IFT159
Analyse et programmation
Chapitre 10 — Introduction aux types abstraits

Gabriel Girard

Département d'informatique
UNIVERSITÉ DE SHERBROOKE

18 novembre 2008

Chapitre 10 — Introduction aux types abstraits

1 Introduction à UML
- Diagramme de cas d'utilisation
- Diagramme de classes
- Diagramme de séquence de messages

2 Exemple 1
- Spécification
- Analyse/conception
- Implantation
- Implantation alternative
- Compilation

3 Exemple 2
- Spécification
- Analyse/conception
- Implantation

4 Surcharge d'opérateurs.

Introduction à UML

- Utilisateurs qui guident la définition des modèles
- Besoins des utilisateurs servent de fil rouge tout au long du cycle de développement
- Élaboration de modèles, pas de barrière stricte entre analyse et conception
- Facilite une démarche d'analyse itérative et incrémentale
- Les modèles d'analyse et de conception ne diffèrent que par leur niveau de détail
- L'élaboration encourage une approche non-linéaire

UML - diagrammes

- Diagrammes UML (vues statiques)
  - Cas d'utilisation
  - D'objets
  - De classes
  - De composants
  - De déploiement
- Diagrammes UML (vues dynamiques)
  - De collaboration
  - De séquence
  - D'états-transitions
  - D'activités

Diagramme de cas d'utilisation

- Pour structurer les besoins des utilisateurs et les objectifs correspondants d'un système
- Se limite aux préoccupations « réelles » des utilisateurs
- Identifie les utilisateurs du système (acteurs) et leur interaction avec le système
- Défini le contour du système é modéliser
- Identifie les fonctionnalités principales (critiques) du système

Diagramme de cas d'utilisation : Éléments

- Acteurs : entité externe qui agit sur le système
- Cas d'utilisations : ensemble d'actions réalisées par le système, en réponse à une action d'un acteur
- Relations : interaction entre acteur et cas d'utilisation

Diagramme de cas d'utilisation : Symboles

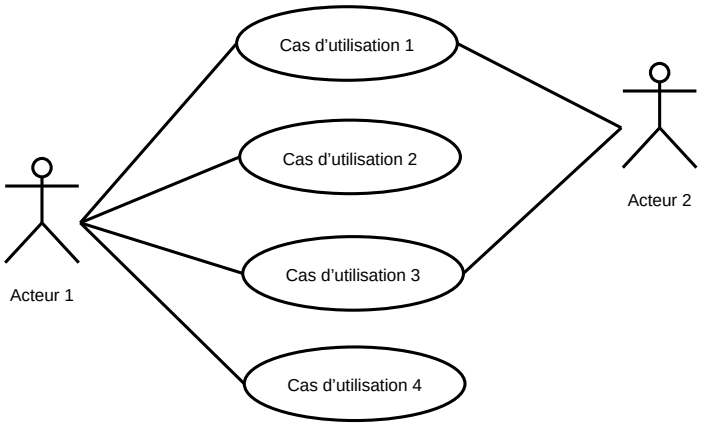


Diagramme de classes

- Une collection d'éléments de modélisation statiques qui montre la structure d'un modèle
- Fait abstraction des aspects dynamiques et temporels
- Pour un modèle complexe, plusieurs diagrammes de classes complémentaires doivent être construits.
  On peut par exemple se focaliser sur :
  - les classes qui participent à un cas d'utilisation
  - les classes associées dans la réalisation d'un scénario précis
- Pour représenter un contexte précis, un diagramme de classes peut être instancié en diagrammes d'objets

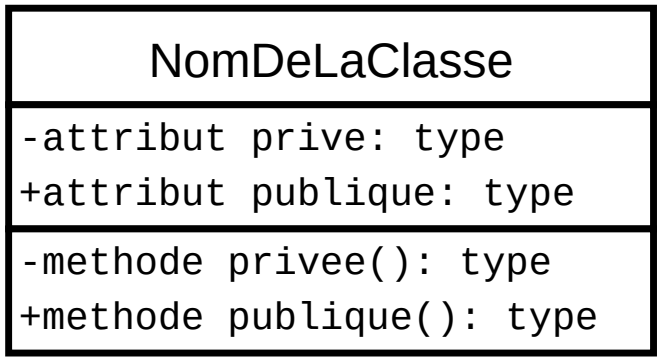
Diagramme de classes : Éléments

- Classes : représentation des entités à partir desquelles des objets seront créés
  - Nom de la classe
  - Attributs de la classe
  - Méthodes de la classe
- Relations : lien entre deux classes
  - Étiquette
  - Rôles des classes en relation
  - Cardinalités

