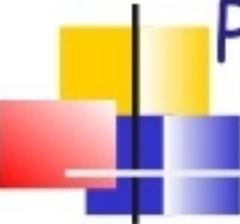


## Chapitre 5 : Compléments

---

### Interfaces Graphiques



# Plan

---

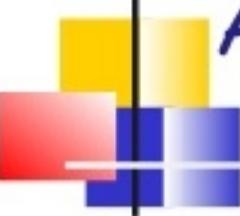
- A-Les Interfaces Graphiques
  - 1-Généralités,
  - 2-Ruby & GUI
  - 3-Tk
  - 4-FoxRuby
  - 5-Gtk+
- B-Exemple
- C-Compléments



## A1-Généralités

---

- la facilité d'utilisation des applications a pris son importance dans les années 90.
  - On ne peut plus écrire d'application sans interface homme-machine graphique.
  - Le confort d'exécution augmente avec l'utilisation de la souris, la sélection de champs dans des listes qui évite bien des erreurs de frappe, l'utilisation de boutons, etc.
    - Ruby est un excellent outil pour écrire des scripts d'administration système par exemple.
    - Il est aussi adapté à l'écriture d'applications complètes plus sophistiquées à destination d'utilisateurs "réels".



## A1-Généralités

---

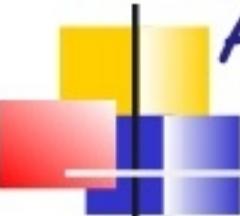
- Un des bénéfices de Ruby est qu'il permet un développement rapide
  - Pas de phase classique *codage-compilation-test*, toujours consommatrice de temps
  - Les changements sont faciles à effectuer permettant de tester rapidement de nouvelles idées.
  - Ces bénéfices deviennent encore plus évident lors du développement d'interface graphique sous Ruby



## A1-Définitions

---

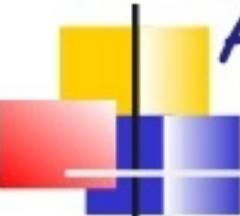
- Widget
  - Une interface graphique est composée de **widgets** (éléments graphiques) placés dans des conteneurs, qui indiquent comment placer ces widgets à l'écran (un conteneur est aussi un widget).
- Signaux
  - Chaque widget dispose d'une panoplie de **signaux** auquel il peut réagir. Par exemple, un bouton peut réagir au signal "clicked", lorsque l'utilisateur a cliqué sur ce bouton



## A1-Généralités

---

- Un programme classique à un déroulement ~ séquentiel.
  - Une application à interface graphique est “dirigée par les évènements”.
    - Elle passe son temps à attendre. (boucle évènementielle)
  - Chaque toolkit
    - Définit ses propres modes de gestion des évènements
    - Utilise un grand nombre de composants graphiques
    - Utilise une représentation hiérarchique des GUI
    - Offre plusieurs gestionnaires de placement des composants. (Layout Manager)



## A2-Ruby & GUI

---

- Plusieurs Toolkits “graphique” sont utilisable “facilement” avec Ruby
  - TK à l'origine conçu pour TCL
    - Très simple à utiliser, pas très joli, et pas très objet
  - FxRuby, une version pour Ruby de FOX
    - Plus compliqué mais offrant aussi plus de possibilités
    - Approche objet
  - Gtk+, la seconde version de la toolkit graphique issu de GIMP.
    - Approche Objet
    - Simple à utiliser
    - Possibilités des interfaces modernes.

# A2-Ruby & GUI



Tk

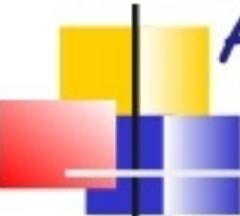


FxRuby



Gtk+

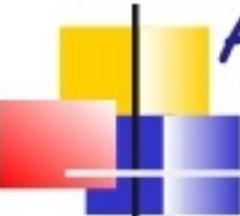
- TK est né dans le milieu des années 80 comme “couche interface” du langage de script TCL développé par John Ousterhout.
  - Il a été depuis adopté comme plateforme d'interfaçage graphique par les langages de scripts les plus populaires (dont Perl & Python).
  - Cependant l'ensemble des composants graphiques proposé par TK est limité comparé aux plus modernes GUI.
  - Il a l'avantage d'être particulièrement bien supporté sous MacOs (au contraire d'autres)



## A3-TK-Les Widgets

---

- Les différents widgets dans Tk
  - **label**, afficher du texte ou des images
  - **button**, afficher des images, déclencher une action
  - **checkbutton**, similaire au button mais qui reste dans un état enfoncé ou non.
  - **radiobutton**, similaire au checkbutton , mais dont un seul est enfoncé à la fois.
  - **listbox**, permettant une sélection unique ou multiple.
  - **entry**, pour créer des zones de saisie .
  - **text**, permettant d'afficher et d'éditer un texte complet, avec possibilité d'afficher différentes fontes et couleurs, avec également possibilité d'hypertexte.

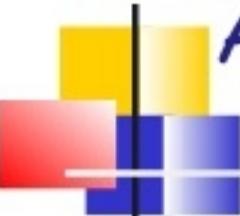


## A3-TK-Les Widgets

---

- Les différents widgets dans Tk

- **canvas**, pour dessiner , lignes , cercles, ellipse etc, on peut également mélanger au dessins d'autres widgets, le canvas est très puissant et permet de l'utiliser pour créer ses propres widgets.
- **frame**, pour regrouper un ensemble de widgets dans des 'boites' au relief divers.
- **scrollbar**, pouvant s'appliquer au listbox, text, entry ,canvas
- **scale**, pour créer des curseurs ou des barres de progression
- **message**, semblable au label , mais avec un contrôle plus fin de l'aspect multiligne
- **menu**, déroulants , cascade , et popup.



## A3-TK-Principe

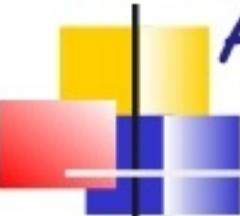
---

- Pour chaque cas le principe est le même, on crée le widget en lui précisant les options voulues, puis on le 'place' en utilisant un des trois gestionnaire de placement :
  - pack, place ou grid.
    - Une fois créé , on peut récupérer/modifier les caractéristiques d'un widget
    - En utilisant ainsi les widgets comme des briques élémentaires d'un jeu de lego, il est très facile et très rapide de créer des interfaces graphiques complètes.

# A3-TK-Principe

- Exemple la réalisation de la calculatrice en Tk





## A4-FxRuby

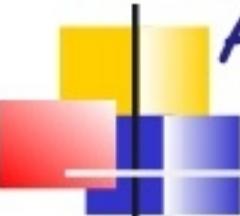
---

- **FxRuby** est un module d'extension de Ruby permettant de développer des interfaces graphiques en utilisant FOX.
  - Cette extension est développée par Lyle Johnson (<http://fxruby.sourceforge.net>).
- **FOX** pour “Free Objects for X” est une bibliothèque graphique multiplateforme (Jeroen van der Zijp, <http://www.fox-toolkit.org/>).
  - Par rapport à TK et GTK+, FxRuby est un petit nouveau mais se répand rapidement auprès des développeurs (C/C++, Python, Ruby, Eiffel,...).

## A4-FxRuby

- Exemple la réalisation de la calculatrice en FxRuby

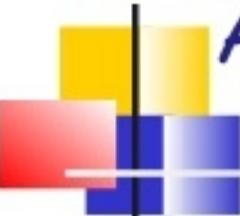




## A5- GTK+

---

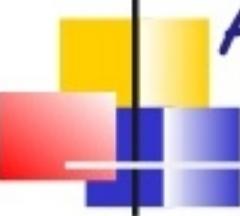
- GTK+ est une bibliothèque permettant de créer des interfaces graphiques (GUI) très facilement.
  - Au départ, GTK+ a été développé pour les besoins du logiciel de traitement d'images GIMP
  - Maintenant plein d'autres applications dont l'environnement GNOME (GNU Network Object Model Environment) sont basées sur GTK+.
- Documentations et tutoriels
  - <http://www.gtk-fr.org/> (pour le C)
  - <http://ruby-gnome2.sourceforge.jp/hiki.cgi?Ruby/GTK>



## A5- GTK+

---

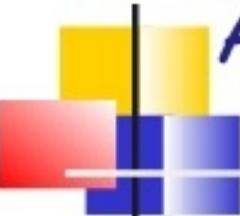
- Caractéristiques intéressante de GTK+ :
  - GTK+ est sous licence GNU LGPL. Cela fait de GTK+ une bibliothèque libre, (Cf. Site GNU)
  - GTK+ existe sur plusieurs plates-formes.
    - UNIX-like, Windows, BeOs;
  - Plusieurs langages de programmation.
    - Même si les créateurs de GTK+ ont écrits cette bibliothèque en C, sa structure orientée objets et sa licence ont permis à d'autres développeurs d'adapter GTK+ à leur langage préféré.
    - il est maintenant possible de programmer des GUI GTK+ en C, C++, Ruby, Ada, Perl, Python, PHP, etc;



## A5- GTK+

---

- Les objets graphiques de GTK+ sont appelés des widgets.
  - Un widget est en fait une structure définissant les propriétés d'un objet associée à une large panoplie de fonctions permettant de manipuler ces objets.
  - Bien que GTK+ soit écrit en C, il introduit la notion d'héritage et les widgets de GTK+ suivent une hiérarchie bien définie. Ainsi tous les objets graphiques héritent des propriétés et des fonctions d'un widget de base qui s'appelle **GtkWidget**.

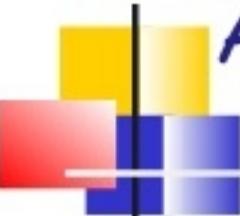


## A51- Les Fenêtres GTK+

---

- Les fenêtres
  - Le premier élément d'une interface graphique est bien entendu la fenêtre.
- Créer et afficher une fenêtre.
  - Le widget permettant d'afficher une fenêtre se nomme `GtkWindow`. Il a bien sûr ses propres méthodes, mais grâce à l'héritage il bénéficie aussi des méthodes de plusieurs autres widgets.

