- メソッド（承前）
- インスタンス変数
- 変数やメソッドへのアクセスを制限する

★7　クラスの作成（承前）

★7.2　メソッド（承前）

★7.2.2　メソッドの多重定義 (p.65)

Turtle クラスの moveTo メソッド（p.14 参照）のように，Java では同じ名前で引数の数や型が異なるメソッドを**多重定義**することができる．HTurtle.java に以下を追加すると，polygon メソッドを多重定義することになる (☆ 1)．どちらの polygon メソッドが使われるかは，このメソッドを呼ぶ側の引数の指定の仕方（引数の数，型，それらの順序）で決まる．

☆ 1) 追加する場所は，1,2 行目の間，8,9 行目の間，14,15 目の間のいずれか

辺の長さ s の n 角形をかめの色 c で描く polygon メソッドの定義

```
public void polygon(int n, int s, java.awt.Color c){
  java.awt.Color prev = tcolor;  // 前のかめの色を記憶
  tcolor = c;       // かめの色を c に
  polygon(n, s);    // 引数 2 つの polygon を呼び出す
  tcolor = prev;    // かめの色を戻す
}
```

Q1. T71.java の 9,11 行目を次のようにすると，実行結果はどうなりますか．

```
 9   m.polygon(5, s / 2, java.awt.Color.red);
11   m.polygon(10, s / 5);
```

★7.2.3　値を返すメソッド (p.65)

メソッドの定義は，一般に次のような形式である．

アクセス修飾子 | 戻り値の型 | メソッド名 (| 引数 1 の型 | 仮引数 1 | , ..., | 引数 n の型 | 仮引数 n |)
メソッド本体のブロック

アクセス修飾子（上記の例では public）については次節で説明する．上記の HTurtle の例では polygon メソッドも house メソッドも返値の型は void であり値を返さないが，ここに void 以外の型を指定すればその型の値を返すメソッドを定義することができる．以下は，かめさんが引いた線の長さを返すように polygon メソッドを修正した例である (☆ 2)．

☆ 2) ここでは polygon メソッドをさらに多重定義するのではなく，前に作った polygon メソッドを書きかえている．

int を返す polygon メソッドの定義

```
public int polygon(int n, int s){
   int a = 360/n;
   for(int j = 0; j < n; j++){
      fd(s); rt(a);
   }
   return n * s;
}
```

polygon を呼ぶ側の例

(T71.java の 9 から 11 行目を修正したもの)

```
int d = 0;
d += m.polygon(5, s / 2);
m.up(); m.moveTo(100,100,0); m.down();
d += m.polygon(10, s / 5);
System.out.println("d = " + d);
```

return 文は，次の例のようにメソッド本体の途中にあってもよいし，条件分岐などのために複数あってもよい．値を返さないメソッドの処理を途中で終わらせるのにも使える．

return の使い方の例 (1)

```
public int hoge(int n){
   if(n % 3 == 0) return 999;
   int x = n * 2;
   return x;
}
```

return の使い方の例 (2)

```
// s < 100 なら何もしない
public void fuga(int s){
   if(s < 100) return;
   house(s);
}
```

Q2. 前頁で定義した 3 引数の polygon メソッドも 2 引数のものと同様に変更しなさい．

★ 7.3 インスタンス変数

★ 7.3.1 インスタンス変数の宣言と利用

次は，インスタンス変数を定義して用いることを考える．再び Turtle クラスを拡張して今度は Stepper というクラスを作成する．Stepper クラスの仕様は p.66 の API 仕様の通りである．以下の Stepper.java が Stepper クラスを定義するプログラムであり，T72.java がそれを用いるプログラムである．

注：古い刷の教科書では，p.66 の本文下の方と Stepper クラスの API 仕様中に「HTurtle クラスを拡張」という記述がありますが，これらは誤りです．p.67 の Stepper.java を見るとわかるように，正しくは「Turtle クラスを拡張」です．

Stepper.java

```
 1  public class Stepper extends Turtle{

 2     public int n;                    //公開されたインスタンス変数 n の宣言
 3     public int size;                 //公開されたインスタンス変数 size の宣言
 4     private int j = 0;               //（非公開）インスタンス変数 j の宣言

 5     public void step() {
 6        if(j >= n)
 7           return;       //描き終えていたならすぐ終了
 8        fd(size);
 9        rt(360/n);
10        j++;              //j の値を 1 増やす
11     }
12  }
```

