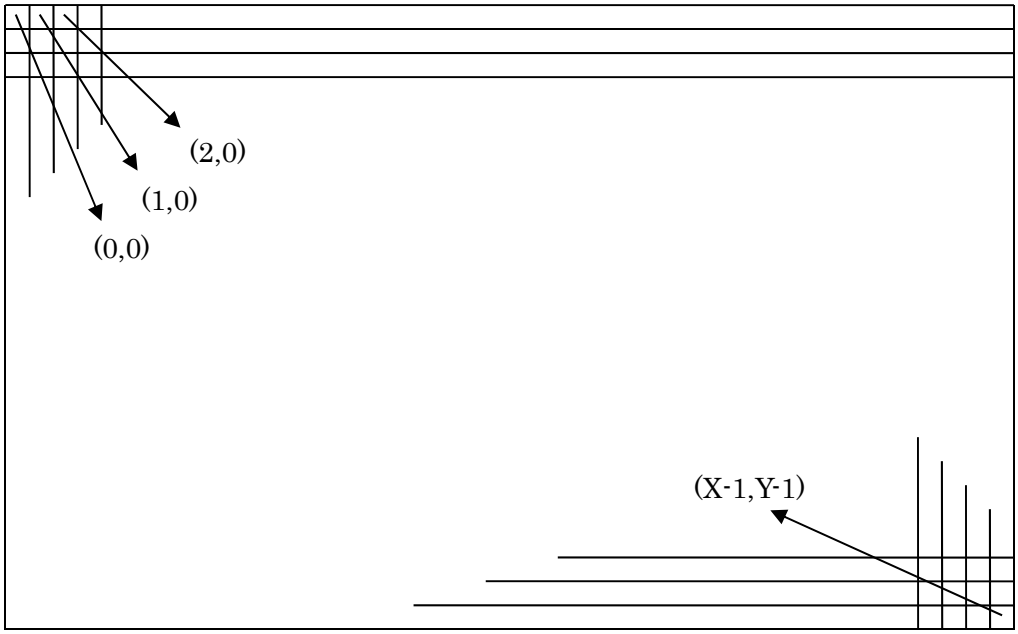
学生証番号：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿　名前：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

4. Java と画像

コンピュータ画像(CG)はマルチメディアにおいて最も大事な要素である。CG は画素またはピクセルと呼ばれる点の集合でできており、色によって白黒、グレー、カラー画像で分類される。コンピュータで CG を表現するために使われる座標系は画面の左上の頂点を原点とし、その点の座標を(0, 0)とする。その原点を基準に右に行くほど x 座標は増加し、下に行くほど y 座標は増加する。画面の右下の頂点の座標を(X-1, Y-1)とするとその画像の解像度は X*Y、x の最大値は X-1、y の最大値は Y-1 になる。



CG と座標

4.1　ビットマップ画像

コンピュータのディスプレイと同様に、点の集まりとして表現されたイメージをビットマップ画像と呼ぶ。白黒(monochrome)イメージは画素(pixel)1 点が 1bit に、256 段階の濃淡(グレー)画像なら画素 1 点が 1byte に、トゥルーカラー(通常の 1600 万色)画像なら画素 1 点が、3byte(RGB それぞれ 1byte ずつ)に対応している。そのためビットマップ画像は拡大すると画像が粗くなったり、画像データの中にむだな部分が多くなったりするといった短所がある。

pgm(portable gray map)はビットマップ画像ファイル形式の一種であり、その構成は非常に単純である。pgm はグレー画像の形式で類似画像形式としては白黒画像用の pbm(portable bit map)、カラー画像用の ppm(portable pixel map)がある。(参考：http://en.wikipedia.org/wiki/Netpbm_format)

