Media Programing 1 - Processing

- proc.02 - 2009.05.12 hrr

今回のテーマ

配列

オブジェクト指向入門

前回の復習

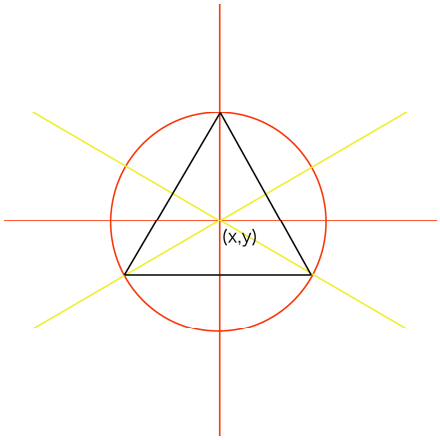
授業内課題０：

正三角形を描画する関数 triangle(float, x, float y,float size) を作成する。

(x,y)を中心として、三角形の中心から各点の距離がsizeの三角形を描く。

マウスをクリックした場所の三角形を描いていく。

正三角形は、以下の図を参考にして独自に描画方法を考えてください。



配列

複数の格納領域を持った変数を名前をつけてまとめたもの。

[] 内の番号（添字）によってアクセスする。

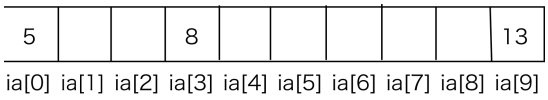
添字の数の分だけ変数が作られて、添字を使う事でプログラム内では

計算した結果に応じた変数アクセスが可能になる。

配列の生成

```
--
   int[] ia = new int[10];
```

```
   // 10個分の整数を大きさをもつ、iaという名前の配列
   ia[0] = 5;
   ia[3] = 8;
   ia[9] = ia[0] + ia[3];

   i1 = 5;
   i2 = 8;



   sum = i1 + i2;

   sum  = 0;
   for ( int i = 0; i < 10 ; i++){
      sum = sum + ia[i];
   }
--
```



練習問題１：

整数の配列を引数として受け取り、配列の中のすべての整数値の平均を返す
関数 average() を作成する。
配列の大きさ（要素の数）は、.lengh を使うと取得できます。
例えば、

```
int[] ary = new int[10];
```

で宣言された配列では、

```
len = ary.length ;
```

とすると、len に配列の大きさ 10 が入る。（ただしlenは整数型の変数）
呼び出し部分は以下のようにする（あくまで例として）

```
--
int N = 10;
int[] ary = new int[N];

void setup(){
    int result;
    for(int i =0; i < N; i++){
        ary[i] = int(random(1000));
        print(ary[i] + " ");
    }
```

```
  result = average(ary); // この関数を作成する
  println(result);
}
```

授業内課題１：

引数で与えられた大きさの菱形を描く関数

rhombus(int x, int y, float size)

