



---

# プログラミング演習2 クラス (1)

---

中村, 高橋, 小林, 橋本

# 本日の流れ

---



- 13:30-13:50 宿題の解説
- 13:50-14:50 座学 + 演習
- 14:50-16:30 課題に取り組む
- 16:30が課題の提出期限
- 16:30-17:00 課題の解説

# 宿題について



- Google Drive上のファイルのタイムスタンプが13:00以降のものはチェック対象外です
- フォームの申請については、採点に利用しますので、申請忘れがある場合は、講義中にやるようにしてください

# 宿題1-1: hw\_boundA11



- 5個の○と, 4個のxと, 3個の△が画面内を動き回るプログラムを作成せよ
  - ただし, その速度はx、y方向それぞれ-5~5の実数値とせよ
  - また, ○は壁で跳ね返り, xと△は跳ね返らずに反対側から出てくるようにせよ

# 宿題1-2: hw\_abababab



- aとbそれぞれに1から9までの数字を当てはめたときにできる8桁の数字について、約数の数が最小になるaとbの組み合わせと約数の数をプログラムで求め標準出力せよ
- ただし、aとbが同じ数字でも良い
  - 例： a=3, b=5のときは、35353535の約数の数を求めることになる
- 出典：数学検定 準1級の試験より
  - 大学受験数学で解けますので、数学好きな人はチャレンジしてみたら良いですよ！

# 宿題1-3: hw\_changeChara



- マウスカーソルの下に何らかのキャラクタを描くプログラム `hw_changeChara` を、`displayActive`、`displayInactive`を完成させることで作成せよ
  - `hw_changeChara.zip` を利用せよ
  - ただし、それぞれの関数内では`mouseX`、`mouseY`を利用しないようにせよ
- ただし、`displayActive`は活性状態、`displayInactive`は非活性状態の見た目となるようにせよ（わかればなんとなくでOK）
- マウスを動かすとキャラクタが動き、マウスのボタンが押されているときに非活性状態となり、そうでないときには活性状態となる

# 配列って使いにくい！？



- オセロの盤面などのように、決まった数しかないものに使うのは便利！
- でも、最初に数が決まってることって少なくなくて、決めるのが無理なこと多くない？
  - サークルの名簿（10人？100人？それ以上？）
  - お絵かきツールで描く線の数
  - じゃんけんの勝敗の記録
  - ハイスコアの記録
  - 素数のリストなどなど

# 配列の数の定義面倒！



- 配列の大きさを事前に指定しておくのは柔軟性がなくて面倒のでもっと手軽に扱いたい！
  - メンバーリスト（下の例だと40人上限）

```
String[] members = new String[40];
```

- ゲームのスコアの記録（下だと100個まで！）

```
float[] score = new float[100];
```

- 位置情報を管理する！（下だと1000個まで！）

```
int[] posX = new int[1000];  
int[] posY = new int[1000];
```



# 配列で...



- 40人分のメンバー用配列作ったぞ！

```
String[] members = new String[40];
```

- 1人追加された！！  
– どうする？

# ArrayList型



- ArrayList型は、いくつでも追加可能で色々なものを格納できるクラス

(例) `ArrayList list = new ArrayList();` で定義

- ArrayListの要素数を取得

```
list.size();
```

- ArrayListに対するaddで要素を追加

```
list.add( value ); // valueを追加
```

- ArrayListに対するremoveで要素を削除

```
list.remove( index ); // index番目を削除
```

- ArrayListに対するgetで要素を取得

```
list.get( index ); // index番目のオブジェクトを取得
```

# ArrayList



- ArrayList型の定義
- add()でリストに追加
- size()でリストのアイテム数を取得
- get(n)でリストからn番目のアイテム（オブジェクト）を取得
  - get(0)なら0番目
  - get(1)なら1番目
- remove(n)でリストからn番目を削除

```
ArrayList member = new ArrayList();

member.add("Nakamura");
member.add("Kobayashi");
member.add("Hashimoto");
member.add("Takahashi");

for(int i=0; i<list.size(); i++){
    println(member.get(i));
}

member.remove(0);
member.add("Miyashita");
member.add("Igarashi");

for(int i=0; i<list.size(); i++){
    println(member.get(i));
}
```

# ArrayListの定義



- 一応こちらでOK ( getのとき要キャスト )

```
ArrayList list = new ArrayList();
```

- でも, こんな感じで型 ( クラス ) もセットにしたほうが後々わかりやすいので推奨
  - <> をGenericsと言います
  - intやfloatはなく, Integer, Floatと書きます

```
ArrayList<型> list = new ArrayList<型>();
```

```
ArrayList<Integer> list = new ArrayList<Integer>();  
ArrayList<Float> list = new ArrayList<Float>();  
ArrayList<String> member = new ArrayList<String>();
```

# ArrayList<型>



```
ArrayList<String> member = new ArrayList<String>();

member.add("Nakamura");
member.add("Kobayashi");
member.add("Hashimoto");
member.add("Takahashi");

for(int i=0; i<list.size(); i++){
    println(member.get(i));
}

member.remove(0);
member.add("Miyashita");
member.add("Igarashi");

for(int i=0; i<list.size(); i++){
    println(member.get(i));
}
```

# ArrayList<型>



- サイコロを1000回ふって、最後の10回分を表示するには？

```
ArrayList<Integer> history = new ArrayList<Integer>();

for(int i=0; i<1000; i++)
{
    int dice = (int)random(1, 7);
    history.add( dice );
}

for(int i=history.size()-10; i<history.size(); i++){
    println(history.get(i));
}
```

# ArrayList<型>



- サイコロを1000回ふって、最後の10回分を表示するには？

```
ArrayList history = new ArrayList ();

for(int i=0; i<1000; i++)
{
    int dice = (int)random(1, 7);
    history.add( dice );
}

for(int i=history.size()-10; i<history.size(); i++){
    int num = (int)history.get(i);
    println( num );
}
```

# 追加/削除が自由！



- どうやって実現しているんだろう？という  
ことを気にせずに，配列として使うことが  
できる！
  - add( 末尾に追加したいもの )
  - remove( 削除したいアイテム番号 )
  - get( 取得したいアイテムの番号 )
  - size()
  - を使えば，大きさを気にせず配列みたいなこと  
を実現することができる！



# 演習: diceHistory



- マウスをクリックするたびにサイコロを振り (ランダムに1~6を生成し), そこまでの記録を全てArrayListで管理し, クリックのたびにそれまでのすべての目を表示せよ

```
4,6,1,5,1,3,3,6,5,1,6,3,6,4,4  
4,6,1,5,1,3,3,6,5,1,6,3,6,4,4,3  
4,6,1,5,1,3,3,6,5,1,6,3,6,4,4,3,4  
4,6,1,5,1,3,3,6,5,1,6,3,6,4,4,3,4,3  
4,6,1,5,1,3,3,6,5,1,6,3,6,4,4,3,4,3,4
```

# (おまけ) 拡張for文



- ArrayList型の定義で、型をちゃんと指定しておくと、拡張for文を使うことができる

```
ArrayList<String> member = new ArrayList<String>();  
  
member.add("Nakamura");  
member.add("Kobayashi");  
member.add("Hashimoto");  
member.add("Takahashi");  
  
for(String name: member)  
{  
    println(name);  
}
```

# PImage 画像を表示する



- 画像を読み込んだり, 格納したり, 大きさを取得したり, みんな大好きフィルタ (エフェクト) をかけたり, 保存したりと行ったことはどうやる?



# PImage型 / 画像型

- 画像を格納および描画するクラス
  - .width や .height で画像の縦横のサイズ取得
  - .resize() で画像サイズを変更可能
  - .save() で画像を保存可能
  - .filter() で各種フィルタをかけることが可能

```
PImage img;  
size(400, 400);  
img = loadImage("画像ファイル名");  
img.filter(BLUR, 6);  
image(img, 0, 0);
```

フィルタ例 ()内はオプション  
THRESHOLD (0-1.0)  
GRAY  
OPAQUE  
INVERT  
POSTERIZE (2-255)  
BLUR (1以上, 半径)  
ERODE  
DILATE



- 準備

- 音楽再生ライブラリ（便利関数群）の読み込み
  - `import ddf.minim.*;`
- Minimの変数を初期化し, 初期化
  - **Minim はサウンド関係を扱うクラス**
  - `Minim minim; // をグローバル変数として用意`
  - `minim.loadFile("ファイル名");` でファイルを読み込む
- AudioPlayer を初期化し loadFile の結果を受け取る
  - **AudioPlayer は音声 / 音楽の再生を司るクラス**
  - `AudioPlayer bgm = minim.loadFile("ファイル名");`
  - `bgm.loop(); // 繰り返し再生`
  - `bgm.stop(); // 停止`



- 終了
  - AudioPlayer の終了
    - `bgm.close();`
  - Minim の終了
    - `minim.stop();`
  - 親の終了 ( `stop()` ) の中では最後に必ず書く )
    - `super.stop();`

# 音楽の再生



```
import ddf.minim.*;
Minim minim;      // Minim型変数であるminimの宣言
AudioPlayer bgm;  // BGM格納用の変数
int x = 0;
int vx = 4;

void setup(){
  size(400, 400);
  minim = new Minim(this); //初期化
  bgm = minim.loadFile("bgm.mp3"); //mp3ファイルを指定する
  bgm.play(); //bgmを再生
}

void draw(){
  background(0, 0, 0);
  x = x + vx;
  ellipse(x, 200, 20, 20);
}

void stop(){
  bgm.close(); //サウンドデータを終了
  minim.stop();
  super.stop(); // 必須
}
```



- 準備

- 音楽再生ライブラリ（便利関数群）の読み込み

- `import ddf.minim.*;`

- Minim の変数を定義し，初期化

- AudioSnippet を初期化し `minim.loadSnippet` の結果を受け取る

- AudioSnippet は音声の再生を司るクラス

- `AudioSnippet clash`

- `= minim.loadSnippet("ファイル名");`

- `clash` は変数名．他の名前でもOK





- 再生処理：
  - `clash.play()`; // `clash`に格納された音の再生
  - `clash.rewind()`; // `clash`に格納された音の巻き戻し
  
- 終了：
  - `AudioSnippet` の終了
    - `clash.close()`;
  - `Minim` の終了
    - `minim.stop()`;
  - 親クラスの終了 ( `stop()` の中では最後に必ず書く )
    - `super.stop()`;



