Interface Utilisateur Graphique

avec scheme

Lien vers la bibliothèque des interfaces utilisateur graphiques (GUI) de racket:
http://docs.racket-lang.org/gui/index.html

L'objectif du TP est de vous faire découvrir la bibliothèque des interfaces utilisateur graphique de scheme (ou plus exactement de racket). Pour cela nous reprendrons le projet du répertoire (dont on vous donne une version). Mais avant de rentrer dans le vif du sujet, un peu de prise en main.

D'abord il est nécessaire d'inclure la bibliothèque :-)

```
(require racket/gui/base)
```

Nous allons commencer par un petit exemple simple : une fenêtre de dialogue, avec un champ de saisie, et un bouton. D'abord commençons par créer la fenêtre :

```
(define $dialog
  (new dialog% [label "Exemple"] [parent #f] [width 200] [height 100]))
```

La fenêtre que nous venons de créer est un *dialog* (cf la doc pour en savoir plus, notamment la différence entre *frame* et *dialog*).Elle a pour titre (dans la barre de titre de la fenêtre) *Exemple* ; est de largeur 200 px et de hauteur 100 px ; et n'a pas de parent (voir la doc pour les autres options). Cette notion de parent est fondamentale pour les interfaces graphiques. En effet, les éléments de l'interface sont disposés dans des conteneurs, qui eux-mêmes sont stockées dans d'autre conteneurs,etc, jusqu'à un conteneur dit top-level : la fenêtre (ici notre dialog). Pour pouvoir retrouver la fenêtre, on la stocke dans une variable, `$dialog`.

Deuxième élément que nous devons créer : le champ de saisie :

```
(define $texte
  (new text-field% [label "Votre nom :"][parent $dialog][init-value "NOM"]))
```

Il s'agit d'un objet de type text-field. Le *label* (qui va s'afficher devant le champ de saisie est *Votre nom :* et le *parent* de cet objet est `$dialog`. Le *init-value* sert à mettre une valeur par défaut dans le champ. Nous l'avons mis dans une variable pour pouvoir le retrouver plus tard.

Il nous faut ensuite faire le bouton :

```
(define $bouton-ok
 (new button% [label "Ok"]  [parent $dialog]
      [callback
       (lambda (b e)
         (let ((nom (send $texte get-value)))
           (set! $nom nom)
           (send $dialog show #f)))
       ]
      ))
```

C'est un objet de type button. Passons sur les paramètres *parent* et *label*. Vous devriez devinez sans peine leur rôle. Intéressons nous de plus prêt au paramètre *callback*:cela permet de définir la fonction qui sera évaluée lorsque le bouton sera cliqué. Ici, on fit trois opérations : on récupère la chaîne de caractères présente dans le champ de saisie `(send $texte get-value)` et on la met dans une variable `nom` ; on change la valeur d'une variable `$nom` définie par ailleurs (avec un `(define $nom "le nom avant")`) ; puis on ferme la fenêtre de dialogue `(send $dialog show #f)`.

Il n'y a plus qu'à définir la fonction qui va lancer notre application :

```
(define (go)
  (printf "Votre nom est initialement : ~a~n" $nom)
  (send $dialog show #t)
  (printf "Votre nom est maintenant : ~a~n" $nom))
```

La ligne importante est : `(send $dialog show #t)`. C'est elle qui permet l'affichage de la fenêtre principale …
Enfin, pour lancer l'application :

```
(go)
```

Pour compléter notre interface on peut ajouter une bouton annuler, dont le rôle est de fermer la fenêtre sans rien faire :

```
(define $bouton-cancel
  (new button% [label "annuler"]  [parent $dialog]
       [callback
        (lambda (b e)
            (send $dialog show #f))
        ]
       ))
```

Comme vous pouvez le constater, les deux boutons sont affichés l'un sous l'autre. Ce n'est pas vraiment esthétique. On préfèrerait qu'ils soient affichés l'un à côté de l'autre. Le problème est que le dialog est un conteneur de type vertical : les objets que l'on met à l'intérieur sont disposés verticalement, dans l'ordre de leur création. Pour avoir l'effet désiré, il faut que l'on créé un conteneur horizontal (un objet de type horizontal-panel, qui accueillera les deux boutons. et ce conteneur sera inclus dans le dialog :

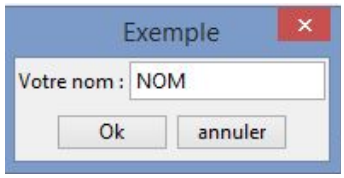
```
(define $panel (new horizontal-panel% [parent $dialog]))
```

Et pour que les deux boutons intègrent le panel, il faut changer leur parent : `[parent $panel]`, à la place de `[parent $dialog]`.