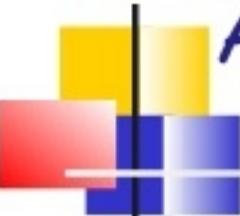
```
 GObject
  ↘ GtkObject
    ↘ GtkWidget
      ↘ GtkContainer
        ↘ GtkBin
          ↘ GtkWindow
```



## A51- Les Fenêtres GTK+

---

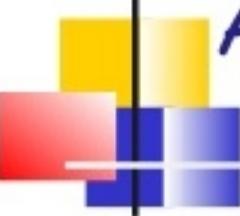
- **Création d'une fenêtre**
  - `monApplication = Gtk::Window.new`
- **Affichage de la fenêtre.**
  - Un widget n'a pas de fonction spécifique liée à son affichage, mais utilise une fonction de `GtkWidget` qui est `show` ou `show_all`
    - `monApplication.show`
  - **Ceci est insuffisant pour voir la fenêtre.**
    - La raison est simple : la destruction de la fenêtre a lieu tout de suite après son affichage.
    - Il faut utiliser les "signaux" afin que l'application ne se termine qu'au moment où l'utilisateur le demande.



## A52- Les Signaux GTK+

---

- Les signaux.
  - lorsque l'utilisateur interagit avec l'application (clique sur un bouton, ferme une fenêtre, ...), le widget concerné émet un signal (par exemple "destroy" lorsque l'on ferme une fenêtre).
    - Chaque widget est associé à un ou plusieurs signaux et permet donc ainsi de programmer toutes les actions possibles.
  - Ce signal va ensuite être intercepté par une boucle évènementielle qui va vérifier qu'une action spécifique à ce signal et ce widget a bien été définie. Si tel est le cas, la fonction associée sera exécutée.

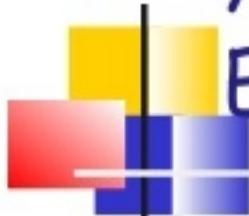


## A52- Les Signaux GTK+

---

- fonction callback.
  - Il faut créer une fonction callback pour chacun des évènements susceptible de survenir pendant l'exécution du programme et associer (ou connecter) cette fonction à un signal.

```
def onDestroy
    puts "Fin de l'application"
    Gtk.main_quit
end
.....
# La fenêtre est détruite il faut quitter
monApplication.signal_connect('destroy')
{onDestroy}
```



## A52- Les Signaux GTK+

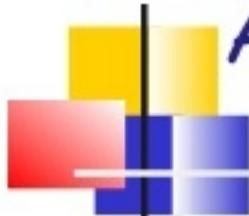
### Exemple1

- Le code Complet

```
require 'gtk2'
Gtk.init
def onDestroy
    puts "Fin de l'application"
    Gtk.main_quit
end

monApplication = Gtk::Window.new
monApplication.show
# la fenêtre est détruite il faut quitter
monApplication.signal_connect('destroy')
{onDestroy}
Gtk.main
```

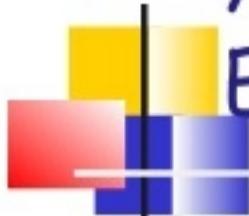




## A53- Créer une application Ruby/GTK+

---

- Inclure le module gtk2 à votre application :
  - require 'rubygems' (ou pas)
  - require "gtk2" # Attention respecter la casse
- Initialiser GTK+ avec cette méthode :
  - Gtk.init # Sans cela il ne se passe rien
- Créer l'interface proprement dite et l'afficher
  - monAppli=.....
- Attendre les évènements
  - => Lancement de la boucle évènementielle
  - Gtk.main()
- Terminer l'application
  - => Arrêt de la boucle évènementielle
  - Gtk.main\_quit()



## A54- Personalisation Exemple2

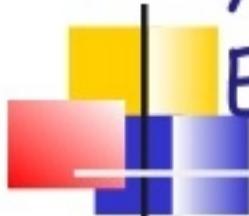


### ■ Personaliser la fenêtre

```
# Titre de la fenêtre
monApp.set_title("Ma Fenêtre en GTK")
# Taille de la fenêtre
monApp.set_default_size(300,100)
# Réglage de la bordure
monApp.border_width=5
# On peut la redimensionner
monApp.set_resizable(true)
# L'application est toujours centrée
monApp.set_window_position(Gtk::Window::POS_CENTER_ALWAYS)
```

### ■ + Beaucoup d'autres possibilités

- <http://ruby-gnome2.sourceforge.jp/hiki.cgi?Ruby/GTK>



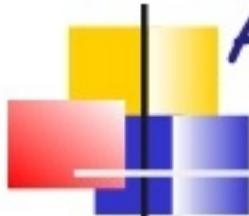
## A55- Les Labels GTK+ Exemple3



- **Les labels**

- Maintenant que nous savons créer une fenêtre, nous allons créer l'incontournable "Hello World!".
  - Pour cela, nous allons utiliser le widget `GtkLabel` qui nous permet d'afficher du texte.
- **Création et affichage d'un label.**

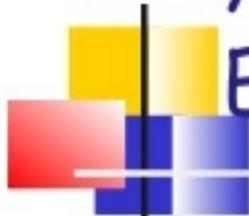
```
# Création du Label  
monLabel=Gtk::Label.new("HelloWorld")  
monApp.add(monLabel)
```



## A55- Les Labels GTK+

---

- Pour afficher du texte, Gtk utilise la librairie Pango qui s'occupe du rendu et de l'affichage du texte.
  - Pango permet l'internationalisation des applications. Pour cela Pango utilise l'encodage UTF8. Ce charset est codée sur 16 bits, ce qui offre la possibilité d'afficher plus 65000 caractères, permettant ainsi d'afficher des caractères accentués et bien plus (ex : pour le grec, le chinois, ...).
  - Il faut donc un éditeur qui puisse travailler en UTF8 pour manipuler les caractères accentués.



## A55- Les Labels GTK+ Exemple4



- Formatage du texte
  - Nous allons maintenant voir comment modifier l'apparence de notre texte (police, couleur, ...).
    - Pour notre application test, nous allons afficher une ligne en *Courier gras taille 10*, une ligne en *Times New Roman italique bleu taille 12*, et une ligne en *Verdana souligné taille 16*, le tout centré
- Alignement du texte
  - Méthode `set_justify(alignment)`
    - `Gtk::JUSTIFY_LEFT` aligner le texte à gauche (par défaut)
    - `Gtk::JUSTIFY_RIGHT` pour aligner le texte à droite ; ·
    - `Gtk::JUSTIFY_CENTER` pour centrer le texte ; ·
    - `Gtk::JUSTIFY_FILL` pour justifier le texte.

# A55- Les Labels GTK+ Exemple4



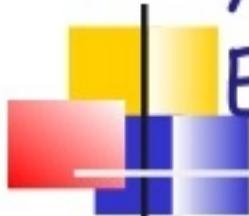
## ■ Définition du format

- Pour définir le format du texte, il suffit d'insérer des balises à l'intérieur même de notre texte. Pango s'occupe ensuite de rechercher les balises puis de formater le texte à notre convenance.

### # Création du Label

```
texte= "<span font_desc=\"Courier New bold 10\">Courier New 10 Gras</span>\n"
texte += "<span font_desc=\"Times New Roman italic 12\" foreground=\"#0000FF\">Times
New Roman 12 Italique</span>\n"
texte += "<span font_desc=\"Comic Sans MS 16\" foreground=\"#0000FF\">Comic Sans MS
16</span>\n"
texte += "<span font_desc=\"Verdana 20\"><u>Verdana 20 Souligne</u></span>"
```

```
monLabel=Gtk::Label.new()
monLabel.set_markup(texte)
monLabel.set_justify(Gtk::JUSTIFY_CENTER)
```



## A56- Les Boutons GTK+ Exemple5



- **Les boutons**

- Un élément essentiel d'une interface graphique.
  - Celui-ci permet à l'utilisateur d'effectuer une action grâce à un simple click de souris. GTK+ permet de les utiliser grâce au widget `GtkButton`

### *# Crédit d'un Bouton*

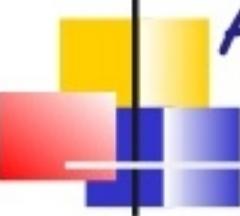
```
monBout=Gtk::Button.new("Mon Bouton")
monApp.add(monLabel)
```

- `Gtk-WARNING **: Attempting to add a widget with type GtkLabel to a container of type GtkWindow, but the widget is already inside a container of type GtkWindow, the GTK+ FAQ at http://www.gtk.org/faq/ explains how to reparent a widget.`
- Cela ne marche pas dans une fenêtre on ne peut mettre qu'un Widget !!!!! Alors ça c'est nul ??????
  - Bien sûr que non

- Object
  - GLib::Instantiatable
    - GLib::Object?
    - Gtk::Object
  - Gtk::Widget
  - Gtk::Container?
  - Gtk::Bin
  - Gtk::Window

## A57- Layout Manager GTK+

- En fait, un widget de type GtkContainer comme GtkWindow ne peut contenir qu'un seul widget.
  - Pour placer plusieurs widgets dans une interface il faut utiliser un gestionnaire de placement (LM).
    - Chacun d'entre eux propose un algorithme de positionnement spécifique
    - Il définit la taille et la position de chaque composant graphique lors de son insertion.
- Pour définir l'apparence de l'interface graphique, on utilise une structure hiérarchique de gestionnaire de placement.



