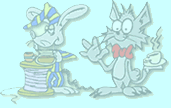
<http://remy-manu.no-ip.biz/Java/Tutoriels/IHM/dialogue.html>

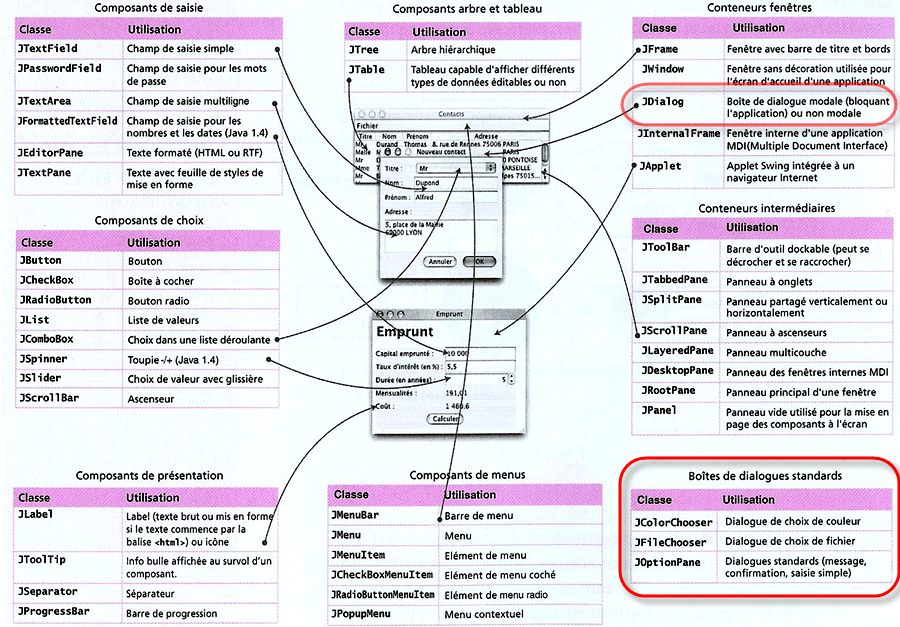
Les boîtes de dialogue

[[http://remy-manu.no-ip.biz/Java/Tutoriels/icones/BoutonAccueil.gif](http://remy-manu.no-ip.biz/index.html)](http://remy-manu.no-ip.biz/index.html)Chapitres traités

Jusqu'ici, nous avons appris à placer des composants dans une fenêtre. En général, ces composants restent affichés en permanence. Or, pour certaines applications, nous avons besoin d'établir un dialogue temporaire avec l'utilisateur. Nous pourrions certes y parvenir en exploitant les possibilités d'ajout ou de suppression dynamique de composants, ou éventuellement en recourant à la méthode setVisible(false). Mais la boîte de dialogue offre une solution beaucoup plus adaptée.

Elle permet en effet de regrouper n'importe quels composants dans une sorte de fenêtre que nous faisons apparaître ou disparaître globalement. Par ailleurs, l'utilisateur est parfaitement habitué à son ergonomie, dans la mesure où elle est très utilisée dans les logiciels du commerce.

Au travers de cette étude, nous allons apprendre à créer et à exploiter de telles boîtes de dialogue en y introduisant les composants de notre choix. Auparavant, nous présenterons quelques boîtes de dialogue standard fournies par Java et qui pourront nous simplifier la tâche dans certaines situations simples : affichage d'un message, demande de confirmation, choix d'une option dans une liste, saisie d'un texte, sélecteur de fichier et sélecteur de couleur.



**Principe général d'une boîte de dialogue**

Â l'image de la plupart des systèmes de fenêtrage, AWT fait la distinction entre les boîtes de dialogue modales et non modales :

1. Une boîte de dialogue modale ne permet pas à l'utilisateur d'interagir avec d'autres fenêtres de l'application tant quelle demeure ouverte. Vous l'utilisez lorsque vous avez besoin de récupérer des informations de la part de l'utilisateur avant de poursuivre l'exécution du programme. Par exemple, quand l'utilisateur demande à lire un fichier pour que le programme puisse entreprendre une opération de lecture. L'application ne peut continuer à s'exécuter que lorsque l'utilisateur a fermé la boîte de dialogue.
2. Une boîte de dialogue non modale autorise l'utilisateur à interagir avec les autres fenêtres de l'application alors qu'elle est ouverte. La boîte de dialogue Rechercher/Remplacer en est un exemple. Elle peut rester apparente aussi longtemps que nécessaire pendant que l'utilisateur manipule les différentes fenêtres.

Nous allons commencer l'étude par une boîte de dialogue modale simple contenant éventuellement au choix : un seul message, une zone de saisie, un ensemble d'options ou une confirmation. Swing dispose d'une classe très utile pour cela, JOptionPane, qui vous permet d'afficher une boîte de dialogue standard sans avoir à écrire de code spécifique.

Il existe également deux autres boîtes de dialogues plus sophistiquées :

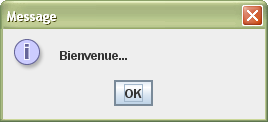
1. JFileChooser : qui permet de gérer et de choisir un ou plusieurs fichiers.
2. JColorChooser : qui permet à un utilisateur de choisir une couleur à partir d'une palette.

Nous verrons ensuite comment créer des boîtes de dialogue plus complexe, à partir de la classe JDialog, en implémentant les notres de façon personnalisées. Nous verrons ainsi comment échanger des données entre une application et une boîte de dialogue.

**Boîtes de dialogue d'options - JOptionPane**

Swing propose un ensemble de boîtes de dialogues standard, prêtes à l'emploi, qui permettent de demander à l'utilisateur de fournir une information. Swing a prévu un jeu de dialogues préfabriqués, accessibles à partir de méthodes statiques de la classe JOptionPane. De nombreuses variantes sont possibles ; JOptionPane les classe en quatre groupes de base :

Dialogue de message : showMessageDialog() - Affiche un message et attend que l'utilisateur clique sur OK (validation de lecture du message).



import javax.swing.\*;

public class Dialogue {

public static void main(String[] args) {

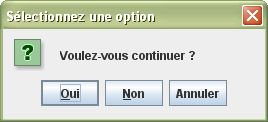
JOptionPane.showMessageDialog(null, "Bienvenue...");

}

}

Cette boîte de dialogue contient les composants suivant : une icône, un message et un bouton OK.  
§

Dialogue de confirmation : showConfirmDialog() - Affiche un message et attend une confirmation de la part de l'utilisateur. Les boutons de réponse sont généralement Oui, Non et Annuler.



import javax.swing.\*;

public class Dialogue {

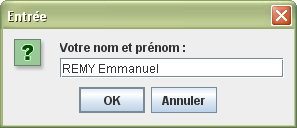
public static void main(String[] args) {

JOptionPane.showConfirmDialog(null, "Voulez-vous continuer ?");

}

}

Dialoque de saisie : showInputDialog() - Affiche un message et récupère une valeur saisie par l'utilisateur.



import javax.swing.\*;

public class Dialogue {

public static void main(String[] args) {

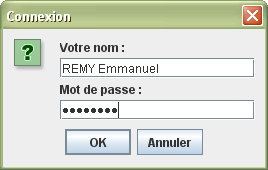
JOptionPane.showInputDialog(null, "Votre nom et prénom :");

}

}

La boîte de dialogue de saisie comprend un composant supplémentaire pour recueillir l'entrée de l'utilisateur. Il peut s'agir d'un champ de texte dans lequel il tape une chaîne quelconque, ou d'une zone de liste déroulante dans laquelle il sélectionne une valeur.

Dialogue d'option : showOptionDialog() - Le type le plus général, vous lui passez vos propres composants, qui se retrouvent affichés dans le dialogue, comme des boîtes combo, un ensemble de boutons, un ensemble d'objets hétérogènes.



import javax.swing.\*;

public class Dialogue {

public static void main(String[] args) {

JTextField utilisateur = new JTextField();

JPasswordField passe = new JPasswordField();

JOptionPane.showOptionDialog(null,

new Object[] {"Votre nom :", utilisateur, "Mot de passe :", passe},

"Connexion",

JOptionPane.OK\_CANCEL\_OPTION,

JOptionPane.QUESTION\_MESSAGE, null, null, null);

}

}

Avec la méthode showOptionDialog(), vous pouvez spécifier un ensemble arbitraire d'options en fournissant un tableau d'objets. Chaque élément du tableau est traité de la façon suivante :

1. String : Un bouton est créé avec la chaîne comme libellé.
2. Icon : Un bouton est créé avec l'icône comme libellé.
3. Component : Le composant est affiché.
4. tout autre objet : La méthode toString() est appliquée et un bouton est créé avec pour libellé la chaîne résultante.

Les dialogues affichés par JOptionPane sont modaux, ce qui signifie, nous l'avons déjà vu, qu'ils bloquent les autres entrées de l'application lorsqu'ils sont affichés.

**Les différents types d'icône**

L'icône placée à gauche dans la boîte de dialogue standard dépend du type de message. Il en existe cinq :

* http://remy-manu.no-ip.biz/Java/Tutoriels/IHM/dialogue/erreur.gifErreur : JOptionPane.ERROR\_MESSAGE.
* http://remy-manu.no-ip.biz/Java/Tutoriels/IHM/dialogue/info.gifInformation : JOptionPane.INFORMATION\_MESSAGE.
* http://remy-manu.no-ip.biz/Java/Tutoriels/IHM/dialogue/warning.gifAvertissement : JOptionPane.WARNING\_MESSAGE.
* http://remy-manu.no-ip.biz/Java/Tutoriels/IHM/dialogue/question.gifQuestion : JOptionPane.QUESTION\_MESSAGE.
* Aucune icône : JOptionPane.PLAIN\_MESSAGE.

Le type PLAIN\_MESSAGE ne possède pas d'icône. Pour chaque type de boîte de dialogue, il existe la méthode setIcon() vous permettant de fournir votre propre icône.

**Choisir son type de message**

Vous pouvez spécifier un message pour chaque type de boîte de dialogue. Ce message peut être une chaîne, une icône, un composant d'interface utilisateur ou tout autre objet. Voici comment est affiché l'objet message suivant son type :

1. String : La chaîne est affichée.
2. Icon : L'icône est affichée.
3. JComponent : Le composant est affiché.
4. Object[] : Tous les objets du tableau sont affichés les uns au-dessus des autres.
5. tout autre objet : La méthode toString() est appliquée et la chaîne résultante est affichée.

Bien sûr, le message le plus fréquent est celui de type chaîne. Un objet Component offre une souplesse optimale puisque vous pouvez redéfinir la méthode paintComponent() pour dessiner tout ce que vous souhaitez.

Astuce : Une chaîne de message peut contenir des caractères de fin de ligne ('\n'). L'affichage est alors réparti sur plusieurs lignes.  
§

**Les différents types de bouton**

Les boutons situés en bas de la boîte de dialogue dépendent de son type et du type d'option. Les méthodes showMessageDialog() et showInputDialog() implémentent uniquement un jeu standard de boutons (respectivement OK et OK/Cancel).

La méthode showConfirmMessage() vous permet de choisir parmi les quatres options suivantes :

* JOptionPane.DEFAULT\_OPTION.
* JOptionPane.YES\_NO\_OPTION.
* JOptionPane.YES\_NO\_CANCEL\_OPTION.
* JOptionPane.OK\_CANCEL\_OPTION.

**Valeurs renvoyées par les méthode statiques**

* showMessageDialog() : aucune.
* showConfirmDialog() : un entier représentant l'option choisie.
* showInputDialog() : la chaîne entrée ou sélectionnée par l'utilisateur.
* showOptionDialog() : un entier représentant l'option choisie.

Les méthodes showConfirmDialog() et showOptionDialog() renvoient des entiers pour indiquer le bouton que l'utilisateur a sélectionné. Pour une boîte de dialogue d'options, il s'agit de l'indice de l'élément choisi ou de la valeur CLOSED\_OPTION si l'utilisateur a fermé la boîte de dialogue au lieu de faire une sélection. Pour une boîte de dialogue de confirmation, la valeur renvoyée peut être l'une des suivantes :

* JOptionPane.OK\_OPTION.
* JOptionPane.CANCEL\_OPTION.
* JOptionPane.YES\_OPTION.
* JOptionPane.NO\_OPTION.
* JOptionPane.CLOSED\_OPTION.

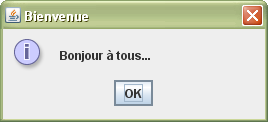
**Création d'une instance de boîte de dialogue**

Au lieu d'utiliser l'une des méthodes statiques pour afficher un dialogue JOptionPane, nous pouvons aussi créer directement un objet (de type JOptionPane) avec les propriétés désirées. Ces réglages sont ainsi conservés dans le temps. Une fois que vous avez créé votre objet en choisissant le constructeur le plus adapté, vous devez faire appel ensuite à la méthode createDialog() qui fabriquera réellement votre boîte de dialogue en fonction des propriétés que vous avez choisies de mettre en place.

Voici la liste des constructeurs qui sont à votre disposition :

1. JOptionPane() : Création d'une boîte d'option avec un message de test "JOptionPane message" avec un bouton OK et sans icône. Boîte de dialogue de type message.
2. JOptionPane(Object message) : Par rapport au précédent, vous choisissez votre message. Boîte de dialogue de type message.
3. JOptionPane(Object message, int typeMessage) : Cette fois-ci vous précisez en plus le type de message (icône correspondante). Boîte de dialogue de type message.
4. JOptionPane(Object message, int typeMessage, int typeOption) : Par rapport au précédent, vous choisissez en plus les boutons qui feront parti de votre boîte de dialogue. Boîte de dialogue de type confirmation.
5. JOptionPane(Object message, int typeMessage, int typeOption, Icon icône) : Par rapport au précédent, vous choisissez en plus votre icône personnalisée. Boîte de dialogue de type confirmation.
6. JOptionPane(Object message, int typeMessage, int typeOption, Icon icône, Object[] options) : Par rapport au précédent, vous placez en plus l'ensemble des objets qui feront parti de votre boîte de dialogue. Boîte de dialogue de type saisie ou option.
7. JOptionPane(Object message, int typeMessage, int typeOption, Icon icône, Object[] options, Object valeurInitiale) : Exactement le même que le précédent, mais vous choisissez votre composant qui sera visualisé par défaut (utile pour les boîtes combo). Boîte de dialogue de type saisie ou option.

**Exemple de codage**



import javax.swing.\*;

public class Dialogue {

public static void main(String[] args) {

JOptionPane dialogue = new JOptionPane("Bonjour à tous", JOptionPane.INFORMATION\_MESSAGE);

JDialog boîte = dialogue.createDialog("Bienvenue");

boîte.setVisible(true);

}

}

Il existe une deuxième méthode createDialog() qui possède deux paramètres et qui prend en premier argument la fenêtre parente afin de bien mettre en oeuvre une boîte de dialogue modale.

Cette façon de faire reste exceptionnelle puisque vous devez systématiquement passer par trois phases :

1. Vous devez précisez pas mal d'informations au constructeur durant la phase de création.
2. Vous devez créer la boîte de dialogue de type JDialog au travers de la méthode createDialog().
3. Il faut ensuite afficher cette boîte de dialogue au travers de la méthode setVisible(true) de JDialog.

Pour toutes ces raisons, il généralement souhaitable d'utiliser plutôt les méthodes statiques de JOptionPane et de choisir la plus adaptée à la situation requise.

**Comment choisir sa boîte de dialogue standard JOptionPane**

La diversité des choix peut sembler déconcertant, mais en pratique c'est très simple :

1. Choisissez le type de boîte de dialogue (message, confirmation, options ou entrée).
2. Choisissez l'icône (erreur, information, avertissement, question, aucune ou personnalisée).
3. Choisissez le message (chaîne, icône, composant personnalisé ou un tableau d'objets).
4. Pour une boîte de dialogue de confirmation, choisissez un type d'option pour le jeu de boutons (default, Oui/Non, Oui/Non/Annuler ou OK/Annuler).
5. Pour une boîte de dialogue d'options, choisissez les options (chaîne, icône, composant personnalisé), et l'option par défaut.
6. Pour une boîte de dialogue de saisie, choisissez entre un champ de texte et une zone de liste déroulante.
7. Déterminez la méthode appropriée à appeler dans la classe JOptionPane.

Dans les chapitres qui suivent, nous allons reprendre chacune de ces boîtes de dialogue, afin d'exploiter au maximun les possibilités qui nous sont offertes.  
§

**[Choix du chapitre](http://remy-manu.no-ip.biz/Java/Tutoriels/IHM/dialogue.html)Les boîtes de message**

Java dispose de méthodes standard vous permettant de fournir à l'utilisateur un message qui reste affiché tant qu'il n'agit pas sur un bouton OK. Plus précisément, la classe JOptionPane dispose de la méthode statique showMessageDialog(), vue plus haut, permettant d'afficher et de gérer automatiquement une telle boîte. Cette méthode dispose de plusieurs variantes dont la liste est fournie ci-dessous.

