



プログラミング演習2 クラス

中村、小林、橋本、辻野

本日の流れ



- 13:30-14:00 宿題の解説
- 14:00-15:40 座学 + 演習 (10分休憩)
- 15:40-15:45 課題提示
- 15:45-16:45 課題に取り組む
- 16:45が課題の提出期限
- 16:45-17:00 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



• 面倒じゃなかったですか？

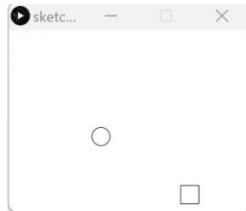
プログラミング演習II 課題

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



• スケッチ名 : basic_BallRect

- 400x300のウィンドウ内を動き回る丸（直径30ピクセルの白色の円）と正方形（一辺の長さが30ピクセルの白色の正方形）を実現せよ
- ただし、その初期値はウィンドウ内のランダムな位置とし、またその速度はx、y方向それぞれ-5~5の実数値とせよ。
- また、丸も正方形も壁で跳ね返るようにせよ。跳ね返り方は図形の端でも図形の中央でもよい。



宿題1-1: hw_boundAll

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



- 10個の赤色の○と、5個の青色の□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、丸も正方形も壁で跳ね返るようにせよ。跳ね返り方は図形の端でも図形の中央でもよい。

これからやること



- 「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



- 変数の定義がめんどい
 - circleX, circleY, circleVX, circleVY
 - squareX, squareY, squareVX, squareVY
- drawが汚くなるので関数で実現する？
 - circleMove();
 - 円を動かす関数
 - squareMove();
 - 正方形を動かす関数

図形の種類が増えたら？



- 変数の定義がめんどい
 - circleX, circleY, circleVX, circleVY
 - squareX, squareY, squareVX, squareVY
 - triangleX, triangleY, triangleVX, triangleVY
- drawが汚くなるので関数で実現する？
 - circleMove();
 - 円を動かす関数
 - squareMove();
 - 正方形を動かす関数
 - triangleMove();
 - 三角形を動かす関数

数が増えたら？



- 変数の定義がめんどい
 - `circleX[i]`, `circleY[i]`, `circleVX[i]`, `circleVY[i]`
 - `squareX[i]`, `squareY[i]`, `squareVX[i]`, `squareVY[i]`
- `draw`が汚くなるので関数で実現する？
 - `circleMove()`;
 - 円を動かす関数
 - `squareMove()`;
 - 正方形を動かす関数

動きは図形に任せたい



- 丸、正方形、三角形を定義

- それぞれの座標は意識したくない

- `circle.x`, `circle.y`, `square.x`, `square.y`, `triangle.x`, `triangle.y`
- 内部で適当に処理してもらう

- それぞれの速度も意識したくない

- `circle.vx`, `circle.vy`, `square.vx`, `square.vy`, `triangle.vx`, `triangle.vy`

- 描画はシンプルにしたい

- `circle.display()`, `square.display()`, `triangle.display()`

- 移動もシンプルにしたい

- `circle.move()`, `square.move()`, `triangle.move()`

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

クラスの定義



```
class Human {
```

```
    int lon, lat;
```

```
    String name;
```

```
    float bodyHeight;
```

```
    int age;
```

```
    void move();
```

```
    void eat();
```

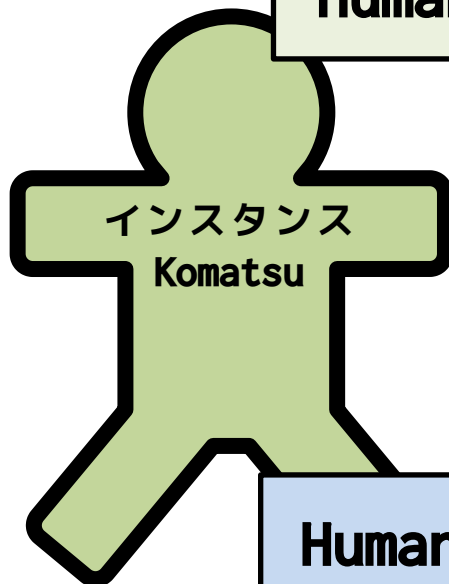
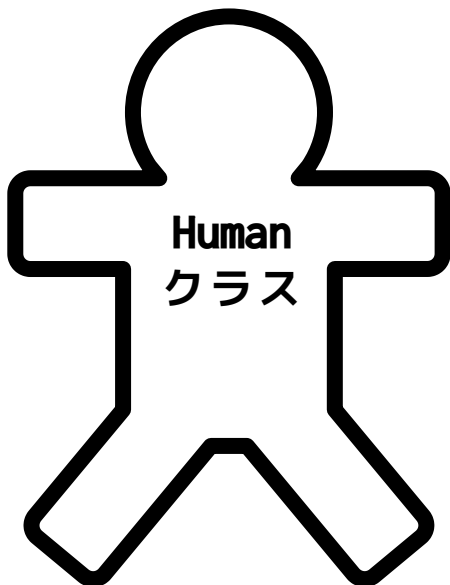
```
    String say();
```

```
}
```

