計算機プログラミングI 2005　久野クラス　＃6

久野　靖*

2005.11.18

はじめに

アプレット作りは皆様にお楽しみ頂けたようで、よかったです。ただ、前回までの方法で複雑な絵や動く絵をやるのは限界がありますし、そろそろ「クラスとは何か」ということをきちんと理解して頂きたいということもありますので、今回はクラスを作ってそれをもとに絵を描く、という考えを学んで頂きます。

1　前回の練習問題の回答例

1.1　演習 2

「Hello, World」をループで数回、場所と色を違えて書き、あと矩形と楕円を描いた。

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

public class r5ex2 extends JApplet {
  Font fn = new Font("Helvetica", Font.BOLD, 24);
  public void paint(Graphics g) {
    Graphics2D g2 = (Graphics2D)g;
    g2.setFont(fn);
    for(int i = 0; i < 10; ++i) {
      g2.setPaint(new Color(100+i*5, 0, 255-i*20));
      g2.drawString("Hello, World", 30+3*i, 30+7*i);
    }
    g2.setPaint(new Color(255, 0, 0));
    g2.fillRect(100, 100, 50, 50);
    g2.setPaint(new Color(0, 255, 0));
    g2.fillOval(150, 150, 100, 50);
  }
}
```

なお、Color.red とか Color.green などはクラス Color に用意されているクラス変数 (実際には書き換えられないので定数) で、その名前の色 (Color オブジェクト) が予め格納されている。その他にどんな色が用意されているかは API ドキュメント参照のこと。

1.2　補足: グラフィクスの座標系

前回の課題で「単位がわからない」という意見が出ていたが、Graphics クラスで扱っている描画機能では単位は基本的にすべて「ピクセル」(画面上の点の並び間隔)。そして、

*筑波大学大学院経営システム科学専攻

左上隅が原点 $(0,0)$ で、右へ行く程 **X** 座標が大きく、下へ行く程 **Y** 座標が大きくなる。

ちなみに、皆様が使っている iMac の画面はおおよそ横 1000 ピクセル、縦 800 ピクセルくらいだと思う。アプレット領域の幅と高さは HTML で指定している (例では幅 300 ×高さ 200 にしてありました)。

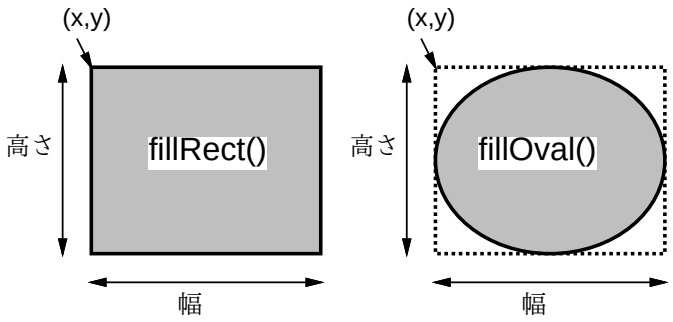

図 1: fillRect や fillOval の座標指定方法

1.3 補足: グラフィクスの基本的なメソッド

`Graphics` クラスのメソッドで矩形 (=長方形∪正方形) を描く `drawRect()`、`fillRect()` のパラメタは、API ドキュメントをちゃんと読めば分かるはずだが…

```
g.fillRect(x, y, 幅, 高さ);
```

で「左上隅の座標 (x,y)」と幅と高さを指定 (図 1 左)。また、円や楕円を描くメソッド `drawOval()`、`fillOval()` は上と同じに

```
g.fillOval(x, y, 幅, 高さ);
```

で矩形を指定して「その矩形に内接する円や楕円を描く」ことになっている (図 1 右)。

また、多角形を塗りつぶすメソッド `fillPolygon()` は配列を 2 つ渡す必要があり、

