Introduction aux méthodes Orientées Objets

Quatrième partie

Modélisation avec UML 2.0

Programmation orientée objet en C++

Pré-requis:

maitrise des bases algorithmiques (cf. 1ier cycle),
maitrise du C (variables, fonctions, pointeurs, structures)

Thomas Grenier

Insa-GE IF3

Plan

I. Premier exemple

II. Définitions de l'Orienté Objet

III. Encapsulation, concept de classe

IV. Héritage (généralisation)

V. Polymorphisme

VI. Généricité (modèles de classe)

V – Polymorphisme – Plan

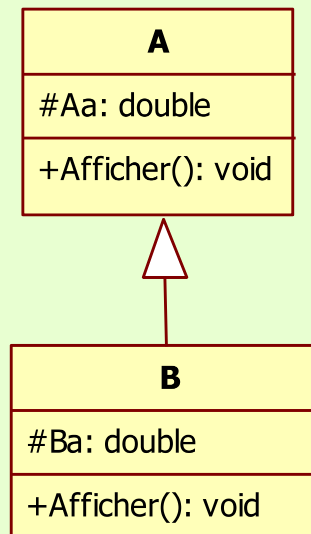
- Concept du polymorphisme
- Définitions
  - UML
  - C++
- Exercice
- Quizz

V – Polymorphisme, concept

Polymorphisme : *qui peut prendre plusieurs formes*

- Concept relatif à la surcharge des <u>opérations</u> des classes d'une même hiérarchie
- Extension des possibilités du mécanisme d'héritage

  Un objet d'une classe fille est ***du même type*** *que sa classe mère*

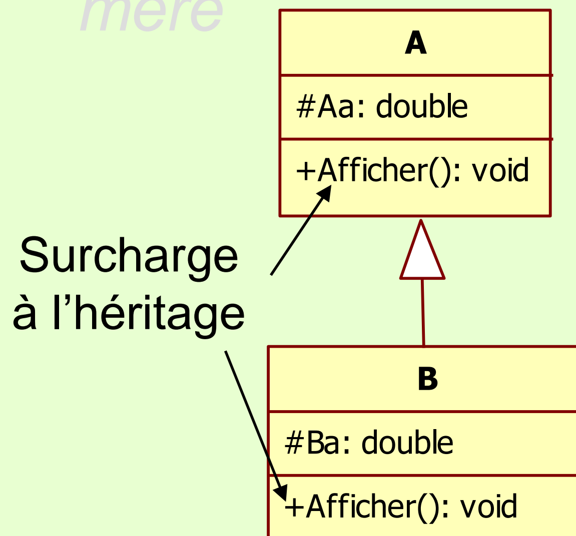


V – Polymorphisme, concept

Polymorphisme : *qui peut prendre plusieurs formes*

- Concept relatif à la surcharge des <u>opérations</u> des classes d'une même hiérarchie
- Extension des possibilités du mécanisme d'héritage

*Un objet d'une classe fille est **du même type** que sa classe mère*



Surcharge de Afficher()

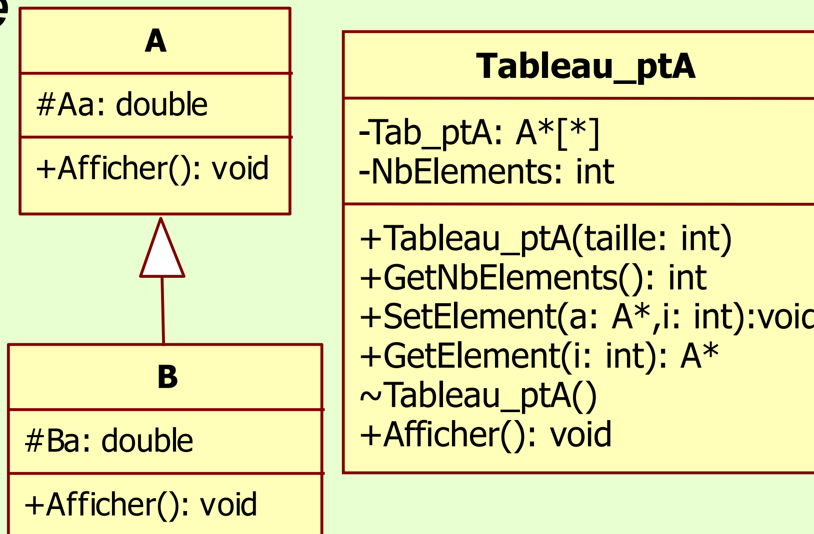
pour un objet de type B, A::Afficher() est remplacée par B::Afficher()

V – Polymorphisme, concept

Polymorphisme : *qui peut prendre plusieurs formes*

- Concept relatif à la surcharge des opérations des classes d'une même hiérarchie
- Extension des possibilités du mécanisme d'héritage

