計算機プログラミングI 2005　久野クラス　＃8

久野　靖*

2005.12.9

はじめに

前回で絵が動かせるようになりましたが、結構大変でしたか。さて、今回はしばらく続いていた「お絵描き」をちょっと離れて「GUI 部品」をやります。GUI 部品が使えると、画面への入力が扱えるようになり、見た目ももっともらしい (実用っぽい) プログラムになると思いますから頑張りましょう。また、GUI 部品を使うにあたっては「インタフェース」「入れ子クラス」も使うので、これらについても復習しながら進めます。

その前に、前々回説明をパスした「`try ... catch`」の説明をする必要があります。ちょっと長いのでさっさとはじめましょう。

1　前回の演習問題の解説

1.1　演習 2

1 つのプログラムでまとめて示そう。円を 2 つにして `paint()` の中でも両方を描画する。あとは `run()` の中の動作を増やすだけ。円 1 は元の円と同様だが Y 方向にも 1 ずつ足して動かし、円 2 は sin/cos で軌道を計算し円運動させる。次に、最初のループが終わったら次のループでは 2 つの円を乱数に従って動かし、なおかつ円 2 は `Color.getHSBColor()` で徐々に変化する色をつくり出して設定する (このため `Circle` に `setPaint()` と `getPaint()` を追加した。後者は使わないけど両方あった方がいいと思うので)。

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

public class r7ex2 extends JApplet {
  Image buf;
  Circle c1 = new Circle(Color.blue, 0, 10, 20);
  Circle c2 = new Circle(Color.red, 150, 110, 15);
  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; c1.draw(g2); c2.draw(g2);
  }
  public void start() { (new Thread(new MyRun())).start(); }
  class MyRun implements Runnable {
    public void run() {
```

*筑波大学大学院経営システム科学専攻

```
      for(int i = 0; i < 100; ++i) {
        try { Thread.sleep(50); } catch(Exception e) { }
        c1.moveTo(c1.getX()+1, c1.getY()+1);
        double r = 50, t = Math.PI/50*i;
        c2.moveTo((int)(150+r*Math.cos(t)), (int)(100+r*Math.sin(t)));
        repaint();
      }
      for(int i = 0; i < 200; ++i) {
        try { Thread.sleep(50); } catch(Exception e) { }
        int dx = (int)(3*Math.random()-1.5), dy = (int)(3*Math.random()-1.5);
        c1.moveTo(c1.getX()+dx, c1.getY()+dy);
        c2.moveTo(c2.getX()+dx*2, c2.getY()+dy*2);
        c2.setPaint(Color.getHSBColor((0.01f*i)%1f, 1f, 1f));
        repaint();
      }
    }
  }
  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 draw(Graphics2D g) {
      g.setPaint(col);
      g.fillOval((int)(gx-rad),(int)(gy-rad),(int)rad*2,(int)rad*2);
    }
    public void moveTo(double x, double y) { gx = x; gy = y; }
    public double getX() { return gx; }
    public double getY() { return gy; }
    public void setPaint(Paint c) { col = c; }
    public Paint getPaint() { return col; }
  }
}
```

1.2 演習 3

こちらは次々に新しいクラスを追加していけばいいという感じ。まず冒頭部分を示す配列 a に作成したクラスのインスタンスを増やしているだけで、あとは前回の R7Sample2 と同じ。

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

public class r7ex3 extends JApplet {
  Image buf;
  boolean go;
  double time;
  Animation[] a = new Animation[20];
  int count = 0;
  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);
  a[count++] = new ResizingSquare(Color.green, 100, 100, 100);
  a[count++] = new GravityCircle(Color.pink, 100, 100, 30, 20, 50);
  a[count++] = new RoundingCircle(Color.black, 100, 100, 10, 50);
  a[count++] = new OscillatingOval(Color.yellow, 210, 40, 30);
  a[count++] = new RotatingNStar(Color.cyan, 150, 140, 5, 20, 40);
}
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();
    }
  }
}
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 > 300 && vx > 0) vx = -vx;
    if(gy < 0 && vy < 0 || gy > 200 && 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);
  }
}
static class ResizingSquare implements Animation {
  Paint col;
  double gx, gy, len, time;
  public ResizingSquare(Paint c, double x, double y, double l) {
    col = c; gx = x; gy = y; len = l; time = 0.0;
  }
  public void addTime(double dt) { time += dt; }
  public void draw(Graphics2D g) {
    int l = (int)(2.0*len + len*Math.sin(time)) / 4;
    g.setPaint(col);
    g.fillRect((int)(gx-l), (int)(gy-l), (int)(l*2), (int)(l*2));
  }
}
```

