IHM: MVC & Swing

Olivier Chapuis
chapuis@lri.fr
Merci à Stéphane Huot

Plan

Plan

Le modèle ‘Modèle-Vue-Contrôleur’

MVC c’est:

- Un patron de conception (une solution standardisée à un problème, indépendante des langages de programmation)
- Une architecture logicielle (une manière de structurer une application ou un ensemble de logiciels)

Introduit en 1979 par Trygve Reenskaug

Très fortement lié aux concepts de la programmation objet (Smalltalk).

MVC: structure du modèle

Organiser, structurer une application interactive en séparant:

- Les données et leurs traitements: **Le Modèle**
- La représentation des données: **La Vue**
- Le comportement de l'application: **Le Contrôleur**

MVC: le modèle

Noyau Fonctionnel de l'application

- Représente les données
- Fournit les accès aux données
- Fournit les traitements applicables aux données
- Expose les fonctionnalités de l'application

MVC: la vue

Sorties de l'application

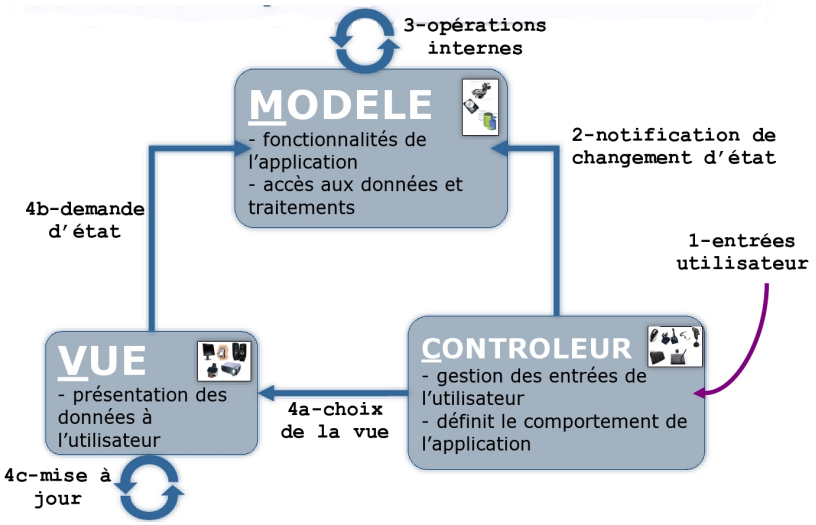
- Représente la (ou une) représentation des données du modèle
- Assure la consistance entre la représentation qu'elle donne et l'état du modèle/le contexte de l'application

MVC: le contrôleur

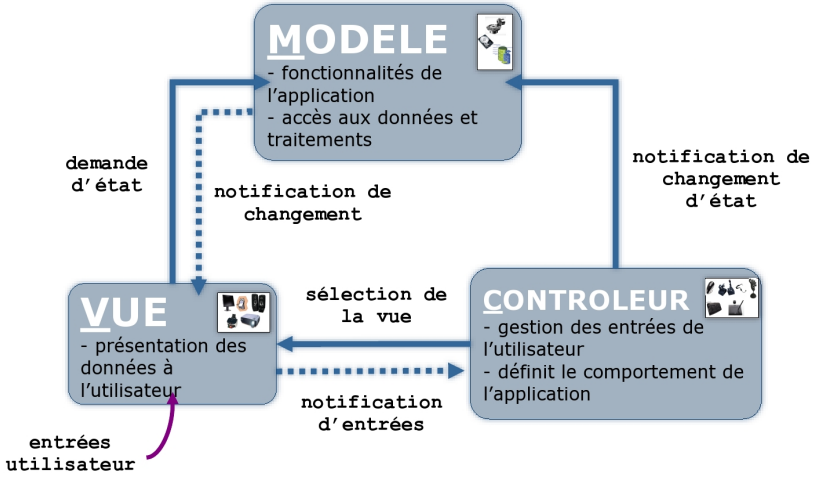
Comportement et gestion des entrées de l'application