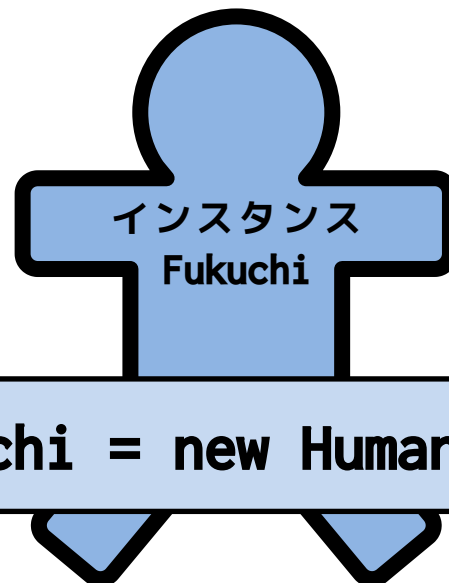
インスタンス変数

インスタンスメソッド

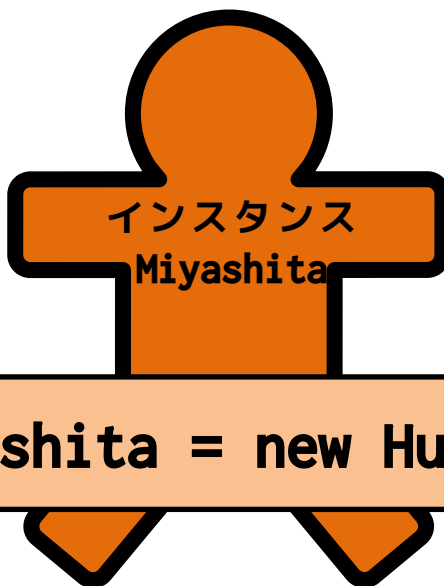
インスタンス化



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



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



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

クラスの定義



- CircleClass というクラスを定義

```
class クラス名 {  
    クラスの諸要素に関する定義  
}
```

```
class CircleClass {  
  
}
```

```
class SquareClass {  
  
}
```

```
class CrossClass {  
  
}
```

端で跳ね返るオブジェクト

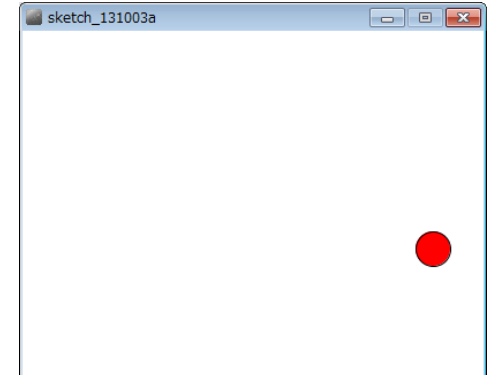
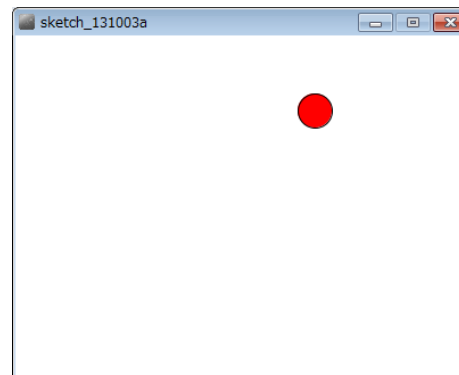
明治大学総合数理学部
先端メディアサイエンス学科
中村研究室



赤色の○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でランダムに設定される実数値とせよ

• 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
 - $v_x = -v_x$;
 - $v_y = -v_y$;



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 CircleClass
{
    float x;
    float y;
    float vx;
    float vy;
}
```

```
CircleClass circleObj;
circleObj = new CircleClass();
circleObj.x += circleObj.vx;
circleObj.y += circleObj.vy;
```

インスタンス化

クラス名 変数名 = new クラス名();

```
float x;  
float y;  
float vx;  
float vy;
```

```
void setup()  
{  
  size(400, 300);  
  x = random(0, width);  
  y = random(0, height);  
  vx = random(-5, 5);  
  vy = random(-5, 5);  
}
```

今までの知識で
プログラムを組むと

```
void draw()  
{  
  background(255);  
  x += vx;  
  y += vy;  
  if(x > width){  
    x = width * 2 - x;  
    vx = -vx;  
  }  
  if(x < 0){  
    x = -x;  
    vx = -vx;  
  }  
  if(y > height){  
    y = height * 2 - y;  
    vy = -vy;  
  }  
  if(y < 0){  
    y = -y;  
    vy = -vy;  
  }  
  fill(255, 0, 0);  
  circle(x, y, 30);  
}
```

```
class CircleClass
```

```
{  
  float x;  
  float y;  
  float vx;  
  float vy;  
}
```

```
CircleClass circleObj  
  = new CircleClass();
```

```
void setup()
```

```
{  
  size(400, 300);  
  circleObj.x = random(0, width);  
  circleObj.y = random(0, height);  
  circleObj.vx = random(-5, 5);  
  circleObj.vy = random(-5, 5);  
}
```

```
void draw()
```

```
{  
  background(255);  
  circleObj.x += circleObj.vx;  
  circleObj.y += circleObj.vy;  
  if(circleObj.x > width){  
    circleObj.x = width * 2 - circleObj.x;  
    circleObj.vx = -circleObj.vx;  
  }  
  if(circleObj.x < 0){  
    circleObj.x = -circleObj.x;  
    circleObj.vx = -circleObj.vx;  
  }  
  if(circleObj.y > height){  
    circleObj.y = height * 2 - circleObj.y;  
    circleObj.vy = -circleObj.vy;  
  }  
  if(circleObj.y < 0){  
    circleObj.y = -circleObj.y;  
    circleObj.vy = -circleObj.vy;  
  }  
  fill(255, 0, 0);  
  circle(circleObj.x, circleObj.y, 30);  
}
```

クラスを使うと

10個の動く円



- 配列で実現することはできるが、それぞれが独立しているのが違和感

```
float[] circleX = new float[10];
float[] circleY = new float[10];
float[] circleVX = new float[10];
float[] circleVY = new float[10];

void setup() {
  size(400, 300);
  for(int i=0; i<10; i++) {
    circleX[i] = random(0, width);
    circleY[i] = random(0, height);
    circleVX[i] = random(-5, 5);
    circleVY[i] = random(-5, 5);
  }
}
```

