

Systeme de focus en Java

par Baptiste Wicht ([home](#))

Date de publication : Le 2 Mai 2007

Dernière mise à jour :

Vous avez envie d'optimiser la gestion du focus pour votre application ? Ce tutoriel est fait pour vous. Vous allez pouvoir demander le focus pour un composant en particulier ou encore configurer l'ordre de focus.

- I - Introduction
- II - Demander le focus
- III - Touches de navigation
- IV - Valider une entrée
- V - Savoir qui a le focus
- VI - Gérer l'ordre de focus
 - VI-A - Simplification
- VII - La classe KeyboardFocusManager
- VIII - Conclusion

I - Introduction

Le focus est le fait qu'un composant soit "sélectionné" ou pas. Le composant ayant le focus est le composant actif. Il est possible de demander le focus sur un composant en particulier, mais normalement cela est géré automatiquement.

Avec ce tutoriel, vous allez apprendre à demander le focus pour un composant précis et à définir un ordre de focus. Nous allons également aborder la validation des champs de saisie, la navigation entre composants, l'écoute du focus et découvrir l'utilité de la classe `KeyboardFocusManager`.

II - Demander le focus


Normalement, le focus est gagné par un composant lors du clic souris ou lorsque qu'on arrive dessus au moyen des touches de sélection clavier. Un composant qui a le focus est souvent mis en évidence, par une bordure plus foncée ou alors par une couleur de fond différente.


Pour gérer le focus des fenêtres, c'est un peu différent et tout dépend du système sur lequel vous êtes, mais rien ne vous garantit d'avoir le focus. Sur Windows, vous pouvez faire obtenir le focus à une fenêtre en utilisant `ToFront()`, mais rien n'est garanti.

Pour ce qui est des composants, il faut que vous utilisiez la méthode `requestFocusInWindow()` :

Obtenir le focus

```
component.requestFocusInWindow();
```

 *Pour obtenir le focus initial, il faut faire attention à demander le focus après l'ajout du composant mais avant l'affichage de la fenêtre.*

 *Avant Java 1.4, il fallait utiliser la méthode `requestFocus()`, mais il est maintenant déconseillé de l'utiliser car elle essaie de donner le focus à la fenêtre en plus du composant, ce qui n'est pas toujours possible.*


III - Touches de navigation

Par défaut, on navigue dans une application avec les touches Tab et Shift+Tab. Ce sont les touches par défaut. Mais on peut tout à fait changer ces touches.

Pour cela, il faut récupérer ces touches avec la méthode `getFocusTraversalKeys(int id)` de n'importe quel composant et y ajouter une nouvelle touche ou combinaison de touches. Voilà un exemple dans lequel on ajoute la touche Enter comme touche de navigation.

Ajouter une touche de navigation

```
Set keys = textField.getFocusTraversalKeys(KeyboardFocusManager.FORWARD_TRAVERSAL_KEYS);
Set newKeys = new HashSet(keys);
newKeys.add(KeyStroke.getKeyStroke(KeyEvent.VK_ENTER, 0));
textField.setFocusTraversalKeys(KeyboardFocusManager.FORWARD_TRAVERSAL_KEYS, newKeys);
```

 *Si vous voulez ajouter une touche pour faire un retour arrière dans la navigation, il vous suffira d'utiliser `KeyboardFocusManager.BACKWARD_TRAVERSAL_KEYS` et non plus `KeyboardFocusManager.FORWARD_TRAVERSAL_KEYS`.*

IV - Valider une entrée

Vous pourriez avoir besoin une fois durant le développement de valider l'entrée utilisateur dans un champ. Vous devriez alors le faire dès que le composant n'a plus le focus. Bien sûr, il y a déjà des `Formatter` qui font ça, mais vous pourriez être amenés à développer un composant qui gère cela lui-même soit parce que les formatters ne font pas ce que vous voulez soit parce que votre composant n'est pas un champ texte.


Pour faire cela, nous allons utiliser des `InputVerifier`. Ces `InputVerifier` permettent de traiter la valeur contenue par le champ une fois que celui-ci va perdre le focus. Si la valeur du composant n'est pas valide, l'`InputVerifier` peut effectuer une action spécifique, par exemple remettre l'ancienne valeur dans le champ à la place de l'invalidé ou alors remettre le focus dans le champ en cours.