効

```
import ddf.minim.*;
Minim minim;          // Minim型変数であるminimの宣言
AudioSnippet crash;  // 衝突サウンド格納用の変数
int x = 0, vx = 4;

void setup(){
  size(400, 400);
  minim = new Minim(this); //初期化
  crash = minim.loadSnippet("clash.mp3"); //mp3ファイルを指定する
}

void draw(){
  background(0, 0, 0);
  x = x + vx;
  if(x == width){
    crash.rewind();
    crash.play(); //再生
  }
  ellipse(x, 200, 20, 20);
}

void stop(){
  crash.close(); // サウンドデータを終了
  minim.stop();
  super.stop(); // 必須
}
```



- String 型は，文字列を扱うためのクラス
  - 「中村聡史」「明治大学 総合数理学部」
- 文字列を扱うにはこういった機能が必要？
  - 文字列の長さを取得する
  - 文字列にある文字が含まれているかを調べる
  - 文字列を部分的に置き換える
  - 文字列が一致しているか調べる
  - n文字目の文字を取得する
  - などなど

# Stringクラスのメソッド



- `charAt(num)`;            `num`文字目の文字を返す（0から始まる）
- `indexOf(文字列)`;        入力された文字列が何文字目か？
- `indexOf(文字列, n)`; `n`文字目以降で入力された文字列が何文字目か？
- `length()`;                入力された文字の文字数を返す
- `substring( x )`;        `x`文字目から最後までを出力
- `substring( x, y )`; `x`文字目から`y-1`文字目までを出力
- `toLowerCase()`;        全てを小文字に変換する
- `toUpperCase()`;        全てを大文字に変換する
- `replace( 文字列A, 文字列B )`;  
  – 文字列Aを文字列Bに変更する
- `split( 文字列 )`;        文字列を分割

<http://processing.org/reference/String.html>

<http://docs.oracle.com/javase/6/docs/api/java/lang/String.html>

```
String str = "Department of Frontier Media Science (FMS), IMS, Meiji University";

println( str.length() );
println( str.charAt( 11 ) );
println( str.indexOf("F") );
println( str.indexOf("S") );
println( str.indexOf("Meiji") );
println( str.substring( str.indexOf("Meiji") ) );
println( str.toLowerCase() );
println( str.toUpperCase() );
```



```
65
o
14
29
49
Meiji University
department of frontier media science (fms), ims, meiji university
DEPARTMENT OF FRONTIER MEDIA SCIENCE (FMS), IMS, MEIJI UNIVERSITY
```



```
String str = "Department of Frontier Media Science (FMS), IMS, Meiji University";

String [] ret = str.split( " " );
println( ret.length );
int i=0;
while(i < ret.length)
{
    println( ret[i] );
    i++;
}
```



9  
Department  
of  
Frontier  
Media  
Science  
(FMS),  
IMS,  
Meiji  
University



- 下記の結果は？

```
println( str.indexOf("Media") );  
println( str.toLowerCase().indexOf("media") );  
println( str.substring(  
    str.indexOf("Meiji") ).length() );
```

- Frontier Media Science を出力するには？

```
str.substring(           ,           );
```

# 塩基配列を扱ってみよう



- DNA塩基配列を文字列処理してみよう！
  - 塩基配列は， ATGCの4文字からなるもの
  - 下記をダウンロードしよう！
    - basic\_Covid19Genome.pde
    - SARS-CoV-2（新型コロナウイルス）の塩基配列
- 演習
  - 塩基配列の長さを出力しよう
  - 最初の100文字を出力しよう
  - 最初にAが4つ連続しているのは何文字目か？
    - <https://www.ncbi.nlm.nih.gov/nuccore/MN908947>



# これからやること



- こんな感じの中身（どんな感じで管理しているか）はよくわからないけど，メソッドを使うことで手軽に使える仕組みを実現していく！
  - 「クラス」という知識を習得する
    - プログラムがよりわかりやすくなる
    - カプセル化などにより問題の切り分けができる
    - 使い回しができる！
    - 他人とプログラムを共有するときに，内部を気にせずに利用することができる
- などなどのメリット

# 例：動物園を実現する



- 猫, 犬, 猿, 象, 熊がいる動物園を再現する
  - 表示場所に関する情報  
猫(catX, catY), 犬(dogX, dogY), 猿(monkeyX, monkeyY),  
象(elephantX, elephantY), 熊(bearX, bearY)
  - それぞれの動物の描画  
drawCat(), drawDog(), drawMonkey(), drawElephant(), drawBear();
  - それぞれの動物の移動  
moveCat(), moveDog(), moveMonkey(), moveElephant(), moveBear();
  - それぞれの動物の睡眠  
sleepCat(), sleepDog(), sleepMonkey(), sleepElephant(),  
sleepBear();
  - などを用意し, それぞれをプログラムの内部から呼び出す必要あり

かなり面倒で混乱のもと

# 愚直に書くとどうなる？



- ball1, ball2, ball3 という3つのボールが動き回るような世界をプログラミングせよ
- [http://nkmr.io/lecture/2020/ball\\_original.pde](http://nkmr.io/lecture/2020/ball_original.pde) をダウンロードして利用しよう

# 動きは動物に任せたい



- 猫, 犬, 猿, 象, 熊を定義
  - それぞれの座標は意識したくない
    - `cat.x`, `cat.y`, `dog.x`, `dog.y`, `monkey.x`, `monkey.y`, ...
    - 内部で適切に処理してもらう
  - 描画はシンプルにしたい
    - `cat.draw()`, `dog.draw()`, `monkey.draw()`, `elephant.draw()`, ...
  - 移動もシンプルにしたい
    - `cat.move()`, `dog.move()`, `monkey.move()`, `elephant.move()`, ...
  - 睡眠も任せてしまう
    - `cat.sleep()`, `dog.sleep()`, `monkey.sleep()`, `elephant.sleep()`, ...

すべてを **XXXX . 機能** という形に！

オブジェクト指向（クラス）

# オブジェクト指向とは



- ざっくり説明すると、色々な値や機能をもつもの  
(例) シューティングゲーム上の敵

- 現在位置 (X座標, Y座標)
- 何らかの移動機能 (移動の関数)
- 何らかの描画機能 (描画の関数)

をもっており、プログラムから移動しろ、描画しろと命令を送るだけで、その中身 (変数の状態) や処理がどうなっているかを意識せずに利用可能

- 他人が何をどう考え実行するかを気にせず、「～をやっておいて」とお願いする感覚

# たとえば



- 人間というクラスを定義する
  - 人間には名前という変数
  - 現在地という場所に関する変数
  - 年齢という変数などがある
- 人間には下記のメソッドがある
  - 移動する
  - 食べる
  - 喋る
  - 聞くなど

# たとえば



- 人間というクラスを使って、「小松」「宮下」というものを具現化する（インスタンス化する）
- 「小松」や「宮下」が持つ機能をインスタンスメソッドと呼ぶ
- 「小松」や「宮下」の情報（変数）をインスタンス変数と呼ぶ（人間に共通のものは静的変数）
- 人間に共通のメソッド（なんという生き物か？というメソッド）を静的メソッドと呼ぶ

Processingには静的メソッド / 静的変数はない

# 変数を直接触らない



- メソッドだけで色々と制御できるように！
- 時計を実現することを考える
  - なかの制御系を直接動かさない
  - 時・分・秒を変更するために、ダイヤルを引っ張り出し、ダイヤルを回す
  - ボタンを押すことでアラームのセットなど

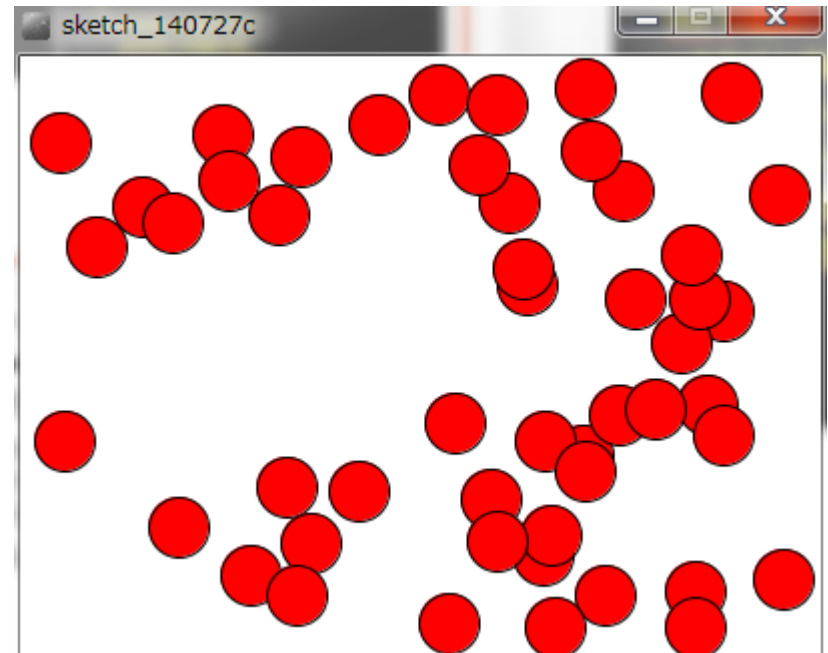
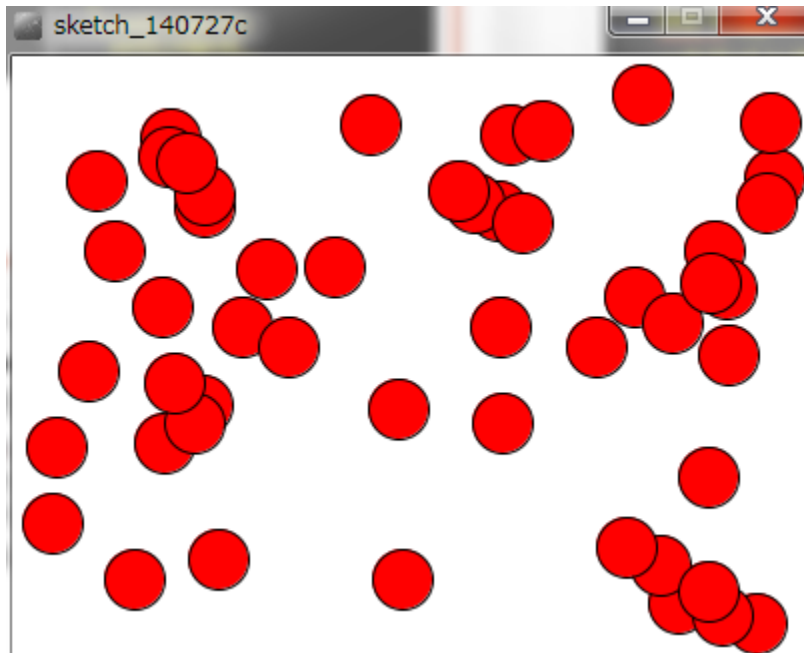