1. static void showMessageDialog(Component composantParent, Object message) : Affiche automatiquement une boîte de dialogue de type message, dont le titre est intitulé "Message", avec une icône de type information http://remy-manu.no-ip.biz/Java/Tutoriels/IHM/dialogue/info.gifet avec un bouton OK. Vous spécifiez éventuellement en argument la fenêtre parente et surtout le contenu de votre message (qui peut être de nature différente).
2. static void showMessageDialog(Component composantParent, Object message, String titre, int typeMessage) : Même boîte de message que précédemment, mais cette fois-ci, vous pouvez choisir votre titre et votre icône.
3. static void showMessageDialog(Component composantParent, Object message, String titre, int typeMessage, Icon icône) : Par rapport à la précédente, vous pouvez placer en plus une icône personnalisée.

Nous pouvons créer un message simple de dialogue avec l'une de ces méthodes showMessageDialog(). Ces méthodes affichent une boîte de dialogue qui contient le message désiré, une icône et un bouton OK qui ferme le dialogue. Le dialogue est modal, ce qui signifie qu'il bloque, en ne revenant que quand l'utilisateur a terminé de consulter le message.

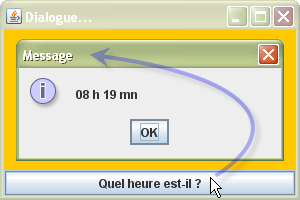
Remarquez au passage que cette méthode statique ne renvoie aucune valeur, ce qui est normal puisqu'il s'agit juste de proposer un message d'information correspondant à la situation actuelle de l'application.

1. L'argument composantParent spécifie le composant servant de parent au dialogue (c'est le composant qui sera rendu actif quand le dialogue sera terminé, généralement la fenêtre de l'application).
2. L'argument titre spécifie une chaîne apparaissant dans la barre de tittre du dialogue.
3. L'argument message est plus complexe. Il est en effet déclaré comme un Object. Nous lui passons généralement une valeur de type String, qui est automatiquement affichée dans un JLabel. Nous pouvons cependant spécifier aussi une icône type Icon qui est également affichée par le JLabel, ou un composant graphique quelconque de JComponent, qui est affiché tel quel. De plus, au lieu de donner un seul objet message, nous pouvons spécifier un tableau d'objets contenant n'importe quelle combinaison de chaînes, d'icônes et de composants.
4. L'argument typeMessage doit être choisi parmi les constantes WARNING\_MESSAGE http://remy-manu.no-ip.biz/Java/Tutoriels/IHM/dialogue/warning.gif, QUESTION\_MESSAGE http://remy-manu.no-ip.biz/Java/Tutoriels/IHM/dialogue/question.gif, INFO\_MESSAGE http://remy-manu.no-ip.biz/Java/Tutoriels/IHM/dialogue/info.gif, ERROR\_MESSAGE http://remy-manu.no-ip.biz/Java/Tutoriels/IHM/dialogue/erreur.gifou PLAIN\_MESSAGE (aucune icône). Ces constantes spécifient le type de message de base à afficher. En général, la personnalisation se limite à l'affichage d'une icône parmi un ensemble standard.
5. Pour surcharger l'icône par défaut du dialogue, nous pouvons également spécifier une icône personnalisée au paramètre icône.

Dans la suite de ce chapitre, nous allons prendre pleins d'exemples, de la boîte de dialogue la plus rudimentaire (souvent suffisante) à la plus sophistiquée, afin de bien montrer tous les réglages et toutes les configurations qu'il est possible de faire sur une simple boîte de dialogue de message. Bien entendu, nous pourrons faire ses mêmes réglages sur les autres types de boîte de dialogue.

**Boîte de message basique activée à partir de la fenêtre principale de l'application**

Venons-en tout de suite à notre premier exemple. Une fenêtre principale d'application possède un bouton qui permet d'activer une boîte de dialogue qui nous renseigne sur l'heure qu'il est. Nous utilisons pour cela la boîte de message la plus rudimentaire :



import java.awt.\*;

import java.awt.event.\*;

import java.text.\*;

import java.util.Date;

import javax.swing.\*;

public class Dialogue extends JFrame implements ActionListener {

private JButton heure = new JButton("Quel heure est-il ?");

public Dialogue() {

super("Dialogue...");

add(heure, BorderLayout.SOUTH);

heure.addActionListener(this);

getContentPane().setBackground(Color.ORANGE);

setSize(300, 200);

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

setVisible(true);

}

public static void main(String[] args) { new Dialogue(); }

public void actionPerformed(ActionEvent e) {

SimpleDateFormat heure = new SimpleDateFormat("HH 'h' mm 'mn'");

JOptionPane.showMessageDialog(this, heure.format(new Date()));

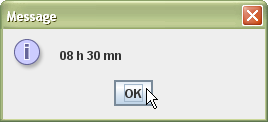
}

}

On notera que le premier argument de la méthode showMessageDialog() représente ce que nous nommons - la fenêtre parent (ou propriétaire) - de la boîte de message ; c'est-à-dire celle dans laquelle elle va s'afficher. Remarquez d'ailleurs que la boîte de dialogue ce centre automatiquement par rapport à la fenêtre propriétaire.

**Même boîte de dialogue mais en mode console**

Il est cependant possible d'afficher une boîte de dialogue indépendamment de toute fenêtre, en donnant à ce premier paramètre la valeur null :



import java.text.SimpleDateFormat;

import java.util.Date;

import javax.swing.JOptionPane;

public class Dialogue {

public static void main(String[] args) {

SimpleDateFormat heure = new SimpleDateFormat("HH 'h' mm 'mn'");

JOptionPane.showMessageDialog(null, heure.format(new Date()));

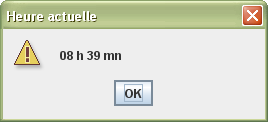
}

}

**Proposer un titre au dialogue et choisir son icône**

La boîte de message usuelle ne permet de définir que le contenu du message. Son titre (Message) et l'icône qu'elle renferme - http://remy-manu.no-ip.biz/Java/Tutoriels/IHM/dialogue/info.gif: lettre i comme "information" - sont imposés. Il existe une autre variante de la méthode showMessageDialog() qui permet de choisir :

1. le contenu du message,
2. le titre de la boîte,
3. le type d'icône.



import java.text.SimpleDateFormat;

import java.util.Date;

import javax.swing.JOptionPane;

public class Dialogue {

public static void main(String[] args) {

SimpleDateFormat heure = new SimpleDateFormat("HH 'h' mm 'mn'");

JOptionPane.showMessageDialog(null,

heure.format(new Date()),

"Heure actuelle",

JOptionPane.WARNING\_MESSAGE);

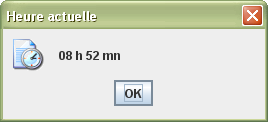
}

}

**Icône personnalisée**

Il existe une troisième variante qui, outre les paramètres précédents, permet de choisir une icône quelconque (objet de la classe Icon) qui remplace l'une des icônes standard.

A ce sujet, il est préférable de stipuler la constante JOptionPane.PLAIN\_MESSAGE pour bien indiquer que nous choisissons une icône personnalisée. Ceci dit, vous avez la possibilité de proposer un autre choix, cela fonctionnera exactement de la même manière.



import java.text.SimpleDateFormat;

import java.util.Date;

import javax.swing.\*;

public class Dialogue {

public static void main(String[] args) {

SimpleDateFormat heure = new SimpleDateFormat("HH 'h' mm 'mn'");

JOptionPane.showMessageDialog(null,

heure.format(new Date()),

"Heure actuelle",

JOptionPane.PLAIN\_MESSAGE,

new ImageIcon("montre.gif"));

}

}

**Choisir son type de message**

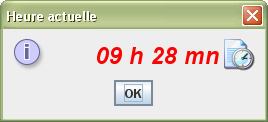
Jusqu'à présent, le type de message que j'utilise est tout simplement une chaîne de caractères (de type String). Je rappelle que ce message peut être une chaîne, une icône, un composant d'interface utilisateur ou tout autre objet. Voici d'ailleurs comment est affiché l'objet message suivant son type :

1. String : La chaîne est affichée.
2. Icon : L'icône est affichée.
3. JComponent : Le composant est affiché.
4. Object[] : Tous les objets du tableau sont affichés les uns au-dessus des autres.
5. tout autre objet : La méthode toString() est appliquée et la chaîne résultante est affichée.

Bien sûr, le message le plus fréquent est celui de type chaîne. Un objet Component offre une souplesse optimale puisque vous pouvez redéfinir la méthode paintComponent() pour dessiner tout ce que vous souhaitez.

Astuce : Une chaîne de message peut contenir des caractères de fin de ligne ('\n'). L'affichage est alors réparti sur plusieurs lignes.  
§

**Utiliser un JLabel comme type de message pour concerver l'icône d'information**



import java.awt.\*;

import java.text.SimpleDateFormat;

import java.util.Date;

import javax.swing.\*;

public class Dialogue {

public static void main(String[] args) {

SimpleDateFormat heure = new SimpleDateFormat("HH 'h' mm 'mn'");

JLabel message = new JLabel(heure.format(new Date()), new ImageIcon("montre.gif"), JLabel.RIGHT);

message.setFont(new Font("Arial", Font.BOLD+Font.ITALIC, 24));

message.setForeground(Color.RED);

message.setHorizontalTextPosition(JLabel.LEFT);

JOptionPane.showMessageDialog(null, message, "Heure actuelle", JOptionPane.INFORMATION\_MESSAGE);

}

}

**Utiliser un tableau de chaîne pour proposer à la fois la date et l'heure sur deux lignes**



import java.text.\*;

import java.util.Date;

import javax.swing.\*;

public class Dialogue {

public static void main(String[] args) {

DateFormat date = DateFormat.getDateInstance(DateFormat.FULL);

SimpleDateFormat heure = new SimpleDateFormat("HH 'h' mm 'mn'");

String[] message = {date.format(new Date()), heure.format(new Date())};

JOptionPane.showMessageDialog(null,

message,

"Aujourd'hui",

JOptionPane.PLAIN\_MESSAGE,

new ImageIcon("montre.gif"));

}

}

Il est possible de prévoir un tableau de JLabel en lieu et place de ce tableau de chaîne afin de permettre le formatage personnalisé du texte.   
**Prévoir une liste déroulante afin de permettre la visualisation de plusieurs paramètres**

import java.text.\*;

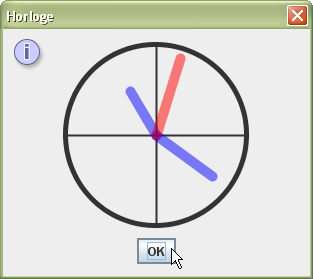
import java.util.Date;

import javax.swing.\*;

public class Dialogue {

public static void main(String[] args) {

DateFormat date = DateFormat.getDateInstance(DateFormat.FULL);

 SimpleDateFormat heure = new SimpleDateFormat("HH 'h' mm 'mn'");

String[] message = {date.format(new Date()), heure.format(new Date())};

JOptionPane.showMessageDialog(null,

new JComboBox(message),

"Aujourd'hui",

JOptionPane.PLAIN\_MESSAGE,

new ImageIcon("montre.gif"));

}

}

**Fabrication d'un nouveau composant qui visualise une horloge en temps réel**

import java.awt.\*;

import java.awt.event.\*;

import java.util.Calendar;

import javax.swing.\*;

public class Dialogue {

public static void main(String[] args) {

Horloge horloge = new Horloge();

JOptionPane.showMessageDialog(null, horloge, "Horloge", JOptionPane.INFORMATION\_MESSAGE);

horloge.arrêt();

}

}

class Horloge extends JComponent implements ActionListener {

private Timer timer = new Timer(100, this);

private double heure;

private double minute, seconde;

public Horloge() {

setPreferredSize(new Dimension(240, 190));

timer.start();

}

public void arrêt() {

timer.stop();

}

@Override

protected void paintComponent(Graphics g) {

Graphics2D dessin = (Graphics2D) g;

dessin.setRenderingHint(RenderingHints.KEY\_ANTIALIASING, RenderingHints.VALUE\_ANTIALIAS\_ON);

dessin.setStroke(new BasicStroke(5));

dessin.drawOval(5, 5, 181, 181);

dessin.setStroke(new BasicStroke(2));

dessin.drawLine(96, 5, 96, 183);

dessin.drawLine(5, 96, 183, 96);

dessin.setStroke(new BasicStroke(10, BasicStroke.CAP\_ROUND, BasicStroke.JOIN\_ROUND));

dessin.setPaint(new Color(0.0F, 0.0F, 1.0F, 0.5F));

dessin.drawLine(96, 96,

(int) (96 + 50\*Math.cos((heure-3) \* 2.0 \* Math.PI / 12)),

(int) (96 + 50\*Math.sin((heure-3) \* 2.0 \* Math.PI / 12)));

dessin.drawLine(96, 96,

(int) (96 + 70\*Math.cos((minute-15) \* 2.0 \* Math.PI / 60)),

(int) (96 + 70\*Math.sin((minute-15) \* 2.0 \* Math.PI / 60)));

dessin.setPaint(new Color(1.0F, 0.0F, 0.0F, 0.5F));

dessin.drawLine(96, 96,

(int) (96 + 80\*Math.cos((seconde-15) \* 2.0 \* Math.PI / 60)),

(int) (96 + 80\*Math.sin((seconde-15) \* 2.0 \* Math.PI / 60)));

}

public void actionPerformed(ActionEvent e) {

Calendar horaire = Calendar.getInstance();

minute = horaire.get(Calendar.MINUTE);

seconde = horaire.get(Calendar.SECOND);

heure = horaire.get(Calendar.HOUR)+minute/60;

repaint();

}

}

Nous voyons ici toute la puissance des boîtes de dialogue standard. Rien empêche donc de créer le composant qui vous plait et de le placer ensuite dans votre boîte de dialogue, qui constitue ainsi votre message principal. Ici, nous voyons notre horloge qui fonctionne en temps réel avec l'aiguille des secondes qui se déplace automatiquement.

**Prévoir un JLabel qui spécifie la date du jour en même temps que l'horloge temps réel que nous venons de mettre en oeuvre**



import java.awt.\*;

import java.awt.event.\*;

import java.text.DateFormat;

import java.util.Calendar;

import javax.swing.\*;

public class Dialogue {

public static void main(String[] args) {

DateFormat aujourdhui = DateFormat.getDateInstance(DateFormat.FULL);

Horloge horloge = new Horloge();

JLabel date = new JLabel(aujourdhui.format(System.currentTimeMillis()));

date.setFont(new Font("ARIAL", Font.BOLD+Font.ITALIC, 20));

Object[] date\_heure = {date, horloge};

JOptionPane.showMessageDialog(null, date\_heure, "Horloge", JOptionPane.INFORMATION\_MESSAGE);

horloge.arrêt();

}

}

class Horloge extends JComponent implements ActionListener {

private Timer timer = new Timer(100, this);

private double heure;

private double minute, seconde;

public Horloge() {

setPreferredSize(new Dimension(240, 190));

timer.start();

}

public void arrêt() {

timer.stop();

}

@Override

protected void paintComponent(Graphics g) {

Graphics2D dessin = (Graphics2D) g;

dessin.setRenderingHint(RenderingHints.KEY\_ANTIALIASING, RenderingHints.VALUE\_ANTIALIAS\_ON);

dessin.setStroke(new BasicStroke(5));

dessin.drawOval(5, 5, 181, 181);

dessin.setStroke(new BasicStroke(2));

dessin.drawLine(96, 5, 96, 183);

dessin.drawLine(5, 96, 183, 96);

dessin.setStroke(new BasicStroke(10, BasicStroke.CAP\_ROUND, BasicStroke.JOIN\_ROUND));

dessin.setPaint(new Color(0.0F, 0.0F, 1.0F, 0.5F));

dessin.drawLine(96, 96,

(int) (96 + 50\*Math.cos((heure-3) \* 2.0 \* Math.PI / 12)),

(int) (96 + 50\*Math.sin((heure-3) \* 2.0 \* Math.PI / 12)));

dessin.drawLine(96, 96,

(int) (96 + 70\*Math.cos((minute-15) \* 2.0 \* Math.PI / 60)),

(int) (96 + 70\*Math.sin((minute-15) \* 2.0 \* Math.PI / 60)));

dessin.setPaint(new Color(1.0F, 0.0F, 0.0F, 0.5F));

dessin.drawLine(96, 96,

(int) (96 + 80\*Math.cos((seconde-15) \* 2.0 \* Math.PI / 60)),

(int) (96 + 80\*Math.sin((seconde-15) \* 2.0 \* Math.PI / 60)));

}

public void actionPerformed(ActionEvent e) {

Calendar horaire = Calendar.getInstance();

minute = horaire.get(Calendar.MINUTE);

seconde = horaire.get(Calendar.SECOND);

heure = horaire.get(Calendar.HOUR)+minute/60;

repaint();

}

}

Là encore, nous pouvons décider d'avoir une boîte de message extrêment riche. Grâce aux tableaux d'objets (Object[]), vous pouvez vraiment placer l'ensemble des éléments qui vous passent par la tête.

