情報メディア基盤ユニット配布資料(2012 年 7 月 10 日分暫定版)

Processing 言語による情報メディア入門

オブジェクト指向入門（その 1）

神奈川工科大学　佐藤尚

最近のプログラミング言語では、オブジェクト指向(object oriented)と呼ばれる機能を持っているものが多くあります。オブジェクト指向は、次の 2 つの仕組みを提供しようとするものです。

1) 複数のデータをまとめて一つに扱う仕組み。

2) 機能拡張を容易に行えるようにする仕組み。

授業では、2 回に分けて、オブジェクト指向の話をします。今回は 1)の仕組みの話です。

そこで、円が上から下に移動するようなプログラムを考えてみます。これはサンプル 12-1 のようになります。どのように、操作するプログラムかはわかりますね。

サンプルプログラム 12-1

```
float xBall;// 円の中心のX座標
float yBall; // 円の中心のY座標
float rBall; // 円の半径
color cBall; // 円の色
void setup(){
  size(400,400);
  smooth();
  colorMode(HSB,359,99,99);
//円の初期状態の決定
  rBall = random(10,20);
  xBall = random(rBall,width-rBall);
  yBall = -rBall;
  cBall = color(random(360),random(50,100),random(50,100));
}
void draw(){
  background(0,0,99);
// 円を移動させる
  yBall += 1;
  if(yBall-rBall > height){
    yBall = -rBall;
  }
// 円を描く
  stroke(cBall);
  fill(cBall);
  ellipse(xBall,yBall,2*rBall,2*rBall);
}
```

今度は 1 つの円ではなく、沢山の円を表示するようなサンプルを考えてみます。円の中心座標の値が入っている xBall や yBall などの値を配列変数にすることで、沢山の円に対する処理を簡単に記述できるようになります。サンプル 12-2 もどのような動作をしているかわかりますね。

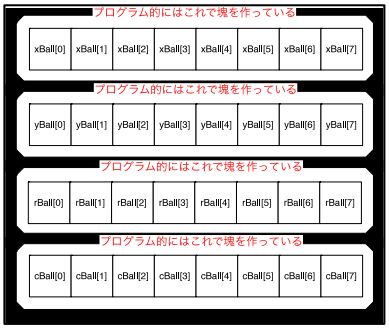
サンプルプログラム 12-2

```
int numberOfBalls = 100;// 円の個数
float[] xBall;// 円の中心のX座標
float[] yBall; // 円の中心のY座標
float[] rBall; // 円の半径
color[] cBall; // 円の色
void setup(){
  size(400,400);
  smooth();
  smooth();
  colorMode(HSB,359,99,99);
// 配列の確保
  xBall = new float[numberOfBalls];
  yBall = new float[numberOfBalls];
  rBall = new float[numberOfBalls];
  cBall = new color[numberOfBalls];
// 円の初期状態の決定
  for(int i=0;i<numberOfBalls;i++){
    rBall[i] = random(10,20);
    xBall[i] = random(rBall[i],width-rBall[i]);
    yBall[i] = -rBall[i];
    cBall[i] = color(random(360),random(50,100),random(50,100));
  }
}
void draw(){
  background(0,0,99);
  for(int i=0;i<numberOfBalls;i++){
// 円を移動させる
    yBall[i] += 1;
    if(yBall[i]-rBall[i] > height){
      yBall[i] = -rBall[i];
    }
// 円を描く
    stroke(cBall[i]);
    fill(cBall[i]);
    ellipse(xBall[i],yBall[i],2*rBall[i],2*rBall[i]);
  }
}
```

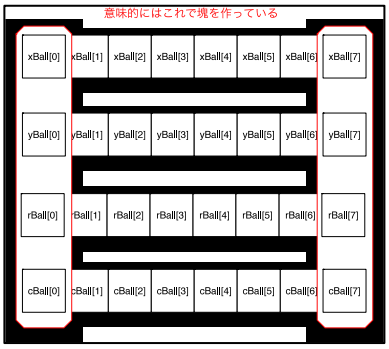
サンプル 12-1 よりは少し複雑になっているように見えますが、沢山の円を処理するために、for 命令による繰り返し処理が付け加わっているだけです。そのために、xBall が xBall[i]などと置き換わっていますが。

