



LA GESTION DES ÉVÈNEMENTS

La gestion des évènements

- L'interaction entre l'utilisateur et une interface graphique engendre des **évènements**. Par exemple, cliquer sur un bouton, appuyer sur une touche du clavier et fermer une fenêtre engendrent des évènements.
- Tous les composants graphiques peuvent engendrer des évènements.
- Les évènements sont représentés par des instances des sous-classes de `java.util.EventObject`
<http://docs.oracle.com/javase/7/docs/api/java/util/EventObject.html>
- Exemples d'évènements
 - `ActionEvent`: lorsque le composant est actionné (exemple clic sur un bouton)
 - `MouseEvent`: lorsqu'on a cliqué ou déplacé la souris sur un composant.
 - `FocusEvent`: lorsqu'un composant gagne ou perd le focus
 - `ComponentEvent`: lorsqu'un composant est déplacé, redimensionné, rendu visible ou invisible
 - `TextEvent`: lorsque le texte(comme un `textField`) est modifié.

La gestion des évènements

- En langage Java, la gestion des évènements est réalisée par l'intermédiaire d'objets spécifiques appelés **écouteurs** (en anglais **listener**). De façon simplifiée, on peut dire que lorsque l'utilisateur clique sur un bouton, ce dernier émet un évènement à l'attention de l'écouteur.
- Les **écouteurs** sont des interfaces qui héritent de l'interface **EventListener** comme `ActionListener`, `WindowListener`, `MouseListener`, ...
- **Exemple**
 - Lorsqu'on actionne un bouton (en utilisant la souris ou le clavier), il se produit un évènement **ActionEvent**.
 - Pour être capable d'écouter et de traiter un tel évènement, un objet écouteur(sa classe) se doit d'implémenter une interface nommée **ActionListener**.
 - L'interface `ActionListener` ne contient qu'une unique méthode à implémenter : `actionPerformed(ActionEvent e)`. Cette méthode doit donc contenir le code à exécuter si jamais on clique sur ce bouton.
 - On doit ensuite enregistrer l'écouteur auprès du bouton qu'il est censé écouter (surveiller l'évènement action).
- **L'interface `ActionListener` :**
 - <https://docs.oracle.com/javase/8/docs/api/java/awt/event/ActionListener.html>

```
public interface ActionListener extends EventListener{  
    void actionPerformed(ActionEvent e);  
}
```

Mise en place d'un gestionnaire d'évènement

- La mise en place d'un écouteur, ou gestionnaire d'évènement nécessite 3 étapes. Nous allons expliquer ces étapes pour un écouteur qui gère des évènements d'action sur un bouton, soit un ActionListener.
- **Étape 1: Créer l'écouteur (gestionnaire d'évènement)**

On peut créer une classe interne à la classe graphique qui implémente l'interface ActionListener. Par exemple, CalculerListener. **Il existe d'autres possibilités** qu'on ne verra pas dans ce cours.

```
class CalculerListener implements ActionListener {  
    @Override  
    public void actionPerformed(ActionEvent e) {  
        // écrire le code qui doit s'exécuter quand un bouton est actionné  
    }  
}
```

- **Étape 2: Instancier l'écouteur (le gestionnaire d'évènement)**

```
ActionListener ecouteur = new CalculerListener();
```

- **Étape 3: Enregistrer l'écouteur (le gestionnaire d'évènement) auprès du bouton qui génère l'évènement avec la méthode addActionListener().**

```
btnCalculer.addActionListener(ecouteur); // on met le bouton sur écoute
```

Ou bien

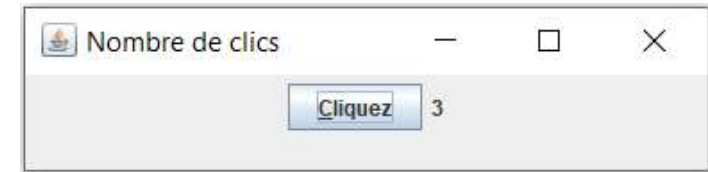
```
btnCalculer.addActionListener(new CalculerListener());
```

