Cours de Programmation Orientée Objet

P.O.O

version. 4.1– 31/08/2020

Michel GRIMALDI
http://grimaldi.univ-tln.fr
grimaldi@univ-tln.fr

Programme Pédagogique National G.E.I.I.

PPN GEII 2013

Référence de l'UE **UE31**	Nom de l'UE Composants, systèmes et applications Matière : **Informatique**	Volume Horaire **30h (6CM, 14TD, 10TP)**
Référence du module **M 3105 C**	Module **Programmation orientée objet**	Semestre **S3**

Objectifs du module :
Comprendre une démarche de conception orientée objet.
Se familiariser avec un langage à objets.

C++

Compétences visées :
Découper une application en objets,
Exprimer un cahier des charges en UML,
Utiliser un paquetage de classes pour construire un objet composite,
Utiliser le polymorphisme,
Programmer en langage objet.

Pré-requis :
Modules M 1103 (Info1).

Contenus :
Penser objet : définir une classe, définir un objet, établir des liaisons entre objets, constructeurs, destructeurs, interfaces, méthodes, propriétés, objets internes.
Construire une application en langage objet.
Les API standard.

API Qt

Modalités de mise en œuvre :
Ce contenu est un canevas et les départements l'adaptent en fonction des applications visées et de la logistique disponible, y compris le choix du langage abordé.

Prolongements possibles :
Développement d'applications d'envergure en utilisant l'approche objet

Mots clés :
Objet, classe

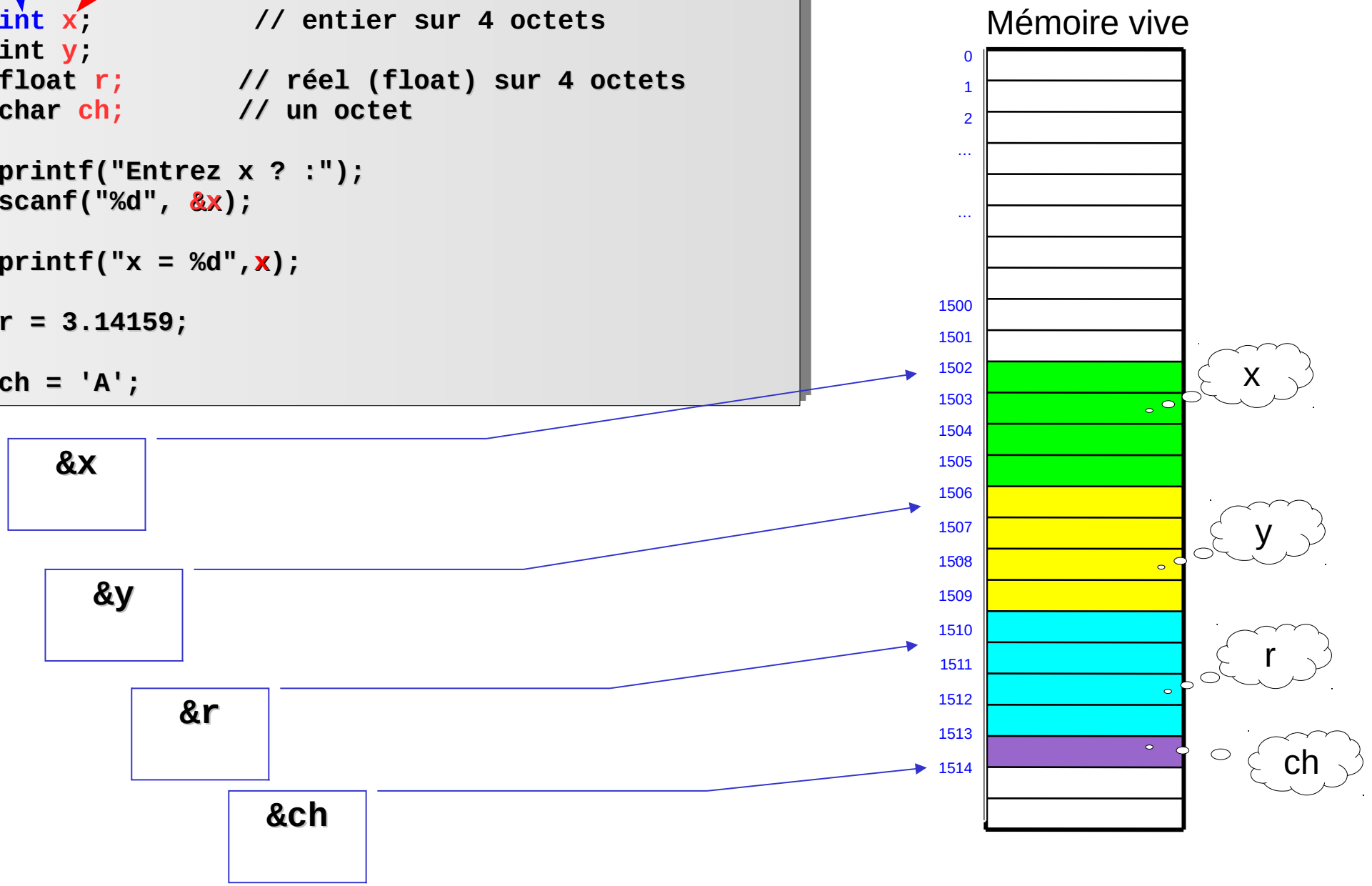
GEII Toulon
4 x 1.5h cours
6 x 3h TD/TP

Prérequis de langage c

Les variables...

type variable

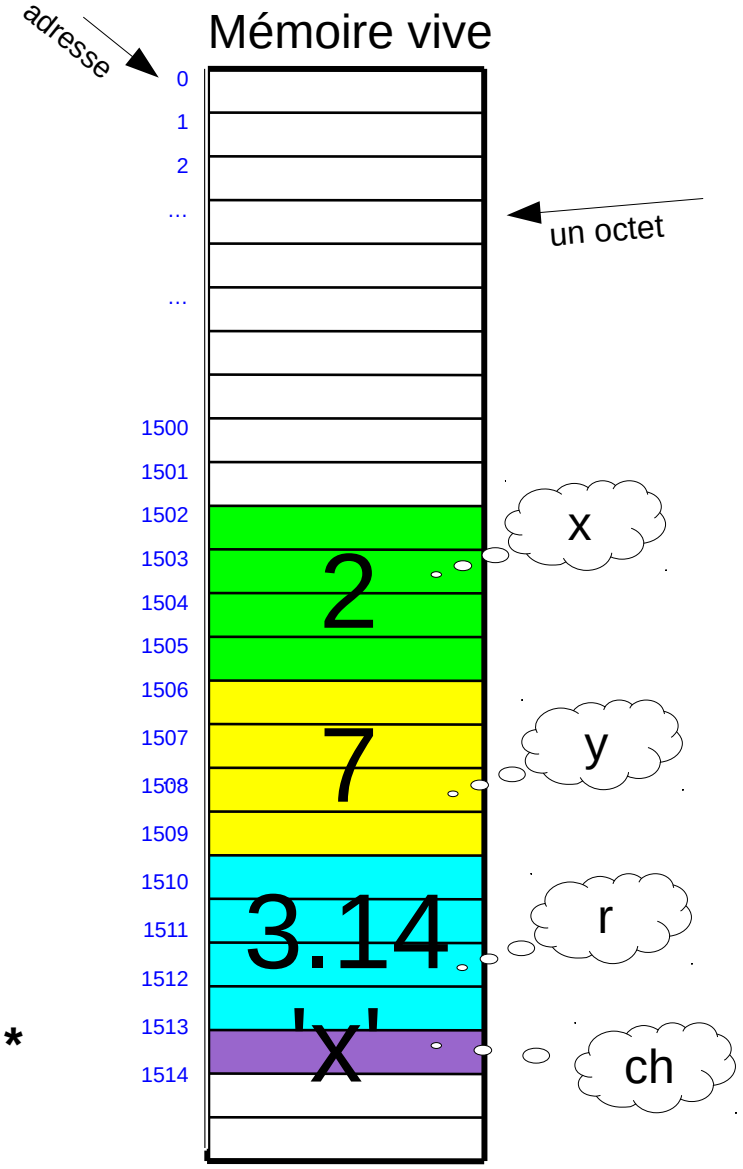
```
int x;          // entier sur 4 octets
int y;
float r;        // réel (float) sur 4 octets
char ch;        // un octet

printf("Entrez x ? :");
scanf("%d", &x);

printf("x = %d",x);

r = 3.14159;

ch = 'A';
```



Chaque variable a une adresse mémoire (du premier octet de la variable → type)

Valeur / adresse

```
int x = 2;                  // entiers sur 4 octets
int y = 7;
float r = 3.14;          // réel (float) sur 4 octets
char ch = 'X';              // un octet
```



Quand j'écris dans le programme

x	→	la valeur de x	→	2
y	→	la valeur de y	→	7
r	→	la valeur de r	→	3.14
ch	→	la valeur de ch	→	'x'
&x	→	l'adresse de x	→	1502
&y	→	l'adresse de y	→	1506
&r	→	l'adresse de r	→	1510
&ch	→	l'adresse de ch	→	1514

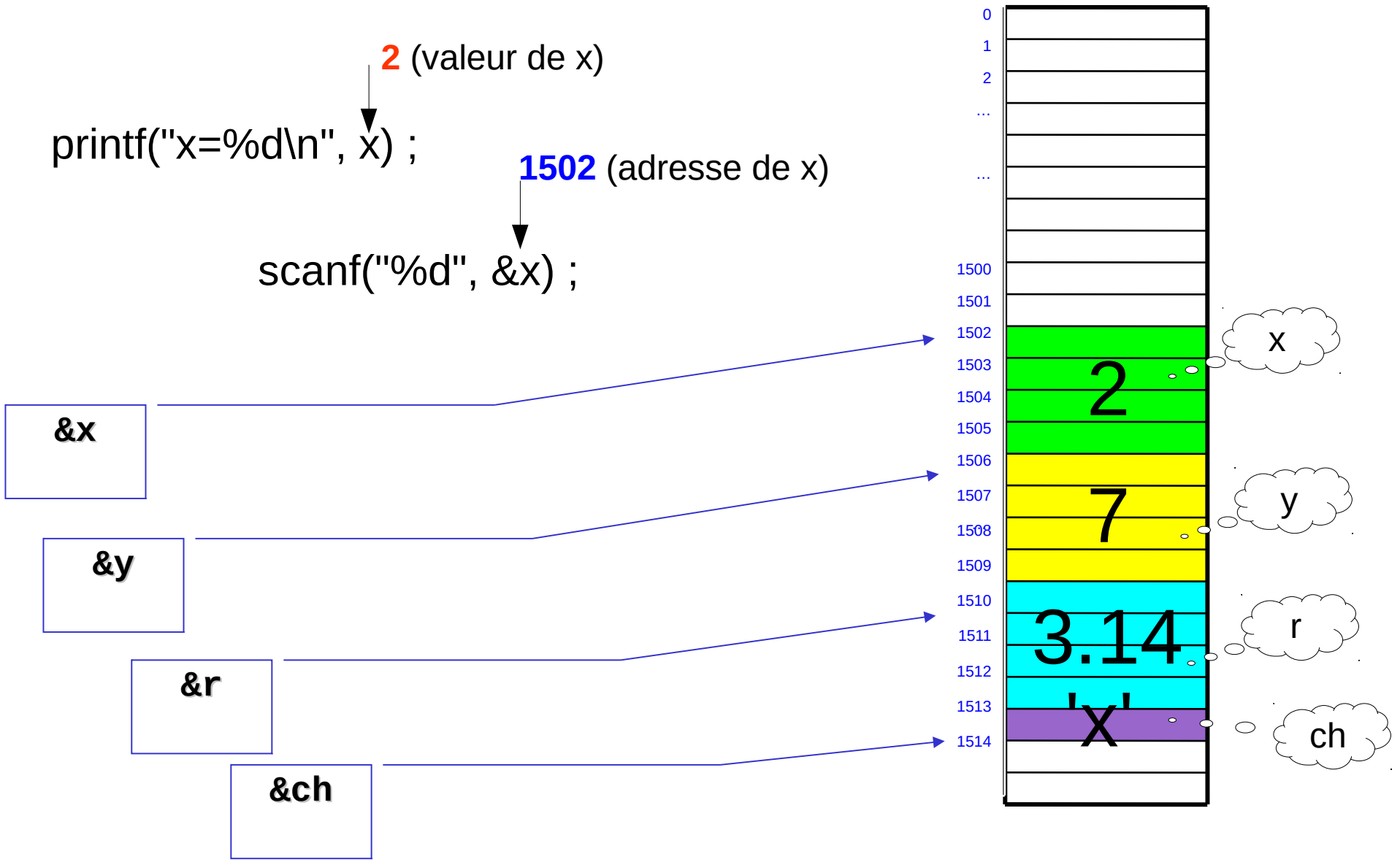
.etc.

Type

&x, &y sont du type "pointeur d'entier" → **`int*`**
&r est du type "pointeur de float" → **`float*`**
&ch est du type "pointeur de char" → **`char*`**

Chaque variable a un type, une adresse et une valeur

Exemple sur deux fonctions bien connues



Chaque variable a un type, une adresse et une valeur

printf et scanf...

2 (valeur de x)

```
printf("x=%d\n", x) ;
```

écrire sur l'écran la **valeur** spécifiée en argument (2) dans le format donné (%d = entier en décimal)

1502 (adresse de x)

```
scanf("%d", &x) ;
```

lire une ligne au clavier, la transformer en fonction du format (%d = entier en décimal) et la ranger à **l'adresse** spécifiée en argument (&x = 1502)

La fonction **printf** ne peut pas modifier la valeur de x car elle ne connaît pas son adresse

Les instructions du c

if (*condition*) *instruction** **else** *instruction** ;
if (*condition*) *instruction** ;

* **si plusieurs *instructions* → un bloc { }**

switch(variable){
    case valeur : *instruction* ;
    case valeur : *instruction* ; **break** ;
    default : *instruction* ;
}

expression conditionnelle
(*condition* **?** *vrai* **:** *faux*)

```
y = (x>10 ? 1 : 0) ;
printf("%s\n", moy>=10 ?"admis" : "ajourné") ;
```

do *instruction** **while** (*condition*);

while (*condition*) *instruction**;

for (*instruction de début ; condition de maintien ; instruction à chaque fois*) *instruction** ;

continue;

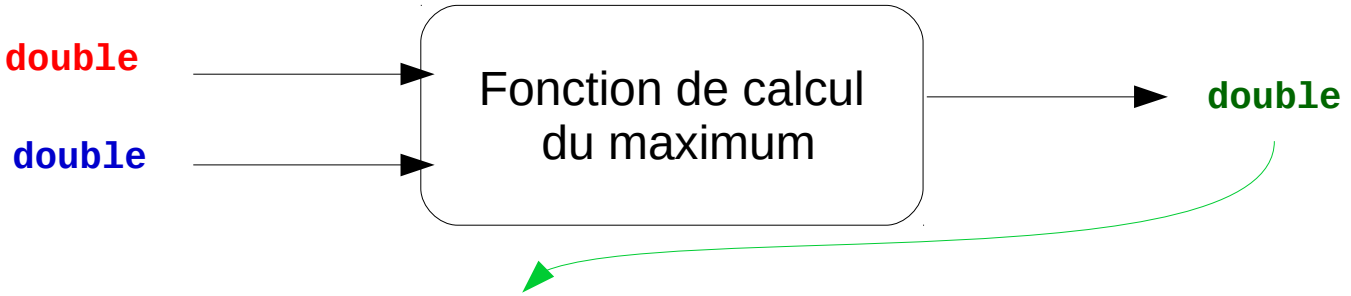
break;

goto;

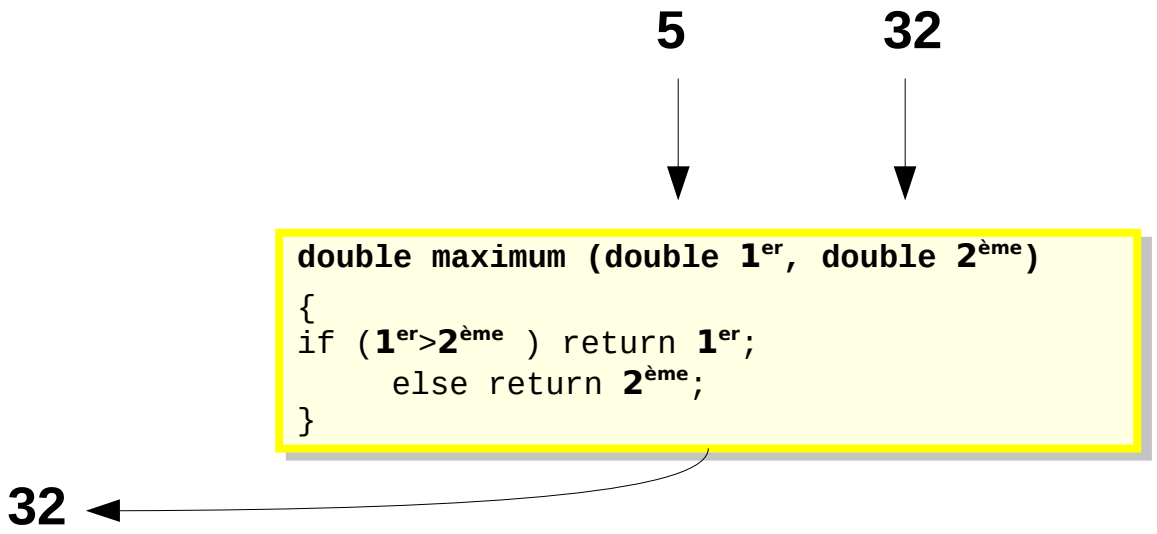
```
int i;
int n=0;

printf("les 20 premières années non bisextiles après 2000\n");
for (i=2000 ; i<3000 ; i=i+1){
          if ((i%4==0 && i%100!=0) || i%400==0) continue;       // bisextile
          if (n>=20) break;
          printf("%04d\t", i);
          n++;
          }
```

```
les 20 premières années non bisextiles après 2000
2001   2002   2003   2005   2006   2007   2009   2010   2011   2013   2014   2015   2017   2018   2019   2021   2022   2023   2025   2026
```

Les fonctions




```
#include <stdio.h>

int main(void){
double x=5, y= 32 ;
double m ;

m = maximum (x, y) ;

printf("le max vaut :%lf", m) ;

return 0 ;
}
```

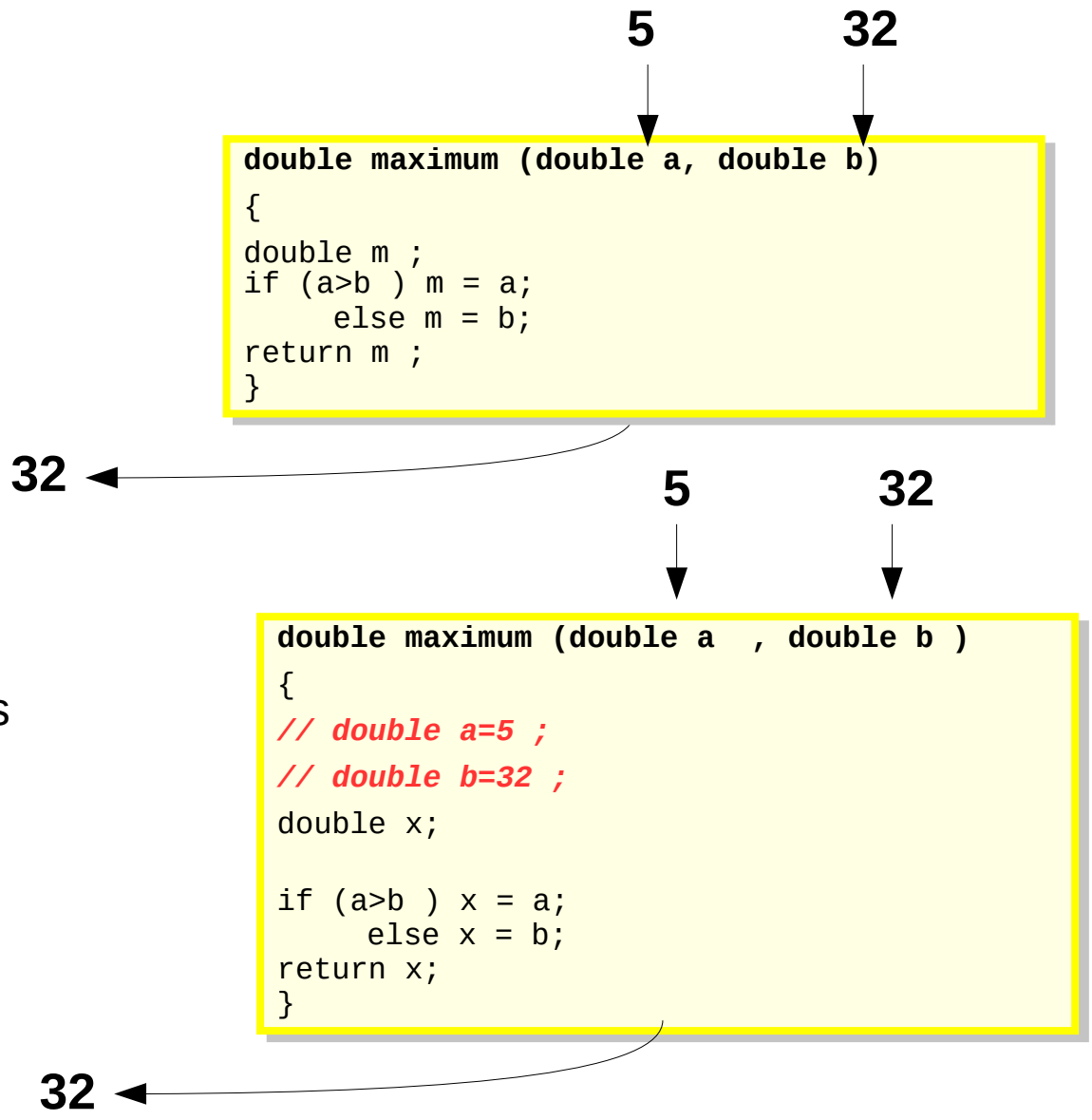



```
double maximum (double a, double b)
{
if (a>b ) return a;
     else return b;
}
```

Arguments = variables locales
muettes

Arguments = variables locales

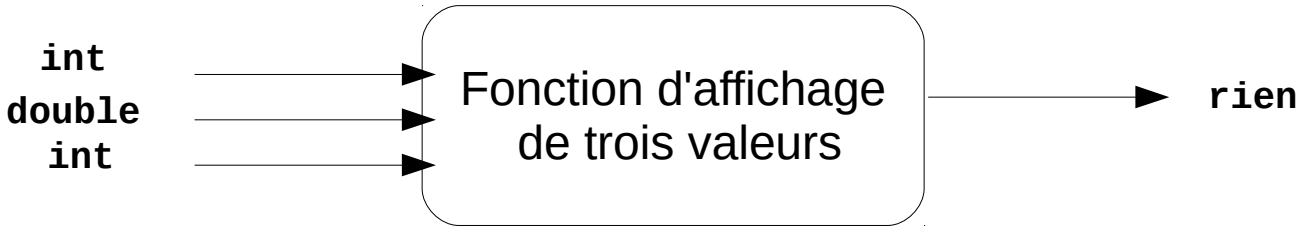
```
#include <stdio.h>

int main(void){
double x=5, y= 32 ;
double m ;

m = maximum (x, y) ;

printf("le max vaut :%lf", m) ;

return 0 ;
}
```



Tout se passe comme si on créait des nouvelles variables locales a et b, visibles uniquement dans la fonction, et qu'on les initialisait avec les valeurs reçues par la fonction (c'est fait automatiquement lors de l'appel)

La variable x du programme n'a pas la même adresse que celle de la fonction !

fonction ne retournant rien (*void*)



Prototype : void affichage (int, double, int)

```
#include <stdio.h>
void affichage (int, double, int);

int main(void){
int x=5, y= 32 ;
affichage(x, 0.333, y) ;

return 0 ;
}
```

5 0.333 32

```
void affichage (int 1er, double 2ème, int 3ème)
{
printf("%d,%f,%d\n", 1er, 2ème , 3ème ) ;
return;
}
```

```
void affichage (int a, double x, int b)
{
printf("%d,%f,%d\n", a, x, b) ;
return;
}
```

La variable x du programme principal, en haut à gauche, est différente de celle de la fonction.

Elles n'ont pas la même adresse mémoire.

fonction sans argument (*void*)

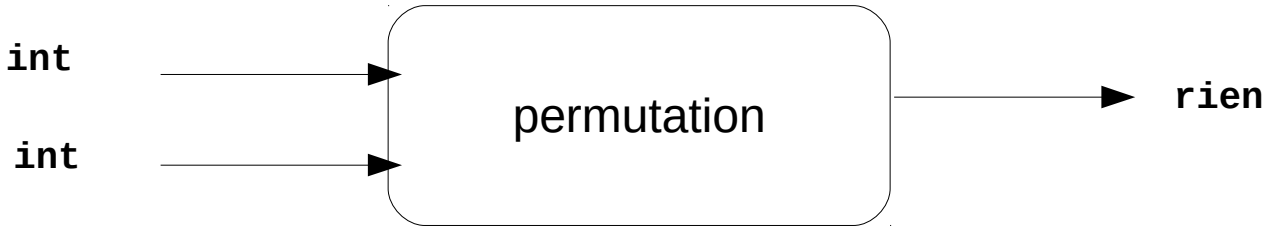
Prototype : double randf (void)

```
#include <stdio.h>
#include <stdlib.h>
double randf(void);

int main(void){
double x;
for (int i=0 ; i<10 ; i++){
                    x = randf();
                    printf("%f\n", x);
                  }
return 0;
}
```

```
0.840188
0.394383
0.783099
0.798440
0.911647
0.197551
0.335223
0.768230
0.277775
0.553970
```

```
// fonction retournant un nombre aléatoire réel
// compris entre 0 et 1
double randf (void)
{
// on utilise la fonction rand() standard qui
// retourne un nombre entier entre 0 et RAND_MAX
return rand()/(double)RAND_MAX ;
}
```

fonction de permutation de deux variables



Prototype : void permute (int, int)

```
#include <stdio.h>
void permute (int, int) ;
int main(void){
int x=5, y= 32 ;

printf("avant :%d,%d\n", x, y) ;
permute(x, y) ;
printf("après :%d,%d\n", x, y) ;
return 0 ;
}
```

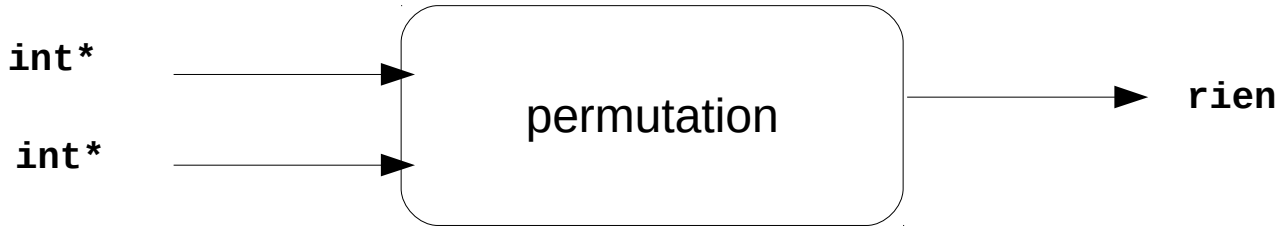
```
avant : 5, 32
après : 5, 32
```

5 32

```
void permute (int ...  , int ... )
{
// int a=5 ;
// int b=32 ;

int c ;
c = a ; a = b ; b = c ;
return ;
}
```

la permutation s'est faite sur les variables locales uniquement.
la fonction ne connaît pas les adresses de x et y du *main*,
elle ne peut donc pas les permuter.

fonction de permutation de deux variables



Prototype : void permute (int*, int*)

```
#include <stdio.h>
void permute (int*, int*) ;
int main(void){
int x=5, y= 32 ;

printf("avant :%d,%d\n", x, y) ;
permute(&x, &y) ;
printf("après :%d,%d\n", x, y) ;
return 0 ;
}
```

```
avant : 5, 32
après : 32, 5
```

l'adresse de x — l'adresse de y

```
void permute (int* ad1  , int* ad2 )
{
int c ;

c = *ad1 ;
*ad1 = *ad2 ;
*ad2 = c ;
return ;
}
```

*ad1 ~ l'entier qui se trouve à l'adresse ad1
*ad2 ~ l'entier qui se trouve à l'adresse ad2

adresse d'une variable ~ pointeur

&x ~ l'adresse de x ~ type **int*** ~ pointeur d'entier
&y ~ l'adresse de y ~ type **int*** ~ pointeur d'entier

ad1 ~ type **int*** ~ pointeur d'entier
ad2 ~ type **int*** ~ pointeur d'entier

l'adresse de x **l'adresse de y**

```
#include <stdio.h>
void permute (int*, int*) ;
int main(void){
int x=5, y= 32 ;

printf("avant :%d,%d\n", x, y) ;
permute(&x, &y) ;
printf("après :%d,%d\n", x, y) ;
return 0 ;
}
```

