情報科学特殊講義'2000 ＃8

久野　靖*

2000.1.24-27

0　はじめに

4日間に渡っておつき合い頂いたこの講義もこれでおしまいです。最後に「落ち穂拾い」として、日本語のちゃんとした扱い方と、継承を利用してこんな風なことができるんだよ、というデモンストレーションをやります。

1　日本語をちゃんと扱うには…

これまで、アプレットで日本語を扱う方法をちゃんと説明していませんでした。文字を扱うときには、必ず文字コード(文字とビット列の対応規則)が問題になります。さらに、ファイルはバイト(1バイトは8ビット)単位でデータを格納するので、その格納方法も問題になります。

英字はASCIIと呼ばれる、1文字8ビットのコードが標準的に使われていて、1バイトに1文字入れるという方法で問題ないのですが、日本語は8ビットでは入りきらないので2バイトを組にして使ったりします。その辺で歴史的事情から、日本語の文字コート系として使われているものに次のものがあります。

- ISO-2022-JP (JISコード) — 切り替え符合で1バイトと2バイトを切り替える。
- EUC (日本語EUC) — 切り替え符合の代りに各文字の8ビット目で区別する。我々のシステムのUnix側でおもに使用している。
- SJIS (MS漢字コード) — EUCと似ているが、ただし8ビットカナ(いわゆる半角カナ)と共存できるようにいじったコード。パソコン系(Windows95/98、MacOSなど)で使われている。

これに対し、Javaでは日本語に限定せずさまざまな国の文字をまとめて扱うコード系であるUNICODEを採用しています。このため、上のどれかのコードでプログラムを書いたとしても、そのままではUNICODEでないのでうまく画面に表示できないことがあります(うまくいってしまうこともあるが、それは偶然であって他のブラウザではだめかもしれない)。

そこで、日本語を使うプログラムの場合は次の方法を使ってください。

- Javaのプログラムは普通に打ち込んでいいが、日本語を含む場合はファイル名として「.java」ではなくたとえば「.java.txt」とか別のものにしておく。
- native2asciiというプログラムでそのファイルを「.java」ファイルに変換する。たとえば次のようにする。

```
native2ascii -encodeing JISAutoDetect X.java.txt X.java
```

- 変換したファイルをこれまで通りにjavacでコンパイルして利用。

では例題を1つだけ見ていただこう(しょうもない内容ですが)。

*筑波大学大学院経営システム科学専攻

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

public class R8Sample1 extends Applet {
  TextField t1 = new TextField("");
  Button b1 = new Button("チエック!");
  Label l1 = new Label("");
  public void init() {
    setLayout(null);
    add(t1); t1.setBounds(20, 50, 200, 40);
    add(b1); b1.setBounds(20, 100, 60, 40);
    add(l1); l1.setBounds(20, 150, 200, 40);
    b1.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        if(t1.getText().equals("れなちゃん")) {
          l1.setText("かわいい!"); t1.setText("");
        } else {
          l1.setText("べっつにいー。"); t1.setText("");
        }
      }
    });
  }
}
```

これを「R8Sample1.java.txt」に打ち込んだとして、コンパイルするのは次のようになる。