  Un objet d'une classe fille est ***du même type*** *que sa classe mère*



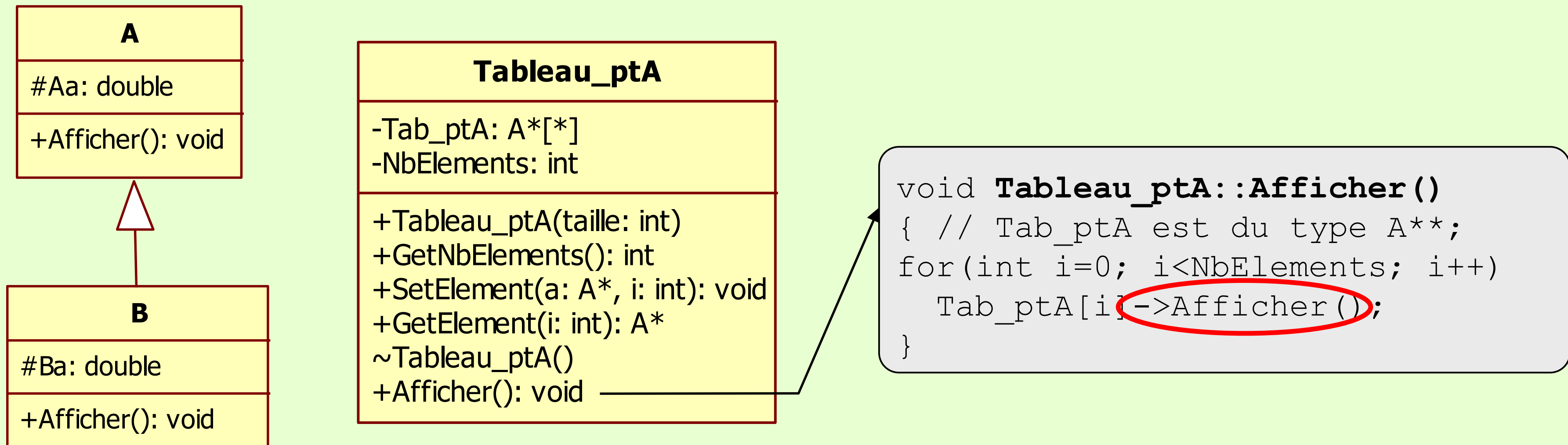
Avec un objet de Tableau_ptA on peut gérer :
- des objets de type A
- des objets de type B !

Polymorphisme

Nb : les notations A sont abusives en UML, un champ A* est une association ou une agrégation*

V – Polymorphisme, concept

Problème: surcharge des opérations à l'héritage



Quelle fonction Afficher() est appelée ?

Si l'élément *i* est un objet A ?
Si l'élément *i* est un objet B ?

A::Afficher()

A::Afficher n'est pas la méthode la plus adaptée pour l'affichage des objets B

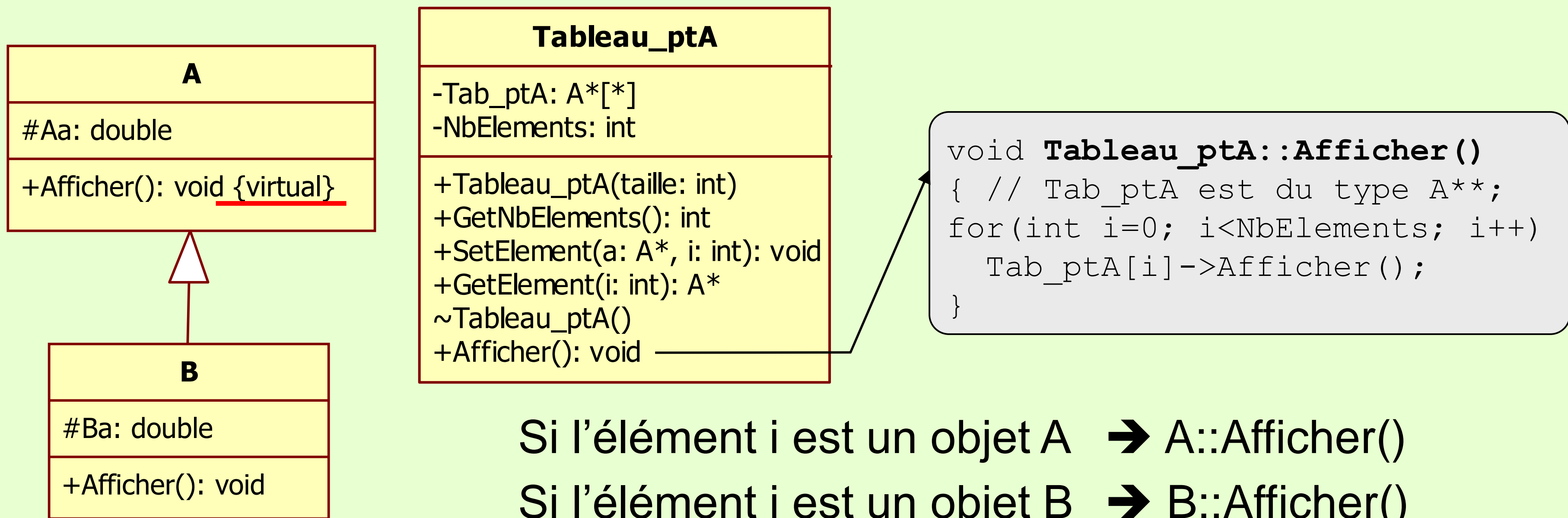
(le champs B::Ba ne peut pas être affiché par A::Afficher())

