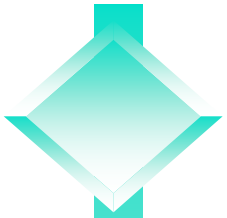
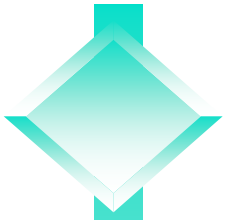


UNIVERSITÉ PARIS SORBONNE PARIS IV

Cours n° 4
Polymorphisme II



Sommaire

Principes

Méthodes de conception et de programmation

• Définition d'un « concept » -> Utilisation du « concept »
  définition d'une fonction + paramètres - > Appel de la fonction ,
  définition d'une classe + nom de variable -> Instanciation de la classe

Mécanisme de généricité

• Niveau supérieur d'abstraction
• Définition d'un « concept de concept » -> Définition d'un concept
  Définition d'un patron de fonctions + {type} -> Définition d'une fonction
  Définition d'un patron de classes + {type} -> Définition d'une classe

Utilisation de bibliothèques de patrons optimisés

• Standard Template Library (STL),
• Common Text Transformation Library (CTTL), Automaton Standard Template Library (ASTL), Matrix Template Library (MTL), Graph Template Library (GTL), Iterative Template Library (ITL), Histogram Template Library (HTL), Bioinformatics Template Library (BTL), Geostatistics Template Library, …

Définition

Famille de fonctions paramétrée par un ou plusieurs types

```
template <typename nomTA[=type], …>      // déclaration du nom du type abstrait
                                                [avec un type par défaut]
        typeTR identificateurF(paramTF) {    // déclaration de la fonction
         blocTF }                            // code de la fonction
```

Exemple : template <typename T=int> T triple {return 3*T};

Déclaration dans un fichier de prototypes (.h)

Pas de compilation sans choix du ou des types (instanciation)

Instanciation implicite

Pas de déclaration spécifique

```
        float i = triple(1.5);
```

Choix par le compilateur du type choisi

Instanciation explicite

Déclaration à l'utilisation du ou des types choisis

```
        float i = triple<float>(1.5);
```

Instanciation par défaut

```
        int i = triple<>(2);
```

Minimum.h

Minimum de 2 et 3 items (1/2)

```
/**
 * @brief Calcul du minimum de 2 items
 * @param[in] 2 items
 * @return minimum
 */
template <typename T>
T Minimum(T x, T y){
return x < y ? x : y;
}

/**
 * @brief Calcul du minimum de 3 items
 * @param[in] 3 items
 * @return minimum
 */
template <typename T>
T Minimum(T x, T y, T z){
return Minimum(Minimum(x, y), z);
}
```

testMinimum.cpp

Minimum de 2 et 3 items (2/2)

```
#include "Minimum.h"

#include <iostream>
using namespace std;

int main (int argc, char *argv[]) {

float x, y; cout << "entrez deux réels : "; cin >> x >> y;
cout << Minimum(x, y) << endl;

int i, j; cout << "entrez deux entiers : "; cin >> i >> j;
cout << Minimum<int>(i, j) << endl;

float z; cout << "entrez trois réels : "; cin >> x >> y >> z;
cout << Minimum<float>(x, y, z) << endl;

int k; cout << "entrez trois entiers : "; cin >> i >> j >> k;
cout << Minimum<>(i, j, k) << endl;

return 0; }
```

Définition

Famille de classes paramétrée par un ou plusieurs types

```
template <typename nomTA[=type], …>      // déclaration des noms du type abstrait
                                                [avec un type par défaut]
        class identificateurC {                  // déclaration de la classe
        };
```

Exemple :

```
template <typename T=float> class Point {
        private:
        T abscisse, ordonnée;
        public:
        Point(T a, T b);
        };
```

Patrons de fonctions membres

Déclaration dans le fichier contenant le patron de classes

Pas de déclaration de type par défaut

type par défaut du patron de classes

Exemple :
```
template <typename T> Point::Point(T a, T b) {
                        abscisse = a;
                        ordonnée = b
                    }
```

Pas de compilation sans choix du ou des types (instanciation)