さて、ここからが新しいクラス。まず重力だが、これはヒントにあった通り「vy' = vy + α*dt」を増やすだけ (*1 のところ)。

```
static class GravityCircle implements Animation {
  Paint col;
  double gx, gy, rad, vx, vy;
  public GravityCircle(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;
    vy += 40*dt; // *1
    if(gx < 0 && vx < 0 || gx > 300 && vx > 0) vx = -vx;
    if(gy < 0 && vy < 0 || gy > 200 && 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);
  }
}
```

つぎに円軌道を周回するのは、軌道の半径と、累積時間をインスタンス変数に増やし、描画するときの座標を sin/cos で計算する (*2)。gx、gy はここでは「回転軌道の中心点」を表すように変わっていることに注意。

```
static class RoundingCircle implements Animation {
  Paint col;
  double gx, gy, rad1, rad2, time = 0.0;
  public RoundingCircle(Paint c, double x, double y, double a, double b) {
    col = c; gx = x; gy = y; rad1 = a; rad2 = b;
  }
  public void addTime(double dt) { time += dt; }
  public void draw(Graphics2D g) {
    int x = (int)(gx + rad2*Math.cos(time) - rad1); // *2
```

```
      int y = (int)(gy + rad2*Math.sin(time) - rad1); // *2
      g.setPaint(col);
      g.fillOval(x, y, (int)rad1*2, (int)rad1*2);
    }
  }
```

形が縦長と横長の間で振動する楕円は、やはり時間を累積し、描画するときの縦と横の半径をサイン関数で変化させている (*3)。

```
  static class OscillatingOval implements Animation {
    Paint col;
    double gx, gy, rad, time = 0.0;
    public OscillatingOval(Paint c, double x, double y, double r) {
      col = c; gx = x; gy = y; rad = r;
    }
    public void addTime(double dt) { time += dt; }
    public void draw(Graphics2D g) {
      int d = (int)(0.5 * rad * Math.cos(time));
      int r1 = (int)rad + d, r2 = (int)rad - d;
      g.setPaint(col);
      g.fillOval((int)(gx-r1), (int)(gy-r2), r1*2, r2*2);
    }
  }
```

回転する N 角星は、fillPolygon() で描画するための配列を用意し、描くときに sin/cos で頂点を決めて行く (1 つおきに長い半径と短い半径を使う)。そして sin/cos に渡す角度を時間とともに変化させるために累積時間 time を加えて計算する (*4)。

```
  static class RotatingNStar implements Animation {
    Paint col;
    double gx, gy, rad1, rad2, time = 0.0;
    int num;
    int[] xpts, ypts;
    public RotatingNStar(Paint c, double x, double y,
                         int n, double a, double b) {
      col = c; gx = x; gy = y; num = n; rad1 = a; rad2 = b;
      xpts = new int[2*n]; ypts = new int[2*n];
    }
    public void addTime(double dt) { time += dt; }
    public void draw(Graphics2D g) {
      double dt = Math.PI / num;
      for(int i = 0; i < 2*num; ++i) {
        double r = rad1; if(i%2 == 0) r = rad2;        // *4
        xpts[i] = (int)(gx + r * Math.cos(time+i*dt)); // *4
        ypts[i] = (int)(gy + r * Math.sin(time+i*dt)); // *4
      }
      g.setPaint(col);
      g.fillPolygon(xpts, ypts, 2*num);
    }
  }
}
```

2　エラーと例外

2.1　アプレットの虫取り

ずっと説明し損なっていましたが、アプレットの中でも `System.out.println(...)` は使えるので、図形の形などがおかしい場合は座標の値などを表示させてチェックできる。ただし、その出力は「Java コンソール」と呼ばれるウィンドウに出るのでこれを開いておくこと (やり方は久野クラスのページの説明を参照)。または、アプレットを appletviewer で動かせば出力は appletviewer を起動した窓に普通に表示される (文字コードが合っていないと文字化けで読めないけど)。

