Programmation orientée objet

Introduction aux classes et objets en C++

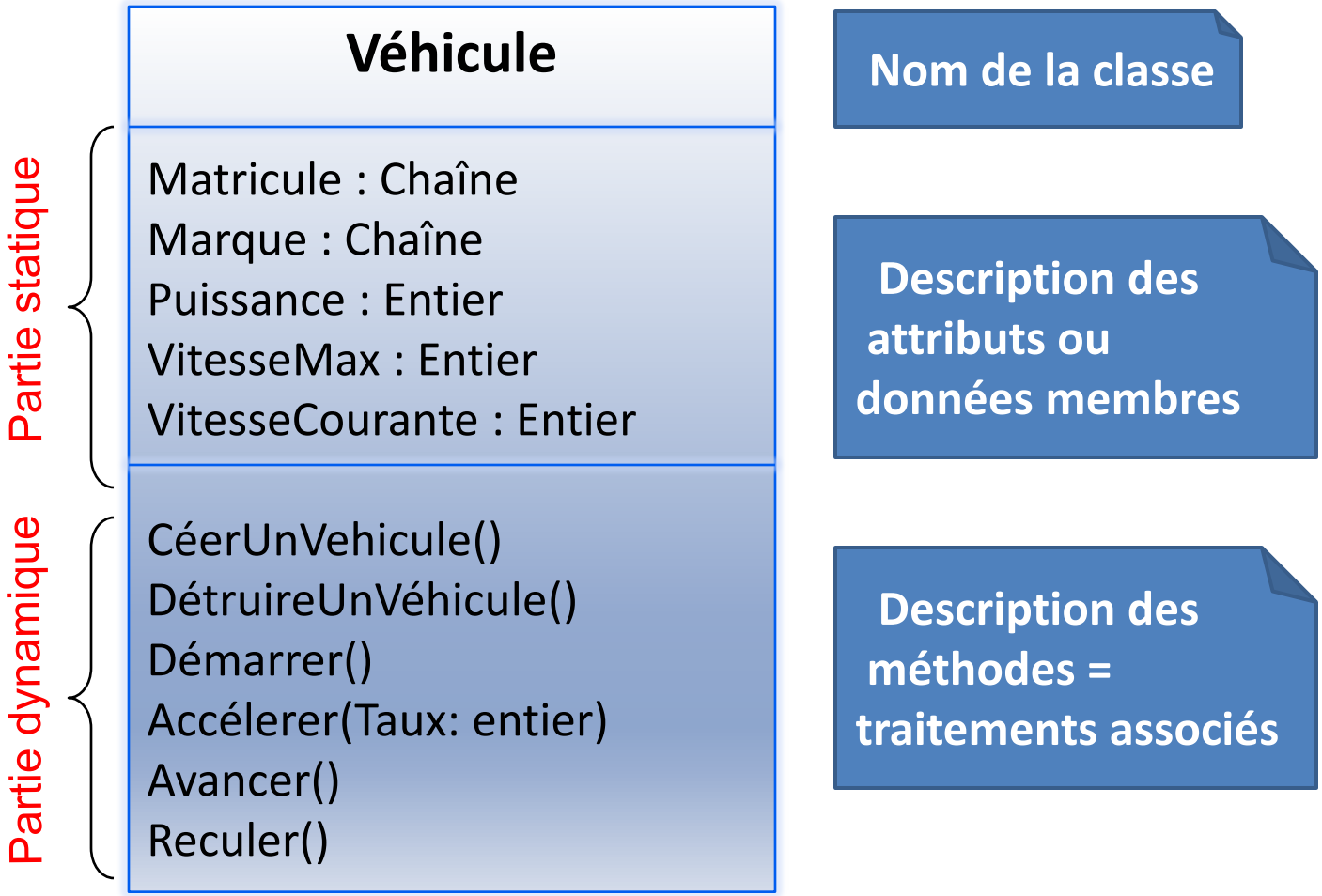
Introduction

- C++, java sont des **langages orientés objet.**
- Qu'est-ce que ça veut dire?
  - Les objets sont de nouvelles **sortes de variables encore plus générales** que les types dérivés (ou structures).
  - Leur caractéristique principale: ils acceptent des **procédures et fonctions** comme composantes.
- A quoi ça sert?
  - Améliorer la **lisibilité d'un code** en utilisant des schémas plus proche du raisonnement humain grâce aux objets.
  - Développer un code au sein d'un grand projet (plusieurs développeurs)