サンプル 12-2 では、配列変数 xBall,yBall,rBall,cBall 毎に塊を作っています。しかし、プログラム

での意味的には、xBall[0], yBall[0], rBall[0], cBall[0]は、1 つの円の中心座標、半径、色を保存しています。同じ添え字の番号の値が組を作って、プログラム内での意味を表しています。このサンプルのように、複数の変数が集まって、一つの意味のある情報を表すことがあります。Processing では、このような複数の情報をまとめて、新たなデータ型を作る仕組みが用意されています。それがクラスと呼ばれる仕組みです。最近のプログラミング言語は、この仕組みを持っていることが一般的になっています。



データをまとめる仕組みとしてのクラス



簡単のために、サンプル 12-1 をクラスの仕組みを用いて書き直してみます。サンプル 12-1 でも、xBall, yBall, rBall, cBall が一塊となって、意味のある情報を表しています。どの情報かを区別するために、名前をつける必要があります。クラスを構成する個々の情報（データ）のことをメンバ(member)やメンバ変数と呼び、その名前のことをメンバ名と呼んでいます。そこで、xBall は円の中心の X 座標の値なので xCenter という名前で表すことにします。同様に、yBall は円の中心の Y 座標の値なので yCenter という名前で表すことにします。また、rBall は円の半径なので radius、cBall は円の色なので col とすることにします。また、この 4 つの情報を一塊にしたものを Ball と名付けることにします。この Ball はクラス名と呼ばれます。一般に、クラス名は大文字から始まる名前にします。また、メンバ変数にはどのようなデータを記録するのかを指定するために、データ型を指定する必要があります。今回は、xCenter, yCenter, radius は float 型、color は color 型とします。
つまり、PImage や PFont などはクラスという仕組みで作られたデータ型でした。

クラスの宣言その 1

クラスの宣言の一般形	Ball クラスの宣言

class クラス名 {
　メンバ変数型 0　メンバ名 0;
　メンバ変数型 1　メンバ名 1;
　メンバ変数型 2　メンバ名 02
　　:
　　:
　　:
}
	class Ball {
	　float xCenter;
	　float yCenter;
	　float radius;
	　color col;
	}

このように定義したクラスは通常のデータ型を同じに利用することが出来ます。サンプル 12-1 を

この Ball クラスを使って書きかえたものがサンプル 12-3 です。

サンプルプログラム 12-3

```
class Ball {
  float xCenter; // 円の中心のX座標
  float yCenter; // 円の中心のY座標
  float radius; // 円の半径
  color col;  // 円の色
}
Ball myBall; // Ball型変数の宣言
void setup(){
  size(400,400);
  smooth();
  colorMode(HSB,359,99,99);
//円の初期状態の決定
  myBall = new Ball();
  myBall.radius = random(10,20);
  myBall.xCenter = random(myBall.radius,width- myBall.radius);
  myBall.yCenter = -myBall.radius;
  myBall.col = color(random(360),random(50,100),random(50,100));
}
void draw(){
  background(0,0,99);
// 円を移動させる
  myBall.yCenter += 1;
  if(myBall.yCenter - myBall.radius > height){
    myBall.yCenter = -myBall.radius;
  }
// 円を描く
  stroke(myBall.col);
  fill(myBall.col);
  ellipse(myBall.xCenter,myBall.yCenter,2*myBall.radius,2*myBall.radius);
}
```

Ball クラスの変数を宣言するためには、通常の変数の宣言と同じように「Ball myBall;」などとします。また、実際にデータを保存する場所を作る必要があります。これを行っているのが、「myBall = new Ball();」の部分です。クラスはどのような種類のデータの集まりかを決める鋳型（テンプレート）のようなものです。この鋳型から new 関数を使って、実際にデータを保存する場所を作りだします。この作り出された場所のことをインスタンス(instance)と呼んでいます。鯛焼き器がクラスで、鯛焼きがインスタンス、鯛焼き器を使って鯛焼きを作る作業が new といった感じでしょうか？