たとえば、次のアプレットを見てみよう。

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

public class R8Sample1 extends JApplet {
  int []x = new int[101];
  int []y = new int[101];
  public void init() {
    for(int i = 0; i <= 100; ++i) {
      double theta = 0.01 * 2 * Math.PI * i;
      x[i] = (int)(100*Math.cos(theta)) + 100;
      y[i] = (int)(100*Math.sin(2*theta)) + 100;
    }
  }
  public void paint(Graphics g) {
    Graphics2D g2 = (Graphics2D)g;
    g2.setStroke(new BasicStroke(3f));
    for(int i = 0; i <= 100; ++i) {
      g2.drawLine(x[i-1], y[i-1], x[i], y[i]);
    }
  }
}
```

このアプレットをブラウザで見ると、絵がちゃんと表示されない。そこで、Java コンソールを開いてみると図 1 のような表示が出ているはず。

これを見ると、最初はごちゃごちゃに見えるが、冒頭に「`ArrayIndexOutOfBoundsException`」とあるので、どうも「配列 (array) の添字 (index) が範囲 (bounds) を超えている (out)」らしいと読める。次の行を見ると `R8Sample1` というクラスの `paint` というメソッドの中で、ファイルでいうと `R8Sample1.java` というファイルの 16 行目だと出ている。つまり※の行ですね。そこでその近辺をよーく見ると、`i` が `0` のとき `x[i-1]` は確かに配列 `x` の `-1` 番目を参照してしまい、これが間違っているのだと分かる。正しくは「`for(int i = 1; …`」とすべきだったのですね。まあ、こう教科書通りには行かないにしても、エラーメッセージの見方に慣れておくとどこに間違いがあるか探す場所を絞ることができる。

2.2　例外

ところで、上にも出て来たように、さまざまな「エラー」が起きるとそのことを表すメッセージが Java コンソールなどに出てくる。実は、これらのエラーは Java では「例外」と呼ばれる機能によって統一的に扱われている。そして、特に指定しなければ例外が起きるとプログラムは終了し、その状況が表示されるので上のようなメッセージを目にするわけだ。

ここで、例外が発生したとき、それを受け止めて処理するようにプログラムを書いておけば、エラーが起きたときにプログラムがそこで止まってしまわないようにもできるので、その方法も学ぼう。

そのためには、例外を受け止めるための構文である try 文というものを使う。具体的には、try 文は

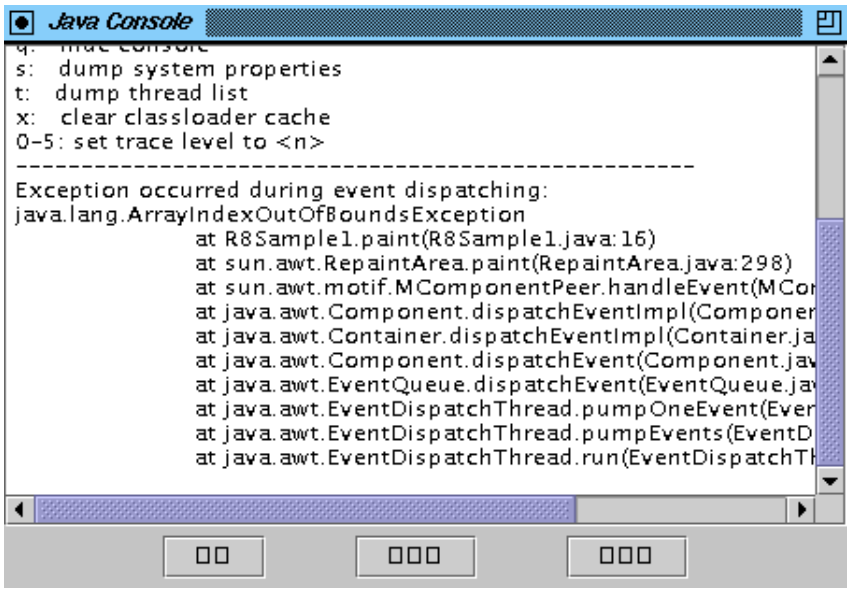

図 1: Java コンソールの表示

```
try {
  文 ... (1)
} catch(例外クラス名 変数) {
  文 ... (2)
}
```

という形をしていて、最初の (1) の部分にある文のならびの中で発生した例外を受け止めることができる。具体的には、例外が起きると実行は (2) の文のならびに「ジャンプ」して、この部分の文を最後まで実行したら try 文全体の処理が終わって次の文に進む。例外が起きなければ (1) の中の文は全部実行されるが、(2) の文は実行されないことになる。つまり (2) の部分はエラー処理専用の動作ということになる。

2.3 例外クラス

ここで「例外クラス」はエラーの種類を表すもので複数あるが、次のように階層構造に (継承を使って) 分類されている。