```
void draw() {
  size(400, 300);
  for(int i=0; i<10; i++) {
    circleX[i] += circleVX[i];
    circleY[i] += circleVY[i];
    if(circleX[i] < 0) {
      circleX[i] = -circleX[i];
      circleVX[i] = -circleVX[i];
    }
    if(circleY[i] < 0) {
      circleY[i] = -circleY[i];
      circleVY[i] = -circleVY[i];
    }
    :
    fill(255, 0, 0);
    circle(circleX[i], circleY[i], 30);
  }
}
```

気持ち悪さ



	0	1	2	3	4	5	6	7	8	9
circleX										
circleY										
circleVX										
circleVY										

 3番目の円

クラスの特徴



- 変数をまとめることができる

```
class CircleClass {
  float x;
  float y;
  float vx;
  float vy;
}

CircleClass[] circles
    = new CircleClass[10];

void setup() {
  size(400, 300);
  for(int i=0; i<10; i++) {
    circles[i] = new CircleClass();
    circles[i].x = random(0, width);
    circles[i].y = random(0, height);
    circles[i].vx = random(-5, 5);
    circles[i].vy = random(-5, 5);
  }
}
```

```
void draw() {
  size(400, 300);
  for(int i=0; i<10; i++){
    circles[i].x += circles[i].vx;
    circles[i].y += circles[i].vy;
    if(circles[i].x < 0){
      circles[i].x = -circles[i].x;
      circles[i].vx = -circles[i].vx;
    }
    if(circles[i].y < 0){
      circles[i].y = -circles[i].y;
      circles[i].vy = -circles[i].vy;
    }
    :
    fill(255, 0, 0);
    circle(circles[i].x, circles[i].y, 30);
  }
}
```

気持ち悪さを解消



	0	1	2	3	4	5	6	7	8	9
circleX										

	0	1	2	3	4	5	6	7	8	9
circleY										

	0	1	2	3	4	5	6	7	8	9
circleVX										

	0	1	2	3	4	5	6	7	8	9
circleVY										

 3番目の円

	0	1	2	3	4	5	6	7	8	9
circle				.x .y .vX .vy						

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



- 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

端で跳ね返るオブジェクト

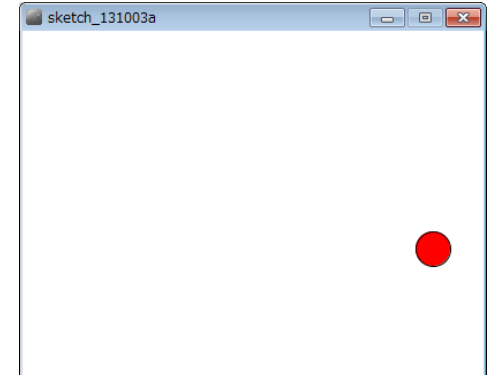
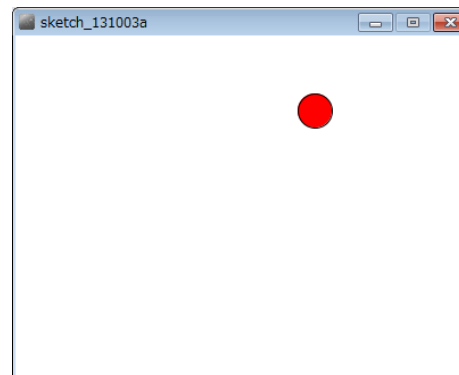
明治大学総合数理学部
先端メディアサイエンス学科
中村研究室



赤色の○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でランダムに設定される実数値とせよ

• 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
 - $v_x = -v_x$;
 - $v_y = -v_y$;



```
float x;  
float y;  
float vx;  
float vy;
```

```
void setup()
```

```
{  
  size(400, 300);  
  x = random(0, width);  
  y = random(0, height);  
  vx = random(-5, 5);  
  vy = random(-5, 5);  
}
```

今までの知識で
プログラムを組むと

```
void draw()
```

```
{  
  background(255);  
  x += vx;  
  y += vy;  
  if(x > width){  
    x = width * 2 - x;  
    vx = -vx;  
  }  
  if(x < 0){  
    x = -x;  
    vx = -vx;  
  }  
  if(y > height){  
    y = height * 2 - y;  
    vy = -vy;  
  }  
  if(y < 0){  
    y = -y;  
    vy = -vy;  
  }  
  fill(255, 0, 0);  
  circle(x, y, 30);  
}
```

```
float x;  
float y;  
float vx;  
float vy;
```

変数を定義

```
void setup()  
{  
  size(400, 300);  
  x = random(0, width);  
  y = random(0, height);  
  vx = random(-5, 5);  
  vy = random(-5, 5);  
}
```

場所と速度を設定

機能ごとに
分解すると

```
void draw()  
{  
  background(255);  
  x += vx;  
  y += vy;  
  if(x > width){  
    x = width * 2 - x;  
    vx = -vx;  
  }  
  if(x < 0){  
    x = -x;  
    vx = -vx;  
  }  
  if(y > height){  
    y = height * 2 - y;  
    vy = -vy;  
  }  
  if(y < 0){  
    y = -y;  
    vy = -vy;  
  }  
  fill(255, 0, 0);  
  circle(x, y, 30);  
}
```

移動

描画

```
float x;  
float y;  
float vx;  
float vy;
```

変数を定義

```
void setup()  
{  
  size(400, 300);  
  x = random(0, width);  
  y = random(0, height);  
  vx = random(-5, 5);  
  vy = random(-5, 5);  
}
```

場所と速度を設定
initialize()

機能ごとに
分解すると