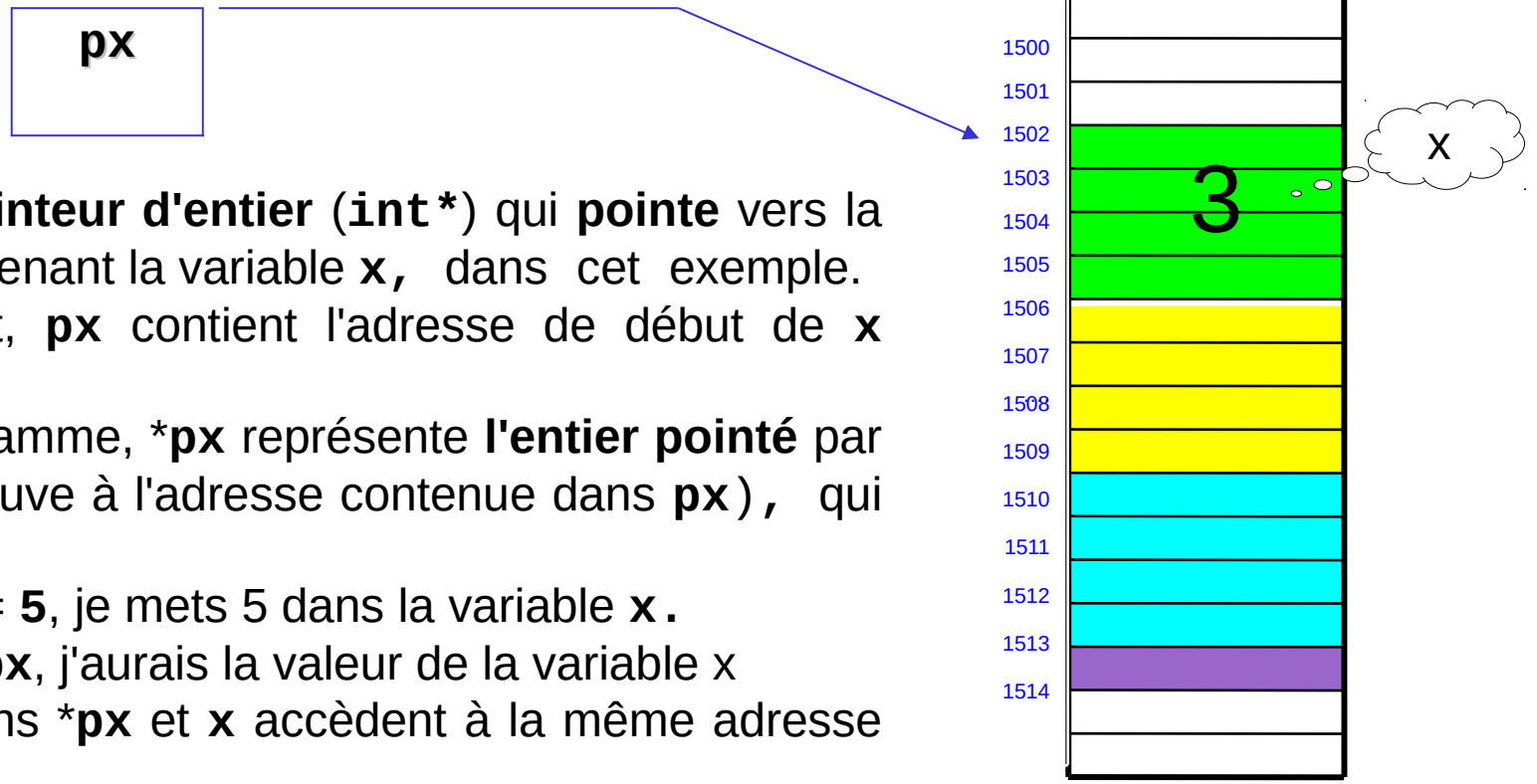
```
avant : 5, 32
après : 32, 5
```

```
void permute (int* ad1  , int* ad2 )
{
int c ;

c = *ad1 ;
*ad1 = *ad2 ;
*ad2 = c ;
return ;
}
```

*ad1 ~ l'entier à l'adresse ad1~ valeur pointée
*ad2 ~ l'entier à l'adresse ad2~ valeur pointée

Pointeur (px) et valeur pointée (*px)

```
int x = 3;          // entier sur 4 octets
int* px;


px = &x;           // px reçoit l'adresse de x

*px = 5;           // 5 dans x !
```



- **px** est un **pointeur d'entier** (**int***) qui **pointe** vers la mémoire contenant la variable **x,** dans cet exemple.
- physiquement, **px** contient l'adresse de début de **x** (1502 ici)
- dans le programme, ***px** représente **l'entier pointé** par **px** (qui se trouve à l'adresse contenue dans **px**), qui vaut 3 ici.
- si j'écris ***px** = **5**, je mets 5 dans la variable **x.**
- si j'imprime ***px**, j'aurais la valeur de la variable x
- les expressions ***px** et **x** accèdent à la même adresse mémoire.

Les pointeurs sont surtout utilisés lors des passages de paramètres par adresse, dans les fonctions.

Tableau de ... ~ pointeur de ...

La variable représentant un **tableau** est un **pointeur** qui contient l'adresse du premier élément

```
#include <stdio.h>
void affiche (int*, int) ;
// void affiche (int [], int) ;

int main(void){
int t[5] = {12, -25, 4, 189, 7} ;

affiche(t, 5) ;
return 0 ;
}
```

t ~ type **int*** ~ pointeur d'entier

l'adresse du tableau — **La valeur du nombre d'éléments**

```
void affiche (int* tab  , int nbre )
{
for (int i=0 ; i<nbre ; i++){
     printf("%d\t", tab[i]) ;
     }
printf("\n") ;
return ;
}
```

```
12   -25  4    189  7
```

écriture équivalente →

```
void affiche (int tab[]  , int nbre )
{
for (int i=0 ; i<nbre ; i++){
     printf("%d\t", tab[i]) ;
     }
printf("\n") ;
return ;
}
```

Tableau ~ pointeur

```c
#include <stdio.h>
int main(void){
int t[5] = {12, -25, 4, 189, 7};

printf("l'adresse de début est %x\n", t);
for (int i=0 ; i<5 ; i++){
     printf("l'element %d est à l'adresse %x\n", i, &t[i]);
     }
return 0;
}
```

```
l'adresse de début est b1369760
l'element 0 est à l'adresse b1369760
l'element 1 est à l'adresse b1369764
l'element 2 est à l'adresse b1369768
l'element 3 est à l'adresse b136976c
l'element 4 est à l'adresse b1369770
```

Afficher la valeur d'un pointeur est possible même si n'a pas d'intérêt, autre que la compréhension.
Nous ne maîtrisons pas les adresses des variables ou des tableaux.

```c
#include <stdio.h>
int main(void){
double t[5] = {12, -25, 4, 189, 7};

printf("l'adresse de début est %x\n", t);
for (int i=0 ; i<5 ; i++){
     printf("l'element %d est à l'adresse %x\n", i, &t[i]);
     }
return 0;
}
```

```
l'adresse de début est eff143e0
l'element 0 est à l'adresse eff143e0
l'element 1 est à l'adresse eff143e8
l'element 2 est à l'adresse eff143f0
l'element 3 est à l'adresse eff143f8
l'element 4 est à l'adresse eff14400
```

Quelle est la taille, en octet, occupée par un ***int*** ?
Quelle est la taille, en octet, occupée par un ***double*** ?

Type structuré

```c
#include <stdio.h>
typedef struct{
     int x;
     int y;
     int z;
} Position3D;

void affiche(Position3D);
void entre(Position3D*);

int main(void){
Position3D p1;
p1.x = 100; p1.y = 200; p1.z = 10;

affiche(p1);
entre(&p1);
affiche(p1);
return 0;
}
void affiche(Position3D p){
     printf("(%d, %d, %d)\n", p.x, p.y, p.z);
}

void entre(Position3D *pt){
     printf("x? :"); scanf("%d", &(pt->x));
     printf("y? :"); scanf("%d", &(pt->y));
     printf("z? :"); scanf("%d", &(pt->z));
     }
```

Déclaration du type

variable

Programme

Fonctions

pour accéder aux éléments d'une **variable** structurée:

p1.x
p1.y
p1.z

```
(100, 200, 10)
x? :10
y? :20
z? :30
(10, 20, 30)
```

pour accéder aux éléments d'une structure à partir d'un **pointeur** pt (de type Position3D*) :

(*pt).x

ou

pt->x

Type structuré, autre exemple

```c
#include <stdio.h>

typedef struct{
     char nom[25] ;
     char prenom[25];
     int jourNaissance ;
     int moisNaissance ;
     int anNaissance;
} Personne;

int main(void){
Personne p1, p2, ptab[10];

printf("nom : ") ; scanf("%25s", p1.nom) ;
printf("prenom : ") ; scanf("%25s", p1.prenom) ;
printf("jour : ") ; scanf("%d", &p1.jourNaissance) ;
printf("mois : ") ; scanf("%d", &p1.moisNaissance) ;
printf("annee : ") ; scanf("%d", &p1.anNaissance) ;

for (int i=0 ; i<10 ; i++){
  printf("nom : ") ; scanf("%25s", ptab[i].nom) ;
  printf("prenom : ") ; scanf("%25s", ptab[i].prenom);
          …
  }
```

```c
#include <stdio.h>
typedef struct{
     int jour;
     int mois;
     int an;
} Date;

typedef struct{
     char nom[25] ;
     char prenom[25];
     Date naissance ;
} Personne;

int main(void){
Personne p1;

printf("nom : ") ;
scanf("%25s", p1.nom) ;
printf("prenom : ") ;
scanf("%25s", p1.prenom) ;
printf("jour : ");
scanf("%d", &p1.naissance.jour) ;
printf("mois : ") ;
scanf("%d", &p1.naissance.mois) ;
printf("annee : ") ;
scanf("%d", &p1.naissance.an) ;
```

Quiz

```
#include <stdio.h>
typedef struct{
     int jour;
     int mois;
     int an;
} Date;

typedef struct{
     char nom[25] ;
     char prenom[25];
     Date naissance ;
} Personne;


int main(){
Personne p1, ptab[10] ;
...
```

Quel est **type** des expressions suivantes ?

p1 →

ptab[3] →

ptab →

p1.nom →

p1.nom[5] →

p1.naissance →

p1.naissance.an →

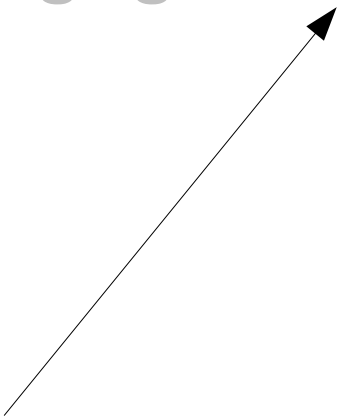
&p1 →

&p1.naissance →

&p1.naissance.jour →

Fin des prérequis de langage c ...

... le langage c++



Ou **C#**, ou **JAVA**, ou **Python ...**

Langage orienté objet

Orienté Objet, pourquoi ?

World of tanks

Orienté Objet, pourquoi ?

scène

Stz5

Crusader

Tank

actions

joueur

Maus

Panzerwerfer

Orienté Objet, pourquoi ?

représentation d'un tank

(*concept de tank*)

BEAUCOUP DE VARIABLES

CARACTERISTIQUES

- modèle → model : string
- armée → army : string
- position/scène → $x, y, z, \theta, \phi, \psi : double$
- orientation de la tourelle → $\alpha, \omega : double$
- vitesse, accélération → $v_x, v_y, v_z : double$
- niveau de carburant → $\gamma_x, \gamma_y, \gamma_z : double$
- type d'armement
- munitions
- ...
- modèle 3D de dessin

Orienté Objet, pourquoi ?

représentation
d'un tank

BEAUCOUP DE FONCTIONS

COMPORTEMENT

- avancer
- reculer
- aller à
- tourner sur place
- orienter le canon
- mitrailler
- tirer au canon
- ...

$avancer(double\ dist)$

$reculer(double\ dist)$

$aller(double\ x, double\ y, double\ z)$

$tourner(double\ alpha)$

Orienté Objet, pourquoi ?

Beaucoup de tanks

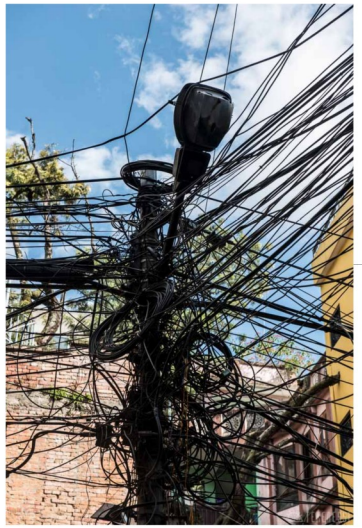
BEAUCOUP, BEAUCOUP DE VARIABLES,
BEAUCOUP BEAUCOUP DE FONCTIONS,

BEAUCOUP BEAUCOUP BEAUCOUP DE COMPLEXITÉ !

Orienté Objet, pourquoi ?

La **Programmation Orientée Objet** permet de fractionner la complexité.

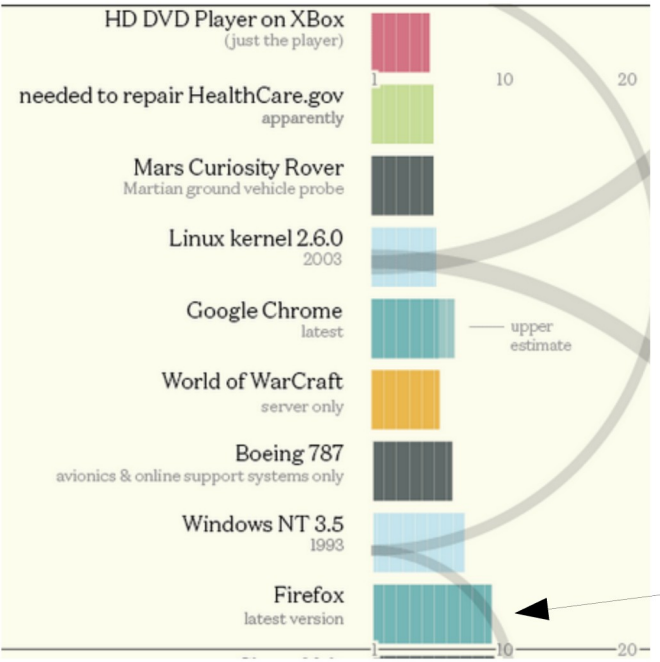
- en représentant totalement chaque concept utilisé par ses **caractéristiques** et son **comportement,** au sein **d'une seule entité** appelée **CLASSE**, de façon indépendante du reste du programme (classe Scène, classe Joueur, classe Tank, *.etc.*).
- en instanciant autant d'**OBJETS** qu'on le désire de ces classes, c'est à dire ayant leurs caractéristiques et leur comportement (ex : 2 joueurs, 1 scène, 50 tanks, *.etc.*)
- en appelant les fonctions de ces objets à souhait pour réaliser le scenario désiré.

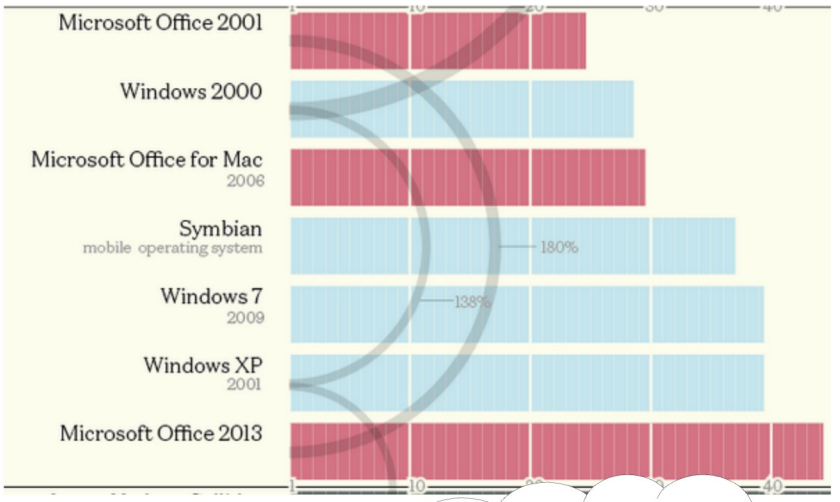


un système
hyper complexe

beaucoup
de systèmes
simples,
plus faciles à
appréhender

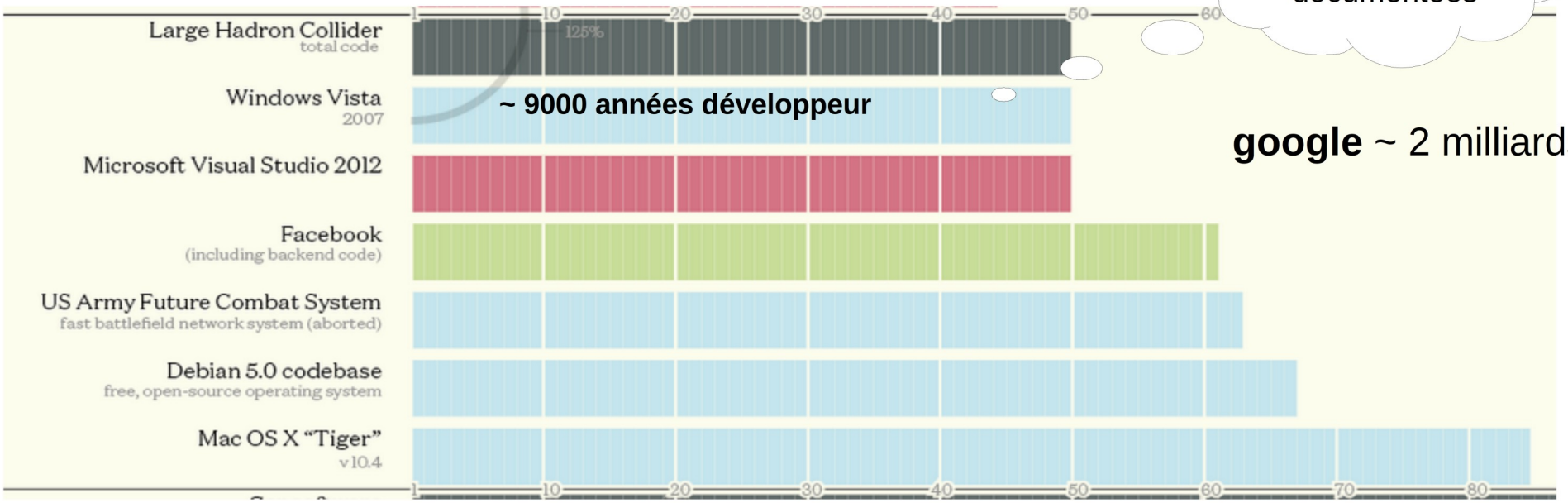
Complexité de quelques programmes connus





9 Millions de lignes de code

<30 lignes
par jour, validées et
documentées



google ~ 2 milliards SLOC

Orienté Objet en c++, comment ?

```
class Tank{
   // attributs
   string model ;
   string army ;
   double x, y, z ;
   double teta, phi, psy ;
   …

   // méthodes
   void avancer(double distance) ;
   void tourner(double angle) ;
   void dessiner()
   …
} ;


// programme
Tank t1;
Tank t2("Stz5", "Russie", …) ;
Tank t3("Cruzader", "GB", …) ;


t1.x = 100 ; t1.y = 100 ; t1.z = 0 ;
t1.avancer(10) ;
t2.tourner(25) ;
...

.etc.
```

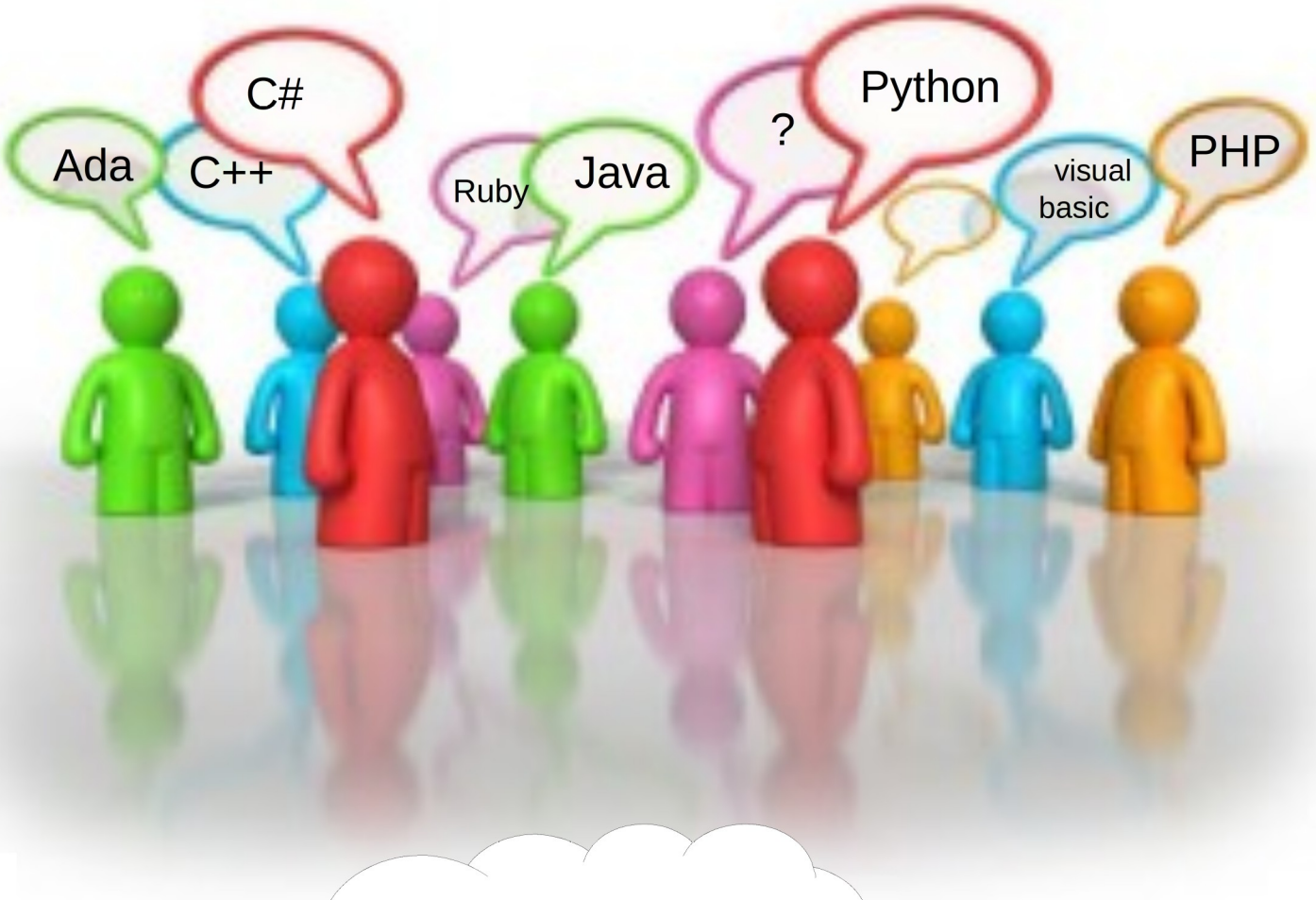
définition de la **classe** Tank

ENCAPSULATION

instanciation **d'objets** de la **classe** Tank

appel des **méthodes** des **objets** de la **classe** Tank

Programmation orientée objet, quel langage ?

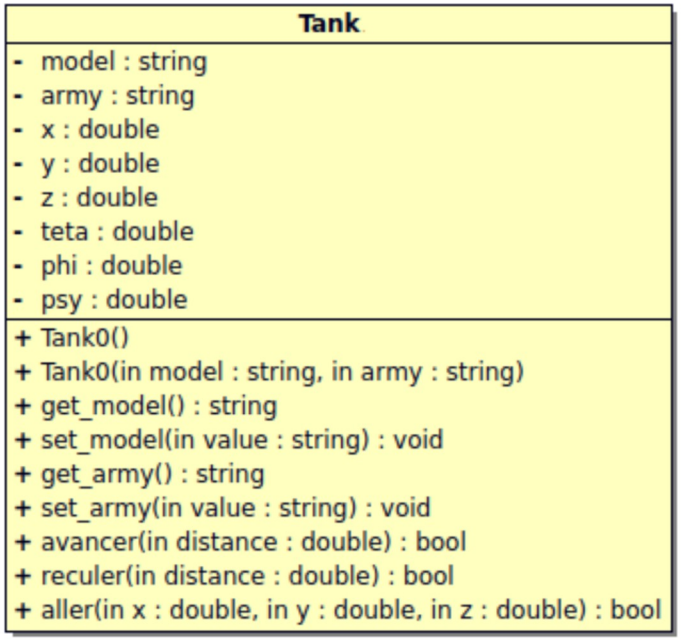


comment faire travailler tous ces gens ensemble ?

Représentation graphique unifiée, UML

Visibilité :
+ public
- private
protected

Nom de la classe



attributs

Méthodes

Analyse conception OO

Découpage du problème en classes, relations, objets, *.etc*.

Génération automatique du code dans le langage désiré

Génération de la documentation

.etc.