# 端で跳ね返る50個のボール



400x300の画面内にX座標Y座標ともにランダムに配置された50個の円を描き，それぞれのX，Y方向へのスピードに応じて移動するようにせよ

- posX, posY という座標の配列と, speedX, speedY というXおよびY方向の速度をもつ配列を導入！



```
float [] posX = new float[50];
float [] posY = new float[50];
float [] speedX = new float[50];
float [] speedY = new float[50];
```

```
void setup()
```

```
{
    size(800, 600);
    int i=0;
    while(i < 50)
    {
        posX[i] = random(0, width);
        posY[i] = random(0, height);
        speedX[i] = random(-5, 5);
        speedY[i] = random(-5, 5);
        i++;
    }
}
```

```
void draw()
```

```
{
    background( 255 );
    fill( 255, 0, 0 );
    int i=0;
    while(i < 50){
        posX[i] += speedX[i];
        posY[i] += speedY[i];
        if(posX[i] > width){
            posX[i] = width * 2 - posX[i];
            speedX[i] = -speedX[i];
        }
        if(posX[i] < 0){
            posX[i] = -posX[i];
            speedX[i] = -speedX[i];
        }
        if(posY[i] > height){
            posY[i] = height * 2 - posY[i];
            speedY[i] = -speedY[i];
        }
        if(posY[i] < 0){
            posY[i] = -posY[i];
            speedY[i] = -speedY[i];
        }
        ellipse( posX[i], posY[i], 30, 30 );
        i++;
    }
}
```

# 問題ないけれど . . .



- 変数（配列）が沢山でちょっと分かりにくい
  - `posX[n]` と `posY[n]` と `speedX[n]` と `speedY[n]` がセットだけれど，それぞれ配列として独立しているし，なんだかよくわからない
  - 条件分岐もいろいろあって `draw` 内が分かりにくい

# クラスの定義



- Ball クラスを定義

```
class クラス名 {
```

クラスの諸要素に関する定義

```
}
```

```
class Ball {  
  
}
```

```
class Human {  
  
}
```

```
class Animal {  
  
}
```

# クラスの定義



人間 {

現在いる場所

名前

身長

年齢

移動する

飲食する

喋る

}

インスタンス変数

インスタンスメソッド

# クラスの定義



```
class Human {
```

```
    int lon, lat;
```

```
    String name;
```

```
    float bodyHeight;
```

```
    int age;
```

```
    void move();
```

```
    void eat();
```

```
    String say();
```

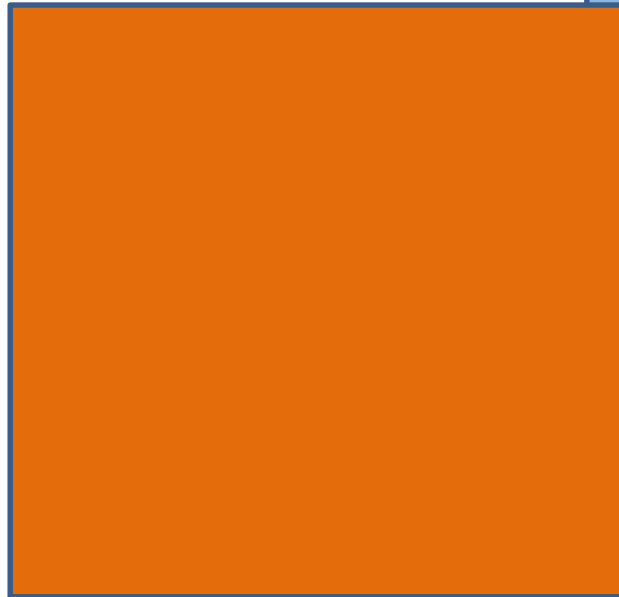
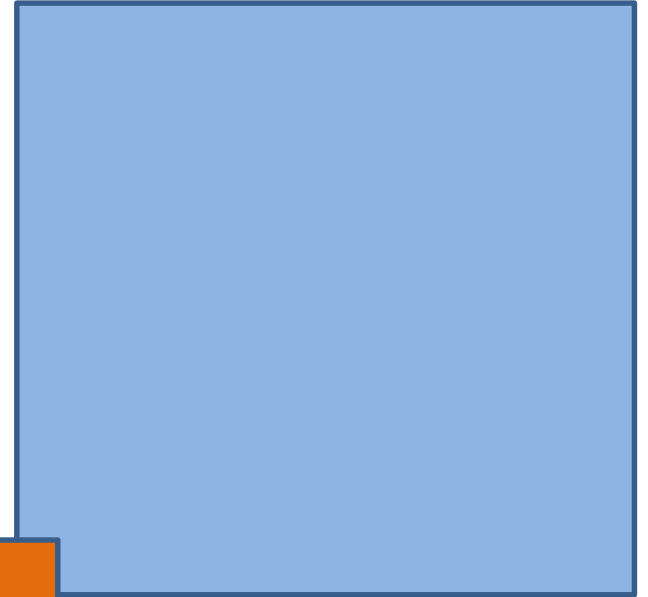
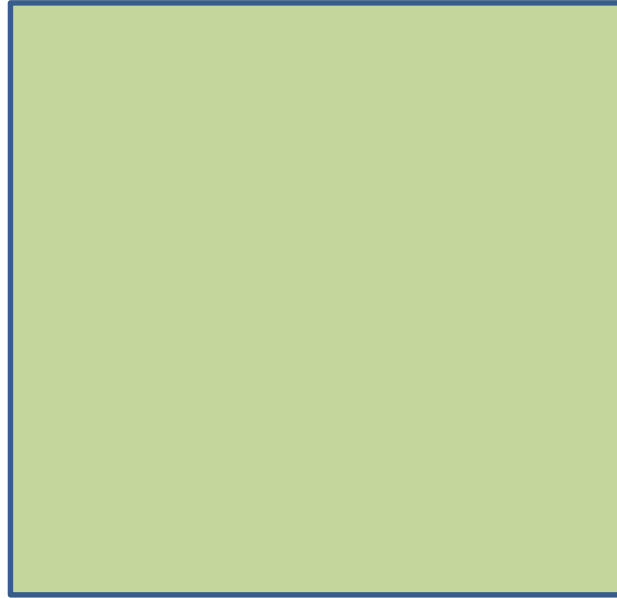
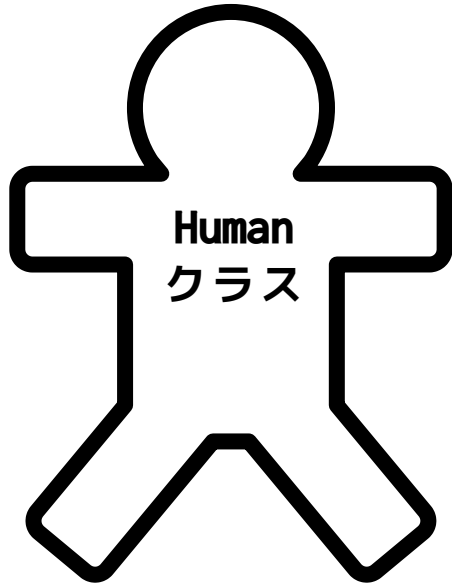
```
}
```

インスタンス変数

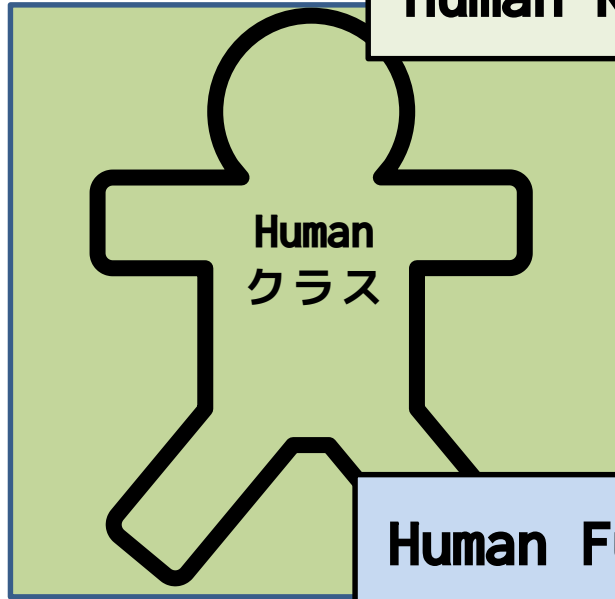
Human  
クラス

インスタンスメソッド

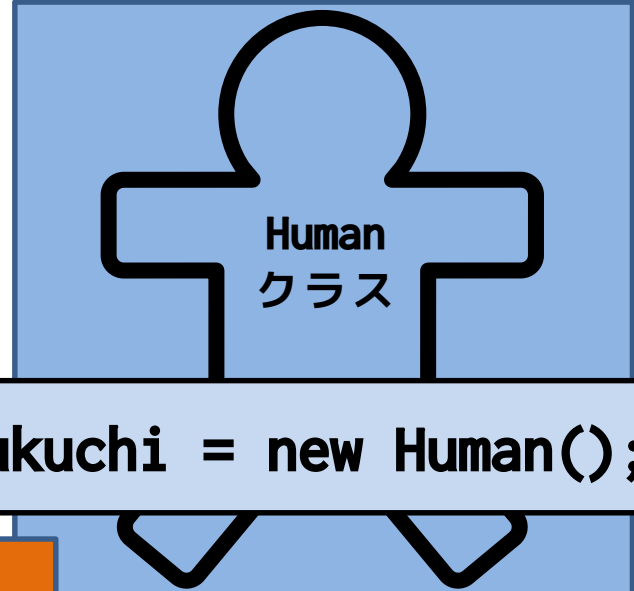
# インスタンス化



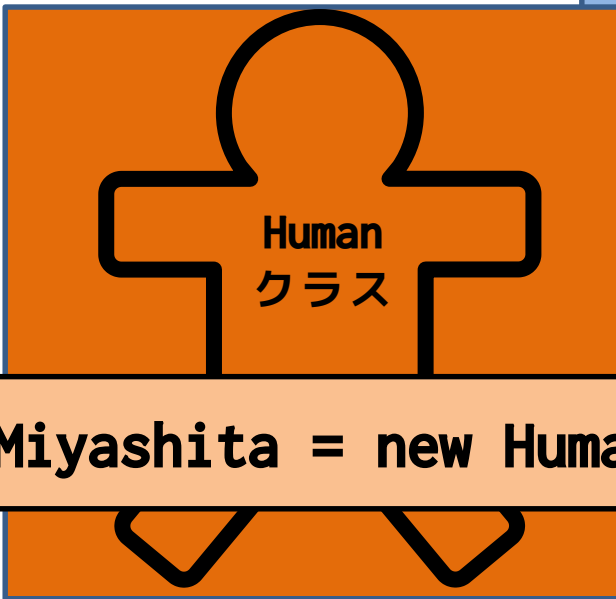
# インスタンス化



```
Human Komatsu = new Human();
```



```
Human Fukuchi = new Human();
```



```
Human Miyashita = new Human();
```



