（2008-05-20 版）

Java によるプログラミング入門 (5)

1　Lesson 3: メソッド，クラス，オブジェクト

問題意識 数学の「関数」では，例えば 2 次関数 $x^2 + 4x + 5$ を記号的に $f(x) = x^2 + 4x + 5$ と定義します．
また，定義では変数 x の式で関数の値を定めますが，実際に使うときには $f(10)$ などと x の代わりに具体的な値を与えて関数値を表します．
このような表現により，より表現が簡潔になり，何度も同じ関数を使うことも容易になります．本章では，Java のプログラムでこれに相当する**メソッド**と呼ばれる機構について学びます．

1.1　流入量の変化が何度も繰り返す場合の水位を計算する

前節ではタンクへの流入量が 10 ステップにわたって増加する例を用いました．この節では 10 ステップにわたる流入量の変化が何度も繰り返す，ということを想定して，より長時間のシミュレーションを行うタスクを考えます．

流入量が，時刻 0 では 1.0，以降 2.0, 3.0, ..., 10.0, 1.0, 2.0, ... と変化していくとしましょう．

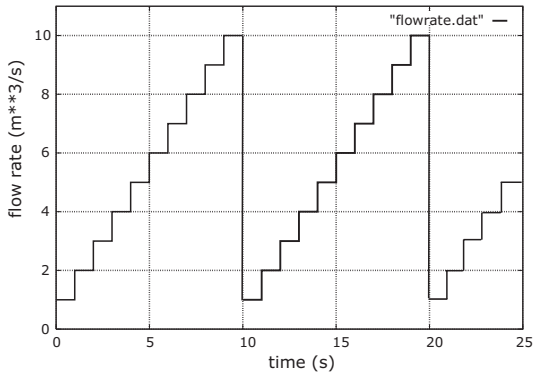

図 1　時間変化する流入量

flow[0] = 1.0, ..., flow[9] = 10.0 として，10 以上の time に対しては流入量はどうやって求めればいいのでしょうか？

時刻 time を整数型（切り捨て）に変換した後，これを 10 で割ってやると，余りは 0, 1, 2, ..., 9, 0, 1, となります．flow[(int)time を 10 で割った余り] が，求めたい流入量です．たとえば，flow[20] = flow[0] = 1.0 です．

```
int index = (int) time % flow.length;
// flow[index] が必要な流入量
```

ここで“ % ”は余り（剰余）を求める演算子です．

この計算をメソッドを使って実装してみましょう．

そのために学ばなければならないことは以下のようなものです：

- メソッドはどう記述するのか？
- メソッドはどう呼び出すのか？
- 関連して，オブジェクトとは何か？
- 引数（ $f(x)$ の x のこと ）はどう扱われるのか

まずは，TankCalculator3.java の 7-14 行目，24-25 行目をエディタで入力して完成させ，コンパイルし，実行してください．ここでは，getFlow というメソッドと report というメソッドが登場します．どのように記述され，どのように使われているか意識しながら打ち込んでみてください．

1.2 TankCalculator3.java

```
public class TankCalculator3 {
  final double TANK_AREA = 20.0;              //20.0 m**2
  final double INITIAL_TANK_LEVEL = 10.0;     //10.0 m
  double flow[]
    = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0}; //m**3/s

  public double getFlow(double t) {
    int index = (int)t % flow.length;
    return  flow[index];
  }

  public void report(double time, double flow, double tankLevel) {
    System.out.println(time + ", " + flow + ", " + tankLevel);
  }

  public static void main(String args[]) {
    TankCalculator3 calculator = new TankCalculator3();
    final int MAX_TIME = 50;
    double tankLevel[] = new double[MAX_TIME+1]; //m
    tankLevel[0] = calculator.INITIAL_TANK_LEVEL;
```

```
21:     System.out.println("Time(s), Flow(m**3/s), Tank Level(m)");
22:     for (int i = 0; i< MAX_TIME; i++) {
23:       double time = i;
24:       tankLevel[i+1] = tankLevel[i] + calculator.getFlow(time)/calculator.TANK_AREA;
25:       calculator.report(time, calculator.getFlow(time), tankLevel[i]);
26:     }
27:     double finalTime = MAX_TIME;
28:     calculator.report(finalTime, 0.0, tankLevel[MAX_TIME]);
29:   }
30: }
```

このプログラムを簡単に解説しておきます.