```
native2ascii -encoding JISAutoDetect R8Sample1.java.txt R8Sample1.java
javac R8Sample1.java
```

なお、R8Sample1.java の中がどうなっているかは見物してみてください。

演習 1 日本語を入力させて取り扱うアプレットを作って動かせ。

2 マウスイベント、キーイベントの利用

これまでは「動作をつける」のは GUI 部品のボタンについてばっかりだったが、実は直接キーの押し下げやマウスの操作を受け取ってそれに対して反応することもできる。そのやり方の概要は、次の通り。

- ActionListener ではなく、KeyListener(キーボードのイベント)、MouseListener(マウスボタンのイベント)、MouseMotionListener(マウスが動いた時のイベント) になる。
- GUI 部品ではなくアプレットに直接動作をつける。つまり、「addKeyListener(...)」を (オブジェクト指定なしで) 直接呼ぶ。
- ActionListener と違って、「押し」「離し」など複数の動作がある。そのため、アダプタクラスを継承して必要なメソッドだけオーバライドする。

最後のところがよく分からないでしょう? つまり、マウスボタンだったら「押す」「離す」があるから、今回はイベントアダプタも「押す」「離す」という複数のメソッドを用意してある。しかし、「押す」場合だけに興味があるのなら、「離す」メソッドは別に「何もしない」ので作らなくていい。しかし、インタフェースとしては「離す」メソッドがあるので作らないとエラーになる。

そこで、あらかじめ「押す」「離す」など全部の場合において「何もしない」クラスを用意しておいて、そのサブクラスを作って、そこで「押す」メソッドだけオーバライドすればよいわけである。と、ごたくは難しいが、書き方としてはこれまでやってきた内部クラスの場合と同じである (implements のほかに extends でも同じ書き方が使えるのでしたね)。

では、これを使ってマウスとキーのイベントを受け取る例をやってみよう。

注意! なぜかこの例題は appletviewer でうまくキーのイベントが取ないことがあります。なぜだろう…

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

public class R8Sample2 extends Applet {
  Font fn = new Font("Helvetica", Font.BOLD, 36);
  String mesg = "?";
  Color col = Color.blue;
  int xpos = 100;
  int ypos = 100;
  public void init() {
    addMouseListener(new MouseAdapter() {
      public void mousePressed(MouseEvent evt) {
        col = Color.pink; xpos = evt.getX(); ypos = evt.getY(); repaint();
      }
    });
    addMouseMotionListener(new MouseMotionAdapter() {
      public void mouseDragged(MouseEvent evt) {
        col = Color.yellow; xpos = evt.getX(); ypos = evt.getY(); repaint();
      }
    });
    addKeyListener(new KeyAdapter() {
      public void keyPressed(KeyEvent evt) {
        if(evt.getKeyChar() != evt.CHAR_UNDEFINED) {
          mesg = mesg + evt.getKeyChar();
        }
        col = Color.red; repaint();
      }
      public void keyReleased(KeyEvent evt) {
        col = Color.green; repaint();
      }
    });
  }
  public void paint(Graphics g) {
    g.setColor(col); g.setFont(fn); g.drawString(mesg, xpos, ypos);
  }
}
```

これはつまり「文字列を表示する」が、「マウスボタンが押されたらそこに文字の位置を動かし、色をピンクにする」し、「ドラグされたらそこに文字の位置を動かしつづけ、色を黄色に」し、「キーが押されたら色を赤にしつつそのキーの文字を文字列に追加 (ただし通常のキーの場合のみ)」し、そして「キーが離されたら色を緑に」する。

演習 2 java.awt.event パッケージにあるインタフェース MouseListener、MouseMotionListener、KeyListener、およびこれらに対応するアダプタクラス MouseAdapter、MouseMotionAdapter、KeyAdapter の API ドキュメントを眺めて使えそうなメソッド等をチェックせよ。

演習 3 上の例題をそのまま打ち込んで動かせ。ただし面倒だったらキーイベントの動作は打ち込みを省略してもよい (以下の演習では使わない)。

演習 4 さらに次のように改良してみよ。

a. マウスポインタがアプレット領域に入って来たときやそこから外へ出て行った時にも何か動作するようにしてみよ (色を変えるとかワープするとか)。
b. ドラグしたときマウスポインタについてくるのではなく「逆らう」(逆の方向に動くとか直角の方向に動くなど) ようにしてみよ。
c. ドラグすると、最初にボタンを押したところとドラグ中のマウスポインタの位置が対角線となる矩形 (長方形) が描けるようなプログラムに直してみよ。