Polymorphisme

Nb : les notations A sont abusives en UML*

V – Polymorphisme, UML

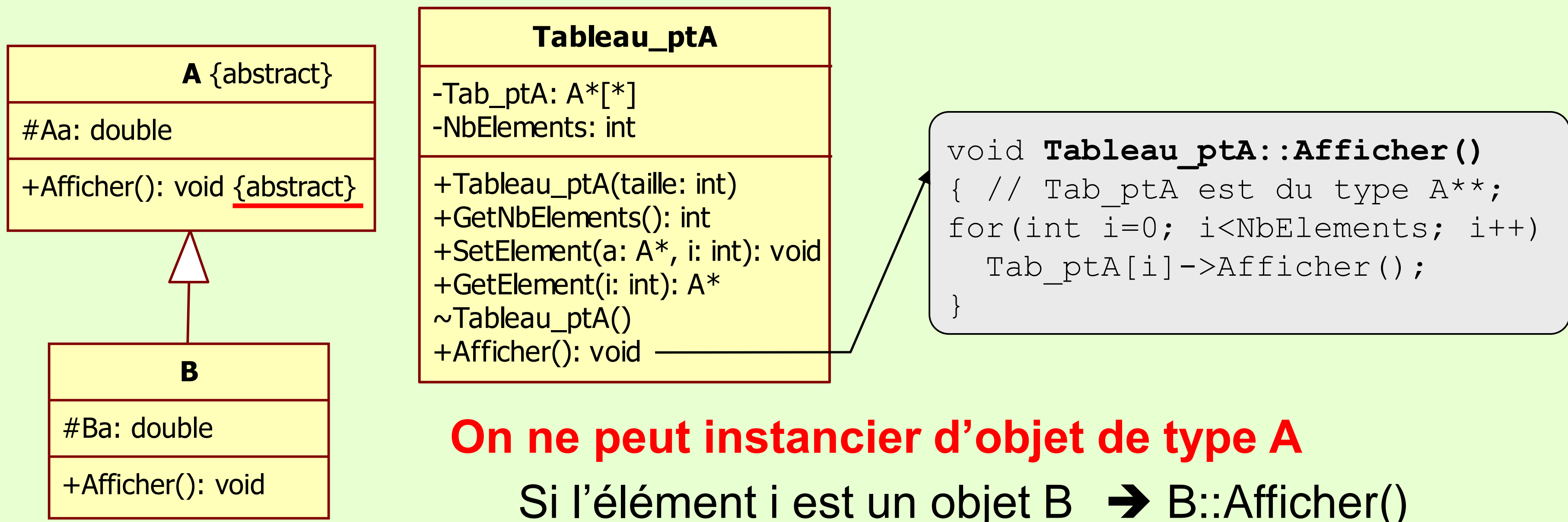
- Opération **virtuelle** *(représentée ~~en italique~~ par {virtual}…)*
  - Opération avec implémentation (en C++)
  - Signifie: « *mes classes filles **peuvent** se charger de la ré-implémentation de ce comportement* »



Nb : les notations A sont abusives en UML, un champ A* est une association ou une agrégation*

V – Polymorphisme, UML

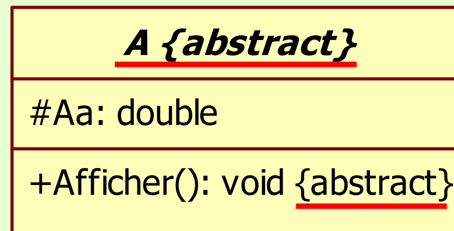
- Opération **abstraite** *(représentée ~~en italique~~* par {abstract}*…)*
  - Opération sans implémentation (en C++)
  - Signifie: « *mes classes filles **doivent** se charger de l'implémentation de ce comportement* »



Nb : les notations A sont abusives en UML, un champ A* est une association ou une agrégation*

V – Polymorphisme, UML

- Classe abstraite *(représentée ~~en italique~~ par {abstract}…)*
  - Une classe possédant une opération abstraite est obligatoirement une classe abstraite
  - Une classe abstraite ne peut pas être instanciée (dérivation obligatoire)

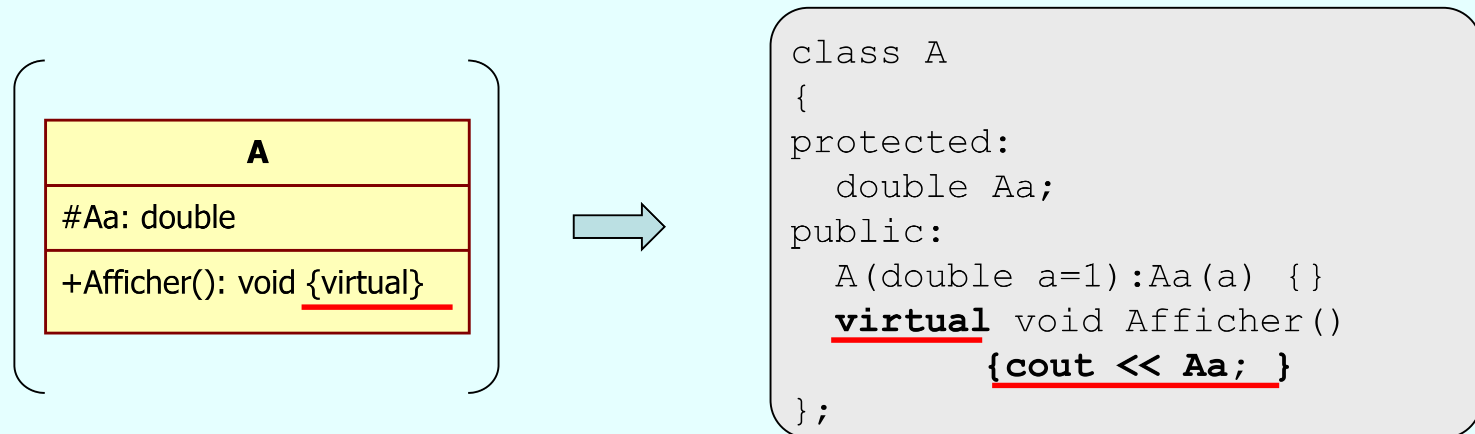


On ne peut pas instancier des objets de type A…

➔ Les classes abstraites sont les classes les plus hautes dans la hiérarchie
➔ Les classes abstraites permettent de définir une architecture et obligent les classes dérivées à respecter cette architecture

V – Polymorphisme, C++

- Méthode virtuelle ({virtual} en UML*)
  - méthode pouvant être implémentée en C++
  - Signifie: « *je propose une implémentation de ce comportement mais mes classes filles peuvent l'améliorer, l'implémentation la plus adaptée au type de l'objet sera exécutée*»



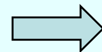
On peut instancier des objets de type A !

