Java による画像処理

（画像をピクセルレベルで操作したい人のために）

これは、ある程度 Java が分かっているが、画像処理は“あまり”やったことのない人を対象にしている。本格的プログラムを意識し、アプレットではなくアプリケーションにしている。話を分かりやすくするため Swing は使っていない。また、Java2D も最小限にとどめている。

第１章　　Image タイプか BufferedImage タイプか？

Java で画像処理をしようとすると、先ず、Image タイプを使ったらよいのか BufferedeImage を使ったらよいのか悩む。ここでは、２つの違いを簡単に紹介する。

（１）Image タイプ：

画像を読み込み、表示する位なら Image タイプの方が簡単でよい。ピクセルレベルで操作するのも pixelgrabber で配列に落とし操作するので直感的で分かりやすい。
Image タイプの変数をファイルに落とそうとすると直接保存できないようで一旦 BufferedImage を経由する必要がある。

（注意）画像ファイルを toolkit の getImage で読込むと、読込み修了前に処理が先に進んでしまうので、MediaTracker でファイル入力処理の修了を待つ必要がある。

（２）BufferedImage タイプ：

画像をピクセルレベルで操作する場合、ファイル入力からファイル出力まで一貫して扱えるのでこのタイプを用いる方がよい。ただし、このタイプでピクセルを扱う場合、BufferedImage の上に DataBuffer や Raster といったタイプがかぶさっているため、これを理解するのが少し面倒。

BufferedImage でのピクセルの操作は次の 3 レベルが存在する。

(1) Image の場合と同様 PixelGrabber が使える。
(2) BufferedImage レベル（低レベルで、ピクセルへのアクセスは getRGB、setRGB を使う）
(3) DataBuffer レベル（中レベルで、ピクセルへのアクセスは getElem、setElem を使う）
(4) Raster レベル（高レベルで、ピクセルへのアクセスは setPixel を使う、getPixel は”ない”)

どれを使ってもかまわない。

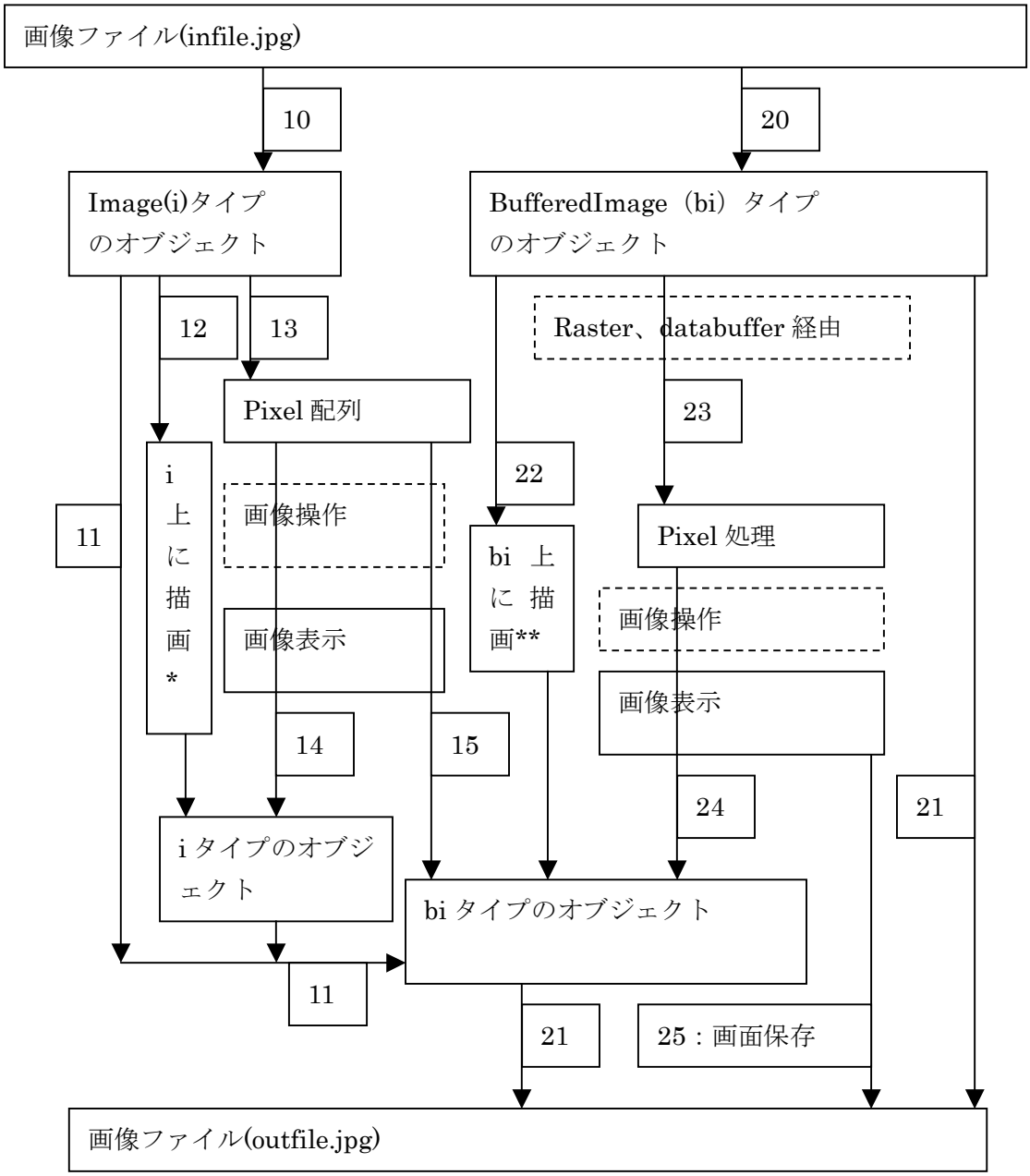


図１　Java による画像処理

第２章　　Image タイプの処理

図１がJavaによる画像処理の全体像である。図の中の矢印には２桁の数値が付してある。サンプルプログラムは Im の後に数字が並んでいるが、この数字は図中の数字に対応している。例えば Im2024 は 20 と 24 の処理を含んでいることを示す。

