Langage C/C++

Cours 5/5 : Surcouche orientée objet

Hubert Godfroy

17 décembre 2014

La dernière fois…

On a vu…

- tout ce que C sait faire (!)
- Paradigme procédural
- Structuré autour de fonctions

C++ ajoute une couche objet à C

Plan

Programmation orientée objet

Nouveautés C++

Application : Interfaces graphiques

Plan

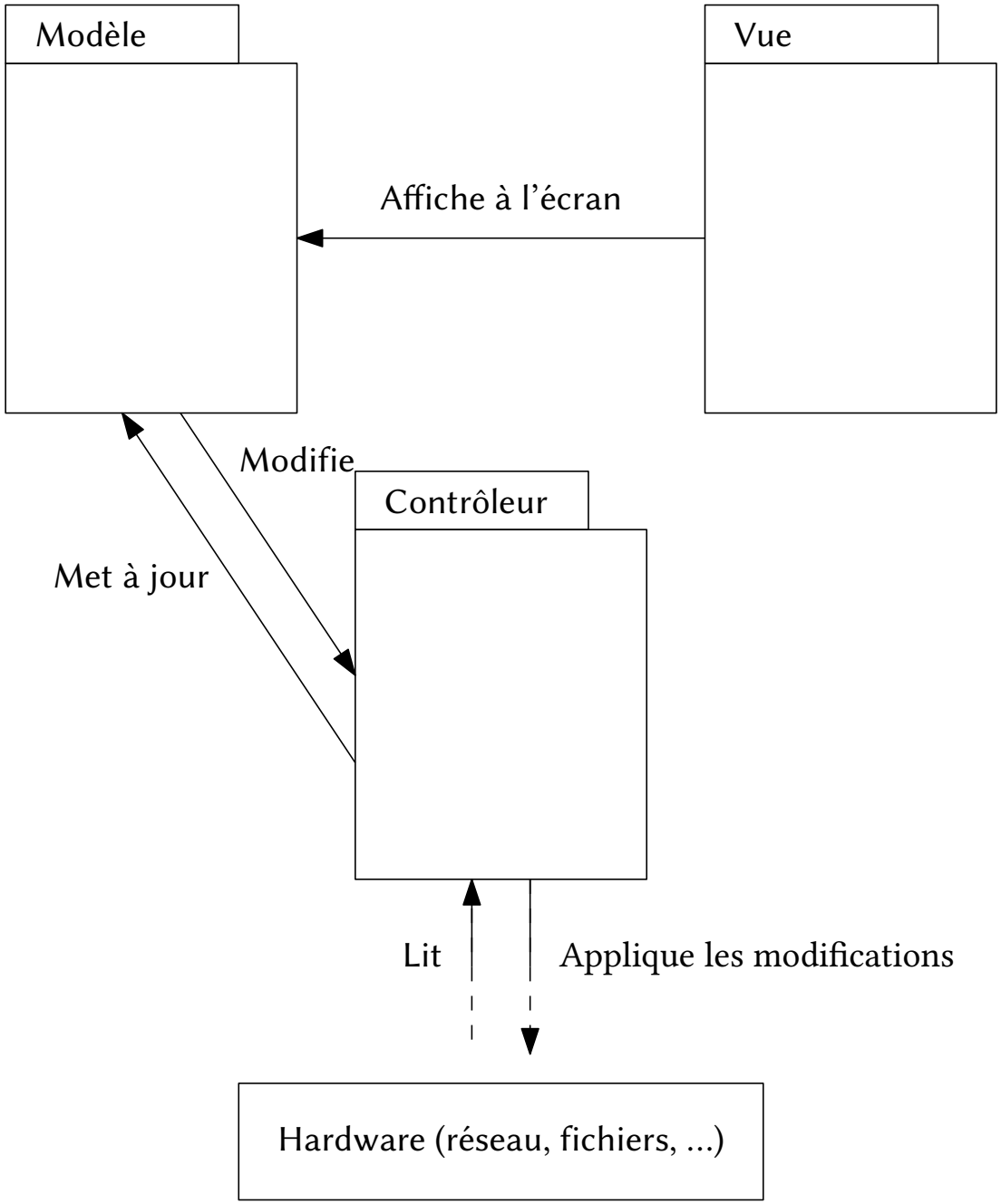
Idée de l'objet

- Associer à chaque objet des fonctions
- Permettre la réflexivité (`this`)

Le but est de cacher les détails d'implémentation

$\Rightarrow$ On se concentre sur l'architecture du logiciel.

