Conception et Programmation Orientée Objet C++

ECE – INGE2
R. FERCOQ
20/10/2018

Durée : 1H30

Calculatrices non autorisées. Aucun document.

Répondez directement sur l'énoncé dans les cadres prévus.

1. Conception orientée objet et UML (pas de code C++) *6 points (a:2 b:4)*

Le but de cet exercice est la conception d'un **diagramme de classes UML** à partir d'un **CDC**. Le diagramme de classes demandé devra couvrir toutes les classes et tous les attributs mais il n'est **pas** demandé de remplir les parties méthodes. Lisez attentivement le CDC...

Cahier Des Charges

Un célèbre site d'archivage de publications scientifiques nommé arXiv.org regroupe des **articles**. On prétendra qu'une application C++ gère les données de cette application. Le site propose différentes **thématiques** qui ont chacune un **nom** : **physique**, **astrophysique**, **mathématiques**... Chaque **article** est rangé dans une **thématique** unique, quand on a un **article** on doit pouvoir connaître sa **thématique**. On doit aussi pouvoir connaître tous les **articles** d'une **thématique**. Pour chaque **article** on a une **date de publication** dans une **revue** scientifique sérieuse (**Nature**, **Science&Vie Junior**...) et une **date de dépôt** sur le site : ces 2 **dates** sont en général distinctes. Les articles ont un **impact** indiquant leur importance, cet **impact** (qui peut changer) se mesure avec un **nombre de citations** et un **nombre de likes** des scientifiques du domaine (on imagine!). Chaque **article** est écrit par un ou plusieurs **auteurs**. Quand on a un **article** on pourra connaître ses **auteurs**, et réciproquement. Pour chaque **auteur** on connaît son **nom** et son **prénom** et son **université** (**MIT**, **Stanford**, **Harvard**...). En revanche il n'est pas demandé de connaître tous les **auteurs** qui sont dans une certaine **université**. Il n'est pas non plus demandé de pouvoir connaître tous les **articles** publiés dans une certaine **revue**. La seule information pour une **université** est son **nom**. La seule information pour une **revue** est son **nom**. En plus des infos précédentes, un **article** a un **titre** et **un nom de fichier** (le fichier .pdf qui contient l'article proprement dit) qui sont des chaînes de caractère. Les **dates** sont stockées avec 3 entiers **jour**, **mois** et **année**. Pour illustrer, un **exemple d'article** est donné page suivante.

a) *Ranger **chaque bloc en gras** du CDC ci-dessus dans une case du tablau ci-dessous. Un même bloc peut éventuellement apparaître plusieurs fois. Grouper horizontalement une classe avec ses attributs et les objets qui peuvent éventuellement y correspondre.* **Crochets : [facultatif]**

Classes	attributs	objets
Article	date publication / date dépôt titre / nom fichier / [impact] [auteurs] / [revue] / [thématique]	[Exemple d'article « Induced parthenogenesis... »]
Thématique	nom (ou « nom thématique ») [articles]	physique / astrophysique mathématiques / biologie
Revue	nom (ou « nom revue ») [articles]	Nature Science&Vie Junior
Date	jour / mois / année	
Impact	nombre de citations nombre de likes	
Auteur	nom / prénom [université] [articles]	
Université	nom (ou « nom université ») [auteurs]	MIT / Stanford / Harvard

Exemple d'article : « Induced parthenogenesis in xenozoans » par Cuiping Chiang (MIT) et Casey Kriegeskorte (Harvard), publié le 01/10/2016 dans Nature, déposé sur le site arXiv.org le 21/11/2016, thématique biologie. Impact de {12 citations et 86 likes}. Fichier " bio02812.pdf "

b) *Faites sur cette page 2 diagrammes de classes, en bas le diagramme de classes des types valeurs, en haut le diagramme de classe des types entités. On ne demande **pas** de justification pour déterminer quels sont les « types valeurs » et quels sont les « types entités ».*
*Il n'est **pas** demandé de faire apparaître les méthodes. **Toutes** les autres infos du diagramme de classes doivent êtres présentes en respect de la norme de notation UML. Pour rappel les types de base en notation UML sont Integer, Real, Boolean, String. Prévoir des boites **larges**.*

