Mise en œuvre d'un popup menu

Un popup menu (une instance de la classe **javax.swing.JPopupMenu**), aussi appelé menu contextuel, se comporte comme un **JMenu**. On utilise donc exactement les mêmes mécanismes pour construire ces éléments : on va notamment y ajouter des instances de **JMenuItem**. Voici un petit extrait de code permettant d'ajouter un élément de menu dans le menu contextuel.

```
JPopupMenu popupMenu = new JPopupMenu();

JMenuItem mnuUndo = new JMenuItem( "Undo" );
mnuUndo.setIcon( new ImageIcon( "icons/undo.png" ) );
mnuUndo.setMnemonic( 'U' );
mnuUndo.setAccelerator( KeyStroke.getKeyStroke(KeyEvent.VK_Z, KeyEvent.CTRL_DOWN_MASK) );
mnuUndo.addActionListener( this::mnuUndoListener );
popupMenu.add(mnuUndo);
```

Figure 1: Constitution d'un popup menu

Là où réside la différence, c'est dans la manière d'afficher le menu. Dans le cas de la barre de menu, la chose est automatique : le simple fait de cliquer sur le label du menu produira son affichage. En revanche, pour un menu contextuel, il nous faudra rajouter quelques lignes de code pour détecter un clic avec le bouton droit de la souris sur un élément graphique de l'interface et lancer l'affichage du menu contextuel.

Exemple

```
textArea.addMouseListener( new MouseAdapter() {
    @Override public void mousePressed( MouseEvent event ) {
        if ( event.isPopupTrigger() ) {
            System.out.println( "Show popup" );
            popupMenu.show( event.getComponent(), event.getX(),
event.getY() );
        }
    }
} );
```

Figure 2: Affichage d'un menu contextuel

Dans le cas de l'exemple proposé ci-dessus, le menu contextuel s'activera à la suite d'un clic bouton-droit sur une instance de type **javax.swing.JTextArea**. Ce menu contextuel proposera des actions relatives à une édition de texte, bien entendu. Voici un exemple de code permettant d'afficher un menu contextuel.

Et voici le résultat qui sera produit par l'exemple proposé ci-dessous.

