

## ProgIHM - Cours 3

# Boîtes de Dialogue, Sélection de Fichiers, ComboBox, Fenêtres modales

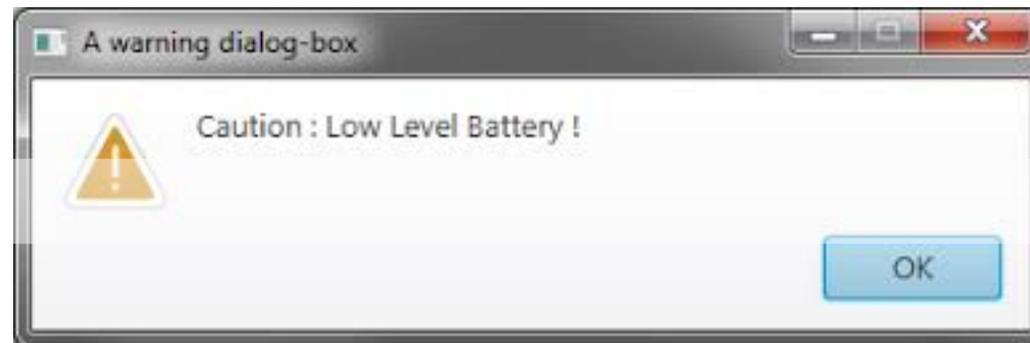
**V. DESLANDRES**

[veronique.deslandres@univ-lyon1.fr](mailto:veronique.deslandres@univ-lyon1.fr)

# Sommaire de ce cours

- Boites de dialogue [3](#)
- Les fenêtres de sélection de fichier [12](#)
- Les menus [16](#)
- La comboBox [23](#)
- Fenêtres modales [27](#)
- Les énumérations [37](#)

# Boites de dialogue



# Les boîtes de dialogue standard

- On utilise la classe `JOptionPane` pour les boîtes de dialogue standard, prêtes à l'emploi
  - Et des méthodes **statiques** : `showXXXDialog()`
- Quatre types de boîtes
  - `MessageDialog` pour afficher un message
  - `ConfirmDialog` pour une réponse de l'utilisateur avec Yes, No et Cancel
  - `InputDialog` pour une invite de saisie
  - `OptionDialog` qui rassemble les caractéristiques des 3 autres types de boîtes de dialogue

# Message Dialog (1/2)

- Un texte, un bouton OK et éventuellement un icône prédéfini
- Ne retourne rien (void)

```
JOptionPane.showMessageDialog(fen, "Le texte Information Msg...", "Un titre", JOptionPane.INFORMATION_MESSAGE);
```

```
JOptionPane.showMessageDialog(fen, "Le texte Warning Msg...", "Un titre", JOptionPane.WARNING_MESSAGE);
```

```
JOptionPane.showMessageDialog(fen, "Le texte Error Msg...", "Un titre", JOptionPane.ERROR_MESSAGE);
```

```
JOptionPane.showMessageDialog(fen, "Le texte Question Msg...", "Un titre", JOptionPane.QUESTION_MESSAGE);
```

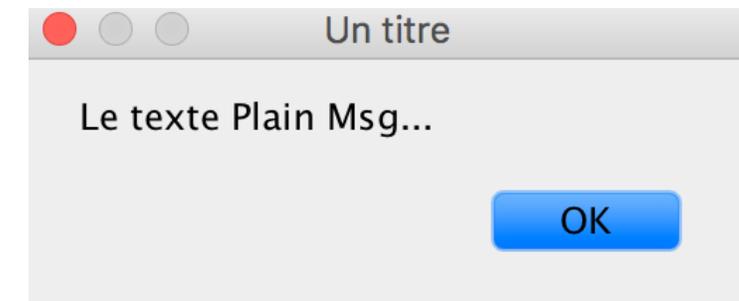
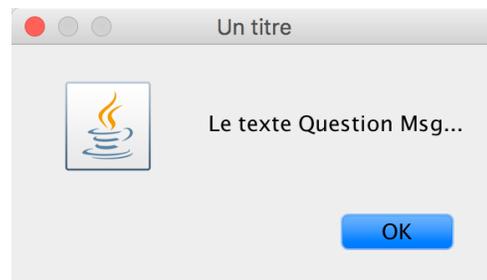
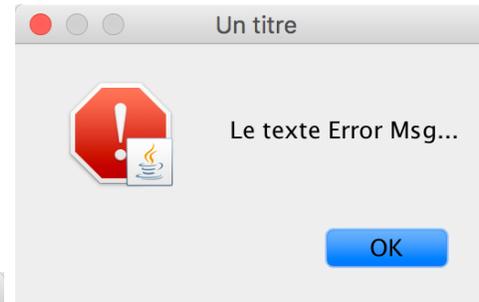
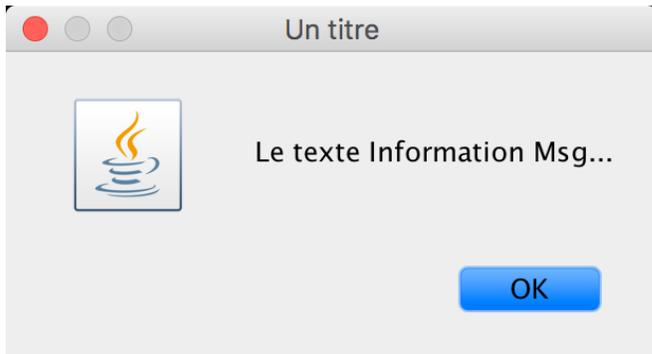
```
JOptionPane.showMessageDialog(fen, "Le texte Plain Msg...", "Un titre", JOptionPane.PLAIN_MESSAGE);
```

Fenêtre parent ↓

le texte ↓

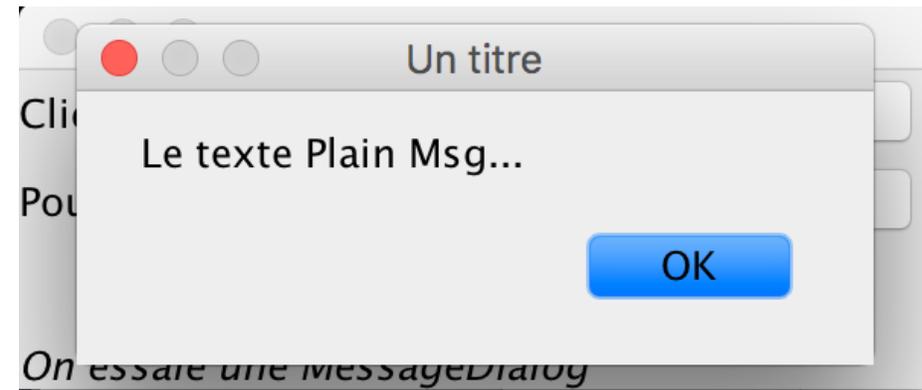
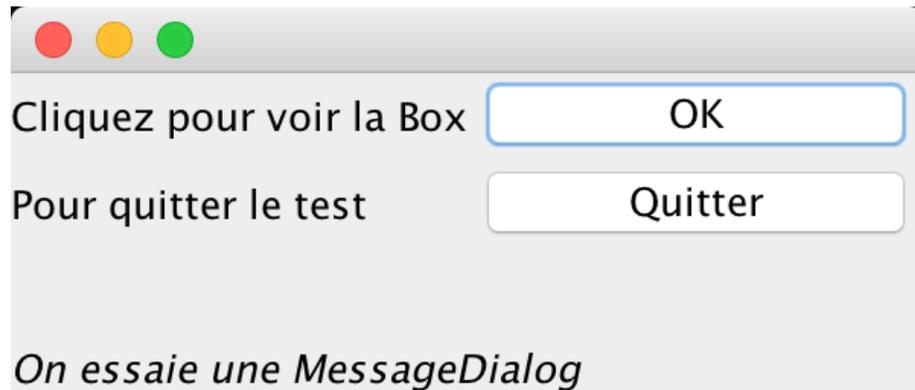
le titre ↓

type d'icône (énumération) ↓



## Message Dialog (2/2)

- Le 1<sup>er</sup> argument est une JFrame
- Si on met **null**, ça marche aussi et la fenêtre sera **centrée sur l'écran**
- Si on met une fenêtre parent, la fenêtre Dialog sera centrée sur cette dernière :

