Programmation Orientée Objets 2
L3 MI
Cours 5

Rémi Watrigant[1]

[1] basé sur le cours de Marianne Huchard

Sommaire du cours 5

Généricité:

Définition : Généricité paramétrique

Définition : Généricité paramétrique

Possibilité d'ennoncer des descriptions de classes ou de fonctions dans lesquelles certains élements (les paramètres) restent formels.

- "Modèle de classe"
- "Modèle de fonction"
- en C++: "template"
- Paramètres = types, classes, fonctions, valeurs

Exemple

Exemple (Modèle de Pile)

`Pile` pour `int`, `String`, `Voiture` ....
⇒ *`modèle de pile` paramétré par le type T des éléments stockés*

Exemple (Modèle de fonction de recherche)

`fonction de recherche` d'un élément dans un tableau
⇒ *`modèle de fonction de recherche` d'un élément d'un type formel T dans un tableau d'éléments de type T.*

Exemple (Modèle d'association)

Dictionnaire, tables de hachages... ⇒ *`modèle d'association`, paramétré par le type de la clef et le type de la valeur.*

La généricité dans les langages de programmation

- Existe notamment en Ada, Eiffel, C++, et Java 5 (tiger)
- Sans généricité paramétrique, des erreurs potentielles à l'exécution dues à :
  - des contrôles de types à l'exécution **plus nombreux**
  - des coercitions de type (`dynamic_cast`) **plus nombreux**
- En C++, la bibliothèque des collections de données (*Standard Template Library*) est implémentée à l'aide de modèles de classes

Modèles de fonctions : Première solution

Modèles de fonctions : Première solution

```
typedef string T ;
//typedef int T;
//typedef Personne T;

void affiche( ostream& os, T tableau[], int taille) {
    for (int i = 0 ; i < taille ; i++)
        os << tableau[i] ;
    os << endl;
}

int main( int argc, char** argv ) {
    string test[3] = {"toto", "titi", "tutu"};
    affiche(cout, test, 3);

    return 0;
}
```

Avec généricité

```
template<typename T>
void affiche( ostream& os, T tableau[], int taille) {
    for (int i = 0 ; i < taille ; i++)
        os << tableau[i] ;
    os << endl;
}
```

Dans l'en-tête de la fonction :

- mot clé *template* signale une déclaration de modèle
- *typename* introduit un paramètre formel (ici : type T)

Avec généricité

```
template<typename T>
void affiche( ostream& os, T tableau[], int taille) {
    for (int i = 0 ; i < taille ; i++)
        os << tableau[i] ;
    os << endl;
}

int main( int argc, char** argv ) {
    string test[3] = {"toto", "titi", "tutu"};

    //explicitation du paramètre
    affiche<string>(cout, test, 3);

    //ou on laisse le compilateur choisir
    affiche(cout, test, 3);

    return 0;
}
```

Laisser le compilateur choisir = ok si pas d'ambiguïté !

Modèle de fonctions

Possibilité de déclarer d'autres paramètres de modèle:

- paramètres du modèle : type des éléments et taille du tableau
- paramètre de fonction : le flux de sortie et le tableau

```
template<typename T, int taille>
void affiche( ostream& os, T tableau[]) {
    for (int i = 0 ; i < taille ; i++)
        os << tableau[i] ;
    os << endl;
}

int main( int argc, char** argv ) {
    string test[3] = {"toto", "titi", "tutu"};

    //ici : explicitation obligatoire
    affiche<string, 3>(cout, test, 3);


    return 0;
}
```

Modèle de fonctions

Possibilité de déclarer d'autres paramètres de modèle:

- paramètres du modèle : type des éléments du tableau et type de sortie
- paramètre de fonction : le tableau et sa taille

```
template<typename T, typename S>
void moyenne(T tableau[], int taille) {
S somme = 0;
    for (int i = 0 ; i < taille ; i++)
        somme += tableau[i];
    return somme/taille;
}

int main( int argc, char** argv ) {
    int test[3] = {12, 14, 8, 17};

    cout << moyenne<int, double>(test, 4) << endl; // affiche 12.75
    cout << moyenne<int, int>(test, 4) << endl; // affiche 12
    cout << moyenne(test, 4) << endl; // ...?

    return 0;
}
```

Expression de contraintes

Contrairement à d'autres langages, notamment UML, Eiffel, Java 5 (et versions suivantes), C++ ne permet pas d'exprimer de contraintes sur les paramètres de modèles.

