AUTRES COMPOSANTS GRAPHIQUES SWING

1. Utilisation de JRadioButton

L'exemple de code suivant vous montre comment définir de boutons radio et comment les associer à un **ButtonGroup** : ainsi les boutons seront en exclusion mutuel dans le groupe et vous ne pourrez avoir au plus qu'un unique bouton coché.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ItemEvent;

public class JRadioButtons extends JFrame {
   JLabel lblMessage = new JLabel("Choisis ta couleur préférée");

   ButtonGroup buttonGroup = new ButtonGroup();
   JRadioButton rdoRed = new JRadioButton("Rouge");
   JRadioButton rdoGreen = new JRadioButton("Verte");
   JRadioButton rdoBlue = new JRadioButton("Bleue");

   JPanel pnlPreview = new JPanel();
   JButton btnQuit = new JButton("Quitter");

   public JRadioButtons() {
      super( "Exemple d'utilisation de la classe JRadioButton" );

      JPanel contentPane = (JPanel) getContentPane();
      contentPane.add( lblMessage, BorderLayout.NORTH );

      JPanel pnlLeft = new JPanel( new GridLayout( 3, 1 ) );
      // On cherche à imposer la largeur
      pnlLeft.setPreferredSize( new Dimension( 100, 0 ) );

      rdoRed.setSelected( true );
      pnlLeft.add( rdoRed );
      buttonGroup.add( rdoRed );
      rdoRed.addItemListener( this::radioButtons_itemStateChanged );

      pnlLeft.add( rdoGreen );
      buttonGroup.add( rdoGreen );
      rdoGreen.addItemListener( this::radioButtons_itemStateChanged );

      pnlLeft.add( rdoBlue );
      this.buttonGroup.add( rdoBlue );
      rdoBlue.addItemListener( this::radioButtons_itemStateChanged );

      contentPane.add( pnlLeft, BorderLayout.WEST );
```

```
        pnlPreview.setBackground(Color.red);
        contentPane.add( pnlPreview, BorderLayout.CENTER );

        btnQuit.addActionListener( (e) -> dispose() );
        contentPane.add( btnQuit, BorderLayout.SOUTH );

        this.setSize(300,160);
        this.setLocationRelativeTo( null );
    }

    private void radioButtons_itemStateChanged(ItemEvent e) {
        Object source = e.getSource();
        if (source == rdoRed) pnlPreview.setBackground(Color.red);
        if (source == rdoGreen) pnlPreview.setBackground(Color.green);
        if (source == rdoBlue) pnlPreview.setBackground(Color.blue);

    }

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

Figure 1: Exemple d'utilisation de la classe JRadioButton et d'un ButtonGroup

2. Utilisation de cases à cocher

L'exemple de code suivant vous montre comment utiliser les cases à cocher de type **JCheckBox.** Contrairement aux cases à cocher de type **JRadioButton**, vous pourrez cocher autant de cases en simultané que vous le souhaiterez.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ItemEvent;

public class JCheckBoxes extends JFrame {
    JLabel lblMessage = new JLabel("Choisis ta couleur");

    JCheckBox chkRed = new JCheckBox("Rouge");
    JCheckBox chkGreen = new JCheckBox("Verte");
    JCheckBox chkBlue = new JCheckBox("Bleue");

    JPanel pnlPreview = new JPanel();
    JButton btnQuit = new JButton("Quitter");
    public JCheckBoxes() {
        super( "Exemple d'utilisation de la classe JRadioButton" );

        JPanel contentPane = (JPanel) getContentPane();
        contentPane.add( lblMessage, BorderLayout.NORTH );
```

```
        JPanel pnlLeft = new JPanel( new GridLayout( 3, 1 ) );
        // On cherche à imposer la largeur
        pnlLeft.setPreferredSize( new Dimension( 100, 0 ) );

        chkRed.setSelected( true );
        pnlLeft.add( chkRed );
        chkRed.addItemListener( this::radioButtons_itemStateChanged );

        pnlLeft.add( chkGreen );
        chkGreen.addItemListener( this::radioButtons_itemStateChanged );

        pnlLeft.add( chkBlue );
        chkBlue.addItemListener( this::radioButtons_itemStateChanged );

        contentPane.add( pnlLeft, BorderLayout.WEST );

        pnlPreview.setBackground(Color.red);
        contentPane.add( pnlPreview, BorderLayout.CENTER );

        btnQuit.addActionListener( (e) -> dispose() );
        contentPane.add( btnQuit, BorderLayout.SOUTH );

        this.setSize(300,160);
        this.setLocationRelativeTo( null );
    }

    void radioButtons_itemStateChanged(ItemEvent e) {
        int red   = chkRed.isSelected() ? 255 : 0;
        int green = chkGreen.isSelected() ? 255 : 0;
        int blue  = chkBlue.isSelected() ? 255 : 0;
        this.pnlPreview.setBackground(new Color(red, green, blue));
    }