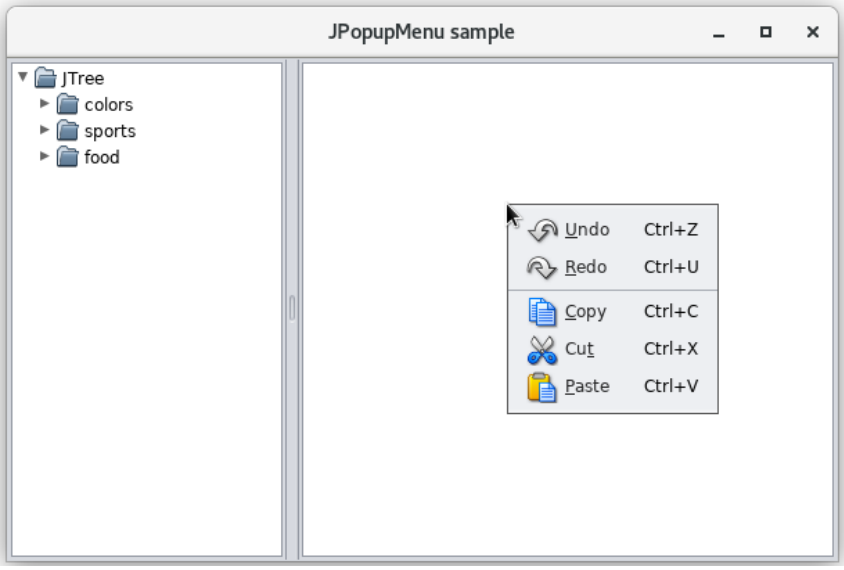

Figure 3: Résultat d'un popup menu

Exemple de code complet

```
import javax.swing.*;
import javax.swing.plaf.nimbus.NimbusLookAndFeel;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.KeyEvent;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;

public class PopupMenuSample extends JFrame {
   /* Construction de l'interface graphique */
   public PopupMenuSample() {
      super( "JPopupMenu sample" );
      this.setSize(600,400);
      this.setLocationRelativeTo( null );
      this.setDefaultCloseOperation( DISPOSE_ON_CLOSE );

      JPanel contentPane = (JPanel) getContentPane();

      // The content of the window
      JScrollPane leftScrollPane = new JScrollPane( new JTree() );
      leftScrollPane.setPreferredSize( new Dimension( 200, 0 ) );

      JTextArea textArea = new JTextArea();
      JScrollPane rightScrollPane = new JScrollPane( textArea );

      JSplitPane splitPane = new JSplitPane(
            JSplitPane.HORIZONTAL_SPLIT, leftScrollPane, rightScrollPane );
      contentPane.add( splitPane /*, BorderLayout.CENTER */ );
      // Association d'un popup menu sur la zone d'édition de texte

      // Attention avant Java SE 8.0, il faut un final au début de la déclaration !!!
      JPopupMenu popupMenu = this.createPopupMenu();
      textArea.addMouseListener( new MouseAdapter() {
```

```
        @Override public void mousePressed( MouseEvent event ) {
            if ( event.isPopupTrigger() ) {
                System.out.println( "Show popup" );
                popupMenu.show( event.getComponent(), event.getX(), event.getY() );
            }
        }
    } );
}
/* Methode de construction du menu contextuel */
private JPopupMenu createPopupMenu() {
    JPopupMenu popupMenu = new JPopupMenu();

    JMenuItem mnuUndo = new JMenuItem( "Undo" );
    mnuUndo.setIcon( new ImageIcon( "icons/undo.png" ) );
    mnuUndo.setMnemonic( 'U' );
    mnuUndo.setAccelerator( KeyStroke.getKeyStroke(KeyEvent.VK_Z, KeyEvent.CTRL_DOWN_MASK) );
    mnuUndo.addActionListener( this::mnuUndoListener );
    popupMenu.add(mnuUndo);

    JMenuItem mnuRedo = new JMenuItem( "Redo" );
    mnuRedo.setIcon( new ImageIcon( "icons/redo.png" ) );
    mnuRedo.setMnemonic( 'R' );
    mnuRedo.setAccelerator( KeyStroke.getKeyStroke(KeyEvent.VK_U, KeyEvent.CTRL_DOWN_MASK) );
    popupMenu.add(mnuRedo);

    popupMenu.addSeparator();

    JMenuItem mnuCopy = new JMenuItem( "Copy" );
    mnuCopy.setIcon( new ImageIcon( "icons/copy.png" ) );
    mnuCopy.setMnemonic( 'C' );
    mnuCopy.setAccelerator( KeyStroke.getKeyStroke(KeyEvent.VK_C, KeyEvent.CTRL_DOWN_MASK) );
    popupMenu.add(mnuCopy);

    JMenuItem mnuCut = new JMenuItem( "Cut" );
    mnuCut.setIcon( new ImageIcon( "icons/cut.png" ) );
    mnuCut.setMnemonic( 't' );
    mnuCut.setAccelerator( KeyStroke.getKeyStroke(KeyEvent.VK_X, KeyEvent.CTRL_DOWN_MASK) );
    popupMenu.add(mnuCut);

    JMenuItem mnuPaste = new JMenuItem( "Paste" );
    mnuPaste.setIcon( new ImageIcon( "icons/paste.png" ) );
    mnuPaste.setMnemonic( 'P' );
    mnuPaste.setAccelerator( KeyStroke.getKeyStroke(KeyEvent.VK_V, KeyEvent.CTRL_DOWN_MASK) );
    popupMenu.add(mnuPaste);


    return popupMenu;
}

private void mnuUndoListener(ActionEvent actionEvent) {
    JOptionPane.showMessageDialog( this, "Undo!" );
}
```

```
    public static void main(String[] args) throws Exception {
        UIManager.setLookAndFeel( new NimbusLookAndFeel() );
        PopupMenuSample frame = new PopupMenuSample();
        frame.setVisible(true);
    }
}
```

Figure 4: Exemple de mise en œuvre d'un popup menu

Utilisation d'actions pour vos éléments de menu

Pourquoi utiliser des actions Swing ?

Dans les trois chapitres précédents (Mise en oeuvre d'une barre de menu, Mise en oeuvre d'une barre d'outils et Mise en oeuvre d'un menu contextuel) nous avons appris à utiliser les principaux éléments de menu.

Le problème est que souvent une même fonctionnalité (dit autrement, une même action) est accessible à partir de la barre de menu, mais aussi à partir de la barre d'outils et enfin à partir d'un menu contextuel. De la manière dont nous avons travaillé, il est nécessaire de respecifier les icons, les accélérateurs, ... pour chaque élément de menu.