# インスタンス化



```
Human Komatsu = new Human();
```

インスタンス  
Komatsu

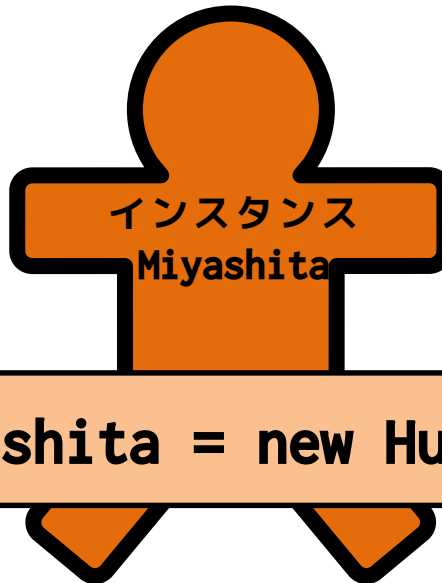
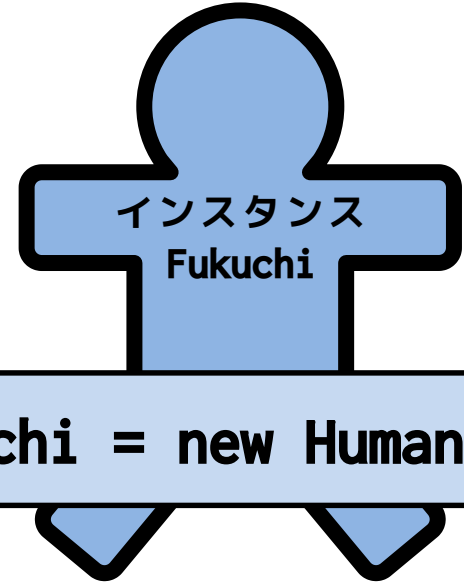
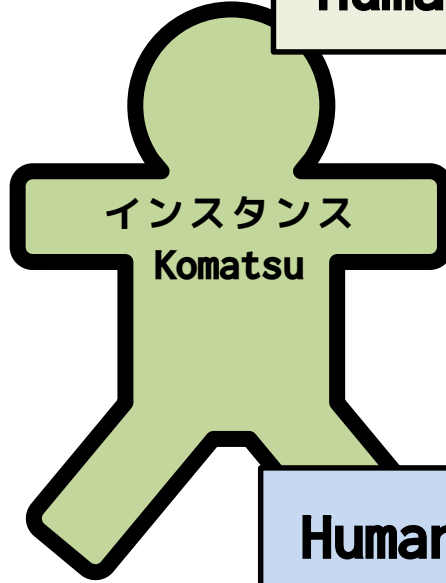
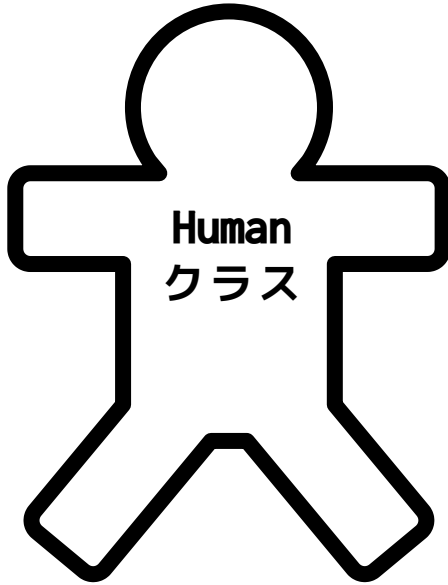
インスタンス  
Fukuchi

```
Human Fukuchi = new Human();
```

インスタンス  
Miyashita

```
Human Miyashita = new Human();
```

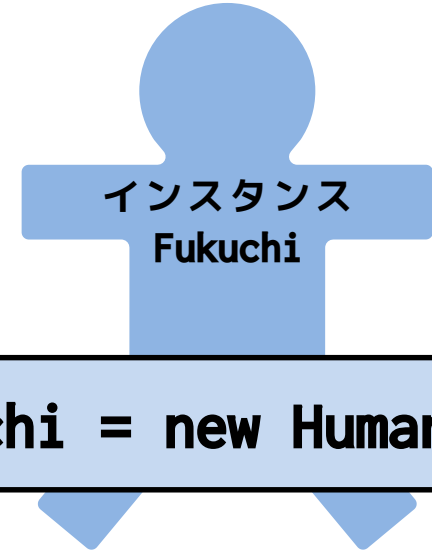
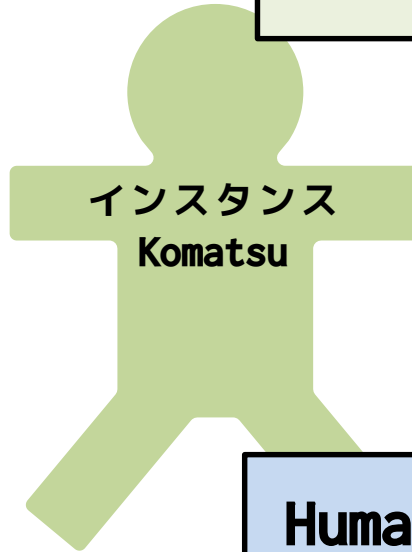
Human  
クラス



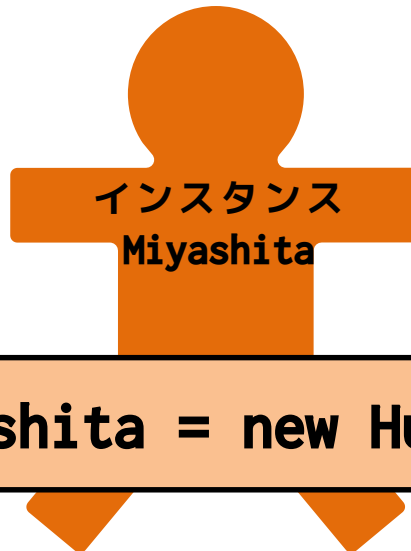
# インスタンス化



```
Human Komatsu = new Human();
```



```
Human Fukuchi = new Human();
```



```
Human Miyashita = new Human();
```

# 使い方



インスタンス  
Komatsu

```
Komatsu.move();
```

インスタンス  
Fukuchi

```
Fukuchi.eat();
```

インスタンス  
Miyashita

```
Miyashita.move();
```

```
Miyashita.eat();
```



- 時計というクラスを作る
  - 時計のインスタンス変数
    - 現在時間（時分秒）の情報
    - 目覚ましのON/OFF状態管理変数
    - 目覚ましの設定時間
  - 時計のインスタンスメソッド
    - 分変更メソッド
    - 時計停止メソッド（秒針停止）
    - 目覚まし設定時間変更メソッド
    - 目覚ましのON/OFF切り替えメソッド

# 定義したクラスの使い方



- クラスの中で変数を定義すると、そのクラス内の変数として使うことができる
  - クラス内で変数を定義
  - クラスを使う場合は new !
  - クラス内の変数を使う場合はドットでつなぐ

```
class Ball
{
    float posX;
    float posY;
    float speedX;
    float speedY;
}

Ball ball;
ball = new Ball();
ball.posX += ball.speedX;
ball.posY += ball.speedY;
```

## インスタンス化

クラス名 変数名 = new クラス名();

# 端で跳ね返るオブジェクト

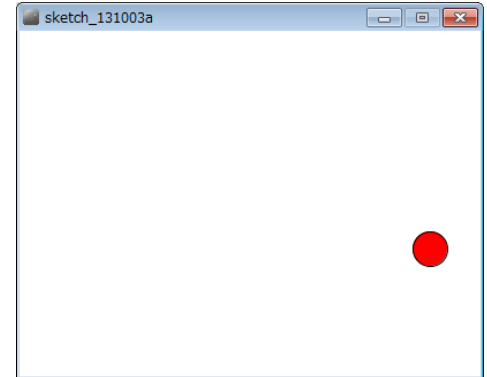
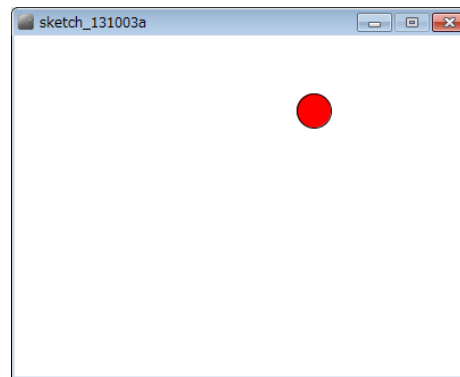
明治大学総合数理学部  
先端メディアサイエンス学科  
中村研究室



(Q1) 400x300のウィンドウ内で、画面中央から毎フレームx方向に2ピクセル、y方向に3ピクセルずつ移動する直径が30の赤い円が右端・左端・上端・下端に来ると跳ね返るようにするには？

## • 考え方

- 右端・左端・上端・下端で衝突する時の条件を整理
- 衝突した時の速度を反転させる
  - $speedX = -speedX;$
  - $speedY = -speedY;$



```
float posX;  
float posY;  
float speedX;  
float speedY;  
  
void setup()  
{  
  size(400, 300);  
  posX = random(0, width);  
  posY = random(0, height);  
  speedX = random(-5, 5);  
  speedY = random(-5, 5);  
  fill(255, 0, 0);  
}
```

今までの知識で  
プログラムを組むと

```
void draw()  
{  
  background(255);  
  posX += speedX;  
  posY += speedY;  
  if(posX > width){  
    posX = width * 2 - posX;  
    speedX = -speedX;  
  }  
  if(posX < 0){  
    posX = -posX;  
    speedX = -speedX;  
  }  
  if(posY > height){  
    posY = height * 2 - posY;  
    speedY = -speedY;  
  }  
  if(posY < 0){  
    posY = -posY;  
    speedY = -speedY;  
  }  
  ellipse(posX, posY, 30, 30);  
}
```

# クラスで考える



- 座標, スピードを持ったオブジェクトを作る
  - ここではx, y座標(posX, posY)とspeedX, speedYを持つBallクラスを考え, その変数を定義する.

