R2.01
DEV2

Développement orienté objets

Etienne Carnovali **Pétra Gomez**

Farid Ammar-Boudjelal Jean-Michel Bohé Rouaa Wannous

Sommaire

Que la lumière soit !

nom de l'objet

caractéristiques

fonctionnalités

Ampoule
état puissance
allumer() éteindre() intensifier() diminuer()

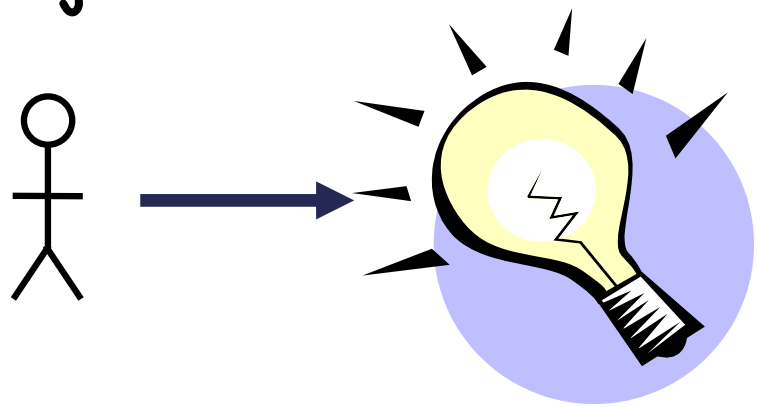
Un programme objet

- Un programme objet est un ensemble d'objets dialoguant entre eux en s'envoyant des messages.
- Un message est un appel de fonction appartenant à un objet particulier.

Ampoule lampe;

lampe.allumer();

lampe : Ampoule
état = OFF puissance = 10
allumer() éteindre() intensifier() diminuer()



Histoire d'eau !

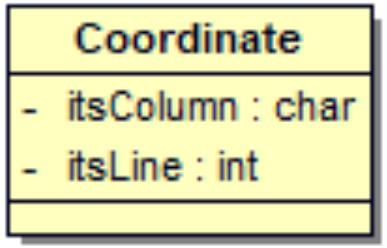
enregistrement

+

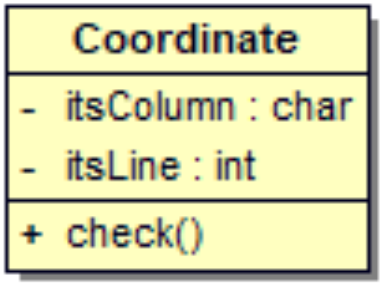
fonctions

=

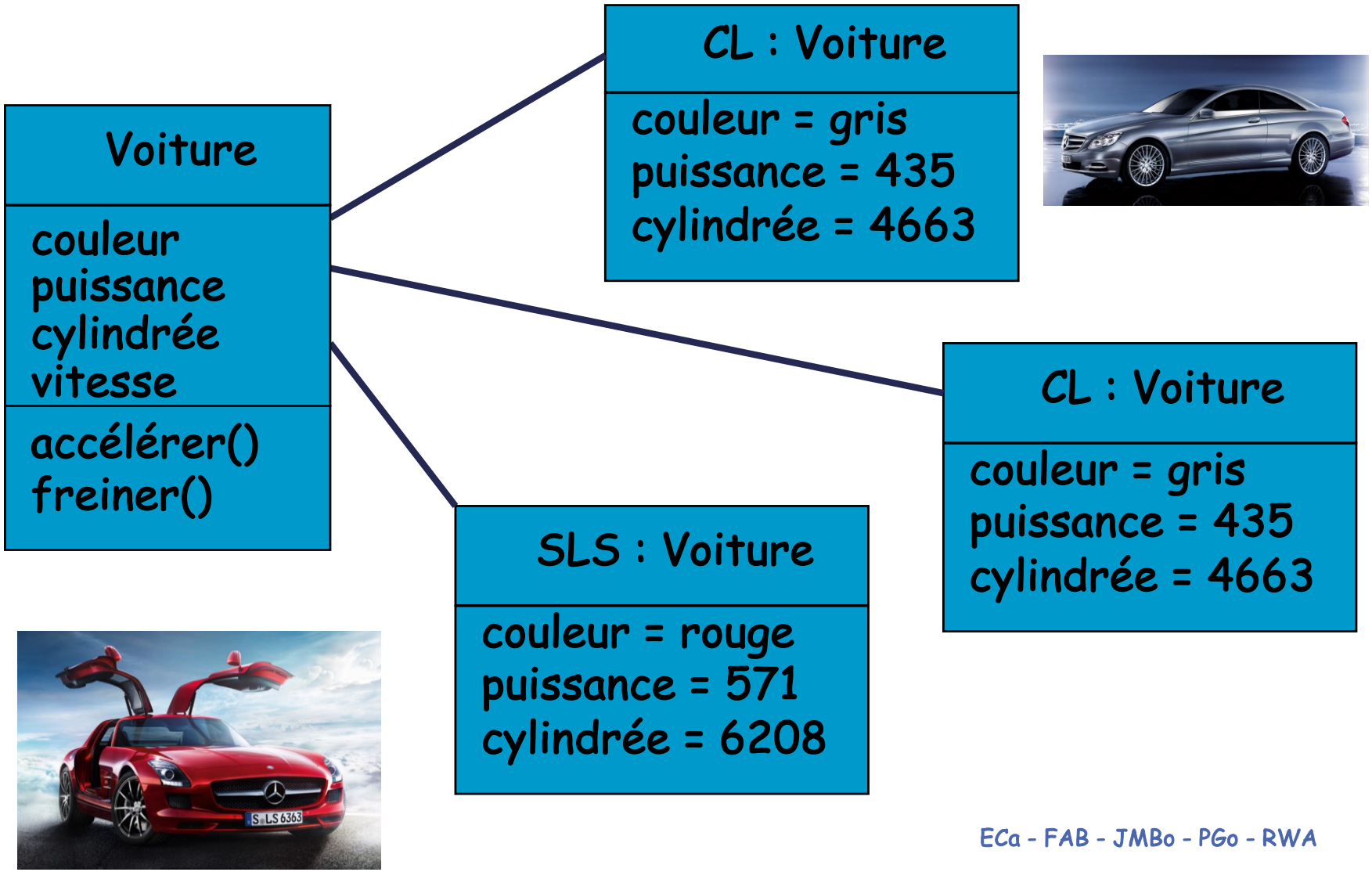
classe




```
bool  checkCoordinate
( string aPlace,
Coordinate& someCoordi )
```