(注意：以下のプログラムでは入力画像ファイル名は infile.jpg（２０ｘ２０ピクセル）とし、出力画像ファイルは outfile.jpg に統一している)

2.1　グラフィクスの基本原理　　(Im12.java)

このプログラムには Java の画像処理の基本的考え方が含まれているので、最初に用意した。特に、画像処理における“紙”と“筆”を意識して読んで欲しい。

アニメーションでチラツキをなくすためにダブルバッファリングという技がある。
これは、オフスクリーンというモニタに直接表示されないスクリーン（Image）に絵を描いておいて、全て描き終わった時にモニタ画面にオフスクリーンの画像を描き出す方法である。
ダブルバッファリングは次のステップで処理が進む。

step1:ダブルバッファリング用三種の神器（紙、筆、サイズ）を宣言
step2:setVisible(true)で Frame を表示
step3:createImage でオフスクリーン紙を作る
step4:offscreen.getGraphics で、gO という offscrean 専用筆を用意
step5:オフスクリーンに長方形、文字列、線、を描画
step6:絵を描き終わったら Frame 専用筆 g で（drawImage で）一気にフレーム（モニタ画面）に描画

ちらつき防止に update を使用。

＝＝

```
import java.awt.Frame;
import java.awt.Graphics;
import java.awt.Image;
import java.awt.Dimension;
import java.awt.Color;

public class Im12 extends Frame{
        /*  ダブルバッファリング用三種の神器  */
        Image offscreen;//紙
```

```
        Graphics gO;//筆
        Dimension d;//紙の大きさ

        public void init(){
                /* オフスクリーン・バッファの用意 */
                d = getSize();
                offscreen = createImage(d.width,d.height);
                gO = offscreen.getGraphics();

                /* 絵を描いておく */
                gO.setColor(Color.white);
                gO.fillRect(0,0,d.width,d.height);
                gO.setColor(Color.black);
                gO.drawString("ダブル・バッファリング",10,120);
                gO.setColor(Color.blue);

                for(int i=0;i<100;i++)
                        gO.drawLine(0,i,100,i);
                /* 絵が出来たので、表示 */
                repaint();
        }
        public void update(Graphics g){
                paint(g);
        }
    public void paint(Graphics g){
                if(offscreen != null){
                        g.drawImage(offscreen,0,0,this);
                }
        }
        public static void main(String args[]){
                Im12 dbf = new Im12();
                dbf.setSize(150,150);
                dbf.setVisible(true);
                dbf.init();
        }
}
==
```

2.2　画像ファイルを Image タイプの変数に読み込み画像ファイルに出力

Open と Save というボタンがあり、

Open を押すと画像ファイルを Image タイプの変数に読み込む。

Save を押すと、BufferedImage タイプの変数に移し換えた後、画像ファイルに出力。

（Image タイプの画像を直接画像ファイルに出力できないため）

```
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import java.io.*;
import com.sun.image.codec.jpeg.*;

public class Im101121 extends Frame implements ActionListener{
    Image img;
    int w=20, h=20;

    //主処理
    public static void main(String ar[]){
        Frame f=new Im101121();
        f.setSize(new Dimension(200,200));
        f.setVisible(true);
    }

    //部品セット
    Im101121(){
        setLayout(new FlowLayout());
        Button b01=new Button("Open");
        Button b02=new Button("Save");
        b01.addActionListener(this);
        b02.addActionListener(this);
        add(b01);
        add(b02);
        addWindowListener(new WinAdapter());
    }
    //閉じる
    class WinAdapter extends WindowAdapter{
        public void windowClosing(WindowEvent we){System.exit(0);}
    }

    //イベント
    public void actionPerformed(ActionEvent ae){
        if(ae.getActionCommand()=="Open"){
                    img = getToolkit().getImage("infile.jpg");
                    repaint();
        }

                    //保存
        if(ae.getActionCommand()=="Save"){
            try{
                    FileOutputStream fo = new FileOutputStream("outfile.jpg");
  BufferedImage bimg = new BufferedImage(w,h,BufferedImage.TYPE_INT_RGB);
                    Graphics gc = bimg.getGraphics();
```

```
                    gc.drawImage(img,0,0,this);
                    JPEGImageEncoder encoder = JPEGCodec.createJPEGEncoder(fo);
                    encoder.encode(bimg);
                    fo.close();
                } catch(Exception ex){}
            }
        }
        public void update(Graphics g){
            paint(g);
        }
        public void paint(Graphics g){
            if(img !=null){
                g.drawImage(img,20,100,this);
            }
        }
}
```

2.3　ピクセルを配列に入れる　(Im101314.java)

“画像読込”と“変色コピー”のボタンがある。

画像読込ボタンを押すと、infile.jpg という画像ファイルを img という Image タイプの変数に読込み、次ぎに pixelGrabber()でピクセルの値を pixels という配列に読み込み、表示。