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

Figure 2: Exemple d'utilisation de la classe JCheckBox

3. Utilisation du composant JEditorPane

Comme son nom l'indique, le composant **javax.swing.JEditorPane** permet de créer un éditeur riche pour votre application Java. Cet éditeur supporte le format HTML. En configurant un **JEditorPane** comme étant non éditable, on obtient alors un composant d'affichage de page HTML.

La méthode **setPage** permet de charger un contenu à partir de différentes sources.

- **editor.setPage**("<h1>Contenu au format HTML</h1>"); : le contenu HTML correspond à la chaîne de caractères passée en paramètre.
- **editor.setPage**("file:docs/index.html"); : le contenu HTML correspond à un fichier accessible à partir du répertoire de travail courant. Dans l'exemple proposé, le répertoire courant doit contenir un dossier docs qui contient lui-même le fichier index.html.
- **editor.setPage**("http://defiteck.fr"); : le contenu HTML correspond à l'URL (Uniform Resource Locator) passée en paramètre.

Exemple

```
import javax.swing.*;
import javax.swing.plaf.nimbus.NimbusLookAndFeel;
import java.awt.*;

public class JEditorPaneSample extends JFrame {
   // --- Construction de l'interface graphique ---
   public JEditorPaneSample() throws Exception {
      super( "JEditorPane sample" );
      this.setSize( 800, 500 );
      this.setLocationRelativeTo( null );
      this.setDefaultCloseOperation( DISPOSE_ON_CLOSE );

      // --- On crée le composant d'affichage JEditorPane ---
      JEditorPane helpPane = new JEditorPane();
      helpPane.setEditable( false );
      helpPane.setPage( "file:docs/index.html" );
      JScrollPane scrollPane = new JScrollPane( helpPane );

      // --- On récupère le contentPane et on y ajoute notre composant ---
      JPanel contentPane = (JPanel) getContentPane();
      contentPane.add( scrollPane, BorderLayout.CENTER );
   }

   // --- Point d'entrée du programme ---
   public static void main(String[] args) throws Exception {
      UIManager.setLookAndFeel( new NimbusLookAndFeel() );
      JEditorPaneSample frame = new JEditorPaneSample();
      frame.setVisible(true);
   }
}
```

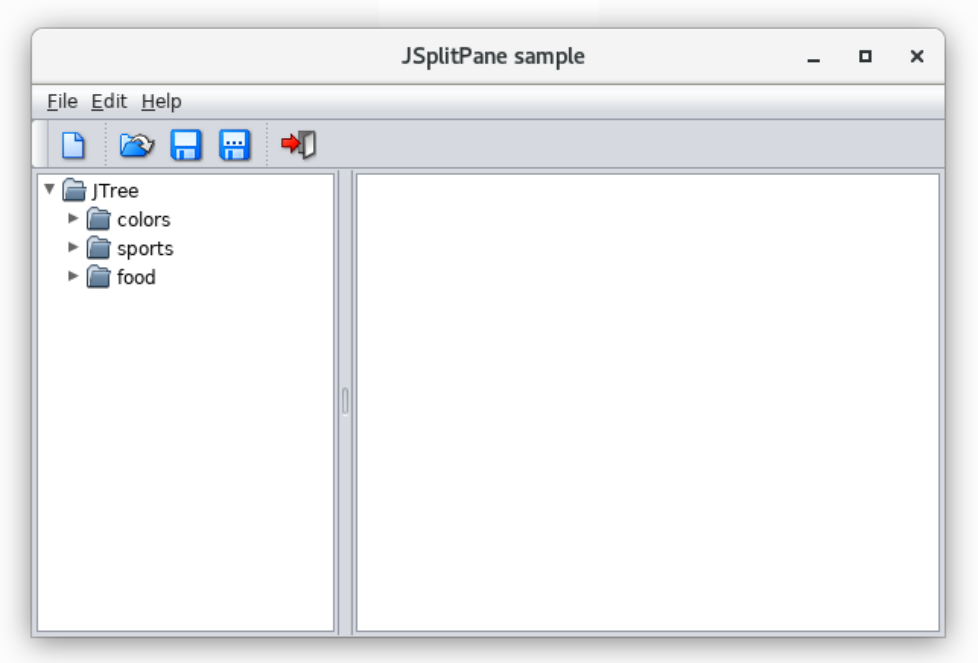
Figure 3: Exemple d'utilisation d'un composant JTabbedPane

