Cours 1 - C++ pour les mathématiques appliquées

Introduction

De quoi ça parle?

C++ moderne

```
[&, n] (int a) mutable { m = ++n + a; }(127);
[](auto a, auto b) { return a + b; }
```

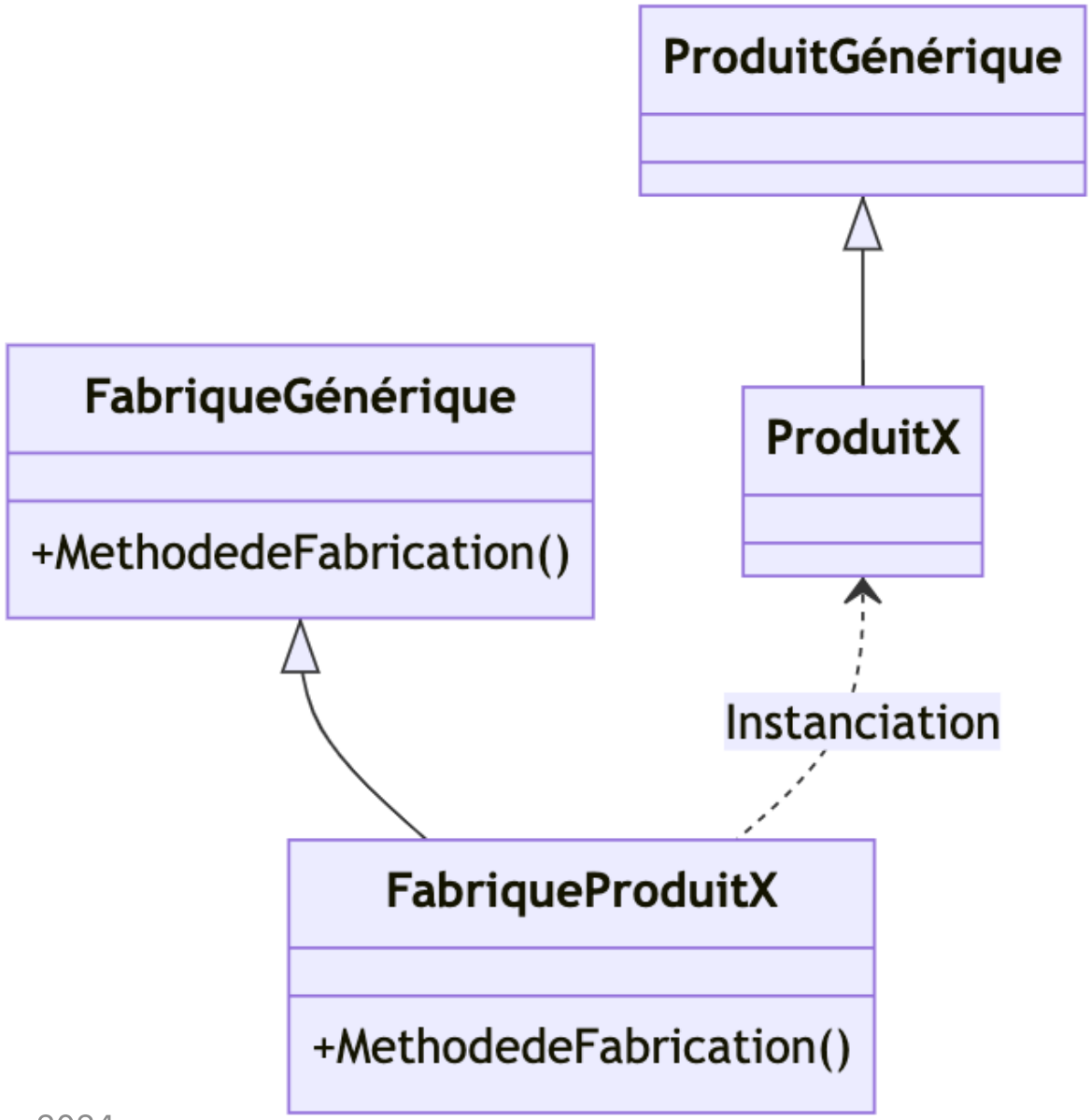
```
export module Particles;
import Logger;
```

