今回は画像を読み込んで，いろいろ扱ってみるスケッチを作っていきます．

画像を表示する

まずは単純に画像を表示するスケッチを作ってみます．

Processing で画像を表示するのは非常に簡単です．まず，新しいスケッチを作る準備をした後，適当な名前を使って保存をしてください．これは，表示する画像を入れておく場所を作るためです．

次に表示する画像ファイルを作成しているスケッチの pde ファイルと同じ場所（スケッチが保存されるフォルダ内にできた現在のスケッチ名のフォルダの中）に保存してください．画像ファイルはなんでも構いません．なにもないという人は下の画像をダウンロードして利用してください．

20081207-3.jpg

この画像を表示するスケッチコードは次のようになります．

画像の読み込み

スケッチで画像を扱うときには，画像を保存・操作するための PImage クラスのインスタンス（通称として PImage オブジェクトと呼びます）を生成します．クラスやインスタンスはプログラミング言語の種類を解説したときに出てきたオブジェクト指向プログラミングの用語です．

クラスとはオブジェクトの性質を定義するためのプロパティとオブジェクトを操作するためのメソッドの集合で定義されます．これらのプロパティやメソッドのことをメンバメソッド，メンバプロパティと呼びます．

まず PImage オブジェクトを格納するための変数を宣言します．PImage オブジェクトはコード全体から利用することになるので，この変数はグローバル変数として宣言しておきます．

```
PImage photo;
```

画像ファイルを読み込んで扱いたいときには，loadImage メソッドを次のように利用します．

```
PImage 型の変数 = loadImage( ファイルパス名 );
```

loadImage メソッドはファイルパス名を引数として受け取り，PImage オブジェクトを生成します．ファイルパス名は，実行フォルダの位置，PDE 環境で実行させているときは pde ファイルがあるフォルダからの相対パスで指定します．たとえば，data というフォルダを作りその中に画像を入れた場合は data/ ファイル名 と書きます．今回は pde ファイルと同じ場所に置いてありますので，単にファイル名を書くだけで構いません．

なおファイルパス名はダブルクォーテーション " で囲みます．ダブルクォーテーションで囲んだものを 文字列リテラル と呼び，Processing における値の一種類です．

画像の表示

PImage オブジェクトを表示するには，image メソッドを利用します．image メソッドは

```
image( PImage オブジェクト , 表示する画像左上の X 座標 , 表示する画像左上の Y 座標 );
```

と使います．

描画ウィンドウの中央に表示

現在，画像は描画ウィンドウの左上に表示されています．これを中央に表示するためには，画像と描画ウィンドウの隙間を，画像と描画ウィンドウの幅の差を二等分した量にすればよいので，image メソッドを用いて描画を行っている文を次のように変更します．

```
image( photo, (width-photo.width)/2, (height-photo.height)/2 );
```

PImage オブジェクトの横幅と縦幅は，PImage オブジェクトが格納されている変数の名前を使って 変数名 .width，変数名 .height で取得できます．表示ウィンドウの横幅と縦幅は width と height で取得でき，それらから画像を中央に表示させるときの左上座標は上記のような式で指定することができます．

画像を動かす

次に画像を動かしてみましょう．以前に作ったボールがバウンドするスケッチで，バウンドするものを図形ではなく画像にしてみます．

次に以前に作ったスケッチのコードを再度掲示しておきます．これは「たくさんの図形を動かす」のところで利用した，時間 t を用いないバージョンです．

画像の準備

まず，動かす画像の準備をします．ボールの代わりになるような画像がよいので，ペイントソフトなどで描いてみてください．ポイントは背景を透明にした GIF 形式の画像にすることです．面倒な人は下の画像を利用してください．画像は先と同じように pde ファイルと同じ場所に保存します．新しくスケッチを作成した場合は，まず保存を行ってフォルダを作成してください．

dora.gif

画像の読み込み

先と同じように, PImage オブジェクトを格納するための変数の宣言と, setup メソッドの中で画像の読み込みを行います.

```
PImage sprite;
   :
void setup()
{
    :
   sprite = loadImage("dora.gif");
    :
}
```

画像の描画

今まで ellipse メソッドで円を描いていた部分を image メソッドで画像を描くように変更します.

```
image( sprite, x, y );
```

跳ね返りの判定

ellipse メソッドは円の中心座標を指定していましたが，image メソッドは画像の左上座標を指定しています．そのため，跳ね返りの判定部分を変更します.

一番簡単なのは左壁にぶつかったときです．画像の左端が描画ウィンドウの左端より左になったときなので，その条件は x < 0 となります.

同様に, 右壁にぶつかったときは, 画像の右端が描画ウィンドウの右端より右になったときで, 画像の横幅は sprite.width なので, 条件は width < x+sprite.width となります．最後に, 地面にバウンドするのは画像の下端が描画ウィンドウの下端より下になったときで, 画像の縦幅は sprite.height なので，条件は y+sprite.height > height となります.

そこで，コードを次のように書き換えます.

```
if( y > height-sprite.height || y < 0 )  ...
if( x > width-sprite.width || x < 0 )  ...
```

最後に, 半径 r の概念がなくなったので, その宣言の削除をします．では, 作って動かしてみましょう．なお，背景は画像の背景色（dora.gif の場合は白色）に合わせた方がよいでしょう.