変色コピーボタンを押すと、配列 pixels を配列 pixels2 にを書き移し、それを img2 という Image タイプの変数に戻して表示。

（ピクセル情報は３２ビットの整数で、８ビットづつの色情報から成り、上位８ビットは 0xff であることに注意）

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

class Im101314 extends Frame implements ActionListener  {
    PixelGrabber pg;
    Button button1,button2;
    Image img,img2;
    int w=20, h=20;

    boolean button1push = false;//画像読み込みボタン
    boolean button2push = false;//処理開始ボタン
    int pixels[]=new int[w*h];//ピクセルデータ（入力）配列
    int pixels2[]=new int[w*h];//色変更したピクセル配列

    Im101314() {
        setLayout(new FlowLayout(FlowLayout.LEFT));
        img = createImage(w,h);//画像読み込み用 Image を生成

        button1 = new Button("画像読込");
        add(button1);
        button1.addActionListener(this);
        button2 = new Button("変色コピー");
        add(button2);
        button2.addActionListener(this);
    }

    public void paint(Graphics g) {

      if(button1push){
          g.drawImage(img,20,100,this);
       }

      if(button2push){
          g.drawImage(img2,100,100,this);
       }
    }

    public void actionPerformed(ActionEvent e) {
```

```
        //ボタン１が押された時の処理（画像ファイル読込）
        if(e.getSource() == button1){
            button1push = true;
            button2push=false;
            img = Toolkit.getDefaultToolkit().getImage("infile.jpg");

            //PixelGrabber を生成
            pg = new PixelGrabber(img,0,0,w,h,pixels,0,w);
            try{
                pg.grabPixels();//画像を配列に読込
            }catch(InterruptedException ie){}

            repaint();
        }
        //ボタン２が押された時の処理（ピクセルごとの処理)
        if(e.getSource() == button2){
            button2push = true;
            button1push=false;
          //単に読出して書戻し(やらなくとも同じ)
            for(int i=0;i<w*h;i++){
                int p = pixels[i];
                int red = 0xff & (p>>16);
                int green = 0xff & (p>>8);
                int blue = 0xff & p;
                pixels2[i] = (0xff000000 | red << 8 | green << 16 | blue);
            }

            img2 = createImage(new MemoryImageSource(w, h, pixels2, 0, w));
            repaint();
        }
    }

    public static void main(String[] args) {
        Im101314 f = new Im101314();
        f.setSize(300,300);
        f.addWindowListener(new WindowAdapter(){public void
                windowClosing(WindowEvent e){System.exit(0);}});
        f.setVisible(true);
    }
}
```

注意：この例ではボタンを押すことで処理が進むので問題ないが、toolkit の getImage では、ファイルの読込を待たずに処理が進むようだ。
ファイル入力の修了を待つにはメディアトラッカーを使う。
次は、bimg という Image 変数にメディアトラッカーを付けて、入力修了を待つ例である。

```
            //メディアトラッカーをセット
           MediaTracker mt = new MediaTracker(new Component(){});
            mt.addImage(bimg, 1);
            try {
                mt.waitForAll();
            } catch (Throwable t) {}
```

2.4　ピクセル配列から drawLine で BufferedImage タイプの変数を作り出力

(Im10131521.java)

infile.jpg を読み込み、ピクセルグラブをして BufferdImage 上に書き出し、ファイル出力。

黄色の長方形に、読み込み画像を重ね描きしている。画面表示は行わないので注意。

（drawImage でもよいが、わざわざ drawLine を使用）

```
import java.awt.*;
import java.awt.Color;
import java.awt.image.*;
import java.awt.image.BufferedImage;
import java.io.*;
import javax.imageio.ImageIO;

class Im10131521 {
public static void main(String[] args) {
      int width = 40;
      int height = 20;
      int[] pixels= new int[width/2*height];
      Image img;
      img = Toolkit.getDefaultToolkit().getImage("infile.jpg");
      PixelGrabber pg= new PixelGrabber(img,0,0,width/2,height,pixels,0,width/2);
      try {
            pg.grabPixels();
      } catch (InterruptedException e) {};
      BufferedImage bimg = new BufferedImage(width,
                                    height, BufferedImage.TYPE_INT_BGR);
      // 出力イメージを描画.
      Graphics g1 = bimg.getGraphics();
      g1.setColor(Color.YELLOW);
      g1.fillRect(0, 0, width, height);
      for(int i=0; i<(width/2);i++){
          for(int j=0; j<height; j++){
                int pt= pixels[i*width/2+j];
                int r= (pt & 0x00ff0000)/0x10000;
                int g= (pt & 0x0000ff00)/0x100;
                int b=   pt & 0x000000ff;
                g1.setColor(new Color(r,g,b));
                g1.drawLine(j, i, j, i);
          }
       }
      g1.dispose();

      // JPEG 形式で保存.
      try {
            ImageIO.write(bimg, "jpeg", new File("outfile.jpg"));
      } catch (IOException exception) {
            exception.printStackTrace();
      }
   }
}
==
```

第３章　BufferedImage タイプの処理

（１）　画像ファイルの入出力と簡単な描画

3.1 画像ファイルを BufferedImage1 タイプの変数に読込み、そのまま出力(Im2021.java)

infile.jpg を一旦 BufferedImage　bimg に保存し、それを出力ファイル outfile.jpg に書出す。

```
import java.awt.*;
import java.awt.event.*;       // For event
import java.io.*;              // For File
import java.awt.image.*;       // For BufferedImage
import javax.imageio.*;        // For ImageIO

public class Im2021 extends Frame {
    BufferedImage bimg;

    public static void main(String[] args) {
        Im2021 f = new Im2021();
        f.setBounds(100,100,300,300);
        f.setVisible(true);
    }

