計算機プログラミング'99　＃3

久野　靖*

1999.9.22

0　はじめに

2週間のごぶさたでしたが、皆様のご意見では「そろそろオブジェクト指向らしいことがしたい」「今のままだと手続き型と変わらない」というのが目立つので、今回からいよいよクラスを作ろうと思います。ただ、クラスを作るに当たってはグラフィクスを題材にしたいので、まず前半でアプレットの作り方をやり、後半でそれをもとにクラスを作って見ることにします。

訂正!: 前回の講義の中で「2つのパッケージに同名のクラスが入っていると両方を「*」で import したら直ちにエラーになる」と説明しましたが、大嘘でした。実際には、その衝突しているクラス名が使われた時はじめてエラーになります。まあその方が親切ではありますね…

1　クラス、オブジェクト、メソッド

一応ですが、上の3大キーワードの復習をしておきます。まず、メソッドは「さまざまな動作」を行うものですね。その書き方は

```
クラス名.メソッド(...)　← クラスメソッド(APIのページの緑丸)
式.メソッド(...)　　← インスタンスメソッド(APIのページの赤丸)
```

でした。とりあえず、おもに後者を使うことになるでしょう。ここで大切なのは、「式」がクラスのインスタンスを表していて、そのクラスが指定したメソッドを持たないといけない、ということです。たとえば「"abcd"」という式は String のインスタンスを表しますから、

```
"abcd".toUpperCase()
```

は許されますが、「1」という式は int 型であり、これはクラス型ではないので (つまりインスタンスではなく値)、

```
1.toUpperCase()
```

は許されません。また、たとえインスタンスであっても

```
System.out.toUpperCase()
```

というのは、許されません。なぜなら、System.out は PrintStream のインスタンスであって、PrintStream クラスは toUpperCase() というメソッドを用意していないからです。たとえば犬に「おすわり!」と言えば分かるけど猫に「おすわり!」といっても分からないのと同様…おわかりかな?

*筑波大学大学院経営システム科学専攻

2 String クラスの練習問題

a. 入力文字列をすべて大文字に

今回は Java コードをいきなり見れば十分ですね。

```
import java.io.*;

public class r2ex4a {
  public static void main(String args[]) {
    try {
      BufferedReader in =
        new BufferedReader(new InputStreamReader(System.in));
      while(true) {
        System.out.print("String> "); System.out.flush();
        String str = in.readLine();
        if(str.equals("")) break;
        String res = str.toUpperCase();
        System.out.println(res);
      }
    } catch(Exception e) { System.err.println("!!"+e); }
  }
}
```

要するに、入力した文字列 str に対してメソッド `toUpperCase()` を呼ぶには「`str.toUpperCase()`」で、その結果 str の小文字をすべて大文字に変換した文字列が返されるので、それを `res` に入れて打出す。なお

```
System.out.println(str.toUpperCase());
```

のように 1 行にしてしまって変数 `res` は使わない、というのでもよい。

授業で説明した例題にあった for ループは文字列の内容を 1 文字ずつ処理するために必要だったのだが、この例のように「使える」メソッドがあるならそれを使ってしまえばループで処理しなくてすみ、楽ちんである。

b. 入力文字列に現われる「a」をすべて「*」に

```
import java.io.*;

public class r2ex4b {
  public static void main(String args[]) {
    try {
      BufferedReader in =
        new BufferedReader(new InputStreamReader(System.in));
      while(true) {
        System.out.print("String> "); System.out.flush();
        String str = in.readLine();
        if(str.equals("")) break;
        String res = str.replace('a', '*');
        System.out.println(res);
      }
    } catch(Exception e) { System.err.println("!!"+e); }
  }
```

```
  }
```

これは上とほとんど同様だが、toUpperCase() ではメソッドに引数がなかったのに対し、replace() では「どの文字を」「どの文字に」という 2 つの引数が必要になる。

なお、Java では「"..."」は String(文字列) を表し、「'?'」は 1 文字を表すという区別がるのに注意。

余談だが、ここで使っている Java の処理系は正式に日本語に対応していない。だからメッセージ出力に日本語を混ぜるだけなら問題ないが、replace() などで日本語を処理するのはうまく行かない。

最後に、先の例題と同様いきなり

```
  System.out.println(str.replace('a', '*'));
```

としても構わない (美観や趣味の問題)。

「a、e、i、o、u」をすべて「*」に

これはヒントにあるように、replace() を 5 回使えばできる。

```
import java.io.*;

public class r2ex4c {
  public static void main(String args[]) {
    try {
      BufferedReader in =
        new BufferedReader(new InputStreamReader(System.in));
      while(true) {
        System.out.print("String> "); System.out.flush();
        String str = in.readLine();
        if(str.equals("")) break;
        String res = str.replace('a', '*');
        res = res.replace('e', '*');
        res = res.replace('i', '*');
        res = res.replace('o', '*');
        res = res.replace('u', '*');
        System.out.println(res);
      }
    } catch(Exception e) { System.err.println("!!"+e); }
  }
}
```