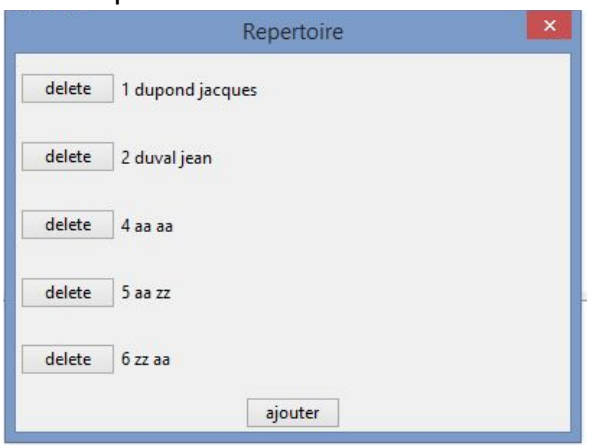
Et pour que les boutons soient centrés dans leur panel, on peut soit mettre l'option adéquate lors de la création du panel, soit le faire après coup :

```
(send $panel set-alignment 'center 'center)
```

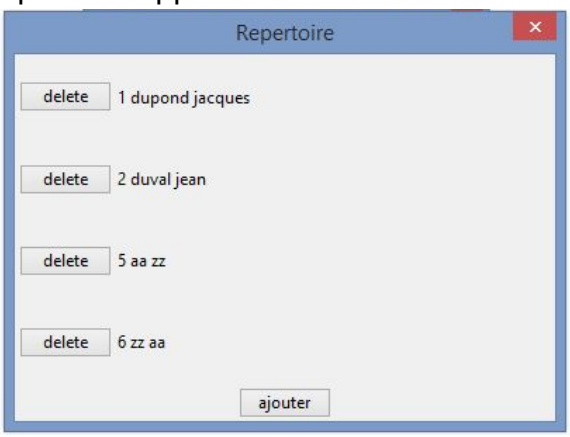
Et vous devriez obtenir quelque chose qui ressemble à ceci :



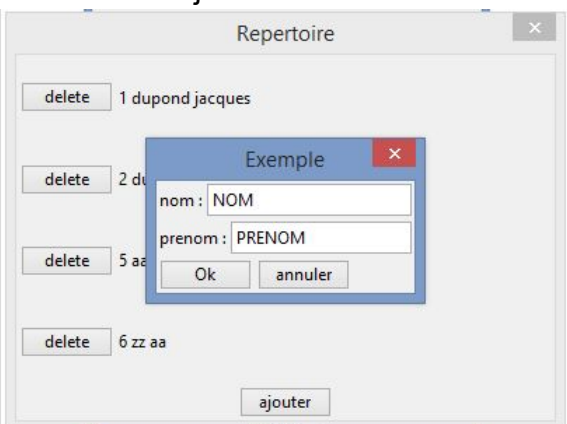
Bon … maintenant le répertoire …
Voici à quoi devrait ressembler l'interface :



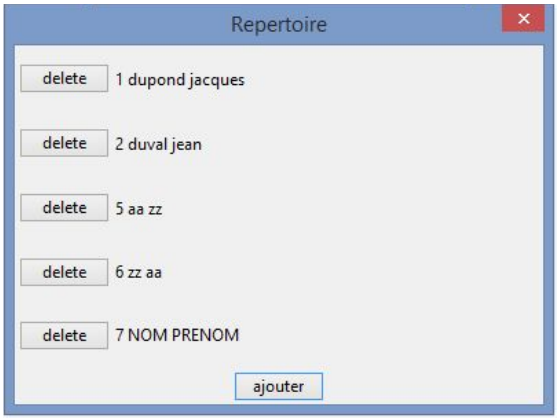
Après la suppression d'un élément :



L'interface d'ajout :



Et après l'ajout :



Le seul élément qu'il vous manque c'est l'objet message. Et vous aurez sûrement besoin des fonctions number->string et string-append. Le reste, c'est juste de bien réfléchir à l'enchaînement des actions (que doit-il se passer quand je clique sur ce bouton) …