1 Les objets Gtk

En Gtk, on manipulera (quasi-) exclusivement des données de type GtkWidget. Parmi celles-ci, vont nous intéresser plus particulièrement les fenêtres (fenêtre classique), les labels (libellé sur une fenêtre), les boutons (zone cliquable par l'utilisateur et engendrant une action), les zones de saisie (zones de texte saisissable par l'utilisateur). Ce sont ces composants qui permettent à la fois d'informer et de mettre en place l'interactivité entre l'application et l'utilisateur.

1.1 Qu'est-ce qu'un Widget ?

Un *Widget* est un objet graphique qui possède des *attributs* et des *méthodes* permettant de manipuler ces attributs. Le nom de Widget vient de la contraction des termes *Window* et *Gadget*. Tous les objets graphiques que nous allons manipuler avec Gtk sont des Widgets particuliers qui héritent des attributs et des méthodes d'un objet GtkWidget : les fenêtres (type GtkWindow), les boutons (type GtkButton) ou encore les labels (type GtkLabel) sont des *spécialisations* des Widgets, c'est-à-dire qu'ils disposent des caractéristiques de la structure GtkWidget, mais avec en plus des attributs et des méthodes qui leur sont propres.

1.2 Arborescence des types gtk

Le schéma qui suit permet de situer les types que nous allons manipuler au sein de l'arborescence des types Gtk et Glib. **En gras** figurent les objets graphiques présentés dans de document.

GObject // *Provient de la bibliothèque Glib sur laquelle Gtk est elle-même construite*
- GtkObject // *Ça y est, on entre dans l'arborescence Gtk à proprement parler*
  - **GtkWidget** //*Le gadget Gtk de base (on ne remontra jamais jusqu'à GtkObject)*
    - GtkContainer
      - GtkBin
        - **GtkWindow**
          - *GtkDialog*
        - **GtkButton**
      - GtkBox
        - **GtkHbox**
        - **GtkVBox**
      - *GtkTable*
    - **GtkEntry**
    - GtkMisc
      - **GtkLabel**
      - *GtkImage*

2 Objets usuels

Remarques préliminaires :

1. Toutes les fonctions ne sont pas listées pour chaque objet ⇒ pour en savoir plus, ne pas hésiter à consulter le fichier ".h" (fichier gtkwindow.h pour l'objet fenêtre, gtkbutton.h pour l'objet bouton,...) !

2. À chaque méthode “set” correspond une méthode “get” ; plus précisément, chaque attribut de chaque objet dispose d'accesseurs en lecture (méthode “get”) ainsi qu'en écriture (méthode “set”) : les *acesseurs en lecture* permettent de *consulter* la valeur d'un attribut, les *acesseurs en écriture* permettent *d'affecter* la valeur d'un attribut. Par souci de concision, on ne listera pas ici les méthodes “get” (ni les méthodes “set” de tous les attributs, d'ailleurs). Illustration, sur l'attribut booléen “resizable” de l'objet graphique “GtkWindow” :

```
gboolean gtk_window_get_resizable(GtkWindow *window);                    // lecture
void gtk_window_set_resizable (GtkWindow *window, gboolean resizable);   // écriture
```

3 Objet GtkWindow

3.1 Fonctions spécifiques à la fenêtre

Constructeur GtkWidget* gtk_window_new(GtkWindowType type);
Deux valeurs sont possibles pour le type énuméré GtkWindowType : GTK_WINDOW_TOPLEVEL (fenêtre “classique” avec toutes ses décorations) et GTK_WINDOW_POPUP (fenêtre “Pop Up”).

```
GtkWidget* pFenetre; // toujours déclarer ses objets comme GtkWidget*
pFenetre = gtk_window_new(GTK_WINDOW_TOPLEVEL);
```

Titre void gtk_window_set_title(GtkWindow* pFenetre, const gchar* strTitle);
On peut affecter autant de fois que l'on veut (quoique que cela n'ait pas grand sens) le titre de la fenêtre.

```
GtkWidget* pFenetre;
pFenetre = gtk_window_new(GTK_WINDOW_TOPLEVEL);
gtk_window_set_title(GTK_WINDOW(pFenetre), "toto le héros"); // toujours préciser la façon dont un objet doit être interprété par le compilateur pour une fonction donnée
gtk_window_set_title(GTK_WINDOW(pFenetre), "tata l'héroïne");
```