C++ efficace

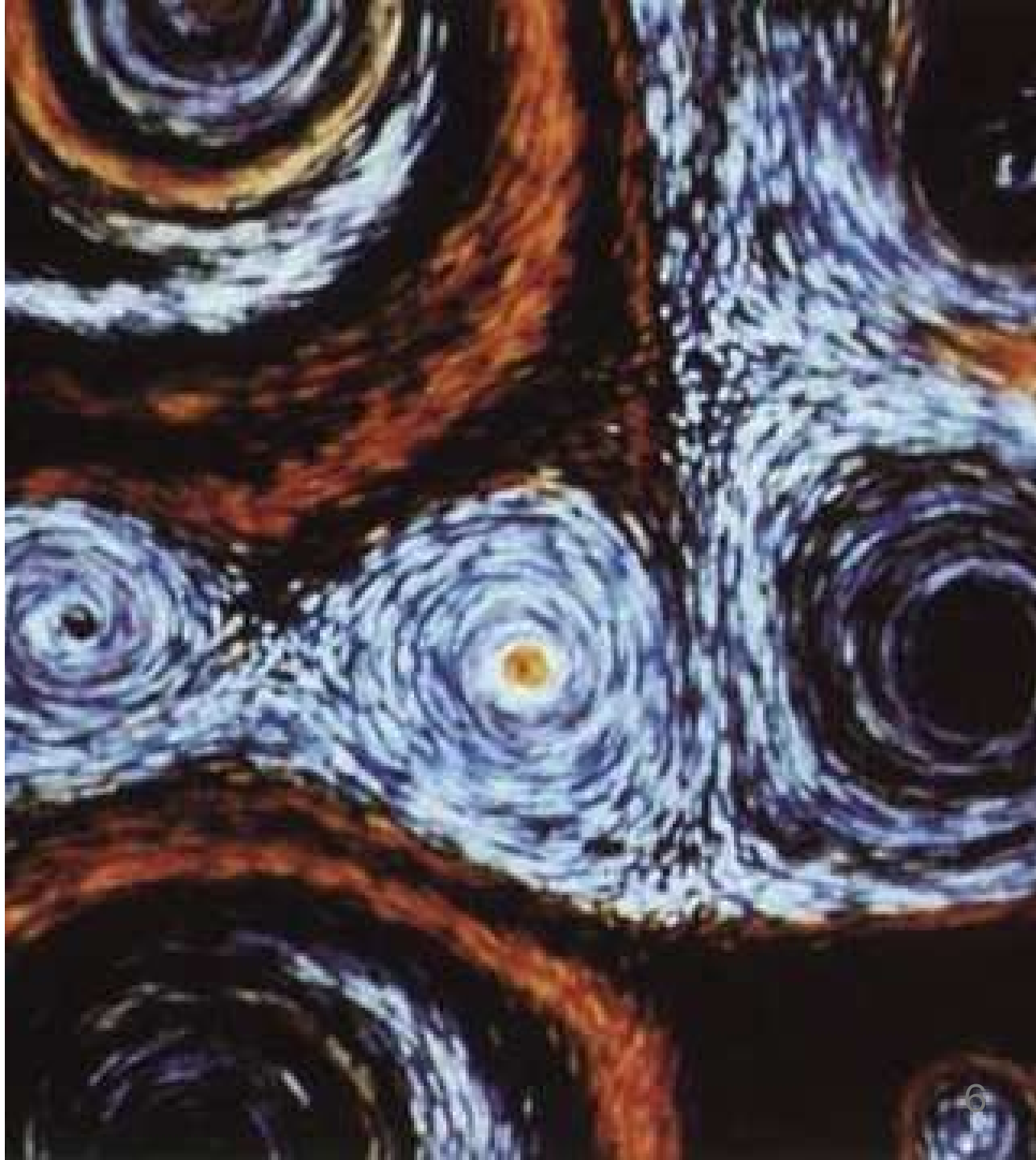
```cpp
template <typename T>
class Total {
    public :
    double getValue() const{
        return static_cast<const T&>(*this).getValue();
    }
};

class Constant42 : public Total<Constant42> {
    public :
    double getValue() const{return 42;}
};
```

Conception . . .


ProduitGénérique
FabriqueGénérique
+MethodedeFabrication()
ProduitX
Instanciation
FabriqueProduitX
+MethodedeFabrication()

... d'applications



Organisation du cours

Équipe pédagogique

Christophe Picard - Amphi E, Groupe 1, Groupe 2

Horaires

Mercredi 11h15 -- 12h45 : Cours en amphi.

Jeudi 8h15 -- 9h45/11h15 -- 12h45 : TP par binômes.

