中級C#言語(ライティング)

MASA

2011年6月

1　はじめに

1.1　おことわり

C#が使えるようになることを目標として説明しているので、マニアからすれば欠かせない要素とかは結構端折っていますし、説明も胡散臭かったりしますがツッコミは無用です。

1.2　目安

C言語の構文はだいたい理解している人に、C#をある程度使えるレベルまで学んでもらうことが目標です。

1.3　C#とはなにか

C#とは、C,C++ときてさらに++した言語です[1]。基本的な文法はC言語から受け継ぎつつそこに新しい様々な概念を盛り込んだ高機能な言語で、とくにその柱となっているのがオブジェクト指向という概念です。オブジェクト指向を中心に据えているためコードの見た目としてはJava等に近いものになるでしょう。

1.4　オブジェクト指向とは何か

オブジェクト指向について詳しく語ろうとするとそれだけで本が何冊も書けてしまうのでここではC#を使うのに最低限のことだけ述べます。オブジェクト指向プログラミングとは、プログラムを物(オブジェクト)単位で作ることで分かりやすくしようという手法です。ここでは私が大好きなシューティングゲームを例に説明しましょう。

[1] #という記号が++++を意味している

まず、シューティングゲームの画面を思い浮かべましょう。色々なものがあります。主人公の機体、敵、自分の弾、敵の弾、アイテム … 。これら一つ一つを個別の「物」として扱うのがオブジェクト指向です。オブジェクト指向では機体や敵、弾ひとつひとつの物のことを「インスタンス」と呼び、また「敵の弾」「アイテム」といった物の種類のことを「クラス」と呼びます。弾を撃つときは、「敵の弾」クラスのインスタンスを生成するわけです。

さて、それでは敵クラスについて詳しく考えてみましょう。敵のインスタンスには、自分のいる位置情報や残り体力の情報、また移動する機能や弾を撃つ機能が必要です。インスタンスにどのような情報を持たせるか、どのような機能を持たせるか、ということを記述するのがクラスを定義するということです。逆に言えば、ある物が持っている情報や機能をまとめたものがクラスであるということになります。

2 実践編

2.1 コードの実行方法

実際に講座に入る前に、プログラムの実行方法についても書いておきましょう。VisualC#やVisualStudioを使ってデフォルトのまま作ると自動で色々なコードが生成されて勉強するには良くないので、原始的にいきます。

「空のプロジェクト」を作成→ソリューションエクスプローラを開き、プロジェクトを右クリック→追加→新しい項目→コードファイル→名前を指定&作成→全て保存

これで作成したコードファイルに実際にプログラムを書いていきます。

2.2 いつものアレ

さあ、プログラミングを学ぶ最初の一歩といえばやっぱりアレですね、アレ。早速アレのコードを掲載しましょう。

ソースコード 1: Hello.cs

```
using System;

class Hogehoge
{
        public static void Main()
        {
                Console.WriteLine("Hello World!");
        }
}
```

こんなコードです。これを実行すると、黒いコマンドプロンプトの画面が一瞬出て消えます。

このコードのように、C#のプログラムはクラスを定義してそのメンバ関数 (あとで詳しく) として記述します。C 言語のように「いちばん外側」に関数や変数定義を書くことは出来ず、関数や変数は必ず何かしらのクラスに所属しなければいけません。

2.3 構文など

基本構文はだいたい C 言語と共通していますが、やや異なっている部分もあります。説明は適当に端折って、コードメインで書いておきます。なお、コードには大して意味はありません。

2.3.1 変数宣言、代入

ソースコード 2: 変数

```
int hoge;
double hogera = 3.14;
char huga = 'A', piyo = 'X';
string foo = "bar";
hoge = -5;
foo = "foobar";
```

C 言語と同じように書けますが、C 言語のように変数宣言はブロックの先頭でしか出来ない、というような制約はありません。string 型は C 言語にはなかった型で解説すべき事項はありますが、ここでは単に文字列を入れられる型、と理解しておけば十分です。

2.3.2 ブロック

構文、と呼ぶかは微妙ですが、中括弧 { } で囲われた中をブロックと呼びます [2]。ひとつのブロックはひとつの文として扱われるので、条件分岐などで複数の文をまとめて書くときに使います。

2.3.3 条件分岐

ソースコード 3: 条件分岐

```
if(hoge == 0)
{
```

[2] 正式ではないかも

```
        hoge += 6;
}
else if (hoge < 5)
{
        hoge += 2;
}
else
{
        hoge -= 10;
}
if (hogera > 1) hogera = hogera * 2;
if (true) piyo = 'Z';
```