**[Choix du chapitre](http://remy-manu.no-ip.biz/Java/Tutoriels/IHM/dialogue.html)Les boîtes de confirmation**

Java permet d'afficher des boîtes dites "de confirmation" offrant à l'utilisateur un choix de type Oui/Non. Il suffit pour cela de recourir à l'une des variantes de la méthode statique showConfirmDialog() de la classe JOptionPane :

1. static int showConfirmDialog(Component composantParent, Object message) : Affiche automatiquement une boîte de dialogue de type confirmation, dont le titre est intitulé "Sélectionnez une option ", avec une icône de type question http://remy-manu.no-ip.biz/Java/Tutoriels/IHM/dialogue/question.gifet avec l'ensemble des boutons Oui/Non/Annuler. Vous spécifiez éventuellement en argument la fenêtre parente et surtout le contenu de votre message (qui peut être de nature différente). Cette méthode retourne une valeur entière qui correspond au choix qui est fait, c'est-à-dire sur quel bouton vous avez cliqué (bouton de clôture compris).
2. static int showConfirmDialog(Component composantParent, Object message, String titre, int typeOption) : Même boîte de confirmation que précédemment, mais cette fois-ci, vous pouvez choisir votre titre et les boutons qui vont constituer votre choix.
3. static int showConfirmDialog(Component composantParent, Object message, String titre, int typeOption, int typeMessage) : Même boîte de confirmation que précédemment, mais cette fois-ci, vous pouvez choisir votre type de message qui se traduit par unes des icônes standard.
4. static int showConfirmDialog(Component composantParent, Object message, String titre, int typeOption, int typeMessage, Icon icône) : Par rapport à la précédente, vous pouvez placer en plus une icône personnalisée.

Les méthodes showConfirmDialog() ressemblent beaucoup aux méthodes showMessageDialog(), si ce n'est qu'elles demandent à l'utilisateur de faire son choix et qu'elles fournissent plusieurs boutons poussoirs représentant les options disponibles. Par exemple, showConfirmDialog() peut être utilisée pour afficher un dialogue demandant "Voulez-vous vraiment quitter ?", en permettant à l'utilisateur de répondre en poussant soit un bouton Oui soit un bouton Non.

Les arguments composantParent, message, titre, typeMessage et icône de ces méthodes sont les mêmes que pour les dialogues de message. Les dialogues de confirmation ajoutent un argument typeOption et une valeur de retour.

L'argument typeOption indique quel bouton doit apparaître dans le dialogue et vous permet de choisir parmi les quatres constantes suivantes :

* JOptionPane.DEFAULT\_OPTION : fournit un unique bouton OK.
* JOptionPane.YES\_NO\_OPTION : fournit les boutons Oui et Non .
* JOptionPane.YES\_NO\_CANCEL\_OPTION : fournit les boutons Oui, Non et Annuler (valeur par défaut).
* JOptionPane.OK\_CANCEL\_OPTION : fournit les boutons OK et Annuler.

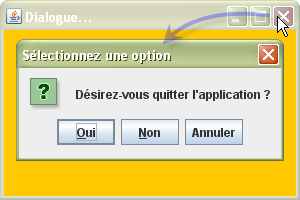
Comme les boîtes de message, les dialogues de confirmation sont modaux et les méthodes statiques qui les affichent bloquent jusqu'à ce que l'utilisateur ferme le dialogue. Comme les dialogues de confirmation présente des choix à l'utilisateur, ils donnent une valeur de retour reflétant le choix de l'utilisateur. Cette valeur est l'une des constantes suivantes :

* JOptionPane.OK\_OPTION.
* JOptionPane.CANCEL\_OPTION.
* JOptionPane.YES\_OPTION.
* JOptionPane.NO\_OPTION.
* JOptionPane.CLOSED\_OPTION.

Cette dernière valeur indique que l'utilisateur a fermé la fenêtre de dialogue sans sélectionner l'un des boutons disponibles ; elle doit généralement être traitée comme une réponse CANCEL\_OPTION.

**Confirmation pour quitter un programme**

A titre d'exemple, je vous propose de prendre la première méthode showConfirmDialog() qui sera dans la plupart des cas largement suffisante. Par ailleurs, très souvent, avant de quitter une application, il est souhaitable de prévenir l'utilisateur pour que ce dernier confirme la sortie du programme, surtout si il doit penser à enregistrer ses dernières modifications effectuées dans un document.



import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

public class Dialogue extends JFrame {

public Dialogue() {

super("Dialogue...");

getContentPane().setBackground(Color.ORANGE);

setSize(300, 200);

addWindowListener(new WindowAdapter() {

@Override

public void windowClosing(WindowEvent e) {

if (JOptionPane.showConfirmDialog(Dialogue.this, "Désirez-vous quitter l'application ?")

== JOptionPane.YES\_OPTION) System.exit(0);

}

});

setDefaultCloseOperation(DO\_NOTHING\_ON\_CLOSE);

setVisible(true);

}

public static void main(String[] args) { new Dialogue(); }

}

Là encore, le premier paramètre de la méthode statique showConfirmDialog() peut prendre la valeur null, auquel cas la boîte est affichée indépendamment d'une quelconque fenêtre.

La boîte reste affichée jusqu'à ce que l'utilisateur agisse sur l'un des boutons ou qu'il ferme la boîte. La valeur de retour de la méthode showConfirmDialog() précise l'action effectuée par l'utilisateur, sous forme d'un entier ayant comme valeur l'une des constantes suivantes de la classe JOptionPane :

1. valeur entière 0 : YES\_OPTION.
2. valeur entière 1 : NO\_OPTION.
3. valeur entière 2 : CANCEL\_OPTION.
4. valeur entière -1 : CLOSED\_OPTION.

Notez que les constantes CANCEL\_OPTION et CLOSED\_OPTION sont différentes. En pratique, nous exploitons rarement cette particularité. Généralement, nous considérons que dans les deux cas, nous avons affaire à un abandon du dialogue.

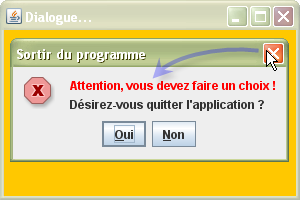
**Imposer à l'utilisateur de faire son choix**

La boîte de confirmation usuelle que nous venons de présenter permet seulement le choix de la question posée à l'utilisateur. Il existe une autre variante de la méthode showConfirmDialog() qui permet en outre de choisir le titre de la boîte, ainsi que la nature des boutons qui s'y trouvent. Ceux-ci sont définis par un paramètre entier dont la valeur est choisie parmi les constantes suivantes :

* JOptionPane.DEFAULT\_OPTION : -1 : fournit un unique bouton OK.
* JOptionPane.YES\_NO\_OPTION : 0 : fournit les boutons Oui et Non .
* JOptionPane.YES\_NO\_CANCEL\_OPTION : 1 : fournit les boutons Oui, Non et Annuler (valeur par défaut).
* JOptionPane.OK\_CANCEL\_OPTION : 2 : fournit les boutons OK et Annuler.

Il existe deux autres versions de la méthode statique showConfirmDialog() qui permettentde choisir en plus l'une des icônes standard où même une icône personnalisée.

A titre d'exemple, je vous propose de mettre en place une boîte de confirmation avec les seuls boutons Oui et Non. Le choix doit se faire impérativement sur l'un de ces deux boutons. Si l'utilisateur clique sur le bouton de fermeture de la boîte de confirmation, une nouvelle boîte de confirmation apparaît avec une alerte en rouge et une icône d'erreur. Tant que l'utilisateur ne choisi pas l'un des deux boutons prévues pour le choix, cette boîte de confirmation réapparaît constamment.



import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

public class Dialogue extends JFrame {

public Dialogue() {

super("Dialogue...");

getContentPane().setBackground(Color.ORANGE);

setSize(300, 200);

addWindowListener(new WindowAdapter() {

@Override

public void windowClosing(WindowEvent e) {

switch (JOptionPane.showConfirmDialog(Dialogue.this,

"Désirez-vous quitter l'application ?","Sortir du programme",

JOptionPane.YES\_NO\_OPTION)) {

case JOptionPane.YES\_OPTION : System.exit(0);

case JOptionPane.CLOSED\_OPTION : {

JLabel alerte = new JLabel("Attention, vous devez faire un choix !");

alerte.setForeground(Color.RED);

int choix;

while ((choix = JOptionPane.showConfirmDialog(Dialogue.this,

new Object[] {alerte, "Désirez-vous quitter l'application ?"},

"Sortir du programme",

JOptionPane.YES\_NO\_OPTION,

JOptionPane.ERROR\_MESSAGE))==JOptionPane.CLOSED\_OPTION);

if (choix==JOptionPane.YES\_OPTION) System.exit(0);

}

}

}

});

setDefaultCloseOperation(DO\_NOTHING\_ON\_CLOSE);

setVisible(true);

}

public static void main(String[] args) { new Dialogue(); }

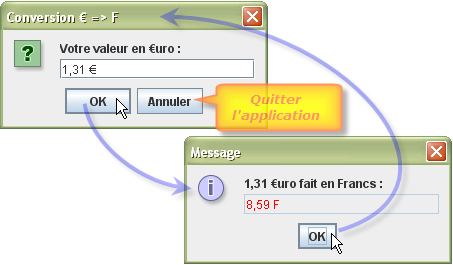
}

**Réaliser une saisie à partir d'une boîte de confirmation**

Bien que cela ne soit pas prévu à priori, il est tout à fait possible de prendre n'importe quel composant pour constituer notre message, et pourquoi pas un composant spécialisé dans la saisie comme un JFormattedTextField.

Bien sûr, dans le chapitre suivant nous ferrons appel à la méthode showInputDialog() de la classe JOptionPane qui est justement prévue pour la saisie, mais c'est encore une fois pour montrer la grande souplesse d'utilisation de ces méthodes statiques, notamment dans le choix du type de message.

A titre d'exemple, je vous propose donc de construire une boîte de confirmation qui permet de récupérer une valeur saisie en €uro. Cette valeur est automatiquement formatée dans la valeur monétaire européenne.



Si l'utilisateur clique sur Annuler le programme s'arrête alors sans effectuer de calcul. Par contre, si l'utilisateur clique sur le bouton OK, un traitement est réalisé et la conversion en franc est alors proposée dans une boîte de message adaptée.

L'utilisateur peut effectuer autant de calculs qu'il le désire jusqu'à ce qu'il clique sur le bouton Annuler de la boîte de confirmation.

import java.awt.Color;

import java.text.\*;

import javax.swing.\*;

public class Dialogue {

public static void main(String[] args) {

JFormattedTextField saisie = new JFormattedTextField(NumberFormat.getCurrencyInstance());

saisie.setValue(0);

Object[] message = {"Votre valeur en €uro :", saisie};

while (JOptionPane.showConfirmDialog(null, message, "Conversion € => F", JOptionPane.OK\_CANCEL\_OPTION)

== JOptionPane.YES\_OPTION) {

double €uro = ((Number)saisie.getValue()).doubleValue();

final double CONVERSION = 6.55957;

JFormattedTextField franc = new JFormattedTextField(new DecimalFormat("#,##0.00 F"));

franc.setEditable(false);

franc.setForeground(Color.RED);

franc.setValue(€uro\*CONVERSION);

String motif = "{0, number, currency}uro{0, choice, 0#|2#s} {0, choice, 0#fait|2#font} en Franc{0, choice, 0#|0.31#s} :";

Object[] résultat = {MessageFormat.format(motif, €uro), franc};

JOptionPane.showMessageDialog(null, résultat);

}

}

}

**[Choix du chapitre](http://remy-manu.no-ip.biz/Java/Tutoriels/IHM/dialogue.html)Les boîtes de saisie**

La boîte de saisie permet à l'utilisateur de fournir une information sous la forme d'une chaîne de caractères. La méthode statique showInputDialog() de la classe JOptionPane vous permet de gérer automatiquement le dialogue avec l'utilisateur.

1. static String showInputDialog(Object message) : Affiche automatiquement une boîte de dialogue de type entrée (saisie), dont le titre est intitulé "Entrée", avec une icône de type question http://remy-manu.no-ip.biz/Java/Tutoriels/IHM/dialogue/question.gifet avec l'ensemble des boutons OK/Annuler. Vous spécifiez le contenu de votre message (qui peut être de nature différente). Cette méthode attend la saisie d'une valeur correspondant à une chaîne de caractères et retourne cette valeur dans le cas où vous avez sélectionné le bouton OK. Dans le cas contraire, c'est la valeur null qui est renvoyée.
2. static String showInputDialog(Object message, Object valeurInitiale) : Même boîte de saisie, en proposant en plus une valeur par défaut.
3. static String showInputDialog(Component composantParent, Object message) :   
   static String showInputDialog(Component composantParent, Object message, Object valeurInitiale) : Mêmes boîtes de saisie que les précédentes, mais cette fois-ci vous spécifiez la fenêtre propriétaire.
4. static String showInputDialog(Component composantParent, Object message, String titre, int typeMessage) : Même boîte de saisie que précédemment, mais cette fois-ci, vous pouvez choisir votre titre ainsi que l'icône prédéfinie.
5. static Object showInputDialog(Component composantParent, Object message, String titre, int typeMessage, Icon icône, Object[] valeursSélection, Object valeurInitiale) : Il s'agit en réalité ici plutôt d'une boîte de sélection qu'une boîte de saisie. En effet, vous proposez à l'argument valeursSélection le tableau de valeurs possibles qui sera généralement représenté par une boîte combo.

Les méthodes showInputDialog() de la classe JOptionPane représentent les dialogues d'entrée. La plupart des versions de ces méthodes prennent les mêmes arguments que les dialogues de message. Cependant, en plus d'afficher un message, ils contiennet également un JTextField dans lequel l'utilisateur peut entrer la valeur demandée.

Ces dialogues sont modaux et les méthodes qui les affichent bloquent jusqu'à ce que l'utilisateur ferme le dialogue.

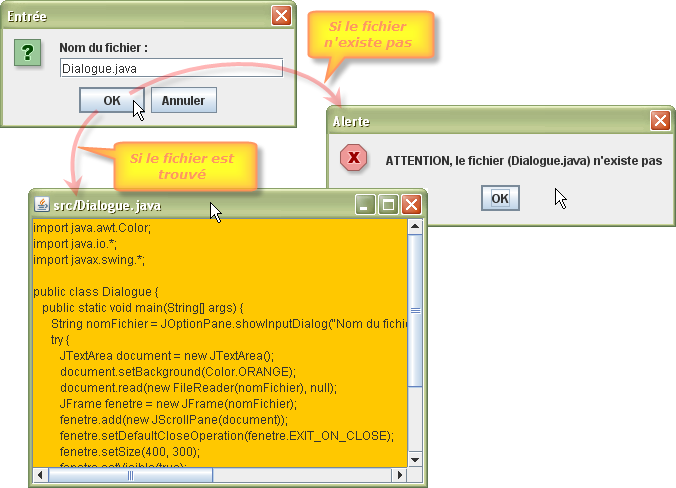
* S'il le ferme avec le bouton OK, les méthodes renvoient ce que l'utilisateur a saisie sous forme de String.
* Si l'utilisateur n'a pas confirmé sa saisie par OK, autrement dit s'il a agit sur Annuler ou s'il a fermé la boîte de saisie (et ce même s'il a commencé à entrer une information dans le champ de texte), ces méthodes renvoient la valeur null.

La dernière version de showInputDialog() prend l'argument supplémentaire valeursSélection. Ainsi, au lieu de demander à l'utilisateur de rentrer une chaîne, elle le fait choisir parmi les valeurs contenues dans le tableau valeursSélection. L'affichage de ces valeurs est laissé au look-and-feel courant, bien qu'elles soient habituellement affichées par un composant JComboBox ou JList. Le tableau valeursSélection contient généralement des chaînes de caractères, mais il peut aussi contenir des objets Icon ou d'autres qui peuvent être affichées de manière significative par les composants JList et JComboBox.

Quand nous passons un tableau de valeursSélection à showInputDialog(), la valeur de retour est celle que l'utilisateur aura choisi, ou null si l'utilisateur n'a pas sélectionné le bouton OK.

**Mise en oeuvre d'une boîte de saisie**

Comme d'habitude, je vous propose de mettre en place une boîte de saisie la plus simple possible. Effectivement, nous allons utiliser la méthode showInputDialog() qui possède comme seul argument le message désiré, ici "Nom du fichier :". Nous construisons une application qui permet de consulter un document enregistré dans un fichier. Dès que le programme débute, la boîte de saisie apparaît en nous demandant d'introduire le nom du fichier désiré. Si ce nom est correct, une fenêtre est alors créée et nous montre le document concerné. Dans le cas contraire, une boîte de message est alors proposée nous indiquant notre erreur de saisie à l'issu de quoi, le programme s'arrête.