Classes et instances (objets)



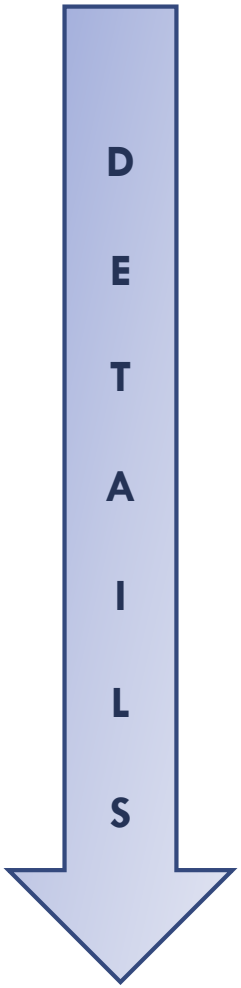
Classes et instances

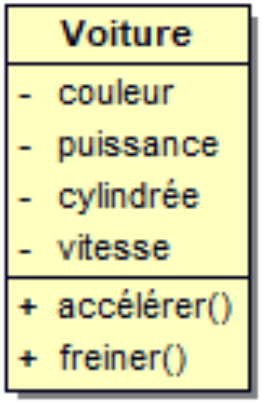
- Une classe est un modèle utilisé pour créer plusieurs objets présentant des caractéristiques communes.
- Un objet est une instance d'une classe donnée.

  - Une classe sert à définir toutes les fonctionnalités d'un objet, l'instance les réalise.
  - On peut créer autant d'instances que l'on veut à partir de la définition d'une classe.

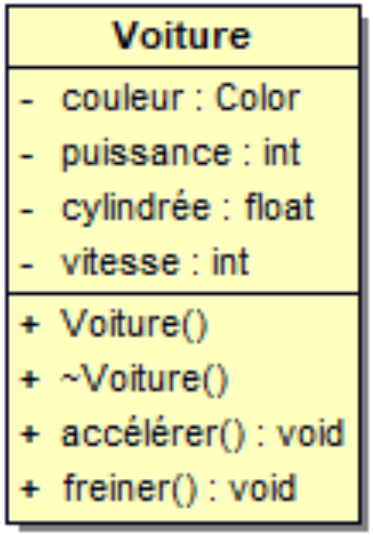
Différents niveaux de détails

- Diagramme d'analyse
  (modèle du domaine)
- Diagramme de conception
- Diagramme d'implémentation
  (au plus proche du langage de programmation)