Diagramme de classes : Éléments

- Types de relation :
  - Association : exprime une connexion sémantique bidirectionnelle entre deux classes
  - Aggrégation : exprime un couplage fort et une relation de subordination
  - Composition : agrégation forte, si l'agrégat est détruit (ou copié), ses composants le sont aussi

Diagramme de classes : Symboles



Associations : Symboles

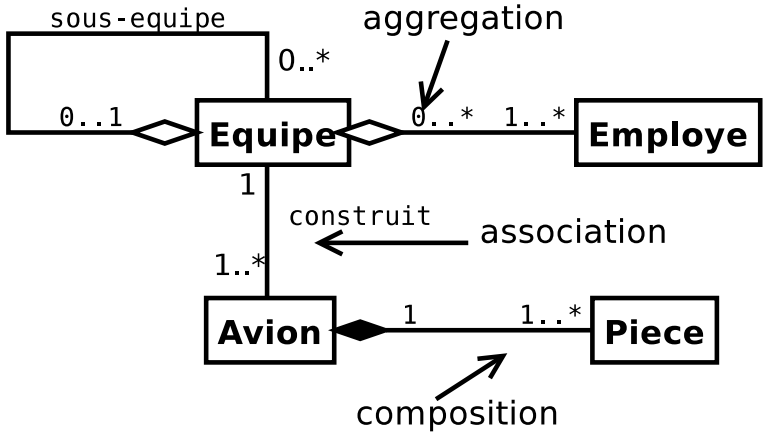


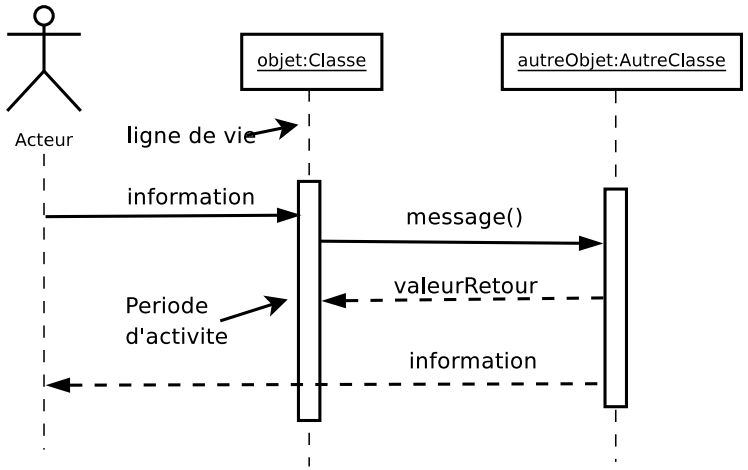
Diagramme de séquence de messages

- Représenter des collaborations entre objets selon un point de vue temporel
- Expression des interactions
- Permet de visualiser la séquence des appels des opérations définies dans les classes
- L'ordre d'envoi d'un message est déterminé par sa position sur l'axe vertical du diagramme
- Peut servir à illustrer un cas d'utilisation

Diagramme de séquence de messages : Éléments

- Acteurs : mêmes entités que dans les cas d'utilisations
- Objets : instanciés pour réaliser le scénario
- Lignes de vie : pour visualiser les période d'activité d'un objet
- Messages : échanges entre les acteurs et les objets et entre les objets
- Valeurs retournées : ce qui est retourné par un objet á la suite d'un appel

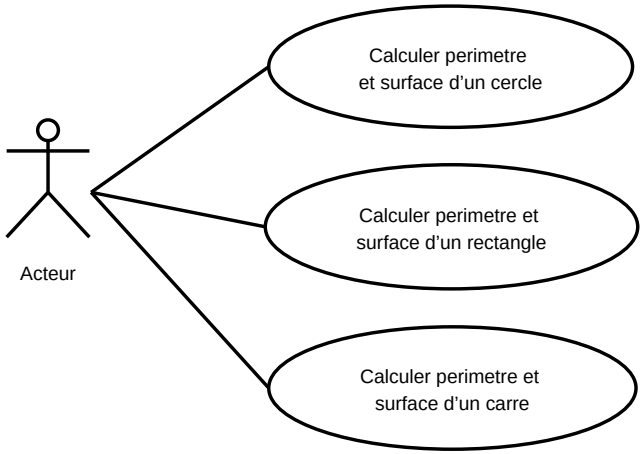
Diagramme de séquence de messages : Symboles



Spécification

Développer un programme qui permet de trouver le périmètre et la surface d'un certain nombre de figures géométriques. Les figures retenues sont le cercle, le rectangle et le carré.

