Projet POO, Dashboard

Romane Sallio *Pierre Fontaine*

Romane Sallio, Pierre Fontaine
UPPA, Licence, Informatique

1er mai 2017

Sommaire

Introduction

Dashboard est le projet que nous avons conçu en C++ avec le paradigme Objet.
Nous avons utilisé le Framework QT pour développer l'interface utilisateur.
Son objectif est simple : permettre à l'utilisateur d'avoir les éléments essentiels à porté de click.

Pourquoi QT

Paradigme Objet

QT est écrit en C++ et est implémenté selon le paradigme objet. Chacun des composants réfère à une classe particulière qui peut ou non dérivé d'une autre classe mère.

Pourquoi QT

JS LIKE

Lors de l'utilisation d'un nouveau *Framework*, une partie crucial du temps est concacré à l'étude du fonctionnement de celui ci. Il semblait évident qu'après avoir étudier le *JavaScript*, le *QT* qui partage la même philosophie de la gestion d'évènement *(Async/Sync)* serait plus digeste.

Template

Utilité ?

La technique consiste à paramétrer le Type. Ainsi l'utilisateur de la classe pourra utiliser le type *int, shorting, string ....*

Utilisé dans *List.h*

Pourquoi ?

1. Créer une liste de n'importe quoi
2. Container important

attention

L'implémentation ne peut être séparée de la signature.

Template

Code


```
template <class T>
class List{...};
```

Forme canonique de coplien

Pourquoi

Il est interressant dans la conception de *container* comme des listes, des sets ... qu'ils puissent s'affecter entre eux, se construire à partir d'un modèle déjà existant.

C'est pourquoi nous intégrons ce modèle dans quelques classes comme :

1. list.h
2. abstractmesureunite.h

Forme canonique de coplien

code 1


```
class AbstractMesureUnite{
protected:
  double _value;
public:
  AbstractMesureUnite();
  AbstractMesureUnite(double);
  AbstractMesureUnite(const AbstractMesureUnite&);
  ~AbstractMesureUnite();
  AbstractMesureUnite &operator=(const AbstractMesureUnite&);
  ...
};
```

Forme canonique de coplien

code 2


```
template <class T>
class List{
protected:
    struct cellule{
        cellule *suivant;
        T valeur;
    };
    typedef cellule* liste;
    liste _l;
public:
    List();
    ~List();
    List(const List<T> &l);
    List<T> operator=(const List<T>);
    ...
};
```

Classe Abstraite

Interet

Factorisation du code :
L'un des objectifs majeur lorsque l'on developpe du code c'est de s'assurer qu'on ne se répetera pas.

Creer une signature que l'on implémente pas :

1. les sous classes fourniront l'implémenteront forcément
2. les sous classes auront chacune une implémentation différente.
3. cette classe abstraite ne sera jamais instancié.