    Im2021() {
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });

        try {
            File imgfile = new File("infile.jpg");   // ファイルオブジェクト作成
            bimg = ImageIO.read(imgfile);        // 読みだす
        } catch (Exception e) { }

        try {
            File imgfile2 = new File("outfile.jpg");   // ファイルオブジェクト
            ImageIO.write(bimg, "jpg", imgfile2);        // 書きだす
        } catch (Exception e) {}

    }
    public void paint(Graphics g) {
        g.drawImage(bimg,50,50,this);
    }
}
==
```

3.2 BufferedImage タイプの変数を作りファイルに出力（Im2221.java）
黄色い正方形を描いてファイル（outfile.jpg）に保存。
（Im2221.java）

```
import java.awt.*;
import java.awt.image.*;
import java.awt.image.BufferedImage;
import java.io.*;
import javax.imageio.ImageIO;
import java.awt.Color;

class Im2221 {
public static void main(String[] args) {
    int width = 20;
    int height = 20;

    BufferedImage bimg = new BufferedImage(width,
                            height, BufferedImage.TYPE_INT_BGR);

    // 描画.
    Graphics gg = bimg.getGraphics();
    gg.setColor(Color.YELLOW);
    gg.fillRect(0, 0, width, height);
    gg.dispose();

    // JPEG 形式で保存.
    try {
        ImageIO.write(bimg, "jpeg", new File("outfile.jpg"));
    } catch (IOException exception) {
        exception.printStackTrace();
    }
 }
}
```

＝＝

（2）　ピクセル操作

ここから本格的 BufferedImage のピクセル操作。

3.3　BufferedImage のピクセル情報を配列に保存　(Im2023.java)

BuffererdImage に対しても PixelGrabber は使える。画像ファイル（infile.jpg）を読み込み、色変換して表示。色変換法は 2.3 と異なる。配列から BufferedImage タイプの変数を作ることはできない（Image タイプの変数は作れるのに)。

```
import java.awt.*;
import java.io.*;
import java.awt.event.*;
import java.awt.image.*;
import javax.imageio.*;
import java.awt.Color;

class Im2023 extends Frame implements ActionListener  {
      PixelGrabber pg;
      Button button1,button2;
      BufferedImage bimg,bimg2;
      Image img2;
      int w=20,h=20;

      boolean button1push = false;//画像読み込みボタン
      boolean button2push = false;//処理開始ボタン
      int pixels[]=new int[w*h];//ピクセルデータ（入力）配列
      int pixels2[]=new int[w*h];//色変更したピクセル配列

      Im2023() {
            setLayout(new FlowLayout(FlowLayout.LEFT));
            button1 = new Button("画像読込");
            add(button1);
            button1.addActionListener(this);
            button2 = new Button("変色コピー");
            add(button2);
            button2.addActionListener(this);
            try {
                File file= new File("infile.jpg");
                bimg= ImageIO.read(file);
            } catch(Exception e){}

      }

      public void paint(Graphics g) {
          if(button1push){
                g.drawImage(bimg,20,100,this);
           }
          if(button2push){
                g.drawImage(img2,100,100,this);
```

```
        }
    }

    public void actionPerformed(ActionEvent e) {
        //ボタン１が押された時の処理（画像ファイル読込）
        if(e.getSource() == button1){
            button1push = true;
            button2push=false;
            //PixelGrabber を生成
            pg = new PixelGrabber(bimg,0,0,w,h,pixels,0,w);
            try{
                pg.grabPixels();//画像を配列に読込
            } catch(InterruptedException ie){}

            repaint();
        }
        //ボタン２が押された時の処理（ピクセルごとの処理）
        if(e.getSource() == button2){
            button2push = true;
            button1push=false;
          //単に読出して書戻し(やらなくとも同じ)
            for(int i=0;i<w*h;i++){
                Color c= new Color(pixels[i]);
                int red = c.getRed();
                int green = c.getGreen();
                int blue = c.getBlue();
                pixels2[i] = (0xff000000 | red   | green <<8 | blue<<16);
            }
            img2 = createImage(new MemoryImageSource(w, h, pixels2, 0, w));
//次の変換はできない
//bimg2 = (BufferedImage)createImage(new MemoryImageSource(w, h, pixels2, 0, w));
            repaint();
        }
    }

    public static void main(String[] args) {
        Im2023 f = new Im2023();
        f.setSize(300,300);
        f.addWindowListener(new WindowAdapter(){public void
                windowClosing(WindowEvent e){System.exit(0);}});
        f.setVisible(true);
    }
}
```

3.4 BufferedImage レベルでのピクセル操作（Im202324.java）

infile.jpg という画像ファイルを読込み、白黒画像に変える。BufferdImage レベルでのピクセル操作は getRGB と setRGB を使う。

```
import java.awt.*;
import java.awt.event.*;       // For event
import java.io.*;              // For File
import java.awt.image.*;       // For BufferedImage
import javax.imageio.*;        // For ImageIO

public class Im202324 extends Frame {
    BufferedImage bimg;

    public static void main(String[] args) {
        Im202324 f = new Im202324();
        f.setBounds(100,100,300,300);
        f.setVisible(true);
    }

    Im202324() {
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });

        try {
            File imgfile = new File("infile.jpg");   // ファイルオブジェクト作成
            bimg = ImageIO.read(imgfile);        // 読みだす
        } catch (Exception e) {}

        for (int y = 0; y < bimg.getHeight(); y++) {
            for (int x = 0; x < bimg.getWidth(); x++) {
                int rgb = bimg.getRGB(x, y);
                int a = (rgb>> 24) & 0xff;
                int r = (rgb >> 16) & 0xff;
                int g = (rgb >> 8) & 0xff;
                int b = rgb & 0xff;
                int m = (2*r + 4*g + b) / 7;
                bimg.setRGB(x, y, new Color(m, m, m, a).getRGB());
        }
    }