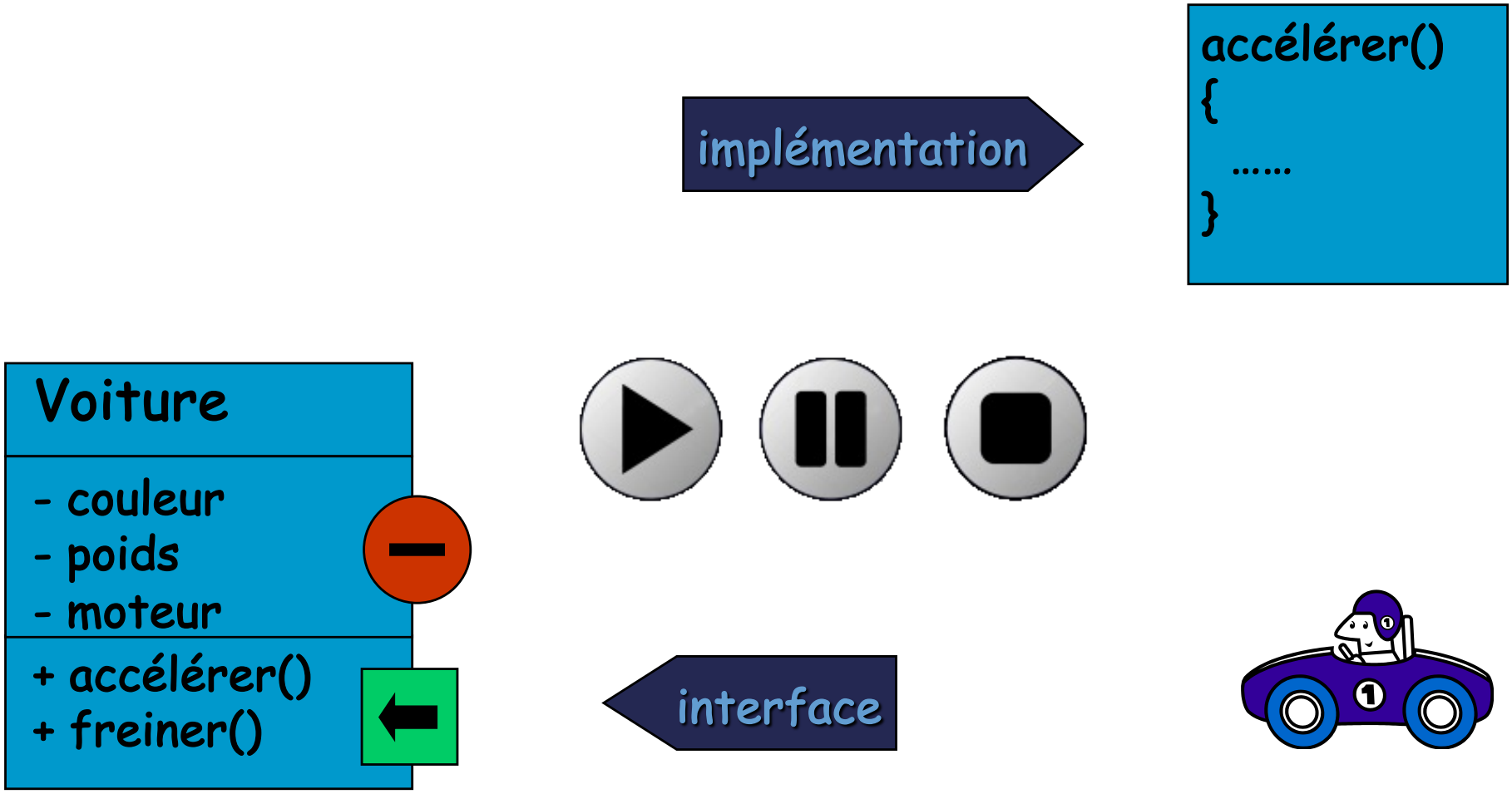


Conception Orientée Objet



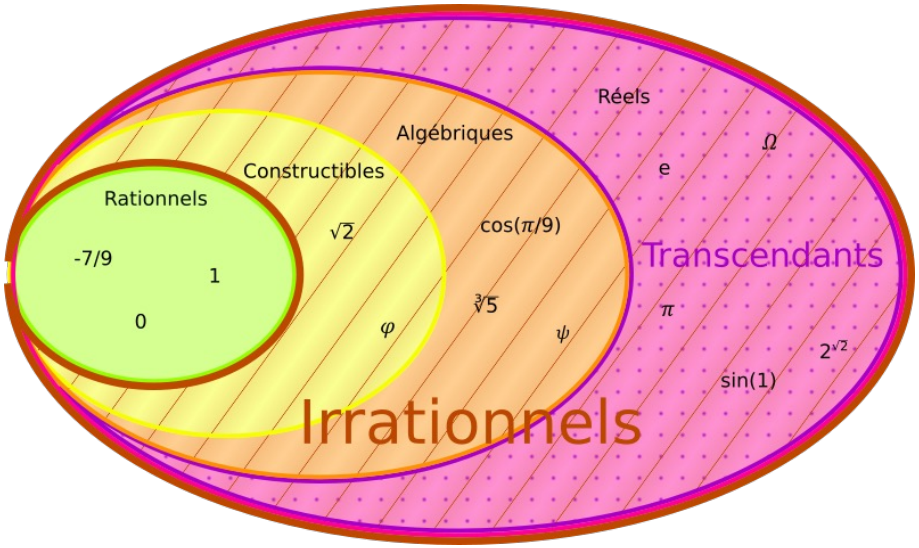
Programmation Orientée Objet

L'interface et l'implémentation



Exemple — fil rouge

- On souhaite créer une classe pour manipuler des nombres rationnels.



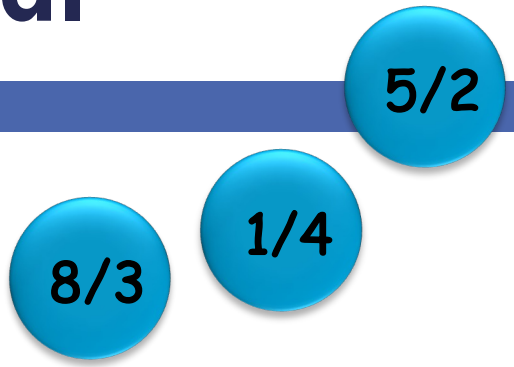
1 - Déterminer le signe de chaque nombre rationnel ?

$$\frac{(-2)\times 3\times(-4)}{6\times(-1)\times 7} \quad ; \quad \frac{0{,}5\times(-1{,}25)\times 13}{(-6)\times 27\times(-1{,}3)}$$

Les nombres rationnels

Maths 4éme

Classe des rationnels Rational

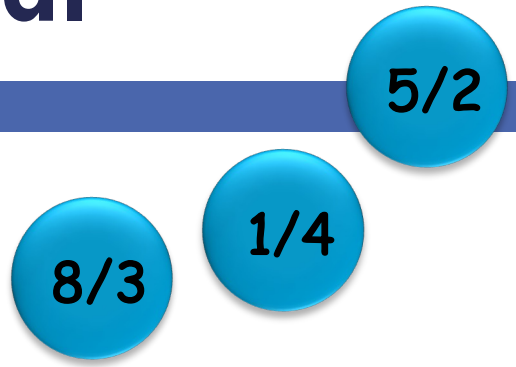


Rational
