2020年9月2日(水)、3日(木)、4日(金)

黒石商業高校

Java言語を利用したゲームアプリケーションソフトウェアの開発

担当：青森大学ソフトウェア情報学部 角田均

協力：八戸工業大学工学部システム情報工学科 小久保温

サポート：青森大学ソフトウェア情報学部

大坂稜弥、横山正市、天内葉月、井熊翔

プロセシング Processing入門

デザイナーとアーティストのための
プログラミング環境

Processingとは？

- MITメディアラボで、ケイシー・リースとベン・フライが開発
  - コンピュータグラフィックス、VR/AR、アートなどの研究所
- デザイナー、アーティスト、プログラミング初心者向けの開発環境
  - とてもシンプルにプログラムが書ける
  - 本当に必要なものだけ書けばいい！

Processingの特徴

- Javaがベース
  - Processingを学ぶと、Javaがよくわかる
  - Javaで学んだことが、ほぼそのまま使える
- オープン・ソース・ソフトウェア
  - 無料でダウンロードできる

https://processing.org/

Java対Processing

- Java

```
class Sample {
    public static void main(String[] args) {
        System.out.println("こんにちは");
    }
}
```

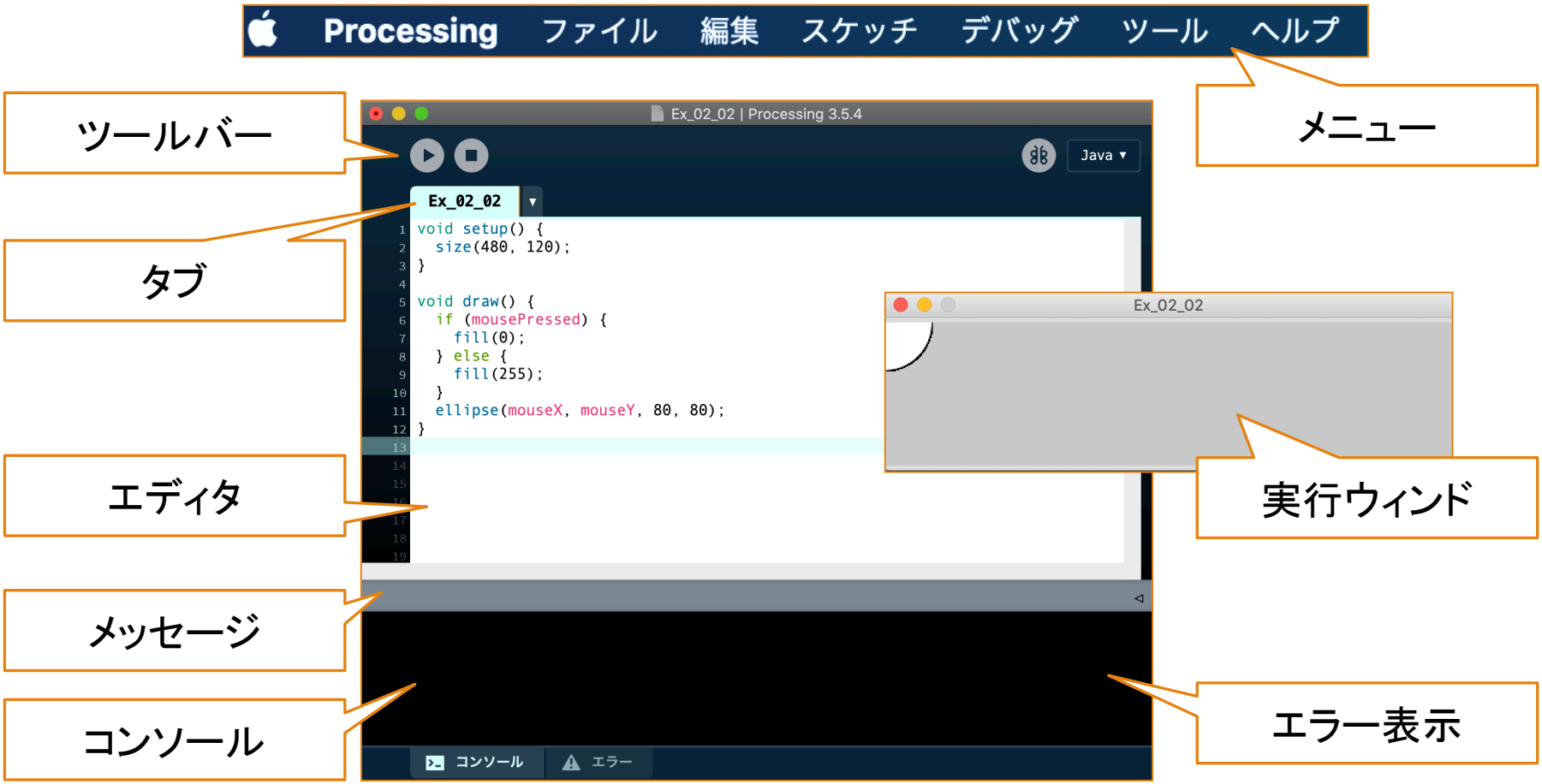
この長い呪文が

- Processing

```
println("こんにちは");
```

これだけでOK！

Processing開発環境



プログラムの作り方

- Processing開発環境を起動
- テキスト・エディタの部分にプログラムを入力
  - Processingでは、プログラムのことを「スケッチ」という
- 実行ボタンを押す

Processingのプログラム

- 基本はJavaと同じ
- ただし、余計なものは書かなくていい

```
for (int i = 0; i < 10; i++) {
  println(i);
}
```

Processingのfor文

自動フォーマット機能

- [編集]→[自動フォーマット]
  - プログラムのインデントを整形してくれる

```
for(int i=0;i<10;i++){println(i);}
```

```
for (int i=0; i<10; i++) {
  println(i);
}
```

ショートカットもOK
⌘T

Processingの限界

- コンソールへ文字の入力ができない
  - 例: 以下のようなプログラムはエラーは出ないが、事実上使えない

```
import java.io.*;
...
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
String str = br.readLine();
...
```

