計算機プログラミングI 2005　久野クラス　＃9

久野　靖*

2004.12.16

はじめに

GUI 部品を使ったプログラムはどうでしたか。前回お話したように、今回からは次回までのレポートがないので、少しゆっくりめに説明ができると思います。今回は、インタフェースと関係の深い (歴史的にはインタフェースより昔からある) 「継承」(これまで何回もお目に掛かっている extends に関係しています) について取り上げます。あと、前回は GUI 部品経由の入力イベントを扱いましたが、もっと直接的なキーボードとマウスからの入力についても説明します。

1　前回の練習問題の解説 — 電卓

電卓をやった人は多そうなので例解を示そう。電卓で面倒くさいのは次の点。

- 数字を打つとそれは表示の後ろにくっつく。
- 「＋」などの演算ボタンを押すと今表示されている数値は一時的にどっかに覚えられる。何の演算ボタンかも覚えられる。
- 「＝」が押されると覚えている数値と今表示している数値の間で覚えている種類の演算をして結果を表示。

これが整理できないと作れない。あと、ボタンが沢山あるので大変だが、ループを使ってうまくこなせば短くなる。ここで、ボタンは動作をつけてしまえばあとは変数に覚えておく必要はないことに注意。

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class r8ex3d extends JApplet {
  JLabel l0 = new JLabel("0"); // 表示窓
  JLabel l1 = new JLabel(); // エラー表示
  String mem = "0";
  char op = '+';

  public void init() {
    Container c = getContentPane(); c.setLayout(null);
    c.add(l0); l0.setBounds(10, 10, 180, 30);
    c.add(l1); l1.setBounds(10, 260, 280, 30);
    String s = "9876543210.c+-*/=";
    for(int i = 0; i < s.length(); ++i) {
      JButton b = new JButton("" + s.charAt(i));
      c.add(b); b.setBounds(10+(i%3)*50, 50+(i/3)*30, 50, 25);
      b.addActionListener(new MyAdapter(s.charAt(i)));
```

*筑波大学大学院経営システム科学専攻

```
    }
  }
  class MyAdapter implements ActionListener {
    char ch;
    public MyAdapter(char c) { ch = c; }
    public void actionPerformed(ActionEvent evt) {
      l1.setText("");
      try {
        if(ch == 'c') {
          l0.setText("0");
        } else if(ch == '+' || ch == '-' || ch == '*' || ch == '/') {
          mem = l0.getText(); l0.setText("0"); op = ch;
        } else if(ch == '=') {
          double x = new Double(mem).doubleValue();
          double y = new Double(l0.getText()).doubleValue();
          if(op == '+')      l0.setText("" + (x+y));
          else if(op == '-') l0.setText("" + (x-y));
          else if(op == '*') l0.setText("" + (x*y));
          else if(op == '/') l0.setText("" + (x/y));
        } else {
          l0.setText(l0.getText() + ch);
        }
      } catch(Exception ex) { l1.setText(ex.toString()); }
    }
  }
}
```

機能に応じて別のアダプタクラスを作ることもできるが、ここでは 1 つのアダプタクラスだけを用意し、そのコンストラクタで「あなたは何のボタン」という情報を渡し (インスタンス変数 ch に保持)、その種類に応じて動作を切り替えるようにした。結構短いでしょ? なお、これも無名内部クラスで書けるが、まだ説明していない技がいるのでやめておいた。

2 インタフェースの復習と継承

2.1 インタフェースと implements

インタフェースがよく分からない、という意見を頂いているので、簡単に復習しよう。まず、ここまでで様々なオブジェクトに遭遇して来たが、その体験を通じて「オブジェクトとは中身は知らなくてもメソッドの呼び出し方を知っておけば使えるもの」ということは分かったと思う。

実はこのような性質は、人間に対してだけでなくプログラムの要素どうしを「くっつける」時にも活用できる。つまり、ちょうど電灯線の「ソケットとプラグ」のように、ある決まった規格に従った「差し込むもの」と「差し込まれるもの」は「差し込んで」くっつけて使え、その規格にさえ合っていればさまざまな「差し込むもの」をとりかえて差し込んで使える、というわけだ。

この状況は、スレッドに自分が作ったアダプタクラス MyRun のインスタンスを渡すところにまさに現われている (図 1)。つまり、Thread という 1 つのクラスにさまざまな動作を (各自の必要に応じて) 行わせるために、「それぞれ別の動作を行うアダプタ」を差し込んではめているわけである。

電灯線のソケットには通常の単相 100V 用、200V 用、3 相用などいろいろな規格がある。同様に、Java の場合はその「はめこみの規格」を「インタフェース」と呼ばれる単位を使って指定する。たとえば、スレッドのはめこみ規格は Runnable というインタフェースで定められている。その内容は (API ドキュメントの java.lang パッケージ中にあるのだがそれを書き写すと) 次の通り。

