INF1010

Programmation Orientée-Objet

Travail pratique #3
Héritage

Objectifs :	Permettre à l'étudiant de se familiariser le concept d'héritage.
Remise du travail :	Mardi 25 Octobre 2016, 8h
Références :	Notes de cours sur Moodle & Chapitre 8 du livre Big C++ 2e éd.
Documents à remettre :	Les fichiers .cpp et .h complétés réunis sous la forme d'une archive au format .zip.
Directives :	Directives de remise des Travaux pratiques sur Moodle Les en-têtes (fichiers, fonctions) et les commentaires sont obligatoires. Les travaux dirigés s'effectuent obligatoirement en équipe de deux personnes faisant partie du même groupe. Veuillez suivre le guide de codage

Informations préalables

Compilation : le cas des inclusions (ou dépendances) circulaires. On appelle inclusion circulaire le fait que deux « points .h » s'incluent mutuellement. L'usage simple des gardes de compilation ne suffit plus à protéger le programmeur d'une erreur de compilation. Les gardes de compilation (***#ifndef***, ***#define*** et ***#endif***) empêchent qu'un même fichier soit inclus plusieurs fois,mais le problème des dépendances circulaires réside dans le fait que chacun des deux fichiers a besoin des déclarations présentes dans l'autre. En particulier, si on imagine deux classes ayant chacune une méthode prenant l'autre type de classe en argument, on a alors un cas de dépendances circulaires. Pour résoudre ce problème, on utilise des déclarations anticipées. C'est-à-dire que

l'on prévient le compilateur de l'existence d'une classe via la déclaration *class maClasse;* on définira la seconde classe après ces directives (dans le même fichier). Et on fera l'opération inverse dans le fichier contenant la déclaration de *maClasse*.

Notez, toutefois, que les inclusions circulaires peuvent ralentir la compilation et doivent être utilisées avec parcimonie.

Par ailleurs, afin de compléter ce TP vous devrez utiliser des *static_cast.* Vous pouvez consulter le lien suivant pour vous documenter sur cette fonction :
http://en.cppreference.com/w/cpp/language/static_cast.

Enfin, certaines fonctionnalités du TP requièrent de l'aléatoire. Pour cela, veuillez utiliser les fonctions *rand* et *srand*. En particulier, soyez très prudent avec l'utilisation de *srand,* une mauvaise utilisation provoquera un résultat déterministe !

Attention, ce TP porte uniquement sur l'héritage, il ne faut pas utiliser le mot clé *virtual*.

Travail à réaliser

Le travail présent a pour but d'ajouter de nouvelles fonctionnalités au jeu Polyland. Les mécanismes que nous allons ajouter sont inspirées d'un célèbre jeu, lui aussi basé sur des combats entre des créatures.

L'ajout de ces fonctionnalités permet d'introduire une notion fondamentale de la programmation orientée objet : *l'héritage*. Une classe représente toujours une idée, un concept or il se peut que deux classes soient deux déclinaisons d'un même concept. Il est alors pertinent de faire intervenir l'héritage plutôt que d'implémenter les mêmes méthodes. Cette notion va de pair avec la notion de polymorphisme que vous verrez au cours des prochaines semaines. Gardez en tête que l'héritage se caractérise par une relation « est un ».

Suite à une réunion avec les responsables design et gameplay, il vous est demandé d'augmenter le nombre d'objets utilisables et de permettre que certaines attaques modifient l'état de votre créature. Ainsi, il sera possible d'endormir une créature adverse… ou que l'adversaire endorme votre créature !

ATTENTION : Tout au long du TP, assurez-vous d'utiliser les opérateurs sur les objets et non sur leurs pointeurs ! Vous devez donc déréférencer les pointeurs si nécessaires.

ATTENTION : Il est fortement recommandé d'utiliser les fichiers fournis, plutôt que de continuer avec vos fichiers du TP2.

ATTENTION : Sauf mention explicite du contraire, c'est à vous de déterminer la visibilité de vos attributs (protected, private, public).

