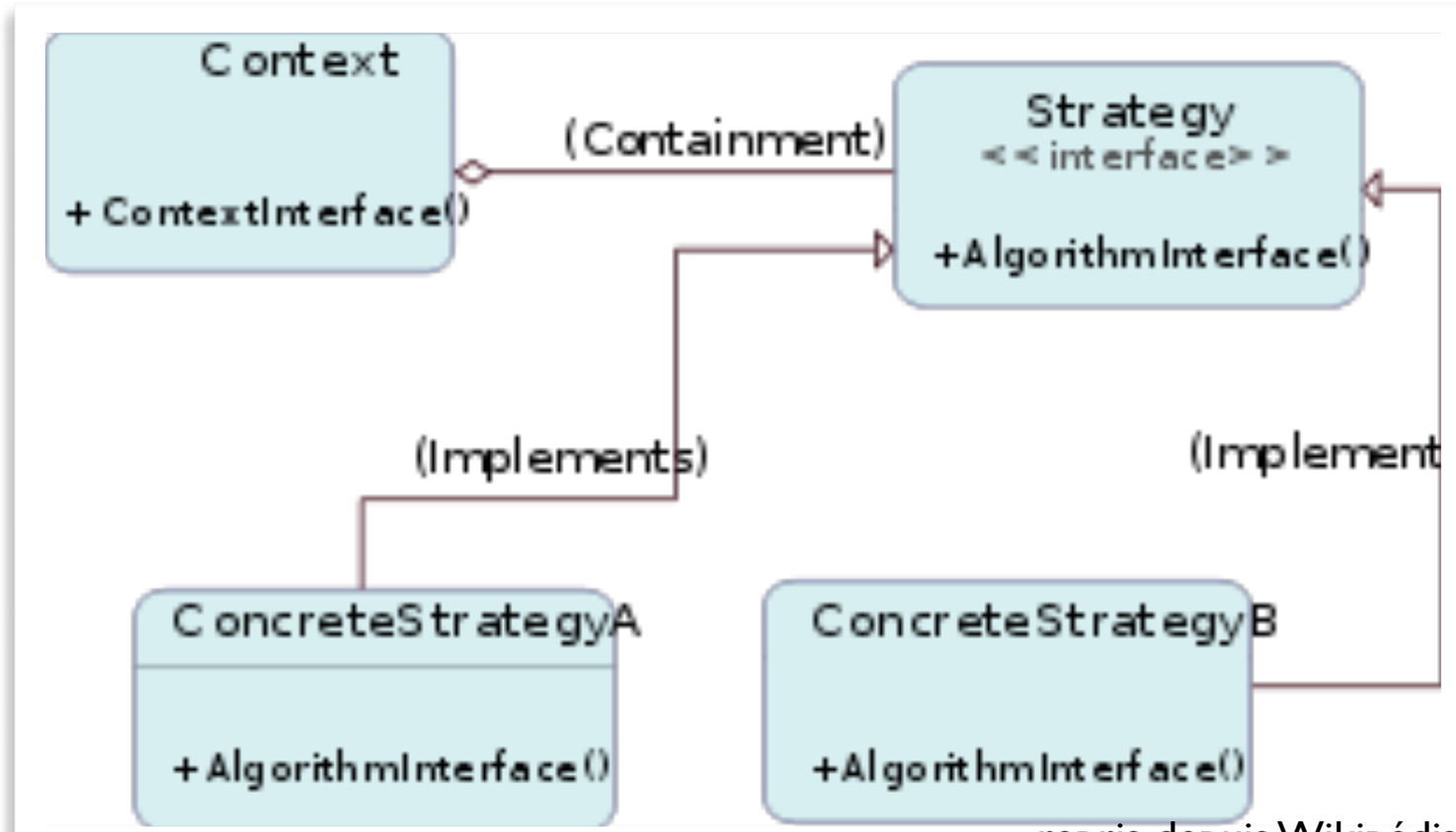
Interfaces Graphiques

Layouts - Le placement

Jean-Baptiste.Yunes@u-paris.fr
Université Paris Cité

- Rappel
  - un Layout est un *composant* du Strategy Pattern
    - sélection dynamique d'un algorithme
      - cet algorithme est chargé du placement des composants dans un container