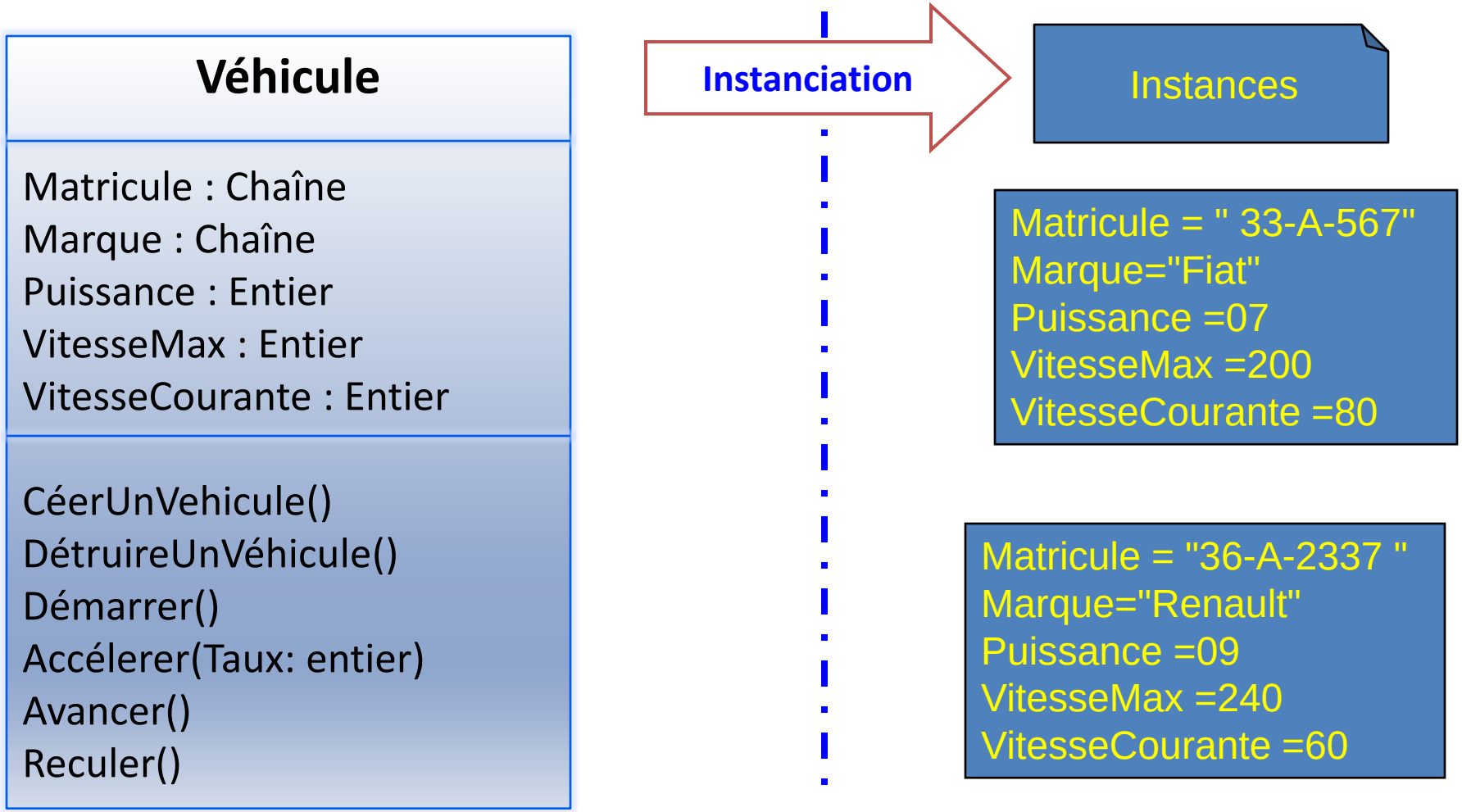
Objet & classe : définition

- Un **objet** est une entité cohérente rassemblant des **données** (partie statique) et des **traitements** (partie dynamique) .
- Une **classe** est une classe d'équivalence d'objets. Une classe est un moule à partir duquel on peut créer des objets.
- Une classe C++ est le prolongement des structures C (mot-clé struct).