3 オブジェクト指向プログラミングの活用

さて、前回クラス継承をやったので、それを活用してアニメーションのもっと構造化 (つまりプログラム上に色々な部品を用意してそれを組み合わせて自在にアニメーションを作る)、ということについて考えてみよう。

それと関連して 1 つだけ新しい内容を説明させて頂く。継承を使って子クラスで親クラスのメソッドを差し換えられるが、それを前提として親クラスを用意するときに「このメソッドは子クラスで必ず差し換えるから作らない」ということにしてもいいはずである。このような親クラスを「抽象クラス」(abstract class) とよぶ。抽象クラスは一部のメソッドがインタフェース (名前と引数の指定) だけなので、インスタンスは作れない。あくまでも、サブクラスを作る「土台」となることだけが目的のクラス、ということになる。

以下の例題でも何か所かに抽象クラスが出てくるのでそのつもりで。ではまず、アプレットクラスから見てみよう。ここでは AnimSet という新しいクラス (アニメーションの絵を沢山入れることができる) を用意して、そこに色々な絵をいれている、とだけ思って頂ければいい。あとは前回までにやったのと変わらない (後で説明する部分はコメントアウトしてある)。

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

public class R8Sample3 extends Applet implements Runnable {
  boolean running = false;
  double time = 0.0;
  AnimSet anim = new AnimSet(100);
  public void init() {
    anim.add(new WaveBackground(0, 250, 300, 100, 50,
                                new Color(190, 200, 100), 60, 20, 0.3, 0));
    anim.add(new WaveBackground(0, 260, 300, 100, 50,
                                new Color(200, 180, 100), 80, 10, 0.2, 0.5));
  //anim.add(new Circle(50, 50, 20, Color.red));
  //anim.add(new LinearMotion(10, 80, 10, 7, new Circle(0,0,20,Color.red)));
  //anim.add(new TimedStopScene(5, 20,
  //         new LinearMotion(10, 80, 10, 7, new Circle(0,0,20,Color.red))));
  //anim.add(new TimedVanishScene(0, 10,
  //           new LinearMotion(200, 100, -10, -4,
  //           new Bird(0,0,20,30,2,Color.blue))));
```

```
    //anim.add(new TimedAppearScene(10.02, 30,
    //           new LinearMotion(100, 50, 10, -2,
    //             new Bird(0,0,60,30,2,Color.green))));
    }
    public void start() { running = true; (new Thread(this)).start(); }
    public void stop() { running = false; }
    public void paint(Graphics g) { anim.draw(g); }
    public void addTime(double dt) { anim.addTime(dt); repaint(); }
    public void run() {
      long basetime = System.currentTimeMillis();
      while(running) {
        try { Thread.sleep(100); } catch(Exception e) { }
        long time = System.currentTimeMillis();
        addTime(0.001*(time-basetime)); basetime = time;
      }
    }
  }
```

インタフェース Animation も前回と同じ。

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

さて、AnimSet は要するに「複数の Animation を入れておいて、それらを順次 setTime()、draw() できる」という機能を持っている。

```
  class AnimSet implements Animation {
    Animation[] a;
    int count = 0;
    public AnimSet(int n) { a = new Animation[n]; }
    public void add(Animation x) {
      if(count < a.length) { a[count] = x; ++count; }
    }
    public void draw(Graphics g) {
      for(int i = 0; i < count; ++i) a[i].draw(g);
    }
    public void addTime(double dt) {
      for(int i = 0; i < count; ++i) a[i].addTime(dt);
    }
  }
```

では、入れるものの方へ進もう。まず、時間を累計していく機能をそれだけでクラスにする。このクラスは具体的な「形」はないのでインスタンスは生成できない (draw() が実装されていないので動かない)。このような「まだ十分具体的でないクラス」を「抽象クラス」といい、Java では abstract というキーワードを指定することになっている。