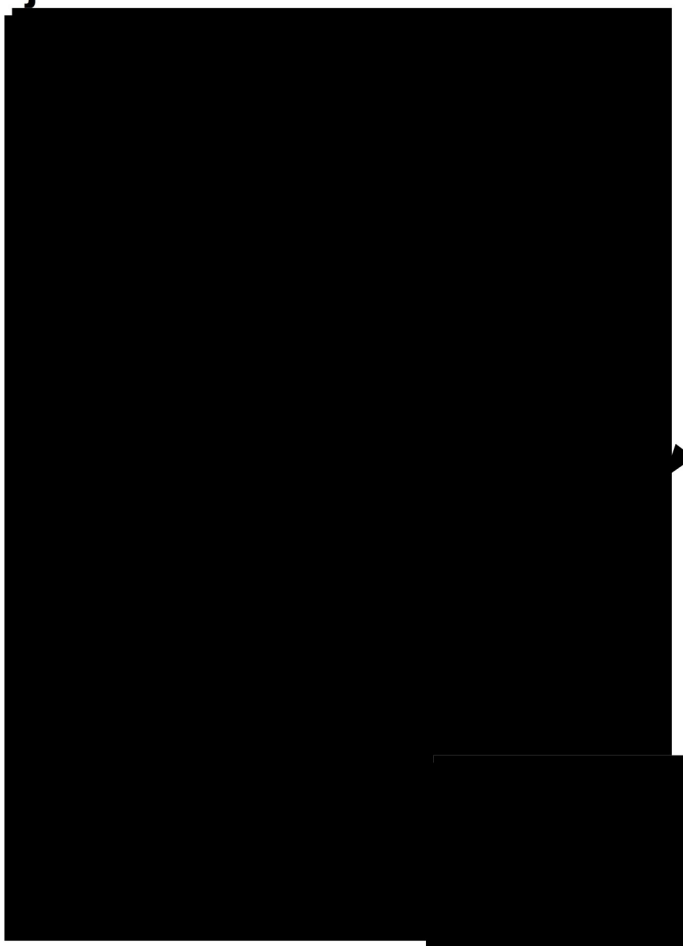
BOUML est une suite d'outils UML 2 gratuits comprenant un modeleur vous permettant de spécifier et générer du code C++, Java, Idl, Php, Python et MySQL.
https://www.bouml.fr

en UML, le type est spécifié après l'identifiant
x : double ~ x est un double

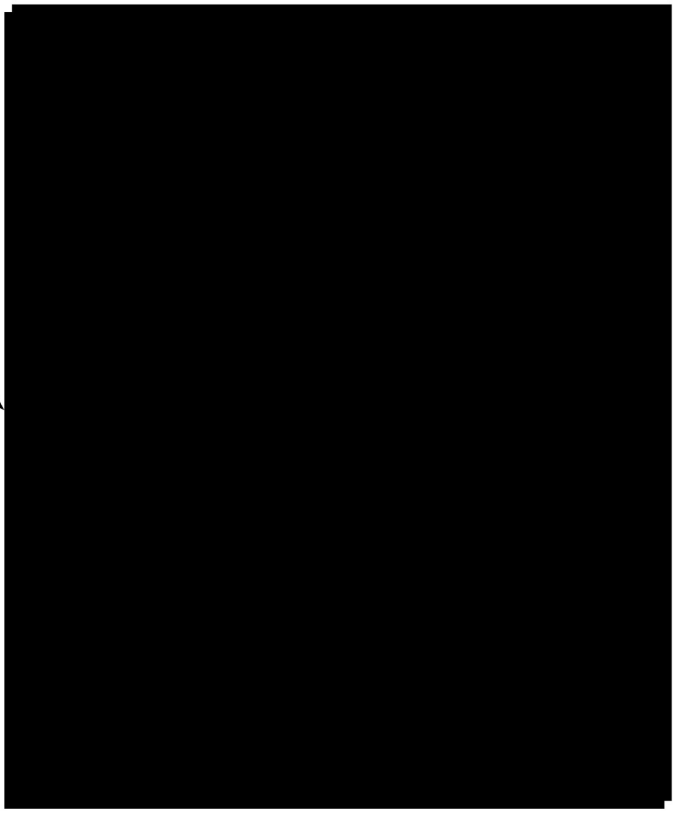
ArgoUML StarUML Modelio Umbrello Visual Paradigm

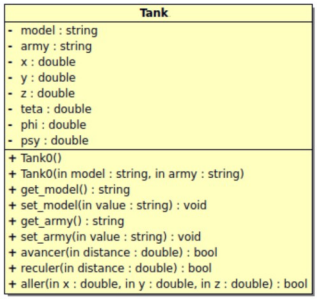
Unified Modeling Language ~ Indépendance du langage utilisé

java

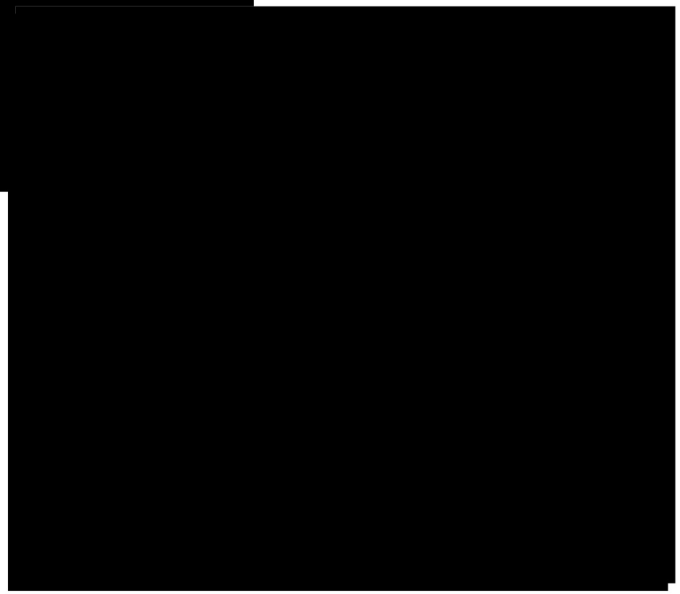


c++

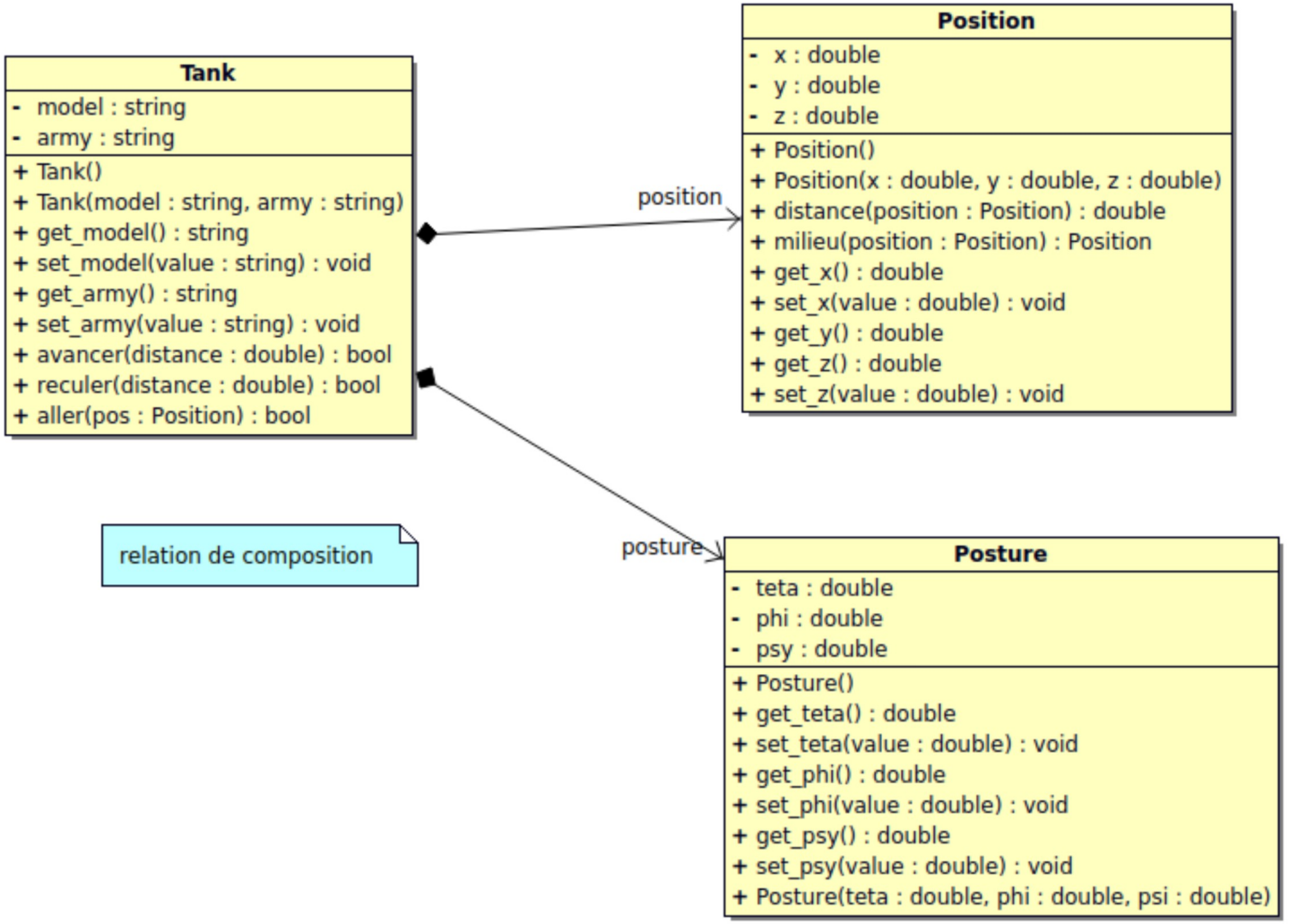




python



Relations entre classes : composition



Pierre

Paul

Règle : **Un seul concept par classe**, une classe pour chaque concept !

Équivalence UML / c++

Position
- x : double - y : double - z : double
+ Position() + Position(x : double, y : double, z : double) + distance(position : Position) : double + milieu(position : Position) : Position + get_x() : double + set_x(value : double) : void + get_y() : double + get_z() : double + set_z(value : double) : void

équivalence

code c++

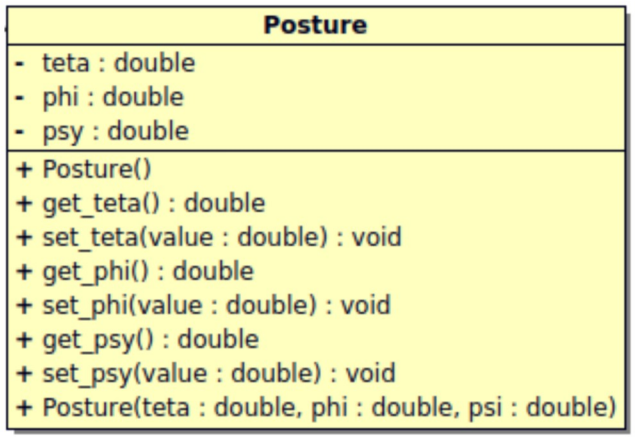
Pierre

Visibilité :

+	public	→	tout le monde peut y accéder
-	private	→	personne d'autre ne peut y accéder
#	protected	→	on verra plus tard !

Déclaration de la classe position.h

Équivalence UML / c++



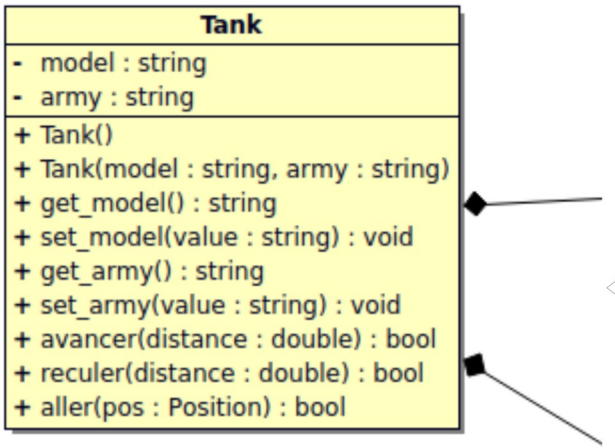
équivalence

code c++

Paul

Le **codage**, la **mise au point** et la **validation** de chaque classe sont indépendants du reste du programme.

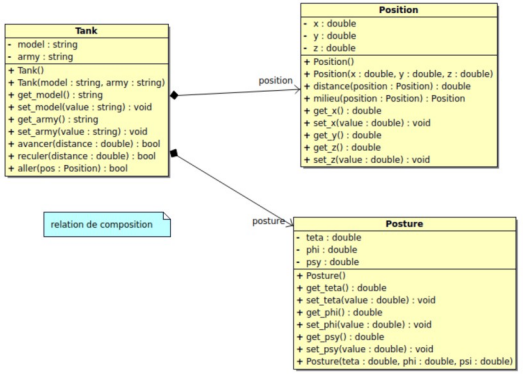
Équivalence c++, relation de composition



équivalence

code c++

```
Position position;
Posture posture;
```



Un Tank **est composé**, entre-autre, d'une Position et d'une Posture

En UML, comme dans tous les langages de POO, on différencie majuscules et minuscules

Position position;

position est un objet (*~variable en c*) de la classe **Position** *(~type en c)*

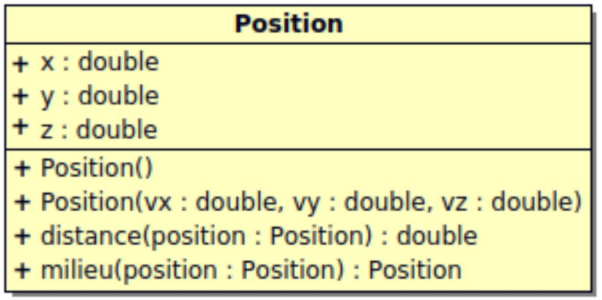
Implémentation des méthodes de la classe

Après avoir déclaré la classe, ses attributs, ses méthodes, et les relations qui la lient à d'autres classes, il faudra écrire, à proprement dit, le code des méthodes.

Ceci s'appelle l'implémentation des méthodes de la classe.

Bien qu'on puisse tout écrire dans un seul et même fichier incluant le programme (.cpp), généralement en c++ la classe est définie dans un fichier avec l'extension .h et l'implémentation des méthodes est faite dans un autre fichier avec l'extension .cpp.

En Java ou en python par exemple, les méthodes sont directement implémentées dans la déclaration (un seul fichier).



Implémentation des méthodes en c++

```
class Position {
  public:
    double x;
    double y;
    double z;

  public:
    Position();
    Position(double vx, double vy, double vz);

    double distance(Position position);
    Position milieu(Position position);
};
```

Déclaration
position.h

de la classe Position

méthode distance

double Position::distance(Position);

renvoie un double

on lui passe une Position

```
#include <stdio.h>
#include "position.h"

int main(void){

Position p1(20,0,30) ;
Position p2(100,0,0) ;

double dist1 ;
dist1 = p1.distance(p2) ;

…
```

p1, donne moi ta distance à p2

Programme
main.cpp

```
#include "position.h"

//constructeur par défaut
 Position::Position() {
       x = y = z = 0.0;
}

//constructeur
 Position::Position(double vx, double vy, double vz)
{
       x = vx ;
       y = vy ;
       z = vz ;
}

//distance à une autre position
double Position::distance(Position position) {
     double dist = sqrt((x-position.x)*
     (x-position.x) + (y-position.y)*
     (y-position.y) + (z-position.z)*
     (z-position.z));
     return dist;
}

//milieu entre la position et une autre position
Position Position::milieu(Position position) {
     Position milieu;
     …
     return milieu;
}
```

Implémentation des méthodes (fonctions)
position.cpp

Constructeur : même nom que la classe

```
class Position {
  public:
    double x;
    double y;
    double z;

  public:
    Position();
    Position(double x, double y, double z);
    …
};
```

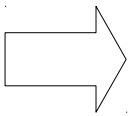
Déclaration

```
//constructeur par défaut
 Position::Position() {
       x = y = z = 0.0;
}

//constructeur
Position::Position(double px, double py, double pz) {
      x = px;
      y = py;
      z = pz;
}
…
```

Implémentation des constructeurs

Programme

```
int main(){
Position p1;
Position p2(100, 50, 20);
Position p3;

p1.x = 900;
p1.y = 300;
p1.z = 100;

printf("p1: %f, %f, %f\n", p1.x, p1.y, p1.z);
printf("p2: %f, %f, %f\n", p2.x, p2.y, p2.z);
printf("p3: %f, %f, %f\n", p3.x, p3.y, p3.z);

return 0;
}
```



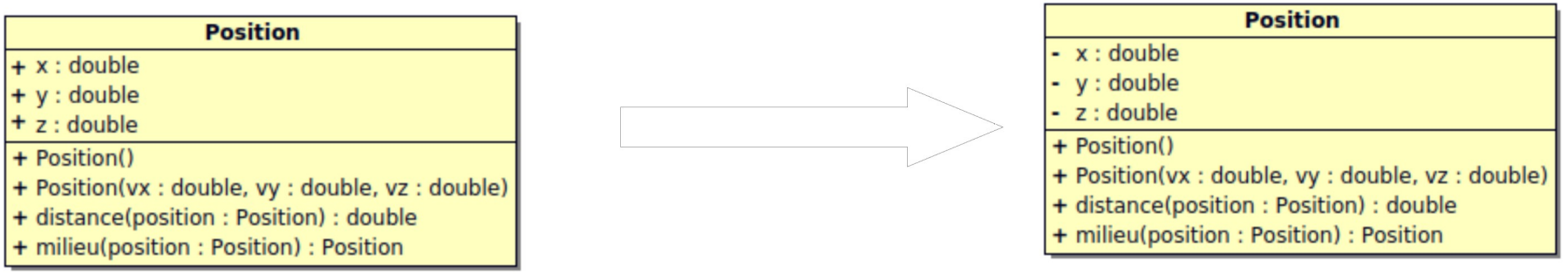
```
p1: 900.000000, 300.000000, 100.000000
p2: 100.000000, 50.000000, 20.000000
p3: 0.000000, 0.000000, 0.000000
```

A chaque fois qu'un objet est créé, son constructeur est automatiquement appelé !

S'il a plusieurs constructeurs, il choisira "celui qui va bien".

Les attributs "public" sont directement accessibles dans le programme !

Accessibilité des attributs




```
public
```

```
private
```

Visibilité :

+	public
-	private
#	protected, on verra plus tard

Accessibilité des attributs

```
class Position {
  private:
    double x;
    double y;
    double z;

  public:
    Position();
    Position(double x, double y, double z);
    …
};
```

Déclaration

```
//constructeur par défaut
 Position::Position() {
       x = y = z = 0.0;
}

//constructeur
Position::Position(double px, double py, double pz) {
      x = px;
      y = py;
      z = pz;
}
…
```

Implémentation des constructeurs

Programme

```
int main(){
Position p1;
Position p2(100, 50, 20);
Position p3;

p1.x = 900;
p1.y = 300;
p1.z = 100;

printf("p1: %f, %f, %f\n", p1.x, p1.y, p1.z);
printf("p2: %f, %f, %f\n", p2.x, p2.y, p2.z);
printf("p3: %f, %f, %f\n", p3.x, p3.y, p3.z);

return 0;
}
```

```
Position.cpp:85:4: error: ‘double Position::x’
is private within this context
 p1.x = 900;
    ^
Position.cpp:12:12: note: declared private
here
     double x;
            ^
Position.cpp:86:4: error: ‘double Position::y’
is private within this context
 p1.y = 300;
    ^
Position.cpp:13:12: note: declared private
here
     double y;
            ^
Position.cpp:87:4: error: ‘double Position::z’
is private within this context
 p1.z = 100;
    ^
…
```

Les attributs "private" ne sont **pas accessibles** dans le programme

Attributs "private", pourquoi ?

En résumé :

- Si je crée un attribut "**`private`**" dans une de mes classes, les programmes qui utiliseront des objets de cette classe ne pourront pas y accéder !
- Quel est l'intérêt ?
- Pourquoi le ferais-je ?
- Comment accéder à ces attributs dans ce cas là ?
- Il faudra bien que je puisse à un moment récupérer ou modifier les coordonnées de ma Position, sinon, à quoi ça sert d'écrire cette classe !
- Encore un truc d'informaticien fou !

Réponse de l'informaticien fou qui écrit la classe :

- Si, par exemple, les coordonnées des positions doivent impérativement être positives ou nulles, ou ne pas dépasser la taille de la scène, comment m'assurer que le programmeur ne va y mettre des valeurs négatives ? Je peux bien le lui dire ou l'écrire dans les consignes, mais nous savons tous que ce n'est pas parce qu'on dit quelque chose que ça va être entendu !
- Je peux lui interdire l'accès direct aux attributs (private), mais lui laisser la possibilité de les modifier en passant par des méthodes "public" que j'écrirai moi-même.
- Il pourra ainsi accéder aux attributs, mais je contrôlerai qu'il n'y met pas de valeurs non autorisées (ex : négatives) !
- Ça complique un peu les choses, mais c'est le prix de la sécurité !
- Tout le monde sera content !

Accesseurs : getter/setter

Déclaration

```
#define XMAX 10000
#define YMAX 10000
#define ZMAX 10000

class Position {
  private:
    double x;
    double y;
    double z;

  public:
    Position();
    Position(double x, double y, double z);

    double get_x(){ return x;};
    void set_x(double value);
    double get_y(){ return y;};
    void set_y(double value);
    double get_z(){ return z;};
    void set_z(double value);
};
```

Implémentation

```
//constructeur par défaut
 Position::Position() {
       x = y = z = 0.0;
}

//constructeur
Position::Position(double px, double py, double pz) {
      x = px;
      y = py;
      z = pz;
}

void Position::set_x(double value) {
  if (value < 0) value = 0;
  if (value > XMAX) value = XMAX;
  x = value;
}

void Position::set_y(double value) {
  if (value < 0) value = 0;
  if (value > YMAX) value = YMAX;
  y = value;
}

void Position::set_z(double value) {
  if (value < 0) value = 0;
  if (value > ZMAX) value = ZMAX;
  z = value;
}
```

Il n'est pas possible de différencier l'accès en lecture et en écriture des attributs
Les accesseurs sont des méthodes, publiques en général, qui permettent l'accès **contrôlé** aux attributs privés

Accesseurs : getter/setter

Programme

```
int main(){
Position p1;
Position p2(100, 50, -20);
Position p3;

p1.set_x(-900);
p1.set_y(300);
p1.set_z(56000);

printf("p1: %f, %f, %f\n", p1.get_x(), p1.get_y(), p1.get_z());
printf("p2: %f, %f, %f\n", p2.get_x(), p2.get_y(), p2.get_z());
printf("p3: %f, %f, %f\n", p3.get_x(), p3.get_y(), p3.get_z());

return 0;
}
```

-900

```
void Position::set_x(double value)
{
  if (value < 0) value = 0;
  if (value > XMAX) value = XMAX;
  x = value;
}
```

```
#define XMAX 10000
#define YMAX 10000
#define ZMAX 10000
```

-20

```
p1: 0.000000, 300.000000, 10000.000000
p2: 100.000000, 50.000000, -20.000000
p3: 0.000000, 0.000000, 0.000000
```

```
//constructeur
 Position::Position(double x, double y, double z) {
      set_x(x);
      set_y(y);
      set_z(z);
}
```

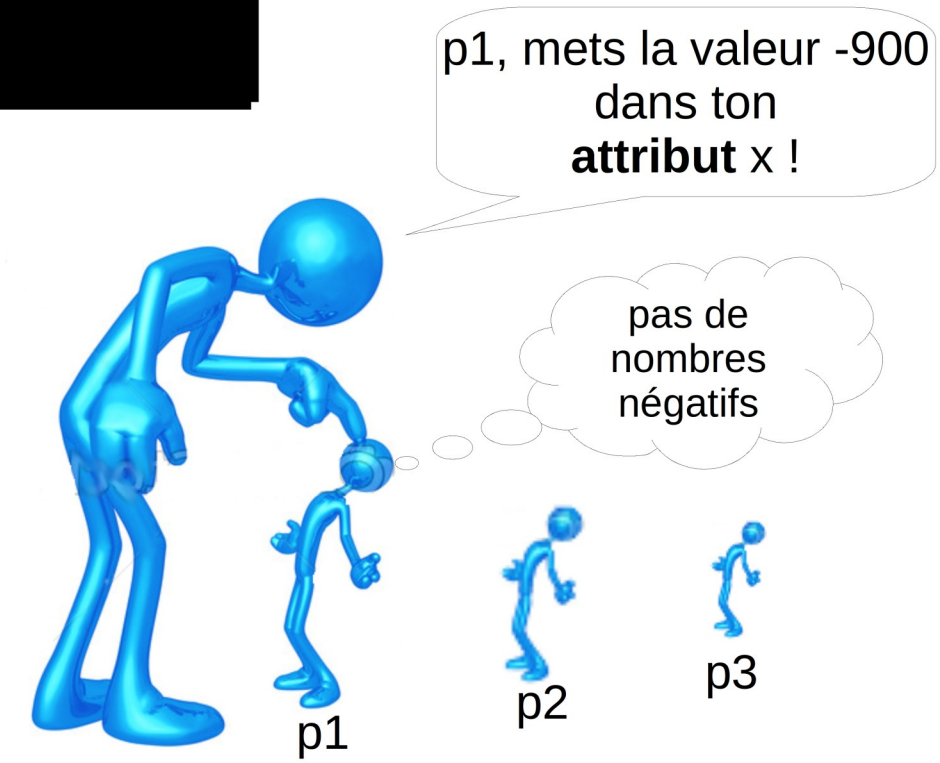
```
p1: 0.000000, 300.000000, 10000.000000
p2: 100.000000, 50.000000, 0.000000
p3: 0.000000, 0.000000, 0.000000
```

Les valeurs des attributs sont **totalement contrôlées** !

Objet actif

```
p1.set_x(-900);
```

En P.O.O, on désigne un **objet** p1, et on lui demande d'exécuter une de ses méthodes



Pendant le déroulement de cette instruction, **p1 est l'objet actif - celui qui travaille !**

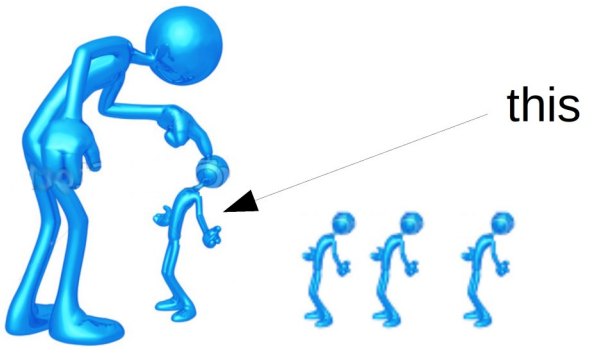
Paramètre ou attribut ? pointeur this

```
x
y
z
```

```
                                x
    x       x
    x           x
this->x     x

                                y
    y       y
    y           y
this->y     y

                                z
    z       z
    z           z
this->z     z
```

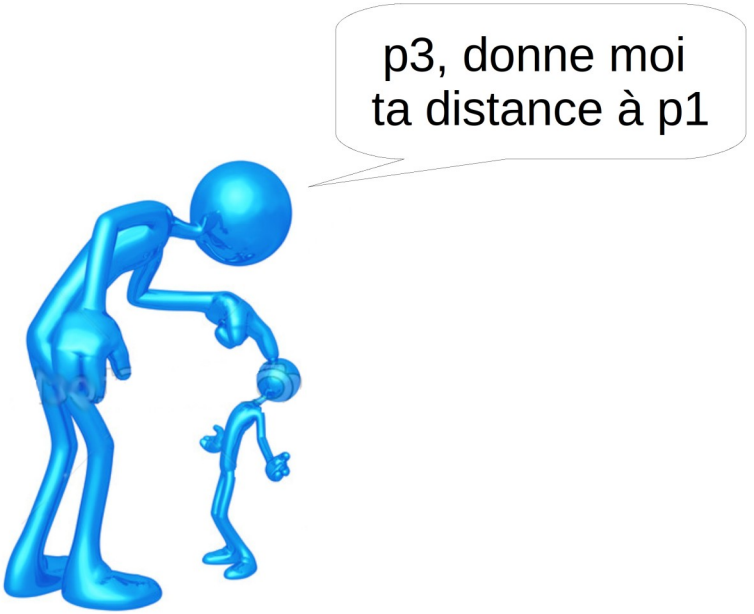
this désigne toujours l'objet actif. (il contient son adresse mémoire, c'est un Position*)

En cas d'ambiguïté (argument et attribut avec le même nom), "**`this->x`**" désigne l'attribut x de l'objet actif, et "**`x`**" désigne l'argument local.