- 2 〜 5 行：クラスの直下で変数を定義しています．これらの変数は**メンバ変数**とか**フィールド**と呼ばれ，クラスから生成されるオブジェクト（インスタンスとも言います）が継続的に保持する変数です.
- 7 〜 10 行：時刻を与えて流入量を得る getFlow() というメソッド（関数）を定義しています.
- 12 〜 14 行：時刻 (time)，流入量 (flow)，水位 (tankLevel) を与えてこれらを端末に出力する report() というメソッドを定義しています.
- 16 〜 29 行：おなじみの main() メソッドです.

- 17 行：TankCalculator3 というクラス（このプログラムそのもの）のオブジェクトが new により作成され，変数 calculator に参照が代入されます．これにより，変数 calculator を対象に，クラス TankCalculator3 で定義されたフィールドやメソッドが使えるようになります．
- 20,24,25,28 行：calculator. フィールド名，calculator. メソッド名という形式でフィールド変数やメソッドにアクセスしています．

1.3　オブジェクトという考え方

Java や C++ は「オブジェクト指向」のプログラミング言語と呼ばれます．**「オブジェクト (object)」**とは「もの」という意味です．プログラムの中で関連した**変数やメソッド（関数）を全部ひとまとめ**にして「もの＝オブジェクト」として扱っていこうという考え方なのです．[1]

- いままで特に説明もせずに書いてきた**「クラス (class)」**は**オブジェクトを生成する型（ハンコのようなもの）**です．
- クラスからオブジェクトを生成する（ハンコをつく）操作は **new** という命令を使います．このように生成されたオブジェクトは**インスタンス**とも言います．

ややこしい横文字が並びますが，ごく簡単にいうと，

- メソッド ： 関数，機能
- フィールド＝メンバ変数 ： 持ち物，継続的に使いたい変数
- オブジェクト＝インスタンス ： メソッドとフィールドをひとまとめにしたもの
- クラス ： オブジェクト（インスタンス）を作るハンコ，型

です．

[1]C 言語の経験がある人は C 言語でデータをひとまとめにして扱う「構造体」という考え方を拡張してメソッド（関数、データに対する操作）も一括して扱えるようにしたものがオブジェクトだと考えると分かりやすいかもしれません

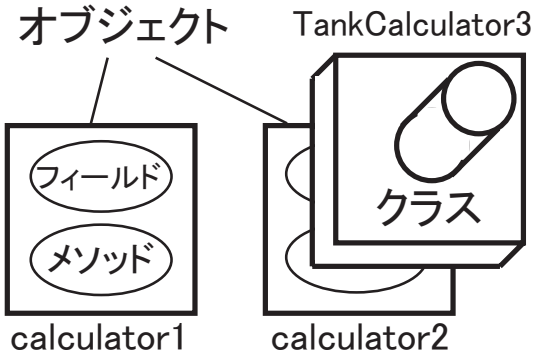

図 2　オブジェクトとクラス

例えば,「車」というクラスについて考えてみます.

- オブジェクト（インスタンス）は，一台一台の車です.
- フィールドは，その車の色，タイヤなどです.
- メソッドは,「進む」「クラクションを鳴らす」などです.

