Université de Bouira
Département d'Informatique
Module : Les Interactions Homme Machine
Niveau 3 info

Série TP 5

JPanel, JButton, et Event Handlers

Etapes à suivre

I. Top-Level Containers - JFrame

- Créer un nouveau projet Java sous le nom *IHMTP4*.
- Créer dans celui-ci une classe *MyJFrame* héritant de JFrame (extends JFrame).
- Pour configurer l'état initial de la fenêtre créée, ajouter une méthode *initJFrame* et appeler celle-ci depuis un constructeur. Utiliser les méthodes suivantes pour l'initialisation :
  - ✓ setTitle(String title)
  - ✓ setSize(int width, int height)
  - ✓ setLocationRelativeTo(null)
  - ✓ setResizable(booblean resizable)
  - ✓ setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)
- Ajouter une classe main nommée *TestJFrame*. Créer dans cette dernière une instance de la classe *MyJFrame* qui devrait s'exécuter dans l'Event Dispatch Thread (en utilisant SwingUtilities).
- Pour afficher cette instance, utiliser la méthode *setVisible*(true).

II. Les Event Handlers

Il est possible d'ajouter plusieurs écouteurs pour un seul événement.

- Créer un JPanel, *panel*, et l'appliquer comme ContentPane de la fenêtre.
- Créer un JLabel, *statusBar*, et initialiser son texte à `"0"`. Ajouter à *statusBar* une bordure de type EchedBorder.

```
JPanel panel = new JPanel();
this.setContentPane(panel);

JLabel statusBar = new JLabel("0");
statusBar.setBorder(BorderFactory.createEtchedBorder());
panel.add(statusBar);
```

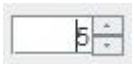
- Créer un JButton *addButton* et lui ajouter deux écouteurs comme suit :

```
JButton addButton = new JButton("+");
addButton.addActionListener(new ButtonListener1());
addButton.addActionListener(new ButtonListener2());
```

Dans cet exemple, la gestion des événements se fait par des classes internes (inner class) à la classe *MyJFrame*, appelées : ButtonListener1 et ButtonListener2. Ce sont ces classes qui implémentent ActionListener et définissent donc la méthode *actionPerformed*.

```
private class ButtonListener1 implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {
            //TODO
        }
}
```

- Créer un composant **JSpinner** en indiquant son modèle numérique, en donnant la valeur initiale, la valeur minimale, la valeur maximale, et le pas.



```
SpinnerModel numModel = new SpinnerNumberModel(0, -100, 100, 1);
JSpinner spinner = new JSpinner(numModel);
panel.add(spinner);
```

- Compléter les deux méthodes *actionPerformed* pour que *spinner* et *statusBar* ajoutent un 1 à leurs valeurs courantes et les affichent.

```
private class ButtonListener1 implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent e) {
         statusBar.setText(Integer.toString(++count));
    }
}

private class ButtonListener2 implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent e) {
         Integer val = (Integer) spinner.getValue();
         spinner.setValue(++val);
    }
}
```

- Afficher votre fenêtre. Cliquer sur le bouton addButton et remarquer l'affichage.

Quand quelque chose se passe dans l'application, un objet événement (Event) est créé. Par exemple, un clic sur un bouton ou une sélection d'un item d'une liste. Il existe par conséquent différents types d'événements - créés selon le composant source et l'interaction avec celui-ci- parmi eux : ActionEvent, TextEvent, MouseEvent, FocusEvent, ComponentEvent, etc. Afin de réagir à ces événements et définir le nouveau comportement, chaque type d'événements a son propre type d'écouteurs (Listener).

- Ajouter à *panel* un composant **JCheckBox** nommé *activeBox*. Initialiser son texte à "Activer/Désactiver" .

- Ajouter à *activeBox* un écouteur de type **ItemListener** en utilisant la méthode *addItemListener*. Utiliser une classe interne anonyme pour définir la méthode *itemStateChange*(ItemEvent). (NB : *itemStateChange* est l'équivalent de *actionPerformed* du type d'écouteurs **ActionListener**).

```
JCheckBox activeBox = new JCheckBox("Activer/Désactiver");
activeBox.addItemListener(new ItemListener() {

            @Override
            public void itemStateChanged(ItemEvent event) {

            }
});
panel.add(activeBox);
```

- Ajouter à *panel* un composant JLabel nommé *activeLabel*. Initialiser son texte à "activeBox Unchecked" .
- Dans la méthode *itemStateChange,* utiliser la méthode isSelected sur le JCheckBox *activeBox* pour changer le texte de *activeLabel* selon l'état activé/désactivé de *activeBox* :

```
activeBox.addItemListener(new ItemListener() {
            @Override
            public void itemStateChanged(ItemEvent event) {
                if (activeBox.isSelected()) {
                    activeLabel.setText("activeBox Checked.");
                } else {
                    activeLabel.setText("activeBox Unchecked.");
                }
            }
        });
```

Il est possible de désactiver un écouteur ajouté avec la méthode *removeItemListener*.