Autres méthodes : milieu et distance

```
double distance(Position position);
Position milieu(Position position);
```




```
p1: 0.000000, 0.000000, 0.000000
p2: 50.000000, 50.000000, 0.000000
p3: 0.000000, 0.000000, 0.000000

p3: 25.000000, 25.000000, 0.000000
d1= 35.355339, d2=35.355339
```

Implémentation de distance et milieu

Déclaration

```
class Position {
  private:
    double x;
    double y;
    double z;

  public:
    Position();
    Position(double x, double y, double z);

    double distance(Position position);
    Position milieu(Position position);

    double get_x(){ return x;};
    void set_x(double x);
    double get_y(){ return y;};
    void set_y(double y);
    double get_z(){ return z;};
    void set_z(double z);
};
```

distance

avec $p_1(x_{1,}\, y_{1,}\, z_1)$

et $p_2(x_{2,}\, y_{2,}\, z_2)$

$$d(p_1, p_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

Milieu

$$m = \left(\frac{(x_2 + x_1)}{2}, \frac{(y_2 + y_1)}{2}, \frac{(z_2 + z_1)}{2}\right)$$

Implémentation

```
//distance à une autre position
double Position::distance(Position position) {
  double dist ;
  dist = sqrt((x-position.x)*(x-position.x)+(y-position.y)*(y-position.y)+(z-position.z)*(z-position.z));
  return dist;
}

//milieu entre la position et une autre position
Position Position::milieu(Position position) {
     Position m;
     m.set_x((x+position.x)/2.);
     m.set_y((y+position.y)/2.);
     m.set_z((y+position.z)/2.);
     return m;
}
```

Implémentation de distance et milieu

```
//distance à une autre position
double Position::distance(Position position) {
  double dist ;
  dist = sqrt((x-position.x)*(x-position.x)+(y-position.y)*(y-position.y)+(z-position.z)*(z-position.z));
  return dist;
}

//milieu entre la position et une autre position
Position Position::milieu(Position position) {
     Position m;
     m.set_x((x+position.x)/2.);
     m.set_y((y+position.y)/2.);
     m.set_z((z+position.z)/2.);
     return m;
}
```

```
//distance à une autre position
double Position::distance(Position position) {
  return sqrt((x-position.x)*(x-position.x)+(y-position.y)*(y-position.y)+(z-position.z)*(z-position.z));
}

//milieu entre la position et une autre position
Position Position::milieu(Position position) {
     Position m((x+position.x)/2., (y+position.y)/2., (z+position.z)/2.);
     return m;
}
```

constructeur de Position

```
//distance à une autre position
double Position::distance(Position position) {
  return sqrt(pow(x-position.x, 2)+pow(y-position.y, 2)+pow(z-position.z, 2));
}

//milieu entre la position et une autre position
Position Position::milieu(Position position) {
     return Position((x+position.x)/2., (y+position.y)/2., (z+position.z)/2.);
}
```

Destructeur - durée de vie des objets

Déclaration

```
class Position {
  private:
    double x;
    double y;
    double z;

  public:
    Position();
    Position(double x, double y, double z);
    ~Position();


    double distance(Position position);
    Position milieu(Position position);

    double get_x(){ return x;};
    void set_x(double x);
    double get_y(){ return y;};
    void set_y(double y);
    double get_z(){ return z;};
    void set_z(double z);
};
```

Implémentation

```
…
//-------------------------------------------------------
//constructeur par défaut
 Position::Position() {
      x = y = z = 0.0;
      printf("%x default says hello!\n", (unsigned)this);
}

//constructeur
 Position::Position(double x, double y, double z) {
       set_x(x);
       set_y(y);
       set_z(z);
      printf("%x says hello!\n", (unsigned)this);
}

// destructeur
Position::~Position(){
      printf("%x says bye, bye!\n", (unsigned)this);
}
…
```

```
int main(){
Position p1;
Position p2(50, 50, 0);
Position p3;

p3 = p1.milieu(p2);

return 0;
}
```

```
df8410f0 default says hello!
df841110 says hello!
df841130 default says hello!

df841170 default says hello!
df841170 says bye, bye!
df841150 says bye, bye!

df841130 says bye, bye!
df841110 says bye, bye!
df8410f0 says bye, bye!
```

p1
p2
p3

m

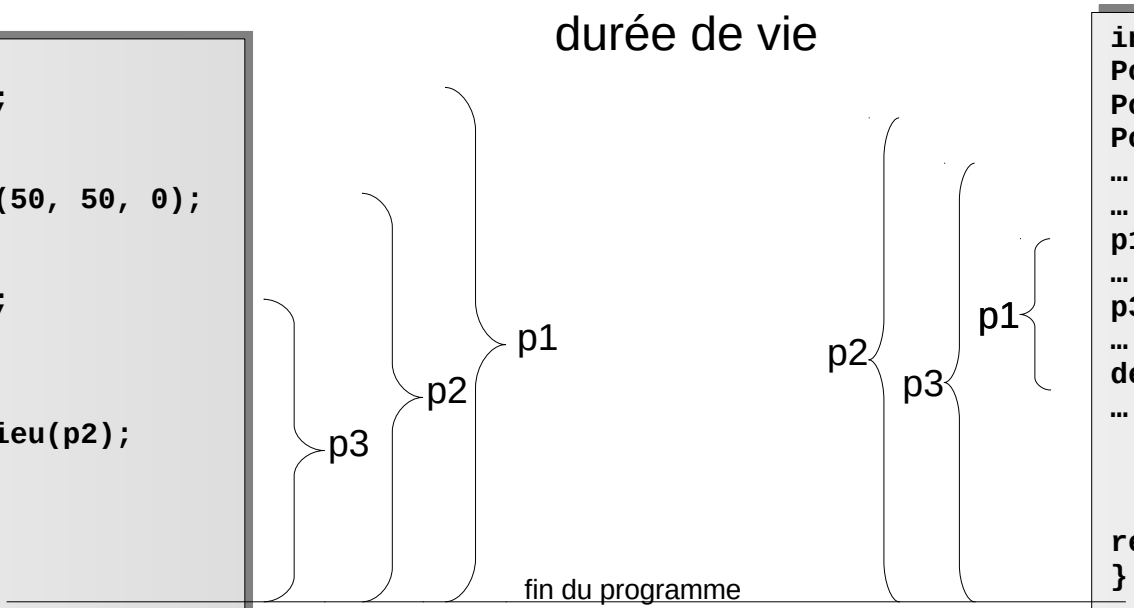
m
?

p3
p2
p1

```
Position Position::milieu(Position position) {
      Position m;
      m.set_x((x+position.x)/2.);
      m.set_y((y+position.y)/2.);
      m.set_z((y+position.z)/2.);
      return m;
}
```

Instanciation statique vs. dynamique

```
int main(){
Position p1;
…

Position p2(50, 50, 0);
…

Position p3;
…
…

p3 = p1.milieu(p2);
…
…
…
return 0;
}
```




```
int main(){
Position *p1;
Position p2(50, 50, 0);
Position p3;
…
…
p1 = new Position(20, 20, 10) ;
…
p3 = p1->milieu(p2);
…
delete p1 ;
…



return 0;
}
```

p1, p2, p3
objets statiques

p2, p3 objets
statiques,
p1 objet dynamique

- un objet dynamique est accessible au moyen d'un pointeur
- l'opérateur **`new`** instancie l'objet dynamique - réservation de la mémoire nécessaire
- l'opérateur **`delete`** détruit l'objet dynamique - restitution de la mémoire allouée

Constructeur, Destructeur, this, durée de vie des objets, allocation dynamique

- **Un constructeur est une méthode qui a le même nom que la classe**
- **A chaque fois qu'un objet est créé, il appelle automatiquement son constructeur - s'il y en a plusieurs dans la classe, il appelle "celui qui va bien" - voir le prototype**

- **Un destructeur est une méthode qui a le même nom que la classe avec un caractère ~ devant**
- **A chaque fois qu'un objet est détruit, il appelle automatiquement son destructeur**

- **Ceci s'applique à tous les objets, y compris à ceux créés localement dans les méthodes.**

- **Le pointeur *this* contient l'adresse mémoire de l'objet actif**

- **L'allocation dynamique (`new`/`delete`) permet de maîtriser la durée de vie des objets**

Convention typographiques

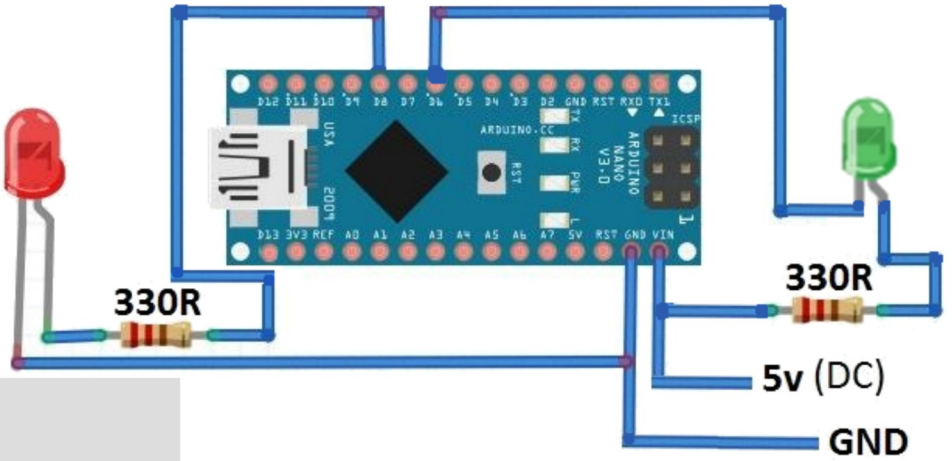
```
// Point header file
#define NUM_POINTS 100


class Point{
private:
   int x;
   int y;

public:
   void affiche(void);
   Point(int x, int y);
   ~Point();

};


int main( void){
Point p1, p2;
```

Exception à la règle :
Les constructeurs/destructeurs : fonctions ayant le même nom que la classe

Les classes, et elles seules, commencent par une majuscule
Les attributs commencent par une minuscule
Les méthodes commencent par une minuscule
Les objets (et toutes les variables) commencent par un minuscule
Seules les expressions constantes (`#define`) sont entièrement en majuscule

Programmation Orientée Objet
aussi, pour écrire des applications c++ dans l'environnement Arduino...

Allumer une LED sous Arduino




```
#define GREEN_PIN 8

void setup() {
  // initialize digital pin GREEN_PIN as an output.
  pinMode(GREEN_PIN, OUTPUT);
  digitalWrite(GREEN_PIN, HIGH) ;        // light OFF
  delay(1000) ;                          // wait for a second
  digitalWrite(GREEN_PIN, LOW) ;         // light ON
  delay(1000) ;                          // wait for a second
  digitalWrite(GREEN_PIN, HIGH) ;        // light OFF
}

void loop() {}
```

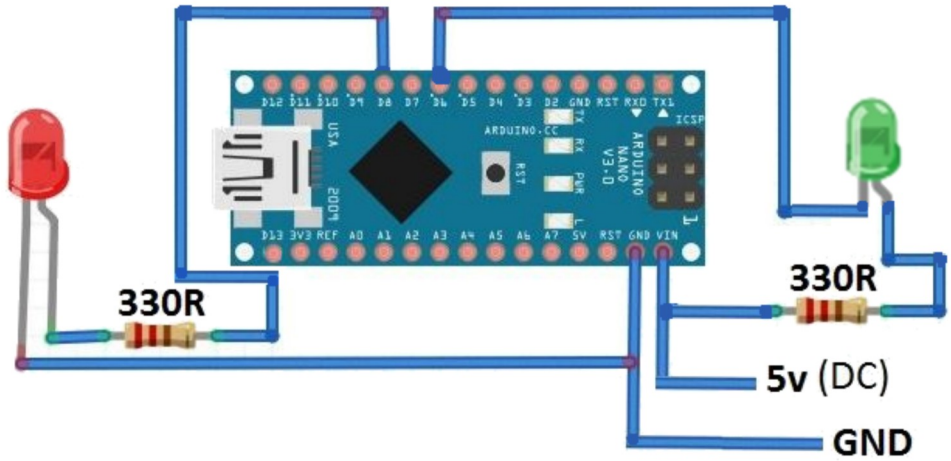
Mode allumée sur niveau bas
«zero logic »

Mode allumée sur niveau haut
«one logic »

```
#define RED_PIN 6

void setup() {
  // initialize digital pin RED_PIN as an output.
  pinMode(RED_PIN, OUTPUT);
  digitalWrite(RED_PIN, LOW) ;           // light OFF
  delay(1000) ;                          // wait for a second
  digitalWrite(RED_PIN, HIGH) ;          // light ON
  delay(1000) ;                          // wait for a second
  digitalWrite(RED_PIN, HIGH) ;          // light ON
}

void loop() {}
```

Piloter une LED

Avec une **période**
donnée de 2 secondes



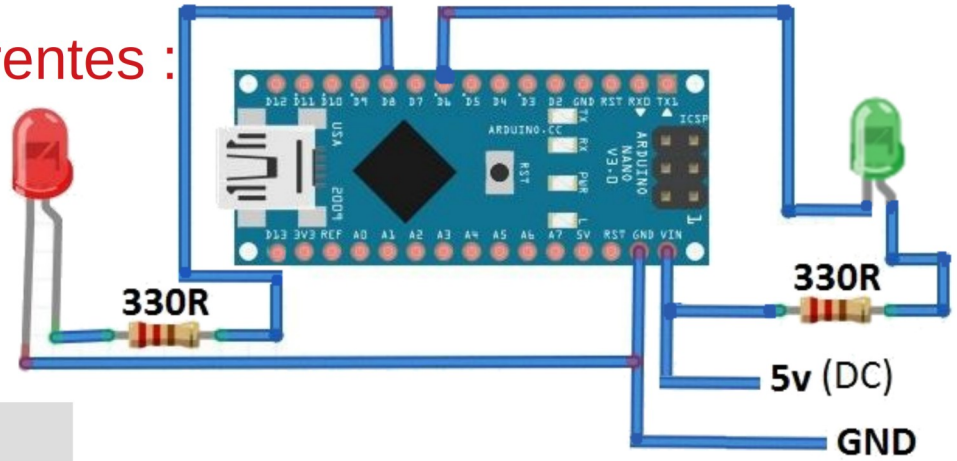

```
#define GREEN_PIN 8

bool state;
int duration = 1000;
unsigned long lastMillis=0;

void setup() {
  // initialize digital pin GREEN_PIN as an output.
  pinMode(GREEN_PIN, OUTPUT);
  digitalWrite(GREEN_PIN, HIGH) ;       // light OFF
  state = false;
}

void loop() {
unsigned long t = milis();
// manage green LED
if (t-lastMillis >= duration){
               digitalWrite(GREEN_PIN, state);
               state = !state;
               lastMillis = t;
     }
}
```

```
#define RED_PIN 6

bool state;
int duration = 1000;
unsigned long lastMillis=0;

void setup() {
  // initialize digital pin GREEN_PIN as an output.
  pinMode(GREEN_PIN, OUTPUT);
  digitalWrite(GREEN_PIN, HIGH) ;       // light OFF
  state = false;
}

void loop() {
unsigned long t = milis();
// manage green LED
if (t-lastMillis >= duration){
               digitalWrite(GREEN_PIN, !state);
               state = !state;
               lastMillis = t;
     }
}
```

Piloter deux LED sous Arduino

Clignotement avec **périodes** différentes :

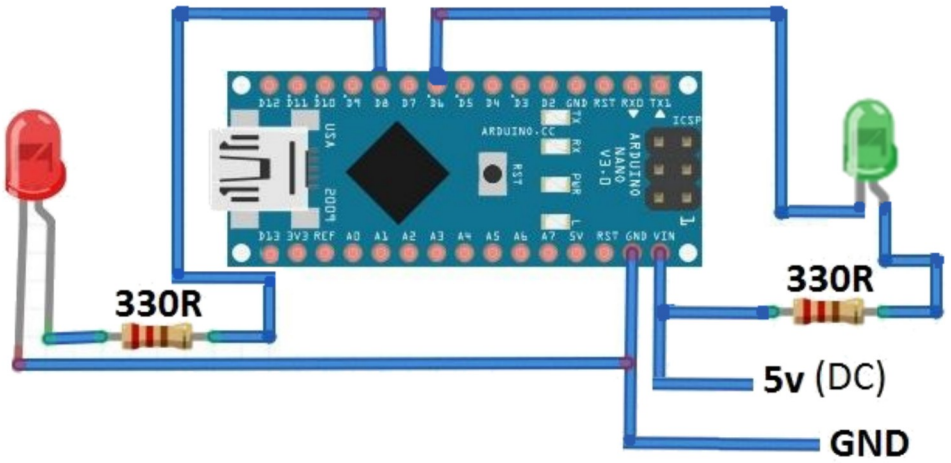
- Rouge, 2 secondes
- Verte, 1 seconde




```
#define GREEN_PIN 8
#define RED_PIN 6

bool redState, greenState;
int redDuration = 1000;
int greenDuration = 500;
unsigned long redLastMillis=0;
unsigned long greenLastMillis=0;

void setup() {
  // initialize digital pin GREEN_PIN as an output.
  pinMode(GREEN_PIN, OUTPUT);
  digitalWrite(GREEN_PIN, HIGH) ;       // light OFF
  greenState = false;

  // initialize digital pin RED_PIN as an output.
  pinMode(RED_PIN, OUTPUT);
  digitalWrite(RED_PIN, LOW) ;          // light OFF
  redState = false;
}
```

```
void loop() {
unsigned long t = milis();
// manage green LED
if (t-greenLastMillis >= greenDuration){
                digitalWrite(GREEN_PIN, greenState);
                greenState = !greenState;
                greenLastMillis = t;
     }

// manage red LED
if (t-redLastMillis >= redDuration){
                digitalWrite(RED_PIN, !redState);
                redState = !redState;
                redLastMillis = t;
     }
}
```

Une classe LED sous Arduino



Caractéristique d'une LED:

- l'état de la LED (*true*=allumée et *false*=éteinte)
- un entier représentant le numéro de la broche Arduino (dépend du type de carte)
- un booléen représentant le type de connexion électrique

Comportement d'une LED:

- *lightOn*, qui allume la LED
- *lightOff*, qui l'éteint
- *invert*, qui inverse son état
- *flash*, qui émet un flash lumineux d'une durée spécifiée en argument (millisecondes)

Led
- state : bool - pin : int - oneLogic : bool
+ get_pin() : int + get_state() : bool + set_pin(in value : int) : void + set_oneLogic(in value : bool) : void + Led() + Led(in pin : int, in oneLogic : bool = true) + lightOn() : void + lightOff() : void + invert() : void + flash(in ms : int = 100) : void - validPin(in pin : int) : bool

Quatre accesseurs (getter/setter) :
get_pin, et *set_pin* pour obtenir et modifier le numéro de la broche
get_state pour obtenir l'état de la LED.
set_oneLogic pour modifier le mode de branchement électrique

Deux constructeurs :
un par défaut mettra la valeur -1 dans le numéro de broche (broche indéfinie), *true* dans le mode de commande (logique "1") et *false* dans l'état (LED éteinte).
un constructeur permettant de préciser le numéro de broche et le mode de commande (logique "1" par défaut)

Une méthode privée validPin qui vérifie que la pin choisie est acceptable.

Piloter une LED sous Arduino

Déclaration
Led.h

```
#ifndef _LED_H
#define _LED_H
//La classe Led représente une led connectée sur une pin d'une carte arduino
class Led {
  private:
    //L'attribut state représente l'état de la led (true=allumée, false=éteinte)
    bool state;
    //l'attribut pin représente le numéro de la pin Arduino sur laquelle la led est branchée
    int pin;
    //L'attribut oneLogic décrit le mode de cablage (allumage par un niveau haut ou par un niveau bas)
    //oneLogic = true si allumage niveau haut
    bool oneLogic;
  public:
    inline const int get_pin() const;
    inline const bool get_state() const;
    void set_pin(int value);
    void set_oneLogic(bool value);
    //constructeur par défaut: pin = -1 (non définie, state=false, oneLogic=true
    Led();
    //constructeur:  pin =numéro de la pin Arduino, , oneLogic= type de logique de commande (high ou low)
    Led(int pin, bool oneLogic = true);
    //allume la led
    void lightOn();
    //Cette méthode éteint la led
    void lightOff();
    //Cette méthode inverse l'état de la led
    void invert();
    //Cette méthode allume la led pendant ms millisecondes puis l'éteint
    void flash(int ms = 100);
  private:
    //cette méthode vérifie si le n° de pin est valide (en fonction du type de carte: nano, uno, .etc.)
    bool validPin(int pin);
};
inline const int Led::get_pin() const {
  return pin;
}
inline const bool Led::get_state() const {
  return state;
}
#endif
```

Piloter une LED sous Arduino

Implémentation Led.cpp

```
#include "Led.h"
#include <Arduino.h>

#define ARDUINO_VALID_PIN pin>=2 && pin<=12      //carte nano

//cette méthode vérifie si le n° de pin est valide (en fonction du type de carte: nano, uno, .etc.)
// pour en changer, il faut modifier la ligne: #define ARDUINO_VALID_PIN ...
bool Led::validPin(int pin) {
    return ARDUINO_VALID_PIN;
}

void Led::set_pin(int value) {
  if (validPin(value)){         // si le n° de pin n'est pas valide, on ne fait rien
       pinMode(value, OUTPUT);
       pin = value;
  }
}

void Led::set_oneLogic(bool value) {
  oneLogic = value;
}

//constructeur par défaut: pin = -1 (non définie, state=false, oneLogic=true
Led::Led(){
  pin=-1;           // -1 = pin invalid a priori
  state=0;
  oneLogic=true;    // logique "1" par défaut
}

//constructeur:  pin =numéro de la pin Arduino, oneLogic= type de logique de commande (high ou low)
Led::Led(int pin, bool oneLogic){
  set_pin(pin);
  state=0;          // eteint par défaut
  this->oneLogic=oneLogic;
}
```

Piloter une LED sous Arduino

Implémentation Led.cpp

```
//allume la led
void Led::lightOn() {
  if (pin==-1) return;      // pin invalide, on ne fait rien
  digitalWrite(pin, (oneLogic ? HIGH : LOW));
  state=true;
}

//Cette méthode éteind la led
 void Led::lightOff() {
  if (pin==-1) return;      // pin invalide, on ne fait rien
  digitalWrite(pin, (oneLogic ? LOW : HIGH));
  state=false;
}

//Cette méthode inverse l'état de la led
 void Led::invert() {
     if (state)lightOff(); else lightOn();
}

//Cette méthode allume la led pendant ms millisecondes puis l'éteint
 void Led::flash(int ms) {
     if (state) return;     // déjà allumé
     if (ms<10) ms = 10 ;   // 10 ms minimum
     lightOn();
     delay(ms);
     lightOff();
}
```

Piloter une LED sous Arduino

Allumage/extinction d'une LED sur la pin 2 en logique "1"

programme test.ino

```
// la broche de la LED
#define    LED_PIN  2
#define    DELAY_MS    100
#include "Led.h"

Led led;       // l'objet led

void setup(){
    // Initialise la bibliothèque Arduino
    init();

    Serial.begin(115200);
    Serial.println("Programme de test !");

    led.set_pin(LED_PIN);
}

void loop(){
    led.flash() ;
    delay(DELAY_MS) ;
    }
```

Quel est l'intérêt de tout ça ?

Allumage/extinction de 3 LED sur pin 2, 4 et 5 en logique "1", "0" et "1", durées : 500, 800 et 300 ms

programme test3.ino

```
// trois LED
#include "Led.h"
#define    DELAY_MS    100

Led leds[3];        // tableau de 3 led
int ledPins[3] = {2, 4, 5} ;
int delays[3] = { 500, 800, 300} ;

void setup(){
    // Initialise la bibliothèque Arduino
    init();

    Serial.begin(115200);
    Serial.println("Programme de test !");

     for (int i=0 ; i<3 ; i++){
       leds[i].set_pin(ledPins[i]);
       }
    // la deuxième est en logique zéro
    leds[1].set_oneLogic(false) ;
}

void loop(){
     for (int i=0 ; i<3 ; i++){
       leds[i].flash(delays[i]) ;
       delay(DELAY_MS) ;
       }
    }
```

Arguments par défaut dans une méthode

```
#ifndef _LED_H
#define _LED_H
//La classe Led représente une led connectée sur une pin d'une carte arduino
class Led {
…
    Led(int pin, bool oneLogic = true);
…
    //Cette méthode allume la led pendant ms millisecondes puis l'éteint
    void flash(int ms = 100);
…
};
#endif
```

```
void Led::flash(int ms) {
     if (state) return;      // déjà allumé
     if (ms<10) ms = 10 ;    // 10 ms minimum
     lightOn();
     delay(ms);
     lightOff();
}
```

programme test.ino

```
// la broche de la LED
#define    LED_PIN  2
#define    DELAY_MS    100
#include "Led.h"

Led led(LED_PIN);          // l'objet led

void setup(){
    // Initialise la bibliothèque Arduino
    init();

    Serial.begin(115200);
    Serial.println("Programme de test !");

}

void loop(){
    led.flash() ;
    delay(DELAY_MS) ;
    led.flash(1000) ;
    delay(DELAY_MS) ;
    led.flash(5) ;
    delay(DELAY_MS) ;
    }
```

`oneLogic = true`

`ms = 100`

`ms = 1000`

`ms = 5`

Si l'argument n'est pas spécifié, il prend la valeur par défaut

Méthodes inline

```
#ifndef _LED_H
#define _LED_H
//La classe Led représente une led connectée sur une pin d'une carte arduino
class Led {
…
    inline int get_pin(){
                    return pin;
                    };

    inline bool get_state();

…
};


inline bool Led::get_state(){
  return state;
}

#endif
```

directement dans la déclaration

séparément avec le mot clef *inline*

Tous les objets d'une même classe partagent les mêmes méthodes - une seule fois le code quelque soit le nombre d'objets instanciés ; ce processus est un peu lent.

Les méthodes *inline* sont dupliquées pour chaque objet, ce qui permet un accès plus rapide.

En général, les accesseurs, qui sont des "petites" méthodes souvent utilisées, sont déclarés *inline*.

Mot clef const

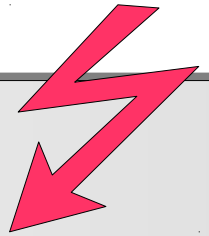
Sur un attribut:

```
const int cst=96 ;
const float pi=3.14159 ;

const int largeurMax = 640;          // Entier constant
const int sizeMax[] = {640, 480};    // Tableau constant
const Point origine(0, 0);           // Objet constant de classe Point
```

Sur la valeur retournée d'une fonction:

```
char *fonc()
    { return “Bonjour”;}
```

```
int main(){
char *resu ;
resu = fonc() ;
resu[1]=’a’ ;
...
```

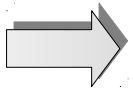


plantage

```
const char *fonc()
    { return "Bonjour";}
```

Sur un paramètre d'une fonction:

```
int calcul(int i){
    int j ;
    j = 2 * i ;
    i = 8 ;
    j = 2 * i;
    return j ;
}
```



```
int calcul(const int i){
    int j ;
    j = 2 * i ;
    i = 8 ;
    j = 2 * i;
    return j ;
}
```

Impossible
(compilation)

Mot clef const

Sur une méthode:

```
class Point {

private:
  double x;
  double y;

public:
  Point(double a, double b) ;

  // Renvoie la valeur des attributs
  double getX() const;
  double getY() const;

  // Assigne une valeur aux attributs : l'objet est modifié
  void setX(double absisse);
  void setY(double ordonnee);
...
```

l'objet courant n'est pas modifiable dans la méthode

```
// Methode constante qui renvoie l'abscisse du point
double Point::getX() const {
  return x ;
}

// Methode constante qui renvoie l'ordonnee du point
double Point::getY() const {
  return y;
}
```

Utilité des arguments par défaut, inline, const, *.etc.*

Lors de modélisation du programme (faite avant de lâcher "les programmeurs fous"), le concepteur peut anticiper la majorité des problèmes risquant d'induire des anomalies de fonctionnement (*bug*).

inline permet de créer des fonctions plus rapides

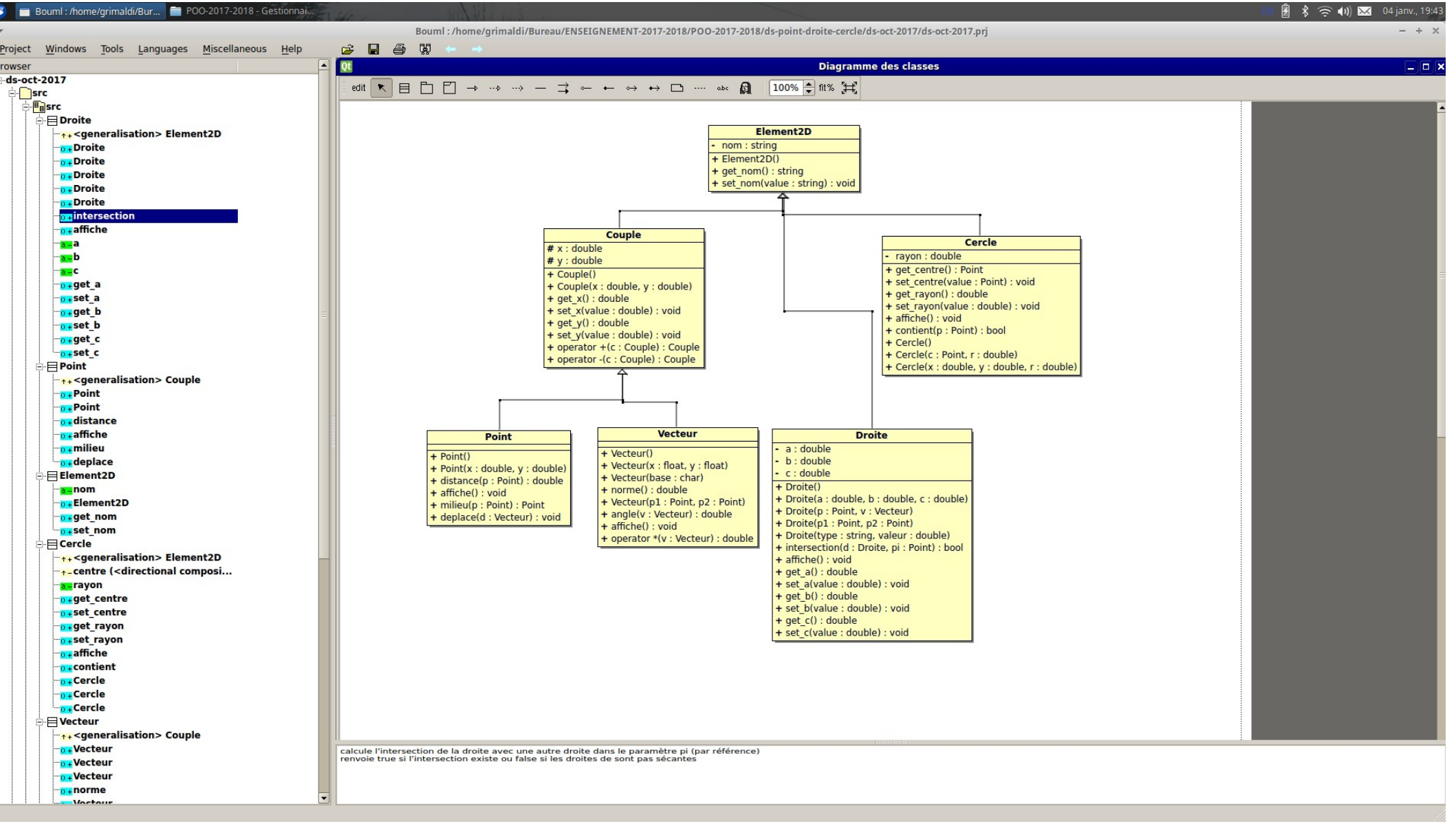
const sur un argument d'une méthode, interdira au programmeur de changer la valeur de cet argument dans la méthode.

const sur méthode, interdira au programmeur de modifier les attributs dans cette méthode

les arguments par défaut, permettra aux utilisateurs de la méthode de ne spécifier qu'une partie des arguments, en gardant pour les autres des valeurs "qui vont bien"

Les programmeurs seront ainsi contraint de respecter le cahier des charges, au risque d'avoir des impossibilités de compilation, *"fatal error"*.