- Représente le comportement de l'application face aux actions de l'utilisateur
- Fournit la traduction des actions de l'utilisateur en actions sur le modèle
- Fournit la vue appropriée par rapport aux actions de l'utilisateur et des réactions du modèle

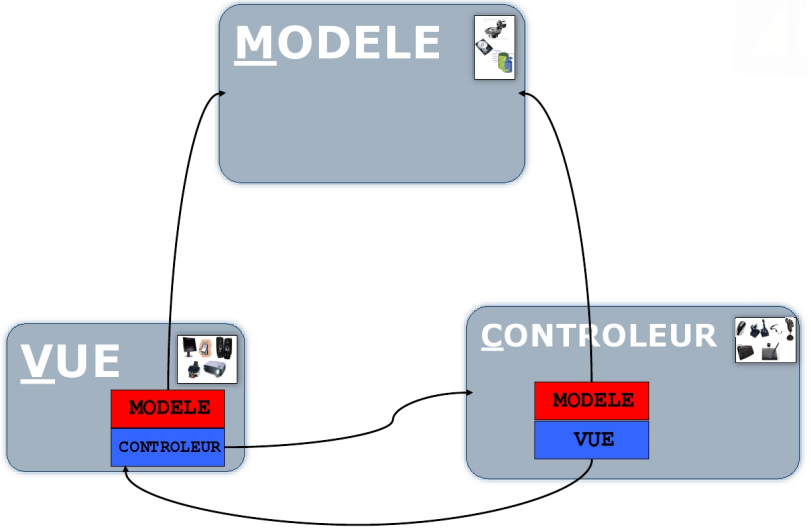
MVC: en résumé



MVC: Vision plus concrète



MVC: Références entre composants



Avantages et Inconvénients

Avantages

- Structure 'propre' de l'application
- Indépendance 'données' 'représentation' 'comportements'
- Modulaire et réutilisable Vues interchangeables – Contrôleurs interchangeables – Changement de 'Look & Feel'
- Facilite les vues et contrôleurs multiples

Inconvénients

- Mise en place complexe dans le cas d'applications importantes
- Mises à jour potentiellement trop nombreuses 'Spaghettis' dans le code – Temps d'exécution
- Contrôleur et Vue restent souvent fortement liés au Modèle

Complément à MVC: Service(s) Minimum

Séparer le noyau fonctionnel de l'interface est un bon principe... mais le noyau fonctionnel doit fournir un minimum de services pour pouvoir implémenter correctement l'interface

Exemples de services utiles (Fekete, 1996):

- la notification: possibilité pour un module externe d'être prévenu lorsque l'état du "modèle" change
- la prévention des erreurs: possibilité de savoir si un appel de fonction est licite dans un certain contexte
- l'annulation: possibilité de revenir à des états antérieurs

D'autres Modèles: Modèles à agents

Un système interactif peut être vu comme une collection d'agents, chaque agent ayant:

- un état
- une expertise
- et étant capable d'émettre et de réagir à des événements

Un interacteur est un agent réactif en contact direct avec l'utilisateur.

Remarque : tous ces agents sont réactifs, pas cognitifs (IA)

D'autres Modèles: PAC

PAC: Présentation - Abstraction - Contrôle. Un agent PAC incarne une compétence à un niveau d'abstraction donné composé de trois facettes

- présentation (le V+C de MVC) : définit le comportement perceptible de l'agent (entrée/sortie)
- abstraction (le M de MVC) : représente l'expertise de l'agent
- contrôle (pas d'équivalent dans MVC) : fait le lien entre A et P (pas de communication directe) et gère les relations avec les autres agents de la hiérarchie

L'interface est une structure hiérarchique d'agents PAC, de l'application à l'utilisateur

Comparaion PAC - MVC

Distinction entrées/sorties dans MVC mais pas dans PAC

- MVC offre une meilleure souplesse/réutilisation possible de composants d'interaction sur des vues différentes, mais nécessite une communication accrue entre les objets Vue et Contrôleur

Distinction entre "modèle" et "communication entre la vue et le modèle"

- cette distinction n'est pas faite dans MVC : risque d'amalgame au niveau du modèle
- dans PAC, distinction entre Abstraction et Contrôle