Diagramme de classes des types entités

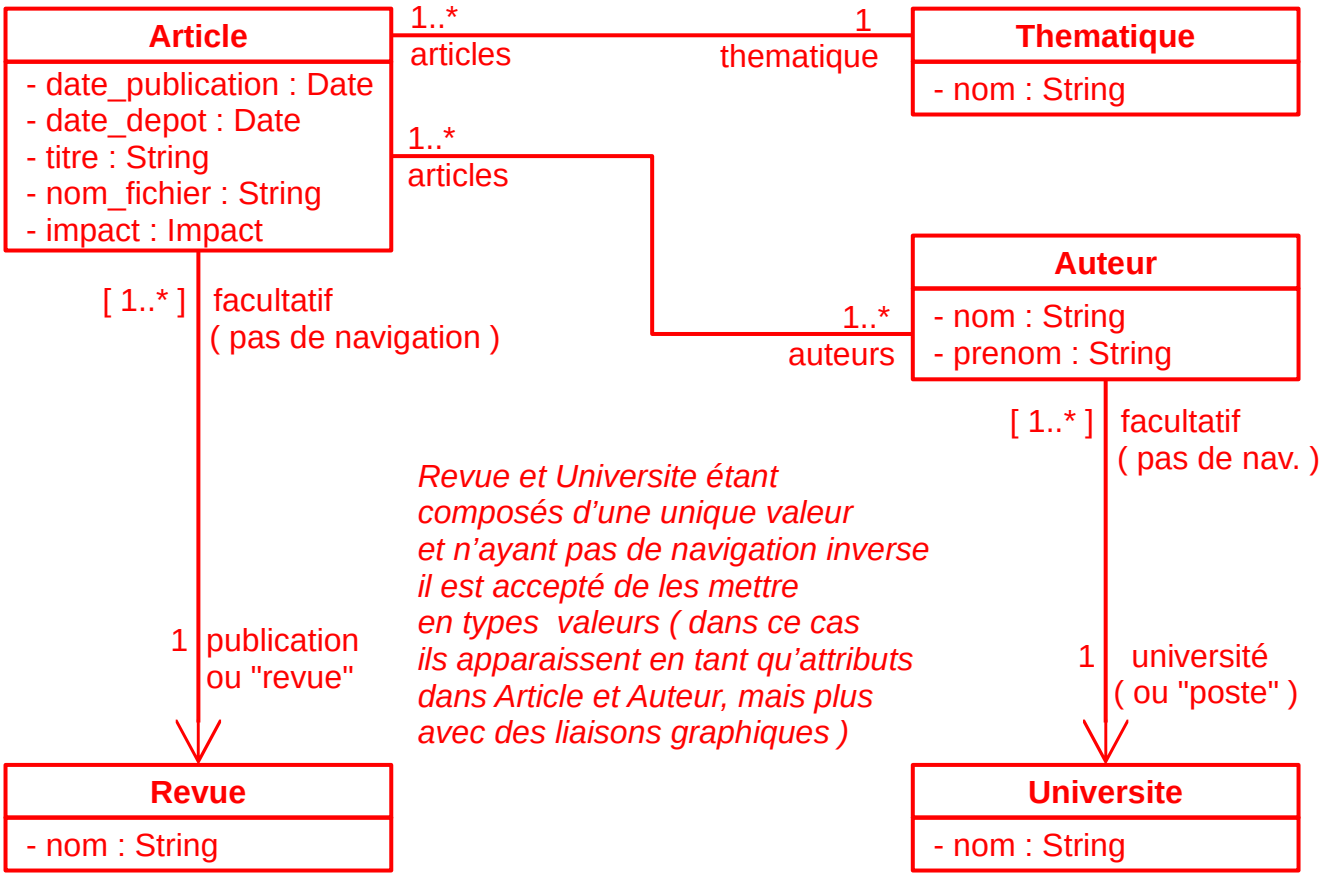

Diagramme de classes des types valeurs

La notion de type valeur n'est pas trop discutable pour Date et Impact : quand 2 Articles ont la même Date ou le même Impact c'est une coincidence,ni une relation entre entités ni un partage. 1 point pour avoir fait la distinction. Si Date et Impact apparaissent en tant que types entités ci dessus et qu'ils sont correctement reliés graphiquement avec des compositions et que le reste est correct 3/4

2. QCM questions de cours *6 points (0.5 point par question)*

***Entourer** une seule bonne réponse (entourer le 1. 2. 3. ou 4.) : 0.5 point*
Plusieurs réponses ou mauvaise réponse ou baratin : 0 point
Pas de stylo rouge SVP. Pas de justification demandée.

Vert : réponse préférée
Bleu : réponse acceptable

a) Dans un programme de dessin on veut des carrés, des ronds, des rectangles...
Un développeur pense à faire hériter une classe Carré d'une classe Rectangle
1. Impossible, il faut que **2 classes filles** au moins héritent d'une classe parente
2. Bof, un Carré **est un** Rectangle mais ce n'est pas un rectangle avec **un truc en plus**
3. C'est l'inverse, Rectangle hérite de Carré auquel **il ajoute** un 2ème attribut de taille
4. Un Carré **est un** (cas spécial) de Rectangle : utilisation typique et correcte d'héritage

b) Dans un affichage complexe avec plusieurs << … << … << chaînés, le bloc **`std::endl`**
1. Indique la fin de la chaîne d'affichage std::endl (end of linked), obligatoire à la fin
2. Indique la fin de la chaîne d'affichage std::endl (end of linked), facultatif à la fin
3. Indique un retour à la ligne dans la console (end of line), facultatif, possible au milieu
