



C++-Module 04

Polymorphisme de sous-type, classes abstraites, interfaces

Résumé : Ce document contient le sujet du module 04 des modules C++.

Version: 10

Contenu

je	Règles générales	2
yl	Exercice 00 : Polymorphisme	4
III	Exercice 01 : Je ne veux pas mettre le feu au monde	6
IV	Exercice 02 : classe abstraite	8
V	Exercice 03: Interface et récapitulatif	9

Chapitre I

Règles générales


Pour les modules C++, vous n'utiliserez et n'apprendrez que C++98. L'objectif est que vous appreniez les bases d'un langage de programmation orienté objet. Nous savons que le C++ moderne est très différent à bien des égards, donc si vous voulez devenir un développeur C++ compétent, vous aurez besoin du C++ standard moderne plus tard. Ce sera le point de départ de votre parcours C++ à vous d'aller plus loin après le 42 Common Core !

- Toute fonction implémentée dans un en-tête (sauf dans le cas des modèles), et toute en-tête non protégé signifie 0 à l'exercice.
- Chaque sortie va à la sortie standard et se terminera par une nouvelle ligne, sauf indication contraire.
- Les noms de fichiers imposés doivent être suivis à la lettre, ainsi que les noms de classes, fonction les noms de fonction et les noms de méthode.
- N'oubliez pas : vous codez en C++ maintenant, et non plus en C. Donc:
 - Les fonctions suivantes sont INTERDITES, et leur utilisation sera punie par 0, aucune question posée : `*alloc`, `*printf` et `free`.
 - Vous êtes autorisé à utiliser tout ce qui se trouve dans la bibliothèque standard. CEPENDANT, il serait judicieux d'essayer d'utiliser les versions C++-ish des fonctions auxquelles vous êtes habitué en C, au lieu de vous en tenir à ce que vous savez, c'est un nouveau langage après tout. Et NON, vous n'êtes pas autorisé à utiliser la STL tant que vous n'êtes pas censé le faire (c'est-à-dire jusqu'au module 08). Cela signifie pas de vecteurs/listes/cartes/etc... ou tout ce qui nécessite un `<algorithme>` d'inclusion jusque-là.
- En fait, l'utilisation de toute fonction ou mécanique explicitement interdite sera punie d'un 0, sans poser de questions.
- Notez également que sauf indication contraire, les mots clés C++ "using namespace" et "friend" sont interdits. Leur utilisation sera sanctionnée par un -42, sans poser de questions.
- Les fichiers associés à une classe seront toujours `ClassName.hpp` et `ClassName.cpp`, sauf indication contraire.
- Les répertoires de restitution sont `ex00/`, `ex01/`, ... , `ex/`.

- Vous devez lire attentivement les exemples. Ils peuvent contenir des exigences qui ne sont pas évidentes dans la description de l'exercice.
- Puisque vous êtes autorisé à utiliser les outils C++ que vous avez appris depuis le début, vous n'êtes pas autorisé à utiliser une bibliothèque externe. Et avant de demander, cela signifie également pas de C++11 et de dérivés, ni de Boost ou quoi que ce soit d'autre.
- Vous devrez peut-être rendre un nombre important de cours. Cela peut sembler fastidieux à moins que vous ne puissiez créer un script avec votre éditeur de texte préféré.
- Lisez **ENTIÈREMENT** chaque exercice avant de le commencer! Fais le.
- Le compilateur à utiliser est c++.
- Votre code doit être compilé avec les drapeaux suivants : -Wall -Wextra -Werror.
- Chacune de vos inclusions doit pouvoir être incluse indépendamment des autres.
Les inclusions doivent contenir toutes les autres inclusions dont elles dépendent.
- Au cas où vous vous poseriez la question, aucun style de codage n'est appliqué en C++. Vous pouvez utiliser n'importe quel style que vous aimez, sans aucune restriction. Mais rappelez-vous qu'un code que votre pair-évaluateur ne peut pas lire est un code qu'il ne peut pas noter.
- Choses importantes maintenant: vous ne serez PAS noté par un programme, sauf si cela est explicitement indiqué dans le sujet. Par conséquent, vous bénéficiez d'une certaine liberté dans la façon dont vous choisissez de faire les exercices. Cependant, soyez conscient des contraintes de chaque exercice, et NE soyez PAS paresseux, vous manquerez BEAUCOUP de ce qu'ils ont à offrir
- Ce n'est pas un problème d'avoir des fichiers superflus dans ce que vous remettez, vous pouvez choisir de séparer votre code dans plus de fichiers que ce qui vous est demandé. N'hésitez pas, tant que le résultat n'est pas noté par un programme.
- Même si le sujet d'un exercice est court, cela vaut la peine d'y consacrer du temps pour être sûr de bien comprendre ce qu'on attend de vous, et que vous l'avez fait de la meilleure façon possible.
- Par Odin, par Thor ! Utilise ton cerveau!!!