```
Throwable --- すべての例外やエラーの全体
  Error --- 回復不可能なエラー
    ClassFromatError --- .class ファイルが変である
    NoClassDefFoundError --- .class ファイルが見つからない
    ... その他いろいろ ...
  Exception --- 例外的なできごと全般
    RuntimeException --- 実行時エラー全般
      NumberFormatException --- 文字列が数値の形式になってない
      NullPointerException --- null 値に対してメソッドを呼ぼうとした
      IndexOutOfBoundsException --- 配列の添字範囲外をアクセスした
      ... その他いろいろ ...
    IOException --- 入出力エラー
    IntrruptedException --- 一時停止中に割り込みが起きた
    IllegalAccessException --- クラスの内容を不正にアクセスしようとした
    ... その他いろいろ ...
```

通常は受け止めて処理するのは Exception 以下なので、「例外クラス名」として Exception を指定することが多い。ただし、ある特定の例外だけ別に扱いたければ次のように try 文を入れ子にして使うことができる。

```
try {
  ... ※A
  try {
    ... ※B
    ...
  } catch(NumberFormatException e) {
    (1) 数値の形式が間違っていた場合の処理
  }
  ... ※C
} catch(Exception e) {
  (2) その他すべての例外の処理
}
```

ここで、※ B で NumberFormatException 例外が起きた場合には (1) の位置にある処理を実行するが、それ以外の例外は (2) の位置で処理する。※ A や※ C は内側の try 文の範囲外なので、すべての例外を (2) で処理する。[1]

2.4 例外を受け止めたところの処理

「例外を処理する」具体的な内容としては、次のようなものが考えられる。

- 何も動作を書かない — 単に例外が起きても無視して先の処理へ進みたい場合は何も動作を書かなくてもよい。
- 簡単なエラーメッセージを表示する。System.err.println(...) でメッセージを出力すればよい。なお、System.err.println() は System.out.println() とほぼ同様だが、エラー表示専用の出力ストリームになっている。
- 例外クラスすべてが共通に持っているメソッド `printStackTrace()` を呼び出して詳しいエラー状況を表示する。

実はこれまでエラーが自動的に表示される場合は 3 番目の `printStackTrace()` による表示が行われていた。

では、エラーを処理する簡単な例題を見てみよう。

```
import java.io.*;

public class R9Sample2 {
  public static void main(String[] args) {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    while(true) {
      try {
        System.out.print("x> ");
        String s = in.readLine();
        if(s.equals("")) break;
        int x = (new Integer(s)).intValue();
        System.out.print("y> ");
        int y = (new Integer(in.readLine())).intValue();
        System.out.println("x + y = " + (x+y));
      } catch(Exception e) {
        System.err.println("Oops! some error occured...");
        e.printStackTrace();
        System.err.println("Continue...");
      }
    }
```

[1] だったら Exception を受け止めれば全部受け止められるのだからそれだけ書けばいいとか思いますか? 無節操に全部受け止めるのは「どのような例外が出るかちゃんと考えていない」兆候なのであまりよくないと言えます。ただ、他人に提供するプログラムで「予定外のことが起きた場合でも動き続けたい」なら Exception を受け止めてそれをエラー表示するのはよいことだと考えます。受け止めて無視する、というのは避けてください。

```
    }
  }
```

この場合、たとえば入力に数字でないものを入れると、NumberFormatException が発生するが、それをループの内側で catch で受け止めているのでプログラムはエラーで終了せず、次のデータを入れることができる。動いている様子を見てみよう。

```
% java R8Sample2
x> 3
y> 5
x + y = 8     ← OK
x> 3
y> a          ←まずいデータ
Oops! some error occured...
java.lang.NumberFormatException: a
        at java.lang.Integer.parseInt(Integer.java:426)
        at java.lang.Integer.<init>(Integer.java:567)
        at R8Sample2.main(R8Sample2.java:13)
Continue...
x> b          ←まずいデータ
Oops! some error occured...
java.lang.NumberFormatException: b
        at java.lang.Integer.parseInt(Integer.java:426)
        at java.lang.Integer.<init>(Integer.java:567)
        at R8Sample2.main(R8Sample2.java:11)
Continue...
x> 1
y> 2
x + y = 3     ← OK
x>            ← [ret] でおしまい
%
```

このように、例外を自前で処理することで何かエラーがあっても終らずに続行することができる。

2.5 例外を投げる

ここまでは例外を受け止める話ばかりだったが、例外は自分で投げることもできる。たとえば処理していて正しくない事態に陥ったと思った時、そこで何かじたばたするより例外を投げてどこか別の場所でまとめて処理した方がきれいにできるのが普通なので、そのような時は自分で例外を投げれば良い。例外を投げるには、throw 文を使う。

