IFT159
Analyse et programmation
Thème 8 — Introduction aux types abstraits

Gabriel Girard

Département d'informatique
UNIVERSITÉ DE SHERBROOKE

16 novembre 2015

Thème 8 — Introduction aux types abstraits

Organisation des données et types

Introduction à UML

- But : définir un processus/méthode de développement complet (de l'analyse à l'implémentation) orienté objet

Introduction à UML

- But : définir un processus/méthode de développement complet (de l'analyse à l'implémentation) orienté objet
- UML est un langage graphique qui permet de représenter et de communiquer les divers aspects d'un système logiciel

Introduction à UML

- But : définir un processus/méthode de développement complet (de l'analyse à l'implémentation) orienté objet
- UML est un langage graphique qui permet de représenter et de communiquer les divers aspects d'un système logiciel
- Il permet une visualisation complète d'un système

Introduction à UML

- UML fournit une notation/syntaxe pour les diagrammes et modèles définis pendant tout le cycle de développement

Introduction à UML

- UML fournit une notation/syntaxe pour les diagrammes et modèles définis pendant tout le cycle de développement
- Les diagrammes et les modèles offrent un ensemble de points de vues complémentaires

Introduction à UML

- UML fournit une notation/syntaxe pour les diagrammes et modèles définis pendant tout le cycle de développement
- Les diagrammes et les modèles offrent un ensemble de points de vues complémentaires
- Cela encourage un développement par raffinements successifs

Introduction à UML

- UML fournit une notation/syntaxe pour les diagrammes et modèles définis pendant tout le cycle de développement
- Les diagrammes et les modèles offrent un ensemble de points de vues complémentaires
- Cela encourage un développement par raffinements successifs
- UML est basé sur un méta-modèle
  il permet d'exprimer des modèles objet en faisant abstraction de leur implémentation
  (indépendance envers les langages de programmation)

Introduction à UML

- Programmation sans programmer
  UML est un langage de modélisation utilisable à la fois par les humains et les machines qui sert à modéliser des applications construites à l'aide d'objets, indépendant de la méthode utilisée

Introduction à UML

- Programmation sans programmer
  UML est un langage de modélisation utilisable à la fois par les humains et les machines qui sert à modéliser des applications construites à l'aide d'objets, indépendant de la méthode utilisée
- Rappelons qu'un modèle est une représentation abstraite de la réalité qui exclut certains détails du monde réel.

Introduction à UML

- Programmation sans programmer
  UML est un langage de modélisation utilisable à la fois par les humains et les machines qui sert à modéliser des applications construites à l'aide d'objets, indépendant de la méthode utilisée
- Rappelons qu'un modèle est une représentation abstraite de la réalité qui exclut certains détails du monde réel.
- UML est un langage

Introduction à UML

- Programmation sans programmer
  UML est un langage de modélisation utilisable à la fois par les humains et les machines qui sert à modéliser des applications construites à l'aide d'objets, indépendant de la méthode utilisée
- Rappelons qu'un modèle est une représentation abstraite de la réalité qui exclut certains détails du monde réel.
- UML est un langage
  - ne propose pas un processus de développement

Introduction à UML

- Programmation sans programmer
  UML est un langage de modélisation utilisable à la fois par les humains et les machines qui sert à modéliser des applications construites à l'aide d'objets, indépendant de la méthode utilisée
- Rappelons qu'un modèle est une représentation abstraite de la réalité qui exclut certains détails du monde réel.
- UML est un langage
  - ne propose pas un processus de développement
  - ni d'ordonnancement des tâches

UML - diagrammes

- Diagrammes UML (vues statiques)
  - Cas d'utilisation
  - D'objets
  - De classes
  - De composants
  - De déploiement

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

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

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

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 à modéliser

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 à modéliser
- Identifie les fonctionnalités principales (critiques) du système

Diagramme de cas d'utilisation : Éléments

- Acteurs : entité externe qui agit sur le 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

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

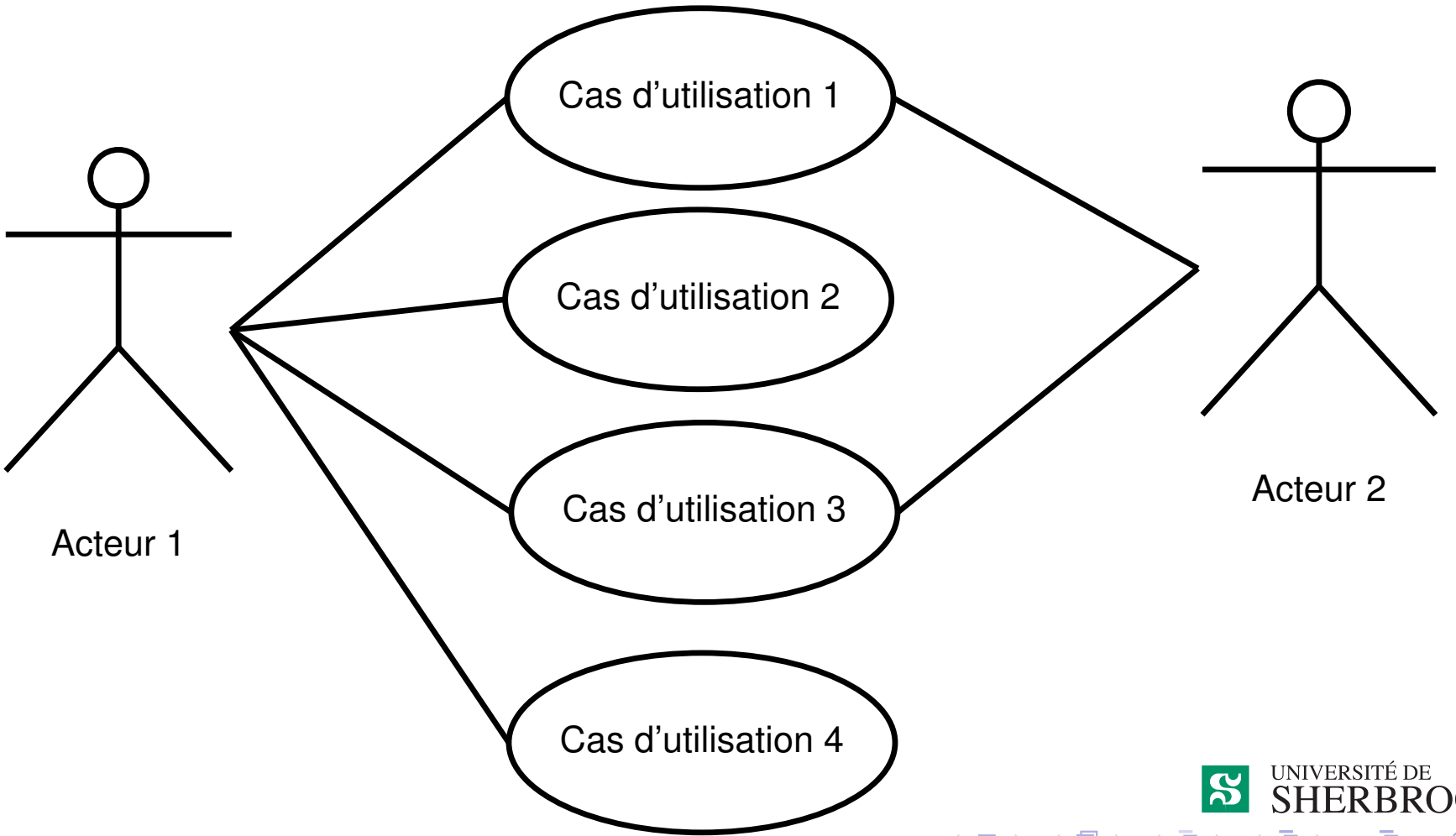


Diagramme de classes

- Une collection d'éléments de modélisation statiques qui montre la structure d'un modèle

Diagramme de classes

- Une collection d'éléments de modélisation statiques qui montre la structure d'un modèle
- Fait abstraction des aspects dynamiques et temporels

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

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

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
  Une relation d'association représente une relation sémantique entre les objets d'une classe.

