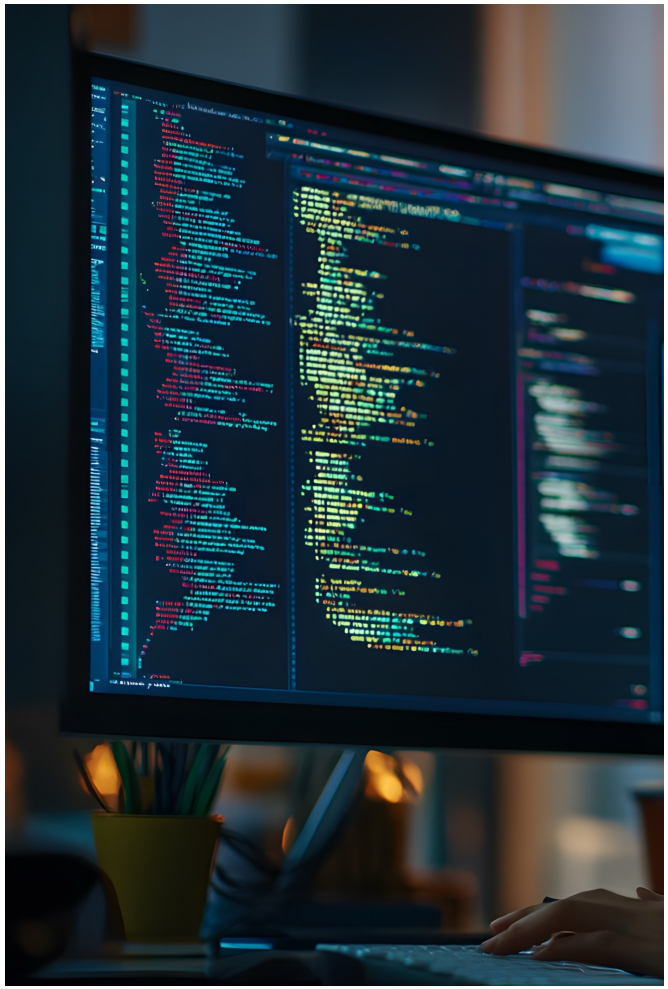
Programmation Orientée Projet
COM-112(a)

Semaine 2

Rafael Pires
rafael.pires@epfl.ch

Lausanne, 27.02.2026

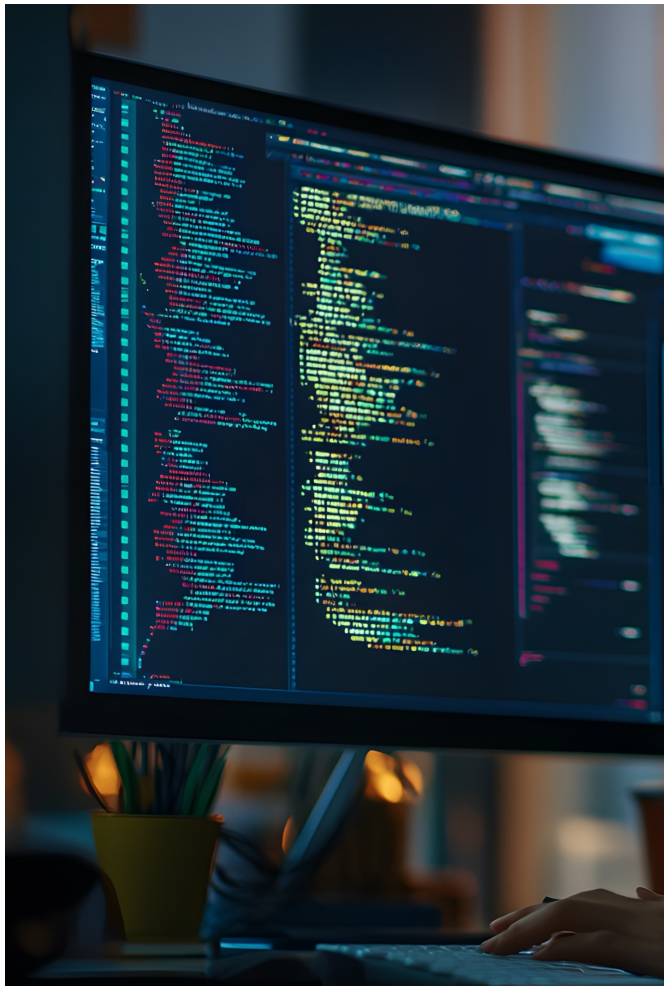
Classe, encapsulation et abstraction



- Question
  - Comment `class` répond aux objectifs de la programmation modulaire ?
- Plan :
  - Interface et implémentation à l'échelle d'une classe
  - Portée et les espaces de noms (`namespace`)
  - Projet