4. Utilisation de composants JSplitPane

Aspects fondamentaux

Un composant de type **JSplitPane** est un conteneur graphique Swing qui divise son espace en deux sous-espaces : dans chacun de ces sous-espaces, vous pourrez-y placer un autre composant graphique. Le **JSplitPane** ajoute un séparateur entre les deux composants. En plaçant la souris sur ce séparateur, puis en maintenant le bouton gauche de la souris enfoncé sur ce séparateur, vous pourrez déplacer le séparateur et donc contrôler la superficie occupée par chacun des deux composants du **JSplitPane**.

Voici une capture d'écran qui montre un exemple d'utilisation du composant **JSplitPane** séparant une arborescence (à gauche) d'un éditeur de texte (à droite).



Un **JSplitPane** peut orienter les composants qu'il contient de deux manières différentes : soit horizontalement (**JSplitPane.HORIZONTAL_SPLIT**), soit verticalement (**JSplitPane.VERTICAL_SPLIT**). Dans l'exemple ci-dessus, c'est bien entendu une séparation horizontale qui a été retenue. L'orientation est spécifiée à la construction de l'instance de type **JSplitPane**.

Agrégation des composants d'un JSplitPane

Deux techniques peuvent être utilisées pour alimenter en composants un **JSplitPane :**

- soit par construction du **JSplitPane**

- soit après construction en appelant une des méthodes suivantes : **setTopComponent, setBottomComponent, setLeftComponent, setRightComponent**. Il faudra, bien entendu, tenir compte de l'orientation du **JSplitPane**

La plupart du temps, c'est la première technique qui est utilisée. Voici un exemple d'appel d'un constructeur de la classe **JSplitPane.**

```
JSplitPane mainSplitPane = new JSplitPane(
JSplitPane.HORIZONTAL_SPLIT, new JTree(), new JTextArea() );
```

Il faut savoir que par défaut, le composant du haut (ou de gauche en fonction de l'orientation) sera dimensionné à sa taille minimale en fonction de son contenu. Le second composant prendra l'espace restant. Si cela ne vous convient pas, vous pouvez fixer une taille préférée (**setPreferredSize** sur le premier composant). Une autre solution consiste à fournir un ratio de taille pour le premier composant par rapport à la taille totale du **JSplitPane** : pour ce faire, vous devez invoquer la méthode **setResizeWeight**. Dans l'exemple suivant, l'arborescence utilisera un tier de l'espace disponible, laissant ainsi les deux autres tiers à la zone d'édition de texte.

```
JSplitPane mainSplitPane = new JSplitPane(
    JSplitPane.HORIZONTAL_SPLIT, new JTree(), new JTextArea() );
mainSplitPane.setResizeWeight( 0.33 );
```

On peut aussi controller la taille du séparateur de zone, affiché par le composant **JSplitPane**. N'oubliez pas qu'en plaçant la souris sur ce séparateur, puis en y maintenant le bouton gauche de la souris enfoncé, vous pourrez déplacer le séparateur et donc contrôler la superficie occupée par chacun des deux composants du **JSplitPane**. Pour changer la taille du séparateur, invoquez la méthode **setDividerSize**

```
JSplitPane mainSplitPane = new JSplitPane(
    JSplitPane.HORIZONTAL_SPLIT, new JTree(), new JTextArea() );
mainSplitPane.setResizeWeight( 0.33 );
mainSplitPane.setDividerSize( 100 );
```

Pour aligner plus que deux éléments

Une difficulté apparait si vous cherchez à séparer plus de deux composants (horizontalement ou vericalement). Effectivement un composant **JSplitPane** ne peut contenir que deux sous-composants.

