オブジェクト指向とゲームプログラミング

○　Java 基礎編 － 第４回　オブジェクトの生成　○

オブジェクトの生成

クラスは、定義しただけでは使用できません。というのも、クラスがどのような属性や動作を持つのかを定義しただけであり、属性の内容といったことについては、まだなにも記述していないからです。そこで必要になるのがオブジェクトの生成という作業です。
たとえば、前回のCharacterクラスを利用して、実際にプログラム上で１つずつキャラクターを作成していきます。すると、そのあとで、その１つずつに座標やアニメーション番号を設定することができます。
このように、あるクラスからメモリに生成されるもののことをオブジェクト(object)またはインスタンス(instance)と呼びます。Characterクラスをもとに生成されるオブジェクトは、Characterクラスのオブジェクトと呼ぶことができます。

実際に、プログラムでオブジェクトを生成するには、次の２つ手順で行います。

１．オブジェクトを格納する変数を宣言する
２．オブジェクトが必要になったら、new演算子で生成する

始めに、通常のint型やdouble型の変数を宣言するのと同じように、型名の部分をクラス名にした変数を宣言します。

```
Character   hinomoto;
```

これがCharacterクラスのオブジェクトを扱うための変数hinomotoを用意する文です。変数hinomotoは、Character型の変数と呼ぶことができます。

次に、new演算子を使い、任意のタイミングでオブジェクトを生成します。

```
hinomoto = new Character();                 // デフォルトコンストラクタ
hinomoto = new Character(320.0, 240.0);     // 引数付きコンストラクタ
```

new演算子は、右辺で生成されたオブジェクトへの参照を返します。上の例では、=演算子によって、左辺で定義されているクラス型変数hinomotoに代入されています。オブジェクトが明示的に代入されていなければ、クラス型変数はnullで初期化されます。

２つの手順をまとめて記述することもできます。

```
Character   hinomoto = new Character();
```

Javaでは、参照されなくなったオブジェクトは、JavaVMによって自動的に破棄されます。そのため、C++のdelete演算子とデストラクタはありません(ただし、破棄するのは仕様上オプション扱いで、破棄するタイミングもJavaVMによって異なります。また、デストラクタの代わりにファイナライザという仕組みがあります)。

メンバへのアクセス

オブジェクトのメンバへのアクセスは、"."演算子で行います。

```
// hinomotoオブジェクトのmoveメソッドを呼び出す
hinomoto.move(3.0, 0.0);
```

参照型

参照型は、int型やdouble型の変数のように直接値を保持するのではなく、メモリのどこかにあるオブジェクトを参照するIDを格納するものです。参照型には、クラス型、配列型、インタフェイス型があります。

参照型変数に保持されるのは、生成されたインスタンス(実体)への参照であり、インスタンスはメモリ上の別の場所に存在しています。変数名で確保されたメモリ領域には、インスタンスの参照(インスタンスを識別する為のID)が保持されています。

参照型変数を別の参照型変数に代入すると、左辺の変数には、右辺の変数が参照しているインスタンスではなく、参照(インスタンスのID)がコピーされます。従って、インスタンスはメモリ上にひとつだけであり、そのインスタンスを参照する変数が２つできたことになります。

代入する参照型変数と、代入される参照型変数があるとき、後者の参照するオブジェクトを操作すると、前者の参照するオブジェクトも操作されたことになり、その逆も成り立ちます。参照型変数を代入すれば、同じインスタンスを参照するIDがコピーされて、同じものを参照することになるわけです。

たとえば、以下のようなクラスがあったとします。

```
class Reference {
    int   x = 0;

    Reference() {}                          // コンストラクタ
    public void setX(int val) { x = val; }  // xに値を設定するメソッド
}
```

このとき、プログラム中で、

```
Reference   ref = new Reference();
```

というようにオブジェクトを生成すると、メモリのどこかにReferenceクラスのインスタンスが生成され、そのIDが変数refに格納されます。このあと、

```
Reference   work = ref;
```

というような代入を行うと、変数refが保持しているIDが変数workに代入されます。この結果、変数refとworkは、同じIDを保持することになります。つまり、２つの変数は同じオブジェクトを指していることになります。この状態で、

```
work.setX(100);      // フィールドxの書き換え
```

のようにworkオブジェクトのフィールドxを書き換えると、refオブジェクトのフィールドxも同じ値に書き変わります。

これに対し、int型やdouble型の変数のように直接値を保持するものをプリミティブ型と呼びます。プリミティブ型では、直接値を保持しています。

```
int   a = 0;
int   b;

a = b;
b = 100;
```

このような場合、「a = b」という代入は、変数aの「値」を変数bにコピーするというものです。この動作により変数aも変数bも0になります。しかし、値は同じでもメモリ上の別々の場所に作られた独立した変数です。ですから、次の行で変数bに100を代入しても変数aは0のままです。

この代入動作の違いは、メソッドの引数として値やオブジェクトを渡したときにも現れます。プリミティブ型の変数の場合は、コピーが作成されます。呼び出し先のメソッドで変数の値を書き換えても、呼び出し元の変数は書き換わりません。オブジェクトや配列を渡した場合は、コピーではなくオブジェクトを参照するIDが渡されます。呼び出し先のメソッドでオブジェクトの内容を書き換える動作を行うと、呼び出し元のオブジェクトの内容も書き換わります。