Chapitre II

Exercice 00 : Polymorphisme

	Exercice : 00
Polymorphisme	
Répertoire de rendu : ex00/	
Fichiers à rendre : Makefile, main.cpp, Tous les autres fichiers dont vous avez	
besoin Fonctions interdites : Aucune	

Pour chaque exercice, votre principal doit tout tester.

Les constructeurs et les destructeurs de chaque classe doivent avoir une sortie spécifique.

Créez une classe de base Animal simple et complète.

La classe animale a un attribut protégé :

- std::type de chaîne ;

Créez une classe Dog qui hérite de Animal.

Créez une classe Cat qui hérite de Animal. (pour la classe

animal le type peut être laissé vide ou mis à n'importe quelle valeur).

Chaque classe doit mettre son nom dans le champ Type par exemple : Le type de classe Chien doit être initialisé en tant que "Chien".

Chaque animal doit pouvoir utiliser la méthode makeSound().

Cette méthode affichera un message approprié sur les sorties standard en fonction de la classe.

```
int main() {

    const Animal* meta = new Animal(); const
    Animal* j = new Chien(); const Animal* i = new
    Chat();

    std::cout << j->getType() << << std::endl; std::cout << i->getType()
    << << std::endl; je->faitSon(); // produira le son d'un chien; meta-
    >makeSound();
```


```
...  
}
```

Cela devrait produire le makeSound spécifique de la classe Dog and cat, pas l'animal un.

Pour être sûr, vous allez créer une classe WrongCat qui hérite d'une classe WrongAnimal qui produira le WrongAnimal makeSound() lors d'un test dans les mêmes conditions.

Chapitre III

Exercice 01 : Je ne veux pas mettre le feu au monde

	Exercice : 01
Je ne veux pas mettre le feu au monde	
Répertoire de rendu : ex01/	
Fichiers à rendre : Makefile, main.cpp, plus les fichiers nécessaires à vos tests Fonctions interdites : Aucune	

Vous réutiliserez les classes Ex00.

Créez une classe appelée Brain.

Brain contiendra un tableau de 100 std::string appelé ideas Maintenant,

Dog et cat auront un attribut Brain* privé.



Tous les animaux n'ont pas de cerveau !

Lors de la construction, Dog et Cat initialiseront leur Brain* avec un nouveau Brain();

Lors de la destruction, le chien et le chat supprimeront leur cerveau.

Votre principal créera et remplira un tableau d'animaux dont la moitié sera un chien et l'autre moitié sera un chat.

Avant de quitter, votre main bouclera sur ce tableau et supprimera tous les animaux.
Vous devez supprimer directement Chat et Chien en tant qu'Animal.


Une copie d'un chat ou d'un chien doit être "profond".
Votre test devrait montrer que les copies sont profondes!

Les constructeurs et les destructeurs de chaque classe doivent avoir une sortie spécifique.
Les destructeurs appropriés doivent être appelés.

```
int main() {  
  
    const Animal* j = new Chien(); const  
    Animal* i = new Chat();  
  
    delete j;//ne devrait pas créer de fuite delete i;  
  
}
```


Chapitre IV

Exercice 02 : classe abstraite

	Exercice : 02
Ce code est impur. PURIFIEZ-LE !	
Répertoire de restitution : ex02/	
Fichiers à rendre : Makefile, main.cpp, plus les fichiers nécessaires Fonctions	
interdites : Aucune	

Un animal général n'a pas de sens après tout.

Par exemple, il ne fait aucun son !


Pour éviter toute erreur future, la classe animale par défaut ne doit pas être instanciable.

Corrigez la classe Animal afin que personne ne l'instancie par erreur.

Le reste devrait fonctionner comme avant.

Chapitre V

Exercice 03 : Interface & récapitulatif

	Exercice : 03
	Interface & récapitulatif
	Répertoire de rendu : ex03/
	Fichiers à rendre : Makefile, main.cpp, plus les fichiers nécessaires Fonctions
	interdites : Aucune

Il n'y a pas d'interface en C++98 (pas même en C++20) mais il est courant d'appeler la classe abstraite pure Interface. Alors pour ce dernier exercice essayons les interfaces et récapitulons tout !

