第三回　オブジェクト指向の基本　～　メッセージとメソッド

1．メッセージを渡す

さて、前回の資料でオブジェクトには「クラス」という設計図があること、オブジェクトを生成するには new 演算子でクラスのインスタンスを生成する必要がある、という説明をしました。ここではオブジェクト指向における基本キーワード「オブジェクト」「メッセージ」のうち、もうひとつの「メッセージ」について解説していこうと思います。

前回四角形クラスとしてこのようなものを作ったのを覚えているでしょうか。

```
public class Rectangle
{
        int height;
        int width;
}
```

そしてこれのインスタンス（便宜上高さ7cm、幅5cmの四角形とします）を生成するときはこのように書きますね。

```
Rectangle RectA = new Rectangle();
```

ここで、前回「このままでは”高さ7cm、幅5cm”の情報がない」という話をしました。確かにそうです。・・・では、どうすればよいのでしょうか？答えは簡単で、情報がなければ情報を渡せばいいんですね。そして、実はこの**「情報を渡す」という行為がそのまま「メッセージ」の定義**に繋がるのです。※1では実際に「メッセージ」にあたるものを上に書いた Rectangle クラスに付け足してみます。

```
public class Rectangle
{
        int height;
        int width;
        public void SetValue(int h, int w){
                height = h;
                width = w;
        }
}
```

2 つのフィールド定義の下に何やら怪しげな部分が追加されましたね。とりあえず一つ一つのキーワードをじっくりと見ていきましょう。まず、「public」ですが、第一行目の public 同様「魔法の言葉」としてスルーします。※2次は「void SetValue」です。（なぜ二つのキーワードを同時に紹介するかと言うと、一個ずつよりもここは一括りにしたほうが意味を説明する上で都合がいいからです。）「SetValue」はそれが何をしたいのかを表す名前（これをメソッド名といいます）、「void」はその処理によりオブジェクトからの返答がないことを表しています。これを要約すると、**「SetValue という処理を行います。返答はいりません。」ということを意味**します。そして次の括弧「(int h, int w)」が SetValue 処理に渡したい値（引数）、を指します。そして{ }の中に書いた 2 行が実際に行われる処理を表します。

何となく勘のいい人は見当がついたかもしれません。Set は「当てはめる」Value は「値」を意味しています。つまり、SetValue は「値を当てはめる」ということです。そして、()に書かれた h, w はそれぞれ height と width に代入されています。このことから、**SetValue とは「高さ(h)、幅(w)の値を受け取ってそれを height と width に代入する」という処理を行っている**ことが分かると思います。なお、SetValue を void（オブジェクトからの返答なし）にしたのは、SetValue は値をセットする処理を行っているだけなので、

返答が必要ないからです。※3

このように定義された SetValue のようなものを**「メソッド」**といいます。この場合、「Rectangle クラスの SetValue メソッド」と呼ぶことが多いです。クラスは複数のメソッドを持つことが可能です。SetValue メソッドは void 型※4なので値を返さないものでしたが、例えば「その四角形の面積を求める」というメソッドが欲しいならば、Rectangle クラスの定義の中に次の行を追加しましょう。

```
public class Rectangle
{
        int height;
        int width;
        public void SetValue(int h, int w){
                height = h;
                width = w;
        }
        public int GetSize(){
                int size = height * width;
                return size;
        }
}
```

今度は void ではなく、int となっていますね。そして、最後の行に return というキーワードがあります。return は「返却する」という意味なので、これは int 型の値 size（高さ×幅）を返すということになります。さらに言えば、この GetSize メソッドはメソッドの呼び出し元に**「メッセージを返している」**ということになります。メッセージは「情報を渡すこと」と上に書きましたが、**渡すだけでなく、渡されることもまたメッセージ**なのです。※1なお、面積を求めるときはあらかじめセットされている height と width の値を使えばいいだけなので、()の中には何も書く必要はありません。（引数がないメソッド）

このように、オブジェクトごとに必要な処理（メソッド）を追加していけるのがオブジェクト指向の便利なところです。

2. メソッドを使ってみよう

上に書いた、SetValue メソッドと GetSize メソッドが追加された Rectangle クラスを使って実際にプログラムを作って見ましょう。話をシンプルにするため擬似コード（C 言語風）で書きます。

```
Rectangle RectA = new Rectangle();
Rectangle RectB = new Rectangle();

RectA.SetValue(7, 5);
RectB.SetValue(4, 8);

int sizeA = RectA.GetSize();
int sizeB = RectB.GetSize();
printf("四角形 A の面積は %d ¥n", sizeA);
printf("四角形 B の面積は %d ¥n", sizeB);
```

```
実行結果
四角形 A の面積は 35
四角形 B の面積は 32
```

こんな感じになります。前回書いたとおり、最初に new 演算子を用いて Rectangle クラスのインスタンスを生成します。この段階では RectA、RectB どちらともまだ height、width の値が定まっていません。次に RectA.SetValue(7, 5)と書きます。これを要約すると、**「RectA というインスタンスの SetValue メソッドを呼び出して、さらに引数として 7, 5 を渡す」**という処理を行っています。これにより RectA の height は 7、width は 5 になりました。RectB も同様の処理を行っています。次に、SizeA という int 型の引数に RectA の面積値を渡すために、RectA.GetSize()と書きます。**GetSize メソッドは int 型の戻り値として height×width を返すので、変数 SizeA には 35 という値が入る**はずです。そして最後に printf という関数を使って、※5sizeA と sizeB の値をそれぞれ画面に表示します。その結果、前ページの「実行結果」のような文字が出力されます。