```
throw 式;
```

ただし「式」は例外クラス (Throwable 以下) のオブジェクトを返すものでなければならない。とりあえずは次のような形で Exception オブジェクトを投げればよいだろう (自分でもっと適切なクラスを選んだり新規に作ってもよいが)。

```
throw new Exception("ほにゃららがまずい。");
```

また、先の catch 節での処理に次のものも追加しておこう。

- 一旦受け止めてエラーメッセージを出すが、さらに外側で処理してもらうため同じ例外を再度投げ直す。

この場合は次のような書き方になるだろう。

```
try {
  ...
} catch(Exception e) {
  System.err.println("....");
  throw e; // 投げなおす
}
```

2.6 throws 宣言

自分で投げた例外はこれまで通り catch で受け止めればよいが、自分のメソッド内では受け止めず、外側 (そのメソッドを呼び出した側) で受け止める場合にはメソッド定義の冒頭部分に

```
... メソッド名 (パラメタ…) throws 例外クラス名,... {
```

の形で宣言しておかなければならない。つまり「私はこれこれの例外を投げますよ」と予め明らかにしておかないと、そのメソッドを呼ぶ側で「心の準備」ができないので呼んでみたら知らない例外が投げられてびっくり、といったことになるからである。

実は、throws 節による宣言は、そのメソッドの中で throw で例外を投げる可能性がある場合だけでなく、そのメソッドの中で例外が発生するような操作を行って、なおかつその例外を catch しない場合にも必要である。というのは、catch しなかった例外はそのままメソッド呼び出し側に伝わって行くことになるから。

これでようやく、第 1 回のおまじないの謎がちょっとだけ解けることになる。つまり、

```
public static void main(String[] args) throws Exception { ...
```

というのは、この中で例外 (実は in.readLine() を呼ぶと IOException が投げられることがある) が発生してそれを捕まえないので外側にもそれが伝わりますよ、ということを断っているのだった。

ただし、この規則にも「例外」があって、Error と RuntimeException およびそれの子孫の例外については throws 節で断らなくてもよい。というのは、これらの例外はあらゆる場所で発生し得るため、いちいち断っていると大変すぎるからである。

あと、前回 `Thread.sleep()` を呼ぶところを try...catch で囲んでいたが、これはこのメソッドは InterruptedException(割り込み例外) を返すと宣言されているので、それを受け止めて処理しないとコンパイラに怒られてしまうからなのだった。

…大変長い説明ですいませんでしたが、一応きちんと説明するとこういうことになってしまうので。まあ、これからも try...catch が出たところでは簡単に復習して行きますから。

3 GUI と GUI 部品

GUI(Graphical User Interface) とは、最も広い意味でいえば「計算機のグラフィクス能力を活用したユーザインタフェース」ということになり、これには、ほとんど無限の多様性がある。しかし現実には GUI をもっと限定的に、「画面上にいろいろな『部品』(GUI 部品) が配置され、マウス等でそれを操作することで計算機とやりとりするようなインタフェース」程度の意味で捉えることが多い。部品の例としては「ボタン」「入力欄」「メニュー」などがある。

このようなスタイルの利点は、利用者にとってはさまざまなプログラムで使われている部品に共通性があり、その操作方法をいちいち覚えなくても済むこと、またソフトウェア作成者にとっては部品の実現部分は誰かが書いたものを持って来て再利用するだけで済んだり、さらに進んで部品を「見たまま方式」で直接配置しながらユーザインタフェースを設計/構築するツールが利用できたりして生産性が上がるということがある (しかしその半面、無批判に GUI 部品によるインタフェースばかり使うことは、人間の能力を殺しているのではないかという気もする)。

まあ疑問はさておき、Java で GUI 部品を使う方法について学ぼう。なお、この部分は JDK 1.1 と Java 2 で変化があったところだが、ここでは Java 2 で整備された「Swing」と呼ばれる部品群を扱う。これらの部品は `javax.swing` パッケージに含まれている (実は JApplet もその 1 つだった)。

どのような GUI 部品でも、画面上に配置し、色やフォントを指定して表示を行わせるというところは共通なので、そのためのメソッド (クラス Component に定義されている) の主なものを挙げておこう。

- setBounds(int, int, int, int) — 画面上の位置 (x,y) と幅と高さを設定
- setForeground(Color) — 前景色を設定
- setBackground(Color) — 背景色を設定
- setFont(Font) — フォントを設定

なお、部品の色はペンキではなくColor(単一の色) のみ指定できるので注意。では早速、「ラベルとボタンを 2 つ持ったアプレット」という例題を見ていただこう。

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

public class R8Sample3 extends JApplet {
  Font fn = new Font("Helvetica", Font.BOLD, 20);
  JTextField t1 = new JTextField("text...");
  JButton b1 = new JButton("B1");
  JButton b2 = new JButton("B2");
  public void init() {
    Container c = getContentPane(); c.setLayout(null);
    c.add(t1); t1.setForeground(Color.blue); t1.setBounds(20, 20, 200, 40);
    c.add(b1); b1.setFont(fn); b1.setBounds(20, 80, 60, 40);
    c.add(b2); b2.setFont(fn); b2.setBounds(120, 80, 60, 40);
  }
}
```