```
g.fillPolygon(a, b, n);
```

で頂点数 n の多角形を塗りつぶす。このとき、2 つの配列は n 個の整数を含んでいて、i 番目の頂点の座標が $(a[i], b[i])$ ということになる。

このように、メソッドにはすべて「私はこれをやります」という「契約事項」があって、その通りに呼べばその通りに仕事をしてくれる。だから、何が「契約事項」かをよく分からないままメソッドをあれこれ呼んでみるのは時間の無駄で、よく API ドキュメントを読んで頂きたい。

1.4 演習 3

別にどれをやってもいいので、とりあえず *sin* 曲線と渦巻きの 2 つを描いた。GeneralPath を使い、translate() と scale() を用いるサンプルから改造。

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

public class r5ex3 extends JApplet {
  public void paint(Graphics g) {
    Graphics2D g2 = (Graphics2D)g;
    GeneralPath p1 = new GeneralPath();
```

```
    for(int i = -15; i <= 15; ++i) {
      float x = i * 0.1f;
      float y = (float)Math.sin(3*x);
      if(i == -15) p1.moveTo(x, y); else p1.lineTo(x, y);
    }
    GeneralPath p2 = new GeneralPath(); p2.moveTo(0f, 0f);
    for(int i = 0; i < 100; ++i) {
      float r = 0.01f * i, theta = 0.2f * i;
      p2.lineTo(r*(float)Math.cos(theta), r*(float)Math.sin(theta));
    }
    g2.setStroke(new BasicStroke(0.03f));
    g2.translate(150f, 100f); g2.scale(100f, -100f);
    g2.setPaint(new Color(100, 0, 255)); g2.draw(p1);
    g2.setPaint(new Color(250, 100, 0)); g2.draw(p2);
  }
}
```

1.5 演習 4

問題文にあったように、乱数で 20 個程度の矩形の位置や色を決めるにしても、その決めた情報を保存しておかないと描き直しの際に同じ場所に同じ色で描くことができない。このため、X/Y 座標、幅、高さ、色を保存しておく配列が必要。

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

public class r5ex4 extends JApplet {
  int[] xpts = new int[20];
  int[] ypts = new int[20];
  int[] width = new int[20];
  int[] height = new int[20];
  Color[] cols = new Color[20];

  public void init() {
    for(int i = 0; i < 20; ++i) {
      xpts[i] = (int)(Math.random()*150);
      ypts[i] = (int)(Math.random()*250);
      width[i] = (int)(Math.random()*100+10);
      height[i] = (int)(Math.random()*100+10);
      cols[i] = new Color((int)(255*Math.random()),
                    (int)(255*Math.random()), (int)(255*Math.random()));
    }
  }
  public void paint(Graphics g) {
    Graphics2D g2 = (Graphics2D)g;
    for(int i = 0; i < 20; ++i) {
      g2.setPaint(cols[i]);
      g2.fillRect(xpts[i], ypts[i], width[i], height[i]);
    }
  }
}
```

これを単に paint() の中で乱数で決めるだけのバージョンと比較してみると面白い。

2 自前のクラスを作る

ここまでで繰り返し「既にあるクラス」(API ドキュメントで調べられる) は使って来たが、依然として「クラスとは何か納得できない」人が多いと思う。そこで今回は、「自分で新たにクラスを作る」ことでこの問題にアプローチしてみよう。

何回も説明してきたが、クラスとは、ある機能を持った「もの」の「種類」に対応している。今回は分かりやすさのため、「もの」として「図形」を取り上げる (だから「種類」は「円」とか「矩形」になる)。つまり、いくつかの種類の図形 (を描く機能) に対応したクラスを作ってみよう。一番簡単な例として、矩形 (長方形) を描くためのクラスを考える。このクラスの名前を仮に Rect と決めたとして、その使い方は次のようにする。