インスタンス変数は名前の通りインスタンス毎に用意される変数であるから，個々の Stepper インスタンス (T72.java の例では m1 と m2) が保持する n，size，j の値はみな別物である．

T72.java

```
 1  public class T72{
 2     public static void main(String[] args){
 3        TurtleFrame f = new TurtleFrame();

 4        Stepper m1 = new Stepper(); f.add(m1);
 5        Stepper m2 = new Stepper(); f.add(m2);

 6        m1.n = 4;  m1.size = 100;
 7        m2.n = 3;  m2.size = 100;  m2.up(); m2.moveTo(100,200,0); m2.down();

 8        for(int i = 0; i < 4; i++){
 9           m1.step();
10           m2.step();
11        }
12     }
13  }
```

次に，ローカル変数とインスタンス変数の違いを考えてみよう．右のような例を考えてみればわかるように，ローカル変数は宣言されたブロックの中でしか有効でないため，メソッド内で宣言したローカル変数はメソッド呼び出しのたびに初期化される．これに対して，インスタンス変数はインスタンス生成時に初期化された後ずっと値を保持することになる．この例では，step メソッドが呼ばれるたびに i の値は 0 に初期化されるけれど，j の値は step が呼ばれるたびに増えていく)．

```
public void step() {
   int i = 0;  // ローカル変数
   if(j >= n) return;
   fd(size);  rt(360/n);
   System.out.println(i+" "+j);
   i++; j++;
}
```

★ 7.3.2 this

メソッドを実行しているインスタンス自身を指す this というキーワードがある．Stepper.java のインスタンス変数 n やインスタンスメソッド fd は，インスタンスメソッドの定義の中では本来はそれぞれ this.n，this.fd と書くべきところであるが，省略しても差し支えないので略記している (☆ 3)．省略しないで書くと，次のようになる．

☆3) インスタンス変数と仮引数やローカル変数の名前を同じにした場合は，互いに区別できるようにインスタンス変数には必ず this を付けないといけない．

this をつけるべき所すべてに this をつけた Stepper.java

```
 1  public class Stepper extends Turtle{
 2     public int n;
 3     public int size;
 4     private int j = 0;
 5     public void step() {
 6        if(this.j >= this.n)
 7           return;
 8        this.fd(this.size);
 9        this.rt(360/this.n);
10        this.j++;
11     }
12  }
```

Q3. HTurtle.java 中で this を付けられる所を指摘しなさい．

★ 7.4 変数やメソッドへのアクセスを制限する (p.68)

★ 7.4.1 何でも公開するのはよくないかも

前節で登場した Stepper.java では，3 つのインスタンス変数のうち j のみを非公開（private をつける）にし，n,size は公開（public をつける）していた．変数を公開することにどんなデメリットがあるのか，次の例で考えよう．

KoukaiStepper.java

```
1  public class KoukaiStepper extends Turtle{

2      public int n;      // n も
3      public int size;   // size も
4      public int j = 0;  // j も public

5      public void step() {
6          if(j >= n) return;
7          fd(size); rt(360/n); j++;
8      }
9  }
```

Iyan.java

```
1  public class Iyan {
2      public static void main(String[] args){
3          TurtleFrame f = new TurtleFrame();
4          KoukaiStepper m = new KoukaiStepper();
5          f.add(m); m.n = 6; m.size = 100;
6          m.step(); m.step(); m.size = 50;  // size の値いじっちゃった
7          m.step(); m.step(); m.j = 10;     // j の値いじっちゃった
8          m.step(); m.step();
9      }
10 }
```

★ 7.4.2 アクセス修飾子

上記の public や private は，クラスやメソッドの定義あるいはフィールド（インスタンス変数とクラス変数）の宣言の先頭に書いて，メソッドや変数の公開度合いを決める役割をもっており，**アクセス修飾子**と呼ばれる．アクセス修飾子は次の 4 つに分けられる (☆ 4)（上のものほど公開してる）．

public	パッケージ外にも公開
protected	同じパッケージに属するクラスとそれらのサブクラスからアクセス可能
指定なし	同じパッケージに属するクラスからアクセス可能
private	そのクラス内からのみ（サブクラスからもアクセスできない）