Taille par défaut void gtk_window_set_default_size(GtkWindow* pFenetre, gint width, gint height);
Le programmeur peut affecter une taille par défaut pour l'affichage initial de la fenêtre :

```
GtkWidget* pFenetre;
pFenetre = gtk_window_new(GTK_WINDOW_TOPLEVEL);
gtk_window_set_default_size(GTK_WINDOW(pFenetre), 200, 150);
```

Position par défaut void gtk_window_set_position(GtkWindow* window, GtkWindowPosition position);
De nouveau, le programmeur peut décider du positionnement de la fenêtre, dans la limite des positions prédéfinies par Gtk au travers du type énuméré GtkWindowPosition ; ces positions sont les suivantes :

GTK_WIN_POS_NONE	la fenêtre est placée n'importe où.
GTK_WIN_POS_CENTER	la fenêtre est centrée sur l'écran.
GTK_WIN_POS_MOUSE	la fenêtre est placée là où se trouve la souris à la création de la fenêtre.
GTK_WIN_POS_CENTER_ALWAYS	la fenêtre est toujours au centre de l'écran.
GTK_WIN_POS_CENTER_ON_PARENT	la fenêtre est centrée sur sa fenêtre parente.

La différence entre “GTK_WIN_POS_CENTER” et “GTK_WIN_POS_CENTER_ALWAYS” réside en le fait qu'avec la seconde valeur, la fenêtre se replace d'elle-même au centre en cas de modification d'affichage.

```
GtkWidget* pFenetre;
pFenetre = gtk_window_new(GTK_WINDOW_TOPLEVEL);
gtk_window_set_position(GTK_WINDOW(pFenetre), GTK_WIN_POS_CENTER_ALWAYS);
```

Comportement void gtk_window_set_resizable (GtkWindow *window, gboolean resizable);
Paramètre "resizable" mis à "FALSE" ⇒ l'utilisateur ne pourra pas redimensionner la fenêtre.

3.2 Fenêtre et Container

Ajouter un objet graphique à une fenêtre void gtk_container_add(GtkContainer*, GtkWidget*);
Quelques remarques :
- (rappel) un objet graphique ne sera pas affiché s'il n'appartient pas (directement ou par l'intermédaire d'autres objets graphiques) à un composant affichable, c'est-à-dire, à un GtkWindow !
- (rappel) seuls les objets GtkContainer (ou en héritant) peuvent contenir d'autres objets ; c'est ce qui explique purquoi la fonction de rattachement d'un objet à un autre est définie au niveau de l'objet GtkCtainer.
- (attention) bien utiliser la macro GTK_CONTAINER() pour que votre fenêtre passée en argument soit effectivement interprétée comme un objet de type GtkContainer.
- (attention) les GtkWindow sont des GtkBin, cas particulier de GtkContainer qui ne peuvent contenir *qu'un* objet !
- (attention) la destruction d'un container (et donc en particulier d'une fenêtre) engendre la destruction des éventuels objets qu'il contient !

4 Objet GtkButton

4.1 Fonctions spécifiques au bouton

4.1.1 Constructeurs

Simple GtkWidget* gtk_button_new();
Cette méthode crée un bouton sans libellé. Pour ajouter un libellé à ce bouton, on pourra faire appel à la méthode "gtk_button_set_label()".

Avec libellé GtkWidget* gtk_button_new_with_label(const gchar* label);
Le bouton porte le libellé donné en argument.

Avec raccourci clavier GtkWidget* gtk_button_new_with_mnemonic(const gchar* label);
Le bouton porte le libellé donné en argument. Ce libellé doit contenir le caractère "underscore". Le caractère L qui suit l'underscore permet d'activer le bouton par le raccourci clavier "ALT+L" (même effet qu'un clic ou que l'appui sur "enter" lorsque le bouton a le focus). Si la chaîne ne contient pas le caractère "underscore", le résultat est le même qu'avec le constructeur "gtk_button_new_with_label()", c'est-à-dire qu'on obtient un bouton avec libellé, mais sans raccourci clavier.

```
GtkWidget* pBoutonQuitter;
pBoutonQuitter = gtk_button_new_with_mnemonic("Cliquer ici pour _Quitter");
g_signal_connect(G_OBJECT(pBoutonQuitter), "clicked", G_CALLBACK(gtk_main_quit), NULL);
```

Si l'utilisateur tape "ALT+'Q"', il quitte l'application. Remarques :

1. les raccourcis clavier ne sont pas sensibles à la casse ;
2. du moment qu'un bouton a été créé par la méthode "gtk_button_new_with_mnemonic()", si l'on change son libellé pour un nouveau libellé qui contient également un raccourci clavier (*i.e.*, présence du caractère "underscore" à l'intérieur de la nouvelle chaîne de caractères), celui-ci sera pris en compte ;

3. en revanche, affecter un libellé comportant un raccourci clavier à un bouton créé par une autre méthode n'aura aucune incidence (le caractère "underscore" est alors traité comme un caractère normal à afficher dans le libellé).