```
void draw()
```

```
{  
  background(255);  
  x += vx;  
  y += vy;  
  if(x > width){  
    x = width * 2 - x;  
    vx = -vx;  
  }  
  if(x < 0){  
    x = -x;  
    vx = -vx;  
  }  
  if(y > height){  
    y = height * 2 - y;  
    vy = -vy;  
  }  
  if(y < 0){  
    y = -y;  
    vy = -vy;  
  }  
  fill(255, 0, 0);  
  circle(x, y, 30);  
}
```

移動: **move()**

描画: **display()**

```
float x, y, vx, vy;
```

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

```
void draw() {  
  background(255);  
  move();  
  display();  
}
```

```
void initialize() {  
  x = random(0, width);  
  y = random(0, height);  
  vx = random(-5, 5);  
  vy = random(-5, 5);  
}
```

```
void display() {  
  fill(255, 0, 0);  
  circle(x, y, 30);  
}
```

場所と速度を設定
initialize()

描画
display()

移動: move()

```
void move() {  
  x += vx;  
  y += vy;  
  if(x > width) {  
    x = width * 2 - x;  
    vx = -vx;  
  }  
  if(x < 0) {  
    x = -x;  
    vx = -vx;  
  }  
  if(y > height) {  
    y = height * 2 - y;  
    vy = -vy;  
  }  
  if(y < 0) {  
    y = -y;  
    vy = -vy;  
  }  
}
```

関数にしてみる

動きは図形に任せたい



- 丸、正方形、三角形を定義

- それぞれの座標は意識したくない

- `circle.x`, `circle.y`, `square.x`, `square.y`, `triangle.x`, `triangle.y`
- 内部で適当に処理してもらう

- それぞれの速度も意識したくない

- `circle.vx`, `circle.vy`, `square.vx`, `square.vy`, `triangle.vx`, `triangle.vy`

- 描画はシンプルにしたい

- `circle.display()`, `square.display()`, `triangle.display()`

- 移動もシンプルにしたい

- `circle.move()`, `square.move()`, `triangle.move()`

すべてを **XXXX . 機能** という形に！

オブジェクト指向（クラス）

CircleClass クラス



```
class CircleClass
```

```
{
```

```
float x;  
float y;  
float vx;  
float vy;
```

**変数を定義
クラス内に**

インスタンス変数

```
void initialize() {  
    x = random(width);  
    y = random(height);  
    vx = random(-5, 5);  
    vy = random(-5, 5);  
}
```

**場所と速度を設定
.initialize()**

インスタンスメソッド

```
void display() {  
    fill(255, 0, 0);  
    circle(x, y, 30);  
}
```

描画: .display()

インスタンスメソッド

```
void move() {
```

```
    x += vx;  
    y += vy;  
    if(x > width) {  
        x = width * 2 - x;  
        vx = -vx;  
    }  
    if(x < 0) {  
        x = -x;  
        vx = -vx;  
    }  
    if(y > height) {  
        y = height * 2 - y;  
        vy = -vy;  
    }  
    if(y < 0) {  
        y = -y;  
        vy = -vy;  
    }  
}
```

移動: .move()

インスタンスメソッド

```
}
```

CircleClass クラス



```
class CircleClass
```

```
{  
  float x;  
  float y;  
  float vx;  
  float vy;
```

変数を定義
クラス内に

インスタンス変数

```
void initialize() {  
  x = random(width);  
  y = random(height);  
  vx = random(-5, 5);  
  vy = random(-5, 5);  
}
```

場所と速度を設定
.initialize()

インスタンスメソッド

```
void display() {  
  fill(255, 0, 0);  
  circle(x, y, 30);  
}
```

描画: .display()

インスタンスメソッド

```
void move() {
```

```
  x += vx;  
  y += vy;  
  if(x > width) {  
    x = width * 2 - x;  
    vx = -vx;  
  }  
  if(x < 0) {  
    x = -x;  
    vx = -vx;  
  }  
  if(y > height) {  
    y = height * 2 - y;  
    vy = -vy;  
  }  
  if(y < 0) {  
    y = -y;  
    vy = -vy;  
  }  
}
```

移動: .move()

インスタンスメソッド

```
}
```



```
CircleClass circleObj = new CircleClass();
```

```
void setup()  
{
```

```
  size(400, 300);
```

```
  circleObj.initialize();
```

```
}
```

```
void draw()  
{
```

```
  background(255);
```

```
  circleObj.move();
```

```
  circleObj.display();
```

```
}
```

変数を定義
インスタンス化

場所と速度を設定
.initialize()

移動: .move()

描画: .display()

配列も簡単！



```
CircleClass[] circles = new CircleClass[10];
```

配列の定義

```
void setup()
```

```
{
```

```
  size(400, 300);
```

```
  for(int i=0; i<circles.length; i++) {
```

```
    circles[i] = new CircleClass();
```

```
    circles[i].initialize();
```

```
  }
```

```
}
```

インスタンス化

場所と速度を設定
.initialize()