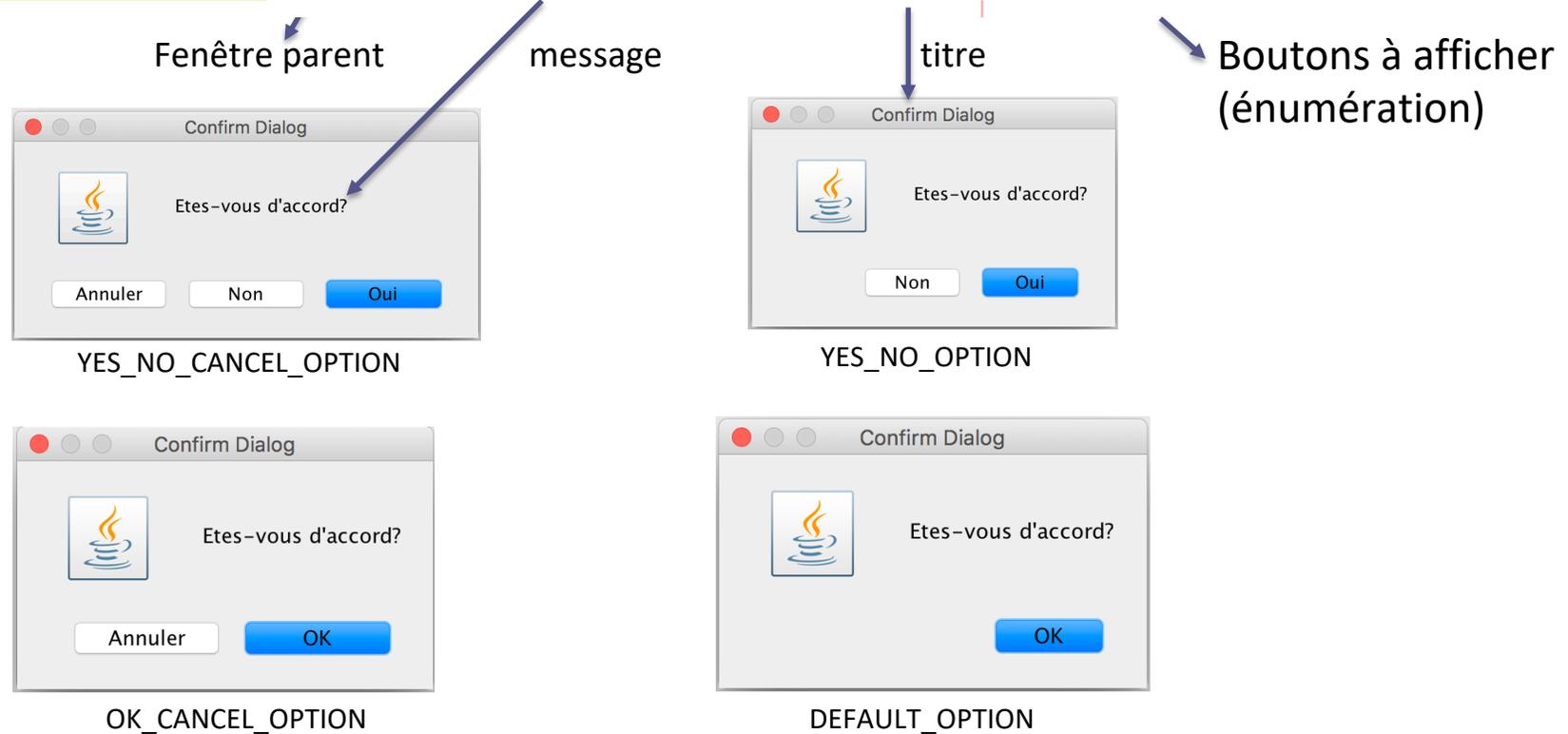


- Ceci est valable pour toutes les fenêtres de dialogue

# Confirm Dialog

- Une question, de 1 à 3 boutons (Oui/OK, Non, Annuler) et un icône
- Retourne un *int* correspondant à l'énumération sélectionnée

```
JOptionPane.showConfirmDialog(fen, "Etes-vous d'accord?", "Confirm Dialog", JOptionPane.YES_NO_CANCEL_OPTION);  
JOptionPane.showConfirmDialog(fen, "Etes-vous d'accord?", "Confirm Dialog", JOptionPane.YES_NO_OPTION);  
JOptionPane.showConfirmDialog(fen, "Etes-vous d'accord?", "Confirm Dialog", JOptionPane.OK_CANCEL_OPTION);  
JOptionPane.showConfirmDialog(fen, "Etes-vous d'accord?", "Confirm Dialog", JOptionPane.DEFAULT_OPTION);
```



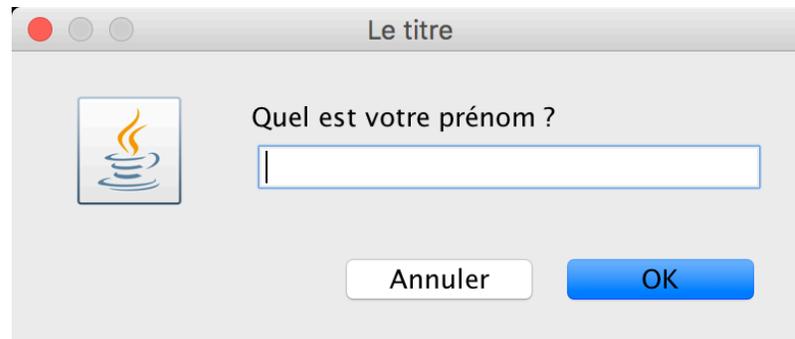
# Input Dialog : boîte de saisie

- Une question, une invite de saisie, 2 boutons (OK et annuler) et un icône
- Retourne une **chaîne**, correspondant au champ saisi

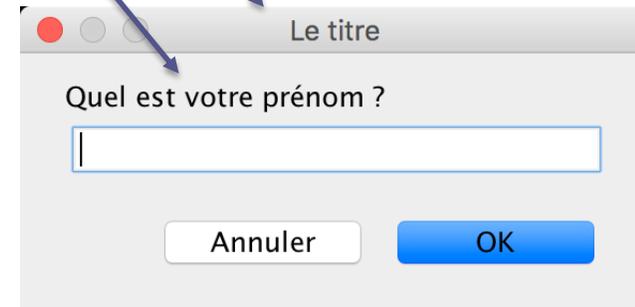
```
String message = "Quel est votre prénom ?";
```

```
String reponse = JOptionPane.showInputDialog(this, message, "Le titre", JOptionPane.PLAIN_MESSAGE);  
// ou WARNING_MESSAGE, INFORMATION_MESSAGE  
// ou ERROR_MESSAGE, PLAIN_MESSAGE);
```

Fenêtre parent



(icône : *INFORMATION\_MESSAGE* ou *QUESTION\_MESSAGE*)



(aucun icône : *JOptionPane.PLAIN\_MESSAGE*)

# Option Dialog (1/2)

## Caractéristiques :

- Une question avec des boutons représentant des choix différents et un icône

```
int showOptionDialog(Component parentComponent,  
                    Object message,  
                    String title,  
                    int optionType,  
                    int messageType,  
                    Icon icon,  
                    Object[] options,  
                    Object initialValue)
```

Soit *optionType* est à 0 et on donne des **options**

Soit un *optionType* est fourni (ex. YES\_NO\_OPTION), et **options** est **null**

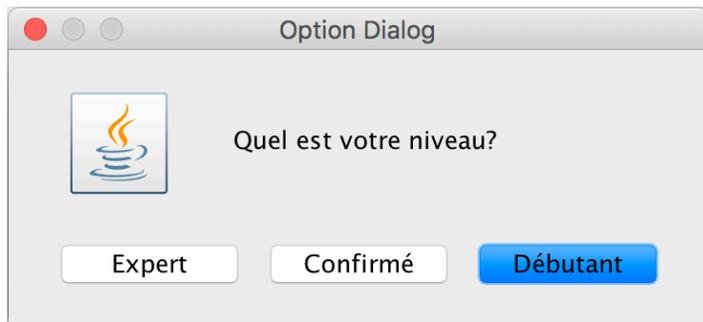
Ouvre une fenêtre de dialogue avec **l'icône proposé**, un texte avec le **message** proposé, les **options** de réponses possibles, et le **choix initial prédéfini** par le paramètre **initialValue**.