Précédemment sur le MOOC – Semaine 1

Classes, objets, attributs et méthodes

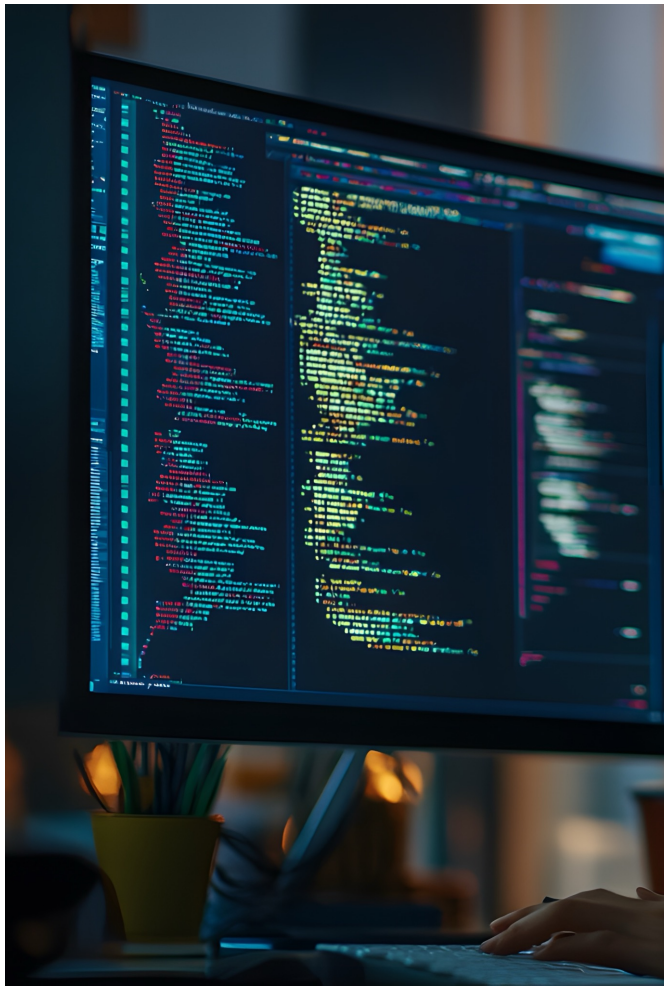


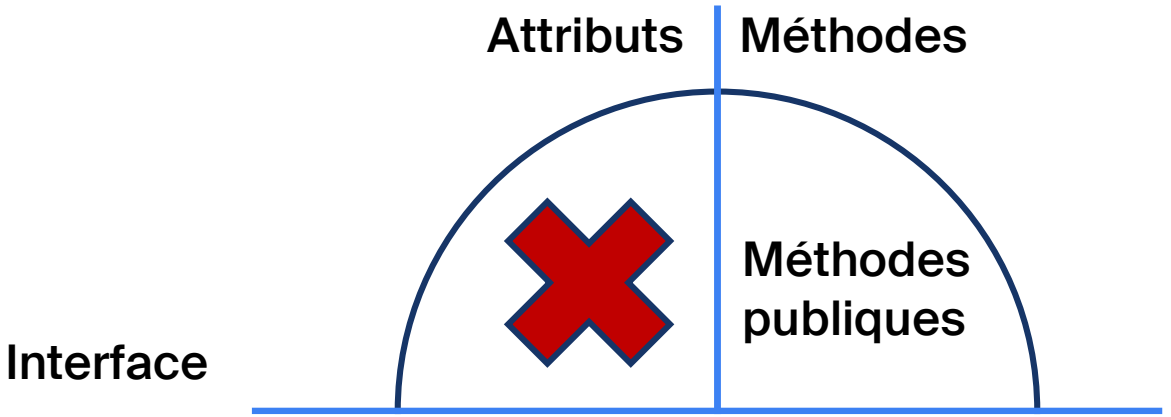
```
class Rectangle {
public:
  double surface() const;
  double getHauteur() const;
  double getLargeur() const;
  void setHauteur(double h);
  void setLargeur(double l);


private:
  void helperMethod();

  double hauteur;
  double largeur;
};
```

```
int main() {
  Rectangle rect1, rect2;
  rect1.setHauteur(5.0);
  rect1.setLargeur(3.0);
  // …
  return 0;
}
```

