Javaによるオブジェクト指向入門
— 教育用プログラミング言語に関するワークショップ2006 —

久野　靖*

2006.3.29

1　プログラミング教育で目指すもの…

□ 人によって様々だが、久野としては…

- 「手順的な自動処理」を体験してもらう
- プログラミングの面白さを知ってもらう (== 「プログラミングができる」+α)
- プログラミング言語のメカニズムの面白さを知ってもらう (やや高度) ←
- プログラミング言語のメカニズムの必然性を知ってもらう (かなり高度) ←

□ 今回の主題:　オブジェクト指向

2　Java言語

□ 歴史:

- Sun が 1991 年頃から開発
- HotJava+Applet(1995) --- Web ページ上で動くプログラム (今の Flash) →インターネットブームで急速に普及
- 今日では汎用の言語として認知

□ 言語としての特徴:

- 汎用のオブジェクト指向言語 (C++と同じ)
- 強い型検査 (C++よりも厳密)
- C++と比べて「行儀よく」「危ないことができない」「言語設計を綺麗に」(?)

□ 教育に使う場合の良い点…

- お行儀よくなるきっかけとなり得る
- 実務で使われる言語というモチベーション (他の言語だってそうなんだけど)
- 長いプログラムになったときに破綻しない (そのための言語ですから)

□ 教育に使う場合の弱点…

- 記述が長くなりやすい (困る)
- 最初からクラス、保護設定の概念が必要 (とても困る←オマジナイ化)
- 強い型は学習者によっては負担 (メリットが分かりにくい)

□ 強い型とは?

- 「Human h = ...」のように変数に型宣言する。
- 「h.getName()」とあったら… (1) メソッドはあるか?　(2) その引数や型は?　…等々をコンパイル時に型検査する (間違いがあればエラーに)
- ×動かさしてくれる前にエラーで止められるのは納得いかない
- ○型が合わないならどのみちエラー←それなら早く分かるのが吉 (納得しづらい)
- ◎型を設計しながらコードを設計しようよ← (一層納得しづらい)

3　実践例:オブジェクト指向グラフィクス

□ 背景

*筑波大学大学院ビジネス科学研究科

- オマジナイを少なく→アプレット前提でグラフィクス
- オブジェクト指向を教えたい→図形==オブジェクト
- アプレットだと学習者どうしの相互鑑賞が容易→切磋琢磨
- 強い型についてはそのつど機会があれば…

□ 前提:簡単な変数/演算の学習は済んでいるものとします

□ 今日の内容:アプレットで図形を表示

- Java 2 Standard Edition 1.4 前提

  http://java.sun.com/j2se/

- HTML ファイルとして次のような 2 行のファイル (名前:なんとか.html) を使用

```
<applet code="Sam1.class" width="400"
   height="200"></applet>
```

- Java プログラムは「なんとか.java」というファイルに入れ (「なんとか」は自由ではなく、クラス名と同じにする)、翻訳コマンド:

```
javac -target 1.1 Sam1.java
```

□ 以下で出て来るプログラムのカタチ:

```
import java.awt.*;
  ↑使う機能を短く書くための指定
public class SamX extends java.applet.Applet{
  ↑クラス開始
  public void paint(Graphics g){
    ↑メソッド開始
    (ここに動作を書く…)
  }                              ←メソッドおわり
}                                ←クラスおわり
```

□ クラス: 「もの」の種類。ここではアプレットとよばれる種別のもの

□ メソッド: 「機能」。ここでは「画面に塗る機能」(paint) を定義。

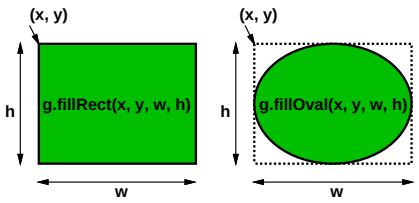
- 各人が「自分は画面をこう塗りたい」と定義→その人のプログラム

□ 最初のプログラム:四角と円を表示

```
import java.awt.*;

public class Sam1 extends java.applet.Applet {
  public void paint(Graphics g) {
    this.setBackground(Color.white);
    g.setColor(Color.green);
    g.fillRect(80, 90, 200, 40);
    g.setColor(Color.pink);
    g.fillOval(100, 100, 50, 50);
  }
}
```

□ メソッドの呼び出し方:「もの. 機能の名前 (パラメタ…)」

- 「this」は「このアプレット」をあらわす
- 「this.setBackground(...)」はアプレットの背景の色を設定
- 「g」は (ブラウザ側から渡される) 画面に描くための「ペン」
- 「g.setColor(...)」はペンの色を切り替える
- 「g.fillRect(...)」「g.fillOval(...)」は四角や楕円の塗りつぶし



□ 「Color. 名前」(できあいの色) は原色なのでいまいち

□ オブジェクトを作る:「new クラス名 (パラメタ…)」

- 色オブジェクト生成「new Color(R, G, B, α)」
- RGB: 赤/緑/青の強さ、α:透明度 (いずれも 0〜255 の整数)

```
import java.awt.*;

public class Sam2 extends java.applet.Applet {
```

```
  public void paint(Graphics g) {
    this.setBackground(Color.white);
    g.setColor(new Color(0, 255, 180, 190));
    g.fillRect(80, 90, 200, 40);
    g.setColor(new Color(200, 0, 100, 130));
    g.fillOval(100, 100, 50, 50);
  }
}
```

□ 演習:例題を参考に、円と長方形で絵を描いてみよう

□ 何がよくないか?