【演習】 何かオブジェクトを考え（例えばあなた自身），そのクラス，フィールド，メソッドとしてどのようなものがあるか，考えてください.（発表してもらいます）

1.4　オブジェクトの生成とフィールド

クラスの中でメソッドを使う際には，そのクラスのオブジェクト（**インスタンス**）を生成して，これのメソッドを呼び出すということが必要になります．

【インスタンスの生成】 インスタンスの生成と変数への代入は次のように行います．クラス名に () をつけたものはコンストラクタと呼ばれ，これによりインスタンスが生成されます[2]．

クラス名 変数名 = new **クラス名** ();

```
17:   TankCalculator3 calculator = new TankCalculator3();
```

順番としては，1. calculator という箱が作られ，2. TankCalculator3 クラスというハンコでインスタンスが生成され，3. そこへの参照が calculator に代入されます．

【フィールド】 オブジェクトで継続的に利用したい変数はクラスの中で**フィールド**として宣言します[3]．TankCalculator3.java では `TANK_AREA`，`INITIAL_TANK_LEVEL`，`flow[]` がフィールドとして宣言されています．

【アクセス法】 フィールドやメソッドは，クラスを表す変数にピリオド（これを「の」と読むと理解しやすいです）を挟んで，フィールド名やメソッド名を付加することでアクセスできます．例： calculator.TANK_AREA, calculator.report(...)

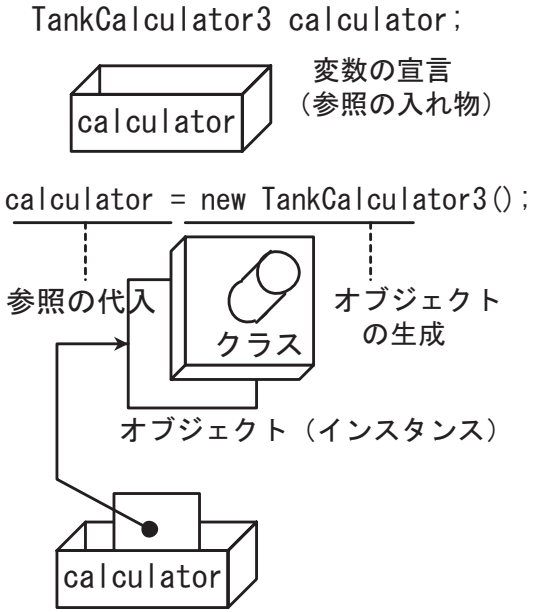

図 3　オブジェクトの生成

[2] 生成の方法の異なる複数のコンストラクタを定義することも可能です．

[3] 一方，メソッド内で宣言された変数はメソッド呼び出しが終了すると失われます．

1.5 メソッド（関数）

java では数学の関数と同じような機能を果たすものは**「メソッド」**と呼ばれます．TankCalculator3 の例では流入量を求めるメソッド getFlow() と データを書き出すメソッド report() が使われています

1.5.1 メソッドの定義

メソッドは class の中で次のように記述します．

アクセスレベル 返り値の型 メソッド名（引数の型 引数の名前，...）{
メソッドの本体
}

```
 7:  public double getFlow(double t) {
 8:      int index = (int)t % flow.length;
 9:      return flow[index];
10:  }
```

- アクセスレベル： public と書けば他のクラスから呼び出すことが可能です．private と書けばそのオブジェクト内からだけ呼び出すことができます．なんでも public にしたほうが便利だと思うかもしれませんが，他からの不用意な利用を避けるため，できるだけ private にするべきです．

- また，static を指定するとオブジェクト（インスタンス）を生成することなく，呼び出すメソッドを作ることができます．次の Lesson で紹介する数学関数はその例です．オブジェクトの持つ状態によって動的に（dynamic に）結果が変わるメソッドに比べ，結果がオブジェクトによらずに静的であるという意味で static と言います．
- 返り値を返さないメソッドは void 型として表現します．[4]
- java では同じ名前でも，引数の型や個数が異なっていれば異なるメソッドとして記述しなければなりません．
- 返り値（関数の計算結果）を返す命令が return です．
- メソッド内で宣言された変数のスコープはそのメソッドが呼び出されてから，return などで値を返して終了するまでです．