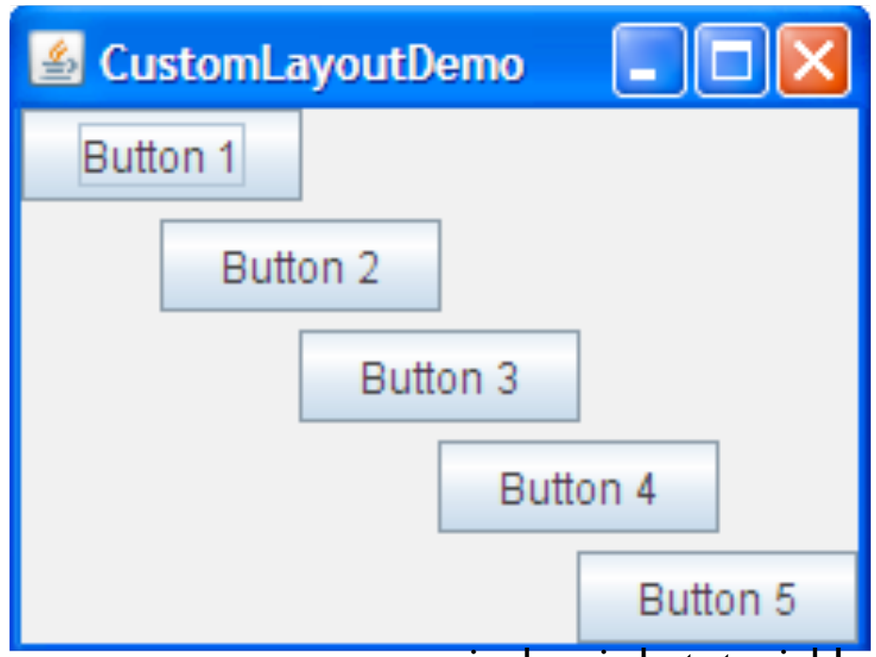
repris depuis Wikipédia

- l'interface `java.awt.LayoutManager`
  - `void addLayoutComponent(String, Component)`
    - gestionnaire annexe de la collection (ex : `BorderLayout`)
  - `void removeLayoutComponent(Component)`
    - quoi faire quand un composant est enlevé
  - `Dimension preferredLayoutSize(Container)`
    - est appelée lorsque il est nécessaire de déterminer la taille *idéale* du container (ex. : `pack()`).

- l'interface `java.awt.LayoutManager`
  - `Dimension minimumLayoutSize(Container)`
    - est appelée à différents moments lorsqu'il est nécessaire de déterminer la taille minimale requise pour afficher le `Container`
  - `void layoutContainer(Container)`
    - méthode *principale* des layout, dans laquelle est effectivement déterminée la taille et la position des composants contenus dans le `Container`, lequel les interrogent et les placent.