を作成する。

マウスをクリックする毎に、その場所に５つまで菱形を描画する。

描かれた菱形はsin波の形状でアニメーションをする。

πの値は、TWO_PI あるいは PI という定数に入っている。

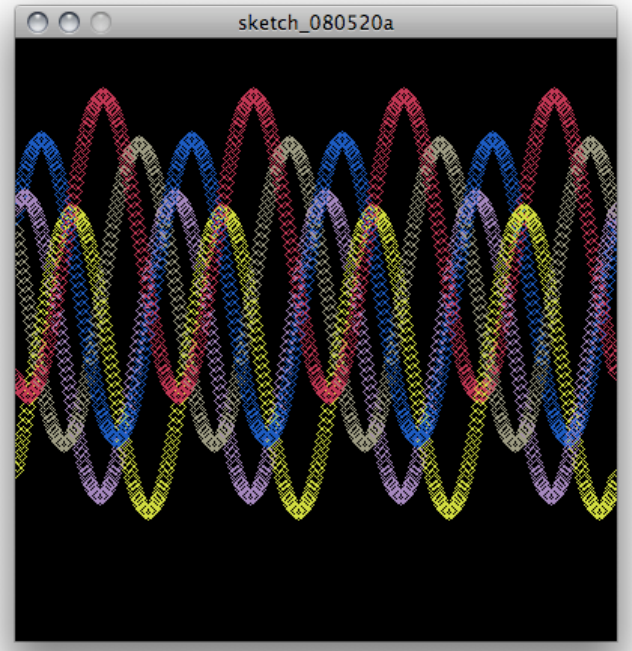
sinの値を求める関数 sin() は、引数としてラジアンの角度を取るので、

PIの値から、角度の変位を作り、sin関数の引数と与えるようにする。

ヒント：まずは、菱形を描く関数を作成する。(x,y) を中心とした菱形の
各頂点の座標を求めれば、line　４つで簡単に実現できる。
その後、draw関数内で、sin波に従った座標値を計算し、
その座標を使って、rhombus関数を呼び出す。
sin波の振幅の大きさは、勝手に決めて良い。
整数の値を浮動小数点に変換するには、float() という関数を使い、
浮動小数点を丸めて整数にするには、round()という関数が使える。

マウスクリックによる処理では、マウスがクリックされる毎に
（mousePressedが呼ばれる毎に）、配列にmouseX, mouseYの値を
代入して保存する。何個描くかも変数の保存しておく。
各菱形について、毎回新しい場所を計算してから rhombus()を呼び
描画処理を行う。
座標値には、x座標を入れる配列、　y座標を入れる配列を用意する。

色も各菱形毎に決めると多少奇麗に見えます。
色は　color[] c = new color[5];　などで配列が作れます。
詳しくはProcessingのreferenceを参照すること。

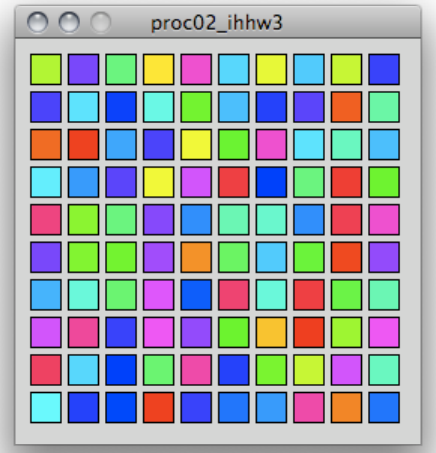


練習問題２：

2次元配列を使って、色が連続的に変化していくアニメーションを作成する。
HSB またはRGBの 1 パラメータを連続的に変化させることで色を変えていく。
例として、プログラムの最初の部分を示すので、setup() および draw()を完成させる。

```
int MX = 10;
int MY = 10;

// color value for each rect
int[][] mtrx = new int[MX][MY];

void setup(){
// 配列の色の初期値の設定
      colorMode(HSB, 100);
}

void draw(){
// 色を変化させるアニメーション
```

```
    //色の指定部分
     fill(color(mtrx[i][j], 100, 100));

}
```



オブジェクト指向

ProcessingはJavaをベースとした言語であるため、オブジェクト指向型の
プログラムが可能である。
オブジェクト指向型言語では、プログラムをオブジェクト（もの）という
概念で扱うことで、より現実に近い形でプログラムを作ることができる。
オブジェクトには、**状態（データ）**とその**振る舞い（関数、メソッド）**が定義されていて、
オブジェクトに対して、「何々する」という**操作を行う**こと（メッセージを送る）ことで、
処理が行われる。
オブジェクト指向言語の特徴としては、
・カプセル化、隠蔽
・継承
・ポリモルフィズム
・メッセージパッシング
などが挙げられる。

例えば、現実の例として、自転車を考える。
自転車には、ハンドルの向き、ギアの段数、ブレーキの状態、ペダルの状態などの
変数がある。

また、ペダルをこぐ、ハンドルを切る、ブレーキをかけるなどの操作がある。
つまり、自転車というオブジェクトをこれらの変数と関数を定義してあげれば
よいことになる。
これらのオブジェクトの定義は、クラスと呼ばれ、クラスを定義することが
オブジェクトの設計図にあたり、オブジェクト自体は設計図から生成された
実体といえる。
オブジェクトを操作するには、そのクラスに定義されている関数を呼ぶことで、
オブジェクトに対しメッセージを送り、対応する処理を行ってもらう。

