- 変数やメソッドへのアクセスを制限する
- コンストラクタ

★5　クラスの作成（承前）

★5.5　変数やメソッドへのアクセスを制限する (p.54)

★5.5.1　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（import 文は省略）

```
 3  public class KoukaiStepper extends Turtle{

 4      public int    n;       // n も
 5      public double size;    // size も
 6      public int    j = 0;   // j も public

 7      public void step() {
 8          if(j >= n)  return;
 9          fd(size); rt(360.0/n); j++;
10      }
11  }
```

Iyan.java（import 文は省略）

```
 3  public class Iyan {
 4      public static void main(String[] args) {
 5          TurtleFrame f = new TurtleFrame();
 6          KoukaiStepper m = new KoukaiStepper();
 7          f.add(m); m.n = 6; m.size = 100;
 8          m.step(); m.step(); m.size = 50;  // size の値いじっちゃった
 9          m.step(); m.step(); m.j = 10;     // j の値いじっちゃった
10          m.step(); m.step();
11      }
12  }
```

★5.5.2　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 (☆ 1)（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 (☆ 2)．

当面は，public（あるいは指定なし (☆ 3)）と private の使い分けを理解していれば十分である．

☆ 1) クラスを修飾できるのは，public と「指定なし」のみ．

☆ 2) Turtle や TurtleFrame は tg パッケージに含まれている．この授業ではパッケージについてはスキップした．教科書 5.4.1 節 (p.54) 参照．

☆ 3) 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）．

★ 5.5.3 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（import 文は省略）

```
 3  public class ImpeiStepper extends Turtle {

 4      private int    n;      // n も
 5      private double size;   // size も
 6      private int    j = 0;  // j も private

 7      public void setN(int n) {
 8              if(j == 0) this.n = n;        // 描き始める前以外は何もしない
 9      }

10      public void setSize(double size) {
11              if(j == 0) this.size = size;  // 同じく
12      }

13      public void step() {
14              if(j >= n) return;
15              fd(size); rt(360/n); j++;
16      }
17  }
```

Iyan2.java（import 文は省略）

```
 3  public class Iyan2 {
 4      public static void main(String[] args){
 5                  TurtleFrame f = new TurtleFrame();
 6                  ImpeiStepper m = new ImpeiStepper();
 7                  f.add(m);
 8                  // m.n = 6;  m.size = 100.0;  これらはコンパイルエラー

 9                  m.setN(6); m.setSize(100);  // 値のセットはメソッド経由で

10                  m.step(); m.step(); m.setSize(50);  // ここで size いじれる？

11                  m.step(); m.step();  // m.j = 10; はコンパイルエラー
12                  m.step(); m.step();
13      }
14  }
```

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 (☆ 4)．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 (☆ 5)．

☆ 4) 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．

☆ 5) 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名前など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．

変数を非公開にした場合，他のクラスからその値を知りたい場合も変更したい場合も直接アクセスできないため，メソッドを経由することになる．ImpeiStepper の setN，setSize メソッドは，このクラスのインスタンス変数 n, size の値を外部から変更できるように（しかもおかしな変更は受け付けず安全に変更できるように）したメソッドである．これらの値を外部から参照することも必要なら，

```
public int getN() {
    return n;
}
```

のようなメソッドを定義しておけばよい（size についても同様）．このように隠蔽したフィールドへアクセスするためのメソッドには「set なんたら」,「get なんたら」という名前を付けることが多いので，setter/getter と呼ぶことがある．

★5.6 コンストラクタ (p.57)

先の ImpeiStepper クラスで隠蔽したインスタンス変数 n,size は，そのオブジェクトが描こうとしている多角形の辺の数と長さを表すものである．これらの値は，上記のようにメソッド経由で設定するのではなく，インスタンス生成時に設定することにしてもよい．その場合，次のように n,size の値を受け取ってインスタンスを生成する**コンストラクタ**を定義してやればよい．

復習: コンストラクタって何？

```
Turtle m = new Turtle();

Turtle m1 = new Turtle(100, 100, 45);
```

