オブジェクト指向と
ゲームプログラミング

○　Java 基礎編 － 第６回　継承　○

継承

これまでのCharacterクラスは、キャラクターの基本的な機能をまとめたものでした。さらに新しいクラスを作成する場合、たとえば自機を表すPlayerクラスや敵を表すEnemyクラスといった、独特の機能を持つクラスを作成するには、Characterクラスでは不十分なので、このクラスを拡張したり、書き直すといった作業が必要になります。しかし、一度完成したクラスに「付け足す」「書き直す」ことは、バグを減らすという観点からはなるべく避けるべきです。書き換えると新たなバグ生む危険性があるためです。一度完成したコードを書き換えるのは、細心の注意と多大な労力が必要になります。

Javaでは、継承(inheritance)という仕組みで、すでに作成したクラスをもとに新しいクラスを効率よく作成できるようになっています。既存のクラスを継承した新しいクラスは、既存のクラスのメンバを「受け継ぐ」ようなかたちになります。既存のクラスには直接さわらずに、新しく必要な属性や動作を付け足すように、新しいクラスのコードを記述することができます。
継承して新しいクラスを作成するには、キーワードextendsを使い、以下のような書式になります。

```
class 新しいクラス名 extends もとになるクラス名 {
    追加・変更のメンバを記述
};
```

「もとになるクラス」をスーパークラス、新しく拡張されたクラスをサブクラスと呼びます。サブクラスでは、スーパークラスの持つprivateとデフォルトアクセス以外のメソッドとフィールドを、あたかも自分のクラスで記述したかのように扱うことができます。

Characterクラスを継承して「残機」の加わったPlayerクラスを作成するとします。Characterクラスに残機を保持するメンバ変数とそれに付随するメンバ関数をつけ加えればよいでしょう。これは、以下のように定義できます。

```
// キャラクタークラス
class Character {
    public final static int   MAX_ANIME = 5;    // アニメーション最大数

    double   posX;                              // x座標
    double   posY;                              // y座標
    int      animeNo = 0;                       // アニメーション番号

    // コンストラクタ
    public Character() {
        this(0.0, 0.0);
    }

    // コンストラクタ
    public Character(double x, double y) {
        posX = x;
        posY = y;
    }

    // 移動
    public void move(double addX, double addY) {
        posX += addX;
        posY += addY;
    }

    // アニメーション
    public void animetion() {
        animeNo++;
        if(MAX_ANIME <= animeNo)
            animeNo = 0;
    }
```

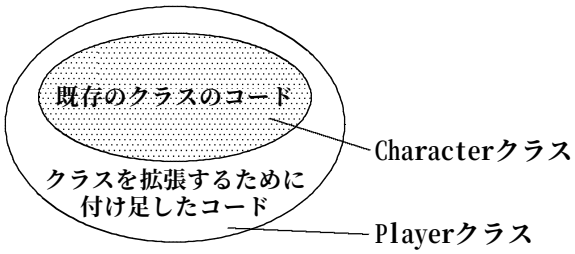
```
    // アクセスメソッド
    public double getX() { return posX; }   // x座標取得
    public double getY() { return posY; }   // y座標取得
}

// プレイヤークラス
class Player extends Character {
    int   rest;                         // 残機

    // コンストラクタ
    public Player(double x, double y, int rest) {
        this.rest = rest;
    }

    int oneUp()   { return ++rest; }      // 1機増やす
    int oneDown() { return --rest; }      // 1機減らす
}
```

こうすると、Characterクラスを内部に含んだPlayerクラスができあがります。この場合、Characterクラスがスーパークラス、Playerクラスがサブクラスとなります。



protected

スーパークラスCharacterは、すべてのフィールドがデフォルトアクセスとして記述されています。privateとデフォルトアクセスのメンバは、自身のクラス以外からアクセスすることはできません。たとえサブクラスであっても、直接アクセスすることができないのです。つまり、サブクラスPlayerが移動処理や座標取得を行うためにフィールドposXやposYに直接アクセスしようとしても、エラーになってしまいます。