Objet & classe : présentation



Objet & classe : présentation



Instanciation

- Pour désigner un objet de la classe, on dit aussi une *instance*
- L'instanciation est le mécanisme de création d'objets à partir d'une classe.
- Ce mécanisme comporte deux phases:
  1. allouer de la mémoire pour l'instance;
  2. initialiser ses attributs d'instance.

Attributs et Méthodes

Attribut ou membre

- L'état d'un objet est l 'ensemble des valeurs de ses attributs. Un attribut est une propriété de l'objet.
- En C++, on dit **membre**.

Méthode ou fonction membre

- Vue de la modélisation objet, une méthode est une opération que l'on peut effectuer sur un objet.
- Vue de la programmation objet, une méthode est une fonction qui s'applique sur une instance de la classe.
- En C++, on dit **fonction membre**.

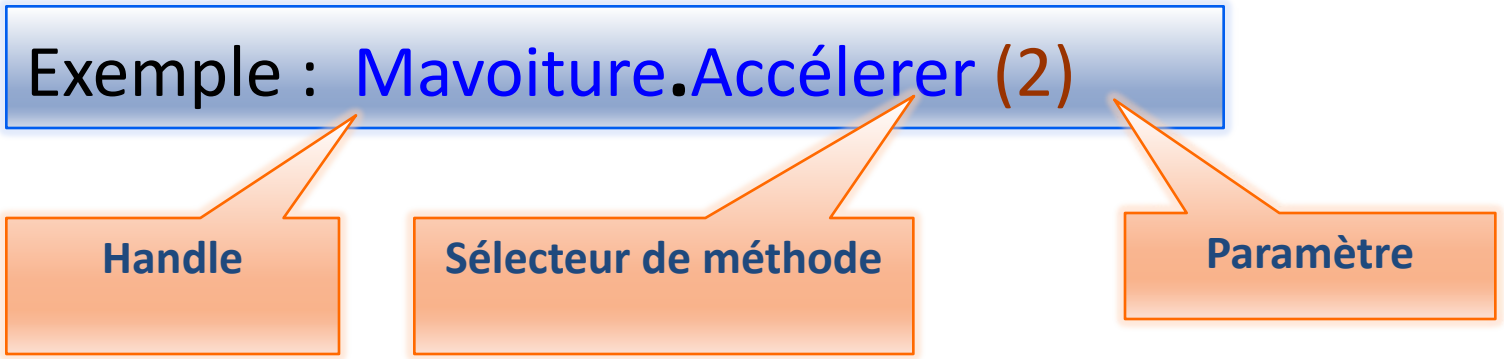
Encapsulation

- L'encapsulation est l'un des grands principes de la POO :
  - Un objet rassemble en lui même ses données et ses traitements.
  - Abstraction de données : la structure d'un objet n'est pas visible de l'extérieur.
  - L'objet protège les données et ne les rendant accessibles qu'au travers des **interfaces.**

L'interface précise quelles sont les services (actions) offerts par l'objet.

Encapsulation : Message

- L'objet est une boite noire. Le message est le moyen de formuler une requête à un objet (une demande de service).
- Le message doit contenir:
  1. Handle de l'objet : une référence ou un pointeur vers l'objet.
  2. Sélecteur de méthode (nom de la méthode).
  3. Liste des paramètres.