```
void draw()
```

```
{
```

```
  background(255);
```

```
  for(int i=0; i<circles.length; i++) {
```

```
    circles[i].move();
```

```
    circles[i].display();
```

```
  }
```

```
}
```

移動: .move()

描画: .display()

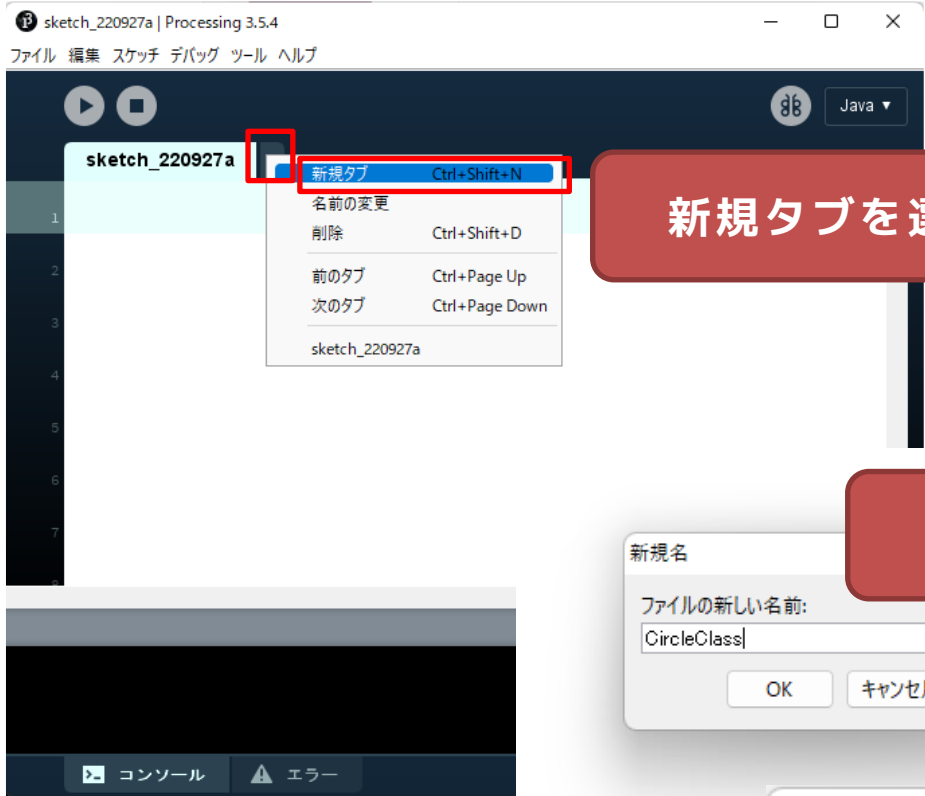
プログラムを動かそう！

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



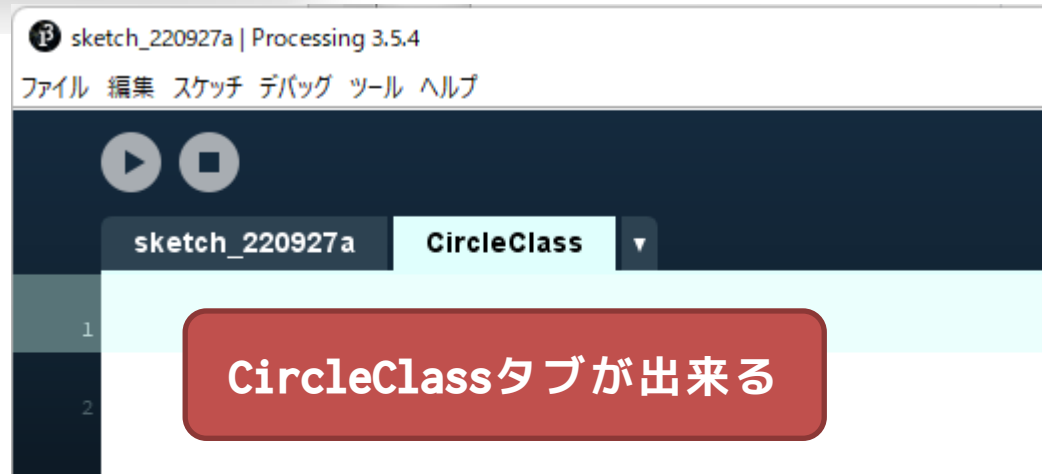
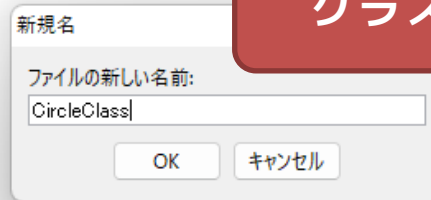
- <https://lecture.nkmr.io/> の第2回講義資料にある CircleClass.txt を利用しよう
 - CircleClass の部分は別のタブに！（次ページで説明）

クラスを作るときは別タブで



新規タブを選択

クラス名を入力



CircleClassタブが出来る

大きさの違う丸

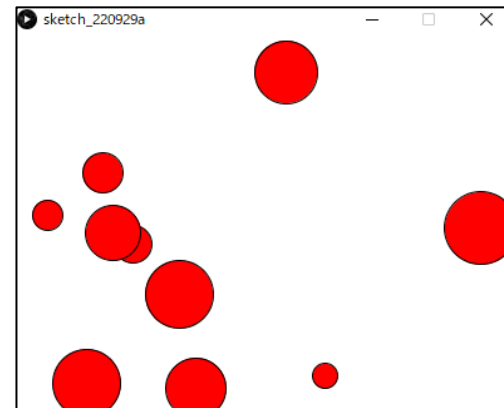


赤色で半径がランダムに10~30の値で設定される○
が10個画面内を動き回るプログラムを作成せよ

ただし、位置は画面内で、速度はx、y方向それぞれ-5~5でランダムに設定される実数値とせよ

• 考え方

- CircleClass を改良して大きさを変数として導入する
- 変数（インスタンス変数と呼ぶ）として r を追加してみよう！



CircleClass クラス改



```
class CircleClass
```

```
{
```

```
float x;
```

```
float y;
```

```
float vx;
```

```
float vy;
```

```
int r;
```

インスタンス変数を追加

```
void initialize() {
```

```
    x = random(width);
```

```
    y = random(height);
```

```
    vx = random(-5, 5);
```

```
    vy = random(-5, 5);
```

```
    r = (int)random(10, 30);
```

```
}
```

.initialize()で
半径も設定

```
void display() {
```

```
    fill(255, 0, 0);
```

```
    circle(x, y, r*2);
```

```
}
```

.display()で半径
を使って描画

```
void move() {
```

```
    x += vx;
```

```
    y += vy;
```

```
    if(x > width) {
```

```
        x = width * 2 - x;
```

```
        vx = -vx;
```

```
    }
```

```
    if(x < 0) {
```

```
        x = -x;
```

```
        vx = -vx;
```

```
    }
```

```
    if(y > height) {
```

```
        y = height * 2 - y;
```

```
        vy = -vy;
```

```
    }
```

```
    if(y < 0) {
```

```
        y = -y;
```

```
        vy = -vy;
```

```
    }
```

```
}
```

```
}
```

宿題1-1: hw_boundA11



- 10個の赤色の○と、5個の青色の□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、丸も正方形も壁で跳ね返るようにせよ。跳ね返り方は図形の端でも図形の中央でもよ。

クラスで解決しよう！

SquareClass