La méthode renvoie **l'indice** de l'option choisie (dans le tableau **options**) par l'utilisateur.

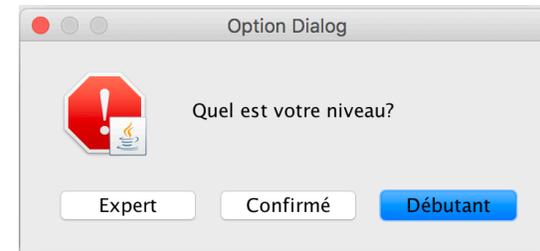
# Option Dialog (2/2)

```
String[] choix={"Débutant", "Confirmé", "Expert"};
JOptionPane.showOptionDialog(fen,"Quel est votre niveau?","Option Dialog",0,JOptionPane.INFORMATION_MESSAGE, null, choix, choix[0]);
JOptionPane.showOptionDialog(fen,"Quel est votre niveau?","Option Dialog",0,JOptionPane.ERROR_MESSAGE, null, choix, choix[0]);
JOptionPane.showOptionDialog(fen,"Quel est votre niveau?","Option Dialog",0,JOptionPane.WARNING_MESSAGE, null, choix, choix[0]);
JOptionPane.showOptionDialog(fen,"Quel est votre niveau?","Option Dialog",0,JOptionPane.QUESTION_MESSAGE, null, choix, choix[0]);
JOptionPane.showOptionDialog(fen,"Quel est votre niveau?","Option Dialog",0,JOptionPane.PLAIN_MESSAGE, null, choix, choix[1]);
```

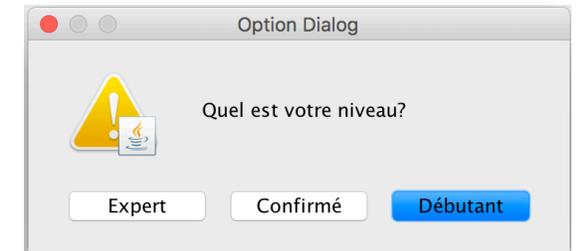
Fenêtre parent    Texte du msg    Titre    (on propose des boutons)    type icône    (icône)    Choix présélectionné  
liste des boutons



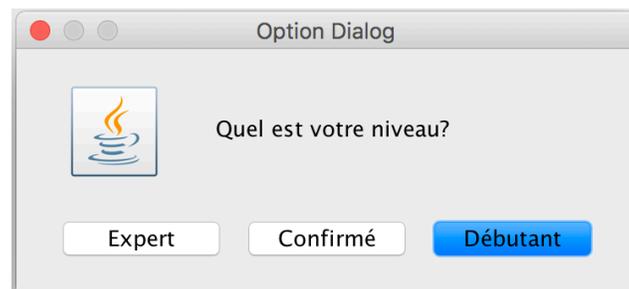
INFORMATION\_MESSAGE



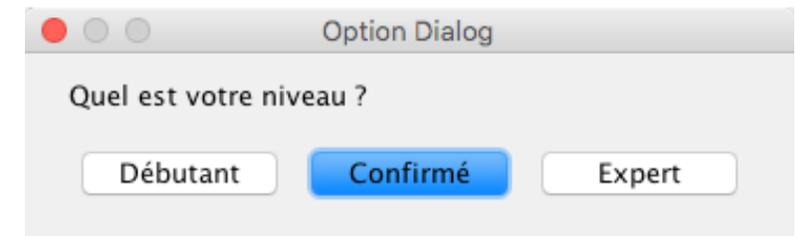
ERROR\_MESSAGE



WARNING\_MESSAGE



QUESTION\_MESSAGE



PLAIN\_MESSAGE

# Exemples d'utilisation

Cf aussi :  
[FenetreDeDialogueDemo.zip](#)

## Confirm Dialog

```
int response=JOptionPane.showConfirmDialog(fen,"Etes-vous d'accord?","Confirm Dialog",
    JOptionPane.YES_NO_CANCEL_OPTION);
if(response==JOptionPane.YES_OPTION)
    System.out.println("Yes Option !");
if(response==JOptionPane.NO_OPTION)
    System.out.println("No Option !");
if(response==JOptionPane.CANCEL_OPTION)
    System.out.println("Cancel Option !");
if(response==JOptionPane.CLOSED_OPTION)
    System.out.println("Closed Option !");
```

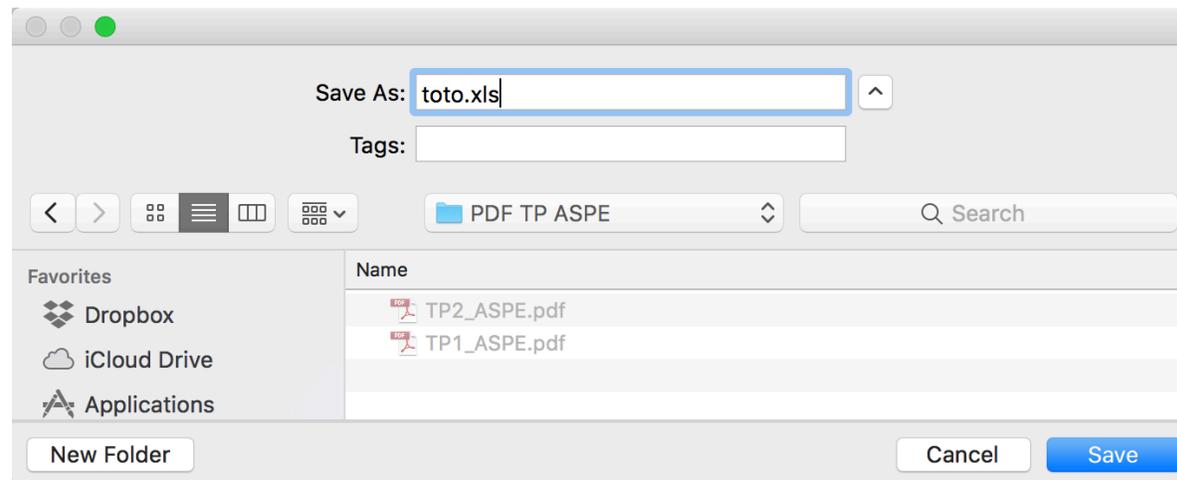
## Input Dialog

```
String reponse = JOptionPane.showInputDialog(fen, "Quel est votre prénom ?", "Le titre",
    JOptionPane.INFORMATION_MESSAGE);
if (reponse.length() != 0) {
    System.out.println("Bonjour " + reponse);
}
```

## Option Dialog

```
int response=JOptionPane.showOptionDialog(fen,"Quel est votre niveau?","Option Dialog",0,
    JOptionPane.INFORMATION_MESSAGE, null, choix, choix[0]);
if(response==JOptionPane.CLOSED_OPTION)
    System.out.println("Pas de formule choisie !");
else
    System.out.println("Vous avez choisi: "+choix[response]);
```