## A57- Layout Manager GTK+

---

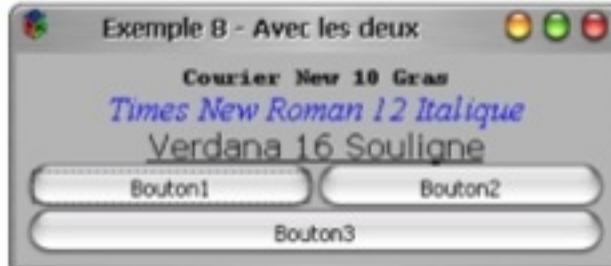
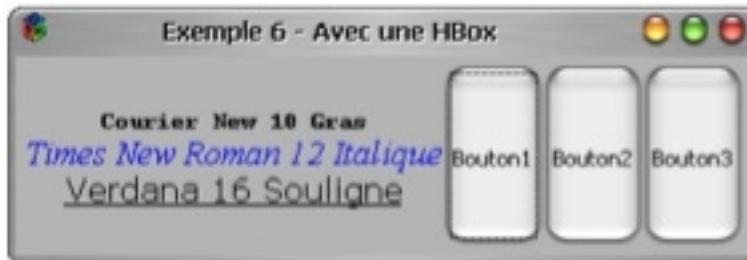
- Sous Gtk il existe plusieurs Layout Managers
  - La `GtkBox` qui permet d'inclure plusieurs widgets à l'intérieur.
    - Il existe deux sous-classes de `GtkBox` :
      - les `GtkHBox` qui permettent de disposer les widgets horizontalement ;
      - les `GtkVBox` pour les disposer verticalement.
  - La `GtkTable`, un `GtkContainer` qui organise les widgets dans une grille (ligne, colonne)

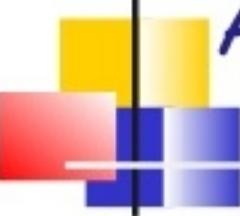
# A57- Layout Manager GTK+

## Exemples 6 à 8

- Exemple

- Obtenir une fenêtre comportant notre label et 3 boutons





## A57- Layout Manager GTK+

---

- Avec le widget `GtkTable`, la première chose à faire sera de choisir le nombre de lignes et de colonnes sur la grille.
- Ensuite il suffit de placer nos widgets sur la grille, en spécifiant en premier les points de départ et d'arrivée suivant les colonnes puis suivant les lignes :
  - On ne donne pas la longueur et la largeur du widget, mais sa position finale sur la grille.
  - Cela devient nettement plus facile et naturel de positionner les widgets sur une grille quand leur nombre est important

# A57- Layout Manager GTK+ Exemple9



# Creation du Layout

```
frame=Gtk::Table.new(2,2,true)
```

```
monApp.add(frame)
```

```
.....
```

# Creation des Boutons

```
monBout1=Gtk::Button.new("Bouton1")
```

```
monBout2=Gtk::Button.new("Bouton2")
```

```
monBout3=Gtk::Button.new("Bouton3")
```

# On ajoute a la Table les widgets

```
frame.attach(monLabel, 0,1,0,2)
```

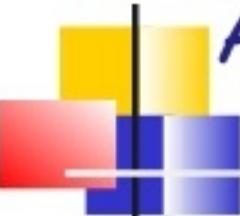
```
frame.attach(monBout1, 1,2,0,1)
```

```
frame.attach(monBout2, 1,2,1,2)
```

```
frame.attach(monBout3, 2,3,0,2)
```



Ca c'est si on  
avait mis false



## A58- Les Boutons (2) GTK+

---

- Retour sur les boutons

- Les constructeurs

- **Gtk::Button::new**
      - Crée et retourne un bouton.
    - **Gtk::Button::new(text, use\_underline = true)**
      - Crée et retourne un Bouton avec un "Child" **Gtk::Label** contenant le texte.
      - **use\_underline**: Permet de spécifier en le faisant précéder d'un \_ le caractère "Accélérateur"
    - **Gtk::Button::new(stock\_id)**
      - Crée et retourne un Bouton contenant l'image et le texte spécifié par le "**stock\_id**" (constantes définies dans le module **Gtk::Stock**).
      - Par Exemple **Gtk::Stock::OK** ou **Gtk::Stock::APPLY**.

# A58- Les Boutons (2) GTK+

## Exemple 10

# Création des Boutons

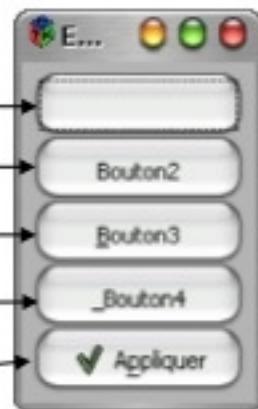
monBout1=Gtk::Button.new()

monBout2=Gtk::Button.new("Bouton2")

monBout3=Gtk::Button.new("\_Bouton3", true)

monBout4=Gtk::Button.new("\_Bouton4", false)

monBout5=Gtk::Button.new(Gtk::Stock::APPLY)



### ■ Les méthodes d'instances

- <http://ruby-gnome2.sourceforge.jp/hiki.cgi?Gtk::Button>

# A58- Les Boutons (2) GTK+

## Exemple 11



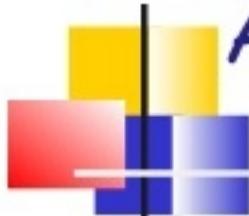
- Les Signaux sur les boutons
  - activate
  - clicked
  - enter
  - leave
  - pressed
  - released

```
def onEvt(label,message)
    puts message
    label.set_text(message)
end
.....
monBout.signal_connect('activate')
    (onEvt(monLabel,"Evenement ACTIVATE"))
monBout.signal_connect('clicked')
    (onEvt(monLabel,"Evenement CLICKED"))
monBout.signal_connect('enter')
    (onEvt(monLabel,"Evenement ENTER"))
monBout.signal_connect('leave')
    (onEvt(monLabel,"Evenement LEAVE"))
monBout.signal_connect('pressed')
    (onEvt(monLabel,"Evenement PRESSED"))
monBout.signal_connect('released')
    (onEvt(monLabel,"Evenement RELEASED"))
```

## A59- Champs de saisie GTK+ Exemple 11



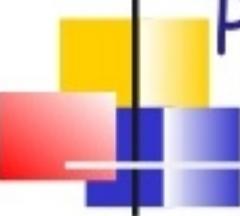
- Le widget `GtkEntry` permet de saisir des données.
  - Il définit une zone de texte (une ligne) dans lequel l'utilisateur peut taper du texte ou dans lequel le programme peut afficher une information.
- Créditation et utilisation d'un `GtkEntry`
  - `maSaisie=Gtk::Entry.new()`
  - <http://ruby-gnome2.sourceforge.jp/hiki.cgi?Gtk::Entry>



## A5- GTK+

---

- Pour les autres Widget, c'est pareil
  - Consulter la doc en ligne
    - <http://ruby-gnome2.sourceforge.jp/hiki.cgi?Ruby/GTK>

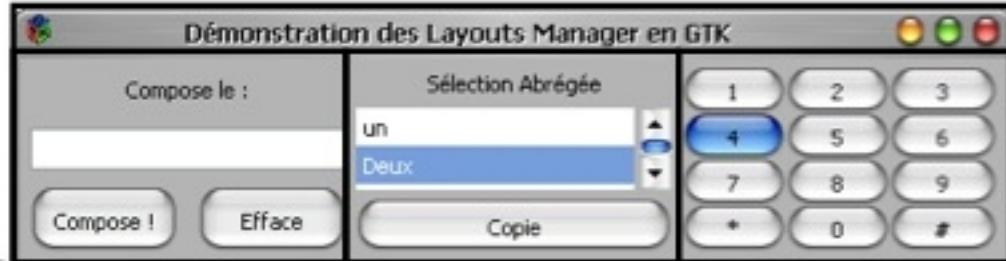


# Plan

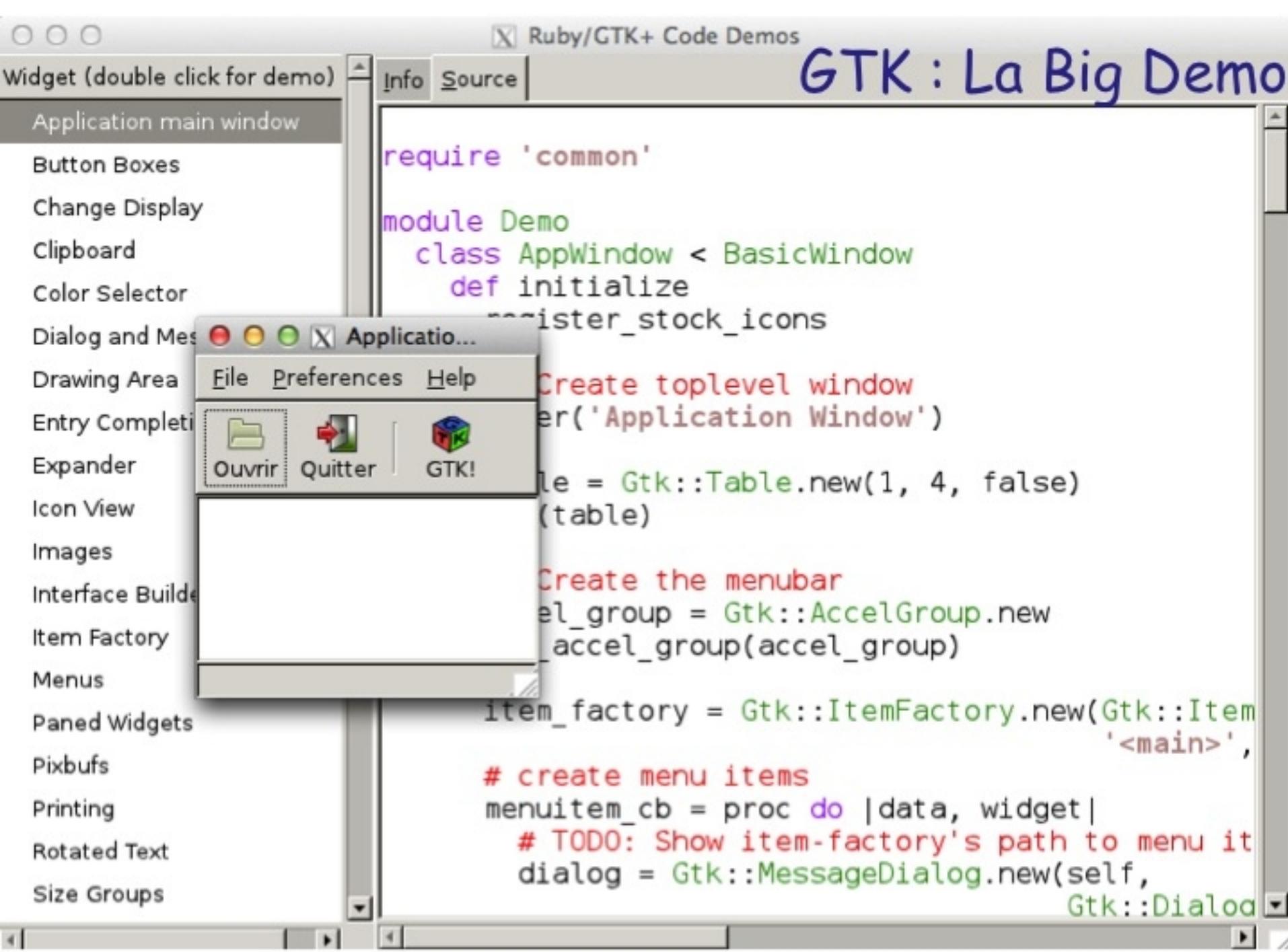
---

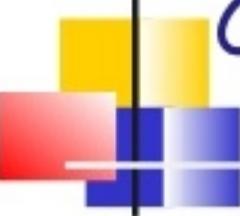
- A-Les Interfaces Graphiques
  - 1-Généralités,
  - 2-Ruby & GUI
  - 3-Tk
  - 4-FoxRuby
  - 5-Gtk+
- B-Exemple
- C-Compléments