カプセル化とは、クラスの定義おいて、クラス内だけ必要な変数や関数を
クラスの外から見えないようにすることで、必要のない情報を隠蔽し、
間違いを起こしにくくすることである。
継承とは、元となるクラスから、その子供となるクラス（サブクラス）を定義
することで、似たような機能をもつクラスを作ることを意味する。
例えば、自転車でいえば、２つの車輪とハンドル、ペダルをもつ自転車クラスから
かごの付いたママチャリクラスや、ギアがたくさんあるMTBクラスとかを定義する
ことにあたる。
継承により、既に用意されたクラスを目的に応じて再利用しながら、
新しいクラスを作るなど、再利用性を高めることができると同時に
概念的にもわかりやすいプログラムを作ることができる。
ポリモルフィズムは、概念的には同じでも処理が異なる機能を同じ名前で
実現することである。
たとえば、「足す」という機能は、数値どうしと文字列同士では処理がことなる。
これを同じ足すという名前で行えるような仕組みのことを示す。
実際の例では、Processing用にたくさんの人が作ってくれたライブラリが
たくさんありますが、それらもほとんどはクラスとして用意されています。
それらのクラスを利用する事、つまり、クラスからオブジェクトを生成して
オブジェクトの機能を呼び出す事で、様々なことが可能となります。
さらに、用意されたクラスからサブクラスを作る事で、自分用に機能拡張や
独自のカスタマイズをすることが簡単にできるようになっています。

クラスの定義

もう少し、Processingらしい例を考える。
配列を使って、１０個の点を作ろうとすると、x[10], y[10] という２つの
配列を作って、x座標とy座標を別々に扱う必要がある。
点を扱うのに、座標値と色をまとめて定義すると点をたくさん扱うときには
便利になりそうである。

```
--
  class Ten{
```

```
   float x;
   float y;
   color c;
}
--
```

これがTenというクラスの最初の**定義**である。
クラスからオブジェクトを生成するには、new という命令を使う。
オブジェクトが生成されるときに自動的に実行される関数は、
クラス名と同じ関数で定義される。これを**コンストラクタ**と呼ぶ。

```
--
class Ten{
   float x;
   float y;
   color c;

   Ten(){                  // Tenクラスを生成するときに自動実行
      x = random(200);
      y = random(200);
      c = color(random(255), random(255), random(255));
   }
}

Ten ten;                   // Ten オブジェクトの宣言
void setup(){
   size(200,200);
   ten = new Ten();        // Ten オブジェクトの生成
}
--
```

これで、Tenクラスから、ランダムな座標値と色を持つtenオブジェクトが
生成された。

このままでは、点を描画できないので、Tenクラスに描画部分を加える。
ついでに、座標値もウインドの大きさに合わせるように変更する。

```
--
class Ten{
   float x;
   float y;
   color c;

   Ten(){                  // Tenクラスを生成するときに自動実行
      x = random(width);
```

```
      y = random(height);
      c = color(random(255), random(255), random(255));
    }

    void tenDraw(){
      stroke(c);
      point(x,y);
    }

  }

  Ten ten;                  // Ten オブジェクトの宣言
  void setup(){
    size(200,200);
    ten = new Ten();        // Ten オブジェクトの生成
    ten.tenDraw();
  }

  void draw(){              // 中身は無くて良いが draw()が必要
                            // draw()がないと描画されない。
  }
--
```

これで一つの点が描画できた。
ここまで、座標値、と色という状態を持ち、描画という機能をもった
点を「もの」を定義するTenというクラスがとりあえず完成した。

練習問題３：

Ten クラスを利用して、マウスをクリックする毎にどこかに点を描く
プログラムを作成する。mousePressed() を使う。

練習問題４：

Ten クラスと同じように円というものを定義するEnクラスを作成する。
座標値、色、直径という状態（属性）と描画する機能を記述する。
直径もrandomで良い。

オーバーロード

クラスの定義では、同じ名前であるが引数や返り値が異なる関数を
重複して定義できる。
引数の数や型が一致した関数が自動的に呼ばれる。
コンストラクタでも、この仕組みを利用して引数によって初期値を設定して
オブジェクトの生成（インスタンス化）が行える。

