- 演算子
- ラッパークラス
- クラスを作るってどういうこと？
- 既存のクラスを拡張したクラスを作ろう
- メソッド

★6　プリミティブ型と演算子（承前）

★6.4　演算子 (p.58)

演算子いろいろ

Java には，+，*，!，&&，=など様々な**演算子**が存在する．そのうち，引数が二つのものを**二項演算子**，一つのものを**単項演算子**という (☆ 1)．演算子 - は，単項／二項どちらの演算子としても使われる．-2-3 という式の場合，左の-は単項，右は二項の演算子である．

前節で述べたように演算子 = は変数の値を変化させるという副作用をもっているが，これ以外にも変数の値を変化させる演算子が存在する．x += 2（x = x + 2 の意味）のように使われる += や -=，*=，/= ，**インクリメント／デクリメント**（変数の値を 1 増やす／減らす）の単項演算子 ++, -- などがその例である．演算子 ++,-- は変数の前に置くか後ろに置くかで意味が違ってくるので注意が必要である（下記の例参照）．

☆ 1) C 言語同様に三項演算子も存在する．例えば
(x > 0) ? x : -x
という式の値は x の絶対値となる．

優先順位，結合の方向

y = 2 + 3 * x という式には，=，+，*と三つの演算子が登場するが，演算の順序を () で示すと y = (2 + (3 * x)) となる．このように演算子の並び順と式の評価順に違いが生じるのは，演算子に**優先順位**があるからである．p.59 の表を見るとわかるように，これら三つの演算子は * > + > = という順位をもつので，その順に式の評価が行われる．

同じ優先順位の演算子が二つ以上ならんでいる x + y - z や x = y += z のようなケースでは，演算子の**結合方向**にしたがって式の評価順が決められる．通常の演算子は左に結合するが，代入演算子は右に結合する．したがって，上の例は (x + y) - z , x = (y += z) という意味になる．

定められた優先順位や結合方向と異なる評価をさせたい場合には，y = (2 + 3) * x のように括弧を補えばよい．優先順位や結合方向がわからない場合も括弧を用いて評価順を明示すればよい．

例

```
int p = q = 37;
System.out.println(p++);   // 出力は 37 （インクリメント前の変数値が値となる）
System.out.println(++q);   // 出力は 38 （インクリメント後の変数値が値となる）

d = m.moveTo(100,0) + m.moveTo(100,100) + m.moveTo(0,100);
                      // かめは (100,0) → (100,100) → (0,100) と動く
e = m.moveTo(100,0) + m.moveTo(100,100) * m.moveTo(0,100);
                      // moveTo() の評価順は変わらないので動き方は上と同じ（d と e の値は異なる）

a = (int)Math.PI * 10;     // 優先順位は . > キャスト > * なので，a は 30 になる
b = (int)(Math.PI * 10)    //                                  b は 31 になる

int x=1, y=2, z=3;
x += y += z;         // (1) y が 3 増える，(2) その値ぶん x も増える → y は 5，x は 6
```

★6.5 ラッパークラス (p.59)

プリミティブ型の値は参照型のオブジェクトとは違うものであるが，プリミティブ値をオブジェクトのように扱いたいことがある．そのために，**ラッパークラス**という，プリミティブ型に対応したクラスが用意されている．前回の Args02 で用いた Integer.parseInt() というメソッドは，int に対応したラッパークラスである Integer クラスのクラスメソッドである（p.59 参照）．

★7 クラスの作成 (p.60)

★7.0 クラスを作るってどういうこと？

★7.0.1 かめさんプログラム再考

これまで作ってきたプログラムのうち，かめさんを動かすものは，図 1 のような構成をしていた．われわれが自分で作るのは Hoge.java とそれをコンパイルした Hoge.class のみで，タートルグラフィックスのための TurtleFrame クラスと Turtle クラスについては，予め用意されたクラスファイル (TurtleFrame.class と Turtle.class(☆2)) を利用していた．今回は，他のプログラムから利用できるような新しいクラスを自分で定義する方法を学ぶ．

☆ 2) これらのクラスファイルは，TurtleFrame.java と Turtle.java というソースをそれぞれコンパイルして作られたものである．

Hoge.java

```
   :
TurtleFrame f = new TurtleFrame();
Turtle m = new Turtle();

f.add(m);
m.fd(100);
   :
```