Classe Abstraite

QT

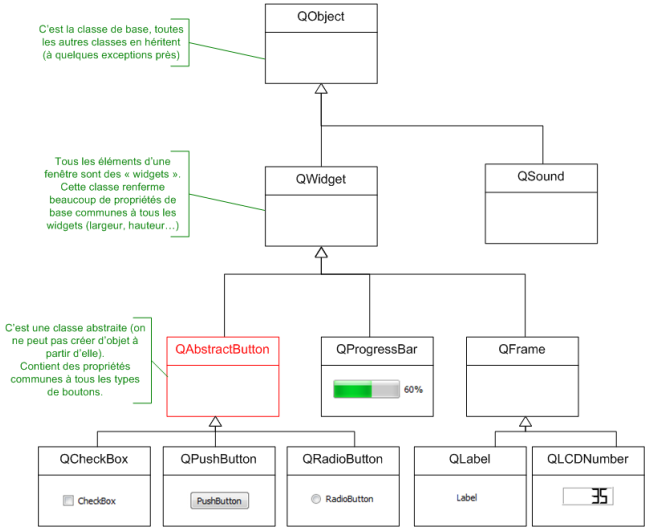


FIGURE: Qt Diagram openclassroom

Classe Abstraite

Code 1


```
class AbstractMesureUnite{
protected:
  double _value;
public:
  AbstractMesureUnite();
  AbstractMesureUnite(double);
  AbstractMesureUnite(const AbstractMesureUnite&);
  ~AbstractMesureUnite();
  AbstractMesureUnite &operator=(const
  ↪ AbstractMesureUnite&);
  ...
  virtual void afficher(ostream &flux) const = 0;

  friend ostream& operator<<(ostream& flux,const
  ↪ AbstractMesureUnite& u){
    u.afficher(flux);
    return flux;
  }
};
```

Classe Abstraite

Code 2


```
class Temperature : public AbstractMesureUnite{
public:
  Temperature();
  Temperature(double);
  virtual void afficher(ostream &flux) const = 0;
  virtual double getCelsius()const = 0;
  virtual double getFahrenheit()const = 0;
  virtual double getKelvin()const = 0;
};
```

Heritage

QT

Dérivation des composants :

1. Chaque composant du framework QT est une classe
2. Créer un composant spécifique se fait en dérivant un composant général

tips

Il est courant que les classes les plus généralistes ne soient pas instanciables, ce sont des classes *abstraites*.

Heritage

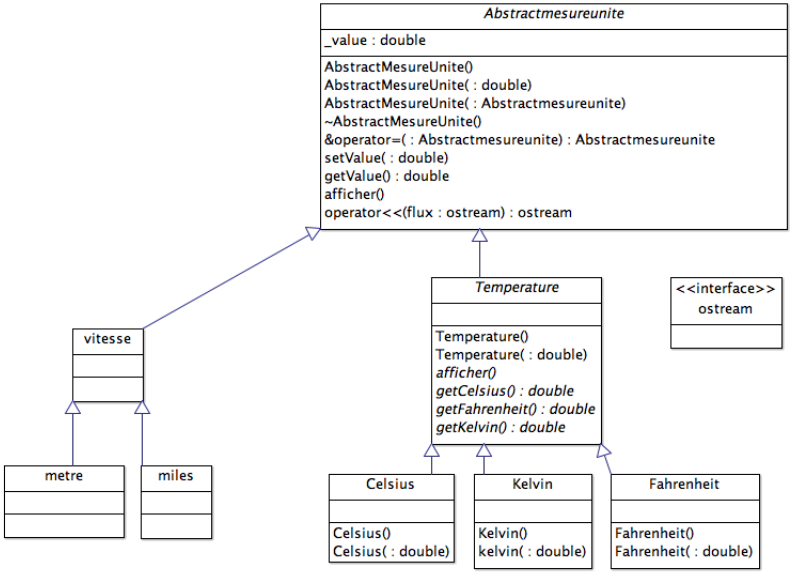
Hériter un composant QT


```
#include <QWidget>

class Module : public QWidget{
    Q_OBJECT
protected:
  //méthodes protégées.
public:
  explicit Module(QWidget *parent = 0,double h = 300,
  ↪  double w = 400);
  //méthodes publiques

signals:
  //signals
public slots:
  //slots
};
```

Heritage

En dehors de QT

- introduction
- Pourquoi QT
- Spécifications techniques du code
- conclusion



Classe Complexe

interet ?

Concept :
Une classe peut utiliser l'instance d'une autre classe.

Exemple d'une vue :
Prenons pour exemple notre *widget Météo*. Celui ci a besoin d'afficher des informations, donc qui fournit l'information ?
On instancie une classe *Meteodata* qui va servir d'intermediaire entre la vue et le modèle.

Classe Complexe

code


```
class MeteoJour : public QWidget{
    Q_OBJECT
protected:
    ...
    //model
    MeteoData *_data;
public:
    explicit MeteoJour(int j,MeteoData *data,QWidget
    ↪ *parent = 0);
signals:
    ...
public slots:
    ...
};
```

Conclusion

- code plus facile à gérer en collaboration
- code plus facile à maintenir
- évolution envisageable
- ré-utilisation de code
- plus efficace pour IHM