    }
    public void paint(Graphics g) {
        g.drawImage(bimg,50,50,this);
    }
}
==
```

3.5　DataBuffer レベルでのピクセル操作(Im202324r.java)

これは、DataBuffer に関して、色々勉強するためのプログラムである。

画像ファイル（infile.jpg）を読み込み、cpyImg に移し、そこに setElem()で青線を描いたり、getElem()でピクセルの値を読み出したりする。

また、別の（BufferedImage） gradImg にグラデーション画像も作っている。

後の方にあるコメントアウトを色々変えて、３つの BufferedImage の画像（bimg,cpyImg,grdImg）を見てみよう。

bimg をわざわざ cpyImage に移したのは、Raster の DataBuffer で getElem()するためである。

```
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;

class Im202324r extends Frame  {

        Button bt;
        BufferedImage biImg;
        BufferedImage image;
        BufferedImage cpyImage;
        public Im202324r() throws Exception{

        int xsize = 20;
        int ysize = 20;
        int[] pixel = new int[xsize*ysize];

//画像ファイル読み込み処理
        FileInputStream fis = new FileInputStream("infile.jpg");
        image = ImageIO.read(fis);//読み込んだイメージ
        cpyImage = new BufferedImage(xsize, ysize,
BufferedImage.TYPE_INT_RGB);//コピー用イメージ
        Graphics g1 = cpyImage.createGraphics();
        g1.drawImage(image, 0, 0, xsize, ysize, null);//イメージをコピー
        g1.setColor(Color.black);
        g1.drawLine(1,1,20,20);
        fis.close();

        WritableRaster ras1 = cpyImage.getRaster();
        DataBufferInt buffer = (DataBufferInt)(ras1.getDataBuffer());

//pixel 情報取得
        for(int k = 200; k < 240; k++)
            ras1.getDataBuffer().setElem(k,255);
        for(int k = 0; k < xsize*ysize; k++)
            pixel[k] = buffer.getElem(k);
        for(int k = 0; k < xsize*ysize; k++)
```

```
            System.out.println(pixel[k]);
//ここから話が変わり、grdImg にグラデーションを作る
        biImg=new BufferedImage(256,256,BufferedImage.TYPE_INT_BGR);
        WritableRaster ras=biImg.getRaster();
        Graphics g2=biImg.createGraphics();

        for (int i=0;i<256;i++)
            for (int j=0;j<256;j++)
                ras.getDataBuffer().setElem(0,i*256+j,(j<<16)+(i<<8)+i);
                g2.drawString("Test",64,64);
        }

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

                Im202324r tes=new Im202324r();
                tes.pack();
                tes.show();
        }


        public void paint(Graphics g) {
//以下のコメント行を変えて色々表示してみよう
//              g.drawImage(biImg,16,24,this);
//              g.drawImage(image,16,24,this);
                g.drawImage(cpyImage,16,24,this);
        }
}
```

（注意）bimg も cpyImage も BufferedImage なのに、infile.jpg から直接作った bimg は getRaster して(DataBufferInt)のキャストを行おうとするとエラーになる。
これは、jpg ファイルが圧縮されているためで、圧縮を解いて image に入れればよい。これが（Im2023x.java）プログラムである。

（Im2023x.java）
画像ファイルを解凍して BufferedImage に格納し、ピクセル値を出力する。
解凍に、
JPEGCodec.createJPEGDecoder、と
decoder.decodeAsBufferedImage を使用。
この時、
import com.sun.image.codec.jpeg.*;
が必要になる。

（Im2023x.java）
import java.awt.Graphics2D;
import java.awt.image.*;

```
import java.awt.image.BufferedImage;
import java.io.*;
import com.sun.image.codec.jpeg.*;

public class Im2023x {

   public static void main(String[] args) throws Exception{
        int xsize = 20;
        int ysize = 20;
        int[] pixel = new int[xsize*ysize];

        /*画像ファイル（20×20）読込み*/
        FileInputStream fis = new FileInputStream("infile.jpg");

//BufferedImage bimg = ImageIO.read(fis);これでは下の getRaster()でエラーになる

     JPEGImageDecoder decoder = JPEGCodec.createJPEGDecoder(fis);
     BufferedImage bimg = decoder.decodeAsBufferedImage();

        fis.close();

        WritableRaster ras = bimg.getRaster();
        DataBufferInt buffer = (DataBufferInt)(ras.getDataBuffer());

        //pixel 情報取得
        for(int i = 0; i < xsize*ysize; i++)
                pixel[i] = buffer.getElem(i);
        //pixel 情報表示
        for(int i = 0; i < xsize*ysize; i++)
                System.out.println(pixel[i]);
        }
}
```

3.6 Raster レベルでの画像生成（Im24.java）

setPixel 命令を使ってピクセルに色書き込む。ただし、getPixel という命令はないので、簡単にピクセル値を読み出せない（と思う）。

setPixel 使用して 4 本の縦線を描いている。

```
import java.awt.*;
import java.awt.image.*;
public class Im24 extends Frame {
    BufferedImage bimg;

    public Im24() {
        setSize( 300, 300 );
        setBackground( Color.white );