ATTENTION : Les fichiers fournis ont été modifiés afin de faciliter l'utilisation de l'héritage, toutefois les techniques employées ne sont pas nécessairement à employer dans vos travaux futurs. Vous étudierez en cours et lors du prochain TP les bonnes manières de faire. En particulier, l'utilisation de cast est normalement à éviter autant que possible. Nous ignorons cette règle durant ce TP dans un but pédagogique.

ATTENTION : L'un des principes de l'héritage étant de limiter la duplication du code, pensez à utiliser au maximum les méthodes des classes mères (ou de base).

En l'absence de précision, vous devez utiliser les valeurs par défaut pour les types suivant :

- **String, une chaîne vide**
- **Pointeur, nullptr**
- **Objet, son constructeur par défaut**

Aide : dans certaines classes il vous est demandé d'appeler l'opérator<< de la classe mère, pour ce faire vous pouvez étudier les fonctions définies au début du main.

Classe *OutilScientifique*

Créez une classe *OutilScientifique qui représente les différents outils que pourra utiliser un professeur*. Il s'agit d'une classe de base qui représente les outils dans leur généralité. Cette classe est fournie.

Classe *Professeur*

Cette classe dérive de la classe dresseur.

Cette classe contient l'attribut suivant :

- Un pointeur vers un OutilScientifique

Les méthodes suivantes doivent être implémentées :

- Constructeur par paramètres semblable à celui de Dresseur
- Un constructeur par copie
- Un destructeur
- Les méthodes de modification de l'attributs
- Une méthode soigner qui rétablit les points de vie et l'énergie de la créature (passée en argument) à leur valeur maximale
- Un opérateur égal
- Une méthode *utiliserOutil* qui s'applique à une créature

Classe Etat*Creature*

Cette classe sert de classe de base pour les différents états dans lesquels une créature peut être. Notez l'absence de constructeur par défaut.

Cette classe contient les attributs suivants :

- Un nom de l'état (string) sert à donner une indication sur la gravité de l'état. Vous pouvez choisir un nom qui vous semble approprié.
- Une durée (entier) qui correspond à la durée pendant laquelle une créature va rester dans cet état. Par défaut, la durée est 0.
- Un attribut type de TypeEtat (énumération dans le fichier TypeEtat.h), il devra être initialisé à la valeur TypeEtat_normal dans tous les constructeurs.

Les méthodes suivantes doivent être implémentées :

- Un constructeur par paramètre qui prend comme argument le nom de l'état,
- Un constructeur par paramètre qui prend comme argument le nom de l'état et une durée
- Une méthode appliquerEtat qui sera surchargée dans les classes dérivées
- Une méthode peutAttaquer qui détermine si une créature peut attaquer à ce tour. Cette méthode sera surchargée, dans la classe de base elle doit renvoyer la valeur « true »
- Une méthode estFini qui caractérise si la créature peut revenir à un état normal. Cette méthode sera surchargée par les classes dérivées. Dans la classe de base, elle renvoie false
- Deux accesseurs pour le nom et l'attribut type

La fonction friend : operateur<< (voir image à la fin du document), pour afficher le nom de l'état.

Classe *EtatConfus*

Cette classe hérite de EtatCreature.

Les méthodes suivantes doivent être implémentées :

- Un constructeur qui prend le nom en argument
- Un constructeur qui prend le nom et la durée de l'état en argument

La fonction globale : opérateur << (voir image à la fin du document)
Les constructeurs devront initialiser l'attribut type de la classe de base à la valeur TypeEtat_Confus

Les méthodes suivantes doivent être surchargées :

- La méthode peutAttaquer qui renvoie true avec une probabilité de 50%
- La méthode appliquerEtat qui inflige des dommages à la créature, cette dernière perd 5% de ses points de vie (ou 5 points de vie si les 5% sont inférieurs à 5) avec une probabilité de 33%. A la fin de cette méthode, l'attribut durée est décrémenté de 1 point.
- La méthode estFini renvoie true
  - si durée est inférieur ou égal à 0, ou
  - elle fait un tirage aléatoire et revoie true avec 33%

Classe *EtatEndormi*

Cette classe hérite de EtatCreature.