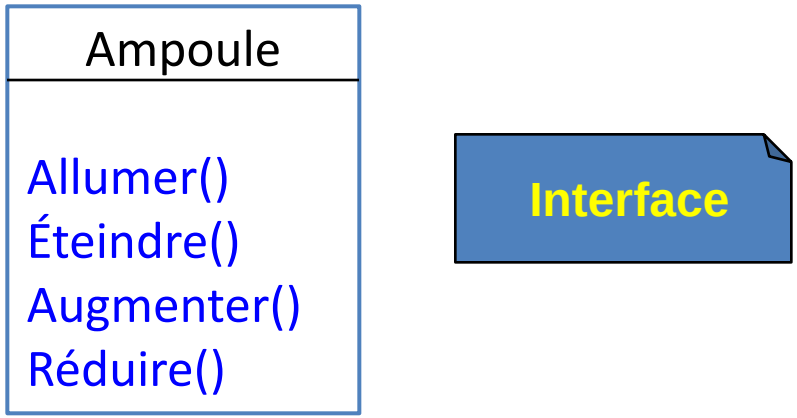
Encapsulation : Interface

- La liste des messages auxquels est capable de répondre un objet constitue son **interface**.

C'est la partie public d'un objet.

- En pratique, dans C++, l'interface représente **la liste des méthodes accessibles (publiques)** au client de l'objet.

C'est au programmeur de veiller au respect du principe d'encapsulation



Les Classes en C++

- **Syntaxe :** de déclaration d'une classe en C++

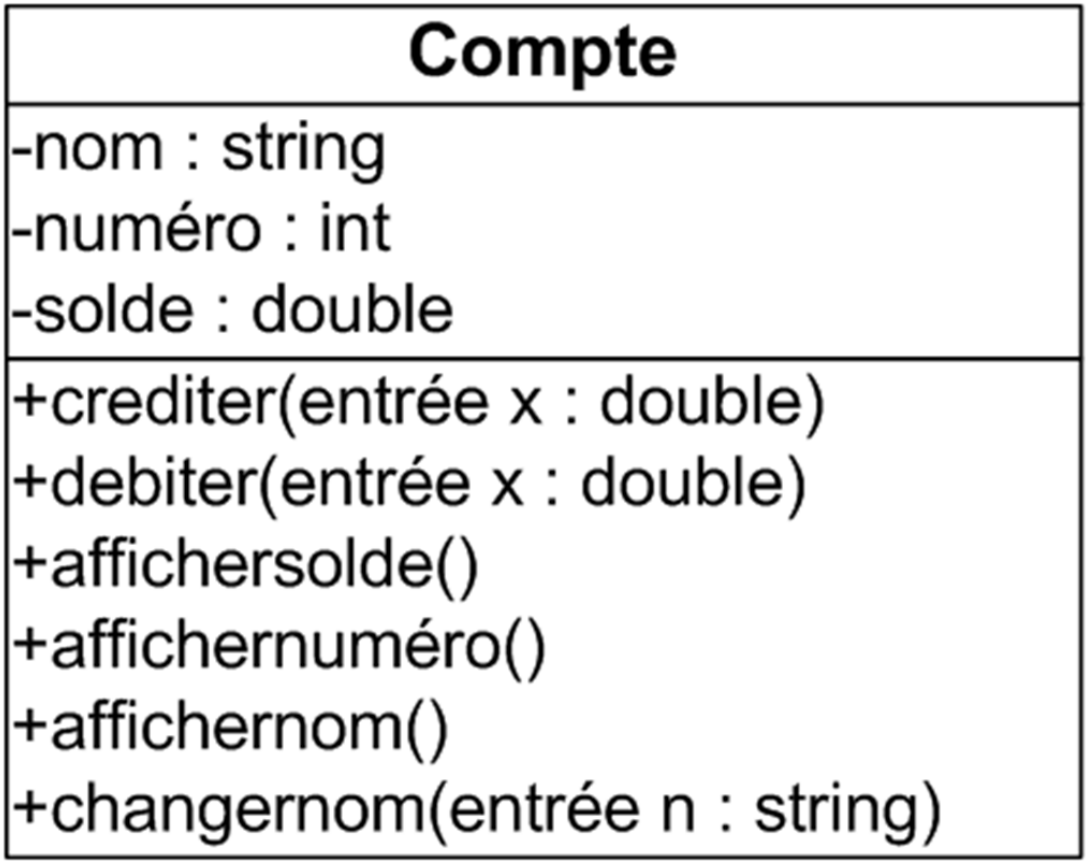
```
class NomDelaClasse
   {
    [protection1] :
      ….
    [protection2] :
    ……
    ……….
    [protection n] :
    ……
  };
```

```
class Cfraction
 {
 private :
          int num;
          int den;
 public :
  Cfraction ();
  Cfraction (int a, int b);
  void Saisir();
  Cfraction AdditionerA (Cfraction F) ;
  Cfraction MultiplierPar (Cfraction F) ;
  void Affiche()
  ~Cfraction() {} ;
  };
```