Diagramme de classes : Éléments

Types de relation :

- Association
  Une relation d'association représente une relation sémantique entre les objets d'une classe.
- Agrégation :
  Une relation d'agrégation décrit une relation de contenance (conteneur/contenu) ou de composition.

Diagramme de classes : Éléments

Types de relation :

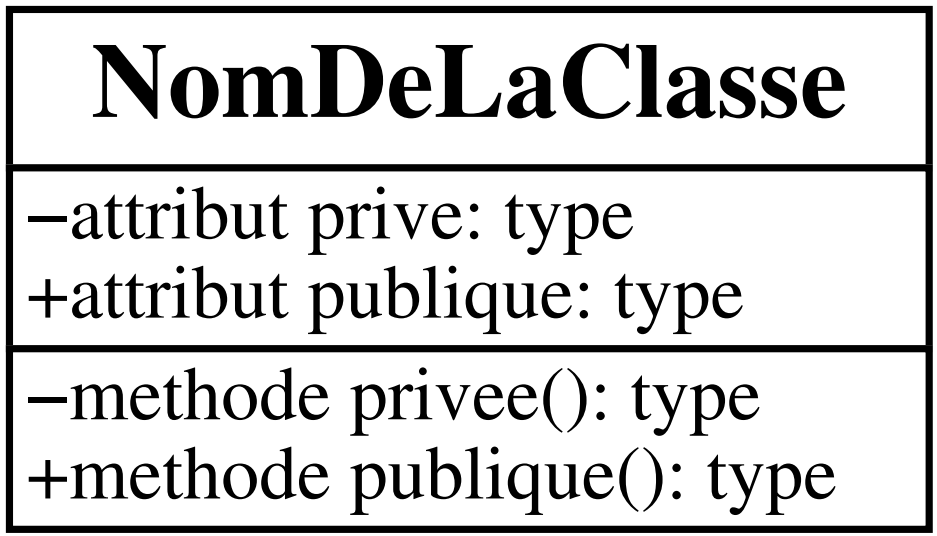
- Association
  Une relation d'association représente une relation sémantique entre les objets d'une classe.
- Agrégation :
  Une relation d'agrégation décrit une relation de contenance (conteneur/contenu) ou de composition.
  - Agrégation faible
    Si le conteneur est détruit (ou copié), le contenu ne l'est pas

Diagramme de classes : Éléments

Types de relation :

- Association
  Une relation d'association représente une relation sémantique entre les objets d'une classe.
- Agrégation :
  Une relation d'agrégation décrit une relation de contenance (conteneur/contenu) ou de composition.
  - Agrégation faible
    Si le conteneur est détruit (ou copié), le contenu ne l'est pas
  - Agrégation forte (composition)
    Si le conteneur est détruit (ou copié), le contenu l'est aussi

Diagramme de classes : Symboles



Associations : Symboles

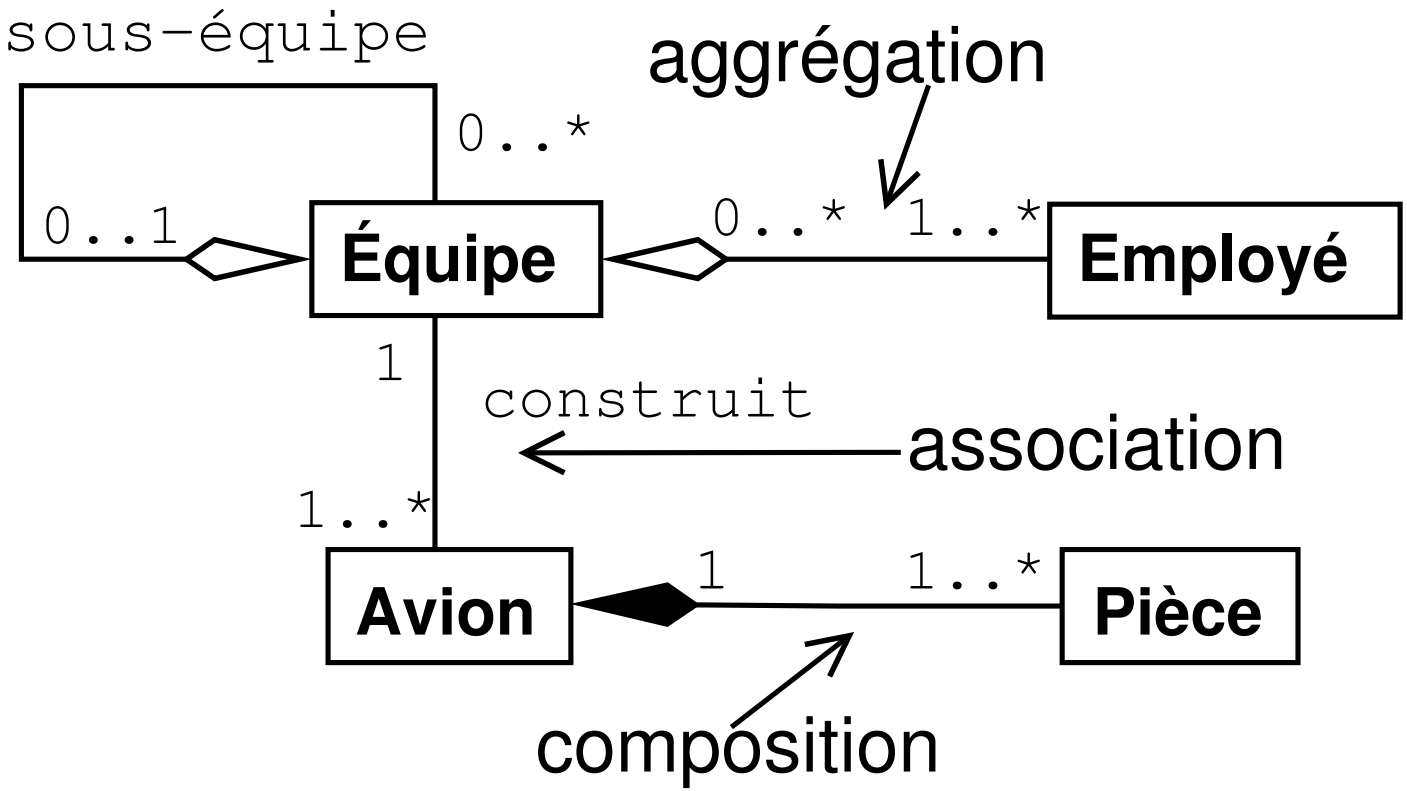


Diagramme de séquence

- Représenter des collaborations entre objets selon un point de vue temporel

Diagramme de séquence

- Représenter des collaborations entre objets selon un point de vue temporel
- Expression des interactions

Diagramme de séquence

- 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

Diagramme de séquence

- 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

Diagramme de séquence

- 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 : Éléments

- Acteurs : mêmes entités que dans les cas d'utilisations

Diagramme de séquence : Éléments

- Acteurs : mêmes entités que dans les cas d'utilisations
- Objets : instanciés pour réaliser le scénario