Exemple


Modèle
Vue
Affiche à l'écran
Modifie
Contrôleur
Met à jour
Lit
Applique les modifications
Hardware (réseau, fichiers, ...)

Syntaxe

Définition

```
class Base {
  public:
    int champ;
    Base();
    Base(int n);
    void afficher();
    void initial();
};
```

Syntaxe

Définition

```
class Base {
  public:
    int champ;
    Base();
    Base(int n);
    void afficher();
    void initial();
};
```

Déclaration

```
Base::Base() {
    cout << "construction d'une instance de Base\n"
        ;
}

Base::Base(int n){
    cout << "construction d'une instance de Base(
        int n)\n";
}

void Base::afficher(){
    cout << "fonction afficher() dans la classe
        Base\n";
}

void Base::initial(){
    cout << "fonction initial() dans la classe Base
        \n";
}
```

Syntaxe

Constructeur par défaut

- `Base::Base()` est le type du constructeur par défaut.
- Il peut être redéfini.

Syntaxe

Constructeur par défaut

- `Base::Base()` est le type du constructeur par défaut.
- Il peut être redéfini.

Utilisation statique

Syntaxe

Constructeur par défaut

- `Base::Base()` est le type du constructeur par défaut.
- Il peut être redéfini.

Utilisation statique (définition sur la pile)

Syntaxe

Constructeur par défaut

- `Base::Base()` est le type du constructeur par défaut.
- Il peut être redéfini.

Utilisation statique (définition sur la pile)

```
Base aBase1;
Base aBase2(10);
```

Utilisation dynamique

Syntaxe

Constructeur par défaut

- `Base::Base()` est le type du constructeur par défaut.
- Il peut être redéfini.

Utilisation statique (définition sur la pile)

```
Base aBase1;
Base aBase2(10);
```

Utilisation dynamique (définition dans le tas)

Syntaxe

Constructeur par défaut

- `Base::Base()` est le type du constructeur par défaut.
- Il peut être redéfini.

Utilisation statique (définition sur la pile)

```
Base aBase1;
Base aBase2(10);
```

Utilisation dynamique (définition dans le tas)

```
Base* aBase1 = new Base();
Base* aBase2 = new Base(10);
```

Plan

Destruction

- On utilise `delete` sur les objets C++.
- Équivalent de `free` sur les objets C.
- Fait appel au destructeur de la classe.
- Le destructeur ne peut pas être surchargé.

Destruction

- On utilise `delete` sur les objets C++.
- Équivalent de `free` sur les objets C.
- Fait appel au destructeur de la classe.
- Le destructeur ne peut pas être surchargé.

Exemple

```
class Derive : public Base {
  public:
    ...
    ~Derive();
};

Base::~Base(){
    cout << "destruction de Base\n";
  }
```

Liaison dynamique *vs* statique

Rappel

- L'adjectif statique désigne un phénomène se déroulant lors de la compilation.
- L'adjectif dynamique désigne un phénomène se déroulant lors de l'exécution.

Liaison dynamique *vs* statique

Rappel

- L'adjectif statique désigne un phénomène se déroulant lors de la compilation.
- L'adjectif dynamique désigne un phénomène se déroulant lors de l'exécution.

Exemple

- Allocation statique *vs* dynamique

Liaison dynamique *vs* statique

Rappel

- L'adjectif statique désigne un phénomène se déroulant lors de la compilation.
- L'adjectif dynamique désigne un phénomène se déroulant lors de l'exécution.