Mise en place d'un gestionnaire d'évènement: exemple I

- On affiche le nombre de clics sur le bouton «Cliquez».

```
public class FrmNombreClics extends JFrame {  
    private JButton btnClic = new JButton("Cliquez");  
    private JLabel lblNbClics = new JLabel("0");  
    private int compteurClics = 0;
```

```
    public FrmNombreClics() {  
        super("Nombre de clics");  
        setLayout(new FlowLayout()); // modifier le layout de la fenêtre  
        setSize(400, 100);  
        btnClic.setMnemonic(KeyEvent.VK_C);  
  
        // mettre btnClic sur écoute  
        btnClic.addActionListener(new ClicListener());  
  
        add(btnClic);  
        add(lblNbClics);  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    }
```



Chaque fois que le bouton est actionné, il cherche à invoquer la méthode `actionPerformed()` sur chacun de ses écouteurs. Ici on a un seul écouteur.

```
class ClicListener implements ActionListener {  
  
    @Override  
    public void actionPerformed(ActionEvent e) {  
        compteurClics++;  
        lblNbClics.setText(String.valueOf(compteurClics));  
    }  
}
```

Le traitement qui sera exécuté quand le bouton est actionné.
Le paramètre `e` qualifie l'évènement qui survient

C'est une classe interne définie dans la classe `FrmNombreClics`. Elle accède à tous les attributs même ceux privés, de cette classe.

Mise en place d'un gestionnaire d'évènement: exemple2

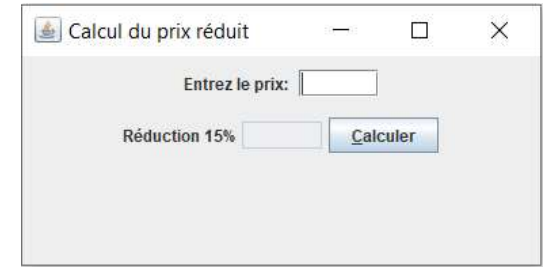
```
public class FrmCalculer extends JFrame {
    private JButton btnCalculer = new JButton("Calculer");
    private JTextField txtPrix = new JTextField(5);
    private JLabel lblPrix = new JLabel("Entrez le prix: ");
    private JLabel lblReduction = new JLabel("Réduction 15%");
    private JTextField txtPrixReduit = new JTextField(5);
    private JPanel panel1 = new JPanel();
    private JPanel panel2 = new JPanel();
```

```
public FrmCalculer() {
    super("Calcul du prix réduit");
    setLayout(new FlowLayout());
    setSize(400, 200);
    txtPrixReduit.setEditable(false);
    btnCalculer.setMnemonic(KeyEvent.VK_C);
```

```
    // mettre btnCalculer sur écoute
    btnCalculer.addActionListener(new CalculerListener());
```

```
    panel1.add(lblPrix);
    panel1.add(txtPrix);
    panel2.add(lblReduction);
    panel2.add(txtPrixReduit);
    panel2.add(btnCalculer);
    add(panel1);
    add(panel2);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
```

```
class CalculerListener implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent e) {
        String prixTxt = txtPrix.getText();
        double prixDouble = Double.parseDouble(prixTxt); // convertir le texte en double
        double prixReduit = prixDouble - prixDouble * 0.15;
        //txtPrixReduit.setText(String.valueOf(prixReduit));
        txtPrixReduit.setText(String.format("%.2f$", prixReduit));
    }
}
```



À chaque fois que le bouton est actionné, il cherche à invoquer la méthode `actionPerformed()` sur chacun de ses écouteurs. Ici on a un seul écouteur.

Le traitement qui sera exécuté quand le bouton est actionné.
Le paramètre `e` qualifie l'évènement qui survient

C'est une classe **interne** définie dans la classe `FrmCalculer`. Elle accède à tous les attributs même ceux privés, de cette classe.

Connaitre la source de l'évènement: getSource()

- Un écouteur (ou gestionnaire d'évènements) peut écouter plusieurs sources d'évènements. Une source d'évènement peut alerter plusieurs écouteurs.
- Comment reconnaître l'objet source de l'évènement?
 - L'objet évènement `e` de la méthode `actionPerformed(ActionEvent e)` permet d'identifier le composant source de l'évènement en appelant `getSource()` :
`e.getSource() = composant source de l'évènement`
 - La méthode `getSource()` fournit une référence de type `Object`.
 - La méthode `getSource()` est présente dans toutes les classes évènements.

