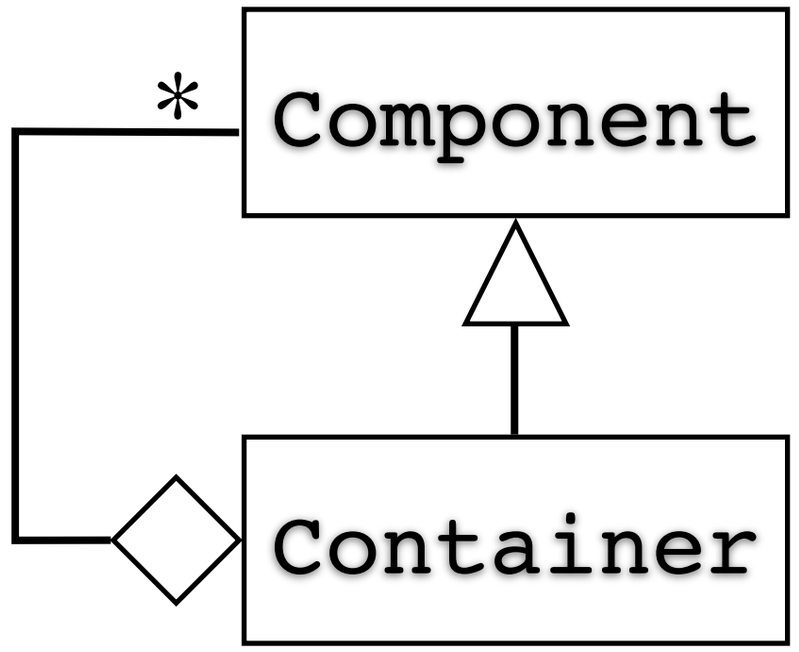
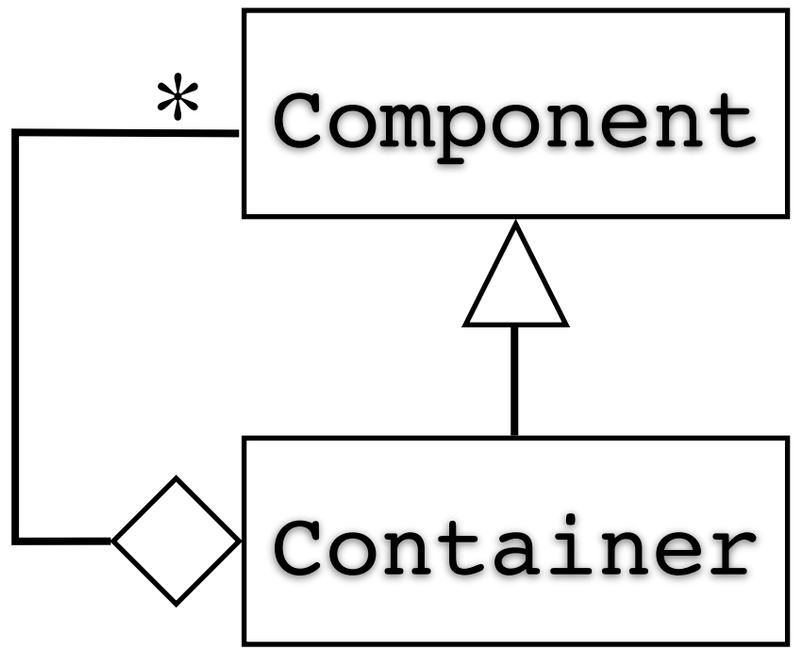
Université Paris Cité

Interfaces Graphiques
Les Composants

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

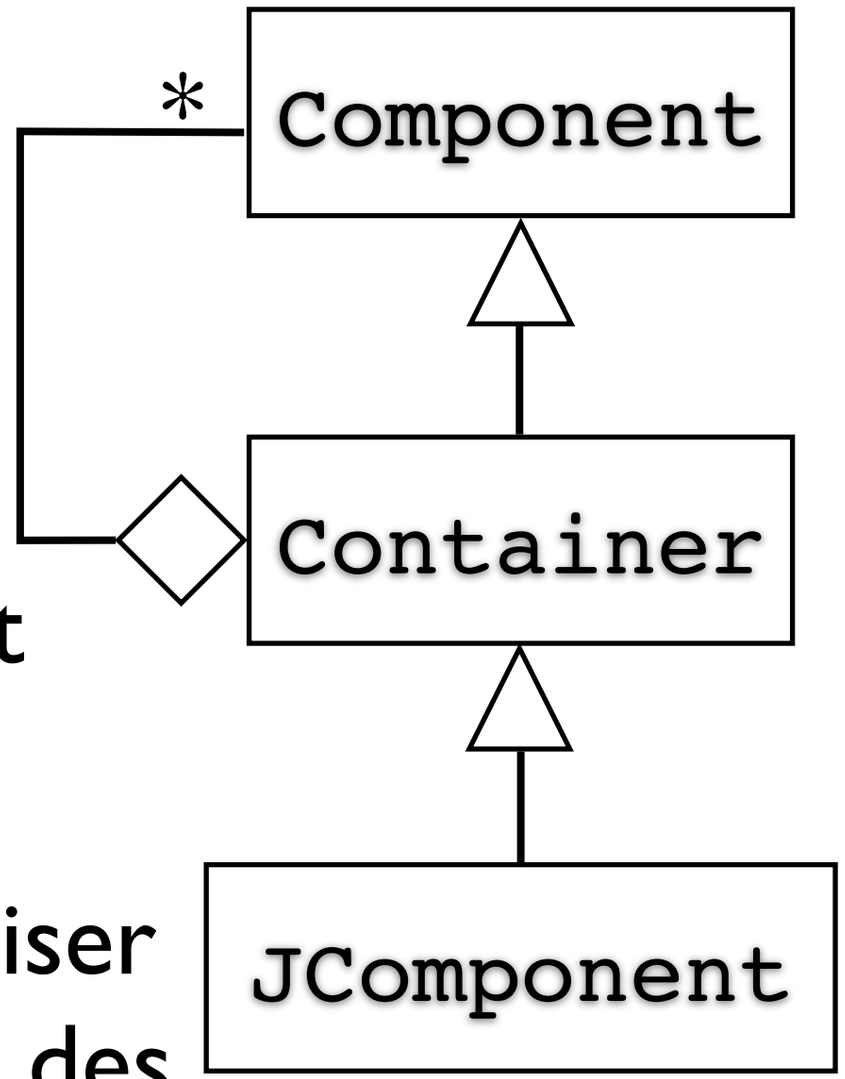


- Interface
  - une interface est obtenue par agrégation de composants
    - des emboîtements successifs de
      - boîtes (containers)
      - objets (composants simples)