Diagramme de cas d'utilisation



Analyse/conception - Les nouveaux types

- Trois nouveaux types sont nécessaires pour résoudre le problème.
- Le cercle, le carré et le rectangle.

Analyse globale

1. Entrées
   1. (clavier) Choix (Caratères)
   2. (clavier) Dimension (cercle, rectangle, carré)
2. Sorties
   1. (écran) périmètre (réel)
   2. (écran) surface (réel)
3. Formules ...
4. Constantes ...

Conception module principal

algorithme

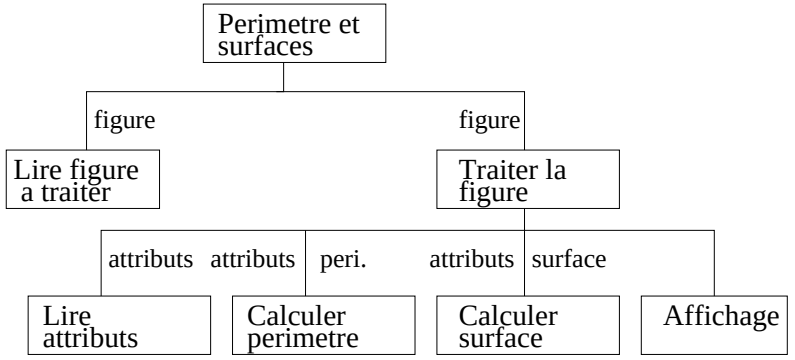
1. Pour chaque choix (trois choix)
   - 1.1 Lire choix de l'utilisateur
   - 1.2 Selon le choix de l'utilisateur (`C, R, K`)
     - 1.2.1 traiter le cercle (Module traiter figure),
     - 1.2.2 traiter le rectangle (Module tratier figure),
     - 1.2.3 traiter le carré (Module traiter figure) ;
     - 1.2.4 si le choix est invalide, le sytème produit un message d'erreur.

Conception module traiter figure

algorithme

1. Selon la figure
   - 1.1 Lire la figure
   - 1.2 Calculer le périmètre
   - 1.3 Calculer la surface
   - 1.4 Afficher
     - le périmètre
     - la surface

Analyse/conception - diagramme



Analyse/conception - le cercle

T.A.D. cercle (Analyse)

- Un cercle est connu par son rayon
- Les formules nécessaires sont :
  périmètre $= 2\pi r$
  surface $= \pi r^2$

Analyse/conception - le rectangle

T.A.D. **rectangle** (Analyse)

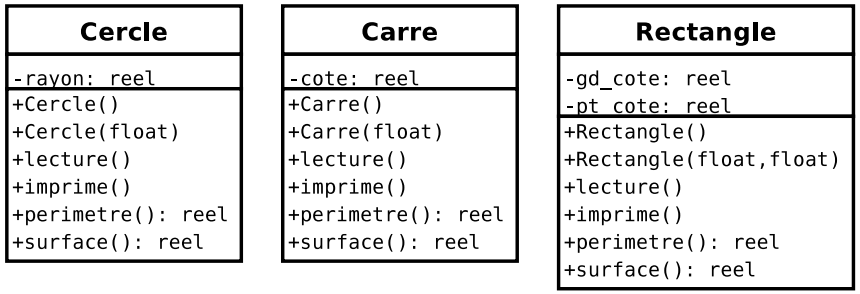
- Un rectangle est connu par son grand coté et son petit coté
- Les formules nécessaires sont :
  périmètre = 2(grand_coté+petit_coté)
  surface = grand_coté*petit_coté

