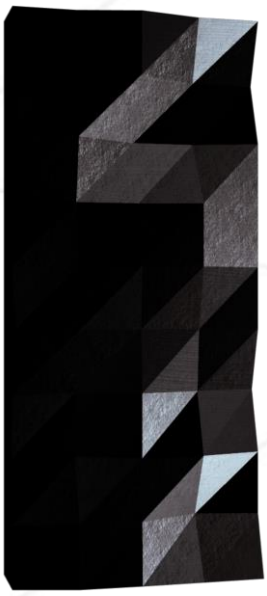

C++ - モジュール 04

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

概要

本書は、C++モジュールのうち、Module 04の演習問題を収録しています。

バージョン：10

内容

第一章 はじめに

C++は、Bjarne Stroustrupが「C with Classes」（出典：Wikipedia）というプログラミング言語の元祖として作った汎用プログラミング言語である。

これらのモジュールの目標は、**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。オブジェクト指向を学ぶには、多くの言語が推奨されます。この言語は複雑な言語であり、物事を単純化するために、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++が多くの側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、もしあなたが熟練した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なのです

第II章 総則

コンパイル

- c++と-Wall -Wextra -Werrorフラグでコンパイルしてください。
- フラグ -std=c++98 を追加しても、あなたのコードはコンパイルできるはずです。

フォーマットと命名規則

- 演習用のディレクトリは、ex00, ex01, ...のように命名されます。　exn
- ファイル、クラス、関数、メンバ関数、属性にガイドラインに必要な名前を付ける。
- クラス名は、**UpperCamelCase** 形式で記述します。クラスコードを含むファイルには、常にクラス名に従った名前が付けられます。例えば例えば、ClassName.hpp/ClassName.h, ClassName.cpp, または ClassName.tpp.例えば、レンガの壁を表すクラス "BrickWall "の定義を含むヘッダーファイルがあれば、そのファイル名はBrickWall.hppとなります。
- 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。あなたの好きなものに従えばいいのです。しかし、同僚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ために、最善を尽くしてください。

許可/禁止

あなたはもうCでコーディングしていない。C++の時代だ!というわけで。

- 標準ライブラリのほとんどすべてを使用することが許されています。したがって、すでに知っているものに固執するのではなく、使い慣れたC関数のC++的なバージョンをできるだけ使うのが賢い方法でしょう。
- ただし、それ以外の外部ライブラリは使用できません。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ます。以下の関数も禁止されています。*printf()、*alloc()、free()。これらを使用した場合、成績は0点、それで終わりです。

- 明示的に指定しない限り、using名前空間<ns_name>と
友人キーワードは禁止です。そうでない場合、あなたの成績は-42となります。
- **STL の使用はモジュール 08 のみ許可されています。**つまり、**コンテナ** (vector/list/map/ など) と**アルゴリズム** (<algorithm> ヘッダを含む必要があるもの) はそれまで使わないでください、ということです。さもなければ、あなたの成績は -42 になります。

いくつかのデザイン要件

- C++でもメモリリークは発生します。メモリを確保するときに（new
キーワード)を使用する場合は、**メモリリークを**回避する必要があります。
- モジュール02からモジュール08までは、**特に明記されている場合を除き、正教会の正書法で**授業を設計する必要があります。
- 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）、演習では0を意味します。
- それぞれのヘッダーは、他のヘッダーと独立して使用できるようにする必要があります。従って、必要な依存関係はすべてインクルードしなければなりません。しかし、**インクルードガードを**追加することによって、二重インクルードの問題を回避しなければなりません。そうでなければ、あなたの成績は0点となります。

リードミー

- 必要であれば、いくつかのファイルを追加することができます（コードを分割するためなど）。これらの課題は、プログラムによる検証を行わないので、必須ファイルを提出する限り、自由に行ってください。
- 演習のガイドラインは短く見えても、例題には明示的に書かれていない要件が示されていることがあります。
- 始める前に、各モジュールを完全に読んでください本当に、そうしてください。
- オーディンによって、トールによって!頭を使え!!!

多くのクラスを実装する必要があります。　　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る人でなければ、退屈に思えるかもしれません。

演習をこなすには、ある程度の自由度が与えられています。ただし、必須のルールは守り、怠慢は禁物です。　　多くの有益な情報を見逃すことになりますよ。　理論的な概念については、ためらわずに読んでください。

第三章

練習問題00：ポリモルフィズム

	演習：00
ポリモルフィズム	
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}.	
禁止されている機能 :なし	

すべての運動について、できる**限り完全なテストを**提供する必要があります。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する必要があります。すべてのクラスで同じメッセージを使用しないでください。

まず、Animalというシンプルな基本クラスを実装します。このクラスは1つのprotected属性を持っています。

- 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する必要があります。そうすると、Dogの型は「Dog」に、Catの型は「Cat」に初期化されます。Animalクラスの型は空のままでも、好きな値を設定することができます。

すべての動物がメンバー関数を使えるようにしなければならない。
makeSound()
適切な音が印刷されます（猫は吠えません）。

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

```
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(); //猫の音を出力します！(笑)
    j->makeSound(); meta->makeSound();
    …

    0を返します。
}
```

