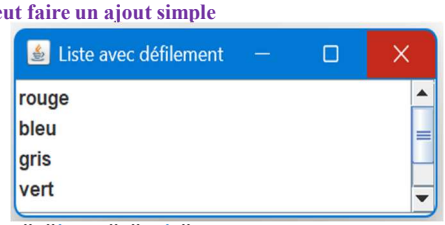
```
public class Liste1 extends JFrame {

     private String[] couleurs = {"rouge", "bleu", "gris", "vert", "jaune", "noir"};
     private JList<String> liste1;

     public Liste1 () {
         setTitle("Choix dans une liste");
         setLayout (new FlowLayout());
         liste1 = new JList<String> (couleurs);
         add(liste1);

         setSize (300, 160);
         setDefaultCloseOperation(3);

     }

     public static void main(String args[]) {
         new Liste1().setVisible(true);
     }
}
```

Remarque: *pour sélectionner plusieurs éléments, laisser appuyer le bouton* **CTRL**

Les listes `JList` (4)

❑ **Exemple 2** : création d'une liste avec une barre de défilement

```
import javax.swing.*;

public class Liste2 extends JFrame {

  public Liste2() {
     setTitle("Liste avec défilement");
     JList<String> liste2 = new JList<String>(couleurs);
     JScrollPane barreDeDefilement = new JScrollPane(liste2);
     add("Center", barreDeDefilement); // on peut faire un ajout simple

     setSize(300, 100);
     setDefaultCloseOperation(3);
  }
  public static void main(String args[]) {
     new Liste2().setVisible(true);;
  }

  String[] couleurs = {"rouge", "bleu", "gris", "vert", "jaune", "noir"};
}
```



Les listes `JList` (5)

- Evénements générés par les éléments d'une liste :

  Une liste génère un seul type d'événement `ListSelctionEvent`, écouté par `ListSelectionListener` de `javax.swing.event.*`.

  La méthode unique à redéfinir est `valueChanged()` :

```
public void valueChanged(ListSelectionEvent ev)
```

Remarque :

Cette méthode génère un événement lors de l'*appui* sur le bouton de la souris et lors de *relâchement*.

Pour pallier à cette redondance, la classe `JList` dispose d'une méthode `getValueIsAdjusting` qui permet de savoir si l'on est ou non en phase de transition, comme suit :

```
public void valueChanged(ListSelectionEvent ev) {
   if(!ev.getValueIsAdjusting()) {
        // accès aux éléments sélectionnés et traitements
   }
}
```


Les listes `JList` (6)

- Pour une liste à **sélection simple**, la méthode `getSelectedValue` rend la **seule chaîne sélectionnée**. Le résultat est de type `Object` et non `String`.

  Il faut par conséquent procéder à une conversion explicite :

```
String ch = (String) liste.getSelectedValue(); // cast obligatoire
```

- Pour une liste à **sélection multiple**, la méthode `getSelectedValues` rend un **tableau de chaînes sélectionnées**. Le résultat est de type `Object` et non `String`.

  Pour afficher toutes les chaînes sélectionnées :

```
Object[] elements = liste.getSelectedValues();
for (int i=0; i<elements.length; i++)
  System.out.println((String) elements[i]); // cast obligatoire
```

Les listes `JList` (7)

❑ **Exemple 3** : l'ajout d'un écouteur d'événements :

```
public class Liste3 extends JFrame implements ListSelectionListener {
   JList<String> liste3;
   public Liste3 () {
         setTitle("Choix dans une liste");
         setLayout (new FlowLayout() );

         liste3 = new JList<String> (couleurs);
         add(liste3);
         liste3.addListSelectionListener(this);
         setSize (300, 160);
         setDefaultCloseOperation(3);
   }
   public void valueChanged (ListSelectionEvent ev) {
         if(!ev.getValueIsAdjusting()) {
              System.out.println ("Action sur la liste - valeur sélectionnée:");
              Object[] elements = liste3.getSelectedValues() ;
              for (int i = 0 ; i<elements.length ; i++)
                        System.out.println ((String) elements[i]);
         }
   }

   public static void main(String args[]) {
         new Liste3().setVisible(true);
   }
   String[] couleurs = {"rouge", "bleu", "gris", "vert", "jaune", "noir"};
}
```


```
Action sur la liste - valeur sélectionnée:
bleu
gris
jaune
```