Avec image GtkWidget* gtk_button_new_from_stock(const gchar* stock_id);
Argument *stock_id* : cet argument peut prendre comme valeur toute constante faisant référence à une image existante. Certaines images sont définies avec Gtk et les constantes associées (leur id) sont définies dans le fichier à entête "gtkstock.h". Le bouton créé contient alors l'image correspondant à l'argument stock_id et peut, selon les cas, comporter un raccourci clavier (en fonction de l'image insérée). Quelques exemples des images dont vous disposez "en stock" :

`GTK_STOCK_DIALOG_WARNING`	Triangle avec point d'exclamation suivi du texte "Warning".
`GTK_STOCK_YES`	Disque suivi du texte "Yes" avec raccourci clavier "ALT+'Y"'.
`GTK_STOCK_QUIT`	Icône et texte "Quit" avec raccourci clavier "ALT+'Q"'.

4.1.2 Autres méthodes

Relief void gtk_button_set_relief(cGtkButton *button, GtkReliefStyle newstyle);
Définit le style de relief du bouton, qui peut prendre 3 valeurs possibles définies dans le type énuméré "GtkReliefStyle" : GTK_RELIEF_NORMAL, GTK_RELIEF_HALF ou GTK_RELIEF_NONE.

4.2 Boutons, événements et signaux

Ci-suit un tableau listant les signaux pouvant être émis par un bouton. Pour chaque signal, on précise l'événement qui le génère.

activate	Le bouton a le focus et que l'on appuie sur la touche "Enter".
clicked	L'utilisateur a cliqué sur le bouton.
enter	Le pointeur de la souris entre dans la zone du bouton.
leave	Le pointeur de la souris quitte la zone du bouton.
pressed	L'utilisateur appuie sur le bouton.
released	L'utilisateur relâche le bouton.

5 Objet GtkBox

5.1 Introduction

Pour palier le fait qu'on ne peut attribuer qu'un GtkWidget à une fenêtre (vrai des GtkBin en général), on peut utiliser les GtkBox. Un objet GtkBox permet de définir un découpage en lignes ou en colonnes et peut contenir autant d'objets qu'il contient de lignes ou de colonnes. Ainsi, en ajoutant un objet GtkBox à une fenêtre, on peut par son intermédiaire associer autant d'objets qu'on le souhaite à la fenêtre !

Ce ne sont pas exactement des objets "GtkBox" que l'on utilise, mais des "GtkHBox" et des "GtkVBox", qui en héritent. Un objet "GtkHBox" sépare l'espace de la fenêtre en colonnes, tandis que l'objet "GtkVBox" sépare l'espace de la fenêtre en lignes. Le nombre d'éléments que l'on place dans une boîte n'est pas prédéfini mais est fonction des insertions successives des objets dans la boîte. Notons qu'une boîte peut contenir tout type d'objet GtkWidget et ainsi, un objet GtkBox peut en contenir un autre (ce qui permet d'obtenir tous les découpages imaginables).

Ci-suit un exemple de code qui affiche les deux boutons "Quitter" et "Traduire" sur une même fenêtre, avec en plus un bouton "Annuler" ; ces trois boutons sont alignés horizontalement, on sépare donc la fenêtre en trois bandes verticales.

```
// Déclaration des trois boutons
GtkWidget* pButtonQuit=NULL;
GtkWidget* pButtonCancel=NULL;
GtkWidget* pButtonTranslate=NULL;
// Déclaration de la boîte horizontale
GtkWidget* pHBox=NULL;
// Creation des boutons
pButtonQuit = gtk_button_new_with_label("Quitter");
pButtonCancel = gtk_button_new_with_label("Annuler");
pButtonTranslate = gtk_button_new_with_label("Traduire");
// Creation de la boîte
pHBox = gtk_hbox_new(TRUE, 0);
// Ajout des trois boutons à la boîte
gtk_box_pack_start(GTK_BOX(pHBox), pButtonTranslate, FALSE, FALSE, 0);
gtk_box_pack_start(GTK_BOX(pHBox), pButtonCancel, FALSE, FALSE, 0);
gtk_box_pack_start(GTK_BOX(pHBox), pButtonQuit, FALSE, FALSE, 0);
// On ajoute la boîte à la fenêtre
gtk_container_add(GTK_CONTAINER(pWindow), pHBox);
// Affichage de la fenêtre et de tout ce qu'elle contient
gtk_widget_show_all(pWindow);
```