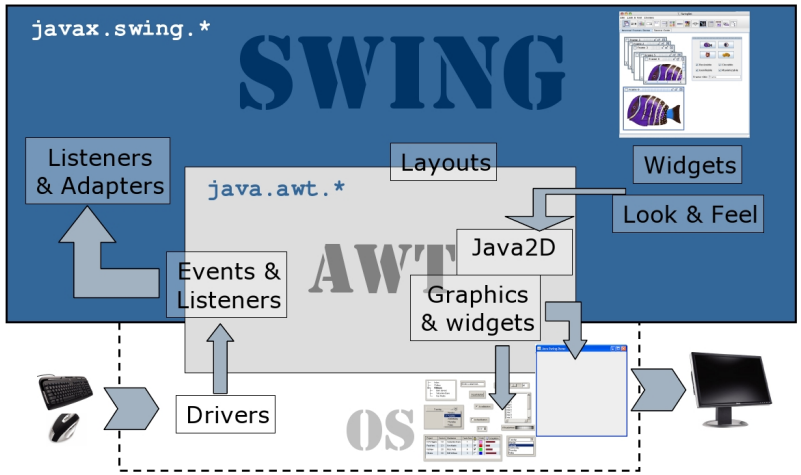
Plan

Boîtes à outils en Java

2 boîtes à outils dans l’API Java:

- AWT (Abstract Window Toolkit):
  - La bibliothèque historique (1995)
  - Bibliothèque graphique de base de l’API Java
- Swing:
  - La ‘nouvelle’ bibliothèque (1998)
  - Améliore les graphismes (Java2D) et les composants (plus complète)
  - MVC

AWT, Swing, Java2D, ...



Swing: composants de base

Représentation de la structure des widgets de l’interface sous forme d’un arbre

Container = widget générique qui peut contenir d’autres widgets
Structuration de l’interface graphique, Ordre et affichage des ‘fils’, Gestion du transfert des événements, Tous les widgets Swing sont des containers

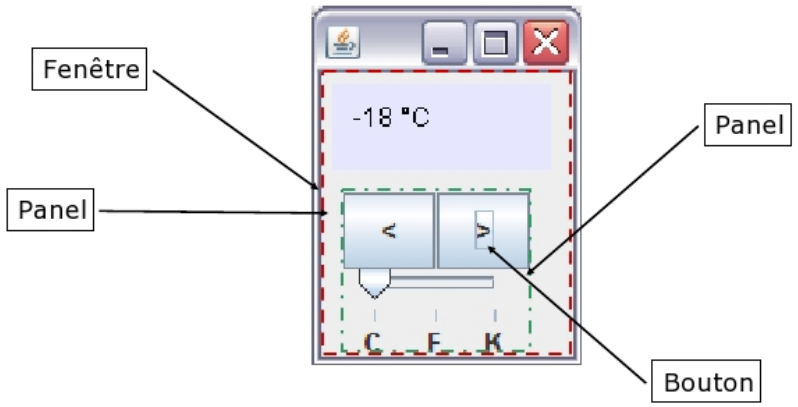
JFrame: Fenêtre Container de plus haut-niveau de la boîte à outils ‘Racine’ de l’interface graphique de l’application (créée dans la méthode main en général)

JComponent: Classe abstraite qui définit et fournit des méthodes de base pour tous les widgets

JPanel: Container concret de base

Ensuite les widgets: JButton, JTextArea, JRadioButton, ...

Container: exemples



Container

Règles:

- Pour apparaître à l'écran, les composants doivent appartenir à une hiérarchie de containers
- Un composant ne peut appartenir qu'à un seul container
- La racine d'une hiérarchie de container est un container de haut-niveau: Top-level container (JFrame, JPanel)

Fournit les méthodes de base pour la manipulation d'un ensemble de composants:

```
container.add(child);
container.remove(child);
container.removeAll();
Component[] container.getComponents();
```

..etc Voir la Javadoc de Container.

