



L'interface `javax.swing.Action`

Gestion avancée des actions utilisateurs

Une fonction – n composants ?

- Interfaces utilisateur classiques : une même fonction est souvent à plusieurs endroits de l'interface (menus et toolbar).
- Comment faire en Java ?
- Première solution : Utiliser le même listener pour plusieurs composants.
- Problème : on ne réutilise pas toutes les informations communes (tooltip, actif ou non, etc.)
- On définit un objet qui regroupe ces informations en implémentant l'interface ***javax.swing.Action***

javax.swing

Interface Action

All Superinterfaces:

[ActionListener](#), [EventListener](#)

- Sous interface de **ActionListener** et de **EventListener**

Method Summary

void	addPropertyChangeListener (PropertyChangeListener listener) Adds a PropertyChange listener.
Object	getValue (String key) Gets one of this object's properties using the associated key.
boolean	isEnabled () Returns the enabled state of the Action.
void	putValue (String key, Object value) Sets one of this object's properties using the associated key.
void	removePropertyChangeListener (PropertyChangeListener listener) Removes a PropertyChange listener.
void	setEnabled (boolean b) Sets the enabled state of the Action.

Interface Action

Field Summary

static String	ACCELERATOR_KEY The key used for storing a <code>KeyStroke</code> to be used as the accelerator for the action.
static String	ACTION_COMMAND_KEY The key used to determine the command <code>String</code> for the <code>ActionEvent</code> that will be created when an <code>Action</code> is going to be notified as the result of residing in a <code>Keymap</code> associated with a <code>JComponent</code> .
static String	DEFAULT Not currently used.
static String	LONG_DESCRIPTION The key used for storing a longer <code>String</code> description for the action, could be used for context-sensitive help.
static String	MNEMONIC_KEY The key used for storing a <code>KeyEvent</code> to be used as the mnemonic for the action.
static String	NAME The key used for storing the <code>String</code> name for the action, used for a menu or button.
static String	SHORT_DESCRIPTION The key used for storing a short <code>String</code> description for the action, used for tooltip text.
static String	SMALL_ICON The key used for storing a small <code>Icon</code> , such as <code>ImageIcon</code> , for the action, used for toolbar buttons.

putValue

```
void putValue(String key,  
             Object value)
```

Sets one of this object's properties using the associated key. If the value has changed, a `PropertyChangeEvent` is sent to listeners.

Parameters:

key - a `String` containing the key
value - an `Object` value

Class AbstractAction

[java.lang.Object](#)

└ [javax.swing.AbstractAction](#)

All Implemented Interfaces:

[ActionListener](#), [Serializable](#), [Cloneable](#), [EventListener](#), [Action](#)

Direct Known Subclasses:

[BasicDesktopPaneUI.CloseAction](#), [BasicDesktopPaneUI.MaximizeAction](#),
[BasicDesktopPaneUI.MinimizeAction](#), [BasicDesktopPaneUI.NavigateAction](#),
[BasicDesktopPaneUI.OpenAction](#), [BasicFileChooserUI.ApproveSelectionAction](#),
[BasicFileChooserUI.CancelSelectionAction](#), [BasicFileChooserUI.ChangeToParentDirectoryAction](#),
[BasicFileChooserUI.GoHomeAction](#), [BasicFileChooserUI.NewFolderAction](#),
[BasicFileChooserUI.UpdateAction](#), [BasicInternalFrameTitlePane.CloseAction](#),
[BasicInternalFrameTitlePane.IconifyAction](#), [BasicInternalFrameTitlePane.MaximizeAction](#),
[BasicInternalFrameTitlePane.MoveAction](#), [BasicInternalFrameTitlePane.RestoreAction](#),
[BasicInternalFrameTitlePane.SizeAction](#), [BasicSliderUI.ActionScroller](#),
[BasicTreeUI.TreeCancelEditingAction](#), [BasicTreeUI.TreeHomeAction](#), [BasicTreeUI.TreeIncrementAction](#),
[BasicTreeUI.TreePageAction](#), [BasicTreeUI.TreeToggleAction](#), [BasicTreeUI.TreeTraverseAction](#),
[MetalFileChooserUI.DirectoryComboBoxAction](#), [TextAction](#)

class in javax.swing.plaf.basic

- Implémente les méthodes de l'interface Action
- sauf actionPerformed (classe abstraite)
- sera reliée à un composant à sa création via le constructeur