このアプレットは初期設定しかしない (あとの動作はGUI 部品が勝手にやってくれる) ので、メソッドinit() だけを持つ。アプレットが持っているメソッドgetContentPane() を呼ぶと、アプレット画面に入れる部品を配置するためのContainerオブジェクトが返され、これに対してメソッドadd() を呼ぶことで部品を追加できる。最初のsetLayout(null) は自動配置機能を off にしている (そうしないと場所指定が効かない)。

部品を追加したら、そのあと部品のメソッドを使って位置、フォントなどを設定している。この例題では部品としてテキスト入力欄 (文字列を入力する部品) とボタン (押す」ことができる部品) を取り上げた。これらを含め、たとえば次のような部品がある。

- JLabel — 文字を表示するだけの部品
- JButton — 押しボタン
- JCheckbox — チェックボックス
- JToggleButton — トグル形式のボタン
- JRadioButton — ラジオボタン
- ButtonGroup — (JRadioButton 用) ラジオボタンをグループ化する
- JComboBox — 選択メニュー
- JSlider — スライドレバー
- JSpinner — 複数の選択肢から値を選ぶ
- SpinnerNumberModel — (JSpinner 用) 数値の選択肢
- SpinnerListModel — (JSpinner 用) 個別値を列挙した選択肢
- JTextField — テキスト入力欄
- JTextArea — 複数行テキスト入力欄
- JList — 複数行の並んだスクロールリスト

演習 1 上の例題を打ち込んでそのまま動かせ。動いたら色やフォントや配置を調整してみよ。

演習 2 上の例題に出てこなかった部品を上の一覧から選び追加してみよ (API ドキュメントでメソッド等を調べられる)。

演習 3 次のような GUI プログラムのインタフェース部分だけを設計し (必ず紙にラフスケッチを描くこと)、その GUI 部分だけを Java で作ってみよ。動作本体は作らなくてよい。

a. 数を入力すると素数かどうか教えてくれる。

b. 華氏の温度を摂氏の温度に変換する。
c. 3 つの数値 (0〜255) を入力してボタンを押すとそれを RGB 値とする色が窓全体の背景色になる。
d. 簡単な電卓 (機能は適当に設計してよい)。
e. 入力も表示も 2 進数で行う簡単な電卓 (機能は適当に設計してよい)。
f. 数値を入力するとその数値までの素数一覧がスクロールリストに現れる。
g. 円や星型その他の図形のパラメタを入力してボタンを押すとそのパラメタに従った図形が現れる。
h. その他自分の好きなもの。

4 部品の動作を指定するには

これまでのところ、部品は「勝手に動作」するけれど、たとえばボタンを押してもそれ以上何も起きなかった (あたりまえだけど)。GUI 部品を役に立てるためには、「ボタンが押されたらこれこれの動作をする」といったコードが必要である。このような「ユーザの動作その他、外部的な要因」のことを「イベント」と呼ぶ。今回のボタン動作のイベントを受け止めるには次のような「アダプタクラス」を使う。

