# オブジェクト指向と
# ゲームプログラミング

## ○　C++編 － 第２回　クラス　○

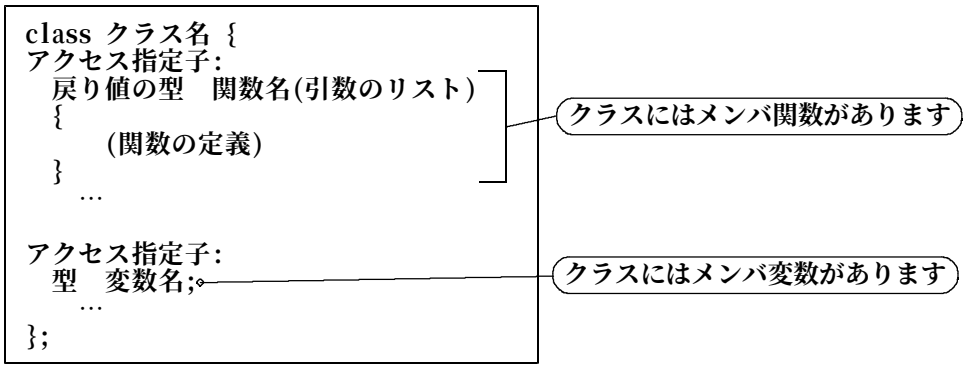
### クラス

クラスとは、動作と属性をまとめたものに名前を付けたものです。オブジェクト指向設計で洗い出されたオブジェクトは、ソースコード上でクラスとして定義されます。オブジェクト指向プログラミングでは、クラスを定義することが基盤となっていますが、ソースコードを記述しながら動作と属性をまとめていくのではなく、設計の行程でオブジェクトすなわちクラスを洗い出すことが先決であり、クラスの動作と属性を洗い出すことが後になることに注意しましょう。

クラスは、オブジェクトを作り出す方法を定義したものであり、実体を持つわけではありません。オブジェクトは、クラスのインスタンス(実体)として作り出されます。

### クラスの定義

クラスの定義には、クラスの属性となるメンバ変数(またはデータメンバ)の定義と、クラスの機能となるメンバ関数の定義があります。特殊なメンバ関数としてコンストラクタやデストラクタを定義することもあります。クラス定義の書式は、以下のようになります。




▲クラスの宣言

クラスの定義は、classというキーワードの後にクラス名を指定し、その後にある{と}で囲まれたブロックの中に、クラスの持つメンバ変数、メンバ関数、およびコンストラクタとデストラクタを定義します。メンバ変数の定義とは、メンバ変数となる変数や定数を記述することです。メンバ関数の定義とは、メンバ関数のプロトタイプを示したり、メンバ関数の処理内容を記述することです。

以下のクラスは、ゲームのキャラクターを表現するCharacterクラスです。ここでは、説明を簡単にするため、メンバ変数は、「ｘ座標」「ｙ座標」「アニメーション番号」で、メンバ関数は「移動」「アニメーション」だけにしています。コンストラクタとデストラクタはありません。継承、ポリモーフィズムも使っていません。

キャラクター

```
属性：
   ｘ座標(double)
   ｙ座標(double)
   アニメーション番号(int)

動作：
   移動
   アニメーション
   (属性の初期化)
```

Characterクラス

```
class Character {
public:
    void Move(const double add_x, const double add_y);
                                    // 移動
    void Animation();               // アニメーション

private:
    double   m_PosX;                // ｘ座標
    double   m_PosY;                // ｙ座標
    int      m_AnimeNo;             // アニメーション番号
};
```

Characterクラスは、double型の変数m_PosX, m_PosYとint型の変数m_AnimeNoを属性として持っています。このようなクラス内部のデータがメンバ変数です。また、このクラスはMove, Animationという２つの動作(関数)を持っています。このようなクラス内部の関数がメンバ関数です。