*: *notation normalement inexistante en UML*

V – Polymorphisme, C++

- Méthode virtuelle pure ou méthode abstraite
  - méthode non implémentée en C++
  - Signifie: « *mes classes filles sont chargées de l'implémentation de ce comportement* »

A {abstract}
#Aa: double
+Afficher(): void {abstract}



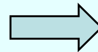
```
class A
{
protected:
  double Aa;
public:
  A(double a =1):Aa(a) {}
  virtual void Afficher() = 0;
};
```

On ne peut pas instancier des objets de type A…

V – Polymorphisme, C++

- Classe abstraite
  - Une classe possédant au moins une méthode virtuelle pure est obligatoirement une classe abstraite
  - Une classe abstraite ne peut pas être instanciée (dérivation obligatoire)

A {abstract}
#Aa: double
+Afficher(): void {abstract}

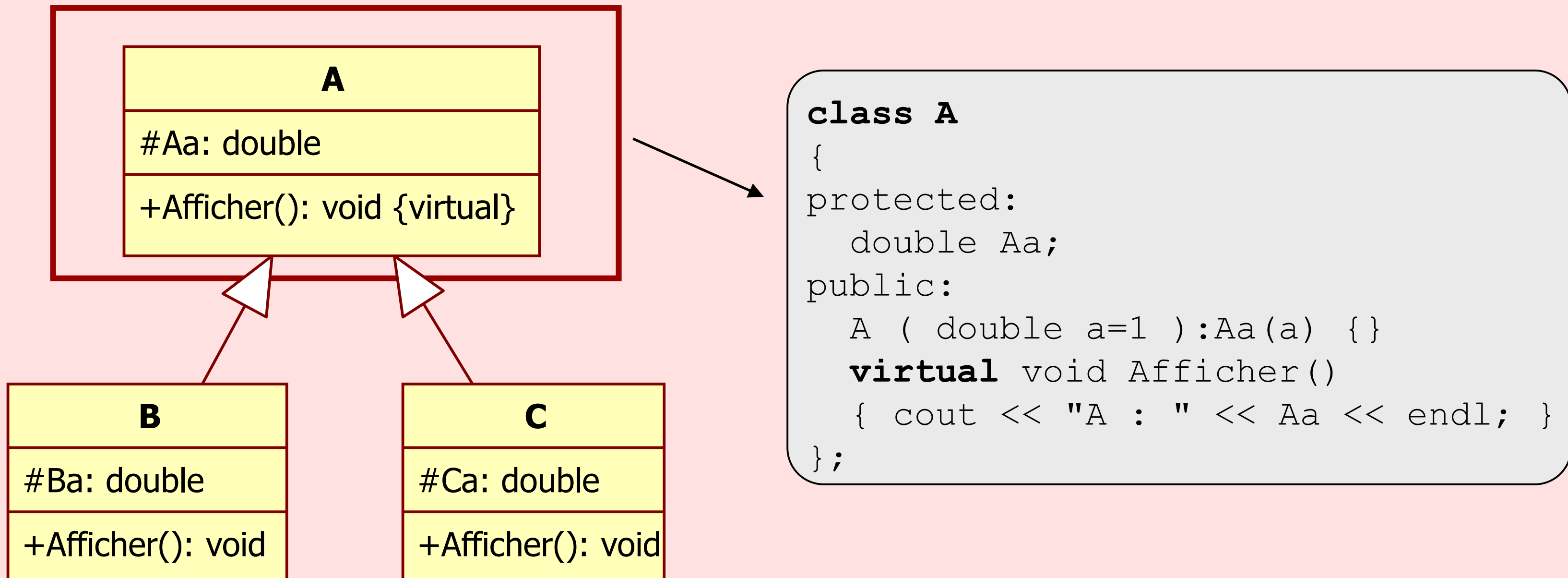


```
class A
{
protected:
  double Aa;
public:
  A(double a =1):Aa(a) {}
  virtual void Afficher() = 0;
};
```

On ne peut pas instancier des objets de type A

V – Polymorphisme, exemple

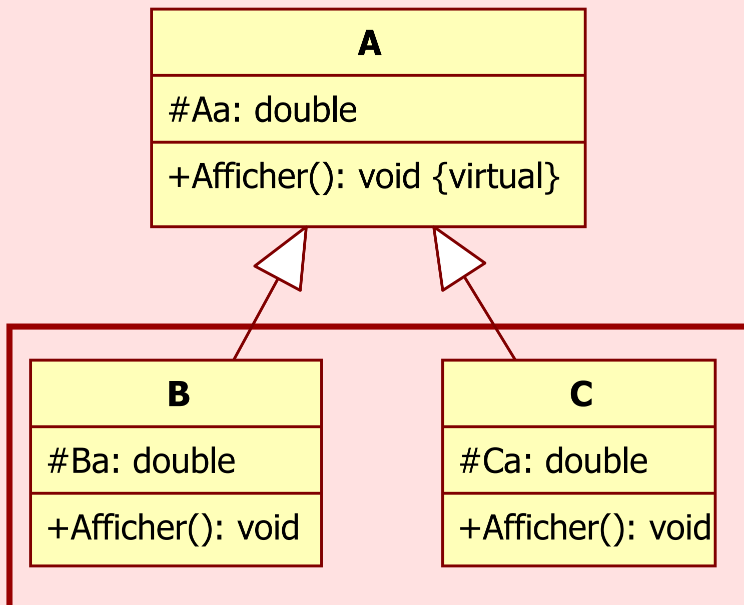
- Exemple complet:
  Tableau de classes A, B et C




```
class A
{
protected:
  double Aa;
public:
  A ( double a=1 ):Aa(a) {}
  virtual void Afficher()
  { cout << "A : " << Aa << endl; }
};
```