## 定義する

```
class Ball {  
    float posX;  
    float posY;  
    float speedX;  
    float speedY;  
}
```

変数を定義

## インスタンス化する(使えるようにする)

```
Ball ball;  
ball = new Ball();  
ball.posX = 200;  
ball.posY = 150;  
ball.speedX = 5;  
ball.speedY = 7;
```

変数の定義(箱を作る)

newで具体化

値を代入

## 描画してみる

```
ellipse( ball.posX, ball.posY, 30, 30 );
```

オブジェクト変数名 . インスタンス変数名





# クラスを試してみる

```
class Ball
{
  float posX;
  float posY;
  float speedX;
  float speedY;
}

Ball ball;

void setup()
{
  size(400, 300);
  ball = new Ball();
  ball.posX = random(0, width);
  ball.posY = random(0, height);
  ball.speedX = random(-5, 5);
  ball.speedY = random(-5, 5);
  fill(255, 0, 0);
}
```

```
void draw()
{
  background(255);

  ball.posX += ball.speedX;
  ball.posY += ball.speedY;
  if(ball.posX > width){
    ball.posX = width * 2 - ball.posX;
    ball.speedX = -ball.speedX;
  }
  if(ball.posX < 0){
    ball.posX = -ball.posX;
    ball.speedX = -ball.speedX;
  }
  if(ball.posY > height){
    ball.posY = height * 2 - ball.posY;
    ball.speedY = -ball.speedY;
  }
  if(ball.posY < 0){
    ball.posY = -ball.posY;
    ball.speedY = -ball.speedY;
  }
  ellipse(ball.posX, ball.posY, 30, 30);
}
```

# ボールを3つにすると？



- 3つの変数を定義
  - new で具体化
    - ball1
    - ball2
    - ball3
  - というボールにする！
  - そのそれぞれの値を参照できるようにする

```
class Ball{
    float posX;
    float posY;
    float speedX;
    float speedY;
}

Ball ball1;
Ball ball2;
Ball ball3;

void setup() {
    size( 400, 300 );

    ball1 = new Ball();
    ball2 = new Ball();
    ball3 = new Ball();
}
```

```
class Ball
{
    float posX;
    float posY;
    float speedX;
    float speedY;
}
```

```
Ball ball1;
Ball ball2;
Ball ball3;
```

```
void setup()
```

```
{
    size( 400, 300 );
    fill( 255, 0, 0 );
    ball1 = new Ball();
    ball2 = new Ball();
    ball3 = new Ball();
    ball1.posX = random(width);
    ball1.posY = random(height);
    ball1.speedX = random(1, 5);
    ball1.speedY = random(1, 5);
    ball2.posX = random(width);
    ball2.posY = random(height);
    ball2.speedX = random(1, 5);
    ball2.speedY = random(1, 5);
    ball3.posX = random(width);
    ball3.posY = random(height);
    ball3.speedX = random(1, 5);
    ball3.speedY = random(1, 5);
}
```

```
void draw()
```

```
{
    background(255);
    ball1.posX = ball1.posX + ball1.speedX;
    ball1.posY = ball1.posY + ball1.speedY;
    ball2.posX = ball2.posX + ball2.speedX;
    ball2.posY = ball2.posY + ball2.speedY;
    ball3.posX = ball3.posX + ball3.speedX;
    ball3.posY = ball3.posY + ball3.speedY;

    if(ball1.posX > width){
        ball1.posX = width * 2 - ball1.posX;
        ball1.speedX = -ball1.speedX;
    }
    if(ball1.posX < 0){
        ball1.posX = -ball1.posX;
        ball1.speedX = -ball1.speedX;
    }
    if(ball2.posX > width){
        ball2.posX = width * 2 - ball2.posX;
        ball2.speedX = -ball2.speedX;
    }
    :
    :
    :
    ellipse(ball1.posX, ball1.posY, 30, 30);
    ellipse(ball2.posX, ball2.posY, 30, 30);
    ellipse(ball3.posX, ball3.posY, 30, 30);
}
```

```
class Ball
{
  float posX;
  float posY;
  float speedX;
  float speedY;
}
```

```
Ball ball1;
Ball ball2;
Ball ball3;
```

```
void setup()
```

```
{
  size( 400, 400);
  fill( 255, 255, 255);
  ball1 = new Ball( random(width), random(height), random(1, 5), random(1, 5));
  ball2 = new Ball( random(width), random(height), random(1, 5), random(1, 5));
  ball3 = new Ball( random(width), random(height), random(1, 5), random(1, 5));
  ball1.posX = random(width);
  ball1.posY = random(height);
  ball1.speedX = random(1, 5);
  ball1.speedY = random(1, 5);
  ball2.posX = random(width);
  ball2.posY = random(height);
  ball2.speedX = random(1, 5);
  ball2.speedY = random(1, 5);
  ball3.posX = random(width);
  ball3.posY = random(height);
  ball3.speedX = random(1, 5);
  ball3.speedY = random(1, 5);
}
```

```
void draw()
```

```
{
  background(255);
  ball1.posX = ball1.posX + ball1.speedX;
  ball1.posY = ball1.posY + ball1.speedY;
  ball2.posX = ball2.posX + ball2.speedX;
  ball2.posY = ball2.posY + ball2.speedY;
  ball3.posX = ball3.posX + ball3.speedX;
  ball3.posY = ball3.posY + ball3.speedY;
}
```

まったく楽になっていない！  
というかむしろ大変になっている！！  
クラス面倒なだけじゃん！！！！

データ型としてしか使っていないため  
面倒で当然

```
ball2.posX = width * 2 - ball2.posX;
ball2.speedX = -ball2.speedX;
```

```
ellipse(ball2.posX, ball2.posY, 30, 30);
ellipse(ball3.posX, ball3.posY, 30, 30);
```

```
}
```

# インスタンスメソッド



- 移動をインスタンスメソッドにしてしまう
  - 全ての円は場所や速度は違うけれど、同じルールで動いているのでまとめることが可能！
  - 内部で勝手に振る舞うメソッド（関数）にしてしまう

下記のように指定するだけで動くように！

```
ball1.move();  
ball2.move();  
ball3.move();
```

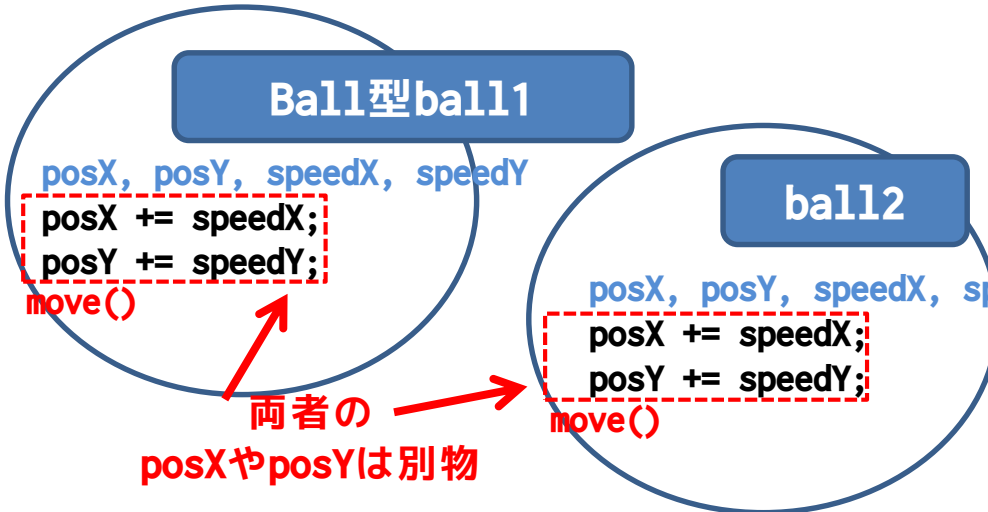
オブジェクト変数名 . インスタンスメソッド名

クラス内で定義されているposXやposY, speedX, speedYを利用したり変更したりできる

# void move() を作る

- 移動用メソッドを追加
  - 位置を変更
  - 端に来ると跳ね返る
- インスタンスメソッドではクラス内変数を活用

```
class Ball{  
    float posX;  
    float posY;  
    float speedX;  
    float speedY;  
    void move(){  
        this.posX += this.speedX;  
        this.posY += this.speedY;  
        if(this.posX > width) {  
            this.posX = width * 2 - posX;  
            this.speedX = -this.speedX;  
        }  
        if(this.posX < 0){  
            this.posX = -this.posX;  
            this.speedX = -this.speedX;  
        }  
        if(this.posY > height){  
            this.posY = height * 2 - this.posY;  
            this.speedY = -this.speedY;  
        }  
        if(this.posY < 0){  
            this.posY = -this.posY;  
            this.speedY = -this.speedY;  
        }  
    }  
}
```