Le concept d'action Swing permet, justement, de répondre à cette problématique en factorisant en un seul endroit la définition du traitement de l'action (l'ActionListener), de son icône, son texte, ... Ensuite, on injecte cette action dans la barre de menu, dans la barre d'outils et dans un éventuel menu contextuel.

Comment utiliser des actions Swing ?

Le type de base, pour coder une action Swing, est l'interface **javax.swing.Action** qui étend l'interface **java.awt.event.ActionListener** : il sera donc nécessaire de redéfinir la méthode **actionPerformed**. Pour les autres méthodes abstraites de cette interface, nous n'aurons à les redéfinir. Effectivement, nous utiliserons la classe abstraite **javax.swing.AbstractAction** qui implémente l'interface initialement présentée.

Grace à la méthode **putValue**, vous pourrez associer à l'action une icône, un accélérateur, un mnémonique et son texte. Cela sera fait dans le constructeur de chacune de nos actions. Voici un exemple de définition d'action pour notre élément de menu « New File... ».

```
private AbstractAction actNew = new AbstractAction() {
  {  // Le constructeur de votre classe anonyme
     putValue( Action.NAME, "New File" );
```

```
        putValue( Action.SMALL_ICON, new ImageIcon( "icons/new.png" ) );
        putValue( Action.MNEMONIC_KEY, KeyEvent.VK_N );
        putValue( Action.SHORT_DESCRIPTION, "New file... (CTRL+N)" );       // ToolTipText sur tool bar
        putValue( Action.ACCELERATOR_KEY, KeyStroke.getKeyStroke(KeyEvent.VK_N,
KeyEvent.CTRL_DOWN_MASK ) );
    }

    @Override public void actionPerformed( ActionEvent e ) {
        System.out.println( "New" );
    }
};
```

Figure 5: Exemple de définition d'une action

Rappel :

Nous avons ici affaire à une implémentation d'une classe anonyme. Comme son nom l'indique cette classe n'a donc pas de nom. En conséquence, nous rencontrons une difficulté : comment coder son constructeur ?

Eh bien la syntaxe Java le permet. ***Il suffit de mettre un bloc de code (entre accolades) telle quelle dans votre classe anonyme.*** *Ce bloc sera bien considéré comme étant le constructeur de votre classe anonyme.*

Ensuite, il ne reste plus qu'à utiliser cette action pour créer un élément de menu dans chaque conteneur considéré (barre de menu, barre d'outils ou menu contextuel).

```
JMenuItem mnuNew = mnuFile.add( actNew );
JButton btnNew = toolBar.add( actNew );
JMenuItem mnuNew2 = popupMenu.add( actNew );
```

Figure 6: Utilisation de l'action pour produire les éléments de menu

Voici le résultat qui sera produit par l'exemple proposé ci-dessous.