Les méthodes suivantes doivent être implémentées :

- Un constructeur qui prend le nom en argument
- Un constructeur qui prend le nom et la durée de l'état en argument

La fonction globale : opérateur << (voir image à la fin du document)
Les constructeurs devront initialiser l'attribut type de la classe de base à la valeur TypeEtat_Endormi

Les méthodes suivantes doivent être surchargées :

- La méthode peutAttaquer qui renvoie false
- La méthode appliquerEtat décrémente la durée si la durée est positive
- La méthode estFini renvoie true si durée est inférieur ou égal à 0

Classe *EtatEmpoisonne*

Cette classe hérite de EtatCreature.

Les méthodes suivantes doivent être implémentées :

- Un constructeur qui prend le nom en argument
- Un constructeur qui prend le nom et la durée de l'état en argument

La fonction globale : opérateur << qui affiche « etat empoisonne »
Les constructeurs devront initialiser l'attribut type de la classe mère à la valeur TypeEtat_Empoisone

Les méthodes suivantes doivent être surchargées :

- La méthode appliquerEtat décrémente la durée et retire 5 points de vie à la créature
- La méthode estFini renvoie true si durée est inférieur ou égal à 0

Classe *ObjetMagique*

Cette classe sert de classe de base pour représenter les objets magiques.

Classe *PotionMagique*

Cette classe hérite de objetMagique, elle représente des potions qui rendent uniquement de la vie.

Les méthodes suivantes doivent être implémentées :

- Un constructeur par défaut semblable à celui de la classe de base
- Un constructeur par paramètre semblable à celui de la classe de base

La fonction globale : operator<< friend dont l'affichage est visible à la fin du document

La méthode suivante doit être surchargée :

- *utiliserSur(Créature)* rend des points de vie à la créature. La valeur des points de vie rendus est celle de l'attribut bonus. (Attention il ne faut pas que les points de vie dépassent les points de vie total)

Classe *GreenBull*

Cette classe hérite de objetMagique, elle représente des boissons énergétiques qui rendent uniquement de l'énergie aux créatures.

Les méthodes suivantes doivent être implémentées :

- Un constructeur par défaut semblable à celui de la classe mère
- Un constructeur par paramètre semblable à celui de la classe mère

La fonction globale operator<< friend dont l'affichage est visible à la fin du document

La méthode suivante doit être surchargée :

- *utiliserSur()* rend autant d'énergie que l'attribut bonus de la classe de base (attention il ne faut pas dépasser l'énergie maximale)

Classe *Elixir*

Cette classe hérite de PotionMagique.

Les méthodes suivantes doivent être implémentées :

- Un constructeur par défaut semblable à celui de la classe mère
- Un constructeur par paramètre semblable à celui de la classe mère

La fonction globale operator<< friend dont l'affichage est visible à la fin du document

La méthode suivante doit être surchargée :

- utiliserSur rend autant d'énergie que l'attribut bonus de la classe de base et 2*bonus points de vie (attention aux valeurs maximales). Pour implémenter cette méthode vous ne devez utiliser uniquement des méthodes des classes de base PotionMagique et ObjetMagique.

Classe *pouvoir*

Un attribut a été rajouté dans cette classe :

- type caractérise les modification d'état que la créature attaque peut subir (type enuméré TypeEtat). Il doit être initialisé à TypeEtat_normal.

Deux méthodes ont été rajoutées dans cette classe :

- AppliquerEffetOffensif : si un pouvoir doit changer l'état d'une créature adverse, c'est cette méthode qui est responsable de cette action. La créature adverse sera passée en arguments.
- AppliquerEffetDefensif : si un pouvoir doit apporter un bonus à l'attaquant c'est cette méthode qui en sera responsable. C'est la créature (l'attaquant) elle-même qui sera passée en argument.
- obtenirType

Cette classe servant de classe de base, ces méthodes ne font rien.

Classe *pouvoirPoison*

Cette classe hérite de Pouvoir.