## B- Exemple



- H Box
  - Vbox
    - Label,
    - Entry,
    - Hbox
      - Button, Button
  - Vbox
    - Label, TreeView, Button
  - Table
    - (4 lignes, 3 colonnes) => 12 Buttons





## C- Complément

---

- Utilisation des listes ListStore
- Utilisation des Arbres TreeStore
- Les vues TreeView et les renderers
- La persistence
- La Manipulation des onglets
  
- Dans TutoGtk

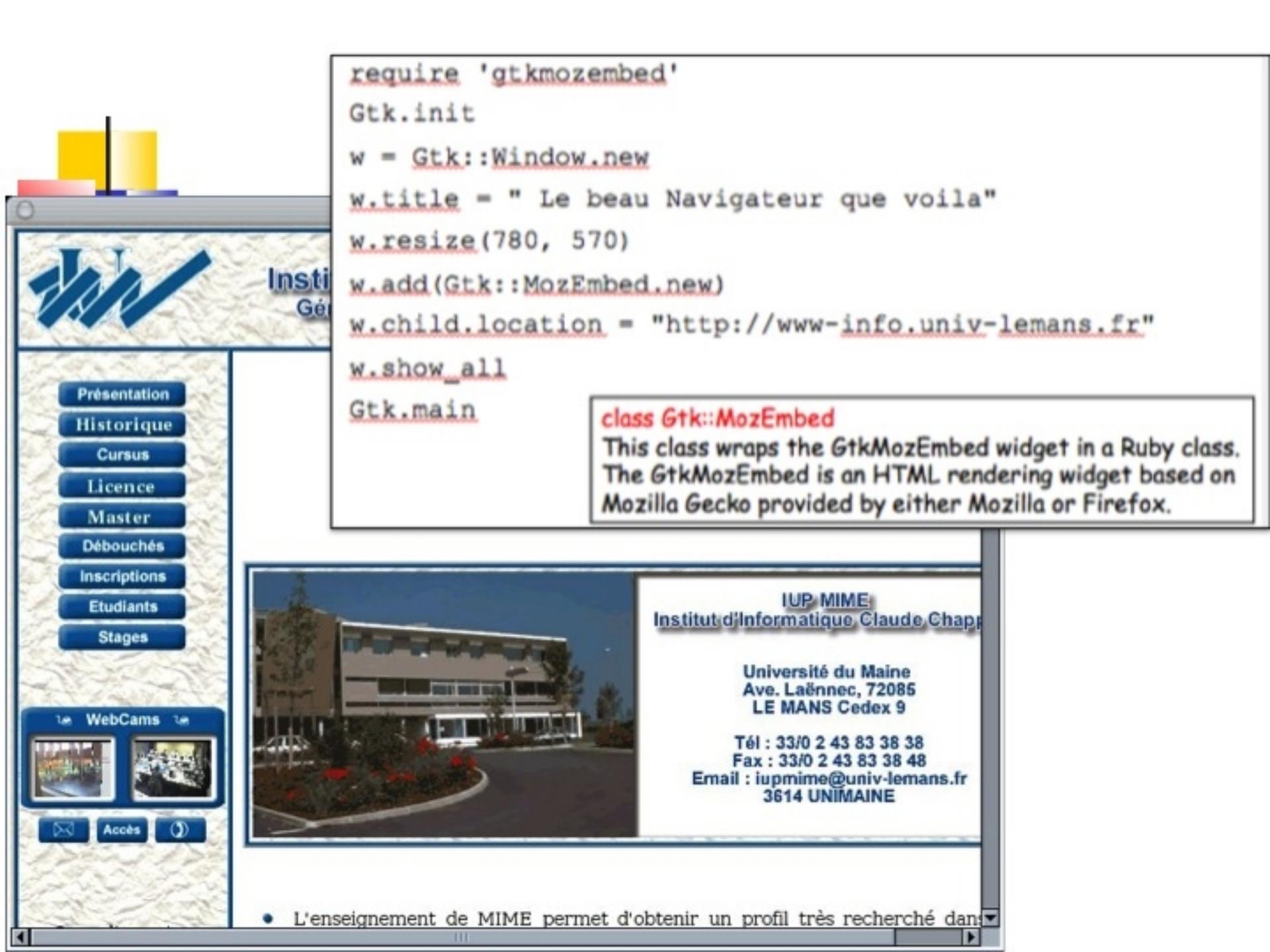
# C-Soyons fous

- Allez on fait un navigateur Web en 9 lignes (Linux)

```
require 'gtkmozembed'  
Gtk.init  
  
w = Gtk::Window.new  
w.title = " Le beau Navigateur que voila"  
w.resize(780, 570)  
w.add(Gtk::MozEmbed.new)  
w.child.location = "http://www-info.univ-lemans.fr"  
w.show_all  
Gtk.main
```

### class Gtk::MozEmbed

This class wraps the `GtkMozEmbed` widget in a Ruby class.  
The `GtkMozEmbed` is an HTML rendering widget based on  
Mozilla Gecko provided by either Mozilla or Firefox.



```
require 'gtkmozembed'

Gtk.init

w = Gtk::Window.new
w.title = " Le beau Navigateur que voila"
w.resize(780, 570)
w.add(Gtk::MozEmbed.new)
w.child.location = "http://www-info.univ-lemans.fr"
w.show_all

Gtk.main
```

### class Gtk::MozEmbed

This class wraps the `GtkMozEmbed` widget in a Ruby class. The `GtkMozEmbed` is an HTML rendering widget based on Mozilla Gecko provided by either Mozilla or Firefox.



### IUP MIME Institut d'Informatique Claude Chapp

Université du Maine  
Ave. Laënnec, 72085  
LE MANS Cedex 9

Tél : 33/0 2 43 83 38 38  
Fax : 33/0 2 43 83 38 48  
Email : iupmime@univ-lemans.fr  
3614 UNIMAINE

- L'enseignement de MIME permet d'obtenir un profil très recherché dan



## C-Complément 1 : Utiliser Glade

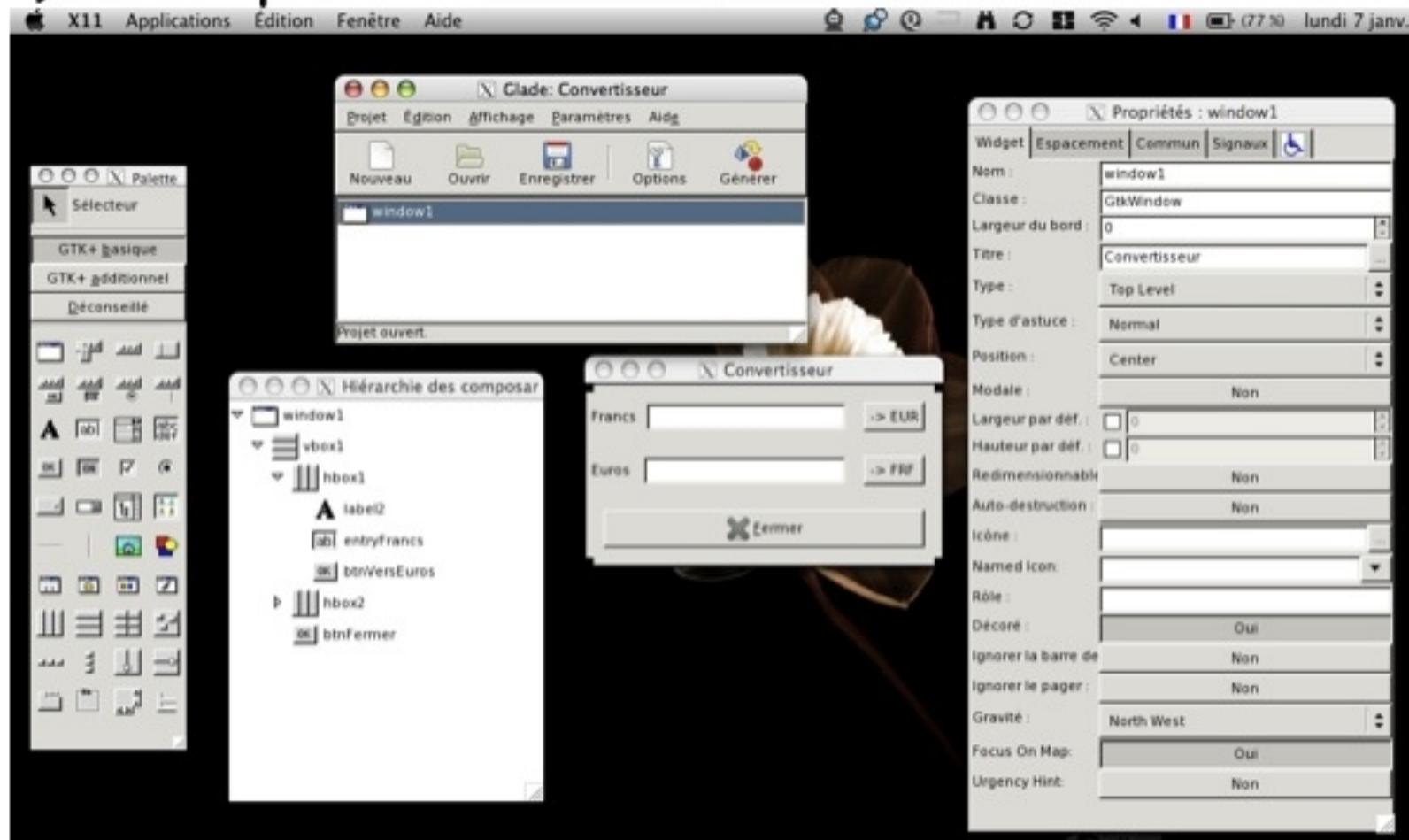
---

### 1) Conception de l'interface dans Glade

- Glade est un outil interactif de construction d'interface graphique pour l'environnement GNOME.
  - Il prend en charge toute la partie de gestion/génération de l'interface pour permettre au développeur de se concentrer sur le code « utile ».
- Glade enregistre les interfaces graphiques en générant des fichiers XML.
  - La bibliothèque `libglade` permet de lire ces fichiers dynamiquement.
  - ces fichiers XML peuvent être utilisés par de nombreux langages de programmation *C*, *C++*, *Java*, *Perl*, *Python*, *C#*, *Pike*, **Ruby**, *Haskell*, *Objective Caml* et *Scheme*.