4. Indique un retour à la ligne dans la console (end of line), facultatif mais seulement à la fin

c) Grâce à l'encapsulation des attributs avec private, un programme C++ est à l'abri
1. D'abîmer les données des objets depuis le code client même en cas d'accès [i] qui déborde
2. De toute tentative de mauvais usage des objets : les objets sont blindés, sauf [i] qui déborde
3. De pouvoir être espionné, le code de bibliothèque est donc à l'abri d'un code « pirate »
4. D'avoir un code client qui dépend des détails de stockage qui peuvent varier

d) Un appel de méthode de classe
1. Part d'un objet unique instance de cette classe : cible de l'appel, paramètre implicite this
2. Part d'un ou deux objets de cette même classe : cible(s) de l'appel, implicite(s) this et that
3. Part d'autant d'objet qu'on veut si la cible de l'appel est un vecteur des objets de la classe : vec.afficher( ) ; et dans la méthode on a this[i] pour accéder à chacun
4. Part de zéro ou un objet : dans le 1er cas le paramètre implicite this vaut nullptr (à tester)

e) On n'a pas d'autre choix que de faire new et delete quand
1. On ne veut pas juste déclarer un objet mais on veut le créer
2. On a un objet qui ne doit pas automatiquement être détruit à la fin du scope (delete + tard)
3. On a un objet qui doit automatiquement être détruit à la fin du scope (destructeur→delete)
4. On a une quantité variable de choses à stocker : l'allocation dynamique est la seule solution

f) Dans un programme graphique qui utilise une sortie fichier SVG on veut pouvoir positionner un carré soit en coordonnées cartésiennes (x,y) soit en coordonnées polaires (r, theta). On a les 2 méthodes suivantes dans la classe Svgfile :

```
void Svgfile::addSquare(double x, double y, double size);
void Svgfile::addSquare(double r, double theta, double size);
```

