



## C++ - モジュール 03

### 継承

#### 概要

本書は、C++モジュールのうち、Module 03の演習を収録したものです。

バージョン : 6

# 内容

I	はじめに	2
II	一般規定	3
III	エクササイズ00：あああ...OPEN!	5
IV	Exercise 01: Serena, my love!	7
V	エクササイズ02：反復作業	8
VI	練習問題03：今度は変だ！？	9

# 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.hpp`。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。

- 標準ライブラリからほとんどすべてのものを使う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。メモリを確保する際に (`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、ツールによって!頭を使い!!!




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、面倒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となりますよ。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

# エクササイズ00：あああ...OPEN!

	演習：00
あああああああ あああああああ あああああああ あああああああ あああああああ あああああああ あああああああ あああああああ あああああああ OPEN!	
ターンインディレクトリ：ex00/	
提出するファイル:Makefile, main.cpp, ClapTrap.{h, hpp}, ClapTrap.cpp.	
禁止されている機能:なし	

まず、クラスを実装するんだ!なんて斬新なんでしょう

**ClapTrap**という名前で、以下のプライベート属性を持ち、括弧内に指定された値で初期化される予定です。

- コンストラクタのパラメータとして渡される名前
- ヒットポイント (10) : ClapTrapの健康状態を表します。
- エネルギーポイント (10)
- 攻撃ダメージ (0)

ClapTrapがよりリアルに見えるように、以下のパブリックメンバー関数を追加します。

- void attack(const std::string& target)。
- void takeDamage(unsigned int amount)。
- void beRepaired(unsigned int amount);

クラップトラックが攻撃するとき、その対象は<攻撃ダメージ>のヒットポイントを失う。クラップトラックが自身を修復すると、それは<量>のヒット・ポイントを取り戻す。攻撃と修理はそれぞれ1エネルギーポイント消費します。もちろん、

ヒットポイントもエネルギーポイントも残っていなければ、何もできません。

これらのメンバ関数では、何が起こったかを説明するメッセージを表示する必要があります。例えば、`attack()`関数は次のような表示をします（もちろん、角括弧は抜きで）。

クラップトラップ<名前>は<ターゲット>を攻撃し、<ダメージ>ポイントを与える!

コンストラクタとデストラクタはメッセージも表示しなければならないので、ピア評価者はそれらが呼び出されたことを簡単に確認できます。


コードが期待通りに動作することを確認するために、自分でテストを実装し、提出すること。





## 第四章

### Exercise 01: Serena, my love!

	演習 : 01
セレナ、私の愛しい人！	
ターンインディレクトリ : <i>ex01/</i>	
提出するファイル: 前回演習のファイル + ScavTrap.{h, hpp}, ScavTrap.cpp	
禁止されている機能 : なし	

クラップトラップはいくらあっても足りないので、今度は派生型のロボットを作成します。このロボットは**ScavTrap**と名付けられ、**ClapTrap**のコンストラクタとデストラクタを継承しています。ただし、コンストラクタ、デストラクタ、**attack()**は異なるメッセージを表示します。結局のところ、**ClapTrap**は自分の個性を自覚しているのです。

適切な建設/破壊の連鎖がテストで示されなければならないことに注意してください。**ScavTrap**が作成されると、プログラムは**ClapTrap**の構築から始まります。破壊は逆の順序で行われます。なぜ？

**ScavTrap**は**ClapTrap**の属性を使用し（結果的に**ClapTrap**を更新する）、それらを次のように初期化しなければならない。

- コンストラクタのパラメータとして渡される名前
- ヒットポイント（100） : **ClapTrap**の健康状態を表します。
- エネルギーポイント（50）
- 攻撃ダメージ(20)

**ScavTrap**も独自の特殊な容量を持つことになります。


**void guardGate()**。

この関数は、**ScavTrap**がゲートキーパモードになったことを通知するメッセージを表示します。

プログラムにテストを追加することを忘れないでください。

## 第五章

### エクササイズ02：反復作業

	演習：02
繰り返しの作業	
ターンインディレクトリ：ex02/	
提出するファイル:過去の演習で使ったファイル + FragTrap.{h, hpp}, FragTrap.cpp	
禁止されている機能 :なし	

ClapTrapsを作るのが面倒くさくなってきたのでしょう。

さて、ClapTrap を継承した **FragTrap** クラスを実装します。ScavTrapと非常によく似ています。しかし、その構築と破壊のメッセージは異なるものでなければなりません。適切な構築・破壊の連鎖がテストで示されなければなりません。FragTrapが生成されると、プログラムはClapTrapの構築から始まります。破壊は逆の順序で行われます。

属性については同じことですが、今回は値を変えています。

- コンストラクタのパラメータとして渡される名前
- ヒットポイント（100）：ClapTrapの健康状態を表します。
- エネルギーポイント（100）
- 攻撃ダメージ(30)

FragTrapも特殊な能力を持っています。

```
void highFivesGuys(void);
```

このメンバ関数は、正のハイファイブリクエストを標準出力に表示する。

もう一度、プログラムにテストを追加してください。

## 第六章

### 練習問題03：今度は変だ！？

	演習：03
今のは変だ!	
ターンインディレクトリ: <i>ex03/</i>	
提出するファイル: 過去の演習のファイル + DiamondTrap.{h, hpp}, DiamondTrap.cpp	
禁止されている機能: なし	

この練習では、半分がFragTrapで半分がScavTrapのClapTrapというモンスターを作成することになります。名前は**DiamondTrap**で、FragTrapとScavTrapの両方を継承しています。これは危険だ！

DiamondTrapクラスは、`name` `private`属性を持つことになります。この属性には、ClapTrap基底クラスの変数名と全く同じ変数名（ここではロボットの名前について話していない）を与える。

より明確にするために、2つの例を挙げます。  
ClapTrapの変数が`name`の場合、DiamondTrapのものに`name`を付与する。  
ClapTrapの変数が`_name`の場合、DiamondTrapのものに`_name`を付与する。

その属性とメンバー関数は、どちらかの親クラスから選ばれます。

- コンストラクタのパラメータとして渡される名前
- `ClapTrap::name` (コンストラクタのパラメータ + `"_clap_name"` サフィックス)
- ヒットポイント (FragTrap)
- エネルギーポイント (ScavTrap)
- 攻撃ダメージ (FragTrap)
- `attack()` (スカブトラップ)

**DiamondTrap**は、その親クラスが持つ特別な機能に加え、独自の特別な能力を持つことになります。

`void whoAmI()`。

このメンバ関数は、その名前と **ClapTrap** 名の両方を表示する。

もちろん、**DiamondTrap**の**ClapTrap**サブオブジェクトは、一度だけ作成されます。そう、仕掛けがあるのだ。

もう一度、プログラムにテストを追加してください。



Wshadow と -Wno-shadow コンパイラフラグをご存知ですか？



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。