  - 後でやるが、キーボードの入力を取得することはできる
- クラス変数やクラス・メソッドは素直には使えない
  - 内部クラスというものを使用しているため

1. println()関数

- 1行表示する　※println: print line 1行表示

```
println(表示したい内容);
```

  - JavaのSystem.out.println()と同じ

- 例

```
println("こんにちは");
```

コメント： Javaと同じ

- // 1行コメント

```
// iを0に
int i = 0;
int j = 0; // jを0に
```

- /* */ 複数行コメント

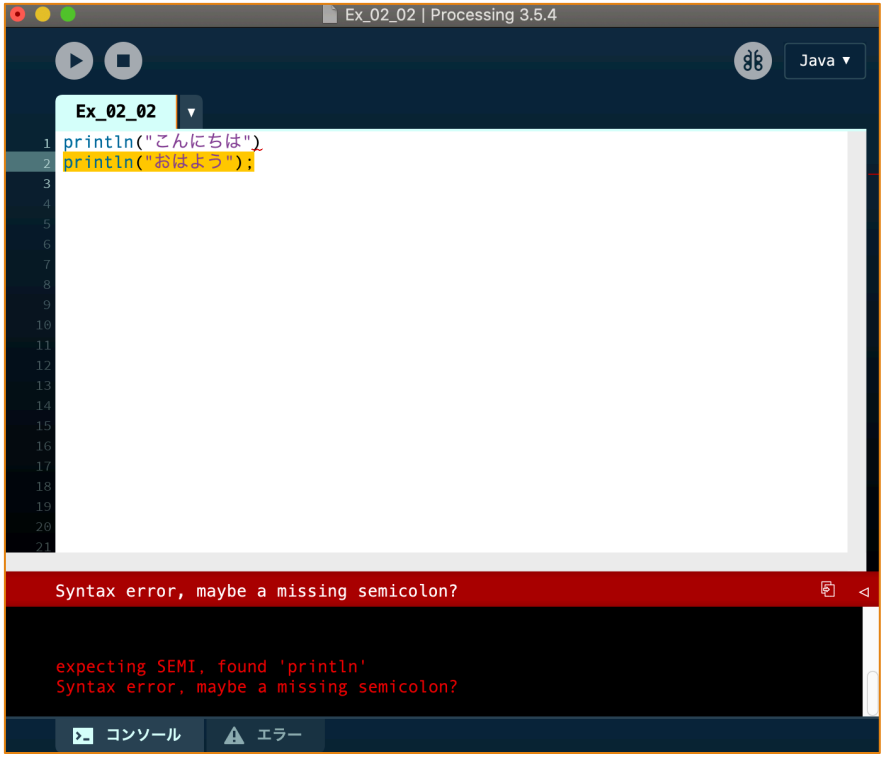
```
/*
   ここがコメント
*/
```

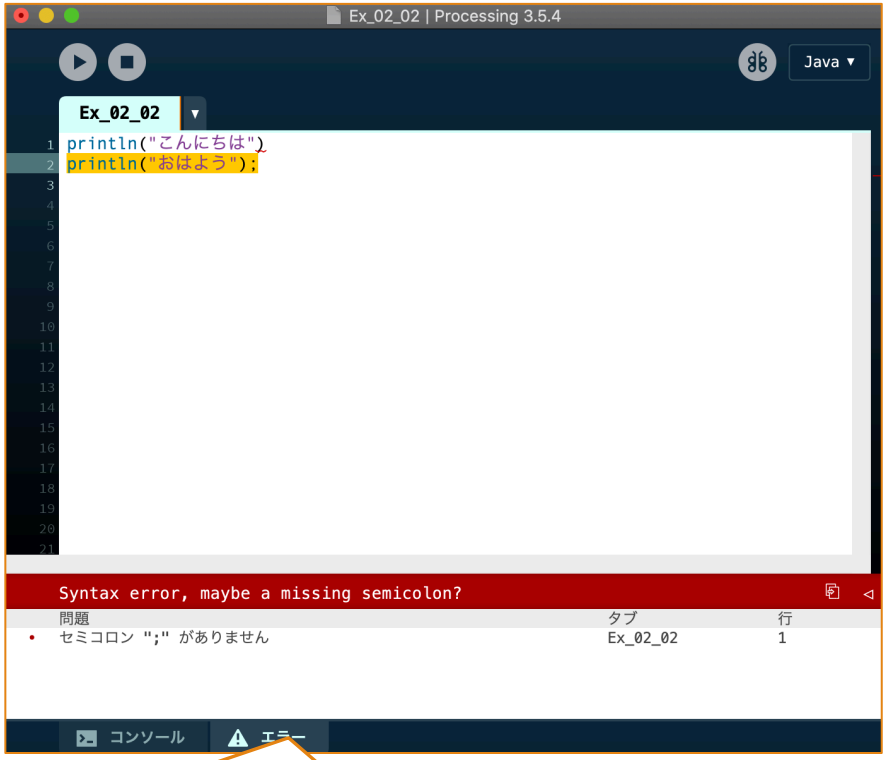
文末記号： Javaと同じ

- 文の終わりには、「;」をつける
  - 「；」… セミコロン

エラーの表示

- エラーの場所、内容





エラー表示タブに切り替え

描画

Processingの得意技

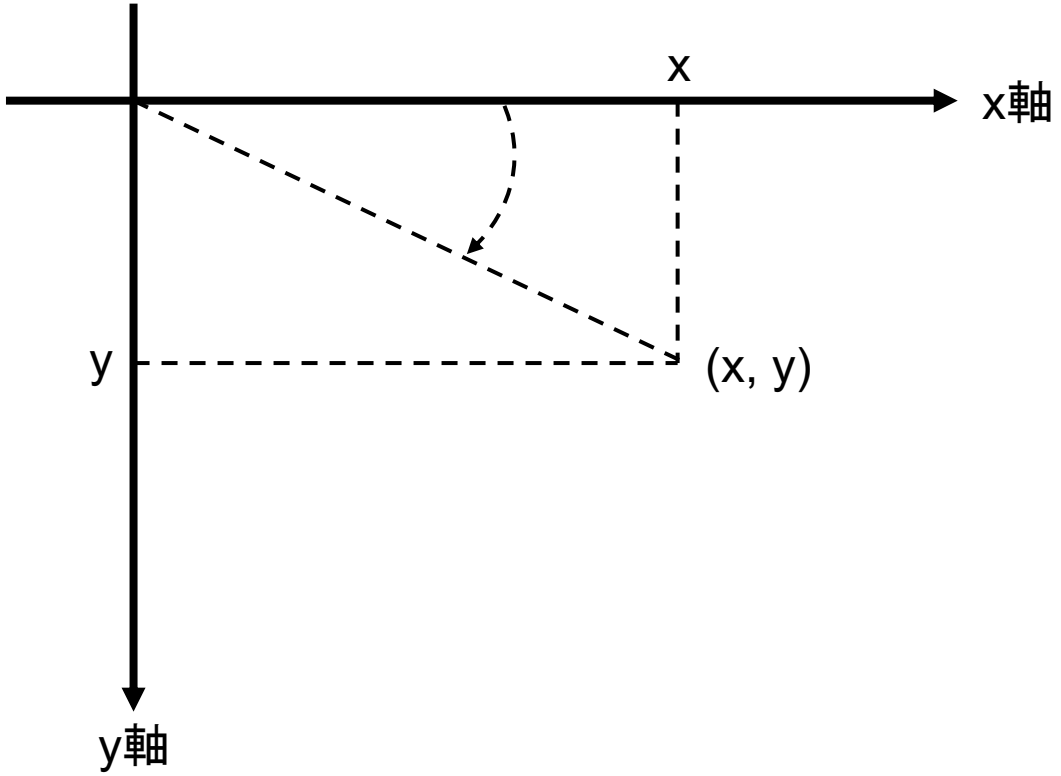
2. ディスプレイ・ウィンドウの大きさ

- ディスプレイ・ウィンドウの大きさを指定
  - size(幅, 高さ);

- 注: 以降も、関数の引数は「横, 縦」の順番で指定することがほとんど

座標系

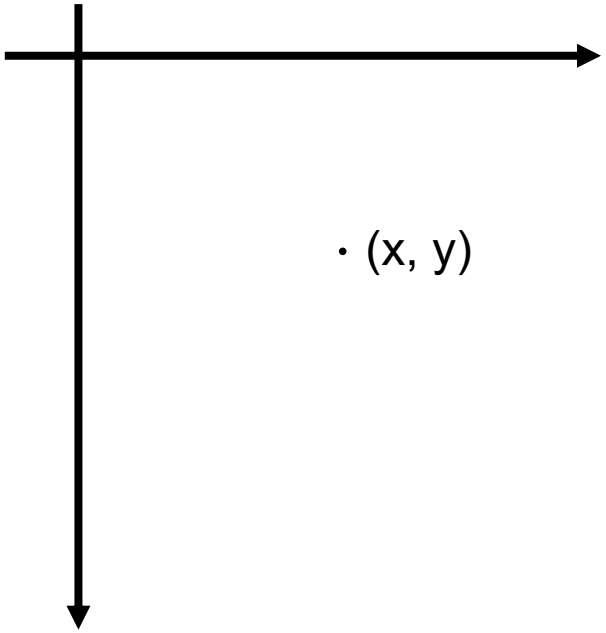
- 高校の数学とは上下と回転方向が逆



図形： 1

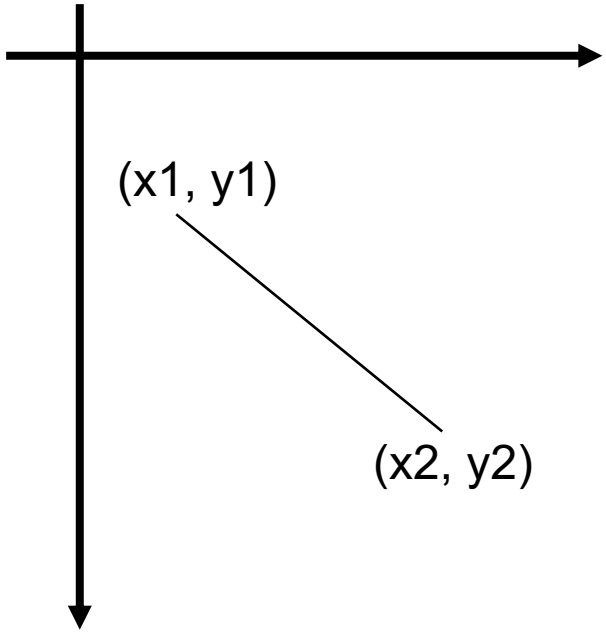
3.1 point(x, y);

