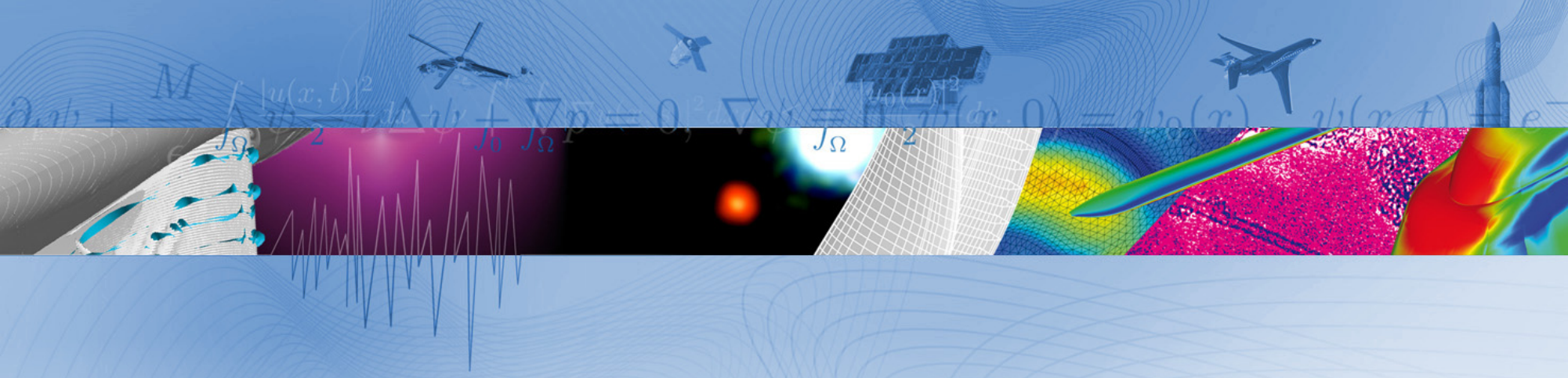


# Introduction à la programmation C++

## Premiers objets

BOULCH Alexandre

ONERA
THE FRENCH AEROSPACE LAB

# Plan de la séance





ONERA
THE FRENCH AEROSPACE LAB

# Plan de la séance





ONERA
THE FRENCH AEROSPACE LAB

# Les objets

## Jusqu'à présent :

- Factoriser du code : **fonctions**, **fichiers**
- Regrouper les éléments cohérents : **tableaux**, **strutures**

Les fonctions agissent sur les données.

## Les objets

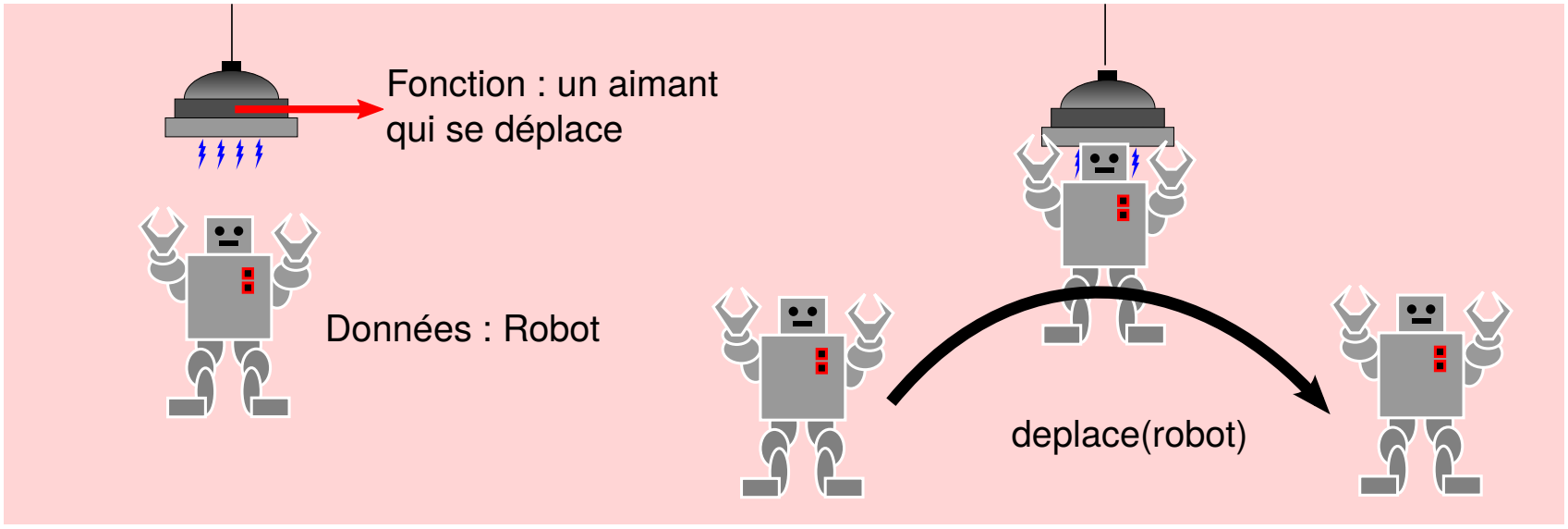
**OBJET = STRUCTURE + MÉTHODES (fonctions)**

