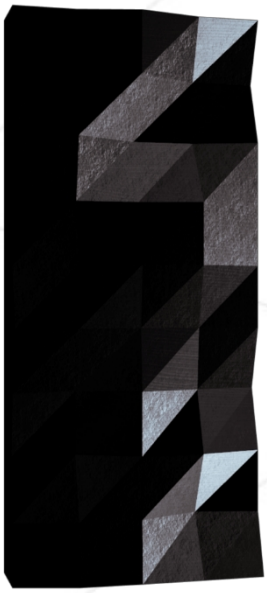

C++ - モジュール 04

サブタイプ多相性、抽象クラス、インターフェース

概要

このドキュメントには、C++ モジュールのモジュール04の練習問題が含まれています。

バージョン: 11

内容

第一章 はじめに

C++は、Bjarne Stroustrup（ビャルネ・ストルストラップ）により、C言語の発展形として開発された汎用プログラミング言語である（出典：Wikipedia）。

これらのモジュールの目的は、**オブジェクト指向プログラミングを**紹介することです。これはあなたのC++の旅の出発点となるでしょう。OOPを学ぶには多くの言語が推奨されています。これは複雑な言語であり、物事をシンプルに保つために、あなたのコードはC++98標準に準拠することになります。

私たちは、現代のC++が多くの面で大きく異なっていることを認識しています。ですから、もしあなたが熟達したC++開発者になりたいのであれば、42のコモン・コアの後にさらに進むのはあなた次第です！

第II章 一般規定

コンパイル

- c++と-Wall -Wextra -Werrorフラグでコードをコンパイルする。
- フラグ-std=c++98を追加しても、コードはコンパイルできるはずだ。

フォーマットと命名規則

- 演習用のディレクトリは次のように命名されます：ex00, ex01, ... , exn.
  exn
- ファイル名、クラス名、関数名、メンバ関数名、属性名は、ガイドラインの要求に従ってください。
- クラス名は**UpperCamelCase**形式で記述する。クラス・コードを含むファイルは常にクラス名に従って命名されます。例えば例えば、ClassName.hpp/ClassName.h、ClassName.cpp、ClassName.tpp。例えば、ClassName.hpp/ClassName.hpp/ClassName.cpp/ClassName.tppのようになります。
- 特に指定がない限り、すべての出力メッセージは改行文字で終了し、標準出力に表示されなければならない。
- *さようならノルミネット*C++モジュールでは、コーディング・スタイルは強制されません。あなたの好きなスタイルに従ってください。しかし、相互評価者が理解できないコードは、評価できないコードであることを心に留めておいてください。きれいで読みやすいコードを書くよう、ベストを尽くしてください。

許可/禁止

あなたはもうC言語でコーディングしていない。C++の出番だ！だから

- 標準ライブラリのほとんどすべてを使うことが許されている。したがって、すでに知っていることに固執するのではなく、使い慣れたC関数のC++っぽいバージョンをできるだけ使うのが賢いやり方だろう。
- ただし、それ以外の外部ライブラリーは使用できない。つまり、C++11（および派生形式）とBoostライブラリは禁止されている。以下の関数も禁止されている：*printf()、*alloc()、free()です。これらを使用した場合、あなたの成績は0点となり、それで終わりです。

- 明示的に指定されていない限り、using 名前空間 <ns_name> と友達キーワードは禁止。さもなければ、あなたの成績は-42点となる。
- **モジュール 08 と 09 でのみ STL を使用できます。**つまり、それまでは**コンテナー**（vector/list/map/など）や**アルゴリズム**（<algorithm>ヘッダーを含む必要があるもの）を使用しないこと。そうでなければ、成績は-42点となる。

いくつかの設計要件

- メモリ・リークはC++でも発生する。メモリを確保するときに（new キーワード）、**メモリー・リークを**避けなければならない。
- モジュール02からモジュール09まで、**特に明記されている場合を除き、**あなたのクラスは**正教会正書式で**デザインされなければなりません。
- ヘッダー・ファイルに書かれた関数の実装は（関数テンプレートを除いて）、練習にとっては0を意味する。
- それぞれのヘッダーは、他のヘッダーから独立して使えるようにしなければならない。したがって、必要な依存関係をすべてインクルードしなければならない。しかし、**インクルード・ガードを**追加することで、二重インクルードの問題を避けなければなりません。そうでなければ、あなたの成績は0点になってしまいます。

続きを読む

- 必要であれば（コードを分割するためなど）ファイルを追加しても構いません。これらの課題はプログラムによって検証されるわけではないので、必須ファイルを提出する限り、自由に行ってください。
- 時には、練習のガイドラインが短く見えても、例題を見れば、指示には明示的に書かれていない要件がわかることもある。
- 始める前に各モジュールを完全に読むこと！本当にそうしてください。

- オーディンによって、ソーによって！頭を使え!