Précédemment sur le MOOC – Semaine 1

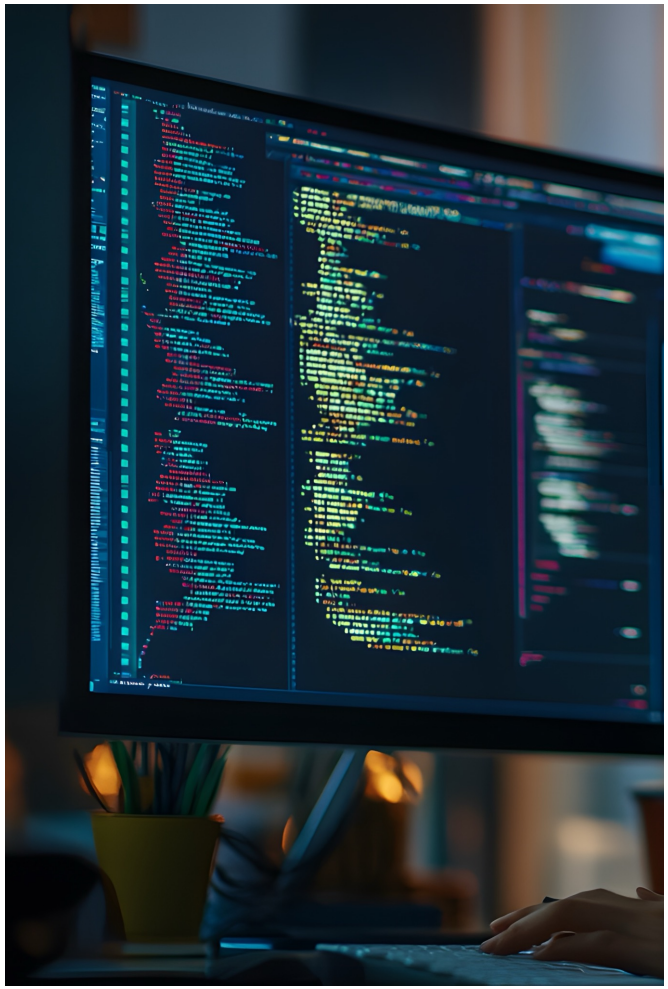
Encapsulation et abstraction





abstraction

Précédemment sur le MOOC – Semaine 1
Encapsulation et abstraction

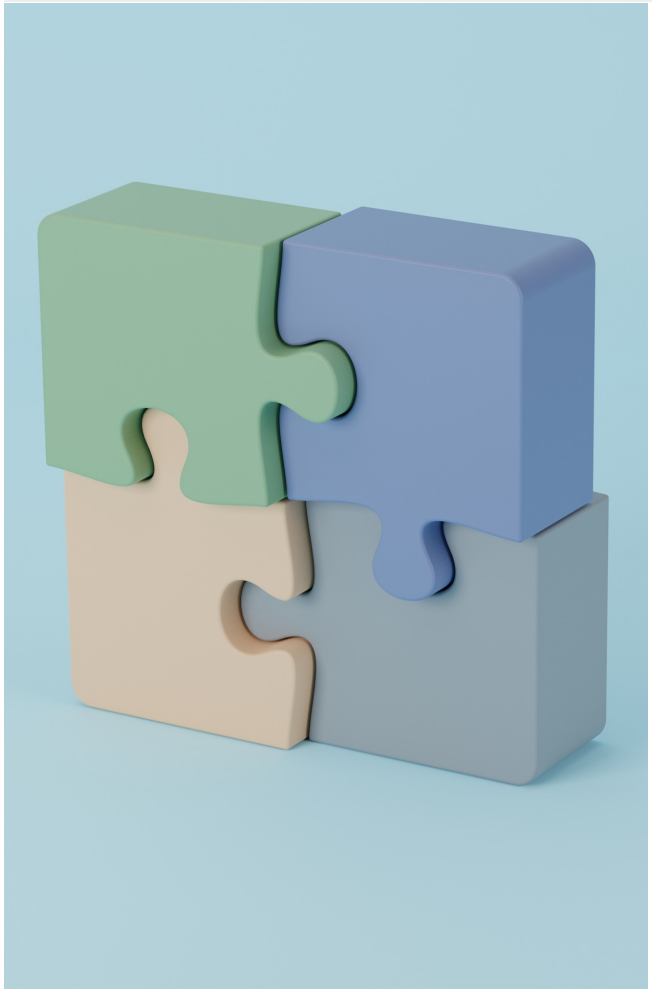


- **Abstraction**
  - Une **classe** gère un **type**
  - Ce type est défini en termes **d'attributs** et de **méthodes**

*Terminologie: un synonyme de méthode est **fonction membre** d'une classe, par opposition à une **fonction libre** indépendente d'une classe.*

- **Encapsulation**
  - La classe permet l'accès exclusivement à certains éléments pour garantir son bon fonctionnement
  - **public** : accès possible en dehors de la classe
  - **private** : accès restreint à la classe

C++ : Structure (`struct`) et classe (`class`)