- アダプタクラスは「`implements ActionListener`」という指定を行う。つまり `ActionListener` インタフェースを実装するクラスである必要がある。
- アダプタクラスは「`public void actionPerformed(ActionEvent evt) { 動作… }`」というメソッドを用意して、そこに動作を書く。これがスレッドの時の `run()`、マウス入力のときの `mousePressed()` のように、「インタフェースで規定するメソッド」にあたる。
- GUI 部品は `addActionListener(…)` というメソッドを持ち、これを読んでアダプタを登録する。

では実際にこの方法で「ボタンが押されるごとにラベルに『*』が増える」というのを実現してみよう。

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

public class R8Sample4 extends JApplet {
  Font fn = new Font("Helvetica", Font.BOLD, 16);
  JLabel l1 = new JLabel("*");
  JButton b1 = new JButton("Press Me!");
  public void init() {
    Container c = getContentPane(); c.setLayout(null);
    c.add(l1); l1.setFont(fn); l1.setBounds(20, 20, 160, 40);
    c.add(b1); b1.setFont(fn); b1.setBounds(20, 80, 100, 40);
    b1.addActionListener(new MyAdapter());
  }
  class MyAdapter implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      l1.setText(l1.getText() + "*");
    }
  }
}
```

すなわち、`MyAdapter` というアダプタクラスは `implements ActionListener` と指定されていて、その中のメソッド `actionPerformed()` でボタンが押された時の動作を指定する。このクラスは「`static` のついていない入れ子クラス」だから、外側クラス (アプレットクラス) の変数 `l1` を参照できる、ということは以前やったが、思い出しておいて欲しい。

なお、上ではボタンが 1 つだったが、ボタンが複数あった場合はボタンごとに別の動作を指定したいだろうから、ボタン 1 つごとに別のアダプタクラスを作って `actionPerformed()` で動作を指定することになるだろう。

5　無名内部クラス

しかし、もっと別の方法で上のコードを簡単にすることができる。それは「無名内部クラス」と呼ばれる機能を使うことである。無名内部クラスは次の条件にあてはまる場合に利用できる。

1. 内部クラスを 1 箇所でしか利用しない (上の例でも `MyAdapter` という内部クラスは 1 箇所だけでしか利用していない)。
2. 内部クラスは `extends XXX` または `implements XXX` という指定を 1 つだけ持つ (上の例もそうなっている)。

この場合、これまでは次のように書いていた:

```
うんたら... new MyAdapter() ... かんたら


class MyAdapter extends/implements XXX {
  // クラス定義の中身
}
```

これを次のように「`new MyAdapter()`」の場所に「埋め込んで」しまう書き方が無名内部クラスである。

```
うんたら... new XXX() {
  // クラス定義の中身
} ... かんたら
```

こうすると、いちいちその内部クラスの名前を考えなくてすむ (だから「無名」内部クラス、と呼ばれる) し、記述も短くなる。見た目は慣れないと異様に感じるかも知れないが。これを利用すれば、上のコードは次のようにできる。

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

public class R8Sample5 extends Applet {
  Font fn = new Font("Helvetica", Font.BOLD, 16);
  JLabel l1 = new JLabel("*");
  JButton b1 = new JButton("Press Me!");
  public void init() {
    setLayout(null); l1.setBackground(Color.white);
    add(l1); l1.setFont(fn); l1.setBounds(20, 20, 160, 40);
    add(b1); b1.setFont(fn); b1.setBounds(20, 80, 100, 40);
    b1.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        l1.setText(l1.getText() + "*");
      }
    });
  }
}
```

6　例外を活用したエラー処理

ところで、`actionPerformed()` の中でさまざまな処理をしていてエラーが起きたときはどうすればいいだろう? アプレットがエラーで止まってしまうのは不親切なので避けたい。それには、先にやった try...catch で例外を受け止めて、どんな例外があったよ、ということだけとりあえずどこかに表示すれば、大変親切ではないにせよ、何とか許されるかと思う。そういう処理を入れた例を見てみよう。