やはりだいたい同じですが、C#では条件式として bool 型[3]のものしか取れないため、if (1) というように数値を条件として渡すことは出来ません。

2.3.4 ループ

ソースコード 4: ループ

```
for(int i = 0; i < 5; i++)
{
        hoge *= 2;
}
while (hoge > 0)
{
        hoge--;
}
do
{
        if ( hoge == 0)
        {
                break;
        }
        else if (hoge < 0)
        {
                hoge++;
                continue;
        }
```

[3]true(真) と flase(偽) の 2 種類だけの値を取る型

```
        hoge -= 2;
}
while (true);
```

条件式がbool型である他は変わりませんが、「ラベルつきbreak」というものが使えたりするので興味があったら自分で調べてください。

2.3.5 switch文

ソースコード 5: switch

```
switch(hoge)
{
        case 0:
                piyo = 'Z';
                break;
        case 1:
                piyo = 'O';
                break;
        case 10:
                piyo = 'T';
                break;
        default:
                piyo = ' ';
                break;
}
```

C言語と大体同じですが、各caseごとにbreakを書かなければならない[4]、defaultを書いていないでどのcaseにも当てはまらないとエラーになる、string型も条件として使える、といった細かな違いがあります。後に出てくる列挙体[5]と組み合わせて使うと相性がいいです。

2.4 クラスの書き方 初級編

続いてはC#の特徴であるクラスの書き方です。まだ基本として配列変数やforeach文が残っていますが、事情により後回しします。最初の方でも述べたとおり、C#のコードはクラスの中に書かれます。そのため、C#でプログラミングするにはクラスの書き方を多少知っている必要があります。まずは最低限必要なことを解説します。

ソースコード 6: Hello.cs

```
using System;
```

[4] fall-throughができない。breakのかわりにreturnなどで抜け出してもいい。
[5] enum

```
class Hoge
{
        public static void Main()
        {
                Man piyo;
                piyo = new Man("piyopiyo");
                piyo.SayHello();
                Man huga = new Man("huga-");
                huga.SayHello();
        }
}

class Man
{
        public string name;
        public Man(string n)
        {
                name = n;
        }
        public string MakeText()
        {
                return "Hello, I am " + name + "!";
        }
        public void SayHello()
        {
                string t = MakeText();
                Console.WriteLine(t);
        }
}
```

先ほど作ったはずのソースコードに上のように記述して実行してみると、黒い画面が一瞬出て消えます。

さて、class Hoge のほうは一旦置いておいて、class Man を見てください。C#では class (クラス名) { (クラスの情報) } という構文でクラスを書きます。なお、最後の中括弧を閉じる部分にセミコロンは不要です [6]。クラスは変数 (メンバ変数と呼びます)(ここでは string 型の name 変数) と関数 (メンバ関数と呼びます)(ここでは string をひとつ引数に取

[6] C++ネタ

るコンストラクタ (後述)、返り値なしの SayHello 関数と、返り値 string 型の MakeText 関数) を持ちます。

メンバ変数の宣言はクラスのブロック内であれば前章に出てきたようにして書け、宣言と同時に初期化することも可能です。メンバ関数の定義も同じくクラスのブロック内に書き、基本的には C 言語での関数定義と同じように記述します。ただし、クラスのメンバ変数・メンバ関数を書く際はその前にいくつか修飾子を付けることがあります [7]。このサンプルでは全てに public 修飾子 [8]を付けていますが、これは意味は気にせずオマジナイとして書いておいてください。

それでは Man クラスのメンバ関数を見ていきます。最初の Man 関数 (っぽいもの) は飛ばします。

次の MakeText 関数は、引数 [9]なしで string 型を返り値としている関数です。返り値は public 型ではないので勘違いしないでよね!さて、この関数で行っている処理は、”Hello, I am ”という文字列に name 変数の値である文字列を結合し、さらに”!”という文字列を結合して返しています。このように、メンバ関数からメンバ変数を読み書きできます。また本筋とは関係ありませんが string 型の値は + 記号で連結できます。

続いての SayHello 関数は、引数、返り値ともにありません。この関数では、まず string 型のローカル変数 t に MakeText 関数の結果を代入して、それを Console.WriteLine という外部の関数を使ってコンソール画面に表示しています。このように、関数内で変数を宣言するとローカル変数というものになります。ローカル変数はその変数を宣言した関数内部のみで使える [10]変数です。