```
public interface Runnable {
```

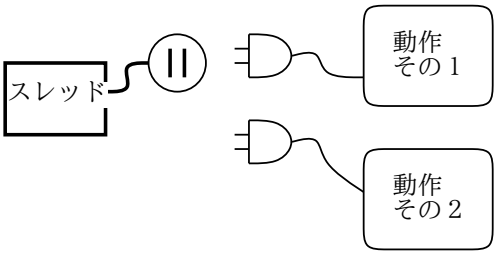

図 1: インタフェースはソケットとプラグ

```
  public void run();
}
```

つまり、このプラグには「run という名前の、引数なしのメソッドを持っているようなオブジェクトが差し込める」わけである。

ただし Java では、単に勝手に上記のメソッドを用意すればいいわけではなく、そのような (差し込むために使う) クラスはその冒頭で「このクラスはこれこれの規格に合っています」という宣言をしておく必要がある。これが「implements Runnable」という指定の持つ意味なのだった。一般に、インタフェース I 型の変数には、I を実装している (implements 指定している) どのクラスのインスタンスでも入れることができる。

2.2 動的分配

もう 1 つインタフェースを使った例として、Animation というインタフェースを自前で定義したところを思い出してほしい。

```
interface Animation {
  public void draw(Graphics2D g);
  putlic void addTime(double dt);
}
```

この方法は何がよかったかというと、さまざまな図形クラスをそれぞれ implements Animation 指定で作ってやれば、それはすべて

```
Animation[] a = new Animation[20];
```

で用意した配列 a に入れることができ、それに対して**メソッドを呼び出す時にも**、いちいちそれが「具体的に何の図形か」に煩わされることがない、という点である。たとえば

```
for(int i = 0; i < count; ++i) a[i].draw(g);
```

というところを実行すると、配列の 0 番目に円、1 番目に星型、2 番目に正方形が入っていたとすれば、円の draw()、星型の draw()、正方形の draw() がそれぞれ**自動的に**呼び出されるし、配列の中身を入れ換えればまた**その時点でそこに入っているオブジェクトのクラスで定義された** draw() が自動的に呼び出される。このように、「実行時にそこに実際に入っているオブジェクトの種類 (クラス) に応じて、そのクラスのメソッドを自動的に呼び出す」機能を「動的分配 (dynamic dispatch)」といい、オブジェクト指向言語の重要な機能である。なぜ重要か?

もし、この機能がないと上のループはこうなってしまう。実際、オブジェクト指向以前のプログラミング言語ではこういう処理が当り前だった。

```
for(int i = 0; i < count; ++i) {
  if(a[i] が円であれば) {
    円の draw(a[i]);
  } else if(a[i] が星型であれば) {
    星型の draw(a[i]);
  } else if(a[i] が正方形であれば) {
```

