

TP C++ - RELATION ENTRE CLASSES

*Objectif :*

Concevoir des classes et leurs relations en fonction d'un diagramme de classe. Etude du mot clé static et du mot clé const.

*Consigne :*

Dans certain cas, il sera peut-être nécessaire de réaliser des méthodes supplémentaires permettant d'accéder aux attributs d'une classe. A vous de juger.

I LA COMPOSITION (AGRÉGATION FORTE)

*Rappel :* Si le composé est détruit alors le composant l'est également. Un composant ne peut être détruit qu'avec la destruction du composé.

*Diagramme de classe :*

*Présentation :*

**CRoue :** l'attribut cptRoue est un attribut de classe de type static. Il est partagé par toutes les instances et permet de compter et connaître le nombre de roues instanciées. (cf Annexe).

**CVehicule :** doit avoir comme attribut un tableau de type CRoue de 4 éléments nommé liste.

*Coder la classe CRoue (Croue.cpp et Croue.h).*

Vous ferez apparaître un message à la construction et destruction contenant le nombre de roues construites et détruites

Exemple : **« Constructeur roue 1 » / « Destructeur roue 1 »**

Testez votre classe en créant un ou des objets de type CRoue dans le main. Exécutez et vérifiez l'apparition des messages.

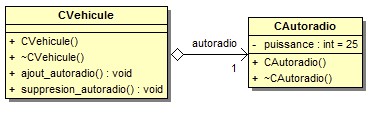
*Coder la classe CVehicule (CVehicule.cpp et CVehicule.h).*

Vous ferez apparaître un message à la construction et destruction d'un objet de type CVehicule. Exemple **: « Constructeur vehicule » / « Destructeur vehicule»**

Testez votre classe en créant un objet v1 de type CVehicule dans le main. Exécutez et vérifiez l'apparition des messages.

Notez l'ordre de construction et de destruction des objets. Que remarquez-vous ?

1 / 7



A quel moment l'objet de type CVehicule est-il détruit ? Peut-on contrôler sa destruction ?

*Faites les modifications suivantes afin de mieux contrôler la destructions des roues :*

Créer de manière dynamique (new) le tableau liste juste après le message de construction et détruisez ce tableau (delete[ ]) juste avant le message de destruction.

II L'AGRÉGATION SIMPLE

*Rappel :* Si le composé est détruit alors le composant l'est également. Cependant, un composant peut être détruit sans la destruction du composé.

*Diagramme de classe :*

*Présentation :*

CAutoradio : Doit avoir comme attribut un entier représentant la puissance et initialiser à la valeur 25. CVehicule : doit avoir comme attribut un pointeur sur un type CAutoradio nommé autoradio.

*Coder la classe CAutoradio.*

Vous ferez apparaître un message à la construction et destruction de l'objet. Exemple : « Constructeur autoradio » / « Destructeur autoradio »

*Modifier la classe CVehicule.*

Vous ajouterez deux méthodes permettant de créer et détruire dynamiquement un autoradio.

Vous ferez apparaître un message à l'ajout et à la destruction dans ces méthodes.

Testez votre classe en créant un objet v1 de type CVehicule dans le main. Manipuler et vérifiez l'apparition

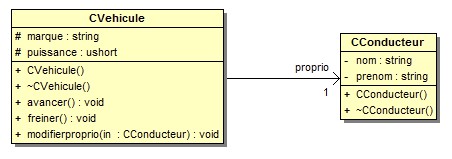
des messages.

Doit-on faire la destruction de l'autoradio dans le destructeur de CVehicule ? Pourquoi ?

2 / 7

III ASSOCIATION

*Rappel :* Lorsqu'un objet a besoin d'un autre objet sans qu'il y ait de dépendance entre eux. *Diagramme de classe :*



+ modifierconducteur(CConducteur \*) : void

*Présentation :*

CVehicule : conducteur sera représenté par un pointeur qui sera initialiser à NULL à la construction d'un véhicule. La méthode modifierconducteur(...) va permettre d'affecter un nouveau conducteur au véhicule.

CConducteur : Cette classe est indépendante d'un véhicule car il peut y avoir un conducteur sans que celui-ci ait une voiture ou que celui-ci soit propriétaire d'une voiture.

*Modifier la classe CVehicule.*

Faites les modifications d'attributs et de méthodes nécessaires.

Avez-vous besoin de détruire l'attribut conducteur une fois le programme terminé ? Pourquoi ?

*Codez la classe CConducteur.*

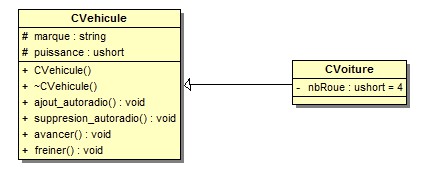
Pensez à soit initialiser les attributs ou à coder des méthodes d'accès à ces attributs !