座標を引数として受け取るコンストラクタを定義すると、座標値を指定して
Tenオブジェクトを生成出来る。

```
--
   class Ten{
      float x;
      float y;
      color c;

      Ten(){                // Tenクラスを生成するときに自動実行
         x = random(200);
         y = random(200);
         c = color(random(255), random(255), random(255));
      }

      Ten(float x, float y){  // 座標を渡してTenを生成
         this.x = x;          // 引数のxをクラスのxに代入する
         this.y = y;
         c = color(random(255), random(255), random(255));
      }

      void tenDraw(){
         stroke(c);
         point(x,y);
      }
   }

   Ten ten;                 // Ten オブジェクトの宣言
   void setup(){
      size(200,200);
      ten = new Ten(20.0, 20.0);  // Ten オブジェクトの生成
      ten.tenDraw();
   }
--
```

同じように引数として整数型も取れるようにする。

```
--
   class Ten{
      float x;
      float y;
      color c;

      Ten(){                // Tenクラスを生成するときに自動実行
         x = random(200);
         y = random(200);
         c = color(random(255), random(255), random(255));
      }
```

```
   Ten(float x, float y){  // 座標を渡してTenを生成
      this.x = x;          // 引数のxをクラスのxに代入する
      this.y = y;
      c = color(random(255), random(255), random(255));
   }

   Ten(int x, int y){      // 座標を渡してTenを生成
      this.x = float(x);   // 引数のxをクラスのxに代入する
      this.y = float(y);
      c = color(random(255), random(255), random(255));
   }

   void tenDraw(){
      stroke(c);
      point(x,y);
   }
}

Ten ten;                // Ten オブジェクトの宣言
void setup(){
   size(200,200);
   ten = new Ten(20, 20);     // Ten オブジェクトの生成
   ten.tenDraw();
}

void draw(){
}
```

--

このように、同じ名前で引数の違うコンストラクタを定義すると、
引数に応じて、対応したコンストラクタを呼んでくれる。

同じように、同じ名前で引数の違う関数（メソッド）を定義すると
引数に応じて適切なメソッドを呼んでくれる。

--

```
class Ten{
   float x;
   float y;
   color c;

   Ten(){                  // Tenクラスを生成するときに自動実行
      x = random(width);
      y = random(height);
      c = color(random(255), random(255), random(255));
   }
```

```
   Ten(float x, float y){  // 座標を渡してTenを生成
      this.x = x;          // 引数のxをクラスのxに代入する
      this.y = y;
      c = color(random(255), random(255), random(255));
   }

   Ten(int x, int y){      // 座標を渡してTenを生成
      this.x = float(x);   // 引数のxをクラスのxに代入する
      this.y = float(y);
      c = color(random(255), random(255), random(255));
   }

   void tenDraw(){
      stroke(c);
      point(x,y);
   }

   void tenDraw(int size){    // 整数を引数に渡すと四角を描く
      stroke(c);
      rect(x, y, size, size);
   }
}

Ten ten;                // Ten オブジェクトの宣言
void setup(){
   size(200,200);
   ten = new Ten(20.0, 20.0);  // Ten オブジェクトの生成
   ten.tenDraw(10);               // 四角を描く
}

void draw(){
}
```

--

これで、基本的な点描画のクラスを作ることができた。
次に、配列を使って複数の点を描画できるようにする。

--

```
Ten[] ten = new Ten[10];       // Tenオブジェクトを入れる配列を生成
void setup(){
   size(200, 200);
   for(int i = 0; i < 10; i++){
      ten[i] = new Ten();
   }
   for(int i = 0; i < 10; i++){
      ten[i].draw(5);
```

```
    }
  }
void draw(){}

--
```

授業内課題２：

一つの整数を引数として与えると、その整数の範囲内のランダムな値で
座標を設定して、オブジェクトを生成するコンストラクタを定義しなさい。

```
  Ten(int dsize){
    ....
  }
```