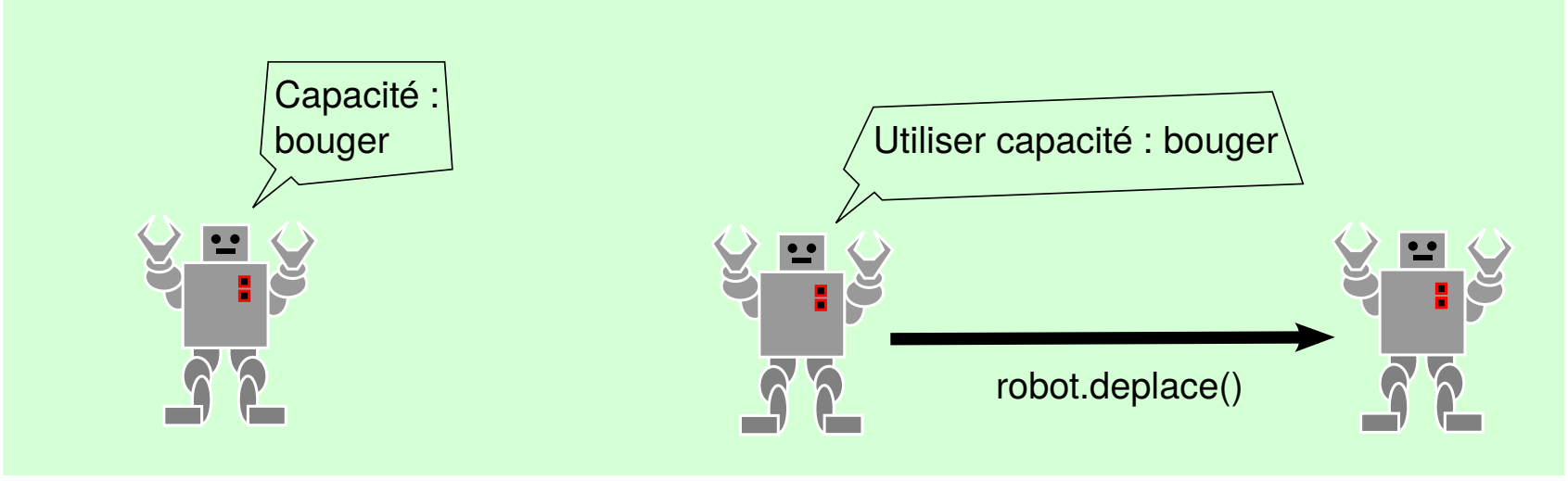
**Idée :** les objets ont des fonctionnalités.