Diagramme de séquence : 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

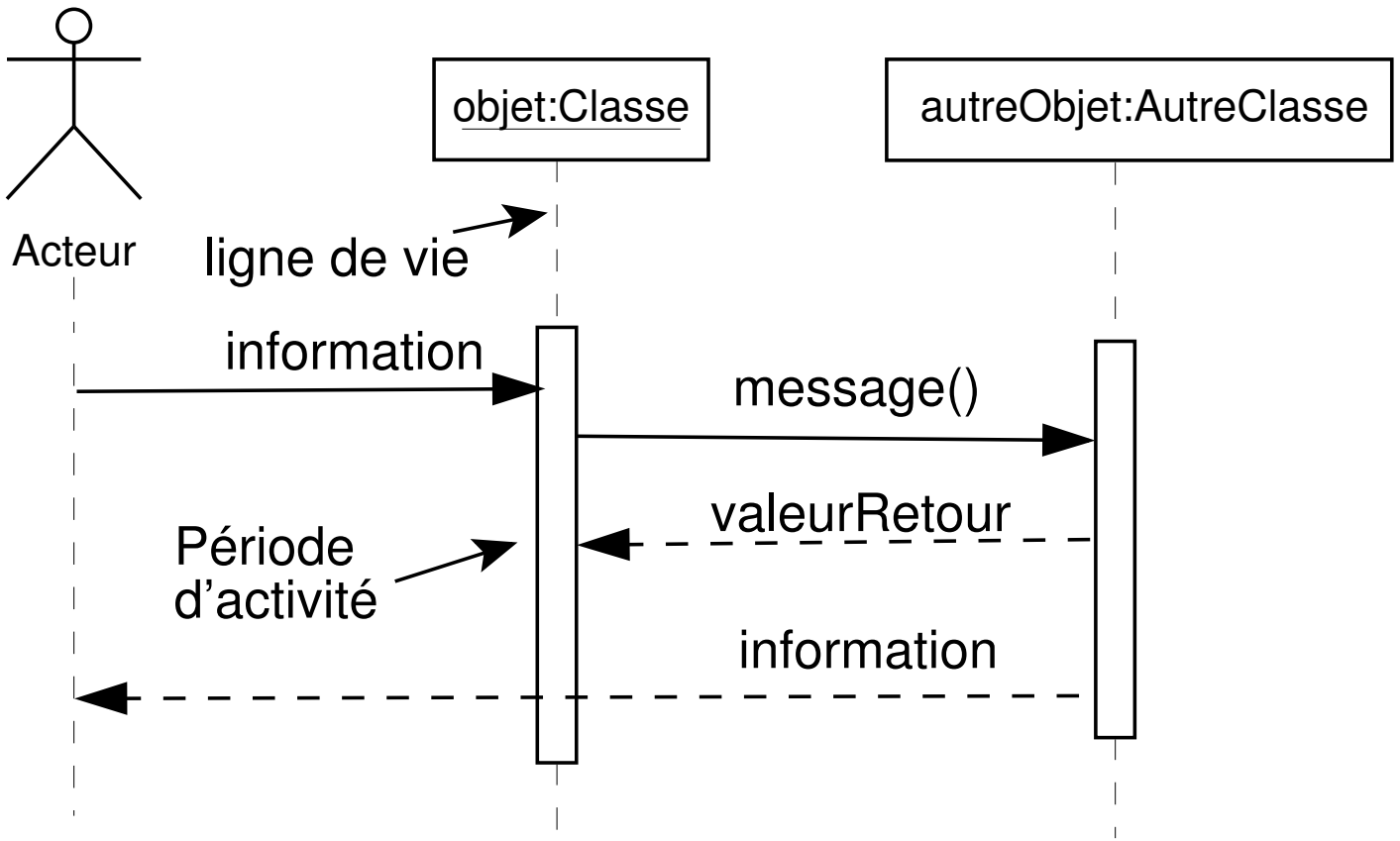
Diagramme de séquence : Éléments

- 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

Diagramme de séquence : 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 à la suite d'un appel

Diagramme de séquence : Symboles

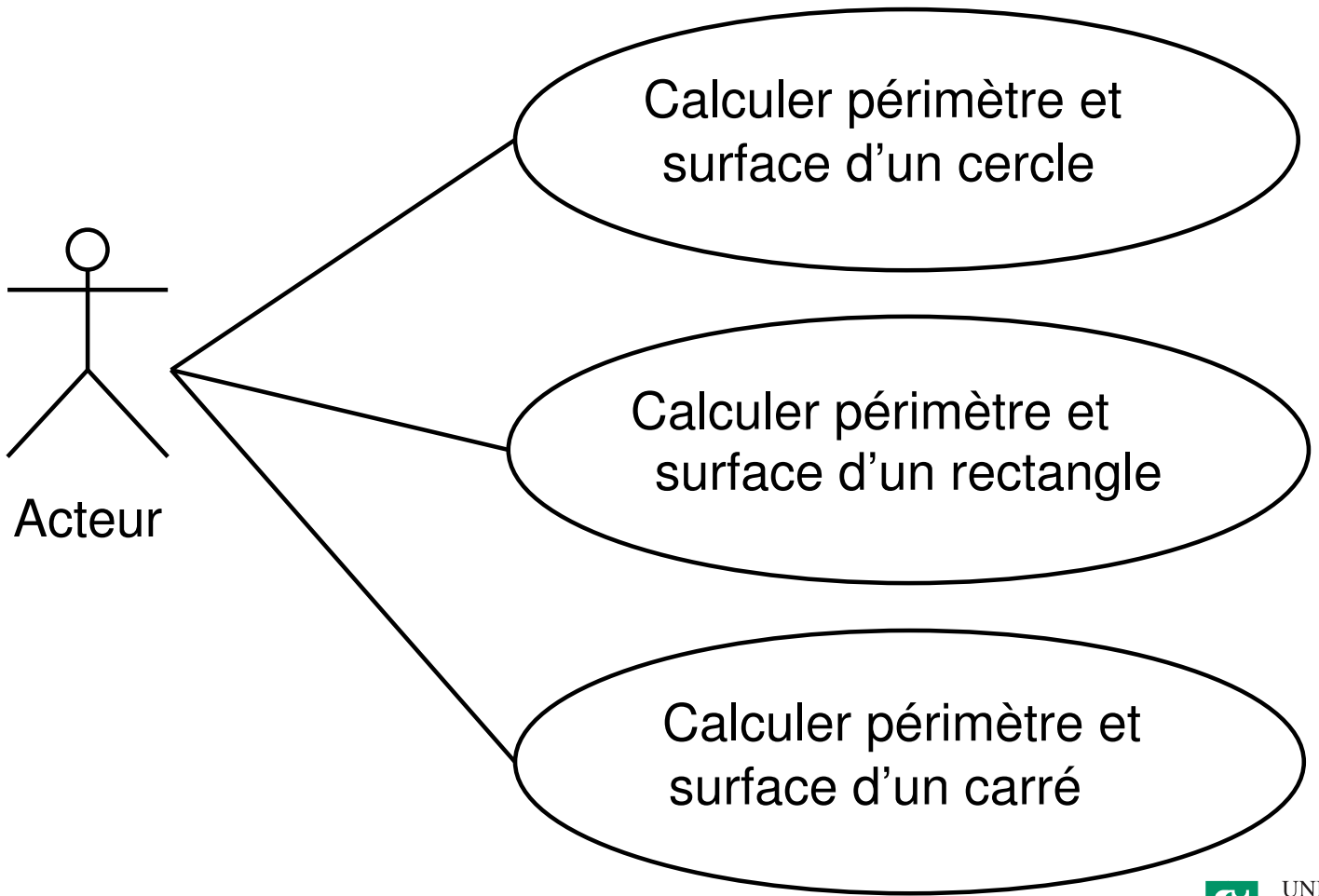


Organisation des données et types

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.