itsNum : int itsDen : int
initialize (int, int) : void convert() : double reverse() : void display() : void

Attributs
données membres

Méthodes
fonctions membres

Classe des rationnels Rational



- private

+ public

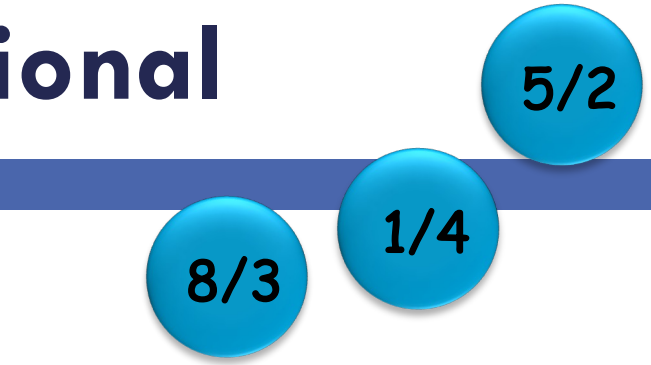
Rational
- itsNum : int - itsDen : int
+ initialize (int, int) : void + convert() : double + reverse() : void + display() : void

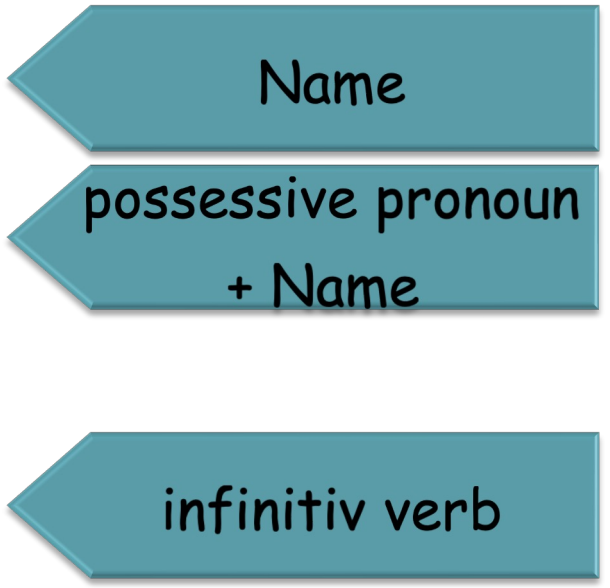
encapsulation

Garantir l'intégrité des données contenues dans l'objet en empêchant l'accès aux données par un autre moyen que les services proposés soit en masquant l'implémentation des données.