Résultat: `Action sur la liste - valeur sélectionnée : bleu gris jaune`

Les listes `JList` (7 bis)

❑ **Exemple 3** : l'ajout d'un écouteur d'événements :

```
import java.util.List;
public class Liste3bis extends JFrame implements ListSelectionListener {
  JList<String> liste3;
  public Liste3bis () {
         setTitle("Choix dans une liste 2");
         setLayout (new FlowLayout() );
         liste3 = new JList<String> (couleurs);
         add(liste3);
         liste3.addListSelectionListener(this);
         setSize (300, 160);
         setDefaultCloseOperation(3);
  }
  public void valueChanged (ListSelectionEvent ev) {
         if(!ev.getValueIsAdjusting()) {
              System.out.println ("Action sur la liste - valeur sélectionnée:");
              List choix = liste3.getSelectedValuesList() ;
              System.out.println (choix);
         }
  }

  public static void main(String args[]) {
         new Liste3bis().setVisible(true);
  }
   String[] couleurs = {"rouge", "bleu", "gris", "vert", "jaune", "noir"};
}
```


```
Action sur la liste - valeur sélectionnée:
[gris]
Action sur la liste - valeur sélectionnée:
[gris, jaune]
```

Résultat:

`Action sur la liste - valeur sélectionnée : bleu gris jaune`

Les listes `JList` (8)

- **Exemple 4** : Choix multiple avec écouteur d'événements

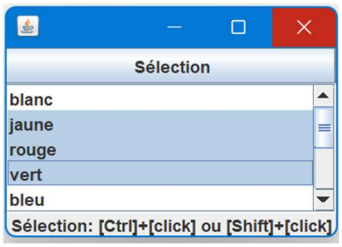
```
public class ListeChoixMultiple extends JFrame {
  //----- variables
  Jlist<String> couleurs;
  JLabel e;
  JButton b;

  public ListeChoixMultiple() {
        couleurs = new JList(T_Couleurs); // définition liste choix
        couleurs.setVisibleRowCount(5); // nombre de lignes visibles
        couleurs.setSelectionMode(
        ListSelectionModel.MULTIPLE_INTERVAL_SELECTION); // autorise la sélection multiple
        JScrollPane spListe = new JScrollPane(couleurs); // ajoute d'un ascenseur
        add("Center", spListe); // ajoute le composant dans la fenêtre
        //--- définition d'une étiquette
        e = new JLabel(" Sélection: [Ctrl]+[click] ou [Shift]+[click] ");
        add("South", e);
        b = new JButton("Sélection"); //--- définition bouton
        b.addActionListener(new Selection());
        add("North", b);
        pack(); setDefaultCloseOperation(3);
  }
  //----- constantes
  final static String[] T_Couleurs =  {"blanc", "jaune", "rouge", "vert", "bleu",
  "orange", "noir", "bleu clair", "rouge clair", "vert clair"};
```


Les listes `JList` (9)

- **Exemple 4 (suite)** : Choix multiple avec écouteur d'événements

```
  //----- classe interne
  private class Selection implements ActionListener {
        public void actionPerformed( ActionEvent ev ) {
              System.out.println( "Votre sélection:" );

              for (String item: couleurs.getSelectedValuesList()) {
                    System.out.println(item);
              }
        }
  }

  public static void main( String[] args ) {
        new ListeChoixMultiple().setVisible(true);
  }

}
```

Résultat :



Avec Ctrl + click : pas nécessairement contigu

Avec Shift + click: choix multiple contigu