● pgm の構成（横幅が X ピクセル、縦幅が Y ピクセルの場合）

P2　X　Y　最大輝度値 (0,0)の輝度データ

(1,0)の輝度データ

(2,0)の輝度データ

…

(X-1,Y-1)の輝度データ

```
/* sample 3.1 */
class Draw
{
	private int sz_x, sz_y;
	private int canvas[ ][ ];

	public Draw(int x, int y)
	{
		canvas = new int[x][y];

		sz_x = x;
		sz_y = y;
	}

	public void init()
	{
		for (int i = 0; i < sz_x; i++)
			for (int j = 0; j < sz_y; j++)
				canvas[i][j] = 0;
	}

	public void dot(int x, int y)
	{
		canvas[x][y] = 255;
	}

	public void line(int x_start, int y_start, int x_end, int y_end)
	{
		int big, tmp_x, tmp_y;

		if (Math.abs(x_end-x_start) > Math.abs(y_end-y_start))
			big = Math.abs(x_end-x_start);
		else
			big = Math.abs(y_end-y_start);

		for (int i = 0; i <= big; i++)
		{
			tmp_x=x_start+i*(x_end-x_start)/big;
			tmp_y=y_start+i*(y_end-y_start)/big;
			canvas[tmp_x][tmp_y] = 255;
		}
	}

	public void out()
	{
		System.out.println("P2 "+sz_x+" "+sz_y+" 255");
		for(int i=0; i < sz_y; i++)
			for(int j=0; j < sz_x; j++)
				System.out.println(canvas[j][i]);
```

```
        }
}

class test
{
        public static void main(String arg[ ])
        {
                int x_size=200, y_size=200;
                Draw cvs = new Draw(x_size, y_size);

                cvs.init();
                cvs.line(100, 10, 10, 100);
                for (int i = 0; i < 100; i++)
                        cvs.dot((int) Math.round(Math.random()*(x_size-1)),
                                                (int)   Math.round(Math.random()*(y_size-
1)));
                cvs.out();
        }
}
```

sample 3.1 は pgm ファイルを生成するサンプルプログラムである。配列を宣言し、一つ一つの要素をピクセルと見立て、配列の要素の値を変えて描画している。init ですべてのピクセルを黒である 0 に設定し、line と dot で白く描画している。
このサンプルプログラムを実行は

```
java test > tmp.pgm ↲
```

とリダイレクトでファイルを生成する。リダイレクトとは出力デバイスを変えるとの意味で、この場合はモニターに出力されるはずのものを、リダイレクト(>)を使い、ファイルに出力することになる。実行後できたファイル tmp.pgm の確認はデスクトップにあるフォルダ「画像系」の中の IrFanView を実行し tmp.pgm をロードして行う。Math.round、Math.random に関しては java API マニュアルの java.lang.Math を参考する。(リダイレクトが良く理解できない人は java test だけを先に実行し，その結果を確認した上で上記のリダイレクトを実行し tmp.pgm の中身を確認する)

- 演習１（＊）

void line()のコードの一部を変更し，void dot()を用いて line()を実現する。また， canvas[ ][ ] = 255; の値を２５５から０～２５５の値にランダムに変化するようにし、結果がどう変わったか確認する。

- 演習２

四角形を描くメソッド void box(int st_x, int st_y, int ed_x, int ed_y)を追加する。st_x と st_y は四角形のある頂点座標、ed_x と ed_y はその頂点の対角頂点座標である。（濃淡値 255 で描く。）

- 演習３（＊）

円を描くメソッド void circle(int x, int y, int radius)を追加する。x と y は円の中心座標、radius は円の半径である。（濃淡値 255 で描く。）

◆ 挑戦１

楕円を描くメソッド void oval(int x, int y, int lng, int shrt, int angle)を追加する。x と y は楕円の中心座標、lng は楕円の長いほうの半径、shrt は楕円の短いほうの半径、angle は楕円の半径の長いほうの角度である。（濃淡値 255 で描く。）

◆ 挑戦２

ある座標を指定するとその点を含んだ領域を塗りつぶすメソッド void paint(int x, int y)を追加する。

◆ 挑戦３

ホームページの資料を参考に pgm を ppm に拡張し，上記の line、dot に色指定機能を追加する。

4.2 java.awt.Graphics

Java には CG のためのクラスとして java.awt.Graphics というクラスが存在し、CG のための様々なメソッドが中に用意されている。

```
/* sample 3.2 */
import java.applet.Applet;
import java.awt.*;

public class Draws extends Applet
{
	Image image;

	public void init()
	{
		image = getImage(getDocumentBase(), "sample.jpg");
	}

	public void paint(Graphics g)
	{
		g.drawImage(image, 1, 80, this);
		g.setColor(Color.RED);
		g.drawLine(10, 10, 160, 170);
		g.setColor(Color.GREEN);
		g.drawOval(50, 70, 90, 90);
		g.setColor(Color.BLUE);
		g.drawRect(130, 110, 40, 60);
		g.setColor(Color.ORANGE);
		g.setFont(new Font("Impact", Font.BOLD, 25));
		g.drawString("demonstration", 20, 30);
	}
}
```