# Les fenêtres modales de sélection de fichiers

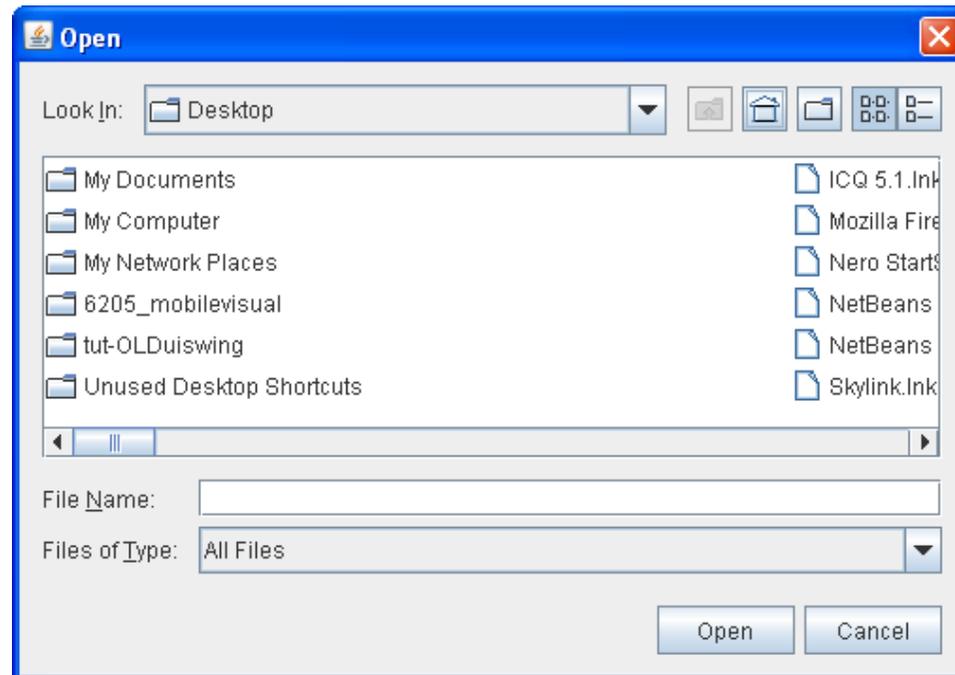


*FileDialog*

## 2 sortes de fenêtre pour sélectionner les Fichiers

- La **FileDialog** d'AWT : fenêtre de base permettant d'ouvrir ou d'enregistrer un fichier
  - Hérite de `java.awt.window`
  - Simple d'utilisation
- Le **JFileChooser** de SWING : fenêtre plus élaborée avec notamment la possibilité de filtrer les fichiers

**JFileChooser**



# FileDialog d'AWT

## Choix du fichier pour ouvrir

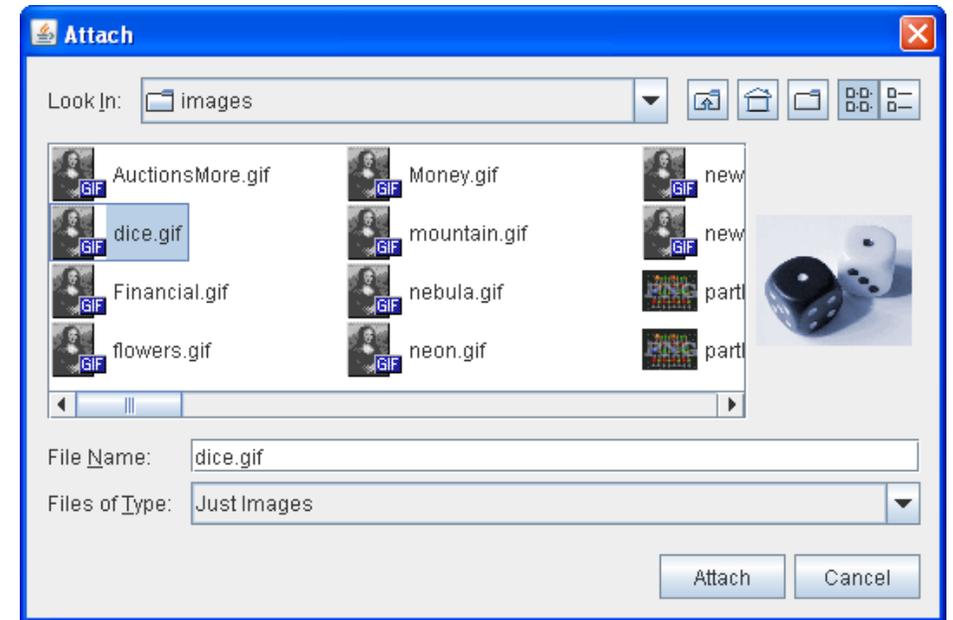
```
String nomFic = new String("");
try {
    // ouvrir un fichier
    FileDialog fd = new FileDialog(this, "Sélectionnez votre fichier...", FileDialog.LOAD);
    fd.setVisible(true);
    nomFic = ((fd.getDirectory()).concat(fd.getFile()));
}
catch (NullPointerException e) {
    System.out.println("Erreur ouverture dossier !");
}
```

## Choix du fichier pour enregistrer

```
String nomFic = new String("");
try {
    FileDialog fd = new FileDialog(this, "Sélectionnez votre fichier...", FileDialog.SAVE);
    fd.setVisible(true);
    nomFic = ((fd.getDirectory()).concat(fd.getFile()));
}
catch (NullPointerException e) {
    System.out.println("Erreur ouverture dossier !");
}
```

# JFileChooser

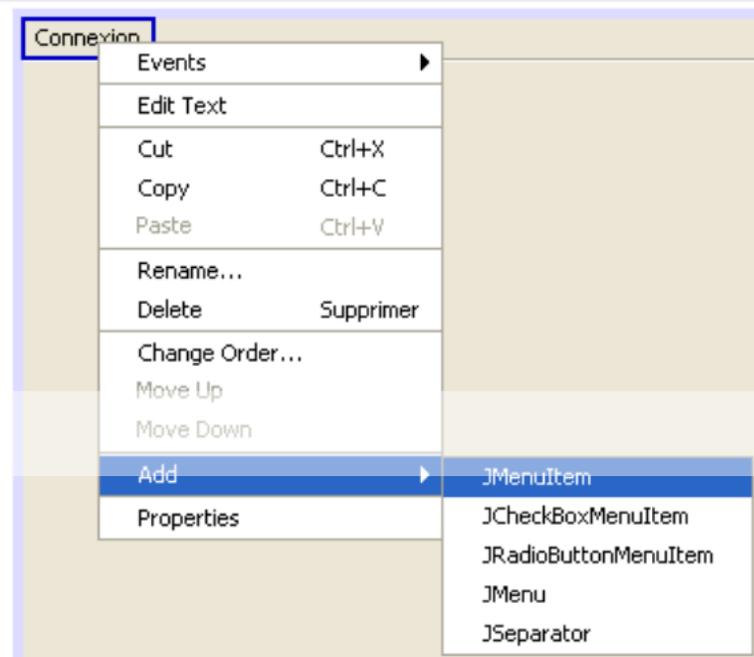
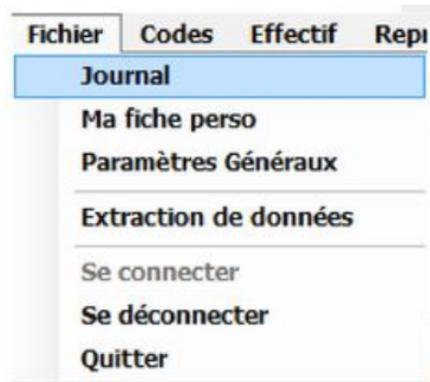
- Beaucoup plus complet
  - Permet de définir des **filtres** : types de fichiers, fichier ou répertoires,...
  - Ici filtre sur fichiers images



- Regarder le tutoriel sur le site Oracle :

