Displayable

ostream & affiche (ostream &) ; ← virtuelle pure.

⇓

```
ostream & operator << (ostream & out, const Displayable & p)
{
    return p.affiche (out);
}
```

classe abstraite ←
```
class Displayable {
public:
    ostream & affiche .....
}
```
classe très générique

méthode non définie. (*)

? → on définit affiche dans chacune des classes filles → sous de Displayable

classe concrètes

```
class Vector : public Displayable
{
    ....
    // définir affiche pour les Vector
```

```
template ....
class Pilegen : public Displayable
{
    ....
    définir affiche pour les Pile Gen.
```

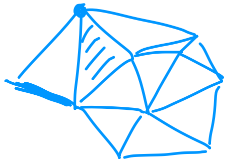
Classe abstraite
↑
méthode "non définie" ⇝ virtuelle pure. → impl. dans les classes dérivées concrètes
|
juste prototypée.

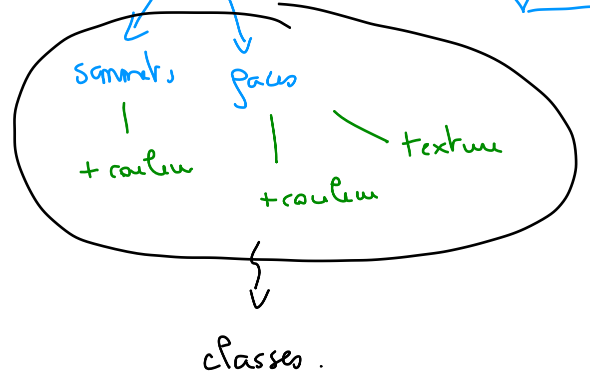
Polymorphisme

```
class Displayable
{
    public:
        virtual ostream & affiche (ostream &) (const) = 0;
}
```
(const) → opt.

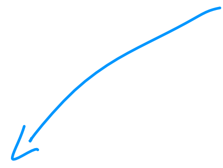
Displayable
↳ afficher

→ 3D - OpenGL
↓
maillages



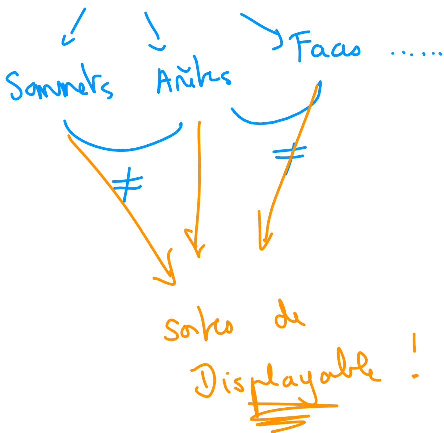


↓
classes.



moteur graphique
↓
liste ou tableau de choses à afficher

Sommets Arêtes Faces .....



sortes de Displayable !

← polymorphisme

Displayable * a_afficher [20];
↓

```
for (int i=0; i<20; i++)
    a_afficher [i] -> afficher ();
```

← polymorphisme ....

class Sommet : public Displayable
↳ afficher

class Sommet_coloré : public Sommet, Displayable
↳ afficher

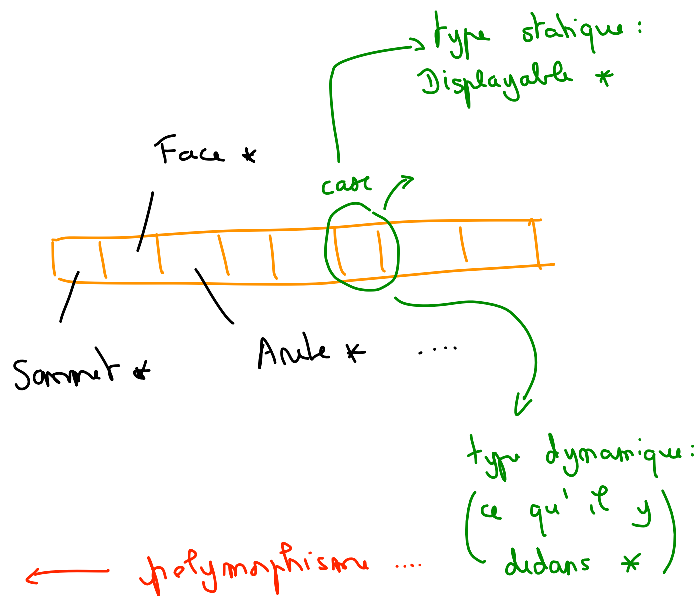
class Face : public Displayable.
↳ afficher

⋮

class Arêtes : public Displayable
↳ afficher

afficher → impl. spécif.

→ type statique : Displayable *



type dynamique : (ce qu'il y dedans *)

→ TP5.

1. Si on définit :
int i = 0 ;
int &j ;
Comment peut-on initialiser j?

j = &i ;	(4) 36%
j = *i ;	(2) 18%
j=i ;	(5) 45%

j = &i

int * R = &i;

i : int
&i : adresse de i
type : int *
~~int &~~

*i int ~~adresse~~
déréférence 1 ptr
~~*&~~ — pas de sens ...

j = i variable entière
réf s/ entier
réf s/ i
ds j

int i = 0;
int &j = i ;
variable.

int &j = 2+3 ; (const)
s/ pile
réf. sur la pile

2. Dans la classe Toto, on définit :
int ma_methode (const Vector & v) const ;
Le premier const signifie que:

La méthode ne change pas l'objet dans lequel on est	(3) 27%
La méthode ne change pas l'argument v	(8) 73%

3. Dans la même classe, le second const signifie que:

La méthode ne change pas l'objet dans lequel on est	(9) 82%
La méthode ne change aucun argument	(2) 18%

se rapporte à v

const Vector & v
v : passé par réf
(mémoire ... pas de copie).
on interdit d'utiliser la réf pour modifier v.

après le nom de la méthode.
⟹ ma_méthode ne modifie pas this

4. Si dans la fonction ma_methode, j'appelle la méthode
this->afficher() ;
Alors afficher doit être déclarée comme :

void afficher() ;	(5) 45%
void afficher() const ;	(6) 55%

2ème const
⟹ on ne modifie pas this ...

```
int ma_méthode (const Vector &) const
{
    ⋮
    this -> afficher ();
}
```

↓ doit être const ⚠

```
class Toto {
  private :
     int _i ;
  public:
     int meth1 (void) { return _i ; }
     void meth2 ( int i) { _i = i ; }
}
```

const → on renvoie 1 val

~~const~~

↓ modifie this

(méthodes const) ↘ modifie pas l'objet.

inline → routine courte | appelées souvent.
↓
get / set

méth1 fct pas inline

main Toto t ;
{
t.méth1 (); → sauvegarde du contexte
← restauration

return _i ......

Héritage "en diamant"