exemple

```
public class LeftAction extends AbstractAction {
    public LeftAction(String text, ImageIcon icon,
                      String desc, Integer mnemonic) {
        super(text, icon);
        putValue(SHORT_DESCRIPTION, desc);
        putValue(MNEMONIC_KEY, mnemonic);
    }
    public void actionPerformed(ActionEvent e) {
        displayResult("Action for first button/menu item", e);
    }
}
```



```
public void createToolBar() {
    JButton button = null;

    //Create the toolbar.
    JToolBar toolBar = new JToolBar();
    add(toolBar, BorderLayout.PAGE_START);

    //first button
    button = new JButton(leftAction);
    if (button.getIcon() != null) {
        button.setText(""); //an icon-only button
    }
    toolBar.add(button);

    //second button
    button = new JButton(middleAction);
    if (button.getIcon() != null) {
        button.setText(""); //an icon-only button
    }
    toolBar.add(button);

    //third button
    button = new JButton(rightAction);
    if (button.getIcon() != null) {
        button.setText(""); //an icon-only button
    }
    toolBar.add(button);
}
```

```
public JMenuBar createMenuBar() {
    JMenuItem menuItem = null;
    JMenuBar menuBar;

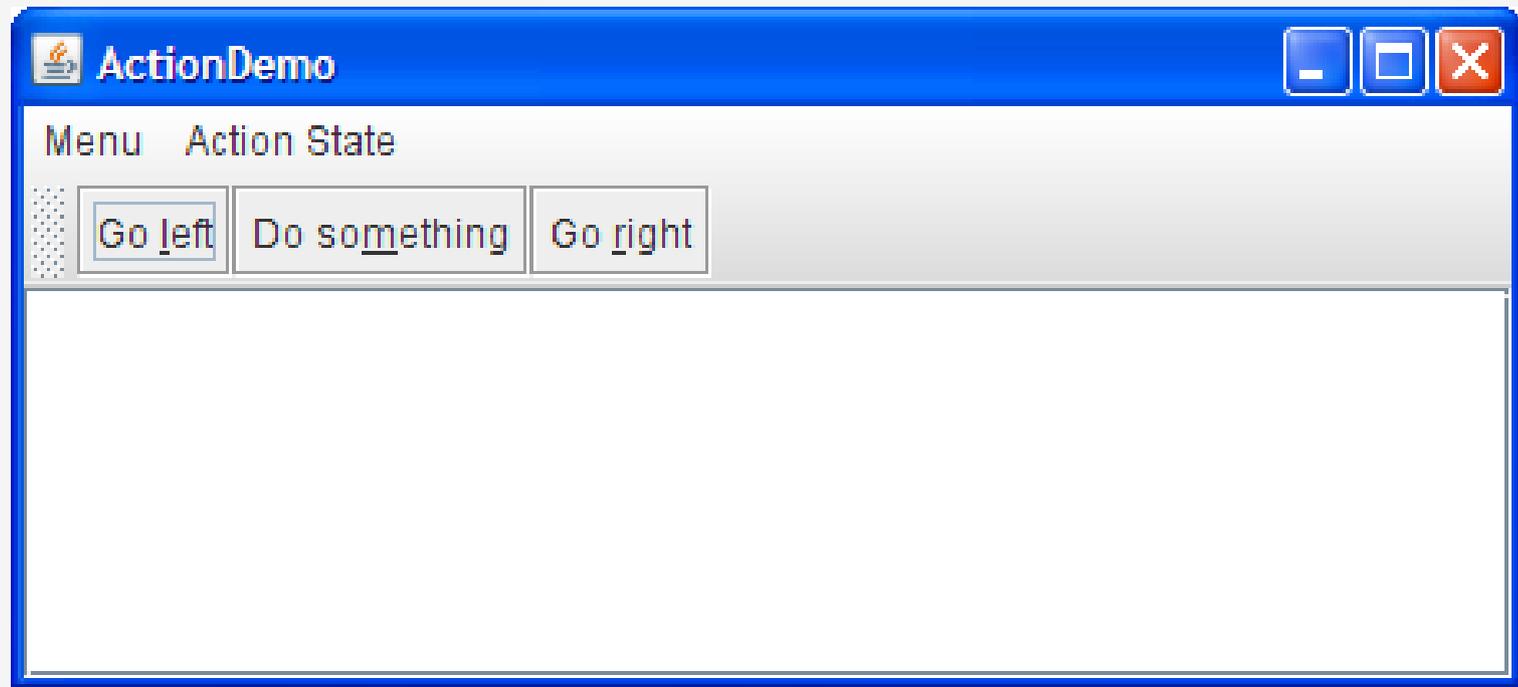
    //Create the menu bar.
    menuBar = new JMenuBar();

    //Create the first menu.
    JMenu mainMenu = new JMenu("Menu");

    Action[] actions = {leftAction, middleAction, rightAction};
    for (int i = 0; i < actions.length; i++) {
        menuItem = new JMenuItem(actions[i]);
        menuItem.setIcon(null); //arbitrarily chose not to use icon
        mainMenu.add(menuItem);
    }

    //Set up the menu bar.
    menuBar.add(mainMenu);
    menuBar.add(createAbleMenu());
    return menuBar;
}
```

Résultat



modifications d'une propriété

- Dans la classe ActionDemo:

```
protected JMenu createAbleMenu() {
    JMenu ableMenu = new JMenu("Action State");
    cbmi = new JCheckBoxMenuItem[3];

    cbmi[0] = new JCheckBoxMenuItem("First action enabled");
    cbmi[1] = new JCheckBoxMenuItem("Second action enabled");
    cbmi[2] = new JCheckBoxMenuItem("Third action enabled");

    for (int i = 0; i < cbmi.length; i++) {
        cbmi[i].setSelected(true);
        cbmi[i].addItemListener(this);
        ableMenu.add(cbmi[i]);
    }

    return ableMenu;
}
```

modification d'une propriété

- Dans la classe ActionDemo:

```
public void itemStateChanged(ItemEvent e) {
    JCheckBoxMenuItem mi = (JCheckBoxMenuItem)(e.getSource());
    boolean selected =
        (e.getStateChange() == ItemEvent.SELECTED);

    //Set the enabled state of the appropriate Action.
    if (mi == cbmi[0]) {
        leftAction.setEnabled(selected);
    } else if (mi == cbmi[1]) {
        middleAction.setEnabled(selected);
    } else if (mi == cbmi[2]) {
        rightAction.setEnabled(selected);
    }
}
```

Résultat