```
      正方形の draw(a[i]);
    }
  }
```

これは書くのが長くて面倒くさいだけでなく、次のような問題がある。

- プログラマにとって、「ここは何をやっているのか」ということが把握しにくくなる
- 図形の種類が増えるごとに、枝分かれを増やさなければならない

このような「負担」があればあるほど、プログラマの労力はそれを正しく処理することに取られてしまい、その分プログラムを発展させるのに回せる労力が減ってしまう (最後にはそれ以上発展させることができなくなる)。これを「ここでは a[i] に入っている図形を描く」とひとこと言えばすみ、また読んでもそう読めるようにしたのが、オブジェクト指向の最大の貢献だと私は考えている。

2.3 継承と extends

さて、インタフェースと implements について整理したところで、これと類似した概念である継承と extends について説明する。

これまでやってきて、いくつもの図形クラスを作るとき、いつも同じような記述を書いているのが気になったことはないだろうか? 同じことを何回も書くのではなく、もっと楽に、既に作ったある部分を「土台にして」新しい部分を足すだけで次のクラスを作る、というわけには行かないだろうか? 実はそのための機能が、これまでに何回も出て来た「extends」によって実現されている。この機能のことを「継承」(inheritance) という。その説明をしよう。

繰り返すと「継承」というのは、あるクラスを下敷き (土台) にして新しいクラスを作ることを言う。なぜそういう機能があるのかというと、既にできている「土台」を元にそこにちょっとだけ継ぎ足すことで、少しずつ機能を増やして行けるとプログラムが作りやすい場合があるから。継承について以下にまとめておく。

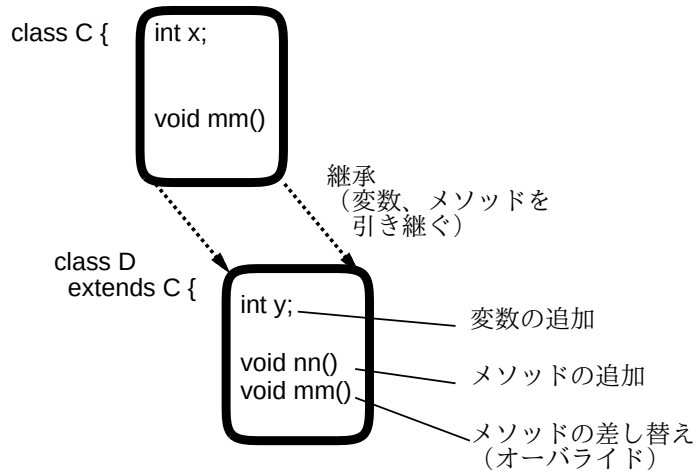

図 2: クラスと継承

- 下敷きになるクラスを「親クラス」「スーパークラス」、新しく作るクラスを「子クラス」「サブクラス」と呼ぶ。
- 継承を指定するには、クラス定義の先頭部で「class ... extends 親クラス ...」と指定する。
- 親クラスの変数、メソッドは子クラスにそのまま引き継がれる (継承される)。
- 子クラスでは、変数やメソッドを新たに追加できる。
- さらに子クラスでは、メソッドを別のものに差し替えることができる (これをオーバライドという)。
- 子クラスのコンストラクタやメソッドの中からは「super」という特別な名前を指定することで親クラスのコンストラクタやメソッドを呼び出すことができる。

ちんぷんかんぷんですか? でも「extends」というのはさんざん見たことがあるでしょう? 実は、皆様が作って来たアプレットも、Applet というクラスを「土台」にして、そのうちの「初期設定の処理」(init() のことですね)、「画面を描く処理」(paint() のことですね) などのメソッドを「オーバライドして」作ったものなのである。

もう 1 つ重要なこととして、「親クラスの型の変数には子クラスのオブジェクトが入れられる」ということがある。これは、インタフェース型の変数にそのインタフェースを実装しているオブジェクトが何でも入れられるのとよく似ている。つまり、親クラスというのは一種のインタフェースとしても使える (この場合もインタフェースと同様、動的分配が使える)。なお、親クラスがインタフェースを implements していれば、子クラスにもその性質は引き継がれる。

さらにまた重要なこととして、「まったく extends を指定していないクラスは Object というクラスのサブクラスになる」という規則がある。つまりクラス Object はすべてのクラスの直接または間接の親クラスであり、したがって Object 型の変数にはすべてのクラスのインスタンスを入れることができる。

では継承とインタフェースの実装がどう違うのかというと、継承では「インスタンス変数やメソッドなどの実装も一緒にくっついて (引き継がれて) くる」という点である。実は、歴史的に言うとインタフェースよりもこの継承機能の方が昔からある機能であり、「くっついて来る」のをやめたいこともあるので、そのためにインタフェースが発明された、という経緯がある。

また、継承に固有の使われ方として、親クラスで基本的なプログラムの構造をひととおり用意しておき、サブクラスでその一部分だけを差し替えてそれぞれの用途にあったプログラムを完成させる、というものがある。実際、アプレットの場合、我々はいつもサブクラスを定義して、そこでインスタンス変数を追加したり、メソッド init()(初期設定部分) や paint()(描画部分) を差し替えることで自分の用途にあったアプレットを作成していた。この考え方はもっとさまざまな方向で使うことができる。

2.4 GUI 部品と絵を共存させる

ここで、前々回までにやった描画と前回の GUI 部品を共存させる方法について説明しておこう。これまでは絵を描くときに、JApplet の paint() をオーバライドして絵を描いていた。しかしこのままだと、GUI 部品と絵が干渉するのであまりよくない。そうでははく、アプレットの中に GUI 部品などと同列に「絵を描く領域」があって、そこに絵が表示される方が扱いやすい (また、そうすれば複数の絵を描くこともできる)。

そのためには、絵を描く領域の土台 (継承元、親クラス) として JApplet の代わりに「何もしない四角い領域」である JPanel を使い、そのサブクラスを定義して paint()(や、アニメーション用の update() をオーバライドすればよい。実際にそのようにして GUI 部品と「飛ぶ円」のアニメーションを共存させる例題を見てみよう。