```
Rect r1 = new Rect(ペンキ, X, Y, 幅, 高さ); // オブジェクトを作る
...
r1.draw(Graphics2D オブジェクト); // ペンを与えて描く
```

まず、矩形オブジェクトを作る時はペンキ (とりあえず色) と座標、幅、高さを指定するものとする。この時、座標は「矩形の重心」を指定することにします (その方が使いやすい)。この、「new XXX(...)」という指定で呼び出す機能を「コンストラクタ」(オブジェクトを「作るもの」という意味) と言う。そしてもう 1 つの機能はインスタンスメソッド draw() で、その矩形を表示する (描く) ものとする。インスタンスメソッドとは、そのオブジェクト (この場合は具体的な矩形) に付随するメソッドだったことを思い出そう。だから、draw() を呼ぶと「その」矩形が、「その」固有の色、位置、大きさで描かれる。このメソッドに対して指定する Graphics2D オブジェクトは、アプレットのメソッド paint() に渡されて来るものを型変換してそれを渡すようにする。

では例題プログラムを見てみる。今回はアプレットのクラスに加えて同じファイルに Rect クラスも入れることにする。このような補助クラスは public 指定はつけない。

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

public class R6Sample1 extends JApplet {
  Rect r1 = new Rect(Color.red, 100, 100, 80, 60);
  Rect r2 = new Rect(new Color(0.5f, 1f, 0f, 0.7f), 150, 120, 60, 90);
  public void paint(Graphics g) {
    Graphics2D g2 = (Graphics2D)g;
    r1.draw(g2); r2.draw(g2);
  }
}
class Rect {
  Paint pat;
  int xpos, ypos, width, height;
  public Rect(Paint p, int x, int y, int w, int h) {
    pat = p; xpos = x; ypos = y; width = w; height = h;
  }
  public void draw(Graphics2D g) {
    g.setPaint(pat);
    g.fillRect(xpos-width/2, ypos-height/2, width, height);
  }
}
```

アプレットのクラスでは、2 つの矩形 (Rect オブジェクト) を用意し、paint() の中でそれらを表示させる。例によって、別のオブジェクトの機能を呼び出すだけにしたので、この部分は非常に簡単になっている。ただし、色の作り方として

「決まった名前の色」と「RGB に加えて透明度を指定する方法」とを使ってみている。このあたりは API ドキュメントをチェックしておくこと。

次に Rect クラスの定義がある。Rect クラスのインスタンス (オブジェクト) は、矩形を表すわけなので、そのための情報一式、つまり色、X/Y 座標、幅、高さを変数 (インスタンス変数) として保持する。次にコンストラクタを定義するが、コンストラクタはクラス名と同じ名前で、型指定を持たないメソッド (の形) として定義する。「public Rect(...)」の「...」の部分には引数指定、つまり「new Rect(...)」で呼び出した時にパラメタとして指定する値を受け取るための変数を書く。コンストラクタの役割りはオブジェクトの初期設定であり、この例のようにインスタンス変数一式に適切な値を入れる形になることが多い。最後にメソッド draw() の定義があるが、これは Graphics オブジェクトを引数 g として受け取り、まず setColor() で色を設定し、次に fillRect() で矩形の塗りつぶしを行う (これらは前回と同様、Graphics オブジェクトのメソッドを呼んでいる)。

しかし長くなっただけでちっとも分かりやすくない? そうだろうか? 確かに Rect クラスを作るところは長くなったが、その分だけアプレットのクラスは簡単になっている。そして、もっと沢山矩形を使う場合を考えると、Rect クラスはもう作ってしまったからいじらないでいいので、アプレット側が簡単になることは全体として非常に有利になるわけである。なお、アプレットを動かすには HTML ファイルもいるので、一応もう 1 回だけ掲載しておこう。

```
<html>
<head><title>sample</title></head>
<body>
<h1>sample</h1>
<applet code="R6Sample1.class" width=300 height=200></applet>
</body>
</html>
```

演習 1 上の例題を打ち込んでそのまま動かせ。うまく行ったら描く矩形を 1 つ増やして 3 つにしてみよ。

演習 2 演習 1 の例題の末尾に次のような図形のクラスを増やし、アプレット側でもそれを画面に描くように指定を追加せよ。

a. 円を表すクラス Circle。その機能は次の 2 つとする。

```
public Circle(Paint p, int x, int y, int r) --- コンストラクタ
public void draw(Graphics2D g) --- 画面に描く
```

なお、コンストラクタではペンキ、中心の X/Y 座標、半径を指定する。

b. 三角形を表すクラス Triangle。その機能は次の 2 つとする。

```
public Triangle(Paint p, int x0, int y0, int x1, int y1,
                int x2, int y2) --- コンストラクタ
public void draw(Graphics2D g) --- 画面に描く
```

なお、コンストラクタではペンキと 3 頂点の座標を指定する。

c. 図 1 のような模様の旗を表すクラス Flag。その機能は次の 2 つとする。

```
public Flag(Paint p1, Paint p2, int x, int y, int r) --- コンストラクタ
public void draw(Graphics2D g) --- 画面に描く
```

コンストラクタでは地 (長方形) のペンキ、中心の円のペンキ、円の中心の X/Y 座標と半径を指定する。地 (長方形) の位置や大きさは円の位置と半径から計算できますね?