これがオブジェクト指向による基本的な書き方です。ちなみに、同じ処理をするために Rectangle クラスを一切使わずに書こうとするとこうなります。

```
int RectAheight = 7;
int RectAwidth  = 5;
int RectBheight = 4;
int RectBwidth  = 8;

int sizeA = RectAheight * RectAwidth;
int sizeB = RectBheight * RectBwidth;
printf("四角形 A の面積は %d ¥n", sizeA);
printf("四角形 B の面積は %d ¥n", sizeB);
```

```
実行結果
四角形 A の面積は 35
四角形 B の面積は 32
```

・・・まぁ、これでも処理が分からなくもないです。しかし上に書いた Rectangle クラスを用いたものと比べると、変数の数が増えているし、変数名も長くなりがちです。面積も今回は高さ×幅という簡単な式だからまだ分かるのですが、もっと複雑な式になると、A, B と書くたびにその都度新しく式を書いて追加しなければならないのです。もし三角錐の体積を計算する、それも100個分やらなければいけない・・・となった場合、クラスを用いない場合は100個分の三角錐の体積を求める計算式を作らなければならないのです。それはかなり面倒ですね。オブジェクト指向では、**処理に必要な「モノ」を部品単位にわけ、処理をクラスの中にメソッドとして記述することで複雑な処理も一度書くだけで済む**というメリットがあるのです。

3. モノには個性がある

ところで上に書いたコードについてですが、当たり前のように RectA RectB とインスタンスを作りました。そして当たり前だとは思いますが、「RectA の height」と「RectB の height」は別物です。（違う四角形の”高さ”情報は異なるもの、当然ですね。）これをややこしい言い方にすると、「インスタンスごとにそれぞれ異なるフィールドを持っている」ということです。

四角形 A（RectA）と四角形 B（RectB）では何が違うのでしょうか。言うまでもなく「存在そのもの」が異なるもの、噛み砕いた

表現をすると「全く別物」です。では「別物である」というのはどうやって判断されるでしょうか。ひとつは変数名です。当然 RectA と RectB は別の変数だし、今さら考えるまでもなく別のオブジェクトなのは明らかです。しかし、こうなるとどうでしょう？

```
Rectangle RectA = new Rectangle();
Rectangle RectB = RectA;
```

この場合、RectA と RectB は異なるものになるでしょうか。答えは「ノー」です。この辺りは言語によって細かい仕様が異なる場合がありますが、この場合だと **RectB に RectA を代入したことで、RectB は RectA と同一のオブジェクトを持っている**ということになるのです。※6ということは「別のオブジェクト」であるか否かに変数名は関係ないのです。

オブジェクト（インスタンス）は new 演算子によって新しく作成されます。ここで作成されるオブジェクトは唯一無二のオブジェクトで、そのオブジェクトにはプログラム中唯一の値である**「オブジェクト ID」**が与えられます。最初の例でそれぞれが別のオブジェクトであったのは **RectA と RectB それぞれに new 演算子で作成された唯一無二オブジェクト（もちろん、それぞれのオブジェクト ID は異なる）が与えられたから**なのです。なお、言語によってはオブジェクト ID というのを見ることが可能な言語もありますし、オブジェクト ID 同士を比較してオブジェクトが同一かどうかをチェックするメソッドが標準で与えられている言語などもあります。

なお、new 演算子に関する詳しい解説は第七章にまとめて行う予定ですので、今は new 演算子を使えば新規のオブジェクトが生成されるのだ、という程度の認識で構いません。（実際、間違いではない。）

4. 終わりに

今回はこれで終了です。今回はメソッドの定義の仕方やメソッドの使い方、そしてオブジェクトの同一性についての話をしました。どちらも非常に大切な話であるとともに、慣れるまではイメージし辛い内容だったと思います。次回以降の内容は第三回までの内容を発展させていくものばかりなので、ここまでに出てきた重要なキーワード（オブジェクト、クラス、インスタンス、フィールド、メッセージ、メソッド、new 演算子など）は確実に理解し、説明できるようになっておいてください。

なお、次回はここまで勉強してきたオブジェクト指向の利点を活かして、簡単なゲームを作ってみようと思います。お楽しみに。

※1　：メッセージの正しい定義は「オブジェクト間の通信でやりとりされる情報」。ただし、現段階では上述の認識で構わない。
※2　：public キーワードの説明は次回の「カプセル化」の話で行います。
※3　：一応「値が正しくセットされたか否か」を返すとか、そういう考え方がないこともないです。
※4　：メソッド前につけるキーワードは実は「変数の型」と同じ。int ○○というメソッドなら int 型の値を返す。
※5　：printf 関数は C 言語のフォーマットを参考に。C 言語が分からない人でも何となく形は掴めると思います。
※6　：正確には RectB は RectA と同一のオブジェクトを「参照」している。参照については次回説明する。