Le attribut suivant est à rajouter:

- Duree (entier)

Les méthodes suivantes doivent être implémentées :

- Un constructeur par paramètres ayant les mêmes paramètres que celui de la classe de base
- Un constructeur comprenant les mêmes paramètres que ceux du constructeur précédent ainsi qu'une durée

La fonction globale : opérateur friend << (cet opérateur doit appeler celui de la classe mère). Voir à la fin du document pour un exemple d'affichage.

Les constructeurs doivent initialiser la variable type à TypeEtat_Empoisonne.

La méthode suivante doit être surchargée :

- appliquerEffetOffensif modifie l'état de la créature cible vers l'état empoisonné

Classe *pouvoirSoporifique*

Cette classe hérite de Pouvoir.

L'attribut suivant est à rajouter:

- Duree (entier)

Les méthodes suivantes doivent être implémentées :

- Un constructeur par paramètres ayant les mêmes paramètres que celui de la classe mère
- Un constructeur comprenant les mêmes paramètres que ceux du constructeur précédent ainsi qu'une durée

La fonction globale : opérateur friend << (cet opérateur doit appeler celui de la classe mère). Voir à la fin du document pour un exemple d'affichage.

Les constructeurs doivent initialiser la variable type à TypeEtat_Endormi.

La méthode suivante doit être surchargée :

- appliquerEffetOffensif modifie l'état de la créature cible vers l'état endormi

Classe *pouvoirHallucinogene*

Cette classe hérite de Pouvoir.

Le attribut suivant est à rajouter:

- Duree (entier)

Les méthodes suivantes doivent être implémentées :

- Un constructeur par paramètres ayant les mêmes paramètres que celui de la classe mère
- Un constructeur comprenant les mêmes paramètres que ceux du constructeur précédent ainsi qu'une durée

La fonction globale : opérateur friend << (cet opérateur doit appeler celui de la classe mère). Voir à la fin du document pour un exemple d'affichage.

Les constructeurs doivent initialiser la variable type à TypeEtat_Confus.

La méthode suivante doit être surchargée :

- appliquerEffetOffensif modifie l'état de la créature cible vers l'état confus

Les différents états correspondant aux différents pouvoirs sont résumés dans le tableau ci-dessous :

Pouvoir	Etat
PouvoirSoporifique	EtatEndormi
PouvoirPoison	EtatEmpoisonne
PouvoirHallucinogene	EtatConfus
Autre	EtatNormal

Classe *Creature*

Les attributs et méthodes liées au TP2 restent inchangées, sauf si spécifié. En particulier, la méthode attaquer n'est pas à modifier pour prendre en compte les différents états.

Notez que les pouvoirs sont désormais en agrégation, ce choix a été fait pour simplifier l'héritage. Considérez ça comme une concession pédagogique plutôt qu'un choix de design.

Cette classe possède l'attribut supplémentaire suivant:

- Etat un pointeur vers la classe EtatCreature

Les méthodes suivantes doivent être implémentées :

- Accesseur pour le nouvel attribut
- Une méthode pour modifier le nouvel attribut

Classe *CreatureMagique*

Créer une classe *Creature* qui représente une créature dans le monde de Polyland.

Cette classe possède l'attribut supplémentaire suivant:

- Bonus (entier)

Les méthodes suivantes doivent être implémentées :

- Constructeurs par paramètres ayant tous les paramètres du constructeur par défaut de la classe de base ainsi qu'un paramètre représentant le bonus
- Un opérateur =
- Un constructeur par copie

La fonction globale : opérateur friend << (pensez à appeler celui de la classe mère !) Un exemple est visible à la fin du document.