まず「a」は先の通り処理したから、次に「res に対して」今度は「e」の処理を同様に行い、その結果を…新しい変数に入れてもいいのだが、同じ変数でもいいので res に入れ直してしまう。これを続ければいいわけだ。

ところで、毎回 res に入れ直さなくても次のようにしてもよい。

```
  String res = str.replace('a', '*').replace('e', '*').
    replace('i', '*').replace('o', '*').replace('u', '*');
```

なぜこれでいいかというと、replace() が返すのは String オブジェクトだから、それに対しまた次の replace() を読んでも構わないから。この方がかっこいいかどうか?

しかし、5 文字くらいだったらこれでも (また先の方法でも) いいけど、20 文字くらいになったらもう書くのがイヤになるでしょう? そこで次のようにしてもよい。

```
String res = str;
String sub = "aeiou";
for(int i = 0; i < sub.length(); ++i) {
  res = res.replace(sub.charAt(i), '*');
}
```

つまり、置き換える文字をまとめて文字列として用意し、その各文字を引数として replace() を適用する、というのを for 文で繰り返すわけである。まあ今回はここまでしなくてもよかったけど。

d. 三角形その 1

```
String> abcd
abcd
abc
ab
a
```

これは substring() を使って文字列の先頭文字から N 文字目までを取り出す、というのを N を変えながら繰り返せばよい。

```
import java.io.*;

public class r2ex4d {
  public static void main(String args[]) {
    try {
      BufferedReader in =
        new BufferedReader(new InputStreamReader(System.in));
      while(true) {
        System.out.print("String> "); System.out.flush();
        String str = in.readLine();
        if(str.equals("")) break;
        for(int i = 0; i < str.length(); ++i) {
          System.out.println(str.substring(0, str.length()-i));
        }
      }
    } catch(Exception e) { System.err.println("!!"+e); }
  }
}
```

つまり、str.length() が 4 であれば、for 文の中で「str.substring(0, 4)」、「str.substring(0, 3)」、「str.substring(0, 2)」、「str.substring(0, 1)」を順に出力するわけである。

e. 三角形その 2

```
String> abcd
abcd
 bcd
  cd
   d
```

これは後ろの方から取り出せばいいのだが、その前に適当な数の空白を用意しないと右側がそろってくれない。そのための処理が余計に必要になる。

```
import java.io.*;

public class r2ex4e {
  public static void main(String args[]) {
    try {
      BufferedReader in =
        new BufferedReader(new InputStreamReader(System.in));
      while(true) {
        System.out.print("String> "); System.out.flush();
        String str = in.readLine();
        if(str.equals("")) break;
        for(int i = 0; i < str.length(); ++i) {
          String space = "";
          for(int j = 0; j < i; ++j) space += " ";
          System.out.println(space + str.substring(i));
        }
      }
    } catch(Exception e) { System.err.println("!!"+e); }
  }
}
```

つまり、外側の for ループの i は「i 文字目から後ろを表示」なので、その前に i 文字ぶんの空白をつければよい。それを行うために、変数 space を用意してそれに i 回空白を追加して必要な長さの文字列を作っている。このために内側で j を 0 から i-1 まで変えながらループする。なお、`substring()` でパラメタ (引数) の数が 1 つのものは、その文字から末尾までを取り出すことになっている。

しかし、せっかくさまざまなメソッドがあるのに「1 文字ずつ空白をくっつける」という細かい操作をやるのは悔しいですね。たとえば、space という変数にうんと長い空白文字列が入れてあったとすると、N 文字の空白は「`space.substring(0, N)`」と書けば済むでしょう?

ここで問題は、どんな入力を持って来てもそれより長いような空白文字列を予め用意することができない、ということ。しかし、使う前に「十分長く」してしまうことは簡単ですね。というわけで次のようにすればよい。

```
import java.io.*;

public class r2ex4e1 {
  public static void main(String args[]) {
    try {
      BufferedReader in =
        new BufferedReader(new InputStreamReader(System.in));
      String space = "                                    "; // 長い空白!
      while(true) {
        System.out.print("String> "); System.out.flush();
        String str = in.readLine();
        if(str.equals("")) break;
        while(space.length() < str.length()) {  // 長さが足りなければ
          space = space + space;                // 長くする!
```

```
        }
        for(int i = 0; i < str.length(); ++i) {
          System.out.println(space.substring(0, i) + str.substring(i));
        }
      }
    } catch(Exception e) { System.err.println("!!"+e); }
  }
}
```