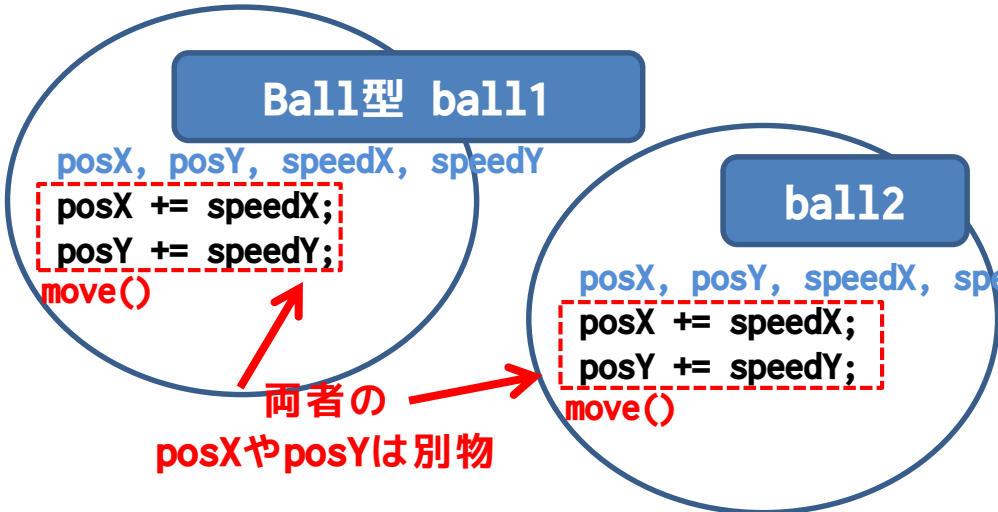
**this.は省略可能**

# void move() を作る

- 移動用メソッドを追加
  - 位置を変更
  - 端に来ると跳ね返る
- インスタンスメソッドではクラス内変数を活用

```
class Ball{
    float posX;
    float posY;
    float speedX;
    float speedY;

    void move(){
        posX += speedX;
        posY += speedY;
        if(posX > width){
            posX = width * 2 - posX;
            speedX = -speedX;
        }
        if(posX < 0){
            posX = -posX;
            speedX = -speedX;
        }
        if(posY > height){
            posY = height * 2 - posY;
            speedY = -speedY;
        }
        if(posY < 0){
            posY = -posY;
            speedY = -speedY;
        }
    }
}
```



# 改良したBallクラスを使うと



```
Ball ball1;
Ball ball2;
Ball ball3;

void setup()
{
    size(400, 300);
    fill(255, 0, 0);

    ball1 = new Ball();
    ball2 = new Ball();
    ball3 = new Ball();

    ball1.posX = random(width);
    ball1.posY = random(height);
    ball1.speedX = random(5);
    ball1.speedY = random(5);
    ball2.posX = random(width);
    ball2.posY = random(height);
    ball2.speedX = random(5);
    ball2.speedY = random(5);
    ball3.posX = random(width);
    ball3.posY = random(height);
    ball3.speedX = random(5);
    ball3.speedY = random(5);
}
```

```
void draw()
{
    background(255);
    ball1.move();
    ball2.move();
    ball3.move();

    ellipse(ball1.posX, ball1.posY, 30, 30);
    ellipse(ball2.posX, ball2.posY, 30, 30);
    ellipse(ball3.posX, ball3.posY, 30, 30);
}
```



**draw() がかなり  
短くなった！**



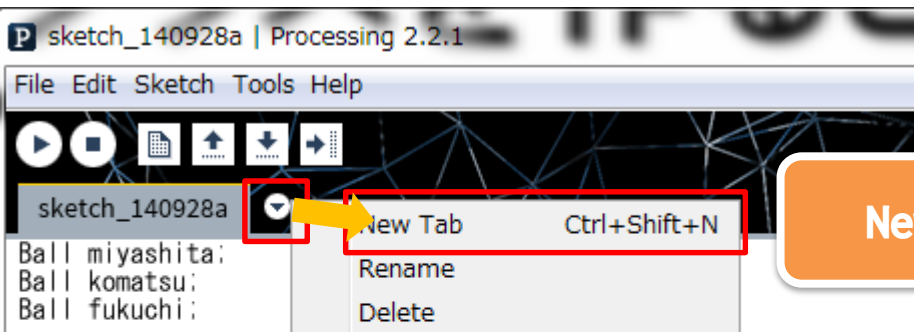
# プログラムを動かそう！

明治大学総合数理学部  
先端メディアサイエンス学科  
中村研究室

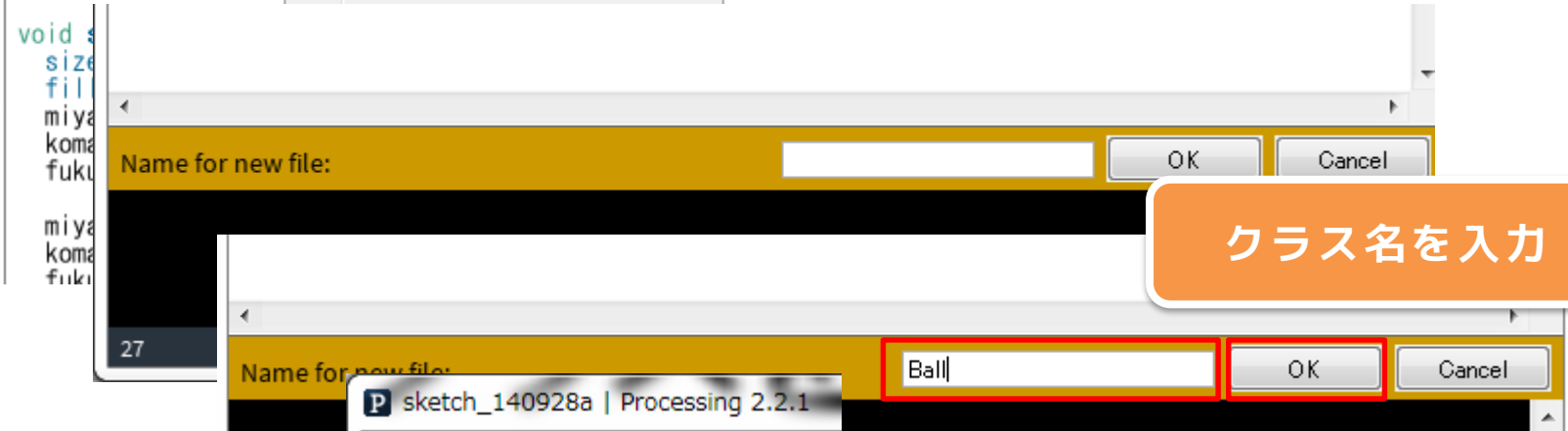


- <http://nkmr.io/lecture/> の第2回講義資料にある Ball.txt を利用しよう
  - Ballクラスの部分は別のタブに！（次ページで説明）

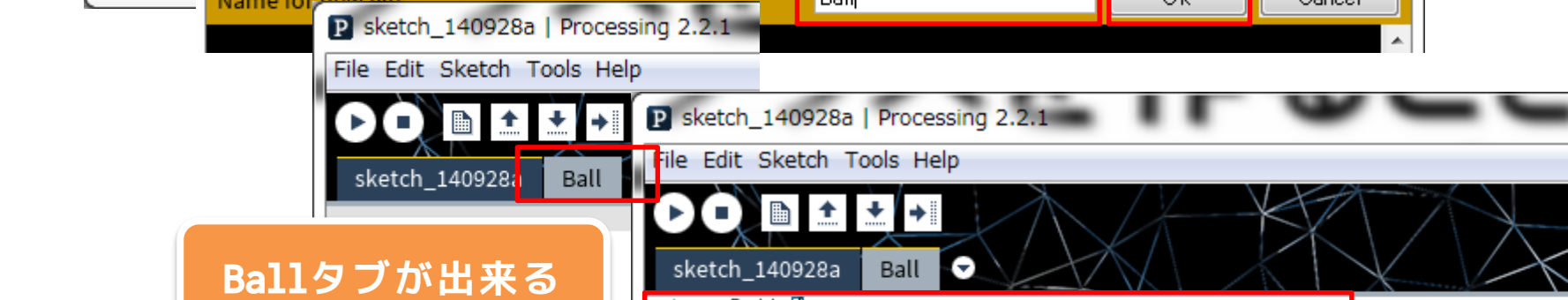
# クラスを作るときは別タブで



New Tabを選択



クラス名を入力



Ballタブが出る



そこに書き込む！

# インスタンスメソッド続き



- 最初の位置を設定する部分もインスタンスメソッドにしてしまおう！
  - 初期位置の設定方法は
    - `XXXXX.posX = random(width);`
    - `XXXXX.posY = random(height);`
    - `XXXXX.speedX = random(1, 5);`
    - `XXXXX.speedY = random(1, 5);`

# initialize() で初期化



```
class Ball
{
    float posX;
    float posY;
    float speedX;
    float speedY;

    void initialize()
    {
        posX = random(width);
        posY = random(height);
        speedX = random(1, 5);
        speedY = random(1, 5);
    }
}
```

```
void move(){
    posX += speedX;
    posY += speedY;
    if(posX > width){
        posX = width * 2 - posX;
        speedX = -speedX;
    }
    if(posX < 0){
        posX = -posX;
        speedX = -speedX;
    }
    if(posY > height){
        posY = height * 2 - posY;
        speedY = -speedY;
    }
    if(posY < 0){
        posY = -posY;
        speedY = -speedY;
    }
}
}
```

# 改良したBallクラスを使うと



```
Ball ball1;  
Ball ball2;  
Ball ball3;  
  
void setup()  
{  
  size(400, 300);  
  fill(255, 0, 0);  
  
  ball1 = new Ball();  
  ball2 = new Ball();  
  ball3 = new Ball();  
  ball1.initialize();  
  ball2.initialize();  
  ball3.initialize();  
}
```

```
void draw()  
{  
  background(255);  
  
  ball1.move();  
  ball2.move();  
  ball3.move();  
  
  ellipse(ball1.posX, ball1.posY, 30, 30);  
  ellipse(ball2.posX, ball2.posY, 30, 30);  
  ellipse(ball3.posX, ball3.posY, 30, 30);  
}
```

**setup() もかなり  
短くなった！**

先ほどのプログラムを改良して  
動かしてみよう！

# コンストラクタ！



- コンストラクタは new されたときに呼び出される場所。initialize() はそこで呼び出したら良いのでは？

```
class Ball
{
    float posX;
    float posY;
    float speedX;
    float speedY;
    Ball(){

    }
}
```