[protection] ≡ < public | private | protected >

Exemple

- **Exemple** : déclarons une classe « Compte » correspondant à ce diagramme UML



Exemple

```
class Compte
{
  private :
     string nom;
     int numero;
     double solde;
  public :
      void crediter (double x);
      void debiter (double x);
      void affichersolde (void);
      void affichernumero (void);
      void affichernom (void);
      void changernom (string n);
};
```

Les Classes en C++

Protections:

- **public** : les membres publics seront accessibles à l'extérieur de la classe (section d'interface)

- **private** : les membres privés ne sont accessibles que par les méthodes membres de la classe
  - Pour l'exemple précédent (la calass Cfraction), une instruction du style de **x.num** dans le main() serait rejetée à la compilation car **num** est un membre privé de la classe Cfraction.

- **protected** : les membres protégés sont comme les membres privés. Mais ils sont aussi accessibles par les méthodes des classes dérivées (Héritage)

- Par défaut les membres d'une classe sont déclarés privés.

Définition des méthodes membres

- La déclaration d'une classe contient simplement les prototypes des méthodes de la classe.
- Toute méthode définie à l'intérieur de la déclaration de la classe est considérée comme une fonction **inline**
- Les méthodes membres sont définies à l'extérieur de la déclaration de la classe ou dans un fichier séparé.

Syntaxe :

```
Type NomDeLaClasse :: NomFonction(liste arguments)
{
  // Corps de la fonction
}
```

Constructeur & Destructeur

- Les ***constructeurs*** et les ***destructeurs*** sont deux types de méthodes particulières qui sont appelées respectivement à la création et à la destruction d'un objet.
- Il est nécessaire de gérer certaines actions (initialisation) qui doivent avoir lieu lors de la création d'un objet et de sa destruction

Par exemple, si l'objet doit contenir des variables dynamiques, il faut leur réserver de la mémoire à la création de l'objet. À la destruction de l'objet, il convient de libérer la mémoire allouée.

Constructeur

- Les constructeurs se définissent comme une méthode. Ils doivent :
  - porter le même nom que la classe ;
  - n'avoir aucun type (*pas même le type void)*.
- Conformément au principe de surcharge des fonctions, une classe peut contenir plusieurs constructeurs.
- Les paramètres des constructeurs peuvent avoir des valeurs par défaut (comme toute fonction en C++).
- On appelle constructeur par défaut un constructeur n'ayant pas de paramètres ou ayant des valeurs par défaut pour tous ses paramètres:

Destructeur

- **Destructeur :** méthode dont le rôle principal est de supprimer de la mémoire d'un objet.
- Un destructeur est appelé implicitement à la fin de la durée de vie d'un objet, typiquement à la fin du bloc d'instructions dans lequel il a été défini.
- Un destructeur est très rarement appelé explicitement par le programmeur.
- Le destructeur est une méthode qui :
  - porte le même nom que la classe précédé du caractère ~ (tilda)
  - n'a aucun type (*pas même le type void)*.
  - n'accepte aucun argument.
  - une classe ne peut posséder qu'un seul destructeur.

Destructeur

- Si le programmeur ne crée pas lui même un destructeur dans une classe, le compilateur en générera un **par défaut.**
- Ce dernier remplira tout à fait son rôle, à l'unique condition que la classe ne possède pas d'attributs de **type pointeurs,** ou que ses méthodes n'en utilisent pas.
- Sinon, il faudra en créer un afin de libérer l'espace mémoire occupée par ces pointeurs.
- **Quand créer un destructeur ?**
  - Quand la classe contient un attribut qui est un pointeur.
  - Quand on veut que la libération de la mémoire s'accompagne d'une autre tâche (affichage, modification d'un attribut de classe…).

```
~Compte() { nbcomptes--; } ;
```