    // 色線のイメージ
        bimg = new BufferedImage( 100,100, BufferedImage.TYPE_INT_RGB );
        WritableRaster raster = bimg.getRaster();
        int[] data = new int[3];                    // RGB データ格納用
        data[0]=255; data[1]=0; data[2]=255;     // RGB のデータを設定する
        for(int i=0; i<100; i++)
                raster.setPixel( 50, i, data );            // ピクセル値の設定する
        data[0]=0; data[1]=128; data[2]=0;       // RGB のデータを設定する
        for(int i=0; i<100; i++)
            raster.setPixel( 51, i, data );          // ピクセル値の設定する
        data[0]=0; data[1]=128; data[2]=0;       // RGB のデータを設定する
        for(int i=0; i<100; i++)
                raster.setPixel( 52, i, data );            // ピクセル値の設定する
        data[0]=255; data[1]=0; data[2]=255;     // RGB のデータを設定する
        for(int i=0; i<100; i++)
                raster.setPixel( 53, i, data );            // ピクセル値の設定する
    }

    public void paint( Graphics g ) {
       g.drawImage(bimg,0,0, null);
    }

    public static void main( String argv[] ) {
        Im24 f = new Im24();
        f.pack();
        f.setVisible( true );
    }
}
```

第４章　　フレームに表示された画像をファイルに保存

モニタのウインドウ(フレーム)に表示された画像をファイルに保存したいケースは多い。ここでは、それらのサンプルを紹介する。

4.1 （復習）フレーム表示画面の保存（Im25.java）

フレームに表示した画面を保存するには、g の上に保存するのではなく、自分のグラフィックス（ここでは myG）を用意し、そこに書いておいて、drawImage で g の上に書き出す。保存するのは drawImage で作った画像。保存ファイルは outfile.jpg である。

基本は 2.1 と同じなので復習と言える。

```
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;

public class Im25 extends Frame implements ActionListener{
    int pc= -1;
    BufferedImage bimg= null;

    public static void main(String args[]){
        Frame f=new Im25();
        f.setTitle("表示画面の保存");
        f.setSize(100,100);
        f.setVisible(true);
    }
    //部品セット
    Im25(){
        setLayout(new FlowLayout());
        Button b0=new Button("draw");
        Button b1=new Button("save");
        b0.addActionListener(this);
        b1.addActionListener(this);
        add(b0);
        add(b1);
        addWindowListener(new WinAdapter());
    }
    //閉じる
    class WinAdapter extends WindowAdapter{
        public void windowClosing(WindowEvent we){System.exit(0);}
    }
    //描画
    public void paint(Graphics g){

         if(bimg==null) bimg =(BufferedImage)createImage(getWidth(), getHeight()) ;
         Graphics myG =bimg.createGraphics();

        if(pc==0){
            myG.drawRect(50,70,20,20);
```

```
            }
            if(pc==1){
                try{
                    ImageIO.write(bimg, "jpeg", new File("outfile.jpg"));
                }catch(IOException ex){};
            }
              g.drawImage(bimg, 0, 0, null);
              myG.dispose();
        }
        //イベント
        public void actionPerformed(ActionEvent ae){
            if(ae.getActionCommand()=="draw"){
                pc=0;
                repaint();
            }
            if(ae.getActionCommand()=="save"){
                pc=1;
                repaint();
            }
        }
//repaint()で画面クリヤーしないようにするには以下を 3 行を生かす
//      public void update(Graphics g){ paint(g); }
}
```

4.2 Java2D のための準備（Im25a.java）

次のアフィン変換表示画面保存の準備で、java2D を使用している他は Im25 と同じ。

```
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import java.awt.Graphics2D;

public class Im25a extends Frame implements ActionListener{
	int pc= -1;
	BufferedImage bimg= null;
	//主処理
	public static void main(String args[]){
		Frame f=new Im25a();
		f.setTitle("表示場面の保存(基本)");
		f.setSize(200,200);
		f.setVisible(true);
	}
	//部品セット
	Im25a(){
		setLayout(new FlowLayout());
		Button b0=new Button("draw");
		Button b1=new Button("save");
		b0.addActionListener(this);
		b1.addActionListener(this);
		add(b0);
		add(b1);
		addWindowListener(new WinAdapter());
	}
	//閉じる
	class WinAdapter extends WindowAdapter{
		public void windowClosing(WindowEvent we){System.exit(0);}
	}
	//描画
	public void paint(Graphics g){
		Graphics2D g2 = (Graphics2D)g;
		 if(bimg==null) bimg =(BufferedImage)createImage(getWidth(), getHeight()) ;
		 Graphics2D myG =bimg.createGraphics();

		if(pc==0){
			myG.drawRect(50,50,20,20);
		}
		if(pc==1){
			try{
				ImageIO.write(bimg, "jpeg", new File("outfile.jpg"));
			}catch(IOException ex){};
		}
		 g2.drawImage(bimg, 0, 0, null);
```

```
            myG.dispose();
        }
        //イベント
        public void actionPerformed(ActionEvent ae){
            if(ae.getActionCommand()=="draw"){
                pc=0;
                repaint();
            }
            if(ae.getActionCommand()=="save"){
                pc=1;
                repaint();
            }
        }

//repaint()で画面クリヤーしないようにするには次の行を生かす
//      public void update(Graphics g){ paint(g); }
}
```

4.3 アフィン変換表示画像の保存（Im25b.java）

“save”ボタンで正方形を描き、

“draw”ボタンでアフィン変換（１０度回転）の準備をして水平線を描く。水平線は１０度回転して表示される。結果は outfile.jpg に保存。

