INFO-F-202
Langages de programmation 2
Université libre de Bruxelles
Laboratoire 10

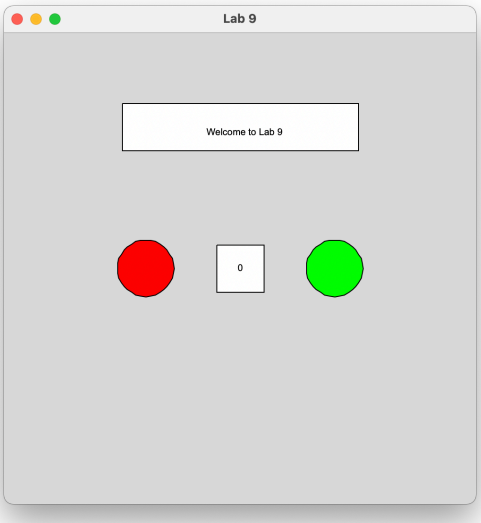
Justin Dallant John Iacono
Yannick Molinghen Alexis Reynouard

1 But

Comprendre et utiliser le patron de conception *observer*.

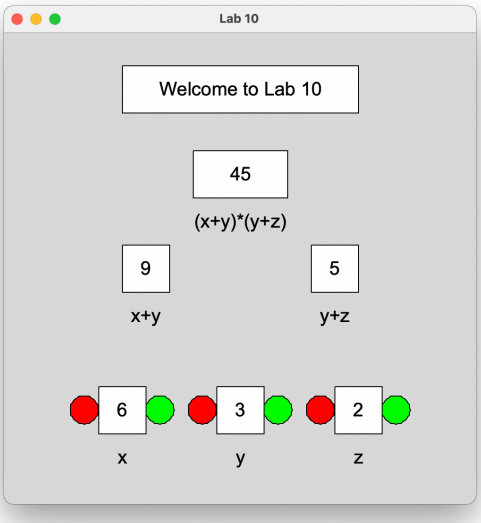
2 Sommaire

Dans le dernier laboratoire, vous deviez coder quelque chose qui ressemblait à ceci, où vous cliquiez sur les cercles rouges et verts pour incrémenter ou décrémenter le nombre.



Cela a été fait en demandant aux cercles d'envoyer des messages au carré avec le décompte.

Ici, vous devriez développer votre solution précédente (ou le code de départ que nous vous donnons) pour créer ce qui suit :



Les principaux ajouts sont les deux sommes et un produit. Vous devez utiliser le patron de conception *Observer* pour les implémenter. Rappelez-vous que dans le patron de conception *Observer*, vous avez un sujet et un observateur, où l'observateur veut être averti chaque fois que le sujet change. Cela a été fait avec le code suivant :

```
struct Observer{
  virtual void subjectChanged()=0;
};

class Subject{
  vector <Observer *> observers;
public:
  void registerObserver(Observer *observer){
    observers.push_back(observer);
  }
  void removeObserver(Observer *observer){
    remove(begin(observers),end(observers),observer);
  }
  void notifyObservers() const {
    for (auto &observer:observers)
      observer->subjectChanged();
  }
};
```

Tout ce qui sera observé devrait hériter du `Sujet`, et tout ce qui observera devrait hériter de `Observer`. L'observateur appellera `registerObserver` sur chaque sujet qui l'intéresse pour indiquer qu'il souhaite être informé des changements. Chaque sous-classe de `Subject` a juste besoin d'appeler `notifyObservers` chaque fois qu'il y a un changement qui devrait entraîner la notification des observateurs.

Notez que les deux rectangles avec les sommes observent chacun deux des compteurs en bas, et sont observés par le rectangle en haut avec le produit !

3 Les Classes : Conseils d'organisation

Vous aurez besoin de plusieurs classes. Aucune d'entre elles ne doit être modélisée.

— Classes abstraites. Toutes proviennent de la solution de la semaine dernière plus l'ajout de Subject et Observer.

Clickable

Drawable

MessageReceiver

Observer

Subject

SendMessessageWhenClicked

— Classes de dessin. Vous n'avez pas besoin de les changer.

Text

Rectangle

Circle

— Les classes qui combinent des fonctionnalités. C'est là que se trouve tout votre travail.

TextRectangle : Un rectangle avec du texte. Par exemple "Welcome to lab 10."

IntRectangle : Un rectangle avec un nombre.

CounterRectangle : Un rectangle avec un numéro qui répond aux messages.

SumRectangle : Un rectangle stockant la somme de ce qu'il observe.

MultRectangle : Un rectangle stockant le produit de ce qu'il observe.

ClickableCircle : Un cercle qui envoie des messages lorsqu'on clique dessus (du dernier laboratoire).

— Canvas et MainWindow

Canvas : Le code de démarrage de cette semaine mettra toutes les cases aux bons endroits sur l'écran.

MainWindow : Ne change pas

Voici une façon possible d'organiser les classes.