```
  abstract class TimedAnimation implements Animation {
    double time = 0.0;
    public void addTime(double dt) { time += dt; }
  }
```

では、これのサブクラスで「波の背景」を用意してみよう。これは多角形を使って、上端がサイン曲線になっている矩形領域を実現する。

```
class WaveBackground extends TimedAnimation {
  int xpos, ypos, width, height;
  int npoints;
  Color col;
  double length, amp, freq, phase;
  int xpts[], ypts[];
  public WaveBackground(int x, int y, int w, int h, int n,
                        Color c, double l, double a, double f, double p) {
    xpos = x; ypos = y; width = w; height = h; npoints = n;
    col = c; length = l; amp = a; freq = f; phase = p;
    xpts = new int[n+2]; ypts = new int[n+2];
    xpts[n] = x+w; ypts[n] = y+h; xpts[n+1] = x; ypts[n+1] = y+h;
  }
  public void draw(Graphics g) {
    for(int i = 0; i < npoints; ++i) {
      double x = xpos + i * (double)width/(npoints-1);
      double t = 2*Math.PI*(time*freq + (x-xpos)/length) + phase;
      double y = ypos - amp * Math.sin(t);
      xpts[i] = (int)x; ypts[i] = (int)y;
    }
    g.setColor(col); g.fillPolygon(xpts, ypts, npoints+2);
  }
}
```

これで「動く波の背景」ができた。さて、次に波は位置を変化させないが、座標を持ち、外部から位置を変化させることができるようなものをクラスとして用意しよう。これも形はまだないので抽象クラス。

```
abstract class PositionedAnimation extends TimedAnimation {
  double xpos, ypos;
  public PositionedAnimation(double x, double y) { xpos = x; ypos = y; }
  public void setPosition(double x, double y) { xpos = x; ypos = y; }
  public double getX() { return xpos; }
  public double getY() { return ypos; }
}
```

では、これをもとに「円」を作ってみる。

```
class Circle extends PositionedAnimation {
  double rad;
  Color col;
  public Circle(double x, double y, double r, Color c) {
    super(x, y); rad = r; col = c;
  }
  public void draw(Graphics g) {
    g.setColor(col);
    g.fillOval((int)(xpos-rad), (int)(ypos-rad), (int)(2*rad), (int)(2*rad));
  }
}
```

さて、なんでこんなことをしたのか? それは、PositionedAnimation の下にあるものなら何でも動かせるようなクラスを作るため。そのような「入れもの」クラスを作ってみよう。これも具体的な動きはないので抽象クラス。

```
abstract class MoveContainer extends PositionedAnimation {
  PositionedAnimation anim;
  public MoveContainer(double x, double y, PositionedAnimation a) {
    super(x, y); anim = a;
  }
  public void draw(Graphics g) { anim.draw(g); }
}
```

ではそのサブクラスで、直線状に動くというクラスを用意する。これにさっきの円を入れると「沈む夕日」とかができる。

```
class LinearMotion extends MoveContainer {
  double vx, vy;
  public LinearMotion(double x, double y, double vx0, double vy0,
                      PositionedAnimation a) {
    super(x, y, a); vx = vx0; vy = vy0; a.setPosition(x, y);
  }
  public void addTime(double dt) {
    super.addTime(dt); anim.addTime(dt);
    anim.setPosition(xpos+vx*time, ypos+vy*time);
  }
}
```

さて、ずーっと動き続けるのではつまらない。そこで、ある時刻になったら動き初めて、その後ある時刻になったら止まる、という入れ物を作ってみよう。

```
class TimedStopScene extends TimedAnimation {
  double time1, time2;
  Animation anim;
  public TimedStopScene(double t1, double t2, Animation a) {
    time1 = t1; time2 = t2; anim = a;
  }
  public void addTime(double dt) {
    super.addTime(dt);
    if(time1 <= time && time <= time2) anim.addTime(dt);
  }
  public void draw(Graphics g) { anim.draw(g); }
}
```

これをちょっと直して、ある時刻より後は描かないようにすると、止まる代りに「突然消える」ようにできる。