3 Java の文法

プログラムの正確な書き方は、その言語の「文法」によって定められている。ここまででクラスとメソッドについて一応学んだので、文法のまとめを示しておこう。なお「[...]」は「あってもなくてもよい」、「|」は「または」、…は「ならんだもの」、,…は「カンマで区切って並んだもの」を表す。こうして見ると、前々回までのスタンドアロンアプリケーションと前回からのクラスとでは一見大きく違うようだが、「クラスが 1 つ以上」という構造は共通していることが分かる。

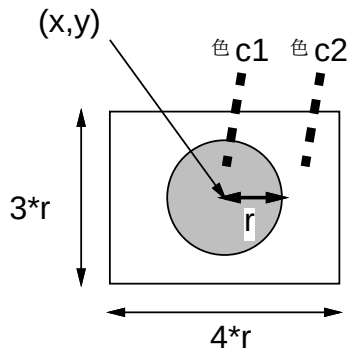

図 2: 旗の指定方法

```
プログラム ::= クラス…
クラス ::= [public] class クラス名 [extends クラス名] { ( 宣言 [= 式]; )… メソッド… }
メソッド::= 修飾… 型指定 メソッド名 (宣言,…) [throws クラス名] { 文… }
修飾 ::= public | static | final
型指定 ::= 型名 | 型名 [] | void
型名 ::= クラス名 | boolean | byte | char
       | int | long | float | double
宣言 ::= 型指定 変数名
文 ::= 宣言 [= 式]; | 式;
       | if(式) 文 [else 文] | while(式) 文
       | for([宣言 =] 式; 式; 式) 文
       | { 文…} | break; | continue;
式 ::= 式 演算子 式 | 演算子 式 | 式 [式]
       | new クラス名 (式,…) | new 型名 [式]
       | クラス名. メソッド名 (式,…) | 式. メソッド名 (式,…)
       | リテラル | 変数名 | クラス名. 変数名 | 式. 変数名 | ( 式 )
リテラル::= 整定数 [l] | 実定数 [f] | "..." | '...'
       | true | false | null
```

一番多くお世話になるのは値を計算したりメソッドを呼び出したりする構文である「式」で、「式;」として 1 つの文にすることもできる (単独メソッド呼び出しはこれに当たる。「`a = b;`」の「`=`」も代入**演算子**を使った式にすぎない。

式には演算子の適用、配列参照 (添字)、メソッド呼び出し、リテラル (定数)、そして new によるオブジェクト生成がある。new の中でも配列生成の new は構文が特別で、既に学んだ要素数を指定する方法のほかに、個々の要素の初期値を指定して配列を作る方法もある。

リテラル (定数) には論理値 (`true`、`false`)、文字列、文字、数値、そして「どのオブジェクトも参照していない」ことを表す特別な値 `null` がある。整数リテラルは「`l`」がついたのは `long`、そうでないのは `int`。実数リテラルは「`f`」がついたのは `float`、そうでないのは `double`。

4 複数のオブジェクトを組み合わせる

先の「旗」の演習問題について考えてみよう。旗を描くには、まず矩形を描いて、その後で円を描けばよい。もちろん、それぞれを「`g.fillRect(...)`」や「`g.fillOval(...)`」で順次描いてもいいのだが、方法としてはそれだけだろうか? もともと「矩形」や「円」という『もの』はクラスとして定義ずみなのだから、それらのクラスを「利用して」作る方が分かりやすいし素直だし楽なのではないだろうか?