- (x, y)に点を描く



3.2 line(x1, y1, x2, y2);

- (x1, y1)から(x2, y2)に線を描く

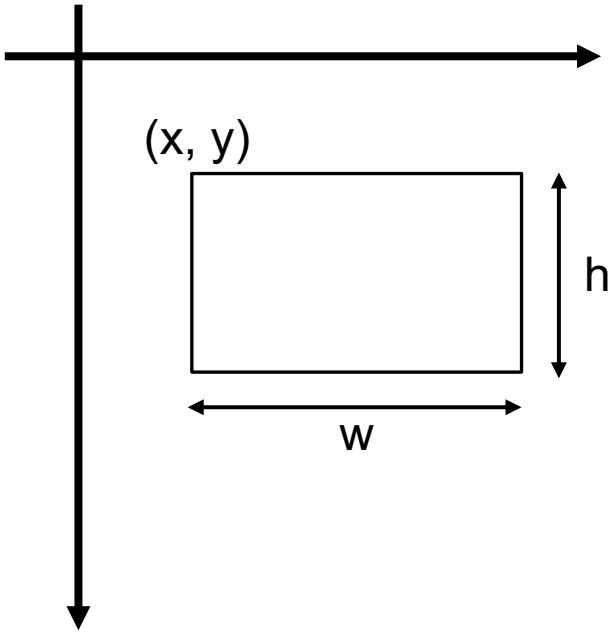


図形： 2

3.3 rect(x, y, w, h);

※rectangle: 長方形

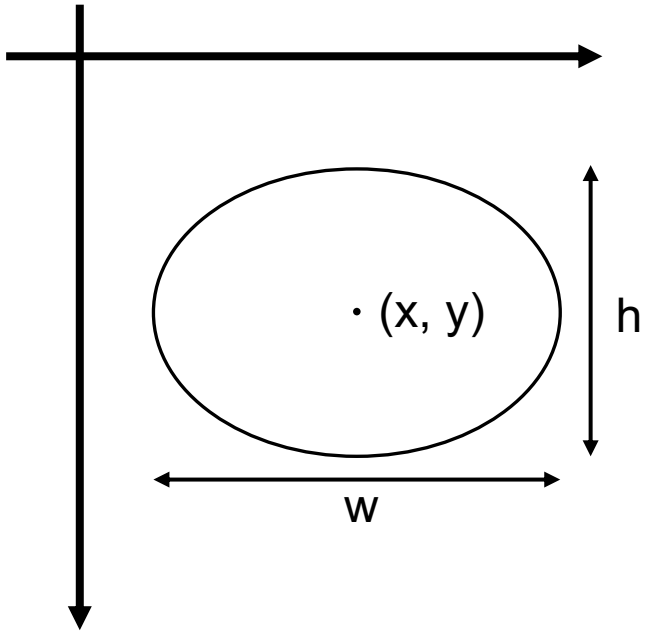
- 左上が(x, y)、幅w、高さhの長方形を描く



3.4 ellipse(x, y, w, h);

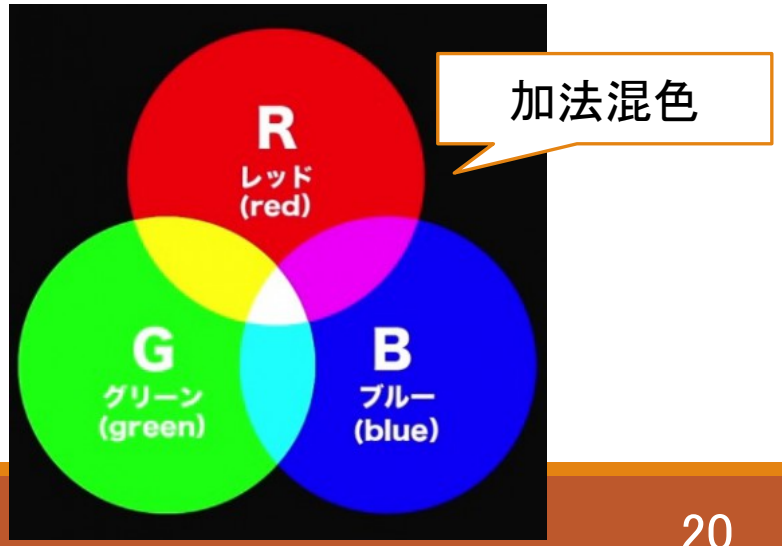
※ellipse: 楕円形

- 中心が(x, y)、幅w、高さhの楕円を描く



色

- コンピュータでは、RGB(赤緑青)の各色8bit(10進数では0～255)の表色系がよく用いられている。
- Processingでは、RGB(デフォルト)とHSBが使える
  ※メソッドのオーバーロードの活用
- RGBモードのとき、以下の4種類の方法で色が指定できる
  - (明るさ)
  - (明るさ, 不透明度)
  - (赤, 緑, 青)
  - (赤, 緑, 青, 不透明度)



背景、塗りつぶしと枠線

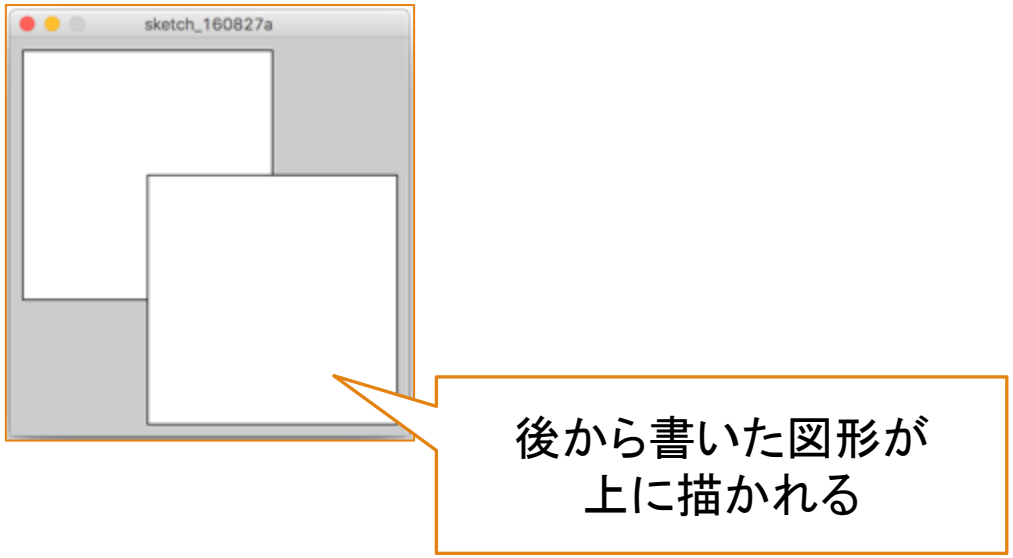
- 3.5 ディスプレイ・ウィンドウの背景
  - 色の指定: background(色);
- 塗りつぶし
  - 3.6 色の指定: fill(色);　　※fill: 塗り
  - 3.7 塗りつぶしなし: noFill();
- 枠線
  - 3.8 枠線の太さ: strokeWeight(太さ);　　※weight: 太さ
  - 3.9 色の指定: stroke(色);　　※stroke: 描線
  - 3.10 枠線なし: noStroke();

キャメル・ケースとスネーク・ケース

- 変数や関数の名前には、英単語が使われている
  - 単語を並べるとき、2種類の流儀がある
- キャメル・ケース (camel case: ラクダ式)
  - 単語の切れ目を大文字に
  - 例: noFill、
- スネーク・ケース(snake case: ヘビ式)
  - 単語の切れ目に「_」(アンダースコア)
  - 例: PI_HALF
- ProcessingやJava
  - 変数や関数はキャメル・ケース、定数はスネーク・ケース