CStepper.java の一部

```
 3  public class CStepper extends Turtle{

 4      private int n;
 5      private double size;
 6      private int j = 0;

 7      public CStepper(int n, double size){
 8          this.n = n;                  //インスタンス変数 n に 引数 n を設定
 9          this.size = size;            //インスタンス変数 size の設定
10      }

11      public CStepper(int n, double size, int x, int y, int angle){
12          super(x, y, angle);     //Turtle のコンストラクタ呼出し
13          this.n = n;
14          this.size = size;
15      }

16      public void step() {
 :        // Stepper.java と同じ
        }
    }
```

Q1. 次のプログラムを CStepper クラスを用いるものに書きかえなさい．

```
Stepper m = new Stepper();
m.n = 6;  m.size = 200;
```

Q2. T52 の m1 は，

```
Stepper m1 = new Stepper(); f.add(m1);
m1.n = 4;  m1.size = 100;
m1.up(); m1.moveTo(100, 200, 0); m1.down();
```

ということになっている．CStepper を用いると，これを以下のように書き換えられる．「???」に入るものを答えなさい．

```
CStepper m1 = ???;
f.add(m1);
```

コンストラクタはオブジェクト生成時に実行するものなので，スーパークラスのコンストラクタの内容を含む必要がある．例えば，CStepper クラスは Turtle クラスのサブクラスである．CStepper のコンストラクタを実行してオブジェクトを生成する際には，Turtle のオブジェクト生成時と同様の初期設定を行った上で，CStepper としての初期設定を行う必要がある．

CStepper.java の 7 行目から 10 行目で定義されている 2 引数のコンストラクタの場合，CStepper の 2 つのインスタンス変数への値の代入のみが書かれているが，実際には，このコンストラクタを呼び出した場合，まず Turtle の引数なしコンストラクタを呼び出した時（`new Turtle()` した時）と同じ処理を行ってからこれらの代入の処理が行われることになる (☆ 6)．この例のように，コンストラクタ内で特に指定しなければ，スーパークラスの引数なしコンストラクタが最初に自動的に呼ばれることになる．

☆ 6) 7 行目と 8 行目の間に `super();` を挿入するのと同じ．

一方，5 引数のコンストラクタの定義（11 行目から 15 行目）では，明示的にスーパークラスのコンストラクタを呼び出している．12 行目がそれである．ここでは，Turtle クラスの 3 引数コンストラクタを呼び出している．

また，コンストラクタの定義では，`this` と書くことで，同じクラスで定義された別のコンストラクタを呼び出すこともできる．例えば，CStepper の 2 引数コンストラクタは，5 引数の方を呼び出す形で

```
public CStepper(int n, double size){
    this(n, size, 200, 200, 0);
}
```

と定義することもできる．このような `super` や `this` の呼び出しは，コンストラクタの定義の先頭に書かなければならない．

ところで，これまでの HTurtle や Stepper などのクラスでは，コンストラクタを定義してなかったのに，引数なしのコンストラクタを用いてインスタンスを生成することができていた．これは,「コンストラクタの定義がなければ，引数なしで中身が `super();` のみのコンストラクタが自動的に定義される」からである．逆に，コンストラクタの定義があればこの手続は行われない．したがって，CStepper の例では引数なしコンストラクタは使えない．CStepper で引数なしコンストラクタも使いたければ，自分で定義する必要がある．

Q3. CStepper に引数なしコンストラクタを定義しよう．仕様は「四角形を一辺の長さ 100 で描く CStepper を (200, 200) の座標に 0 度の角度で作成する」とする．`super` や `this` の使い方で何通りも書けるが，次の二通りで考えてみよう：

(1) 2 引数の `this` を呼ぶ

(2) `super` も `this` も使わない

宿題？

- 授業中に解説されなかった Q を解いてみよう．
- 5 章の残り（内部クラス，まとめの問題）はこの授業では説明しません．次回は 6 章の予定．