Les méthodes suivantes doivent être surchargées :

- Attaquer doit effectuer une attaque normale et augmenter sa vie de bonus point de vie. (On doit tout de même respecter les valeurs maximales)

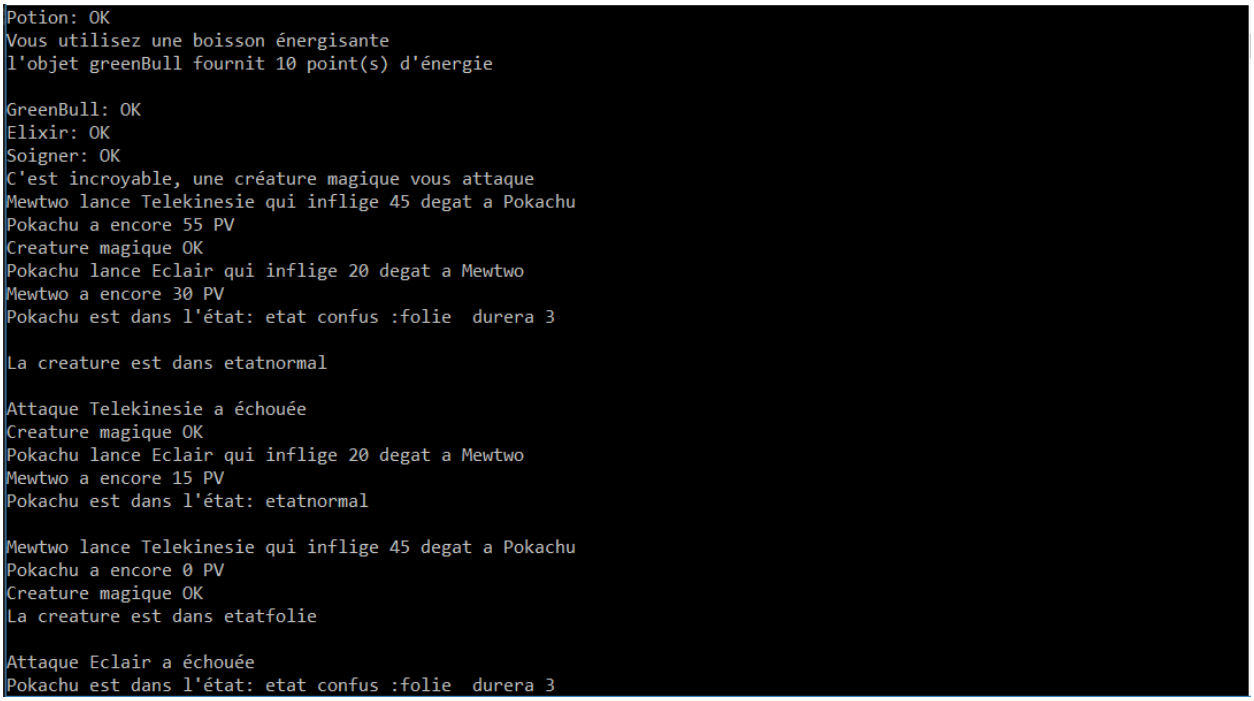
Main.cpp

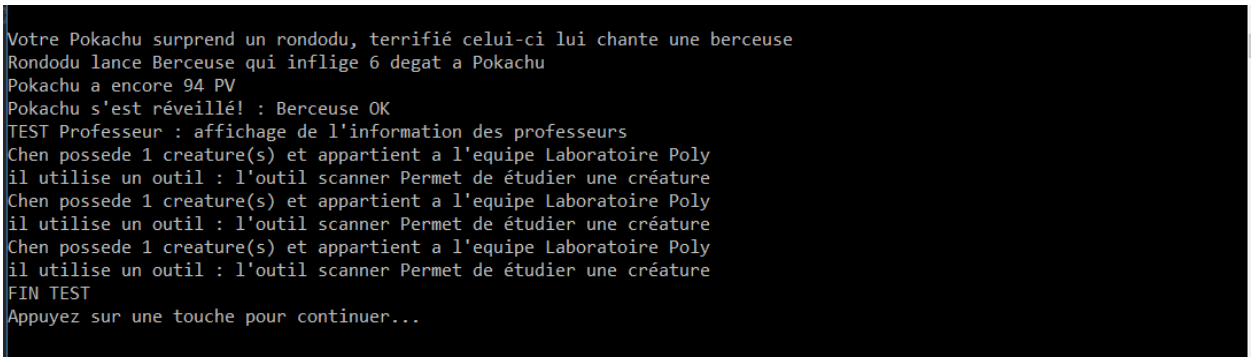
Le programme principal contient des directives à suivre pour instancier différents objets et essayer les différentes méthodes implémentées. Il vous est également demandé de compléter les parties identifiées par un *TODO*.