Classe des rationnels Rational

Rational
- itsNum : int - itsDen : int
+ initialize (int, int) : void + convert() : double + reverse() : void + display() : void





Rational : programmation

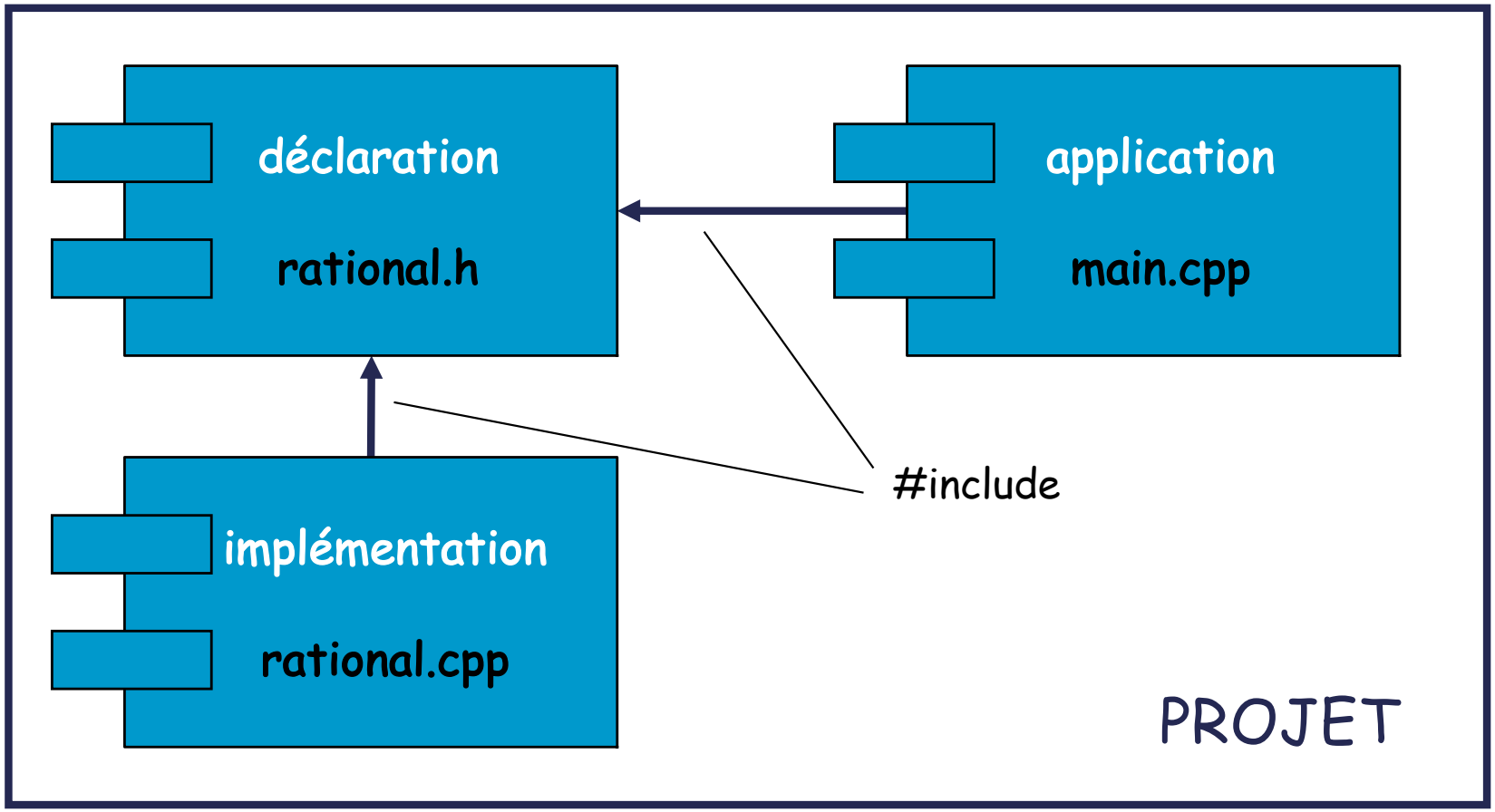


Diagramme de composants UML

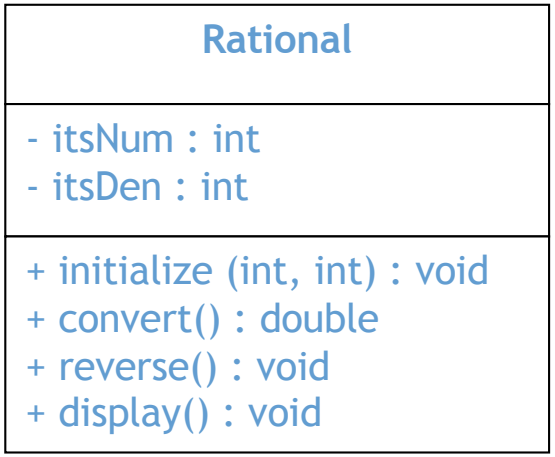
Rational : déclaration

rational.h *déclaration*

déclaration des fonctions

```
class Rational
{
    private :
        int itsNum ;
        int itsDen ;

    public :
        void initialize(int, int) ;
        double convert() ;
        void reverse() ;
        void display() ;
};
```



Rational : implémentation

rational.cpp *implémentation*
définition des fonctions

```
#include "rational.h"

void Rational::initialize(int num, int den)
{
    itsNum = num ;
    itsDen = den ;
}

double Rational::convert()
{
    return (double) itsNum / itsDen ;
}
```

```