Modélisation, Analyse/Conception



http://www.bouml.fr

analyse/conception orientée objet en UML

client

€ acompte

développeurs

accord client

choix du langage de programmation (c++, java, .*etc*.)

identification des besoins

analyse fonctionnelle

génération automatique du code source

classes, attributs, méthodes, types, relations, critères de validation, .*etc*.

conception UML

implémentation des méthodes

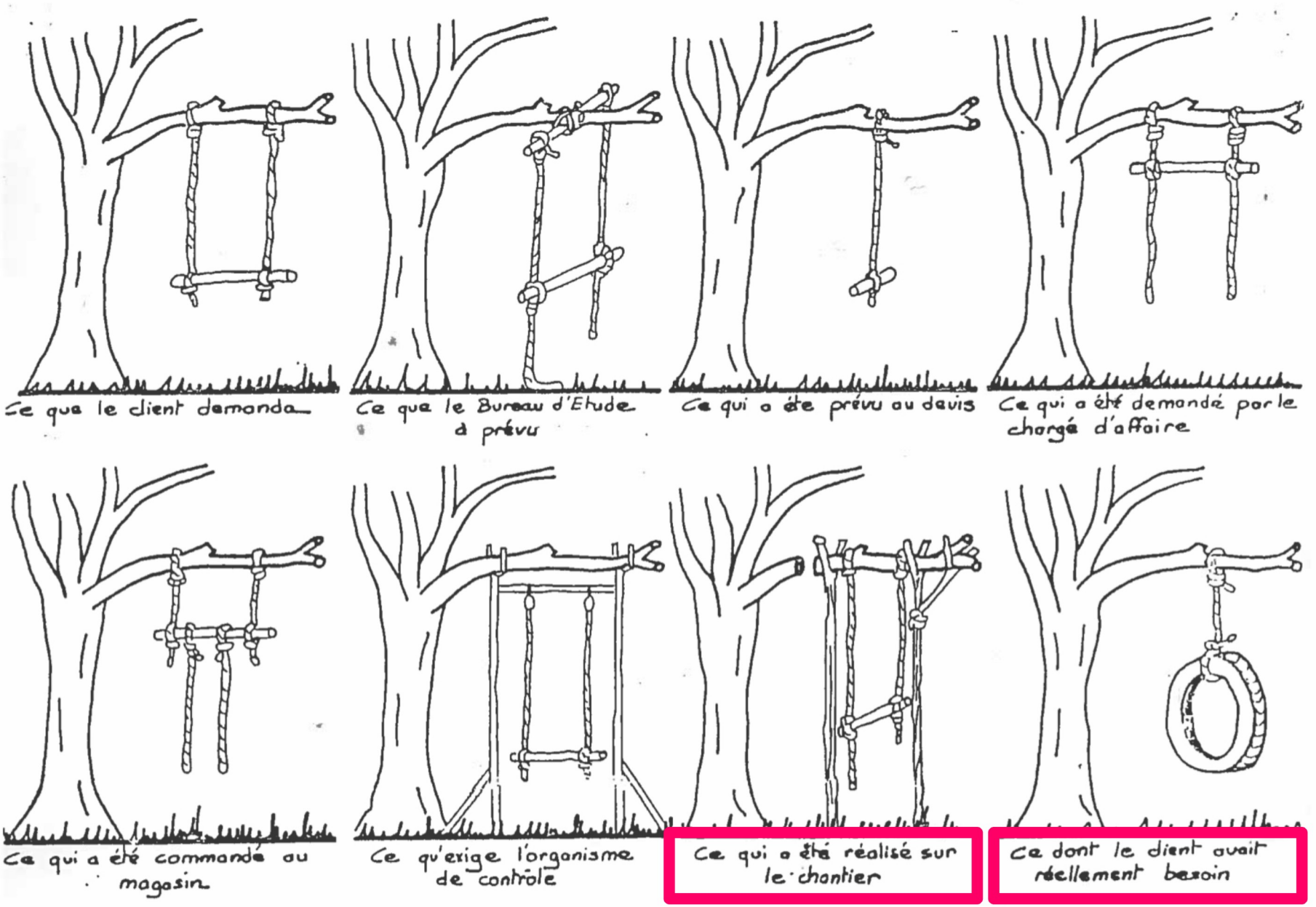
tests

validation

€

paiement

pour éviter ceci ...



Autre exemple dans l'environnement Arduino.

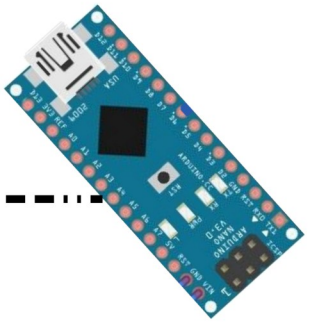
"Feu rouge" basé sur 3 LED

On veut créer et commander un mini "feu rouge" basé sur trois LED, connectées directement à des ports d'une carte Arduino

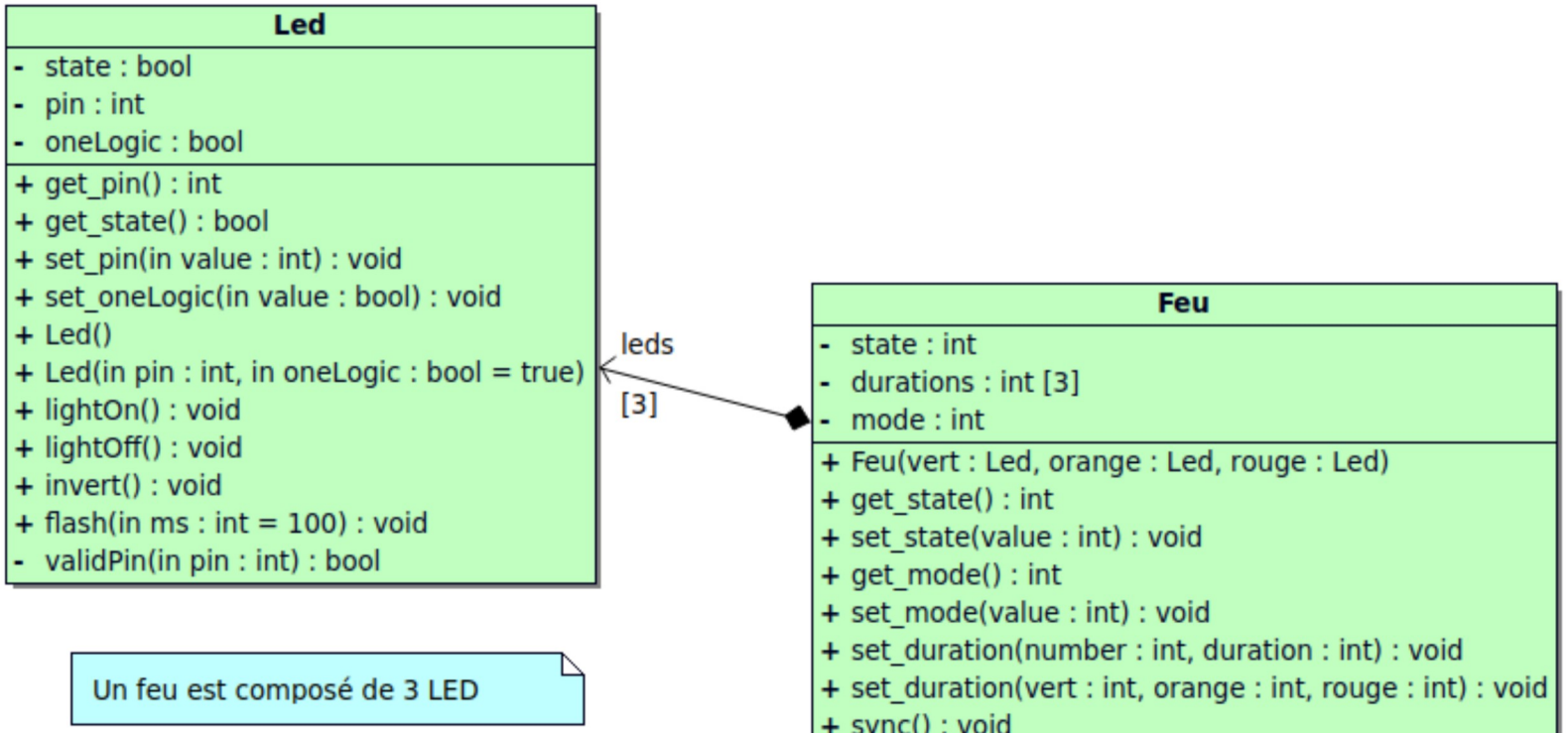
Fonctionnalités de base :

- 3 états : vert, orange et rouge
- 3 modes de fonctionnement (0=manuel, 1=automatique, 2=orange clignotant).
- 3 durées d'allumage réglables
- régler ses durées d'allumage (rouge, orange, vert)
- passer en mode automatique ou manuel ou clignotant.

LED verte
LED orange
LED rouge
GND



Modélisation d'un feu de signalisation basé sur 3 LED



Réutilisation du code effectué précédemment sur la LED

La classe Feu

Déclaration Feu.h

```
#ifndef _FEU_H
#define _FEU_H

#include "Led.h"
#define VERT 0
#define ORANGE 1
#define ROUGE 2
#define MODE_MANUEL 0
#define MODE_AUTO 1
#define MODE_ORANGE_CLI 2

//Cette classe représente un feu de signalisation basé sur trois leds directement connectées
//sur des ports d'une carte arduino
class Feu {
  private:
    //les trois leds du feu de signalisation: leds[0]=vert, leds[1]=orange, leds[2]=rouge
    Led leds[3];

    //cet attribut représente l'état du feu: 0=vert, 1=orange, 2=rouge
    int state;

    // cet attribut est un tableau qui contiendra les durées d'allumage des différentes leds
    // (0=vert, 1=orange, 2=rouge)
    int durations[3];

    //mode de fonctionnement du Feu (0=manuel, 1=automatique, 2=orange clignotant)
    int mode;

  public:
    // Constructeur permettant d'initialiser un Feu à partir de trois Led - state=0(vert),
    //  durations=1000 (ms pour les trois)
    Feu(Led vert, Led orange, Led rouge);
```

La classe Feu...

Déclaration Feu.h

```
    //getter: permet de récupérer l'état courant du Feu (0=vert, 1=orange, 2=rouge)
     inline int get_state() const;

    //setter: permet de modifier l'état courant du Feu(0=vert, 1=orange, 2=rouge)
     void set_state(int value);

     //getter: renvoir le mode de fonctionnement courant (0=manuel, 1=automatique, 2=orange
     // clignotant)
     inline int get_mode() const;

     // setter: permet de modifier le mode de fonctionnement courant
     // ( 0=manuel, 1=automatique, 2=orange clignotant)
     void set_mode(int value);

     //setter: permet de régler la durée d'une couleur
     void set_duration(int number, int duration);

     //setter: permet de régler la durée des trois couleurs vert, orange, rouge
     void set_duration(int vert, int orange, int rouge);

     // Cette méthode permet de synchroniser le feu avec la boucle principal. Pour que le
     // Feu fonctionne automatiquement, il faut l'appeler dans la boucle principale du
     // programme. La fonction sync n'est pas bloquante.
     void sync();

};
#endif
```

La classe Feu...

Implémentation Feu.cpp

```
#include "Feu.h"

//getter: permet de récupérer l'état courant du Feu (0=vert, 1=orange, 2=rouge)
inline int Feu::get_state() const {
  return state;
}

//getter: renvoie le mode de fonctionnement courant (0=manuel, 1=automatique,
//2=orange clignotant)
inline int Feu::get_mode() const {
  return mode;
}

//Constructeur permettant d'initialiser un Feu à partir de trois Led - state=0(vert),
//durations=1000(ms pour les trois)
Feu::Feu(Led vert, Led orange, Led rouge) {
    leds[0]=vert; leds[1]=orange ; leds[2]= rouge;
    state=0;        // vert
    mode=0;         // manuel
    for (int i=0 ; i<3 ; i++)durations[i]=1000;     // 1 seconde à tout le monde
}

//setter: permet de modifier l'état (0=vert, 1=orange, 2=rouge)
void Feu::set_state(int value) {
    if (value>=0 && value<=2) {     // un état valide
        state = value;
        for (int i=0 ; i<3 ; i++)       // on met les leds dans cet état
            if (i==state)leds[i].lightOn();
                else leds[i].lightOff();
    }
}
```

La classe Feu...

```
// setter: permet de modifier le mode de fonctionnement courant (0=manuel, 1=automatique,
// 2=orange clignotant)
void Feu::set_mode(int value) {
    if (value>=0 && value <=2)  mode = value;
    if (mode == MODE_ORANGE_CLI) set_state(ORANGE);
}

//setter: permet de régler la durée d'une couleur
void Feu::set_duration(int number, int duration){
    if (number>=0 && number<=2)durations[number] = duration;
}

//setter: permet de régler la durée des trois couleurs vert, orange, rouge
void Feu::set_duration(int vert, int orange, int rouge){
    durations[VERT]=vert;
    durations[ORANGE]=orange;
    durations[ROUGE]=rouge;
}
```

Implémentation Feu.cpp

Vous noterez qu'il est possible d'avoir plusieurs méthodes avec le même nom, comme ici **`set_duration`**, à condition qu'elles n'aient pas le même prototype :

```
set_duration(int, int) ;
set_duration(int, int, int) ;
```

La classe Feu

```
// Cette méthode permet de synchroniser le feu avec la boucle principale. Pour que le Feu
// fonctionne automatiquement, il faut l'appeler dans la boucle principale du programme.
// La fonction sync n'est pas bloquante.
void Feu::sync() {
    static long lastMillis=0;
    long now = millis();
    switch(mode){
    case 0:     // manuel on ne fait rien
    case 1:     // automatique on change d'état
        if (now-lastMillis>=durations[state]){      // c'est fini pour cet état
            set_state((state+1)%3);                 // on passe au suivant
            lastMillis = now;
        }
        break;
    case 2:     // orange clignotant
        if (now-lastMillis>=durations[1]){
                leds[1].invert();     // l'orange clignote
                lastMillis = now;
        }
        break;
    }
}
```

Implémentation Feu.cpp

Dans l'environnement Arduino IDE, le répertoire du projet doit contenir les fichiers :

- Led.h et Led.cpp - pour la classe Led
- Feu.h et Feu.cpp - pour la classe Feu
- main.ino - le programme principal

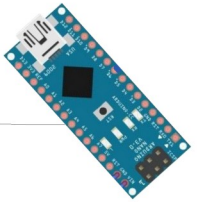
Exemple de programme

```
#include "Led.h"
#include "Feu.h"

Led ledR(2, false);      // pin2, zeroLogic
Led ledV(4, true);       // pin4, oneLogic
Led ledO(3, true);       // pin3, zeroLogic

Feu f(ledV, ledO, ledR); // un feu de signalisation basé sur les 3 LED

void setup(){
    // Initialise la bibliothèque Arduino
    init();
    Serial.begin(115200);

    f.set_duration(4000, 1000, 4000);   // durées des allumages
    f.set_mode(MODE_AUTO);              // mode automatique
}

void loop(){
    // lecture d'un caractère sur la liaison série
    char c = Serial.read();

    // changement du mode de fonctionnement
    switch (c){
    case 'M': f.set_mode(MODE_MANUEL); break;
    case 'A': f.set_mode(MODE_AUTO); break;
    case 'O': f.set_mode(MODE_ORANGE_CLI);break;
    }
 f.sync();   // mise à jour du feu
 }
```

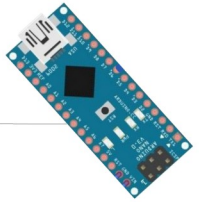
Programme main.ino

Généralisation/spécialisation

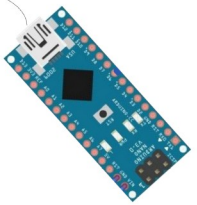
connexion directe
aux ports



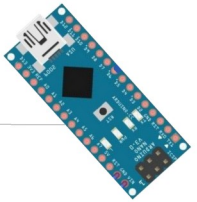
liaison analogique



led RGB



liaison I2C



Étudié précédemment !
Comment réutiliser le code ?

HÉRITAGE DE CLASSE

Généralisation :
classe mère, contenant tout ce qui est commun à tous les feux.

Spécialisation :
classes filles, contenant ***en plus***, tout ce qui est propre à chaque feu.

Héritage de classes

classe mère : **`Feu`** (généralisation)

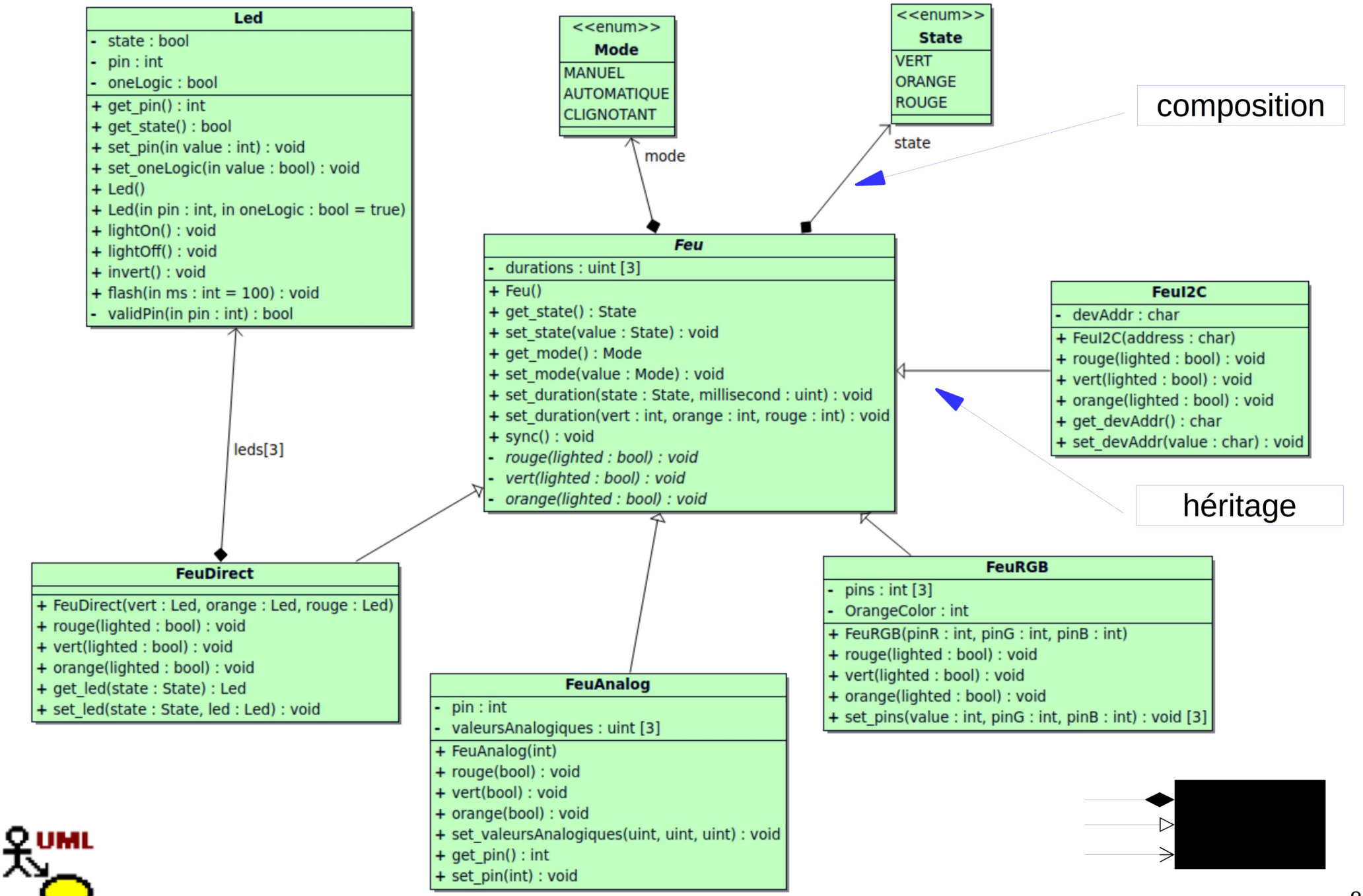
classes filles :

un **`FeuDirect`** est un **`Feu`** particulier, basé sur 3 LED, directement connectées au port (spécialisation)
un **`FeuI2C`** est un **`Feu`** particulier, connecté à l'I2C (spécialisation)
un **`FeuAnalog`** est un **`Feu`** particulier, commandé par une unique tension U analogique, ex : vert pour 0v<U<1v, orange pour 1.1v<U<2v, rouge pour U>2.1v, (spécialisation)
un **`FeuRGB`** est un **`Feu`** particulier, basé sur une LED RGB qui change de couleur (spécialisation)

Les classes **`FeuDirect, FeuI2C, FeuAnalog`**, et **`FeuRGB`**, sont héritées de la classe **`Feu.`**

On peut envisager un héritage entre deux classes quand la liaison verbale **"est un.e"** est possible entre-elles.

Diagramme des classes UML



Les types d'héritage

Le type public : `FeuDirect : public Feu`

- Les membres publiques de la classe mère restent publiques dans la classe fille.
- Les membres protégés de la classe mère restent protégés dans la classe fille.
- Les membres privés de la classe mère sont inaccessibles.

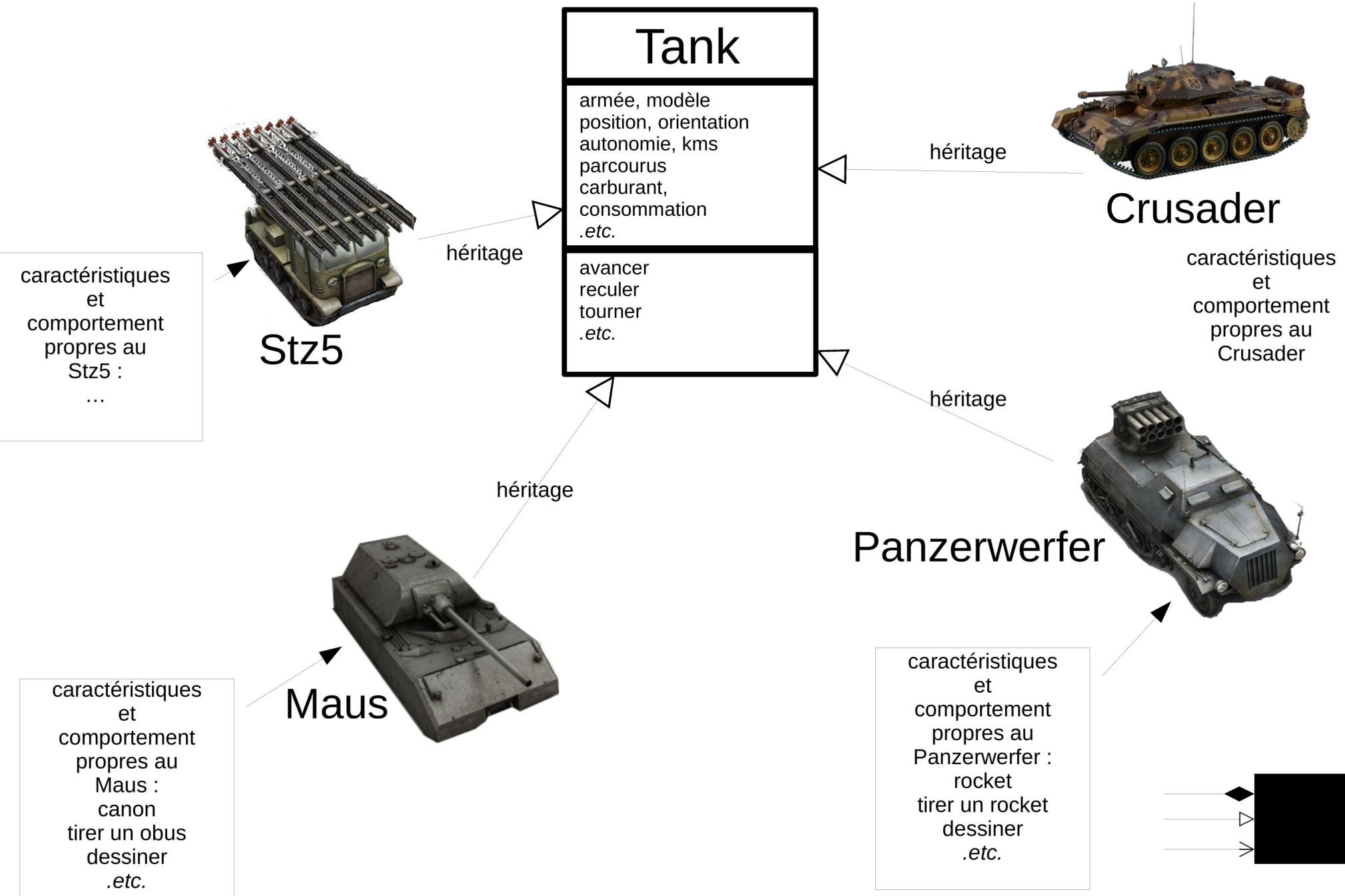
Le type protected : `FeuDirect : protected Feu`

- Les membres publiques de la classe mère deviennent protégés dans la classe fille.
- Les membres protégés de la classe mère restent protégés dans la classe fille.
- Les membres privés de la classe mère sont inaccessibles.

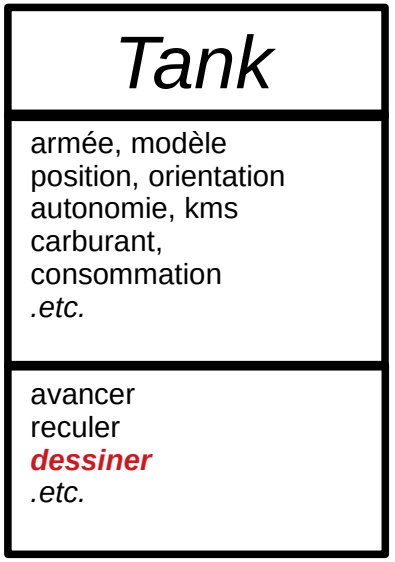
Le type private : `FeuDirect : private Feu`

- Les membres publiques de la classe mère deviennent privés dans la classe fille.
- Les membres protégés de la classe mère deviennent privés dans la classe fille.
- Les membres privés de la classe mère sont inaccessibles.

Est-ce applicable dans *World of Tanks* ?



Méthode/classe abstraite (*virtual*)



Si dans la classe mère Tank on définie une ou plusieurs méthodes **abstraites** (mot clef **`virtual`** en c++, *italique* en UML), la classe Tank devient une classe **abstraite** (virtuelle).

- Elle ne peut pas être instanciée (on ne peut pas créer d'objets Tank)
- Elle ne peut être qu'héritée
- Les classes filles de Tank, devront **obligatoirement** définir totalement (déclarer et implémenter) les méthodes abstraites, sous peine d'être à leur tour **abstraites** avec ce qui en découle (deux points précédents)

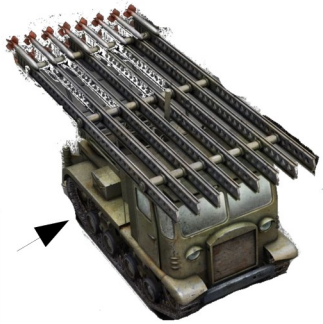
Une classe fille peut être à son tour classe mère d'une autre classe autant de fois qu'on le désire - Hiérarchie de classes

Méthode/classe abstraite (*virtual*)

Tank

armée, modèle
position, orientation
autonomie, kms
carburant,
consommation
.etc.

avancer
reculer
dessiner
.etc.



Stz5

caractéristiques
et
comportement
propres au
Stz5 :
…
dessiner

Crusader

caractéristiques
et
comportement
propres au
Crusader
dessiner

Maus

caractéristiques
et
comportement
propres au
Maus :
canon
tirer un obus
dessiner
.etc.