もう 1 つ別の例として、「ハロウィンの顔」を描いてみよう (最近まで知らなかったのですが、あのカボチャをくりぬいたランタンは Jack-o-lantern と言うのだそうです。知ってました?)。

```
import java.applet.Applet;
```

```
import java.awt.*;

public class R6Sample2 extends Applet {
  Jack j1 = new Jack(new Color(80, 120, 200), new Color(40, 80, 0, 200),
                     100, 100, 60);
  Jack j2 = new Jack(new Color(200,180,50,200),new Color(240, 200, 90),
                     180, 120, 50);
  public void paint(Graphics g) {
    Graphics2D g2 = (Graphics2D)g;
    j1.draw(g2); j2.draw(g2);
  }
}

class Jack {
  Circle cir;
  Triangle eyeL, eyeR, mouth;
  public Jack(Paint p1, Paint p2, int x, int y, int r) {
    int u = r / 4;
    cir = new Circle(p1, x, y, r);
    eyeL = new Triangle(p2, x-3*u, y-u, x-u, y-u, x-2*u, y-2*u);
    eyeR = new Triangle(p2, x+3*u, y-u, x+u, y-u, x+2*u, y-2*u);
    mouth = new Triangle(p2, x-u, y+2*u, x+u, y+2*u, x, y+3*u);
  }
  public void draw(Graphics2D g) {
    cir.draw(g); eyeL.draw(g); eyeR.draw(g); mouth.draw(g);
  }
}
class Circle {
  Paint pat;
  int xpos, ypos, rad;
  public Circle(Paint p, int x, int y, int r) {
    pat = p; xpos = x; ypos = y; rad = r;
  }
  public void draw(Graphics2D g) {
    g.setPaint(pat); g.fillOval(xpos-rad, ypos-rad, 2*rad, 2*rad);
  }
}
class Triangle {
  Paint pat;
  int[] xpts, ypts;
  public Triangle(Paint p, int x0, int y0, int x1, int y1, int x2, int  y2) {
    pat = p; xpts = new int[]{x0,x1,x2}; ypts = new int[]{y0,y1,y2};
  }
  public void draw(Graphics2D g) {
    g.setPaint(pat); g.fillPolygon(xpts, ypts, 3);
  }
}
```

クラス Jack は丸い輪郭と三角形の両目、口を持つ顔を表しているが、その中では 1 つの円、3 つの三角形のオブジェクトを保持しているだけで、描画もこれらの「下請け」に任せている。このように、込み入った構造を持つものでもその

部分を下請けに任せてしまえれば簡単に用意できることがお分かりいただけると思う。

あと、「new int[]{ ○, ○, ...} 」というのは配列を割り当てるとともに各要素の初期値を設定する「配列リテラル」で、今回のように配列を用意してすぐに値も入れる場合には便利だと思うので使ってみた。

演習 3 円、矩形、三角形を適宜組み合わせた簡単な「絵」をデザインし、その絵を表すようなクラスを作ってアプレットで描画してみよ。

注意!: 以下に出て来る高度な機能はマスターしなくてもレポートをやる上では差し支えない。ただ、ある種の絵が描きたい人は知っておかないと不便だと思うので説明しておくということで。

5 画像の扱い

ところで、ここまでで描いて来た「絵」はなんか図形みたいで写真や絵とは違うと思ったかも知れません。それはそうなので、これまでのところ、画面に描くための機能として線など幾何学的なものしか学んでいないので。

そこで次は、絵や写真などのような「画像 (Image)」を扱う方法を説明しておきましょう。画像というのは、要するに「縦横にさまざまな色の点がぎっしり並んだもの」と考えればよい。

それで、Java での画像の扱いだが、どういうわけか歴史的にさまざまな機能がつぎはぎに用意されたため、どんなクラスを何のために使えばよいのか非常に分かりにくい状況になっている。ここではとりあえず、Java 2 になってから用意された新しいクラスだが、 一番分かりやすいと思う BufferedImage クラスを使うことだけ取り上げる。BufferedImage クラスは java.awt.image パッケージに含まれているので、import 文を指定するのを忘れないこと。

BufferedImage クラスのコンストラクタは 3 つあるが、一番良く使うのは次の形のもの。

```
new BufferedImage(幅, 高さ, 種別)
```

ここで「種別」としてさまざまなものが指定できるが、今回は決め打ちで「TYPE_INT_ARGB」という種別 (この定数も BufferedImage クラスで定義されている) を指定する。メソッドについては API ドキュメントを見ていただきたいが、とりあえず次の 2 つを知っておけば色々できる。