姉ヶ崎寧々さんというキャラクタはクラスのようなもので、姉ヶ崎寧々さんは

俺の嫁と思っている人の３DS にはインスタンスとしての“姉ヶ崎寧々さん”が存在しています。
インスタンスのメンバ変数にアクセスするためには、”.”を使います。例えば、m y Ball の xCenter にアクセスするためには、myBall.xCenter などします。その他のメンバの値に対しても、同じようにアクセス出来ます。
サンプル 12-3 は、クラスを使ったプログラム例としては少し不自然なものです。実は、メンバには単なる変数だけなく、関数を持ってくることも出来ます。クラスに付随している関数のことは、メソッド(method)と呼びます。メソッドの定義は、通常の関数の定義と同じです。一つ異なっている点は、class クラス名{ 〜}の中に書くことになっている点です。また、メンバ変数の初期化などはコンストラクタ(constructor)と呼ばれる特殊なメソッド（戻り値無し、名前はクラス名と同じ）を利用します。

クラスの宣言その２

クラスの宣言の一般形	Ball クラスの宣言

class クラス名 {
メンバ変数型０ メンバ名 0;
メンバ変数型１ メンバ名 1;
メンバ変数型２ メンバ名 02
：
：
：
クラス名(){
〜
}
}
	class Ball {
	float xCenter;
	float yCenter;
	float radius;
	color col;
	Ball(){
	〜
	}
	}

コンストラクタを使ってサンプル 12-3 を書きかえたものがサンプル 12-4 です。
この例では、class Ball {〜}内の Ball(){〜}の部分がコンストラクタです。コンストラクタやそのメソッドが付随している class クラス名{〜}の部分でメソッドの定義を書く場合には、直接メンバ名を書けば大丈夫です。

サンプルプログラム 12-4

```
class Ball {
  float xCenter; // 円の中心のX座標
  float yCenter; // 円の中心のY座標
  float radius; // 円の半径
  color col;  // 円の色
  Ball(){
    radius = random(10,20);
    xCenter = random(radius,width- radius);
    yCenter = -radius;
    col = color(random(360),random(50,100),random(50,100));
  }
}

Ball myBall; // Ball型変数の宣言
void setup(){
  size(400,400);
  smooth();
  colorMode(HSB,359,99,99);
//円の初期状態の決定
  myBall = new Ball();
}

void draw(){
  background(0,0,99);
// 円を移動させる,update
  myBall.yCenter += 1;
  if(myBall.yCenter - myBall.radius > height){
    myBall.yCenter = -myBall.radius;
  }
// 円を描く,draw
  stroke(myBall.col);
  fill(myBall.col);
  ellipse(myBall.xCenter,myBall.yCenter,2*myBall.radius,2*myBall.radius);
}
```

サンプル 12-4 の「円を移動させる」や「円を描く」などの部分は、一つのインスタンスだけの情報を利用して作られています。このような場合には、クラスのメソッドとして書くことが一般的です。そこので、このような方針でサンプル 12-4 を書きかえたものがサンプル 12-5 です。クラスに付随するメソッドを呼び出す場合にも、メンバ変数と同じように”.”を使って使用します。

クラスの宣言その 3

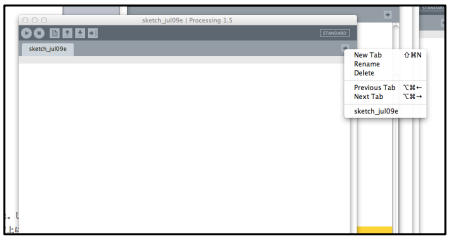
クラスの宣言の一般形	Ball クラスの宣言
class クラス名 { 　メンバ変数型 0　メンバ名 0; 　メンバ変数型 1　メンバ名 1; 　メンバ変数型 2　メンバ名 02	class Ball { 　float xCenter; 　float yCenter; 　float radius;

:
:
:
クラス名(){
～
}
戻り値型０　メソッド名０(仮引数の並び){
～
}
戻り値型１　メソッド名１(仮引数の並び){
～
}
:
}
	color col;
	Ball(){
	～
	}
	void update(){
	～
	}
	void draw(){
	～
	}
	}