第四章

Exercise 01: 世界に火をつけるのはいやだ

	演習：01
世界を燃やしたくはない	
ターンインディレクトリ : *ex01/*	
提出するファイル:前回演習のファイル + *.cpp, *.{h, hpp}。	
禁止されている機能 :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する必要があります。

Brainクラスを実装します。100個のstd::stringからなるideaという配列が入っています。
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ります。
構築時に、DogとCatはnew Brain()を使ってBrainを作成します。
破壊されると、DogとCatは自分のBrainを削除します。

メイン関数の中で、**Animal**オブジェクトの配列を作成し、それを埋めてください。半分は**Dog**オブジェクト、もう半分は**Cat**オブジェクトになります。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。このとき、Animalとして、DogとCatを直接削除してください。適切なデストラクタが期待される順序で呼び出される必要があります。

メモリーリークのチェックも忘れずに。

犬や猫のコピーが浅いものであってはならない。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する必要があるのです

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


    delete j; //リークを発生させてはいけない
    を削除する。
    ...

    0を返します。
}
```

第五章

練習問題02：抽象クラス

	演習：02
抽象クラス	
ターンインディレクトリ : *ex02/*	
提出するファイル:前回演習のファイル + *.cpp, *.{h, hpp}。	
禁止されている機能 :なし	

アニマルオブジェクトを作っても、やっぱり意味がない。そうなんです、音を出さないんです!

間違いを防ぐために、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ます。
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。すべて以前と同じように動作するはずです。

必要であれば、AnimalにAという接頭辞を付けてクラス名を更新することができます。

第六章

演習03：インターフェイス＆リキャップ

	演習：03
インターフェース＆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}.	
禁止されている機能 :なし	

C++98にはインターフェイスは存在しません（C++20にも存在しません）。しかし、純粋な抽象クラスは、一般にインターフェイスと呼ばれます。したがって、この最後の演習では、このモジュールを手に入れたことを確認するために、インターフェイスの実装に挑戦してみましょう。

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してください。

```
class AMateria
{
    を保護します。
        [...]

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

        std::string const & getType() const; //マテリアの種類を返す

        virtual AMateria* clone() const = 0.マテリアの種類を返す。
        virtual void use(ICharacter& target)。
};
```

Materias の具象クラス **Ice** と **Cure** を実装します。それぞれの名前を小文字にして（Iceなら "ice"、Cureなら "cure"）型を設定します。もちろん、メンバ関数clone()は同じ型の新しいインスタンスを返します（例：Ice Materiaをcloneすると、新しいIce Materiaが得られます）。

use(ICharacter&)メンバ関数が表示されます。

- アイス："* アイスボルトを<名前>に撃つ *"
- Cure："＊＜名＞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。

マテリアを割り当てる際に、型をコピーしても意味がない。

次のインタフェースを実装する具象クラス**Characterを**作成しなさい。

```
class ICharacter
{
    を公開します。
            virtual ~ICharacter() {}
            virtual std::string const & getName() const = 0; 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つのインベントリを所持しており、最大で4つのマテリアを所持していることになります。建設時、インベントリは空です。最初に見つけた空のスロットにマテリアを装備します。つまり、スロット0からスロット3までの順番で装備することになります。万が一、満杯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ても、何もしないこと（ただし、バグは禁止されている）。unequip()関数はマテリアを削除してはならない!

キャラクターが置いていったマテリアを好きなように処理してください。unequip()を呼び出す前にアドレスを保存するなどして、メモリリークを回避することを忘れないようにしましょう。

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さなければならないでしょう。

キャラクターのインベントリに、あらゆる種類のアマテリアが対応するようになります。

キャラクターは、その名前をパラメータとするコンストラクタを持つ必要があります。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、**深いもので**なければなりません。コピー中は、新しいマテリアルがインベントリに追加される前に、そのキャラクターのマテリアルを削除する必要があります。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。

以下のインタフェースを実装した具象クラス**MateriaSourceを**作成しなさい。

```
class IMateriaSource
{
    を公開します。
        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てメモリに保存し、後でクローンできるようにします。キャラクターと同様、**MateriaSourceは**最大で4つのマテリアを知ることができます。それらは必ずしもユニークなものではありません。
- createMateria(std::string const &)
  新しいMateriaを返します。後者は**MateriaSourceが**以前に学習したMateriaのコピーで、その型はパラメータとして渡されたものと同じである。型が不明な場合は0を返します。

一言で言えば、MateriaSourceはMateriaの「テンプレート」を学習して、必要な時に作成できるようにしなければならない。そして、その種類を特定する文字列だけで、新しいマテリアを生成できるようにする。

このコードを実行する。

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

```
$> clang++ -W -Wall -Werror *.cpp
$> ./a.out | cat -e
* ボブにアイスボルトを放つ。
* ボブの傷を癒す*$。
```

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

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

    me->use(0, *bob);
```

このモジュールは、演習03を行わずに合格することができます。

```
    削除しま
    す。

    0を返します。
}
```