Panzerwerfer

caractéristiques
et
comportement
propres au
Panzerwerfer :
rocket
tirer un rocket
dessiner
.etc.

la méthode **`dessiner`** doit être définie dans chaque classe fille

Concept abstrait → classe abstraite

Tank est un **concept abstrait**, représenté donc par une **classe abstraite**, car sa méthode ***`dessiner`*** est **abstraite.**

Dessine un Tank

Dessine un Stz5

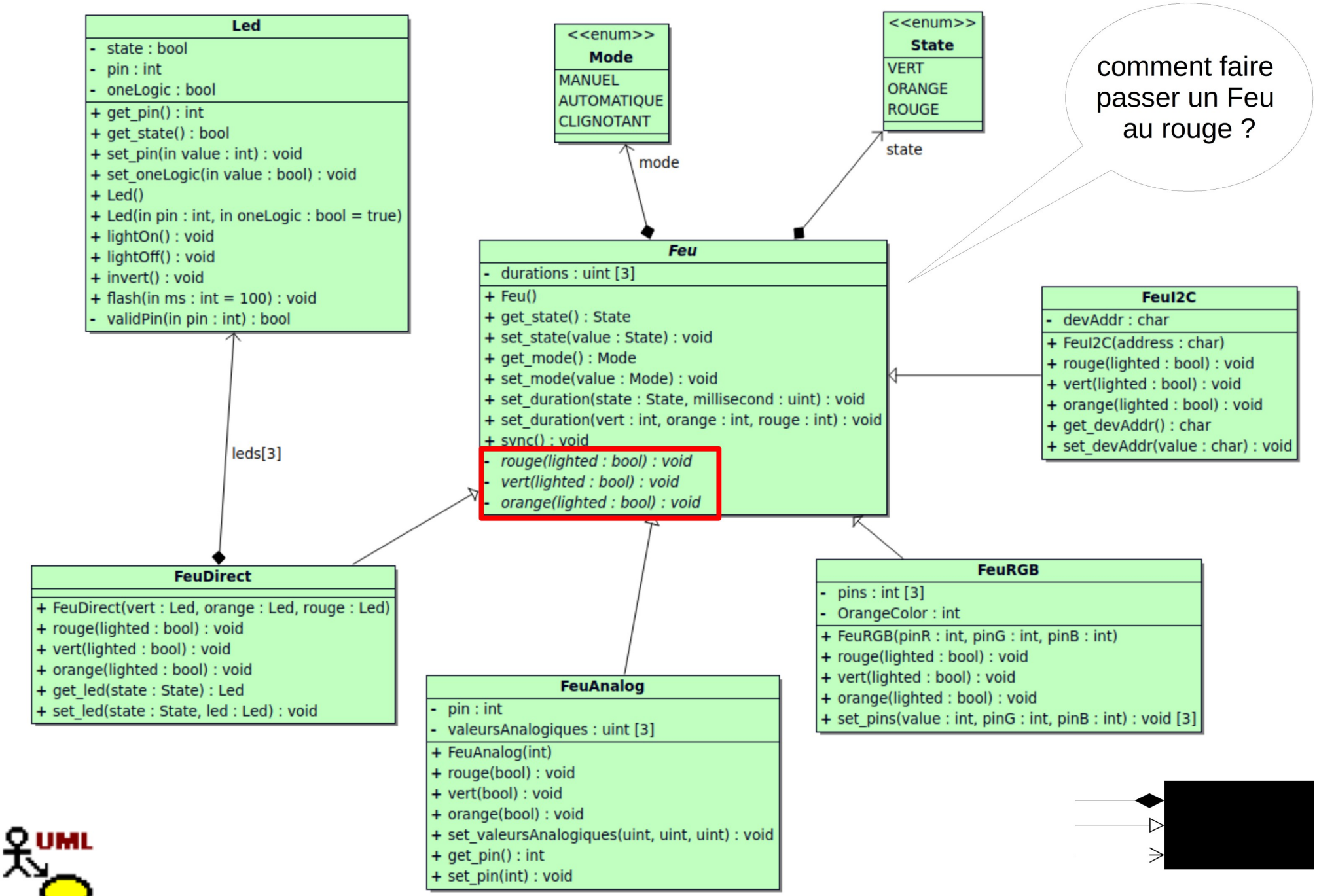
Dessine un Maus

Dessine un Panzerwerfer

Dessine un Crusader

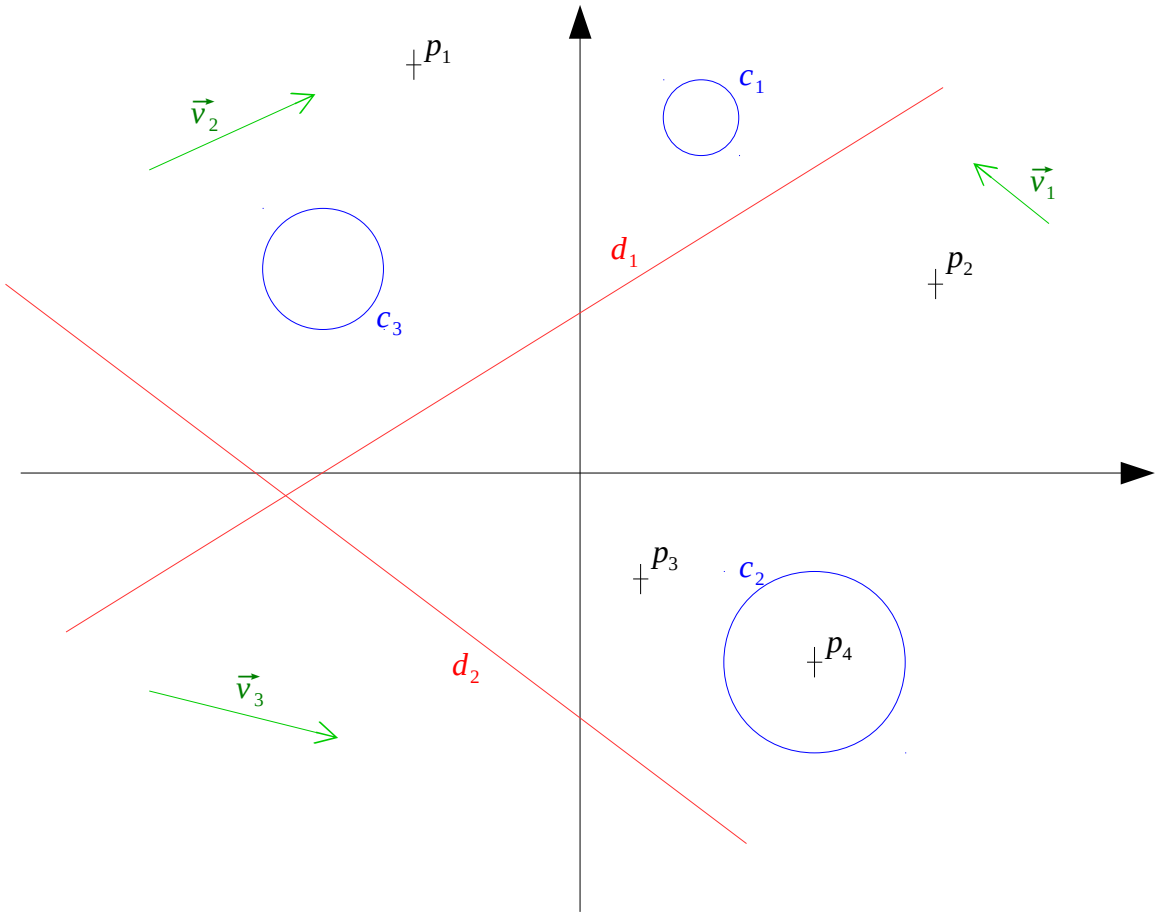
- La méthode *`dessine`* de la classe *Tank* est bien **abstraite** :
- On ne peut pas dessiner un *Tank* en tant que tel !
- *Tank* est bien un **concept abstrait**

Feu est aussi une classe abstraite !



Autre exemple utilisant l'héritage

points, droites, cercles, vecteurs, ...



Modélisation d'éléments géométriques du plan

élément géométrique du plan, nommé

couple du plan (x, y)

droite du plan $ax+by+c=0$

Element2D
- nom : string
+ Element2D()
+ get_nom() : string
+ set_nom(value : string) : void

Droite
- a : double
- b : double
- c : double
+ Droite()
+ Droite(a : double, b : double, c : double)
+ Droite(p : Point, v : Vecteur)
+ Droite(p1 : Point, p2 : Point)
+ Droite(type : string, valeur : double)
+ intersection(d : Droite, pi : Point) : bool
+ affiche() : void
+ get_a() : double
+ set_a(value : double) : void
+ get_b() : double
+ set_b(value : double) : void
+ get_c() : double
+ set_c(value : double) : void

Couple
x : double
y : double
+ Couple()
+ Couple(x : double, y : double)
+ get_x() : double
+ set_x(value : double) : void
+ get_y() : double
+ set_y(value : double) : void
+ operator +(c : Couple) : Couple
+ operator -(c : Couple) : Couple

Cercle
- rayon : double
+ get_centre() : Point
+ set_centre(value : Point) : void
+ get_rayon() : double
+ set_rayon(value : double) : void
+ affiche() : void
+ contient(p : Point) : bool
+ Cercle()
+ Cercle(c : Point, r : double)
+ Cercle(x : double, y : double, r : double)

centre

Point
+ Point()
+ Point(x : double, y : double)
+ distance(p : Point) : double
+ affiche() : void
+ milieu(p : Point) : Point
+ deplace(d : Vecteur) : void

Vecteur
+ Vecteur()
+ Vecteur(x : float, y : float)
+ Vecteur(base : char)
+ norme() : double
+ Vecteur(p1 : Point, p2 : Point)
+ angle(v : Vecteur) : double
+ affiche() : void
+ operator *(v : Vecteur) : double

vecteur du plan $\vec{v}=\begin{pmatrix} x \\ y \end{pmatrix}$

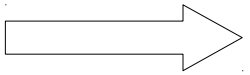
cercle du plan

point du plan $p=(x, y)$

Surcharge des opérateurs

Tous les opérateurs du langage (*cf. à droite)* sont définis pour les types prédéfinis (**`int, float, double, char, .etc`**).

En langage orienté objet (c++), on peut les redéfinir pour des classes que nous avons créées nous même.



Cela ne change rien pour les types prédéfinis ! (ex : l'addition de deux entiers reste inchangée)

operator =
operator ==
operator !=
operator <
operator >
operator <=
operator >=
operator +
operator -
operator *
operator /
operator %
operator +=
operator -=
operator *=
operator /=
operator %=
operator !
operator &&
operator ||
operator ++
operator --
operator &
operator |
operator ^
operator ~
operator &=
operator |=
operator ^=
operator <<
operator >>
operator <<=
operator >>=
operator ->
operator ()
operator []
operator new
operator delete
operator bool
operator int

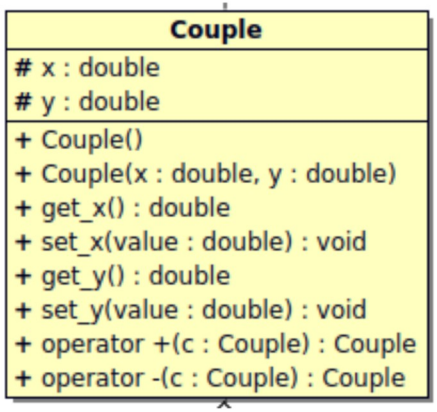
exemple:

	opérateur
opérateur d'addition de la classe Couple (somme des composantes)	+
opérateur de soustraction de la classe Couple (différence des composantes)	-
opérateur multiplication de la classe Vecteur (produit scalaire)	*

mais, on pourrait aussi surcharger

les opérateurs relationnels de la classe Cercle (comparaison des rayons)	>
	<
	==
l'opérateur d'addition de la classe Cercle (un cercle de centre le milieu des deux centres et de rayon la somme des deux rayons)	+

Surcharge des opérateurs de Couple



notez les attributs protégés

héritage

```
: public Element2D
```

2 opérateurs surchargés

```
Couple operator +(Couple c);
Couple operator -(Couple c);
```

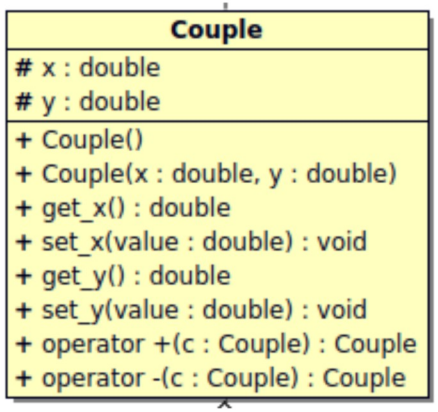
Pourquoi ces prototypes ?

```
c3 = c1.operator +(c2)
```

```
Couple operator +(Couple c)
```

L'utilisation d'un opérateur revient à utiliser une méthode de la classe.

Surcharge des opérateurs de Couple



appel du constructeur par défaut de la classe mère

2 opérateurs surchargés

```
Couple Couple::operator +(Couple c) {
    Couple resu(x+c.x, y+c.y);
    return resu;
}

Couple Couple::operator -(Couple c) {
    Couple resu(x-c.x, y-c.y);
    return resu;
}
```

surcharge des opérateurs de Cercle

```
bool Cercle::operator >(Cercle c) {
return rayon>c.rayon ;
}

bool Cercle::operator <(Cercle c) {
return rayon<c.rayon;
}

bool Cercle::operator ==(Cercle c) {
return rayon==c.rayon;
}

Cercle Cercle::operator +(Cercle c) {
Cercle value ;
value.set_centre(centre.milieu(c.centre));
value.set_rayon(rayon + c.rayon);
return value;
}
```

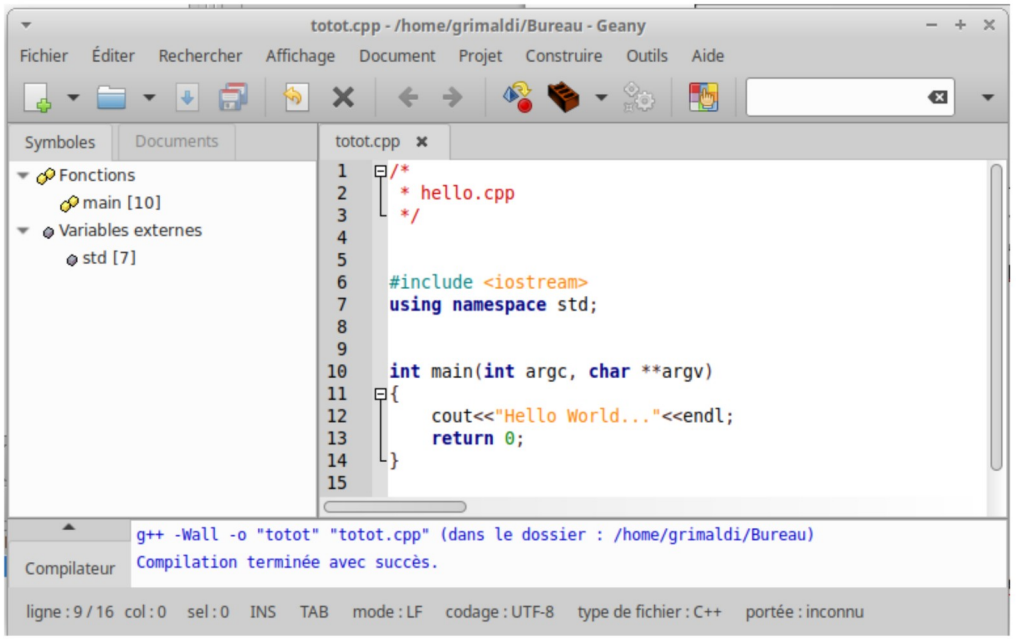
opérateur
>
<
==
+

```
c2 + c3    ~   c2.operator +(c3)
```

notez que le l'opérateur + de la classe Point est héritée de sa classe mère Couple (un Point *est un* Couple)

```
// programme principal
int main(void){
Point p1, p2(100, 20), p3(30, 157);
Cercle c1, c2(p2, 100), c3( p2+p3, 20, 5);

p1 = p2 + p3 ;                  // ~ p2.operator +(p3)
c1 = c2 + c3 ;                  // ~ c2.operator +(c3)
if (c1>c2) {…}                  // ~ c1.operator >(c2)
    else if (c1==c2){…}         // ~ c1.operator ==(c2)
}
```

Programmation Orientée Objet pour écrire des applications c++ en mode console sur Windows ou Linux...



Les flux d'entrée/sortie

On trouve dans **iostream.h**, la déclaration des flux standards, qui sont des objets des classes istream_withassign et ostream_withassign.

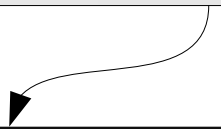
cin qui est l'analogue du flux stdin (entrée à partir du clavier)
cout qui est l'analogue de stdout (sortie standard sur l'écran)
cerr qui est l'analogue de stderr (sortie standard pour les erreurs)
clog qui n'a pas d'équivalent en langage c (sorties d'erreurs avec buffer)

On trouve dans ces 2 classes, respectivement, 2 opérateurs surchargés
<< et **>>** (opérateurs de décalage de bits du langage c)
L'opérateur << est associé à cout
L'opérateur >> est associé à cin

cout (console out ~ sortie console)

```
#include <iostream>
int main()
{
  std::cout << "Hello world!" << std::endl;
  return 0;
}
```

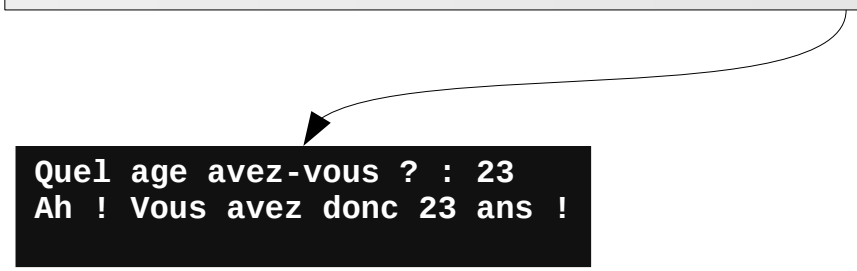
```
#include <stdio.h>   // pour printf
#include <iostream>
using namespace std;

int main(){
double age = 22.5;
int jour=12;
int mois= 6;
char departement[] = "GEII";

cout << "Salut, j'ai " << age << " ans et je suis en " << departement;
cout<<" mon aniversaire est le "<<jour<<'/'<<mois<<endl;

printf("Salut, j'ai %0.1f ans et je suis en %s mon aniversaire est le %d/%d\n",
                              age, departement, jour, mois);
return 0;
}
```



```
Salut, j'ai 22.5 ans et je suis en GEII mon aniversaire est le 12/6
Salut, j'ai 22.5 ans et je suis en GEII mon aniversaire est le 12/6
```

cin (console in ~ entrée console)

```
#include <iostream>
using namespace std;
int main()
{
    int age = 0;
    cout << "Quel age avez-vous ? : ";
    cin >> age;
    cout << "Ah ! Vous avez donc " << age << " ans !" << endl;
    return 0;
}
```



Le sens des chevrons correspond au flux d'information :

console << donnée	valeurs à écrire vers la console	`cout<<age`
clavier >> donnée	information entrée au clavier vers la variable	`cin>>age`

Formatage de l'affichage

endl	(sorties) Passe à la ligne et vide le tampon.
ends	(sorties) Insère un caractère nul.
flush	(sorties) Vide le tampon.
dec	Mode décimal.
hex	Mode hexadécimal.
oct	Mode octal.
ws	(entrées) Supprime les espaces.

```
cout <<dec<<i<<hex<<j<<endl;
```

Flags

ios::basefield
(ios::dec | ios::oct | ios::hex)

ios::adjustfield
(ios::left | ios::right | ios::internal)

ios::floatfield
(ios::scientific | ios::fixed)

```
int i = 245;
double d = 75.8901;

cout.precision(2);
cout.setf(ios::scientific, ios::floatfield);
cout << d;    // écrit 7.59e+01

cout.width(7);
cout.fill('$');
cout << i;    // écrit $$$$245

cout.width(9);
cout.fill('#');
cout.setf(ios::left|ios::hex,
          ios::adjustfield|ios::basefield);
cout << i;    // écrit f5#######

cout.fill(' ');
cout.width(6);
cout.setf(ios::internal|ios::showpos|ios::dec,
          ios::adjustfield|ios::showpos|
           ios::basefield);
cout << i;    // écrit +  245
```

Des classes prêtes à l'emploi : la STL "Standard Template Library"

Un très grand nombre de classes déjà écrites en c++ sont mises à notre disposition pour être utilisées dans nos programmes

http://www.cplusplus.com/

Ex : la classe std ::string pour les chaînes de caractères

```
// exemple de constructeur de la classe string
#include <iostream>
#include <string>
using namespace std ;

int main ()
{
  string s0 ("Initial string");
  // plusieurs constructeurs utilisable comme suit:
  string s1;
  string s2 (s0);
  string s3 (s0, 8, 3);
  string s4 ("A character sequence");
  string s5 ("Another character sequence", 12);
  string s6a (10, 'x');
  string s6b (10, 42);      // 42 is the ASCII code for '*'
  string s7 (s0.begin(), s0.begin()+7);

  cout << "s1: " << s1 << endl;
  cout << "s2: " << s2 << endl;
  cout << "s3: " << s3 << endl;
  cout << "s4: " << s4 << endl;
  cout << "s5: " << s5 << endl;
  cout << "s6a: " << s6a << endl;
  cout << "s6b: " << s6b << endl;
  cout << "s7: " << s7 << endl;
  return 0;
}
```

```
s1:
s2: Initial string
s3: str
s4: A character sequence
s5: Another char
s6a: xxxxxxxxxx
s6b: **********
s7: Initial
```

http://www.cplusplus.com/reference/string/string/

Surcharge de l'addition : concaténation

```
// opérateurs = et + de la classe string
#include <iostream>
#include <string>

using namespace std ;

int main ()
{
  string str1, str2, str3;

  str1 = "Test string: ";   // c-string
  str2 = 'x';               // single character
  str3 = str1 + str2;       // string

  cout << str3  << endl;
  return 0;
}
```

```
Test string: x
```

Et une multitude de méthodes et d'opérateurs simplifiant la gestion des chaînes de caractères :

- accéder aux caractères, extraire, insérer, supprimer une sous-chaîne
- ajouter des mots au début, à la fin
- taille de la chaîne
- *.etc.*

http://www.cplusplus.com/reference/string/string/

et plein d'autres choses...

Sequence containers:

array C++11	Array class (class template)
vector	Vector (class template)
deque	Double ended queue (class template)
forward_list C++11	Forward list (class template)
list	List (class template)

Container adaptors:

stack	LIFO stack (class template)
queue	FIFO queue (class template)
priority_queue	Priority queue (class template)

Associative containers:

set	Set (class template)
multiset	Multiple-key set (class template)
map	Map (class template)
multimap	Multiple-key map (class template)

Unordered associative containers:

unordered_set C++11	Unordered Set (class template)
unordered_multiset C++11	Unordered Multiset (class template)
unordered_map C++11	Unordered Map (class template)
unordered_multimap C++11	Unordered Multimap (class template)

http://www.cplusplus.com/reference/stl

Exercices utilisant la classe string

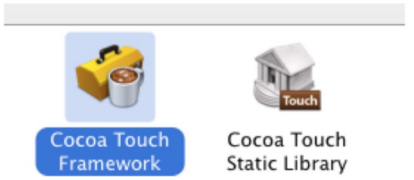
1) Écrire un programme qui :
   instancie une chaîne de caractères s1 contenant la phrase : "Au clair de la lune mon ami Pierrot"
   affiche s1
   remplace, dans s1, la sous chaîne "de la lune" par "du soleil" et l'affiche
   instancie une chaîne de caractère s2 contenant " grand "
   insère la chaîne s2 dans s1 avant le mot "ami" pour obtenir "mon grand ami" et affiche les deux chaînes
   ajoute à la fin de s1 la phrase ", prête moi ta plume" et l'affiche
   supprime dans s1 le mot " grand " qu'on avait ajouté précédemment et l'affiche
2) Définir une classe Personne qui représente une personne :
      nom, prénom, date de naissance, adresse, code postal, ville
   permettant de
      saisir une personne au clavier
      afficher la personne
      modifier les données de la personne
      comparer une personne à une autre (nom, prénom, date de naissance)
3) Définir une classe Etudiant (une personne+un numéro+une formation+groupe)
4)Définir une classe Enseignant (une personne+une matière+des groupes)

Consulter le site : http://www.cplusplus.com/reference/string/string/

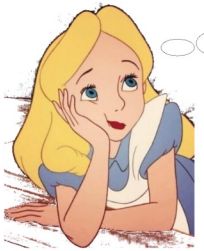
Encore plus de classes prêtes à l'emploi : utilisation d'un framework orienté objet

Applications de haut niveau :

- graphiques
- en réseau
- bases de données
- mobiles
- .*etc*.



quel framework ?
pour quel OS ?



ça recommence !

Le framework Qt

Qt est un **framework** orienté objet et développé en C++ par Qt Development Frameworks, filiale de Nokia.
Elle offre des composants d'interface graphique (**widgets**), d'accès aux données, de connexions réseaux, de gestion des fils d'exécution, d'analyse XML, etc.
Qt est par certains aspects un framework lorsqu'on l'utilise pour concevoir des i**nterfaces graphiques** ou que l'on architecture son application en utilisant les mécanismes des **signaux et slots.**

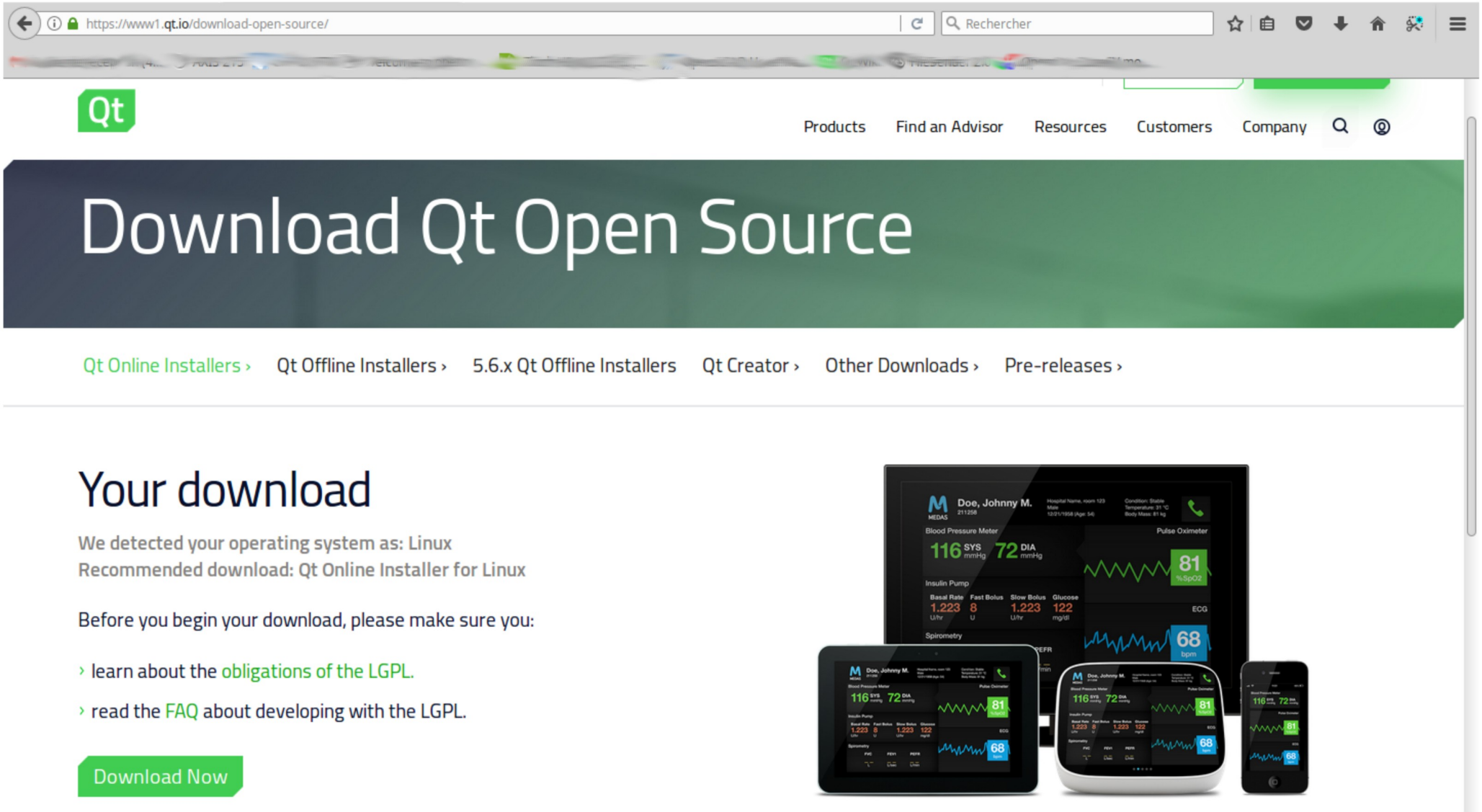
Qt **permet la portabilité des applications** qui n'utilisent que ses composants par simple recompilation du code source. Les environnements supportés sont les **Unix** (dont Linux) qui utilisent le système graphique X Window System, **Windows** et **Mac OS X**.

source : http://fr.wikipedia.org/wiki/Qt

Obtenir le SDK de Qt

https://www1.qt.io/download-open-source/

https://www1.qt.io/download-open-source/#section-2



Environnement de développement : Qt Creator

Qt Creator

Qt Creator est un environnement de développement intégré multiplate-forme faisant partie du framework Qt. Il est donc orienté pour la programmation en C++.

Il intègre directement dans l'interface un **débogueur**, un **outil de création d'interfaces graphiques**, des **outils pour la publication de code sur Git et Mercurial** ainsi que la **documentation Qt.**

L'éditeur de texte intégré permet l'autocomplétion ainsi que la coloration syntaxique.

Qt Creator utilise sous Linux le compilateur gcc et MinGW par défaut sous Windows.