ところで、Java では「//」を書くとその右側には何を書いてもよい。これをコメント (注釈) 機能といい、後でコードを読む人 (1 週間後の自分?) に対する説明を書くのに使える。今後は、わかりにくい箇所は積極的にコメントを書いておこう。また、プログラムの先頭にはこのプログラムが何であるかのコメントを書く習慣をつけよう。

f. 巡回表示

```
String> abcde
bcda
cdab
dabc
abcd
```

これはじつは簡単で、単に「2 文字目から最後まで」と「先頭」とをくっつければ 1 文字ずれるので、それを何回も単に繰り返す。

```
// #2-演習 4-f 解答例 by Kuno, 1999.9.22
import java.io.*;

public class r2ex4f {
  public static void main(String args[]) {
    try {
      BufferedReader in =
        new BufferedReader(new InputStreamReader(System.in));
      while(true) {
        System.out.print("String> "); System.out.flush();
        String str = in.readLine();
        if(str.equals("")) break;
        String res = str;
        for(int i = 0; i < str.length(); ++i) {
          res = res.substring(1) + res.charAt(0);
          System.out.println(res);
        }
      }
    } catch(Exception e) { System.err.println("!!"+e); }
  }
}
```

3 配列の練習問題

演習 6

3 つの小問とも似たようなパターンなので、1 つにまとめて示す。

```
// #2-演習-abc 解答例 by Y.kuno, 1999.9.22
import java.io.*;

public class r2ex6 {
  public static void main(String args[]) {
    try {
      BufferedReader in =
        new BufferedReader(new InputStreamReader(System.in));
      // 配列にデータを読み込む
      int[] a = new int[100];
      int count = 0;
      while(count+1 < a.length) {
        System.out.print("" + count + "> "); System.out.flush();
        int v = (new Integer(in.readLine())).intValue();
        if(v == 0) break;
        a[count] = v;
        ++count;
      }
```

ここまでは例題と全然変わらないでよい。この先で最大と最少を求める。

```
      // 最大、最少を求める。
      int max = a[0];
      int min = a[0];
      for(int i = 1; i < count; ++i) {
        if(a[i] > max) max = a[i];
        if(a[i] < min) min = a[i];
      }
      System.out.println("max: " + max + ", min: "+min);
```

見ての通り、まず 0 番目を max、min 双方に入れておき、残りの各要素と照らしあわせて max、min を更新する。では次に合計と平均。

```
      // 合計、平均を求める。
      int total = 0;
      for(int i = 0; i < count; ++i) total += a[i];
      double average = (new Integer(total)).doubleValue() /
                       (new Integer(count)).doubleValue();
      System.out.println("total: " + total + ", average: " + average);
```

合計は変数 total に「足し込んで」行けばできる。さて、平均はそれを個数で割るのだけど、整数のままで割り算すると切捨て除算になるのであんまりよくない。Java では整数を実数にするには

- 単に実数の変数に格納する。
- 実数と混ぜて演算する。たとえば「count + 0.0」でもよい。

- きちんとクラス (ここでは Integer) のメソッドを使う。

などの方法があるが、解答例としてはきちんとしたいので 3 番目の方法を使っている。さて、最後に平均より大きいものを出力。

```
        // 平均より大きいものを出力。
        for(int i = 0; i < count; ++i) {
          if(a[i] > average) {
            System.out.println("larger than average: " + a[i]);
          }
        }
      } catch(Exception e) { System.err.println("!!"+e); }
    }
  }
```

ここでも「きちんとするなら」a[i] を double に変換してから average と比較するのだけど、そうしてもあんまり見やすくないのでここでは直接比較した (この場合も int が double に変換される)。

演習 7

これは Java そのものは疑似コードの通り作ればよい。ただし、最後の「a[k] を出力」は当然、「a[i] を出力」が正しかったです。すいません。

```
// #2-演習 7 解答例 by Y.kuno, 1999.9.22
import java.io.*;

public class r2ex7 {
  public static void main(String args[]) {
    try {
      BufferedReader in =
        new BufferedReader(new InputStreamReader(System.in));
      // 配列にデータを入力する
      int[] a = new int[100];
      int count = 0;
      while(count+1 < a.length) {
        System.out.print("" + count + "> "); System.out.flush();
        int v = (new Integer(in.readLine())).intValue();
        if(v == 0) break;
        a[count] = v;
        ++count;
      }
      // 昇順に並べ替える
      for(int i = 0; i < count; ++i) {
        for(int k = 1; k < count; ++k) {
          if(a[k-1] > a[k]) {
            int x = a[k-1]; a[k-1] = a[k]; a[k] = x;
          }
        }
      }
      // 結果を出力する
```

```
        for(int i = 0; i < count; ++i) {
          System.out.println(a[i]);
        }
      } catch(Exception e) { System.err.println("!!"+e); }
    }
  }
```