Pour l'utiliser, il suffit de créer une classe étendant `InputVerifier` et d'utiliser la méthode `setInputVerifier` sur le composant à tester :

```
EtendInputVerifier verifier = new EtendInputVerifier();  
  
...  
  
monComposant.setInputVerifier(verifier);
```

La classe `InputVerifier` ne possède que deux méthodes :

- `verify(JComponent input)` : Indique si la valeur du composant est correcte. Elle doit être surchargée.
- `shouldYieldFocus(JComponent input)` : Cette méthode permet d'indiquer si le composant peut perdre le focus ou non. Si elle retourne `true`, le focus change de composant normalement, si elle retourne `false`, le focus reste sur ce composant. Par défaut, elle ne fait que retourner le résultat de la méthode `verify()`.

 *Pour des raisons de lisibilité et de logique, je vous conseille de n'implémenter que du test de la valeur dans la méthode `verify` et d'écrire les autres fonctionnalités de l'`inputVerifier` dans la méthode `shouldYieldFocus`, par exemple émettre un bip si c'est faux.*

Je vais vous donner un simple exemple d'`InputVerifier`. Celui-ci vérifie que la valeur est bien un nombre, que ce nombre soit plus grand que zéro et inférieur à une valeur X. Si la valeur est fautive, on émet un simple bip.

```
import java.awt.Toolkit;  
  
import javax.swing.InputVerifier;  
import javax.swing.JComponent;  
import javax.swing.JTextField;  
  
public class NumberInputVerifier extends InputVerifier {  
    private int max = 0;  
  
    public NumberInputVerifier() {  
        this(100);  
    }  
  
    public NumberInputVerifier(int maximum) {  
        super();  
  
        this.max = maximum;  
    }  
  
    public boolean shouldYieldFocus(JComponent input) {  
        boolean valid = verify(input);
```

```
        if (valid) {
            return true;
        } else {
            Toolkit.getDefaultToolkit().beep();
            return false;
        }
    }

    public boolean verify(JComponent input) {
        JTextField field = (JTextField)input;


        String value = field.getText();
        int intValue;


        try {
            intValue = Integer.parseInt(value);
        } catch (NumberFormatException pe) {
            return false;
        }

        if(intValue < 0 || intValue > max){
            return false;
        }

        return true;
    }
}
```

Vous pouvez bien sûr faire beaucoup d'autres choses avec cette classe, vous pouvez en profiter pour changer le format d'un nombre en ajoutant des guillemets ou des virgules ou alors modifier n'importe quoi sur le composant d'entrée vu que le composant concerné est passé en paramètre

 *Vous pouvez tout à fait utiliser le même `InputVerifier` pour plusieurs composants. Il vous suffira ensuite de tester le composant concerné via le composant passé en paramètre des méthodes. Seulement, ne surchargez pas trop non plus vos `InputVerifier`, s'ils prennent en charge trop de composants, vous aurez de la peine à les réutiliser ensuite dans d'autres programmes.*

 *Si vous êtes sûr de n'utiliser l'`InputVerifier` que pour un seul composant, vous pouvez évidemment utiliser une classe interne voire une classe interne anonyme.*

V - Savoir qui a le focus


Une première manière de faire pour savoir quel est le composant qui détient le focus actuellement est d'utiliser un focus listener. Il suffit alors de créer une classe implémentant FocusListener et d'ajouter ce listener à chacun des composants que vous voulez gérer. Ensuite, il vous suffit d'agir dans les méthodes focusGained et focusLost pour savoir qui obtient et perd le focus. Cette manière de faire est très simple, mais si vous avez beaucoup de composants, elle se révélera vite très lourde à mettre en oeuvre.

Une seconde technique est d'utiliser la classe KeyboardFocusManager. Il vous suffit d'ajouter un PropertyChangeListener sur cette classe et ensuite de vérifier la propriété focusOwner. Vous pouvez aussi écouter beaucoup d'autres choses avec cette classe : le changement de focus sur une fenêtre, le changement de l'ordre de focus, ... Vous pouvez trouver une liste des propriétés de cette classe [ici](#). Cette fois, vous aurez directement tous les changements de focus pour la totalité des composants de votre application. Ceci peut se révéler trop, dans ce cas, utiliser plutôt un focus listener pour des composants spécifiques.