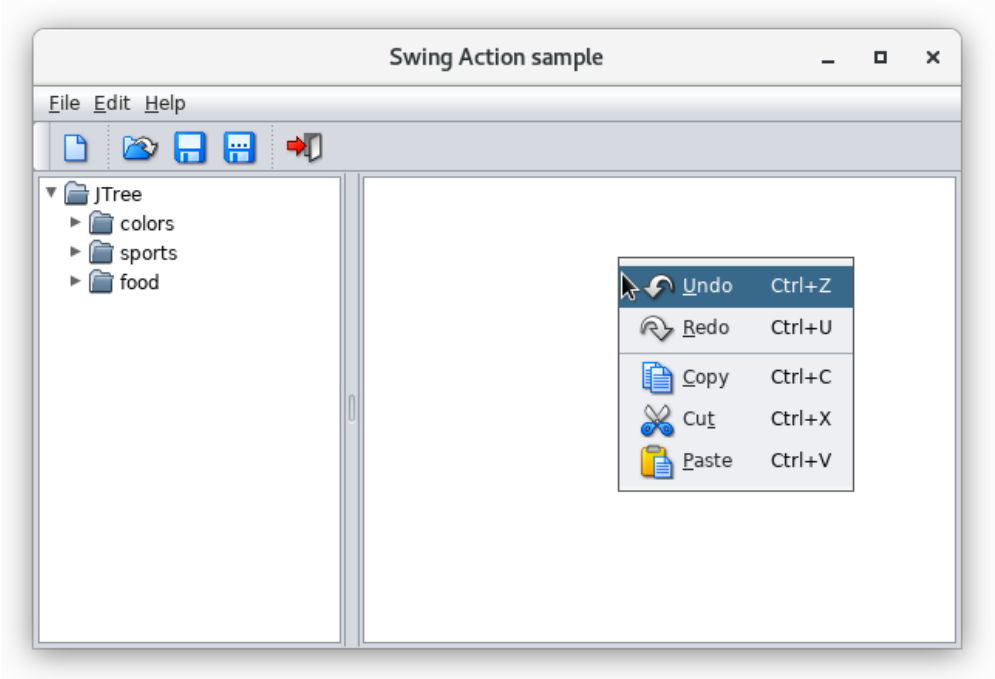

Figure 7: Résultat d'utilisation d'action

Cacher le texte de l'action dans une barre d'outils

Il y a un petit subtil à gérer. Par défaut, une barre d'outils (**JToolBar**) affiche le texte de chaque action, ce qui n'est pas habituelle. Le texte de l'action doit être défini pour que l'on puisse l'afficher dans une barre de menu ou un menu contextuel, mail il faut demander à ne pas l'afficher pour une barre d'outils. Cela se fait grâce à la méthode **setHideActionText**. Deux styles de codage peuvent être employés. Voici un premier style de codage, qualifié de décomposé, qui consiste à récupérer un pointeur sur le bouton de la barre de menu dans une variable, puis d'utiliser ensuite cette variable pour invoquer la méthode **setHideActionText.**

```
JButton btnNew = toolBar.add( actNew );
btnNew.setHideActionText( true );
/* ... do other calls on btnNew */
```

Figure 8:Comment cacher le texte de l'action dans une barre d'outils - Style 1

Cette première technique vous permettra d'invoquer d'autres méthodes sur le bouton, si cela est nécessaire. La seconde technique est pratique si vous n'avez rien d'autre à demander au bouton : dans ce cas, il est inutile de produire une déclaration d'une variable pour un unique appel sur votre bouton. Voilà comment procéder.

```
toolBar.add( actNew ).setHideActionText( true );
```