# コンストラクタ！



```
class Ball{
    float posX;
    float posY;
    float speedX;
    float speedY;
    Ball(){
        initialize();
    }

    void initialize(){
        posX = random(width);
        posY = random(height);
        speedX = random(1,5);
        speedY = random(1,5);
    }
}
```

コンストラクタで  
initializeメソッドを  
呼び出す！

```
void move(){
    posX += speedX;
    posY += speedY;
    if(posX > width){
        posX = width * 2 - posX;
        speedX = -speedX;
    }
    if(posX < 0){
        posX = -posX;
        speedX = -speedX;
    }
    if(posY > height){
        posY = height * 2 - posY;
        speedY = -speedY;
    }
    if(posY < 0){
        posY = -posY;
        speedY = -speedY;
    }
}
```

# コンストラクタに移動



```
class Ball{
    float posX;
    float posY;
    float speedX;
    float speedY;

    Ball(){
        posX = random(width);
        posY = random(height);
        speedX = random(1,5);
        speedY = random(1,5);
    }

    void initialize(){
    }
```

**initializeの中身をコンストラクタの中に移動して、なくしてしまってもよい!**

```
void move(){
    posX += speedX;
    posY += speedY;
    if(posX > width){
        posX = width * 2 - posX;
        speedX = -speedX;
    }
    if(posX < 0){
        posX = -posX;
        speedX = -speedX;
    }
    if(posY > height){
        posY = height * 2 - posY;
        speedY = -speedY;
    }
    if(posY < 0){
        posY = -posY;
        speedY = -speedY;
    }
}
```



# 改良したBallクラスを使うと



```
Ball ball1;  
Ball ball2;  
Ball ball3;  
  
void setup() {  
  size(400, 300);  
  fill(255, 0, 0);  
  
  ball1 = new Ball();  
  ball2 = new Ball();  
  ball3 = new Ball();  
}
```

```
void draw() {  
  background(255);  
  
  ball1.move();  
  ball2.move();  
  ball3.move();  
  
  ellipse(ball1.posX, ball1.posY, 30, 30);  
  ellipse(ball2.posX, ball2.posY, 30, 30);  
  ellipse(ball3.posX, ball3.posY, 30, 30);  
}
```

↑  
**setup() がさらに  
短くなった！**

先ほどのプログラムを改良して  
動かしてみよう！



- 下みたいなのはあまり好ましくない
  - ellipse(ball1.posX, ball1.posY, 30, 30);
  - ellipse(ball2.posX, ball2.posY, 30, 30);
  - ellipse(ball3.posX, ball3.posY, 30, 30);
  - あと, 色の指定もdisplayにもっていく!

```
class Ball{
  float posX;
  float posY;
  float speedX;
  float speedY;
  void display(){
    fill(255, 0, 0);
    ellipse( posX, posY, 30, 30 );
  }
}
```

データはなるべく隠蔽  
したい (カプセル化)

```
void draw() {
  background(255);
  ball1.move();
  ball2.move();
  ball3.move();

  ball1.display();
  ball2.display();
  ball3.display();
}
```

先ほどのプログラムを改良して  
動かしてみよう!

# 端で跳ね返る50個のボール



400x300の画面内にX座標Y座標ともにランダムに配置された50個の円を描き，それぞれのX，Y方向へのスピードに応じて移動するようにせよ

- posX, posY という座標の配列と, speedX, speedY というXおよびY方向の速度をもつ配列を導入！



# オブジェクト + 配列



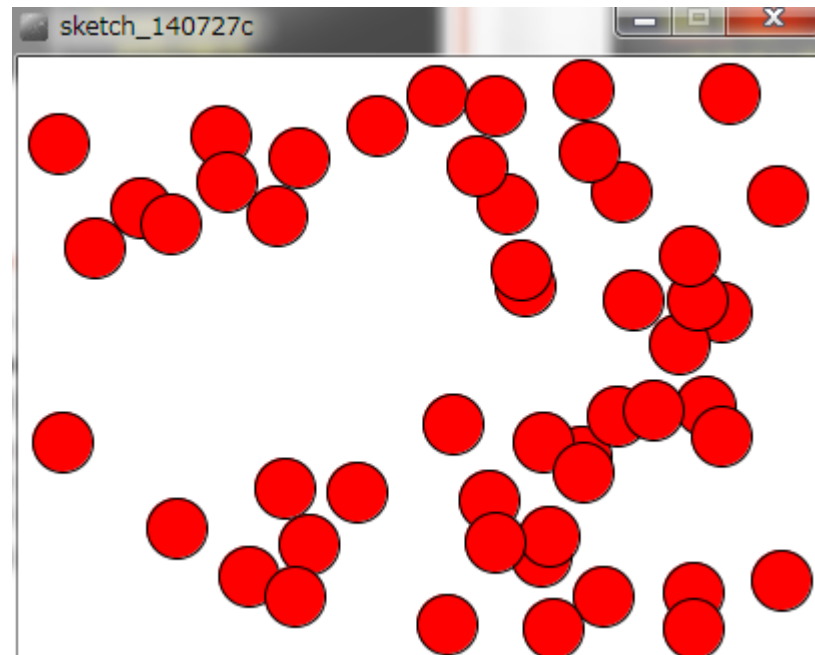
- 50個の丸を動かすには配列を使う！

```
Ball[] balls = new Ball[50];
```

```
void setup(){  
  size(400, 300);  
  for(int i=0; i<50; i++){  
    balls[i] = new Ball();  
  }  
}
```

```
void draw(){  
  background(255);  
  for(int i=0; i<50; i++){  
    balls[i].move();  
    balls[i].display();  
  }  
}
```

型 [] 配列名 = new 型 [要素数];



# オブジェクト + 配列



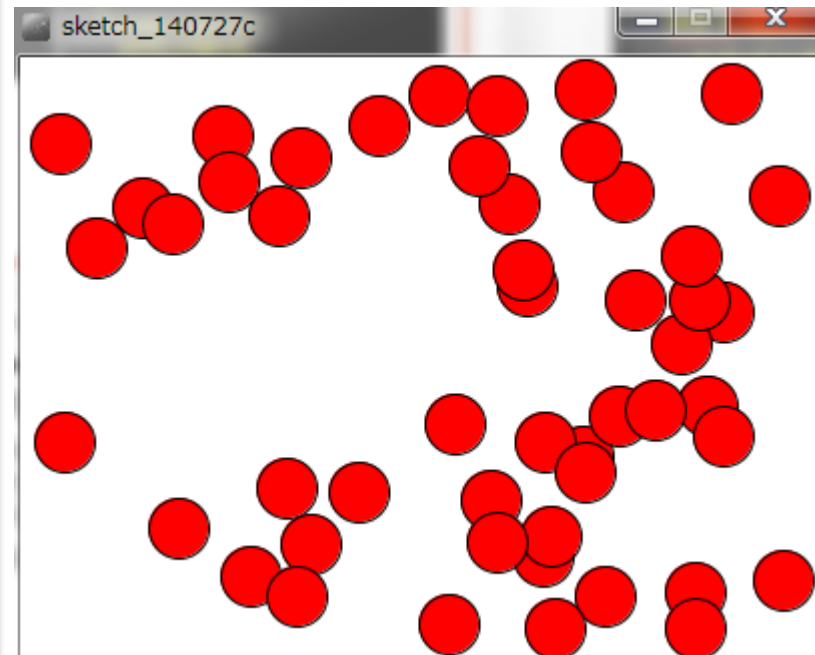
- 50個の丸を動かすには配列を使う！

```
Ball[] balls = new Ball[50];

void setup() {
  size( 400, 300 );
  for(int i=0; i<balls.length; i++){
    balls[i] = new Ball();
  }
}

void draw(){
  background(255);
  for(int i=0; i<balls.length; i++){
    balls[i].move();
    balls[i].display();
  }
}
```

配列変数名.length  
で配列の長さを取得



# オブジェクト + 配列



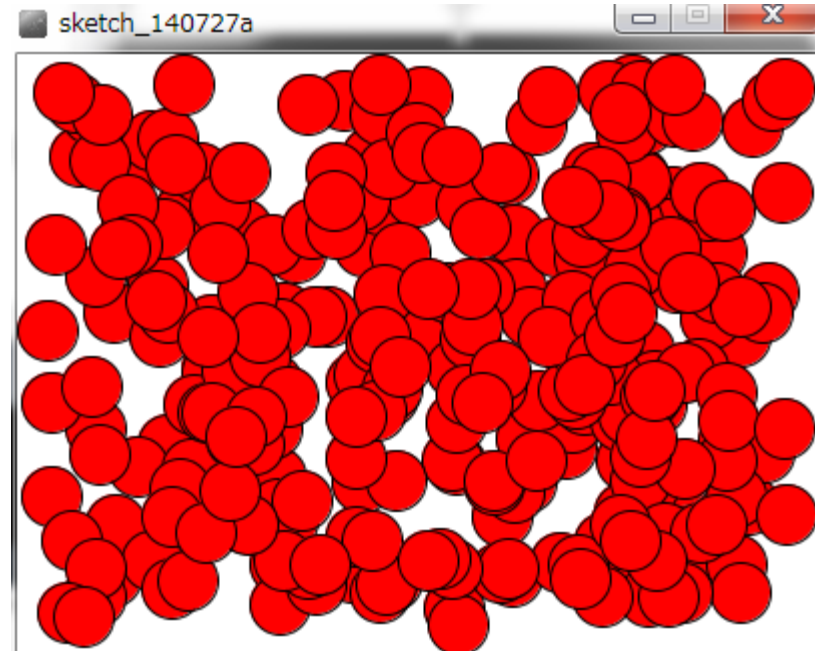
- 300個の丸を動かすには配列の定義を変更

```
Ball[] balls = new Ball[300];

void setup() {
  size(400, 300);
  for(int i=0; i<balls.length; i++){
    balls[i] = new Ball();
  }
}

void draw(){
  background(255);
  for(int i=0; i<balls.length; i++){
    balls[i].move();
    balls[i].display();
  }
}
```