サブクラスの作成

前回のオブジェクト指向の特徴であるクラスの継承機能を使ってサブクラスを
作成する。
サブクラスは、既存のクラスをその属性や機能（変数やメソッド）を引き継いだ上で
必要に応じた拡張を行ったクラスである。

前回のクラスTenを拡張して, Areaクラスを作成してみる。
Areaクラスを作ったので、Tenクラスでは点の描画を行い、四角の描画はAreaクラスに
移動した。描画処理の名前（メソッド名）をわかりやすくするため
paintというメソッドに変更している。

```
--
  class Ten{
    float x;
    float y;
    color c;
    boolean isFill = false;  // 課題で追加した塗りつぶしのon/off

    Ten(){                 // Tenクラスを生成するときに自動実行
      x = random(width);
      y = random(height);
      c = color(random(255), random(255), random(255));
    }

    Ten(float x, float y){  // 座標を渡してTenを生成
      this.x = x;           // 引数のxをクラスのxに代入する
      this.y = y;
      c = color(random(255), random(255), random(255));
    }
```

```
    Ten(int x, int y){      // 座標を渡してTenを生成
      this.x = float(x);    // 引数のxをクラスのxに代入する
      this.y = float(y);
      c = color(random(255), random(255), random(255));
    }

    void paint(){
      stroke(c);
      point(x,y);
    }
  }


// ここからサブクラス Area

  class Area extends Ten {

    float dsize; // エリアなのでサイズが必要

    Area(){
      dsize = 20.0;
      isFill = true;
      c = color(0,0,255);     // 初期値は青
    }

    Area(int x, int y){       // 座標を指定してオブジェクトの生成
      super(x,y);
      dsize = 20.0;
      isFill = true;
      c = color(0,0,255);
    }

    void paint(){
      if(isFill)
        fill(c);
      else
        noFill();
      rect(this.x, this.y, dsize, dsize);
    }
  }


  Area aa1, aa2;              // Tama オブジェクトの宣言
  void setup(){
    size(200,200);
    aa1 = new Area();         // Tama オブジェクトの生成
```

```
    aa1.paint();                // 弾を描く
    aa2 = new Area(100, 100);
    aa2.paint();

  }

  void draw(){
  }
--
```

練習問題５：

Areaにfloat で x,y座標を受け付けられるようにコンストラクタを追加する。

動きを制御するメソッドの追加

既に弾の座標や色などを持つクラスを作ったが、動き方自体もクラスの中に
入れてしまう。
例えば、左右に往復する移動を示すメソッドとして repeatH()という関数を作成すると
draw()内で、aa1.repeatH() を繰り返す事で、aa1オブジェクトの座標などを意識する
ことなく、移動のアニメーションを行う事ができる。
どうせなら、Areaクラスではなく、Ten クラスにメソッドを定義すると良い。

```
--
  class Ten{
    float x;
    float y;
    color c;
    boolean isFill = false;  // 課題で追加した塗りつぶしのon/off
    float speed = 1.0;           // 移動速度
    float dx = 1.0;
    float dy = 1.0;             // 移動する際の差分

    Ten(){                 // Tenクラスを生成するときに自動実行
      x = random(width);
      y = random(height);
      c = color(random(255), random(255), random(255));
      speed = 1.0;
      dx = dy = 1.0;
    }

    Ten(float x, float y){  // 座標を渡してTenを生成
      this.x = x;           // 引数のxをクラスのxに代入する
      this.y = y;
      c = color(random(255), random(255), random(255));
    }
```

```
    Ten(int x, int y){      // 座標を渡してTenを生成
      this.x = float(x);    // 引数のxをクラスのxに代入する
      this.y = float(y);
      c = color(random(255), random(255), random(255));
    }

    void paint(){
      stroke(c);
      point(x,y);
    }

    void repeatH(){           // 水平に行ったり来たり
      x = x + dx;
      if( (x > width) || (x < 0)){
        dx = dx * -1.0;
      }
    }

    void repeatV(){

    }

  }

// ここからサブクラス Area

  class Area extends Ten {

    float dsize; // エリアなのでサイズが必要

    Area(){
      dsize = 20.0;
      isFill = true;
      c = color(0,0,255);     // 青
    }

    Area(int x, int y){       // 座標を指定してオブジェクトの生成
      dsize = 20.0;
      super(x,y);
      isFill = true;
      c = color(0,0,255);
    }

    void paint(){
```

```
      if(isFill)
         fill(c);
      else
         noFill();
      rect(this.x, this.y, dsize, dsize);
   }
}


Area aa, aa2;              // Tama オブジェクトの宣言
void setup(){
   size(200,200);
   background(0);
   aa = new Area();           // Tama オブジェクトの生成
   aa.paint();                // 弾を描く
   aa2 = new Area(100, 100);
   aa2.paint();

}

void draw(){
   background(0);
   aa.paint();
   aa.repeatH();
}
--
```