5.2 Boîte et fonctions

Constructeurs

`GtkWidget* gtk_hbox_new(gboolean homogeneous, gint spacing);`
Création d'une boîte horizontale.
`GtkWidget* gtk_vbox_new(gboolean homogeneous, gint spacing);`
Création d'une boîte verticale.

- Paramètre "homogeneous" : (TRUE/FALSE)
  Si ce paramètre est mis à "TRUE", l'espace occupé par la boîte est équitablement distribué parmi ses différents contenus : si on insère 5 objets, chacun disposera de 20% de la hauteur dans le cas d'une boîte verticale, de la largeur dans le cas d'une boîte horizontale.

- Paramètre "spacing" : entier
  Ce paramètre permet de définir, en pixels, une marge entre les espaces réservés à deux contenus successifs.

Ajouter un objet à une boîte

On peut considérer une boîte comme une liste, verticale ou horizontale, d'objets. On peut alors insérer les objets soit en début (haut/gauche), soit en fin (bas/droite) de liste :
`void gtk_box_pack_start(GtkBox*, GtkWidget*, gboolean expand, gboolean fill, guint padding)`
insère l'objet GtkWidget en bas (*resp.*, à droite) dans la boîte verticale (*resp.*, horizontale).
`void gtk_box_pack_end(GtkBox*, GtkWidget*, gboolean expand, gboolean fill, guint padding)`
insère l'objet GtkWidget en haut (*resp.*, à gauche) dans la boîte verticale (*resp.*, horizontale).

- Paramètre "expand" : (TRUE/FALSE)
  Ce paramètre permet encore de définir la répartition de l'espace entre objets insérés. Attention : il ne sera pris en compte que si la boîte a été créée avec le paramètre "homogeneous" mis à "FALSE".
  Si donc la boîte a été créée avec le paramètre "homogeneous" mis à "FALSE", les objets insérés avec "expand" mis à "FALSE" prendront l'espace (hauteur pour une boîte verticale, largeur pour une boîte horizontale) qui leur est juste nécessaire. Les objets insérés avec le paramètre "expand" mis à "TRUE" se partagent alors équitablement l'espace restant.

- Paramètre “fill” : (TRUE/FALSE)
  Si fill est mis à “TRUE”, alors le contenu occupe tout l’espace qui lui est réservé.

- Paramètre “padding” : entier non signé
  Ce paramètre permet de définir, en pixels, une marge entre le contenu et le bord de l’espace qu’il occupe.

Gestion de l’espace
Bon, c’est pas très simple tout ça... Les paramètres des fonctions de construction et d’insertion permettent d’agir sur la taille *du contenant* et *du contenu*. Pour chaque objet, on définit d’abord un espace (contenant) dont il dispose, par les paramètres “homogeneous” du constructeur et “expand” de la fonction d’insertion. On définit ensuite la taille du contenu par le paramètre “fill” de la fonction d’insertion. Enfin, on définit deux types de marge : la première, paramètre “spacing” du constructeur, permet de séparer les contenants, tandis que la seconde, paramètre “padding” de la fonction d’insertion, permet de séparer le contenu du contenant.