配列変数名.length  
で配列の長さを取得



# コンストラクタの不思議



- 何故 `void Ball(){ ... }` じゃないの？
  - コンストラクタは、そもそも返り値（`return`で返されるもの）が存在しない。そのため、返り値に関する設定が不要！
  - ちなみに、`Ball(){...}`だけじゃなく、`Ball(float x, float y){...}`みたいなパターンもあります！（これは次回に）

# ArrayListと組み合わせ



- クリックするたびにBallがうまれる！

```
ArrayList<Ball> balls = new ArrayList<Ball>();

void setup() {
  size(400, 300);
}

void draw(){
  background(255);
  for(int i=0; i<balls.size(); i++){
    balls.get(i).move();
    balls.get(i).display();
  }
}

void mousePressed(){
  Ball ball = new Ball();
  balls.add( ball );
}
```



# ArrayListと組み合わせ



- クリックするたびにBallがうまれる！

```
ArrayList<Ball> balls = new ArrayList<Ball>();

void setup() {
  size(400, 300);
}

void draw(){
  background(255);
  for(int i=0; i<balls.size(); i++){
    balls.get(i).move();
    balls.get(i).display();
  }
}

void mousePressed(){
  balls.add(new Ball());
}
```

こう書いてもOK

# ArrayListと組み合わせ



- クリックするたびにBallがうまれる！

```
ArrayList balls = new ArrayList();
```

```
void setup() {  
    size(400, 300);  
}
```

定義をこうする場合は

```
void draw(){  
    background(255);  
    for(int i=0; i<balls.size(); i++){  
        Ball ball = balls.get(i);  
        ball.move();  
        ball.display();  
    }  
}
```

キャストして使う！

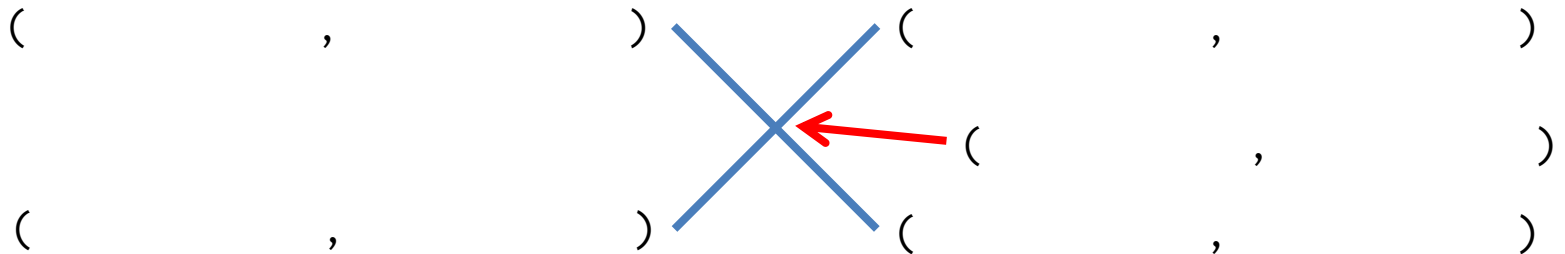
```
void mousePressed(){  
    balls.add(new Ball());  
}
```

# xが動き回るクラスを作ろう



Ballクラスを利用して、Crossクラスを作ろう！

- Ballクラスと、Crossクラスの違いは、表示される図形が「○」か「x」かなだけ！！
- Crossクラスのタブを作成し、Ballクラスをコピー！
- BallをCrossに書き換える！
- 表示だけを変更したいので、displayの中身を変更！
- メインのプログラムで Cross を使っていこう！



# 課題2-1: basic\_diceHistory



- 1~6までの数字をランダムに生成することで1~6の目からなるサイコロをひたすらふり, 6が連続で3回出るまで繰り返せ
- 6が連続で3回出たら終了し, 最後の10回分と, サイコロを何回ふったかを標準出力せよ(ただし, 10回以下の場合はその目だけを出力せよ)
- ArrayListを利用して実現せよ

最後の10回

1  
3  
5  
3  
3  
6  
2  
6  
6  
6

サイコロを振った回数 616

# 課題2-2: basic\_boundA112



- Ball クラスを改良し,  $x$ が動き回るCrossクラスと $\triangle$ が動き回るTriangleクラスを作成せよ
- またこれを利用して5個の $\circ$ と, 4個の $x$ と, 3個の $\triangle$ が動き回るプログラムを作成せよ
  - ただし, その速度は $x$ 、 $y$ 方向それぞれ $-5\sim 5$ の実数値とせよ
  - また,  $\circ$ は壁で跳ね返り,  $x$ と $\triangle$ は跳ね返らずに反対側から出てくるようにせよ

# 宿題2-1: hw\_listBall

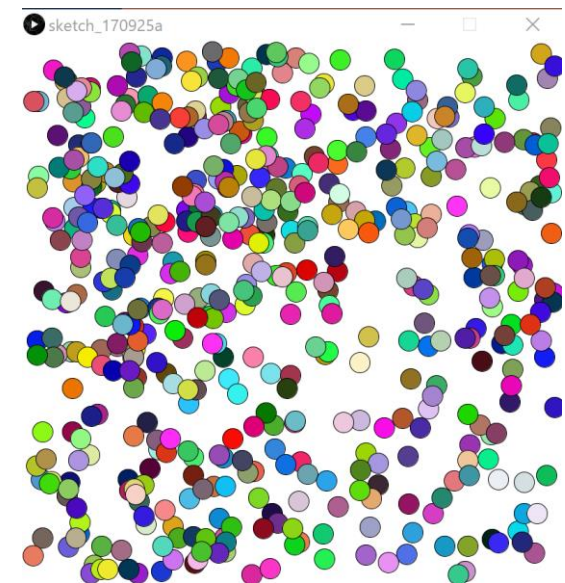


- ウィンドウ800x800を用意し, Ballクラスを利用してマウスをクリックするたびに丸が生まれ, 上下左右にランダムな速度で動き, 上下左右の端で跳ね返るプログラムを作成せよ. なお, クリックでいくらでも作成できるようにせよ
- また, Ballクラスを改良してそれぞれの丸の色は初期値としてランダムに設定するようにせよ

[ヒント] 色情報を持つインスタンス変数を追加する

- int red;
- int green;
- int blue;

- たくさん増やして管理するには ArrayListを使う!



# 宿題2-2: hw\_Covid19Genome

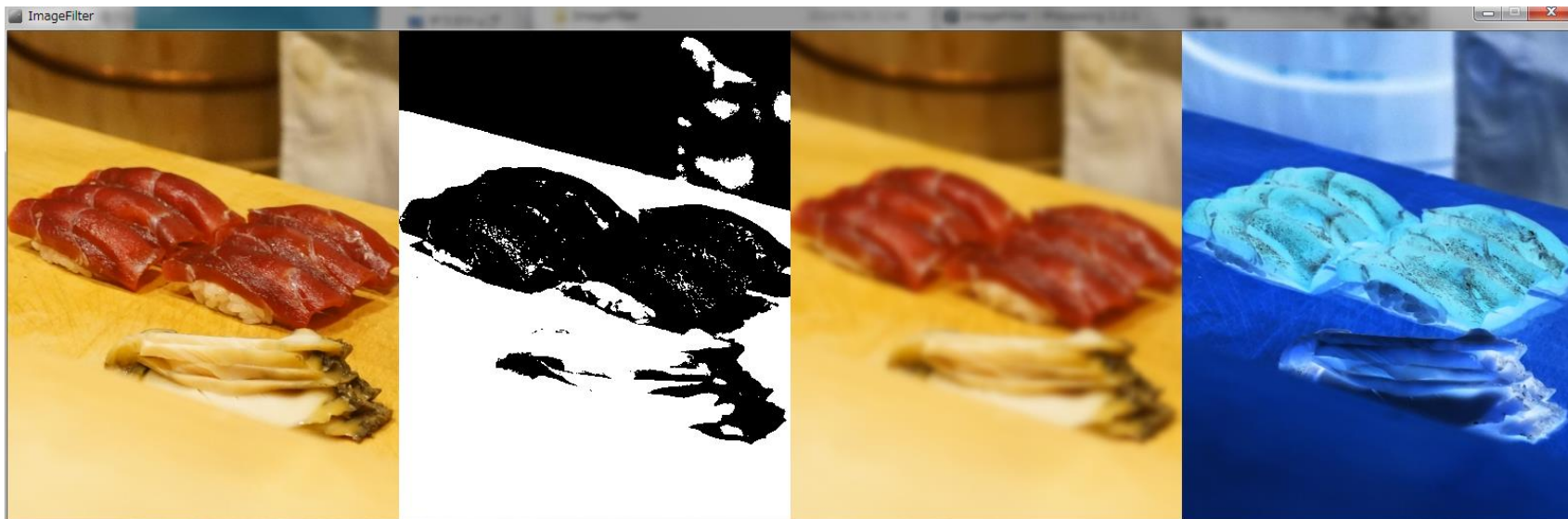


- 下記のPDEを利用し, 指定の要件を満たすようにプログラムを完成させよ
  - basic\_Covid19Genome.pde
    - SARS-CoV-2 ( 新型コロナウイルス ) の塩基配列
  - 部分点あります
- 要件
  - 塩基配列の長さを標準出力
  - 最初の100文字と最後の100文字を標準出力
  - A, T, G, Cのそれぞれの出現率を標準出力
  - AAA, TTT, GGG, CCCのそれぞれの出現回数を出力
    - <https://www.ncbi.nlm.nih.gov/nuccore/MN908947>

# 宿題2-3: hw\_imageProc



- PImage クラスを使って画像処理をしてみよう
    - 適当な画像をダウンロードし, その画像に対してフィルタを掛けてみましょう!
    - 一番左にオリジナル画像, 次にTHRESHOLDで二値化したもの, さらに次にBLURでぼかしたもの, 一番右にINVERTでネガポジ反転したものを表示しよう
- (ヒント) PImage を4つ定義すると良いよ!







- オブジェクト指向のさわりを学んだ
  - 色々なクラス
    - ArrayList, PImage, Minim, AudioPlayer, String
  - インスタンス化
    - `Human nkmr = new Human();`
  - インスタンス
    - `nkmr`
  - インスタンス変数
    - `nkmr.speed`
  - インスタンスメソッド
    - `nkmr.move()`
  - コンストラクタ
    - `Human(){ 初期化处理 }`