さて、後回しにした Man 関数もどきを見てみます。これは一見関数と似ていますが、返り値の型が指定されていないので関数とは別物です public は型名ではなく修飾子。これはコンストラクタというもので、クラスのインスタンスが生成されるとき (あとで述べます) に実行される特別な関数もどきです。コンストラクタの書き方は、

public クラス名 (引数リスト) 処理

という書式で、返り値の型を書かないこと、名前がクラスの名前と同じであること、が特徴です。コンストラクタでは普通初期化処理を行います。この場合では、string 型の引数 n をとって、その値をメンバ変数 name に代入しています。

これをふまえて class Hoge の定義を見てみます。このクラスにはメンバ変数がなく、Main というメンバ関数を持っています。C 言語で main 関数からプログラムが実行されたように、C#では Main 関数からプログラムが実行される (エントリポイントと言います) のです。ただ C#はクラスの中にしか関数を書けないため Main 関数を適当なクラスのメンバ関数として定義しているわけです。Main 関数の書き方は、このソースコードの

[7]関数内でのローカル変数定義では付けません

[8]簡単にするため

[9]「いんすう」ではなく「ひきすう」

[10]正確にはローカル変数はブロックごとに使える範囲=スコープが決まる。例えば if 文に続くブロック内で宣言したローカル変数はそのブロックの外では使えないし、そのブロック内に更に別のブロックがあればそのブロックでは使える

ように

public static void Main() 処理の中身

と適当なクラスの中に書くのが一般的です。返り値や引数を持たせることも可能ですが、ここでは説明しません。なお、Main という名前でこのような形の関数であればコンパイラは勝手にプログラムのエントリポイントだと認識するのでどのクラスに書いても構いませんが、逆に言えばプログラムのエントリポイントにするつもりがなくても Main という名前の関数だとエントリポイントだと認識されかねないので Main という名前のメンバ関数は作らないほうが無難です。

続いて Main 関数の処理内容を見ていきましょう。まず Man 型のローカル変数 piyo を宣言しています。はじめの方でクラスのことを「物の種類」と書きましたが、このように変数の型としてクラスを使えるのです。さらには、これまで使ってきた int や string なども実は特殊なクラスなのでした深く考えると難しい問題が出てくるので、混乱したくなければ深く考えないほうがいい。次に、Man クラスのインスタンスを生成して piyo に代入しています。インスタンスを生成するときは、

new クラス名 (コンストラクタに渡す引数)

と書きます。その結果作られたインスタンスを変数に代入して扱っているわけです。クラスを生成するときには、先に書いたようにそのクラスのコンストラクタが実行されます。よって、このコードでは Man クラスの Man 関数もどきが引数 ”piyopiyo”を渡されて呼ばれ、生成されたインスタンスの name 変数に”piyopiyo”が代入されています。

次の行では Man 型の piyo 変数を使って、Man クラスの SayHello 関数を呼んでいます。インスタンスが持っているメンバ変数やメンバ関数を参照するには、

インスタンスが代入されている変数名. 参照したい変数/関数の名前

というように . を使って書きます正確には「インスタンスが代入されている変数名」ではなくインスタンスへの参照であればいいので、new Man(”hoge”).SayHello(); というようなことも可能。

このように、クラスを定義→クラスの型の変数を定義→ new を使ってインスタンスを生成して変数に代入→変数を使ってクラスの変数、関数を操作する、という流れが C#でのプログラミングの基本です。

その後の 2 行では同様のことを行っています。Man 型の huga 変数を宣言と同時に Man クラスのインスタンスを代入しています。piyo 変数に入っているインスタンスと huga 変数に入っているインスタンスは別物なので、それぞれの name 変数には別の値 (”piyopiyo”と”huga-”) が入っていますし、SayHello 関数の結果もそれぞれで違います。

ところで、Main 関数の前に書かれている static という言葉、これは修飾子の一種です。static 修飾子が付いたメンバ変数やメンバ関数などは、「インスタンスではなくクラスに属するもの」になり、それぞれ静的変数、静的関数などと呼ばれます。インスタンスに属していないため、どのインスタンスから参照しても同一の存在であり、またインスタンスがなくても参照することが可能になっています。static な変数/関数を参照するには普通のメンバ変数/関数を参照するときと同様に

インスタンスの代入されている変数名.静的変数/関数名
という書き方も出来ますし、
クラス名.静的変数/関数名
というように書くことも出来ます。実は、Console.WriteLine というのも、既に定義されている Console クラスの静的関数である WriteLine 関数のことなのです。

