PoP Série 5 niveau 0

Usage de GTKmm pour l'interface graphique utilisateur : Création d'une petite interface

Adapté du manuel de référence en-ligne de GTKmm 4

Exercice 1.(niveau 0) : **Layout et gestion d'evenement**

Dans cet exemple nous allons avoir une interaction entre les boutons et l'affichage.

1.1) Le programme principal définit un widget de la classe MyEvent que nous définissons pour cette série. Il suit le même modèle que les exemples de la semaine dernière.

```
#include "myevent.h"
#include <gtkmm/application.h>

int main(int argc, char** argv)
{
   auto app = Gtk::Application::create();

   return app->make_window_and_run<MyEvent>(argc,argv);
}
```

1.2) Passons à l'interface du module **myevent**. Il contient l'interface de deux classes :

- La classe **MyArea** que nous avons étendue avec 2 méthodes publiques **clear**() et **draw**(), et un attribut privé (booléen **empty**).

- La classe **Myevent** contient 2 attributs Button avec leur signal_handler. Elle possède aussi un attribut **MyArea** et 2 **Gt ::Box** pour le layout.

```
#ifndef GTKMM_MY_EVENT_H
#define GTKMM_MY_EVENT_H

#include <gtkmm/window.h>
#include <gtkmm/box.h>
#include <gtkmm/button.h>
#include <gtkmm/drawingarea.h>

class MyArea : public Gtk::DrawingArea
{
public:
      MyArea();
      virtual ~MyArea();

      void clear();
      void draw();

protected:
      void on_draw(const Cairo::RefPtr<Cairo::Context>& cr,
                   int width, int height);

private:
      bool empty;
};
```

```
class MyEvent : public Gtk::Window
{
public:
      MyEvent();

protected:
      //Button Signal handlers:
      void on_button_clicked_clear();
      void on_button_clicked_draw();

      MyArea m_Area;

      Gtk::Box m_Main_Box;
      Gtk::Box m_Buttons_Box;
      Gtk::Button m_Button_Clear;
      Gtk::Button m_Button_Draw;
};
#endif // GTKMM_MY_EVENT_H
```

L'implémentation myevent.cc définit d'abord **MyArea**. Une taille de 200 x 200 est imposée dans son constructeur.

- Ses méthodes publiques **clear**() et **draw**() sont utilisées pour le contrôle de l'interaction avec les boutons. Elles modifient l'attribut booléen **empty** qui agit comme une variable d'état ; cet attribut est utilisé par la méthode **on_draw()** pour savoir quel dessin il faut produire **:**
    - La méthode **clear** met le booleen **empty** à vrai et appelle **queue_draw()**
    - La méthode **draw** met le boolen **empty** à faux et appelle **queue_draw()**

- La méthode **queue_draw()** de GTKmm produit un *signal* qui va causer un appel du signal handler **on_draw().** Ainsi lorsque la méthode **on_draw()** est finalement appelée, elle utilise l'état actuel de l'attribut **empty** pour décider ce qu'il faut faire :
    - Si **empty** vaut *true* la méthode on_draw() ne dessine rien
    - Si **empty** vaut *false* la méthode on_draw() fait un dessin

Concernant **MyEvent**, le constructeur définit tout d'abord le texte des attributs **Button** et l'option d'organisation HORIZONTAL ou VERTICAL des attributs **Box** dans la liste d'initialisation. Ces attributs **Box** permettent de d'organiser la position relative des éléments de l'interface graphique. Si un attribut est initialisé avec l'option HORIZONTAL, tous les éléments qu'il contiendra seront sur une même ligne horizontale (resp. pour VERTICAL).

Ensuite on définit le titre de la fenêtre et (nouveau) on l'empêche de changer de taille pendant l'exécution avec la méthode **set_resizable(false).**

L'attribut **m_Main_Box** étant choisi comme conteneur de notre interface, c'est lui qui est fourni à la méthode **set_child(m_Main_Box)**. Ensuite on y ajoute avec la méthode **append** le widget de dessin **m_Area** puis le conteneur Box **m_Button_Box** : ces 2 éléments apparaitront l'un au-dessus de l'autre car **m_Main_Box** a été initialisé avec l'option VERTICAL.

On passe ensuite au remplissage du conteneur **m_Button_Box** avec la méthode **append**. On y ajoute les 2 boutons qui vont apparaitre horizontalement conformément à son initialisation.

Enfin on initialise les signal handlers des 2 boutons ; chaque **Button** fournit l'adresse d'une méthode de la classe **MyEvent**. Quand elle sera appelée, cette méthode appellera elle-même une méthode de l'attribut m_Area. Notez qu'il n'est pas possible d'indiquer directement l'adresse de la méthode de m_Area à l'initialisation du signal handler.

```
#include "myevent.h"
#include <cairomm/context.h>
#include <gtkmm/label.h>
#include <iostream>

constexpr int area_side(200);

MyArea::MyArea(): empty(false)
{
    set_content_width(area_side);
    set_content_height(area_side);

    set_draw_func(sigc::mem_fun(*this, &MyArea::on_draw));
}
MyArea::~MyArea()
{
}

void MyArea::clear()
{
   empty = true;
   queue_draw();
}
void MyArea::draw()
{
   empty = false;
   queue_draw();
}

void MyArea::on_draw(const Cairo::RefPtr<Cairo::Context>& cr,
                     int width, int height)
{
   if(not empty)
   {
      // coordinates for the center of the window
      int xc(width / 2), yc(height / 2);

      cr->set_line_width(10.0);

      // draw red lines out from the center of the window
      cr->set_source_rgb(0.8, 0.0, 0.0);
      cr->move_to(0, 0);
      cr->line_to(xc, yc);
      cr->line_to(0, height);
      cr->move_to(xc, yc);
      cr->line_to(width, yc);
      cr->stroke();
   }
   else
   {
      std::cout << "Empty !" << std::endl;
   }
}
```

```
MyEvent::MyEvent():
    m_Main_Box(Gtk::Orientation::VERTICAL, 0),
    m_Buttons_Box(Gtk::Orientation::HORIZONTAL, 2),
    m_Button_Clear("Clear"),
    m_Button_Draw("Draw")
{
    set_title("DrawingArea");
    set_resizable(false);
    set_child(m_Main_Box);

    m_Main_Box.append(m_Area);
    m_Main_Box.append(m_Buttons_Box);

    m_Buttons_Box.append(m_Button_Clear);
    m_Buttons_Box.append(m_Button_Draw);

    m_Button_Clear.signal_clicked().connect(
      sigc::mem_fun(*this, &MyEvent::on_button_clicked_clear));

    m_Button_Draw.signal_clicked().connect(
      sigc::mem_fun(*this, &MyEvent::on_button_clicked_draw));
}

void MyEvent::on_button_clicked_clear()
{
    std::cout << "Clear" << std::endl;
    m_Area.clear();
}

void MyEvent::on_button_clicked_draw()
{
    std::cout << "Draw" << std::endl;
    m_Area.draw();
}
```

Activité :

1) inversez les options HORIZONTAL et VERTICAL pour les Box et constatez le résultat
2) En plus, changez aussi l'ordre des éléments dans m_Main_box, c'est-à-dire ajoutez d'abord m_Button_box avant m_Area