3. 11　順次

- プログラムは書いた順に実行される



変数と型

Javaと同じ

4. 変数

- 変数は値をとっておいて、使いまわすときに使う
- 変数の準備
  - 変数は使う前に宣言する必要がある
- 変数の宣言

```
型 名前;
```

- 整数の”a”と浮動小数点数の”x”と”y”を宣言する例

```
int a;
float x, y;
```

型

種類	型	リテラル(データの書き方)
整数	int	0, 24, -18
浮動小数点数	float	12.234, 0.0012, -99.021
~~倍精度浮動小数点数~~	~~double~~	~~12.234, 0.0012, -99.021~~
真偽値	boolean	true, false
文字(1文字)	char	'a', 'A'
文字列	String	"Hello, world!", "こんにちは"
画像	PImage	
フォント	PFont	

整数: integer
浮動小数点数: floating point number
倍精度: double precision
　※Processingではdoubleは使わない方が無難
文字: character

プリミティブとオブジェクト

- プリミティブ ※primitive 素朴な
  - 単純に値だけを持っている
  - 型名は小文字ではじまる: int、float、boolean、char
- オブジェクト(後で詳しく)
  - 複数の値を持っている
  - メソッドを持っている
  - 型名は大文字ではじまる: String、PImage、PFont

4.5 ディスプレイ・ウィンドウ変数

- ディスプレイ・ウィンドウの大きさ
  - width: 幅　　ウィドゥス
  - height: 高さ　ハイト

5. 演算子

Javaと同じ

演算子

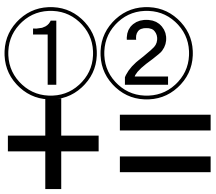
種類	演算子	例
代入(右辺を左辺に)	=	x = 3
足す	+	x + 3
引く	-	x - 3
掛ける	*	x * 3
割る	/	x / 3
割った余り	%	x % 3
文字列の結合	+	"Hello" + "world"

複合代入演算子

- 計算して代入

```
x += 10; // xに10を足し、それをxに代入
y -= 15; // yから15を引き、それをyに代入
```

- 読み方



①足して、②代入する

インクリメント、デクリメント演算子

- インクリメント: 1つ増やす

※increment: 増加

```
x++; // xを1つ増やす
```

- デクリメント: 1つ減らす

※decrement: 減少

```
x--; // xを1つ減らす
```

5.2 演算と型

- 整数と整数の演算→整数
  - 3 / 2 → 1
- 小数と小数の演算→小数
  - 3.0 / 2.0 → 1.5
- 整数と小数の演算→小数 ※自動型変換
  - 3 / 2.0 → 1.5

自動型変換

- 自動的に型が変換されることがある
  - 「小数と整数の演算」や「小数←整数の代入」は小数になる

```
float x = 2;
```
→　xは2.0になる

  - 「+」演算子は、文字列が混ざると文字列になる

```
"合計は" + 1 + 3
```
→
```
"合計は13"
```

    - 計算は()でくくっておくとよい

```
"合計は" + (1 + 3)
```
→
```
"合計は4"
```

5.3 代入の制限

- データが落ちるような代入はできない
  - 「整数←小数の代入」はダメ
  - 型を変換する必要がある
    - int()関数を利用

```
整数 = int(小数);
```

    - キャストを利用

```
整数 = (int)小数;
```

単精度と倍精度の浮動小数点数

- 浮動小数点数には、データ量の異なる2種類がある
  - 単精度 32bit float
  - 倍精度 64bit double
- Processingでは、小数にはfloatが使用されている
  - 関数の引数などにdoubleを使用するとエラーが出る
  - 「(float)倍精度浮動小数点数」のようにキャストを利用する

```
double x = 100.0;
double y = 100.0;
double d = 20.0;
ellipse((float)x, (float)y, (float)d, (float)d);
```

6. くりかえし

Javaと同じ

Processingのくりかえし

- Javaと同じで、以下のくりかえしが使える
  - for文
  - while文
  - do～while文
- ※ forもwhileも「～の間」という意味

6.1 for文

- for文の書き方

```
for（初期設定；くりかえしを続ける条件；変化）{
  くりかえしたい内容
}
```

- 例

```
for (int i = 0; i < 10; i++) {
  println(i);
}
```

条件式

- 条件式の書き方

```
値  関係演算子  値
```

- 例

```
x > 0
```

- 注意
  - 条件式は2つのものの比較しか書けない。
  - 組み合わせたい場合は、論理演算子や()を使ってつなげる

```
×  0 < x < 10
◯  (0 < x) && (x < 10)
```

関係演算子

種類	演算子	例
等しい	==	
等しくない	!=	
より大きい	>	
以上	>=	
以下	<=	
より小さい	<	

数学の記号のうち、キーボードにないもの(≦、≧)に注意

論理演算子

- かつ &&
- または ||
- ではない！
- 条件式は、2つずつしか比較できない
  - 例: 0以上10以下
    - ×: 0 <= x <= 10
    - ◯: (0 <= x) && (x <= 10)

7. アニメーション

パラパラまんが

コンピュータによるアニメーション

- 「パラパラまんが」と同じ
  - 仮現運動
    - 1コマ1コマを高速に切り替えて、「あたかも動いているかのよう」に見せる
- 1コマ1コマのことを「フレーム」という

※frame: コマ

Processingによるアニメーション

- プログラムの書き方

```
全体で使用する変数の宣言

void setup() {
  初期設定(最初に1回だけ実行する内容)
}

void draw() {
  毎フレーム実行する内容
}
```

7.2 フレーム・レート

- フレーム・レート(1秒当たりのコマ数)の設定

`frameRate(値)` ※rate: 割合

- フレーム・レートの実測値

`変数frameRate`

7.3 経過時間

- 経過時間 millis()

※milli second ミリ秒

  - 単位はミリ秒
  - 1秒=1000ミリ秒

- 経過時間を秒で取得したい場合
  - millis() / 1000

7.4 マウス変数

- マウスの現在位置
  - mouseX、mouseY
- マウスの1フレーム前の位置
  - pmouseX、pmouseY

※previous 前の

7.5 残像とbackground()関数

- Processingでは、描画は上書きされていく
- そのままでは残像が残る
- 残像を消すには、draw()関数の中で background()関数を実行する

8. 条件分岐

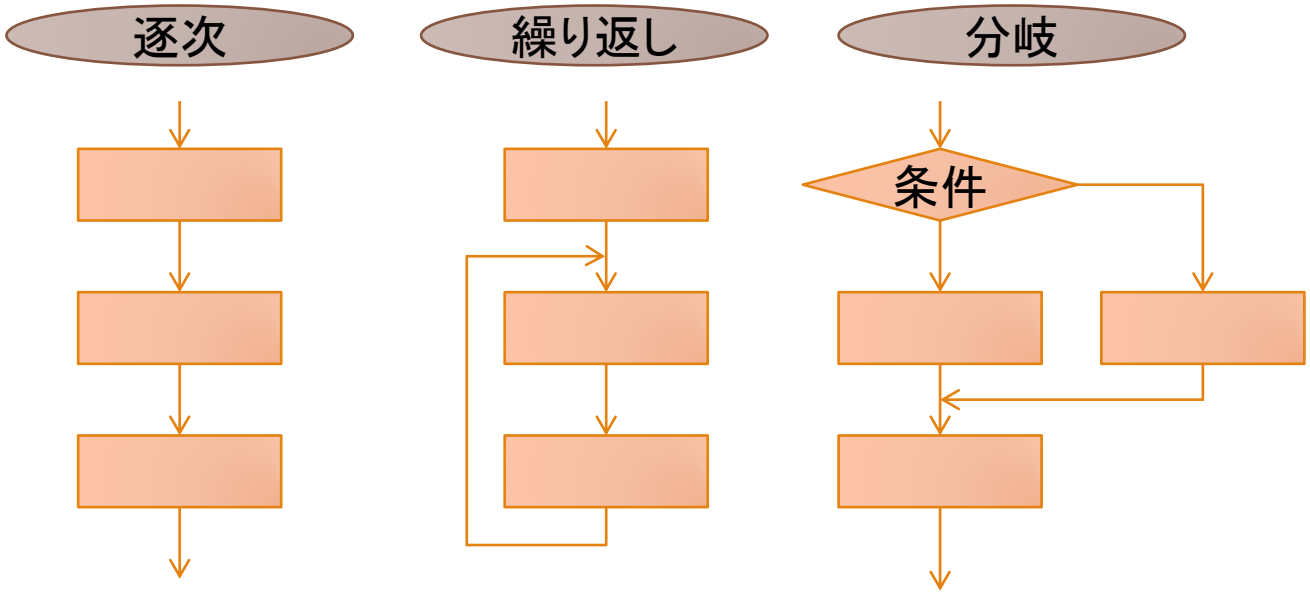
Javaと同じ

Processingの条件分岐

- Javaと同じで、以下の条件分岐が使える
  - if～else文
  - switch文

構造化プログラミング

- 3つの制御構造（プログラムの流れ）



どんなプログラムでも3つの流れの組み合わせで表現できる

エドガー・ダイクストラ
（1930-2002）

9. マウス

マウスの位置、マウスのクリック

9.1 マウス変数

- マウスのクリック mousePressed
  - 押している true、押していない false

※press 押す

- 最後にクリックしたボタン mouseButton
  - 左 LEFT、真ん中 CENTER、右 RIGHT
- マウスの現在位置 mouseX、mouseY
- マウスの1フレーム前の位置 pmouseX、pmouseY