- Exemple:

```
@Override
public void actionPerformed (ActionEvent e)
{
    if (e.getSource() == bouton1)
        action1
    else if (e.getSource() == bouton2)
        action2
    ...
}
```

Connaitre la source de l'évènement: getSource()

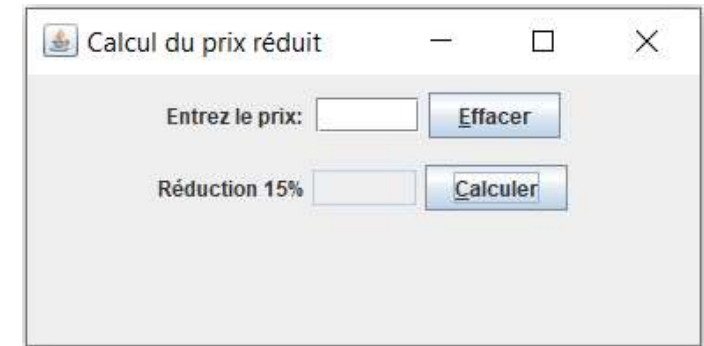
- Exemple: On ajoute un deuxième bouton qui consiste à effacer les 2 zones de texte. Si on utilise le même écouteur pour les boutons **btnCalculer** et **btnEffacer**, pour savoir quel bouton a été actionné, on utilise **e.getSource()**.

```
public class FrmCalculer2 extends JFrame {
    // créer les composants de base
    private JButton btnCalculer = new JButton("Calculer");
    private JButton btnEffacer = new JButton("Effacer");
    private JTextField txtPrix = new JTextField(5);
    private JLabel lblPrix = new JLabel("Entrez le prix: ");
    private JLabel lblReduction = new JLabel("Réduction 15%");
    private JTextField txtPrixReduit = new JTextField(5);
    // créer le composant intermédiaire: le panneau
    private JPanel panel1 = new JPanel();
    private JPanel panel2 = new JPanel();

    public FrmCalculer2() {
        super("Calcul du prix réduit");
        setLayout(new FlowLayout());
        setSize(400, 200);
        txtPrixReduit.setEditable(false);
        btnEffacer.setMnemonic(KeyEvent.VK_E);
        btnCalculer.setMnemonic(KeyEvent.VK_C);

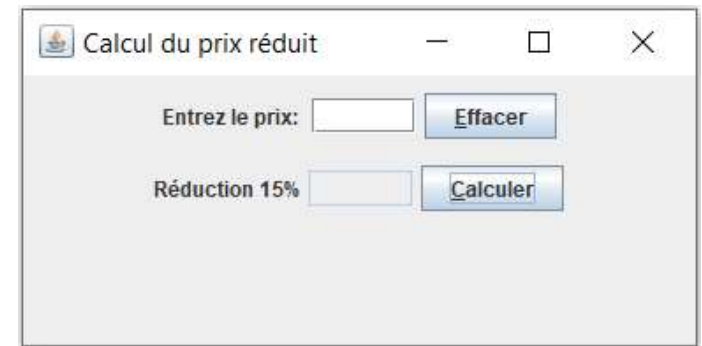
        // mettre les 2 boutons sur écoute
        btnEffacer.addActionListener(new CalculerListener());
        btnCalculer.addActionListener(new CalculerListener());

        panel1.add(lblPrix);
        panel1.add(txtPrix);
        panel1.add(btnEffacer);
        panel2.add(lblReduction);
        panel2.add(txtPrixReduit);
        panel2.add(btnCalculer);
        // ajouter le panneau à la fenêtre
        add(panel1);
        add(panel2);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```



Connaitre la source de l'évènement: getSource()