Voici un exemple de listener avec KeyboardFocusManager :

```
KeyboardFocusManager focusManager = KeyboardFocusManager.getCurrentKeyboardFocusManager();
focusManager.addPropertyChangeListener(
    new PropertyChangeListener() {
        public void propertyChange(PropertyChangeEvent e) {
            String properties = e.getPropertyName();
            if (("focusOwner".equals(properties)) && (e.getNewValue() != null)) {
                Component component = (Component)e.getNewValue();
                String name = component.getName();

                System.out.println(name + " a pris le focus");
            }
        }
    }
);
```

 *Si vous voulez limiter le listener à un seul type de composant, il vous suffit par exemple de rajouter "e.getNewValue instanceof VotreClasse" en condition dans la méthode propertyChange, vous n'aurez ainsi que les changements de focus pour les composants de type VotreClasse.*

VI - Gérer l'ordre de focus

Swing a un ordre de focus par défaut déjà très correct et il est normalement suffisant, mais il peut arriver que vous vouliez changer cet ordre, par exemple pour naviguer dans un formulaire. Par défaut, Swing détermine l'ordre de focus par rapport à l'ordre dans lequel vous ajoutez vos composants au content pane.

Avant la version 1.4, vous pouviez définir l'ordre de focus avec la méthode `setNextFocusableComponent` de la classe `JComponent`. Cette méthode est maintenant dépréciée. Il faut donc utiliser la classe `LayoutFocusTraversalPolicy` qui définit l'ordre de focus pour une application Swing.

Pour cela, il faut créer une nouvelle classe étendant `FocusTraversalPolicy` et redéfinir les méthodes `getComponentAfter(Container focusCycleRoot, Component aComponent)`, `getComponentBefore(Container focusCycleRoot, Component aComponent)`, `getDefaultComponent(Container focusCycleRoot)`, `getLastComponent(Container focusCycleRoot)` et `getFirstComponent(Container focusCycleRoot)`.

Voici ce que fait chacune de ces méthodes :

- `getComponentAfter` : Permet d'indiquer quel composant va avoir le focus après le composant passé en paramètre.
- `getComponentBefore` : Permet d'indiquer quel composant va avoir le focus avant le composant passé en paramètre.
- `getDefaultComponent` : Permet d'indiquer quel est le composant par défaut du container auquel on a appliqué la policy. C'est utilisé quand on passe d'un autre cycle de focus à celui-ci
- `getLastComponent` : Permet d'indiquer quel est le dernier composant du container auquel on a appliqué la policy. C'est utilisé quand on passe d'un autre cycle de focus à celui-ci
- `getFirstComponent` : Permet d'indiquer quel est le premier composant du container auquel on a appliqué la policy. C'est utilisé quand on passe d'un autre cycle de focus à celui-ci
- `getInitialComponent` : Permet d'indiquer quel est le composant qui doit avoir le focus quand la fenêtre est rendue visible.

Voilà un petit exemple :

```
public class TestFocusTraversalPolicy extends FocusTraversalPolicy {
    public Component getComponentAfter(Container focusCycleRoot, Component aComponent) {
        if (aComponent.equals(component1)) {
            return component2;
        } else if (aComponent.equals(component2)) {
            return component3;
        } else if (aComponent.equals(component3)) {
            return component4;
        } else if (aComponent.equals(component4)) {
            return component5;
        } else if (aComponent.equals(tf5)) {
            return component1;
        }

        return component1;
    }

    public Component getComponentBefore(Container focusCycleRoot, Component aComponent) {
        if (aComponent.equals(component1)) {
            return component5;
        } else if (aComponent.equals(component2)) {
            return component1;
        } else if (aComponent.equals(component3)) {
            return component2;
        } else if (aComponent.equals(component4)) {
            return component3;
        }
    }
}
```

```

    } else if (aComponent.equals(component5)) {
        return component4;
    }

    return component1;
}

public Component getDefaultComponent(Container focusCycleRoot) {
    return component1;
}

public Component getLastComponent(Container focusCycleRoot) {
    return component5;
}

public Component getFirstComponent(Container focusCycleRoot) {
    return component1;
}
}
    
```

L'inconvénient de cette technique est que les polices doivent connaître tous les composants entrant en jeu. C'est pour cela qu'on utilise très souvent une classe interne, elle aura ainsi directement accès aux composants nécessaires. Voici donc comment faire avec une classe interne :

```

public class Interface {
    // Code de la classe principale

    class TestFocusTraversalPolicy extends FocusTraversalPolicy {
        //Code du traversal policy
    }
}
    
```


Vous pouvez aussi éventuellement utiliser une classe interne anonyme, mais dans ce cas de figure, vous ne pourrez pas l'utiliser pour plusieurs composants.


Pour appliquer cette Policy, il suffit d'utiliser la méthode `setFocusTraversalPolicy` de la classe `Container`.

```

TestFocusTraversalPolicy newPolicy = new TestFocusTraversalPolicy();
maFrame.setFocusTraversalPolicy(newPolicy);
    
```

Vous pouvez bien sûr définir des stratégies pour plusieurs panels et non pas pour toute la frame. Vous pouvez aussi utiliser la même classe pour plusieurs containers, il suffira simplement pour cela d'utiliser le `focusCycleRoot` passé en paramètre des méthodes.