Les classes du framework Qt

Le framework Qt est basé sur une hiérarchie de plusieurs milliers de classes toutes héritées d'une classe QObject, et regroupées par module :

Module	Description
Qt Core	Core non-graphical classes used by other modules.
Qt GUI	Base classes for graphical user interface (GUI) components. Includes OpenGL.
Qt Multimedia	Classes for audio, video, radio and camera functionality.
Qt Multimedia Widgets	Widget-based classes for implementing multimedia functionality.
Qt Network	Classes to make network programming easier and more portable.
Qt QML	Classes for QML and JavaScript languages.
Qt Quick	A declarative framework for building highly dynamic applications with custom user interfaces.
Qt Quick Controls	Reusable Qt Quick based UI controls to create classic desktop-style user interfaces.
Qt Quick Dialogs	Types for creating and interacting with system dialogs from a Qt Quick application.
Qt Quick Layouts	Layouts are items that are used to arrange Qt Quick 2 based items in the user interface.
Qt SQL	Classes for database integration using SQL.
Qt Test	Classes for unit testing Qt applications and libraries.
Qt Widgets	Classes to extend Qt GUI with C++ widgets.

http://doc.qt.io/qt-5/classes.html


Copyright © 2006 Trolltech ASA.
All rights reserved.
Main Classes in Qt 4.2

Quelques classes de base prises au hasard...

La classe QString

Une chaîne de caractère internationale UNICODE

```
#include "QString.h"

const QString dept = "GEII";
QString message;

message = "Vous avez réussi !";
message = dept;
message = 'W';

QString phrase = "Il était une fois une jolie princesse";
phrase.insert(20, " très");
//phrase contient donc "Il était une fois une très jolie princesse"

phrase.remove(21, 5);
//On revient donc à "Il était une fois une jolie princesse"

phrase.replace(21, 5, "affreuse");
//phrase contient maintenant "Il était une fois une affreuse princesse"

QString suite;
suite = phrase + " qui attendait son prince" ;
//suite contient maintenant "Il était une fois une affreuse princesse qui
attendait son prince"
```

Source : http://sites.univ-provence.fr/wcpp/V2/index.htm

QString : conversions numériques

Conversion d'une chaîne en nombre
les fonctions toShort(), toInt(), toLong(), toFloat() et toDouble()

```
QString chiffres = "123";
int n = chiffres.toInt();          // n contient maintenant 123
chiffres.replace(1, 1, ".");       //chiffres contient maintenant "1.3"
double k = chiffres.toDouble();    //k contient maintenant 1.3
n = chiffres.toInt();              //n contient maintenant 0

bool ok ;
chiffre = "1SS23";
n = chiffres.toInt(&ok);           // on tente la conversion

// ok devient faux si impossible
```

Conversion d'un nombre en chaine
la fonction setNum(x)

```
QString resultat;
double pi = 3.14159 ;
resultat.setNum(pi) ;    //resultat  contient maintenant 3.14159
```

Source : http://sites.univ-provence.fr/wcpp/V2/index.htm

La classe QPoint et QPointF

Un point du plan en précision entière (QPoint) ou réelle (QPointF)

	QPoint()
	QPoint(int *xpos*, int *ypos*)
bool	**isNull**() const
int	**manhattanLength**() const
int &	**rx**()
int &	**ry**()
void	**setX**(int *x*)
void	**setY**(int *y*)
int	**x**() const
int	**y**() const
QPoint &	**operator*=**(float *factor*)
QPoint &	**operator*=**(double *factor*)
QPoint &	**operator*=**(int *factor*)
QPoint &	**operator+=**(const QPoint & *point*)
QPoint &	**operator-=**(const QPoint & *point*)
QPoint &	**operator/=**(qreal *divisor*)

bool	**operator!=**(const QPoint & *p1*, const QPoint & *p2*)
const QPoint	**operator***(const QPoint & *point*, float *factor*)
const QPoint	**operator***(const QPoint & *point*, double *factor*)
const QPoint	**operator***(const QPoint & *point*, int *factor*)
const QPoint	**operator***(float *factor*, const QPoint & *point*)
const QPoint	**operator***(double *factor*, const QPoint & *point*)
const QPoint	**operator***(int *factor*, const QPoint & *point*)
const QPoint	**operator+**(const QPoint & *p1*, const QPoint & *p2*)
const QPoint	**operator+**(const QPoint & *point*)
const QPoint	**operator-**(const QPoint & *p1*, const QPoint & *p2*)
const QPoint	**operator-**(const QPoint & *point*)
const QPoint	**operator/**(const QPoint & *point*, qreal *divisor*)
QDataStream &	**operator<<**(QDataStream & *stream*, const QPoint & *point*)
bool	**operator==**(const QPoint & *p1*, const QPoint & *p2*)
QDataStream &	**operator>>**(QDataStream & *stream*, QPoint & *point*)

```
p1=(0,0)
p2=(100,230)
distance de Mahanattann de p1=330
p1=(1000,2300)
p1=(100,230)
p1=(80,280)
```

```
#include <QPoint>
#include <iostream>
using namespace std;

int main()
{
QPoint p1, p2(100, 230);

cout<<"p1=("<<p1.x()<<','<<p1.y()<<')'<<endl;
cout<<"p2=("<<p2.x()<<','<<p2.y()<<')'<<endl;

cout<<"distance de Mahattann de
p1="<<p2.manhattanLength()<<endl;

p1 = p2*10;
cout<<"p1=("<<p1.x()<<','<<p1.y()<<')'<<endl;

p1 = p1/10;
cout<<"p1=("<<p1.x()<<','<<p1.y()<<')'<<endl;

p1 = p2+QPoint(-20, 50);
cout<<"p1=("<<p1.x()<<','<<p1.y()<<')'<<endl;

return 0;
}
```

getters

Les classes QRect et QSize

Un rectangle du plan, une taille (largeur, hauteur)

	QRect()
	QRect(const QPoint & *topLeft*, const QPoint & *bottomRight*)
	QRect(const QPoint & *topLeft*, const QSize & *size*)
	QRect(int *x*, int *y*, int *width*, int *height*)
void	**adjust**(int *dx1*, int *dy1*, int *dx2*, int *dy2*)
QRect	**adjusted**(int *dx1*, int *dy1*, int *dx2*, int *dy2*) const
int	**bottom**() const
QPoint	**bottomLeft**() const
QPoint	**bottomRight**() const
QPoint	**center**() const
bool	**contains**(const QPoint & *point*, bool *proper* = false) const
bool	**contains**(int *x*, int *y*, bool *proper*) const
bool	**contains**(int *x*, int *y*) const
bool	**contains**(const QRect & *rectangle*, bool *proper* = false) const
void	**getCoords**(int * *x1*, int * *y1*, int * *x2*, int * *y2*) const
void	**getRect**(int * *x*, int * *y*, int * *width*, int * *height*) const
int	**height**() const
QRect	**intersected**(const QRect & *rectangle*) const
bool	**intersects**(const QRect & *rectangle*) const
bool	**isEmpty**() const
bool	**isNull**() const
bool	**isValid**() const
int	**left**() const
void	**moveBottom**(int *y*)
void	**moveBottomLeft**(const QPoint & *position*)
void	**moveBottomRight**(const QPoint & *position*)
void	**moveCenter**(const QPoint & *position*)
void	**moveLeft**(int *x*)
void	**moveRight**(int *x*)
void	**moveTo**(int *x*, int *y*)
void	**moveTo**(const QPoint & *position*)
void	**moveTop**(int *y*)
void	**moveTopLeft**(const QPoint & *position*)
void	**moveTopRight**(const QPoint & *position*)
QRect	**normalized**() const
int	**right**() const
void	**setBottom**(int *y*)
void	**setBottomLeft**(const QPoint & *position*)

```
#include <QPoint>
#include <QRect>
#include <iostream>
using namespace std;
// affichage console de données d'un QRect et d'un QPoint
void affiche_rect(string titre, QRect r){
cout<<titre<<":("<<r.x()<<','<<r.y()<<",largeur="<<r.width()
     <<",hauteur="<<r.height()<<')'<<endl;
}
void affiche_point(string titre, QPoint p){
    cout<<titre<<"=("<<p.x()<<','<<p.y()<<')'<<endl;
    }

int main()
{
QPoint p1(100, 230), p2(200, 300), p3, p4;
QRect r1, r2(p1, p2), r3(p1, QSize(20, 30));

affiche_rect("R1", r1);
affiche_rect("R2", r2);
affiche_rect("R3", r3);

p3=r3.bottomRight();              // point en bas à droite
affiche_point("P3", p3);
p4 = r3.bottomLeft();           // point en bas à gauche
affiche_point("P4", p4);

r3.translate(20, -20);  // translaton du rectangle
affiche_rect("R3", r3);

if (r2.intersects(r3)){
        r1=r2.intersected(r3);    // intersection de rectangles
        affiche_rect("R1", r1);
}
else cout<<"pas d'intersection R2, R3"<<endl;

if (r1.contains(p3))cout<<"P3 dans R1"; else cout<<"P3 n'est pas dans R1";
cout<<endl;
return 0;
}
```

```
R1:(0,0,largeur=0,hauteur=0)
R2:(100,230,largeur=101,hauteur=71)
R3:(100,230,largeur=20,hauteur=30)
P3=(119,259)
P4=(100,259)
R3:(120,210,largeur=20,hauteur=30)
R1:(120,230,largeur=20,hauteur=10)
P3 n'est pas dans R1
```

La classe QDate

Une date sous toutes ses formes

	QDate()
	QDate(int *y*, int *m*, int
QDate	addDays(qint64 *nday*
QDate	addMonths(int *nmon*
QDate	addYears(int *nyears*)
int	day() const
int	dayOfWeek() const
int	dayOfYear() const
int	daysInMonth() const
int	daysInYear() const
qint64	daysTo(const QDate
void	getDate(int * *year*, int
bool	isNull() const
bool	isValid() const
int	month() const
bool	setDate(int *year*, int
qint64	toJulianDay() const
QString	toString(const QStrin
QString	toString(Qt::DateFor
int	weekNumber(int * *ye*
int	year() const
bool	operator!=(const QD
bool	operator<(const QDa
bool	operator<=(const QD
bool	operator==(const QD
bool	operator>(const QDa
bool	operator>=(const QD

```
date ? :31/8/2021
d1: 15.08.2019, 227 ème jour de l'année 2019
d2: 31.08.2021, mardi, dans 747 jours/aujourd'hui'
d3: 01.08.2021 dimanche, 717 jours/aujourd'hui, dimanche, 2021 a 365 jours
```

```
 d2(

QDate::currentDate()

   toString("dd.MM.yyyy")
                                        d1.dayOfYear()


        d2.toString
      QDate::longDayName d2.dayOfWeek
          d1.daysTo

d2.addDays


 d3.year          d3.daysInYear
```

La classe QTime

L'heure sous toutes ses formes

```
t1: 09:39:27
t2: 15:12:03
t1 est avant t2
il y a 19956 secondes d'écart
```

```
void affiche_time(string titre, QTime t){
cout<<titre<<": "<<t.toString().toUtf8().constData()<<endl;
}
```

```
t2(
QTime::currentTime
t1>
t1==
t1.secsTo(
```

	QTime()
	QTime(int *h*, int *m*, int *s* = 0, int *ms* = 0)
QTime	**addMSecs**(int *ms*) const
QTime	**addSecs**(int *s*) const
int	**elapsed**() const
int	**hour**() const
bool	**isNull**() const
bool	**isValid**() const
int	**minute**() const
int	**msec**() const
int	**msecsSinceStartOfDay**() const
int	**msecsTo**(const QTime & *t*) const
int	**restart**()
int	**second**() const
int	**secsTo**(const QTime & *t*) const
bool	**setHMS**(int *h*, int *m*, int *s*, int *ms* = 0)
void	**start**()
QString	**toString**(const QString & *format*) const
QString	**toString**(Qt::DateFormat *format* = Qt::Te
bool	**operator!=**(const QTime & *t*) const
bool	**operator<**(const QTime & *t*) const
bool	**operator<=**(const QTime & *t*) const
bool	**operator==**(const QTime & *t*) const
bool	**operator>**(const QTime & *t*) const
bool	**operator>=**(const QTime & *t*) const

La classe QTime

L'heure sous toutes ses formes

	QTime()
	QTime(int *h*, int *m*, int *s* = 0, int *ms* = 0)
QTime	**addMSecs**(int *ms*) const
QTime	**addSecs**(int *s*) const
int	**elapsed**() const
int	**hour**() const
bool	**isNull**() const
bool	**isValid**() const
int	**minute**() const
int	**msec**() const
int	**msecsSinceStartOfDay**() const
int	**msecsTo**(const QTime & *t*) const
int	**restart**()
int	**second**() const
int	**secsTo**(const QTime & *t*) const
bool	**setHMS**(int *h*, int *m*, int *s*, int *ms* = 0)
void	**start**()
QString	**toString**(const QString & *format*) cons
QString	**toString**(Qt::DateFormat *format* = Qt::T
bool	**operator!=**(const QTime & *t*) const
bool	**operator<**(const QTime & *t*) const
bool	**operator<=**(const QTime & *t*) const
bool	**operator==**(const QTime & *t*) const
bool	**operator>**(const QTime & *t*) const
bool	**operator>=**(const QTime & *t*) const

```
les boucles seules ont pris 24 ms
la fonction rand toute seule prend 8.6 ns
```

```
void affiche_time(string titre, QTime t){
cout<<titre<<": "<<t.toString().toUtf8().constData()<<endl;
}
```

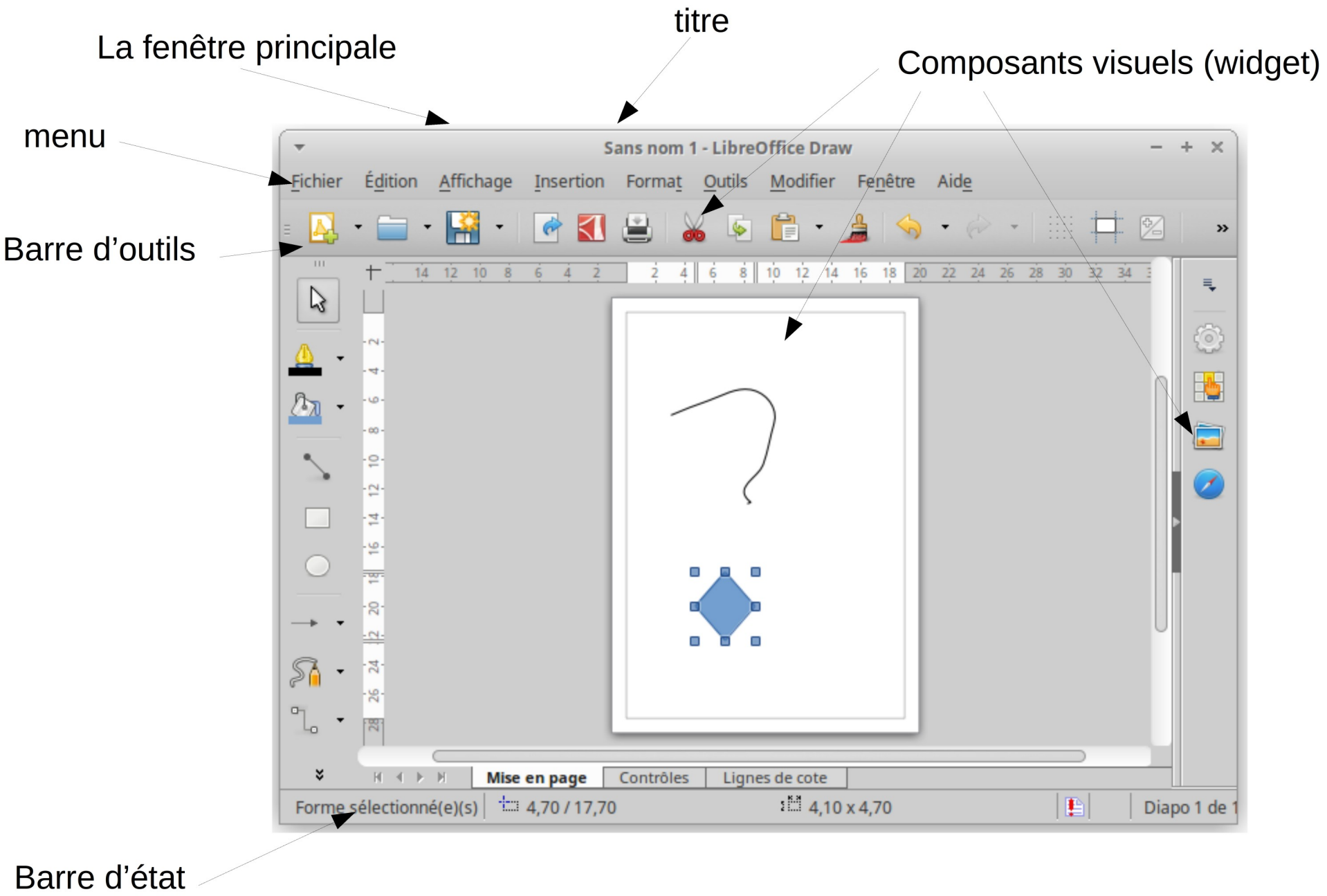
```
t3.start(
```

```
    t3.restart
```

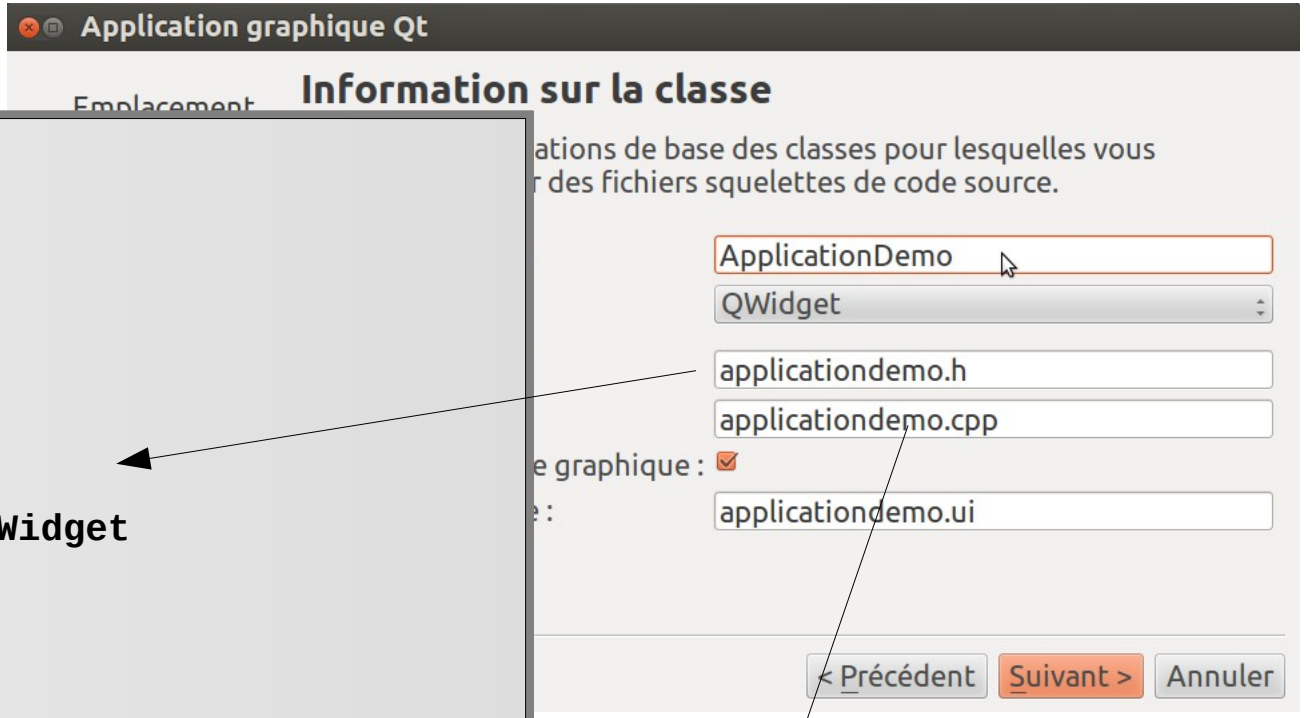
```
    t3.elapsed
```

Application graphique Qt

Application graphique : exemple



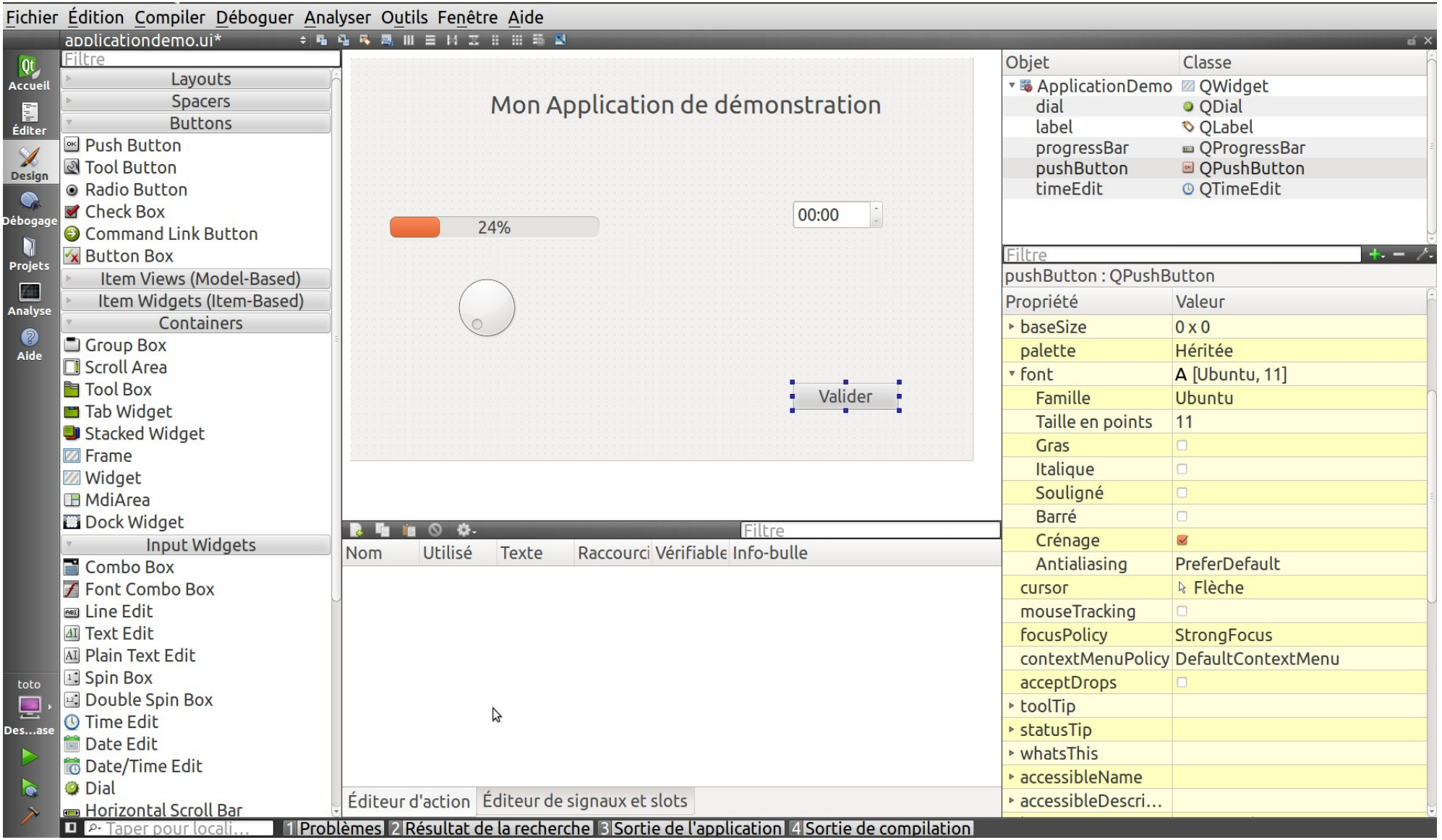
Création d'une application graphique Qt




```
#ifndef APPLICATIONDEMO_H
#define APPLICATIONDEMO_H

#include <QWidget>

namespace Ui {
class ApplicationDemo;
}

class ApplicationDemo : public QWidget
{
    Q_OBJECT

public:
    explicit ApplicationDemo(QWidget *parent = 0);
    ~ApplicationDemo();

private:
    Ui::ApplicationDemo *ui;
};

#endif // APPLICATIONDEMO_H
```

```
#include "applicationdemo.h"
#include "ui_applicationdemo.h"

ApplicationDemo::ApplicationDemo(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::ApplicationDemo)
{
    ui->setupUi(this);
}

ApplicationDemo::~ApplicationDemo()
{
    delete ui;
}
```

L'interface utilisateur (*user interface*)



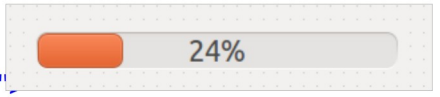
Qt Designer : dessin de l'interface (fichier : applicationdemo.ui)

Le fichier de design de l'UI

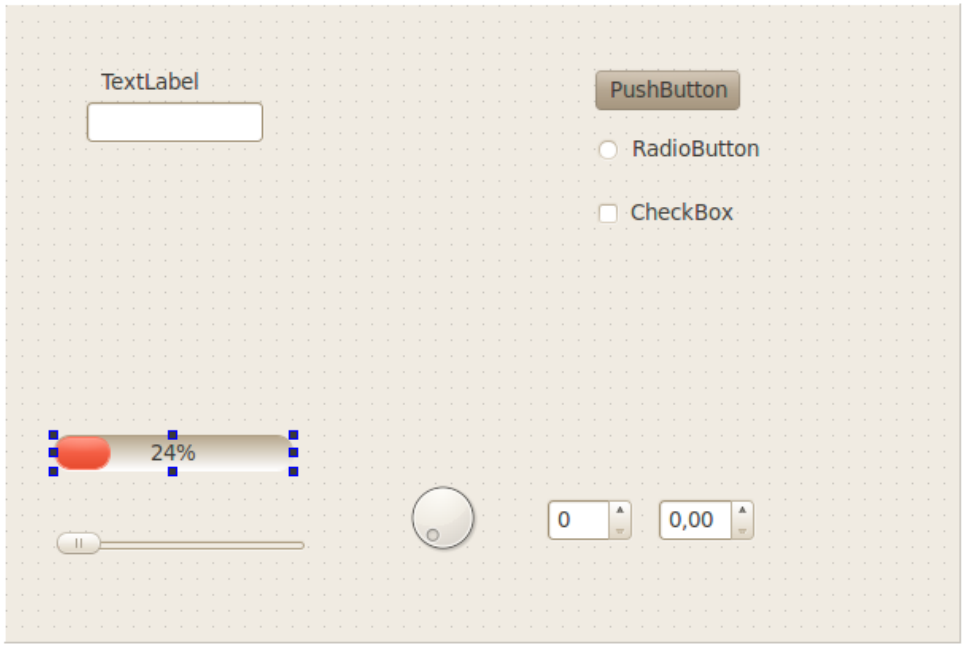
```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>ApplicationDemo</class>
 <widget class="QWidget" name="ApplicationDemo">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>803</width>
    <height>509</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>ApplicationDemo</string>
  </property>
  <widget class="QLabel" name="label">
   <property name="geometry">
    <rect>
     <x>180</x>
     <y>40</y>
     <width>505</width>
     <height>37</height>
    </rect>
   </property>
   <property name="font">
    <font>
     <pointsize>16</pointsize>
    </font>
   </property>
   <property name="text">
    <string>Mon Application de démonstration</string>
```

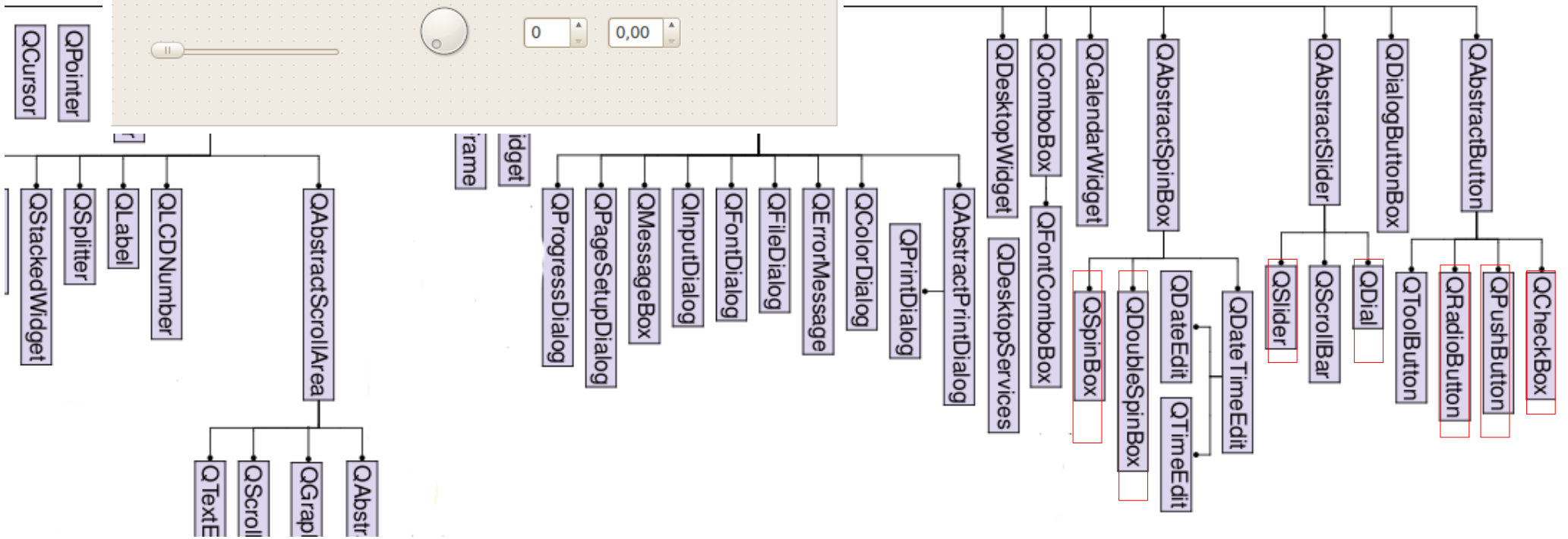
Mon Application de démonstration

```
<?xml version="1.0" encoding="UTF-8"?
   </property>
  </widget>
  <widget class="QDial" name="dial">
   <property name="geometry">
    <rect>
     <x>130</x>
     <y>270</y>
     <width>91</width>
     <height>91</height>
    </rect>
   </property>
  </widget>
  <widget class="QProgressBar" name="progressBar">
   <property name="geometry">
    <rect>
     <x>50</x>
     <y>200</y>
     <width>271</width>
     <height>28</height>
    </rect>
   </property>
   <property name="value">
    <number>24</number>
   </property>
  </widget>
  <widget class="QTimeEdit" name="timeEdit">
   <property name="geometry">
    <rect . . . . . .
…....
```



Quelques classes remarquables...