2.5 配列と for 文、foreach 文

クラスの基礎を学んだところで今度は配列です。まずはサンプルコードから。

ソースコード 7: Hello.cs

```
using System;

class Hoge
{
        public static void Main()
        {
                int[] homu;
                homu = new int[5];
                for (int i = 0; i < homu.Length; i++)
                {
                        homu[i] = i * 2;
                }
                foreach (int item in homu)
                {
                        Console.WriteLine(item.ToString());
                }
        }
}
```

配列は C 言語とは結構違った使い方になっています。C#での配列はクラスの一種として作られているからです。

コードで見ると、まず int の配列型の変数 homu を宣言しています。型名 [] と書くことで、その型名の配列型のクラスとして扱われます。配列はクラスなので変数を宣言しただけでは使えず、また C 言語のように要素数も宣言の時点では指定しません。次の行で、int 型で要素 5 個 C 言語とは違い、配列の要素数は定数でなくても構わないの配列クラスのインスタンスを生成して変数 homu に代入しています。配列は特殊なクラスなので、インスタンスを生成するときに new int[5]() のように () をつける必要はありません。

配列を生成しただけでは値が設定されていない[11]ので、次の for 文で値を設定しています。配列の添字 (index) は、0 から始まり (要素数 - 1) までです。

for 文内の条件式に使われている homu.Length というのは配列の要素数 (ここでは 5) を表すプロパティです。プロパティというのはインスタンスが持つメンバ変数に似たもので、読み書き両用、読み込み専用、書き込み専用、というように操作を制限する機能を持ったものです正確に言うとかなり違うが簡単のため。配列の Length プロパティは読み込み専用なため値を代入しようとするとコンパイルエラーが出ます。

この for 文の条件では、i が 0 から homu.Length=5 よりも小さい間、つまり i が 0 から 4 まで実行されます。配列 homu の添字も 0 から 4 までなので、このように書けば配列の要素全てを扱うことが出来るわけです。

続いて foreach 文という C 言語にはない新しい構文が出てきました。foreach 文は、配列などの要素をひとつずつ取り出してそれを操作する、という機能を持った文です。。書き方は、

foreach(配列などの要素の型 適当な変数名 in 配列など)

というものです。このコードでは int 型の変数 item を宣言してそれに配列 homu の要素を 1 つずつ代入してブロック内の文を実行しています。foreach 文だと配列の要素数をいちいち意識しなくても全要素に対しての処理を簡単に書けるため便利ですが、変数に要素を代入しているというところにひとつトラップがあります。それは、変数は要素そのものではないため書きこむときに意図とは違う動作をする場合があるということです。具体的には、下記のコードは上手く動きません。

ソースコード 8: 失敗例

```
foreach (int item in homu)
{
        item = 1;
}
```

こうすると一見配列 homu の全要素に 1 が代入されそうですが、実際には配列 homu の要素の値は全く変化してません。この理由を書くと値渡しと参照渡し、というようなやや面倒な概念が出てくるので割愛します。とりあえずは「foreach の変数には代入できない」と覚えておくのがいいでしょう。上記のコードで配列の要素の初期化にわざわざ for 文を用いたのもこれが理由です。

さて、foreach 文の中で実行しているコードを見てみましょう。Console.WriteLine 関数の引数に item.ToString() というものを渡しています。先ほどちらっと述べたように int 型もクラスの一種であるためメンバ関数を持っています。ここでは item 変数は int 型なので、int クラスの ToString 関数を呼んでいるわけです。ToString 関数はその値を文字列と

[11] int の場合は初期値 0 が代入されている

して出力するという関数で、item の値が 2 であれば 2 という文字列が返されます [12]。

2.6 列挙体

書き忘れていたのでここで列挙体です。

ソースコード 9: 列挙体

```
using System;

enum MagicalGirl
{
        Madoka,
        Homuhomu,
        Sayaka,
        Mami,
        Anko,
}

class Hogera
{
        public static void Main()
        {
                MagicalGirl mg;
                mg = MagicalGirl.Homuhomu;
                switch (mg)
                {
                        case MagicalGirl.Madoka:
                                break;
                        case MagicalGirl.Homuhomu:
                                Console.WriteLine("boku ha homuhomu ha desu
                                break;
                        case MagicalGirl.Anko:
                                Console.WriteLine("umaui");
                                break;
                        default:
                                Console.WriteLine("dare?");
```

[12] ちなみに数値自体も int 型として扱われるので 12.ToString() ということも可能で、この場合戻り値は 12 という文字列です

```
                    }
                }
}
```

列挙体とは値に名前をつける物で、この場合はMagicalGirl.Madokaは0、MagicalGirl.Homuhomuは1という値が内部的に割りふられています。列挙体は定数として用いたり、switch文と組み合わせて使います。