9.2 マウス関数

- マウスの操作時に呼び出される関数
  - クリックした　　mouseClicked()
  - ボタンを押した　　mousePressed()
  - ボタンを離した　　mouseReleased()
  - 移動した　　mouseMoved()
  - ドラッグ操作　　mouseDragged()
  - ホイール操作　　mouseWheel()

真偽値の条件判定

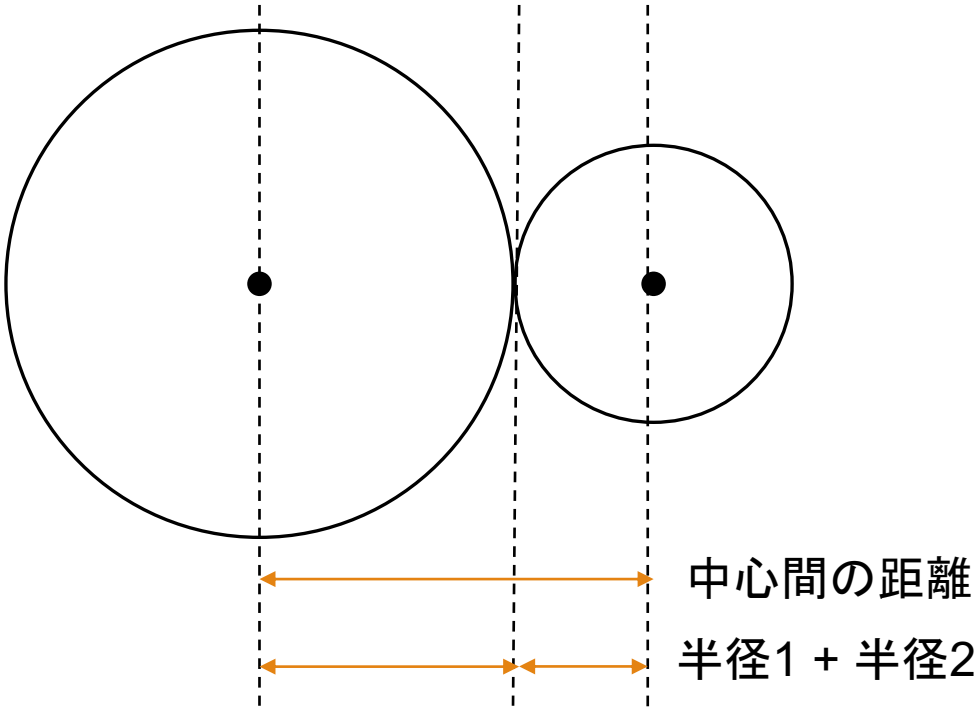
- if (mousePressed)と

if (mousePressed == true)は同じ

mousePressed	mousePressed == true
true	true
false	false

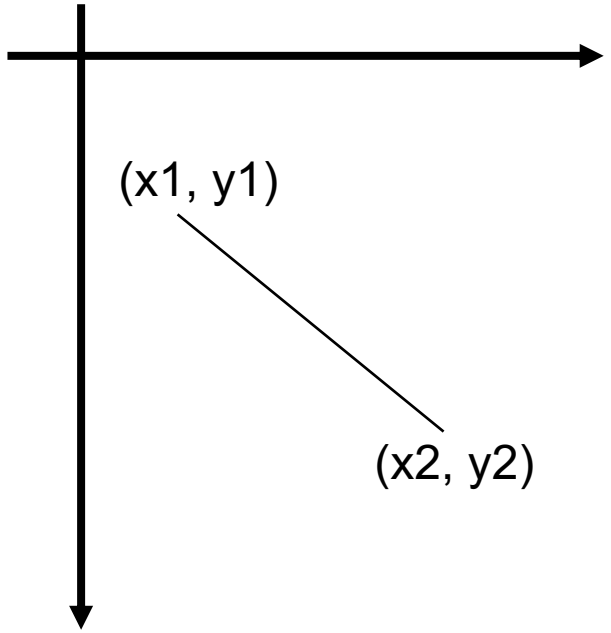
9.2 円形の当たり判定

- if (中心間の距離 < 半径1 + 半径2)



dist() 関数 ※distance: 距離

- 点(x1, y1)と点(x2, y2)の間の距離
  - dist(x1, y1, x2, y2)



10. キー

キーの押し下げ、押したキー

10.1 キー変数

- キーの押し下げ keyPressed
  - 押している true、押していない false
- 押したキー key
  - 'a'、'b'、...
  - 修飾キーの場合は値がCODEDになる
- 押した修飾キー keyCode
  - ←LEFT、→RIGHT、↑UP、↓DOWN、SHIFT、SPACE、ALTほか
- 注意
  - 今押しているキーではなく、「最後に」押したキーのデータが入っている

10.2 キー関数

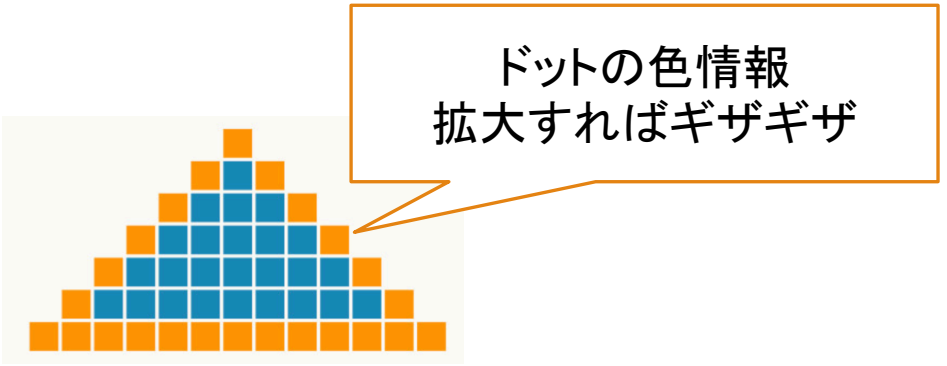
- キーボードの操作時に呼び出される関数
  - キー入力した　　keyTyped()
  - キーを押した　　keyPressed()
  - キーを離した　　keyReleased()