ONERA
THE FRENCH AEROSPACE LAB

# Les objets






ONERA
THE FRENCH AEROSPACE LAB

# Les objets

## Attention

Il ne faut pas voir des objets partout :

- Les données et les fonctions ne sont pas toujours liées.
- Il faut bien penser à l'organisation des données.
- Les fonctions sont souvent plus adaptées lorsqu'elles concernent plusieurs objets.

```
Obj1 a;
Obj2 b;
int i = f(a,b) // fonction f sur a et b

Obj1 a;
Obj2 b;
int i = a.f(b); // methode f de a appliquee a b
                // ou b.f(a) ???
```



ONERA
THE FRENCH AEROSPACE LAB

# Exemple

```
// Structure + fonctions
struct Obj1{
    int x;
};
int f(Obj1 &x);
int g(Obj1 &x, int y);

...
Obj1 a;
cout << f(a) << endl;
int i = g(a,10);
```

```
// Objet
struct Obj1{
    int x;
    int f();
    int g(int y);
};

...
Obj1 a;
cout << a.f() << endl;
int i = a.g(10);
```

On met simplement les déclarations **dans** la structure. **On ne met plus en argument** l'objet en question.



ONERA
THE FRENCH AEROSPACE LAB

# Exemple 2

Dans la définition de la struture précédente, on **déclare** les méthodes, on ne les **définit** pas. Pour définir les méthodes, on utilise les **::**

```
// OBJET
struct Obj1{
    int x;
    int f();
    int g(int y);
};

// SOURCE Obj1.cpp
int Obj1::f(){
    ...
}
int Obj1::g(int y){
    ...
}
```

```
int main(){

    Obj1 a;
    Obj1 b = {5}; //intialisation

    a.x = 2:

    cout << a.f() << endl;
    cout << b.g(a.f()) << endl;

    ...
}
```



ONERA
THE FRENCH AEROSPACE LAB

# Organisation

## Header

Ils reçoivent les **déclarations des structures**.

Ex : `struct Obj1{...};` dans `obj1.h`

## Source

On y place les **déclarations des méthodes**.

Ex : `int Obj1::f(){...}` dans `obj1.cpp`



ONERA
THE FRENCH AEROSPACE LAB

# Plan de la séance





ONERA
THE FRENCH AEROSPACE LAB

# Namespace

Les espaces de nom (**namespace**) définissent un container pour les fonctions ou des objets.
Nous en avons recontré deux : **std** et **Imagine**.
Pour se placer à l'intérieur du namespace on utilise **using namespace xxx;**. De l'extérieur on utilise **::**

```
//de l'interieur du namespace
#include <iostream>
using namespace std;

#include <Imagine/Graphics.h>
using namespace Imagine;

...
cout << i << endl;
click();
...
```

```
// de l'exterieur du namespace
#include <iostream>


#include <Imagine/Graphics.h>


...
std::cout << i << std::endl;
Imagine::click();
...
```



ONERA
THE FRENCH AEROSPACE LAB

# Et pour les objets ?

C'est le même principe : lorsqu'on est dans l'objet, on a accès **aux champs** et **aux méthodes**.

```
struct Obj1{

    int x;

    void double_x();

    int quadruple_et_renvoie_x();

};
```

```
void Obj1::double_x(){
    // on est dans Obj1
    x = 2*x; // on peut modifier les champs
}
int Obj1::quadruple_et_renvoie_x(){
    // on est dans Obj1
    double_x(); // on peut utiliser les
    double_x(); // autres methodes
    return x;
}
```

```
int main(){ // par contre en dehors de l'objet :
    Obj1 a; // il faut creer un objet
    a.x = 3; // et on accede aux champs et aux methodes avec .
    cout << a.quadruple_et_renvoie_x() << endl;
}
```



ONERA
THE FRENCH AEROSPACE LAB

# Plan de la séance





ONERA
THE FRENCH AEROSPACE LAB

# Définition de l'objet

```
// matrice.h
struct Matrice{
    //champs
    int m,n;
    double* t;

    //methodes
    void cree(int m1,int n1);
    void detruit();
    void get(int i,int j);
    void set(int i,int j,double x);
    void affiche();
};
```



ONERA
THE FRENCH AEROSPACE LAB

# Méthodes

```
// matrice.h
struct Matrice{
  //champs
  int m,n;
  double* t;

  //methodes
  void cree(int m1,int n1);
  void detruit();
  double get(int i,int j);
  void set(int i,int j,double x);
  void affiche();
};
```

```
// matrice.cpp
void Matrice::cree(int m1,int n1){
    m = m1;
    n = n1;
    t = new double[m1*n1];
}
void Matrice::detruit(){
    delete[] t;
}
double Matrice::get(int i,int j){
    return t[i+j*m];
}
void Matrice::set(int i,int j,double x){
    t[i+j*m] = x;
}
void Matrice::affiche(){
    for(int i=0; i<m; i++){
        for(int j=0; j<n; j++){
            cout << get(i,j) << " ";
        cout << endl;
    }
}
```