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

public class R8Sample6 extends Applet {
  Font fn = new Font("Helvetica", Font.BOLD, 16);
  JTextField t1 = new JTextField("1");
  JLabel l1 = new JLabel("");
  JButton b1 = new JButton("+1");
  JButton b2 = new JButton("-1");
  public void init() {
    setLayout(null); l1.setBackground(Color.white);
    add(t1); t1.setFont(fn); t1.setBounds(20, 20, 160, 40);
    add(b1); b1.setFont(fn); b1.setBounds(20, 80, 60, 40);
    add(b2); b2.setFont(fn); b2.setBounds(100, 80, 60, 40);
    add(l1); l1.setBounds(20, 140, 300, 40);
    b1.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        try {
          t1.setText("" + (new Integer(t1.getText()).intValue() + 1));
        } catch(Exception ex) { l1.setText(ex.toString()); }
      }
    });
    b2.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        try {
          t1.setText("" + (new Integer(t1.getText()).intValue() - 1));
        } catch(Exception ex) { l1.setText(ex.toString()); }
      }
    });
  }
}
```

すなわち、例外が起きたらそれを受け止め、`toString()` で文字列に変換してラベル `l1` に `setText()` することで表示している。たとえば入力欄にある文字列が数値の形をしていないとちゃんとエラーが表示される。

演習 4 R8Sample4〜6 の例題から好きなものを 1 つ打ち込んで動かしてみよ。

演習 5 演習 3 で作ったプログラムに動作をつけてみよ。無名内部クラスを使うかどうかなどは好きに決めてよい。

演習 6 その他、自分で面白いと思う、オリジナルな動作つき GUI プログラムを作れ。

A　本日の課題 8A

「演習 2」または「演習 3」(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 `report8a.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 8A」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. GUI 部品とはどういうものか分かりましたか?
   Q2. GUI 部品を使った画面の設計方法はどうするのがいいと感じます?
   Q3. その他感想、要望等あればどうぞ。

B　次回までの課題 8B

次回までの課題は、「演習 5」または「演習 6」のアプレット (GUI 部品だけでなく、必ず動作がついていること) を 1 つ、作成することです。どのような動作をするか等は各自にまかせます。アプレットそのものはそれを表示するページの HTML ファイルを自分の `cp1` ディレクトリの下に `report8b.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、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 8B」とする。
2. 学籍番号、氏名、投稿日時を書く。
3. プログラムのソース。
4. その簡単な説明。
5. 下記のアンケートの回答。

   Q1. GUI 部品に動作をつける方法は分かりましたか? 動作をつける際に難しいところは何だったでしょうか?
   Q2. 実際に GUI 部品プログラムを設計したり、作ってみて、難しいところや勘どころはどこだと思いましたか?
   Q3. 感想と今後の要望をお書きください。

C　予告: 冬休み課題 9B

冬休みの課題のテーマは「絵を表示する」「GUI 部品を活用する」「アニメーションを行う」「マウスまたはキーイベントを受け取る」という 4 要素のうちから 2 つ以上を兼ね備えた、[2] 『楽しい』アプレットを作成することです。『楽しい』の定義は各自に任されますので、自分の腕前と相談して適切な水準のものを期待します。

レポートは紙のレポートになりますので、プログラムもさることながら、「レポートをきちんと書く」という点で努力をお願いします。レポートは A4 版の紙を縦づかいとし、必ず綴じること。さらに、次の内容構成とすること。

1. 表紙。表紙には表題「Report9B」、学籍番号、氏名、提出日付のみを書くこと。
2. 課題の再掲。この場合は「「絵を表示する」「GUI 部品を活用する」「アニメーションを行う」「マウスまたはキーイベントを受け取る」という 4 要素のうちから 2 つ以上を兼ね備えた、『楽しい』アプレットないしアプリケーションを作成する」ですね。
3. 方針の説明。自分がその課題をどのようにしてこなそうと計画したかの、方針を書く。
4. 回答。この場合は Java プログラムのプリントアウトと、その中の要点を説明したもの。
5. 考察。この課題をやってどんなことが分かったか、まだ疑問な点は何かを書く。
6. 感想。「おまけ」として、この科目/クラスを選択したことはどうだったか、よかったこと、悪かったことなど自由に記述してください。
7. 付録。たとえば表示の画面ハードコピーとか。[3]

アプレット本体はいつも通り `report9a.html` から見られるようにしておいてください。レポートの〆切は 1 月末日、提出場所は大演習室前のレポートボックスとなります。では頑張ってください。(なお、1/20 は講義休講ですが、TA さんに出勤してもらってプログラムの直し相談をやっていただきます。おおよそその時までにはプログラムが書けていないと、紙のレポートを書く時間が足りないと思います。)

[2] ということは、アニメーションはそれだけで絵が含まれるはずなので単独でよいが、絵単独や GUI 部品単独ではだめで、GUI 部品と絵かマウスイベントと絵を組み合わせる等の必要があるわけです。
[3] 窓のハードコピーをファイルに取るには「`gcopy`」コマンドを使ってください。