## アクセス指定

クラスのメンバは、必ずアクセス指定を行います。キーワードpublicより下にあるメンバが公開、キーワードprivateより下にあるメンバが非公開であることを意味しています。public, privateは、クラス(あるいは構造体)の文法を壊さない位置ならどこでも、何度でも記述することができます。

```
class クラス名 {
public:
    公開するメンバ

private:
    非公開なメンバ
};
```

クラスの宣言は、構造体の宣言によく似ています。C++では、classとstructでクラスと構造体を区別しますが、それ以外はクラスと構造体でまったく同じ文法が使用できます。違いはデフォルトのアクセス制限で、クラスのデフォルトはprivate、構造体のデフォルトはpublicとなっています。

## カプセル化

クラスではメンバの公開および非公開が、バグを減らす重要な役割を果たすことになります。

public以降に書いたメンバは「公開」となりますが、その意味は「それらのメンバはクラスの内でも外からでも直接アクセスできる」ということです。クラスの外とは、キーワードclassの後のカッコ{}で囲まれた部分の外側です。たとえば、ほかのクラスやグローバル関数(クラスに属していない関数)などは、クラスの外になります。前述のCharacterクラスでは、Move関数とAnimation関数はpublic(＝公開)であるため、クラスの外でも呼び出すことができます。

一方、privateの下に書いたメンバは「非公開」になります。その意味は、クラスの内部でしかアクセスできないということです。前述のCharacterクラスでは、変数m_PosX, m_PosY, m_AnimeNoがprivate(＝非公開)であるため、ほかのクラスやグローバル関数などといったCharacterクラスに属さないものからアクセスしようとすると、コンパイルエラーになります。
Characterクラスでは、メンバ変数がすべて非公開になっています。これは、Characterクラスを使うプログラマに「メンバ変数を外部から直接アクセスしてはならない」という制約を課していることになります。
なぜ、このように面倒なことをするのかというと、バグを減らすという意味があります。このような制約を課す方法は、不便に感じることも多々あります。しかし、メンバ関数をとおして間接的にしかアクセスさせないようにし、さらにエラーチェックを厳重にすることにより、「非現実的な操作」や「不正な操作」ができなくなり、バグを減らすことができます。メンバ変数を外部から直接変えられるようにしておくと、後々とんでもない数値を代入するといった操作から不具合が発生するかもしれません。クラスを設計する人とクラスを利用する人が異なる場合もあるからです。

このように、データを内部に置いて、外部から触れさせない仕組みをカプセル化(データ隠蔽または情報隠蔽)といいます。メンバ変数は、原則として非公開にします。メンバ変数への操作はメンバ関数をとおして間接的に行うように設計します。

また、すべてのメンバ関数を公開しなければならないということはありません。クラスの外部からまったく利用せず、クラスの内部だけで呼び出されるだけの関数(クラスの作業用の関数など)をわざわざ公開にする必要はありません。このような場合は非公開にし、隠蔽しておきます。

クラスを設計する段階で適切にprivateとpublicを分類しておけば、ほかの人がそのクラスを利用しても、不具合の発生しにくいプログラムが作成できるわけです。

## メンバ変数の定義

メンバ変数は、クラスの持つ属性であり、クラスの中でメンバ関数の外に記述された変数や定数として定義されます。メンバ変数は、クラスの持つメンバ関数が読み書きすることを想定しています。
メンバ変数の定義は、データ型を指定して変数や定数を宣言するだけです。定数の場合には、データ型の前にconstを指定するか、列挙体(enum)を利用します。constは値を変更できない変数(すなわち定数)であることを示し、コンストラクタ初期化子で初期値を設定した後は、いっさいの変更を行うことができません。
メンバ変数は、宣言と同時に初期値を与えることができません。初期値の設定は、コンストラクタの処理として行うのが一般的です。

メンバ変数のデータ型に構造体やほかのクラスを指定することもできます。この場合、ほかのクラスを利用していることになりますが、継承ではありません。メンバ変数のデータ型がほかのクラスになっているだけです。このような「持っている」関係を集約または所有と呼びます。