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class R9Sample1 extends JApplet {
  boolean go;
  double time;
  Animation[] a = new Animation[20];
  int count = 0;
  JButton b1 = new JButton("start");
  JButton b2 = new JButton("stop");
  MyPanel p1 = new MyPanel();
  public void init() {
    a[count++] = new FlyingCircle(Color.red, 100, 100, 40, 30, -40);
    a[count++] = new FlyingCircle(Color.blue, 100, 100, 30, 40, 50);
    Container c = getContentPane(); c.setLayout(null);
    c.add(b1); b1.setBounds(10, 10, 70, 30);
    c.add(b2); b2.setBounds(10, 40, 70, 30);
    c.add(p1); p1.setBounds(90, 10, 200, 180);
    b1.addActionListener(new ActionListener() {
```

```
  public void stop() { go = false; }
    public void actionPerformed(ActionEvent evt) {
      go = true; (new Thread(new MyRun())).start();
    }
  });
  b2.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent evt) { go = false; }
  });
}
class MyPanel extends JPanel {
  Image buf;
  public void update(Graphics g) {
    if(buf == null) { buf = createImage(getWidth(), getHeight()); }
    Graphics2D g2 = (Graphics2D)buf.getGraphics();
    g2.clearRect(0, 0, getWidth(), getHeight()); paint(g2);
    g.drawImage(buf, 0, 0, this);
  }
  public void paint(Graphics g) {
    Graphics2D g2 = (Graphics2D)g;
    for(int i = 0; i < count; ++i) a[i].draw(g2);
  }
}
class MyRun implements Runnable {
  public void run() {
    time = 0.001 * System.currentTimeMillis();
    while(go) {
      try { Thread.sleep(100); } catch(Exception e) { }
      double dt = 0.001 * System.currentTimeMillis() - time;
      for(int i = 0; i < count; ++i) a[i].addTime(dt);
      time += dt; repaint();
    }
  }
}
interface Animation {
  public void addTime(double dt);
  public void draw(Graphics2D g);
}
static class FlyingCircle implements Animation {
  Paint col;
  double gx, gy, rad, vx, vy;
  public FlyingCircle(Paint c, double x, double y, double r,
                      double vx1, double vy1) {
    col = c; gx = x; gy = y; rad = r; vx = vx1; vy = vy1;
  }
  public void addTime(double dt) {
    gx += vx*dt; gy += vy*dt;
    if(gx < 0 && vx < 0 || gx > 200 && vx > 0) vx = -vx;
    if(gy < 0 && vy < 0 || gy > 180 && vy > 0) vy = -vy;
  }
  public void draw(Graphics2D g) {
```

```
        g.setPaint(col);
        g.fillOval((int)(gx-rad), (int)(gy-rad), (int)rad*2, (int)rad*2);
      }
    }
  }
```