- 円は「中心と半径」で指定したい (長方形も同様がいいかも)
- 「円」「長方形」を直接扱いたい
- →これらもクラスとして用意

□ 作り方: 「new Circle(色, x, y, 半径)」、「new Circle(色, x, y, w, h)」ただし (x,y) は中心 (重心)

□ 描画は「図形.draw(ペン)」

```
import java.awt.*;

public class Sam3 extends java.applet.Applet{
  Circle c1 = new Circle(
    new Color(200,0,100,130),125,125,25);
  Rect r1 = new Rect(
    new Color(0,255,180,190),180,110,200,40);
  public void paint(Graphics g) {
    this.setBackground(Color.white);
    r1.draw(g); c1.draw(g);
  }
```

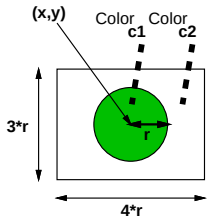
□ これらの図形クラスも一緒に用意する (一度用意すれば何回でも再利用)

- 「その図形を表す情報」→インスタンス変数。「色/x/y/半径」「色/x/y/w/h」
- 初期化する (new で呼び出す) メソッド、描画するメソッド

```
  class Circle {
    Color c; int x, y, r;
    public Circle(Color c0,int x0,int y0,int r0){
      c = c0; x = x0; y = y0; r = r0;
    }
    public void draw(Graphics g) {
      g.setColor(c); g.fillOval(x-r, y-r, 2*r, 2*r);
    }
  }
  class Rect {
    Color c; int x, y, w, h;
    public Rect(Color c0,
                int x0, int y0, int w0, int h0) {
      c = c0; x = x0; y = y0; w = w0; h = h0;
    }
    public void draw(Graphics g) {
      g.setColor(c); g.fillRect(x-w/2, y-h/2, w, h);
    }
  }
}
```

□ 演習:今度は、Circle クラスと Rect クラスを使って絵を描いてみよう

□ 次のような「旗」を描くには? (注:特定の国の旗ではありません)



□ Flag クラスがあるものとしてそれを利用→カンタンですね!

- 「new Flag(色 1, 色 2, x, y r)」

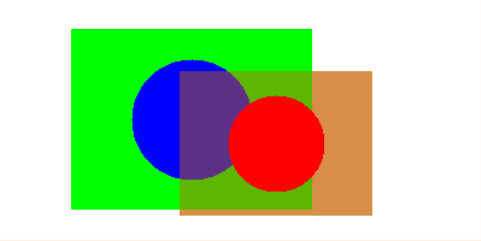
```
import java.awt.*;

public class Sam4 extends java.applet.Applet {
  Flag f1 = new Flag(
    Color.blue, Color.green, 160, 100, 50);
  Flag f2 = new Flag(
    Color.red,new Color(200,100,0,120),230,120,40);
  public void paint(Graphics g) {
    this.setBackground(Color.white);
    f1.draw(g); f2.draw(g);
  }
```

□ Flag クラスは Cicle と Rect を保持するだけ→カンタンですね!

```
  class Flag {
    Circle c; Rect r;
    public Flag(Color c1, Color c2,
                int x1, int y1, int r1) {
      c = new Circle(c1, x1, y1, r1);
      r = new Rect(c2, x1, y1, 4*r1, 3*r1);
    }
    public void draw(Graphics g) {
      r.draw(g); c.draw(g);
    }
  }
```

□ Circle、Rect は前と同じでよい→再利用 (何もしなくてよい)

```
  class Circle {
    Color c; int x, y, r;
    public Circle(Color c0,int x0,int y0,int r0){
      c = c0; x = x0; y = y0; r = r0;
    }
    public void draw(Graphics g) {
      g.setColor(c); g.fillOval(x-r,y-r,2*r,2*r);
    }
  }
  class Rect {
    Color c; int x, y, w, h;
    public Rect(Color c0,
                int x0, int y0, int w0, int h0) {
      c = c0; x = x0; y = y0; w = w0; h = h0;
    }
    public void draw(Graphics g) {
      g.setColor(c); g.fillRect(x-w/2,y-h/2,w,h);
    }
  }
}
```