Alors comment séparer 3 composants ou plus ? C'est simple, en utilisant plusieurs **JSplitPane**. Voici un petit exemple simple.

```
JSplitPane secondSplitPane = new JSplitPane(
        JSplitPane.HORIZONTAL_SPLIT, new JTextArea(), new JTree() );
secondSplitPane.setResizeWeight( 0.5 );
JSplitPane firstSplitPane = new JSplitPane(
        JSplitPane.HORIZONTAL_SPLIT, new JTree(), secondSplitPane );
firstSplitPane.setResizeWeight( 0.33 );
```

Aligner plus que deux composants normalement, les poids de retaillage devraient garantir que chaque composants (**JTree** ou **JTextArea**) devrait occuper un tier de l'espace totale de la fenêtre.

L'exemple de code complet

```
import javax.swing.*;
import javax.swing.plaf.nimbus.NimbusLookAndFeel;
import java.awt.*;

public class JSplitPaneSample extends JFrame {
   // --- Construction de l'interface graphique ---
   public JSplitPaneSample() {
      super( "JSplitPane sample" );
      this.setSize(800,500);
      this.setLocationRelativeTo( null );
      this.setDefaultCloseOperation( DISPOSE_ON_CLOSE );

      // --- On récupère le contentPane ---
      JPanel contentPane = (JPanel) getContentPane();


      // --- L'explorateur de projet et ses scrollbars ---
      JTree projectExplorerTree = new JTree();
      JScrollPane projectScrollPane = new JScrollPane( projectExplorerTree );
      projectScrollPane.setPreferredSize( new Dimension( 200, 0 ) );

      // --- L'éditeur principal et ses scrollbars ---
      JTextArea textArea = new JTextArea();
      JScrollPane editorScrollPane = new JScrollPane( textArea );

      // --- Le panneau de propriétés et ses scrollbars ---
      JTable propertyPane = new JTable();
      JScrollPane propertyScrollPane = new JScrollPane( propertyPane );
```

```
        // --- La vue outline et ses scrollbars ---
        JTree outlineTree = new JTree();
        JScrollPane outlineScrollPane = new JScrollPane( outlineTree );


        // --- Et maintenant on assemble le tout avec de JSplitPane ---
        JSplitPane documentSplitPane = new JSplitPane(
                JSplitPane.HORIZONTAL_SPLIT, editorScrollPane, outlineScrollPane );
        documentSplitPane.setResizeWeight( 0.8 );

        JSplitPane rightSplitPane = new JSplitPane(
                JSplitPane.VERTICAL_SPLIT, documentSplitPane, propertyScrollPane );
        rightSplitPane.setResizeWeight( 0.8 );

        JSplitPane mainSplitPane = new JSplitPane(
                JSplitPane.HORIZONTAL_SPLIT, projectScrollPane, rightSplitPane );
        contentPane.add( mainSplitPane /*, BorderLayout.CENTER */ );

        // Pour changer la taille d'un séperateur de panneaux
        // mainSplitPane.setDividerSize( 100 );        // Bof

        // Pour changer un composant dans le JSplitPane après sa construction
        // setTopComponent, setBottomComponent, setLeftComponent, setRightComponent
    }

    // --- Point d'entrée du programme ---
    public static void main(String[] args) throws Exception {
        UIManager.setLookAndFeel( new NimbusLookAndFeel() );
        JSplitPaneSample frame = new JSplitPaneSample();
        frame.setVisible(true);
    }
}
```

5. Utilisation du composant JTabbedPane

Le composant **javax.swing. JTabbedPane** permet d'instancier un composant de type conteneur d'onglets. Ce type de composants peut contenir plusieurs sous-composants, que vous insérerez en invoquant l'une des méthodes **addTab** (il y a plusieurs signatures disponibles : avec ou sans icône, ...). Un seul des sous-composants sera visible à un instant donné et vous pourrez choisir lequel sera affiché en avant-plan en cliquant sur l'onglet qui lui est associé.