**codage correspondant**

import java.awt.Color;

import java.io.\*;

import javax.swing.\*;

public class Dialogue {

public static void main(String[] args) {

String nomFichier = JOptionPane.showInputDialog("Nom du fichier :");

try {

JTextArea document = new JTextArea();

document.setBackground(Color.ORANGE);

document.read(new FileReader(nomFichier), null);

JFrame fenetre = new JFrame(nomFichier);

fenetre.add(new JScrollPane(document));

fenetre.setDefaultCloseOperation(fenetre.EXIT\_ON\_CLOSE);

fenetre.setSize(400, 300);

fenetre.setVisible(true);

}

catch (IOException ex) {

JOptionPane.showMessageDialog(null,

"ATTENTION, le fichier ("+nomFichier+") n'existe pas",

"Alerte",

JOptionPane.ERROR\_MESSAGE);

}

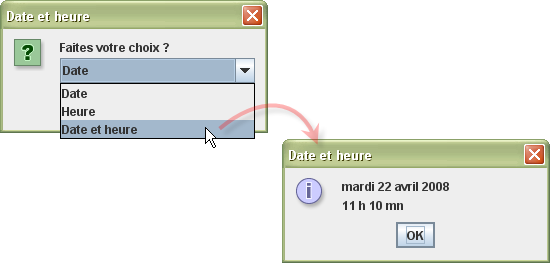
}

}

**Boîte de sélection**

Nous remarquons que c'est vraiment très facile de construire une boîte de saisie. Il existe bien entendu plein de variantes dont nous connaissons bien les principes maintenant, et je ne vais d'ailleurs pas m'y attarder. Je vous propose toutefois de travailler avec la dernière méthode showInputDialog() puisqu'elle est un peu plus particulière dans le sens où il est possible de proposer un choix parmi un ensemble de valeurs prédéfinies.

A titre d'exemple, voici une boîte de sélection qui permet de choisir l'affichage entre la date et/ou l'heure actuelles :



**codage correspondant**

import java.text.\*;

import java.util.Date;

import javax.swing.\*;

public class Dialogue {

public static void main(String[] args) {

DateFormat date = DateFormat.getDateInstance(DateFormat.FULL);

SimpleDateFormat heure = new SimpleDateFormat("HH 'h' mm 'mn'");

String[] sélections = {"Date", "Heure", "Date et heure"};

String choix = (String)JOptionPane.showInputDialog(null,

"Faites votre choix ?",

"Date et heure",

JOptionPane.QUESTION\_MESSAGE,

null,

sélections,

null);

if (choix.equals("Date")) {

String message = date.format(new Date());

JOptionPane.showMessageDialog(null, message, choix, JOptionPane.INFORMATION\_MESSAGE);

}

else if (choix.equals("Heure")) {

String message = heure.format(new Date());

JOptionPane.showMessageDialog(null, message, choix, JOptionPane.INFORMATION\_MESSAGE);

}

else {

String[] message = {date.format(new Date()), heure.format(new Date())};

JOptionPane.showMessageDialog(null, message, choix, JOptionPane.INFORMATION\_MESSAGE);

}

}

}

**[Choix du chapitre](http://remy-manu.no-ip.biz/Java/Tutoriels/IHM/dialogue.html)Les boîtes d'options**

Il existe une dernière boîte de dialogue prédéfinie qui s'appelle tout simplement boîte d'options. Cette boîte d'options, représentée par la méthode statique showOptionDialog(), généralise en fait la boîte de dialogue de confirmation. Il existe effectivement un argument supplémentaire, appelé options, qui indique quels boutons afficher dans la boîte de dialogue. Ils remplacent alors les boutons traditionnels, comme OK/Annuler. Vous pouvez ainsi proposer des libellés qui correspondent plus précisément à votre application.

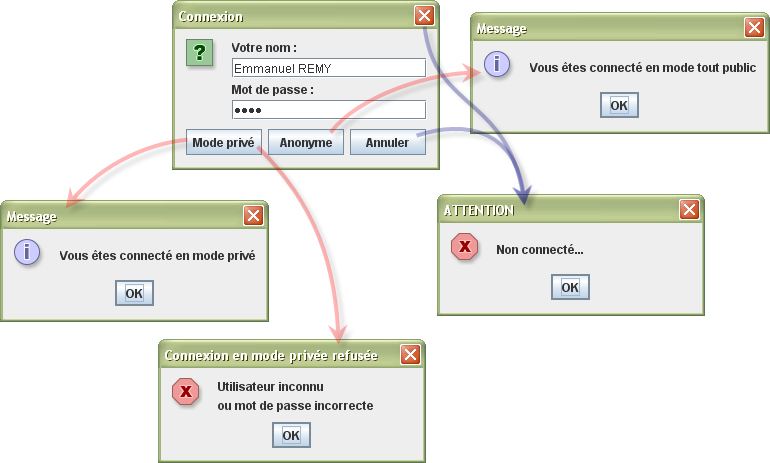
static Object showOptionDialog(Component composantParent, Object message, String titre, int typeOption, int typeMessage, Icon icône, Object[] options, Object valeurInitiale) : Boîte de dialogue de type option. Il existe une seule méthode. Du coup, vous devez spécifier un nombre considérable d'arguments. Nous connaissons déjà au travers des autres méthodes le rôle de chacun de ces arguments. Celui qui est le plus important ici, et c'est ce qui fait sa spécificité, c'est l'argument options au travers duquel vous passez l'ensemble des éléments qui constituerons les boutons de la boîte de dialogue, sous forme de tableaux d'objets.

Dans cet argument options, nous spécifions typiquement un tableau de chaînes que JOptionPane affiche dans des composants JButton. Ces boutons s'affichent alors sur la même ligne. Rien n'empêche toutefois de choisir d'autres types de composants, comme des zones de saisie, des libellés, etc.

La valeur de retour est, comme pour la boîte de confirmation, de type entier. Le nombre retourné correpond à l'ordre de placement des boutons en commençant par 0. Ainsi, si vous cliquez sur le troisième bouton, c'est le chiffre 2 qui est renvoyée par la boîte de dialogue.

**Mise en oeuvre d'une boîte d'option**

A titre d'exemple, je vous propose de fabriquer une boîte de dialogue qui permet de nous connecter à un service particulier. Nous pouvons nous connecter à ce service soit de façon anonyme, c'est-à-dire tout public, ou éventuellement en mode privé auquel cas, vous devez alors donner votre nom d'utilisateur et spécifier votre mot de passe.



**codage correspondant**

import javax.swing.\*;

public class Dialogue {

public static void main(String[] args) {

JTextField utilisateur = new JTextField();

JPasswordField motDePasse = new JPasswordField();

int choix = JOptionPane.showOptionDialog(null,

new Object[] {"Votre nom :", utilisateur, "Mot de passe :", motDePasse},

"Connexion",

JOptionPane.NO\_OPTION,

JOptionPane.QUESTION\_MESSAGE, null,

new String[] {"Mode privé", "Anonyme", "Annuler"}, null);

switch (choix) {

case 0 : if (utilisateur.getText().equalsIgnoreCase("Emmanuel REMY") && motDePasse.getText().equals("manu"))

JOptionPane.showMessageDialog(null, "Vous êtes connecté en mode privé");

else

JOptionPane.showMessageDialog(null,

new String[] {"Utilisateur inconnu", "ou mot de passe incorrecte"},

"Connexion en mode privée refusée",

JOptionPane.ERROR\_MESSAGE);

break;

case 1 : JOptionPane.showMessageDialog(null, "Vous êtes connecté en mode tout public"); break;

default : JOptionPane.showMessageDialog(null, "Non connecté...", "ATTENTION", JOptionPane.ERROR\_MESSAGE); break;

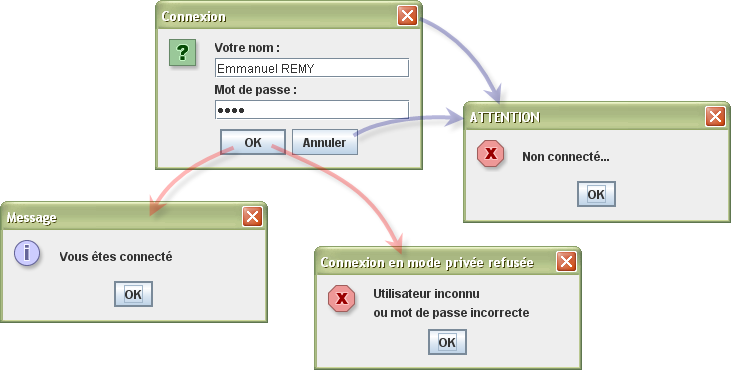
}

}

}

**Même type de connexion mais avec des boutons standards**

Bien entendu, vous n'êtes pas obligé de prendre en compte l'argument options (vous proposez alors la valeur null) de cette méthode showOptionDialog(). Dans ce cas là, il vous faut préciser les boutons standards que vous désirez voir apparaître au travers de l'argument typeOption.



**codage correspondant**

import javax.swing.\*;

public class Dialogue {

public static void main(String[] args) {

JTextField utilisateur = new JTextField();

JPasswordField motDePasse = new JPasswordField();

int choix = JOptionPane.showOptionDialog(null,

new Object[] {"Votre nom :", utilisateur, "Mot de passe :", motDePasse},

"Connexion",

JOptionPane.OK\_CANCEL\_OPTION,

JOptionPane.QUESTION\_MESSAGE, null, null, null);

if (choix == JOptionPane.OK\_OPTION)

if (utilisateur.getText().equalsIgnoreCase("Emmanuel REMY") && motDePasse.getText().equals("manu"))

JOptionPane.showMessageDialog(null, "Vous êtes connecté");

else

JOptionPane.showMessageDialog(null,

new String[] {"Utilisateur inconnu", "ou mot de passe incorrecte"},

"Connexion refusée",

JOptionPane.ERROR\_MESSAGE);

else

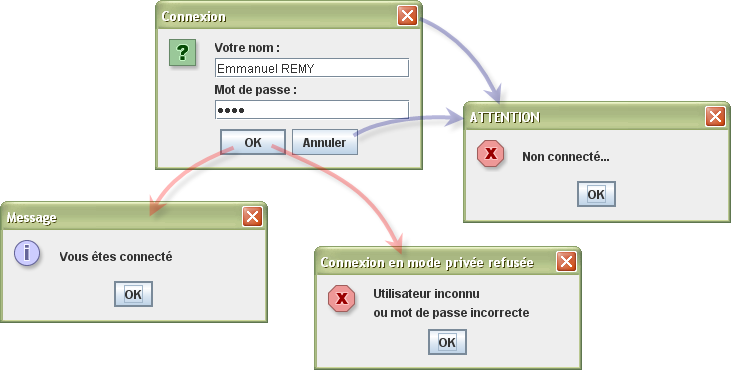
JOptionPane.showMessageDialog(null, "Non connecté...", "ATTENTION", JOptionPane.ERROR\_MESSAGE);

}

}

**Prendre à la place une boîte de confirmation avec la méthode showConfirmDialog()**

La particularité de la boîte d'options par rapport à la boîte de confirmation c'est uniquement le nombre de boutons que vous désirez placer dans la boîte de dialogue et le choix de leurs libellés. Si vous ne désirez pas créer de nouveaux boutons et si vous préférez plutôt utiliser les boutons standards, autant choisir une boîte de dialogue de type confirmation, dont la méthode showConfirmDialog() possède moins d'arguments à prendre en compte.



**codage correspondant**

import javax.swing.\*;

public class Dialogue {

public static void main(String[] args) {

JTextField utilisateur = new JTextField();

JPasswordField motDePasse = new JPasswordField();

int choix = JOptionPane.showConfirmDialog(null,

new Object[] {"Votre nom :", utilisateur, "Mot de passe :", motDePasse},

"Connexion",

JOptionPane.OK\_CANCEL\_OPTION);

if (choix == JOptionPane.OK\_OPTION)

if (utilisateur.getText().equalsIgnoreCase("Emmanuel REMY") && motDePasse.getText().equals("manu"))

JOptionPane.showMessageDialog(null, "Vous êtes connecté");

else

JOptionPane.showMessageDialog(null,

new String[] {"Utilisateur inconnu", "ou mot de passe incorrecte"},

"Connexion refusée",

JOptionPane.ERROR\_MESSAGE);

else

JOptionPane.showMessageDialog(null, "Non connecté...", "ATTENTION", JOptionPane.ERROR\_MESSAGE);

}

}

**[Choix du chapitre](http://remy-manu.no-ip.biz/Java/Tutoriels/IHM/dialogue.html)Boîte de dialogue Fichier (sélecteur de fichier)**

Lorsque vous écrivez une application, il est souvent nécessaire d'ouvrir ou d'enregistrer des fichiers. Une boîte de dialogue Fichier (File) bien conçue, qui affiche les fichiers et les répertoires et permet à l'utilisateur de naviguer dans le système de fichiers, est difficile à développer. Il est de plus totalement inutile de vouloir "réinventer la roue", puisque Swing propose une classe JFileChooser qui permet d'afficher une boîte de dialogue Fichier semblable à celle qu'utilisent la plupart des applications natives.

Un JFileChooser est donc une boîte de sélection de fichier standard. Comme les autres composants Swing, il est implémenté en pur Java, paraît et agit donc de la même façon sur des plateformes différentes.

Les boîtes de sélection de fichier JFileChooser sont toujours modales. Notez que cette classe n'est pas une sous-classe de JDialog. Au lieu d'appeler setVisible(true), vous appelez :

1. la méthode showOpenDialog() pour afficher une boîte de dialogue d'ouverture de fichier.
2. la méthode showSaveDialog() pour afficher une boîte de dialogue d'enregistrement de fichier.

Le bouton utilisé pour accepter un fichier est automatiquement libellé Ouvrir (Open) ou Enregistrer (Save). Vous pouvez également spécifier votre propre intitulé de bouton à l'aide de la méthode showDialog().

1. int showDialog(Component parent, String libelléBouton) : affichage d'une boîte de sélection de fichier avec un bouton de validation personnalisé.
2. int showOpenDialog(Component parent) : affichage d'une boîte de sélection de fichier en mode ouverture.
3. int showSaveDialog(Component parent) : affichage d'une boîte de sélection de fichier en mode enregistrement.

**Séquence d'utilisation**

Voici les étapes à suivre pour mettre en place une boîte de dialogue de sélection de fichiers et récupérer ensuite le choix réalisé par l'utilisateur :

1°) Créer un objet de type JFileChooser. A la différence du constructeur de la classe JDialog, vous n'avez pas besoin d'indiquer le composant parent, ce qui vous permet de réutiliser une même boîte de dialogue avec plusieurs cadres :

JFileChooser sélection = new JFileChooser();

Réutiliser un objet sélecteur de fichier est une bonne idée, car le constructeur de JFileChooser peut se révéler assez lent, surtout sous Windows, notamment si l'utilisateur possède de nombreux mappages d'unités réseau.

2°) Définissez un répertoire de travail en appelant la méthode setCurrentDirectory() :

sélection.setCurrentDirectory(new File(".")); // travailler avec le répertoire courant.

Reportez-vous à l'étude suivante ([la gestion des fichiers](http://remy-manu.no-ip.biz/Java/Tutoriels/TechniquesSup/GestionFichiers.html)) pour bien comprendre le fonctionnement de la classe File.  
§

Si cette boîte de sélection de fichier n'est utilisée qu'une seule fois, il est possible de préciser le répertoire de travail durant la phase de création en choisissant le bon constructeur :

JFileChooser sélection = new JFileChooser(new File("."));  
ou encore plus facile :   
JFileChooser sélection = new JFileChooser(".");

3°) Si l'utilisateur peut choisir un nom de fichier par défaut, appelez alors la méthode setSelectedFile() de la façon suivante :

sélection.setSelectedFile(new File("fichier")); // sélection automatique d'un fichier par défaut

4°) Pour permettre à l'utilisateur de sélectionner plusieurs fichiers, appelez la méthode setMultiSelectionEnabled(). Son utilisation est peu courante, et bien sûr optionnelle :

sélection.setMultiSelectionEnabled(true);

5°) Si vous désirez limiter l'affichage à un type de fichier particulier (par exemple tous les fichiers avec l'extension .gif), vous devez alors définir un filtre de fichier, dont nous reparlerons plus loin dans cette section.

6°) Par défaut, un utilisateur ne peut sélectionner que des fichiers dans un sélecteur de fichier. Si vous désirez qu'il soit capable de sélectionner des répertoires, utiliser la méthode setFileSelectionMode(), de la façon suivante :

* JFileChooser.FILES\_ONLY : fichiers seulement (fonctionnement par défaut).
* JFileChooser.DIRECTORIES\_ONLY : répertoires seulement.
* JFileChooser.FILES\_AND\_DIRECTORIES : les deux.