V – Polymorphisme, exemple

- Exemple complet: classes A, B et C




```
class B : public A
{
protected:
  double Ba;
public:
  B( double a=1, double b=1):A(a),Ba(b) {}
  void Afficher()
   { cout<<"B :"<< Aa <<", "<< Ba << endl; }
};
```

```
class C : public A
{
protected:
  double Ca;
public:
  C( double a=1, double c=1):A(a),Ca(c) {}
  void Afficher()
   { cout<<"C :"<< Aa <<", "<< Ca << endl; }
};
```

V – Polymorphisme, exemple

- Exemple complet: classe Tableau_ptA

Tableau_ptA
-Tab_ptA: A*[*] -NbElements: int
+Tableau_ptA(taille: int) +GetNbElements(): int +SetElement(a: A*, i: int): void +GetElement(i: int): A* ~Tableau_ptA() +Afficher(): void

```
class Tableau_ptA
{
  A **Tab_ptA;
  int NbElements;
public:
  Tableau_ptA(int taille)
    { Tab_ptA = new A*[taille];
      NbElements = taille;   }
  void SetElement( A *a, int i )
    { Tab_ptA[i] = a; }
  A* GetElement( int i )
    { return Tab_ptA[i]; }
  int GetNbElements()
    { return NbElements; }
  void Afficher();
  ~Tableau_ptA()
    { delete[] Tab_ptA; }
};
```

```
void Tableau_ptA::Afficher()
{
for( int i=0; i<NbElements; i++)
  {
  cout << "Tab["<< i << "] :";
  Tab_ptA[i]->Afficher();
  }
}
```

V – Polymorphisme, exemple

- Exemple complet: utilisation

```
int main(void)
{
B b1( 1, 1);
B b2( 2, 2);

C c1(-1, -1);
C c2(-2, -2);

Tableau_ptA tab(4);

tab.SetElement( &b1, 0);
tab.SetElement( &c1, 1);
tab.SetElement( &c2, 2);
tab.SetElement( &b2, 3);

tab.Afficher();

return 0;
}
```

Création des objets de types B et C

Création d'un tableau de 4 éléments

Affectation des éléments du tableau

Affichage de tous les éléments du tableau

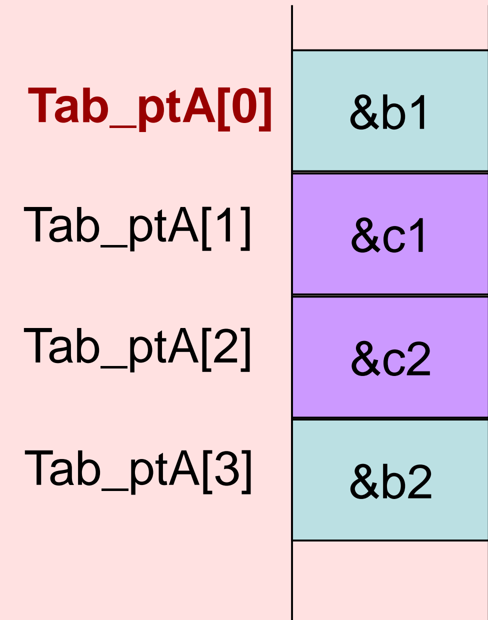
Exécution :

```
Tab[0] :B :1, 1
Tab[1] :C :-1, -1
Tab[2] :C :-2, -2
Tab[3] :B :2, 2
```

V – Polymorphisme, exemple

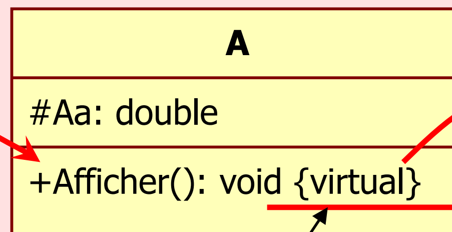
- exécution de `tab.Afficher();`

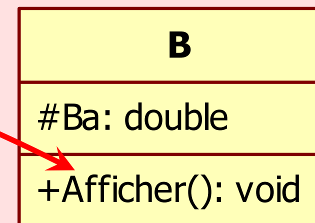
```
void Tableau_ptA::Afficher()
{
for( int i=0; i<NbElements; i++)
  {
  cout << "Tab["<< i << "] :";
  Tab_ptA[i]->Afficher();
  }
}
```

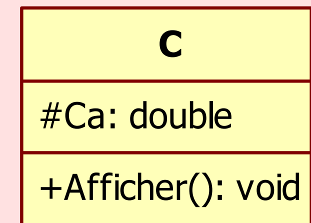


Tab_ptA[0]->Afficher()

type A*







Quelle instance ?