Exemple

```
class Cfraction
 {
 private :
          int num;
          int den;
 public :
  Cfraction ();
  Cfraction (int, int);
  void Saisir();
  void Affiche();
  ~Cfraction() {} ;
  };
```

```
Cfraction::Cfraction() // constructeur par défaut
  {
      num = 0;
      den = 1;
  }

Cfraction::Cfraction(int newnum, int
newden)
  {
      num=newnum;
      den=newden;
  }
```

Exemple

- **Exemple de déclaration des objets fractions dans la fonction main**

```
Cfraction F;  // utilisation du constructeur par défaut;
Cfraction F2(1,2) ; //Appel du constructeur Cfraction(int
                    // newnum, int newden)
Cfraction *pF ; // pas de constructeur
Cfraction Ftable[5] ;//Appel du constructeur par défaut
                      // pour chaque élément
 pF=new Cfraction ; // utilisation du constructeur par défaut
 delete pF; // destructeur
pF = new Cfraction (2,5); //Appel du constructeur
                          // Cfraction(intnewnum, int newden)
```

Remarques

- Les méthodes d'une classe disposent d'un accès direct aux attributs de cette même classe, c'est pourquoi il n'est pas nécessaire de passer ces derniers en paramètres pour qu'une méthode puisse les manipuler.

```
void Compte::debiter (double x)
{ solde -= x; }
```

Règles d'utilisation des fichiers .cpp et .h

- Pour pouvoir effectivement réutiliser les classes programmées et les compiler séparément, il est indispensable de les exploiter proprement et les ranger dans des fichiers qui portent des noms corrects.
  - La déclaration d'une classe **MaClasse** doit être mise dans un fichier **maclasse.h**
  - La définition d'une classe **MaClasse** doit être mise dans un fichier **maclasse.cpp**

Règles d'utilisation des fichiers .cpp et .h

- Pour l'instant, retenir qu'un fichier maclasse.h possède la structure suivante :

```
// fichier  maclasse.h
#ifndef  MACLASSE_H
#define MACLASSE_H
#include ... // includes de classes eventuelles
...
class MaClasse {
...
} ;
#endif
```

- **#ifndef #define et #endif** dans le **.h** pour que le fichier ne soit effectivement inclus qu'une seule fois lors d'une compilation

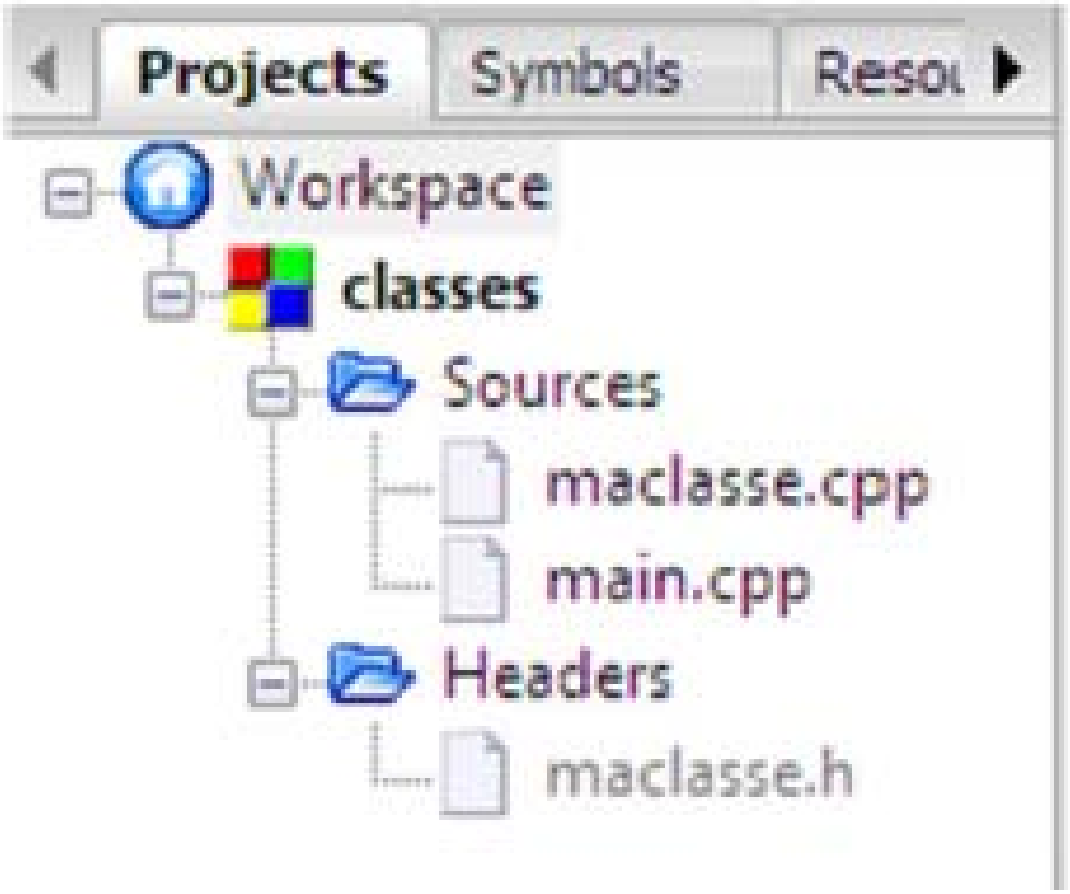
Règles d'utilisation des fichiers .cpp et .h