Objectifs

- Se familiariser avec les différents concepts.
- Comprendre les concepts présentés dans le cours.
- Développer une maîtrise des concepts nécessaires aux applications.
- Appréhender divers aspects de la programmation scientifique.
- Développer un code scientifique en langage .
- Mettre en oeuvre des outils de travail performants.

Consignes Généralités

- Les TP sont en binômes.
- Les rendus doivent se présenter sous la forme d'une archive au format `.tar.gz` qui créera à l'extraction un répertoire ***TPi_nom1_nom2***.
- Les rapports doivent être au format uniquement `.pdf` et ne contiennent pas de code.
- Les codes doivent être réfléchis.

Consignes Présentation du code

Le code doit :

- être facile à lire.
- s'articuler logiquement.
- être déboggable.
- être transmissible.

Vous devez :

- commenter le code abondamment, précisément, clairement et immédiatement.
- indenter proprement et uniformément.
- donner des noms significatifs aux "objets" que vous manipulez.

Quelques hypothèses

- Vous savez programmer dans un langage quelconque.
- Vous savez utiliser un terminal.
- Vous avez accès un à compilateur C++ moderne.
- Vous connaissez vos algorithmes et structures de données de bases.
- Vous savez résoudre une EDO.

Calcul scientifique

- Le calcul scientifique, les sciences des données, l'IA consistent à déplacer une grande quantité d'information entre des mémoires et des unités de traitement, les traiter et les déplacer vers la mémoire à nouveau

C'est la raison de l'existence de **FORTRAN**

Mais cela reste un langage avec des capacités génériques réduites.

C++ est un langage qui permet de construire des abstractions et de les combiner.

Motivations

- Simulation d'un processus physique
  Processus physiques $\Longrightarrow$ modèle mathématique $\Longrightarrow$ algorithme $\Longrightarrow$ logiciel $\Longrightarrow$ simulation numérique

Application d'algorithmes numériques

Utilisation massive du `C++`

- Simulation de phénomènes naturels (Code de météo France)
- Applications industrielles (Dassault System)
- Application en médecine.
- En finance.
- En cybersécurité (Linbox).
- Deep Learning.

Logiciel scientifique

Quelques propriétés

- Validation
- Efficacité : vitesse, mémoire, stockage, énergétique
- Maintenabilité
- Extensibilité.

Compétence

- Comprendre le modèle mathématiques.
- Comprendre l'approche numérique.
- Concevoir les algorithmes et les structures de données.
- Choisir et utiliser les bibliothèques et outils adaptés.
- Adaptabilité aux nouveaux langages et concepts.

Code de calcul type

- point de départ : problème computationnel
- Pre-processing :
    - Données d'entrée et préparation.
    - Mise en place des structure de données
- Coeur du calcul
- Post-processing :
    - Analyse des résultats
    - Affichage, sortie et visualisation

Logique de développement

L'implementation correcte de problèmes numériques évolués est une tâche difficile.

Elle se divise en deux étapes :

- Exprimé le problème numérique sans forme d'un algorithme.
- Traduire l'algorithme en langage machine par l'intermédiaire d'un langage de programmation

Quel langage de programmation?

- Il existe plusieurs centaines de langages de programmation.
- Pour le calcul scientifique, les choix sont beaucoup plus restreints : Fortran (77, 95, 2003, 2008), C, C++, Matlab (GNU Octave), Python, Julia, Maple/Mathematica.
- Comment le choisir
    - Efficacité calculatoire.
    - Coût de développement, maintenabilité.
    - Eco-système efficace et évolué.
    - Support pour des types utilisateurs.
- Les projets sont protéiformes. Des langages différents sont requis.

  L'interopérabilité est nécessaire.

Propriétés

Les langages compilés

- Sont en général plus rapides.
- possèdent des cycles de développement plus long.

Langages interprétés

- Sont en général plus lents.
- Cycles de développement plus rapides. Les fonctionnalités numériques sont dans des langages compilés.

Des langages différents ont des efficacités différentes.

Types utilisateurs

Les primitives de types ne sont pas toujours suffisantes pour les codes numériques!

Les primitives sont groupées dans un nouveau type de données

- `struct` en C (pas de fonctions)
- `class` en Java, Python, et C++

Les hiérarchies de classes sont indispensables en programmation scientifique.

La programmation orientée objet peut conduire à des pertes de performances importantes.

Le C++ c'est quoi?