Exemple

- Allocation statique *vs* dynamique
- Pour les liaisons aussi !

Liaison en C

- On sait ***sans exécuter le programme*** quelle fonctions sera utilisée

Liaison en C

- On sait ***sans exécuter le programme*** quelle fonctions sera utilisée

Exemple

```
void main(){
  ...
  afficher();
  ...
}
```

Liaison en C

- On sait ***sans exécuter le programme*** quelle fonctions sera utilisée

Exemple

```
void main(){
  ...
  afficher();   ← connue sans exécution
  ...
}
```

$\Rightarrow$ Les liaisons sont statiques.

Liaison en C++

- On ne sait pas dans le cas général (Comme en Java)

Exemple (JAVA)

```
public class Base {
  public void afficher(){
    System.out.println("Base.afficher()");
  }
}
```

Liaison en C++

- On ne sait pas dans le cas général (Comme en Java)

Exemple (JAVA)

```
public class Base {
  public void afficher(){
    System.out.println("Base.afficher()");
  }
}
public class Derive extends Base {
  public void afficher(){
    System.out.println("Derive.afficher()");
  }
}
```

Liaison en C++

- On ne sait pas dans le cas général (Comme en Java)

Exemple (JAVA)

```
public class Base {
  public void afficher(){
    System.out.println("Base.afficher()");
  }
}
public class Derive extends Base {
  public void afficher(){
    System.out.println("Derive.afficher()");
  }
}
public static void main(String[] args) {
  Base truc;
  if ((int)(Math.random()*2) == 1){
    truc = new Base();
  } else {
    truc = new Derive();
  }
  truc.afficher();
}
```

$\Rightarrow$ Les liaisons sont dynamiques.

Implémentation en C++

Fonctions virtuelles

- On doit spécifier si la liaison doit être faite statiquement ou dynamiquement.
- On utilise le mot clef `virtual` pour imposer une liaison dynamique (cf exercices).
- Une fonction est virtuelle pure si elle n'a pas d'implémentation :

```
virtual void afficher()=0;
```

Implémentation en C++

Fonctions virtuelles

- On doit spécifier si la liaison doit être faite statiquement ou dynamiquement.
- On utilise le mot clef `virtual` pour imposer une liaison dynamique (cf exercices).
- Une fonction est virtuelle pure si elle n'a pas d'implémentation :

```
virtual void afficher()=0;
```

Classes abstraites

- C'est une classe comportant une fonction virtuelle pure.
- Ne peux pas être instanciée.
- Une classe est abstraite pure si toute ses méthodes sont virtuelles pures.

Méthode de résolution

- statique : en fonction de la classe de l'objet "danslecode" (lors de la compilation)
- dynamique : en fonction de la classe de l'objet lors de l'exécution

(cf exercices)

Héritage

Définition

```
class Derive: Base {
   ...
}
```

Héritage

Définition

```
class Derive: Base {
  ...
}
```

Construction

- Commence par construire la classe mère
- Fait par défaut appel au constructeur par défaut de la classe mère.
- On peut spécifier le constructeur que l'on souhaite :

```
Derive::Derive(int n) : Base(n){...};
```

Héritage

Liaison (rappel)

- statique : en fonction de la classe de l'objet "dans le code" (lors de la compilation)
- dynamique : en fonction de la classe de l'objet lors de l'exécution.

Héritage

Liaison (rappel)

- statique : en fonction de la classe de l'objet "dans le code" (lors de la compilation)
- dynamique : en fonction de la classe de l'objet lors de l'exécution.

Destruction

- Ordre d'appel inverse de celui des constructeurs : commence par détruire la classe fille, puis la classe mêre.
- Attention au type de liaison du destructeur (cf exercices).