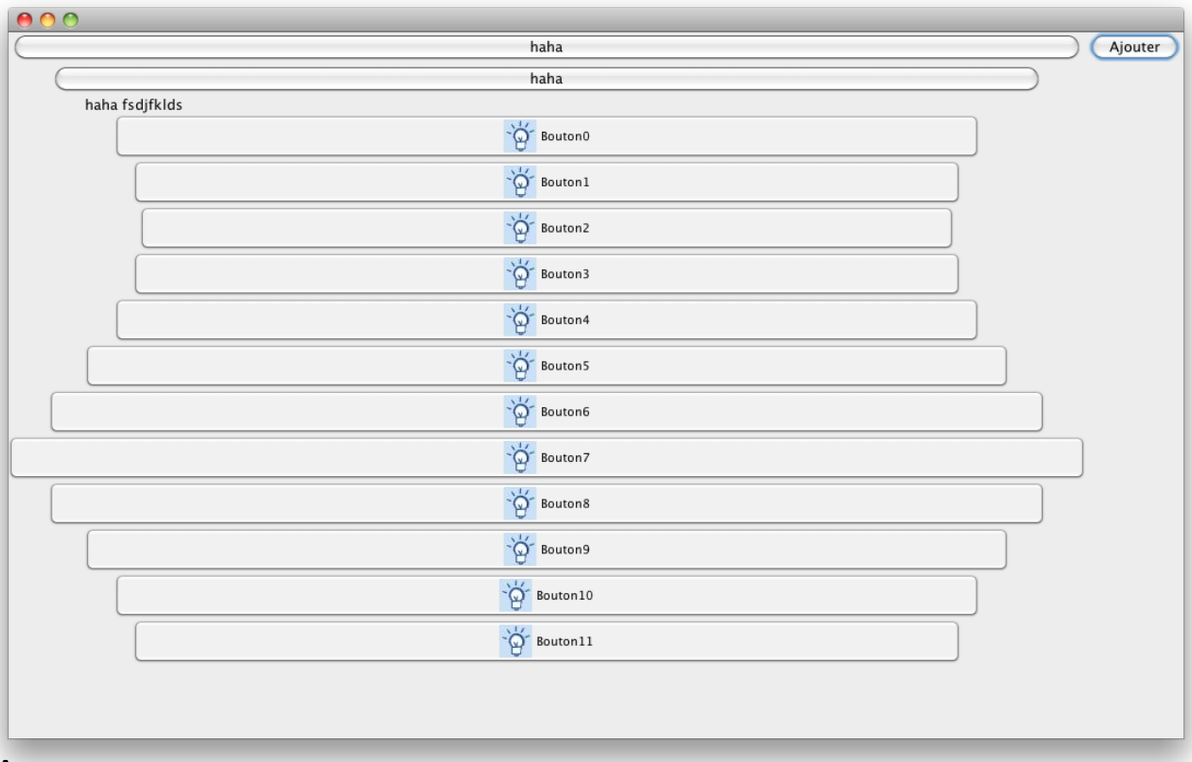
- l'interface `LayoutManager2` fournit d'autres méthodes :
  - `void addLayoutComponent(Component, Object);`
  - `float getLayoutAlignmentX(Container)`
  - `float getLayoutAlignementY(Container)`
  - `void invalidateLayout(Container)`
  - `Dimension maximumLayoutSize(Container)`

- deux exemples :
  - tutoriel Java : `DiagonalLayout`
    - code source fourni



repris depuis le tutoriel Java

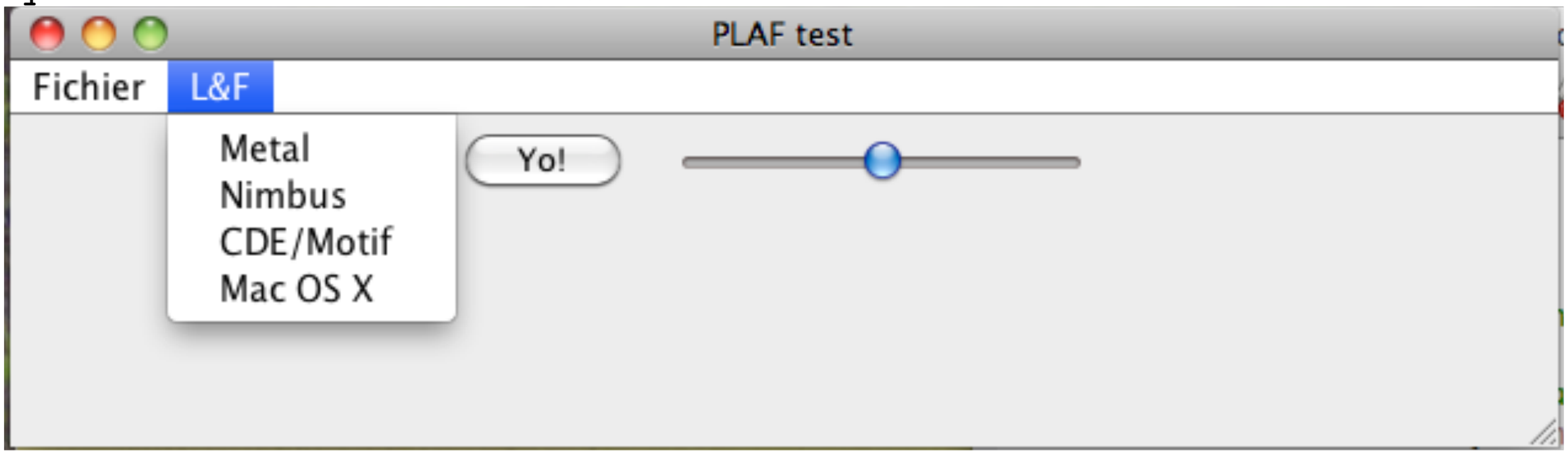
- exemples du cours : `SinLayout`
  - disponible depuis la page web du cours