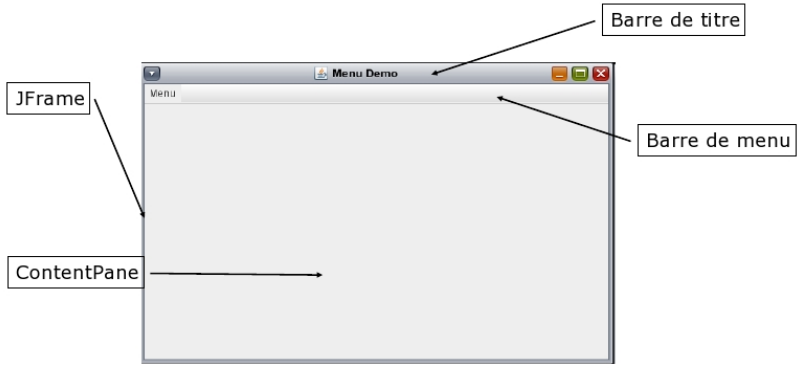
Fenêtre: JFrame

Fenêtres des applications, crées à partir du système de fenêtrage. Container de plus haut-niveau de la boîte à outils. C'est la 'racine' de l'interface graphique de l'application (créée dans la méthode main en général)

- Le contour et la barre de titre: système
- Le 'ContentPane': partie qui va contenir les composants de l'interface (Top-Level Container)
- Possibilité d'ajouter une barre de menu (JMenuBar)

```
JFrame frame = new JFrame();
frame.setLayout(monLayout); // Changement du layout
frame.setTitle("Mon Appli"); // Changement du titre (dans la barre)
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
...  etc ..
frame.setJMenuBar(maMenuBar); // Ajout d'une barre de menu
...  etc ..  // Ajout d'autres composants
frame.pack(); // Compactage
frame.setVisible(true); // Affichage de la fenêtre
```

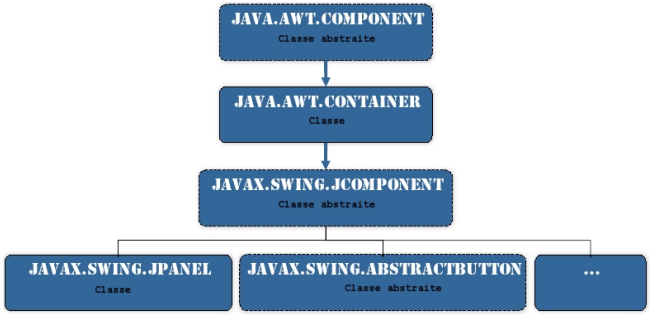

JFrame



JComponent

Classe abstraite qui définit et fournit des méthodes de base pour tous les widgets

- Mécanismes de 'Look & Feel' (apparence et comportement)
- Entrées souris et clavier
- Mécanisme de dessin et d'affichage (painting, borders, etc.)
- Gestion de la position/orientation, la taille, les couleurs, la police de texte, etc.



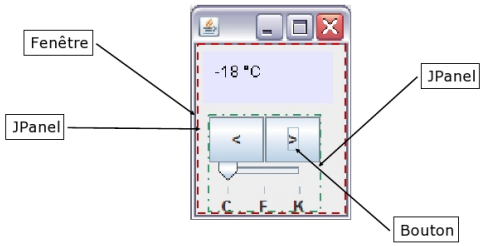
JComponent, méthodes de base

- Position et taille (peuvent dépendre du layout du container parent) : `Point getLocation()` ou `int getX()` et `int getY()`
  `Point setLocation(int x, int y)` `int getWidth()`, `int getHeight()` (largeur et hauteur)
  `int Rectangle getBounds()` (rectangle englobant)
  `Dimension getSize()` `setSize(Dimension d)`, `setPreferredSize(Dimension d)`, `setMaximumSize(Dimension d)`, `setMinimumSize(Dimension d)` (taille)
- Couleur de fond: `setBackground(Color c)` et `Color getBackground()`
- Couleur de premier plan (texte): `setForeground(Color c)` et `Color getForeground()`
- Police du texte: `setFont(Font f)` et `Font getFont()`
- Méthodes d'affichage: `paint(Graphics2D g)` (appelée par Swing)
  `paintComponent(Graphics2D g)` `paintBorder(Graphics2D g)` `paintChildren(Graphics2D g)` (appelées par paint, celles que l'on surcharge en général)

JPanel

Container concret de base. Permet de 'regrouper' des composants pour:

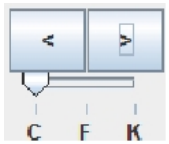
- Structurer l'interface graphique
  - Tâches de l'utilisateur
  - Placements
  - Couleurs
  - ...
- Structurer le code
  - Sections de codes / Classes
  - Comportement (contrôleur)
  - ...



