SCALE FOR PROJECT CPP MODULE 04 (/PROJECTS/CPP-MODULE-04)

You should evaluate 1 student in this team

★

Git repository

```
git@vogsphere-v2.42.fr:vogsphere/intra-uuid-f931dde4-a498-4d86-bde7-l
```

Introduction

Merci de respecter les règles suivantes:

- Restez polis, courtois, respectueux et constructifs pendant le processus d'évaluation. Le bien-être de la communauté repose là-dessus.

- Identifiez avec la personne évaluée ou le groupe évalué les éventuels dysfonctionnements de son travail. Prenez le temps d'en discuter et débattez des problèmes identifiés.

- Vous devez prendre en compte qu'il peut y avoir de légères différences d'interprétation entre les instructions du projet, son scope et ses fonctionnalités. Gardez un esprit ouvert et notez de la manière la plus honnête possible. La pédagogie n'est valide que si la peer-évaluation est faite sérieusement.

Guidelines

- Ne notez que ce qui est contenu dans le dépôt Git cloné de l'étudiant(e) ou du groupe.

- Vérifiez que le dépôt Git appartient bien à l'étudiant(e) ou au groupe, que le projet est bien celui attendu, et que "git clone" est utilisé dans un dossier vide.

- Vérifiez scrupuleusement qu'aucun alias n'a été utilisé pour vous tromper et assurez-vous que vous évaluez bien le rendu officiel.

- Afin d'éviter toute surprise, vérifiez avec l'étudiant(e) ou le groupe les potentiels scripts utilisés pour faciliter l'évaluation (par exemple, des scripts de tests ou d'automatisation).

- Si vous n'avez pas fait le projet que vous allez évaluer, vous devez lire le sujet en entier avant de commencer l'évaluation.

- Utilisez les flags disponibles pour signaler un rendu vide, un programme ne fonctionnant pas, une erreur de Norme, de la triche... Dans ces situations, l'évaluation est terminée et la note est 0, ou -42 en cas de triche. Cependant, à l'exception des cas de triche, vous êtes encouragé(e)s à continuer la discussion sur le travail rendu, même si ce dernier est incomplet. Ceci afin d'identifier les erreurs à ne pas reproduire dans le futur.

- Si le sujet requiert un fichier de configuration, vous ne devriez jamais avoir à le modifier. Si vous souhaitez éditer un fichier, prenez le temps d'expliquer pourquoi à la personne évaluée et de vous assurer que vous avez son accord.

- Vous devez aussi vérifier l'absence de fuites mémoire. Toute mémoire allouée sur le tas doit être libérée proprement avant la fin de l'exécution du programme.
Vous avez le droit d'utiliser tout outil disponible sur la machine tel que leaks, valgrind ou e_fence. En cas de fuites mémoire, cochez le flag approprié.

Attachments

subject.pdf (https://cdn.intra.42.fr/pdf/pdf/41320/fr.subject.pdf)

Tests préliminaires

Si un cas de triche est suspecté, la notation et l'évaluation prennent fin immédiatement. Pour le signaler, sélectionnez le flag "Cheat". Faites attention à l'utiliser avec calme, précaution et discernement.

Prérequis

Le code doit compiler avec c++ et les flags -Wall -Wextra -Werror
Pour rappel, ce projet doit suivre le standard C++98. Par conséquent, des fonctions C++11 (ou autre standard) et les containers ne sont PAS attendus.

Ne notez pas l'exercice si vous trouvez :
- Une fonction implémentée dans un fichier d'en-tête (sauf pour les fonctions templates).
- Un Makefile compilant sans les flags demandés et/ou avec autre chose que c++.

Sélectionnez le flag "Fonction interdite" (Forbidden function) si vous rencontrez :
- L'utilisation d'une fonction "C" (*alloc, *printf, free).
- L'utilisation d'une fonction interdite dans le projet.
- L'utilisation de "using namespace " ou du mot-clé "friend".
- L'utilisation d'une bibliothèque externe, ou de fonctionnalités propres aux versions postérieures à C++98.

Yes No

Ex00 : Polymorphisme

Comme d'habitude, il doit y avoir assez de tests pour prouver que le programme fonctionne comme demandé. S'il n'y en a pas, ou que les tests ne sont pas assez précis, ou qu'une des classes n'est pas sous la forme canonique de Coplien, ne notez pas cet exercice.

Premières vérifications

Il y a une classe Animal qui possède un attribut :
Une string appelée type.
Vous devez pouvoir instantier cette classe et l'utiliser.

Yes No

Héritage

Il y a au minimum deux classes héritant d'Animal : Cat et Dog.
Les messages affichés par leur constructeur et leur destructeur doivent être cohérents.
Demandez des explications sur l'ordre d'enchaînement du constructeur et du destructeur.