Analyse/conception - Les nouveaux types

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

Analyse globale

Analyse globale

1. Entrées
    1. (clavier) Choix (Caratères)

Analyse globale

1. Entrées
    1. (clavier) Choix (Caratères)
    2. (clavier) Dimension (cercle, rectangle, carré)

Analyse globale

1. Entrées
   1. (clavier) Choix (Caratères)
   2. (clavier) Dimension (cercle, rectangle, carré)
2. Sorties

Analyse globale

1. Entrées
   1. (clavier) Choix (Caratères)
   2. (clavier) Dimension (cercle, rectangle, carré)
2. Sorties
   1. (écran) périmètre (réel)

Analyse globale

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

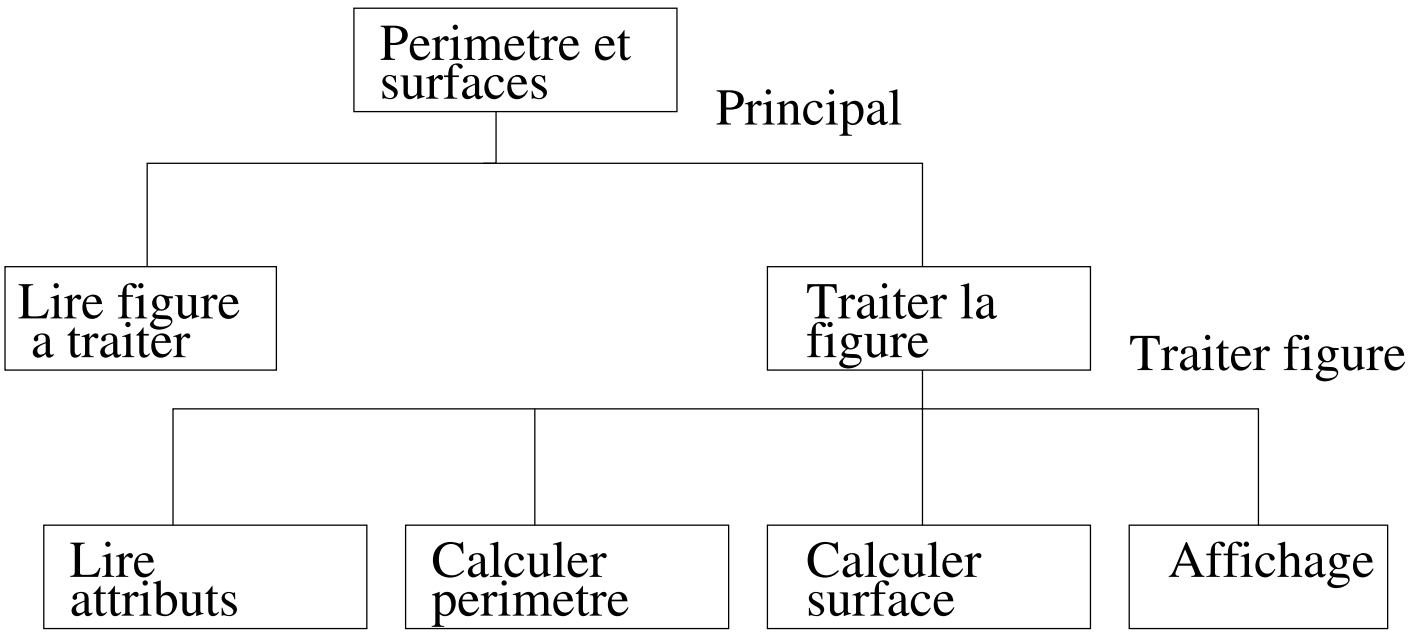
Analyse globale

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

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 ...

Analyse/conception - diagramme



Conception module principal

algorithme

1. Pour chaque choix (trois choix)
   - 1.1 Lire choix de l'utilisateur
   - 1.2 Traiter la figure

Conception module traiter figure

algorithme

1. Selon la figure choisie par l'utilisateur (`C, R, K`)
   - 1.1 Lire la figure
   - 1.2 Calculer le périmètre
   - 1.3 Calculer la surface
   - 1.4 Afficher
     - le périmètre
     - la surface

Analyse/conception - le cercle

T.A.D. cercle (Analyse)

- Un cercle est connu par son rayon

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

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)