JPanel, bases

- Création d'un JPanel: `JPanel panel = new JPanel();`
- Ajout d'un composant:
  `panel.add(child); //child est un Component`
- Retrait d'un composant:
  `panel.remove(child); //child est un Component`
- Ajout à un autre container: `container.add(panel);`
- Changement du layout : `panel.setLayout(monLayout);`
- Ajout d'une bordure:
  `panel.setBorder(new LineBorder(Color.BLACK));`
- Changement de la couleur de fond:
  `panel.setBackground(Color.RED);`
- Rendre le fond transparent: `panel.setOpaque(false);`
- Etc.
- Voir la Javadoc de JPanel...

JPanel. Exemple




```
JPanel buttons = new JPanel();
//Ajout de boutons au panel...
buttons.add(mButton);
buttons.add(pButton);
buttons.add(mSlider);
//...
//Ajout du panel à un autre container...
```


JButton

Un widget... bouton!

- Etend AbstractButton
- Affiche un bouton avec:
  - Du texte
  - Une image
  - Du texte et une image
- Mécanisme de raccourcis clavier (mnemonic)
- Comportement programmé à l'aide
  - D'Action
  - De Listeners

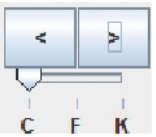
JButton, bases

- Création d'un JButton:
  ```
  //un bouton sans texte ni image
  JButton button = new JButton();
  //un bouton avec du texte
  JButton button = new JButton(String text);
  //un bouton avec une image
  JButton button = new JButton(Icon icon);
  ```
- Activation/désactivation:
  `button.setEnabled(boolean b);`
- Comportement:
  - Configuration de l'action:
    `button.setAction(Action a);`
  - Ajout d'un ActionListener:
    ```
    button.addActionListener(ActionListener l);
    //L'action à réaliser est programmée dans une
    //classe Action ou ActionListener
    ```


JButton. Exemple

```
JButton pButton = new JButton(">");
JButton mButton = new JButton("<");

//Ajout d'un 'contrôleur' sur le bouton "UP"
pButton.addActionListener(new
    ControleurThermometreButtons(modele, vue
    BUTTONS.UP));
//Ajout d'un 'contrôleur' sur le bouton "DOW
mButton.addActionListener(new
    ControleurThermometreButtons(modele, vue
    BUTTONS.DOWN));
//Ajout des boutons au panel
buttons.add(mButton);
buttons.add(pButton);
```



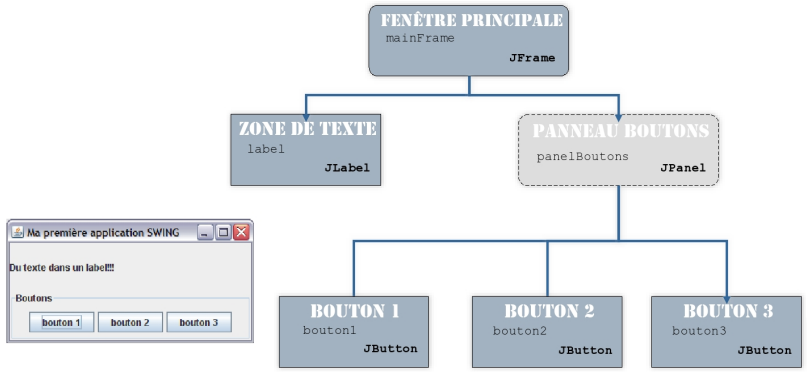
JButton, un peu plus

- Changement du texte :
  `button.setText(``Texte'');`
- 'Rollover':
  `button.setRolloverEnabled(true);`
- Images: `button.setIcon(Icon i);`
  `button.setPressedIcon(Icon i);`
  `button.setRolloverIcon(Icon i);`
  `button.setRolloverSelectedIcon(Icon i);`
  `button.setDisabledIcon(Icon i);`
- Etc.
- Voir la Javadoc de JButton...