Patrons

- Une classe peut-être paramétrée par un type T (interface en JAVA) :

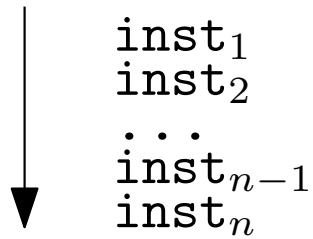
```
template<typename T> class Classe {
  T champ;
  T function(T val, int n);
}
```

- Utilisation :

```
Classe<int> obj;
```

Plan

Style habituel



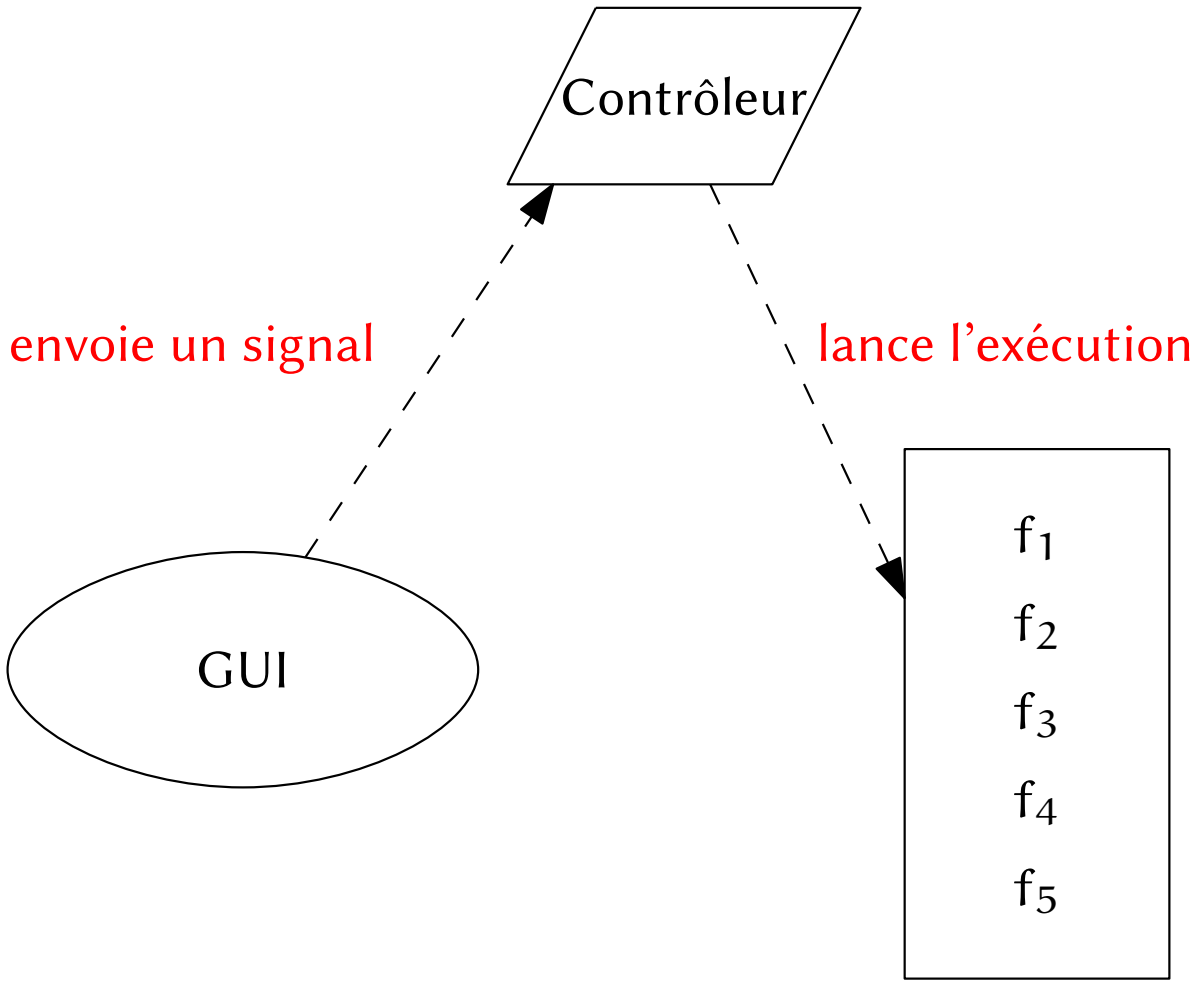
- Lecture linéaire
- De haut en bas
- On voit “tout ce qui se passe”.
- Rythmé par les appels de fonction/méthodes

