Chapter 2

クラス

C++と C の大きな違いはクラスという概念をサポートしているかどうかになります。このチャプターではオブジェクト指向プログラムの肝となるクラスについて見ていきます。

2.1 クラスの概念

クラスというのは、プログラムで実装する必要のある機能をモデル化してプログラムの可読性、保守性、再利用性を向上させるために用いられる概念です。昔はクラスとは現実世界のモノをモデル化するものだという説明が入門書などに記述されていたのですが、これは間違った説明で、これがクラスを理解できない原因になっているとまで言われていたものです。このような話を聞いたことがある人は頭から叩き出してください。

2.1.1 可読性

可読性とはコードの読みやすさということです。ソフトウェアの開発というのは複数人で行われるもので、一人で開発を行うということは稀です。そのため、他人のソースコードを読むということが必ず発生します。そのためコードの読みやすさというのはソフトウェア開発を潤滑に進めるために重要なファクターになります。また、ソースコートというものは他人が読むだけではありません。間違いなく自分の書いたコードを一番多く読むのは自分自身です。昨日の自分は他人という言葉がソフトウェア開発の世界ではよく使われます。可読性の高いプログラムを書くということは、他人のためというよりも自分のためという側面の方が強くなります。

ではなぜクラスを使うと可読性が上がるのでしょうか？例えば、あなたがゲーム開発でプレイヤーの処理を実装していると考えてみて下さい。クラスを扱うことに少し慣れている人なら class Player{}などのようにプレイヤーのクラスを作ることをすんなりと考えられるはずです。なぜなら、Player をクラス化しておけば、プレイヤーの処理は基本的には Player クラスのソースを読めばいいし、そこに書けばいいことが分かるからです。大規模な開発ならソースコードがウン十万行あることは珍しくありません。そこでクラスの概念がない言語(C 言語とか)で開発を行うことを考えてみてください。恐らくあなたはプレイヤーの処理がどこに書かれているのかを把握することすら困難になるでしょう。

2.1.2 保守性

保守性というのはプログラムのメンテナス、拡張のしやすさという意味です。例えばアクションゲームを作っているケースを考えてみてください。そのゲームでは当初は A ボタンが押されるとプレイヤーがジャンプするという仕様でした。しかし、開発が半年ほど経過し

て中盤に差し掛かった頃にクライアントから、ジャンプ中に A ボタンが押されたら 2 段ジャンプができるようにして欲しいという要望が来ました。そのためプレイヤーのプログラムを変更する必要に迫られます。プレイヤーのジャンプのプログラムを作成したのは数ヶ月前になり、コードの場所の記憶も薄れてきています。しかし、このプロジェクトが C++ のようなクラスが使えるプログラム言語で作成されているのであれば、すぐに Player クラスが見つかることでしょう。そしてすぐにプログラマは Player クラスの Jump 関数を見つけ出して、その関数のプログラムを変更すればいいことを思い出すはずです。もしこれが C 言語のようなクラスが使えないプログラム言語で作成されていたのであれば、プログラマはウン十万という膨大なソースの中からプレイヤーの処理を探すことになるでしょう。

もちろん C 言語を使っていても、ソースファイル名に Player.cpp などのような分かりやすい名前をつけていれば該当する処理を探し出すことは容易です。しかし、クラスという機能をモデル化する概念がない言語では往々にして、プレイヤーの処理はいたるところに記述されていきます。そしていつかそれを把握するのが困難になります。とくに記憶が薄れてきたころに。

2.1.3 再利用性

オブジェクト指向言語で言われる再利用とはプログラムを再利用と、設計の再度利用をさします。オブジェクト指向がもてはやされた時に、クラスや継承を使用するとプログラムの再利用性が向上して開発効率が高くなると言われていました。しかし現実問題、作成したクラスの再利用は一部のライブラリなどを作成するプログラマが考えることで、アプリケーション層のプログラムを書いている場合は再利用性をそこまで考える必要はありません。再利用されるかどうかも分からないプログラムで、再利用されたときのことを考えても無駄になります。オブジェクト指向のキモは可読性と保守性の向上です。ただし、設計の再利用は重要です。これがデザインパターンと呼ばれるものです。デザインパターンとは先人の考えた優れた設計をカタログ化して再利用しようという考えです。

2.2 C++でのクラスの作成の仕方

では 3D アクションゲームでプレイヤークラスを作成する場合を考えてクラスを作成してみましょう。まず、クラスの名前は Player などといった名前にすることが思いつくかと思います。

```
class Player{
};
```

次はメンバ変数を考えてみましょう。3D アクションゲームなので、当然プレイヤーは 3D 空間場で位置を表すための変数を保持しているはずです。そのため、プレイヤーに位置を表す変数を保持させてみましょう。

```
class Player{
private:
    float   m_positionX;    //X 座標
    float   m_positionY;    //Y 座標
    float   m_positionZ;    //Z 座標
};
```

また、このゲームは体力という概念があり、敵から攻撃を受けると体力が減るという仕様があります。そのためプレイヤーは体力というメンバ変数を保持しているはずです。

```
class Player{
private:
    float m_positionX;      //X 座標
    float m_positionY;      //Y 座標
    float m_positionZ;      //Z 座標
    int m_hitPoint;         //体力
};
```

また、画面にプレイヤーを描画する必要があるため、Draw というメンバ関数も必要なはずです。そして、ユーザーのキー入力によりプレイヤーは移動するため、Move というメンバ関数も必要になるでしょう。

```
 class Player{
    float m_positionX;      //X 座標
    float m_positionY;      //Y 座標
    float m_positionZ;      //Z 座標
    int m_hitPoint;         //体力
 public:
    void Move()
    {
        //移動処理。
    }
    void Draw()
    {
        //描画する処理を記述する。
    }
 };
```

ではこの Player クラスを使用して、簡単なゲームプログラムを書いてみましょう。

```
int main()
{
    Enemy enemy;            //敵
    Player player;          //プレイヤー
  //ゲームループ
  while(true){
      enemy.Move();  //敵の移動処理。
      player.Move();  //プレイヤの移動処理
      enemy.Draw();  //敵の描画処理。
      player.Draw();   //プレイヤの描画処理。
      WaitVSync();   //垂直同期待ち。おまじない。
  }
}
```

このクラスの作成の仕方の流れは私の思考をトレースした一例でしかありません。このような流れでクラスを作成する必要があるわけではないので注意してください。

実習

Lesson_01 はクラスを使用せずにプレイヤーの処理を記述しています。このプログラムのリファクタリングを行い、プレイヤークラスを作成しなさい。

Lesson_01/Player.h の#define PLAYER_CPP を無効にした状態で同じ挙動になっていたら完成しているものとする。