<https://docs.oracle.com/javase/tutorial/uiswing/components/filechooser.html>

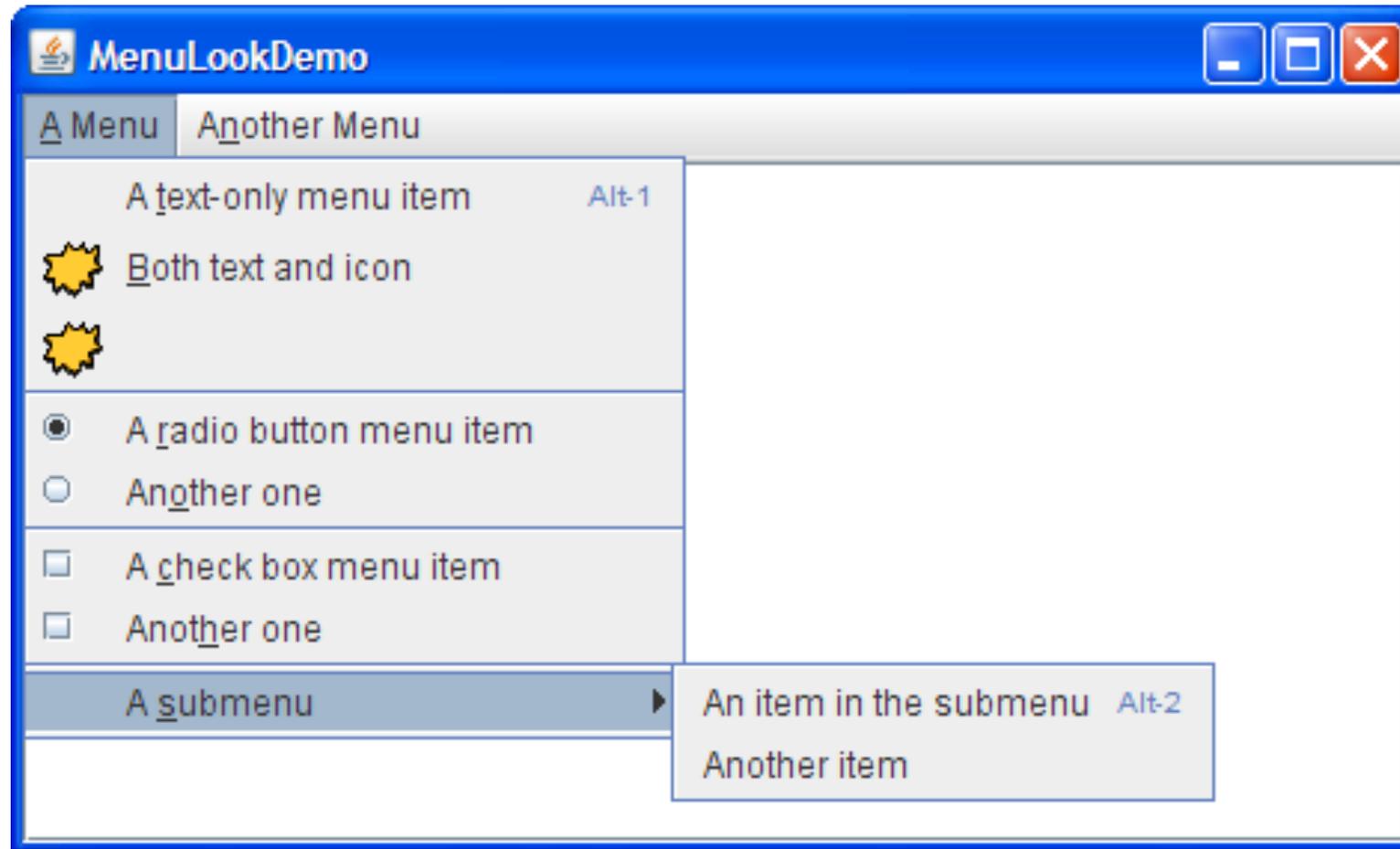
# Les Menus



# Les classes de menus

- Il existe plusieurs classes pour créer des menus en Java
  - `JMenu` qui sert à créer un **menu**
  - `JMenuBar` sert à créer une **barre** de menus
  - `JPopupMenu` sert à créer un menu **déroulant**
  - `JMenuItem` est un **item** d'un menu
  - `JCheckBoxMenuItem` est un item de menu, à **cocher**
  - `JRadioButtonMenuItem` est un item **Radio**

# Les menus





# Les menus

- Par convention, les menus ne sont pas placés dans d'autres composants de l'interface
  - **pas d'ajout au contentPane**
- Ils apparaissent :
  - soit dans une **barre de menus**
  - soit dans un menu **déroulant (JPopupMenu)**
- Le menu **déroulant** n'a pas besoin **d'écouteur**
  - C'est géré automatiquement par l'API Swing
  - On place juste des écouteurs **sur les items de menu**

# Les menus (exemple)

Dans la fenêtre

```
protected JMenuItem quitter = new JMenuItem("Quitter");  
protected JMenuItem option1 = new JMenuItem("Option-1");  
protected JMenuItem option2 = new JMenuItem("Option-2");  
protected JMenuItem option3 = new JMenuItem("Option-3");
```

On crée les 4 items de menu

Dans le constructeur

```
JMenu fichier = new JMenu("Fichier");  
fichier.add(quitter);  
JMenu edition = new JMenu("Edition");  
edition.add(option1);  
edition.add(option2);  
edition.addSeparator();  
edition.add(option3);
```

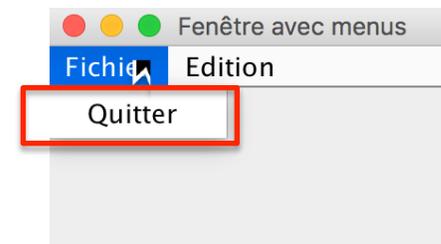
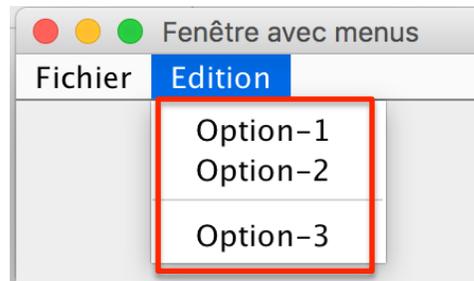
On crée les 2 menus (Fichier et Edition)  
et on ajoute les items aux menus avec  
une ligne séparatrice

Dans le constructeur

```
JMenuBar mb = new JMenuBar();  
mb.add(fichier);  
mb.add(edition);  
this.setJMenuBar(mb);
```

On crée la barre de menu,  
on y ajoute les 2 menus

Méthode de JFrame : ajoute la barre  
de menu à la fenêtre



# Les écouteurs des items de menu

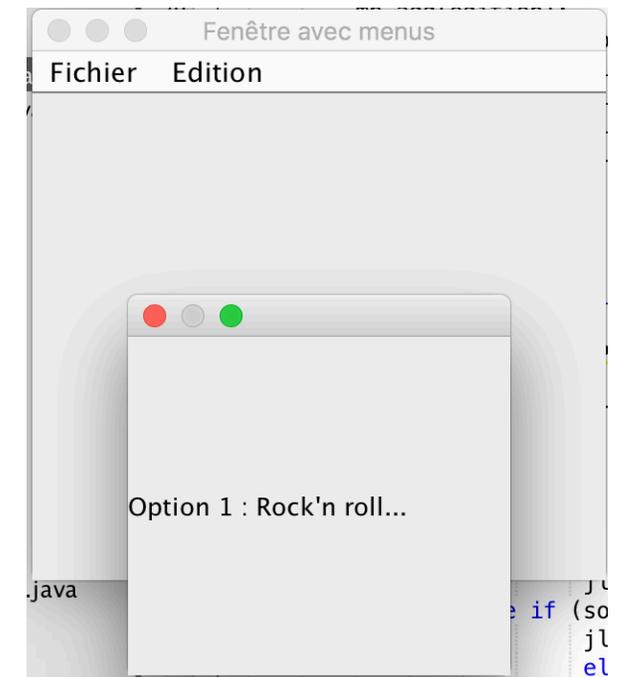
Dans le constructeur

```
quitter.addActionListener( new MonActionListener());  
option3.addActionListener( new MonActionListener());  
option2.addActionListener( new MonActionListener());  
option1.addActionListener( new MonActionListener());
```

On ajoute l'écouteur  
à chaque item

```
class MonActionListener implements ActionListener {  
    public void actionPerformed(ActionEvent e) {  
        Object source = e.getSource();  
        if (source == quitter) System.exit(0);  
  
        JDialog jd = new JDialog();  
        JLabel jl;  
        if (source == option1)  
            jl = new JLabel("Option 1 : Rock'n roll...");  
        else if (source == option2)  
            jl = new JLabel("Option 2 : Choubidou waa...");  
        else  
            jl = new JLabel("Option 3 : Yeah... Rasta man");  
        jd.setSize(200,200);  
        jd.setLocation(250,300);  
        jd.getContentPane().add(jl); // on ajoute le JLabel ici  
        jd.setVisible(true);  
    }  
} // fin classe ecouteur
```

Exemple d'écouteur  
(ici en classe interne)

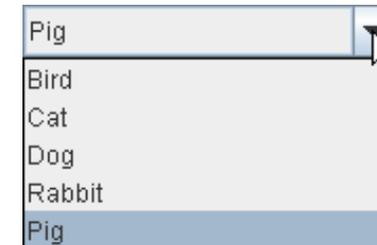


# ComboBox



# JComboBox : les caractéristiques de base

- **Une JComboBox** est un composant très souvent utilisé : il permet d'éviter les erreurs de saisie
  - Par rapport à un JTextField par ex.
- Il permet aux utilisateurs de choisir une des options proposées.
- Lorsque l'utilisateur clique sur la **ComboBox**, une liste d'options à sélectionner apparaît et il choisit un item.