```
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import java.awt.geom.*;
import java.awt.Graphics2D;

public class Im25b extends Frame implements ActionListener{
    int pc= -1;
    BufferedImage bimg= null;
    //主処理
    public static void main(String args[]){
        Frame f=new Im25b();
        f.setTitle("画面の保存");
        f.setSize(100,100);
        f.setVisible(true);
    }
    //部品セット
    Im25b(){
        setLayout(new FlowLayout());
        Button b0=new Button("draw");
        Button b1=new Button("save");
        b0.addActionListener(this);
        b1.addActionListener(this);
        add(b0);
        add(b1);
        addWindowListener(new WinAdapter());
    }
    //閉じる
    class WinAdapter extends WindowAdapter{
        public void windowClosing(WindowEvent we){System.exit(0);}
    }
    //描画
    public void paint(Graphics g){
        Graphics2D g2 = (Graphics2D)g;
         if(bimg==null)bimg =(BufferedImage)createImage(getWidth(), getHeight()) ;
         Graphics2D myG =bimg.createGraphics();

        if(pc==0){
           myG.drawRect(60,60,30,30);
        }
        if(pc==1){
           AffineTransform af = new AffineTransform();
           af.setToRotation(10 * Math.PI/180);
           myG.setTransform(af);
```

```
                myG.setColor(Color.red);
                myG.drawLine(0, 50, 90, 50);
                try{
                   ImageIO.write(bimg, "jpeg", new File("outfile.jpg"));
                }catch(IOException ex){};
            }
             g2.drawImage(bimg, 0, 0, null);
             myG.dispose();
        }
        //イベント
        public void actionPerformed(ActionEvent ae){
            if(ae.getActionCommand()=="draw"){
                pc=0;
                repaint();
            }
            if(ae.getActionCommand()=="save"){
                pc=1;
                repaint();
            }
        }
//repaint()で画面クリヤーしないようにするには次の行を生かす
//      public void update(Graphics g){ paint(g); }
}
==
```

第５章　　色々な画像変換
Java2D では、フィルター機能を使うとアフィン変換など色々な画像変換が可能である。
ここでは、簡単な画像の縮小拡大を行うアフィン変換プログラムを挙げておく。
この分野に関してはネット等に沢山資料があるので、そちらを参照されたい。

5.1 Java2D のフィルター処理（Imaf.java）
関数（メソッド）rescale は org のサイズを変えて return してくる。

```
import java.awt.geom.AffineTransform;
import java.awt.image.AffineTransformOp;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;

public class Imaf {

    public Imaf() {
        try {
            File ifile = new File("infile.jpg");
            BufferedImage org = ImageIO.read(ifile);

            //次の第２、第３引数が拡大縮小後のサイズ
            BufferedImage dst = rescale(org, 10, 10);

            ImageIO.write(dst,"jpg",new File("outfile.jpg"));
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }

    public BufferedImage rescale(BufferedImage org, int width, int height) {
        BufferedImage image = new BufferedImage(width, height, org.getType());

        double sx = (double)width / (double)org.getWidth();
        double sy = (double)height / (double)org.getHeight();
        AffineTransform trans = AffineTransform.getScaleInstance(sx, sy);
        AffineTransformOp operation = new AffineTransformOp(trans,
                        AffineTransformOp.TYPE_BILINEAR);

        operation.filter(org, image);

        return image;
    }

    public static void main(String[] args) {
        new Imaf();
    }
}
==
```

Java によるシリアル通信

COM ポートを通じての通信は RS232C の通信などとも呼ばれる。
但し、これを行うには Java communication API をインストールする必要がある。
多分、今は手に入らないので私に連絡してくれれば、お送りします。
＝＝インストール手順
STEP1: javacomm20-win32.zip を入手・解凍する。

STEP2: 3 つのファイルを以下のフォルダーに入れる。

comm.jar
JDK では、j2sdk1.4.2_xx¥jre¥lib¥ext
JRE では、j2re1.4.2_xx¥lib¥ext

javax.comm.propaties
JDK では、j2sdk1.4.2_xx¥jre¥lib
JRE では、j2re1.4.2_xx¥lib

win32com.dll
JDK では、j2sdk1.4.2_xx¥jre¥bin
JRE では、j2re1.4.2_xx¥bin

（注意）普通 j2sdk しか使っていないので、JDK だけに格納した場合、うまくいかず、JRE にも格納してうまくいった経験がある。
＝＝
次にシリアル通信チャットプログラムを示す。
まず、２台の PC の COM ポートをクロスケーブルで結ぶ。
転送速度と方式を設定（プログラム中に速度などを書く）する。実行するとデータ到着待ちになる。キー入力すると相手に送られる。
一方はハイパーターミナルでもよい。
（ComChat.java）

```
import java.io.BufferedReader;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.Socket;
import java.net.SocketException;
import java.net.UnknownHostException;
import java.util.Arrays;
import java.util.Properties;
import java.util.StringTokenizer;
import java.util.TooManyListenersException;
import java.util.Vector;
import javax.comm.CommPortIdentifier;
```