La barre d'onglets, permettant la sélection du composant placé en avant-plan, est affichée par défaut en haut du conteneur. Mais vous pouvez choisir de la placer sur le bord de votre choix en invoquant la méthode **setTabPlacement.**

Exemple complet

```
import javax.swing.*;
import javax.swing.plaf.nimbus.NimbusLookAndFeel;
import java.awt.*;

public class JTabbedPaneSample extends JFrame {
   // --- Construction de l'interface graphique ---
   public JTabbedPaneSample() throws Exception {
      super( "JTabbedPane sample" );
      this.setSize( 800, 500 );
      this.setLocationRelativeTo( null );
      this.setDefaultCloseOperation( DISPOSE_ON_CLOSE );

      // --- On crée le conteneur d'onglets (partie gauche) ---
      JTabbedPane tabs = new JTabbedPane();
      tabs.setPreferredSize( new Dimension( 260, 0 ) );
      tabs.setTabPlacement( JTabbedPane.BOTTOM );

      // --- Premier onglet et son composant associé ---
      tabs.addTab( "Explorer", new JScrollPane( new JTree() ) );

      // --- Second onglet et son composant associé ---
      JEditorPane helpPane = new JEditorPane();
      helpPane.setEditable( false );
      helpPane.setPage( "file:docs/index.html" );
      tabs.addTab( "Help", new JScrollPane( helpPane ) );

      // --- On crée un éditeur (partie droite) ---
      JTextArea editor = new JTextArea();
      JScrollPane scrollEditor = new JScrollPane( editor );

      // -- On assemble la partie gauche et la partir droite dans un splitter ---
      JSplitPane splitter = new JSplitPane( JSplitPane.HORIZONTAL_SPLIT, tabs, scrollEditor );

      // --- On récupère le contentPane et on y ajoute le splitter ---
      JPanel contentPane = (JPanel) getContentPane();
      contentPane.add( splitter, BorderLayout.CENTER );
   }

   // --- Point d'entrée du programme ---
   public static void main(String[] args) throws Exception {
      UIManager.setLookAndFeel( new NimbusLookAndFeel() );
```

```
        JTabbedPaneSample frame = new JTabbedPaneSample();
        frame.setVisible(true);
    }
}
```