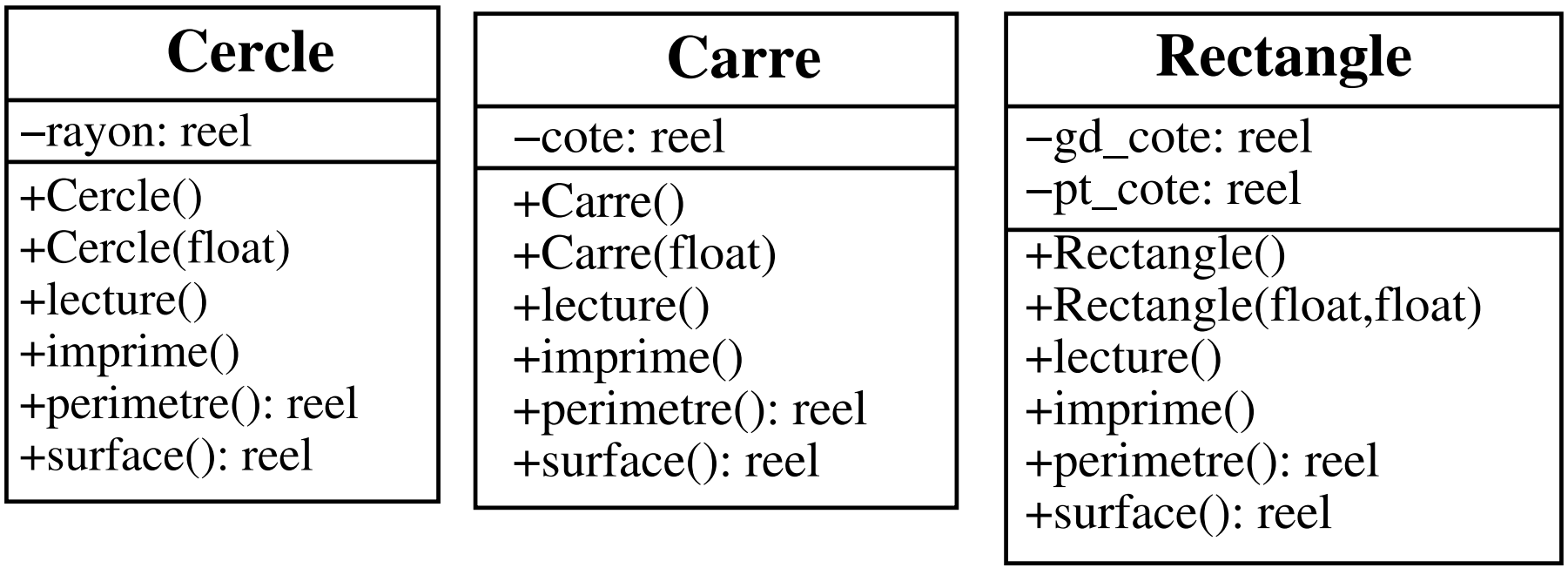
- Un carré est connu par son coté

Analyse/conception - le carré

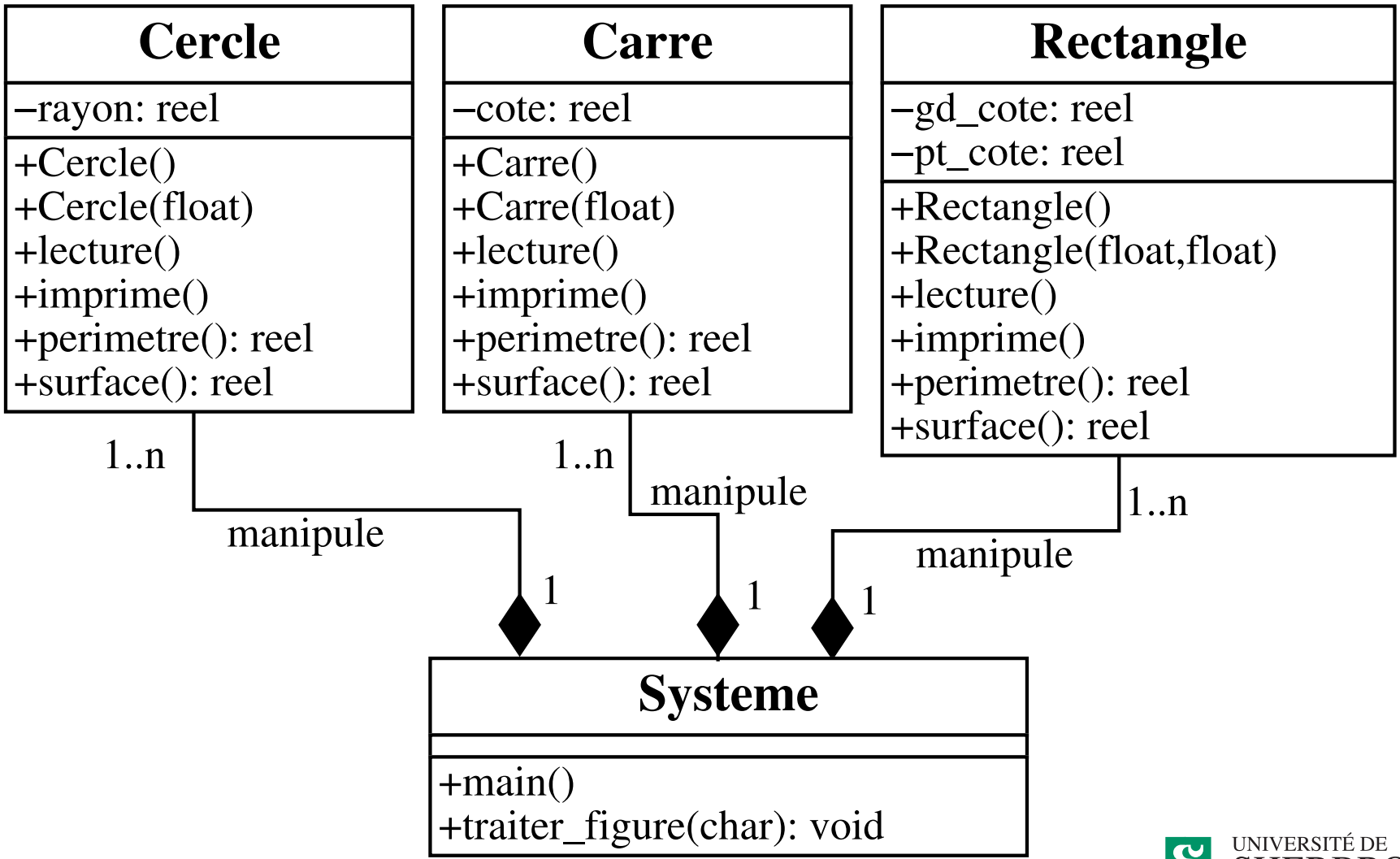
T.A.D. **Carré** (Analyse)

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

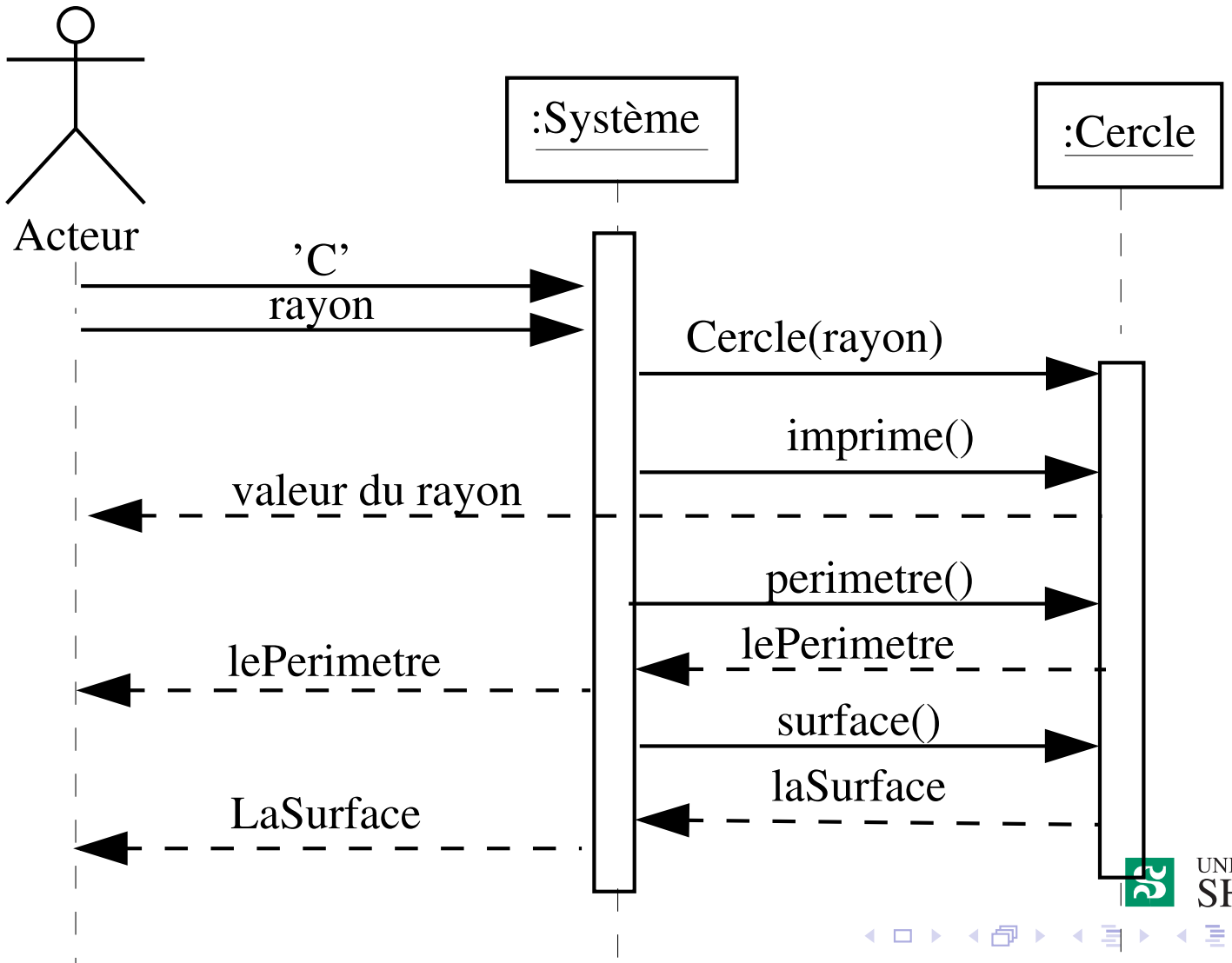
Analyse/conception - diagrammes de classes