```
class SquareClass
```

```
{
```

```
float x;  
float y;  
float vx;  
float vy;
```

変数を定義
クラス内に

```
void initialize() {  
    x = random(width);  
    y = random(height);  
    vx = random(-5, 5);  
    vy = random(-5, 5);  
}
```

場所と速度を設定
.initialize()

```
void display() {  
    fill(0, 0, 255);  
    rect(x-15, y-15, 30, 30);  
}
```

描画: .display()

```
void move() {  
    x += vx;  
    y += vy;  
    if(x > width) {  
        x = width * 2 - x;  
        vx = -vx;  
    }  
    if(x < 0) {  
        x = -x;  
        vx = -vx;  
    }  
    if(y > height) {  
        y = height * 2 - y;  
        vy = -vy;  
    }  
    if(y < 0) {  
        y = -y;  
        vy = -vy;  
    }  
}
```

移動: .move()

丸と正方形の描画



```
CircleClass circleObj = new CircleClass();  
SquareClass squareObj = new SquareClass();
```

変数を定義
インスタンス化

```
void setup()
```

```
{
```

```
  size(400, 300);
```

```
  circleObj.initialize();
```

```
  squareObj.initialize();
```

```
}
```

場所と速度を設定
.initialize()

```
void draw()
```

```
{
```

```
  background(255);
```

```
  circleObj.move();
```

```
  squareObj.move();
```

```
  circleObj.display();
```

```
  squareObj.display();
```

```
}
```

移動: .move()

描画: .display()

丸と正方形の描画：配列



```
CircleClass[] circles = new CircleClass[10];  
SquareClass[] squares = new SquareClass[5];
```

配列の定義

```
void setup()  
{  
  size(400, 300);  
  for(int i=0; i<circles.length; i++){  
    circles[i] = new CircleClass();  
    circles[i].initialize();  
  }  
  for(int i=0; i<squares.length; i++){  
    squares[i] = new SquareClass();  
    squares[i].initialize();  
  }  
}
```

インスタンス化

場所と速度を設定
.initialize()

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

移動: .move()

描画: .display()

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関数
 - 変数の初期化などによく利用される

```
class CircleClass
{
    float x;
    float y;
    float vx;
    float vy;
    CircleClass() {
    }
}
```

コンストラクタ！



```
class CircleClass {  
    float x;  
    float y;  
    float vx;  
    float vy;  
  
    CircleClass() {  
        initialize();  
    }  
  
    void initialize(){  
        x = random(width);  
        y = random(height);  
        vx = random(-5, 5);  
        vy = random(-5, 5);  
    }  
};
```

```
CircleClass circleObj  
= new CircleClass();
```

CircleClass()使っ
てインスタンス作
ってるよってこと

```
void setup()  
{  
    size(400, 300);  
    circleObj.initialize();  
    fill(255, 0, 0);  
}  
  
void draw()  
{  
    background(255);  
    circleObj.move();  
    circleObj.display();  
}  
  
void mousePressed()  
{  
    circleObj.click(mouseX, mouseY);  
}
```

コンストラクタで initialize
メソッドを呼び出す！

コンストラクタ！



```
class CircleClass {  
    float x;  
    float y;  
    float vx;  
    float vy;
```

```
    CircleClass() {  
        x = random(width);  
        y = random(height);  
        vx = random(-5, 5);  
        vy = random(-5, 5);  
    }  
    :
```

コンストラクタで
初期設定する！

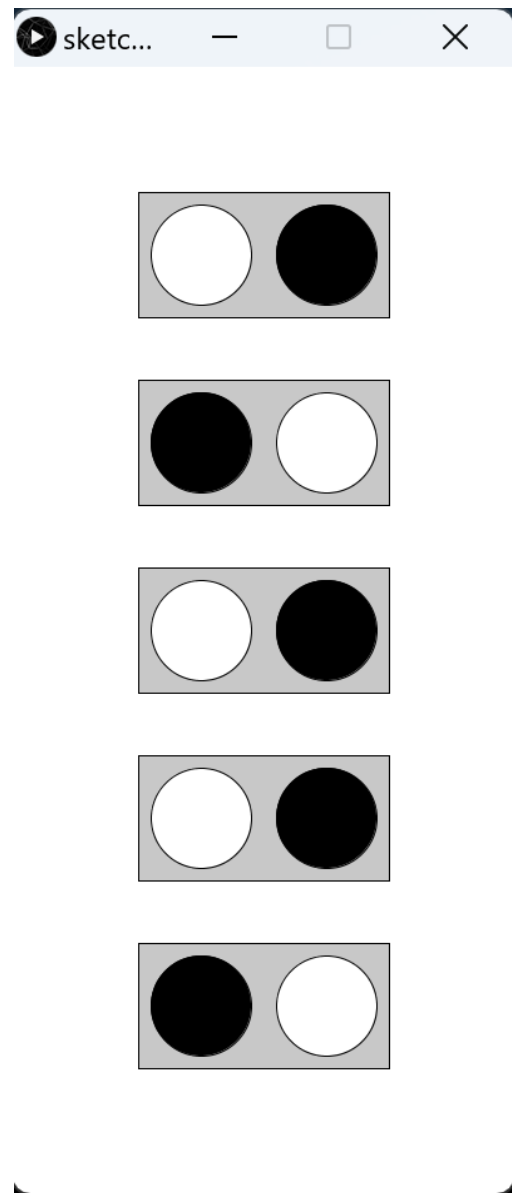
```
CircleClass circleObj  
    = new CircleClass();  
  