1. Impossible parce que les 2 versions surchargées ont les mêmes types de paramètres
2. Impossible à moins de mettre les paramètres surchargés en dernier (après size)
3. C'est possible avec la surcharge mais il faut se placer dans le bon scope avec ::
4. C'est possible avec la surcharge mais il faut faire l'appel avec les bons noms en paramètres

g) Un itérateur est
1. Une surcharge de l'opérateur ++ pour des types non numériques (++monElephant)
2. Une syntaxe moderne du C++ pour répéter des opérations identiques for(auto x: Voiture)...
3. Un objet « pointeur de case » pour des classes conteneurs d'objets
4. Un objet « copieur de d'attributs » pour des objets qui utilisent le constructeur par copie

h) Ce code écrit par un expert (voir page suivante)

```
void test(int &x, int &y)
{
    x=y;
}
```

```
int main()
{
    int x = 4;
    int y = 3;
    test(&x, &y);
    y = 5;
    std::cout << x << " " << y;
```

1. Ne compile pas : les paramètres attendus sont des int et l'appel envoie des int*
2. Compile mais l'appel ne sert à rien parce que les références sont copiées, affiche 4 5
3. Compile et marche par référence : x appelant prend bien la valeur 3 de y, affiche 3 5
4. Compile et marche par référence : la référence x devient la même que y , affiche 5 5

i) Dans un code source C++ de simulateur de conduite on trouve dans le main la ligne suivante :

```
bolide.setVitesse(bolide.getVitesse() + 1);
```

1. C'est obligatoire, pour modifier l'objet on est obligé de passer par des accesseurs get/set
2. C'est maladroit, on pourrait utiliser une méthode accelerer : la coder si possible
3. C'est maladroit, le private de la classe rend les données inaccessibles : utiliser une struct
4. C'est interdit, Voiture est un type valeur. Construire une copie avec une nouvelle vitesse

j) Une classe Quelconque est déclarée comme suit (code abrégé à gauche) et on veut l'utiliser comme suit (code à droite) pour mettre des valeurs dans les attributs alpha et beta :

```
class Quelconque
{
    private :
        int alpha;
        float beta;
    public : // méthodes etc …
```

```
int main()
{
    Quelconque::alpha = 3;
    Quelconque::beta = 3.14;
```

1. N'importe quoi ! Il n'y a pas d'objet : on ne met pas des valeurs dans une classe
2. Non à cause du private : il faut utiliser des accesseurs Quelconque.setAlpha(3); etc...
3. Non à cause du private : il faut utiliser des accesseurs Quelconque::setAlpha(3); etc...
4. Aucun problème ! Grâce à :: (résolution de portée) on accède directement à alpha et beta

k) On suppose qu'on dispose d'une classe Date déjà correctement implémentée. Le code client :

```
int main()
{
    Date noel{"25/12/2018"}, unAn{"0/0/1"}; // ça marche
    Date noelSuivant = noel + unAn; // ça peut marcher ou pas ?
```

1. N'importe quoi ! Date n'est pas un type numérique, operator+ ne peut pas marcher
2. Oui à condition de transformer explicitement la std::string en trois attributs entiers, par exemple dans le constructeur, puis de coder explicitement operator+ pour Date
3. Oui si en interne operator+ de Date utilise explicitement operator+ de std::string
4. Aucun problème ! La conversion vers le type string est implicite, rien à faire

l) Le code suivant dimensionne un vecteur à 2 dimensions de mots dans le constructeur d'une classe matrice, le développeur de la classe n'a pas encore codé de destructeur...

```
class Matrice
{
    private :
        std::vector<std::vector<std::string>> o_o;
        ...
    public :
        Matrice(int largeur, int hauteur);
        ... // Pas de destructeur
```