ONERA
THE FRENCH AEROSPACE LAB

# Méthodes : opérateurs mathématiques

```
// matrice.h
struct Matrice{
  //champs
  int m,n;
  double* t;

  //methodes
  void cree(int m1,int n1);
  void detruit();
  double get(int i,int j);
  void set(int i,int j,double x);
  void affiche();
};

Matrice operator*(Matrice A,Matrice B);
```

```
Matrice operator*(Matrice A,Matrice B){
    assert(A.n == B.m);
    Matrice C;
    C.cree(A.m, B.n);
    for(int i=0; i<A.m; i++){
        for(int j=0; j<B.n; j++){
          double d=0:
          for(int k=0; k<A.n; k++){
            d+= A.get(i,k)*B.get(k,j);
          }
          C.set(i,j,d);
        }
    }
    return C;
}
```



ONERA
THE FRENCH AEROSPACE LAB

# Utilisation

```
// matrice.h
struct Matrice{
  //champs
  int m,n;
  double* t;

  //methodes
  void cree(int m1,int n1);
  void detruit();
  double get(int i,int j);
  void set(int i,int j,double x);
  void affiche();
};

Matrice operator*(Matrice A,Matrice B);
```

```
int main(){
    Matrice M1;
    M1.cree(2,3);
    for(int i=0; i<2; i++)
        for(int j=0; j<3; j++)
            M1.set(i,j,i+j);
    M1.affiche();
    Matrice M2;
    M2.cree(3,5);
    for(int i=0; i<3; i++)
        for(int j=0; j<5; j++)
            M1.set(i,j,i*j);
    M2.affiche();
    Matrice M3 = M1 * M2;
    M3.affiche();
    M1.detruit();
    M2.detruit();
    M3.detruit();
}
```



ONERA
THE FRENCH AEROSPACE LAB

# Retour sur les opérateurs

Il est possible de mettre les opérateurs dans les objets. Par convention l'opérateur méthode d'un objet `A` de type Obj1 :

```
operatorOp(Obj2 B)
```

définit l'opération `A Op B` (dans cet ordre)

**Attention**

Pour définir `B Op A`, il faut définir l'opérateur dans l'objet de type Obj2.



ONERA
THE FRENCH AEROSPACE LAB

# Retour sur les opérateurs

```
// matrice.h
struct Matrice{
  ...

  Matrice operator+(Matrice B);
  Matrice operator*(double l);
};

// pour definir l * A obligatoirement
// a l'exterieur
Matrice operator*(double l,Matrice A);
```

```
Matrice Matrice::operator+(Matrice B){
  Matrice C;
  C.cree(m,n);
  for(int i=0; i<m; i++)
    for(int j=0; j<n; j++)
      C.set(i,j,get(i,j)+B.get(i,j));
  return C;
}
Matrice Matrice::operator*(double l){
  Matrice C;
  C.cree(m,n);
  for(int i=0; i<m; i++)
    for(int j=0; j<n; j++)
      C.set(i,j,l*get(i,j));
  return C;
}
Matrice operator*(double l,Matrice A){
    return A*l;
}
```



ONERA
THE FRENCH AEROSPACE LAB

# Interface

```
int main(){
    Matrice M1;
    M1.cree(2,3);
    for(int i=0; i<2; i++)
        for(int j=0; j<3; j++)
            M1.set(i,j,i+j);
    M1.affiche();
    Matrice M2;
    M2.cree(3,5);
    for(int i=0; i<3; i++)
        for(int j=0; j<5; j++)
            M1.set(i,j,i*j);
    M2.affiche();
    Matrice M3 = M1 * M2;
    M3.affiche();
    M1.detruit();
    M2.detruit();
    M3.detruit();
}
```

Si on regarde attentivement, l'utilisateur n'utilisé que :

```
struct Matrice{
    void cree(int m1,int n1);
    void detruit();
    double get(int i,int j);
    void set(int i,int j,double x);
    void affiche();
};
```

C'est **l'interface** de l'objet Matrice.