Changer la couleur d'une bordure - L'événement MouseEvent

En utilisant **MouseListener**, ajouter et changer la bordure de *activeLabel* selon le survol de la sourie : rouge en entrant (méthode *mouseEntered*) et noire en sortant (*mouseExited*).

```
activeLabel.addMouseListener(new MouseListener() {
    @Override
    public void mouseEntered(MouseEvent e) {
       activeLabel.setBorder(BorderFactory.createMatteBorder(1, 1,
            1, 1, Color.red));
    }
          . . . .
});
```

Déplacer une fenêtre – L'événement ComponentEvent

Cet exemple récupère et affiche la position (les coordonnées X et Y) de la fenêtre lorsqu'on la déplace en utilisant un écouteur de type **ComponentListener**.

- Créer un JPanel *panel* et l'appliquer comme ContentPane de la JFrame.
- Créer deux JLabel, *labelx* et *labely* et les ajouter à *panel*. Le texte de *labelx* initialisé à "X = " et *labely* à " Y = ".
- Personnaliser la police et la taille des deux JLabel créés.
- Ajouter un écouteur ComponentListener à la fenêtre en utilisant la méthode *addComponentListener(this).*

```
JLabel labelx = new JLabel("X = ");
labelx.setFont(new Font("Serif", Font.BOLD, 16));
panel.add(labelx);

JLabel labely = new JLabel("Y = ");
labely.setFont(new Font("Serif", Font.BOLD, 16));
panel.add(labely);

addComponentListener(this);
```

- Changer *MyJFrame* pour qu'elle implémente ComponentListener et ajouter toutes les méthodes à implémenter de l'interface.
- Changer la méthode *componentMoved* comme suit :

```
@Override
public void componentMoved(ComponentEvent e) {

        int x = e.getComponent().getX();
        int y = e.getComponent().getY();

        labelx.setText("X = " + x);
        labely.setText("y = " + y);
}
```

Les adaptateurs

Certains types d'écouteurs, comme dans l'exemple précédent, toutes les méthodes de l'interface (ex : ComponentListener) devraient être ajoutées et définies, même celles qui ne nous sont pas utiles. Pour éviter ces lignes de code non-nécessaire, les adaptateurs peuvent être utilisés. Une classe Adapter sert donc à n'implémenter que la méthode qui nous intéresse.

Par exemple, la classe ComponentAdapter implémente l'interface ComponentListener et en conséquence définit toutes ses méthodes, avec pour chacune un "corps vide". Si une classe étend la classe ComponentAdapter, elle possède par héritage des définitions pour les

méthodes déclarée dans l'interface ComponentListener ; ces méthodes ne faisant rien, il suffit de redéfinir uniquement les méthodes que l'on souhaite utiliser.

- Réécrire l'exemple précédent comme suit :

```
addComponentListener(new MoveAdapter());

…

private class MoveAdapter extends ComponentAdapter {

        @Override
        public void componentMoved(ComponentEvent e) {

            int x = e.getComponent().getX();
            int y = e.getComponent().getY();

            labelx.setText("X = " + x);
            labely.setText("Y = " + y);
}
```

III. Event Handling - Tableau récapitulatif des exemples utilisés

Exemple Composant Source	Type d'Evénements	Ecouteur d'événements (= Interface)	Méthode d'un écouteur d'événements
JButton	ActionEvent	ActionListener	actionPerformed
JCheckBox	ItemEvent	ItemListener	itemStateChanged
JFrame	ComponentEvent	ComponentListener	componentHidden componentMoved componentResized componentShown
JLabel	MouseEvent	MouseListener	mouseClicked mouseEntered mouseExited mousePressed mouseReleased

- Le code source complet sur **GitHub** :

https://github.com/GitTeaching/MesTPIHM