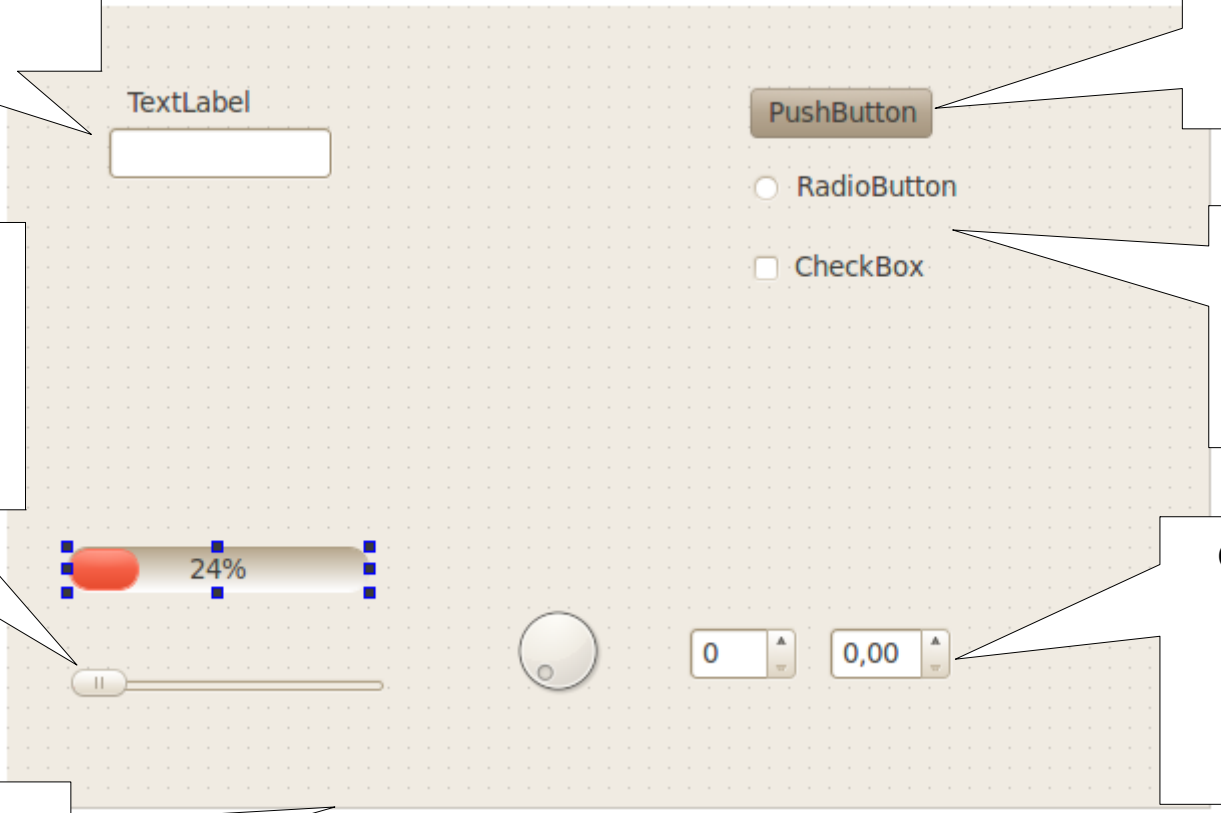




Événements signal/slots

On m’a modifié mon contenu
On a fait « entrée » sur moi
On m’a détruit
.etc.

On m’a appuyé dessus
On m’a relâché
On m’a double-cliqué
On m’a détruit
.etc.



On a modifié ma valeur
On m’a déplacé
On m’a cliqué
On m’a relâché
On m’a détruit
.etc.

On m’a appuyé dessus
On m’a relâché
On m’a double-cliqué
On m’a détruit
.etc.

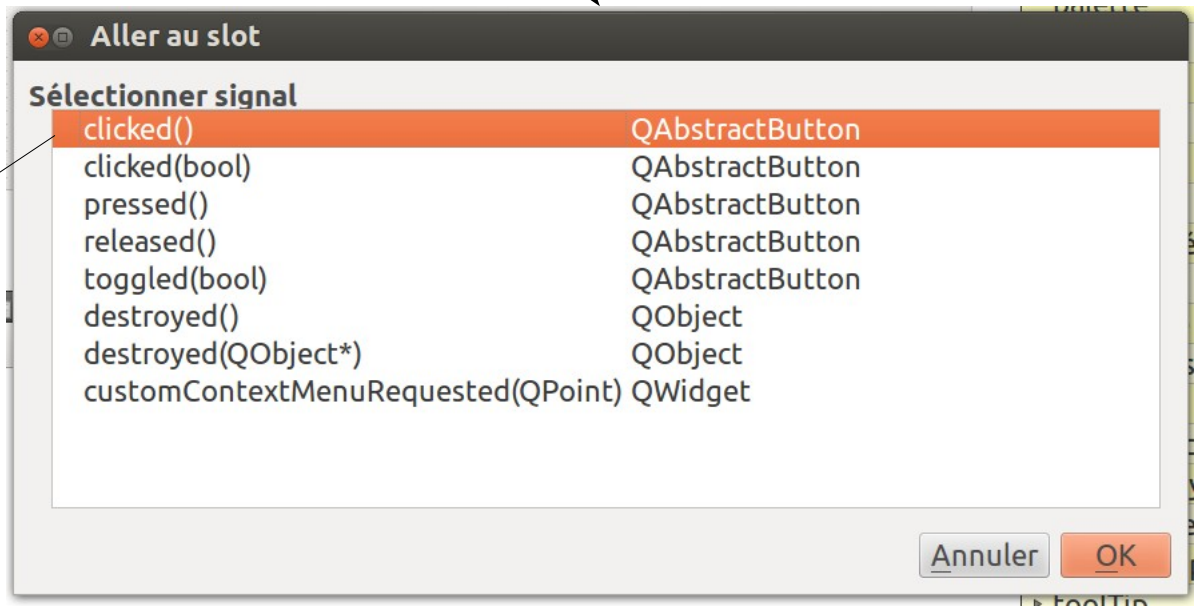
On a modifié ma valeur
On m’a déplacé
On m’a cliqué
On m’a relâché
On m’a détruit
.etc.

On m’a déplacé
On m’a retaillé
On veut me fermer
On m’a détruit
.etc.

Chaque composant Qt (représenté par une classe) peut émettre des **signaux,** correspondant à des actions, au noyau de l’application
On peut connecter un de ces signaux à une méthode d'une classe quelconque (**slot**)

connexion d'un signal à un slot dans le designer

Click droit



On choisit le signal que l'on veut gérer,

Fichier d'entête : une nouvelle méthode est crée

```
private slots:
    void on_toolButton_clicked();
```

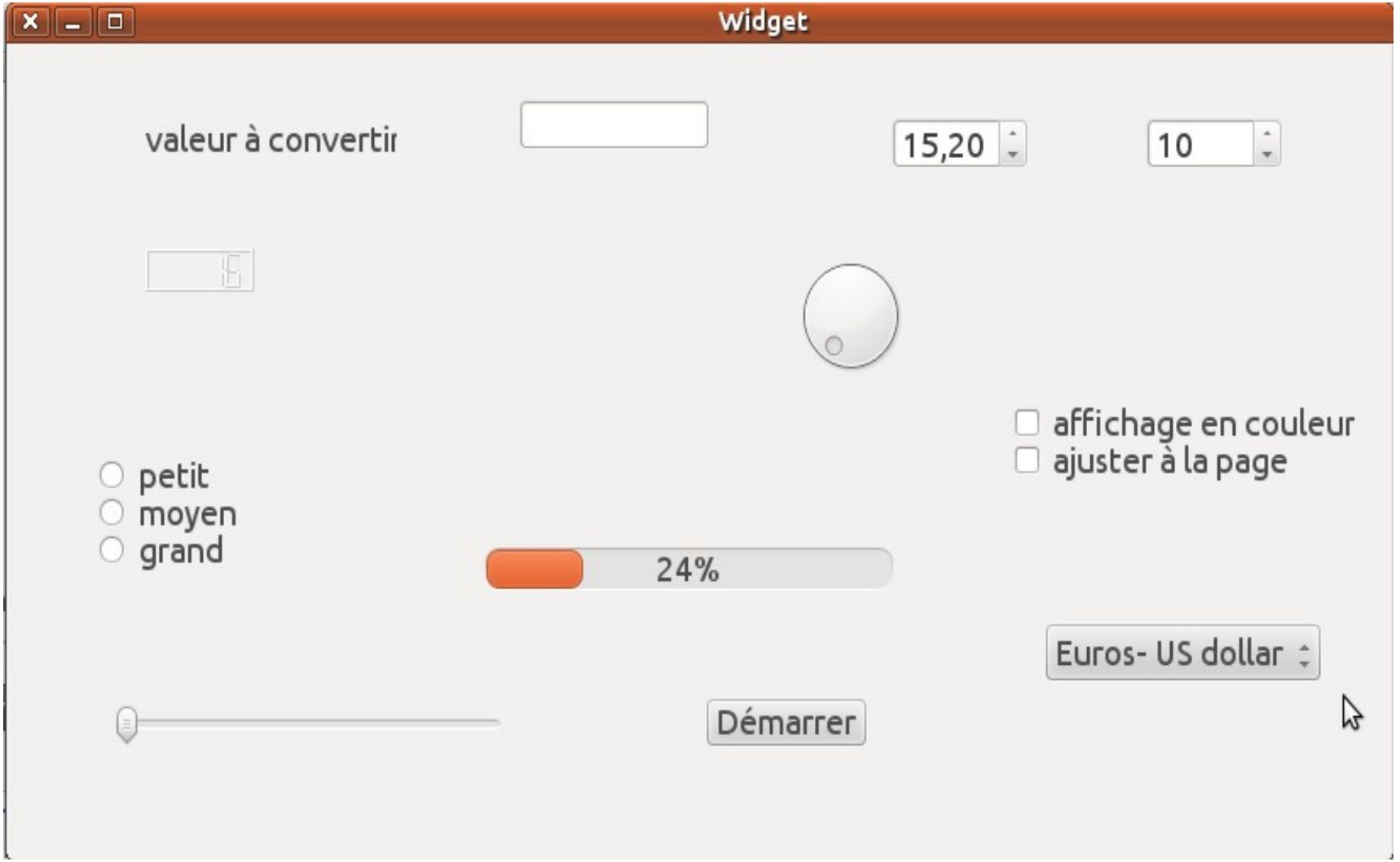
Fichier source : implémentation de la méthode

```
void ApplicationDemo::on_toolButton_clicked()
{

// Écrire ici le code qui se déroulera lorsqu'on
// aura clické sur le bouton « Valider »
}
```

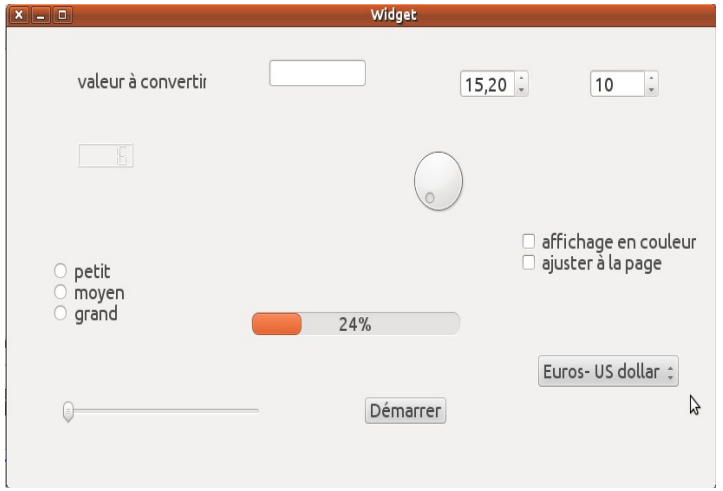
On pourrait également le faire par programme :

```
connect (objet_ émetteur, signal, objet_récepteur, slot)
```

Actions courantes sur les composants de base




```
this->ui->composant->methode(arg1, arg2, ...);
```



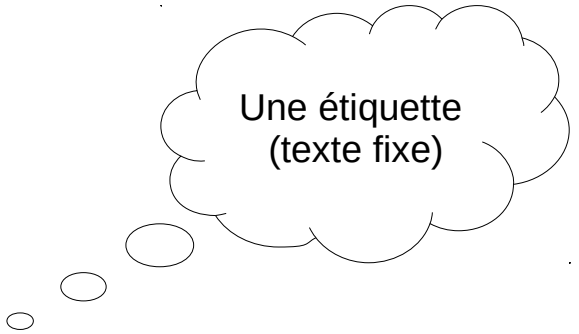
QWidget

La fenêtre principale de l'application

```
this->setVisible(true);                 // rend visible
this->setVisible(false);                // rend invisible
…
QFont serifFont("Times", 10, QFont::Bold);
this->setFont(serifFont);
…

this->setCursor(Qt::WaitCursor);    // curseur sablier
…
    // la taille et la position par rapport au coin haut gauche de l'écran
QRect r;                                // un rectangle
r = this->geometry();
r.setWidth(r.width()+30);               // augmente la largeur de 30 pixels
this->setGeometry(r);
…
this->setWindowTitle("Essai - version 1.00");
…
this->close();   // ferme le widget
…
```

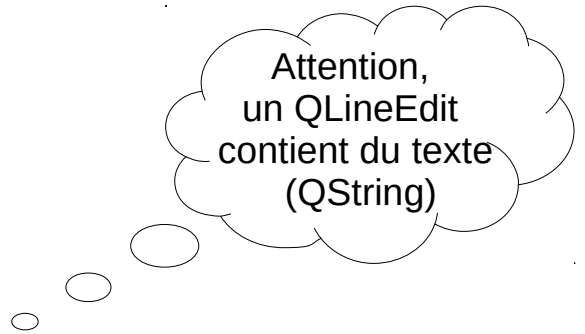
QLabel

valeur à convertir




```
ui->label->setText("salut");    // modifie le texte


ui->label->setVisible(true);    // visible
ui->label->setVisible(false);    // invisible


QString contenu = ui->label->text();    // le texte
contenu = contenu + " les GEII";        // concaténation
ui->label->setText(contenu);            // salut les GEII


if (ui->label>isVisible()){       // est il visible?
                           }
      else {  // pas visible
                 }
```

QLineEdit

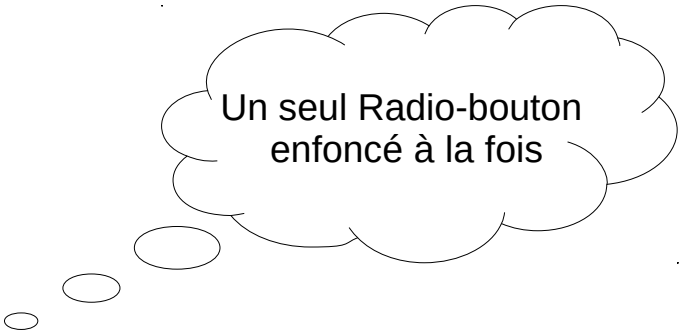



```
ui->lineEdit->setVisible(true);

QRect r = ui->lineEdit->geometry();

ui->lineEdit->setCursor(Qt::CrossCursor);

double valeur;
QString chaine;
bool conversionOk;
chaine = ui->lineEdit->text();

valeur = chaine.toDouble(&conversionOk);

if (conversionOk && valeur>=0){
        valeur = sqrt(valeur);
        chaine.setNum(valeur);
        ui->lineEdit->setText(chaine);
        }
    else ui->lineEdit->setText("impossible");
```

QPushButton

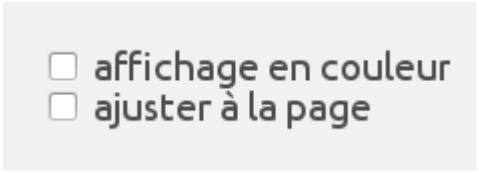

```
ui->pushButton->setText("start");

ui->pushButton->setVisible(true);   // ou true

ui->pushButton->setEnabled(false);  // ou true

QRect r = ui->pushButton->geometry();


if (ui->pushButton->text()=="start")
                            ui->pushButton ->setText("stop");
                     else  ui->pushButton->setText("start");
```

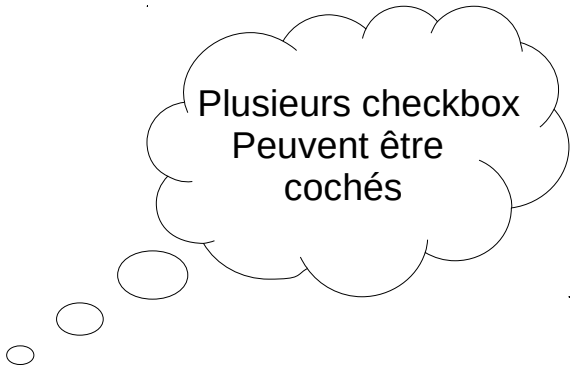
QRadioButton




```
ui->radioButton->setVisible(true);

ui->radioButton->setText("trés grand");

ui->radioButton->setChecked(true); // ou false


if (ui->radioButton->isChecked()){  // il est enfoncé
                                    …
                                  }
else                              { // il n'est pas enfoncé
                                  …
                                              }
```

QCheckBox





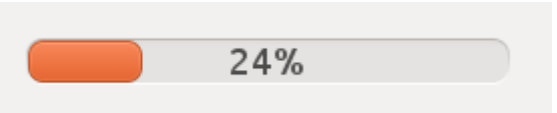

```
ui->checkBox->setVisible(true);

ui->checkBox->setText("affichage en couleur");

ui->checkBox->setChecked(true); // ou false

if (ui->checkBox->isChecked()){  // il est enfoncé
                                 }
else                             { // il n'est pas enfoncé
                                 }
```

QSpinBox


```
ui->spinBox->setVisible(true);  // ou false

// intervalle de définition
ui->spinBox->setMaximum(360);
ui->spinBox->setMinimum(-360);

// récupération de la valeur
int valeur = ui->spinBox->value();

valeur *=2;

ui->spinBox->setValue(valeur);
```

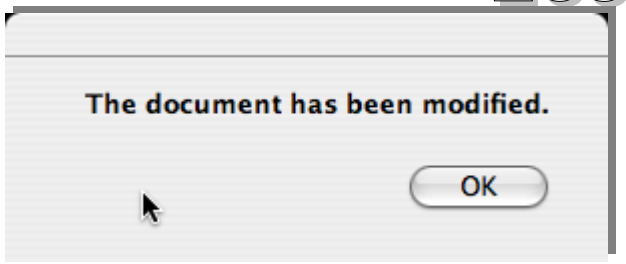
QDoubleSpinBox idem mais en double

QSlider- QDial - QProgressBar

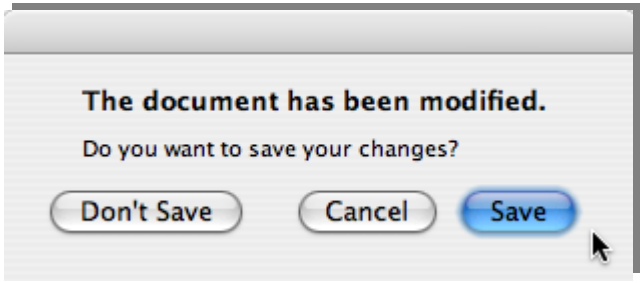



```
ui->horizontalSlider->setMaximum(200);  // on minimum
ui->progressBar->setMaximum(200);

int valeur = ui->horizontalSlider->value();

ui->progressBar->setValue(valeur);

ui->dial->setValue(200-valeur);

// mais aussi :
// la géométrie
// visible ou pas
// le cursor
// la font
// .etc.
```

Les MessagesBox prédéfinis




```
QMessageBox msgBox;
 msgBox.setText("The document has been modified.");
 msgBox.exec();
```




```
QMessageBox msgBox;
 msgBox.setText("The document has been modified.");
 msgBox.setInformativeText("Do you want to save your changes?");
 msgBox.setStandardButtons(QMessageBox::Save |
                           QMessageBox::Discard |
                           QMessageBox::Cancel);
 msgBox.setDefaultButton(QMessageBox::Save);
 int ret = msgBox.exec();
```

```
int ret = QMessageBox::warning(this, tr("My Application"),
                      tr("The document has been modified.\n",
                      "Do you want to save your changes?"),
                      QMessageBox::Save |
                      QmessageBox::Discard |
                      QMessageBox::Cancel,
                      QMessageBox::Save);
```

Les dialogues d'entrée prédéfinis

Static Public Members

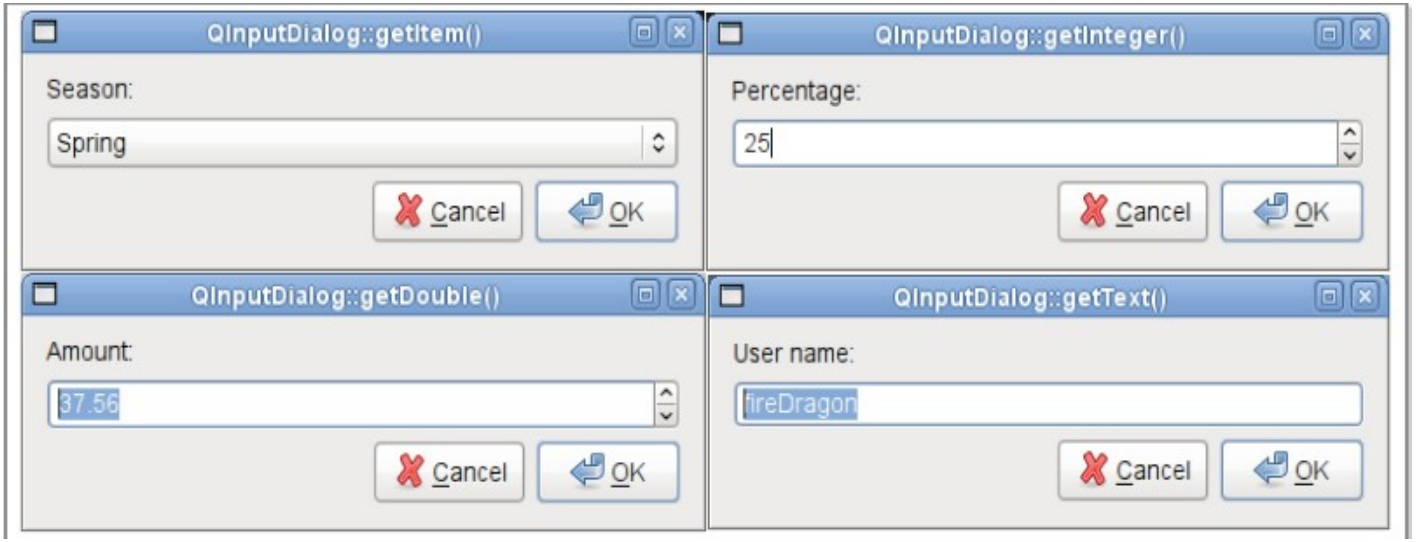
double **getDouble**(QWidget * *parent*, const QString & *title*, const QString & *label*, double *value* = 0, double *min* = -2147483647, double *max* = 2147483647, int *decimals* = 1, bool * *ok* = 0, Qt::WindowFlags *flags* = 0)

int **getInt**(QWidget * *parent*, const QString & *title*, const QString & *label*, int *value* = 0, int *min* = -2147483647, int *max* = 2147483647, int *step* = 1, bool * *ok* = 0, Qt::WindowFlags *flags* = 0)

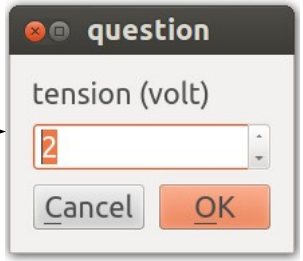
int **getInteger**(QWidget * *parent*, const QString & *title*, const QString & *label*, int *value* = 0, int *min* = -2147483647, int *max* = 2147483647, int *step* = 1, bool * *ok* = 0, Qt::WindowFlags *flags* = 0) (deprecated)

QString **getItem**(QWidget * *parent*, const QString & *title*, const QString & *label*, const QStringList & *items*, int *current* = 0, bool *editable* = true, bool * *ok* = 0, Qt::WindowFlags *flags* = 0, Qt::InputMethodHints *inputMethodHints* = Qt::ImhNone)

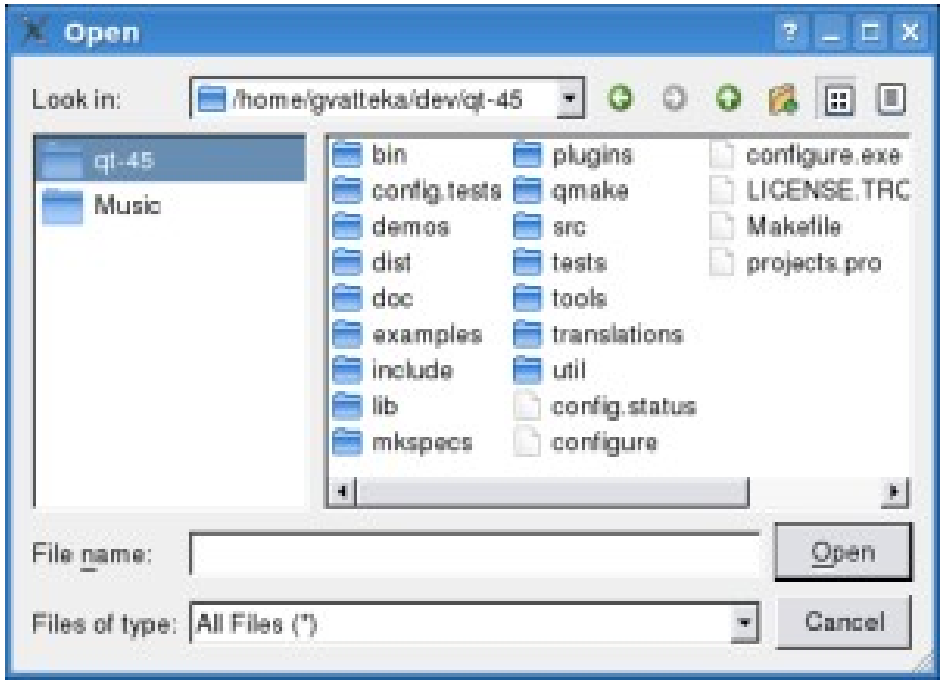
QString **getText**(QWidget * *parent*, const QString & *title*, const QString & *label*, QLineEdit::EchoMode *mode* = QLineEdit::Normal, const QString & *text* = QString(), bool * *ok* = 0, Qt::WindowFlags *flags* = 0, Qt::InputMethodHints *inputMethodHints* = Qt::ImhNone)




```
#include <QinputDialog>

…

int x = QInputDialog::getInt(this, "question","tension (volt)" );
```



Les dialogues de fichiers prédéfinis




```
QList<QUrl> urls;
urls << QUrl::fromLocalFile("/home/grimaldi/qt-45")
     << QUrl::fromLocalFile(QDesktopServices::storageLocation(QDesktopServices::MusicLocation));

 QFileDialog dialog;
 dialog.setSidebarUrls(urls);
 dialog.setFileMode(QFileDialog::AnyFile);
 if(dialog.exec()) {
      // ...
 }
```

Pour le reste voir l'aide de Qt .....