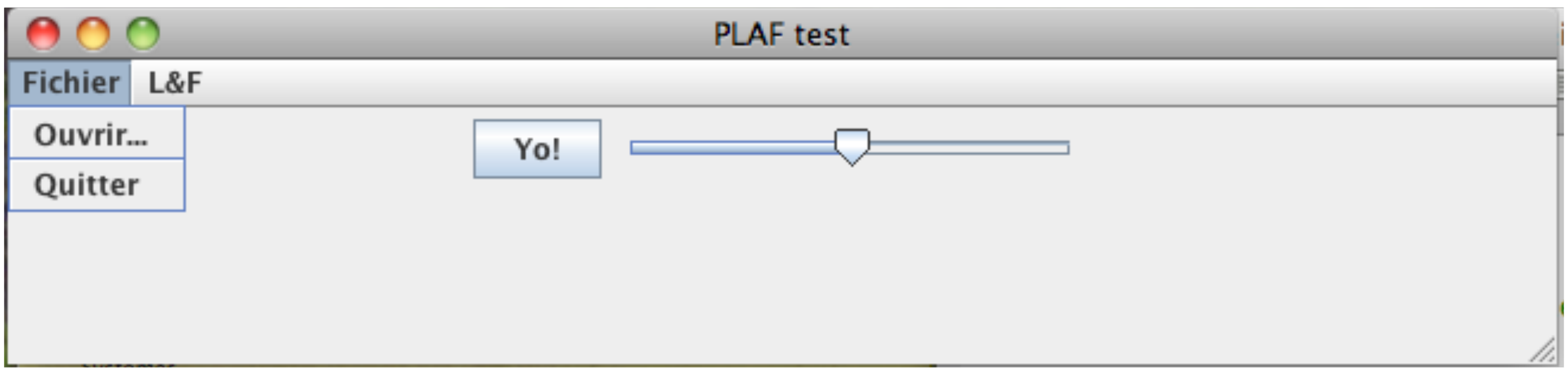
`SinLayout`

Les apparences de MacOSX Snow Leopard

Aqua Par défaut sous MacOSX

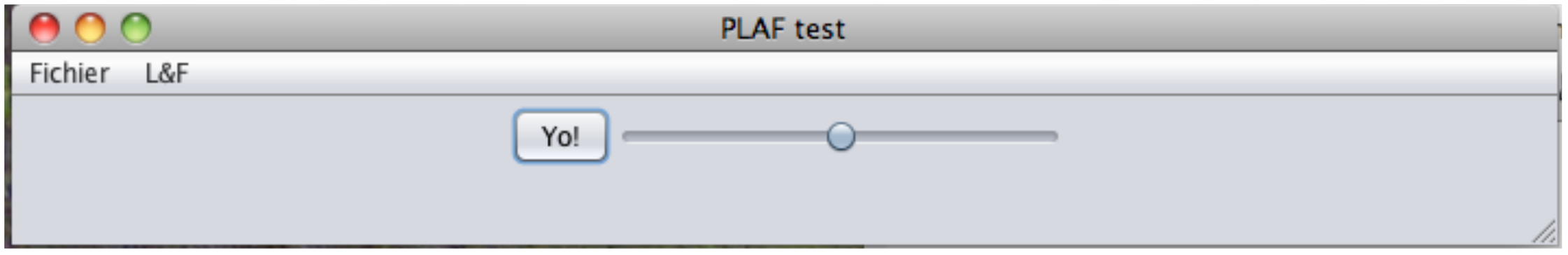


Metal

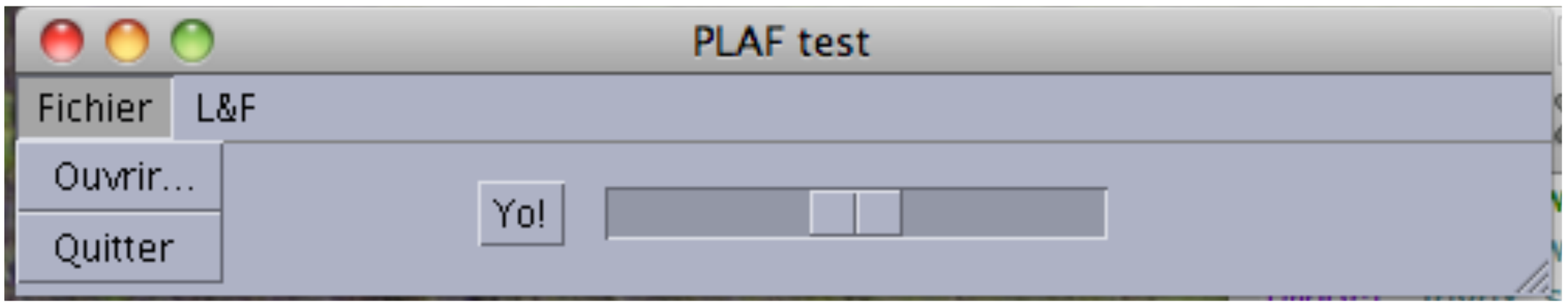