- getGraphics() — その画像に描くためのペンを取得。ペンを取得した後はこれまで画面に描いたのと同様にして画像にいろいろなものを描くことができる。
- setRGB(x, y, p) — 画像の点 (x, y) の色を p にする。TYPE_INT_ARGB の画像であれば、p は 32 ビットの整数で上から 8 ビットずつ透明度、赤、緑、青の値 (当然 0〜255 の範囲) を指定する。簡単には、Color オブジェクトのメソッド getRGB() で取り出した整数値を入れる方法が書きやすい。

あと、用意した画像を画面に描くときは、Graphics クラスのインスタンスメソッド drawImage() を次のようにして使う。

- g.drawImage(img, x, y, null) — 位置 (x, y) に画像を描く。
- g.drawImage(img, x, y, w, h, null) — 位置 (x, y) に幅を w、高さを h に伸縮した形で画像を描く。

2 番目の方法を使えば、小さい画像を大きく表示することもできる。なお、最後のパラメタは歴史的事情により指定するもので、BufferedImage 以外の種別の画像では適宜オブジェクトを指定する必要があるが、ここでは常に null でよい。

では簡単なサンプルを見ていただこう。

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

public class R6Sample3 extends JApplet {
  BufferedImage im = new BufferedImage(40, 40, BufferedImage.TYPE_INT_ARGB);
  public void init() {
    for(int x = 0; x < 40; ++x) {
      for(int y = 0; y < 40; ++y) {
        int r = (int)(256*Math.random());
```

```
        int g = (int)(256*Math.random());
        int b = (int)(256*Math.random());
        im.setRGB(x, y, new Color(r, g, b, 150).getRGB());
      }
    }
    Graphics2D g2 = (Graphics2D)im.getGraphics();
    g2.drawLine(0, 40, 40, 0);
  }
  public void paint(Graphics g) {
    Graphics2D g2 = (Graphics2D)g;
    g2.drawImage(im, 10, 10, null);
    g2.drawImage(im, 30, 30, 200, 200, null);
  }
}
```

このアプレットでは、40x40 サイズの画像を用意し、メソッド `init()` の中でその各点にランダムな色を半分くらいの透明度で割り当てる。その後、この画像に描くためのペンを用意し、斜めに線を 1 本だけ引く。`paint()` ではこの画像をそのままのサイズと 5 倍したサイズで画面に描く。これを見ると、確かに半透明なランダム画像に斜め線が引けているのが分かる。

演習 4 この例題を打ち込んでそのまま動かせ。動いたら何でも好きな画像を作って複数回表示するように直してみよ。

6 ペンキの謎

しかしここまでで、何となく釈然としないことが色々あったのではないだろうか? たとえば、クラスを作る側では `Paint` を受け取ることにしているのに、利用する側では `Color` を渡している。そんな違うものを渡してもいいのだろうか?

実は API ドキュメントを見れば分かるように、`Paint` というのはクラスではなく**インタフェース**と呼ばれるものである。インタフェースとは簡単に言えば「切り口」であり、ペンキとして使える「切り口」を持つクラスならどれでも `Paint` 型として扱うことができる。そして、そのようなクラス (API ドキュメントでは「実装している」「implements している」という言い方をする) の 1 つとして、`Color` クラスがあるわけだ。つまり、均一の色で塗るというのもペンキの 1 種には違いない。

では、それ以外の `Paint` を実装するクラスにはどんなものがあるのだろうか。それは次の 2 つがある。

- `GradientPaint` — グラデーション (階調変化) のあるペンキ。
- `TexturePaint` — 画像を模様として持つペンキ。

前者については、2 つの点とそれぞれの点での色を指定すると、その間の範囲で色が連続的に変化するようになる。そのコンストラクタの呼び出し方は次の通り (座標の指定には `float` 型を使う)。

`new GradientPaint(`x_1`,` y_1`,` c_1`,` x_2`,` y_2`,` c_2`)`

さらに最後に `true` を追加すると、上記のパターンが反復した縞模様になる。

`new GradientPaint(`x_1`,` y_1`,` c_1`,` x_2`,` y_2`,` c_2`, true)`

後者については、画像と矩形領域を指定することで、画像を矩形領域の位置と形に合わせてぎっしり敷き詰める (図 3)。

矩形領域 $(x,y),w,h$ を指定するのに `Rectangle2D.Float` を使うのでちょっと API ドキュメントを読んでも分かりづらいが、次のようにすればよい。

`new TexturePaint(`img`, new Rectangle2D.Float(`x`,`y`,`w`,`h`))`

ではこれらを使った例を示しておく。

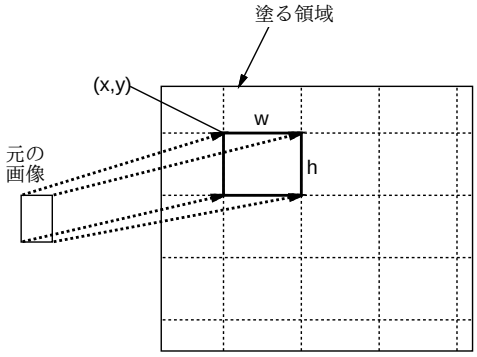

図 3: TexturePaint の指定方法

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

public class R6Sample3 extends JApplet {
  Rect r0, r1, r2;
  public void init() {
    r0 = new Rect(Color.green, 80, 80, 100, 40);
    BufferedImage im =
      new BufferedImage(20, 20, BufferedImage.TYPE_INT_ARGB);
    Graphics2D g2 = (Graphics2D)im.getGraphics();
    g2.setPaint(new Color(1f, 0f, 0f, 0.7f)); g2.fillOval(0, 0, 20, 20);
    for(int x = 0; x < 20; ++x)
      for(int y = 0; y < x; ++y) im.setRGB(x, y, 0x00FFFFFF);
    TexturePaint tp = new TexturePaint(im,
                        new Rectangle2D.Float(0, 0, 10, 20));
    r1 = new Rect(tp, 100, 100, 80, 160);
    GradientPaint gp = new GradientPaint(90f, 90f, new Color(1f,1f,1f,0.1f),
                                         180f, 180f, Color.blue);
    r2 = new Rect(gp, 150, 120, 100, 90);
  }
  public void paint(Graphics g) {
    Graphics2D g2 = (Graphics2D)g;
    r0.draw(g2); r1.draw(g2); r2.draw(g2);
 }
}
class Rect {
  Paint pat;
  int xpos, ypos, width, height;
  public Rect(Paint p, int x, int y, int w, int h) {
    pat = p; xpos = x; ypos = y; width = w; height = h;
  }
  public void draw(Graphics2D g) {
    g.setPaint(pat);
    g.fillRect(xpos-width/2, ypos-height/2, width, height);
  }
```

```
}
```