Analyse/conception - le carré

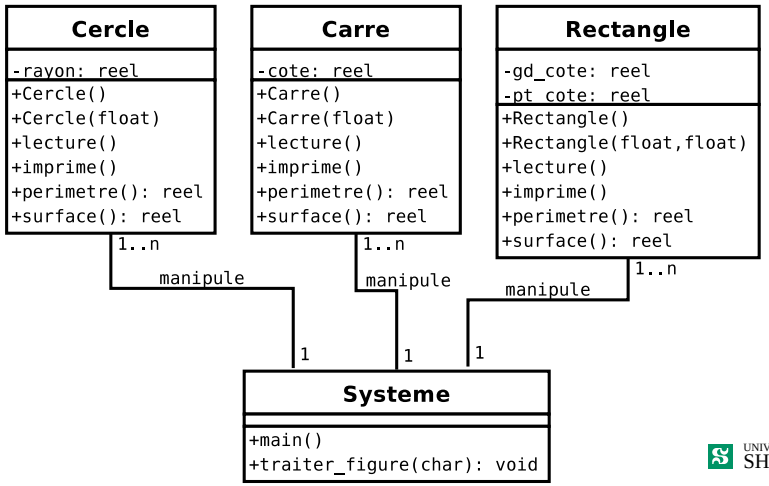
T.A.D. **Carré** (Analyse)

- Un carré est connu par son coté
- Les formules nécessaires sont :
  périmètre = 4*coté
  surface = coté^2

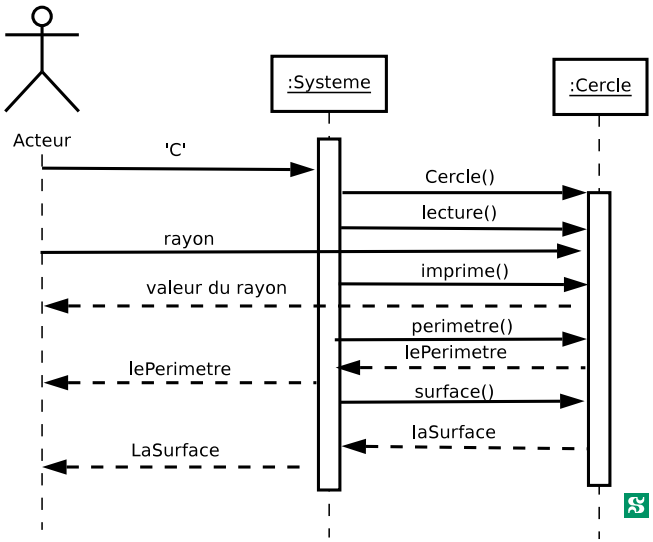
Analyse/conception - diagrammes de classes