それで、なぜ並べ替えられるのか? まず、ループの中の if 文のところを見てみよう。これは 1 のように、ある箱と次の箱の間で「大小順が逆」だったら、「その 2 つの箱の内容を入れ換える」働きをする。

内側のループではそれが終わったら、今度は次の 2 つの箱の間で同じ動作をする…ということは、「大きい値」はつぎつぎに配列の後ろ側に向かってずれて行くことになる。ただし途中で自分より大きい値にぶつかったら、そこでは交換はされないので、その大きい値がその後右の方にずれて行くことになる。

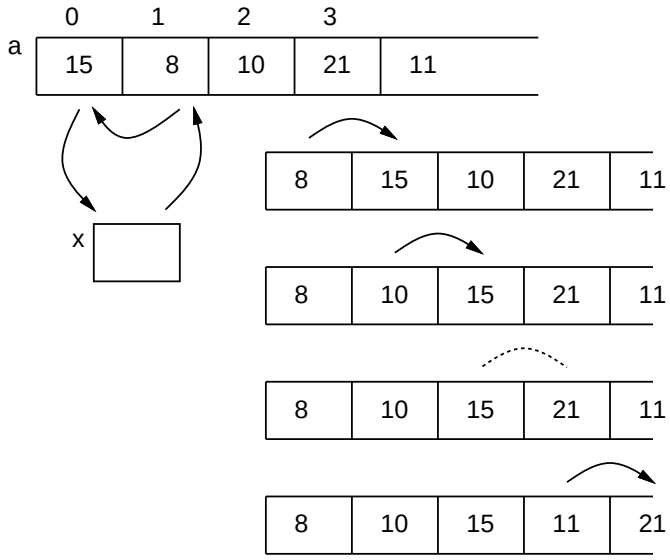

図 1: バブルソートの原理 (1)

ではこれを外側のループで何回も繰り返すことの意味は何だろう? それは 2 のように、1 回目の繰り返しで「最大の値」は必ず右端に来る。ということは、2 回目の繰り返しで「2 番手の値」がその左隣に落ち着くことは確実である。というようにして、count 回やれば最後の値まであるべき位置に落ち着く。

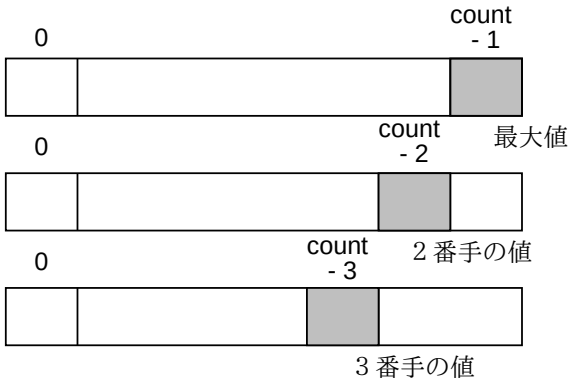

図 2: バブルソートの原理 (2)

ここでさらに考えると、実は内側のループは毎回「k < count - i」のところまでやれば十分である。というのは、そこから先は既に大きい奴が落ち着いているので、交換が起きることはないからである。

もっと考えると、以上のは「最も運が悪くても」大丈夫なようにしたわけであり、実際にはこんなに回数を回さなくても並び終わっているはずですね。そこで、「どこかから先は全部並んでいた」というのを変数 i に覚えておくようにしたら、もっと回数を節約できる。

```
int i = count;
while(i > 0) {
  int j = 0;
  for(int k = 1; k < i; ++k) {
    if(a[k-1] > a[k]) {
      int x = a[k-1]; a[k-1] = a[k]; a[k] = x;
      j = k - 1;
    }
  }
  i = j;
}
```

変数 j は内側ループの始まる前に 0 にして、それから交換が起きるたびに交換された左側の要素の番号を覚え直す。なので、内側ループが終わったあとは、最後に交換された箇所の左側が入っていて、それより右はすべて並び終わっている。この値を i に入れて、外側ループを繰り返す。最後は 1 回も交換が起きないと j も i も 0 になるので、ループが終わる。なかなかパズルみたいで難しいでしょう?