ここで矩形 r0 はただの緑色のペンキ、r1 は赤丸を描いたあと対角線の上半分を透明に消した画像を縦長に伸ばして敷き詰めた模様のペンキ、r2 はほぼ透明な白から不透明の青に変化するグラデーションのペンキを指定している。

演習 5 上の例題を打ち込んで動かせ。動いたら自分なりのペンキを作ってそれで矩形や他の図形を塗ってみよ。

演習 6 自分で作ったクラス (群) のインスタンスを組み合わせて「美しい」または「かっこいい」絵を描くアプレットを作成せよ。自分の腕前に照らして不当に易しくないこと。「美しさ」「かっこよさ」の基準については各自の判断に任せる。

A　本日の課題 6A

「演習 2」または「演習 3」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 report6a.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 6A」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. クラスが作れるようになりましたか?
   Q2. クラスを組み合わせて絵を作る手法についてはどう思いますか?
   Q3. その他感想、要望等あればどうぞ。

B　次回までの課題 6B

次回までの課題は、「演習 6」をお願いします。作成したアプレットのための HTML ファイルを、自分の cp1 ディレクトリの下に report6b.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。期限は次回授業開始の 10 分前 (10:30 まで) です。具体的な内容は次の通り。

1. Subject: は「Report 6B」とする。
2. 学籍番号、氏名、投稿日時を書く。
3. 選んだプログラムのソース。
4. その説明。どのような点を工夫したか書くこと。
5. 下記のアンケートの回答。

   Q1. 自分で美しい/かっこいい絵をデザインし製作しようとして、どういうところに苦心しましたか?
   Q2. クラスを作りはじめると「疑似コード」とはお別れという感じになったと思いますが、それについてはどう思いますか?
   Q3. その他感想、要望等あればどうぞ。