Le résultat final devrait être similaire à ce qui suit :

```
BIENVENU DANS LE MONDE MERVEILLEUX DE POLYLAND
l'outil scanner Permet de étudier une créature
TEST : affichage de la créature magique
La créature Magique Mewtwo a 10 en attaque et 3 en defense,
Il a 50/50 PV et 25/25 Energie
Il est au niveau 1 avec 0d'XP
Il lui manque 100 jusqu'au prochain niveau
Pouvoirs :
Mewtwo ne connait aucun pouvoir

FIN TEST : affichage de la créature magique
TEST AFFICHAGE Pouvoir
Eclair possede un nombre de dégat de 10 et une energie necessaire de 5

Morsure Venin possede un nombre de dégat de 10 et une energie necessaire de 5
 il peut empoisonner la cible

onde Folie possede un nombre de dégat de 4 et une energie necessaire de 5
  il peut rendre confus la cible

Berceuse possede un nombre de dégat de 2 et une energie necessaire de 5
  il peut emdormir la cible

Telekinesie possede un nombre de dégat de 15 et une energie necessaire de 5
  il peut rendre confus la cible

FIN : TEST AFFICHAGE Pouvoir
TEST PolyLand
Chen a bien été ajouté !
Jessie a bien été ajouté !
```

```
Chen possede 1 creature(s) et appartient a l'equipe Laboratoire Poly
Jessie possede 1 creature(s) et appartient a l'equipe Team Rocket

FIN TEST PolyLand
LA TEAM MISSILE VOUS ATTAQUE
Miaouss lance Morsure Venin qui inflige 30 degat a Pokachu
Pokachu a encore 70 PV
Pokachu lance Eclair qui inflige 30 degat a Miaouss
Miaouss a encore 20 PV
affichage de l'etat de Pokachu
Pokachu est dans l'état: etatEmpoisonne

affichage de l'etat de Pokachu (autre méthode)
Pokachu est dans l'état: etat empoisonne :Empoisonne durera 2

Miaouss lance Morsure Venin qui inflige 30 degat a Pokachu
Pokachu a encore 35 PV
Pokachu lance Eclair qui inflige 30 degat a Miaouss
Pokachu a gagné 30 XP
Miaouss a encore 0 PV
affichage de l'etat de Pokachu
Pokachu est dans l'état: etatEmpoisonne

affichage de l'etat de Pokachu (autre méthode)
Pokachu est dans l'état: etat empoisonne :Empoisonne durera 2

Felicitation vous avez gagne
Vous utilisez une potion magique
l'objet Potionsoigne de 10 points de vie
```





Questions

1. Il y a parfois une incohérence dans l'affichage des données deux appels à cout sur le même objet semblent produire des résultats différents. Pouvez-vous l'expliquer ? (Pokachu premier combat).
2. Quel est l'ordre d'appel des différents constructeurs lors d'un appel au constructeur par paramètres de Professeur?
3. Remplissez le tableau suivant :
   Considérons les déclarations suivantes :
   Class Mere;
   Class fille : public Mere;
   fonctionSurMere(Mere& mere);
   fonctionSurFille(Fille& fille);
   Mere m;
   Fille f;

Mere* mPointeur;
Fille* fPointeur;

Situation	Est valide?
m = f;	
f = m;	
mPointeur = &f;	
fPointeur = &m;	
fonctionSurMere(f);	
fonctionSurFille(m);	

Correction

La correction du TP3 se fera sur 20 points.

Voici les détails de la correction :

- (03 points) Compilation du programme;
- (03 points) Exécution du programme;
- (04 points) Comportement exact des méthodes du programme;
- (04 points) Utilisation adéquate de l'héritage;
- (01 points) Utilisation correcte de la portée des attributs;
- (01 point) Documentation du code;
- (01 point) Utilisation correcte du mot-clé *this* et const;
- (03 points) Réponses aux questions.