Les apparences de MacOSX Snow Leopard

`Nimbus`



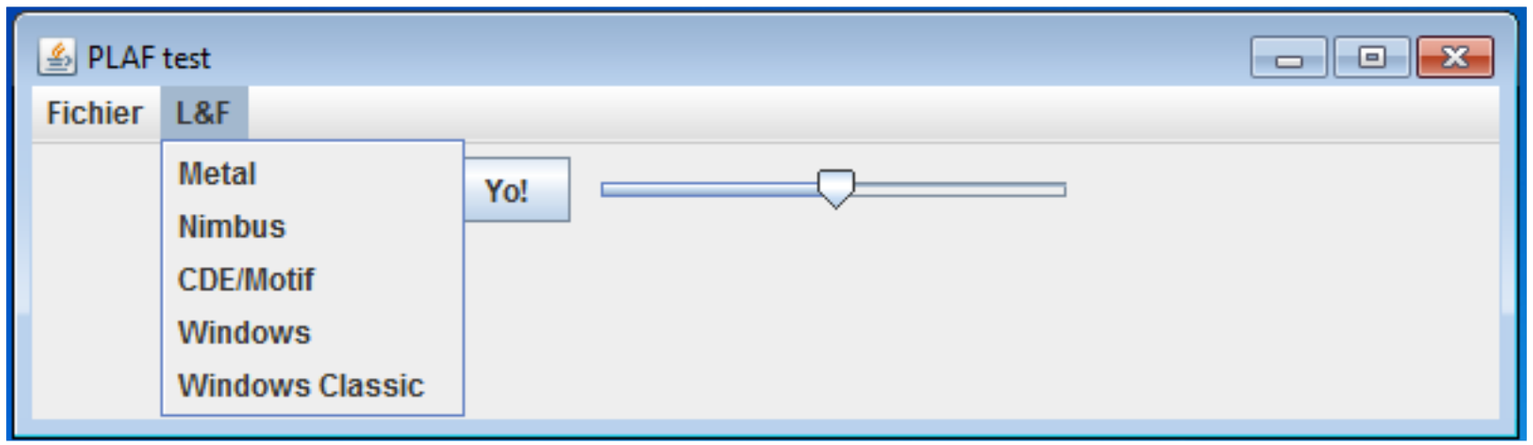
`CDE/Motif`



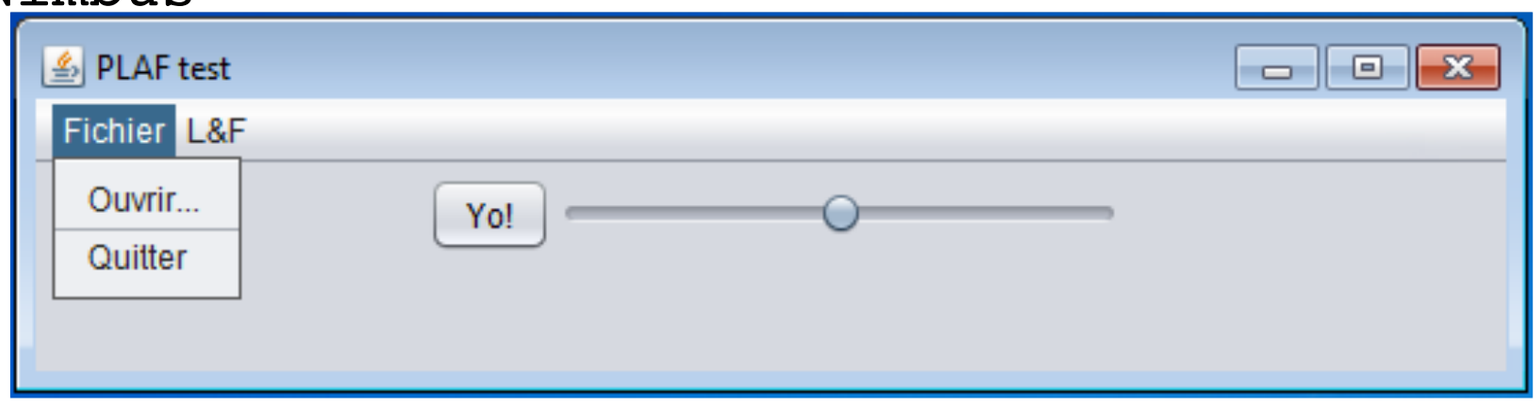
Look and Feel

Les apparences sous Windows 7

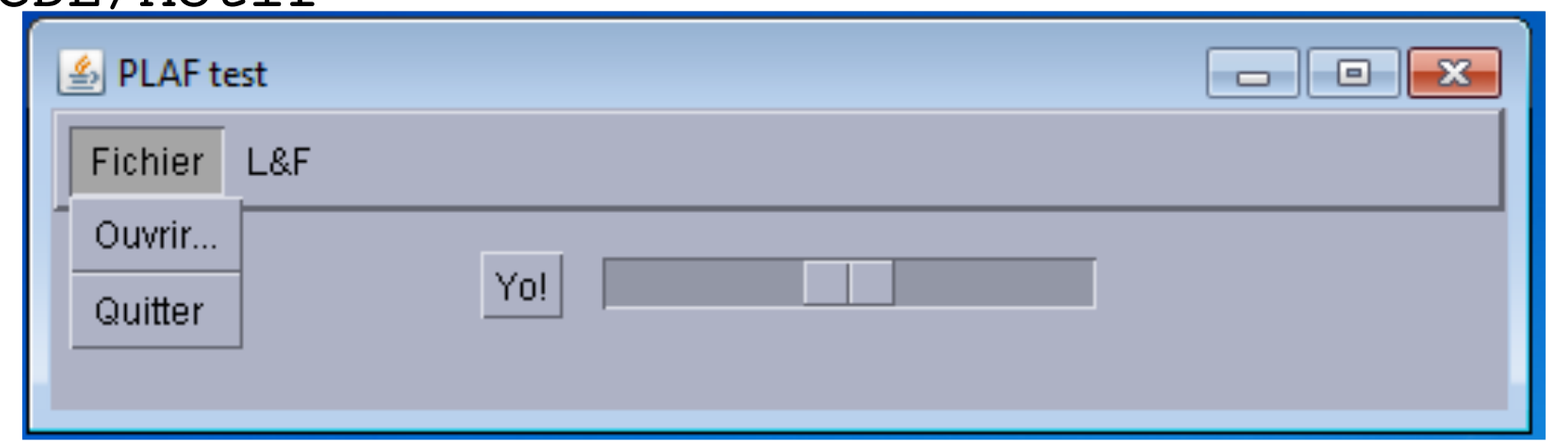