Instanciation explicite

Déclaration à l'utilisation du ou des types choisis

```
Point<int> p(1, 2);
```

Instanciation par défaut

```
Point<> p(1.5, 2.1);
```

TableauDyn.h

Patron de classes d'un tableau dynamique

```
#include <iostream> // déclaration des flots standard
#include <cassert>
using namespace std;

// classe générique de tableau dynamique
template <typename T> class TableauDyn {
private :
unsigned int capacite;    // capacité du tableau tab
unsigned int pasExtension; // pas d'extension du tableau
T* tab;   // tableau alloué en mémoire dynamique
void agrandir();
public :
TableauDyn(unsigned int c, int p);
~TableauDyn();
T lire(unsigned int i) const;
void ecrire(unsigned int i, T it);
int getCapacite();
};
```

TableauDyn.h

Patrons de méthodes d'un tableau dynamique (1/3)

```
/** @brief constructeur du tableau d'items dynamique
 * caractérisé par un pas d'extension (p)
 * Allocation en mémoire dynamique du tableau d'items
 * de capacité (c) caractérisé par un pas d'extension (p)
 * @param [in] c : capacité du tableau
 * @param [in] p : capacité du tableau */
template <typename T> TableauDyn<T>::TableauDyn(unsigned int c, int p)
{
assert((c>=0) && (p>0));
capacite = c;
pasExtension = p;
// arrêt du programme en cas d'erreur d'allocation
tab = new T[capacite];
}

/** @brief destructeur du tableau d'items en mémoire dynamique */
template <typename T>TableauDyn<T>::~TableauDyn() {
delete [] tab;
tab = NULL;
}
```

TableauDyn.h

Patrons de méthodes d'un tableau dynamique (2/3)

```
/**@brief Lecture d'un item d'un tableau d'items
 * @param[in] t : le tableau d'items
 * @param[in] i : l'indice de l'item dans le tableau
 * @return l'item au poste i
 * @pre i < nbItems  */
template <typename T> T TableauDyn<T>::lire(unsigned int i) const {
assert(i < capacite);
return tab[i];
}

/**@brief Ecrire un item dans un tableau d'items
 * @param[in] i : l'indice où écrire l'item
 * @param[in] item : l'item à écrire
 * @pre i <= capacite  */
template <typename T> void TableauDyn<T>::ecrire(unsigned int i, T it)
{
assert(i>=0);
while (i >= capacite) agrandir();
tab[i] = it;
}
```

TableauDyn.h

Patrons de méthodes d'un tableau dynamique (3/3)

```
/**
 * @brief Agrandir un tableau d'items de son pas d'extension
 */
template <typename T> void TableauDyn<T>::agrandir() {
/* Allocation d'un tableau de (capacite + pasExtension) items */
T* newT = new T[capacite + pasExtension];
/* Recopie des (capacite) éléments du tableau dans le nouveau tableau
*/
for (unsigned int i = 0; i < capacite; ++i)
newT[i] = tab[i];
/* Désallocation de l'ancien tableau d'items */
delete [] tab;
/* Mise à jour de tab et de la capacité du nouveau tableau
 * résultant de l'extension de capacité */
tab = newT;
capacite += pasExtension;
}
```

testTableauDyn.cpp

Test du tableau dynamique générique

```
#include "TableauDyn.h"
#include "../Cours01/Etudiant.h"
#include "Complexe.h"

int main() {
TableauDyn <Etudiant> t1(0,1);
Etudiant Pascal; Pascal.putnote(3, 15); t1.ecrire(0, Pascal);
Etudiant e = t1.lire(0); cout << e.getnote(3) << endl;

TableauDyn <Complexe> t2(1,2); Complexe c1(-0.8,0.7), c2;
c2 = c1 * c1; t2.ecrire(1, c2);
c1 = t2.lire(1); cout << c1 << endl;

return 0;}
```