void Rational::reverse()
{
    // int temp = itsNum ;
    // itsNum = itsDen ;
    // itsDen = temp ;
    swap(itsNum, itsDen);
}
void Rational::display()
{
    cout<< itsNum <<"/"<<
itsDen ;
}
```

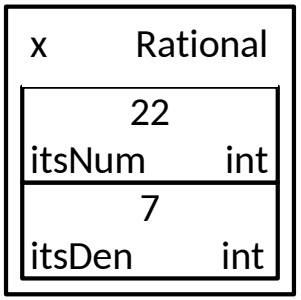
Rational : application

main.cpp — *application*
programme principal

```
#include "rational.h"

int main()
{
    Rational x ;  // instanciation de la classe CRatio
                  // déclaration d'un objet de type CRatio
    x.initialize(22,7) ;
    cout << "x = " ;
    x.display() ;
    cout << " = " << x.convert() << endl;
    x.reverse() ;
    cout <<  "1/x = " << x.display() ;

    return 0 ;
}
```

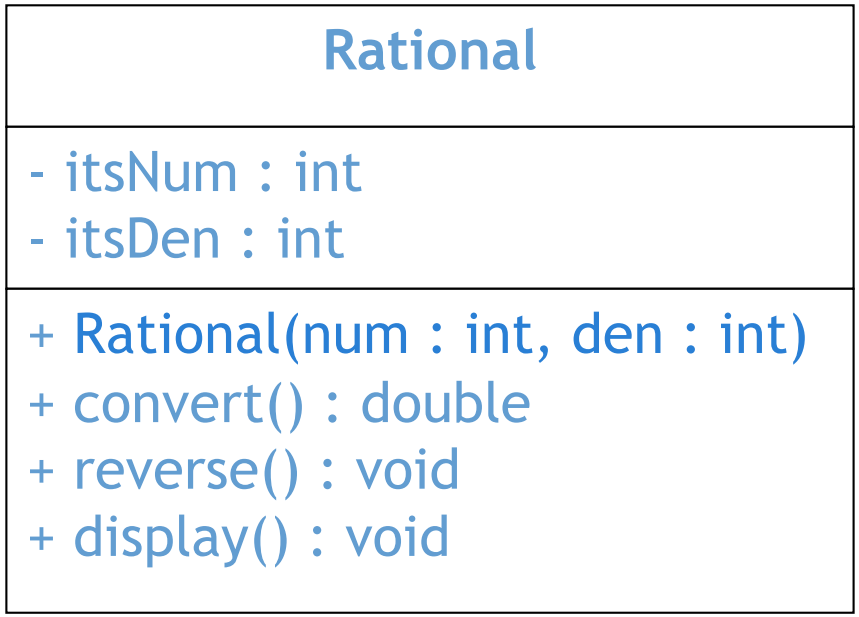



```
x = 22/7 = 3.142

1/x = 7/22
```

Rational : le constructeur

Le constructeur est une méthode particulière qui a pour rôle d'initialiser les attributs de l'objet lors de son instanciation.