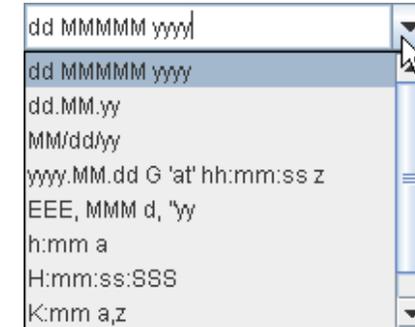


4 constructeurs :

- **JComboBox()** crée une JComboBox avec un modèle par défaut.
- **JComboBox(ComboBoxModel<E> unModele)** Crée une JComboBox avec les items fournis par le ComboBoxModel.
- **JComboBox(E[] tablItems)** Crée une JComboBox qui contient les éléments du tableau *tablItems*
- **JComboBox(Vector<E> vecItems)** Crée une JComboBox qui contient les éléments du Vector *vectorItems*

# JComboBox : les caractéristiques avancées

- Une JComboBox peut être **éditable**, comme ici :
  - ou **non modifiable**
- Pour gérer les actions de l'utilisateur sur une JComboBox :
  - les interfaces **ActionListener**, **ChangeListener** ou **ItemListener** peuvent être utilisées
- Une méthode **getSelectedItem()** permet de récupérer l'élément sélectionné
- Une méthode **setEditable()** peut être utilisée pour activer ou désactiver la partie saisie du texte
  - (sans que ça permette de trouver l'item qui s'en rapproche\*)
  - Permet d'ajouter des items à la liste



\* Pour faire de l'auto-complétion avec une Combo, cf <http://www.orbital-computer.de/JComboBox/>

# JComboBox : les caractéristiques

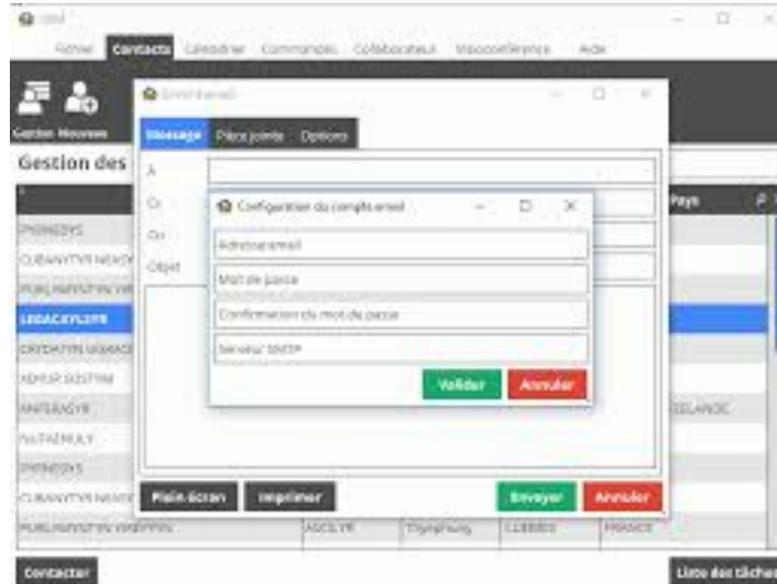
- On peut créer une instance de JComboBox à partir d'un **tableau d'objets**, ou d'une instance de **Vector**.
- Souvent on utilise un **ComboBoxModel** pour stocker les données à afficher de la comboBox
  - Cela permet de charger dynamiquement la liste des éléments de la ComboBox
    - Par ex. quand une comboBox dépend d'une **autre variable** (Master / Detail)
- (Pour aller plus loin : on peut aussi préférer un **MutableComboBoxModel**
  - Quand on veut pouvoir modifier dynamiquement la liste des items : ajouter, modifier, supprimer des éléments, etc.)

- On peut récupérer la **partie éditeur de la ComboBox** avec :

```
JTextComponent editor = (JTextComponent) comboBox.getEditor().getEditorComponent();
```

Et ainsi placer des écouteurs sur le texte, par ex. pour permettre l'auto-complétion.

# Construire ses propres Fenêtres Modales



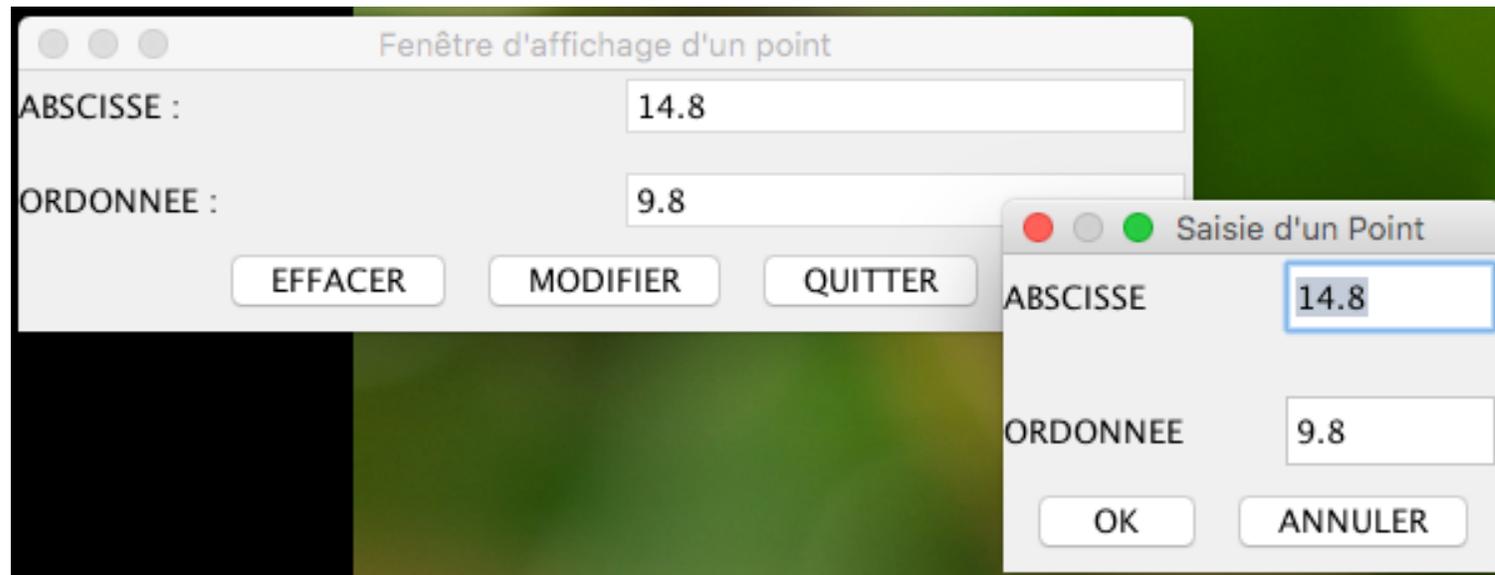
# Fenêtre Modale

- DEF – C'est une fenêtre qui **dépend** d'une autre fenêtre et qui **prend le contrôle** du clavier et de l'écran.
  - Par ex.: quand on pose une question à laquelle il est impératif que l'utilisateur réponde, avant de pouvoir à nouveau interagir avec le reste du programme
- La fenêtre modale permet :
  - **d'obtenir des données** de l'utilisateur,
  - de **fournir une information** à l'utilisateur
- Pour saisir des données de l'application, on utilise souvent la classe `JDialog`
  - **Semblable à celle de `JFrame`**
  - **Utilisées pour des fenêtres modales et *non* modales**

# Fenêtre Modale (Exemple)

- Exemple:

- on a la classe `Point` (d'AWT) avec 2 coordonnées réelles  $X$  et  $Y$
- on réalise un petit programme d'affichage ou de saisie/modification des coordonnées d'un point
- Ici l'utilisateur doit fermer la boîte de dialogue avant de pouvoir à nouveau interagir avec le reste du programme



# Fenêtre Parent (1/2)

```
public class FenetreModale extends JFrame implements ActionListener{
    JTextField xa, ya;
    JLabel abscisse, ordonnée, d;
    JButton btnE, btnM, btnQ;
    Point pointA=new Point(0,0);
    JPanel pan;
    FenetreModale(){ //Constructeur
        pan=(JPanel) getContentPane();
        pan.setLayout(new BorderLayout());
        pan.add(panneau_bas(), BorderLayout.SOUTH);
        pan.add(panneau_milieu(), BorderLayout.CENTER);
    }
}
```