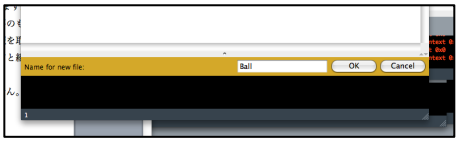
サンプルプログラム 12-5

```
class Ball {
  float xCenter; // 円の中心のX座標
  float yCenter; // 円の中心のY座標
  float radius; // 円の半径
  color col;  // 円の色
// コンストラクタの定義
  Ball(){
    radius = random(10,20);
    xCenter = random(radius,width- radius);
    yCenter = -radius;
    col = color(random(360),random(50,100),random(50,100));
  }
// メソッドの定義
  void update(){
    yCenter += 1;
    if(yCenter - radius > height){
      yCenter = -radius;
    }
  }
  void draw(){
    stroke(col);
    fill(col);
    ellipse(xCenter,yCenter,2*radius,2*radius);
  }
}

Ball myBall; // Ball型変数の宣言
```

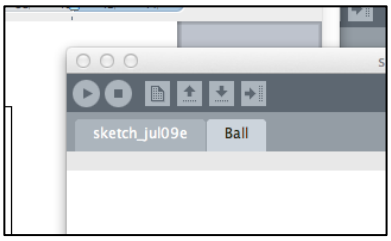
```
void setup(){
  size(400,400);
  smooth();
  colorMode(HSB,359,99,99);
//円の初期状態の決定
  myBall = new Ball();
}

void draw(){
  background(0,0,99);
// 円を移動させる,update
  myBall.update(); // myBallのupdateメソッドの呼び出し
// 円を描く,draw
  myBall.draw();  // myBallのdrawメソッドの呼び出し
}
```



今までのクラスを使ったサンプルでは、一つのタブの中に全てのプログラムを書いていました。しかし、通常はクラス毎に別々のタブに記述します。新たのタブを作るためには、次の様に行います。



1. ウインドウの右上にある矢印状のボタンを押します。
2. すると、メニューが出てきますので、「New Tab」を選択します。
3. そして、新たに作るタブの名前を入力し、OK ボタンを押します。
4. 新しいタブが作られます。
5. タブの名前は、クラスの名前と同じにするのが一般的です。



Ball クラスを使って、サンプル 12-2 を書きかえてみます。この結果がサンプル 12-6 です。

サンプルプログラム 12-6

Ballクラスのタブ

```
class Ball {
  float xCenter; // 円の中心のX座標
  float yCenter; // 円の中心のY座標
  float radius; // 円の半径
  color col;  // 円の色
// コンストラクタの定義
  Ball(){
    radius = random(10,20);
    xCenter = random(radius,width- radius);
    yCenter = -radius;
    col = color(random(360),random(50,100),random(50,100));
  }
// メソッドの定義
  void update(){
    yCenter += 1;
    if(yCenter - radius > height){
      yCenter = -radius;
    }
  }
  void draw(){
    stroke(col);
    fill(col);
    ellipse(xCenter,yCenter,2*radius,2*radius);
  }
}
```

メインのタブ

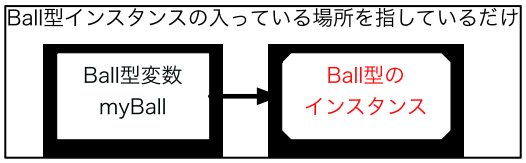
```
int numberOfBalls=100;
Ball[] myBalls; // Ball型変数の宣言
void setup(){
  size(400,400);
  smooth();
  colorMode(HSB,359,99,99);
// 配列の確保
  myBalls = new Ball[numberOfBalls];
  for(int i=0;i<numberOfBalls;i++){
    myBalls[i] = new Ball();
  }
}
void draw(){
  background(0,0,99);
  for(int i=0;i<numberOfBalls;i++){
    myBalls[i].update(); // myBalls[i]のupdateメソッドの呼び出し
    myBalls[i].draw();  // myBall[i]のdrawメソッドの呼び出し
  }
}
```