6 Objet GtkEntry

6.1 Fonctions

`GtkWidget* gtk_entry_new();`
constructeur
`G_CONST_RETURN gchar* gtk_entry_get_text(GtkEntry* entry);`
récupération du texte saisi
`void gtk_entry_set_text(GtkEntry* entry, const gchar* text);`
écrire un texte dans la zone de saisie
`void gtk_entry_set_max_length(GtkEntry* entry, gint max);`
borner le nombre de caractères
`void gtk_entry_set_editable(GtkEntry* entry, gboolean editable);`
décider de l’activation de la zone de saisie
`void gtk_entry_set_visibility(GtkEntry* entry, gboolean visible);`
décider de la visibilité des caractères saisis
`void gtk_entry_set_invisible_char(GtkEntry* entry, gunichar ch);`
si “gtk_entry_get_visibility(GTK_ENTRY(pEntry)) == FALSE”,
remplace les caractères saisis par le caractère “ch” donné en argument

6.2 Exemples

```
GtkWidget* pSaisie; // Declaration d'un GtkWidget*... comme toujours !
pSaisie = gtk_entry_new(); // Creation d'une GtkEntry (bien qu'un pointeur sur GtkWidget soit renvoye)
gtk_entry_set_max_length(GTK_ENTRY(pSaisie), 12); // On borne le nombre de carcateres a 12
gtk_entry_set_text(GTK_ENTRY(pSaisie),"login"); // On ecrit "login" dans la zone
gtk_entry_set_visibility(GTK_ENTRY(pSaisie), TRUE); // Les caracteres de la zone sont visibles
gtk_entry_set_editable(GTK_ENTRY(pSaisie), TRUE); // L'utilisateur peut ecrire dans la zone
```

6.3 Quelques événements

activate void user_function(GtkEntry* entry, gpointer user_data);
Ce signal est émis lorsque la zone de texte a le focus et que l’on appuie sur la touche "Enter".
delete-from-cursor void user_function(GtkEntry* entry, gpointer user_data);
Ce signal est émis lorsque l’utilsateur supprime du texte (“delete” ou “backslash”).
insert-at-cursor void user_function(GtkEntry* entry, gpointer user_data);
Ce signal est émis lorsque l’utilsateur écrit.
move-cursor void user_function(GtkEntry* entry, gpointer user_data);
Ce signal est émis lorsque l’utilsateur déplace le curseur.

Exercices

Pour l'utilisation des boîtes (exercices 1 et 2), on utilisera dans un premier temps les valeurs de paramètre :
- `homogeneous = FALSE` et `spacing = 0` pour les constructeurs `gtk_hbox_new()` et `gtk_vbox_new()` ;
- `expand = fill = FALSE` et `padding = 0` pour la méthode d'insertion `gtk_box_pack_start()`.

Exercice 0

À partir de l'exercice 3 du TP précédent :

1. Faire en sorte que le bouton "Traduire" puisse être activé en tappant "ALT+'T"'.
2. À peine plus fantaisiste, faire en sorte que le bouton "Traduire" puisse être activé en tappant "ALT+'D"' dans sa version française, par "ALT+'L"' dans sa version anglaise.
3. Centrer (toujours !) la fenêtre à l'écran, et interdire à l'utilisateur facétieux de modifier sa taille.

Exercice 1

1. Dans une fenêtre, placer un label en haut, un bouton au mileu et un bouton en bas ; le titre de la fenêtre est "Ma merveilleuse application Gtk", le label contient le texte "Bonjour !", le premier bouton a pour libellé "Traduire" (avec raccourci clavier sur le 'T'), le second a pour libellé "Quitter" (avec raccourci clavier que le 'Q').
2. Le premier bouton permet de quitter l'application, le second de traduire son propre libellé.
3. Faire en sorte que le second bouton permette *aussi* de traduire le libellé du premier bouton.
4. Faire en sorte que le second bouton permette *aussi* de traduire le libellé du label.
5. Faire en sorte que le second bouton permette *aussi* de traduire le titre de la fenêtre.
6. Faire en sorte que le second bouton permette de *tout* traduire (libellé du label, libellé des boutons, titre de la fenêtre).

Exercice 2 : des boutons en série *et* en parallèle

1. Créer une fenêtre, de taille 800×100, qui contient 4 boutons alignés horizontalement. Le titre de la fenêtre est "Qui est clicked ?" et les boutons, à l'initialisation, doivent afficher le texte "Je ne suis pas clicked" (désolée pour l'anglicisme, c'est le problème des accents qui ne sont pas gérés trivialement en Gtk).
2. En gérant seulement l'événement "clicked" survenant aux objets GtkButton, faire en sorte que, quand un bouton est cliqué, celui-ci affiche "Je suis clicked", tandis que quand il ne l'est plus, il affiche de nouveau "Je ne suis pas clicked".
3. Au lieu d'une fenêtre 800×100, créer une fenêtre carrée de 400 pixels de côté, qui contient 2 rangées de 2 boutons.