ONERA
THE FRENCH AEROSPACE LAB

# Interface

L'utilisateur n'a besoin de connaître que l'interface pour utiliser l'objet Matrice.

## Intérêt

Les interfaces permettent de séparer l'**utilisation** de l'objet de sa **conception**.

Une fois l'interface créée, le concepteur peut modifier l'organisation interne de l'objet (changer les champs sans modification apparente pour l'utilisateur).



ONERA
THE FRENCH AEROSPACE LAB

# Plan de la séance





ONERA
THE FRENCH AEROSPACE LAB

# Constat

Rien n'empêche l'utilisateur d'écrire :

```
Matrice A;
A.cree(5,7);
A.t[10] = 1000;
A.m = 50; // il va y avoir des problemes
```

Si le concepteur change `t` en `tab`, le programme de l'utilisateur ne fonctionne plus.
$\longrightarrow$ Il faut empêcher l'utilisateur d'accéder à l'organisation interne de l'objet, il faut **protéger** l'objet.


ONERA
THE FRENCH AEROSPACE LAB

# Principe

On va rendre **privé** une partie de l'objet. Cette partie ne sera accessible que de l'intérieur de l'objet.
Pour cela :

- on remplace `struct` par `class`
- on utilise les mots clés `private:` et `public:` pour définir les zones privées et publiques.
- par défaut, tout est privé dans une classe.



ONERA
THE FRENCH AEROSPACE LAB

# Classes

```
// matrice.h
class Matrice{
  // prive par defaut
  int m,n;

public: //public a partir d'ici

  //methodes
  void cree(int m1,int n1);
  void detruit();
  double get(int i,int j);
  void set(int i,int j,double x);
  void affiche();

private: //prive a partir d'ici
    double* t;

};
```

Cela ne change rien aux définition des méthodes.

L'utilisateur n'a plus accès aux champs `m`, `n` et `t`.

Il est possible de mettre des méthodes dans les zones privées.



ONERA
THE FRENCH AEROSPACE LAB

# Classes vs structures

Une structure est une classe ou tout est public.



ONERA
THE FRENCH AEROSPACE LAB

# Accesseurs

Les **accesseurs** permettent de lire ou d'écrire dans les champs privés des objets.

```
double get(int i, int j);
void set(int i, int j, double x);
```

Maintenant que `m` et `n` sont aussi privés il faut aussi définir des accesseurs en lecture pour ces champs (pas en écriture).