11. 画像

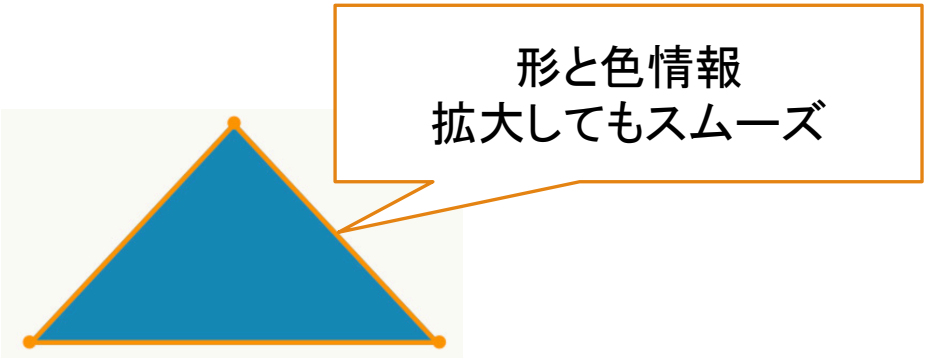
画像を表示しよう

コンピュータで扱う画像

- ラスター画像



- ベクター画像



Processingで利用できる画像形式

- ラスター画像

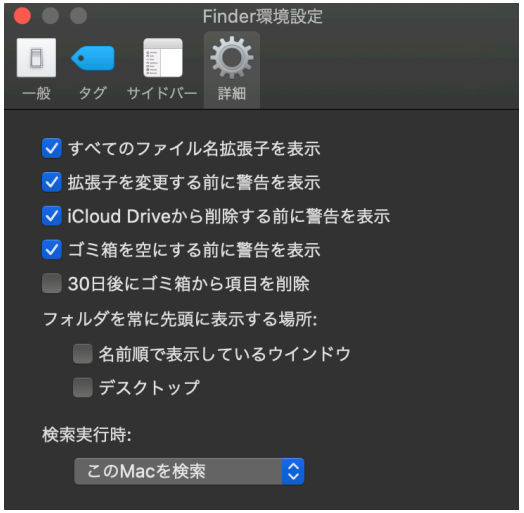
種類	特徴
GIF	256色まで。アニメーションができる。透明にできる。
JPEG	およそ1670万色。写真に適している。ファイルサイズが小さくなる。
PNG	およそ1670万色。半透明にできる。

- ベクター画像

種類	特徴
SVG	拡大縮小しても粗くならない

画像の用意

- 画像の拡張子を表示するようにFinderを設定
  - Finder → 環境設定(⌘,) → 詳細
  - 「すべてのファイル名拡張子を表示」



- 画像をダウンロードする
  - 指定したフォルダか[ダウンロード]フォルダなどに保存される

画像のプログラミング

- 画像をProcessingのスケッチにドラッグ・アンド・ドロップ
- 画像を表す変数を宣言

```
PImage 変数名;
```

- 変数に画像を読み込む

```
変数名 = loadImage("画像ファイル名");
```

※画像ファイル名は大文字・小文字を区別するので注意

- 画像を表示する

```
image(変数名, 左上のx座標, y座標);
image(変数名, 左上のx座標, y座標, 幅, 高さ);
```

11.2 imageMode()関数

- image()関数で画像を表示するとき、指定する座標を設定
  - 画像の左上の座標を指定(デフォルト)
    - imageMode(CORNER);
  - 画像の中心の座標を指定
    - imageMode(CENTER);

12.　フォントと文字列の表示

文字列を表示しよう

フォントのプログラミング

- フォントを表す変数を宣言

```
PFont 変数名;
```

- 変数にフォントを作って読み込む

```
変数名 = createFont(フォント名, サイズ);
```

※サイズの単位はポイント
PCでは1ポイントは0.75ピクセル(ドット)相当

- フォントを指定する

```
textFont(変数名);
```

文字列を表示する

- 塗りつぶしの色を指定

```
fill(色);
```

- 文字列の表示

```
text(文字列, x座標, y座標);
```

- 注意
  - 文字列の座標はベースラインの左端

13. 乱数

一見でたらめに見える数を作る

乱数

- 実は奥が深い問題
  - 決まったことしかできないコンピュータで乱数を作る？？
  - 実際には「擬似」乱数
- 作り方
  - 線形合同法、メルセンヌ・ツイスタ法、ほか
- 暗号などとも深い関係が

random

- Processingにはrandomが用意

random(上限)	0「以上」、上限「未満」の乱数を作る
random(下限, 上限)	下限「以上」、k上限「未満」の乱数を作る

- 例

random(0, 10)	0から9.999...の間の乱数を作る

型変換

- キャスト演算子
  - 整数に変換　　(int)変数名
  - 浮動小数点数に変換　　(float)変数名
- 型変換関数(Processingのみ)
  - 整数に変換　　int(変数名)
  - 浮動小数点すうに変換　　float(変数名)

14. 関数

Javaと同じ

14.1 関数

- 一連の「プログラムに名前をつけて、簡単に呼び出せるようにした」もの
- 関数を使うには
  - 事前に宣言する
  - それから名前を呼んで使う

関数の使い方（Javaと同じ）

- 関数の宣言

```
戻り値の型　関数名（仮引数の型 仮引数の名前，．．．）｛
　一連のプログラム
｝
```

- 関数の呼び出し

```
戻り値を入れる変数 = 関数名（実引数 実引数，．．．）；
```

14.2 関数の例：1

- 「こんにちは」とコンソールに表示

関数の宣言

```
void hello() {
  println("こんにちは");
}
```

関数の呼び出し

```
hello();
```

- 戻り値も引数もない場合の例
- 戻り値がないので、戻り値の型は void 無

14.3 関数の例：2

- 「こんにちは○○」とコンソールに表示

関数の宣言

```
void helloTo(String name) {
  println("こんにちは" + name);
}
```

関数の呼び出し

```
helloTo("太郎");
```

- 戻り値がなく、引数がある場合の例

14.4 関数の例: 3

- 2つの整数を足した結果を返す

  関数の宣言

```
int add(int a, int b) {
  return a + b;
}
```

  関数の呼び出し

```
int result = add(1, 2);
```

- 戻り値と引数がある場合の例
- 戻り値は整数なので、戻り値の型は int

15. オブジェクト

パーツとして使えるプログラム

Javaと同じ

「ソフトウェア危機」

- 1960年代後半
  - どんどんコンピュータが高度になって、
    ソフトウェアが複雑になってヤバい、死にそう(?)

オブジェクト指向

- プログラムの再利用
  - プログラムをパーツ化したものがオブジェクト
    - たとえば現実世界のモノや概念を参考にパーツを設計(？)
  - すごい人の書いたプログラムをパーツとして提供し、
    それを組み合わせれば簡単にプログラムできる(？)
- メッセージ送受信
  - オブジェクト間でメッセージをやりとりをしているかのように
    プログラムを書く(？)

15.1.1 オブジェクト

- オブジェクトを使うには
  - クラスを宣言する　※class 種族、グループ
    - クラスは「オブジェクトの設計図」
  - インスタンスを表す変数を用意する
    - オブジェクトのことを「インスタンス」とも言う
  - クラスからインスタンスを作り、変数に代入する
    ※instance 設計図から作った実体
  - インスタンスを操作する

15.1.2 クラスの宣言
中身が空の場合

- 書き方

```
class クラス名 {
}
```

  - クラス名は大文字で
    はじめる

- 例: ボール

```
class Ball {
}
```

15.1.3 クラスの宣言 フィールドのみの場合

- 書き方

```
class クラス名 {
  // フィールドの宣言
  型 名前;
  …
}
```

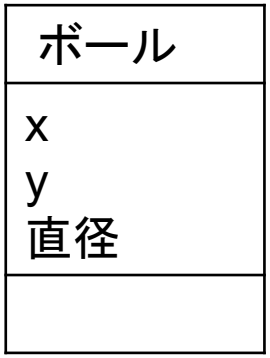
- 例: ボール

```
class Ball {
  float x; // 中心のx座標
  float y; // 中心のy座標
  float d; // 直径
}
```

- フィールドの宣言は変数の宣言と同じ

※field 欄

- クラス図



15.1.4 クラスの宣言 フィールドとメソッドがある場合

- 書き方

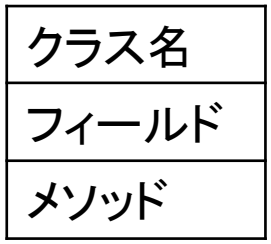
```
class クラス名 {
   // フィールドの宣言
   …

   // メソッドの宣言
   戻り値の型 メソッド名(引数の型 引数の名前,...) {
      メソッドの内容
   }
   …
}
```

- メソッドの書き方は、関数と同じ

※method 方法。オブジェクトに対する命令

クラス図の書き方

- 長方形の中に、上から「クラス名」「フィールド」「メソッド」を書く



クラスの宣言

- ボールの例

```
class Ball {
  // フィールドの宣言
  float x; // 中心のx座標
  float y; // 中心のy座標
  float d; // 直径

  // メソッドの宣言
  // 表示
  void display() {
    ellipse(x, y, d, d);
  }
}
```