```
class CalculerListener implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == btnCalculer) {
            String prixTxt = txtPrix.getText();
            double prixDouble = Double.parseDouble(prixTxt);
            double prixReduit = prixDouble - prixDouble * 0.15;
            // txtPrixReduit.setText(String.valueOf(prixReduit));
            txtPrixReduit.setText(String.format("%.2f$", prixReduit));
            txtPrix.requestFocus(); // déplacer le curseur sur la zone de texte
            txtPrix.selectAll(); // sélectionner le contenu de la zone de texte
        } else { // la source est btnEffacer
            txtPrix.setText("");
            txtPrixReduit.setText("");
            txtPrix.requestFocus(); // déplacer le curseur sur zone de texte
        }
    }
}
```

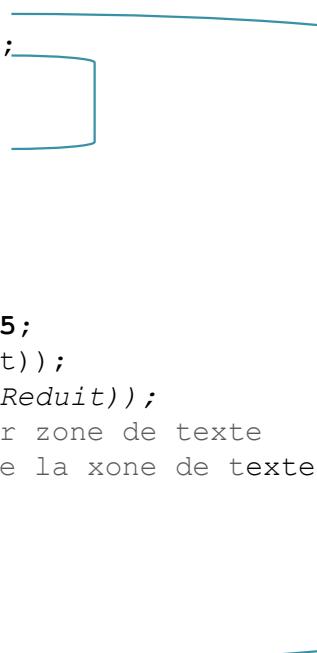


Utiliser plusieurs écouteurs

- Dans l'exemple précédent, on peut **utiliser deux gestionnaires (écouteurs)**:
 - Un gestionnaire pour effacer les zones de texte: **EffacerListener**
 - Un gestionnaire qui calcule et affiche le prix réduit: **CalculerListener**

```
public class FrmCalculer3 extends JFrame {
    private JButton btnCalculer = new JButton("Calculer");
    private JButton btnEffacer = new JButton("Effacer");
    ...
    public FrmCalculer3() {
        ...
        // mettre les 2 boutons sur écoute
        btnEffacer.addActionListener(new EffacerListener());
        btnCalculer.addActionListener(new CalculerListener());
        ...
    }
    class CalculerListener implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {
            String prixTxt = txtPrix.getText();
            double prixDouble = Double.parseDouble(prixTxt);
            double prixReduit = prixDouble - prixDouble * 0.15;
            // txtPrixReduit.setText(String.valueOf(prixReduit));
            txtPrixReduit.setText(String.format("%.2f$", prixReduit));
            txtPrix.requestFocus(); // déplacer le curseur sur zone de texte
            txtPrix.selectAll(); // sélectionner le contenu de la zone de texte
        }
    }

    class EffacerListener implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {
            txtPrix.setText("");
            txtPrixReduit.setText("");
            txtPrix.requestFocus(); // déplacer le curseur sur zone de texte
        }
    }
}
```



Traiter l'exception: le nombre saisi n'est pas du bon format

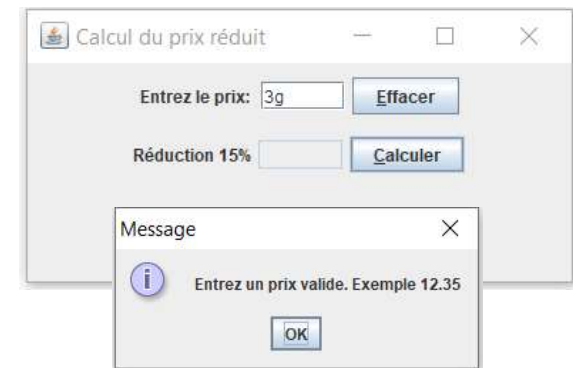
- Si on ne veut pas que le programme plante quand l'utilisateur saisit un prix qui n'est pas du bon format, on intègre la gestion des exceptions.
- L'instruction `Double.parseDouble(prixStr);` convertit la chaîne **prixStr** fournie comme argument en un nombre de type double. Cette instruction peut déclencher une exception de type **NumberFormatException** si l'argument n'est pas du bon format. Pour traiter cette exception et éviter ainsi que le programme plante, on l'entoure d'un bloc