授業内課題３：
同じようにして垂直に移動するメソッド repeatV() を作成する。

授業内課題４：
このままでは、反射するときに壁にめり込んでしまう。
sizeを考慮することで、めり込まずに反射できるように、
Areaクラスに repeatH, repeatV を再定義（オーバーロード）する。

キーボードによる操作

キーボード入力イベントを扱うための関数（メソッド）として keyPressed() を使う。
マウスと同様にキーボードを押すと、この関数が自動的に呼ばれる。
どのキーが押されたかの情報は、keyCode という変数に入っている。
特殊なキーについては、LEFT, DOWN, RIGHT, UP などのように特別な名前がつけれている。詳しくはreference を参照。

```
--
   void keyPressed(){
      if(key == CODED){
         if(keyCode == LEFT){
            ...
         } else if(keyCode == RIGHT){
            ...
         }
      } else if(key == ' '){
         ...
      }
   }
--
```

授業内課題５：

ウインドウの下の方に、ラケットとなるような長方形を描画し、キーボードの
カーソルキーで移動出来るようにする。
Tenクラスのサブクラスとして、Itaクラスを作成して実現すると良い。
右に移動するメソッド、左に移動するメソッドの追加（Tenクラスに追加しても可）、
板を描画するpaint()のオーバーロード、コンストラクタの作成で、ほぼ完成する。
ウインドウをはみ出ないように移動を制限すること。

テキストの表示

まずは、事前にフォントデータの生成をしておく必要がある。
Processing のメニューの Toolsから Create Font... を選択し、フォントを選ん
で、vlw形式のデータに変換する。
変換すると、そのsketchのdataフォルダに自動的に置かれる。

フォントを使うには、PFont形式の変数にロードする。
　font = loadFont("フォントファイル名");
表示に使うフォントとサイズを指定。
　textFont(font, 24);
最後に表示するには、
　text(文字列、5, 20);　　//と書くと文字列をその座標に表示

```
--
   PFont font;
   int  elapse= 0;

   void setup(){
      size(200,200);
      background(0);
      // setup内で、フォントデータのロード
```

```
      font = loadFont("Arial-BoldItalicMT-48.vlw");
      // 使うフォントと、その大きさの指定
      textFont(font, 12);
      elapse= minute() * 60 + second();
   }

   void draw(){
      int elapsed;

      elapsed = minute() * 60 + second()  - elapse;

      background(0);
      String str = "elapseTime = " + int(elapsed/60) + ":"   + elapsed
%60;                  // 見た目では改行されてますが、改行無し
      text(str, 5, 20);       // 経過時間を表示
}
--
```

課題

練習問題２をAreaクラスを使って書き直す。
クラスの配列を使って、単純にrectで四角を描く代わりに
areaオブジェクトを生成し描画する。
色の変化をするアニメーションを描画しているが、
マウスをクリックすると、初期状態に戻るようにする。
初期状態の保持のため、Areaクラスは適宜変更する必要がある。

ちなみに、クラスの配列は以下のように上でやってます。
２次元配列にするのは、練習問題２と同じです。

```
--
   Area areas[] = new Area[10];   // Areaオブジェクトを入れる配列を生成
   void setup(){
      size(200, 200);
      for(int i = 0; i < 10; i++){
         areas[i] = new Area();
      }
      for(int i = 0; i < 10; i++){
         areas[i].paint();
      }
   }
void draw(){}

--
```