POLYTECHNIQUE MONTRÉAL

CHAPITRE 13

Héritage et abstractions

Identification des abstractions

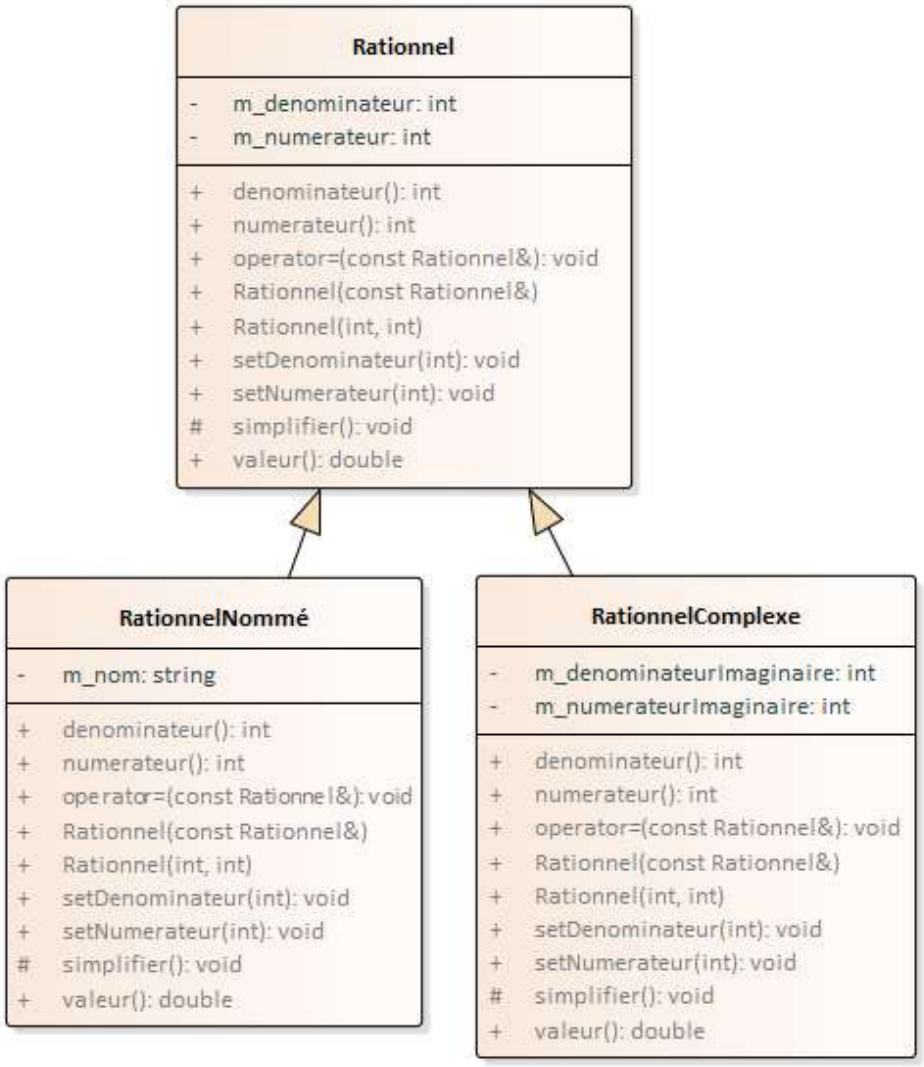
Identifier les abstractions.

- Identifier les abstractions pertinentes à un problème et à sa solution est **LA tâche** la plus importante et la plus difficile du travail d'analyse et de conception,
- Ce n'est que dans la mesure où un design aura correctement identifié les abstractions qu'il sera réutilisable.
- La forme la plus tangible des abstractions présentes dans un design sont les classes abstraites qui le constituent.