[4] main() メソッドがそうですね．

1.5.2 補足．引数について

メソッドに何らかの変数を引数として渡した場合，それがメソッドの中で変更されると，もとの変数はどうなるのでしょうか？

- int 型や double 型などの引数は，呼び出す際に，その値のコピーが作られてメソッドに渡されます．メソッド内で，対応する引数の値を操作しても，呼び出し側の変数の値は影響を受けません．
- これに対して配列やクラスのオブジェクトなどを渡す場合，コピーではなくて参照が渡されます．従って，配列の要素などをメソッド内で変更すると，もともとの配列の要素の値まで変更されてしまいます．[5]

[5] このような動作は**「副作用」**と呼ばれ，エラーも出ず禁止もされていませんが，多用するとプログラムを分かりにくくするとされます．

1.6 読みやすいプログラムを書くための工夫 — 注釈

ここまでは紙面の関係上，注釈（コメント）をあまりつけていませんでした．しかしながら，メンバ変数やメソッドなどを利用するようになると注釈を付けることが望まれます．java では注釈は以下の 3 通りの方法で書けます．このように付けられた注釈はプログラムの実行には影響を与えません．

1. “/*” と “*/” で挟まれた領域.
2. “//” から行末まで.
3. “/**” と “*/” で挟まれた領域.

3. は 1. に似ていますが，後で述べる javadoc 用です．プログラムの保守に便利な方法ですので身につけておくとよいでしょう．

良いコメントをつけるために

- 変数名などを適切につけることによって，極力コメントに頼らないプログラムにすべきです．

```
tankLevel = INITIAL_TANK_LEVEL + FLOW_RATE * time / TANK_AREA;
tl = ITL + FR * t / TA;     // これはダメ
```

- 何にでもコメントをつければよいというものではなく，そのプログラムを理解するために必要な部分にだけつけます．
- クラスやメソッドが何をしてくれるものなのか，どのように使われるべきなのかをコメントします．
- プログラム中にテストやデバッグのために一時的に挿入した部分にはコメン

トをつけておかないと，時間がたてばその意図などがわからなくなります[6].

- java ではあまり推奨されない使い方をあえてする場合は，それがミスでないことを示すためにコメントします.
- 英語でコメントするか日本語でコメントするかは悩みどころです.

[6] "//" 実行できるコードの前に書いて無効化すること（コメントアウト）がデバッグなどの際によく行われますが，そのまま放置すべきではありません.

メソッドの利用に関連しては

- 変数名と同様，メソッド名もよく考えて適切なものをつけます.
- 自然に設計するとメソッド名は通常，**動詞**を用います.
- （これに対して変数名やクラス名は通常，**名詞**になります.）
- 慣習として，以下のような動詞を前置した名前がよく用いられます．たとえばフィールドの値を設定するメソッドは "set"（例えば setTime）, 値を得るメソッドは "get"（例えば getFlow）, そのオブジェクトがある状態にあるかどうかを問い合わせるメソッドは "is"（例えば isAlive）,　何かを保有しているかどうかを問い合わせるメソッドは "has"（例えば hasChild）などです.

1.7 Javadoc を利用してみよう

Java にはプログラムのドキュメントを自動生成する javadoc というユーティリティがありプログラムの文書化を支援してくれます[7]．基本的な考え方は

> **クラス，メンバ，メソッドの直前にコメントを書いておけば，javadoc コマンドが自動的に web 用の html 形式のマニュアルを作成してくれる**

というものです．プログラムのドキュメントはコーディング中に書くのが最も効率的なのでスマートな考え方だと言えます．

使い方は簡単で

- クラス，メンバ，メソッドの直前に “`/**`”, “`*/`” で囲んでコメントを書く．
- `@param` や `@return` など， `@` で始まる予約語でメソッドのパラメータや返り値にコメントを書くことができます．
- javadoc の使い方は例えば TankCalculator4.java のドキュメントを生成する場合は

  ```
  javadoc -d docs TankCalculator4.java
  ```