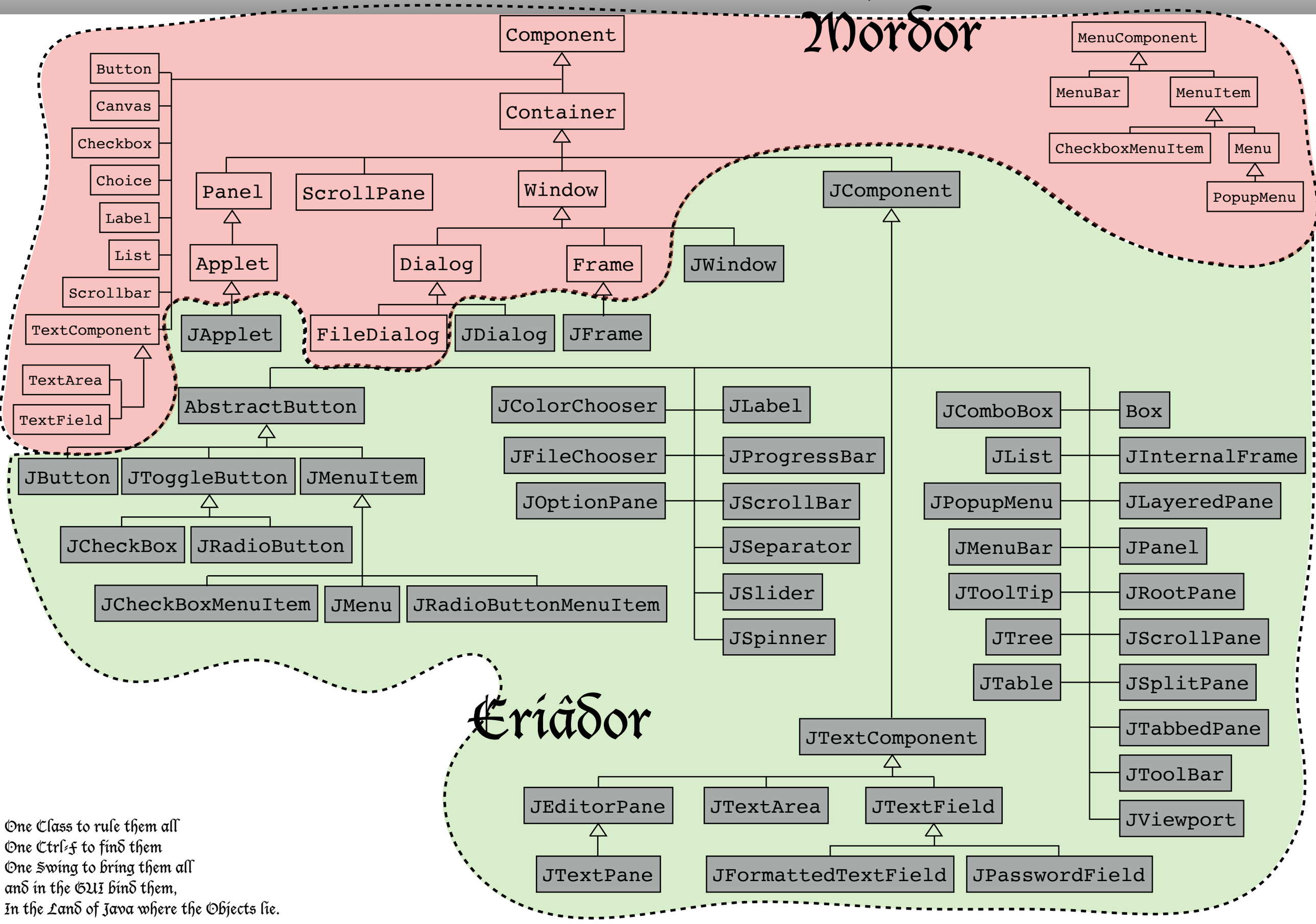


- Donc deux types d'objets GUI
  - les composants
  - les containers
    - qui sont aussi des composants...

- (bizarrerie?) Swing
  - les `JComponents` Swing sont des `Container` AWT…
  - il n'est pas conseillé de les utiliser comme tels… sauf dans le cas des *containers* Swing…



Panorama (non-exhaustif) aka les-terres-du-milieu.java in JTolkien package



One Class to rule them all
One Ctrl-F to find them
One Swing to bring them all
and in the GUI bind them,
In the Land of Java where the Objects lie.

- Les composants (`java.awt.Component`) :
  - un nom (`name`)
  - une taille et position (`x`/`y`/`width`/`height` - `size`/`location`)
  - visible ou non (`visible`)
  - réceptacle de divers événements
- Il est **très important de rendre visible** les composants sous peine… d'invisibilité !

- Les containers (java.`awt.Container`) :
  - des méthodes de gestion de la relation d'agrégation Container/Composant
    - `add/remove/getComponent/ getComponentCount/getComponentAt/ getComponents...`
  - des méthodes de gestion de la disposition (layout)
  - une police par défaut (font)

JComponent

- Les composants Swing (`javax.swing.JComponent`) :
  - support pour une apparence dynamique (pluggable look-and-feel)
  - amélioration de la gestion du clavier
  - support pour info-bulles
  - support pour l'accessibilité
  - support pour stockage de propriétés spécifiques
  - support amélioré pour le dessin (double-buffering, bords)

désormais

sauf exception nécessaire

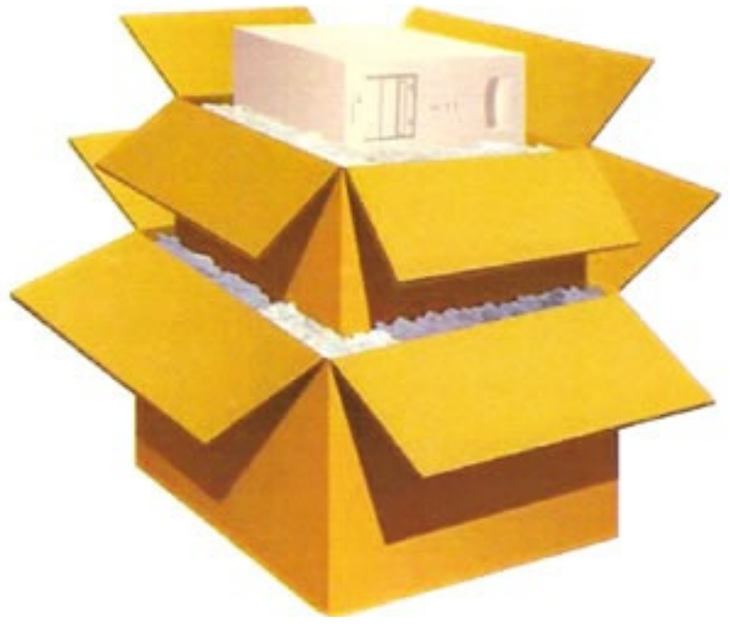
nous ne préoccuperons plus que de Swing…

- Quelques propriétés des `JComponents` :
  - `opaque` (`boolean`)
  - `background`/`foreground` (`Color`)
  - `font` (`Font`)
  - toolTip (`String`)

- illustration avec un `JLabel` (`JComponentExemple.java`)
  - précaution : rendre le `JLabel` opaque car par défaut son fond est transparent…

Containers

- Les Containers
  - des boîtes (2D)
  - ont pour rôle de contenir d'autres composants



Containers racine

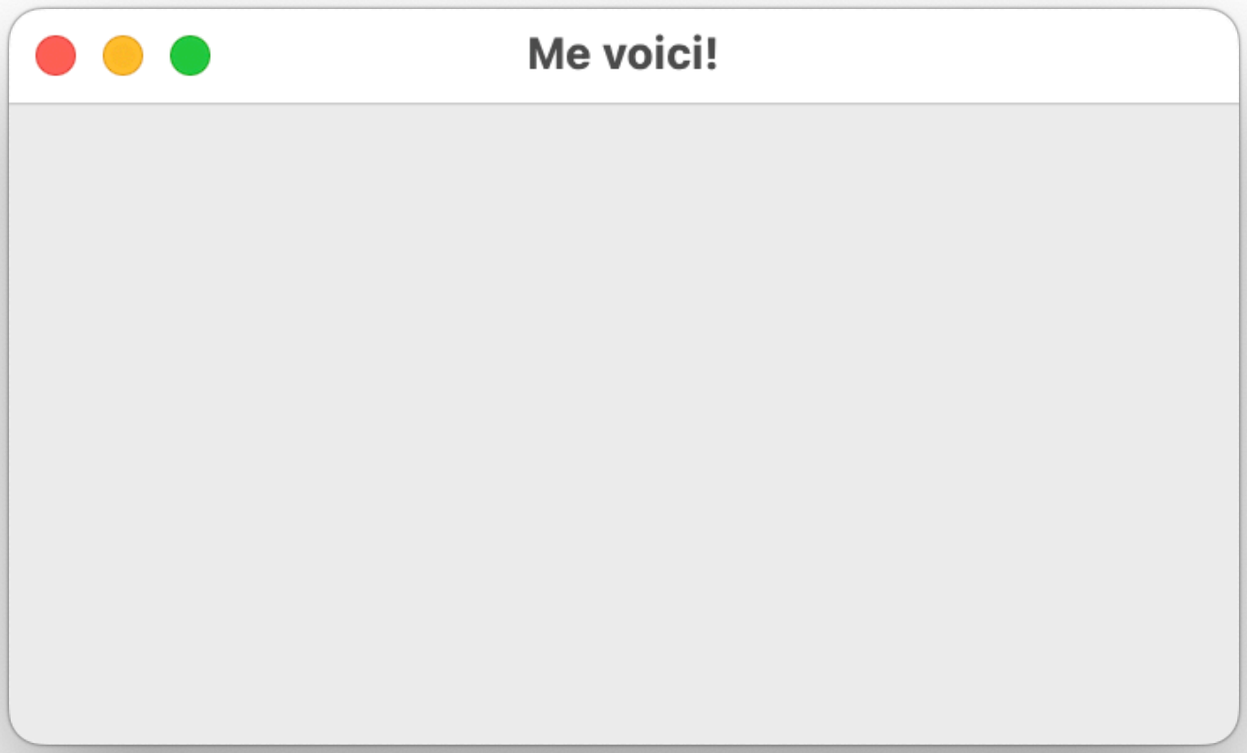
- Containers racine
  - ~~`JApplet`~~
  - `JDialog`
  - `JFrame`
  - `JWindow`
- au moins un par application
- manipulables directement par le window manager

Containers racine

- `JFrame` **vs** `JWindow`
  - décoration...
- `JDialog` ?
  - en général utile dans le contexte d'une autre fenêtre, on verra plus tard…
- ~~`JApplet`~~ ?