4 アプレット

さて、ここまでに作ってきた Java プログラム — 「java」コマンドで動かす奴 — は「スタンドアロンアプリケーション」と呼ばれる種類のものである。これとは別に、見たことはある人が多いと思うが、Web ブラウザの画面内で動くような Java プログラムもあり、これを「アプレット」と呼んでいる。

アプレットも (スタンドアロン) アプリケーションも Java プログラムであることに変りはないが、おもに外側の部分の書き方がちがっている。具体的にはアプレットでは次のような形になる。

```
import java.applet.*;
import java.awt.*;

public class 名前 extends Applet {
  型 変数 = 初期値;
  型 変数 = 初期値;
  ...

  public void paint(Graphics g) {
    文…
  }
  その他必要なメソッド定義…
}
```

もう少し細かく説明しよう。

- まず、これまでと使うライブラリが違うので、import 文で java.applet パッケージと java.awt パッケージを必ず指定する (他に必要なものがあれば import 文はいくつでも追加してよい)。
- 次に、アプレット用に作るクラスは最初に「public」という指定、そしてクラス名の後に「extends Applet」という指定が必要 (その具体的な意味は長くなるのでもっと後で説明する)。

- アプレットではメソッドを複数定義して使うので、それらのメソッドが共通に使う変数を次に書く。
- アプレットではさまざまなことができるが、最低限「何かを画面に描く」ことはしないと何も表示されないので意味がない。そのため、画面に描くためのメソッドを必ず用意する。これが `paint()` である。
- `paint()` は、ブラウザがアプレットに画面を描かせようと思った時何回でも繰り返し呼び出される。だから、この中であまりたくさん仕事をすると画面表示が遅くなってうまくない。呼び出されるとき、`paint()` には引数として `Graphics` クラスのインスタンスが渡される。このオブジェクトのメソッドを呼び出すことで、画面にさまざまな描画を行うことができる。
- アプレットでは `paint()` 以外にもいくつかメソッドを定義することがあるが、話が長くなるのでこれも後回しにする。

では、最初のアプレットの例題を見てみよう。

```
import java.applet.Applet;
import java.awt.*;

public class R3Sample1 extends Applet {
  Font fn = new Font("Helvetica", Font.BOLD, 24);
  public void paint(Graphics g) {
    g.setFont(fn);
    g.setColor(new Color(100, 0, 255));
    g.drawString("Hello, World", 30, 30);
  }
}
```

やってることは次の通り。

- まず、表示に使うフォント (字形) を大きなものにするため、Font オブジェクトを作成して変数 fn に入れる。これはアプレットの実行開始時に最初に行う。あとは全部 paint() が呼ばれたときの動作。
- `paint()` の中では、まず先に用意したフォントを使うように設定。
- 次に、描画に使う色を設定。
- そして最後に、画面上の位置を指定して文字列を表示する。

さて、アプレットを表示するためには、アプレットを埋め込んだ表示用の HTML ファイルが必要である。アプレットしか入っていない簡単なテスト用の HTML ファイルを示しておこう。

```
<html>
<head><title>r3sample1</title></head>
<body>
<h1>r3sample1</h1>
<applet code="R3Sample1.class" width=300 height=200></applet>
</body>
</html>
```

ここではこのファイルもアプレットの名前に合わせて R3Sample1.html というファイル名をつけることにしておく。

演習 1 以下の手順に従って、上の例題アプレットを含んだページを作ってブラウザで表示してみよ。

1. まず、ホームディレクトリの下の「WWW」というディレクトリへ行く。

```
cd
mkdir WWW          ←まだ作ってない人だけ
chmod go+rx WWW    ←〃
cd WWW
```

2. エディタでこのディレクトリの下にファイル R3Sample1.java を作成する。(別のディレクトリに作っても無意味だから注意!)
3. 「javac R3Sample1.java」でコンパイルする。
4. エディタでこのディレクトリの下にファイル R3Sample1.html を作成する。(別のディレクトリに作っても無意味だから注意!)
5. 作成したファイルが誰でも読めるように保護設定を変更。つまり「chmod go+r R3Sample1.*」
6. 完成したら Netscape でこのページを開く。URL は次の通り:

   http://smm/~自分のユーザ名/R3Sample1.html

なお、コードを手直しした場合は javac でコンパイルするのは当然だが、その後で「Shift キーを押しながら」Netscape の再読み込みボタンをクリックすること (そうしないとファイルを読み直してくれない)。

演習 2 次のことを行え。

a. java.awt.Graphics のドキュメントを見て、drawString() の使い方をチェックせよ。次に上の例題を修正して、文字の表示位置を変更してみよ。
b. java.awt.Font のドキュメントを見て、フォントの作り方をチェックせよ。次に上の例題を修正して、文字の大きさや形を変更してみよ。
c. java.awt.Color のドキュメントを見て、色の作り方をチェックせよ。次に上の例題を修正して、文字の色を変更してみよ。Hello, と World, を別々の色にできるとなおよい。