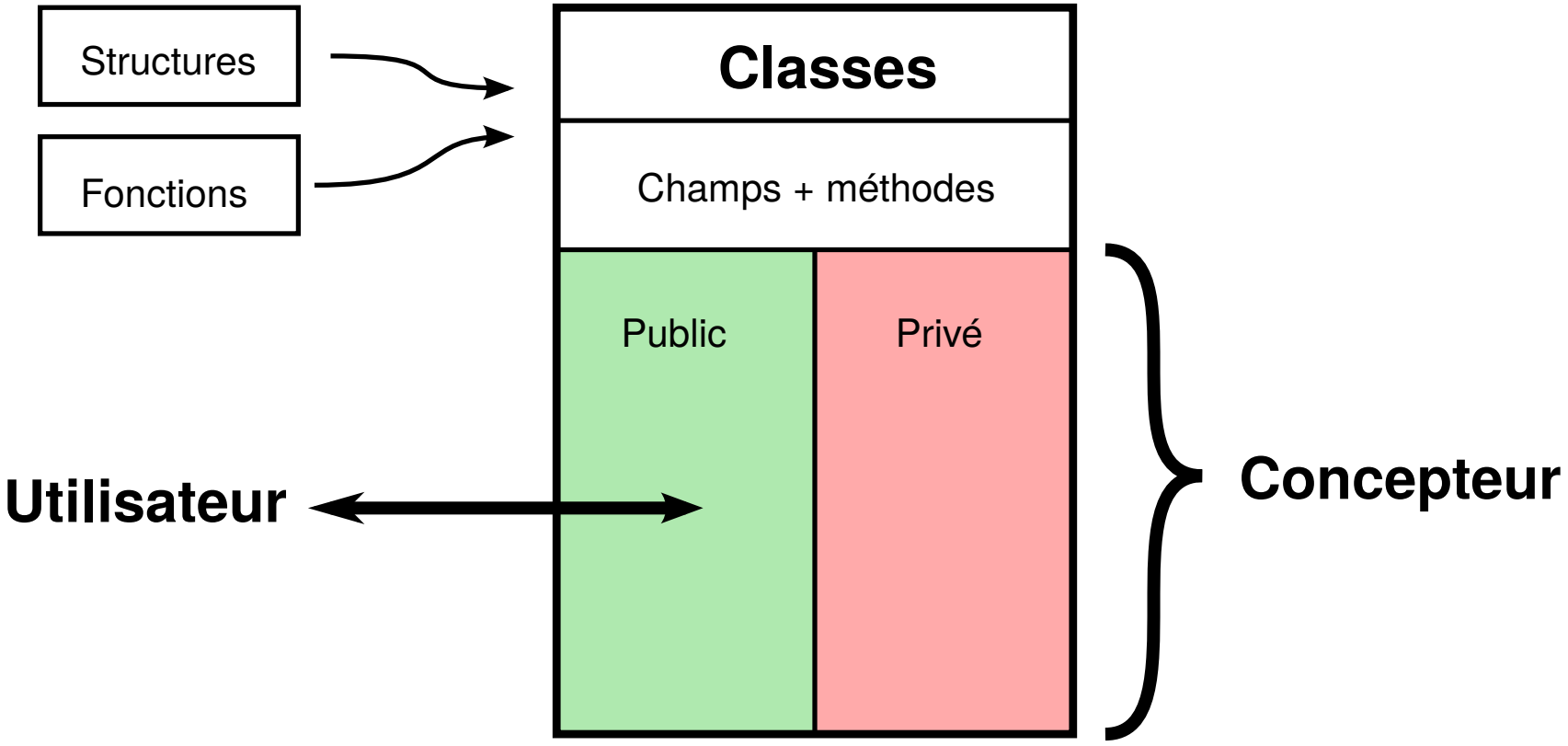
```
int get_m();
int get_n();
```

et les placés dans la partie publique de Matrice.



ONERA
THE FRENCH AEROSPACE LAB

# Résumé






ONERA
THE FRENCH AEROSPACE LAB

# Plan de la séance



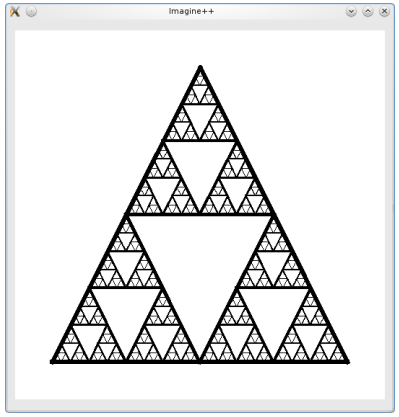


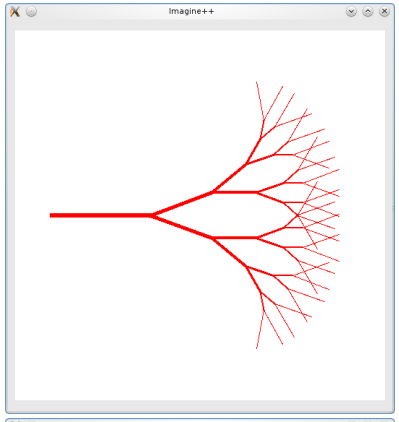
ONERA
THE FRENCH AEROSPACE LAB

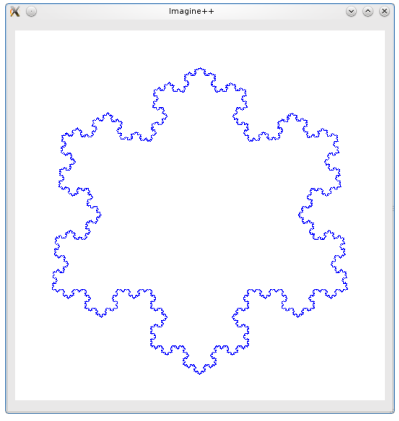
# TP

## Fractales

- Objets
- Fonctions récursives






ONERA
THE FRENCH AEROSPACE LAB