 *Si vous voulez rétablir les propriétés de focus par défaut, il suffit d'utiliser `setFocusTraversalPolicy` avec comme paramètre `null`.*

 *Vous pouvez aussi modifier le `FocusTraversalPolicy` utilisé par défaut par un autre avec la méthode `setDefaultTraversalPolicy` de la classe `KeyboardFocusManager`.*

VI-A - Simplification

Certains d'entre vous pourraient trouver cette méthode assez compliquée et c'est vrai qu'elle n'est pas tout ce qu'il y a de plus intuitif, mais il est facile de simplifier cette classe. Pour cela, vous pouvez par exemple utiliser une `HashMap` déterminant l'ordre des composants. Voici un exemple avec cette technique :

```
import java.awt.Component;
import java.awt.Container;
import java.awt.FocusTraversalPolicy;
import java.util.HashMap;

/**
 * Permet de gérer l'ordre de focus via une position sous forme de int. Le premier composant doit
 * avoir la
 * position 0 et tous les composants doivent se suivre avec un intervalle de 1. Le dernier
 * composant est donc à
 * la position égale au nombre de composants moins 1.
 *
 * @author Baptiste Wicht
 */
public class MapFocusTraversalPolicy extends FocusTraversalPolicy {
    private HashMap<Integer, Component> components = new HashMap<Integer, Component>();
    private HashMap<Component, Integer> positions = new HashMap<Component, Integer>();

    /**
     * Ajoute le composant à l'ordre de focus en lui donnant une position de focus qui déterminera
     * l'ordre dans
     * lequel il sera "focusé".
     *
     * @param component Le composant
     * @param position Sa position dans le cycle de focus
     */
    public void addComponent(Component component, int position){
        components.put(position, component);
        positions.put(component, position);
    }

    @Override
    public Component getComponentAfter(Container parent, Component component) {
        int position = positions.get(component);

        return components.get(position + 1);
    }

    @Override
    public Component getComponentBefore(Container parent, Component component) {
        int position = positions.get(component);

        return components.get(position - 1);
    }

    @Override
    public Component getDefaultComponent(Container parent) {
        return components.get(0);
    }

    @Override
    public Component getFirstComponent(Container parent) {
        return components.get(0);
    }

    @Override
    public Component getLastComponent(Container parent) {
        return components.get(components.size() + 1);
    }
}
```

Ici on a employé 2 maps pour pouvoir récupérer la clé aussi à partir d'un composant. Si vous le pouvez, vous pouvez employer directement la classe **BidiMap** de **Commons Collections**, cette classe permet de faire une relation bidirectionnelle et de récupérer soit la clé depuis la valeur soit le contraire. Vous pouvez aussi tout développer votre propre Map bidirectionnelle.

Une autre technique que vous pourriez employer, c'est faire implémenter à tous vos composants une interface avec une méthode `getFocusPosition()` et jouer là-dessus dans le Policy pour déterminer l'ordre.

VII - La classe KeyboardFocusManager

En plus des utilisations qu'on en a fait plus loin, cette a aussi d'autres utilités.

Premièrement, elle permet de changer de manière programmatique le composant ayant le focus. Vous pouvez utiliser les méthodes `focusXXX` pour faire cela. Par exemple, vous pouvez utiliser `focusNextComponent` pour dire de passer le focus sur le prochain composant. Vous pouvez aussi indiquer quel est le possesseur du focus avec la méthode `setFocusOwner` et faire pareil avec la méthode `setActiveWindow` pour une fenêtre.

Deuxièmement, vous pouvez aussi récupérer la fenêtre active, le composant ayant le focus actuellement et le container dans lequel est le focus. Si ces composants sont dans un contexte différent que dans le thread appelé, il faut utiliser les méthodes `getGlobalXXX` pour les récupérer.

VIII - Conclusion

En conclusion, vous pouvez faire presque ce que vous voulez avec le système de focus. Vous pouvez modifier l'ordre de focus des composants, dire quel composant doit avoir le focus, faire des tests sur vos composants d'entrée, savoir quel composant a le focus à tout moment, ... Mais cela est parfois assez déroutant de prime abord. Une fois que l'on a compris vers quelles classes se tourner pour effectuer nos manoeuvres, cela se révèle assez simple et intuitif.

Si vous voulez plus d'informations sur le système de focus, je vous conseille de lire le **How to use the focus subsystems** de Sun.

Merci à **Zedros** pour toutes ses corrections.