演習 3 java.awt.Graphics のドキュメントを見ると、円、矩形、線分、多角形などさまざまなものを表示するメソッドがあることが分かる。どれか 1 つ好きなものを選び、上の例題に追加してその図形も描くようにしてみよ。文字とは色を変えること。

5 メソッド

ここまで、「メソッド」とはどういうものかあまり説明しないままとりあえずやって来た。ここでまとめて説明しよう。メソッドが「一連の動作」(これまで散々やった計算とか代入とか枝分かれとか繰り返しとか) をひとまとめにしたものである、ということはお分かりになっていることと思う。

ここまで出て来た例題はどれもさほど複雑ではなかったので、すべての動作を 1 つのメソッドに入れてしまっても問題なかった。なので、アプリケーションでは main()、アプレットでは paint() というメソッドだけを作ればよかった。

しかし、もっと動作が込み入って来ると、自分の書く動作も複数のメソッドに分けた方が見通しがよくなる。たとえば、アプレットで「三角形」を描こうと思うと、fillPoligon() というメソッドが利用できるが、これは一般に多角形を描くためのものなので、座標を配列として渡さなければならず、その準備を含めるとかなり長い処理が必要になる。三角形を描くごとにそういう長い処理を書くとプログラムがごちゃごちゃで分かりにくくなる。

そこで、「三角形を描く」というメソッドを定義する。そうすれば、三角形が必要なところでは、どこでもこのメソッドを呼ぶだけで済む。そして実際に三角形を描くこちゃごちゃした作業は、そのメソッドの中で 1 処理を記述すればよい。具体的に見ていただこう。

```
import java.applet.Applet;
import java.awt.*;
```

```
public class R3Sample2 extends Applet {
  public void paint(Graphics g) {
    Color c = new Color(200, 110, 255);
    for(int i = 0; i < 6; ++i) {
      sankaku(g, c, 50, 180, 250-i*20, 200-i*10, 50+i*25, 100-i*10);
      c = c.darker();
    }
  }
  void sankaku(Graphics g, Color c,
      int x0, int y0, int x1, int y1, int x2, int y2) {
    int[] xpos = new int[]{x0, x1, x2};
    int[] ypos = new int[]{y0, y1, y2};
    g.setColor(c); g.fillPolygon(xpos, ypos, 3);
  }
}
```

ところで、これまでインスタンスメソッドは必ず「式. メソッド名 (...)」で呼び出す、という説明をしてきたが、インスタンスメソッドの中で同じクラスのインスタンスメソッドを呼び出す場合には単に「メソッド名 (...)」で呼び出すこともできるのだった。

6 メソッドと引数

ところで、「三角形を描く」メソッドを定義したとしても、いつも同じ場所に同じ色の三角形しか描けないのでは役に立たない。さまざまな場所に、さまざまな色の三角形を描きたいからメソッドを用意するわけである。

このため、メソッドには「引数」ないし「パラメタ」と呼ばれる仕組みが備わっている。つまり、メソッドを呼ぶ時には、呼ぶ側で「これこれの座標に、これこれの色で」という情報をつけて呼ぶ。これを「実引数」という。そして、メソッドの先頭部分で実引数を受け取るための「変数のようなもの」の宣言を並べる。これを「仮引数」という。実際にメソッドの本体が実行されるときは、1 番目の仮引数 (の名前を持った変数) には 1 番目の実引数の値、2 番目の仮引数 (の名前を持った変数) には 2 番目の実引数の値、…が入っていて、それを自由に参照できる (図 3)。これを利用することで、`sankaku()` は呼ばれるごとに色や位置の違う三角形が描けるわけである。

そしてもちろん、メソッドは複数定義してもよく、あるメソッドの中で別のメソッドを呼び出すことも自由である。たとえば、「ハロウィンの顔」を描くメソッド `kao()` で、目と口の部分は三角形だから `sankaku()` を呼び出して利用する、といった場合である。

```
import java.applet.Applet;
import java.awt.*;

public class R3Sample3 extends Applet {
  public void paint(Graphics g) {
    for(int i = 0; i < 4; ++i) {
       kao(g, new Color(200-i*20, 100+i*30, i*50),
           40+i*60, 50+i*30, 40+i*10);
    }
  }
  void kao(Graphics g, Color c, int x, int y, int r) {
    g.setColor(c); g.fillOval(x-r, y-r, 2*r, 2*r);
```