この例題では、各クラスの役割は次のようになっている。

- R9Sample1 — JApplet のサブクラスであり、アプレット全体を表す。他の補助クラス類をすべて含み、また主要なデータ構造もこのクラスのインスタンス変数として保持する。やることは init() で各種部品を配置するだけ。また、ボタンの動作としてアニメーションの開始/終了を行うようにしている。
- MyPanel — JPanel のサブクラスであり、中に「飛ぶ円」のアニメーションを表示する領域。このために paint() と update() をオーバライドしている。
- MyRun — Runnable を implements するクラスであり、メソッド run() として、配列 a に入っている図形の時刻を定期的に進めて repaint() する動作を提供している。
- Animation — アニメーション可能な図形のインタフェース。
- FlyingCircle — 飛ぶ円のクラス。

3 入力イベントの受け取り

3.1 入力イベント受け取りの原理

ウィンドウを持つプログラム (アプレットを含む) では、最初の方でやった、1 行ずつプロンプトを入力しては入力を待つプログラムと違って、プログラムはいつでも動ける状態にあって、ユーザ側から「ボタン押し」「マウスクリック」などの入力動作を行うと、そのことをプログラム側で受け取って対応する動作を行う、という形になっている。このような「入力があったよ」という事象のことを一般に「イベント」と呼ぶのだった。

前回は GUI 部品に関連してボタンのイベントを受け取る方法をやったが、今回はもっと低レベルな、マウスとキーボードのイベントを受け取る方法を学ぶ。これらは、プログラムの領域 (Component) 単位で受け取りが指定できる (ここまでに出て来た JApplet や JPanel などはすべて Component のサブクラスであり、同じメソッドが使える)。そのためには、次のメソッドを利用する。

addMouseListener()

マウスボタンの押し/離しに関係するイベントを受け取るためのアダプタクラスを設定する。アダプタクラスはインタフェース MouseListener を実装している必要がある。このインタフェースは次の 5 つのメソッドを定義している。

- mouseClicked(MouseEvent e) — マウスボタンのクリック
- mousePressed(MouseEvent e) — マウスボタンの押し下げ
- mouseReleased(MouseEvent e) — マウスボタンの離し
- mouseEntered(MouseEvent e) — マウスが領域内に入った
- mouseExited(MouseEvent e) — マウスが領域から出た

メソッドに渡されてくる MouseEvent オブジェクトからは、メソッド getX()、getY() でマウスの座標を取り出すことができる (詳しくは API ドキュメント参照)。

addMouseMotionListener()

マウスの動きに関係するイベントを受け取るためのアダプタクラスを設定する。アダプタクラスはインタフェース MouseMotionListener を実装している必要がある。このインタフェースは次の 2 つのメソッドを定義している。

- mouseDragged(MouseEvent e) — マウスボタンを押し下げた状態での移動 (ドラグ)
- mouseMoved(MouseEvent e) — マウスボタンが押されていない状態での移動

これらのメソッドはマウスが動いている間、適当な時間間隔で繰り返し呼び出される。

addKeyListener()

キーボードの打鍵に関係するイベントを受け取るためのアダプタクラスを設定する。アダプタクラスはインタフェース KeyListener を実装している必要がある。このインタフェースは次の 3 つのメソッドを定義している。