ここで言う「アクセス」とは，メソッドなら呼び出すこと，変数なら値を参照したり代入したりすることを指している．「パッケージ」とは，関連したクラスをひとまとめにしたもののことである (☆ 5)．

当面は，public（あるいは指定なし (☆ 6)）と private の使い分けを理解していれば十分である．

☆ 4) クラスを修飾できるのは，public と「指定なし」のみ．

☆ 5) 以前登場した，java.awt.Color クラスを含む java.awt パッケージや，Math クラスを含む java.lang パッケージなどがその例．パッケージに関する詳細は教科書第 9 章参照．

☆ 6) この授業では，同じディレクトリ内にあるクラスファイルを利用するようなプログラム例しか考えない．一般にディレクトリとパッケージは対応しているので，その場合には public も指定なしも変わりないことになる．ただし，ソースファイル名の付け方には影響する（詳細は教科書参照）．

★ 7.4.3 隠蔽とカプセル化

上記の問題を解決するには, 変数 n，size，j の宣言にアクセス修飾子 private をつけて非公開にしてやればよい．ただし，そうすると変数 n，size に直接値を代入できなくなるので，次のように値をセットするメソッドも用意する．

ImpeiStepper.java

```
 1  public class ImpeiStepper extends Turtle{

 2      private int n;      // n も
 3      private int size;   // size も
 4      private int j = 0;  // j も private

 5      public void setN(int n) {
 6          if(j == 0) this.n = n;        // 描き始める前以外は何もしない
 7      }

 8      public void setSize(int size) {
 9          if(j == 0) this.size = size;  // 同じく
10      }

11      public void step() {
12          if(j >= n) return;
13          fd(size); rt(360/n); j++;
14      }
15  }
```

Iyan2.java

```
 1  public class Iyan2 {
 2      public static void main(String[] args){
 3          TurtleFrame f = new TurtleFrame();
 4          ImpeiStepper m = new ImpeiStepper();

 5          f.add(m); m.setN(6); m.setSize(100);  // n や size はメソッド経由で

 6          m.step(); m.step(); m.setSize(50);  // size いじれる？

 7          m.step(); m.step();    // m.j = 10; はコンパイルエラー
 8          m.step(); m.step();
 9      }
10  }
```

上記のように変数を非公開にして外部からは直接アクセスできなくすることを，データを**隠蔽する**という．この例では，変数 n,size を非公開にした上で SetN,SetSize メソッドを工夫して，多角形を描いている途中にはこれらの値を変更できなくしている．このように，データを隠蔽しておくと，プログラマが予期せぬデータの変化を防いだりチェックすることが容易になる (☆ 7)．また，非公開の変数は API 仕様に記さないことにしておけば，クラスを利用する側は，ソースを見ない限りは，このような変数が存在することを知らずに利用することができる (☆ 8)．

☆ 7) 他にも，データを抽象化できる (内部でどんなデータ構造を採用しているか外部からは気にしないで済むようにできる）というメリットもある．

☆ 8) Turtle クラスの場合，かめの位置や向きを表す変数は隠蔽されており，API 仕様にも記されていない．

また，このような設計とすることで，ImpeiStepper クラスの作成者は，他への影響を気にしないでプログラムの内部を容易にいじれるようにもなっている（例えば KoukaiStepper の場合，別のプログラムで変数 n,size が利用されている可能性を考えるとこれらの変数名等をうかつに変更できないが，ImpeiStepper ならいつでも変更できる）．同様に，内部でのみ利用するメソッドを非公開としておけば，外部からそのメソッドを使われる心配がないので，安心して仕様（例えば引数の数）を変更することができる．このように，プログラムの外部からは内部の様子が分からないように（分からなくてすむように）することを，**カプセル化**するという．複数人で大きなプログラムを作成したり，過去に作成したプログラムを再利用することを考えると，カプセル化することには非常に大きなメリットがあることがわかる．

カプセル化できるような仕組みが提供されていることは，Java を含めたオブジェクト指向言語の大きな特徴の一つである．

宿題？

次回は教科書第 7 章の残り（ただし「7.7 内部クラス」と「7.8 まとめの例題」はとばします）の内容を説明します．あらかじめ読んでおくこと．例題のプログラムを作成し実行してみておくこと．