## メンバ関数の定義

メンバ関数は、クラスの持つ動作であり、クラスの中にメンバ関数のプロトタイプまたは処理内容を記述します。
C++の慣習では、メンバ変数を変更しないメンバ関数の後ろに、キーワードconstをつけます。このconstは「誤ったメンバ変数の変更」を防ぐ働きをします。このキーワードのついた関数内でメンバ変数を書き換えると、コンパイルエラーになります。

メンバ関数は、クラスの内部でも外部でも定義することができます。クラスの内部でメンバ関数を定義する場合、次のように記述します。

```
class Character {
public:
    // 移動
    void Move(const double add_x, const double add_y)
    {
        m_PosX += add_x;
        m_PosY += add_y;
    }

    // アニメーション
    void Animetion()
    {
        m_AnimeNo++;
        if(MAX_ANIME <= m_AnimeNo)
            m_AnimeNo = 0;
    }

    // アクセス関数
    double GetPosX() const { return m_PosX; }   // ｘ座標取得
    double GetPosY() const { return m_PosY; }   // ｙ座標取得

private:
    double   m_PosX;            // ｘ座標
    double   m_PosY;            // ｙ座標
    int      m_AnimeNo;         // アニメーション番号

    enum { MAX_ANIME = 5 };     // アニメーション最大数
};
```

クラス内部で定義したメンバ関数は、インライン関数として扱われます。インライン関数とは、「関数のコードを独立して作り、関数を使う場所ではそれを呼び出す」のではなく、「関数を使う場所にその関数のコードを展開する」というものです。インライン関数を使うと、実行可能ファイルは大きくなる傾向にありますが、「関数の呼び出し」と「呼び出された場所に戻る」というオーバーヘッド(余分な処理)がなくなるぶんだけ実行速度が向上します。
どんな関数でもインライン展開されるわけではありません。一般に、値の代入、簡単な演算、値の返却といった短い関数はインライン向きですが、複雑な処理をする関数は、コンパイラが「インライン展開できない」と拒否することがあります。

クラスのメンバ数が多い場合や大きな関数の場合、クラス内部で直接定義してしまうと、非常に大きく扱いにくいソースコードになってしまいます。これを避けるため、C++ではクラスの外部(ほかのソースファイル)でメンバ関数を定義できるようになっています。
クラスの外部でメンバ関数を定義する場合、通常はクラスの宣言をしたヘッダファイル(.hまたは.hpp)と、メンバ関数の処理内容を記述したソースファイル(.cppまたは.cxx)に分割して作成します。Characterクラスの場合は、ヘッダファイルとソースファイルを以下のように定義します。

―Character.h―
```
class Character {
public:
    void Move(const double add_x, const double add_y);  // 移動
```

```
        void Animetion();                                   // アニメーション

        // アクセス関数
        double GetPosX() const { return m_PosX; }           // ｘ座標取得
        double GetPosY() const { return m_PosY; }           // ｙ座標取得

    private:
        double   m_PosX;            // ｘ座標
        double   m_PosY;            // ｙ座標
        int      m_AnimeNo;         // アニメーション番号

        enum { MAX_ANIME = 5 };     // アニメーション最大数
    };
```

－Character.cpp－

```
    #include "Character.h"

    // 移動
    void Character::Move(const double add_x, const double add_y)
    {
        m_PosX += add_x;
        m_PosY += add_y;
    }

    // アニメーション
    void Character::Animetion()
    {
        m_AnimeNo++;
        if(MAX_ANIME <= m_AnimeNo)
            m_AnimeNo = 0;
    }
```

## スコープ

C++でのスコープについて簡単に説明します。C言語では、変数名の検索は「自動変数→関数内の静的変数→ファイル内の静的変数→外部変数」の順で行われますが、C++では"クラスの変数"の検索が加わり、「自動変数→関数内の静的変数→メンバ変数→ファイル内の静的変数→外部変数」の順となります。

