## UML

Diagramme de Classe :

<http://dia-installer.de/shapes/UML/index.html.fr>

<https://staruml.io/download>

~ 🡪 protected

+ 🡪 public

\*-[LOSANGE]\* 🡪

-[triangle] 🡪 héritage

Diagramme de Séquence :

éviter de se perdre entre toutes les classes

## STATIC

Normalement les variables des fonctions existent au niveau des **objets**

mais avec *static*, la variable existe au niveau de la **classe**, elle est commune entre tous les objets créés par cette classe

peut permettre de compter le nombre d’objets qu’on a, par exemple, avec un *static int* qui s’incrémente à chaque fois

cette variable est créée au moment où on lance le programme

## FONCTIONS PAR DEFAUT POUR TOUTES LES CLASSES

[fonction] constructeur par défaut

‘’ constructeur par copie 🡪 s’utilise très facilement sans s’en rendre compte, ‘’

‘’ assignement 🡪 ‘’, il est donc très important d’avoir le réflexe de les *delete*, lorsqu’on ne veut pas s’en servir

exemple : Window& operator~(const Window&) = delete ;

‘’ destruction

## FONCTIONS SDL

SDL\_LogError(static\_cast<int>(category), "%s | SDL: %s", message.c\_str(), SDL\_GetError());

% ~ seront remplacés par les arguments qui suivent dans la fonction, s pour string

(extrait de Log.cpp)

## CONSEILS ARCHITECTURE

de manière générale, essayez de créer des interfaces qui sont faciles à utiliser

càd, avoir dans le header des fonctions claires

ce qui se cache dans le .cpp, ‘’on s’en fiche’’

## CONSEILS GENERAUX

il vaut mieux utiliser les accolades { } que les parenthèses ( ), dans les versions récentes

/!\ 🡪

si tu sais qu’une variable ne va pas changer, il faut quasi systématiquement préciser un *const* devant

*constexpr* 🡪 variante de *const*, qui va se créer au moment de la compilation, sans avoir besoin de lancer le programme

les fonctions à une seule ligne, laissez-les dans le .h, ne vous embêtez pas à la mettre dans le .cpp

exemple : int getWidth() const { return width; }

**Design pattern** : is a “good” solution that can be standardized when programmers meet often a certain class of problems (singleton, par exemple)

**Singleton** : permet d’avoir un objet unique, on n’aura qu’une seule instance d’une classe

permet d’avoir accès à l’objet de manière globale

souvent, les architectes logiciels ont tendance à ne pas trop l’aimer, justement parce qu’on peut y accéder de manière globale

**Polling** :

Conversions :

static\_cast<TYPE>(VARIABLE) 🡪 permet de convertir un type en un autre

reinterpret\_cast… 🡪 permet de changer de type de pointeur

quand vous voulez des variables statiques, il faut les déclarer dans le ficher .cpp aussi, pour éviter plein de soucis

## QUESTIONS

for (auto actor : actors) { }