Yes No

Classe dérivée simple

L'attribut type est set à la valeur appropriée à la création de chaque animal.
Ainsi, les chats doivent avoir le type "Cat" et les chiens "Dog".

Yes No

Animal

La fonction makeSound() doit toujours appeler la fonction makeSound() correspondante. makeSound() doit être virtuelle ! Vérifiez la présence du mot-clé "virtual" dans le code :
virtual void makeSound() const
La valeur de retour n'est pas importante, par contre, "virtual" est obligatoire.

Il doit y avoir un exemple avec un WrongAnimal et un WrongCat n'utilisant pas le mot-clé "virtual" (cf. sujet).
Le WrongCat doit afficher le makeSound() du WrongCat seulement quand utilisé comme WrongCat.

Yes No

Ex01 : Je ne veux pas brûler le monde

Comme d'habitude, il doit y avoir assez de tests pour prouver que le programme fonctionne comme demandé. S'il n'y en a pas, ou que les tests ne sont pas assez précis, ou qu'une des classes n'est pas sous la forme canonique de Coplien, ne notez pas cet exercice.

Classe concrète Animal

Il y a une nouvelle classe Brain.
Les classes Cat et Dog possèdent désormais l'attribut Brain comme demandé.
L'attribut Brain ne doit pas être dans la classe Animal.
La classe Brain doit avoir ses propres messages affichés par son constructeur et son destructeur.

Yes No

Classe concrète Brain

La copie d'un Cat ou d'un Dog doit toujours être profonde.
Essayez quelque chose comme :
Dog basic;
{
Dog tmp = basic
}
Si la copie est superficielle, tmp et basic auront le même Brain. Celui-ci sera supprimé à la fin du scope avec tmp.
Le constructeur par recopie doit aussi faire une copie profonde.
C'est pourquoi une implémentation propre suivant la forme canonique de

Coplien vous épargnera des heures de souffrance.

Yes No

Enchaînement des destructeurs

Les destructeurs de la classe Animal et de ses classes dérivées sont virtuels.
Demandez à l'étudiant(e) ce qui se passerait sans le mot-clé "virtual".
Ensuite, vérifiez en testant sans "virtual".

Yes No

Affectation et copie

Le constructeur par recopie et l'opérateur d'affectation des classes Cat et Dog marchent comme demandé.
La copie profonde créé un nouveau Brain pour le nouveau Cat ou Dog.
Assurez-vous que la forme canonique de Coplien est vraiment implémentée (pas d'opérateur d'affectation vide, etc.). Rien ne devrait être public sans raison valable. De plus, ce code est très simple et doit donc être propre !

Yes No

Ex02 : Classe abstraite

Comme d'habitude, il doit y avoir assez de tests pour prouver que le programme fonctionne comme demandé. S'il n'y en a pas, ou que les tests ne sont pas assez précis, ou qu'une des classes n'est pas sous la forme canonique de Coplien, ne notez pas cet exercice.

Classe abstraite

Il y a une classe Animal qui se comporte exactement comme attendu.
La fonction Animal::makeSound() est une fonction virtuelle pure :
virtual void makeSound() const = 0;
Le "= 0" est obligatoire.
Par conséquent, vous ne devriez pas pouvoir instantier un Animal.
Animal test; // résultera en une erreur de compilation disant que la classe est abstraite

Yes No

Concrete Animal

Class Cat and Dog are still present and work exactly like in ex02.

Yes No

Ex03 : Interface and recap

Comme d'habitude, il doit y avoir assez de tests pour prouver que le programme fonctionne comme demandé. S'il n'y en a pas, ou que les tests ne sont pas assez précis, ou qu'une des classes n'est pas sous la forme canonique de Coplien, ne notez pas cet exercice.

Interfaces

Il y a des interfaces ICharacter et IMateriaSource qui sont exactement comme demandé dans le sujet.

Yes No

MateriaSource

La classe MateriaSource est présente et implémente IMateriaSource. Les fonctions membres marchent comme attendu.

Yes No

Classe concrète Materia

Il y a des classe concrètes Ice et Cure qui héritent d'AMateria. Leur méthode clone() est correctement implémentée. Les messages affichés sont comme demandé.
La classe AMateria est abstraite (clone() est une fonction pure).
Le destructeur est virtuel : virtual ~AMateria()
AMateria possède un attribut protégé de type string pour le type de Materia.

Yes No

Personnage

La classe Character est présente et implémente ICharacter. Elle possède un inventaire pouvant contenir 4 Materias maximum.
Ses fonctions membres sont implémentées comme demandé.
La copie et l'affectation d'un Character sont implémentées comme demandé (copie profonde).

Yes No

Ratings

Don't forget to check the flag corresponding to the defense

Ok Outstanding project

Empty work No author file Invalid compilation Norme Cheat Crash Leaks

Forbidden function

Conclusion

Leave a comment on this evaluation

Finish evaluation