- keyTyped(KeyEvent e) — キーが打鍵された
- keyPressed(KeyEvent e) — キーが押し下げられた
- keyReleased(KeyEvent e) — キーが離された

メソッドに渡されてくる KeyEvent オブジェクトからは、メソッド getKeyChar() で打鍵された文字を取り出すことができる (詳しくは API ドキュメント参照)。

お手伝い用のクラス

上の 3 つのインタフェースはどれも、複数のメソッドを持っているので、アダプタクラスを作るときもそれらのメソッドをすべて用意する必要がある。しかし、たとえばマウスクリックだけに興味がある (動作をつける) のに、残りの 4 つの「何もしない」メソッドを用意するのは苦痛である。そこで、あらかじめそれぞれのインタフェースを実装したクラス MouseAdapter、MouseMotionAdapter、KeyAdapter が用意してある。これらのクラスでは、インタフェースの各メソッドを「何もしない」内容として定義しているので、自分ではこれのサブクラスを作って興味のある (動作をつけたい) メソッドだけを差し替えればいいわけである。

3.2 例題: マウスボタン押し下げを受け取る

とりあえず簡単な例題として、2 つ円が表示されているが、1 つの円の上でマウスボタンを押すとその円が横に動くというものを示そう。

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class R9Sample2 extends JApplet {
  Circle c1 = new Circle(Color.red, 100, 100, 15);
  Circle c2 = new Circle(Color.blue, 150, 100, 20);
  public void init() {
    addMouseListener(new MouseAdapter() {
      public void mousePressed(MouseEvent e) {
        if(c1.hit(e.getX(), e.getY())) {
          c1.moveTo(c1.getX()+3, c1.getY()); repaint();
        }
      }
    });
  }
  public void paint(Graphics g) {
    Graphics2D g2 = (Graphics2D)g;
    g2.clearRect(0, 0, getWidth(), getHeight()); c1.draw(g2); c2.draw(g2);
  }
```

```
  static class Circle {
    Paint col;
    double gx, gy, rad;
    public Circle(Paint c, double x, double y, double r) {
      col = c; gx = x; gy = y; rad = r;
    }
    public void moveTo(double x, double y) { gx = x; gy = y; }
    public double getX() { return gx; }
    public double getY() { return gy; }
    public boolean hit(double x, double y) {
      return (x-gx)*(x-gx) + (y-gy)*(y-gy) <= rad*rad;
    }
    public void draw(Graphics2D g) {
      g.setPaint(col);
      g.fillOval((int)(gx-rad), (int)(gy-rad), (int)rad*2, (int)rad*2);
    }
  }
}
```

Circle クラスに追加したメソッド hit() は、X/Y 座標を指定するとその座標が円の範囲に入っているか否かを真偽値として返す。アプレット本体では、メソッド init() でマウスイベントを受け取るアダプタを (無名内部クラスとして定義して) 設定し、その mousePressed() の中では「マウスボタンが押された時の座標が円 c1 の上だったら、円 c1 を 3 ピクセル横に動かす」動作を行っている。

演習 1 上の例題をそのまま打ち込んで動かせ。

演習 2 動いたら次のように改造してみよ。

a. 円 c2 も動くようにしてみよ。
b. 円の外をつつくと、その位置に円 c1 がワープするようにしてみよ (最後につついた円がワープするともっとよい)。
c. ついた円が重なり順の上に出るようにしてみよ。
d. マウスをドラグすると円 c1 がついてくるようにしてみよ (最後につついた円がついてくるともっとよい)。(ヒント: addMouseMotionListener() でマウス移動イベントを受け取る。)
e. キーボードから「<」「>」を打鍵すると円 c1 がちょっとずつ大きく/小さくなるようにする (最後につついた円の大きさが変化するともっとよい) (ヒント: addKeyListener() で打鍵イベントを受け取る。)
f. 円の外をつつくと、そこに新しい円が追加されるようにする。色や大きさは乱数で決める。(ヒント: 円を配列に保持するようにして、その配列に追加していく。)
g. f に c、d、e の機能を追加する。

3.3 例題: マウスでものをドラグする

ではもう少し高度な例題として、「飛ぶ円」をマウスでドラグ可能にしてみる。ただし、飛ぶ円以外のものも増やして行けるように、アニメーション可能かつドラグ可能なものをあらわすインタフェース DraggableAnimation を定めて、アプレット側はこのインタフェースにあてはまるオブジェクトなら何でも扱えるように作成している。冒頭部分は通常通り。

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
```

```
public class R9Sample3 extends JApplet {
  Image buf;
  boolean go;
  double time;
  DraggableAnimation[] a = new DraggableAnimation[20];
  DraggableAnimation target = null;
  int count = 0;
  public void init() {
    a[count++] = new FlyingCircle(Color.red, 100, 100, 15, 30, -40);
    a[count++] = new FlyingCircle(Color.blue, 150, 100, 20, 40, 50);
    addMouseListener(new MyAdapter());
    addMouseMotionListener(new MyAdapter2());
  }
```