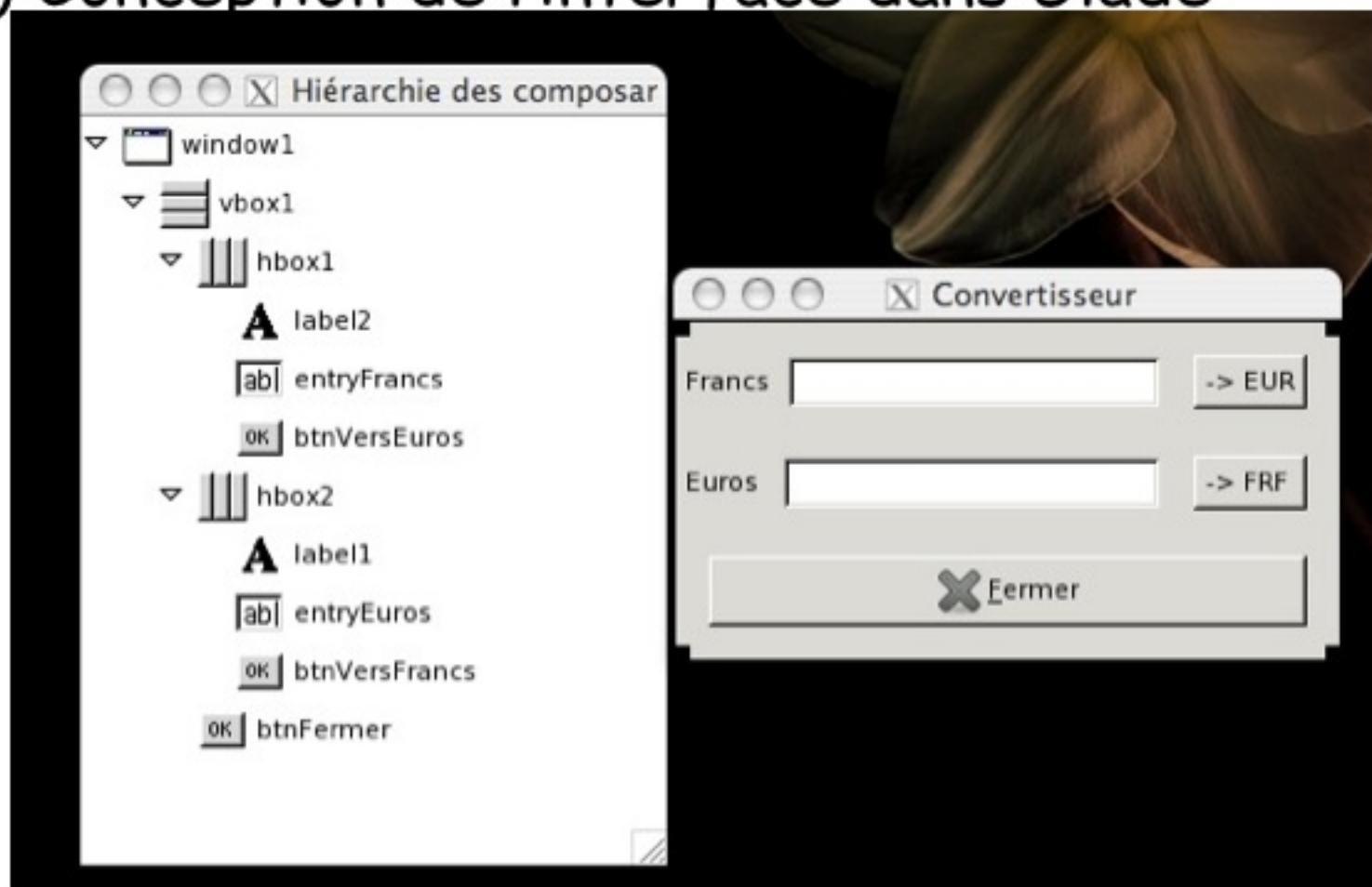
# C-Complément 1 : Utiliser Glade

## 1) Conception de l'interface dans Glade



# C-Complément 1 : Utiliser Glade

## 1) Conception de l'interface dans Glade



## C-Complément 1 : Utiliser Glade

### 1) Conception de l'interface dans Glade

- On dessine d'abord l'interface sans s'occuper des noms des widgets
- quand ça ressemble à ce qu'on veut :
  - on nomme les objets qui ont un rôle
    - Dans l'exemple : on donne des noms
      - aux 3 boutons (**button**)  
btnVersEuros, btnVersFrancs et btnFermer
      - aux 2 champs de saisies (**entry**)  
entryFrancs et entryEuros
  - on fait générer les handlers d'évènements qui nous intéressent.
    - clicked : **boutons**, destroy : **fenetre**



# C-Complément 1 : Utiliser Glade

## 2) On sauve le fichier glade,

- on obtient une description Xml de l'interface.
- Ce fichier pourra être utiliser avec tout langage de programmation supportant la bibliothèque libglade.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <interface>
3   <!-- interface-requires gtk+ 2.6 -->
4   <!-- interface-naming-policy toplevel-contextual -->
5   <object class="GtkWindow" id="window1">
6     <property name="visible">True</property>
7     <property name="can_focus">False</property>
8     <property name="title" translatable="yes">Convertisseur</property>
9     <property name="resizable">False</property>
10    <property name="window_position">center</property>
11    <signal name="destroy" handler="on_window1_destroy" swapped="no"/>
12    <child>
13      <object class="GtkVBox" id="vbox1">
14        <property name="visible">True</property>
15        <property name="can_focus">False</property>
16        <property name="border_width">5</property>
17        <child>
18          <object class="GtkHBox" id="hbox1">
19            <property name="visible">True</property>
20            <property name="can_focus">False</property>
21            <child>
22              <object class="GtkLabel" id="label2">
```

# C-Complément 1 : Utiliser Glade

## 3) On crée le fichier ruby pour utiliser l'interface

```
1 ##  
2 # Auteur Pierre Jacoboni  
3 # Version 0.1 : Date : Mon Jul 01 1  
4 #  
5 require 'gtk2'  
6  
7 class Convertisseur < Gtk::Builder  
8  
9   def initialize  
10     super()  
11     self.add_from_file(__FILE__.sub(".rb",".glade"))  
12  
13     self['window1'].set_window_position Gtk::Window::POS_CENTER  
14     self['window1'].signal_connect('destroy') { Gtk.main_quit }  
15     self['window1'].show_all  
16     # Creation d'une variable d'instance par composant glade  
17     self.objects.each() { |p|  
18       instance_variable_set("@#{p.builder_name}".intern, self[p.builder_name])  
19     }  
20  
21     self.connect_signals{ |handler|  
22       method(handler)  
23     }  
24     @taux=6.55957  
25   end
```

The code block is highlighted with a red border. A red arrow points from the bottom right towards the highlighted code, specifically pointing at the final block of code starting with '# Creation d'une variable d'instance par composant glade'.

l'objet Glade nommé vbox1 devient une VI de nom @vbox1  
l'objet Glade nommé window1 devient une VI de nom @window1  
l'objet Glade nommé btnFermer devient une VI de nom @btnFermer  
l'objet Glade nommé btnVersEuros devient une VI de nom @btnVersEuros  
l'objet Glade nommé label1 devient une VI de nom @label1  
l'objet Glade nommé label2 devient une VI de nom @label2  
l'objet Glade nommé entryEuros devient une VI de nom @entryEuros  
l'objet Glade nommé btnVersFrancs devient une VI de nom @btnVersFrancs  
l'objet Glade nommé hbox1 devient une VI de nom @hbox1  
l'objet Glade nommé hbox2 devient une VI de nom @hbox2  
l'objet Glade nommé entryFrancs devient une VI de nom @entryFrancs

# C-Complément 1 : Utiliser Glade

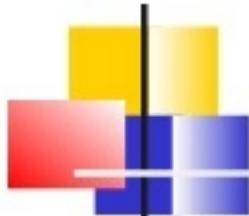
## ■ Les Signaux pour quitter