- un constructeur porte le même nom que la classe
- un constructeur n'a pas de type de retour (même pas void)
- un constructeur peut avoir des arguments

Rational : le constructeur

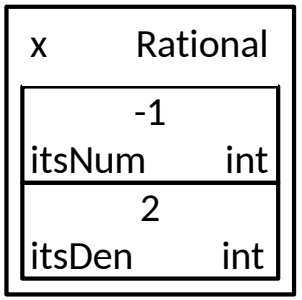
rational.cpp

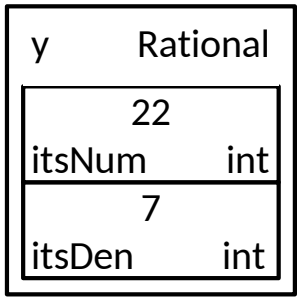
```
Rational::Rational
    (int num, int den)
{
    itsNum = num ;
    itsDen = den ;
}
```

main.cpp

```
int main()
{
    Rational x(-1,2), y(22,7) ;
    cout << "x = ";
    x.display() ;
    cout << " et y = " ;
    y. display() ;

    return 0 ;
}
```





x = -1/2 et y = 22/7

Rational : constructeurs multiples = surcharge

CRatio.cpp

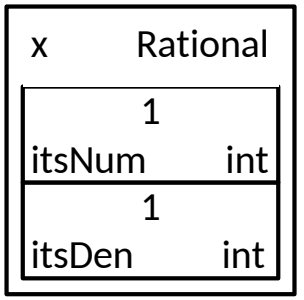
```
// constructeur par défaut (sans paramètre)
Rational::Rational ()
{
    itsNum = 1 ;
    itsDen = 1 ;
}
// constructeur paramétrique (1 param.)
Rational::Rational (int num)
{
    itsNum = num ;
    itsDen = 1 ;
}
// constructeur paramétrique (2 param.)
Rational::Rational (int num, int den)
{
    itsNum = num ;
    itsDen = den ;
}
```

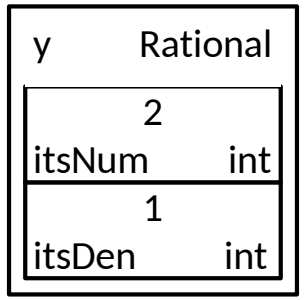
main.cpp

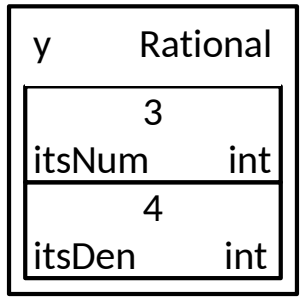
```
int main()
{
 Rational x, y(2), z(3,4) ;
    cout << "x = ";
    x.display() ;
    cout << "\ny = " ;
    y.display() ;
    cout << "\nz = " ;
    z.display() ;

     return 0 ;
}
```

```
x = 1/1
y = 2/1
z = 3/4
```







Rational : constructeurs multiples = initialisation

rational.cpp

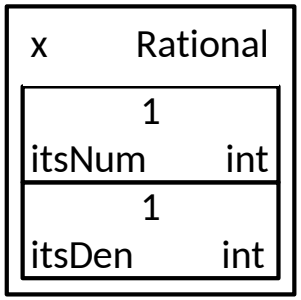
```
// constructeur paramétrique avec
      valeurs par défaut
Rational::Rational(int
     num = 1, int den =
     1)
{
    itsNum = num ;
    itsDen = den ;
}
```

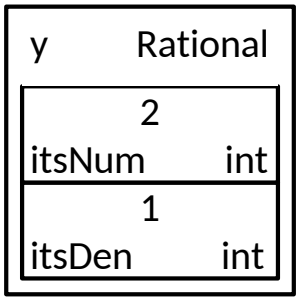
main.cpp

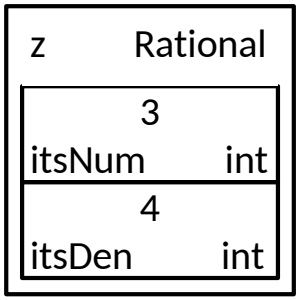
```
int main()
{
    Rational x, y(2), z(3,4) ;
    cout << "x = ";
    x.display() ;
    cout << "\ny = " ;
    y.display() ;
    cout << "\nz = " ;
    z.display() ;

     return 0 ;
}
```

```
x = 1/1
y = 2/1
z = 3/4
```







Rational : le destructeur

Le destructeur est une méthode particulière qui permet de réaliser des opérations (libération de mémoire) lors de la destruction de l'objet.