変数名の衝突はしばしば根の深いバグの原因となりますが、C++ではさらにネストが深くなるため、注意深い名称設定が必要です。

## スコープ解決演算子

CharacterクラスのAnimation関数の正式な名前は「Character::Animation」です。先頭のCharacterというクラス名は、Animation関数がCharacterクラスのメンバであることを示しています。クラス名の付かないAnimation関数は、グローバル関数となり、CharacterクラスのAnimation関数とはべつの関数と見なされます。

「::」はスコープ解決演算子と呼ばれ、これによってメンバがどのクラスに属しているのかがわかります。::演算子は、グローバル関数を指定するときも使うことができます。この場合、クラス名の部分には何も書きません。たとえば、グローバル関数Animationを参照する場合は、「::Animation()」と記述します。

グローバル関数のスコープ解決演算子は通常は使っても使わなくてもかまいませんが、次のように同名の関数が２つある場合は必要になります。

```
    // グローバルのAnimation関数
    void Animation()
    {
        (処理内容省略)
    }
```

```
class Character {
public:
    // 移動
    void Move(const double add_x, const double add_y)
    {
        (移動処理：省略)
        Animation();    // グローバル関数Animationの呼び出し？
    }

    // アニメーション
    void Animation() { m_AnimeNo++; }

private:
    double   m_PosX;            // ｘ座標
    double   m_PosY;            // ｙ座標
    int      m_AnimeNo;         // アニメーション番号
};
```

Characterクラスのメンバ関数Moveの内部からグローバル関数Animation()を呼び出したいとします。このような場合、Animation()に::演算子をつけないでこれを呼び出すと、Character::Animation()が呼び出されてしまいます。つまり、メンバ関数が優先されてしまうのです。次のように::演算子を付け加えると、期待どおりの結果が得られます。

```
class Character {
public:
    // 移動
    void Move(const double add_x, const double add_y)
    {
        (移動処理：省略)
        ::Animation();  // グローバル関数Animationの呼び出し
    }

    // アニメーション
    void Animation() { m_AnimeNo++; }

private:
    double   m_PosX;            // ｘ座標
    double   m_PosY;            // ｙ座標
    int      m_AnimeNo;         // アニメーション番号
};
```

## 練習問題

**1　以下のクラスのなかからすべてのメンバ変数とメンバ関数を答えましょう。**

```
class Character {
public:
    void Move(const double add_x, const double add_y);  // 移動
    void Animation();                                    // アニメーション

private:
    double   m_PosX;                                     // ｘ座標
    double   m_PosY;                                     // ｙ座標
    int      m_AnimeNo;                                  // アニメーション番号
};
```

**・メンバ変数**

**・メンバ関数**

**2　１のクラスのなかでCharacterクラスの外部からアクセスできないメンバを答えましょう。**

**3　以下の文章の内容が正しい場合には○、間違っている場合には×を付けましょう。**

**(１)public指定されたメンバには、クラスの外からアクセスすることができる。**
**(２)private指定されたメンバには、クラスの外からアクセスすることができない。**
**(３)メンバ関数の中で定義されている変数をメンバ変数と呼ぶ。**
**(４)メンバ変数は、オブジェクトが作成されていないとアクセスすることができない。**
**(５)メンバ変数を持たず、メンバ関数のみのクラスを定義することはできない。**
**(６)メンバ変数の定義でconstを指定すると、定数のメンバ変数となる。**
**(７)メンバ変数の型に、ほかのクラスを指定することができる。**
**(８)あるクラスがほかのクラスを持っている関係を委譲と呼ぶ。**

**4　クラスの定義において、メンバ変数をprivate指定にする理由を説明しましょう。**

**5　次のように整数値の座標を表すCPointクラスを作成しましょう。ただし、座標の範囲はｘが０～６４０、ｙが０～４８０となるようにしましょう。**

**メンバ変数**
**ｘ座標…pos_x**
**ｙ座標…pos_y**

**メンバ関数**

```
void SetX(const int px);      // ｘ座標を設定する
void SetY(const int py);      // ｙ座標を設定する
int GetX();                   // ｘ座標を得る
int GetY();                   // ｙ座標を得る
```