Containers racine

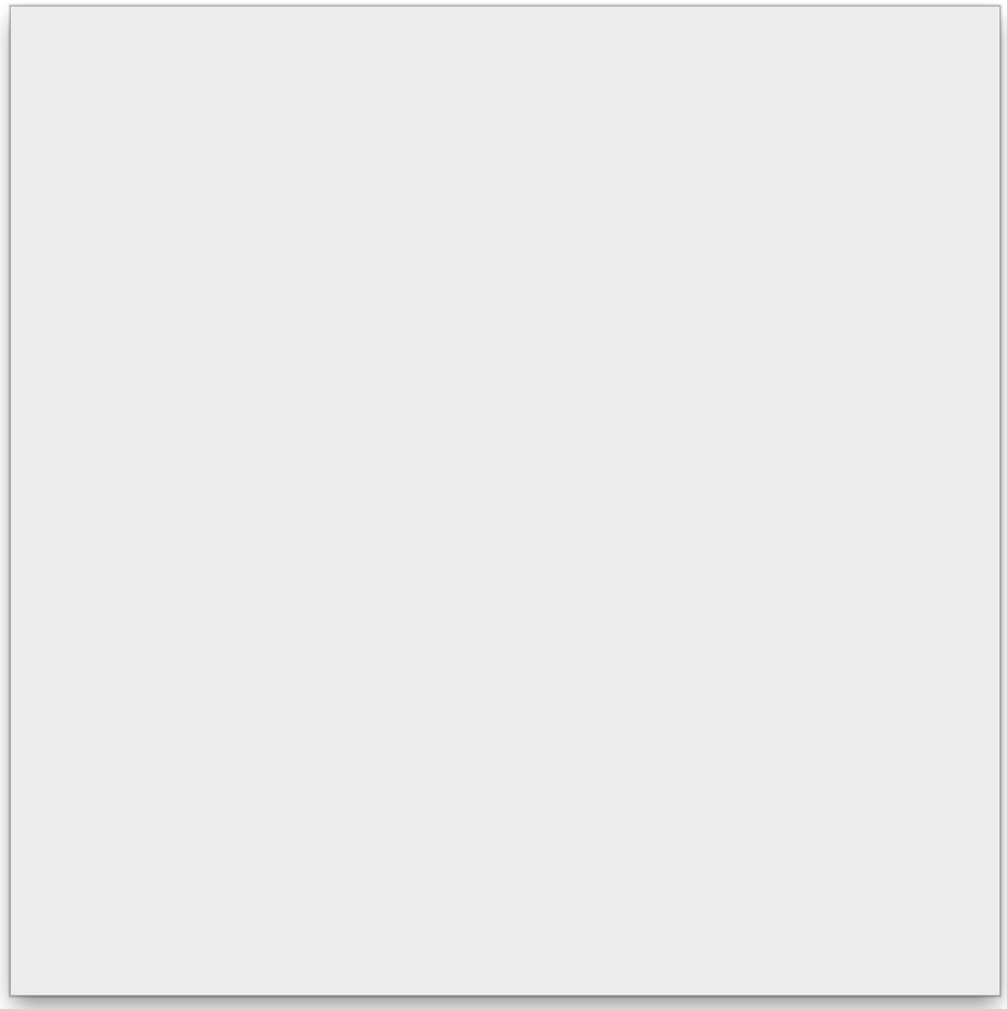
- `JFrame(String title)`
  - contient un unique `JRootPane`
    - peut être remplacé
    - ne peut être enlevé
  - peut être associée à une `JMenuBar`



`JFrameExemple.java`

Containers racine

- `JWindow()` / `JWindow(Frame owner)` / `JWindow(Window owner)`
  - contient un unique `JRootPane`
    - peut être remplacé
    - ne peut être absent
  - pas de barre de menu…



`JWindowExemple.java`

Containers racine

- Rappel : les containers racines peuvent être visibles ou non :
  - `setVisible(boolean)`
- il **ne faut pas oublier** des les rendre visibles sous peine d'invisibilité…
- il **ne faut pas** les rendre visible trop tôt! Pour éviter des effets désagréables de construction visible de l'interface et de performances…

Containers racine

- `JWindow`, `JFrame`
  - des capsules pour un container *utilisateur*
  - le container principal (`JRootPane`) est accessible *via*
    - `Container getContentPane()`
    - `setContentPane(Container)`

Containers

- Containers ordinaires
  - `JPanel`
  - `JScrollPane`
  - `JSplitPane`
  - `JTabbedPane`
  - `JToolBar`
- permettent la division d'un espace existant
  - ne sont utilisables que dans d'autres containers

- Les containers ordinaires
  - un espace de rangement d'autres composants, l'agrégation vit grâce à :
    - `add(Component)` / `remove(Component)`
  - comment les composants sont-ils rangés/placés ?
  - comme on veut… on le verra plus tard…

- `JPanel`
  - le plus simple, un espace rectangulaire
  - un simple panneau d'affichage

`JPanelExemple.java`

`JPanelExemple2.java`

- `JScrollPane`
  - une fenêtre sur un espace rectangulaire déplaçable

`JScrollPaneExemple.java`

- `JSplitPane`
  - divise un espace verticalement ou horizontalement en deux parties dont la somme est l'espace entier

`JSplitPaneExemple.java`

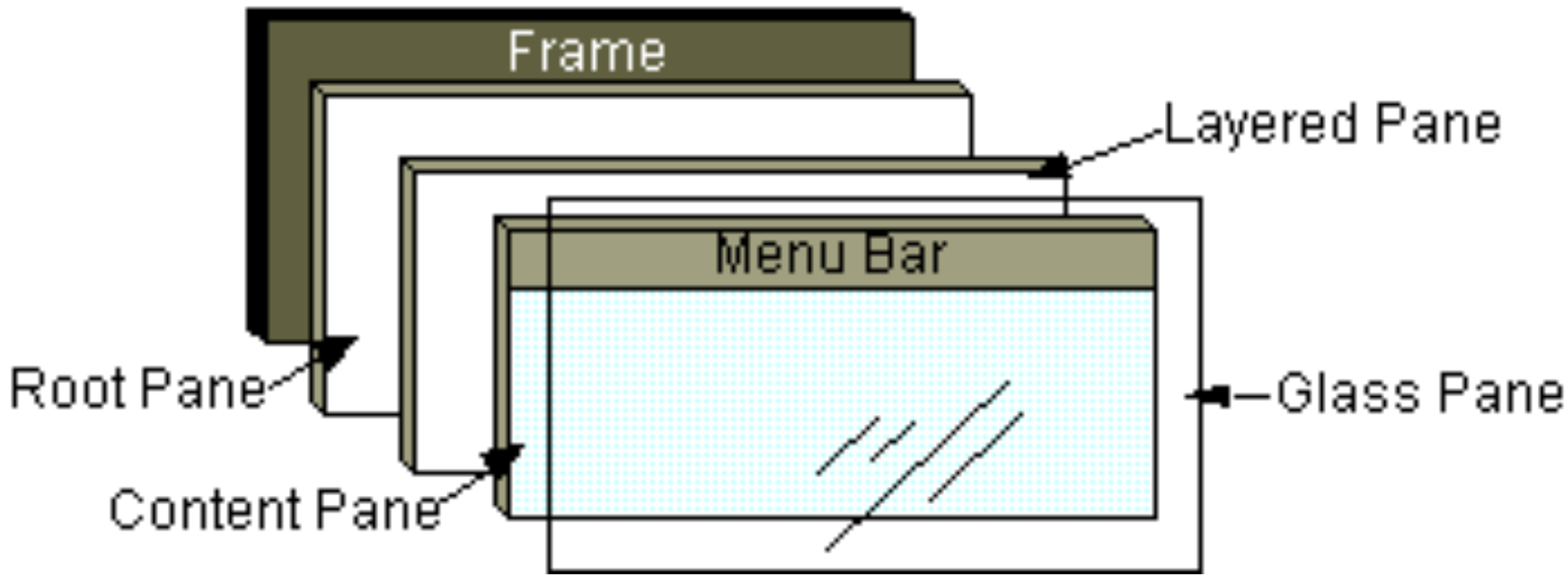
- `JTabbedPane`
  - une pile d'espaces tous de même dimension