```
try ..catch (NumberFormatException ex).
```

Exemple: FrmCalculer4.java

```
class CalculerListener implements ActionListener {
```

```
    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == btnCalculer) {
            String prixStr = txtPrix.getText();
            String messageErreur = " Entrez un prix valide. Exemple 12.35";
            try {
                double prixDouble = Double.parseDouble(prixStr);
                double prixReduit = prixDouble - prixDouble * 0.15;
                // txtPrixReduit.setText(String.valueOf(txtPrix));
                txtPrixReduit.setText(String.format("%.2f$", prixReduit));
                txtPrix.requestFocus(); // déplacer le curseur sur zone de texte
                txtPrix.selectAll(); // sélectionner le contenu de la zone de texte
            } catch (NumberFormatException ex) {
                JOptionPane.showMessageDialog(FrmCalculer4.this, messageErreur);
                txtPrix.requestFocus();
            }
        } else { // la source est btnEffacer
            txtPrix.setText("");
            txtPrixReduit.setText("");
            txtPrix.requestFocus();
        }
    }
}
```



Si la conversion plante, le bloc catch est exécuté,

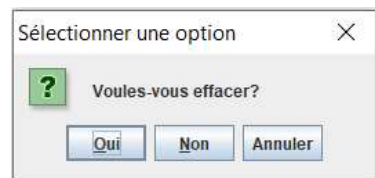
Les boîtes de dialogue

- Une boîte de dialogue est une fenêtre affichée par un programme ou par le système d'exploitation pour établir le dialogue avec l'utilisateur pour:

- informer l'utilisateur d'un évènement



- obtenir une information de l'utilisateur



- La classe **JOptionPane** et ses différentes **méthodes statiques** permettent de créer des boîtes de dialogue fournies par Java.

Les boîtes de dialogue: la boîte de message

- `static void showMessageDialog(Component parentComponent, Object message)`
 - **parentComponent** est le parent de la boîte de dialogue. Si le paramètre est null, la boîte de dialogue sera centrée à l'écran.
 - **message**: le message à afficher sur la boîte de dialogue.
- Exemple:

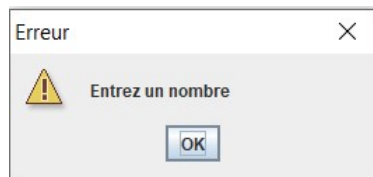
```
String messageErreur = " Entrez un prix valide. Exemple 12.35";
JOptionPane.showMessageDialog(null, messageErreur);
// la fenêtre s'affiche au milieu de l'écran.
```



- Si on veut que la boîte de dialogue s'affiche au milieu d'une fenêtre, on utilise:

```
JOptionPane.showMessageDialog(nomFenetre.this, messageErreur);
```
- La méthode `showMessageDialog` est surchargée et offre des paramètres supplémentaires qui permettent de personnaliser la boîte de dialogue.
- Exemple:

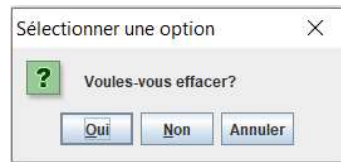
```
JOptionPane.showMessageDialog(FrmCalculer4.this, "Entrez un nombre", "Erreur" ,
JOptionPane.WARNING_MESSAGE);
```



Les boîtes de dialogue: la boîte de confirmation

- static **int** **showConfirmDialog**(**Component** parentComponent, **Object** message),
- La méthode **showConfirmDialog** retourne un entier qui indique sur quel bouton on a cliqué.
- Exemple: CalculerListener.java

```
int choix = JOptionPane.showConfirmDialog(null, "Voulez-vous effacer?");
//on récupère l'entier correspondant au bouton cliqué(yes, no, cancel, X)
if (choix == JOptionPane.YES_OPTION) {
    txtPrix.setText("");
    txtPrixReduit.setText("");
    txtPrix.requestFocus();
}
```



- Une fois la fenêtre de confirmation fermée, la méthode **showConfirmDialog** renvoie une valeur entière représentant le choix de l'utilisateur.
 - **JOptionPane.YES_OPTION**: le bouton **Oui** a été cliqué
 - **JOptionPane.NO_OPTION**: le bouton **Non** a été cliqué
 - **JOptionPane.CANCEL_OPTION**: le bouton **Annuler** a été cliqué
 - **JOptionPane.CLOSED_OPTION**: le bouton de fermeture (X) a été cliqué
-