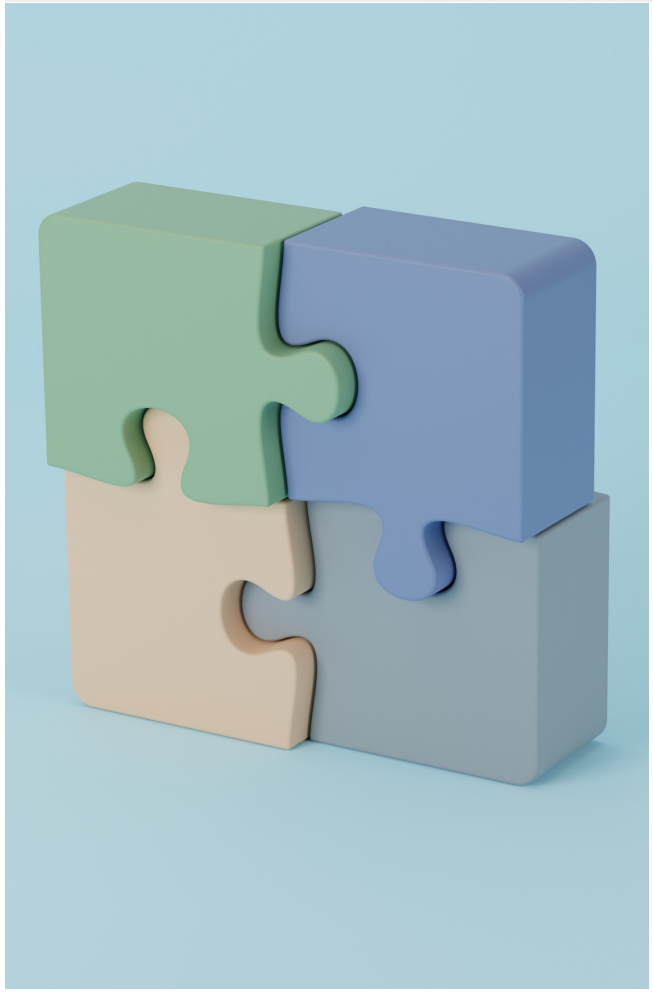
```
struct Date
{
  int day;
  int month;
  int year;
};
```

```
int main()
{
  Date birth;
  birth.day = 30;
  birth.month = 2;
  birth.year = 2026;
  …
```

```
class Date
{
  int day;
  int month;
  int year;
};
```

```
int main()
{
  Date birth;
  birth.day = 1;
  birth.month = 3;
  birth.year = 2026;
  …
```

C++ : Éviter les attributs publics



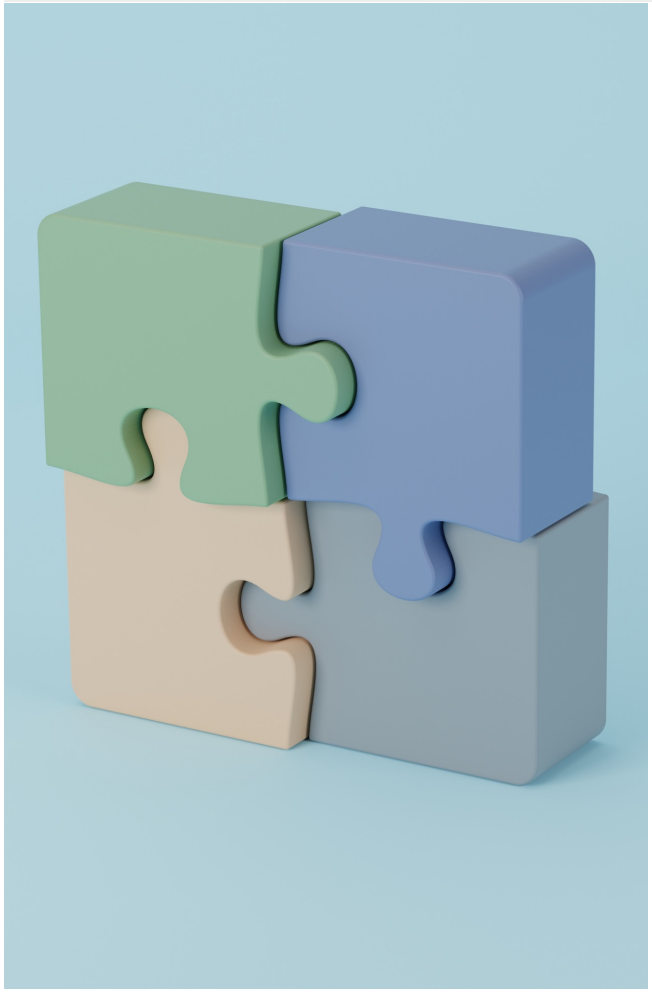
```
class Date
{
public:
  int day;
  int month;
  int year;
};
```

```
class Date
{
private:
  int day;
  int month;
  int year;
};
```

C++ laisse la liberté de mettre ce qu'on veut en accès **`public`**, aussi bien des méthodes (OK) que des attributs (très mauvaise pratique).

La responsabilité de réaliser une encapsulation correcte revient à la *personne qui définit la classe*.

C++ : Éviter les attributs publics



date.h

```
class Date{
public:
  void setDate(int d, int m, int y){
    if (m < 1 || m > 12) {
      // Erreur "Mois invalide"
    }
    int maxDays = daysInMonth(m, y);
    if (d < 1 || d > maxDays) {
      // Erreur "Jour invalide pour le mois et l'année donnés"
    }
    day = d;
    month = m;
    year = y;
  }

private:
  int daysInMonth(int m, int y);
  int day;
  int month;
  int year;
};
```

Précédemment sur POP-01 : C'est quoi un module ?