`JTabbedPaneExemple.java`

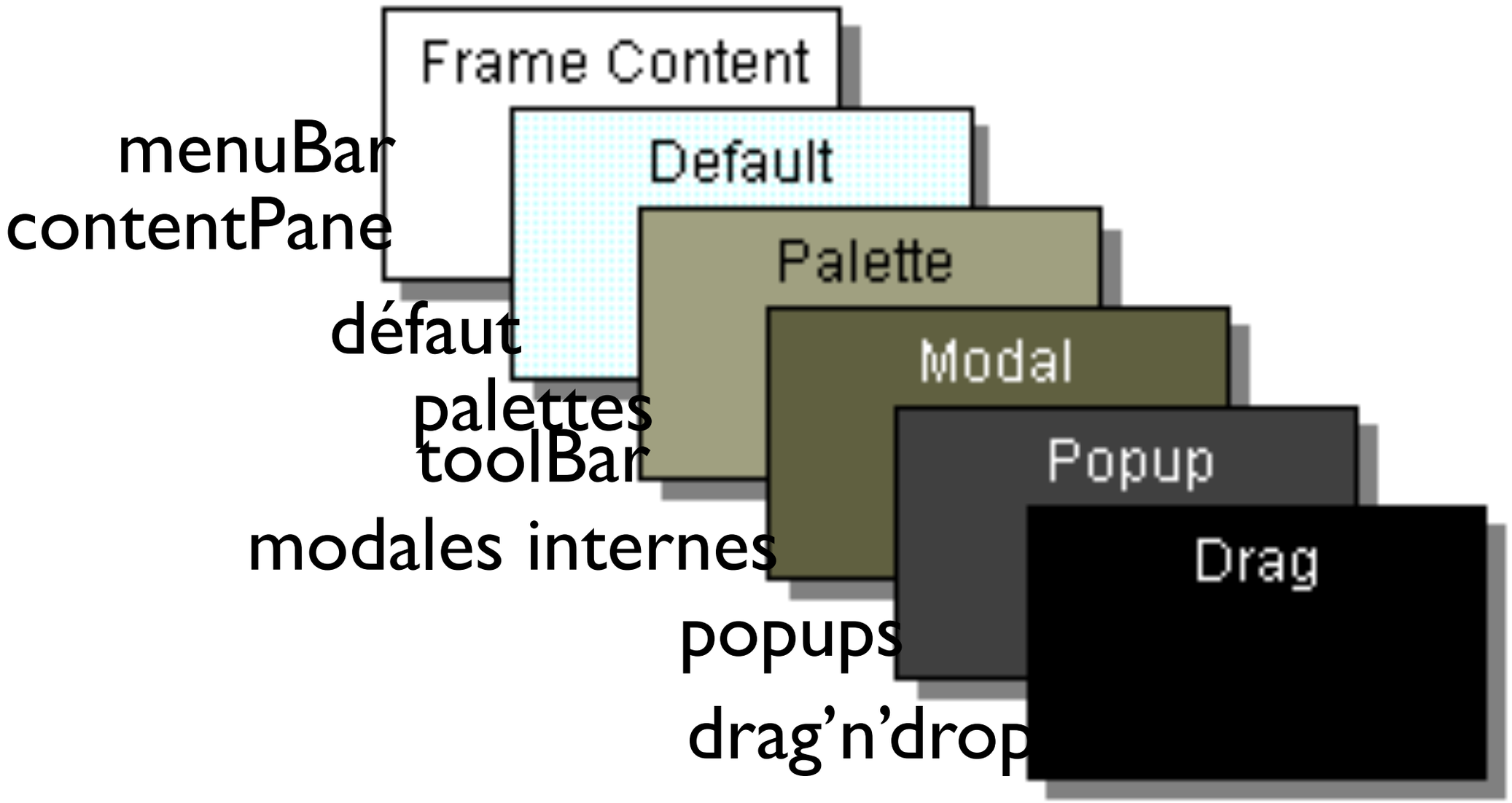
- `JToolBar`
  - un espace linéaire de rangement
  - utilise des `Actions`

- Containers spéciaux :
  - `JRootPane`
  - `JLayeredPane`
    - `JInternalFrame`
    - `JDesktopPane`
- Leur usage est plus anecdotique ou technique…

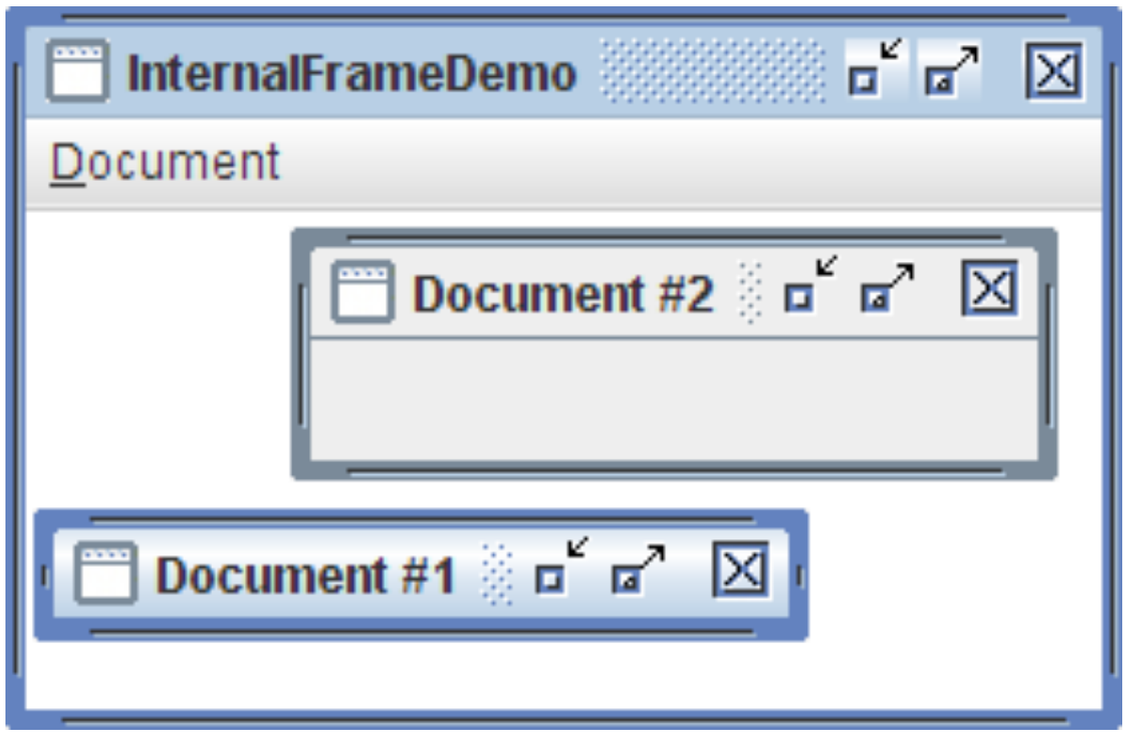
- `JRootPane`
  - on ne les crée jamais soi-même
    - les containers racine s'occupent d'en fournir
      - leur structure est très particulière…



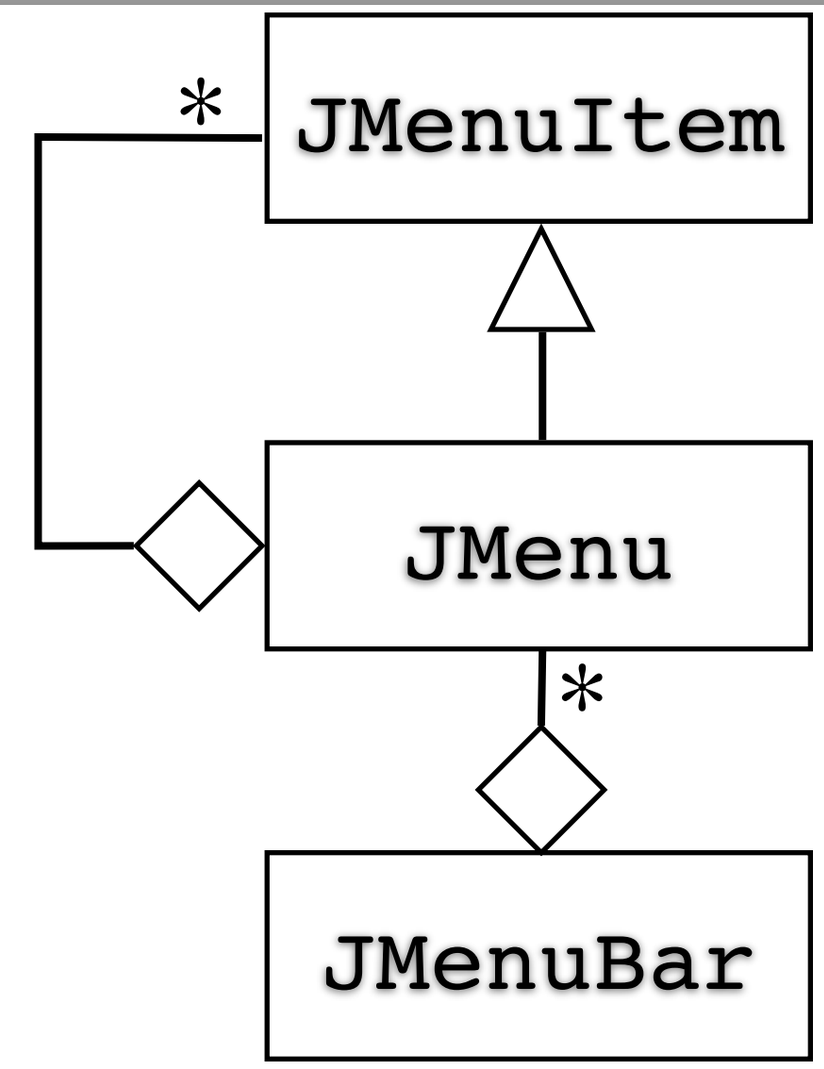
- `JLayeredPane`
  - autorise la superposition « en Z-stack » de différent panneaux à usages particuliers…