1. N'importe quoi ! Le destructeur doit être codé pour libérer les mots, les vecteurs lignes, et le vecteur colonne, sinon ça plante à la sortie de scope
2. Mauvais, ça ne plante pas mais le vecteur dans l'objet n'est jamais libéré : fuite mémoire
3. Pas terrible, si le code client oublie de faire delete sur un objet matrice alors fuite mémoire
4. Aucun problème ! Le destructeur implicite convient bien, rien à faire

Codes à corriger *total 8 points (exos 3. à 7. 1pt chacun, exo 8 3pts)*

Pour chacun des exercices qui suit il peut y avoir une ou plusieurs choses à corriger. Si vous ne corrigez qu'une partie des problème vous n'aurez qu'une partie des points. Si vous corrigez des aspects qui ne sont pas des problèmes mais que le code résultant fonctionne aussi bien, vous ne gagnez pas de point, vous n'en perdez pas non plus. Si vous proposez des corrections qui introduisent des erreurs ou des problèmes supplémentaires vous perdez des points (sur l'exo).

Pour chaque correction proposée, ***rayer directement*** *l'erreur et corriger sur le code si possible.* ***Ne pas utiliser de stylo rouge SVP****. Justifier* ***très brièvement*** *chaque correction dans les cadres.*

*Dans les exercices qui suivent on appelle « anomalie » un aspect du code qui ne respecte pas les consignes explicites données en cours. C'est différent d'une erreur : l'anomalie n'empêche pas le code de fonctionner, mais il est demandé de la corriger, et d'****expliquer le rôle*** *de la consigne.*

3. ***Le code suivant ne compile pas. Expliquer et corriger. (1pt)***

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

Il manque l'opérateur de résolution de portée std:: pour cout et pour endl (cout et endl sont dans la bib. Standard, mettre std:: etc accepté)
« using namespace std; » marche mais n'est pas conseillé : 0.5 point seulement

4. ***Le code suivant compile, s'exécute, mais ne fait pas ce qu'on veut. Expliquer et corriger. (1pt)***

```
#include <iostream>                          &
#include <vector>

void doubler(std::vector<double> vec)
{
    for (size_t i=0; i<vec.size(); ++i)
        vec[i] = 2 * vec[i];
}

int main()
{
    std::vector<double> vec{3.1, 4.2, 5.3};

    doubler(vec);

    for (size_t i=0; i<vec.size(); ++i)
        std::cout << vec[i] << " ";

    return 0;
}
```

Affiche
3.1 4.2 5.3

Au lieu de ce qu'on veut
6.2 8.4 10.6

Le paramètre reçoit une copie, c'est la copie qui est modifiée

Il suffit de passer par référence &

Alternativement « par adresse » (rock'n roll, déréférencer...)

Alternativement par retour valeur vec = doubler(vec) (changer la fonction : return vec etc)

5. ***Le code suivant compile, s'exécute, fait ce qu'on veut, mais a une ou des anomalie(s). (1pt)***

```
#include <iostream>
#include <vector>
#include <string>          const

void afficher(std::vector<std::string>& words)
{
    for (size_t i=0; i<words.size(); ++i)
        if ( words[i]=="beau" )
            std::cout << "beautiful" << " ";
        else
            std::cout << words[i] << " ";
}
```

```
int main()
{
 std::vector<std::string>
   mots{"il",
        "fait",
        "beau"};
  afficher(mots);
  return 0;
}
```

La seule anomalie que je vois ici c'est que ce sous programme d'affichage n'a pas vocation à modifier les valeurs qu'on lui confie, le paramètre doit donc être const (si le == est par accident un = le problème sera signalé)