Programmation par événements

- Envoi de signaux
- Un contrôleur s'occupe d'envoyer les signaux au bon destinataire

Programmation par événements

- Envoi de signaux
- Un contrôleur s'occupe d'envoyer les signaux au bon destinataire



Modèle - Vue - Contrôleur

Signaux

Ils sont déclenchés par des instances extérieurs au programme (clique de l'utilisateur, branchement d'un périphérique, ...).

Modèle - Vue - Contrôleur

Signaux

Ils sont déclenchés par des instances extérieurs au programme (clique de l'utilisateur, branchement d'un périphérique, ...).

Slot

Ils réceptionnent les signaux. Ce sont en général des méthodes de la classe modélisant la vue.

Modèle - Vue - Contrôleur

Signaux

Ils sont déclenchés par des instances extérieurs au programme (clique de l'utilisateur, branchement d'un périphérique, ...).

Slot

Ils réceptionnent les signaux. Ce sont en général des méthodes de la classe modélisant la vue.

Vue

- Compose l'interface utilisateur
- Différents éléments d'interaction (bouton, champs de texte, labels, ...)

En Qt

```
class Fenetre : public QWidget {
    Fenetre();

    QPushButton* bouton;
};
```

En Qt

```
class Fenetre : public QWidget {
    Fenetre();

    QPushButton* bouton;
};


Fenetre::Fenetre() {
    setFixedSize(300, 150);
    bouton = new QPushButton("Quitter", this);
    bouton->move(110, 50);

    QObject::connect(
      bouton,
      SIGNAL(clicked()),
      qApp,
      SLOT(quit())
    );
}
```

En Qt

```
class Fenetre : public QWidget {
    Fenetre();

    QPushButton* bouton;
};


Fenetre::Fenetre() {
    setFixedSize(300, 150);
    bouton = new QPushButton("Quitter", this);
    bouton->move(110, 50);

    QObject::connect(
      bouton,                    ← émetteur du signal
      SIGNAL(clicked()),
      qApp,
      SLOT(quit())
    );
}
```

En Qt

```
class Fenetre : public QWidget {
    Fenetre();

    QPushButton* bouton;
};


Fenetre::Fenetre() {
    setFixedSize(300, 150);
    bouton = new QPushButton("Quitter", this);
    bouton->move(110, 50);

    QObject::connect(
      bouton,                  ← émetteur du signal
      SIGNAL(clicked()),       ←signal
      qApp,
      SLOT(quit())
    );
}
```

En Qt

```
class Fenetre : public QWidget {
    Fenetre();

    QPushButton* bouton;
};


Fenetre::Fenetre() {
    setFixedSize(300, 150);
    bouton = new QPushButton("Quitter", this);
    bouton->move(110, 50);

    QObject::connect(
      bouton,                  ← émetteur du signal
      SIGNAL(clicked()),       ←signal
      qApp,                    ← récepteur du signal
      SLOT(quit())
    );
}
```

En Qt

```
class Fenetre : public QWidget {
    Fenetre();

    QPushButton* bouton;
};


Fenetre::Fenetre() {
    setFixedSize(300, 150);
    bouton = new QPushButton("Quitter", this);
    bouton->move(110, 50);

    QObject::connect(
      bouton,                  ← émetteur du signal
      SIGNAL(clicked()),       ←signal
      qApp,                    ← récepteur du signal
      SLOT(quit())             ← slot d'accueil
    );
}
```