このようにクラスを利用してプログラムを作成すると、プログラムの見通しが良くなります。
また、コンストラクタにも引数を渡すことが出来ます。

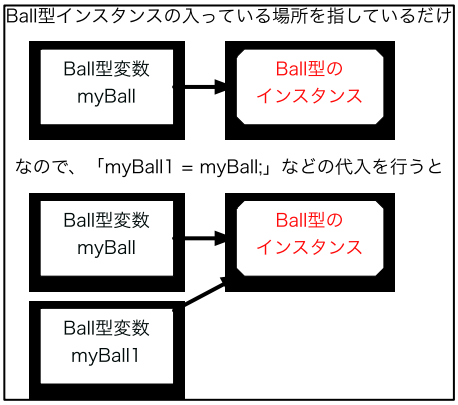
クラスの宣言その４

クラスの宣言の一般形	Ball クラスの宣言

class クラス名 {
　メンバ変数型 0　メンバ名 0;
　メンバ変数型 1　メンバ名 1;
　メンバ変数型 2　メンバ名 02
　　:
　　:
　　:
　クラス名(){
　　～
　}
　戻り値型 0　メソッド名 0(仮引数の並び){
　　～
　}
　戻り値型 1　メソッド名 1(仮引数の並び){
　　～
　}
　　:
}
	class Ball {
	　float xCenter;
	　float yCenter;
	　float radius;
	　color col;
	　Ball(){
	　　～
	　}
	　void update(){
	　　～
	　}
	　void draw(){
	　　～
	　}
	}



今まで、説明をサボってきましたが、クラス型の変数は、そのクラスのインスタンスへの参照となっています。普通は気にしなくても大丈夫ですが、時々問題が起きることがあります。つまり、Ball 型変数同士の代入を行っても、変数が指しているインスタンスの情報そのものが複製される訳ではありません。インスタンスの情報もコピーするような代入を浅いコピー(shallow c opy)と呼んでいます。メンバ変数の型が何らかのクラス型になっている場合には、単に参照がコピーされるだけです。Processing でも、浅いコピーを実現するための clone メソッドが用意されています。逆に、完全なコピーを作るような代入を深いコピー(deep copy)と呼ばれています。深いコピーを実現するためには、複製を作るために時間



がかかるので、どうしても深いコピーを使いたいときだけ、利用します。ちょっと難しい話で、詳しくは触れません。

サンプル 12-6 をまねして、円の代わりに正方形を落ちてくるようなプログラムを作成してみます。サンプル 12-7 を見るとわかるように、クラスを使ってプログラムを作成しておくと、どの部分を変更すれば良いかが見やすくなっていることがわかると思います。

サンプルプログラム 12-7

Squareクラスのタブ

```
class Square {
  float xCenter; // 中心のX座標
  float yCenter; // 中心のY座標
  float length; // 一辺の長さ
  color col;  // 色
// コンストラクタの定義
  Square(){
    length = random(10,20);
    xCenter = random(length/2,width- length/2);
    yCenter = -length/2;
    col = color(random(360),random(50,100),random(50,100));
  }
// メソッドの定義
  void update(){
    yCenter += 1;
    if(yCenter - length/2 > height){
      yCenter = -length/2;
    }
  }
  void draw(){
    rectMode(CENTER);
    stroke(col);
    fill(col);
    rect(xCenter,yCenter,length,length);
  }
}
```

メインのタブ

```
int numberOfSquares=100;
Square[] mySquares; // Square型変数の宣言
void setup(){
  size(400,400);
  smooth();
  colorMode(HSB,359,99,99);
// 配列の確保
  mySquares = new Square[numberOfSquares];
  for(int i=0;i<numberOfSquares;i++){
    mySquares[i] = new Square();
  }
}
void draw(){
  background(0,0,99);
  for(int i=0;i<numberOfSquares;i++){
    mySquares[i].update(); // mySquares[i]のupdateメソッドの呼び出し
    mySquares[i].draw();  // mySquare[i]のdrawメソッドの呼び出し
  }
}
```