POLYTECHNIQUE MONTRÉAL

Identification des abstractions

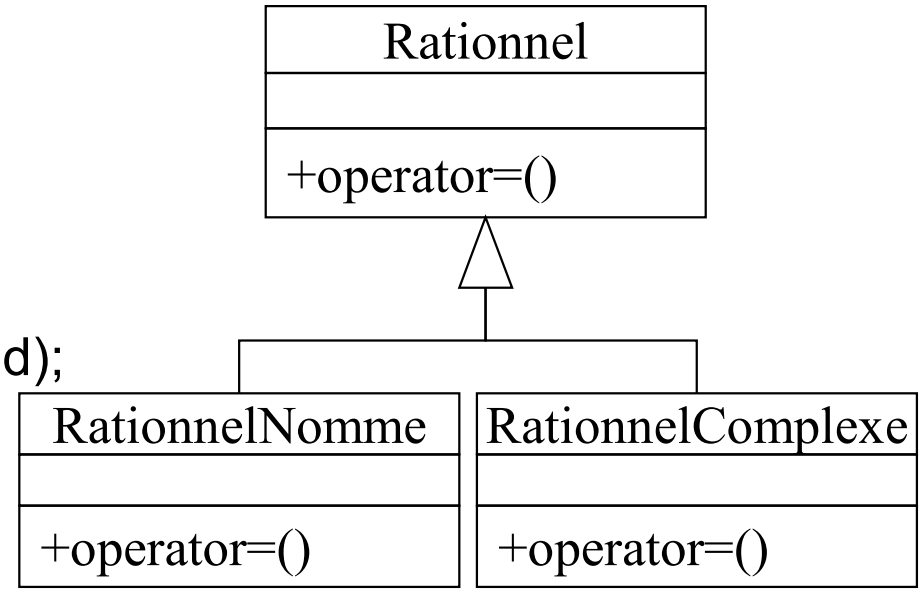
Soit une hiérarchie de 3 classes concrètes pour traiter des nombres rationnels:



POLYTECHNIQUE MONTRÉAL

Identification des abstractions

Concentrons nous uniquement sur les opérateurs d'assignation:




```
class Rationnel {
public:
    Rationnel& operator=(const Rationnel& mdd);
    ...
};

class RationnelNomme: public Rationnel {
public:
    RationnelNomme& operator=(const RationnelNomme& mdd);
    ...
};

class RationnelComplexe: public Rationnel {
public:
    RationnelComplexe& operator=(const RationnelComplexe& mdd);
    ...
};
```

POLYTECHNIQUE MONTRÉAL

Identification des abstractions

Utilisation des classes pour traiter des rationnels:

```
RationnelNomme rn1;
RationnelNomme rn2;

Rationnel* pRationnel1 = &rn1;
Rationnel* pRationnel2 = &rn2;
...
*pRationnel1 = *pRationnel2;
```

C'est l'opérateur d'assignation de la classe Rationnel qui est utilisé, même si les objets sont des RationnelNommes.

- ➔L'objet n'est que **partiellement modifié** (seule la partie Rationnel du RationnelNomme est modifiée),
- ➔Une classe devrait être «facile à utiliser correctement, et difficile à utiliser incorrectement»: ce n'est pas le cas ici !

Identification des abstractions

On pourrait rendre les opérateurs d'assignation virtuels:

```
class Rationnel {
public:
  virtual Rationnel& operator=(const Rationnel& mdd);
  ...
};
class RationnelNomme: public Rationnel {
public:
  virtual RationnelNomme& operator=(const Rationnel& mdd);
  virtual RationnelNomme& operator=(const RationnelNomme& mdd);
  ...
};
class RationnelComplexe: public Rationnel {
public:
  virtual RationnelComplexe& operator=(const Rationnel& mdd);
  virtual RationnelComplexe& operator=(const RationnelComplexe& mdd);
  ...
};
```

POLYTECHNIQUE MONTRÉAL

Identification des abstractions

On ouvre la porte à des assignations mixtes et partielles:

```
RationnelNomme rn1, rn2;
RationnelComplexe rc1;
RationnelNomme* rnp1 = &rn1;

Rationnel* rp1 = &rn1;
Rationnel* rp2 = &rn2;
Rationnel* rp3 = &rc1;

rn1     = rn2;     // assignation correcte d'un RationnelNomme dans
                   // un autre (RationnelNommé& operator=(const RationnelNomme& mdd))
*rnp1 = rn2; // idem
*rp1   = rn2;   // assignation partielle d'un RationnelNomme dans
                // un autre (RationnelNommé& operator=(const Rationnel& mdd))
*rp1   = *rp2; // assignation partielle d'un RationnelNomme dans un
                // autre avec RationnelNommé& operator=(const Rationnel& mdd)
*rp1   = *rp3; // assignation mixte d'un RationnelComplexe dans un
                // RationnelNommé avec RationnelNommé& operator=(const Rationnel& mdd)
```

Comment faire pour que tout ceci fonctionne correctement ?