Metal



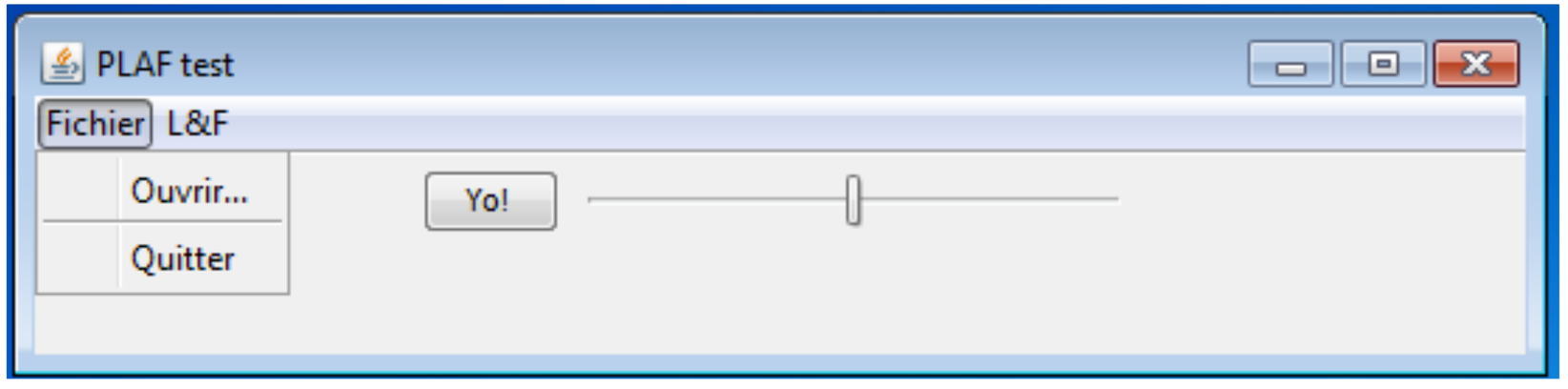
Nimbus



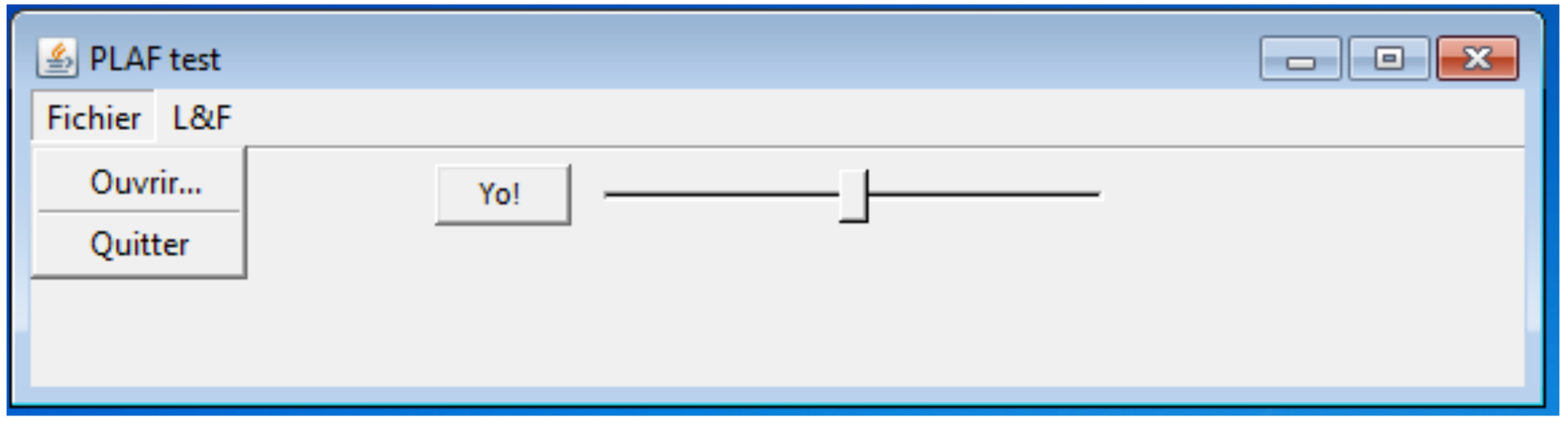
CDE/Motif



Les apparences sous Windows 7



Classic



- La classe `UIManager` propose des services utilitaires de gestion des Apparences (Look-and-Feels)
  - apparences disponibles
    - `UIManager.LookAndFeelInfo [] getInstalledLookAndFeels();`
  - modification de l'apparence courante
    - `setCurrentLookAndFeel(LookAndFeel)`
  - etc.