- Tout usage.
- Flexible en permettant aux développeurs/utilisateurs de construire des abstractions.
- Performance et efficacité.
- Se faire comprendre par l'utilisateur.

On en est où?

- 1979 : Création des classes pour C par Bjarne Stroustrup
- 1983 : Changement de nom.
- 1985 : livre de référence.
- 1998 : Standardisation ISO/IEC 14882:1998 $\Longrightarrow$ `C++98`
- 2003 : révision mineure
- 2011 : nouveau standard $\Longrightarrow$ `C++11`
- 2014 : revision mineure $\Longrightarrow$ `C++14`
- **2017 : révision intermédiaire** $\Longrightarrow$ `C++17`
- 2020 : révision majeure $\Longrightarrow$ (C++ 20) $\Longrightarrow$ `gcc-12`
- 2023 : révision mineur `

On apprends quoi dans le cours?

Des concepts du `C++`

- Outils de gestion de code : compilation, liens and gestion des dépendances, integration continue, tests.
- Gestion de ressources: RAII, smart pointers, references, copy/move.
- Programmation procédurale : fonctions, paramètres , lambdas, surcharges, et gestion d'erreur..
- OOP: classes, héritage, polymorphisme.
- Programmation générique : templates et variations ...
- Programmation à la compilation : `template` récursifs, `constexpr` , et traits.
- Conteneurs et itérateurs : STL, `iterator` , vues.

Et on les applique en vrai!

Premier exemple de code

On commence par quoi?

```cpp
#include <iostream>
void main() {
    std::cout << "Hello, world!" << std::endl;
}
```

```
$ g++ --std=c++11 hello.cpp -o hello
$ ./hello
Hello, world!
```

Ce que fait le compilateur

1. Le préprocesseur s'exécute en premier.
2. La directive `#include` recopie l'ensemble du fichier dans l'unité de compilation courante.
3. Les symboles `<...>` indiquent de rechercher dans les répertoires systèmes.
4. Le fichier `iostream` inclus est de la bibliothèque standard.

Ce que fait l'OS

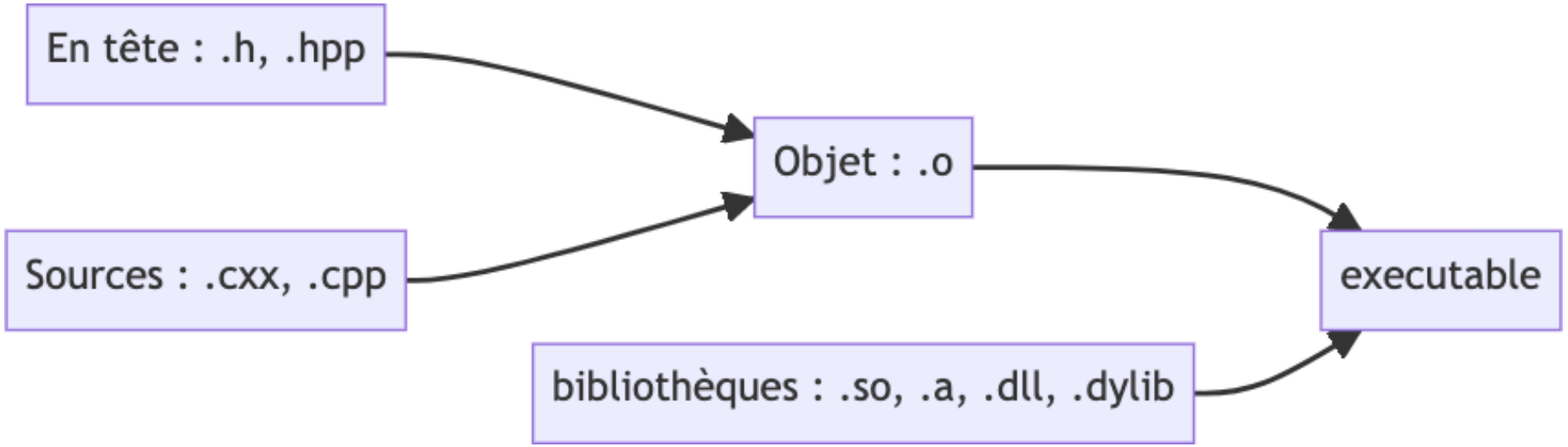
- Chaque programme ne peut contenir qu'une et une seule fonction .
- Le compilateur et le système d'exploitation identifie cette fonction spéciale afin qu'elle soit exécutée lors du lancement.