Exercice 3 : un peu de mise en forme

1. À partir de la question 2 de l'exercie 2 (boutons en lignes) :
   (a) faire en sorte que les boutons remplissent tout l'espace qui leur est réservé ;
   (b) aérer néanmoins la fenêtre en espaçant deux espaces consécutifs de la boîte par une bande 20 pixels ;
   (c) annuler la modification précédente et aérer de nouveau la fenêtre, en définissant cette fois-ci des marges de 20 pixels autour des contenus des boîtes ;
   (d) comment arriver à obtenir une marge de 20 pixels, non seulement entre le bord gauche (*resp.*, droit) de la fenêtre et le premier (*resp.*, le dernier) bouton, mais aussi entre les bords de deux boutons successifs ?
2. À partir de la question 3 de l'exercie 2 (quadrillage de boutons) :
   (a) faire en sorte que les boutons se répartissent tout l'espace de la fenêtre ;
   (b) aérer ensuite la fenêtre de sorte à avoir une marge de 20 pixels, d'une part entre les bords de la fenêtre et les boutons, d'autre part entre deux boutons consécutifs (en ligne ou en colonne).

Exercice 4 : mieux gérer l'événementiel

Sur la base de l'exercice précédent :

1. Si l'on considère qu'un bouton est en état "non cliqué" à l'ouverture de la fenêtre et lorsque la souris l'a quitté, modifier la gestion événementielle de l'alternance des labels sur les boutons en exploitant les deux événements "clicked" et "leave".
2. Pour tester un peu les émissions de signaux produites par la souris et le clavier, associer à chaque événement pouvant survenir à un bouton une fonction callback qui indique simplement sur le bouton concerné (émetteur de l'événement) le dernier signal émis.
3. Ne peut-on pas faire de même en utilisant une seule fonction de callback ?

Exercice 5 : la guerre des boutons

Sur la base de la fenêtre précédente :

1. Le premier bouton doit être créé à l'aide du constructeur avec raccourci clavier : il doit être écrit "Quitter" sur ce bouton et le raccourci clavier pour l'activer doit être "ALT+'U"'. Pour les trois derniers boutons, incorporer respectivement les images d'identifiant `GTK_STOCK_QUIT`, `GTK_STOCK_DIALOG_WARNING` et `GTK_STOCK_YES`.
2. Pour que les raccourcis clavier (certains du moins) servent à quelque chose, associer aux deux premiers boutons le callback qu'il vous faut.

Exercice 6 : "Qui êtes-vous ?"

1. Créer une fenêtre, de taille 200×100, qui contient deux zones de saisie alignées verticalement ; le titre de la fenêtre est "Qui etes-vous ?", la première zone de saisie contient le texte "NOM", la seconde, le texte "PRENOM" ; enfin, les deux zones de saisie limitées à 50 caractères.
2. Pour chaque zone de saisie, faire en sorte que l'utilisateur ne puisse saisir que des caractères alphabétiques.
3. Pour chaque zone de saisie, faire en sorte qu'après saisie, les caractères soient tous des caractères majuscules.
4. Ajouter un bouton "Valider" en bas de la fenêtre. Lorsque l'on clique sur ce bouton, le nom et le prénom saisis sont affichés sur la console (méthode "printf()"), puis on quitte l'application graphique.

Exercice 7 : "Connected people"

1. Créer une fenêtre de titre "Se connecter" contenant les éléments suivants :
   (a) le libellé "Login :" suivi d'une zone de saisie limitée à 8 caractères ;
   (b) en-dessous, le libellé "Mot de passe :" suivi d'une zone de saisie également limitée à 8 caractères ;
   (c) en-dessous encore, les trois boutons "Quitter", "Annuler" et "Valider".
2. Naturellement, le mot de passe ne doit pas apparaître à l'écran !
3. Imaginons de plus que l'utilisateur souhaite rester anonyme et qu'il soit vénal de sucroît : pour le login, ne pas afficher le texte saisi, mais seulement une suite de caractères "$".
4. Le bouton "Quitter" doit permettre de quitter la fenêtre, tandis que le bouton "Annuler" efface les zones de saisie.
5. Le bouton "Valider" change le titre de la fenêtre en "Utilisateur connecté" et rend inactifs les champs de saisie.
6. Mieux, le bouton "Valider" précise le login de l'utilisateur : "Utilisateur *toto* connecte" si *toto* est le login saisi.