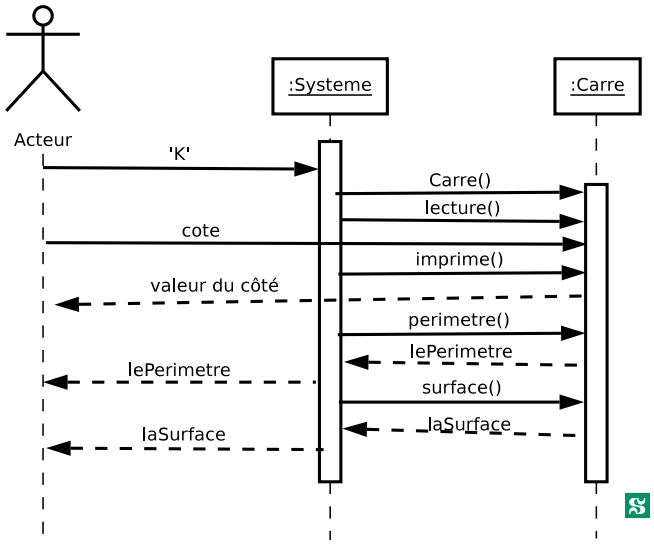
Analyse/conception



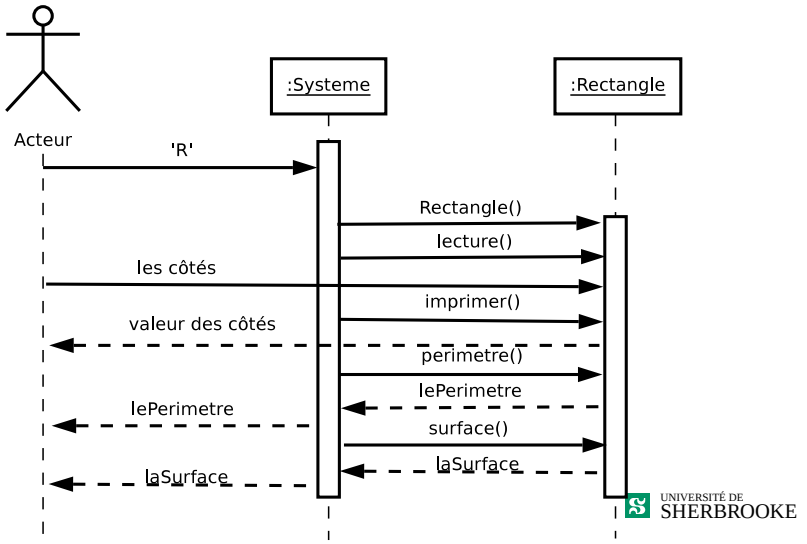
Diaglamme de séquence - scénario 1



Diaglamme de séquence - scénario 2



Diaglamme de séquence - scénario 3



Implantation - main.cpp

```
#include <iostream>
#include <cctype>
#include "Cercle.h"
#include "Rectangle.h"
#include "Carre.h"

using namespace std;
```


Implantation - main.cpp

```
int main()
{
 void traite_figure(char) ;
 char figure_choisie ;

 for (int i = 0 ; i < 3 ; i++)
 {
   cout<< "Entrer la figure " << endl
       << "  C pour cercle" << endl
       << "  R pour rectangle" << endl
       << "  K pour carre" << endl ;
   cin >> figure_choisie ;
   traite_figure(figure_choisie) ;
 }
 cout << "Fin du traitement" << endl ;
}
```

Implantation - traiter figure

```
void traite_figure(char figure)
{
  float rayon, cote, gr_cote, pt_cote;
  Cercle rond;
  Carre de;
  Rectangle boite;

  switch(toupper(figure))
  {
   case 'C': cout << "Donner le rayon en cms: " ;
             rond.lecture();
             cout << "Le cercle de rayon ";
             rond.imprime()
             cout << " a pour perimetre " ;
                  << rond.perimetre() << "cms" ;
                  << endl <<" et pour surface ";
                  << rond.surface() << "cms2" << endl ;
             break ;
```


Implantation - traiter figure

```
   case 'R': cout << "Donner les cotes en cms: " ;
             boite.lecture()
             cout << "Le rectangle de taille "
             boite.imprime()
             cout << " a pour perimetre ";
                  << boite.perimetre() << "cms";
                  << endl <<" et pour surface ";
                  << boite.surface() <<"cms2" << endl ;
             break ;
```


Implantation - traiter figure

```
   case 'K': cout << "Donner le cote en cms: " ;
             de.lecture();
             cout << "Le carre de taille "
             de.imprime();
             cout << " a pour perimetre ";
                  << de.perimetre() << "cms";
                  << endl <<"  et pour surface ";
                  << de.surface() << "cms2" << endl;
            break ;
  default:cerr << "Forme inconnue. Recommencer"
               << endl ;
  }
}
```


Implémentation du T.A.D. **Cercle** - cercle.h

```
const float PI = 3.14159 ;

class Cercle
{
        float rayon ;
   public:
        Cercle() ;
        Cercle(float);
        void lecture()
        void imprime();
        float perimetre() ;
        float surface() ;
} ;
```

Implémentation du T.A.D. **Cercle** - cercle.h

```
Cercle::Cercle()
{
     rayon = 0 ;
}
Cercle::Cercle(float ray)
{
     rayon = ray ;
}
float Cercle::lecture()
{
     cin >> rayon ;
}
```


Implémentation du T.A.D. **Cercle** - cercle.h

```
float Cercle::imprime()
{
     cout << rayon ;
}

float Cercle::perimetre()
{
     return 2 * PI * rayon ;
}


float Cercle::surface()
{
     return PI * rayon * rayon ;
}
```


Implémentation du T.A.D. **Rectangle** - rectangle.h

```
class Rectangle
{
        float gd_cote ;
        float pt_cote ;
public:
        Rectangle() ;
        Rectangle(float,float);
        void lecture()
        void imprime();
        float perimetre() ;
        float surface() ;
} ;
```


Implémentation du T.A.D. **Rectangle** - rectangle.cpp

```
Rectangle::Rectangle()
{
     gd_cote = 0 ;
     pt_cote = 0 ;
}
void Rectangle::Rectangle(float cote1, float cote2)
{
   if (cote1 > cote2)
   {
        gd_cote = cote1 ; pt_cote = cote2 ;
   }
   else
   {
     gd_cote = cote2 ; pt_cote = cote1 ;
   }
}
```