Complétez la définition de la classe AMateria suivante et implémentez les besoins fonctions membres saires.

```

classe AMateria
{
    protégé : [...]

    public:
        AMateria(std::string const & type); [...]

        std ::string const & getType() const; // Retourne le type de matière

        virtual AMateria* clone() const = 0; utilisation du
        vide virtuel (ICharacter& cible);
};

```

Créer le béton Materias Ice and Cure. Leur type sera leur nom dans minuscule ("ice" pour Ice, etc...).

Leur méthode clone() renverra bien sûr une nouvelle instance du type réel de Materia.

Concernant la méthode `use(ICharacter&)`, elle affichera :

- `Glacé`: `"* tire un éclair de glace sur NOM *"`
- `Guérison`: `"* guérit les blessures de NOM *"`

(Bien sûr, remplacez `NOM` par le nom du Personnage donné en paramètre.)



Lors de l'attribution d'une `Materia` à une autre, copier le type ne fait pas sens...

Créez la classe `Character`, qui implémentera l'interface suivante :

```

classe ICaractère
{
    public:
        virtual ~ICharacter() {} virtual
        std::string const & getName() const = 0; vide virtuel equip(AMateria*
            m) = 0; vide virtuel unequip(int idx) = 0; utilisation du vide virtuel
            (int idx, ICharacter & cible) = 0;
};

```

Le Personnage possède un inventaire de 4 `Materia` au maximum, vide au départ. Il équipera la `Materia` dans les emplacements 0 à 3, dans cet ordre.

Dans le cas où nous essayons d'équiper une `Materia` dans un inventaire complet, ou d'utiliser/déséquiper une inexistante `Materia`, ne fais rien.

La méthode de déséquipement ne doit PAS supprimer `Materia`!

La méthode `use(int, ICharacter&)` devra utiliser la `Materia` à l'emplacement `idx`, et passer la cible en paramètre à la méthode `AMateria::use`.



Bien sûr, vous devrez être en mesure de prendre en charge N'IMPORTE QUELLE `AMateria` dans l'inventaire d'un personnage.

Votre personnage doit avoir un constructeur prenant son nom en paramètre. La copie ou l'attribution d'un personnage doit être profonde, bien sûr. L'ancienne `Materia` d'un Personnage doit être supprimée. Idem lors de la destruction d'un Personnage.

Créez la classe `MateriaSource`, qui devra implémenter l'interface suivante :

```
classe IMateriaSource
{
    public:
        virtual ~IMateriaSource() {} virtual void
        learnMateria(AMateria*) = 0; virtual AMateria*
        createMateria(std::string const & type) = 0;
};
```

`learnMateria` doit copier la `Materia` passée en paramètre, et la stocker en mémoire pour la cloner plus tard. De la même manière que pour le `Personnage`, la `Source` peut contenir plusieurs `Materia` qui ne sont pas forcément uniques.

`createMateria(std::string const &)` renverra une nouvelle `Materia`, qui sera une copie de la `Materia` (précédemment apprise par la `Source`) dont le type est égal au paramètre. Renvoie 0 si le type est inconnu.

En un mot, votre `Source` doit être capable d'apprendre des "modèles" de `Materia` et de les recréer à la demande. Vous pourrez alors créer une `Materia` sans connaître son "vrai" type, juste une chaîne d'identifiant.

Comme d'habitude, voici une main de test que vous devrez améliorer :

```
int main() {  
  
    IMateriaSource* src = new MateriaSource(); src-  
    >learnMateria(nouvelle glace()); src->learnMateria(nouveau  
    remède());  
  
    ICaractère* moi = nouveau Caractère("moi");  
  
    AMatéria* tmp;   
    tmp = src->createMateria("ice"); moi->  
    équiper(tmp); tmp = src->createMateria("cure");  
    moi-> équiper(tmp);  
  
    ICaractère* bob = nouveau Caractère("bob");  
  
    moi->use(0, *bob); moi-  
    >use(1, *bob);  
  
    supprimer Bob;   
    supprime moi;  
    supprimer source;  
  
    renvoie 0;  
}
```

Sortir:

```
$> clang++ -W -Wall -Werror *.cpp $> ./a.out |  
cat -e * tire un éclair de glace sur bob *$ *  
soigne les blessures de bob *$
```