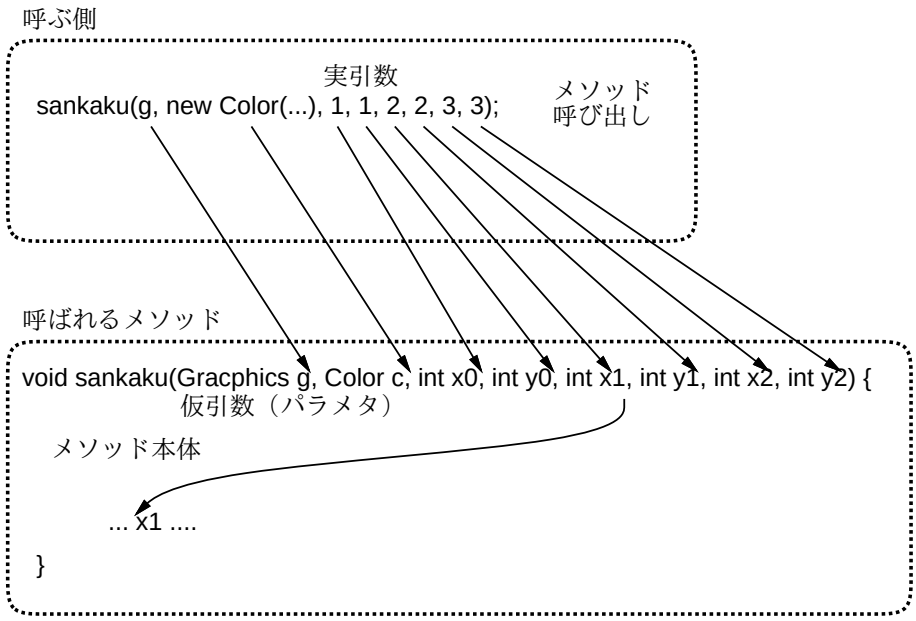

図 3: メソッドとパラメタ

```
    int u = r / 4;
    sankaku(g, c.brighter(), x-3*u, y-u, x-u, y-u, x-2*u, y-2*u);
    sankaku(g, c.brighter(), x+3*u, y-u, x+u, y-u, x+2*u, y-2*u);
    sankaku(g, c.brighter(), x-u, y+2*u, x+u, y+2*u, x, y+3*u);
  }
  void sankaku(Graphics g, Color c,
      int x0, int y0, int x1, int y1, int x2, int y2) {
    int[] xpos = new int[]{x0, x1, x2};
    int[] ypos = new int[]{y0, y1, y2};
    g.setColor(c); g.fillPolygon(xpos, ypos, 3);
  }
}
```

演習 4 上で出て来たメソッドの例題アプレットのどちらでも好きな方を打ち込んで動かせ。

演習 5 自分独自の絵を表示するオリジナルアプレットを作成せよ。できればメソッドを利用することが望ましい。もちろん、絵がかっこいい?美しい?ことは強く望まれる。

7 クラスを作る

前節までのプログラムでは、「顔」はあくまでも手続きによって描かれた「結果」ないし「痕跡」であって「もの」では全然なかった。これに対し、オブジェクト指向の考え方ではそれぞれの「顔」を「もの」として扱うのが当然ということになる。いよいよそのために、クラスを作ってみよう。クラスの形は既にさんざんやっているが再度整理しよう。

```
[public] class 名前 {
  型 変数 = 初期値;
  ...
  メソッド定義
  ...
```

```
  }
```

ここで、「変数」はインスタンス変数、つまり個々のオブジェクト (インスタンス) がそれぞれ持つ値を保持する変数を定義する。または先頭に static をつければクラス変数が定義できる。

メソッドも特に何も指定しなければインスタンスメソッド、つまり個々のオブジェクトに何らかの操作を行うメソッドになる。または先頭に static をつければクラスメソッドが定義できる。

実は、Java や C++ではメソッド (厳密にはそうは呼ばないが) にはもう 1 種類、「コンストラクタ」がある。コンストラクタはインスタンスの生成時の初期化を行う特別なメソッドで、必ずクラス名と同じ名前を持つことになっている。そして、「`new Integer(...)`」など new によりインスタンスを生成するときはコンストラクタが呼ばれ、「`...`」の部分はコンストラクタに引数として渡される。

では、さっき出て来た「顔」をクラスにしてみよう。名前は `Face` クラスとし、次の 6 つのメソッドを持つ。

- コンストラクタ — 色、X/Y 座標、半径を指定して顔を生成する。
- `draw()` — `Graphics` オブジェクトを受け取り、それを用いて顔を実際に画面に描く。
- `getX()`、`getY()`、`setX()`、`setY()` — X 座標と Y 座標の値を調べたり変更する。