Implémentation du T.A.D. **Rectangle** - rectangle.cpp

```
float Rectangle::lecture()
{
     cin >> gd_cote >>  pt_cote ;
}
float Rectangle::imprime()
{
     cout << gd_cote << ' x ' <<  pt_cote ;
}
float Rectangle::perimetre()
{
     return 2 * (gd_cote + pt_cote) ;
}
float Rectangle::surface()
{
    return gd_cote * pt_cote;
}
```


Implémentation du T.A.D. **Carre** - carre.h

```
class Carre
{
        float cote ;
public:
        Carre() ;
        Carre(float);
        void lecture()
        void imprime();
        float perimetre() ;
        float surface() ;
} ;
```


Implémentation du T.A.D. **Carre** - carre.cpp

```
Carre::Carre()
{
     cote = 0 ;
}

void Carre::Carre(float c)
{
    cote =c;
}

float Carre::lecture()
{
     cin >> cote ;
}
```


Implémentation du T.A.D. **Carre** - carre.cpp

```
float Carre::imprime()
{
     cout << cote ;
}

float Carre::perimetre()
{
     return 4 * cote ;
}

float Carre::surface()
{
     return cote * cote ;
}
```

Alternative

Un carré ressemble beaucoup à un rectangle

1. On peut utiliser un rectangle pour représenter le carré : un objet de type `Carre` cachera alors une instance de `Rectangle` (agrégation).
2. On peut dire qu'un carré est un rectangle particulier : la classe `Carre` est construire en « héritant » des caractéristiques de la classe `Rectangle`.

Cette dernière solution sera vu en IFT339.

Implémentation du T.A.D. **Carre** - carre.h

```
class Carre
{
        Rectangle boite ;
    public:
        Carre() ;
        Carre(float) ;
        void lecture()
        void imprime();
        float perimetre() ;
        float surface() ;
} ;
```


Implémentation du T.A.D. **Carre** - carre.cpp

```
Carre::Carre()
{
     cote = 0 ;
}

Carre::Carre(float cote)
{
     Rectangle boite_carre (cote,cote) ;
     boite = boite_carre ;
}
```


Implémentation du T.A.D. **Carre** - carre.cpp

```
float Carre::perimetre()
{
     return boite.perimetre() ;
}

float Carre::surface()
{
     return boite.surface() ;
}
```

Compilation en ligne

Commandes de compilation

```
Rigel > g++ -c main.cpp -o main.o
Rigel > g++ -c Carre.cpp -o Carre.o
Rigel > g++ -c Cercle.cpp -o Cercle.o
Rigel > g++ -c Rectangle.cpp -o Rectangle.o
Rigel > g++ main.o Rectangle.o Cercle.o Carre.o
        -o figure.out
Rigel > ./figure.out
```

autre commande possible

```
Rigel > g++ main.cpp Rectangle.cpp Cercle.cpp Carre.cpp
        -o figure.out
Rigel > ./figure.out
```


Spécification

On veut bâtir un nouveau type de données qui soit une abstraction des nombres rationnels.

Analyse/conception

- Un nombre rationnel est un nombre constitué d'un numérateur et d'un dénominateur.
- Opérations :
  addition, multiplication, égalité, convertion pour affichage.
- Fonctions utilitaires :
  mettre au méme dénominateur et simplifier.

Implémentation du T.A.D. **Rationnel** - rationnel.h

```
#include <iostream>

class Rationnel
{
    int num ;
    int denom ;
    void simplifier() ;
    void m_deno ( Rationnel& ) ;
 public :
    Rationnel(int n=0,int d=1); //valeur par defaut
    void lecture()
    void imprime();
    Rationnel add (Rationnel) ;
    Rationnel mul (Rationnel) ;
    bool est_egal (Rationnel) ;
    string en_string () ;
};
```

Implémentation du T.A.D. **Rationnel** - rationnel.cpp

```
Rationnel::Rationnel(int n , int d)
{
    num =   n ;
    if(d != 0) denom  = d ;
    else
    {
       cerr <<"denominateur nul" << endl ;
       num = 0  ;
       denom = 1 ;
    }
}
string Rationnel::en_string()
{
   return entier_en_string(numerateur) + "/"
                 + entier_en_string(denominateur) ;
}
```