*Tab_ptA[0]
est de type B

V – Polymorphisme, exemple

- exécution de `tab.Afficher();`

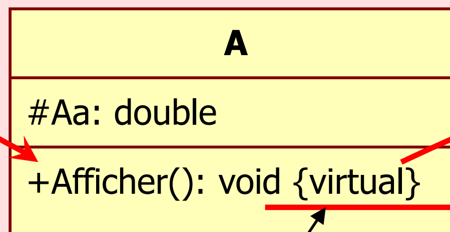
En mémoire

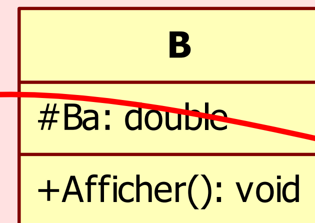
Tab_ptA[0]	&b1
Tab_ptA[1]	&c1
Tab_ptA[2]	&c2
Tab_ptA[3]	&b2

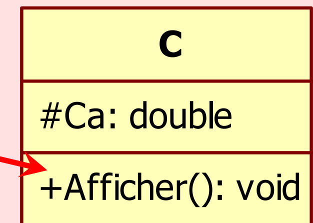
```
void Tableau_ptA::Afficher()
{
for( int i=0; i<NbElements; i++)
  {
  cout << "Tab["<< i << "] :";
  Tab_ptA[i]->Afficher();
  }
}
```

Tab_ptA[1]->Afficher()

type A*







Quelle instance ?

*Tab_ptA[1]
est de type C

V – Polymorphisme, exemple

- Comparaison: avec et sans méthode virtuelle

Sans méthode virtuelle

A
#Aa: double
+Afficher(): void

```
class A
{
protected:
  double Aa;
public:
  A ( double a=1 ):Aa(a) {}
  void Afficher()
  { cout << "A : " << Aa << endl; }
};
```

Exécution du même « main »

sans méthode virtuelle

```
Tab[0] :A : 1
Tab[1] :A : -1
Tab[2] :A : -2
Tab[3] :A : 2
```

avec méthode virtuelle

```
Tab[0] :B :1, 1
Tab[1] :C :-1, -1
Tab[2] :C :-2, -2
Tab[3] :B :2, 2
```

V – Polymorphisme - Quizz

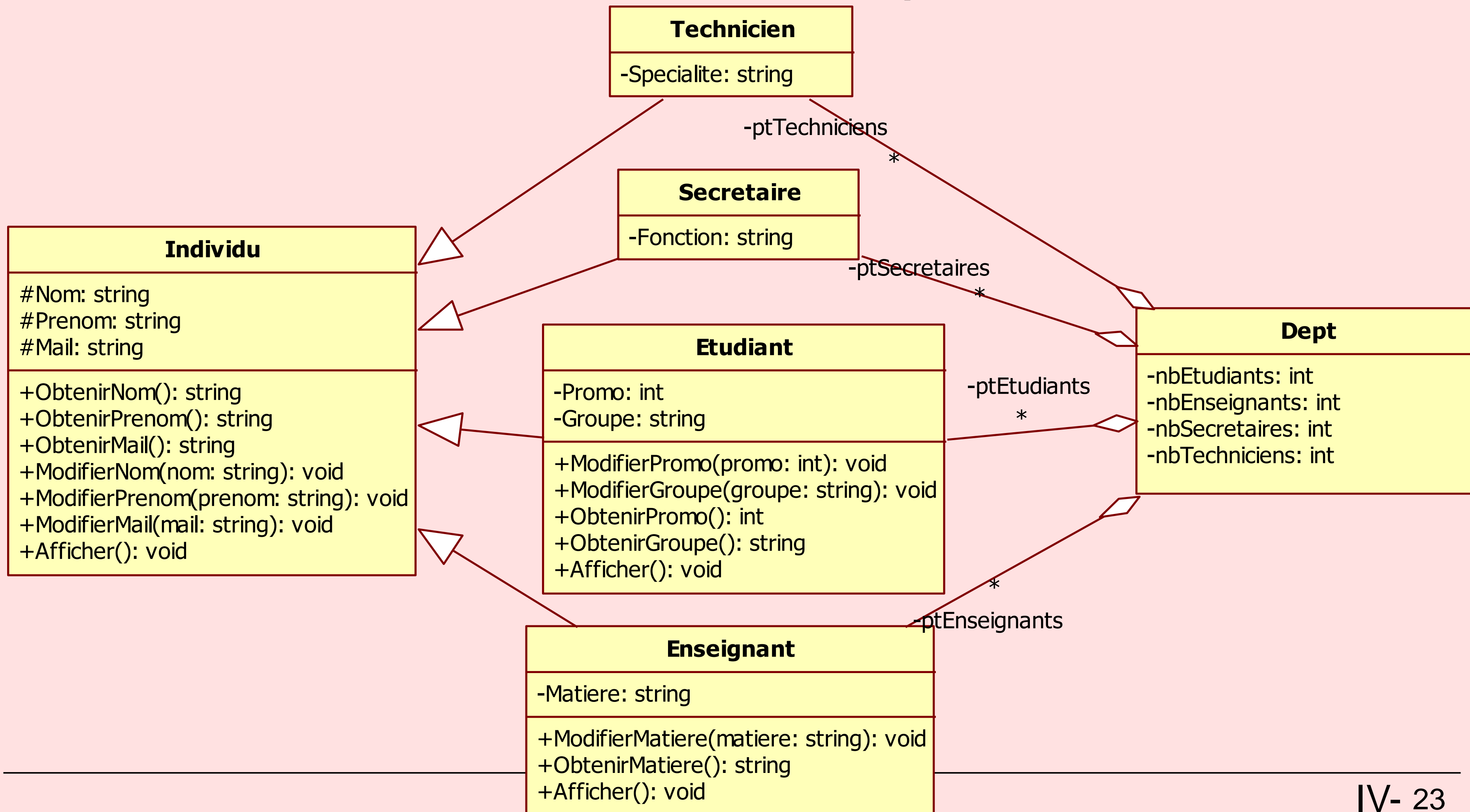
- Quels sont les objectifs/buts du polymorphisme ?
  -
  -
  -
  -
- Une méthode devrait toujours être virtuelle. Laquelle ?