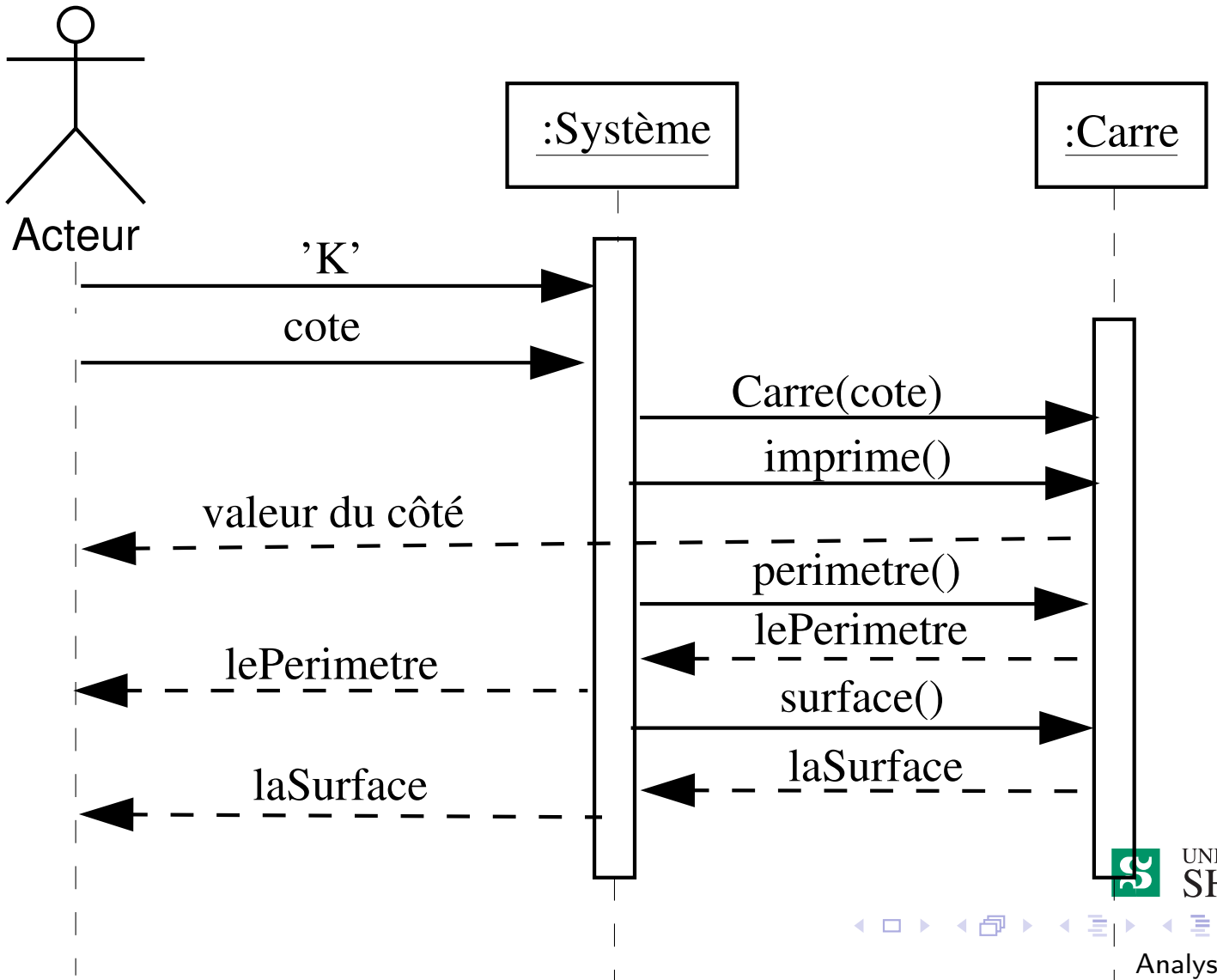
Analyse/conception



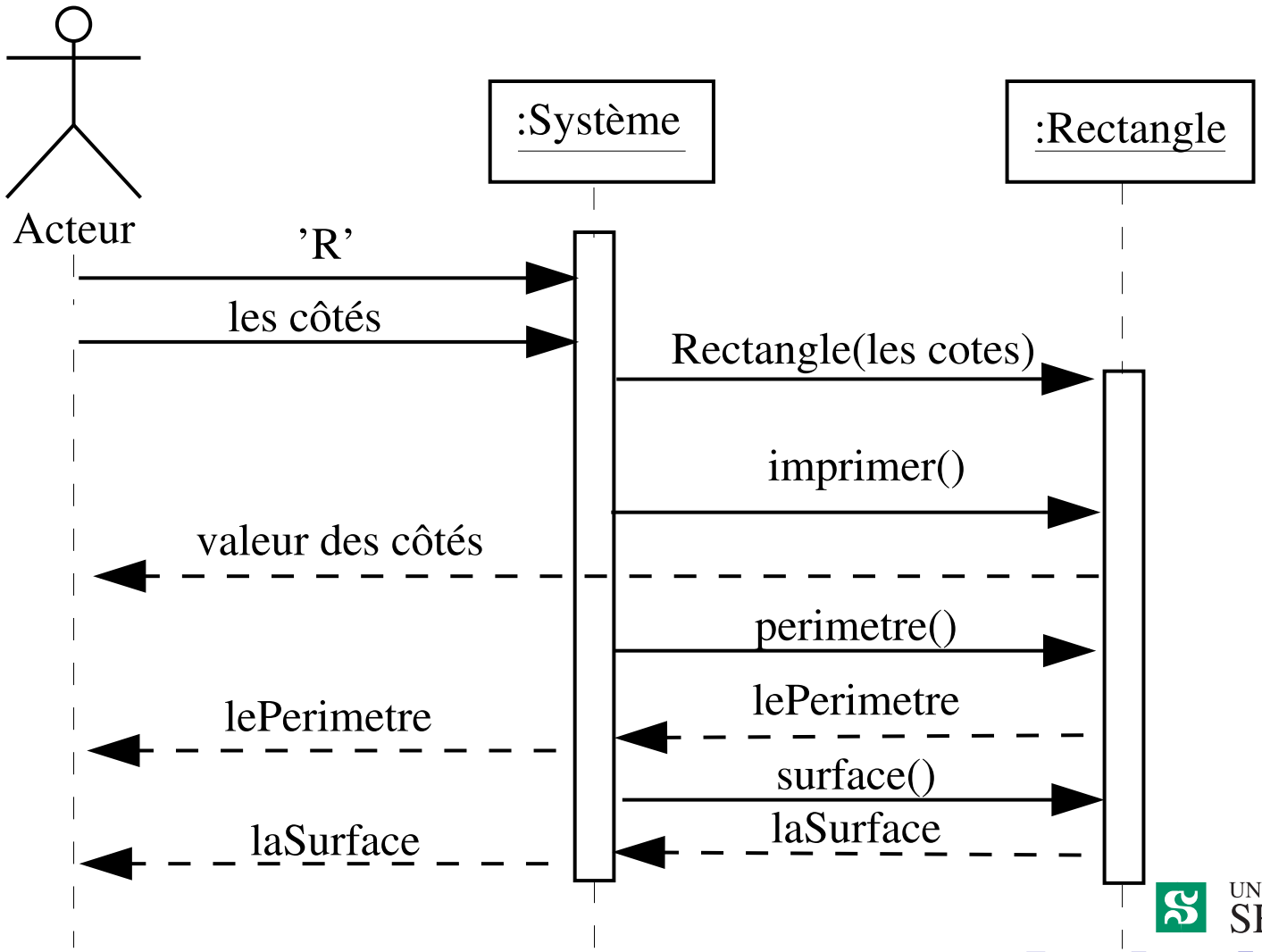
Diaglamme de séquence - scénario 1



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;