- `UIManager.LookAndFeelInfo` :
  - décrit une apparence
    - `getName();`
      - un nom court à employer pour représenter l'apparence dans une interface
    - `getClassName();`
      - le nom complet à employer pour sélectionner une apparence

- Si la modification est opérée dynamiquement
  - demander à l'interface de se remettre entièrement au goût du jour
    - `SwingUtilities.updateComponentTreeUI(Component);`
      - éventuellement `pack()`...
- Exemple : `PLAF.java`

- L'apparence `Metal` autorise la création de variations
  - Thèmes
    - `DefaultMetalTheme`, `OceanTheme`
      - à sous-classer et modifier pour obtenir une variante...
    - `MetalLookAndFeel.setCurrentTheme(MetalTheme)`
- Exemple : `PLAFTheme.java`

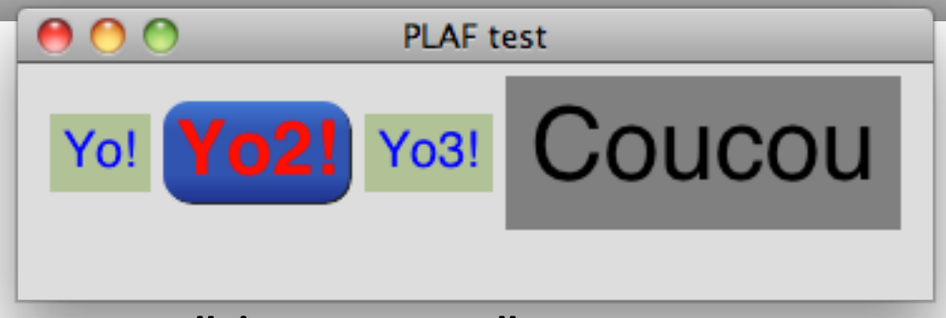
- La création d'une apparence s'obtient :
  - par programmation
  - par une définition externe (XML)

- Technique XML :
  - créer une instance de `SynthLookAndFeel`
  - lui associer une description XML :
    - `load(InputStream,Class<?>);`
      - le premier paramètre est le flux de lecture du fichier XML
      - le second paramètre servira à déterminer comment charger des resources associées (`ClassLoader`, etc.)

- les composants appartiennent tous à une classe et l'on peut définir une apparence pour toutes les instances d'une classe donnée
- mais, les composants peuvent être nommés *via*
  - `setName(String);`
    - ce qui autorise à définir certaines particularités pour certaines instances particulières (boutons de dialogues, etc)
- mécanisme similaire à class/id de HTML

- la balise `<style id="identificateur">…</style>`
  - permet de définir des caractéristiques d'apparence
- la balise `<bind style="id" type="name|region" value="exp"/>`
  - permet d'associer le style d'identité *id* aux objets correspondants à l'*exp*ression
    - `name` : objets nommés *exp*
    - `region` : objets génériquement nommés *exp*

Création d'un Look and Feel via XML




```
<style id="yo2Style">
  <!--opaque value="TRUE"/-->
  <property key="Button.textShiftOffset" type="integer"
            value="1"/>
  <font name="Helvetica-Bold" size="30"/>
  <insets top="5" left="5" right="5" bottom="5"/>
  <state>
    <imagePainter method="buttonBackground" path="button.png"
                  sourceInsets="20 20 20 20"/>
    <color value="GREEN" type="BACKGROUND"/>
    <color value="RED" type="TEXT_FOREGROUND"/>
  </state>
  <state value="PRESSED">
    <imagePainter method="buttonBackground" path="button3.png"
                  sourceInsets="20 20 20 20"/>
    <color value="GREEN" type="BACKGROUND"/>
    <color value="#FF6666" type="TEXT_FOREGROUND"/>
  </state>
</style>
<bind style="yo2Style" type="name" key="YO2"/>
```

Création d'un Look and Feel via XML

- Un exemple : `PLAFSynth.java`