A la compilation : instanciation du patron de classes

A l'exécution : instanciation de la classe

Héritage (1/3)

testTableauDyn2Chat.cpp

Classe dérivée d'un patron de classes

class B public A<type>

```
#include "TableauDyn.h"
#include "../Cours03/Chat.h"

class TableauDyn2Chat : public TableauDyn<Chat> {
public:
TableauDyn2Chat(unsigned int c, int p) : TableauDyn <Chat> (c, p) {}
void presente() {for (int i= 0;i < capacite;i++) tab[i].presente();}
};

int main() {
TableauDyn2Chat t(0,1);
Chat c1("Felix"); t.ecrire(0, c1);
Chat c2("Potam"); t.ecrire(1, c2);
t.presente(); }
```

```
je m'appelle Felix

je m'appelle Potam
```

Héritage (2/3)

testSofstream.cpp

Patron de classes dérivée d'une classe

template <typename T> class B: public A

```
#include "../Cours03/Animal.h"

template <typename T> class Sofstream : public ofstream {
public:
void serialiser(T d) {write((char*) &d, sizeof(T));}
};

int main() {
Sofstream <Animal> fc;
fc.open("FileAnimal.bin", ofstream::out|ofstream::binary);
Animal a("Blaireau", 4);
fc.serialiser(a);
fc.close();
}
```

Héritage (3/3)

testTableauDynQ.cpp

Patron de classes dérivée d'un patron de classes

template < typename T, typename U> class B : public A<T>

```
#include "TableauDyn.h"
#include "../tp01/Date.h"
#include "../Cours01/Etudiant.h"
template <typename U, typename T> class TableauDynQ : public
TableauDyn<T> {
protected:
U quality;
public:
TableauDynQ<U,T>(unsigned int c, int p, U q) : TableauDyn <T> (c, p) {
quality = q;}
U getQuality() {return quality;}
};
int main() {
Etudiant Pascal; Pascal.putnote(3, 15); Date d(30,4,2006);
TableauDynQ<Etudiant, Date> tq(0,1, Pascal);
tq.ecrire(0, d); tq.getQuality(); }
```

CompareTo.h

Principes

Extension de la notion de fonction

Patron de fonction avec mémorisation
    base de la programmation fonctionnelle
Redéfinition de l'opérateur ()
Foncteur unaire (un paramètre)
Foncteur binaire (deux paramètres)
Foncteur logique (prédicat)
    retourne un booléen

```
// Définition du foncteur binaire de comparaison avec mémorisation
template <typename T> class compareTo {
  private:
  int res; // mémorisation du résultat
  public:
    int operator () (const T& x, const T& y) {
    res = x.compareTo(y);
      return res;
    }
  };
```

MaximumTableau.h

Utilisation du foncteur

Méthode générique de recherche de l'élément maximum d'un conteneur d'Item

```
// Fonction template prenant en paramètre un tableau d'Item et sa
taille et un foncteur binaire :
template <typename T, typename F>
int MaximumTableau(T t, int taille, F foncteurCmp) {
int max = 0;
for (int i = 1;i < taille;i++) {
if (foncteurCmp(t.lire(max), t.lire(i)) < 0)
max = i;
}
    return max;
}
```

Identite.cpp

Instanciation pour une classe donnée

Opérateur de comparaison pour Identite

```
/ Comparaison entre deux objets de type Identite
int Identite::compareTo(const Identite & y) const {
     if (nom_ < y.nom()) return -1;
     if (nom_ > y.nom()) return 1;
     if (prenom_ < y.prenom()) return -1;
     if (prenom_ > y.prenom()) return 1;
     return 0;
}
```

Exemple d'utilisation

testMaximumTableauIdentite.cpp

```
#include "MaximumTableau.h"
#include "MaximumTableau.h"
#include "compareTo.h"
#include "TableauDyn.h"
#include "../cours03/Identite.h"

int main() {

   compareTo<Identite> cmpIdentite;
   TableauDyn <Identite> t(0,1);
   Identite i1("Montacié","Claude"); t.ecrire(0, i1);
   Identite i2("Seddah", "Djamé"); t.ecrire(1, i2);
   Identite i3("Dzerdz","Anastazija"); t.ecrire(2, i3);

   int j = MaximumTableau(t, t.getCapacite(), cmpIdentite);

   cout << j << " " << t.lire(j) << endl; return 0;
}
```

```
1 Seddah Djamé
```