Classe Point  
du package awt

Ici on choisit de passer par des fonctions  
qui retournent le JPanel désiré

```
JPanel panneau_milieu(){ //Définir le panneau au milieu
    JPanel milieu=new JPanel();
    milieu.setLayout(new GridLayout(2,2,20,20));
```

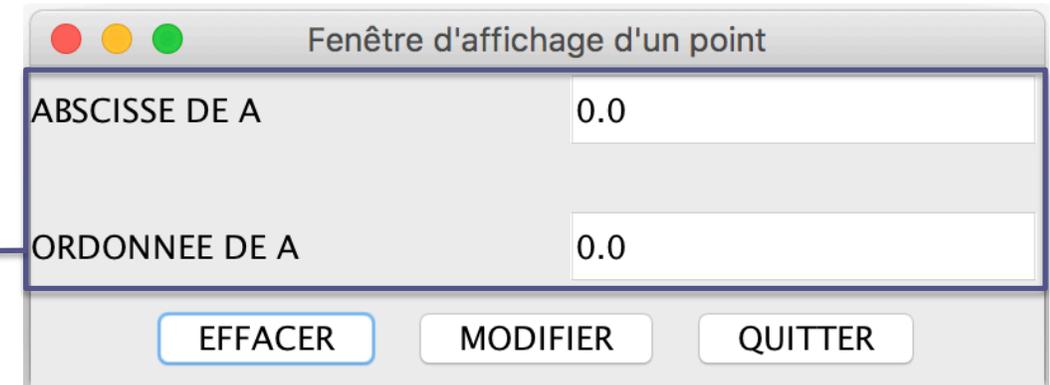
```
        abscisse=new JLabel("ABSCISSE DE A");
        ordonnée=new JLabel("ORDONNEE DE A");

        xa=new JTextField(20);
        ya=new JTextField(20);

        xa.setEditable(false);
        ya.setEditable(false);

        milieu.add(abscisse);
        milieu.add(xa);
        milieu.add(ordonnée);
        milieu.add(ya);

        return milieu;
    }
```

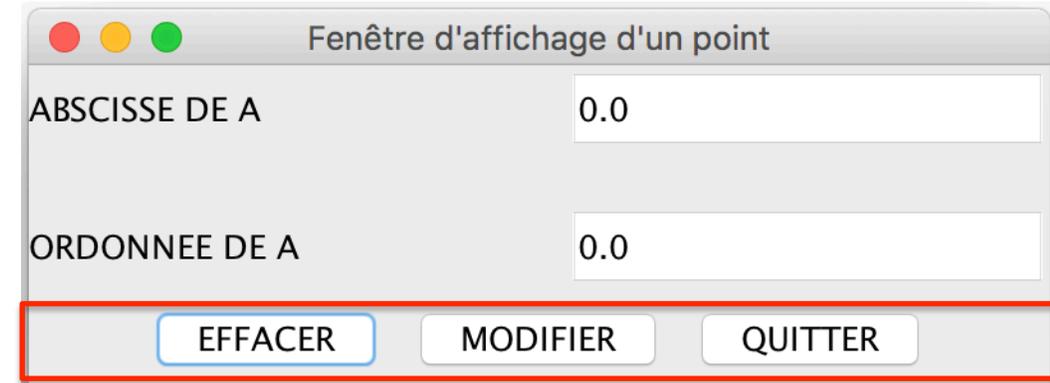


## Fenêtre Parent (2/2)

```
JPanel panneau_bas(){
    JPanel bas=new JPanel();
    btnE=new JButton("EFFACER");
    btnM=new JButton("MODIFIER");
    btnQ=new JButton("QUITTER");
    bas.add(btnE);
    bas.add(btnM);
    bas.add(btnQ);
    btnE.addActionListener(this);
    btnM.addActionListener(this);
    btnQ.addActionListener(this);
    return bas;
}

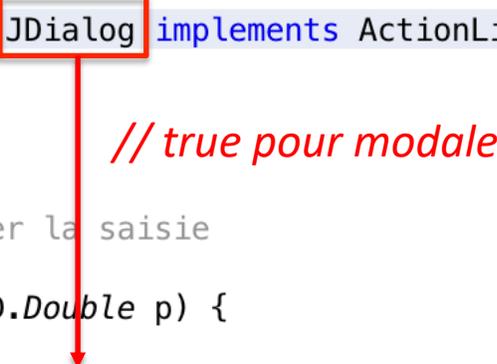
public void actionPerformed(ActionEvent e){
    if(e.getSource()==btnE){
        pointA.setLocation(0,0);
        xa.setText(String.valueOf(pointA.getX()));
        ya.setText(String.valueOf(pointA.getY()));
    }
    else if(e.getSource()==btnM){
        FenSaisiePoint fenSaisie= new FenSaisiePoint(this, pointA);
        if(fenSaisie.afficheModale()){
            xa.setText(String.valueOf(pointA.getX()));
            ya.setText(String.valueOf(pointA.getY()));
        }
    }
    else if(e.getSource()==btnQ){
        this.dispose();
    }
}

public static void main(String[] args){
    JFrame fen=new FenetreModale();
    fen.setBounds(10, 20, 400, 150);
    fen.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    fen.setTitle("Fenêtre d'affichage d'un point");
    fen.setVisible(true);
}
```



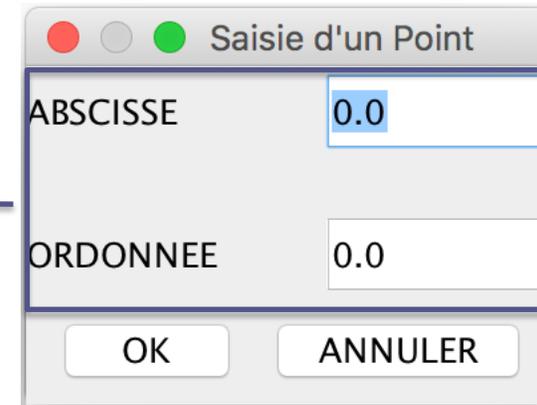
# Fenêtre Saisie (1/4)

```
public class FenSaisiePoint extends JDialog implements ActionListener {  
  
    Point2D.Double p;  
    JTextField tf_x, tf_y;  
    JButton btOK, btANNUL;  
    boolean OKchoisi; // pour valider la saisie  
  
    FenSaisiePoint(JFrame f, Point2D.Double p) {  
  
        super(f, "Saisie d'un Point", true); // construction de la fenêtre modale  
        this.p = p; // on récupère le point envoyé lors de l'appel  
  
        this.setBounds(450, 100, 200, 150);  
        this.setDefaultCloseOperation(JDialog.DO_NOTHING_ON_CLOSE);  
        // on laisse gerer la fenetre mere  
        Container c = getContentPane();  
        c.setLayout(new BorderLayout());  
        c.add(panneau_milieu(), BorderLayout.CENTER);  
        c.add(panneau_bas(), BorderLayout.SOUTH);  
  
        OKchoisi = false; // est à FALSE tant qu'on n'a pas validé une saisie  
        this.setVisible(true);  
    }  
}
```



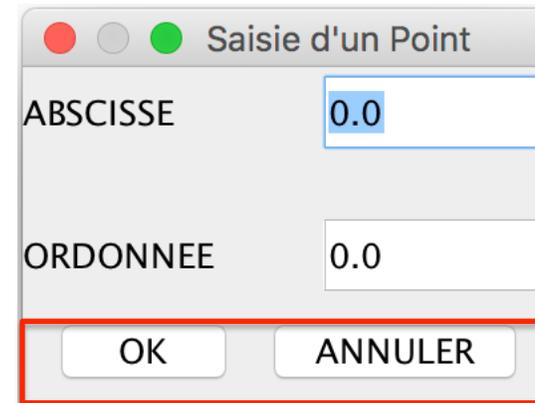
## Fenêtre Saisie (2/4)

```
JPanel panneau_milieu() {  
    JPanel milieu = new JPanel();  
    milieu.setLayout(new GridLayout(2, 2));  
  
    JLabel l = new JLabel(" ABSCISSE");  
    milieu.add(l);  
    tf_x = new JTextField(20);  
    milieu.add(tf_x);  
    l = new JLabel(" ORDONNEE");  
    milieu.add(l);  
    tf_y = new JTextField(20);  
    milieu.add(tf_y);  
  
    tf_x.setText(String.valueOf(p.getX()));  
    tf_y.setText(String.valueOf(p.getY()));  
  
    return milieu;  
}
```