*un **module** = une **interface** + une **implémentation***

- Interface
- Implémentation

module calcul

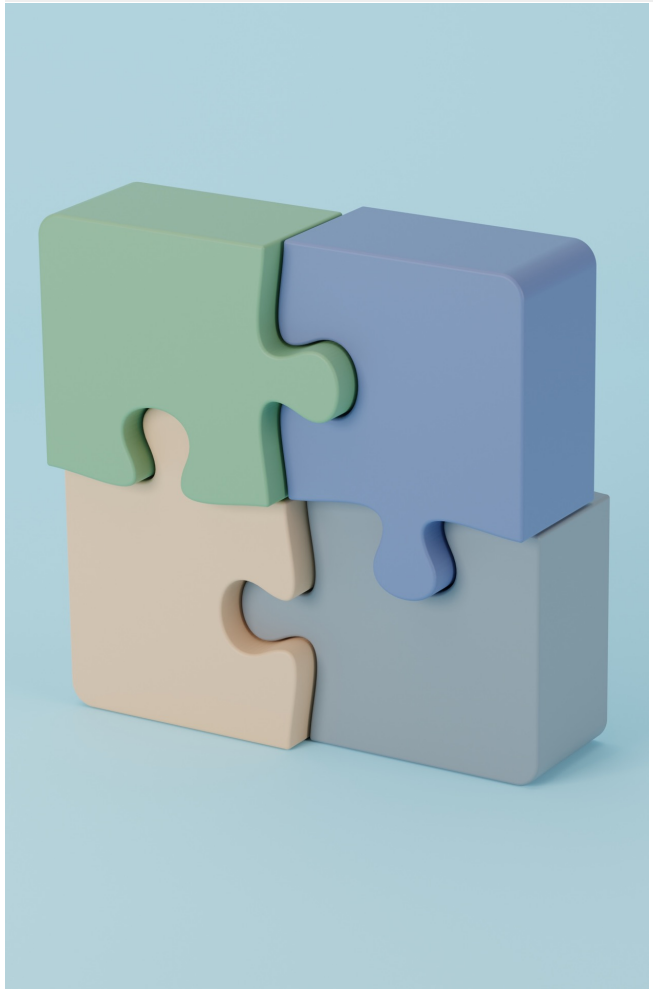
calcul.h

```
int div(int num, int denom);
```

calcul.cc

```
#include "calcul.h"
int div(int num, int denom)
{
  if(denom != 0)
    return num/denom;
  return 0;
}
```

C++ : Externaliser la définition des méthodes



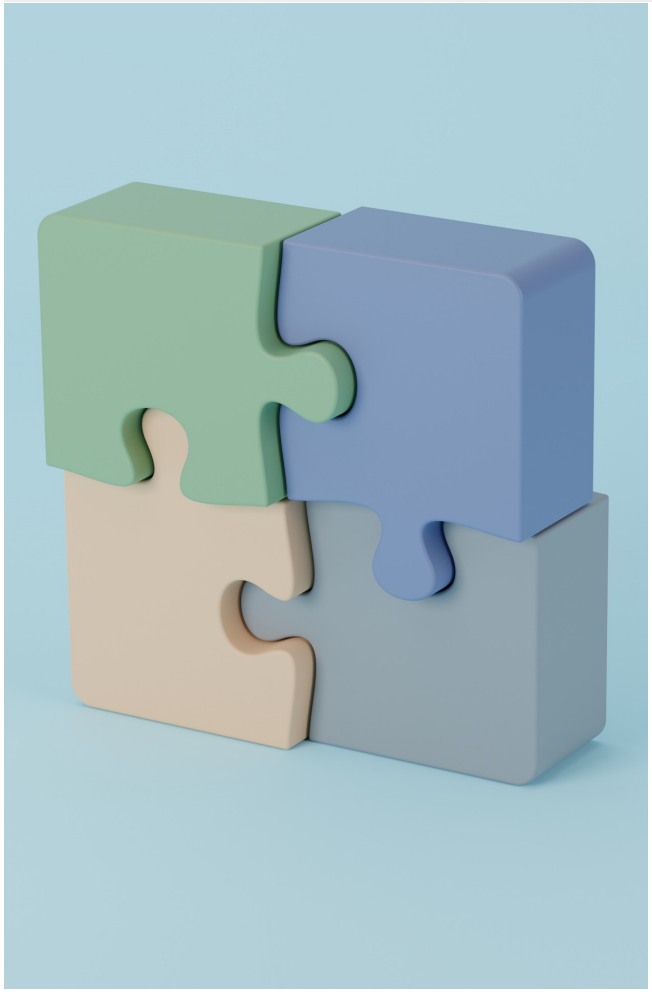
rectangle.h

```
class Rectangle {
public:
  double surface() const { return hauteur * largeur; }
 double getHauteur() const { return hauteur; }
 double getLargeur() const { return largeur; }
 void setHauteur(double h) { hauteur = h; }
 void setLargeur(double l) { largeur = l; }

private:
  double hauteur;
 double largeur;
};
```

rectangle.h

```
class Rectangle {
public:
  double surface() const;
  double getHauteur() const;
  double getLargeur() const;
  void setHauteur(double h);
  void setLargeur(double l);

private:
  double hauteur;
  double largeur;
};
```