Implémentation du T.A.D. **Rationnel** - rationnel.cpp

```
void Rationnel::lecture()
{
    int n,d;
    cin >> n >> d;
    num =   n ;
    if(d != 0) denom  = d ;
    else
    {
       cerr <<"denominateur nul" << endl ;
       num = 0  ;
       denom = 1 ;
    }
}
void Rationnel::imprime()
{
   cout << num << "/" << denom ;
}
```


Implémentation du T.A.D. **Rationnel** - rationnel.cpp

```
void Rationnel::simplifier()
{
   int  i, j ;
   if( num == 0 ) denom = 1 ;
   else {
      i = num ;
      if( i < 0 )  i = -i ;
      j =  denom ;

      //recherche du plus grand commun diviseur
      while (i  !=  j) {
        if (i > j) i = i - j ;
        else j = j - i ;
      }
      num = num / i ;
      denom = denom / i  ;
  }
}
```


Implémentation du T.A.D. **Rationnel** - rationnel.cpp

```
void Rationnel::m_deno( Rationnel& r)
{
  if ( denom !=   r.denom ) {
     num  = num * r.denom ;
     r.num  = r.num * denom ;
     denom =  denom * r.denom ;
     r.denom = denom ;
  }
}
Rationnel Rationnel::add ( rationnel r )
{
   Rationnel t ;
   t.num  = num ;
   t.denom = denom ;
   t.m_deno(r) ;
   t.num =  t.num + r.num ;
   t.simplifier() ;
   return t ;
}
```

Implémentation du T.A.D. **Rationnel** - rationnel.cpp

```
Rationnel Rationnel::mul( Rationnel r)
{
  Rationnel t  ;
  t.num = num*r.num ;
  t.denom = denom*r.denom ;
  t.simplifier() ;
  return t ;
}
bool  Rationnel::est_egal(Rationnel  r)
{
   Rationnel t1,  t2 ;
   t1.num = num  ;
   t1.denom = denom  ;
   t2 = r ;
   t1.m_deno(t2)  ;
   return( t1.num  == t2.num )  ;
}
```


Type rationnel - Utilisation

```
void main()
{
  Rationnel  nb1(2, 3), nb2(-4, 0), nb3(2, 3);
  Rationnel  a(2, 3), b(1, 3), c(3, 4)  ;

  cout << "nb1 vaut: "; nb1.impr();
  cout << endl ;
  cout << "nb2 vaut: "; nb2.impr();
  cout << endl ;
  cout << "nb3 vaut: "; nb3.impr();
  cout << endl ;

  nb2 = nb1.mul(nb3) ;
  cout << "nb2 vaut: "; nb2.impr();
  cout << endl ;

  nb2 = nb1.add(b) ;
  cout << "nb2 vaut: " ; nb2.impr() ;
  cout<< endl ;
```


Type rationnel - Utilisation

```
  nb2 = nb1.add(c) ;
  cout << "nb2 vaut: " ; nb2.impr() ;
  cout << endl ;

  cout << "Les deux rationnels sont " << endl ;
  if (a.egal(c))
       cout << "egaux" << endl ;
  else
       cout << "differents" << endl ;

  cout << "Les deux rationnels sont " << endl ;
  if (a.egal(nb3))
       cout << "egaux" << endl ;
  else
     cout << "differents" << endl ;

}
```


Remarques

- Les objets ne s'utilisent pas de la méme maniére que les variables. cote = 0 ;
  Exemples :
  `int i, j, k ;` ⟶ `i = j + k ;`

  `Rationnel nb1, nb2, nb3`
  ⟶ `nb3 = nb1.add(nb2) ;`
  et non `nb3 = nb1 + nb2 ;`
- On peut changer cela ! ! ! !

Surcharge d'opérateurs.

- Tous les opérateurs de base du langage peuvent étre redéfinis pour de nouveaux types définis via des classes.
- Tout opérateur garde sa priorité et s'il était binaire reste binaire (comme unaire et ternaire).
- Pour surcharger un opérateur il faut utiliser le mot réservé « operator » suivi de l'opération comme nom de méthode. Exemple : « operator+ »

Implémentation du T.A.D. **Rationnel** - rationnel.h

```
#include <iostream>

class Rationnel {
    int num ;
    int denom ;
    void simplifier() ;
    void m_deno (Rationnel&) ;

public :
    Rationnel(int n = 0, int d = 1);
    Rationnel operator+(Rationnel) ;
    Rationnel operator*(Rationnel) ;
    bool operator==(Rationnel) ;
    string en_string () ;
} ;
```