画像をいじくる

画素情報を得る，変更する

PImage オブジェクトは，画像内の 1 画素ごとにどんな色なのかを調べることができます．また, 1 画素ごとに変更することも可能です．たとえば, PImage 型の変数 photo に画像が読み込んであるとき，その画像の左上を原点とし，右方向を X 軸性方向，下方向を Y 軸正方向とする座標系において座標が (x,y) の点の色は，PImage クラスのメンバメソッド get か，メンバプロパティ

pixels を用いて,

```
photo.get( x, y )
または
photo.pixels[ y*image.width + x ]
```

として取得することができます．得られる値の型は color 型です．image.pixels は color 型の配列で，(0,0), (1,0), (2,0),...(photo.width-1,0), (0,1), (1,1), (2,1),...,(photo.width-1,photo.height-1) の画素の色が順番に格納されています.

画素の色を変更するときは, PImage クラスのメンバメソッド set, メンバプロパティ pixels を用いて,

```
photo.set( x, y, color( 255, 0, 0 ) );
または
photo.pixels[ y*photo.width + x ] = color( 255,0,0 );
```

とすると, 座標が (x,y) の点の色が赤色になります．なお, get や set よりも pixels を利用した方が, 高速に動作するスケッチになります．ただし，pixels を利用する場合には前には PImage クラスのメンバメソッドである loadPixels を呼び出し，また変更後には updatePixels を呼び出します.

最初のスケッチを若干変更した下のスケッチは画像の上半分が赤色に塗りつぶされます.

次のスケッチも同じ結果を得られます.

また,

```
color cl = photo.get( x, y );
photo.set( x, y, color( red(cl)/2, green(cl), blue(cl) ) );
```

とすると, 青と緑の成分はそのままに, 赤の成分だけ 50% にすることができます．red メソッドは

```
red( color 型の値 )
```

と書くことで color 型の値の赤成分の値を得ることができるものです．green，blue メソッドも同様です．また，hue，saturation，brightness メソッドはそれぞれ，色相，彩度，明度を得るためのメソッドです.

マウスカーソルの周辺だけ赤成分を落としてみる

上記までのスケッチは読み込んだ画像自体を変換してしまいましたが，今回は一部分だけ，それも，一時的に書き換えるということをして，マウスカーソルの周辺だけグレースケールにするというスケッチにしてみます.

描画ウィンドウに表示されているものをいじくる

上記のことを実現するには，PImage の内容（画像）を書き換えるのではなく，描画ウィンドウに表示されている内容を書き換えるという方法を用います．

そのためには組み込みメソッドの get，set メソッドを利用します．PImage クラスのメンバメソッドである get と set が PImage に読み込まれた画像を操作できたのに対して，組み込みメソッドの get と set は，描画ウィンドウ全体の画像を操作できます．同様に組み込みメソッドの loadImage を呼び出すと, システムプロパティの pixels には描画ウィンドウ全体の画像が読み込まれ，操作できるようになります．

この方法を用いて上半分の色の赤成分を落とすスケッチは次のようになります．for 文の中で画像の幅（photo.width, photo.height）を使っている部分は，描画ウィンドウの幅（width, height）に変更しています．

このスケッチは，draw メソッドが呼び出されるたびに PImage オブジェクトである photo に読み込まれた画像を表示しなおし，その一部分だけ，描画ウィンドウに表示されている内容を直接変更するという動きになっています．

マウスを中心とした四角領域の左上座標と右上座標を求める

目的のスケッチは，draw メソッドが呼び出されるたびに，マウスの周辺だけを書き換えればよいので，まずは，draw が呼び出された時のマウスを中心とした四角領域の左上座標と右下座標を計算します．

マウスの座標は mouseX と mouseY で取得できます．四角領域の１辺の長さを len に入れておくとすると，目的の座標は (mouseX-len/2, mouseY-len/2), (mouseX+len/2, mouseY+len/2) となります．

ただ，この座標が描画ウィンドウの外側になると，指定する座標がウィンドウの外に出てしまいエラーが発生してしまいます．そこで, X 座標は 0 から width の範囲, Y 座標は 0 から height の範囲になるように書きかえる必要があります．すると，たとえば，左上の X 座標は

```
x = mouseX-len/2;
if( x < 0 ) x = 0;
if( x >= width ) x = width-1;
```

と修正する必要があります．これは面倒ですね．このように範囲を制限して値を用いたいときに便利なのが constrain メソッドです．constrain メソッドは

```
constrain( 値，最小値，最大値 )
```

とかくと，最小値と最大値の範囲内であれば値そのものを，範囲外であれば最小値もしくは最大値を生成します．これを用いると，先の左上の X 座標を求める 3 行は

```
x = constrain( mouseX-len/2, 0, width-1 );
```

と書きかえることができます．

マウスを中心とした四角領域だけを書き換える

ここまでで，マウスを中心とした四角領域の範囲を求めることができました．あとは，先ほどのスケッチが (0,0)-(width-1,height/2) の範囲の色を変更していたのを，マウスを中心とした四角領域の範囲に変更するだけです．

色を変更しているのは描画ウィンドウに描かれているものに対してで, draw の中で毎回画像を描きなおしていますので，マウスカーソルが動けば，前回色を変えたところも元に戻って，マウスカーソルがあるところだけ色が変わってみえるようになっているはずです．

演習課題

上記マウスカーソルのまわりだけ赤みを落とすスケッチを提出してください．スケッチ名は半角英数字で 学籍番号 _K10 とし，pde ファイルを WebClass から提出してください．