Ce que ça signifie

- `std` est l'**espace de nom** standard.
- Un espace de nom est un regroupement d'objets avec une portée limitée.
- Pour accéder à un élément d'un espace de noms, on utilise l'**opérateur** `::` .
- `cout(cin)` représente la sortie standard (l'entrée standard).
- La bibliothèque standard utilise l'opérateur `<<` pour l'insertion dans un flux.

Quelques Outils

Compilation



Compilation et édition de lien

```
g++ -std=c++-17 -Wall -Wextra -03 -c -o source.o source.cxx
g++ -std=c++-17 -Wall -Wextra -03 -o exe source.o source.a
```

Automatisation : cmake

Il s'agit maintenant de générer des qui soient multi plateformes. Nous allons donc utiliser l'outils .

- Fichier équivalent

```
cmake_minimum_required (VERSION 3.16.3)
project(Demo CXX)
set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_FLAGS "-Weverything -Wall")
# Commentaire
add_executable(exe source.cxx)
```

- Invocation

```
mkdir build; cd build; cmake ..; make
```

Deboggage

Afin de trouver les erreurs dans votre code, l'utilisation de est fortement recommandée.

Quelques commandes pour

- Lancer gdb : `gdb ./exe`
- Lancer le programme sous `gdb` avec argument et redirection de l'affichage:

```
(gdb) run arg1 arg2 > sortie
```

- Afficher les données : `(gdb) print i j`
- Fonction : `(gdb) call fonction(arg1,arg2,...)`
- Variable : `(gdb) set variable name = exp`
- Naviguer dans la liste des appels `up` ou `down` .

Le programme doit être compilé avec l'option **-g**.

Gestion de la mémoire

- L'un des aspects critiques de la programmation dans des langages compilés évolués est la gestion de la mémoire.
- `valgrind` est un outil qui permet de vérifier que la mémoire est gérée de façon correcte, et que toute la mémoire utilisée a bien été libérée.
- Quelques options pour `valgrind`
  - Analyser les fuites mémoires : `valgrind --leak-check=yes myprog arg1 arg2`
  - Montrer les zones encore accessibl `--show-reachable`
  - Être bavard `-v`
  - Enregistrer dans un fichier `--log-file=filename`

La documentation

- Il existe plusieurs types de documentations : besoin, conception, technique, utilisateur.
- La documentation technique peut-être générée en partie automatiquement
- Quelques commandes pour `doxygen`
  - `@param` En instrumentant votre code, vous pouvez, par exemple, spécifier les arguments retour de vos fonctions.
  - `@brief` Résumé de description.
  - `@fn` Documentation de fonction.
  - `@todo` Un bon moyen de communiquer sur l'état d'un morceaux de code.
  - `@bug` Un bug a été identifié. Donner les informations nécessaire.

Types

String

- La bibliothèque standard possède une classe qui permet de manipuler une chaîne de caractères.
- Elle s'utilise avec l'entête approprié `cpp #include <string>`

```cpp
#include <iostream>
#include <string>
int main() {
    std::string message = "Hello, world";
    std::cout << message << std::endl;
}
```

Fonctions

Fonction 101

Une fonction encapsule un morceau de code entre des accolades. La fonction peut être réutilisée ultérieurement.

```
void say_hello() {
    std::cout << "Hello, world!" << std::endl;
}
int main(int argc, char* argv[]) {
    say_hello();
}
```

Signature de fonction

Une fonction déclare ses types de retour et de paramètre.

Les paramètres sont des variables locales initialises par la fonction d'appel

Une valeur est renvoyée avec la mot clé . Le type de l'expression retournée doit être compatible avec le type de retour de la fonction.

```
int sum(int a, int b) {
    return a + b; }
```

Le type de retour peut être induit:

```
auto sum(int a, int b) -> int {
    return a + b; }
```

Surcharge

Des fonctions avec le même nom, mais des arguments différents peuvent coexister.

```
int sum(int a, int b) {
    return a + b;
}
double sum(double a, double b) {
    return a + b;
}
```

Lorsque la fonction est invoquée, le compilateur identifie les arguments et recherche le meilleur candidat.

Le meilleur candidat peut-être une fonction standard ou une fonction utilisateur.

Conclusion

En résumé

- Quelques éléments de syntaxes.
- Premier programme en .
- Compilation et exécution.
- Outils de programmation.

La prochaine fois

Les classes...