7°) Affichez la boîte de dialogue en appelant la méthode showOpenDialog() ou showSaveDialog(). Vous devez indiquer le composant propriétaire pour ces appels :

JFrame propriétaire = new JFrame();   
int résultat = sélection.showOpenDialog(propriétaire);   
ou  
int résultat = sélection.showSaveDialog(propriétaire);

La seule différence entre ces appels est le libellé du "bouton d'approbation" (ainsi que le titre de la boîte de dialogue), celui sur lequel l'utilisateur clique pour mettre fin à la sélection du fichier. Vous avez la possibilité de personnaliser votre intitulé en appelant la méthode showDialog() et en passant le texte explicite, pour le "bouton d'approbation" , et pour le titre de la boîte de dialogue :

JFrame propriétaire = new JFrame();   
int résultat = sélection.showDialog(propriétaire, "Valider votre choix de fichier");

A la suite de ces appels, le flux d'exécution du programme ne revient pas tant que l'utilisateur n'a pas accepté un fichier ou annulé son action. La valeur renvoyée est :

JFileChooser.APPROVE\_OPTION : si l'utilisateur clique sur le bouton d'approbation, suivant le cas : "Ouvrir", "Enregistrer", "Intitulé personnalisé".

JFileChooser.CANCEL\_OPTION : si l'utilisateur clique sur le bouton "Annuler" ou s'il choisit de sortir par le bouton de fermeture de la boîte de dialogue.

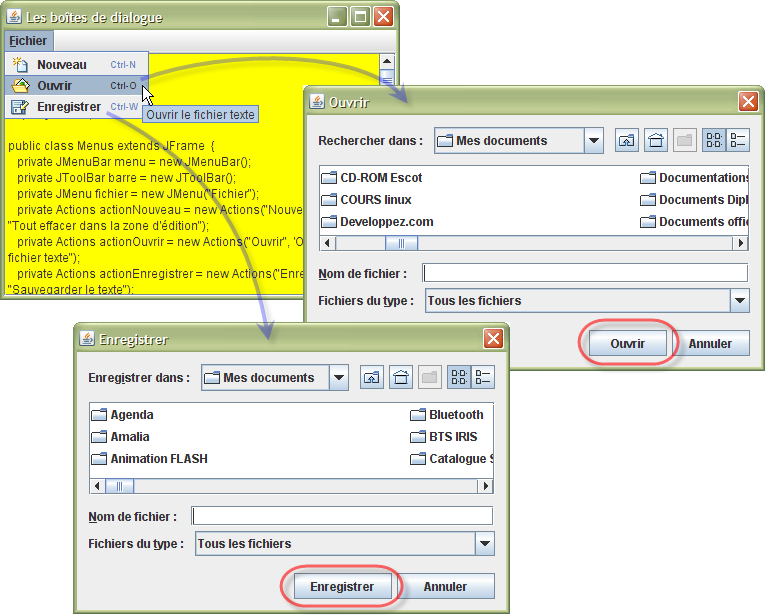
JFileChooser.ERROR\_OPTION : si une erreur se produit lors de la sélection du fichier.

8°) Vous récupérez le ou les fichiers sélectionnés au moyen des méthodes getSelectedFile() ou getSelectedFiles(). Ces méthodes renvoient soit un seul objet File, soit un tableau d'objets File. Si vous avez uniquement besoin du nom de l'objet fichier, appelez sa méthode getName() pour avoir juste le nom du fichier ou sa méthode getPath() pour récupérer le nom du fichier avec tout le chemin complet :

String nomFichier = sélection.getSelectedFile().getPath();

**Premier exemple d'utilisation avec mise en oeuvre d'un simple éditeur de texte**

Afin de bien illustrer l'utilisation des boîtes de sélection de fichier, je vous propose de mettre en oeuvre un simple éditeur de texte. Cet éditeur comporte un menu classique de fichier permettant ainsi l'ouverture et la sauvegarde d'un texte. Chacun des éléments possède un libellé, une icône, une bulle d'aide et un raccourci clavier.



**Codage correspondant**

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

import java.io.\*;

public class Dialogue extends JFrame {

private JMenuBar menu = new JMenuBar();

private JMenu fichier = new JMenu("Fichier");

private Actions actionNouveau = new Actions("Nouveau", 'N', "Tout effacer dans la zone d'édition");

private Actions actionOuvrir = new Actions("Ouvrir", 'O', "Ouvrir le fichier texte");

private Actions actionEnregistrer = new Actions("Enregistrer", 'W', "Sauvegarder le texte");

private JTextPane éditeur = new JTextPane();

public Dialogue() {

super("Les boîtes de dialogue");

setJMenuBar(menu);

menu.add(fichier);

fichier.setMnemonic('F');

fichier.add(actionNouveau);

fichier.add(actionOuvrir);

fichier.add(actionEnregistrer);

éditeur.setBackground(Color.YELLOW);

add(new JScrollPane(éditeur));

setSize(400, 300);

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

setVisible(true);

}

public static void main(String[] args) { new Dialogue(); }

private class Actions extends AbstractAction {

private String méthode;

public Actions(String libellé, char raccourci, String description) {

super(libellé, new ImageIcon(libellé.toLowerCase()+".gif"));

putValue(SHORT\_DESCRIPTION, description)

putValue(ACCELERATOR\_KEY, KeyStroke.getKeyStroke("control "+raccourci));

méthode = libellé.toLowerCase();

}

public void actionPerformed(ActionEvent e) {

try {

this.getClass().getDeclaredMethod(méthode).invoke(this);

}

catch (Exception ex) { Dialogue.this.setTitle("Problème");}

}

private void nouveau() {

Dialogue.this.setTitle("Nouveau document");

éditeur.setText("");

}

private void ouvrir() throws IOException {

JFileChooser boîte = new JFileChooser();

if (boîte.showOpenDialog(Dialogue.this)==JFileChooser.APPROVE\_OPTION) {

File fichier = boîte.getSelectedFile();

Dialogue.this.setTitle(fichier.getPath());

éditeur.read(new FileInputStream(fichier), null);

}

}

private void enregistrer() throws IOException {

JFileChooser boîte = new JFileChooser();

if (boîte.showSaveDialog(Dialogue.this)==JFileChooser.APPROVE\_OPTION) {

File fichier = boîte.getSelectedFile();

Dialogue.this.setTitle(fichier.getPath());

if (!fichier.exists()) fichier.createNewFile();

PrintWriter écriture = new PrintWriter(fichier);

écriture.println(éditeur.getText());

écriture.close();

}

}

}

}

**Choisir ses types de fichier automatiquement**

Ces étapes sont dans l'ensemble assez simples. La principale difficulté de l'implémentation d'une boîte de dialogue Fichier réside dans la spécification d'un sous-ensemble de fichiers à partir duquel l'utilisateur pourra faire sa sélection.

Nous pouvons sélectivement filtrer les fichiers afin que certains choix seulements soient affichés en passant un objet javax.swing.filechooser.FileFilter à la méthode setFileFilter() de la classe JFileChooser.

Supposons effectivement qu'un utilisateur ait à choisir un fichier d'image GIF. Le sélecteur de fichier ne devrait alors afficher que les fichiers dotés de l'extension ".gif". Il faudrait aussi qu'il puisse signifier à l'utilisateur que les fichiers affichés sont d'une certaine catégorie, comme "Image GIF". Mais la situation est parfois plus complexe. Si l'utilisateur doit choisir un fichier d'images JPEG, l'extension peut être soit ".jpg" soit ".jpeg".

Plutôt que d'élaborer un mécanisme permettant de codifier ces complexités, les concepteurs fournissent un mécanisme plus élégant : pour limiter les fichiers affichés, vous fournissez un objet qui étend la classe abstraite javax.swing.filechooser.FilFilter. Le sélecteur de fichier passe chaque fichier au filtre et affiche seulement ceux qu'il a acceptés.

Vous pouvez ainsi facilement définir vos propres filtres. Il suffit d'implémenter les deux méthodes abstraites de la superclasse FileFilter :

1. public boolean accept(File fichier) : cette méthode vérifie si un fichier doit être acceptée.
2. public String getDescription() : celle-ci renvoie une description du type de fichier qui peut être affiché dans la boîte de dialogue.

Attention, il existe dans le paquetage java.io une interface FileFilter, sans aucun rapport, et qui possède la seule méthode accept(). Elle est notamment utilisée par la méthode listFiles() de la classe File pour énumérer les fichiers d'un répertoire. Vous devrez résoudre le conflit de nom entre ces deux types homonymes, si vous importez à la fois les paqutages java.io et javax.swing.filechooser. Le remède le plus simple consiste à importer javax.swing.filechooser.FileFilter et non javax.swing.filechooser.\*.

Reportez-vous à l'étude suivante ([la gestion des fichiers](http://remy-manu.no-ip.biz/Java/Tutoriels/TechniquesSup/GestionFichiers.html#filtres)) pour bien comprendre le fonctionnement des filtres dans la classe File.  
§

Par exemple, pour filtrer les fichiers GIF, procédez de la façon suivante :

class FiltreGIF extends FileFilter {

@Override

public boolean accept(File fichier) {

return fichier.getName().toLowerCase().endsWith(".gif") || fichier.isDirectory();

}

@Override

public String getDescription() {

return "Images GIF";

}

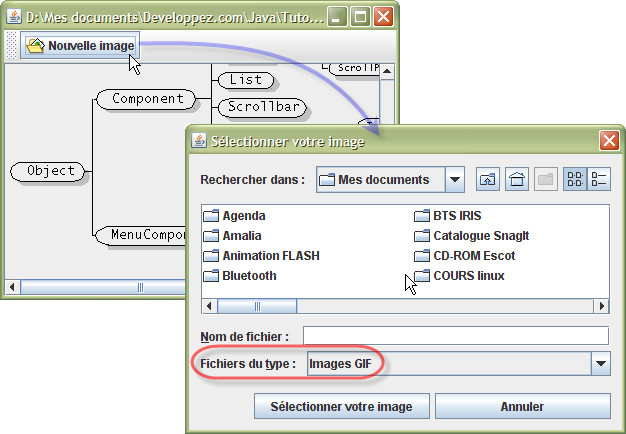
}

Une fois que vous diposez de votre filtre de fichier, utilisez la méthode setFileFilter() de la classe JFileChooser pour l'implémenter dans l'objet sélecteur de fichier :

sélection.setFileFilter(new FiltreGiF());

**Visionneuse**

A titre d'exemple, je vous propose de réaliser une petite application qui permet de visualiser une image en format GIF. La sélection de fichier se fait uniquement au travers de ces types de fichier GIF.



**Codage correspondant**

import java.io.File;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.filechooser.FileFilter;

public class Dialogue extends JFrame implements ActionListener {

private JToolBar barreBoutons = new JToolBar();

private JButton ouvrir = new JButton("Nouvelle image", new ImageIcon("ouvrir.gif"));

private JLabel image = new JLabel();

public Dialogue() {

super("Les boîtes de dialogue");

ouvrir.addActionListener(this);

barreBoutons.add(ouvrir);

add(BorderLayout.NORTH, barreBoutons);

add(new JScrollPane(image));

setSize(400, 300);

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

setVisible(true);

}

public static void main(String[] args) { new Dialogue(); }

public void actionPerformed(ActionEvent e) {

JFileChooser sélecteur = new JFileChooser();

sélecteur.setFileFilter(new FiltreGIF());

if (sélecteur.showDialog(this, "Sélectionner votre image")==JFileChooser.APPROVE\_OPTION) {

File fichier = sélecteur.getSelectedFile();

setTitle(fichier.getPath());

image.setIcon(new ImageIcon(fichier.getPath()));

}

}

private class FiltreGIF extends FileFilter {

@Override

public boolean accept(File fichier) {

return fichier.getName().toLowerCase().endsWith(".gif") || fichier.isDirectory();

}

@Override

public String getDescription() {

return "Images GIF";

}

}

}

Vous pouvez installer plusieurs filtres dans le sélecteur de fichier en utilisant la méthode addChoosableFileFilter() :

sélection.addChoosableFileFilter(new FiltreGIF());   
sélection.addChoosableFileFilter(new FiltreJPEG());

L'utilisateur sélectionne donc un filtre dans la zone liste déroulante située en bas de la boîte de dialogue. Par défaut, le filtre "Tous les fichiers" est toujours présent dans la liste. Il est recommandé de procéder ainsi pour toutes les boîtes de dialogue Fichier, dans le cas où un utilisateur de votre programme aurait besoin de sélectionner un fichier dont l'extension n'est pas standard.

Si toutefois vous vouliez supprimer cette option "Tous les fichiers", servez-vous de la méthode setAcceptAllFileFilterUsed() :

sélection.setAcceptAllFileFilterUsed(false);

Si vous utilisez un sélecteur de fichier unique pour charger et enregistrer différents types de fichiers, utilisez la méthode resetChoosableFilters() pour effacer tout filtre de fichier ancien avant d'en ajouter de nouveaux.

**Classe FileNameExtensionFilter qui permet gérer automatiquement les extensions de fichier**

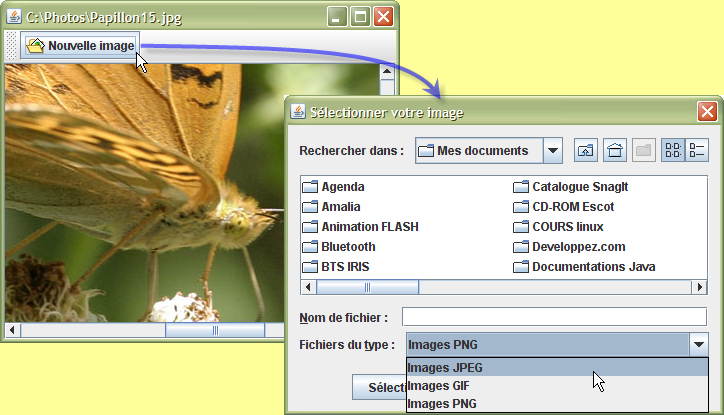
Nous pouvons réaliser des filtres très élaborés en créant une nouvelle classe personnalisée qui hérite de cette classe abstraite FileFilter et proposer alors de nouvelles fonctionnalités dans les méthodes que vous devez redéfinir : accept() et getDescription(). Mise à part les extensions de fichier, vous pouvez également contrôler plus spécialement le nom de chaque fichier pour que, par exemple, il commence tous par la lettre 'c' et que le nom du fichier ne possède pas de caractères espace (' ').

Si vous devez juste contrôler l'extension des fichiers, depuis la version JRE 6, il n'est plus nécessaire de passer par ce mécanisme. Il existe en effet une classe toute prète, FileNameExtensionFilter qui hérite de la classe FileFilter qui, comme son nom l'indique, est spécialisée sur les extensions des fichiers. Il suffit simplement de spécifier la description du type de fichier ainsi que les extensions possibles au constructeur de cette classe et le tour est joué :

FileFilter filtre = new FileNameExtensionFilter("Images JPEG", "jpg", "jpeg");  
JFileChooser sélecteur = ...;  
sélecteur.addChoosableFileFilter(filtre);

**Constructeur**

FileNameExtensionFilter(String description, String ... extensions)



import java.io.File;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.filechooser.FileFilter;

import javax.swing.filechooser.FileNameExtensionFilter;

public class Dialogue extends JFrame implements ActionListener {

private JToolBar barreBoutons = new JToolBar();

private JButton ouvrir = new JButton("Nouvelle image", new ImageIcon("ouvrir.gif"));

private JLabel image = new JLabel();

public Dialogue() {

super("Les boîtes de dialogue");

ouvrir.addActionListener(this);

barreBoutons.add(ouvrir);

add(BorderLayout.NORTH, barreBoutons);

add(new JScrollPane(image));

setSize(400, 300);

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

setVisible(true);

}

public static void main(String[] args) { new Dialogue(); }

public void actionPerformed(ActionEvent e) {

JFileChooser sélecteur = new JFileChooser();

sélecteur.addChoosableFileFilter(new FileNameExtensionFilter("Images JPEG", "jpg", "jpeg"));

sélecteur.addChoosableFileFilter(new FileNameExtensionFilter("Images GIF", "gif"));

sélecteur.addChoosableFileFilter(new FileNameExtensionFilter("Images PNG", "png"));

sélecteur.setAcceptAllFileFilterUsed(false);

if (sélecteur.showDialog(this, "Sélectionner votre image")==JFileChooser.APPROVE\_OPTION) {

File fichier = sélecteur.getSelectedFile();

setTitle(fichier.getPath());

image.setIcon(new ImageIcon(fichier.getPath()));

}

}

}

**Personnalisation de la boîte de sélection de fichiers**

Vous pouvez également personnaliser la boîte de dialogue de sélection de fichier en y ajoutant des icônes spéciales et une description pour chaque fichier affiché. Pour cela, vous devez fournir un objet d'une classe qui étend la classe FileView du paquetage javax.swing.filechooser.

Normalement, vous n'avez pas à fournir une vue de fichier, car le style d'interface dynamique (look-and-feel) s'en charge à votre place. Mais si vous souhaitez afficher différentes icônes pour des types de fichiers particuliers, vous pouvez implémenter votre propre vue de fichier.

Pour cela, vous devez étendre la classe FileView et implémenter les méthodes qui vous intéressent parmi les cinq méthodes suivantes :

1. Icon getIcon(File fichier).
2. String getName(File fichier).
3. String getDescription(File fichier).
4. String getTypeDescription(File fichier).
5. Boolean isTraversable(File fichier).

Vous appelez ensuite la méthode setFileView() pour implémenter votre vue de fichier dans le sélecteur de fichier.