  とします．ここで `-d docs` というオプションは生成する html ファイルの格納場所を指定するものです．
- これにより docs というフォルダの下にクラスに対応した TankCalculator4.html というファイルが出来ているはずですので web ブラウザで見ることができます．

[7] Java の ドキュメント自体もこれで書かれています

Javadoc の記述サンプル（TankCalculator4.java）を以下に付けておきます．最初にダウンロードしたファイル群の中にあります．

```
 1: /**
 2:  * タンクに水をためるシミュレーションです．
 3:  * @author 喜多 一
 4:  * @version 3.0
 5:  */
 6: public class TankCalculator4 {
 7: /**
 8:   * タンクの底面積，単位は m**2
 9:   */
10:   final double TANK_AREA = 20.0;
11: /**
12:   * タンクの初期水位，単位は m
13:   */
14:   final double INITIAL_TANK_LEVEL = 10.0;     //10.0 m
15: /**
16:   * タンクへの流入量パターン，一秒毎，単位は m**3/s
17:   */
18:   double flow[]
19:       = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0}; //m**3/s
20:
```

```
21: /**
22:   タンクへの流入量 (m**3/s) を得る.
23:   @param t 時刻，単位は秒
24:   @return その時刻の流入量，単位は mm*3/s
25: */
26:   public double getFlow(double t) {
27:     int index = (int)t % flow.length;
28:     return  flow[index];
29:   }
30:
31: /**
32:   タンクの状態を標準出力に書き出す.
33:   @param time 時刻
34:   @param flow 流入量
35:   @param tankLevel 水位
36: */
37:   public void report(double time, double flow, double tankLevel) {
38:     System.out.println(time + ", " + flow + ", " + tankLevel);
39:   }
40:
41:   public static void main(String args[]) {
```

```
42:     TankCalculator4 tankCalculator = new TankCalculator();
43:     final int TIME_HORIZON = 100;
44:     double tankLevel[] = new double[TIME_HORIZON+1]; //m
45:     tankLevel[0] = tankCalculator.INITIAL_TANK_LEVEL;
46:     System.out.println("Time(s), Flow(m**3/s), Tank Level(m)");
47:     for (int i = 0; i< TIME_HORIZON; i++) {
48:       double time = i;
49:       tankLevel[i+1] = tankLevel[i] + tankCalculator.getFlow(time)/tankCalculator.TANK_AREA;
50:       tankCalculator.report(time, tankCalculator.getFlow(time), tankLevel[i]);
51:     }
52:     double finalTime = TIME_HORIZON;
53:     tankCalculator.report(finalTime, 0.0, tankLevel[TIME_HORIZON]);
54:   } // End of main() method.
55: } // End of class TankCalculator4
```

先ほどの javadoc コマンドを入力すると，M:¥java¥docs というフォルダの下に TankCalculator4.html ファイルが得られます．見てみましょう．