通常、サブクラスとスーパークラスは密接な関係にあるため、このアクセス制限が不便な場合もあります。このような場合に用いるのがprotectedです。

protected指定すると、サブクラスでも直接アクセスできるメンバになります。ただし、これを多用するとカプセル化のメリットを損なってしまうので、注意しましょう。

```
// キャラクタークラス
class Character {
    public final static int   MAX_ANIME = 5;    // アニメーション最大数
    int         animeNo   = 0;                  // アニメーション番号

    protected double   posX;                    // x座標
    protected double   posY;                    // y座標

    public Character() { …省略… }
    public Character(double x, double y) { …省略… }

    public void move(double addX, double addY) { …省略… }
    public void animetion() { …省略… }

    public double getX() { return posX; }       // x座標取得
    public double getY() { return posY; }       // y座標取得
}
```

継承とコンストラクタ

コンストラクタは、インスタンスが生成されるときに呼び出される特別なメソッドであり、フィールドなどの初期化を行い、インスタンスを生成します。サブクラスのインスタンスを作成する場合には、何らかの方法でスーパークラスのコンストラクタを呼び出し、スーパークラスのインスタンスを生成しなければなりません。

スーパークラスのコンストラクタは、サブクラスから自動的に呼び出されるようになっています。サブクラスのコンストラクタが呼び出されると、何も記述されていない場合でも、最初にスーパークラスのコンストラクタが呼び出され、実行されます。スーパークラスのコンストラクタの処理がすべて終了したあと、サブクラスのコンストラクタが実行されます。

サブクラスのコンストラクタでスーパークラスのコンストラクタが明示されていない場合、スーパークラスのデフォルトコンストラクタが暗黙のうちに呼び出されます。スーパークラスのデフォルトでないコンストラクタを呼び出すには、キーワードsuperを用います。
たとえば、サブクラスPlayerのコンストラクタからスーパークラスCharacterの引数付きコンストラクタを呼び出すには、以下のように記述します。

```
Player(double x, double y, int rest) {
    super(x, y);
    this.rest = rest;
}
```

スーパークラスのコンストラクタ呼び出しは、サブクラスが生成される前に行わなければならないので、サブクラスのコンストラクタの最初に書かなければなりません。

is-a関係とhas-a関係

継承された２つのクラス(スーパークラスとサブクラス)の関係のことをis-a関係またはa-kind-of関係と呼びます。これは、「サブクラス is a(kind of) スーパークラス」すなわち「サブクラスはスーパークラス(の一種)である」という関係です。オブジェクト指向設計において、２つのクラスの関係が「○○は△△である」と考えられる場合には、継承を使ってクラスを関連付けます。

継承に対し、あるオブジェクトをいくつかのオブジェクトの組み合わせとして実現したり、あるオブジェクトの一部を別のオブジェクトで構成したりする方法を集約(aggregation)と呼び、この関係が特に強いものを合成(composition)と呼びます。
また、構成要素のオブジェクトとそれらを組み合わせてできたオブジェクトとの関係を、has-a関係またはpart-of関係と呼びます。これは、「クラスA has a(part of) クラスB」すなわち「クラスAはクラスBを持っている(またはその一部である)」という関係です。オブジェクト指向設計において、２つのクラスの関係が「他方がもう一方を持っている」と考えられる場合には、集約を使ってクラスを関連付けます。

プログラムを構成するすべてのクラスが、継承または集約のいずれかで必ず関連付けられるわけではありません。継承も集約も使わず、単独のクラスとなる場合もあります。継承や集約は、プログラムの構造を整理し、効率的に作成するための手段にすぎません。継承や集約を使うかどうかは、プログラム設計者の判断によります。

練習問題

1　以下の文章(１)～(６)に示したClassAクラスとClassBクラスの関係が「is-a関係」の場合はA，「has-a関係」の場合はB，どちらでもない場合はCを付けましょう。

(１)ClassAクラスは、ClassBクラスを継承している。
(２)ClassAクラスのメソッド内で、ClassBクラスのオブジェクトを作成した。
(３)ClassAクラスのフィールドに、ClassBクラスをデータ型とした変数がある。
(４)ClassAクラスとClassBクラスのフィールドに、別の同じクラスをデータ型とした変数がある。
(５)ClassAクラスとClassBクラスは、同じスーパークラスから継承されている。
(６)ClassAクラスは、ClassBクラスの一種だと考えられる。