PoP Série 5

H1 : Usage de GTKmm 4 pour l'interface graphique utilisateur

H2 : MOOC MOOC Introduction à la programmation orientée objet (en C++)

Série MOOC5: Polymorphisme (exercice 18 hors programme du cours ; remplacé par la suite du « type paramétré » en page 2 => passage à une hiérarchie de classes pour Form)

Usage de GTKmm 4 pour l'interface graphique utilisateur

Exercice 1.1 (niveau 0) : **usage de boutons => variable d'état => impact sur le dessin**
Exemple 1.2 (niveau 0) : **usage d'un timer (dés)-activé avec des boutons**
Exemple 1.3 (niveau 0) : **opérations de lecture/écriture de fichier activées avec des boutons**

Exercice 2 (niveau 1) : **Coordinates conversion from the Model space to the Application's window width and height / preventing distortion (in English)**

Let's suppose that we are designing a X red cross within a Model space respecting the conventions from the course: X is positive rightwards and Y is positive upwards. The X shape has to be built with two orthogonal lines : from (-50,-50) to (50,50) and from (-50, 50) to (50, -50).

a) Extend the example from week 5 (GTKddrawingarea) to create an Application window of size (300, 200) that draws the red cross ("X") with the following Model framing of [-150, 150] along X and [-100, 100] along Y in the Model space. Set the line width to 10 to ensure visibility. Use the approach presented in last week course with the calls to the methods **translate** and **scale** before making any calls to **move_to** and **line_to** for drawing the X shape. Verify that the cross is displayed in the center of the graphic window. Then if you manually change the size of the window[1] the X shape should deform so that it is always inside the new window size. However the 2 lines may not be orthogonal anymore.

b) Here we want to have the same initial state = the X cross is centered in the window AND we want the 2 lines to remain always orthogonal. Preventing a distortion means that we must enforce *the same scaling factor along X and Y* (in absolute value). Generalize the solution presented in week5 course so that we see the full X in the center of the window and without introducing any distortion (the 2 branches of the X remain orthogonal whatever the shape of the window).

[1] on_draw() from MyArea class is called each time you modify the size of your window (XXXX à modifier)

Polymorphisme (exercice 18 hors programme du cours ; remplacé par la suite du « type paramétré » => passage à une hiérarchie de classes pour Form)

Exercice 3 (niveau 1) : **convertir le type paramétré Form de la Série3 en une hiérarchie de classes tirant parti du Polymorphisme**

Se reporter à l'exercice 1.3 de la Série PoP_s3 pour les détails du contexte. Une hiérarchie à deux niveaux est suffisante : on conserve le nom de **Form** pour la superclasse qui va devenir une classe abstraite parce que certaines méthodes seront virtuelles pures (listées plus bas). A partir de la superclasse Form, on demande de dériver 3 classes de **Circle**, **Square** et **Rectangle**.

Dans cet exercice on pose que l'ensemble des classes de la hiérarchie sont dans un même module. Le type **Bbox** (pour *Bounding Box*) devient synonyme de **Rectangle** avec les pré-déclarations suivantes qui doivent être faites au début du fichier **form.h** avant de déclarer les classes :

```
class Rectangle ;
typedef Bbox Rectangle ;
```

a) La superclasse **Form** contient les attributs x et y d'un point de référence qui peut être son centre où un autre point remarquable. Avec la hiérarchie de classes, on peut maintenant utiliser des noms d'attributs plus explicites dans les classes dérivées spécialisées. Par exemple : rayon, side, largeur, hauteur, width, height.

Définissez l'interface de cette hiérarchie de classes (dans l'unique fichier form.h) :

- **Form** :
  - Offrir un seul constructeur qui définit des valeurs par défaut pour les attributs **protected** x et y .
  - Ajouter un destructeur virtuel avec un corps vide comme définition.
  - Conserver les accesseurs pour x et pour y
  - Rendre *virtuel pur* les 2 méthodes suivantes : **affiche**, **get_bbox**

- Pour les 3 classes dérivées, préciser leur constructeur et déclarer les méthodes virtuelles pures héritées de la classe parente.
  - La classe **Rectangle** déclare en plus les méthodes **merge_bbox** et **intersect_bbox**.

Puis définissez les méthodes dans l'implémentation form.cc en adaptant le code de la solution de l'exercice 1.3 de la Série3.

b) Intégrer le nouveau module **form** avec **formset** et **test**. Vérifier en lisant les fichiers de configuration