## Fenêtre Saisie (3/4)

```
JPanel panneau_bas() {  
    JPanel pan = new JPanel();  
  
    btOK = new JButton("OK");  
    btOK.addActionListener(this);  
    btANNUL = new JButton("ANNULER");  
    btANNUL.addActionListener(this);  
    pan.add(btANNUL);  
    pan.add(btOK);  
  
    return pan;  
}
```



## Fenêtre Saisie (4/4)

```
public void actionPerformed(ActionEvent e) {
    double vx, vy;
    if (e.getSource() == btOK) {
        // lecture de l'abscisse :
        try {
            vx = Double.parseDouble(tf_x.getText());
            // ou vx = new Double(tf_x.getText());
        } catch (RuntimeException ex) {
            // en cas d'erreur sur la saisie, reassigne les coordonnees d'origine
            vx = p.getX();
            tf_x.setText(String.valueOf(p.getX()));
        }

        // idem pour l'ordonnee :
        try {
            vy = Double.parseDouble(tf_y.getText());
        } catch (RuntimeException ex) {
            vy = p.getY();
            tf_y.setText(String.valueOf(p.getY()));
        }

        // affecte nouvelles valeurs au Point p :
        p.setLocation(vx, vy);
        OKchoisi = true;
    } else { // clic btANNUL : on rend la main à la fenetre parent
        OKchoisi = false;
    }
    dispose(); // ferme la fenetre modale et toutes les autres fenetres
               //eventuellement ouvertes à partir d'elle
               // et rend la main à la fenetre parent
} // fin écouteur
```

On récupère la valeur des champs X et Y et on les transmet au Point p

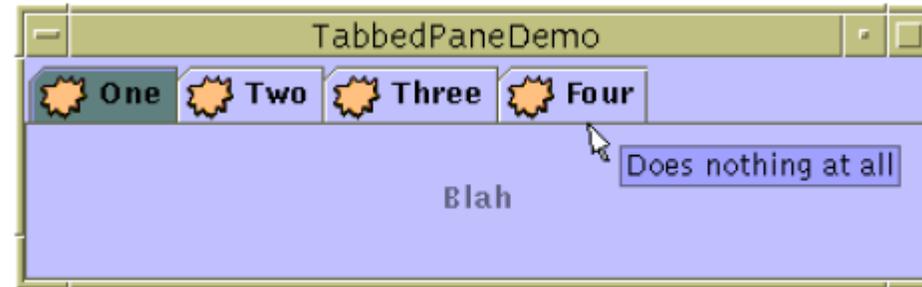
En cas de pb (si une exception est levée), on prend les anciennes valeurs du point

# Autres composants utiles

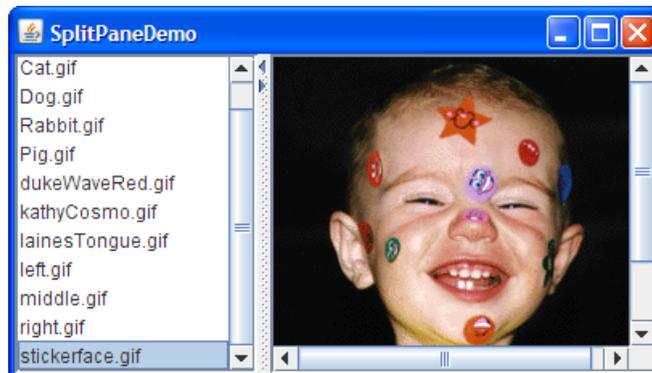
**JColorChooser** : fenêtre de dialogue permettant de choisir une couleur



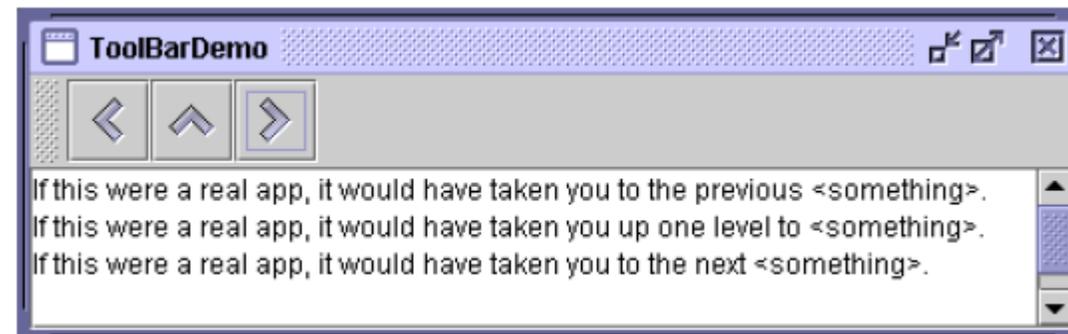
**JTabbedPane** : permet de mettre plusieurs JPanel dans des onglets



**JSplitPane** : il s'agit d'un double conteneur permettant une interaction entre deux composants



**JScrollPane** : un conteneur permettant le défilement (ascenseur) si nécessaire



# Les énumérations

# Les énumérations

- Existe depuis le JDK Java 5 (Tiger)
- Mot-clefs : **enum** ou **Enum**
- Les énumérations sont un ensemble de constantes liées
- Plus propres que les constantes du langage Java
  - Vérifie les valeurs lors de la **compilation**
- Il existe des *enum* and des classes *Enum*
  - Une **enum** est un type dont les champs sont des constantes fixes
  - **Enum** est un type de classe particulier qui hérite de `java.lang.Enum`

# Les énumérations simples

- Pour représenter le jour de la semaine :

```
public enum JoursSemaine {  
    LUNDI, MARDI, MERCREDI, JEUDI, VENDREDI, SAMEDI, DIMANCHE  
}
```

- Utilisation :

```
JoursSemaine jour = JoursSemaine.LUNDI;  
if (jour == JoursSemaine.SAMEDI) {  
    // ...  
}
```

```
switch (jour) {  
    case DIMANCHE:  
        break;  
  
    case LUNDI:  
        break;  
  
    // ...  
}
```

Les variables de type **enum** sont implicitement **comparables** et **sérialisables**

# Les Classes Enumeration

- On peut leur ajouter :
    - Des méthodes
    - Une interface
- ➔ cf illustration ci-après...

# Classe Enumeration : un exemple

- On veut représenter les continents et leur superficie
- Imaginer 2 méthodes:
  - `getArea()` : donne la superficie
  - `getCoverage()` : renvoie le ratio de la Région par rapport à la superficie globale

Nom	Superficie (km <sup>2</sup> )
Amérique du Nord	24.490.000
Amérique du Sud	17.840.000
Antarctique	13.720.000
Asie	43.810.000
Europe	10.400.000
Afrique	30.370.000
Océanie	9.010.000

```
public enum Continent {  
    NORTHAMERICA (24490000),  
    SOUTHAMERICA (17840000),  
    ANTARCTICA (13720000),  
    ASIA (43810000),  
    EUROPE (10400000),  
    AFRICA (30370000),  
    OCEANIA (9010000);  
  
    private final int area;  
  
    private static final int TOTALAREA = 149640000;
```

La variable numérique est définie après :  
ici, c'est la superficie du continent (un  
entier)

NOTE: le compilateur crée une liste des valeurs que  
l'on récupère avec la méthode **values()** :

```
for (Continent c: Continent.values() ) {  
    System.out.println("continent: "  
+ c);  
}
```

Cela affiche les valeurs dans **l'ordre de leur  
définition.**

Le constructeur de la classe énumération doit aussi être définie :

```
private Continent (int area) {  
    this.area = area;  
}  
  
public int getArea() {  
    return area;  
}  
  
public double getCoverage() {  
    return area / (double) TOTALAREA * 100;  
}  
}
```

#### USE

```
Continent c = Continent.EUROPE;
```

```
System.out.printf ("%s continent  
couvre %.2f%% de la surface mondiale\n" , c,  
c.getCoverage() );
```

*Affichage : « EUROPE couvre 6,95% de la surface mondiale »*