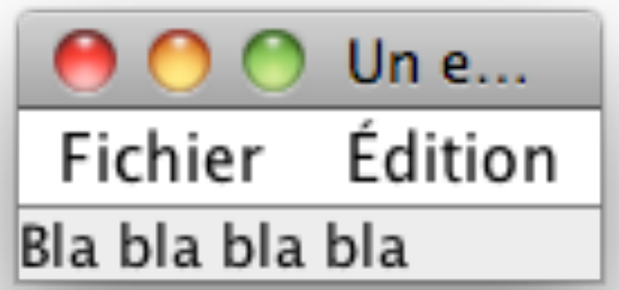
- `JInternalFrame`
  - autorise la création de frames internes, c'est-à-dire de fenêtres à l'intérieur d'une autre
- `JDesktopPane`
  - une version spéciale de `JInternalFrame` adaptée pour gérer des fenêtres internes multiples

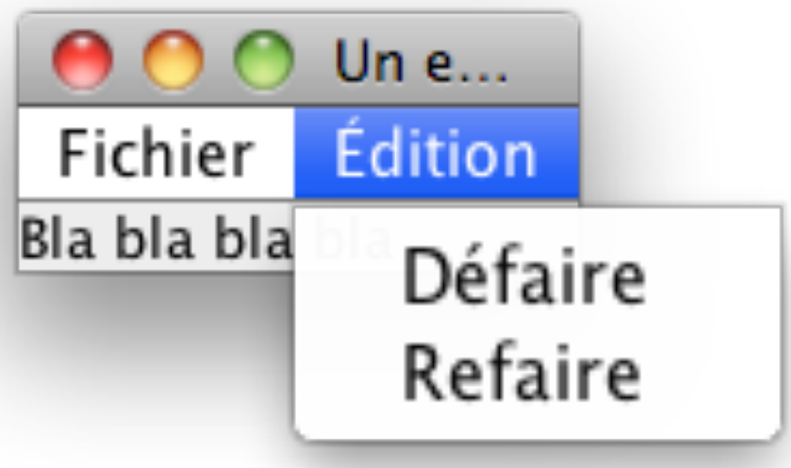


- Les menus
  - `JMenuItem`
    - un choix dans un menu
  - `JMenu`
    - un container dédié qui hérite de `JMenuItem`
  - `JMenuBar`
    - un container dédié à l'accueil de menus



- Attention
  - une seule barre de menu par container racine (`setJMenuBar(JMenuBar)`)
  - le placement de la barre n'est pas contrôlable (WindowManager dépendant)
    - Mac OS `java -Dapple.laf.useScreenMenuBar=true`
  - pas de layout modifiable





`JMenuExemple.java`

Containers : Menu

- Les séparation logiques entre groupes d'items peuvent être obtenues par utilisation de

  - `JSeparator`

`JSeparatorExemple.java`

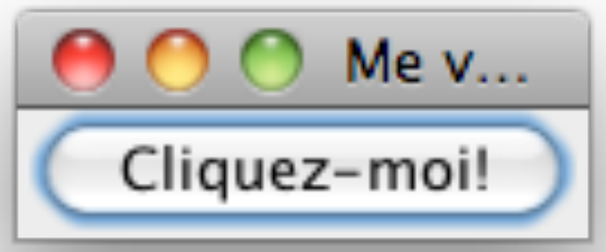
- Si le container est visible
  - `validate()` permet d'obtenir un replacement correct de tous les composants après ajout de nouveaux composants
- Container racine
  - `pack()` permet d'obtenir un rangement *optimal*

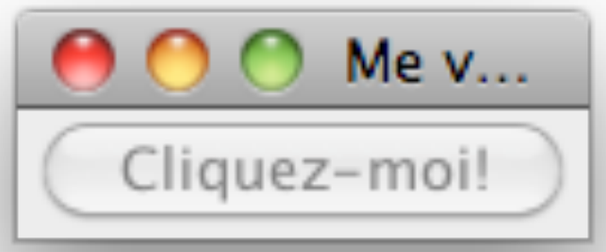
Composants

- Les Composants
  - proposent une interaction avec l'utilisateur
    - affichage
    - interaction en entrée
    - les deux combinés

Composants

- Les composants :
  - peuvent être actifs ou non, *i.e.* autorisent l'interactivité
    - `setEnabled(boolean)`
    - `boolean getEnabled()`
  - l'effet obtenu est en général un grisé

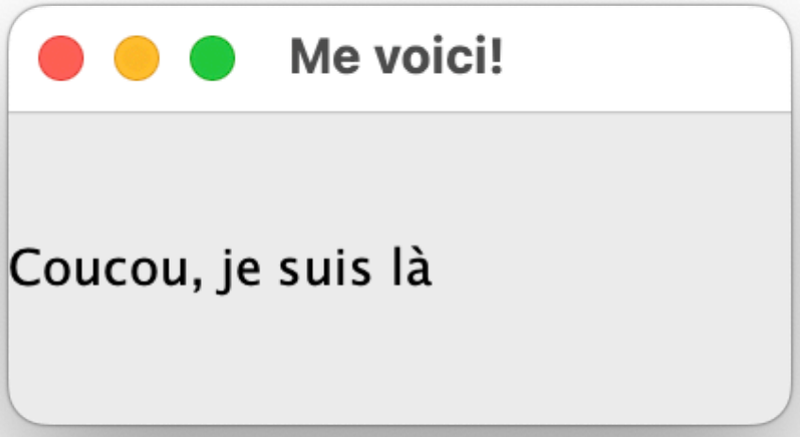


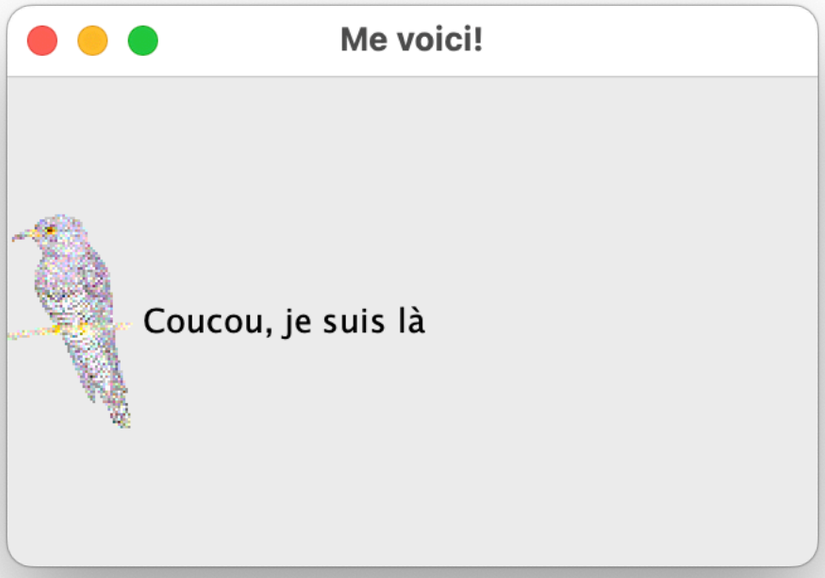


`JButtonExemple.java` `JButtonDisabledExemple.java`

- Composants simples non-interactifs
  - `JLabel`
  - `JProgressBar`
  - `JSeparator`
  - `JToolTip`
- fournissent une information à l'utilisateur

- `JLabel`
  - permet d'afficher une icône et/ou un texte
  - supporte un sous-ensemble de HTML 3.2