```
import javax.comm.NoSuchPortException;
import javax.comm.PortInUseException;
import javax.comm.SerialPort;
import javax.comm.SerialPortEvent;
import javax.comm.SerialPortEventListener;
import javax.comm.UnsupportedCommOperationException;
import javax.print.attribute.standard.Chromaticity;
public class ComChat implements SerialPortEventListener {
	// SerialPort
	private SerialPort	serialport = null;
	private InputStream	in		= null;
	private OutputStream out		= null;

	public ComChat() {
		super();
	}
	public static void main(String[] args) {
		ComChat cc = new ComChat();
		cc.send();
		cc.close();
	}
	/*
	 * ASCII データを受信し表示
	 * ETX(0x02)がデータ送信終了コード
	 */
	public void serialEvent(SerialPortEvent event) {
		try {
			if (event.getEventType() == SerialPortEvent.DATA_AVAILABLE) {
				StringBuffer inputBuffer = new StringBuffer();
				int newData = 0;
				System.out.println("Serial Receive START->");
				while (true) {
					try {
						newData = in.read();// 入力ストリームから読み込み
						if (newData == -1 || newData == 0x02) {// EOF or ETX?
							System.out.println(inputBuffer+"<-END");
							break;
						}
						if ('¥r' == (char)newData) {
							inputBuffer.append('¥n');
							System.out.print(inputBuffer);
						} else {
							inputBuffer.append((char)newData);
						}
					} catch (IOException ex) {
						System.err.println(ex);
						return;
					}
				}
			}
		}
		catch (Exception e) {
			e.printStackTrace();
```

```
        }
    }
    /*
     * シリアルポートを初期化後
     * Key 入力された文字列を送信
     */
    public void send() {
        try {
            // Open new Serial Port.
            openSerialPort();
            BufferedReader keyin = new BufferedReader(new
InputStreamReader(System.in));
            String keyString;
            while (true) {
                try {
                    // Key 入力待ち
                    keyString = keyin.readLine();
                    // EXIT と入力された場合は終了
                    if(keyString.equals("EXIT")){
                        break;
                    }
                                                        keyString= keyString + '¥r';
                    byte[] bytes = keyString.getBytes();
                    out.write(bytes);       // 出力ストリームにバイト列を書込
                    out.flush();            // 出力ストリームをフラッシュ
                }
                catch (Exception e) {
                    e.printStackTrace();
                    continue;
                }
            }
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
    /*
     * ストリーム、シリアルポートのクローズ処理
     */
    public void close() {
        try {
            if(in != null){
                in.close();
            }
            if(out != null){
                out.close();
            }
            if (serialport != null) {
                serialport.close();
            }
        }
        catch (Exception e) {
            e.printStackTrace();
```

```
        }
    }
    /**
     * シリアルポートの初期化
     * ポート：COM1
     * ボーレート：9600
     * データビット：8
     * ストップビット：1
     * パリティ：なし
     * フロー制御：なし
     */
    private void openSerialPort() {
        try{
            CommPortIdentifier port = CommPortIdentifier.getPortIdentifier("COM1");
            // ユニークな名前でシリアルポートを開く
            SerialPort serialPort = (SerialPort) port.open("ComChat", 30000);
            // シリアルポートのパラメータを設定
            serialPort.setSerialPortParams(     9600,           // Baudrate
                                SerialPort.DATABITS_8,          // Data Bits
                                SerialPort.STOPBITS_1,          // Stop Bits
                                SerialPort.PARITY_NONE);    // Parity Bit
            // Flow Control Mode
            serialPort.setFlowControlMode(SerialPort.FLOWCONTROL_NONE);
            // イベントリスナー登録
            serialPort.addEventListener(this);
            // シリアルポートがデータを受信した際に教えてねと設定
            serialPort.notifyOnDataAvailable(true);
            // 入力ストリームを取得
            in = serialPort.getInputStream();
            // 出力ストリームを取得
            out = serialPort.getOutputStream();
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
    private void dumpBytes(byte[] data){
        for(int i = 0; i < data.length; i++){
            System.out.print("[" + Integer.toHexString(data[i]) + "]");
        }
        System.out.println();
    }
}
```

＝＝終わり

Java による USBIO プログラム

準備：
ｋｍ２社の USBIO を使う場合、古谷の持つ usbio_km2 フォルダーを用意。
その中の、
BinaryOnly フォルダーから usbio.jar と usbio.dll を自分の使用するフォルダーに移す。

自分のフォルダーにソースプログラムを入れる。

もし、ソースプログラムが Led.java だったら、

コンパイルは

```
javac -classpath usbio.jar Led.java
```

とします。

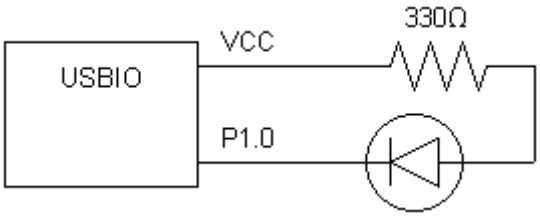
実行は、

```
java -cp .;usbio.jar Led
```

となります。

Led を点滅させるプログラム（Led.java）

この例は、USB-IO に LED を一個だけ接続し、それを点滅させるものです。LED は Port #1 のビット 0 に接続します。具体的な回路図は以下の通りです。



プログラムは以下です。

```
import xyzzy.hardware.UsbIo;

class Led
{
    public static void main ( String[] argv )
    {
        try {
            final UsbIo.Device usbio = UsbIo.getDevice();
```

```
                try {
                    int bits = 0xFF;

                    for ( int count = 25; count > 0; --count ) {
                        bits ^= 0x01;
                        usbio.invoke( UsbIo.WRITE1, bits );
                        Thread.sleep( 200 );
                    }

                } finally {

                    usbio.invoke( UsbIo.WRITE1, 0xFF );
                    usbio.close();
                }

            } catch ( Throwable e ) {
                /* Show any Error/Exception.  */
                e.printStackTrace();
            }
        }
    }
```

USBIO への出力は、
usbio.invoke(UsbIo.WRITE1, 0xFF);
のようにし、

入力は、
１度“１”を出力した後、読み込みます。即ち、
usbio.invoke(UsbIo.WRITE1, 0xFF);
usbio.invoke(UsbIo.READ1,0);
となります。