```
class TimedVanishScene extends TimedStopScene {
  public TimedVanishScene(double t1, double t2, Animation a) {
    super(t1, t2, a);
  }
  public void draw(Graphics g) {
    if(time <= time2) anim.draw(g);
```

```
    }
  }
```

もちろん、逆に「突然現れる」ようにしてもよい。

```
  class TimedAppearScene extends TimedStopScene {
    public TimedAppearScene(double t1, double t2, Animation a) {
      super(t1, t2, a);
    }
    public void draw(Graphics g) {
      if(time1 <= time) anim.draw(g);
    }
  }
```

では、鳥を飛ばそう。この鳥は時刻とともに形が変わる (羽ばたく) けど、そういうのをどうやって作るかは既にやったことがありますね?

```
  class Bird extends PositionedAnimation {
    double deg, len, freq;
    Color col;
    int[] x = new int[4];
    int[] y = new int[4];
    public Bird(double x, double y, double d, double l, double f, Color c) {
      super(x, y); deg = d; len = l; freq = f; col = c;
    }
    public void draw(Graphics g) {
      g.setColor(col);
      double d = 10 + 5*(Math.sin(time*freq));
      x[0] = (int)xpos;
      y[0] = (int)ypos;
      x[1] = (int)(xpos + 1.3*len*Math.cos(2*Math.PI*(deg-d)/180.0));
      y[1] = (int)(ypos + 1.3*len*Math.sin(2*Math.PI*(deg-d)/180.0));
      x[2] = (int)(xpos + len*Math.cos(2*Math.PI*(deg)/180.0));
      y[2] = (int)(ypos + len*Math.sin(2*Math.PI*(deg)/180.0));
      x[3] = (int)(xpos + 1.3*len*Math.cos(2*Math.PI*(deg+d)/180.0));
      y[3] = (int)(ypos + 1.3*len*Math.sin(2*Math.PI*(deg+d)/180.0));
      g.fillPolygon(x, y, 4);
    }
  }
```

このように、インタフェース、クラス (抽象クラス)、継承を使ってさまざまな「部品」のクラスを用意していくことで、場面転換のあるアニメーションを作ることができる。では、冬休みレポート頑張ってください。

4 さいごに

以上で 4 日間に渡っておつき合い頂いた「情報科学特殊講義」はおしまいである。Java のような新しい言語を使うことで、これまでやったことがないようなさまざまなプログラミングが可能になることがお分かり頂けたことと思う。その「新しいことができる」という能力はどこから来ているのだろうか? それは…

- 「オブジェクト」(もの) という考え方に基づいて考えることで、これまでのプログラミング言語よりも自然な形でプログラムの動作やプログラム内のデータの様子を把握/設計/制御できる。

- 「クラス」という「かたまり」を導入することで、プログラムのさまざまな部分を「部品」として扱えるようになり、より大きなプログラムでも混乱せずにちゃんと書けるようになる。
- 自分が作れないような機能でも「ライブラリクラス」という形で予め用意されているものを利用することができるので、それによってこれまでにない機能が利用できる。
- 「継承」を利用することで、既にあるクラスを土台としてそれにプラス α することが簡単にでき、さまざまなクラスのバリエーションが容易に作れるようになる。
- インタフェース (や継承) の機能を利用することで、さまざまな種類のものを「これらのどれでもよい」という形でまとめて取り扱うことができ、短いプログラムで豊富な機能が実現できる。

まだ他にもあるかも知れないが、とにかく「新しいプログラミング言語」は機能が増えて複雑になっている面もあるが、その増えた機能は最終的には「人間 (プログラマ) の能力をより引き出すことでさらに高度なソフトウェアを作成することを (その人にとって) 可能にする」ためのものである、ということを理解して頂けたら幸いである。

では、ご静聴ありがとうございました。