POLYTECHNIQUE MONTRÉAL

Identification des abstractions

Parfois l'assignation d'un Rationnel dans un autre est correcte, et parfois non:

- Le comportement correct de ce code ne peut être vérifié qu'au moment de l'exécution.

```
RationnelNomme& RationnelNomme::operator=( const Rationnel& mdd )
{
    if( this == &mdd ) return *this;

    try
    {
       return operator=( dynamic_cast< const RationnelNomme& >( mdd ) );
    }

    catch( bad_cast& exception )
    {
       // traiter l'erreur
    }
}
```

Identification des abstractions

Cette façon de faire a plusieurs faiblesses:

- Privilégie les erreurs à l'exécution plutôt que les erreurs de compilation
- Exige des programmeurs qu'ils traitent les exceptions… qu'est-ce qu'on fait quand quelqu'un essaie d'assigner un RationnelComplexe dans un RationnelNomme,
- Implique un coût accru à l'exécution.

Identification des abstractions

Une meilleure solution serait de déclarer l'opérateur non-virtuel dans toutes les classes, et de le déclarer «*private*» au niveau de la classe de base `Rationnel` :

- Non-virtuel puisqu'il est impossible de surcharger correctement l'opérateur sans passer par des conversions de type
- Private dans la classe Rationnel pour empêcher les clients de faire des assignations à partir de pointeurs sur la classe de base et ainsi éviter les assignations mixtes ou partielles. // ou même **deleted**

POLYTECHNIQUE MONTRÉAL

Identification des abstractions

```
class Rationnel
{
private:
   Rationnel& operator=(const Rationnel& mdd);
   ...
};

class RationnelNomme: public Rationnel {
public:
   RationnelNomme& operator=(const RationnelNomme& mdd);
   ...
};

class RationnelComplexe: public Rationnel {
public:
   RationnelComplexe& operator=(const RationnelComplexe& mdd);
   ...
};
```

Identification des abstractions

Cette solution n'est envisageable que si l'on n'a pas besoin d'accéder à l'opérateur d'assignation de Rationnel:

```
Rationnel& operator=(const Rationnel& mdd)
```

Ce n'est pas le cas si `Rationnel` est une classe concrète (ce qui était une donnée du problème).

De plus, cela rend impossible l'implantation correcte des opérateurs pour RationnelNomme et RationnelComplexe, qui doivent appeler l'opérateur d'assignation de leur classe de base.

Identification des abstractions

Il faut donc éliminer la nécessité d'utiliser directement l'opérateur d'assignation entre deux objets de la classe Rationnel.

Si l'on transformait la classe Rationnel en classe abstraite, notre problème serait réglé.