アプレットの初期設定の最後に、マウスイベント受け取り用のアダプタクラスを設定している。2 つ必要なのは、押し/離しとドラグは別のアダプタを使うようになっているため。

次に paint() とかアニメーション関係はこれまでと変わらない。

```
  public void update(Graphics g) {
    if(buf == null) { buf = createImage(getWidth(), getHeight()); }
    Graphics2D g2 = (Graphics2D)buf.getGraphics();
    g2.clearRect(0, 0, getWidth(), getHeight()); paint(g2);
    g.drawImage(buf, 0, 0, this);
  }
  public void paint(Graphics g) {
    Graphics2D g2 = (Graphics2D)g;
    for(int i = 0; i < count; ++i) a[i].draw(g2);
  }
  public void start() { go = true; (new Thread(new MyRun())).start(); }
  public void stop() { go = false; }
  class MyRun implements Runnable {
    public void run() {
      time = 0.001 * System.currentTimeMillis();
      while(go) {
        try { Thread.sleep(100); } catch(Exception e) { }
        double dt = 0.001 * System.currentTimeMillis() - time;
        for(int i = 0; i < count; ++i) a[i].addTime(dt);
        time += dt; repaint();
      }
    }
  }
```

ではいよいよ、マウス操作の部分に進む。マウスボタンの押し/離しについては、押した時に一番上に描かれているものから順に「当たっているもの」を探し、見つかったらそれを変数 target に入れて、ドラグ状態にする。離した場合はドラグ終了として、target を null にする。

```
  class MyAdapter extends MouseAdapter {
    public void mousePressed(MouseEvent e) {
      for(int i = count-1; i >= 0; --i)
        if(a[i].hit(e.getX(), e.getY())) {
          target = a[i]; target.setDrag(true); break;
        }
```

```
    }
    public void mouseReleased(MouseEvent e) {
      if(target != null) { target.setDrag(false); target = null; }
    }
  }
```

ドラグ中の場合は、繰り返し mouseDragged() が呼ばれるので、そのつど現在時刻と X/Y 座標をパラメタとして dragTo() を呼ぶ。

```
  class MyAdapter2 extends MouseMotionAdapter {
    public void mouseDragged(MouseEvent e) {
      if(target != null) {
        double t = 0.001 * System.currentTimeMillis();
        target.dragTo(e.getX(), e.getY(), t); repaint();
      }
    }
  }
```

インタフェース DraggableAnimation では、前回の Animation に比べて、当たり判定、ドラグ開始/終了、そしてドラグのメソッドが追加されている。

```
  interface DraggableAnimation {
    public void addTime(double dt);
    public void draw(Graphics2D g);
    public boolean hit(double x, double y);
    public void setDrag(boolean b);
    public void dragTo(double x, double y, double t);
  }
```