Testez votre classe en créant seulement un objet v3 de type CVoiture dans le main. Manipulez les méthodes afin de vérifier que l'on peut céder la voiture à quelqu'un d'autre.

IV HÉRITAGE

*Rappel :* La classe dérivée (fille) hérite des membres public et protégés de la classe de base (mère). *Diagramme de classe :*

3 / 7



*Présentation :*

CVehicule : contient deux attributs : marque du véhicule et puissance du véhicule.

Justifiez la visibilité de ces deux attributs :

CVoiture : contient un attribut nommé nbroue car celui-ci est une caractéristique spécifique des voitures.

Comment se fait-il que CVoiture n'ai pas de constructeur et de destructeur ?

*Modifier la classe CVehicule.*

Vous ajouterez les deux attributs présentés plus haut.

Vous ajouterez également 2 méthodes avancer(...) et freiner(...) qui afficheront respectivement les messages suivants : « avance », « freine »

*Codez la classe CVoiture héritant de Cvehicule.*

Pensez bien que celle-ci ne contient quasiment rien (pas de constructeur/destructeur) puisqu'elle hérite de Cvehicule avec seulement une spécialisation.

Testez votre classe en créant seulement un objet **v2** de type **CVoiture** dans le main. Manipulez les méthodes afin de vérifier que l'objet v2 accède à l'ensemble des méthodes de CVehicule.

Dans quelle classe doit-on coder les méthodes permettant d'accéder aux attributs marque et puissance ? Codez-les et utilisez-les !

4 / 7

5 / 7

ANNEXE

V ATTRIBUTS ET MÉTHODES DE CLASSE

Une classe est un type et chaque objet de la classe détient sa propre copie des données membres. Il peut être intéressant que des objets d'une même classe partagent certaines données. On peut utiliser à cette fin des variables globales. Cependant il est généralement plus judicieux et plus élégant que ces données soient déclarées comme faisant partie de la classe : ce sont les attributs de classe. On utilise le mot clé static pour déclarer un attribut de classe.

Revenons sur la classe Etudiant précédemment déclarée. On désire compter le nombre d'instances de   
classe (i.e. le nombre d'objets de cette classe). Pour cela, on va déclarer un attribut de classe compteur à la   
classe Etudiant qui sera incrémenté à la création d'un objet et décrémenté à la destruction. Cet attribut de   
classe doit pouvoir être récupéré depuis l'extérieur mais il ne faut surtout pas qu'on puisse le modifier : on le   
déclare donc en private et on définit une méthode d'accès getNbEtudiants() qui renvoie la valeur du   
compteur. Cette méthode ne doit pas être détenu par chaque instance de la classe Etudiant, c'est une   
méthode partagée par tous les objets de la classe Etudiant : c'est une méthode de classe, déclarée elle   
aussi avec le mot clé static.

// Déclaration de la classe Etudiant class Etudiant {

private:

// Attribut de classe   
static int compteur;

// Attribut d'instance double moyenne;

public:

// Méthode de classe qui renvoie le nombre d'instances de la classe static int getNbEtudiants();

// Constructeur sans paramètre (constructeur par défaut) Etudiant() {

moyenne = 0;

// Incrémentation du compteur d'instances compteur++;

}

// Constructeur avec un paramètre Etudiant(double note) {

moyenne = note;

// Incrémentation du compteur d'instances compteur++;

}

// Destructeur   
~Etudiant() {   
 compteur--; }

// Méthodes d'accès à l'attribut (d'objet) moyenne   
 double getMoyenne() { return moyenne; }   
 void setMoyenne(double note) { moyenne = note; } };

Dans cette déclaration de la classe Etudiant, le membre statique compteur est seulement déclaré et non

6 / 7

défini. Il faut que la définition (et en particulier son initialisation qui est ici essentielle) apparaissent quelque part dans le programme (en général dans le .cpp lié à la classe), et de façon unique :

// Définition de l'attribut de classe

int Etudiant::compteur = 0;

// Définition la méthode de classe

int Etudiant::getNbEtudiants() { return compteur; }

Pour accéder à la méthode de classe getEtudiant dans le programme, on la désignera par Etudiant::getNbEtudiants() :

int main() {

Etudiant bob, john, jenny;

cout << Etudiant::getNbEtudiant() << " ";

Etudiant \* ptr\_etudiant = new Etudiant;

cout << Etudiant::getNbEtudiant() << " ";

delete ptr\_etudiant;

cout << Etudiant::getNbEtudiant() << endl;   
}

Ce petit programme affichera donc sur la sortie standard : 3 4 3.

Attention, une méthode static ne peut accéder qu'aux attributs static de la classe.

Il faut bien distinguer membre d'instance (que l'on désigne souvent abusivement par membre) et membre de classe. Un membre est un attribut (une donnée membre) ou une méthode (une fonction membre).

7 / 7