C++ : Externaliser la définition des méthodes



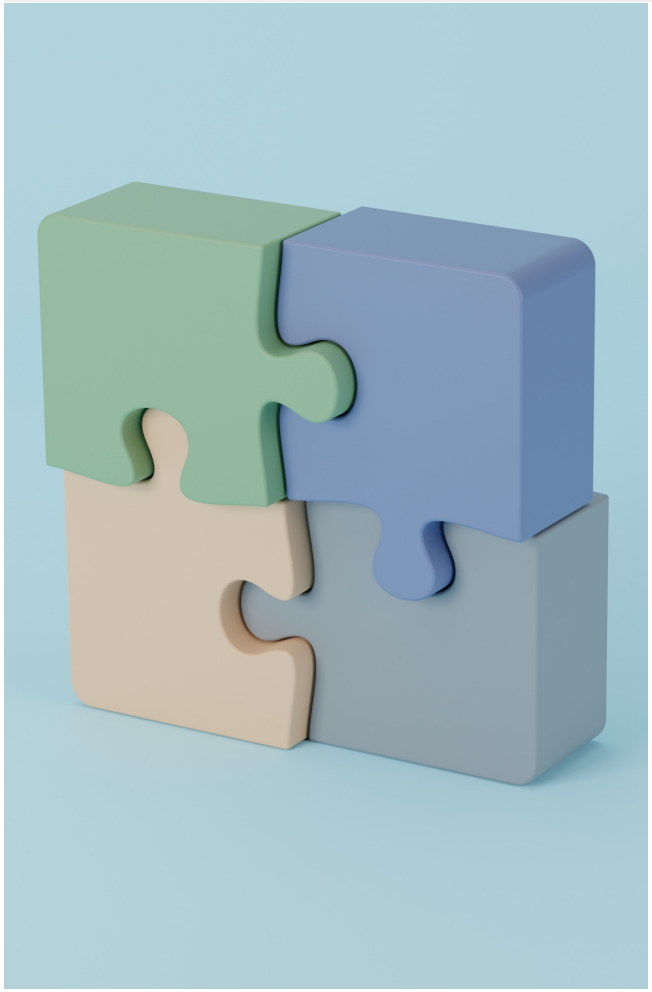
rectangle.h

```
class Rectangle {
public:
  double surface() const;
  double getHauteur() const;
  double getLargeur() const;
  void setHauteur(double h);
  void setLargeur(double l);

private:
  double hauteur;
  double largeur;
};
```

rectangle.cc

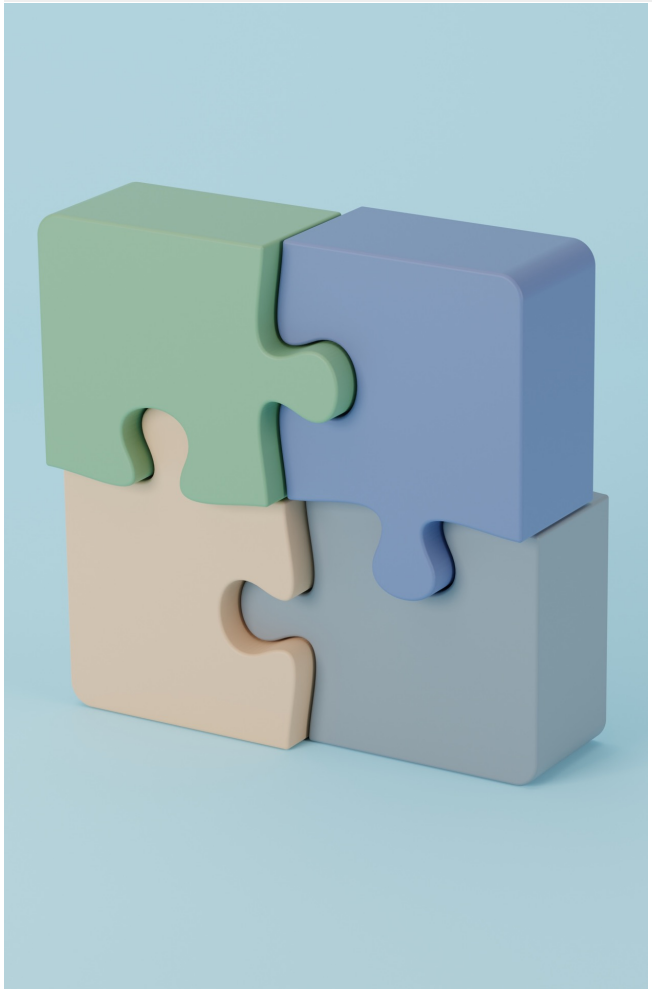
```
#include "rectangle.h"

double Rectangle::surface() const {
  return hauteur * largeur;
}

double Rectangle::getHauteur() const {
  return hauteur;
}

double Rectangle::getLargeur() const {
  return largeur;
}

void Rectangle::setHauteur(double h) {
  hauteur = h;
}

void Rectangle::setLargeur(double l) {
  largeur = l;
}
```