  switch(toupper(figure))
  {
   case 'C': cout << "Donner le rayon en cms: " ;
             cin >> rayon;
             Cercle rond(rayon);
             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: " ;
          cin >> gr_cote >> pt_cote;
          Rectangle boite(gr_cote, pt_cote);
          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: " ;
              cin >> cote;
              Carre de(cote);
              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 ;
    }
}
```

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

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).

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`.

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 IFT232 et 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 (idem sur Linux)

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

autre commande possible

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

Organisation des données et types

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.

Analyse/conception

- Un nombre rationnel est un nombre constitué d'un numérateur et d'un dénominateur.
- Opérations :
  addition, multiplication, égalité, convertion pour affichage.

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.

Diagramme de cas d'utilisation

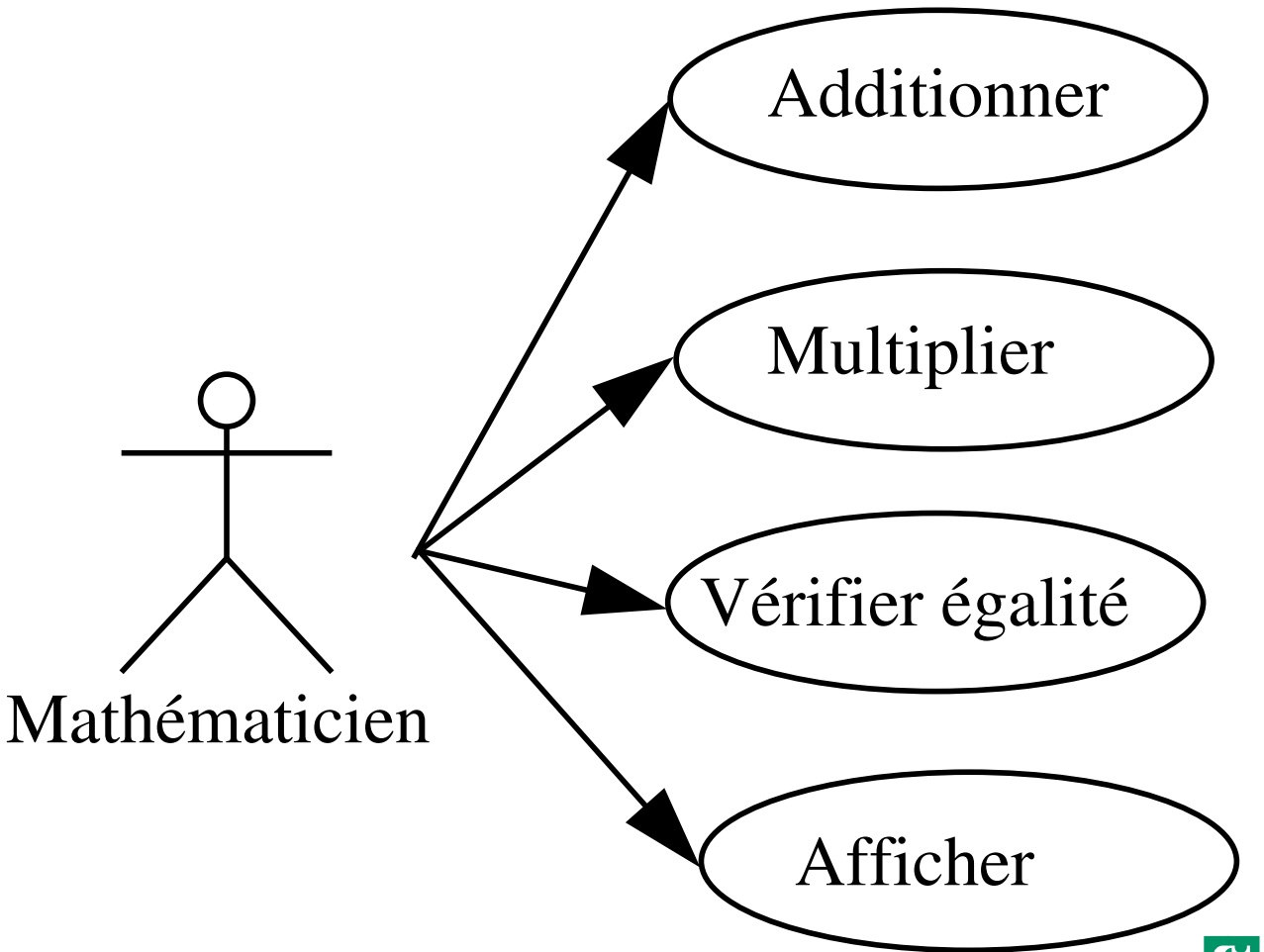
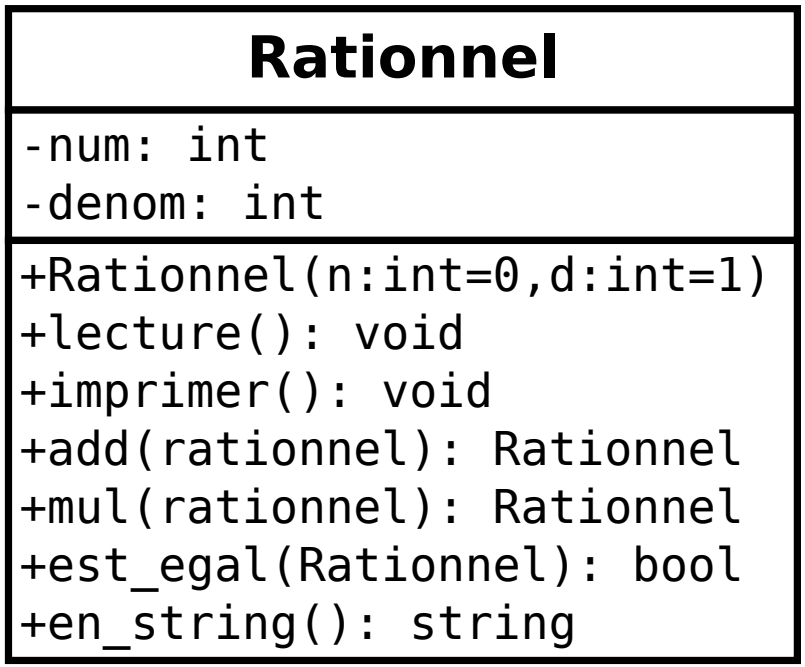


Diagramme de classe



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 même manière que les variables.

Exemple d'utilisation des variables

```
int i, j, k;
i = j + k;
```

Remarques

Les objets ne s'utilisent pas de la même manière que les variables.

Exemple d'utilisation des variables