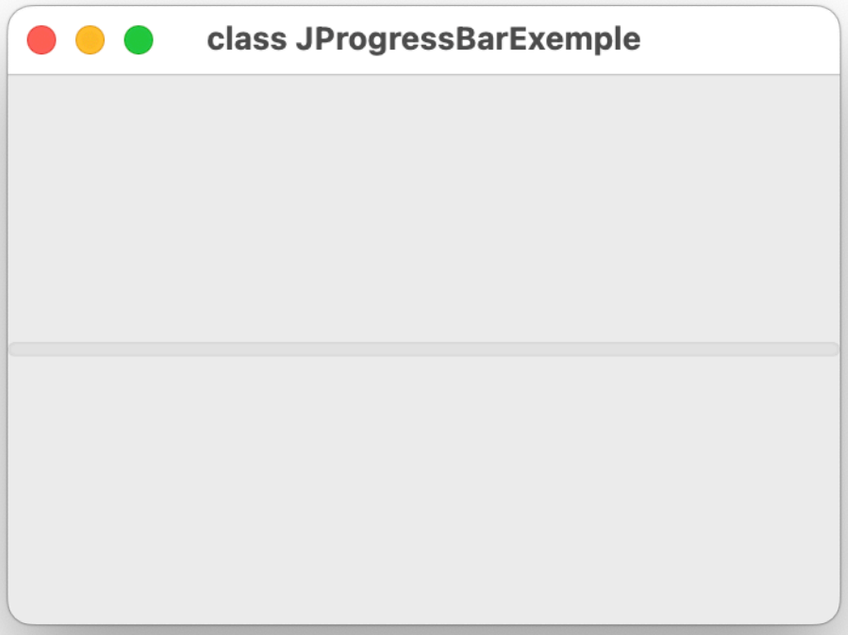


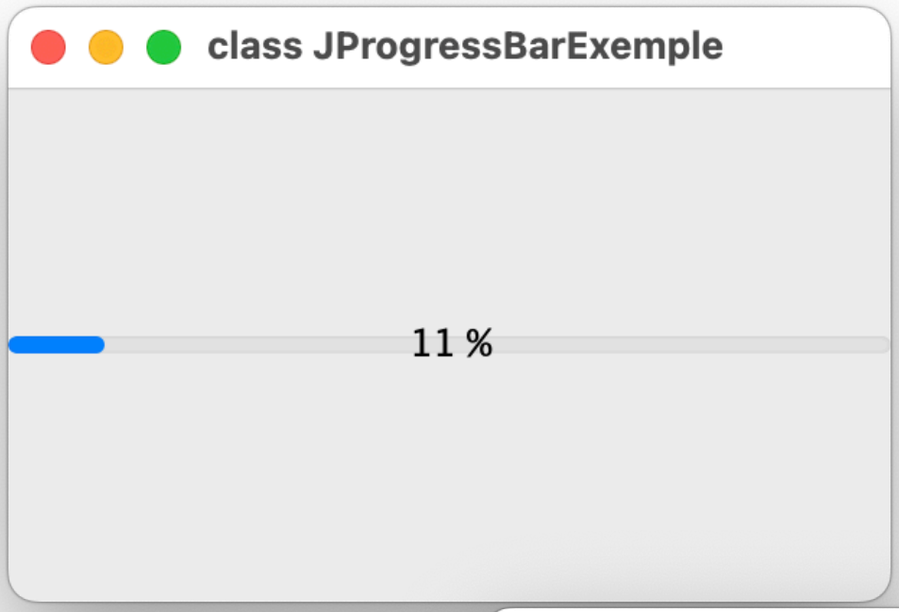


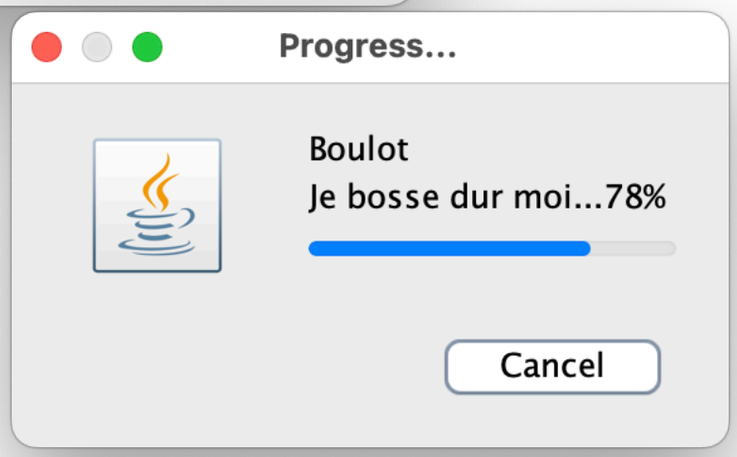
`JLabelExemple.java` `JLabelIconeExemple.java`

Composants

- `JProgressBar`
  - permet de rendre compte d'une progression
  - cousins : `ProgressMonitor` / `ProgressMonitorInputStream`





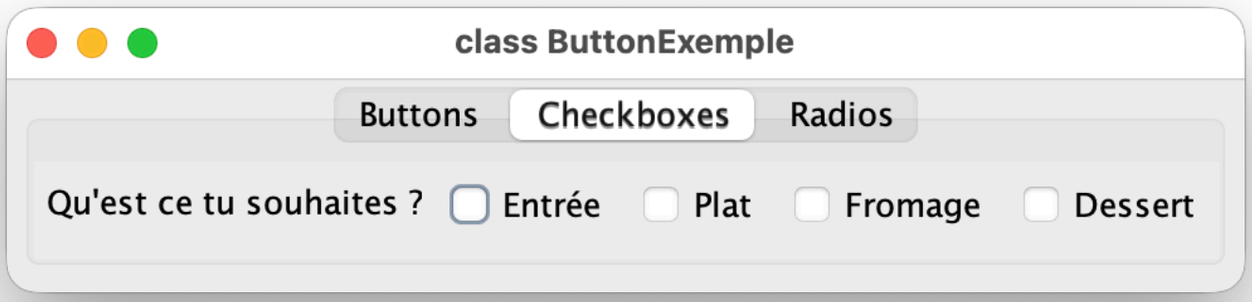


`JProgressBarExemple.java`
`ProgressMonitorExemple.java`

- Composants interactifs simples :
  - Boutons
    - `JButton`
    - `JMenuItem`
    - `JCheckBox JCheckBoxMenuItem`
    - `JRadioButton JRadioButtonMenuItem`
    - `JToggleButton`
  - Listes
    - `JComboBox`
    - `JList`
  - Texte
    - `JTextField JFormattedTextField/JPasswordField`
  - Divers
    - `JSlider`
    - `JSpinner`

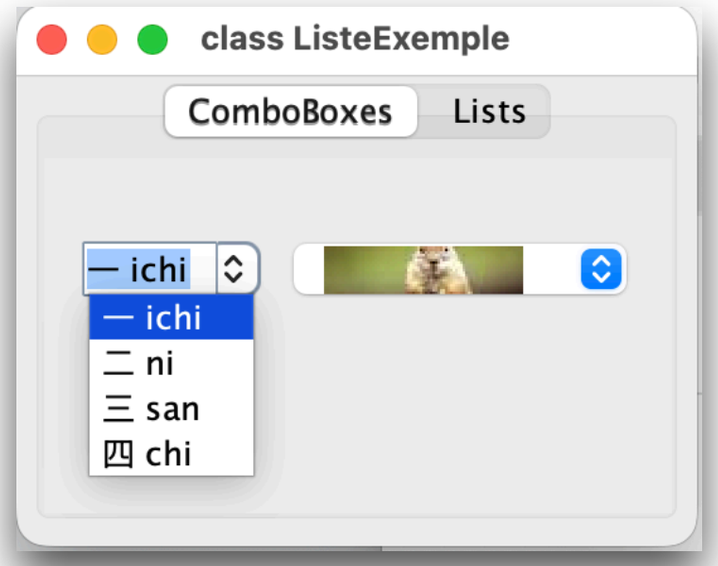
Boutons

- Les boutons (`AbstractButton`) peuvent être regroupés logiquement *via* des `ButtonGroup`
  - cela n'a vraiment de sens que pour les boutons qui ont un état de sélection
    - donc ni `JButton`, ni `JMenuItem`
      - le plus souvent utilisé avec les `RadioButton`
- Les `ButtonGroup` permettent de contrôler l'exclusion mutuelle lors de sélection



`ButtonExemple.java`

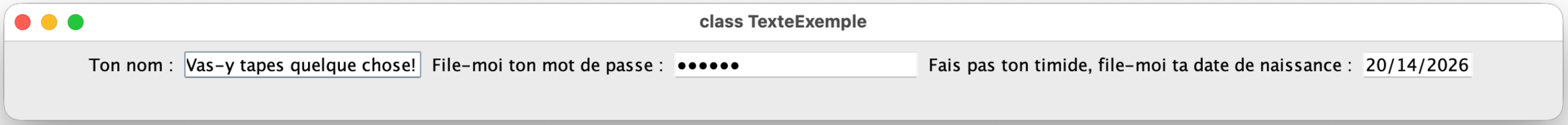
- Les listes
  - Les `JComboBoxs` peuvent être éditables
  - on verra plus tard comment personnaliser les rendus de ces objets… Les curieux peuvent aller jeter un œil sur `ListCellRenderer<E>`



`ListeExemple.java`

Texte

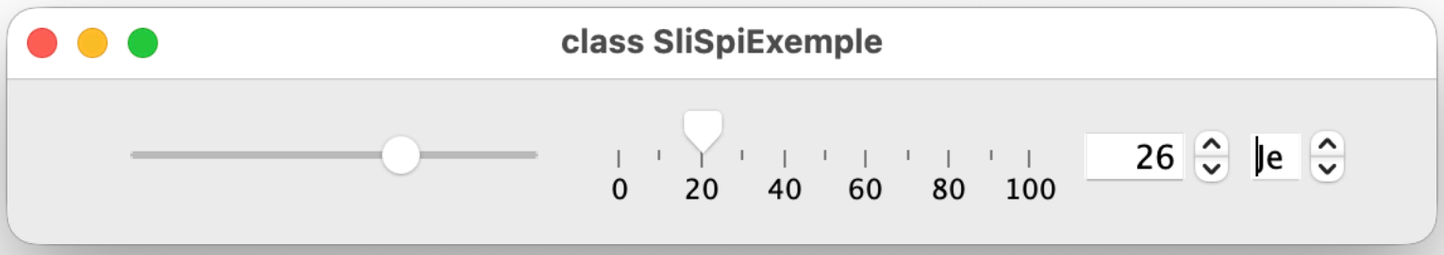
- Les champs de saisie de texte
  - ordinaire
  - mot de passe (saisie masquée)
  - champ formatés (date, etc)