Autres widgets...

- Texte:
  - JLabel
  - JTextField
  - JTextArea
  - ...
- Listes et arbres
  - JList
  - JTree
  - JComboBox
  - JMenu/JPopupMenu
- Choix
  - CheckBox
  - JRadioButton
- Dialogues
  - JDialog
  - JFileChooser
  - JColorChooser
  - ...
- ...

Arbre de widgets



Code 1

```
package gui;
import java.awt.FlowLayout;
import java.awt.GridLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.border.TitledBorder;
public class ApplicationSimple {
    public static void main(String[] args) {
       //Création de la fenêtre de l'application
       JFrame mainFrame = new JFrame("Ma première application SWING");
       //Changement du layout de la fenêtre
       mainFrame.setLayout(new GridLayout(2, 1));
       //Création du label contenant le texte
       JLabel label = new JLabel("Du texte dans un label!!!");
       //Création du panel de boutons
       JPanel panelBoutons = new JPanel();
       //Changement du bord du panel
       panelBoutons.setBorder(new TitledBorder("Boutons"));
       //Suite au prochain transparent...
```

Code 2

```
        //Création des 3 boutons
        JButton bouton1 = new JButton("bouton 1");
        JButton bouton2 = new JButton("bouton 2");
        JButton bouton3 = new JButton("bouton 3");
        //Changement du layout du panel de boutons et ajout des boutons
        panelBoutons.setLayout(new FlowLayout());
        panelBoutons.add(bouton1);
        panelBoutons.add(bouton2);
        panelBoutons.add(bouton3);
        //Ajout du label à la fenêtre
        mainFrame.add(label);
        //Ajout du panel de boutons à la fenêtre
        mainFrame.add(panelBoutons);
        //'Compactage' de la fenêtre
        mainFrame.pack();
        //On quitte l'application quand la fenêtre est fermée
        mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        //Affichage de la fenêtre
        mainFrame.setVisible(true);
    }
}
```


Layout

Structurer une interface graphique:

- Regrouper les contrôles de manière cohérente par tâches/fonctionnalités
- S'assurer du maintien de la cohérence
  - Plateforme et résolution d'affichage
  - Redimensionnement par l'utilisateur

Arrangement "semi"-automatique:

- Les LayoutManager

LayoutManager

Mécanisme de Swing pour:

- Placer les widgets dans un container
- Gérer les redimensionnements

Concerne les Containers

- Méthodes add spécialisées (paramètres de layout):
  `add(Component comp, Object constraints)`

A une influence sur les widgets (propriétés Size et Location)

Interface de AWT, implantée dans plusieurs classes de AWT ou Swing

LayoutManagers concrets

- BorderLayout (AWT):
  Divise le container en 5 zones (Centre, Nord, Sud, Est et Ouest)
- BoxLayout (Swing):
  Alignement des composants (axe X, axe Y, Line, Page)
- FlowLayout (AWT):
  Positionnement en flux selon la place disponible (Centré, Gauche ou Droite)
- GridLayout (AWT):
  Positionnement des composants dans une grille (avec tailles des cases égales)
- GridBagLayout(AWT):
  Positionnement dans une grille où les composants peuvent prendre plusieurs cases (utilisation de contraintes)
- Null:
  Pas de LayoutManager (positionnement des composants 'à la main')

LayoutManagers

- le layout est un problème complexe ... Encore des activités de recherche sur le placement des widgets!
- Le choix du LayoutManager dépend de ce que l'on veut faire... beaucoup de possibilités et besoin de pratique
- 'Détourner' et 'Jouer' avec les LayoutManagers pour arrive à ses fins ... Essayer, expérimenter... pratiquer
- Intérêt des JPanel pour structurer l'interface:
- Il existe des constructeurs d'interfaces pour java (InterfaceBuilders) mais besoin de savoir ce qu'il se passe 'sous le capot' pour pouvoir ajuster, paramétrer et prévoir
- Construction dynamique: ajout de composants et changement des layouts à l'exécution