6. Utilisation d'un JDesktopPane

Cet exemple de code vous montre comment utiliser la classe **JDestopPane** : il s'agit d'un conteneur MDI (Multiple Document Interface). Vous pourrez ouvrir, à l'intérieur de ce conteneur des sous fenêtres de type **JInternalFrame**.

```
import javax.swing.*;
import javax.swing.plaf.nimbus.NimbusLookAndFeel;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.beans.PropertyVetoException;

public class MdiFrame extends JFrame {
    private JDesktopPane desktopPane = new JDesktopPane();


    /* Construction de l'interface graphique */
    public MdiFrame() {
        super( "JDesktopPane / JInternalFrame sample" );
        this.setSize(600,400);
        this.setLocationRelativeTo( null );
        this.setDefaultCloseOperation( DISPOSE_ON_CLOSE );

        // Mise en place du conteneur de sous-fenêtres
        desktopPane.setBackground(Color.gray);
        JPanel contentPane = (JPanel)this.getContentPane();
        contentPane.add(desktopPane, BorderLayout.CENTER);

        // Construction de la barre de menu
        this.createMenuBar();

        // Creation d'une sous-fenêtre
        JInternalFrame firstWindow = new JInternalFrame("Première fenêtre");
        firstWindow.setSize(300,200);
        firstWindow.setVisible(true);
        firstWindow.setResizable(true);
        desktopPane.add( firstWindow );

        // Creation d'une autre sous-fenêtre sans les boutons
        JInternalFrame secondWindow = new JInternalFrame("Seconde fenêtre");
        secondWindow.setSize(300,200);
```

```
        secondWindow.setVisible(true);
        secondWindow.setResizable(true);
        secondWindow.setClosable( true );
        secondWindow.setIconifiable( true );
        secondWindow.setMaximizable( true );
        secondWindow.setLocation(20,20);
        desktopPane.add( secondWindow );

        // Pass the second frame to front
        try {
            secondWindow.moveToFront();
            secondWindow.setSelected( true );
        } catch ( PropertyVetoException exception ) {
            exception.printStackTrace();
        }

        // Autres Décorations
        contentPane.add(new JTree(), BorderLayout.WEST);
        contentPane.add(new JLabel("La barre de status"), BorderLayout.SOUTH);
    }


    /* Methode de construction de la barre de menu */
    private void createMenuBar() {
        JMenuBar menuBar = new JMenuBar();

        JMenu mnuFile = new JMenu( "File" );
        JMenuItem mnuNewFile = new JMenuItem( "New File" );
        mnuFile.add(mnuNewFile);
        JMenuItem mnuOpenFile = new JMenuItem( "Open File ..." );
        mnuFile.add(mnuOpenFile);
        JMenuItem mnuSaveFile = new JMenuItem( "Save File ..." );
        mnuFile.add(mnuSaveFile);
        JMenuItem mnuSaveFileAs = new JMenuItem( "Save File As ..." );
        mnuFile.add(mnuSaveFileAs);
        menuBar.add(mnuFile);

        JMenu mnuEdit = new JMenu( "Edit" );
        JMenuItem mnuUndo = new JMenuItem( "Undo" );
        mnuEdit.add(mnuUndo);
        JMenuItem mnuRedo = new JMenuItem( "Redo" );
        mnuEdit.add(mnuRedo);
        mnuEdit.addSeparator();
        JMenuItem mnuCopy = new JMenuItem( "Copy" );
        mnuEdit.add(mnuCopy);
        JMenuItem mnuCut = new JMenuItem( "Cut" );
        mnuEdit.add(mnuCut);
        JMenuItem mnuPaste = new JMenuItem( "Paste" );
        mnuEdit.add(mnuPaste);
        menuBar.add(mnuEdit);
```

```
    JMenu mnuWindow = new JMenu( "Window" );
    JMenuItem mnuCascade = new JMenuItem( "Cascade" );
    mnuCascade.addActionListener( this::mnuCascadeListener );
    mnuWindow.add(  mnuCascade );
    JMenuItem mnuTileHorizontaly = new JMenuItem( "Tile horizontaly" );
    mnuTileHorizontaly.addActionListener( this::mnuTileHorizontalyListener );
    mnuWindow.add(  mnuTileHorizontaly );
    JMenuItem mnuTileVerticaly = new JMenuItem( "Tile verticaly" );
    mnuTileVerticaly.addActionListener( this::mnuTileVerticalyListener );
    mnuWindow.add(  mnuTileVerticaly );
    JMenuItem mnuIconifyAll = new JMenuItem( "Iconify all" );
    mnuIconifyAll.addActionListener( this::mnuIconifyAll );
    mnuWindow.add(  mnuIconifyAll );
    menuBar.add( mnuWindow );

    JMenu mnuHelp = new JMenu( "Help" );
    menuBar.add( mnuHelp );

    this.setJMenuBar( menuBar );
}

/* Permet de  cascader toutes les fenêtres internes */
private void mnuCascadeListener( ActionEvent event ) {
    JInternalFrame internalFrames[] = desktopPane.getAllFrames();
    for (int i = 0; i < internalFrames.length; i++) {
        internalFrames[i].setBounds( i*20, i*20, 300, 200 );
    }
}

/* Permet d'aligner horizontalement toutes les fenêtres internes */
private void mnuTileHorizontalyListener( ActionEvent event ) {
    JInternalFrame internalFrames[] = desktopPane.getAllFrames();
    int frameWidth = desktopPane.getWidth() / internalFrames.length;
    for (int i = 0; i < internalFrames.length; i++) {
        internalFrames[i].setBounds( i*frameWidth, 0, frameWidth, desktopPane.getHeight() );
    }
}

/* Permet d'aligner verticalement toutes les fenêtres internes */
private void mnuTileVerticalyListener( ActionEvent event ) {
    JInternalFrame internalFrames[] = desktopPane.getAllFrames();
    int frameHeight = desktopPane.getHeight() / internalFrames.length;
    for (int i = 0; i < internalFrames.length; i++) {
        internalFrames[i].setBounds( 0, i*frameHeight, desktopPane.getWidth(), frameHeight );
    }
}

/* Permet d'iconifier toutes les fenêtres internes */
private void mnuIconifyAll( ActionEvent event ) {
```