Figure 9: Comment cacher le texte de l'action dans une barre d'outils - Style 2

Exemple complet

```
import javax.swing.*;
import javax.swing.plaf.nimbus.NimbusLookAndFeel;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.KeyEvent;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;

public class ActionSample extends JFrame {
   /* Construction de l'interface graphique */
   public ActionSample(){
      super("Action Sample Illustration");
      this.setSize(600, 400);
      this.setLocationRelativeTo(null);
      this.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
      // Construction et injection de la barre de menu
      this.setJMenuBar(this.createMenuBar());
      // Construction et injection de la barre d'outils
      JPanel contentPane = (JPanel) getContentPane();
      contentPane.add(this.createToolBar(), BorderLayout.NORTH);
      // Le contenu de la fenêtre
      JScrollPane leftScrollPane = new JScrollPane(new JTree());
      leftScrollPane.setPreferredSize(new Dimension(200,0));
```

```
    JTextArea textArea = new JTextArea();
    JScrollPane rightScrollPane = new JScrollPane( textArea );

    JSplitPane splitPane = new JSplitPane(
            JSplitPane.HORIZONTAL_SPLIT, leftScrollPane, rightScrollPane );
    contentPane.add( splitPane /*, BorderLayout.CENTER */ );

    // Association d'un popup menu sur la zone d'édition de texte
    JPopupMenu popupMenu = this.createPopupMenu();
    textArea.addMouseListener(new MouseAdapter() {
        @Override
        public void mousePressed(MouseEvent event) {
            if(event.isPopupTrigger()){
                popupMenu.show(event.getComponent(), event.getX(), event.getX());
            }
        }
    });
}


/* Methode de construction de la barre de menu */
private JMenuBar createMenuBar() {
    // La barre de menu à proprement parler
    JMenuBar menuBar = new JMenuBar();

    // Définition du menu déroulant "File" et de son contenu
    JMenu mnuFile = new JMenu( "File" );
    mnuFile.setMnemonic( 'F' );
    /*JMenuItem mnuNewFile =*/ mnuFile.add( actNew );
    mnuFile.addSeparator();
    mnuFile.add( actOpen );
    mnuFile.add( actSave );
    mnuFile.add( actSaveAs );
    mnuFile.addSeparator();
    mnuFile.add( actExit );
    menuBar.add(mnuFile);

    // Définition du menu déroulant "Edit" et de son contenu
    JMenu mnuEdit = new JMenu( "Edit" );
    mnuEdit.setMnemonic( 'E' );

    mnuEdit.add( actUndo );
    mnuEdit.add( actRedo );
    mnuEdit.addSeparator();
    mnuEdit.add( actCopy );
    mnuEdit.add( actCut );
    mnuEdit.add( actPaste );

    menuBar.add(mnuEdit);

    // Définition du menu déroulant "Help" et de son contenu
    JMenu mnuHelp = new JMenu( "Help" );
    mnuHelp.setMnemonic( 'H' );
```

```
        menuBar.add( mnuHelp );

        return menuBar;
    }
    /* Methode de construction de la barre d'outils */
    private JToolBar createToolBar() {
        JToolBar toolBar = new JToolBar();

        toolBar.add( actNew ).setHideActionText( true );
        toolBar.addSeparator();
        toolBar.add( actOpen ).setHideActionText( true );
        toolBar.add( actSave ).setHideActionText( true );
        toolBar.add( actSaveAs ).setHideActionText( true );
        toolBar.addSeparator();
        toolBar.add( actExit ).setHideActionText( true );

        return toolBar;

    }
    /* Methode de construction du menu contextuel */
    private JPopupMenu createPopupMenu() {
        JPopupMenu popupMenu = new JPopupMenu();

        popupMenu.add( actUndo );
        popupMenu.add( actRedo );
        popupMenu.addSeparator();
        popupMenu.add( actCopy );
        popupMenu.add( actCut );
        popupMenu.add( actPaste );

        return popupMenu;
    }

    /* Nos diverses actions */
    private AbstractAction actNew = new AbstractAction() {
        {
            putValue( Action.NAME, "New File..." );
            putValue( Action.SMALL_ICON, new ImageIcon( "icons/new.png" ) );
            putValue(Action.MNEMONIC_KEY, KeyEvent.VK_N);
            putValue( Action.SHORT_DESCRIPTION, "New file (CTRL+N)" );
            putValue( Action.ACCELERATOR_KEY, KeyStroke.getKeyStroke(KeyEvent.VK_N,
KeyEvent.CTRL_DOWN_MASK ));

        }
        @Override
        public void actionPerformed(ActionEvent e) {
            System.out.println("Nouveau");
        }
    };

    private AbstractAction actOpen = new AbstractAction() {
        {
```

```
            putValue( Action.NAME, "Open File..." );
            putValue( Action.SMALL_ICON, new ImageIcon( "icons/open.png" ) );
            putValue(Action.MNEMONIC_KEY, KeyEvent.VK_O);
            putValue( Action.SHORT_DESCRIPTION, "Open file (CTRL+O)" );
            putValue( Action.ACCELERATOR_KEY, KeyStroke.getKeyStroke(KeyEvent.VK_O,