Sample3.2 はアプレットなので appletviewer で実行する。（本資料２．９参照）API 仕様のホームページから java.awt.Graphics を開き、drawImage、setColor、drawLine、drawRect、setFont、drawString 等のメソッドの使い方を確認する。getImage、getDocumentBase は API 仕様のホームページから java.applet.Applet を開き、メソッドの用途を確認する。

● 演習４（＊）

Sample3.2 で表示された CG をベクタグラフィックスで保存したら，その保存したファイルの中身は

どのようなものになるか各自任意に想定して（定義して）テキストエディタで作成してみよ。

- 演習５

複数の画像ファイルを、座標を変更しながらスクリーン上に順番に描画し、動画を実現せよ。(画像ファイルはホームページからダウンロードできる。)(できるだけ動画に見えるように工夫をする。)

◆ 挑戦４

sample 3.2 に float degree の宣言を追加し、実行結果を degree 度回転させる。(ただし、写真画像と文字列は回転しなくて良い)

4.3 画像処理

CG を目的に合わせて処理を行うことを画像処理と呼ぶ。解像度を変える、コントラストを上げる、カラー画像をグレー画像に変換する、エッジを抽出するといった様々な画像処理が存在する。Sample3.3 はカラーの gif イメージをロードし、グレー画像と白黒画像に変換するプログラムである。

```
/* sample 3.3 */
import java.awt.*;
import java.awt.image.*;
import java.applet.Applet;


public class ImageProcess extends Applet
{
	int WIDTH, HEIGHT;
	Image imageInput,imageOutput1, imageOutput2;
	int[ ] pixelInput, pixelOutput1, pixelOutput2;

	public void loadImage(String s)
	{
		MediaTracker mt;

		imageInput = getImage(getDocumentBase(), s); // どこのどのファイル化指定
		mt = new MediaTracker(this);
		mt.addImage(imageInput,0); // 画像の id を 0 番に登録
		try
		{
			mt.waitForID(0); // 画像のロードを開始、読み込まれるまで待つ
		}
		catch (InterruptedException e)
		{
			System.out.println(e);
		}


		HEIGHT = imageInput.getHeight(this);
		WIDTH  = imageInput.getWidth(this);

		pixelInput  = new int[WIDTH * HEIGHT];
		pixelOutput1 = new int[WIDTH * HEIGHT];
		pixelOutput2 = new int[WIDTH * HEIGHT];
```

```
    // imageInput の画像を配列 pixelInput に入れる
        PixelGrabber pg = new PixelGrabber(imageInput,0,0,
                                    WIDTH,HEIGHT, pixelInput,0,WIDTH);
        try
        {
            pg.grabPixels();
        }
        catch (InterruptedException e)
        {
            System.out.println(e);
        }
}

public int RGBtoGray(int rgb_value)
{
        int r, g, b;

        r = (rgb_value & 0x00FF0000) / 0x00010000;
        g = (rgb_value & 0x0000FF00) / 0x00000100;
        b = (rgb_value & 0x000000FF);

        return (int)(r * 0.299 + g * 0.587 + b * 0.114);
}

public void threshold(int thres)
{
        int cnt = 0, x, y, value;
        int white = 0xFFFFFFFF;
        int black = 0xFF000000;

        for (y = 0; y < HEIGHT; y++)
        {
            for (x = 0; x < WIDTH; x++)
            {
                value = RGBtoGray(pixelInput[y*WIDTH + x]);
                pixelOutput2[y*WIDTH + x] =
                                    0xFF000000 + value * 0x00010101;
                if (value > thres)
                    pixelOutput1[y*WIDTH + x] = white;
                else
                    pixelOutput1[y*WIDTH + x] = black;
            }
        }
        imageOutput1 = createImage(new MemoryImageSource(WIDTH, HEIGHT,
                                    pixelOutput1,0,WIDTH));
        imageOutput2 = createImage(new MemoryImageSource(WIDTH, HEIGHT,
                                    pixelOutput2,0,WIDTH));
}

public void init()
{
        loadImage("image.GIF");
        threshold(127);
        repaint();
}
```

```
        public void paint(Graphics g)
        {
                g.drawImage(imageInput, 30, 30, this);
                g.drawImage(imageOutput2, 30, 50+HEIGHT, this);
                g.drawImage(imageOutput1, 30, 70+HEIGHT+HEIGHT, this);
        }
}
```