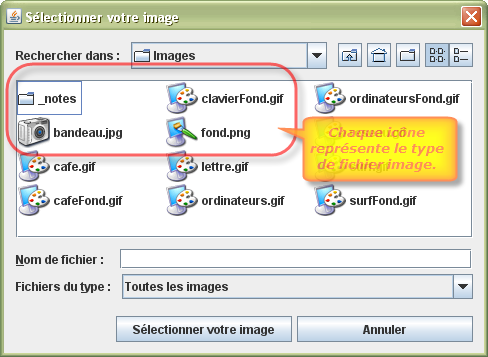
Le sélecteur appelle vos méthodes pour chaque fichier ou répertoire qu'il souhaite afficher. Si une méthode relative à l'icône, au nom ou à la description renvoie null, le sélecteur consulte alors la vue de fichier par défaut du style d'interface implémenté. Tout cela signifie que vous ne devez gérer que les types de fichiers auquels vous réservez un traitement différent.

Le sélecteur de fichier appelle la méthode isTraversable() pour décider de l'ouverture d'un répertoire lorsqu'un utilisateur clique dessus. Notez que cette méthode renvoie un objet Boolean et non une valeur boolean ! L'opération paraît étrange, elle est pourtant appropriée. Si vous voulez conserver la vue de fichier par défaut, contentez-vous de renvoyer la valeur null. Le sélecteur de fichier consultera alors la vue par défaut. En d'autres termes, la méthode renvoie un objet Boolean pour vous donner le choix entre trois options :

* vrai : Boolean.TRUE.
* faux : Boolean.FALSE.
* null : si cela n'a pas d'importance.

**Mise en oeuvre d'une vue de fichier personnalisée sur l'application précédente**

Nous allons reprendre l'application précédente afin de proposer une vue de fichier personnalisée. Nous allons ainsi faire en sorte de reconnaître le type de fichier image au moyen d'une icône spécifique.



**Codage correspondant**

import java.io.File;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.filechooser.FileFilter;

import javax.swing.filechooser.FileNameExtensionFilter;

import javax.swing.filechooser.FileView;

public class Dialogue extends JFrame implements ActionListener {

private JToolBar barreBoutons = new JToolBar();

private JButton ouvrir = new JButton("Nouvelle image", new ImageIcon("ouvrir.gif"));

private JLabel image = new JLabel();

public Dialogue() {

super("Les boîtes de dialogue");

ouvrir.addActionListener(this);

barreBoutons.add(ouvrir);

add(BorderLayout.NORTH, barreBoutons);

add(new JScrollPane(image));

setSize(400, 300);

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

setVisible(true);

}

public static void main(String[] args) { new Dialogue(); }

public void actionPerformed(ActionEvent e) {

JFileChooser sélecteur = new JFileChooser();

sélecteur.addChoosableFileFilter(new FileNameExtensionFilter("Images GIF", "gif"));

sélecteur.addChoosableFileFilter(new FileNameExtensionFilter("Images PNG", "png"));

sélecteur.addChoosableFileFilter(new FileNameExtensionFilter("Images JPEG", "jpg", "jpeg"));

sélecteur.addChoosableFileFilter(new FileNameExtensionFilter("Toutes les images", "gif", "png", "jpg", "jpeg"));

sélecteur.setFileView(new VueDeFichier());

if (sélecteur.showDialog(this, "Sélectionner votre image")==JFileChooser.APPROVE\_OPTION) {

File fichier = sélecteur.getSelectedFile();

setTitle(fichier.getPath());

image.setIcon(new ImageIcon(fichier.getPath()));

}

}

private class VueDeFichier extends FileView {

private Icon photo = new ImageIcon("photo.gif");

private Icon palette = new ImageIcon("palette.gif");

private Icon pinceau = new ImageIcon("pinceau.gif");

@Override

public Icon getIcon(File fichier) {

if (fichier.isFile()) {

String nom = fichier.getName().toLowerCase();

if (nom.endsWith(".gif")) return palette;

else if (nom.endsWith(".jpeg") || nom.endsWith(".jpg")) return photo;

else if (nom.endsWith(".png")) return pinceau;

else return null;

}

else return null;

}

}

}

**Prévisualisation de fichier**

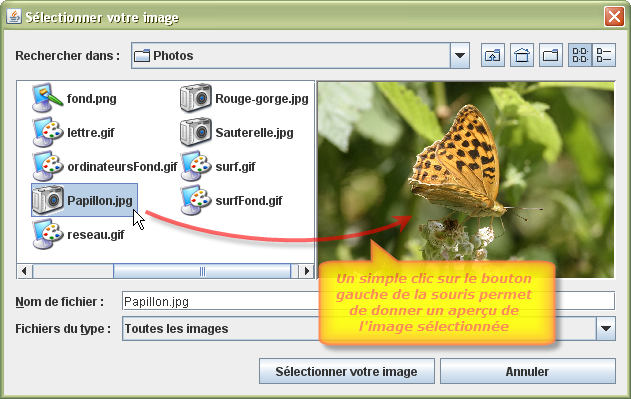
Enfin, vous pouvez personnaliser la boîte de dialogue de sélection de fichiers en ajoutant un composant accessoire. Cet accessoire peut, par exemple, donner un aperçu du fichier sélectionné en affichant une vue miniature du fichier actuellement sélectionné. L'accessoire se place systématiquement sur la partie EST de la boîte de dialogue.

Un accessoire peut être n'importe quel composant Swing. Il suffit de l'associer ensuite à votre boîte de sélection de fichier au travers de la méthode setAccessory() de la classe JFileChooser.

Attention : Nous voulons mettre à jour l'image d'aperçu chaque fois que l'utilisateur sélectionne un fichier différent. Le sélecteur de fichier utilise le mécanisme des JavaBeans de notification aux écouteurs intéressés chaque fois qu'une de ses propriétés change. Le fichier sélectionné est une propriété que vous pouvez surveillé en installant un PropertyChangeListener et en redéfinissant la méthode propertyChange().

**Proposer un aperçu sur chaque fichier image sélectionnée**

Pour bien comprendre ce mécanisme, je vous propose de reprendre l'application précédente à laquelle nous allons rajouter un accessoire qui donne un aperçu de l'image sélectionnée. Cet accessoire est un simple JLabel dont l'icône évolue au court du temps (suivant la sélection du fichier) :



**codage correspondant**

import java.beans.\*;

import java.io.File;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.border.EtchedBorder;

import javax.swing.filechooser.FileNameExtensionFilter;

import javax.swing.filechooser.FileView;

public class Dialogue extends JFrame implements ActionListener {

private JToolBar barreBoutons = new JToolBar();

private JButton ouvrir = new JButton("Nouvelle image", new ImageIcon("ouvrir.gif"));

private JLabel image = new JLabel();

public Dialogue() {

super("Les boîtes de dialogue");

ouvrir.addActionListener(this);

barreBoutons.add(ouvrir);

add(BorderLayout.NORTH, barreBoutons);

add(new JScrollPane(image));

setSize(400, 300);

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

setVisible(true);

}

public static void main(String[] args) { new Dialogue(); }

public void actionPerformed(ActionEvent e) {

JFileChooser sélecteur = new JFileChooser();

sélecteur.addChoosableFileFilter(new FileNameExtensionFilter("Images GIF", "gif"));

sélecteur.addChoosableFileFilter(new FileNameExtensionFilter("Images PNG", "png"));

sélecteur.addChoosableFileFilter(new FileNameExtensionFilter("Images JPEG", "jpg", "jpeg"));

sélecteur.addChoosableFileFilter(new FileNameExtensionFilter("Toutes les images", "gif", "png", "jpg", "jpeg"));

sélecteur.setFileView(new VueDeFichier());

sélecteur.setAccessory(new Aperçu(sélecteur));

if (sélecteur.showDialog(this, "Sélectionner votre image")==JFileChooser.APPROVE\_OPTION) {

File fichier = sélecteur.getSelectedFile();

setTitle(fichier.getPath());

image.setIcon(new ImageIcon(fichier.getPath()));

}

}

private class VueDeFichier extends FileView {

private Icon photo = new ImageIcon("photo.gif");

private Icon palette = new ImageIcon("palette.gif");

private Icon pinceau = new ImageIcon("pinceau.gif");

@Override

public Icon getIcon(File fichier) {

if (fichier.isFile()) {

String nom = fichier.getName().toLowerCase();

if (nom.endsWith(".gif")) return palette;

else if (nom.endsWith(".jpeg") || nom.endsWith(".jpg")) return photo;

else if (nom.endsWith(".png")) return pinceau;

else return null;

}

else return null;

}

}

private class Aperçu extends JLabel {

public Aperçu(JFileChooser sélecteur) {

setPreferredSize(new Dimension(300, 200));

setBorder(BorderFactory.createLoweredBevelBorder());

sélecteur.addPropertyChangeListener(new PropertyChangeListener() {

public void propertyChange(PropertyChangeEvent evt) {

if (evt.getPropertyName().equals(JFileChooser.SELECTED\_FILE\_CHANGED\_PROPERTY)) {

File fichier = (File)evt.getNewValue();

if (fichier==null) { setIcon(null); return; }

ImageIcon vignette = new ImageIcon(fichier.getPath());

setIcon(new ImageIcon(vignette.getImage().getScaledInstance(getWidth(), -1, Image.SCALE\_DEFAULT)));

}

}

});

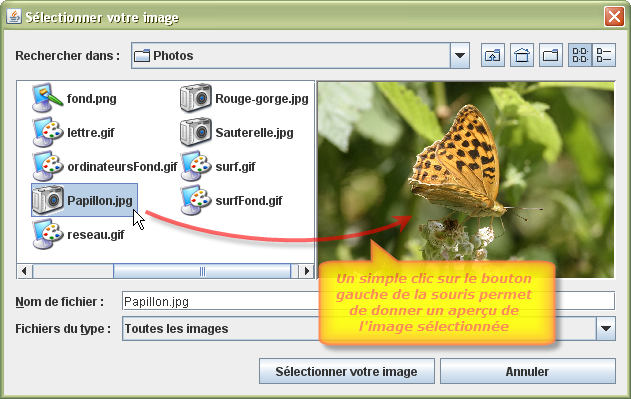
};

}

}

**Ouvrir la boîte de sélection de fichier sur le dernier répertoire consulté**

Généralement, il est souhaitable de faire en sorte que lorsque nous consultons de nouveau notre boîte de sélection de fichier, nous soyons automatiquement replacé sur le dernier répertoire sur lequel nous avons effectué notre sélection. Rassurez-vous la classe JFileChooser le fait automatiquement. Par contre, il faut ne pas détruire l'objet correspondant, ce qui sous-entends qu'il est nécessaire que votre objet de type JFileChooser soit un attribut de la classe qui l'utilise et qu'il ne soit donc pas créé dans la méthode qui gére l'événement.



**Codage correspondant**

import java.beans.\*;

import java.io.File;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.filechooser.FileNameExtensionFilter;

import javax.swing.filechooser.FileView;

public class Dialogue extends JFrame implements ActionListener {

private JToolBar barreBoutons = new JToolBar();

private JButton ouvrir = new JButton("Nouvelle image", new ImageIcon("ouvrir.gif"));

private JLabel image = new JLabel();

JFileChooser sélecteur = new JFileChooser("C:/Photos");

public Dialogue() {

super("Les boîtes de dialogue");

sélecteur.addChoosableFileFilter(new FileNameExtensionFilter("Images GIF", "gif"));

sélecteur.addChoosableFileFilter(new FileNameExtensionFilter("Images PNG", "png"));

sélecteur.addChoosableFileFilter(new FileNameExtensionFilter("Images JPEG", "jpg", "jpeg"));

sélecteur.addChoosableFileFilter(new FileNameExtensionFilter("Toutes les images", "gif", "png", "jpg", "jpeg"));

sélecteur.setFileView(new VueDeFichier());

sélecteur.setAccessory(new Aperçu());

ouvrir.addActionListener(this);

barreBoutons.add(ouvrir);

add(BorderLayout.NORTH, barreBoutons);

add(new JScrollPane(image));

setSize(400, 300);

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

setVisible(true);

}

public static void main(String[] args) { new Dialogue(); }

public void actionPerformed(ActionEvent e) {

if (sélecteur.showDialog(this, "Sélectionner votre image")==JFileChooser.APPROVE\_OPTION) {

File fichier = sélecteur.getSelectedFile();

setTitle(fichier.getPath());

image.setIcon(new ImageIcon(fichier.getPath()));

}

}

private class VueDeFichier extends FileView {

private Icon photo = new ImageIcon("photo.gif");

private Icon palette = new ImageIcon("palette.gif");

private Icon pinceau = new ImageIcon("pinceau.gif");

@Override

public Icon getIcon(File fichier) {

if (fichier.isFile()) {

String nom = fichier.getName().toLowerCase();

if (nom.endsWith(".gif")) return palette;

else if (nom.endsWith(".jpeg") || nom.endsWith(".jpg")) return photo;

else if (nom.endsWith(".png")) return pinceau;

else return null;

}

else return null;

}

}

private class Aperçu extends JLabel {

public Aperçu() {

setPreferredSize(new Dimension(300, 200));

setBorder(BorderFactory.createLoweredBevelBorder());

sélecteur.addPropertyChangeListener(new PropertyChangeListener() {

public void propertyChange(PropertyChangeEvent evt) {

if (evt.getPropertyName().equals(JFileChooser.SELECTED\_FILE\_CHANGED\_PROPERTY)) {

File fichier = (File)evt.getNewValue();

if (fichier==null) { setIcon(null); return; }

ImageIcon vignette = new ImageIcon(fichier.getPath());

setIcon(new ImageIcon(vignette.getImage().getScaledInstance(getWidth(), -1, Image.SCALE\_DEFAULT)));

}

}

});

};

}

}

**[Choix du chapitre](http://remy-manu.no-ip.biz/Java/Tutoriels/IHM/dialogue.html)Sélecteurs de couleur**

Comme vous l'avez vu dans la section précédente, un sélecteur de fichier performant est un composant d'interface complexe, et il est hors de question que vous l'implémentiez vous-même. De nombreuses boîtes à outils d'interface utilisateur fournissent d'autres boîtes de dialogue communes pour sélectionner une date/heure, une valeur monétaire, une police, une couleur, etc. L'intérêt est double. Les programmeurs peuvent simplement utiliser une implémentation de qualité, plutôt que de créer la leur propre. Et les utilisateurs ont une expérience intuitive de ces outils de sélection.

Actuellement Swing ne fournit qu'un seul sélecteur supplémentaire, JColorChooser. Il permet aux utilisateurs de choisir une valeur de couleur. Tout comme la classe JFileChooser, le sélecteur de couleur est un composant, et non une boîte de dialogue, mais il contient des méthodes pour créer des boîtes de dialogue contenant un composant pour la sélection d'une couleur (voir plus loin).

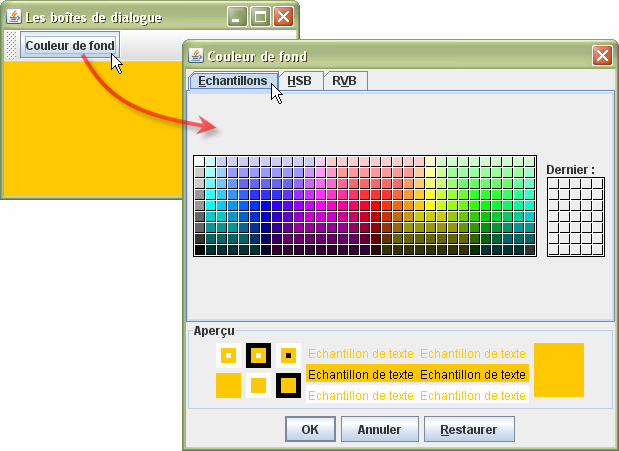
**Passer par la méthode showDialog()**

Ce composant JColorChooser permet donc à l'utilisateur de sélectionner une couleur. La manière la plus simple de l'employer consiste à appelez la méthode statique showDialog(), en spécifiant un composant parent (propriétaire) pour le dialogue, un titre et une couleur initiale par défaut :

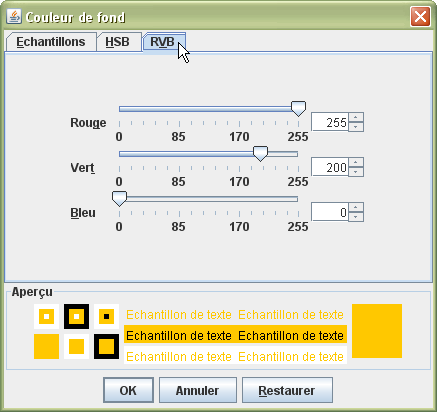
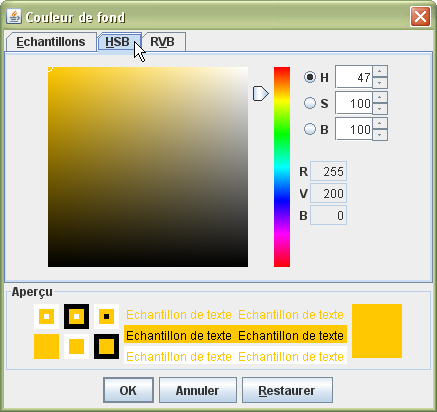
Color couleur = JColorChooser.showDialog(parent, "Titre du dialogue", couleurInitiale);

Cette méthode crée un JColorChooser dans un JDialog modal et bloque jusqu'à ce que l'utilisateur le ferme. S'il a été fermé par le bouton OK, la méthode renvoie la couleur sélectionnée. Si l'utilisateur le ferme d'une autre façon, showDialog() renvoie null.