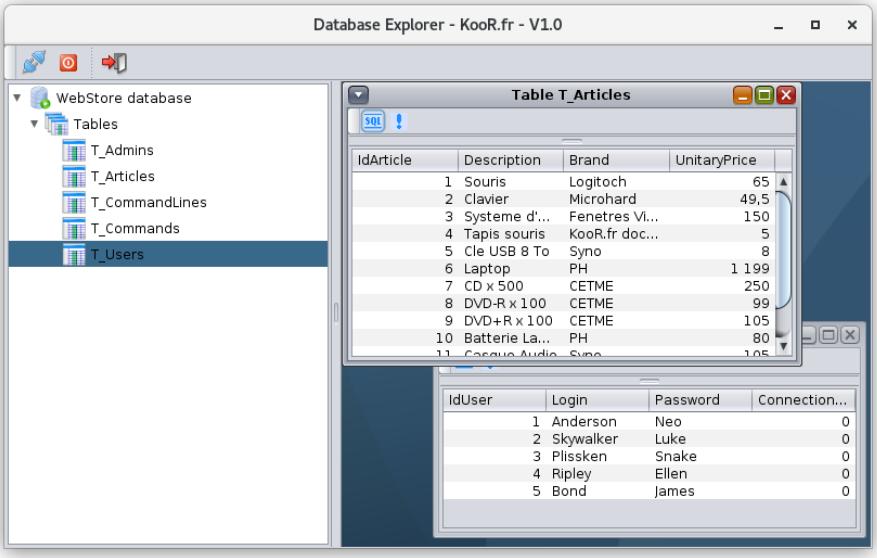
```
        JInternalFrame internalFrames[] = desktopPane.getAllFrames();
        for (int i = 0; i < internalFrames.length; i++) {
            try {
                internalFrames[i].setIcon( true );
            } catch ( PropertyVetoException exception ) {
                exception.printStackTrace();
            }
        }
    }


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

7. Coder un explorateur de bases de données SQL

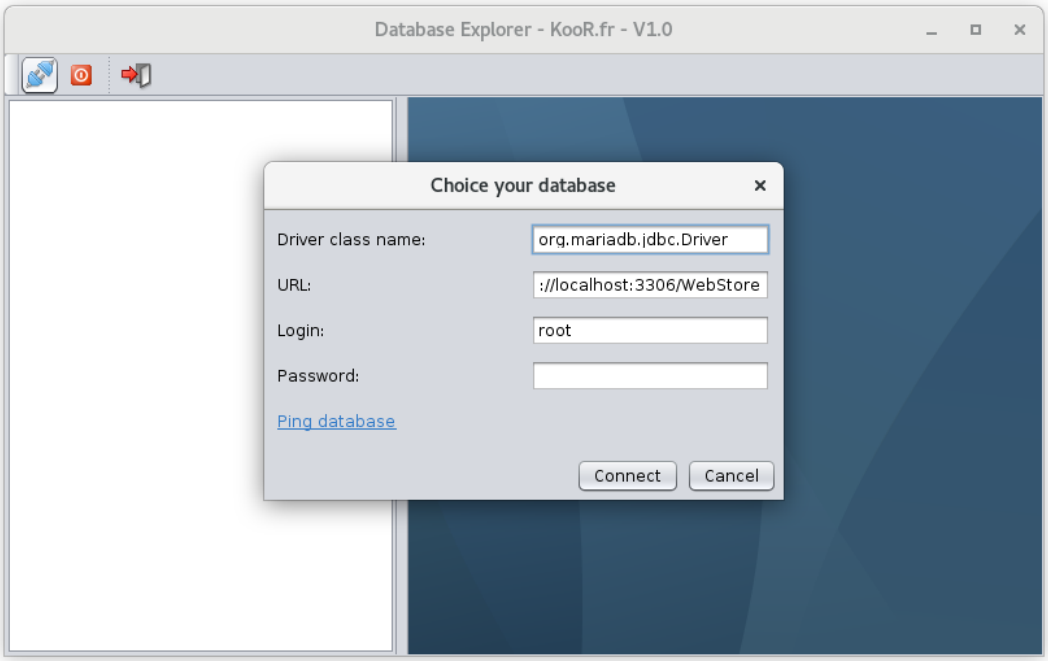
Présentation de l'application DatabaseExplorer

Il s'agit donc d'une application Java/Swing permettant de se connecter à une base de données et d'afficher les données stockées dans les tables de la base de données. Bien entendu, il s'agit d'un prototype et il vous appartient de poursuivre le développement de l'application. Voici une capture d'écran de l'application.



Une boite de dialogue pour configurer les informations de connexion à la base de données.

Il s'agit d'une boîte de dialogue permettant de saisir les quatre informations nécessaires à une connexion JDBC : le nom du driver, l'URL de connexion, le login et le mot de passe. Voici une capture d'écran de cette boîte de dialogue.



On intègre les composants graphiques requis dans une fenêtre

Nous allons donc utiliser deux principaux composants d'affichage de données à partir d'une base. Le premier sera basé sur la classe **swing.jdbc.KDatabaseTree** et permettra d'afficher les tables présentes dans la base de données sélectionnée. Le second, basé sur la classe **swing.jdbc.KSqlQueryTable**, permettra d'afficher les données stockées dans une table.