```
int i, j, k;
i = j + k;
```

Exemple d'utilisation des objets

```
Rationnel nb1, nb2, nb3
nb3 = nb1.add(nb2);
```

Remarques

Les objets ne s'utilisent pas de la même manière que les variables.

Exemple d'utilisation des variables

```
int i, j, k;
i = j + k;
```

Exemple d'utilisation des objets

```
Rationnel nb1, nb2, nb3
nb3 = nb1.add(nb2);
```

Il serait intéressant de faire :

```
Rationnel nb1, nb2, nb3
nb3 = nb1 + nb2;
```

Organisation des données et types

Surcharge d'opérateurs.

- Tous les opérateurs de base du langage peuvent étre redéfinis pour de nouveaux types définis via des classes.

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).

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 >>

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;
}
```

Organisation des données et types

Exercice 1

On veut développer un programme qui gère un bottin téléphonique (nom et téléphone). Les opérations permises sur ce bottin sont :

- l'ajout

Exercice 1

On veut développer un programme qui gère un bottin téléphonique (nom et téléphone). Les opérations permises sur ce bottin sont :

- l'ajout
- le retrait

Exercice 1

On veut développer un programme qui gère un bottin téléphonique (nom et téléphone). Les opérations permises sur ce bottin sont :

- l'ajout
- le retrait
- la modification

Exercice 1

On veut développer un programme qui gère un bottin téléphonique (nom et téléphone). Les opérations permises sur ce bottin sont :

- l'ajout
- le retrait
- la modification
- la connaissance du nom à partir du téléphone

Exercice 1

On veut développer un programme qui gère un bottin téléphonique (nom et téléphone). Les opérations permises sur ce bottin sont :

- l'ajout
- le retrait
- la modification
- la connaissance du nom à partir du téléphone
- la connaissance de téléphone à partir du nom

Exercice 2

Reprenez le programme de gestion l'enregistrement liste que vous avez fait pour le laboratoire 8. Modifier le programme afin que l'enregistrement liste devienne la classe liste.

Exercice 2

```
#include "liste.h"
int main()
{
    liste_t liste1;
    int code;
    liste1.init();
    liste1.insere(1);
    liste1.insere(2);
    liste1.insere(3);
    liste1.insere(4);
    liste1.insere(5);
    liste1.affiche();
    code = liste1.recherche(3);
    if (code >= 0)
        cout << "Position de l'élément  est " << code << endl
    else cout << "Element 3 absent de la liste " << endl;
    code = liste1.retire(3);
    code = liste1.retire(4);
    liste1.affiche();
    return 0;
}
```

Exercice 3

Développer la classe complexe où les opérations permises sont : l'addition, la soustraction, la multiplication, la division, le conjugué, l'inverse, l'égalité en plus de pouvoir connaître la partie réelle, la partie imaginaire, la norme et l'argument. Tout complexe peut être fourni en coordonnées polaires ou cartésiennes.

- (a,b) + (c,d) = (a+c,b+d)

Exercice 3

Développer la classe complexe où les opérations permises sont : l'addition, la soustraction, la multiplication, la division, le conjugué, l'inverse, l'égalité en plus de pouvoir connaître la partie réelle, la partie imaginaire, la norme et l'argument. Tout complexe peut être fourni en coordonnées polaires ou cartésiennes.

- (a,b) + (c,d) = (a+c,b+d)
- (a,b) - (c,d) = (a-c, b-d)

Exercice 3

Développer la classe complexe où les opérations permises sont : l'addition, la soustraction, la multiplication, la division, le conjugué, l'inverse, l'égalité en plus de pouvoir connaître la partie réelle, la partie imaginaire, la norme et l'argument. Tout complexe peut être fourni en coordonnées polaires ou cartésiennes.

- (a,b) + (c,d) = (a+c,b+d)
- (a,b) - (c,d) = (a-c, b-d)
- (a,b) * (c,d) = (ac-bd, bc-ad)

Exercice 3

Développer la classe complexe où les opérations permises sont : l'addition, la soustraction, la multiplication, la division, le conjugué, l'inverse, l'égalité en plus de pouvoir connaître la partie réelle, la partie imaginaire, la norme et l'argument. Tout complexe peut être fourni en coordonnées polaires ou cartésiennes.

- (a,b) + (c,d) = (a+c,b+d)
- (a,b) - (c,d) = (a-c, b-d)
- (a,b) * (c,d) = (ac-bd, bc-ad)
- (a,b) / (c,d) = ((ac+bd)/(c2+d2) , (bc-ad)/(c2+d2))

Exercice 3

Développer la classe complexe où les opérations permises sont : l'addition, la soustraction, la multiplication, la division, le conjugué, l'inverse, l'égalité en plus de pouvoir connaître la partie réelle, la partie imaginaire, la norme et l'argument. Tout complexe peut être fourni en coordonnées polaires ou cartésiennes.

- (a,b) + (c,d) = (a+c,b+d)
- (a,b) - (c,d) = (a-c, b-d)
- (a,b) * (c,d) = (ac-bd, bc-ad)
- (a,b) / (c,d) = ((ac+bd)/(c2+d2) , (bc-ad)/(c2+d2))
- conjugue(a,b) = (a, -b)

Exercice 3

Développer la classe complexe où les opérations permises sont : l'addition, la soustraction, la multiplication, la division, le conjugué, l'inverse, l'égalité en plus de pouvoir connaître la partie réelle, la partie imaginaire, la norme et l'argument. Tout complexe peut être fourni en coordonnées polaires ou cartésiennes.

- (a,b) + (c,d) = (a+c,b+d)
- (a,b) - (c,d) = (a-c, b-d)
- (a,b) * (c,d) = (ac-bd, bc-ad)
- (a,b) / (c,d) = ((ac+bd)/(c2+d2) , (bc-ad)/(c2+d2))
- conjugue(a,b) = (a, -b)
- inverse(a,b) = (-a, -b)

Exercice 3

Développer la classe complexe où les opérations permises sont : l'addition, la soustraction, la multiplication, la division, le conjugué, l'inverse, l'égalité en plus de pouvoir connaître la partie réelle, la partie imaginaire, la norme et l'argument. Tout complexe peut être fourni en coordonnées polaires ou cartésiennes.

- (a,b) + (c,d) = (a+c,b+d)
- (a,b) - (c,d) = (a-c, b-d)
- (a,b) * (c,d) = (ac-bd, bc-ad)
- (a,b) / (c,d) = ((ac+bd)/(c2+d2) , (bc-ad)/(c2+d2))
- conjugue(a,b) = (a, -b)
- inverse(a,b) = (-a, -b)
- norme(a,b) = rac_carree(a2+b2)

Exercice 3

Développer la classe complexe où les opérations permises sont : l'addition, la soustraction, la multiplication, la division, le conjugué, l'inverse, l'égalité en plus de pouvoir connaître la partie réelle, la partie imaginaire, la norme et l'argument. Tout complexe peut être fourni en coordonnées polaires ou cartésiennes.

- (a,b) + (c,d) = (a+c,b+d)
- (a,b) - (c,d) = (a-c, b-d)
- (a,b) * (c,d) = (ac-bd, bc-ad)
- (a,b) / (c,d) = ((ac+bd)/(c2+d2) , (bc-ad)/(c2+d2))
- conjugue(a,b) = (a, -b)
- inverse(a,b) = (-a, -b)
- norme(a,b) = rac_carree(a2+b2)
- argument(a, b) = arctan(b/a)