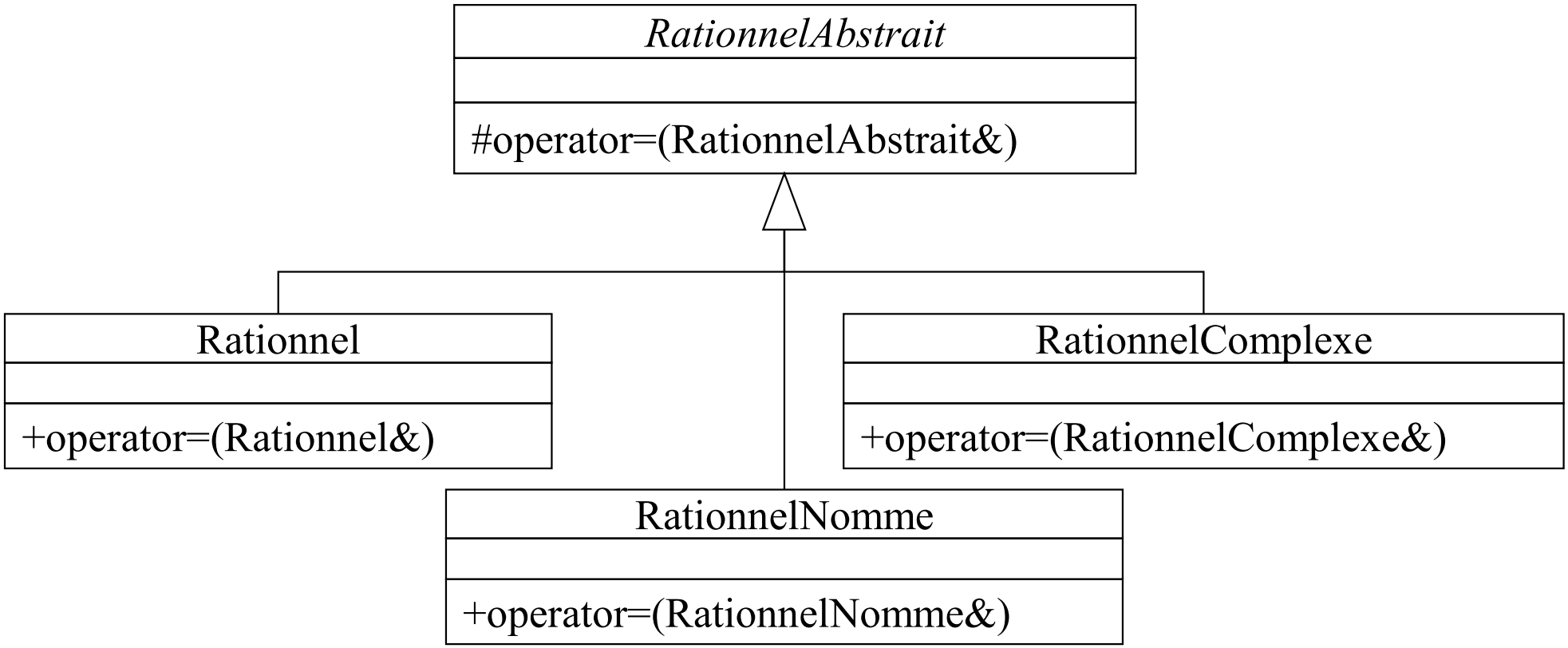
Dans la conception initiale, Rationnel est une classe concrète pour laquelle nous avons besoin de l'opérateur d'assignation:

- On ne peut pas simplement transformer Rationnel en classe abstraite.

POLYTECHNIQUE MONTRÉAL

Identification des abstractions

Nouvelle Hiérarchie de classe pour traiter des rationnels:



Tous les opérateurs d'assignation peuvent être déclarés **non-virtuels**

Identification des abstractions

```
RationnelAbstrait&
RationnelAbstrait::operator=(const RationnelAbstrait& mdd)
{
   if ( this == &mdd ) return *this;

   // Copier les attributs de la classe RationnelAbstrait
   [...]

   return *this;
}

RationnelComplexe&
RationnelComplexe::operator=( const RationnelComplexe& mdd )
{
   if ( this == &mdd ) return *this;

   // 1- Copier les attributs de la classe de base avec l'opérateur d'assignation
   RationnelAbstrait::operator=( mdd );

   // 2- Copier les attributs de la classe RationnelComplexe
   [...]

   return *this;
}
```

Identification des abstractions

Cette nouvelle conception nous donne tout ce dont nous avons besoin:

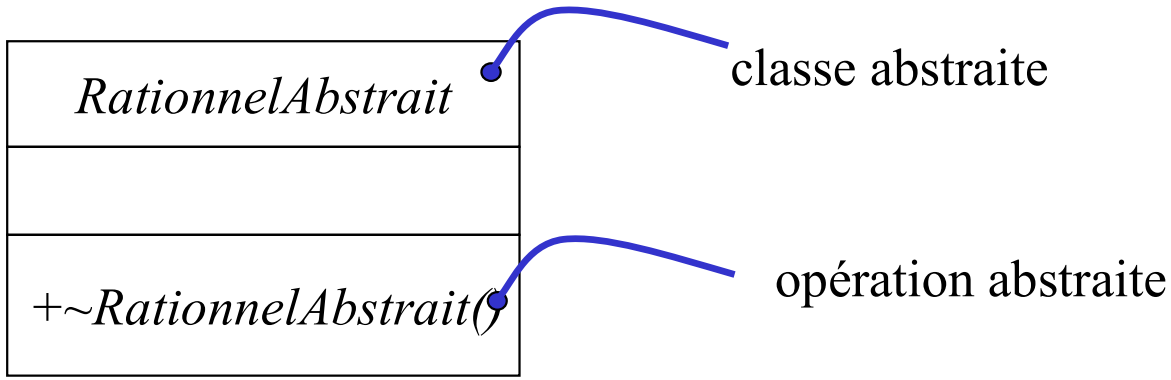
- Les assignations homogènes sont permises pour les classes Rationnel, RationnelNomme et RationnelComplexe,
- Les assignations partielles et hétérogènes sont interdites,
- Les sous-classes peuvent appeler l'opérateur d'assignation de la classe de base,
- Les tentatives d'assignation mixtes et partielles seront détectées À LA COMPILATION.

POLYTECHNIQUE MONTRÉAL

Identification des abstractions

Une classe abstraite est une classe dont certaines méthodes virtuelles (au moins 1) doivent être redéfinies (virtuelles pures):

- En déclarant une classe abstraite, on empêche les clients de la classe de construire directement des objets de cette classe,
- Pour que notre design fonctionne, il faut que *RationnelAbstrait* soit une classe abstraite et, donc, qu'elle ait au moins une méthode abstraite.



Identification des abstractions

En général, trouver au moins une opération abstraite est facile. Si ce n'est pas le cas, une bonne méthode candidate est le destructeur.

On peut déclarer le destructeur d'une classe comme une fonction virtuelle pure, mais dans le cas spécial du destructeur, la méthode doit **obligatoirement** être implémentée.

POLYTECHNIQUE MONTRÉAL

Identification des abstractions

Sur le plan de la conception, cet exemple illustre une règle importante:

Règle: dans une hiérarchie de classes, **seules les classes terminales (les feuilles) devraient être des classes concrètes**.

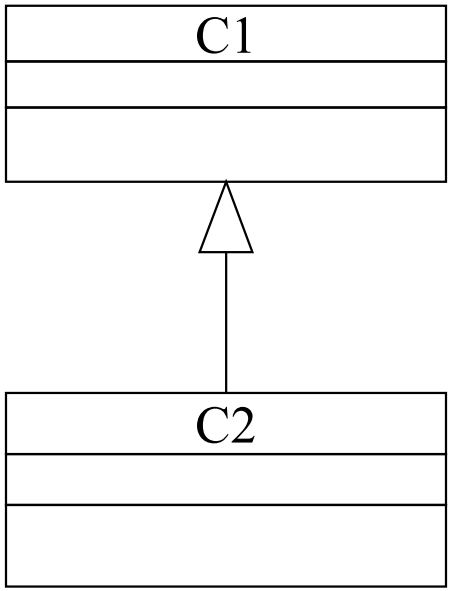
Dit autrement:

On ne devrait pas dériver une classe concrète d'une autre classe concrète.