ボール
x y 直径
表示()

15.1.5 オブジェクトを使う インスタンスを表す変数の用意

- インスタンスを表す変数を用意する

```
クラス名 変数名;
```

```
Ball b;
```

- プログラム全体で使うインスタンスは、一番外側で宣言

```
クラス名 変数名;

void setup() {
  …
}

void draw() {
  …
}
```

インスタンスを作り変数に代入

- newで作り、=で変数に代入

```
変数名 = new クラス名();
```

```
b = new Ball();
```

- 初期化のときにインスタンスを作るには、setupの中で

```
クラス名 変数名;

void setup() {
  変数名 = new クラス名();
  …
}

void draw() {
  …
}
```

インスタンスを操作

- 「インスタンス.フィールド」でフィールドにアクセス

```
// ballのxフィールドに10を代入
b.x = 10;

// ballのxフィールドの値をコンソールに表示
println(b.x);
```

- 「インスタンス.メソッド()」でメソッドを実行

```
// ボールのdisplay()メソッドを実行
b.display();
```

15.2 クラスの宣言 コンストラクタがある場合

- 書き方

```
class クラス名 {
   // フィールドの宣言
   …

   // コンストラクタの宣言
   クラス名(引数の型 引数の名前, ...) {
      コンストラクタの内容
   }

   // メソッドの宣言
   …
}
```

- コンストラクタの書き方は、戻り値のない関数

※constructor 生成するもの。初期化

コンストラクタのある場合のインスタンスの作り方

- 引数のないコンストラクタ

```
変数名 = new クラス名();
```

```
b = new Ball();
```

- 引数のあるコンストラクタ

```
変数名 = new クラス名(引数, …);
```

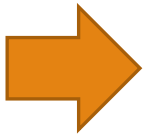
```
b = new Ball(100, 200, 150);
```

15.2.4 thisキーワード

- クラスの中で、自分自身を表すのにthisを使用する
- コンストラクタで、フィールドと引数を明確に区別するのに便利

```
// 人間を表すクラス
class Person {
  // フィールドの宣言
  int age; // 年齢

  // コンストラクタの宣言
  Person(int age) {
    age = age;
  }

  [中略]
}
```

フィールドageに
引数ageを代入したいが
見てもよくわからない



```
// 人間を表すクラス
class Person {
  // フィールドの宣言
  int age; // 年齢

  // コンストラクタの宣言
  Person(int age) {
    this.age = age;
  }

  [中略]
}
```

フィールドの方を
this.ageと書けばいい

クラスの宣言

- ボールの例

```
class Ball {
  // フィールドの宣言
  [中略]

  // コンストラクタの宣言
  Ball(float x, float y, float d) {
    this.x = x;
    this.y = y;
    this.d = d;
  }

  // メソッドの宣言
  [中略]
}
```

ボール
x y 直径
表示() ボール(x, y, 直径)

15.3 オーバーロード

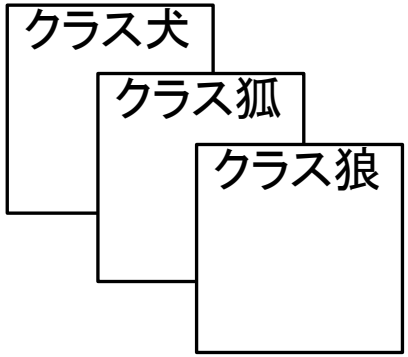
※overload: 多重定義

- 引数が異なれば、同じ名前のコンストラクタやメソッドを書ける
- 例: background関数
  - 引数が1つ background(明るさ);
    - 明るさを指定して背景を塗る
  - 引数が3つ background(赤, 緑, 青);
    - RGBを指定して背景を塗る
- 例: +演算子: 数値同士なら足し算、文字列とは結合