たくさんのクラスを実装しなければならない。これは、お気に入りのテキストエディタでスクリプトを書くことができない限り、退屈に思えるかもしれない。

練習をこなすにはある程度の自由が与えられている。しかし、義務的なルールは守り、怠けてはいけません。多くの有益な情報を見逃すことになります！理論的な概念を読むことをためらわないでください。

第三章

練習00：ポリモーフィズム

	エクササイズ：00
ポリモーフィズム	
ターンイン・ディレクトリ : *ex00/*	
提出するファイル：Makefile、main.cpp、*.cpp、*.{h、hpp}。	
禁止されている関数： なし	

すべての運動について、できる**限り完全なテストを**提供しなければならない。
各クラスのコンストラクタとデストラクタは特定のメッセージを表示しなければならない。すべてのクラスに同じメッセージを使わないでください。

Animalというシンプルな基本クラスを実装することから始めましょう。このクラスには1つのprotected属性があります：

- std::string型；

Animalを継承した**Dogクラスを**実装する。Animalを継承した**Catクラスを**実装する。

これら2つの派生クラスは、その名前に応じて型フィールドを設定しなければならない。すると、Dogの型は "Dog "に、Catの型は "Cat "に初期化される。Animalクラスの型は空のままでもいいし、好きな値に設定してもいい。

すべての動物はメンバー関数を使うことができなければならない：
makeSound()

適切な音が印刷される（猫は吠えない）。

このコードを実行すると、Animalの音ではなく、DogとCatクラスの固有の音が表示されます。

```
int main()
{
    const Animal* meta = new Animal();
    const Animal* j = new Dog();
    const Animal* i = new Cat();

    std::cout << j->getType() << " " << std::endl;
    std::cout << i->getType() << " " << std::endl; i-
    >makeSound(); //猫の鳴き声を出力します
    !
    j->makeSound();
    meta->makeSound()
    ;
    ---

    0を返す;
}
```

WrongAnimalクラスを継承した**WrongCatクラスを**実装します。上のコードでAnimalとCatを間違ったものに置き換えると、WrongCatはWrongAnimalの音を出力するはずです。

上記以外のテストも実施し、提出すること。

第四章

練習01：世界に火をつけたくない

	エクササイズ：01
世界に火をつけたくはない	
ターンイン・ディレクトリ : *ex01/*	
提出ファイル：前回の演習で使用したファイル + *.cpp, *.{h, hpp}.	
禁止されている関数： なし	

各クラスのコンストラクタとデストラクタは特定のメッセージを表示しなければならない。**Brain**クラスを実装する。ideasと呼ばれる100個のstd::stringの配列を含む。

こうすることで、DogとCatはプライベートなBrain*属性を持つことになる。
構築されると、DogとCatはnew Brain()を使ってBrainを作成する；
破壊されると、犬と猫は自分の脳を削除する。

メイン関数の中で、**Animal**オブジェクトの配列を作り、それを埋める。半分は**Dog**オブジェクトで、もう半分は**Cat**オブジェクトにします。プログラム実行の最後に、この配列をループしてすべてのAnimalを削除してください。Animalとして犬と猫を直接削除しなければなりません。適切なデストラクタを期待される順序で呼び出さなければならない。

メモリーリークのチェックをお忘れなく

。

犬や猫のコピーは浅くてはならない。したがって、自分のコピーが深いコピーであることをテストしなければならない!

```
int main()
{
    const Animal* j = new Dog();
    const Animal* i = new Cat();


    delete j;//リークを起こしてはならない
    を削除する;
    ---

    0を返す;
}
```

C++ - モジュール 04　　サブタイプポリモーフィズム、抽象クラス、インターフェース

上記以外のテストも実施し、提出すること。

第五章

練習02：抽象クラス

	エクササイズ：02
抽象クラス	
ターンイン・ディレクトリ : *ex02/*	
提出ファイル：前回の演習で使用したファイル + `*.cpp, *.{h, hpp}.`	
禁止されている関数： なし	

アニマルオブジェクトを作っても、結局は意味がない。確かに、彼らは音を出さない！間違いを避けるために、デフォルトのAnimalクラスはインスタンス化できないようにしておく。

Animalクラスを修正し、誰もインスタンス化できないようにする。すべてが以前と同じように動くはずだ。

必要であれば、Animalに接頭辞Aを付けてクラス名を更新することができる。

第六章

練習03：インターフェイスとまとめ

	練習：03
インターフェイスと総括	
ターンイン・ディレクトリ : *ex03/*	
提出するファイル：Makefile、main.cpp、*.cpp、*.{h、hpp}。	
禁止されている関数： なし	

インターフェイスはC++98には存在しない（C++20にも存在しない）。しかし、純粋な抽象クラスは一般にインターフェースと呼ばれる。従って、この最後の練習では、このモジュールが理解できたかどうかを確認するために、インターフェイスを実装してみましょう。