では最後に飛ぶ円のクラス。冒頭部分は前回の例題と変わらない。ただ、インスタンス変数として「ドラグ中」を表す dragging と、ドラグイベントの時刻を記録する lasttime とが追加されている。addTime() ではドラグ中は飛ばないようにしている。文「return;」は、メソッドの実行を終らせて呼び出し元に帰ることを意味する。

```
  static class FlyingCircle implements DraggableAnimation {
    static final int MAX_X = 300, MAX_Y = 200;
    Paint col;
    double gx, gy, rad, vx, vy, lasttime = 0.0;
    boolean dragging = false;
    public FlyingCircle(Paint c, double x, double y, double r,
                        double vx1, double vy1) {
      col = c; gx = x; gy = y; rad = r; vx = vx1; vy = vy1;
    }
    public boolean hit(double x, double y) {
      return (x-gx)*(x-gx) + (y-gy)*(y-gy) <= rad*rad;
    }
    public void draw(Graphics2D g) {
      g.setPaint(col);
      g.fillOval((int)(gx-rad), (int)(gy-rad), (int)rad*2, (int)rad*2);
    }
    public void addTime(double dt) {
      if(dragging) return;
      gx += vx*dt; gy += vy*dt;
      if(gx < 0 && vx < 0 || gx > MAX_X && vx > 0) vx = -vx;
```

```
      if(gy < 0 && vy < 0 || gy > MAX_Y && vy > 0) vy = -vy;
    }
```

最後にドラグ処理。1 つ前の時刻が記録されていたら、そこから今までの時刻差と位置の差を用いて速度を計算する。なめらかに速度が変化するように、これまでの速度と 7:3 で移動平均を取っている。

```
    public void setDrag(boolean b) { dragging = b; }
    public void dragTo(double x, double y, double t) {
      if(lasttime != 0.0 && t > lasttime) {
      double dt = t - lasttime;
      vx = 0.7*vx + 0.3*((x-gx)/dt); vy = 0.7*vy + 0.3*((y-gy)/dt);
      }
      gx = x; gy = y; lasttime = t;
    }
  }
}
```

かなり長かったけど、これでマウスイベントとアニメーションを組み合わせることもできると分かりました。よかったら冬休み課題などでも活用してみてください。

演習 3 上の例題をそのまま打ち込んで動かせ。

演習 4 動いたら次のように改造してみよ。

a. つついた円が重なり順の上に出るようにしてみよ。
b. 円以外の図形を追加してみよ。
c. ドラグするとドラグされたのと反対方向に動く天邪鬼な円にしてみよ。
d. 等速直線運動でない動きをするものを追加してみよ。それも何らかの形でドラグできるようにすること。
e. 最後につついたものをキーボードで操れるようにしてみよ。(インタフェースの追加が必要。)
f. 飛んでいるものが衝突して跳ね返るようにしてみよ。(インタフェースの追加が必要、かつかなり高度。)

A 本日の課題 9A

「演習 2」または「演習 4」(の小課題 1 つ) で作成したアプレットを格納した WWW ページのための HTML ファイルを、自分の `cp1` ディレクトリの下に `report9a.html` という名前で作成すること。また、そのプログラムのコードはいつも通り、「本日中に」久野までメールで送付してください。具体的な内容は次の通り。

1. Subject: は「Report 9A」とする。
2. 学籍番号、氏名、投稿日時を書く。
3. 選んだプログラム 1 つのソース。
4. その簡単な説明。
5. 下記のアンケートの回答。

   Q1. 継承について分かりましたか?
   Q2. マウスイベント、キーイベントの受け取り方が分かりましたか?
   Q3. その他感想、要望等あればどうぞ。

B 次回までの課題

次回までの課題はありません。前回資料に掲載した、冬休み課題 9B に取り掛かってください。(アプレットのファイル名が `report9a.html` となっていましたが、当然 `report9b.html` のつもりでしたので訂正します。) では皆様、よいお年を。