TurtleFrame.class

TurtleFrame クラスを定義したクラスファイル

Hoge.class

Hoge クラスを定義したクラスファイル

Turtle.class

Turtle クラスを定義したクラスファイル

図 1: これまでのかめさんプログラムの構成例

★ 7.0.2 作る話の前に使う話の復習をしよう

自分でクラスを作る方法を学ぶ準備として，他人が作ったクラスを利用する例で復習しよう．

HW クラスの API 仕様 個人の身長体重のデータを扱うためのクラス

● コンストラクタ

HW() 名前 ”ほげお”，身長 170.0[cm]，体重 62.0[kg] のインスタンスを生成．
HW(String n, double h, double w) 名前 n，身長 h[cm]，体重 w[kg] のインスタンスを生成．

● メソッド

void print() このインスタンスの名前，身長，体重を表示する．
double calcBMI() このインスタンスの BMI を計算して返す． BMI = (体重 [kg])/(身長 [cm]/100)^2
static void printHimando(double bmi) bmi が Border1 未満，Border1 以上 Border2 未満，Border2 以上の三つの場合に分けてメッセージを出力．順に，「やせてんなぁ」，「ふつー」，「太ってる…かな？」．

● フィールド

String name 名前． **double height** 身長．**double weight** 体重．
static double Border1 BMI の境界値のうち小さい方．初期値は 20.
static double Border2 BMI の境界値のうち大きい方．初期値は 25.

Fugayo.java

```
 1  /** HW クラスを使うプログラム */
 2  public class Fugayo{
 3      public static void main(String[] args){
 4          HW p1 = new HW();
 5          p1.print();
 6          double bmi = p1.calcBMI();
 7          HW.printHimando(bmi);

 8          HW p2 = new HW("ふがよ", 234.5, 50.0);
 9          p2.print();
10          HW.printHimando(p2.calcBMI());

11          p2.weight = 120.0;
12          p2.print();
13          HW.printHimando(p2.calcBMI());

14          HW.Border1 = 24.0;
15          HW.printHimando(p1.calcBMI());
16          HW.printHimando(p2.calcBMI());
17      }
18  }
```

Fugayo の実行結果 (注)

```
ほげおさんの身長は 170.0[cm], 体重は 62.0[kg] です
BMI は 21.453287197231838 ふつー

ふがよさんの身長は 234.5[cm], 体重は 50.0[kg] です
BMI は 9.09252094689513 やせてんなぁ

ふがよさんの身長は 234.5[cm], 体重は 120.0[kg] です
BMI は 21.822050272548314 ふつー

BMI は 21.453287197231838 やせてんなぁ
BMI は 21.822050272548314 やせてんなぁ
```

注) Fugayo.class と同じディレクトリ内に HW.class があると想定している．また，この実行結果には，見やすくするために改行が挿入されている

★7.1 既存のクラスを拡張したクラスを作ろう

次節以降ではじめに考える例は，図 2 に示すような構成のものである (☆ 3)．T71.java と HTurtle.java の二つのソースを作成する．HTurtle.java は新しく作成する HTurtle というクラスの定義を書いたものであり，T71.java はそれを利用するものである．

☆ 3) これらの例では，全てのソースファイルは一つだけクラスを定義しており，ソースファイル名とクラス名が一致している．実際には，一つのソースファイル中に複数のクラスを定義したり，ソースファイル名と異なるクラス名をつけることも可能であるが，当面は説明しない．