Dans les modèles de fonction précédents, l'opérateur `operator<<(ostream&,const TypeElt&)` doit exister pour que les instanciations soient compilables, mais il n'y a pas de moyen d'écrire cette contrainte dans le code source.

Modèles de classes

Modèles de classes

```
template<typename TypeCle, typename TypeValeur>
class Assoc{
private:
  TypeCle cle; TypeValeur valeur;
public:
  Assoc();
  Assoc(TypeCle, TypeValeur);
  virtual ~Assoc ();
  virtual TypeCle getCle()const;
  virtual void setCle(TypeCle);
  virtual TypeValeur getValeur()const;
  virtual void setValeur(TypeValeur);
  virtual void affiche(ostream&)const;
};

template<typename TypeCle, typename TypeValeur>
ostream& operator<<(ostream&, const Assoc<TypeCle,TypeValeur>&);
```

Définition du modèle de classe Assoc

```
#include "Assoc.h"

template<typename TypeCle, typename TypeValeur>
Assoc<TypeCle,TypeValeur>::Assoc()   {}

template<typename TypeCle, typename TypeValeur>
Assoc<TypeCle,TypeValeur>::Assoc(TypeCle c, TypeValeur v)
:cle(c), valeur(v)    {}

template<typename TypeCle, typename TypeValeur>
Assoc<TypeCle,TypeValeur>::~Assoc ()   {}\\

template<typename TypeCle, typename TypeValeur>
TypeCle Assoc<TypeCle,TypeValeur>::getCle()const   {return cle;}

template<typename TypeCle, typename TypeValeur>
void Assoc<TypeCle,TypeValeur>::setCle(TypeCle c)   {cle=c;}
```

```
template<typename TypeCle, typename TypeValeur>
TypeValeur Assoc<TypeCle,TypeValeur>::getValeur()const {return valeur;}

template<typename TypeCle, typename TypeValeur>
void Assoc<TypeCle,TypeValeur>::setValeur(TypeValeur v)   {valeur=v;}

template<typename TypeCle, typename TypeValeur>
void Assoc<TypeCle,TypeValeur>::affiche(ostream &os) const
  {os <<getCle() << ", " <<getValeur();}

template<typename TypeCle, typename TypeValeur>
ostream& operator<<(ostream& os, const Assoc<TypeCle,TypeValeur>& a)
  {a.affiche(os); return os;}
```

Modèles de classes

/!\ pas de fichiers .h et .cpp séparés /!\

Instanciation

Cas de la création d'un objet

```
// ................. mainAssoc1.cpp ........................
#include <iostream>
#include <string>
#include "Assoc.h"

int main()
{
  Assoc<char, int> a( 'y', 5 );
  Assoc<string,string>* pass = new Assoc<string,string>("citron","jaune");
}
```

Héritage

Héritage

Exemple (un modèle dérive d'un autre modèle)

```
template <typename T>
class Conteneur{
    ...
};

template <typename T>
class Liste : public virtual Conteneur<T>{};
```

Héritage

Exemple (un modèle dérive d'une classe)

```
class Graphe{
   ...
};

template <typename TypeEtiquette>
class GrapheEtiquete : public virtual Graphe{};
```

Héritage

Exemple (une classe dérive d'un modèle)

```
template<typename cle, typename valeur>
class Assoc{
   ...
};


class Point : virtual public Assoc<int,int>{};
```