```
26  
27     def on_window1_destroy(widget) →  
28         Gtk.main_quit  
29     end  
30  
31     def on_btnFermer_clicked(widget) →  
32         Gtk.main_quit  
33     end
```



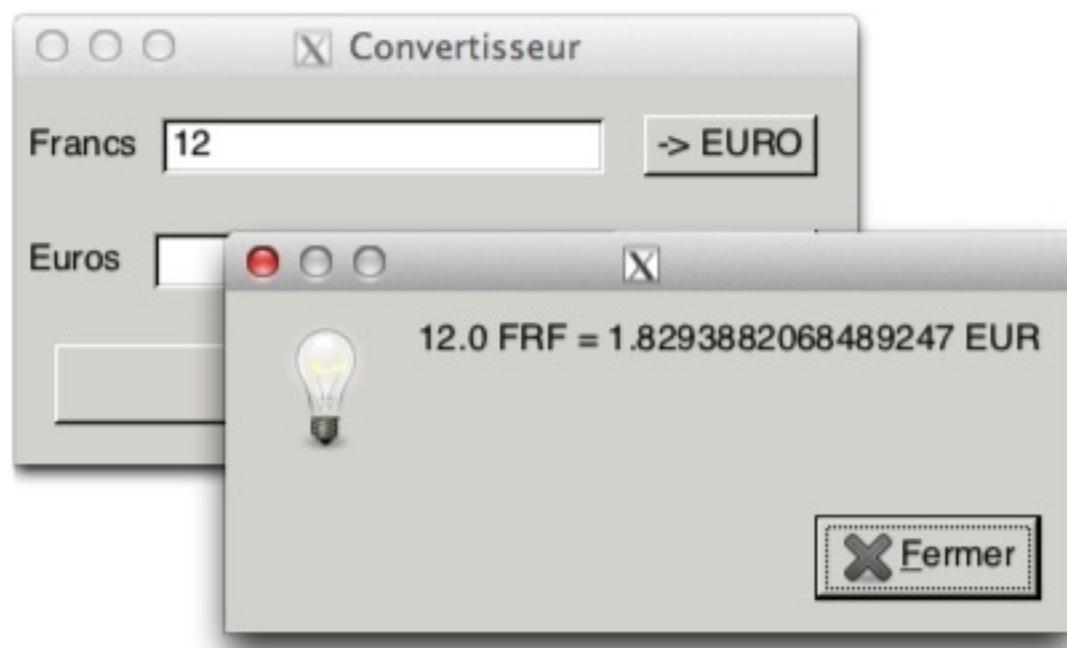
## ■ Les signaux pour les boutons et les champs

```
35     def on_btnVersFrancs_clicked(widget) →  
36         on_entryEuros_activate(widget)  
37     end  
38  
39     def on_entryEuros_activate(widget) →  
40         f = @entryEuros.text.to_f * @taux  
41         d = Gtk::MessageDialog.new(@window1, Gtk::Dialog::DESTROY_WITH_PARENT,  
42                                     Gtk::MessageDialog::INFO,  
43                                     Gtk::MessageDialog::BUTTONS_CLOSE,  
44                                     "#{@entryEuros.text.to_f} EUR = #{f} FRF")  
45         d.run  
46         d.destroy  
47     end
```

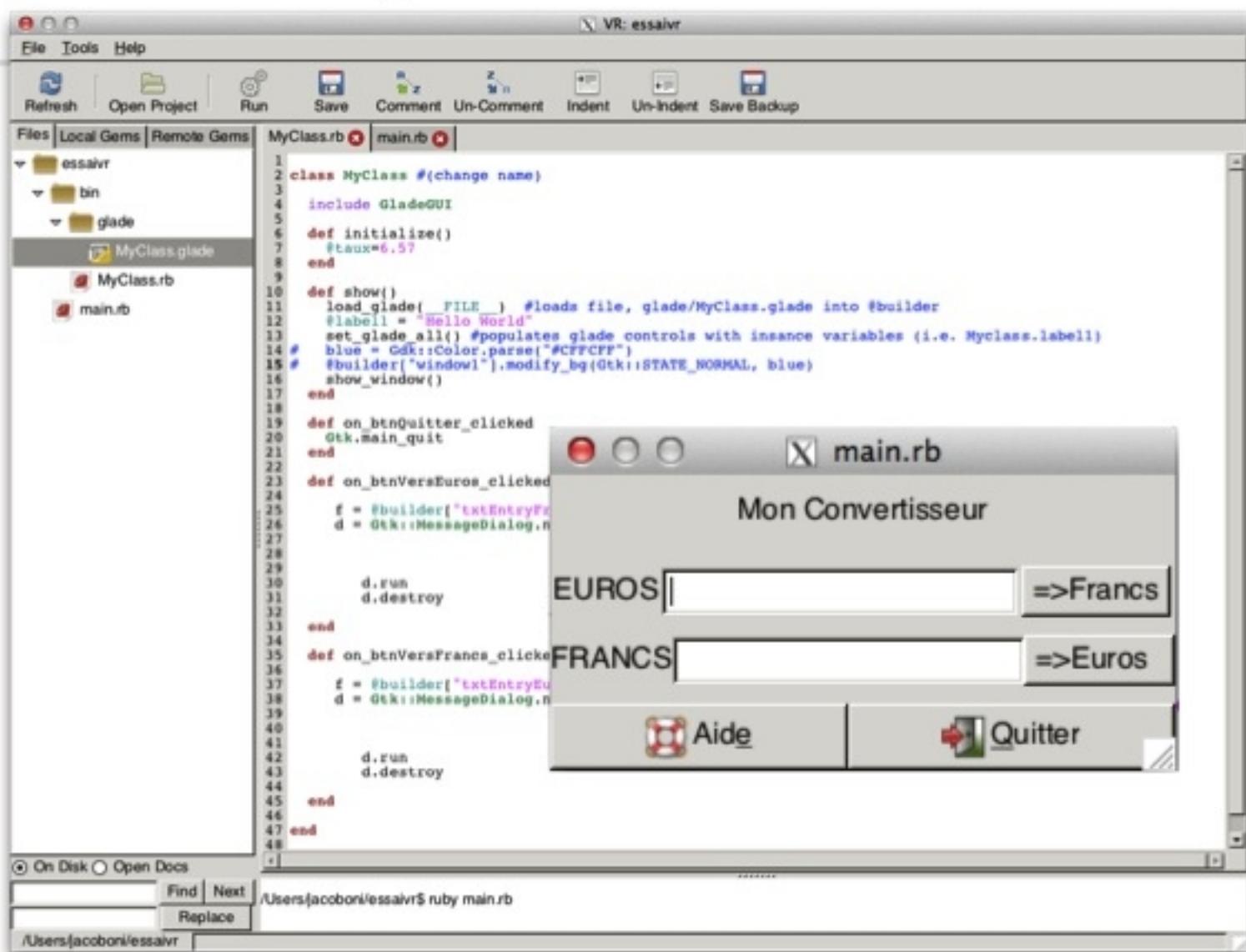


## Lancer l'exécution

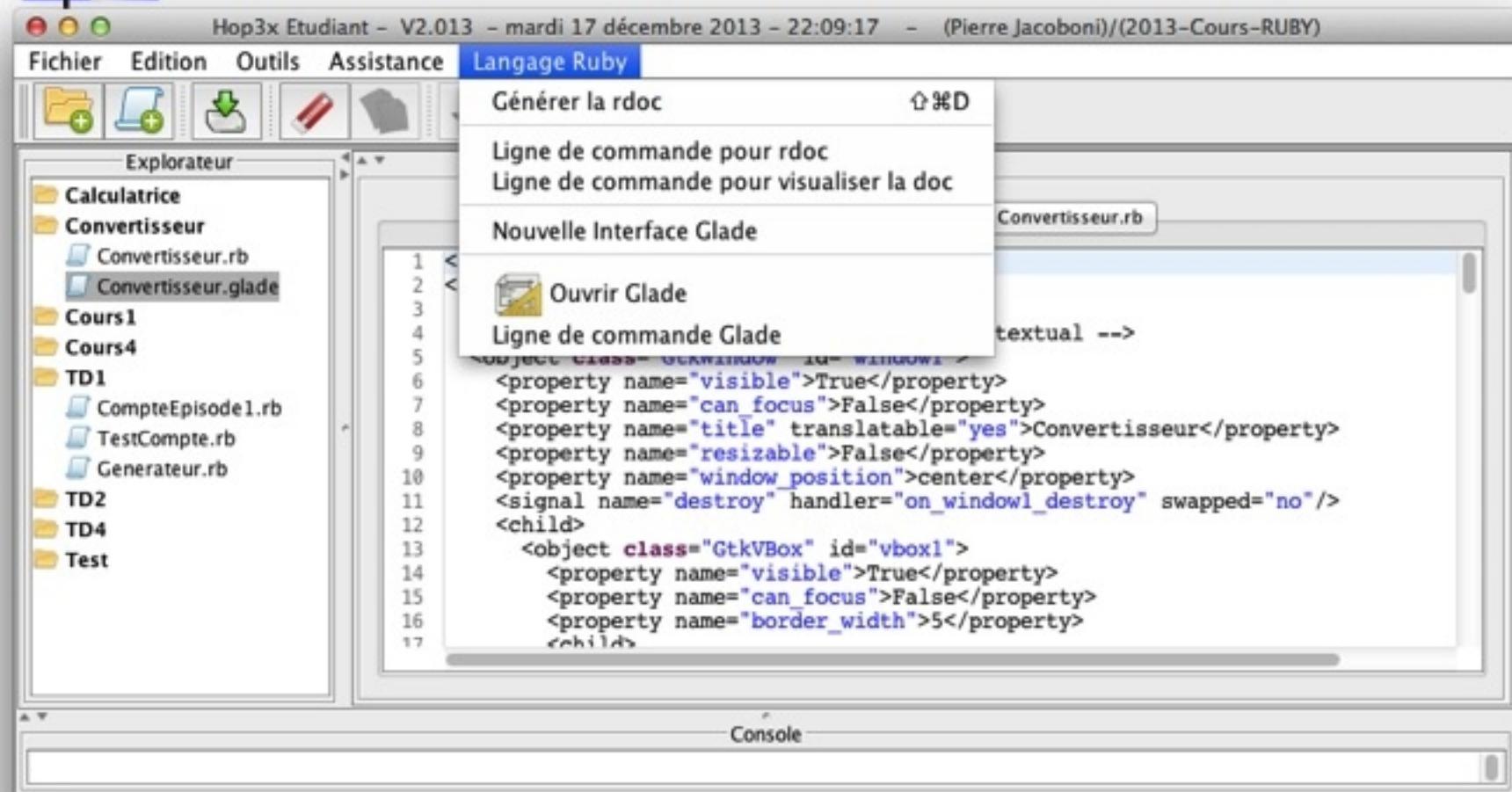
```
65  
66 Gtk.init  
67 convertisseur = Convertisseur.new()  
68 Gtk.main
```



# Avec VisualRuby



# Evidemment avec Hop3x

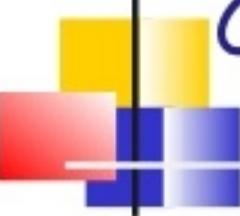




## C-Complément 2 : WxRuby

---

- wxRuby est un autre toolkit permettant de définir des interfaces graphiques notamment en Ruby
- Il autorise des applications au look natif pour windows, OsX, Gnome et toute autre plateforme en utilisant uniquement Ruby.
- Il est basé sur le framework d'application graphique multiplateforme WxWidget



## C-Complément 2 : WxRuby

---

### ■ Pourquoi WxRuby?

- multi-plateforme : Windows, Linux, Mac OS X, et BSD -
- L&F Natif : les Windows widgets sous Win32, Aqua sous OS X et GTK sous Linux
- Installation Facile: un « gem » sans dépendance externe. (les gems sont des « paquets Ruby », gem est un gestionnaire de paquets, cf yum, apt, ...)
- Ensemble complet de widget: Basiques et avancés (par exemple, HTML viewer)
- Internationalisation: wxRuby permet un support complet d'UTF8
- Repose sur wxWidgets un ensemble de widget graphique très mature, largement utilisé et dans une communauté active
- Licence libre

# C-Complément 2 : WxRuby

## ■ Exemple 1 (extrait)

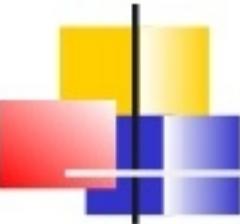
```
def on_init
    @accumulated, @elapsed= 0, 0
    @frame=Wx::Frame.new(nil,-1,'Chronomètre WxRuby')
    menu_bar=Wx::MenuBar.new
    program_menu=Wx::Menu.new
    menu_bar.append(program_menu,'&Chrono')
    program_menu.append(START_MENU,'&Start', 'Démarrer le Chrono')
    @frame.evt_menu(START_MENU){start}
    program_menu.append(STOP_MENU,'&Stop', 'Arrêter le Chrono')
    @frame.evt_menu(STOP_MENU){stop}
    menu_exit=program_menu.append(EXIT_MENU,'E&xit Alt_X', 'Quitter')
    @frame.evt_menu(EXIT_MENU){exit}
    reset_menu=Wx::Menu.new
    menu_bar.append(reset_menu,'&Reset')
    reset_menu.append(RESET_MENU,'&Reset', 'Reset du Chrono')
```



# WxRuby

## Exemple

### 2