6. ***Le code suivant compile, commence à s'exécuter, mais plante. Expliquer et corriger. (1pt)***
Le problème à régler n'est pas l'absence de blindage de la saisie, saisir 6 fait planter. Ce que ça devrait afficher : fibo(0)=0 fibo(1)=1 fibo(2)=1 fibo(3)=2 fibo(4)=3 fibo(5)=5 fibo(6)=8

```
#include <iostream>
#include <vector>

int main()
{
    std::vector<int> fibo;
    std::cout << "Fibonacci au rang : ";
    int n;
    std::cin >> n;

    fibo[0] = 0;    fibo[1] = 1;
    for (int i=2; i<=n; ++i)
        fibo[i] = fibo[i-1] + fibo[i-2];

    for (int i=0; i<=n; ++i)
        std::cout << "fibo(" << i << ")="
                  << fibo[i] << "   ";

    return 0;
}
```

Les cases [i] du vecteur n'existent pas ! Avant de parler d'une case [i] il faut la créer
→ Soit en initialisant le vecteur à la bonne taille (après la saisie de n !)
`std::vector<int> fibo(n+1);`
→ Soit en ajoutant les cases avec push_back
```
fibo.push_back(0);
fibo.push_back(1);
for (…)
    fibo.push_back(
            fibo[i-1]+fibo[i-2]);
```
Le reste est correct
(en particulier l'affichage peut faire [i] maintenant que ces cases existent)

7. ***Le code qui suit est un essais de « blindage » de la saisie de l'exo 6. Ça ne compile pas. (1pt)***

```
    /// Saisie blindée
    do
    {
        int n;
        std::cin >> n;
        if ( n<2 )
            std::cout << "n>=2 SVP : ";
    } while (n<2);
```

La variable int n est locale au scope de la boucle, dès l'accolade fermante elle n'existe plus et donc on ne peut plus en parler (ni dans le test du while, ni l'utiliser après). Il faut la déclarer en dehors du bloc : avant le do !

8. ***Le code suivant compile, s'exécute, ne fait pas tout ce qu'on veut et plante après. (3 pts)***
Corriger les problèmes, les erreurs, les anomalies. A cet exo. pas d'explications demandées

```
Au fait (importance = 3) : Le frigo est vide
Et puis (importance = 5) : La box est en panne
On a faim (importance = 3) : Le frigo est vide
```

⇐ Ce que ça affiche avant de planter

Ce qu'on veut ici : 7

message.h

```
#include <string>

class message
{
    private :
        std::string texte;       // Le texte du message
        unsigned int importance; // L'importance du message

    public :
        message(std::string tex, unsigned int imp);
        void afficher(std::string commentaire, message m);
        int getImportance(message m);
        void setImportance(int importance); //besoin de setter !
};
```

m_

const

C'est un contresens de demander l'objet en paramètre : l'objet principal est la cible de l'appel (this …)

// On peut barrer le getter au lieu de le corriger, le client a besoin d'un setter !

message.cpp

```
#include "message.h"
#include <iostream>

message::message(std::string tex, unsigned int imp)
{
    texte = tex;
    importance = imp;
}

void message::afficher(std::string commentaire, message m)
{
    std::cout << commentaire << " (importance = "
              << m.importance << ") : "
              << m.texte << std::endl;
}

int message::getImportance(message m)
{
    return m.importance;
}
```

On préfère une liste d'initialisation mais le code de départ est correct

C'est un contresens de demander l'objet en paramètre : l'objet principal est la cible de l'appel (this …)

const

m_

// On peut barrer le getter au lieu de le corriger, le client a besoin d'un setter !

const

```
//besoin de setter !
void setImportance(int importance)
{
    m_importance = importance;
}
```

---

main.cpp

```
#include "message.h"
#include <iostream>

int main()
{
    message A{"Le frigo est vide",   3};
    message B{"La box est en panne", 5};

    A.afficher("Au fait", A);
    B.afficher("Et puis", B);

    A.setImportance(7);  // Corriger le = = par = ne suffit pas !
    A.getImportance(A) == 7;
    A.afficher("On a faim", A);

    delete &A;
    delete &B;

    return 0;
}
```

Pas de new => pas de delete (le plantage vient de là)
Les objets sont « automatiques » leur libération est automatique à la fin du scope, on ne doit/peut pas forcer leur libération