KeyEvent.CTRL_DOWN_MASK ));

        }
        @Override
        public void actionPerformed(ActionEvent e) {
            System.out.println("Ouvrir");
        }
    };
    private AbstractAction actSave = new AbstractAction() {
        {
            putValue( Action.NAME, "Save File" );
            putValue( Action.SMALL_ICON, new ImageIcon( "icons/save.png" ) );
            putValue( Action.MNEMONIC_KEY, KeyEvent.VK_S );
            putValue( Action.SHORT_DESCRIPTION, "Save file (CTRL+S)" );
            putValue( Action.ACCELERATOR_KEY, KeyStroke.getKeyStroke(KeyEvent.VK_S,
KeyEvent.CTRL_DOWN_MASK ) );
        }

        @Override public void actionPerformed( ActionEvent e ) {
            System.out.println( "Enrégistrer" );
        }
    };

    private AbstractAction actSaveAs = new AbstractAction() {
        {
            putValue( Action.NAME, "Save As..." );
            putValue( Action.SMALL_ICON, new ImageIcon( "icons/save_as.png" ) );
            putValue( Action.MNEMONIC_KEY, KeyEvent.VK_A );
            putValue( Action.SHORT_DESCRIPTION, "Save file" );
        }

        @Override public void actionPerformed( ActionEvent e ) {
            System.out.println( "Save as" );
        }
    };

    private AbstractAction actExit = new AbstractAction() {
        {
            putValue( Action.NAME, "Exit" );
            putValue( Action.SMALL_ICON, new ImageIcon( "icons/exit.png" ) );
            putValue( Action.MNEMONIC_KEY, KeyEvent.VK_X );
            putValue( Action.SHORT_DESCRIPTION, "Exit (ALT+F4)" );
            putValue( Action.ACCELERATOR_KEY, KeyStroke.getKeyStroke(KeyEvent.VK_F4,
KeyEvent.ALT_DOWN_MASK ) );
        }

        @Override public void actionPerformed( ActionEvent e ) {
            System.out.println( "Exit" );
```

```
    }
  };

  private AbstractAction actUndo = new AbstractAction() {
    {
      putValue( Action.NAME, "Undo" );
      putValue( Action.SMALL_ICON, new ImageIcon( "icons/undo.png" ) );
      putValue( Action.MNEMONIC_KEY, KeyEvent.VK_U );
      putValue( Action.SHORT_DESCRIPTION, "Undo (CTRL+Z)" );
      putValue( Action.ACCELERATOR_KEY, KeyStroke.getKeyStroke(KeyEvent.VK_Z,
KeyEvent.CTRL_DOWN_MASK ) );
    }

    @Override public void actionPerformed( ActionEvent e ) {
      System.out.println( "Undo" );
    }
  };

  private AbstractAction actRedo = new AbstractAction() {
    {
      putValue( Action.NAME, "Redo" );
      putValue( Action.SMALL_ICON, new ImageIcon( "icons/redo.png" ) );
      putValue( Action.MNEMONIC_KEY, KeyEvent.VK_R );
      putValue( Action.SHORT_DESCRIPTION, "Redo (CTRL+U)" );
      putValue( Action.ACCELERATOR_KEY, KeyStroke.getKeyStroke(KeyEvent.VK_U,
KeyEvent.CTRL_DOWN_MASK ) );
    }

    @Override public void actionPerformed( ActionEvent e ) {
      System.out.println( "Redo" );
    }
  };

  private AbstractAction actCopy = new AbstractAction() {
    {
      putValue( Action.NAME, "Copy" );
      putValue( Action.SMALL_ICON, new ImageIcon( "icons/copy.png" ) );
      putValue( Action.MNEMONIC_KEY, KeyEvent.VK_C );
      putValue( Action.SHORT_DESCRIPTION, "Copy (CTRL+C)" );
      putValue( Action.ACCELERATOR_KEY, KeyStroke.getKeyStroke(KeyEvent.VK_C,
KeyEvent.CTRL_DOWN_MASK ) );
    }

    @Override public void actionPerformed( ActionEvent e ) {
      System.out.println( "Copier" );
    }
  };

  private AbstractAction actCut = new AbstractAction() {
    {
      putValue( Action.NAME, "Cut" );
      putValue( Action.SMALL_ICON, new ImageIcon( "icons/cut.png" ) );
      putValue( Action.MNEMONIC_KEY, KeyEvent.VK_T );
```

```
            putValue( Action.SHORT_DESCRIPTION, "Cut (CTRL+X)" );
            putValue( Action.ACCELERATOR_KEY, KeyStroke.getKeyStroke(KeyEvent.VK_X,
KeyEvent.CTRL_DOWN_MASK ) );
        }

        @Override public void actionPerformed( ActionEvent e ) {
            System.out.println( "Couper" );
        }
    };

    private AbstractAction actPaste = new AbstractAction() {
        {
            putValue( Action.NAME, "Paste" );
            putValue( Action.SMALL_ICON, new ImageIcon( "icons/paste.png" ) );
            putValue( Action.MNEMONIC_KEY, KeyEvent.VK_P );
            putValue( Action.SHORT_DESCRIPTION, "Paste (CTRL+V)" );
            putValue( Action.ACCELERATOR_KEY, KeyStroke.getKeyStroke(KeyEvent.VK_V,
KeyEvent.CTRL_DOWN_MASK ) );
        }

        @Override public void actionPerformed( ActionEvent e ) {
            System.out.println( "Coller" );
        }
    };

    public static void main(String[] args) throws Exception {
        UIManager.setLookAndFeel(new NimbusLookAndFeel());
        ActionSample frame = new ActionSample();
        frame.setVisible(true);
    }

}
```