TexteExemple.java

Divers

- Le curseur (slider)
  - un curseur sur une règle
  - les graduations peuvent être activées ou non (mineures/majeures)
- Le spinner
  - la *roulette* (attention son aspect graphique n'est pas celui généralement attendu…)



`SliSpiExemple.java`

- Composants avancés (complexes, *i.e.* plus d'une interaction) :
  - `JColorChooser`
  - `JEditorPane`
  - `JTextPane`
  - `JFileChooser`
  - `JTable`
  - `JTextArea`
  - `JTree`
- On les étudiera plus tard…

Layouts

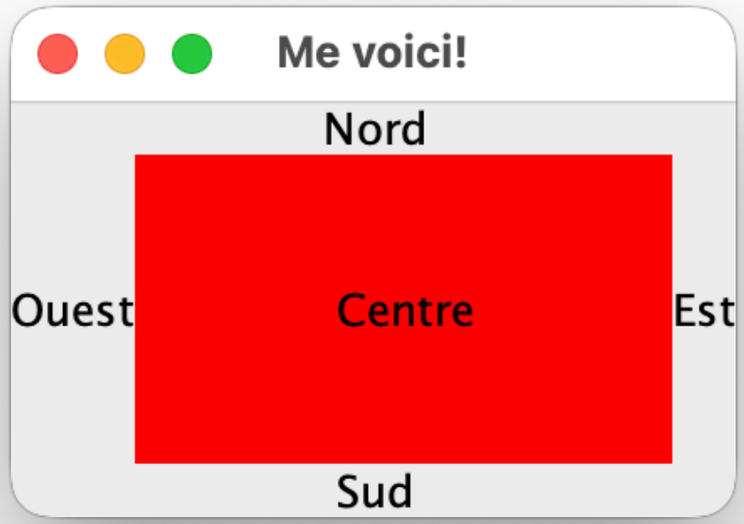
- Les Layouts
  - algorithmes de placement de composants dans des containers...
    - problème : ranger des bagages dans un coffre

Layouts

- `LayoutManager`
- on peut toujours essayer de ranger les éléments soi-même mais c'est généralement non-portable et parfaitement déconseillé…
- on peut choisir la politique de placement associée à un container donné
- méthode `setLayout(LayoutManager)`

- Layouts de base
  - AWT
    - `BorderLayout`
    - `CardLayout`
    - `FlowLayout`
    - `GridLayout`
    - `GridBagLayout`
  - Swing
    - `BoxLayout`
    - `GroupLayout`
    - `OverlayLayout`
    - `ScrollPaneLayout`
    - `SpringLayout`
    - `ViewportLayout`

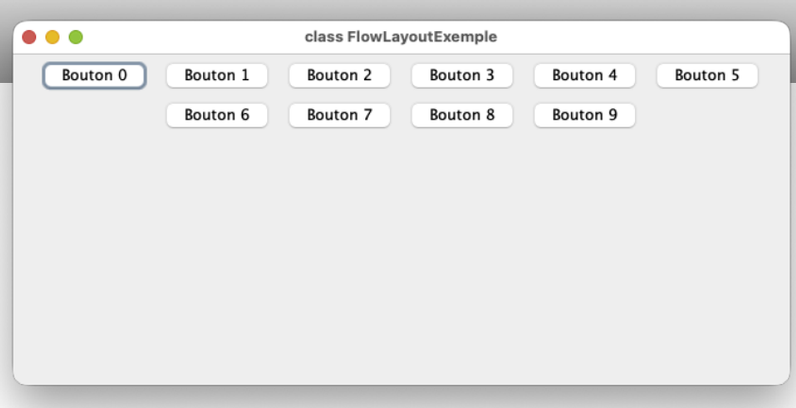
Layouts

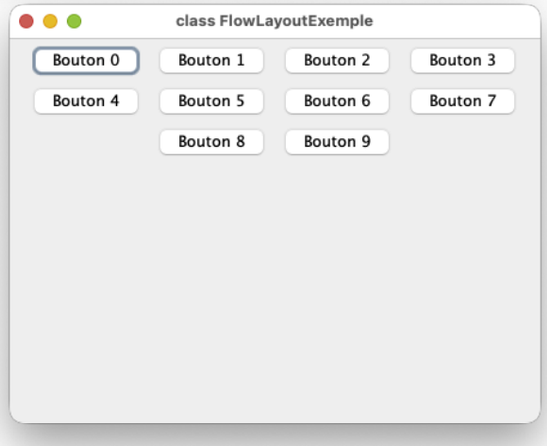


- `BorderLayout`
  - par défaut dans :
    - les `JRootPane` des `JWindow`
    - les `contentPane` des `JFrame`
      - Cinq composants au plus : nord, sud, est, ouest, centre
      - conserve dynamiquement son aspect
        - retaille les composants si nécessaire

`BorderLayoutExemple.java`

Layouts





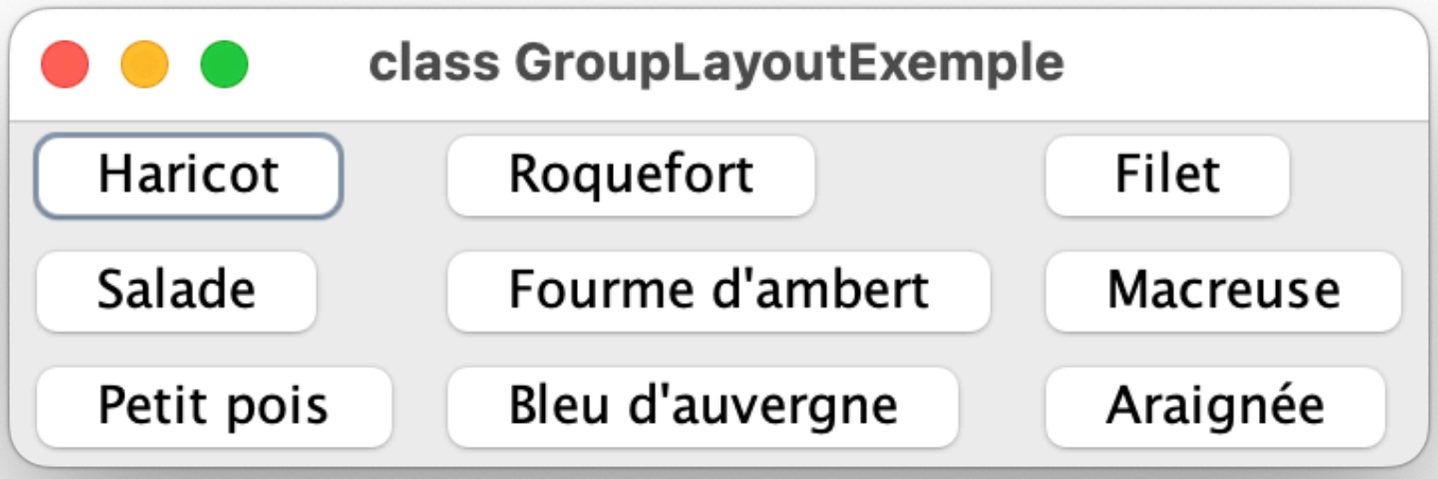
- `FlowLayout`
  - par défaut dans les panneaux
  - les composants sont placés comme dans un flot d'écriture
  - ne retaille pas les composants

`FlowLayoutExemple.java`

- `BoxLayout`
  - par défaut dans les `Box`
  - les composants sont rangés horizontalement ou verticalement dans des espaces tous de même taille
  - ne retaille pas les composants

`BoxLayoutExemple.java`

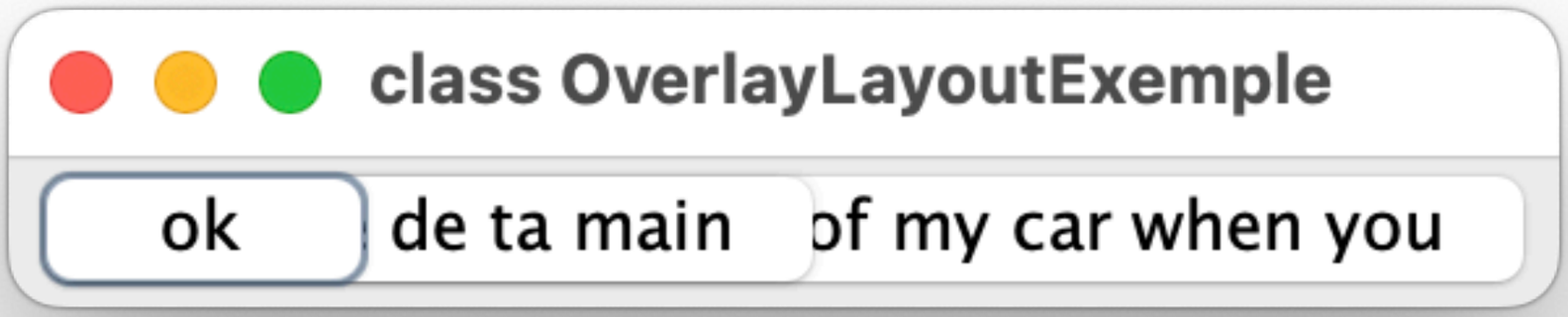
- `GroupLayout`
  - permet d'obtenir des effets d'alignement
  - ne retaille pas les composants
  - un poil complexe à utiliser