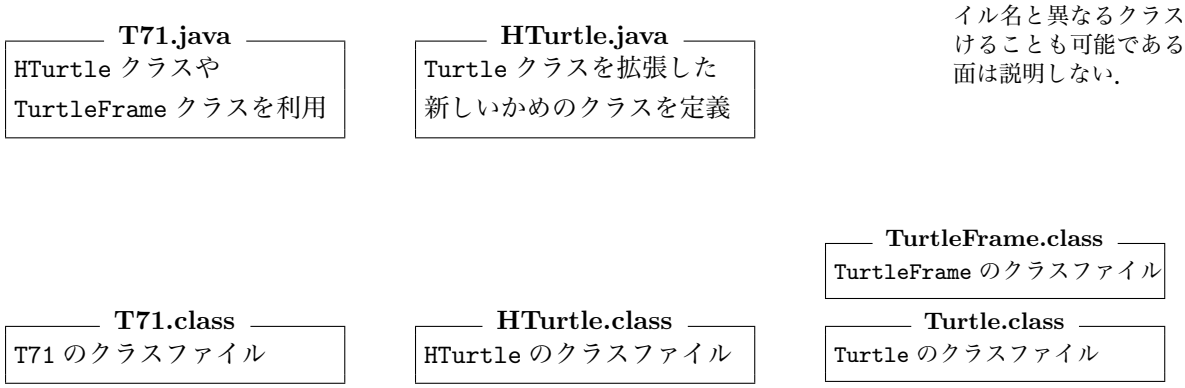

図 2: HTurtle クラスを用いるプログラムの構成例

HTurtle クラスは，今まで用いてきた Turtle クラスに新たな機能を追加して作成する．このように既存のクラスに機能を追加して新しいクラスを定義することを，既存のクラスを**拡張する**という．また，クラス A を拡張してクラス B が定義されているとき，B は A の**サブクラス（子クラス）**であるといい，A は B の**スーパークラス（親クラス）**であるという．HTurtle クラスは Turtle クラスを拡張しており，HTurtle クラスは Turtle クラスのサブクラスである．

クラス A $\xrightarrow{\text{拡張}}$ クラス B $\xrightarrow{\text{拡張}}$ クラス C という関係の場合，A も B も C のスーパークラスであり，B も C も A のサブクラスである．Java には，あらゆるクラスのスーパークラスとして **Object クラス**が存在しており，全てのクラスは Object クラスのサブクラスである (☆ 4)．

あるクラスのオブジェクトは，自分のインスタンスメソッドやインスタンス変数の他に，自身のスーパークラスで定義されたインスタンスメソッドやインスタンス変数も引き継いでもっている．このことを，メソッドや変数をスーパークラスから**継承する**という．したがって，HTurtle クラスは，Turtle クラスとそのスーパークラスの全てのインスタンスメソッド／変数を継承している (☆ 5)．クラスメソッド／クラス変数についても同じ様なことがいえる (☆ 6)．

☆ 4)「extends クラス名」を省略すると，「extends Object」とみなされる．

☆ 5) Java API 仕様のページから java.lang パッケージの Object クラスを探せば，Object クラスのメソッドなど (あらゆるクラスに継承されている) を一覧することができる．

☆ 6) 詳しいことは省略するが，例えば HTurtle は Turtle クラスのサブクラスなので，HTurtle.speedAll というクラスメソッドや HTurtle.withTurtleAll というクラス変数を使える．

★7.2 メソッド

HTurtle クラスの中身を考えていこう．まずは新しいメソッドの追加から．

★ 7.2.1 メソッドの追加

Turtle クラスを拡張した HTurtle クラスを定義し，p.60 の API 仕様が示すような機能をもつ二つのメソッド polygon と house を作ることを考える．以下の HTurtle.java が HTurtle クラスを定義するプログラムであり，T71.java がそれを用いるプログラムである．

HTurtle クラスは Turtle クラスのインスタンスメソッド／変数を継承しているので，HTurtle クラスを利用する T71.java では，HTurtle の二つのインスタンスメソッドとともに fd などの Turtle のインスタンスメソッドも呼び出すことができる．

T71.java

```
 1  public class T71 {
 2     public static void main(String[] args){
 3        TurtleFrame f = new TurtleFrame();
 4        HTurtle m = new HTurtle();
 5        int s = 50;
 6        f.add(m);
 7        m.house(s);
 8        m.up(); m.lt(90); m.fd(50); m.rt(72); m.down();
 9        m.polygon(5, s / 2);
10        m.up(); m.moveTo(100,100,0); m.down();
11        m.polygon(10, s / 5);
12     }
13  }
```

HTurtle.java

```
 1  public class HTurtle extends Turtle {    //Turtle を拡張する



 2     public void polygon(int n, int s){   //polygon メソッドの定義

 3        int a = 360/n;                     //曲がる角度を求めておく
 4        for(int j = 0; j < n; j++){        //n 回繰り返す
 5           fd(s);                           //s 前に進んで
 6           rt(a);                           //a 曲がるのを


 7        }
 8     }

 9     public void house(int s){   //house メソッドの定義

10        polygon(4,s);                      //polygon を利用
11        fd(s); rt(30);
12        polygon(3,s);
13        lt(30); bk(s);                     //元の場所に戻しておく
14     }
15  }
```