C++ : Externaliser la définition des méthodes



- Pourquoi il faut externaliser la définition des méthodes ?
  - Les bonnes pratiques de la programmation modulaire cherchent à minimiser les dépendances entre les modules d'un programme
  - Pour cela, l'interface d'un module doit être la plus légère possible
  - Un module associé à une classe doit montrer l'interface de la classe dans l'interface du module (.h) car elle contient les méthodes publiques
  - Monter les prototypes des méthodes publiques est nécessaire et suffisant pour les appeler dans un autre module

C++ : Portée de classe



- L'opérateur de résolution de portée `::` indique que les méthodes appartiennent à l'**espace de noms** de la classe

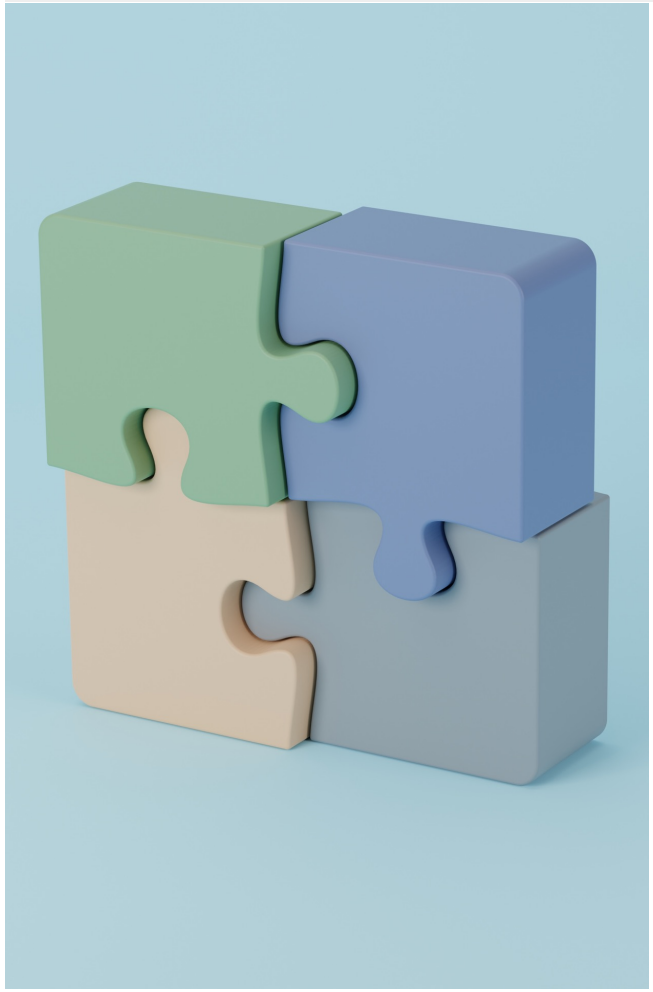
rectangle.h

```
class Rectangle {
public:
  double surface() const;
  double getHauteur() const;
  double getLargeur() const;
  void setHauteur(double h);
  void setLargeur(double l);

private:
  double hauteur;
  double largeur;
};
```

rectangle.cc

```
#include "rectangle.h"

double Rectangle::surface() const {
  return hauteur * largeur;
}

double Rectangle::getHauteur() const {
  return hauteur;
}

// …
```