以下の **AMateria** クラスの定義を完成させ、必要なメンバ関数を実装する。

```
クラス AMateria
{
    保護されている:
            [...]

    を公開した:
            AMateria(std::string const &
            type); [...].

            std::string const & getType() const; //マテリアのタイプを返す

            virtual AMateria* clone() const = 0;
            virtual void use(ICharacter&target);
};
```

Materiasの具象クラスIceと**Cureを**実装する。それらの名前を小 文字（Iceなら "ice"、Cureなら "cure"）にして型を設定する。もちろん、メンバ関数のclone()は同じ型の新しいインスタンスを返す（つまり、Ice Materiaをクローンすると、新しいIce Materiaが得られる）。

use(ICharacter&)メンバ関数が表示されます：

- Ice: "* <名前>に氷の稲妻を放つ *"
- Cure: "* <名前>の傷を癒す *"

<name>はパラメータとして渡されるキャラクタの名前です。角括弧（<と>）は表示しない。

マテリアを別のマテリアに割り当てる際、タイプをコピーしても意味がない。

次のインターフェイスを実装する具象クラス **Character を**書きなさい：

```
クラス ICharacter
{
    を公開した：
            virtual ~ICharacter() {}
            virtual void equip(AMateria* m) = 0;
            virtual void unequip(int idx) = 0;
            virtual void use(int idx, ICharacter& target) = 0;
};
```

このキャラクターは4スロットのインベントリを所持している。建設時のインベントリは空である。最初に空 いた スロットにマテリアを装備する。 つまり、スロット0からスロット3までの順番となる。 満タンのインベントリにマテリアを追加しようとしたり、既存のマテリアを使用/装備解除しようとしたりしても、何もしないこと（それでもバグは禁止されている）。unequip() メンバ関数はマテリアを削除してはならない！

キャラクターが床に置いていったマテリアを好きなように処理する。unequip()を呼び出したりする前にアドレスを保存しておくこと。ただし、メモリリークを避けなければならないことをお忘れなく。

use(int, ICharacter&)メンバ関数はスロット[idx]でマテリアを使用し、AMateria::use関数にターゲットパラメータを渡す必要があります。

キャラクターのインベントリは、あらゆる種類のアマテリアに対応する。

Characterは、その名前をパラメータとするコンストラクタを持たなければならない。キャラクターのコピー（コピーコンストラクタまたはコピー代入演算子を使用）は、**deep**でなければなりません。コピー中は、新しいマテリアルをインベントリに追加する前に、そのキャラクターのマテリアルを削除しなければなりません。もちろん、キャラクターが破壊された場合もマテリアは削除されなければなりません。

次のインターフェイスを実装する具象クラス**MateriaSourceを**書きなさい：

```
クラス IMateriaSource
{
    を公開した:
           virtual ~IMateriaSource() {}.
           virtual void learnMateria(AMateria*) = 0;
           virtual AMateria* createMateria(std::string const & type) = 0;
};
```

- learnMateria(AMateria*)
  パラメータとして渡されたマテリアをコピーし、後でクローンできるようにメモリに保存する。Characterと同様、**MateriaSourceは**最大4つのMateriaを知ることができる。それらは必ずしも一意ではありません。
- createMateria(std::string const &)
  新しいMateriaを返す。後者は**MateriaSourceによって**以前に学習されたMateriaのコピーであり、その型はパラメータとして渡されたものと等しい。型が不明な場合は0を返す。

一言で言えば、MateriaSourceはMateriaの "テンプレート "を学習して、必要なときにMateriaを生成できなければならない。そうすれば、タイプを識別

する文字列だけを使って新しいマテリアを生成できるようになる。

このコードを実行する：

```
int main()
{
    IMateriaSource* src = new MateriaSource();
    src->learnMateria(new Ice());
    src->learnMateria(new Cure());

    ICharacter* me = new Character("me");

    AMateria* tmp;
    tmp = src->createMateria("ice");
    me->equip(tmp);
    tmp = src->createMateria("cure");
    me->equip(tmp);

    ICharacter* bob = new Character("bob");

    me->use(0, *bob);

    me->use(1, *bob);

    ボブを削除
    、私を削除
    、srcを削
    除;

    0を返す;
}
```

出力する必要がある：

```
clang++ -W -Wall -Werror *.cpp
$> ./a.out | cat -e
* ボブに氷の稲妻を放つ。
* ボブの傷を癒す*$
```

例によって、上記以外のテストも実施し、提出すること。

練習問題03を解かなくても、このモジュールに合格することができます。

第七章

提出と相互評価

通常通り、Gitリポジトリに課題を提出してください。あなたのリポジトリ内の作品だけが、ディフェンス中に評価されます。フォルダ名やファイル名が正しいかどうか、ためらわずに再確認してください。

???????????XXXXXXXXXX = $3$$6b616b915363971573e58914295d42