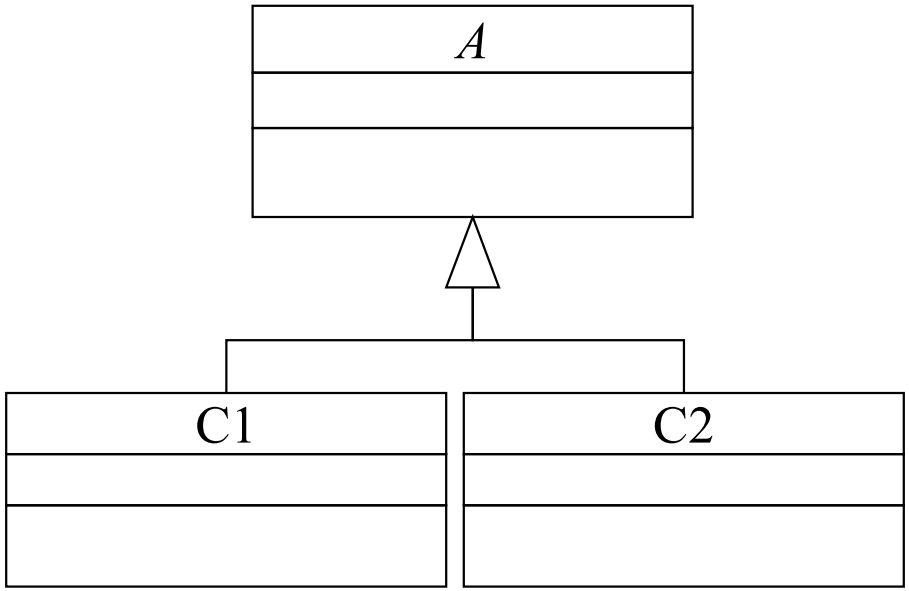
POLYTECHNIQUE MONTRÉAL

Identification des abstractions

Si, en cours de conception, on identifie deux classes concrètes C1 et C2 et que l'on voudrait faire dériver C2 publiquement de C1, on devrait transformer cette hiérarchie de 2 classes en une hiérarchie à trois classes, où C1 et C2 dérivent toutes les deux d'une classe abstraite A:



Idée initiale



Hiérarchie transformée

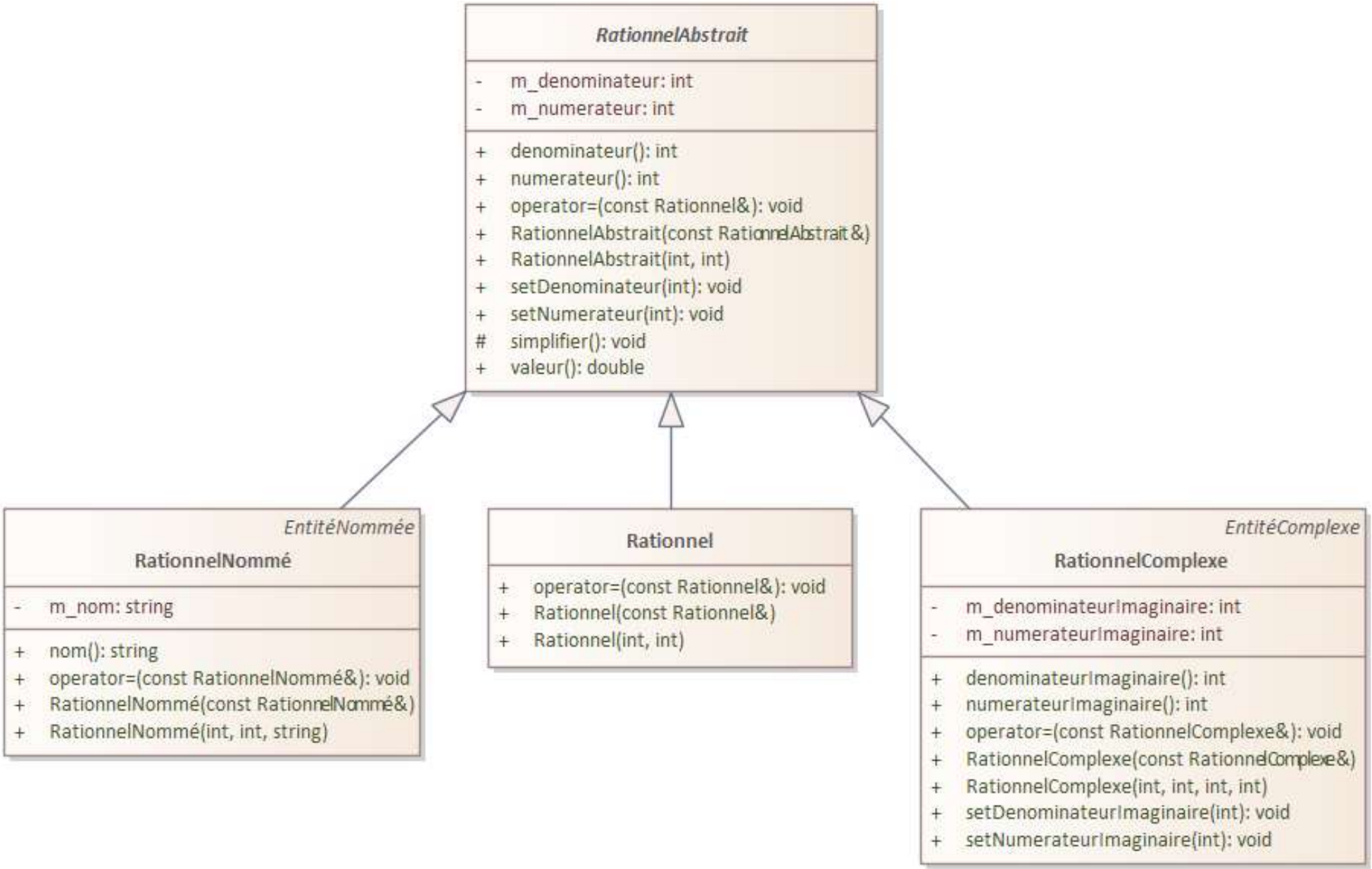
Identification des abstractions

Cette transformation force l'identification des abstractions qui sont immédiatement utiles dans le système en cours de conception:

- Ces classes illustrent et documentent dans le modèle les abstractions importantes du système,
- Ces classes abstraites fournissent des points de départ tout désigné pour des extensions futures du système.

POLYTECHNIQUE MONTRÉAL

Identification des abstractions

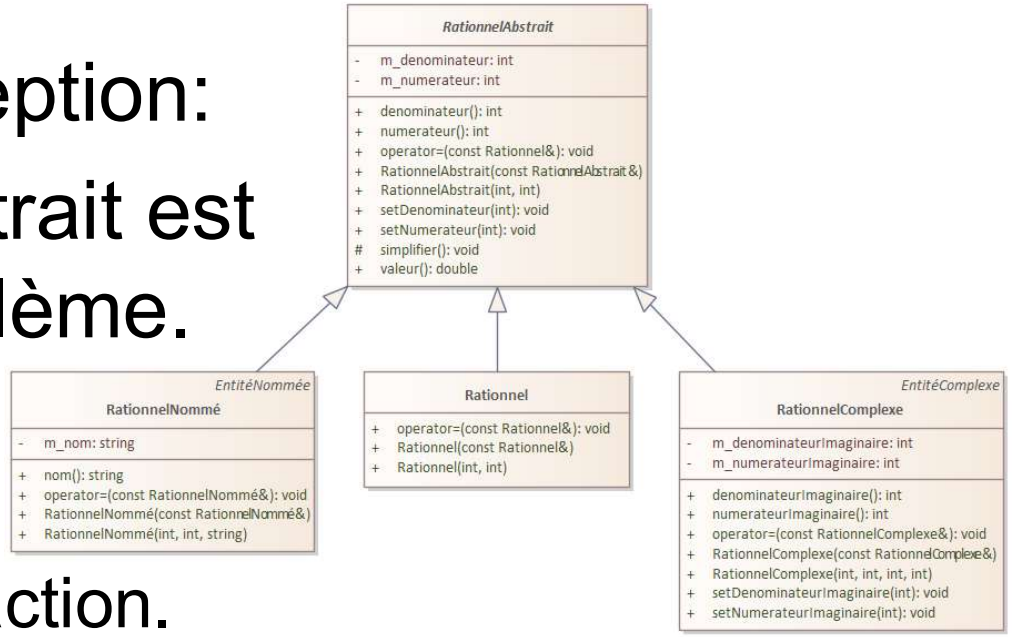


POLYTECHNIQUE MONTRÉAL

Identification des abstractions

Dans cette nouvelle conception:

- La classe RationnelAbstrait est une abstraction du problème.
  - ➢ Ce n'est pas la seule !
  - ➢ Une classe qui porte un nom est aussi une abstraction.
  - ➢ Une classe qui a une partie imaginaire aussi.
- Si ces caractéristiques peuvent être utiles ailleurs, on pourrait aussi les extraire comme des classes abstraites.



POLYTECHNIQUE MONTRÉAL

Notion d'interface

Chaque caractéristique est isolée dans une interface spécialisée :

- Ségrégation des interfaces, le 'I' des principes SOLID,
- Les caractéristiques sont combinées par héritage multiple.