```
#!/usr/bin/env arch -i386 ruby
require 'rubygems'
require 'wx'
include Wx

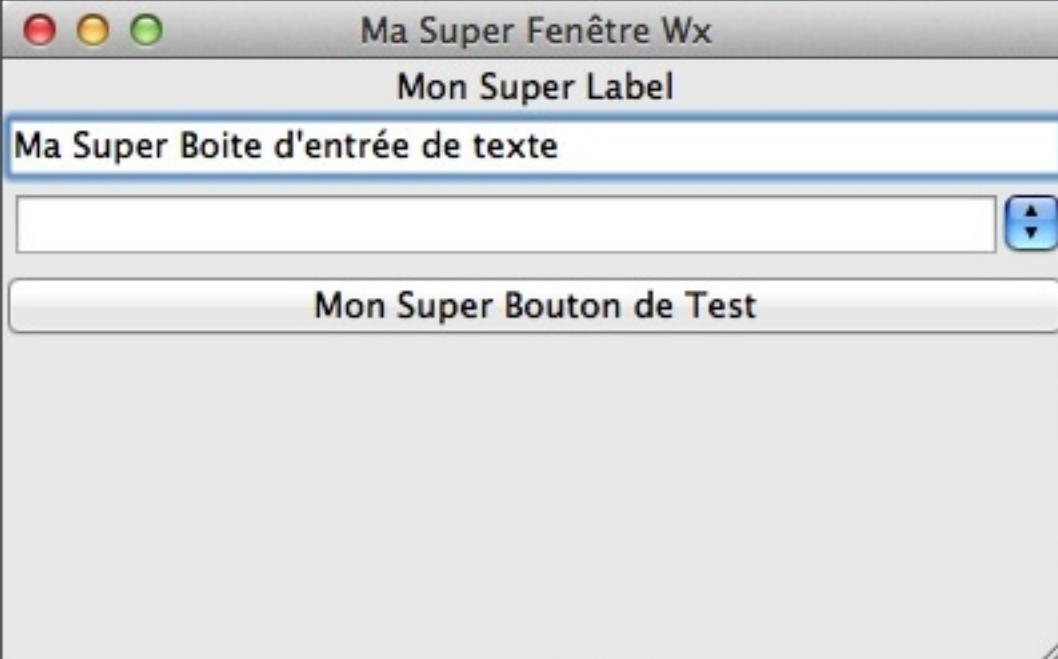
class MyFrame < Frame
  def initialize()
    super(nil, -1, 'Ma Super Fenêtre Wx')
    # Création des controles
    @my_panel = Panel.new(self)
    @my_label = StaticText.new(@my_panel, -1, 'Mon Super Label',
      DEFAULT_POSITION, DEFAULT_SIZE, ALIGN_CENTER)
    @my_textbox = TextBox.new(@my_panel, -1, 'Ma Super Boite d\'entrée de texte')
    @my_combo = ComboBox.new(@my_panel, -1, 'Text par défaut',
      DEFAULT_SIZE, ['Item 1', 'Item 2', 'Item 3'])
    @my_button = Button.new(@my_panel, -1, 'Mon Super Bouton de Test')

    @my_sizer = BoxSizer.new(VERTICAL)
    @my_sizer.Add(@my_label, 0, GROWIALL, 2)
    @my_sizer.Add(@my_textbox, 0, GROWIALL, 2)
    @my_sizer.Add(@my_combo, 0, GROWIALL, 2)
    @my_sizer.Add(@my_button, 0, GROWIALL, 2)

    get_id() { |event| my_button_click(event)}
  end
end

class MyApp < App
  def on_init
    MyFrame.new
  end
end

MyApp.new.main_loop()
```



# Wxruby : La big Demo

WxRuby BIG combined demo

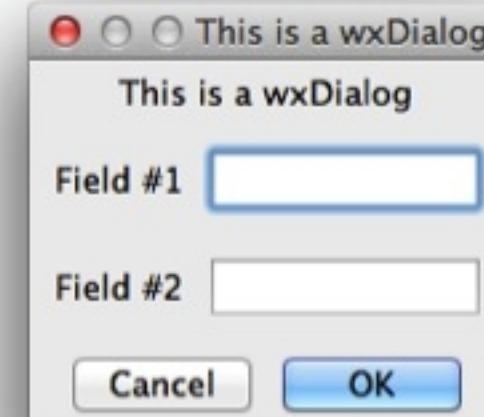
- ▼ wxRuby Overview
- ▼ Base Frames and Dialogs
  - wxDialog
  - wxFrame
  - wxMDIWindows
  - wxMiniFrame
- ▼ Common Dialogs
  - wxColourDialog
  - wxDirDialog
  - wxFFileDialog
  - wxFFileDialog\_Save
  - wxFindReplaceDialog
  - wxFontDialog
  - wxMessageDialog
  - wxProgressDialog
  - wxSingleChoiceDialog
  - wxTextEntryDialog
- ▶ More Dialogs
- ▶ Core Windows/Controls
- ▶ More Windows/Controls
- ▶ Window Layout
- ▶ Using Images
- ▶ Miscellaneous
- Check out the samples dir

wxDialog Overview

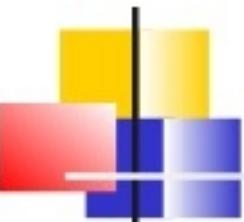
Demo Code

```
#!/usr/bin/env ruby
# wxRuby2 Sample Code. Copyright (c) 2004-2008 wxRuby development team
# Freely reusable code: see SAMPLES-LICENSE.TXT for details
begin
  require 'rubygems'
rescue LoadError
end
require 'wx'