Listeners

Littéralement: 'écouteurs'. Représentent le(s) contrôleur(s) de l'application

Parties du code de l'application qui vont être exécutées en réaction à des événements dans le modèle MVC (Entrées utilisateur, Changements d'état d'un composant de MVC)

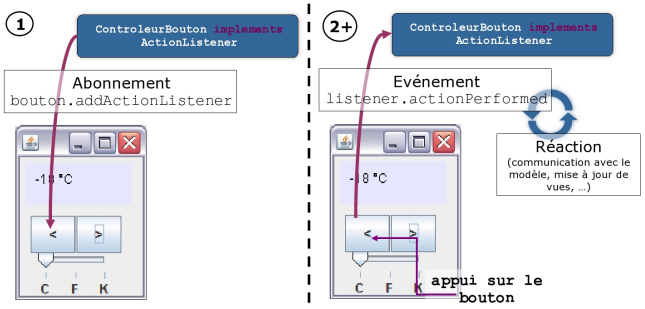
Observés = Widgets

- Mécanismes d'abonnement
- Mécanismes de notification

Observateurs = Listeners

- Interface EventListener de AWT et ses interfaces dérivées: ActionListener, ChangeListener, WindowListener, MouseListener, MouseMotionListener, etc.
- Implanter la (ou les) méthodes que doit appeler l'observé pour la notification: actionPerformed, stateChanged, etc.

Exemple




```
JLable temp = new JLabel("");
VueThermo vt =
    new VueThermo(temp);
JButton boutonM =
    new JButton("<");
...
bouton.addActionListener(
       new ctrlBouton(vt)));
...
```

```
public class ctrlBouton implements ActionListener {
    VueThermo vt;
    public ctrlBouton(VueThermo v) {
        this.vt = v;
    }
    public void actionPerformed(ActionEvent e) {
        vt.addTemp(-1);
    }
}
```


Exemple ... compressé

```
JLable temp = new JLabel("");
VueThermo vt = new VueThermo(temp);
JButton boutonM = new JButton("<");
...
bouton.addActionListener(
 new ActionListener() {
    public void actionPerformed(ActionEvent e) {
       vt.addTemp(-1);
    }
  }
);
```

Exemple ... plusieurs boutons

```
JLable temp = new JLabel("");
VueThermo vt = new VueThermo(temp);
JButton boutonM = new JButton("<");
JButton boutonP = new JButton(">");
...
...
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == boutonP) {
       vt.raddTemp(+1);
    }
    else if (e.getSource() == boutonM) {
       vt.raddTemp(-1);
    }
}
```

Variante: utiliser
`e.getActionCommand()` et `boutonM.setActionCommand("action")`

Listeners: utilisation

Les interfaces décrivent des Listeners avec une 'sémantique' différente, selon les événements écoutés:

- ActionListener: écouter des actions avec `actionPerformed (ActionEvent e)`
- ChangeListener: écouter des changements d'état avec `stateChanged(ChangeEvent e)`
- MouseMotionListener: écouter les mouvements de souris avec `mouseMoved(MouseEvent e)` et `mouseDragged(MouseEvent e)`
- MouseListener: écouter les actions sur la souris avec `mouseClicked(MouseEvent e)` `mouseEntered(MouseEvent e)`, etc.
- KeyListener: écouter les événements clavier avec `keyPressed(KeyEvent e)`, `keyReleased (KeyEvent e)`, etc.
- Etc... voir Javadoc...

Listeners: utilisation

Les widgets permettent de s'abonner à certains types d'événements:

- Component: addKeyListener, addMouseListener, addMouseMotionListener, etc.
- JFrame: addWindowListener, etc.
- JButton: addActionListener, addChangeListener
- ...

Javadoc: décrit pour chaque widget quels Listeners peuvent être attachés et quels événements sont déclenchés à quels moments