void setup()  
{  
    size(400, 300);  
    circleObj.initialize();  
    fill(255, 0, 0);  
}  
  
void draw()  
{  
    background(255);  
    circleObj.move();  
    circleObj.display();  
}  
  
void mousePressed()  
{  
    circleObj.click(mouseX, mouseY);  
}
```

独立したラジオボタン

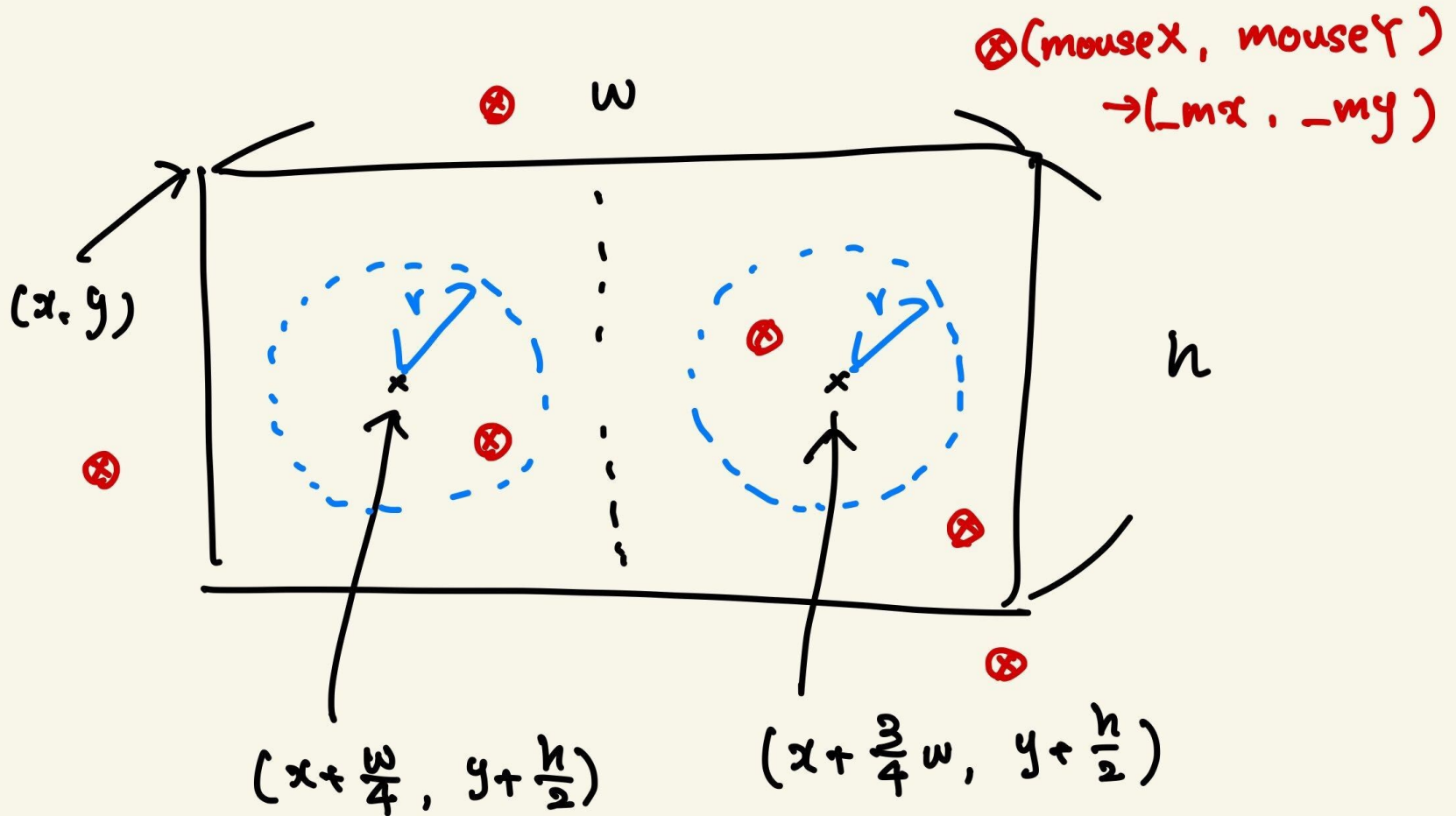


2つの選択肢をもつ、独立したラジオボタンUIのパーツを作成する

- UIのパーツは左右にボタン（丸）が2つならんでおり、四角形により囲まれている
- クリックされたボタン（丸）が黒色になり、他方のボタン（丸）が白色になるという挙動をする
- このパーツをウィンドウ内に縦に5つ並べよ



ボタンの当たり判定



```

class RadioButtonClass {
    int x, y;
    int w, h, r;
    int leftOrRight;

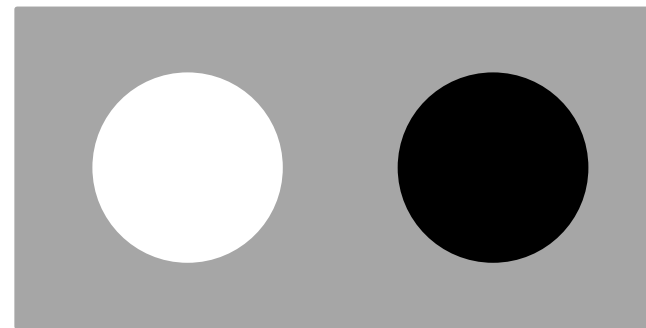
    RadioButtonClass() {
        x = 0;
        y = 0;
        w = 200;
        h = 100;
        r = 40;
        // 0ならleft, 1ならrightとする
        leftOrRight = 0;
    }

    void display() {
        fill(200, 200, 200);
        rect(x, y, w, h);
        if(leftOrRight == 0) fill(0);
        else fill(255);
        circle(x+w/4, y+h/2, r*2);
        if(leftOrRight == 1) fill(0);
        else fill(255);
        circle(x+w*3/4, y+h/2, r*2);
    }
}
    