`GroupLayoutExemple.java`

Layouts

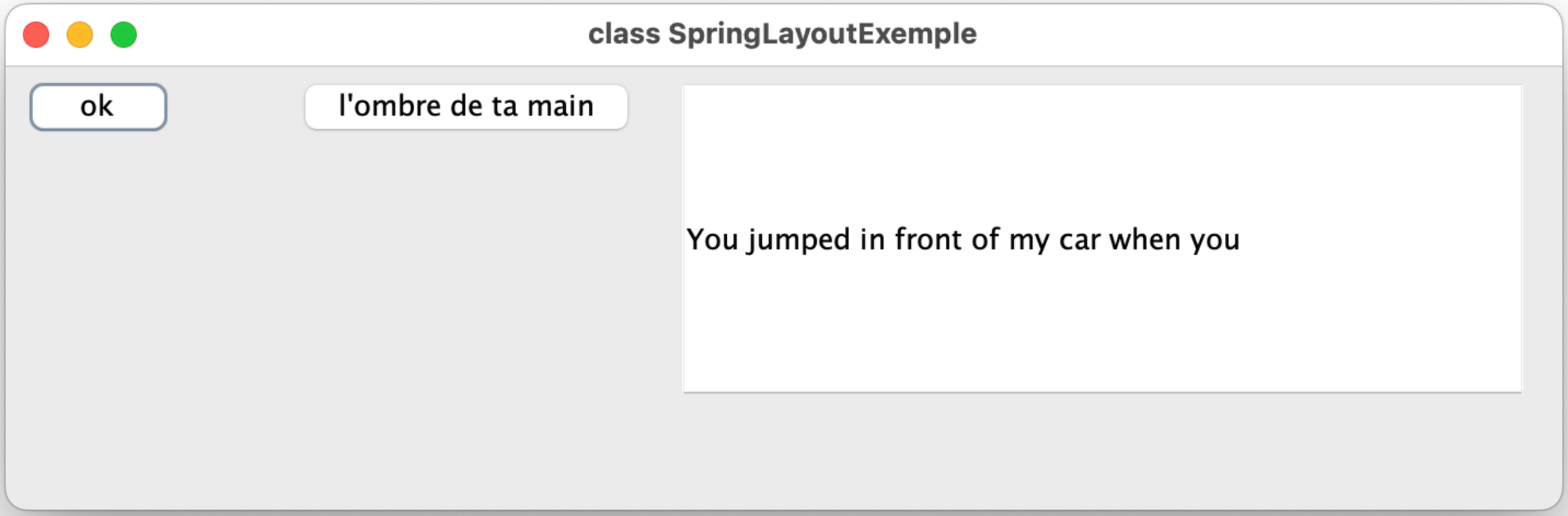
- `OverlayLayout`
  - superpose des composants comme `CardLayout`
  - mais autorise la visualisation/manipulation par transparence…



`OverlayLayoutExemple.java`

Layouts

- `SpringLayout`
  - exprime des contraintes entre composants
  - simple en apparence…



`SpringLayoutExemple.java`

Internationalisation

- Régionalisation (*Localization*) l10n
  - adaptation d'une application aux caractéristiques culturelles locales
  - travail de l'ordre de la **traduction**

- Internationalisation (*Internationalization*) i18n
  - processus de développement conduisant à produire une application localisable
  - travail spécifique de **développement**

- Idée (simple)
  - ne pas afficher un message en dur
  - utiliser une fonction réalisant la traduction adéquate en fonction d'un environnement donné

- `java.util.ResourceBundle`
  - une classe d'encapsulation de données régionalisées, la sélection des données est opérée par un `ResourceBundle.Control`
  - des méthodes pour obtenir les messages depuis une liste (une classe adéquate ou un fichier adéquat)

- Récupérer des données régionalisées depuis un fichier
  - `ResourceBundle.getBundle(String name)`
    - le fichier recherché aura pour nom
      - `name` + *suffixe régionalisé* + "`.properties`"
        - ex : `Messages_fr_FR.properties`
          - possibilité d'internaliser le Bundle...

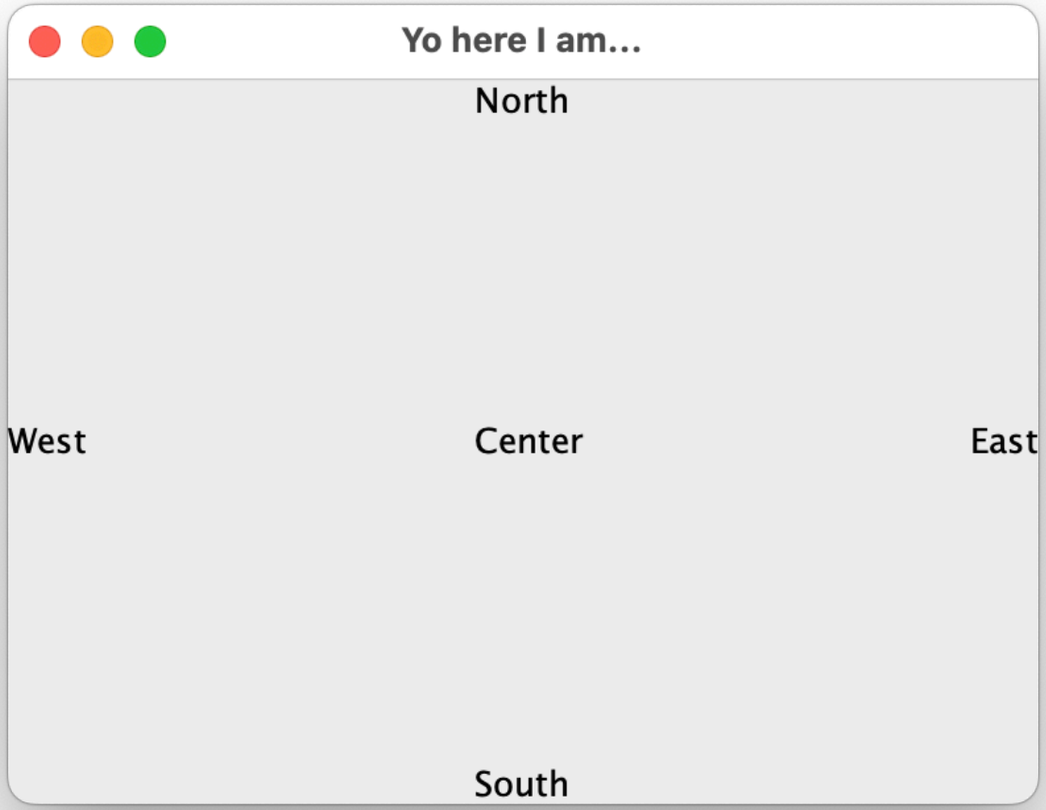
- Récupérer la traduction d'un terme
  - `unResourceBundle.getString(String clé)`
    - permet de récupérer la chaîne associée à la clé donnée, et ce dans la base précédemment sélectionnée

Internationalisation

- Internationalisation
  - MacOSX
    - préférences système
  - Windows
  - Unix
    - environnement : `LANG`, `LC_*`
- dans Eclipse : sélection possible dans
  `Run Configurations...` ➧ `Environment`

`InternationalisationExemple.java`

- Internationalisation



`InternationalisationExemple.java`

Mnémoniques

- Les menus et la navigation au clavier
  - on peut associer aux menus, items et boutons un **mnémonique**
    - une lettre (associée à un modificateur en général <ALT>) permettant de se placer (naviguer) et sélectionner l'objet associé
      - attention, l'action associée n'est pas réalisée (<RET>)
- ce n'est pas un raccourci!
  - un **raccourci** permet de déclencher l'action associée à un objet d'interface sans passer par l'objet
  - c'est de l'accessibilité
- La RFC 1345 est dédiée au sujet « Character Mnemonics & Character Sets »

Mnémoniques

- pour associer un mnémonique
  - `setMnemonic(int)`
  - l'entier est normalement l'identité d'une touche du clavier
    - `java.awt.event.KeyEvent.VK_*`
      - si le caractère est présent dans le texte affiché par l'objet associé, ce caractère est souligné par l'interface
- Attention: macOS ne supporte pas les mnémoniques Swing

`MnemmoniquesExemple.java`

Raccourcis

- un **raccourci** permet de déclencher l'action associée à un objet d'interface sans passer par l'objet
  - c'est de l'ergonomie
    - `setAccelerator(KeyStroke)`
- pour l'observer, il faut être capable d'associer des actions à des objets d'interface (très bientôt traité…)