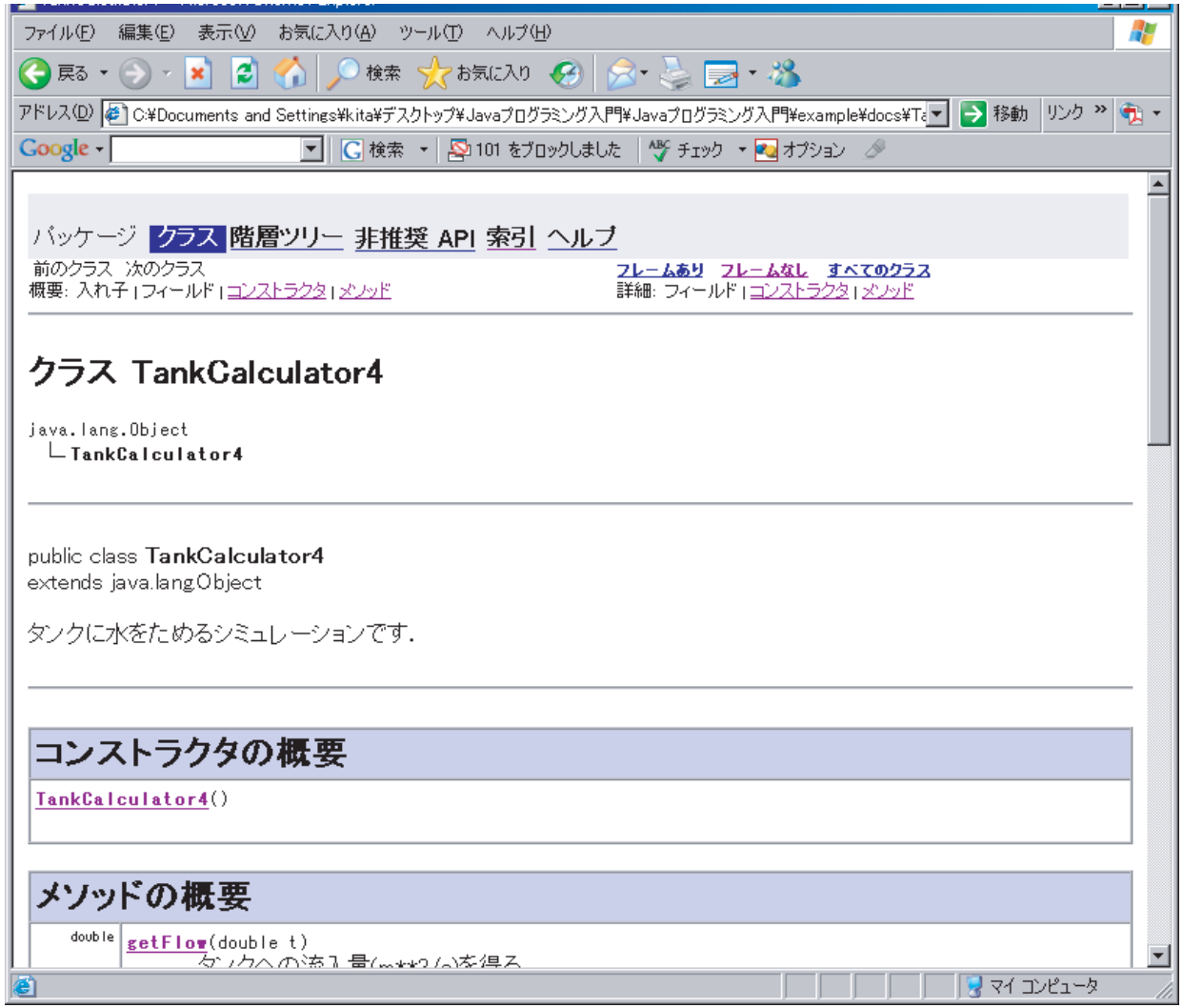

図 4　javadoc で得られたマニュアル

ここまで学んだこと

- プログラムの流れを制御するために，while, for, if などの構文を使う．
- ブロック`{ }`の中で宣言された変数は，ブロックの中でのみ有効.
- 多くのデータを扱うために，配列を使う．
- 配列は参照型なので，代入に注意が必要.
- 関数を用いるためにメソッドを使う．
- クラスの中で継続的に使う変数をフィールドという．
- クラスはハンコ．押されたものがオブジェクト（インスタンス）.
- メソッドやフィールドは，クラス名. メソッド名 のようにアクセスする．
- 読みやすいプログラムのために注釈をつけましょう．
- 注釈をうまくつけると，javadoc を用いて HTML にできる．