```

```

void setPos(int _x, int _y) {
    x = _x;
    y = _y;
}

// (_mx, _my)の位置がクリックされた
void click(int _mx, int _my) {
    if(dist(_mx, _my, x+w/4, y+h/2) <= r) {
        leftOrRight = 0;
    } else if(dist(_mx, _my, x+w*3/4, y+h/2) <= r) {
        leftOrRight = 1;
    } else {
        // ボタンの中じゃないので何もしない(不要)
    }
}
    
```



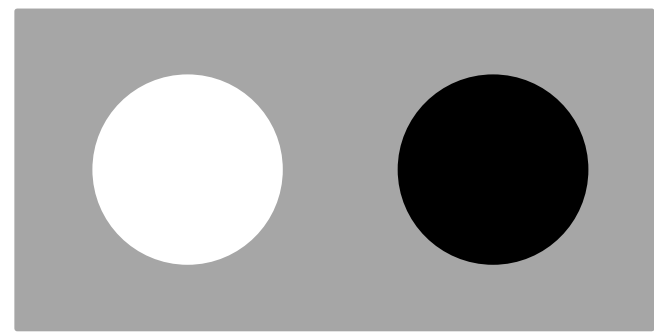
```
class RadioButtonClass {
  int x, y;
  int w, h, r;
  int leftOrRight;

  RadioButtonClass() {
    x = 0;
    y = 0;
    w = 200;
    h = 100;
    r = 40;
    // 0ならleft, 1ならrightとする
    leftOrRight = 0;
  }

  void display() {
    fill(200, 200, 200);
    rect(x, y, w, h);
    fill(255 * leftOrRight);
    circle(x+w/4, y+h/2, r*2);
    fill(255 * (1 - leftOrRight));
    circle(x+w*3/4, y+h/2, r*2);
  }
}
```

```
void setPos(int _x, int _y) {
  x = _x;
  y = _y;
}

// (_mx, _my)の位置がクリックされた
void click(int _mx, int _my) {
  if(dist(_mx, _my, x+w/4, y+h/2) <= r) {
    leftOrRight = 0;
  } else if(dist(_mx, _my, x+w*3/4, y+h/2) <= r) {
    leftOrRight = 1;
  } else {
    // ボタンの中じゃないので何もしない(不要)
  }
}
}
```



配列 + インスタンス化

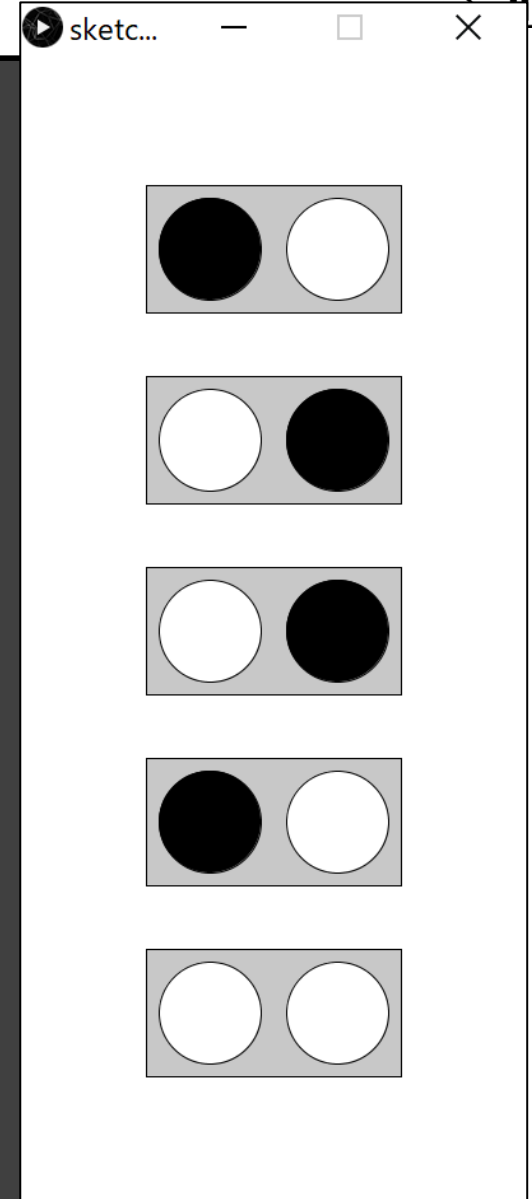


```
RadioButtonClass[] UIs = new RadioButtonClass[5];
```

```
void setup(){  
  size(400, 900);  
  for(int i=0; i<5; i++){  
    UIs[i] = new RadioButtonClass();  
    UIs[i].setPos(100, 100+150*i);  
  }  
}
```

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

```
void mousePressed(){  
  for(int i=0; i<UIs.length; i++){  
    UIs[i].click(mouseX, mouseY);  
  }  
}
```





- オブジェクト指向のさわりを学んだ
 - インスタンス化
 - `CircleClass circleObj = new CircleClass();`
 - インスタンス
 - `circleObj`
 - インスタンス変数
 - `circleObj.speed`
 - インスタンスメソッド
 - `circleObj.move()`
 - コンストラクタ
 - `CircleClass(){ 初期化処理 }`