A titre d'exemple, je vous propose une application très rudimentaire qui permet à l'utilisateur de choisir la couleur de fond de la fenêtre principale :



Voici les autres vues de la boîte de sélections de couleur :



**codage correspondant**

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

public class Dialogue extends JFrame {

private Color couleurFond = Color.ORANGE;

private JToolBar barre = new JToolBar();

private JButton choix = new JButton("Couleur de fond");

public Dialogue() {

super("Les boîtes de dialogue");

choix.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

Color couleur = JColorChooser.showDialog(Dialogue.this, "Couleur de fond", couleurFond);

if (couleur!=null) getContentPane().setBackground(couleurFond = couleur);

}

});

barre.add(choix);

add(barre, BorderLayout.NORTH);

getContentPane().setBackground(couleurFond);

setSize(300, 200);

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

setVisible(true);

}

public static void main(String[] args) { new Dialogue(); }

}

**Boîte de dialogue de sélection de couleur non modale - createDialog()**

Si nous désirons plus de contrôle sur le dialogue de sélection de couleur, nous pouvons appelez la méthode createDialog(). Cette méthode statique crée un JDialog qui contient un composant JColorChooser donné. Elle permet de spécifier les objets java.awt.event.ActionListener devant répondre aux boutons OK et Annuler. Le dialogue ne s'affiche pas automatiquement, et il n'attend pas que l'utilisateur ait fait une sélection.

Vous pouvez ainsi afficher une boîte de dialogue sélecteur de couleur non modale à l'aide de cette méthode createDialog(). Vous devez alors fournir :

1. un composant parent ;
2. le titre de la boîte de dialogue ;
3. un indicateur pour séletionner une boîte de dialogue, soit modale, soit non modale ;
4. un sélecteur de couleur ;
5. des écouteurs pour les boutons OK et Annuler (ou null si vous ne voulez pas d'écouteur).

JDialog dialogue = JColorChooser.showDialog(

Composant parent, // Le composant où doit apparaître la boîte de dialogue.

String titre, // Le titre de la boîte de dialogue.

boolean modale, // true si cet appel doit empêcher toute autre action jusqu'à la fermeture de la boîte de dialogue.

JColorChooser couleur, // Le sélecteur de couleur à ajouter à la boîte de dialogue.

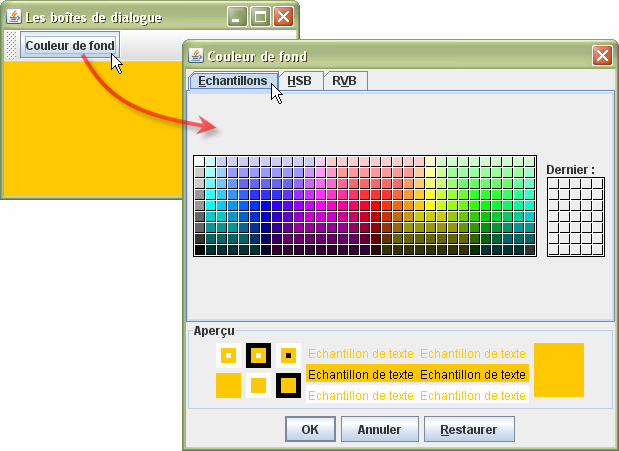
ActionListener écouteurOK, // écouteur spécifique au bouton OK.

ActionListener écouteurAnnuler // écouteur spécifique au bouton Annuler.

);

J'en profite pour vous indiquer des méthodes souvent utiles de cette classe JColorChooser :

1. JColorChooser() : Construit un sélecteur de couleur avec le blanc comme couleur initiale.
2. JColorChooser(Color couleurInitiale) : Construit également un sélecteur en spécifiant cette fois-ci la couleur initiale.
3. Color getColor() et void setColor(Color nouvelleCouleur): Permettent d'obtenir et de définir la couleur actuelle de ce sélecteur de couleur.
4. ColorSelectionModel getSelectionModel() : Récupère le modèle de sélection intégré dans la boîte de sélection de couleur (voir plus loin).



**codage correspondant**

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

public class Dialogue extends JFrame {

private JToolBar barre = new JToolBar();

private JButton choix = new JButton("Couleur de fond");

private JColorChooser couleurFond = new JColorChooser(Color.ORANGE);

private JDialog dialogue;

public Dialogue() {

super("Les boîtes de dialogue");

dialogue = JColorChooser.createDialog(Dialogue.this, "Couleur de fond", false, couleurFond, new ActionListener() {

public void actionPerformed(ActionEvent e) {

Dialogue.this.getContentPane().setBackground(couleurFond.getColor());

}

}, null);

choix.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

dialogue.setVisible(true);

}

});

barre.add(choix);

add(barre, BorderLayout.NORTH);

getContentPane().setBackground(couleurFond.getColor());

setSize(300, 200);

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

setVisible(true);

}

public static void main(String[] args) { new Dialogue(); }

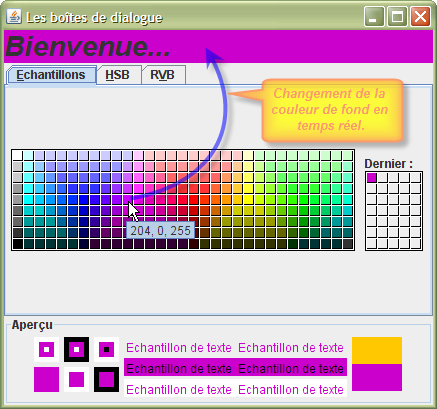
}

**Intégrer un sélecteur de couleur dans un autre composant**

Vous pouvez faire encore mieux, et renvoyer immédiatement à l'utilisateur des informations sur la sélection de couleur. Pour surveiller les sélections de couleur, vous devez alors récupérer le modèle de sélection du sélecteur de couleur au moyen de la méthode getSelectionModel() de la classe JColorChooser et ajouter un écouteur de changement ChangeListener au moyen de la méthode addChangeListener() prévu par ce modèle.

Dans ce cas, les boutons OK et Annuler fournis par la boite de dialogue de sélection de couleur n'ont aucun intérêt. Nous pouvons ainsi créer et manipuler un JColorChooser spécifique en le plaçant dans une boîte de dialogue non modale ou même sur n'importe quel type de conteneur.

A titre d'exemple, Nous reprenons l'application précédente et nous intégrons directement le sélecteur de couleur sur la partie sud de la fenêtre principale :



**codage correspondant**

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

import javax.swing.event.\*;

public class Dialogue extends JFrame {

private JLabel bienvenue = new JLabel("Bienvenue...");

private JColorChooser couleurFond = new JColorChooser(Color.ORANGE);

public Dialogue() {

super("Les boîtes de dialogue");

bienvenue.setFont(new Font("Arial", Font.BOLD+Font.ITALIC, 28));

add(bienvenue);

couleurFond.getSelectionModel().addChangeListener(new ChangeListener() {

public void stateChanged(ChangeEvent e) {

Dialogue.this.getContentPane().setBackground(couleurFond.getColor());

}

});

add(couleurFond, BorderLayout.SOUTH);

getContentPane().setBackground(couleurFond.getColor());

pack();

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

setVisible(true);

}

public static void main(String[] args) { new Dialogue(); }

}

**[Choix du chapitre](http://remy-manu.no-ip.biz/Java/Tutoriels/IHM/dialogue.html)Les boîtes de dialogue personnalisées**

Dans les paragraphes précédents, nous vous avons présenté des boîtes de dialogue standard. La plupart du temps, pour les utiliser, il suffisait de faire appel à une des méthodes statiques, comme par exemple showInputDialog() de JOptionPane, et qui réalisait les opérations suivantes :

1. Création de l'objet boîte de dialogue.
2. Affichage.
3. Gestion du dialogue avec l'utilisateur.
4. Fermeture de la boîte de dialogue.
5. Transfert éventuel d'information (par le biais de valeur de retour).

Mais il vous arrivera d'avoir besoin de boîtes de dialogue plus élaborées. Pour ce faire, Java dispose d'une classe JDialog qui vous permettra de créer vos propre boîtes de dialogue. Nous allons voir, durant ce chapitre, comment la mettre en oeuvre et comment prendre nous-mêmes en charge les différentes opérations évoquées ci-dessus.

Toutefois, nous avons vu, qu'avec les boîtes de dialogue standard, il est tout à fait possible de réaliser des boîtes de dialogue déjà très élaborées, comme, par exemple, la mise en place d'une connexion réseau avec authentification de l'utilisateur. Il est souvent plus simple de prendre ces boîtes de dialogue standard puisque tous les points évoqués plus haut se font automatiquement. Dans le cas de boîte de dialogue totalement personnalisées, à l'aide de la classe JDialog, vous êtes obligé de traiter chacun de ces points.

**Processus de mise en oeuvre d'une boîte de dialogue personnalisée**

Pour implémenter une boîte de dialogue, vous devez dériver une classe de JDialog. Ce processus est identique à celui qui permet de dériver la fenêtre principale à partir de la classe JFrame pour une application. Voici les étapes à suivre :

1. Dans le constructeur de la boîte de dialogue, appelez le constructeur de la classe de base JDialog. Vous devez fournir le nom du cadre propriétaire (la fenêtre cadre dans laquelle s'affiche la boîte de dialogue), le titre de la boîte de dialogue et un indicateur booléen spécifiant si la boîte de dialogue doit être modale ou non.

Vous devez indiquer le cadre propriétaire afin que la boîte de dialogue soit affichée par-dessus. Les systèmes de fenêtrage requièrent que toute fenêtre affichable appartienne à un autre cadre. Vous pouvez aussi fournir un propriétaire null, mais c'est un peu risqué, la boîte de dialogue pourrait être masquée derrière d'autre fenêtres (les boîtes de dialogue ayant un propriétaire null appartiennent en fait à un cadre masqué et partagé).

1. Ajoutez les composants d'interface de la boîte de dialogue.
2. Ajoutez les gestionnaires d'événement.
3. Définissez la taille de la boîte de dialogue.

**Phase de construction**

La façon la plus usuelle de construire un objet boîte de dialogue se présente de la façon suivante :

JFrame application = new JFrame("Fenêtre principale");

JDialog dialogue = new JDialog(

Composant parent, // Le composant où doit apparaître la boîte de dialogue.

String titre, // Le titre de la boîte de dialogue.

boolean modale, // true si cet appel doit empêcher toute autre action jusqu'à la fermeture de la boîte de dialogue.

);

Le dernier argument du constructeur précise donc si la boîte doit être modale (true) ou non modale (false). Avec une boîte modale, l'utilisateur ne peut pas agir sur d'autres composants que ceux de la boîte tant qu'il n'a pas mis fin au dialogue. Comme nous l'avons vu, les boîtes de dialogue standard de Java sont modales comme la plupart des boîtes de dialogue des logiciels du commerce.

**Affichage**

Comme pour une fenêtre, il est nécessaire d'attribuer une taille à une boîte de dialogue par la méthode setSize() (ou éventuellement la méthode pack()) et d'en provoquer l'affichage par la méthode setVisible().

dialogue.setSize(300, 250);  
dialogue.setVisible(true);

Il est très important de noter que, à partir du moment où nous avons affaire à une boîte modale, le système n'exécutera pas l'instruction suivant l'appel de setVisible(), tant que l'utilisateur n'aura pas mis fin au dialogue.

**Mise en oeuvre d'une classe dérivée de JDialog**

En pratique, nous serons souvent amené, comme pour JFrame, à créer une classe dérivée de JDialog, que nous pourrons ainsi spécialiser en introduisant les attributs et les fonctionnalités dont nous aurons besoin.

Pour qu'elle présente un intérêt, une boîte de dialogue devra bien entendu comporter des composants et, généralement, être en mesure de transmettre des informations. La plupart du temps, elle comportera un bouton OK destiné à valider les informations que l'utilisateur aura pu y entrer ; parfois, elle comportera aussi un bouton Annuler, permettant à l'utilisateur d'abandonner le dialogue sans que les informations éventuellement saisies soient prises en compte.

**Introduction des composants**

Nous introduisons les composants dans une boîte de dialogue exactement comme dans une fenêtre. Autrement dit, nous ajoutons un composant au moyen de la méthode add(). Par défaut la boîte de dialogue est dotée d'un gestionnaire de mise en forme de type BorderLayout.

Bien entendu, ces différentes opérations peuvent tout à fait être réalisées dans le constructeur de la boîte de dialogue, pour peu que nous fassions une classe spécialisée, dérivée de JDialog.

**Gestion du dialogue**

En général, nous n'auront pas à nous préoccuper de gérer les différents contrôles contenus dans la boîte de dialogue, exception faite du bouton OK qui devra mettre fin au dialogue. Cette action doit alors être gérée par le gestionnaire d'événement associé au bouton OK en activant la méthode setVisible(false) de la boîte de dialogue.

Notez bien le double fonction de cet appel de méthode : il rend invisible la boîte de dialogue mais, de plus, il met fin au dialogue, permettant ainsi à l'exécution de se poursuivre après l'instruction d'affichage de la boîte (si nous avons choisi d'avoir une boîte modale).

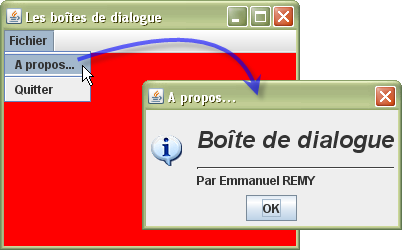
Sachez également qu'un tel appel est réalisé automatiquement par la classe JDialog en cas de fermeture de la boîte par l'utilisateur. Il n'a donc pas à être prévu explicitement.

En général, l'action sur OK sert à valider les informations fournies par l'utilisateur alors qu'une fermeture de la boîte (ou, éventuellement, une action sur le bouton Annuler) annule le dialogue. Nous serons donc souvent amené à utiliser dans la boîte de dialogue spécialisée, un attribut booléen (nommé par exemple ok) qui sera initialisé à false à l'affichage de la boîte et que nous placerons à true en cas d'action sur le bouton OK.

Si l'utilisateur clique sur la case de fermeture (X) de la fenêtre, la boîte de dialogue est également fermée. Comme pour la classe JFrame, vous pouvez redéfinir ce comportement au moyen de la méthode setDefaultCloseOperation().

**Mise en oeuvre d'une boîte "A propos... "**

Afin de valider tout ce qui vient d'être évoqué, je vous propose de mettre en oeuvre une boîte de dialogue "A propos..." dont voici le fonctionnement ci-dessous :



**codage correspondant**

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

public class Dialogue extends JFrame {

private JMenuBar barre = new JMenuBar();

private JMenu fichier = new JMenu("Fichier");

private APropos boîte;

public Dialogue() {

super("Les boîtes de dialogue");

setJMenuBar(barre);

barre.add(fichier);

fichier.add("A propos...").addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

if (boîte==null) boîte = new APropos();

boîte.setVisible(true);

}

});

fichier.addSeparator();

fichier.add("Quitter").addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

System.exit(0);

}

});

getContentPane().setBackground(Color.RED);

setSize(300, 250);

setDefaultCloseOperation(DO\_NOTHING\_ON\_CLOSE);

setVisible(true);

}

public static void main(String[] args) { new Dialogue(); }

private class APropos extends JDialog implements ActionListener {

private String html = "<html><h1><i>Boîte de dialogue</i></h1><hr>Par Emmanuel REMY</html>";

private JLabel texte = new JLabel(html, new ImageIcon("info.gif"), JLabel.CENTER);

private JPanel sud = new JPanel();

private JButton ok = new JButton("OK");

public APropos() {

super(Dialogue.this, "A propos...", true);

setDefaultCloseOperation(DO\_NOTHING\_ON\_CLOSE);

setSize(260, 150);

ok.addActionListener(this);

sud.add(ok);

add(texte);

add(sud, BorderLayout.SOUTH);

texte.setIconTextGap(15);

}

public void actionPerformed(ActionEvent e) {

setVisible(false);

}

}

}

Par rapport aux boîtes de dialogue standard, vous avez beaucoup de choses à prendre en compte. Pour cette boîte "A propos...", il aurait mieux valu passer une boîte de message standard. Pour justifier le choix d'une boîte de dialogue personnalisée, il faut que vous ayez besoin de beaucoup de composants internes avec un placement bien précis.