では見てみよう。

```
class Face {
  Color c;
  int x, y, r, u;
  public Face(Color c0, int x0, int y0, int r0) {
    c = c0; x = x0; y = y0; r = r0; u = r / 4;
  }
  public void draw(Graphics g) {
    g.setColor(c); g.fillOval(x-r, y-r, 2*r, 2*r);
    sankaku(g, c.brighter(), x-3*u, y-u, x-u, y-u, x-2*u, y-2*u);
    sankaku(g, c.brighter(), x+3*u, y-u, x+u, y-u, x+2*u, y-2*u);
    sankaku(g, c.brighter(), x-u, y+2*u, x+u, y+2*u, x, y+3*u);
  }
  public int getX() { return x; }
  public int getY() { return y; }
  public void setX(int i) { x = i; }
  public void setY(int i) { y = i; }
  void sankaku(Graphics g, Color c,
      int x0, int y0, int x1, int y1, int x2, int y2) {
    g.setColor(c);
    g.fillPolygon(new int[]{x0, x1, x2}, new int[]{y0, y1, y2}, 3);
  }
}
```

ちなみに `sankaku()` はさっきと同じ下請けのメソッドだが、ちょっと書き方を短く工夫している。

さて、このクラスを用いてアプレットを書き直してみた。

```
import java.applet.Applet;
import java.awt.*;

public class R3Sample4 extends Applet {
  Face[] a = new Face[4];
  public void init() {
```

```
    for(int i = 0; i < 4; ++i) {
       a[i] = new Face(new Color(200-i*20, 100+i*30, i*50),
                       40+i*60, 50+i*30, 40+i*10);
    }
  }
  public void paint(Graphics g) {
    for(int i = 0; i < 4; ++i) a[i].draw(g);
  }
  // ここに Face クラスのコードを入れる
}
// (またはここに入れてもよい)
```

メソッド init() はアプレットの初期設定を行うためのメソッドで、ここで 4 つの顔を生成し、配列 a に格納する。paint() の方では 4 つの顔を順次描画する。とても簡単でしょう?

なお、Face クラスのコードは、アプレットクラスの「中に」入れても、外に出して並べてもよい。中に入れた場合は、Face クラスはアプレットクラスの「内部クラス」として外からは直接参照できないので他のクラスとの干渉が避けられる。外に出した場合は同じディレクトリに置いた他のクラスからも Face クラスが参照できる (ので、注意しないと名前が干渉したりバージョンの不一致が起きたりする)。ここでは内部クラスにしておいた方がよいと思う。

8 顔をキー操作で操る

上の例では描ける顔はさっきの例題と同じで、ちっともありがた味がなかった。そこで次は、4 つの顔をキーボードで自由に操るようにしてみよう。

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

public class R3Sample5 extends Applet {
  Face[] a = new Face[4];
  int cur = 0;
  public void init() {
    for(int i = 0; i < 4; ++i) {
       a[i] = new Face(new Color(200-i*20, 100+i*30, i*50),
                       40+i*60, 50+i*30, 40+i*10);
    }
    addKeyListener(new KeyAdapter() {
      public void keyPressed(KeyEvent e) {
        char c = e.getKeyChar();
        if(c >= '0' && c < '4') {
          cur = c - '0';
        } else if(c == 'h') {
          a[cur].setX(a[cur].getX()-5);
        } else if(c == 'j') {
          a[cur].setY(a[cur].getY()+5);
        } else if(c == 'k') {
          a[cur].setY(a[cur].getY()-5);
```

```
        } else if(c == 'l') {
          a[cur].setX(a[cur].getX()+5);
        }
        repaint();
      }
    });
  }
  public void paint(Graphics g) {
    for(int i = 0; i < 4; ++i) a[i].draw(g);
  }
  // ここに Face クラスのコードを入れる
}
```

キーボードのキーを押した時の情報をどうやって取るかという話を詳しく始めると大変だが、簡単に言うと「キーの押し/離しなどが起きた時にその情報を受け取るための内部クラスを定義し、そのインスタンスを生成して、受け取り用オブジェクトとして設定する」ことをやっている。よく分からなくてもまた後で説明するので、とりあえずメソッド `keyPressed()` の中で押したキーの情報が受け取れるものと思っておいてください。

さて、ここではキーが「0」〜「3」の場合はその番号をインスタンス変数 `cur` に覚え、「hjkl」のどれかだった場合は `cur` で指定された「顔」の位置を上下左右に 5 ピクセル移動ている。いずれの場合も、最後に `repaint()` を呼んでいるが、このメソッドは画面の再描画を行うように実行系に依頼する (その結果実行系が `paint()` を呼び出してくれる)。これだけで、「顔」がだいぶ「もの」らしくなったでしょう?

演習 6 「顔を上下左右に動かす」例題を打ち込んで動かせ。

演習 7 上下左右に動かすだけでなく、もっと別な変化も起こせるように直してみよ。たとえば顔の大きさ、色、表情を変化させるキーを指定するとか。

演習 8 「顔」以外の「もの」も現れるように変更してみよ。当然、新しいクラスを追加することになる (その「もの」もキーで操作できるとなおよい)。