C++ : Namespaces

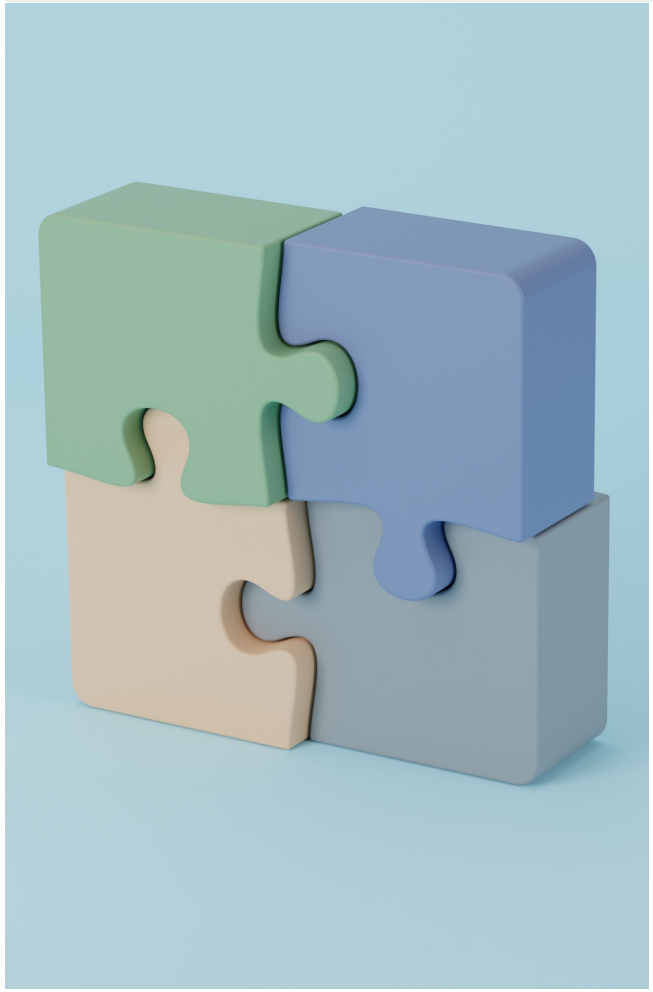


- L'intégration des nombreux modules objets dans l'exécutable par l'éditeur de liens peut faire apparaître des collisions de noms alors que l'étape de compilation n'en a pas détectées.

```
struct Date{
  int day;
  int month;
  int year;
};

// renvoie la date du premier
// jour du calendrier Grégorien
Date reset(){
  return {25, 2, 1582};
}
```

```
struct Coordonnees{
  int x;
  int y;
};


Coordonnees reset(){
  return {0, 0};
}
```

C++ : Namespaces

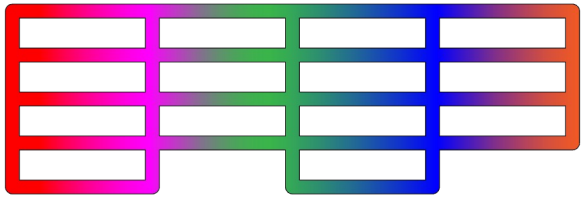


- Remède : introduire un espace noms distinct avec le mot clef **namespace**

```
namespace date{

struct Date{
  int day;
  int month;
  int year;
};

// renvoye la date du premier
// jour du calendrier Grégorien
Date reset(){
  return {25, 2, 1582};
}

}
```

```
namespace coordonnes{

struct Coordonnees{
  int x;
  int y;
};


Coordonnees reset(){
  return {0, 0};
}

}
```

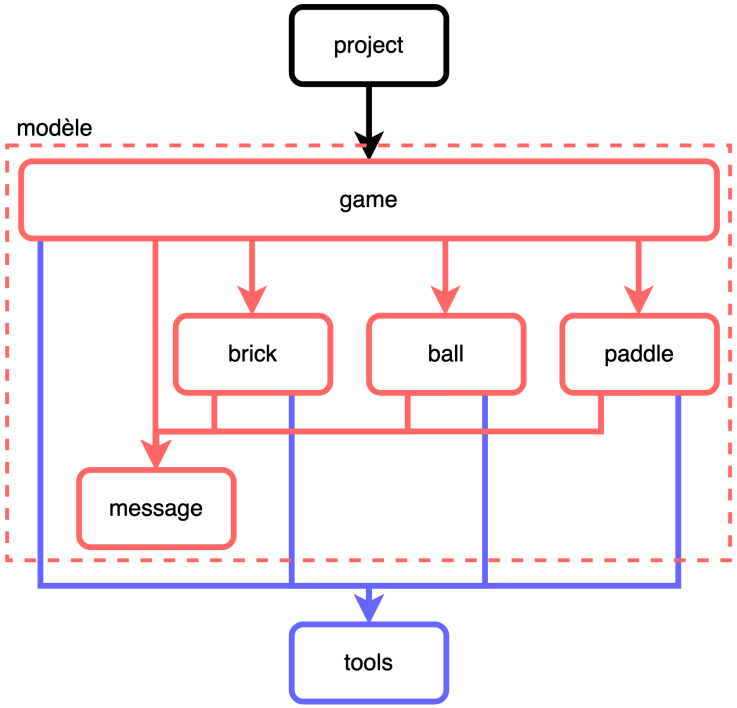
Projet – Brick Breaker



- *3 rendus valant 55% de la note globale :*
  - Rendu1 : 29 mars 15%
    - mise au point d'un module de bas niveau **tools** et initialisation par lecture d'un fichier avec detection d'erreurs.
    - Syntaxe de test: passage d'un nom de fichier

      `./projet t01.txt`
  - Rendu2 : 26 avril 15%
    - Affichage graphique GTKmm et fonctionnement minimal de l'interface (code GTKmm partiellement fourni)
    - Fonctionnement limité
  - Rendu3 : 24 mai 25%
    - Jeu complet avec interface graphique GTKmm

Brick Breaker : Rendu 1



- aucune interface graphique
  (pas de GTKmm)
- lire un fichier, instantier les élements,
  afficher des erreurs
- Livrables :
  - ❖ Code source et Makefile
  - ❖ Rapport (1/2 page)

Résumé Cours 2

- Les principes de la programmation modulaire sont efficacement mis en oeuvre en C++ avec le concept de **class**.
- Une class est constituée d'une **interface** (la partie `public`) et d'une **implementation**
- Il ne faut pas rendre les attributs `public`
- Une bonne correspondance entre module et class implique **d'externaliser** la définition des méthodes dans **l'implémentation** du module qui lui est associé.

rafael.pires@epfl.ch

Merci