Implémentation du T.A.D. **Rationnel** - rationnel.cpp

```
Rationnel::Rationnel(int n , int d)
{
    num =   n ;
    if(d != 0) denom  = d ;
    else
    {
       cerr <<"denominateur nul" << endl ;
       num = 0  ;
       denom = 1 ;
    }
}

string Rationnel::en_string()
{
   return entier_en_string(numerateur) + "/"
                + entier_en_string(denominateur) ;
}
```


Implémentation du T.A.D. **Rationnel** - rationnel.cpp

```
void Rationnel::simplifier()
{
  int  i, j ;
  if( num == 0 ) denom = 1 ;
  else {
     i = num ;
     if( i < 0 )  i = -i ;
     j =  denom ;
     //recherche du plus grand commun diviseur
     while ( i  !=  j )  {
        if (i > j) i = i - j ;
        else j = j - i ;
     }
     num = num / i ;
     denom = denom / i  ;
  }
}
```

Implémentation du T.A.D. **Rationnel** - rationnel.cpp

```
void Rationnel::m_deno( Rationnel& r)
{
  if ( denom !=   r.denom ) {
       num  = num * r.denom ;
       r.num  = r.num * denom ;
       denom =  denom * r.denom ;
       r.denom = denom ;
  }
}
Rationnel Rationnel::operator+(Rationnel r)
{
   Rationnel t ;
   t.num  = num ;
   t.denom = denom ;
   t.m_deno(r) ;
   t.num =  t.num + r.num ;
   t.simplifier() ;
   return t ;
}
```


Implémentation du T.A.D. **Rationnel** - rationnel.cpp

```
Rationnel Rationnel::operator*(Rationnel r)
{
    Rationnel t  ;
    t.num = num*r.num ;
    t.denom = denom*r.denom ;
    t.simplifier() ;
    return t ;
}
booleen  Rationnel::operator==(Rationnel  r)
{
    Rationnel t1,  t2 ;
    t1.num = num  ;
    t1.denom = denom  ;
    t2 = r ;
    t1.m_deno(t2)  ;
    return( t1.num  == t2.num )  ;
}
```


Type Rationnel - Utilisation

```
void main()
{
   Rationnel  nb1(2, 3), nb2(-4, 0), nb3(2, 3);
   Rationnel  a(2, 3), b(1, 3), c(3, 4)  ;

   cout << "nb1 vaut: " ; nb1.impr();
   cout << endl ;
   cout << "nb2 vaut: " ; nb2.impr();
   cout << endl ;
   cout << "nb3 vaut: " ; nb3.impr();
   cout << endl ;

   nb2 = nb1 * nb3 ;
   cout << "nb2 vaut: "; nb2.impr();
   cout << endl ;

   nb2 = nb1 + b ;
   cout << "nb2 vaut: "; nb2.impr();
   cout<< endl ;
```


Type Rationnel - Utilisation

```
   nb2 = nb1 + c ;
   cout << "nb2 vaut: "; nb2.impr();
   cout << endl ;

   cout << "Les deux rationnels sont " << endl ;

   if(a == c)  cout << "egaux" << endl ;
   else
     cout << "differents" << endl ;

   cout << "Les deux rationnels sont " << endl ;

   if(a == nb3) cout << "egaux" << endl ;
   else
     cout << "differents" << endl ;
}
```

Surcharge << et >>

- Il est possible de surcharger les opérateurs << et >>
- Cela diffère toutefois des autres car l'objet est à droite et non à gauche de l'opérateur.

Exemples de la figure

```
Cercle rond ;
   ...
  cout << "Entrez les donnees du cercle: " ;
  cin >> rond;
  cout << "Le cercle de rayon " << rond
       << "cms a pour perimetre "
       << rond.perimetre() << "cms"
       << endl <<"       et pour surface "
       << rond.surface() << "cms2" << endl ;
  break ;
```


Exemples de la figure

```
void Cercle::lecture(istream& in)
{
     in >> rayon ;
}

void Cercle::ecrire(ostream& out)
{
     out << rayon ;
}
```


Exemples de la figure

```
istream& operator>>(istream& is, Cercle& rd)
{
   rd.lecture(is);
   return is;
}

ostream& operator<<(ostream& os, Cercle rd)
{
   rd.ecrire(os);
   return os;
}
```