Exercice équipe de sport Dept

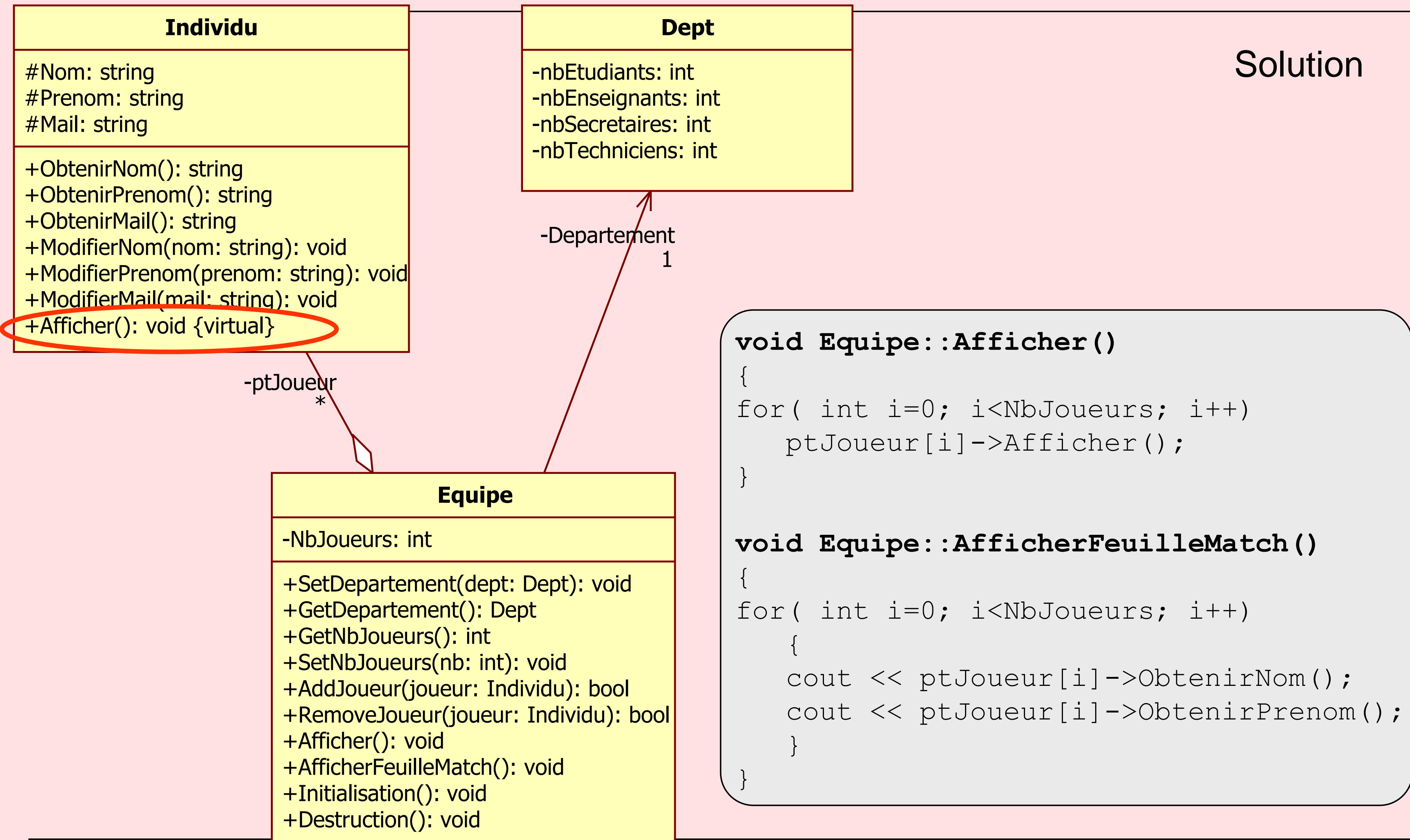
- A partir de la modélisation d’un département (classes Dept, Individu, Etudiant, Enseignant, …), modéliser **<u>une équipe</u>** dont les joueurs sont un mélange d’étudiants, d’enseignants, de secrétaires et de techniciens appartenant tous à un même département.
  - On voudra afficher la liste des joueurs d’une équipe, en précisant tous les attributs de chaque joueur (méthode afficher)
  - Pour les feuilles de match, on ne souhaite afficher que le nom et le prénom (méthode AfficherFeuilleMatch())

Exercice équipe de sport Dept

- Modélisation initiale d'un département :



Exercice équipe de sport Dept



Plan

I. Premier exemple

II. Définitions de l'Orienté Objet

III. Encapsulation, concept de classe

IV. Héritage (généralisation)

V. Polymorphisme

VI. Généricité (modèles de classe)

VI – Généricité – Plan

- Intérêt de la généricité
- Définitions
  - UML
  - C++
- Exercice

VI – Généricité – Plan

- Intérêt de la généricité
- Définitions
  - UML
  - C++
- Exercice

VI – Généricité – Intérêt

- Modéliser les classes Tableau_Double, Tableau_Int, Tableau_ptA.