class TestDialog < Wx::Dialog
  def initialize(parent, id, title)
    super(parent, id, title)
    sizer = Wx::BoxSizer.new(Wx::VERTICAL)
    label = Wx::StaticText.new(self, :label => "This is a wxDialog")
  end
end
```



10:04:57: Running Demo: wxMDIWindows.rbw  
10:05:03: Running Demo: wxMiniFrame.rbw  
10:05:04: Running Demo: wxFrame.rbw  
10:05:08: on\_item\_expanded: Common Dialogs  
10:05:09: Running Demo: wxFFileDialog.rbw  
10:05:09: CWD: /Users/jacoboni/Dropbox/Enseignement/Enseignement 11-1:  
10:05:19: Running Demo: wxDialog.rbw  
10:05:25: You pressed Cancel  
10:05:28: Running Demo: wxDialog.rbw



# wxRuby Documentation: Class Reference

This is the class reference for WxRuby. The classes listed below are those currently available in WxRuby2. This documentation was seeded by converting the WxWidgets C++ documentation, so it has some shortcomings.

- A small number of methods have inaccurate descriptions of the parameters they accept
- Some of the example code is still in C++
- There are links to C++ tutorials and topic overviews that are not available – many are not relevant to wxRuby.

The following tutorials contain helpful overviews specific to the wxRuby API:

- [wxRuby introduction](#) : wxRuby syntax conventions
- [window styles overview](#) : window styles and wxRuby constants.

## Outline

---

wxRuby includes a large number of classes. Some represent a GUI element represented on the screen, such as a Window, Frame, Control or Menu:

- [Window](#) : Anything drawn on the screen
- [Frames](#) : Top-level windows that contain controls
- [Controls](#) : Windows for user interaction, eg buttons
- [Miscellaneous windows](#) : Miscellaneous other on-screen items
- [Common dialogs](#) : Task-specific standalone windows
- [Menus](#) : Drop-down menus associated with a Frame
- [Rich text](#) : Editing, viewing and printing formatted text
- [HTML classes](#) : Displaying simple hypertext on screen
- [AUI: advanced user interface](#) : IDE-like floatable panes and toolbars

# Complément 3 : Installer tout ça (GiMF et LTeMA) sous Windows



## Installation de ruby 1.9.3, GTK et VisualRuby sous Windows

Testée Ok sur Windows 7 et 8 le 20 juin 2013

Supprimer toute installation préalable de Ruby et Gtk. Il faut que gcc soit installé et dans le Path

### 1- Installer Ruby

Télécharger et installer le '[One-Click Installer](#)' de ruby1.9.3 pour windows, par exemple  
[rubyinstaller-1.9.3-p429.exe](#)

Cocher les 3 cases et notamment celle qui concerne le Path

### 2- Installer Glade 3

### 3- Installer VisualRuby

Dans un terminal

`gem install visualruby`

### 4- Vérification

Dans un terminal

`vr` doit lancer quelque chose

# Complément 3 : Installer tout ça (GiMF et LTeMA) sous MacOsX

The screenshot shows a web browser window with the following details:

- Title bar: Document sans nom
- Address bar: hop3x.univ-lemans.fr/MacOsX-RubyGtk.html
- Content area:
  - Installation de ruby 1.9.3, GTK et VisualRuby sous MacOsX pour utilisation dans Hop3x.**
  - Text: Testée Ok sur Mountain lion (10.8) le 26 novembre 2013
  - Text: Testée Ok sur Mavericks (10.9) le 27 novembre 2013
  - Text: Les commandes en rouge doivent être tapées/copiées dans un terminal :

## A- *Gcc, Xcode et Xquartz doivent être installés*

## B- Installer *brew*

(Si les "command line tools" ne sont pas installées la commande 01 proposera de le faire)

- 01) `ruby -e "$(curl -fsSL https://raw.githubusercontent.com/mxcl/homebrew/go/install)"`
- 02) `brew doctor`

Si vous comprenez les recommandations données par la seconde commande vous pouvez les suivre  
(éventuellement suppression de fichier existant, mise à jour xcode, etc...)

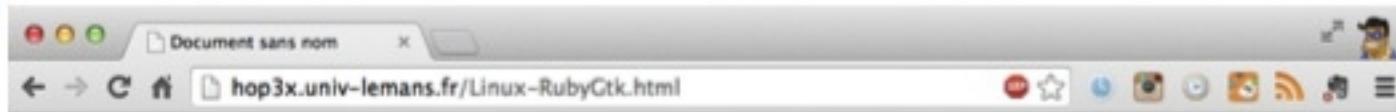
## C - Installer la version 1.9.3 de *ruby*

Les commandes suivantes servent à mettre à jour les sources, à se placer dans le bon répertoire (attention dans 04 c'est le caractère BackQuote) et à installer la version 1.9.3 de Ruby

- 03) `brew tap homebrew/versions`

- 04) `cd 'brew --prefix'`

# Complément 3 : Installer tout ça (GiMF et LTeMA) sous Linux



## Installation de ruby 1.9.3, GTK et VisualRuby sous Linux

Testée Ok sur Debian et Ubuntu et Fedora le 20 juin 2013

Il faut que gcc soit installé et dans le Path

**1- Installer Ruby 1.9.3 (le numéro de version est important) avec votre gestionnaire de paquet préféré.**

**2- Dans un terminal**

```
sudo apt-get install -y glade libgtksourceview2.0-dev gtk2-engines-pixbuf
```

(pour Fedora les noms des paquets sont un peu différents)

**3- Dans un terminal**

```
sudo gem install visualruby
```

**4- Vérification dans un terminal**

**vr** doit lancer quelque chose



## Complément 3 : Installer tout ça (GiMF et LTeMA)

---

- **Installer** Fox

- Windows :

```
gem install fxruby
```

- MacOsX :

```
brew install fox
```

```
sudo gem install fxruby
```

- **Installer** WxRuby

```
[sudo] gem install wxruby
```



## Complément 3 : Installer tout ça (GiMF et LTeMA) sous MacOsX

---

- wxruby sous MacOsX
  - Comme c'est une version 32 bit pour le faire tourner il faut utiliser ruby en 32 bit

```
arch -i386 ruby testWx.rb
```
- Solution Idéale : Installer rvm qui permet de gérer plusieurs versions de Ruby / gems pour les utilisateurs. (GiMF)



## Complément 3 : Installer tout ça (GiMF et LTeMA)

---

- Pour les bases de données

```
# sequel permet de travailler avec la
# base sqlite en mémoire

[sudo] gem install sequel

# avec sqlite sur fichier

[sudo] gem install sqlite

# Il faut ActiveRecord

[sudo] gem install activerecord
```



## Complément 3 : Installer tout ça (GiMF ou GeMA)

---

- Pour Mysql

- Sous MacOsX

```
brew update
```

```
brew install mysql
```

```
sudo gem install mysql2
```

- Pour Démarrer le serveur Sql

```
mysql -uroot
```

```
mysql.server start
```

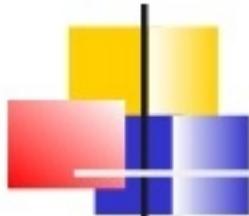
- Pour l'arreter

```
mysql.server stop
```

- Sous Windows/Linux (GiMF)

- Il faut installer un serveur mysql

```
[sudo] gem install mysql
```



FIN ?

---

# Complément 4 : Prog Dynamique



```
2000
#<X:0x10e3fedf0>
["@@maVariableDeClasse"]
#<X:0x10e3fedf0 @monNom="Després">
Mon Nom est Lemeunier
#<X:0x10e3fec38 @monNom="Lemeunier">
500
#<X:0x10e3fedf0 @monNom="Després">
```

```
class X
  def self.ajouteVarClasse( unSymbole, uneValeur )
    class_variable_set( unSymbole, uneValeur )
  end
  def self.retourneValeurVarClasse( unSymbole )
    return class_variable_get( unSymbole )
  end
  def ajouteMethode( m, &unBloc )
    self.class.send( :define_method, m, &unBloc )
  end
end

X.ajouteVarClasse( :@@maVariableDeClasse, 2000 )

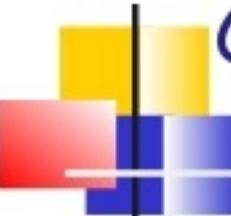
puts( X.retourneValeurVarClasse( :@@maVariableDeClasse ) )

ob=X.new
p ob
p( X.class_variables )

ob.instance_variable_set("@monNom", "Després")
p ob
ob.aj
outeMethode( :nouvelleMethode ) { puts("Mon Nom est #{@monNom}") }

ob2 = X.new
ob2.instance_variable_set("@monNom", "Lemeunier")
ob2.nouvelleMethode
p ob2

X::const_set( :NUM, 500 )
puts( X::const_get( :NUM ) )
p ob
```



# Complément 4 : Prog Dynamique

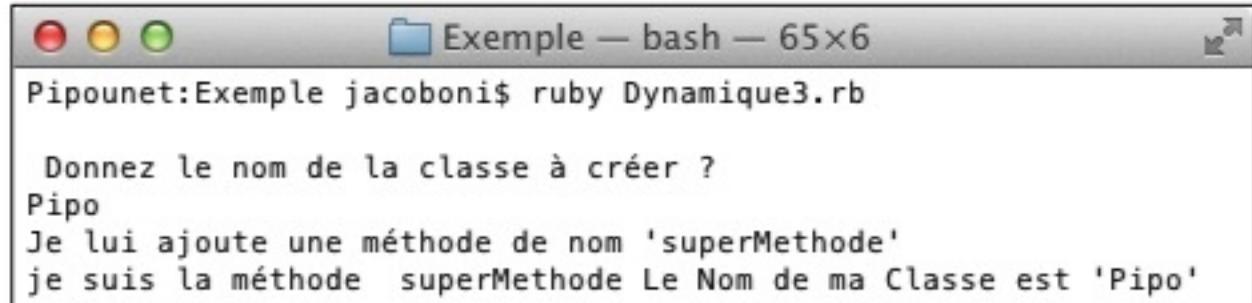
---

- Je veux créer une classe

```
puts("\n Donnez le nom de la classe à créer ? ")
nomClasse = gets.strip().capitalize()
Object.const_set(nomClasse, Class.new)
```

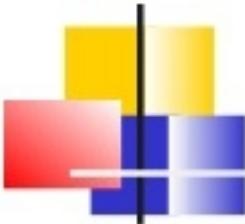
- Je veux lui ajouter une méthode

```
puts("Je lui ajoute une méthode de nom 'superMethode'" )
laClasse = Object.const_get(nomClasse)
laClasse::module_eval{ define_method(:superMethode){
    puts("je suis la méthode superMethode Le Nom de ma Classe est '#{self.class}'" )
}}
x = laClasse.new
x.superMethode
```



```
Pipounet:Exemple jacoboni$ ruby Dynamique3.rb
Donnez le nom de la classe à créer ?
Pipo
Je lui ajoute une méthode de nom 'superMethode'
je suis la méthode superMethode Le Nom de ma Classe est 'Pipo'
```

J'en  
veux  
plus



SapphireSteel Software: The Book Of Ruby

www.sapphiresteel.com/The-Book-Of-Ruby

Boutique de vente en ligne | Essais, review et avis | Comment figurer sur Google

Autres favoris

SapphireSteel Software

Products

Programming

Back to [Tutorials](#) > [Ruby In Steel Tutorials](#) > [Programming](#)

## The Book Of Ruby

The Book Of Ruby (free edition) is a comprehensive free tutorial to the Ruby language. It contains in the form of a PDF document in which each chapter is accompanied by ready-to-run source code. Introduction which explains how to use the source code in Ruby In Steel or any other editor/IDE or index.

bookofruby.pdf - Adobe Acrobat Pro

File Edit View Document Comments Forms Tools Advanced Window Help

387 / 425

Bookmarks

- Chapter Seven
- Chapter Eight
- Chapter Nine
- Chapter Ten
- Chapter Eleven
- Chapter Twelve

CREATING CLASSES AT RUNTIME

So far we have modified classes and created new objects. But how would you go about creating a completely new class? Just as `const_get` can be used to access an existing class, `const_set` can be used to create a new class. Here's an example of how to prompt the user for the name of a new class before creating that class, adding a method to it, creating an instance, `x`, of that class and calling its `mymethod` method:

```
puts("What shall we call this class? ")
className = gets.strip.capitalize()
```