Sample3.3 はディスクドライブにある画像ファイルを MediaTracker と getImage 等を使いメモリにロードし、PixelGrabber で画像からピクセル情報の配列を取り出している。これで出来上がった配列 pixelInput には配列の要素一つ一つが画素を表し、その構成は次のとおりである。

P = α : R : G : B （α、R、G、B はそれぞれ 1 byte）

α は透明度、R、G、B はそれぞれ赤、緑、青成分の濃度を表している。たとえば、α が 255(不透明)、R が 50、G が 60、B が 150 の場合、P は 255*256^3 + 50*256^2 + 60*256 + 150 となるが、16 進数を使うと計算が楽になる。それぞれの値を 16 進数で直すとアルファが 0xFF、R が 0x32、G が 0x3C、B が 0x96 のため P は 0xFF323C96 になる。また P から RGB の値を取り出す場合は次の式が便利である。

```
r = (P & 0x00FF0000) / 0x00010000;
g = (P & 0x0000FF00) / 0x00000100;
b = (P & 0x000000FF);
```

必要ではない部分は 0 と&することで 0 にし、0x10000, 0x100 などで割ることで桁を移動させる。10 進数で例えると 3400 を 100 で割り、34 にするのと同じである。例えば P が 0xFF323C96 の場合、P & 0x00FF0000 を計算すると 0x00320000 となる。さらに/ 0x00010000 を計算すると 0x32 となる。

public void threshold(int thres)はカラーイメージをグレーイメージに、またグレーイメージを白黒イメージに変換する。RGB データから輝度値を求めるため次の式が用いられている。

(int)(r * 0.299 + g * 0.587 + b * 0.114)

この式は RGB の濃度値から輝度を求める式であり、人間の目の特性を調査して定めたものである。新たに出来上がったグレーイメージと白黒イメージはそれぞれ画素値が配列 pixelOutput2 と pixelOutput1 に格納され、MemoryImageSource と createImage で画面に出力可能な形に変換される。

MemoryImageSource(int width, int height, int[ ] pixel, int offset, int scan)は MemoryImageSource のコンストラクタである。width と height はそれぞれ作成しようとするイメージの幅と高さで、pixel はα、R、G、B 値を格納した配列である。offset は、配列の全体ではなく一部でイメージを作成する時、どこからイメージを作るか、という開始座標を指定するとき使う。従って、配列全体で画像を作る場合は 0 でよい。scan は pixel 配列に入っているイメージの横幅を指す。MemoryImageSource は

createImage と一緒に使い、メモリ(配列)に入っているイメージで画面、ファイル等に出力可能なイメージを生成できる。

- 演習６（＊）

init()にある threshold(127)の値 127 を 0～255 の値に変えて実行し結果を確認せよ。(値３種類以上)

- 演習７（＊）

sample3.3を修正し、グレースケールイメージの輝度ヒストグラムを作成するプログラムを開発する。(自信がある人は sample3.3 を用いなくても良い。)

- 演習８

グレースケールイメージのコントラストを高くするプログラムを作成せよ。

- 演習９

ホームページのソースファイルをダウンロードしグレースケールイメージに 3×3 の加重平均値フィルタ処理を行い、その結果を出力するプログラムを完成せよ。(加重平均値フィルタは着目画素の部分の重みを２とする。)

◆ 挑戦５

演習７を拡張し、カラー画像のヒストグラム(RGB それぞろの濃度値)を作成するプログラムを完成せよ。

◆ 挑戦６

3×3 のメディアンフィルタ処理を行うプログラムを作成し、演習９の結果と比較せよ。

◆ 挑戦７

画像のエッジを検出するプログラムを作成せよ。