Ces classes disposeront des mêmes fonctionnalités

Tableau_Int
-Tab: int[0..*] -Taille: int
+SetElement(i: int, v: int): void +GetElement(i: int): int +GetTaille(): int +SetTaille(taille: int): void <<create>>+Tableau_Int(taille: int) <<destroy>>+Tableau_Int() +Copie(t: Tableau_Int): void

Tableau_Double
-Tab: double[0..*] -Taille: int
+SetElement(i: int, v: double): void +GetElement(i: int): double +GetTaille(): int +SetTaille(taille: int): void <<create>>+Tableau_Double(taille: int) <<destroy>>+Tableau_Double() +Copie(t: Tableau_Double): void

Tableau_ptA
-Tab: A*[0..*] -Taille: int
+SetElement(i: int, v: A*): void +GetElement(i: int): A* +GetTaille(): int +SetTaille(taille: int): void <<create>>+Tableau_ptA(taille: int) <<destroy>>+Tableau_ptA() +Copie(t: Tableau_ptA): void

Alors? qu'est ce qu'on fait? Architecture + polymorphisme ?

Le concept de polymorphisme est relatif aux opérations… et non aux types (paramètres ou attributs)

VI – Généricité – Intérêt

- **Obtenir une généricité du comportement d'une classe vis-à-vis des types des attributs**

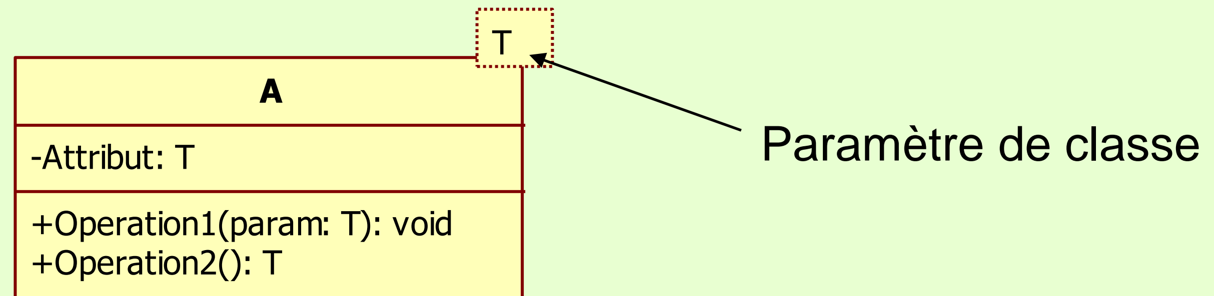
  le comportement interne d'une classe « générique » doit être indépendant du type des objets manipulés

- Retarder le choix des classes avec lesquelles une classe fonctionnera

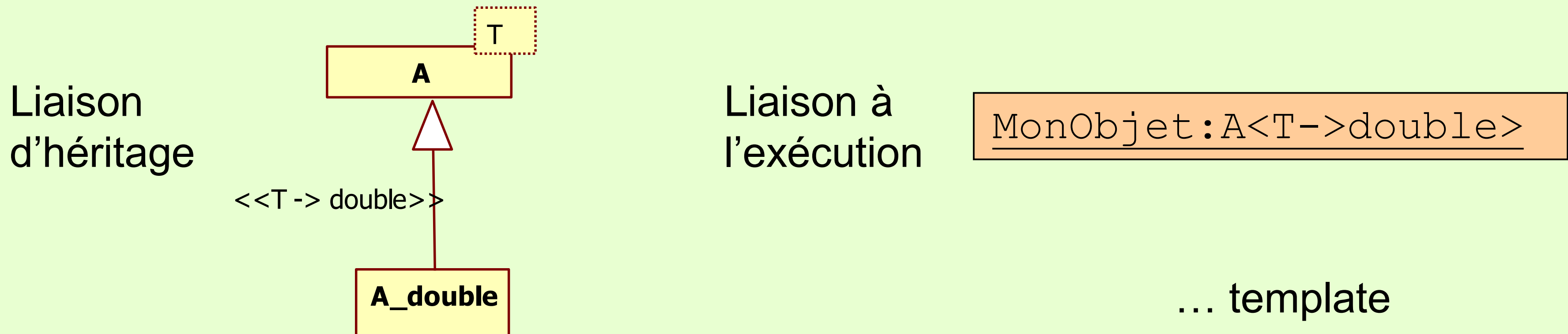
« je sais que cette classe fonctionnera avec d'autres classes, mais je ne connais pas, ou cela m'est égal, de connaître ces classes »

VI – Généricité – Définition UML

- modèle de classe ou classe paramétrée



- Deux manières d’associer un ensemble de classe à un modèle



Exemple vector<T>

```
#include <iostream>
#include <vector>
using namespace std;
int main () {
  vector<int> myvector;
  int sum (0);
  myvector.push_back (100);
  myvector.push_back (200);
  myvector.push_back (300);
  while (!myvector.empty())
             {
              sum+=myvector.back();
              myvector.pop_back();
             }
  cout << "The elements of myvector summed " << sum << endl;
  return 0;
}
```

Exemple vector<T>

```
// vector::operator[]
#include <iostream>
#include <vector>
 using namespace std;
 int main () {
 vector<int> myvector (10); // 10 zero-initialized elements
unsigned int i;
vector<int>::size_type sz = myvector.size();
// assign some values:
 for (i=0; i<sz; i++)
            myvector[i]=i;
// reverse vector using operator[]:
for (i=0; i<sz/2; i++) {
            int temp;
            temp = myvector[sz-1-i];
            myvector[sz-1-i]=myvector[i];
            myvector[i]=temp; }
cout << "myvector contains:";
for (i=0; i<sz; i++) cout << " " << myvector[i]; cout << endl;
 return 0; }
```