**Echange de données**

Nous concevons le plus souvent une boîte de dialogue dans le but de recueillir des informations de la part de l'utilisateur. Vous avez pu constater que la création d'un objet boîte de dialogue est une opération qui demeure simple : il suffit de fournir les données initiales, puis d'appeler la méthode setVisible(true) pour afficher la boîte à l'écran. Nous allons maintenant étudier comment des données peuvent être transférées dans une boîte de dialogue, ou extraites.

Dans le cas le plus général, il faudra prévoir :

1. Transmettre des informations à la boîte de dialogue afin qu'elle initialise les valeurs affichées par certains de ses contrôles : pour cela, la plupart du temps, la boîte de dialogue doit fournir des méthodes pour définir les valeurs par défaut.
2. Une fois que vous avez, le cas échéant, défini les valeurs par défaut, vous affichez la boîte dialogue en appelant la méthode setVisible(true).

Même si vous n'avez pas besoin d'initaliser certaines valeurs, il est souvent souhaitable de créer une méthode spécifique dans votre boîte de dialogue personnalisée qui s'occupera de lancer cette méthode setVisible(true). Ainsi, lorsque vous désirerez afficher votre boîte de dialogue, vous passerez par cette méthode particulière. Vous profiterez de cette méthode pour gérer un attribut interne à la boîte de dialogue qui vous indiquera si l'utilisateur à cliquer sur le bouton OK ou sur le bouton Annuler. Il serait d'ailleurs souhaitable que cette méthode retourne un booléen relatif au choix du bouton de l'utilisateur.

1. L'utilisateur fournit les informations et clique sur le bouton OK ou Annuler. Les gestionnaires d'événement pour les deux boutons appellent setVisible(false), qui termine l'appel à setVisible(true). L'utilisateur peut aussi fermer la boîte de dialogue. Si vous n'avez pas installé d'écouteur de fenêtre pour la boîte de dialogue, le processus de fermeture de fenêtre par défaut s'applique : la boîte de dialogue devient invisible, ce qui a ausi pour effet de mettre fin à l'appel à setVisible(true).

Ce dont il faut se souvenir, c'est que l'appel à setVisible(true) se bloque jusqu'à ce que l'utilisateur ait renvoyé la boîte de dialogue. Ceci facilite l'implémentation des boîtes de dialogue modales.

Le transfert de données à partir d'une boîte de dialogue non modale n'est pas une opération facile. Lors de l'affichage d'une telle boîte de dialogue, l'appel de la méthode setVisible() n'est pas bloquant et le programme poursuit son exécution. Si l'utilisateur sélectionne une option puis clique sur OK, la boîte de dialogue doit envoyer un événement à un écouteur dans le programme.

1. A partir de maintenant, il faudra savoir si l'utilisateur a accepté ou annulé la boîte de dialogue. Généralement vous définissez un attribut ok sur false avant d'afficher le boîte de dialogue. Seul le gestionnaire d'événement du bouton OK définit l'attribut ok sur true. Dans ce cas, vous pouvez récupérer la saisie de l'utilisateur dans la boîte de dialogue.
2. Dans le cas d'une validation, il faut récupérer les informations entrées par l'utilisateur dans la boîte de dialogue, au moment de sa fermeture. La plupart du temps, la boîte de dialogue doit alors fournir des méthodes pour récupérer les valeurs saisies (en les formatant éventuellement).

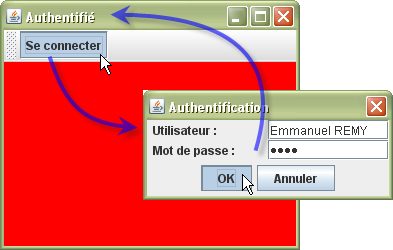
Une autre façon de parvenir à cet échange de données (en entrée et en sortie), consiste à créer un objet, par exemple d'une classe Infos, destiné à assurer cet archivage d'informations. En général, nous pourrons nous contenter d'attributs publics représentant les valeurs des contrôles concernés (en entrée ou en sortie), même si cela ne respecte pas le principe d'encapsulation.

Une autre approche consiste à réaliser le maximum de traitement directement à l'intérieur de la boîte de dialogue et de renvoyer juste la ou les valeurs finalisées.

Une dernière façon de procéder consiste à mettre en oeuvre votre boîte de dialogue personnalisée comme classe interne de la classe principale de l'application. De cette manière, il devient plus facile d'échanger des informations puisque les attributs de la classe principale sont toujours visibles depuis une classe interne. Par ailleurs, nous pouvons envisager de placer la gestion d'exception directement sur la classe principale de l'application afin de permettre une interactivité en temps réel entre la boîte de dialogue et la fenêtre principale. Dans ce cas, nous pouvons éventuellement nous passer de boutons de validation.

**Mise en oeuvre d'une boîte d'authentification**

Afin de bien maîtriser les échanges d'information entre la boîte de dialogue personnalisée et la fenêtre principale de l'application, je vous propose de fabriquer une boîte de dialogue d'authentification dont le focntionnement est décrit ci-desous :



**codage correspondant**

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

public class Dialogue extends JFrame implements ActionListener {

private JToolBar barre = new JToolBar();

private JButton connecter = new JButton("Se connecter");

private Authentification authentifier;

public Dialogue() {

super("Les boîtes de dialogue");

connecter.addActionListener(this);

barre.add(connecter);

add(barre, BorderLayout.NORTH);

getContentPane().setBackground(Color.RED);

setSize(300, 250);

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

setVisible(true);

}

public void actionPerformed(ActionEvent e) {

if (authentifier==null) authentifier = new Authentification();

if (authentifier.afficher()) setTitle(authentifier.getRésultat());

}

public static void main(String[] args) { new Dialogue(); }

private class Authentification extends JDialog {

private JTextField utilisateur = new JTextField();

private JPasswordField motDePasse = new JPasswordField();

private JPanel panneau = new JPanel();

private JButton ok = new JButton("OK");

private JButton annuler = new JButton("Annuler");

private JPanel sud = new JPanel();

private boolean validation;

private String résultat;

public Authentification() {

super(Dialogue.this, "Authentification", true);

setSize(250, 110);

panneau.setLayout(new GridLayout(2, 2));

panneau.add(new JLabel(" Utilisateur :"));

panneau.add(utilisateur);

panneau.add(new JLabel(" Mot de passe :"));

panneau.add(motDePasse);

add(panneau);

ok.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

validation = true;

résultat = utilisateur.getText().equalsIgnoreCase("Emmanuel REMY") ?

motDePasse.getText().equals("manu") ? "Authentifié" : "Mot de passe incorrect" :

"Utilisateur inconnu";

setVisible(false);

}

});

sud.add(ok);

annuler.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

setVisible(false);

}

});

sud.add(annuler);

add(sud, BorderLayout.SOUTH);

}

public boolean afficher() {

validation = false;

résultat = "Aucun";

motDePasse.setText("");

setVisible(true);

return validation;

}

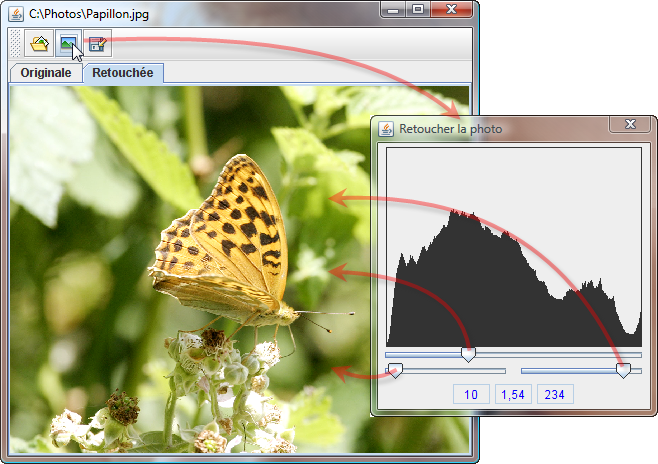
public String getRésultat() { return résultat; }

}

}

**Mise en oeuvre d'un petit logiciel de retouche**

Pour conclure ce chapitre, je vous propose de mettre en place une boîte de dialogue non modale qui représente un histogramme afin de permettre le réglage des niveaux sur une photo que nous désirons retoucher. L'intérêt ici de prendre une boîte non modale permet de vérifier la différence entre la photo retouchée et la photo originale. L'intérêt également d'utiliser la classe JDialog au lieu d'une simple classe JFrame, c'est que la boîte de dialogue est toujours au dessus de la fenêtre principale, elle est donc toujours accessible pour permettre les différents réglages.



**codage correspondant**

import java.beans.\*;

import java.io.File;

import java.io.IOException;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

import java.awt.geom.\*;

import java.awt.image.\*;

import java.text.\*;

import javax.imageio.ImageIO;

import javax.swing.event.\*;

import javax.swing.filechooser.FileNameExtensionFilter;

public class Dialogue extends JFrame {

private JToolBar barreBoutons = new JToolBar();

private JFileChooser sélecteur = new JFileChooser("C:/Photos");

private Photo originale = new Photo();

private Photo retouche = new Photo();

private JTabbedPane onglets = new JTabbedPane();

private Retouche retoucher = new Retouche();

public Dialogue() {

super("Les boîtes de dialogue");

sélecteur.addChoosableFileFilter(new FileNameExtensionFilter("Images JPEG", "jpg", "jpeg"));

sélecteur.setAccessory(new Aperçu());

barreBoutons.add(new AbstractAction("Nouvelle image", new ImageIcon("ouvrir.gif")) {

public void actionPerformed(ActionEvent e) {

if (sélecteur.showDialog(Dialogue.this, "Sélectionner votre image")==JFileChooser.APPROVE\_OPTION) {

File fichier = sélecteur.getSelectedFile();

setTitle(fichier.getPath());

BufferedImage source;

try {

source = ImageIO.read(fichier);

originale.setPhoto(source);

BufferedImage copie = new BufferedImage(source.getWidth(), source.getHeight(), source.getType());

copie.setData(source.getData());

retouche.setPhoto(copie);

retoucher.réajuster();

}

catch (IOException ex) {

setTitle("Problème avec le fichier image");

}

if (retoucher.isVisible()) retoucher.afficher();

}

}

});

barreBoutons.add(new AbstractAction("Retoucher", new ImageIcon("retouche.gif")) {

public void actionPerformed(ActionEvent e) {

retoucher.afficher();

}

});

barreBoutons.add(new AbstractAction("Enregistrer", new ImageIcon("enregistrer.gif")) {

public void actionPerformed(ActionEvent e) {

if (sélecteur.showSaveDialog(Dialogue.this)==JFileChooser.APPROVE\_OPTION) {

try {

File fichier = sélecteur.getSelectedFile();

ImageIO.write(retouche.getPhoto(), "JPEG", fichier);

}

catch (IOException ex) {

setTitle("Problème d'enregistrement de fichier");

}

}

}

});

add(BorderLayout.NORTH, barreBoutons);

onglets.addTab("Originale", originale);

onglets.addTab("Retouchée", retouche);

add(onglets);

setSize(400, 380);

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

setVisible(true);

}

public static void main(String[] args) { new Dialogue(); }

private class Photo extends JComponent {

private BufferedImage photo;

private double ratio;

@Override

protected void paintComponent(Graphics g) {

if (photo!=null) g.drawImage(photo, 0, 0, getWidth(), (int)(getWidth()/ratio), null);

}

public void setPhoto(BufferedImage photo) {

this.photo = photo;

ratio = (double)photo.getWidth() / photo.getHeight();

repaint();

}

public BufferedImage getPhoto() { return photo; }

}

private class Histogramme extends JComponent {

private final int largeur = 256;

private final int hauteur = 200;

private BufferedImage image;

private int[] rvb;

public Histogramme() {

this.setPreferredSize(new Dimension(largeur+16 , hauteur+4));

}

public void setImage(BufferedImage image) {

this.image = image;

récupérerRVB();

repaint();

}

@Override

protected void paintComponent(Graphics surface) {

super.paintComponent(surface);

if (image!=null) {

Graphics2D dessin = (Graphics2D) surface;

Rectangle2D rectangle = new Rectangle2D.Double(8, 4, largeur-1, hauteur-1);

dessin.draw(rectangle);

dessin.translate(0, hauteur+4);

double surfaceImage = image.getWidth()\*image.getHeight();

double surfaceHistogramme = largeur\*hauteur;

dessin.scale(1, -surfaceHistogramme/surfaceImage/3.7);

for (int i=0; i<255; i++) dessin.drawLine(i+8, 0, i+8, rvb[i]);

}

}

private void récupérerRVB() {

int[] rouge = new int[256];

int[] vert = new int[256];

int[] bleu = new int[256];

rvb = new int[256];

Raster trame = image.getRaster();

ColorModel modèle = image.getColorModel();

for (int y=0; y<image.getHeight(); y++)

for (int x=0; x<image.getWidth(); x++) {

Object données = trame.getDataElements(x, y, null);

rouge[modèle.getRed(données)]++;

vert[modèle.getGreen(données)]++;

bleu[modèle.getBlue(données)]++;

}

for (int i=0; i<256; i++) rvb[i] = (rouge[i]+vert[i]+bleu[i])/2;

}

}

private class Retouche extends JDialog implements ChangeListener {

private Histogramme histogramme = new Histogramme();

private JSlider gris = new JSlider(0, 255);

private JSlider noir = new JSlider(0, 127, 0);

private JSlider blanc = new JSlider(128, 255, 255);

private Niveaux vgris = new Niveaux(1.0);

private Niveaux vnoir = new Niveaux(0.0);

private Niveaux vblanc = new Niveaux(255.0);

private JPanel sud = new JPanel(new BorderLayout());

private JPanel niveaux = new JPanel();

private double proportion = 0.5;

private boolean extrême = false;

public Retouche() {

super(Dialogue.this, "Retoucher la photo", false);

add(histogramme);

add(sud, BorderLayout.SOUTH);

sud.add(gris, BorderLayout.NORTH);

noir.setPreferredSize(new Dimension(136, 16));

blanc.setPreferredSize(new Dimension(136, 16));

noir.addChangeListener(this);

blanc.addChangeListener(this);

gris.addChangeListener(this);

sud.add(noir, BorderLayout.WEST);

sud.add(blanc, BorderLayout.EAST);

sud.add(niveaux, BorderLayout.SOUTH);

niveaux.add(vnoir);

niveaux.add(vgris);

niveaux.add(vblanc);

pack();

}

public void réajuster() {

histogramme.setImage(originale.getPhoto());

extrême = true;

noir.setValue(0);

blanc.setValue(255);

extrême = false;

gris.setValue(128);

}

public void afficher() {

if (originale.getPhoto()!=null) {

setVisible(true);

}

}

private class Niveaux extends JFormattedTextField {

public Niveaux(double valeur) {

super(new DecimalFormat("#.##"));

setValue(valeur);

setColumns(3);

setHorizontalAlignment(CENTER);

setEditable(false);

setForeground(Color.BLUE);

}

public double getValeur() {

return ((Number)getValue()).doubleValue();

}

}

public void stateChanged(ChangeEvent e) {

double début = noir.getValue();

double fin = blanc.getValue();

double position = début + (fin - début) \* proportion;

if (e.getSource()==noir) {

extrême = true;

vnoir.setValue(début);

gris.setValue((int)position);

extrême = false;

}

else if (e.getSource()==blanc) {

extrême = true;

vblanc.setValue(fin);

gris.setValue((int)position);

extrême = false;

}

else {

if (!extrême) {

int x = gris.getValue();

proportion = (x-début) / (fin-début);

vgris.setValue(proportion<=0.5 ? 0.5/proportion : (1-proportion)/0.5);

}

}

calcul();

}

public void calcul() {

double moyen = vgris.getValeur();

double basse = vnoir.getValeur();

double haute = vblanc.getValeur();

int[] courbeInitiale = new int[256];

byte[] courbe = new byte[256];

for (int i=0; i<256; i++)

courbeInitiale[i] = (int) (255\*(basse-i)/(basse-haute)+64\*(moyen-1)\*Math.sin((i-basse)\*Math.PI/(haute-basse)));

for (int i=0; i<256; i++) {

if (courbeInitiale[i]<0) courbe[i] = (byte)0;

else if (courbeInitiale[i]>255) courbe[i] = (byte)255;

else courbe[i] = (byte)courbeInitiale[i];

}

ByteLookupTable table = new ByteLookupTable(0, courbe);

LookupOp opération = new LookupOp(table, null);

opération.filter(originale.getPhoto(), retouche.getPhoto());

retouche.repaint();

}

}

private class Aperçu extends JLabel {

public Aperçu() {

setPreferredSize(new Dimension(300, 200));

setBorder(BorderFactory.createLoweredBevelBorder());

sélecteur.addPropertyChangeListener(new PropertyChangeListener() {

public void propertyChange(PropertyChangeEvent evt) {

if (evt.getPropertyName().equals(JFileChooser.SELECTED\_FILE\_CHANGED\_PROPERTY)) {

File fichier = (File)evt.getNewValue();

if (fichier==null) { setIcon(null); return; }

ImageIcon vignette = new ImageIcon(fichier.getPath());

setIcon(new ImageIcon(vignette.getImage().getScaledInstance(getWidth(), -1, Image.SCALE\_DEFAULT)));

}

}

});

};

}

}