Rational
- itsNum : int - itsDen : int
+ Rational(num=1, den=1) **+ ~Rational()** + convert() : double + void reverse() : void + display() : void

- un destructeur porte le même nom que la classe précédé de ~
- un destructeur n'a pas de type de retour (même pas void)
- un destructeur ne peut pas avoir d'argument

Rational : le destructeur

Rational.h

```
class Rational
{
  private :
    int itsNum, itsDen ;

  public :
      Rational() { cout << "Bonjour"; }
      ~Rational() { cout << "Au revoir"; }
      …
} ;
```

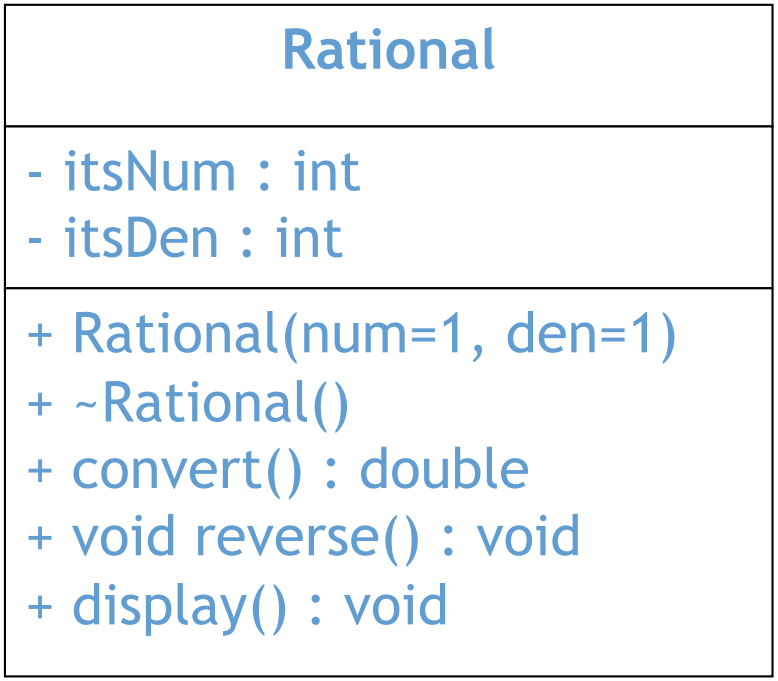
main.cpp

```
int main()
{
    {
        Rational x ;
        cout << "bonjour x !" ;
    }
    Rational y ;

    return 0 ;
}
```

```
Bonjour
bonjour x !
Au revoir
Bonjour
Au revoir
```

La classe Rational



Les getties = accesseurs

Un accesseur est une fonction membre permettant de récupérer le contenu d'une donnée membre en accès privé.

Rational
- itsNum : int - itsDen : int
+ Rational(num=1, den=1) + ~Rational() + convert() : double + void reverse() : void + display() : void **+ getNum() : int** **+ getDen() : int**

Un accesseur, pour accomplir sa fonction :

- doit avoir comme type de retour le type de la variable à renvoyer
- ne doit pas nécessairement posséder d'arguments.

Les getties = accesseurs

rational.cpp

```
…
int Rational::getNum()
{
     return itsNum ;
}
int Rational:: getDen() {
     return itsDen ; }
…
```

main.cpp

```
int main()
{
   {
      Rational x(3,4) ;
      cout << "x = " << x.getNum()
      << "/" << x.getDen() ;
   }

   return 0 ;
}
```

x = 3/4

Les setties = mutateurs

Un mutateur est une fonction membre permettant de modifier le contenu d'une donnée membre en accès privé.

Rational
- itsNum : int - itsDen : int
+ Rational(num=1, den=1) + ~Rational() + convert() : double + void reverse() : void + display() : void + getNum() : int + getDen() : int **+ setNum(num : int) : void** **+ setDen(den : int) : void**

Un mutateur, pour accomplir sa fonction :

- doit avoir comme paramètre la valeur à assigner à la donnée membre qui est donc du type de la donnée membre
- ne doit pas nécessairement renvoyer de valeur ; il possède dans sa plus simple expression le type void

Les setties = mutateurs

Rational.cpp

```
…
void Rational::setNum (int num)
{
     itsNum = num ;
}
void Rational::setDen (int den) {
     if (den==0) den = 1 ;
     itsDen = den ;
}
  …
```

main.cpp

```
int main()
{
   {
      CRatio x(3,4) ;
      x.display() ;
      x.setNum(2) ;
      x.setDen(0) ;
      x.display() ;
   }

   return 0 ;
}
```

```
x = 3/4
x = 2/1
```

R2.01
DEV2

Développement orienté objets

Etienne Carnovali **Pétra Gomez**

Farid Ammar-Boudjelal Jean-Michel Bohé Rouaa Wannous