□ 演習:　自分なりの Human クラス、Car クラス、または House クラスを定義し、それを画面に描いてみなさい。

□ 一度作ったものは「そのまま」部品として利用可能→オブジェクト指向の利点

- これを納得してもらえるには「ある程度継続的に制作する」場面が必要
- カリキュラム的には高校の選択科目レベル、大学初年度などが適切

□ それぞれの「もの」がプログラム上でそのまま表現→それを直接扱える

- たとえば:　個々のオブジェクトを動かす→アニメーション
- 例:　日没と月の出 (太陽と月の動きがヘンだと文句を言わないように)

□ 「海」「空」「太陽」「月」は既に作った Rect/Circle

```
import java.awt.*;

public class Sam5 extends java.applet.Applet {
  int time = 0;
  Rect sky = new Rect(
    new Color(100, 200, 255, 255), 200, 80, 400, 160);
  Rect sea = new Rect(
    new Color(0, 100, 180, 255), 200, 180, 400, 40);
  Circle sun = new Circle(Color.red, 220, 30, 20);
  Circle moon = new Circle(Color.yellow, 180, 250, 15);
  public void paint(Graphics g) {
    this.setBackground(Color.white);
    sky.draw(g); sun.draw(g); moon.draw(g); sea.draw(g);
  }
```

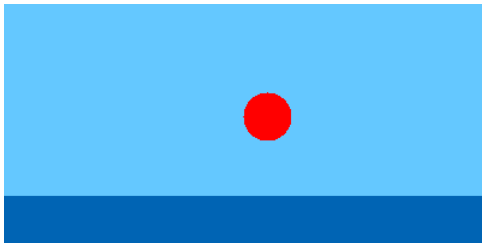
□ スレッドの詳細はここでは省略。概要は次の通り

- init() でアプレット開始時の動作を指定
- Thread オブジェクトの run() で並行動作を指定
- その中では「時間待ち→オブジェクトの変更→表示」の反復

```
  public void init() {
    (new Thread() {
      public void run() {
        while(true) {
          try {Thread.sleep(250);} catch(Exception ex){}
          time = time + 1;
          sun.moveBy(0, 2);
          if(time > 75 && time < 80) sky.darker();
          if(time > 75 && time < 78) sea.darker();
          moon.moveBy(1, -1);
          if(time > 150) return;
          repaint();
        }
      }
    }).start();
  }
```

□ Circle にはメソッド moveBy()、Rect にはメソッド darker() を増やした

```
  class Circle {
    Color c; int x, y, r;
    public Circle(Color c0,int x0,int y0,int r0){
      c = c0; x = x0; y = y0; r = r0;
    }
    public void draw(Graphics g) {
      g.setColor(c); g.fillOval(x-r,y-r,2*r,2*r);
    }
  }
  class Rect {
    Color c; int x, y, w, h;
    public Rect(Color c0,
                int x0,int y0,int w0,int h0){
      c = c0; x = x0; y = y0; w = w0; h = h0;
    }
    public void draw(Graphics g) {
      g.setColor(c); g.fillRect(x-w/2,y-h/2,w,h);
    }
  }
}
```



□ 演習： 自分の好きなアニメーションを作ってみよ。

4 ここから先の進め方…

□ プログラムが大きくなってくる→型検査のありがた味が分かる (と期待)

□ このように「ものの種類」「様々な機能」を増やして行く→ゴチャゴチャに→次の「言語的手段」の必要性

- さまざまな「図形」を統一的に扱う→動的分配→ Java のインタフェース
- さまざまな「図形」の共通機能をくくり出す→ Java の継承、抽象クラス
- さまざまな「図形」の動的生成/加工→自己反映計算→ Java のリフレクション

□ その他…

- Java には何でもあるが、どこまで教えれば十分かは難しいところ
- どこかで止まって「アルゴリズム」「計算量」などコンピュータ科学的側面も必要

5 まとめ

□ プログラミング言語において大切なこと→人間が「考えやすい」枠組みを提供する

□ オブジェクト指向言語の場合…

- クラス→「ものの種類」
- オブジェクト→「個々のもの」
- メソッド→「個々のものが持つ機能」

□ 自分のアイデアを素直にこれらに対応づけられるような導入が大切

6 質疑

- 大岩 (慶應義塾大学)： こうすればこんなことができます、だけに見える。これでクラスが書けるようになるのか?
- 久野： もちろん、もっと時間を掛けゆっくりやる。それで実際に書けるようになる。ただ、東大教養の 1 年選択なので「それは東大生だから」と言われると窮する。ぜひ他の先生もやってみて欲しい。

  休憩時間などに大岩先生とさらに議論しました。大岩先生としては「単にサンプルを見せるだけではメソッドがないではないか、それでは教えていないしできるようにもならないだろう」というご主張のようでした。久野としては、メソッドは必要だがそれが UML やオブジェクト指向設計というわけではない、ともかく言語機構を遊び倒して慣

れないと使えるようにならないのだから、その遊ぶ段階としてこのようなものは必要である、と反論しました。ただ、今後適切なメソッドの探求が必要という点は久野と大岩先生で一致しているものと感じました。