- Le fichier **maclasse.cpp** possède la structure suivante :

```
// fichier nomclasse.cpp
#include "maclasse.h" // include de sa propre classe
#include ... // autres includes optionnels
...
MaClasse : : MaClasse (...) { // definition du constructeur
...
}
MaClasse : :~ MaClasse () { // definition du destructeur
...
}
.. // autres definitions de fonctions membres
```

Règles d'utilisation des fichiers .cpp et .h

- L'arborescence du projet ressemble alors à cela :

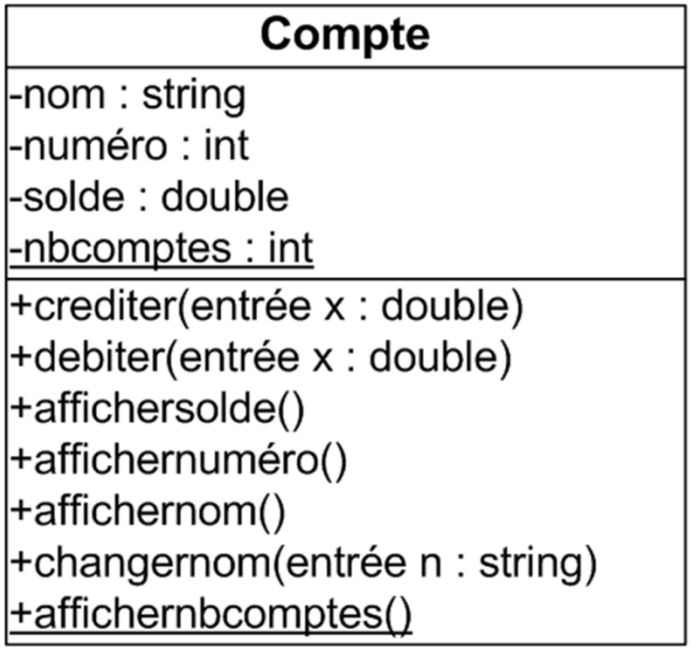


Attributs et méthodes de classe

- **un attribut de classe est** un attribut qui possède la même valeur pour toutes les instances de la classe,
- une **méthode de classe est** une méthode qui porte sur des attributs de classe.
- **Déclaration :** les attributs et méthodes de classes se déclarent avec le préfixe « static ». Ils peuvent être **publics ou privés.**

attribut de classe

méthode de classe



Attributs et méthodes de classe

- Exemple

```
class Compte
{   private :
        std::string nom;
        int numero;
        double solde;
        static int nbcomptes;
    public :
        void crediter (double x);
        void debiter (double x);
        void affichersolde (void);
        void affichernumero (void);
        void affichernom (void);
        void changernom (std::string n);
        static void affichernbcomptes(void);};
```