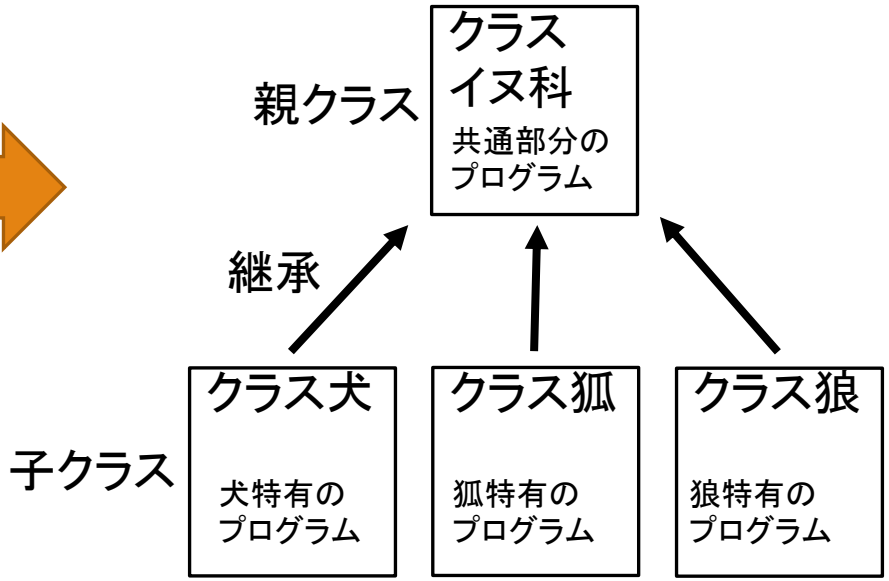
15.4 継承

- 差分プログラミング

だいたい同じ内容のクラス

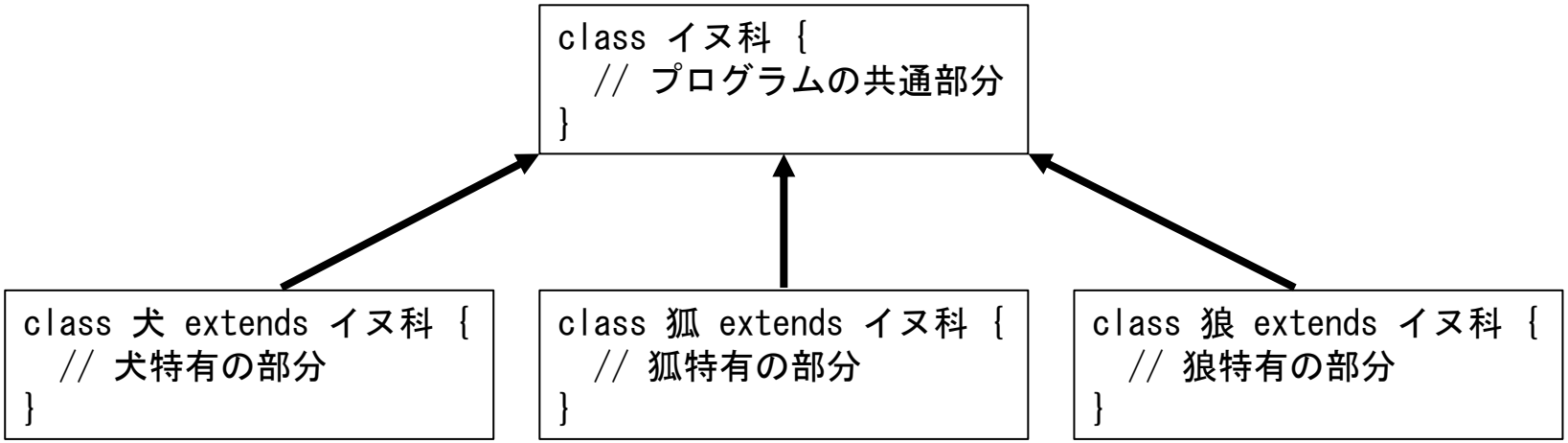


共通部分を取り出せる機能



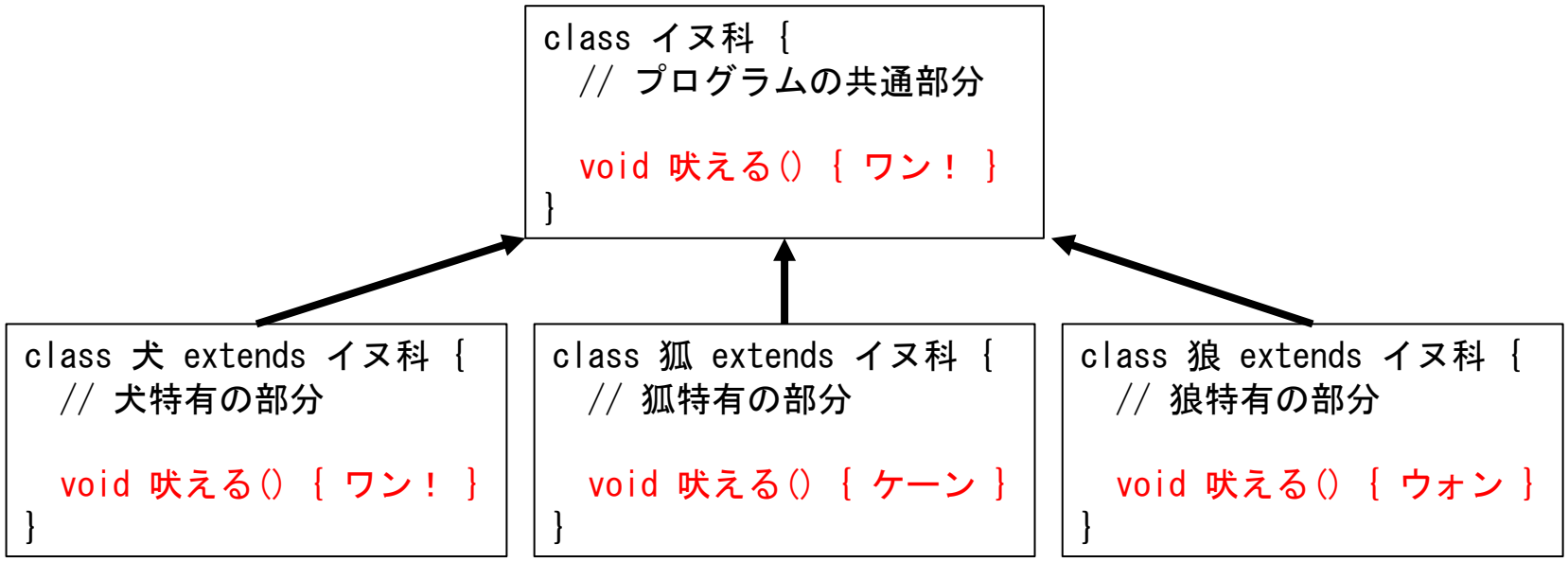
継承のプログラムの書き方

- extendsで継承できる ※extend: 拡張する



15.5 オーバーライド※override: 上書き

- 親クラスのメソッドやコンストラクタを子クラスで上書きできる



参考: Androidアプリのプログラミング

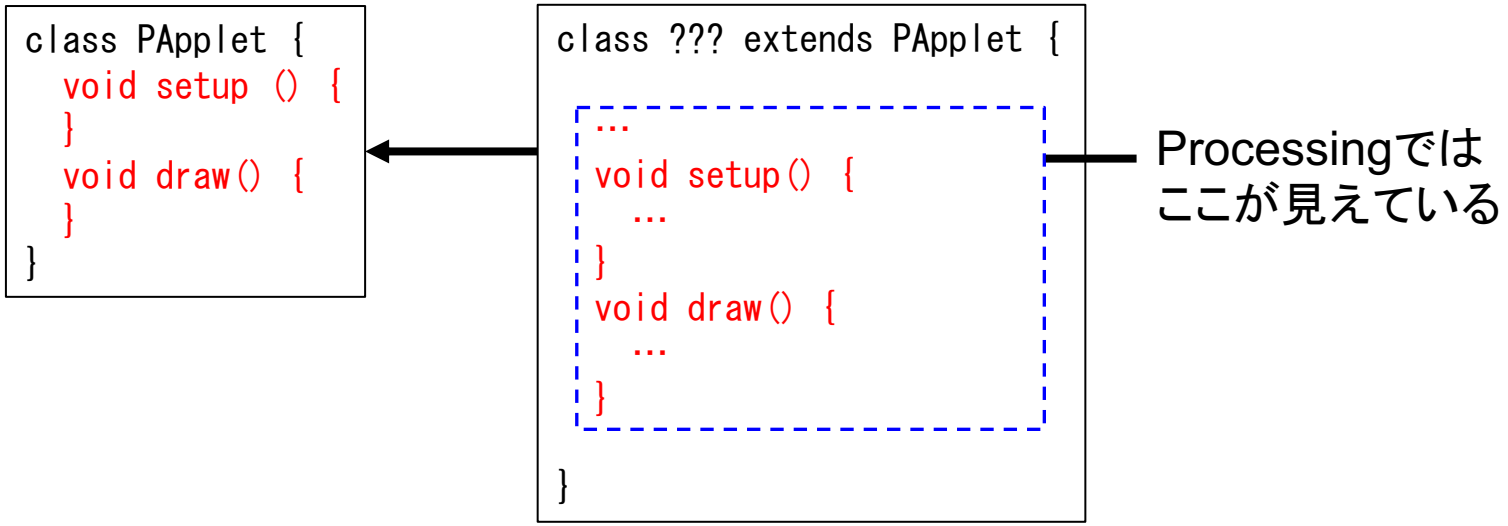
- 画面を表すクラスがフレームワーク(枠組み)として用意
- それを継承して、メソッドをオーバーライドして自分のアプリを作る

```
class 画面 {
  void 画面を表示したら実行 () {
  }
}
```

```
class 俺アプリ extends 画面 {
  void 画面を表示したら実行 () {
    タイトルバーを表示
    メッセージを表示
    ボタンを表示
  }
}
```

参考：Processingのプログラミング

- Processingのプログラムは PAppletクラスを継承したクラスの内側の部分
- setup()やdraw()メソッドをオーバーライドして自分のアプリを作っている



superキーワード

- superで、子クラスから親クラスを使うことができる
- 親クラスのコンストラクタを使う
  - super(引数);

※必ずコンストラクタの最初で使う必要がある

- 親クラスのメソッドを使う
  - super.メソッド(引数);

多態性 ※polymorphism: 同じ何かが多様な性質を持つ

- 親クラスの変数に、
  子クラスのインスタンスが代入できる
  - 例: イヌ科 イヌ科の動物 = new 狐();
- そのときに、メソッドを起動すると、
  子クラスのメソッドが起動される
  - 例: イヌ科の動物.吠える(); →「ケーン」

多態性の実例

- Androidアプリのプログラミング
  - ボタン、入力欄などは、共通の親クラスを持っている
  - その親クラスは画面に配置できるようになっている
  - ボタンや入力欄は配置するときに親クラスに代入される
  - その結果、どのパーツも全く同じように画面に配置できる
- Windowsの右クリック
  - アイコンごとに、対応したクラスがある
  - それらのクラスは共通の親クラスを持っている
  - 右クリックすると、子クラスの右クリック・メソッドが起動され、アイコンごとに異なったメニューが表示される

16. 配列

たくさんのデータ

16.1 配列のプログラミング

- 配列を利用すると、たくさんの同じ型のデータを扱える
- 配列を表す変数を用意
  - 型[] 変数名;
- 配列を生成して変数に代入
  - 変数名 = new 型[要素の個数];
    - 変数名[0]から変数名[要素の個数 - 1]までが使用できるようになる
- 配列の要素を1つずつ代入
  - 変数名[添字] = 値; // これをくりかえす

配列の例

```
// 整数の配列を表す変数dataを宣言
int[] data;

// 5つの要素を持った整数の配列を生成して変数dataに代入
data = new int[5];

// 要素に1つずつ値を代入
data[0] = 13;
data[1] = 210;
data[2] = -15;
data[3] = 8;
data[4] = 10;
```

配列の初期化

- 配列を用意して代入するところまでを一発でできる
  - 型[] 変数名 = { 要素, 要素, 要素, ... };
- 例
  - int[] data = { 13, 210, -15, 8, 10 };

配列の要素の個数

- 配列はオブジェクト
  - newで配列を生成
  - int[] data = new int[100];
- 配列はフィールドやメソッドを持つ
- 配列の要素の個数はlengthフィールド
  - data.length // dataの要素の個数