# Javaプログラミング入門講座
## ～オブジェクト指向を理解する～

足立研セミナー／Keio AI Othello Competition


2013/2/21

小野雅裕 （理工学部助教）

# 目的

「オブジェクト指向」の主要な概念を理解する

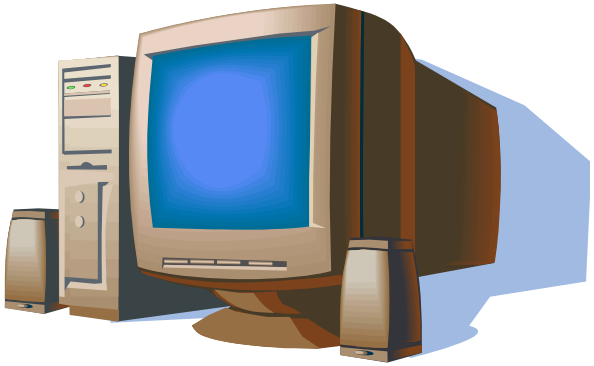
# おしながき

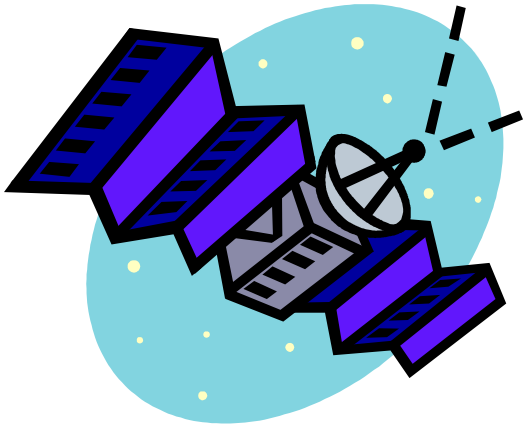
- What is Java? Why Java?
- オブジェクト指向入門
  - クラスとインスタンス
  - カプセル化
  - クラスの継承
  - ポリモーフィズム
- オセロ大会でのポリモーフィズムの応用
- プログラミング上達のためのコツ

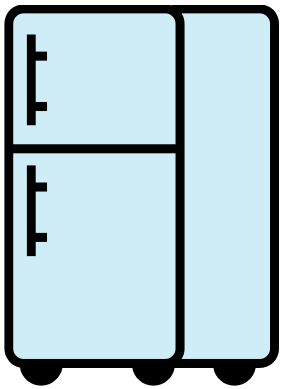
# おしながき

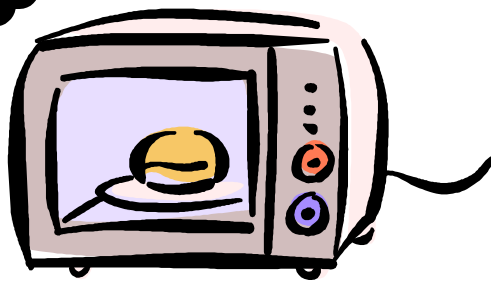
- **What is Java? Why Java?**
- オブジェクト指向入門
  - クラスとインスタンス
  - カプセル化
  - クラスの継承
  - ポリモーフィズム
- オセロ大会でのポリモーフィズムの応用
- プログラミング上達のためのコツ

# コンピューター・プログラムは
# あらゆる場所で動いている







# 背景1：組み込みシステムの普及

ＰＩＣマイコン
ＰＩＣ１２Ｆ６２
９－Ｉ／Ｐ

[PIC12F629-I/P]
[I-00252]
1個 ￥70
(税込)

- 現代の電気製品の殆どにマイコンが乗っている
  - 電子レンジでも冷蔵庫でも時計でも
- マイコン＝超小型コンピューター
  - ひとつのチップにCPU、メモリなどコンピューターの機能を全てつめこんだもの
  - コンピューターと同様にプログラムを搭載できる
  - 安い！！

# 背景2：スマホの普及

- 誰でもアプリを作れる
  - 開発キットが無償配布されている
- 誰でも作ったアプリを売れる


Mushroom Garden ...

BEEWORKSGAMES

Popular with Mushroom Garden Deluxe users

INSTALL

Ultima Reversi

ULTIMA ARCHITECT IN

Popular with Othello Pr

INSTALL

gContacts

Y.GPHONEBOOK

Popular in your area

INSTALL


楽天市場

RAKUTEN,INC.

Popular with YAMADA users

INSTALL

# エンジニア採用に新基準、グリーが“GREE Programming Challenge”を導入

グリーは、エンジニアの採用において、新ツール“GREE Programming Challenge”をグリーのコーポレートサイトにて提供を始めた。

カテゴリー GREE

2012-06-01 14:26:55



## ●新ツールによってエンジニアの技術アピールが可能に

グリーは、エンジニアの採用において、新ツール“GREE Programming Challenge”をグリーのコーポレートサイトにて提供を始めた。これは、従来のWebエントリーや書類選考と並行して、プログラミングの技術に重点を置いた新たな選考方法で、米国では数多くの情報技術、インターネットサービス関連企業に採用されているプロセス。日本企業としてはグリーが初めてのサービス導入となる。

～以下、リリースより～

■GREE Programming Challengeとは

応募者が従来の書類選考と一次面接に参加する代わりに、グリーのコーポレートサイトにアクセスし、出題されたプログラミング問題を一定の時間内で回答するシステムです。プログラミングの途中経過のすべてがシステムで記録され、問題解決のための思考プロセス、速度および正確さなどが総合的に評価されます。応募者は、選考の初期段階から自身の専門能力をアピールすることが可能となると同時に、グリーのオフィスがある地域以外からでも物理的な移動なしに、選考に参加することができるようになります。

速報 MarkeZineニュース

# サイバーエージェント、プログラミングスキルのみで内定決定する「コード採用」

MarkeZine編集部 [著] 2012/05/14 12:45



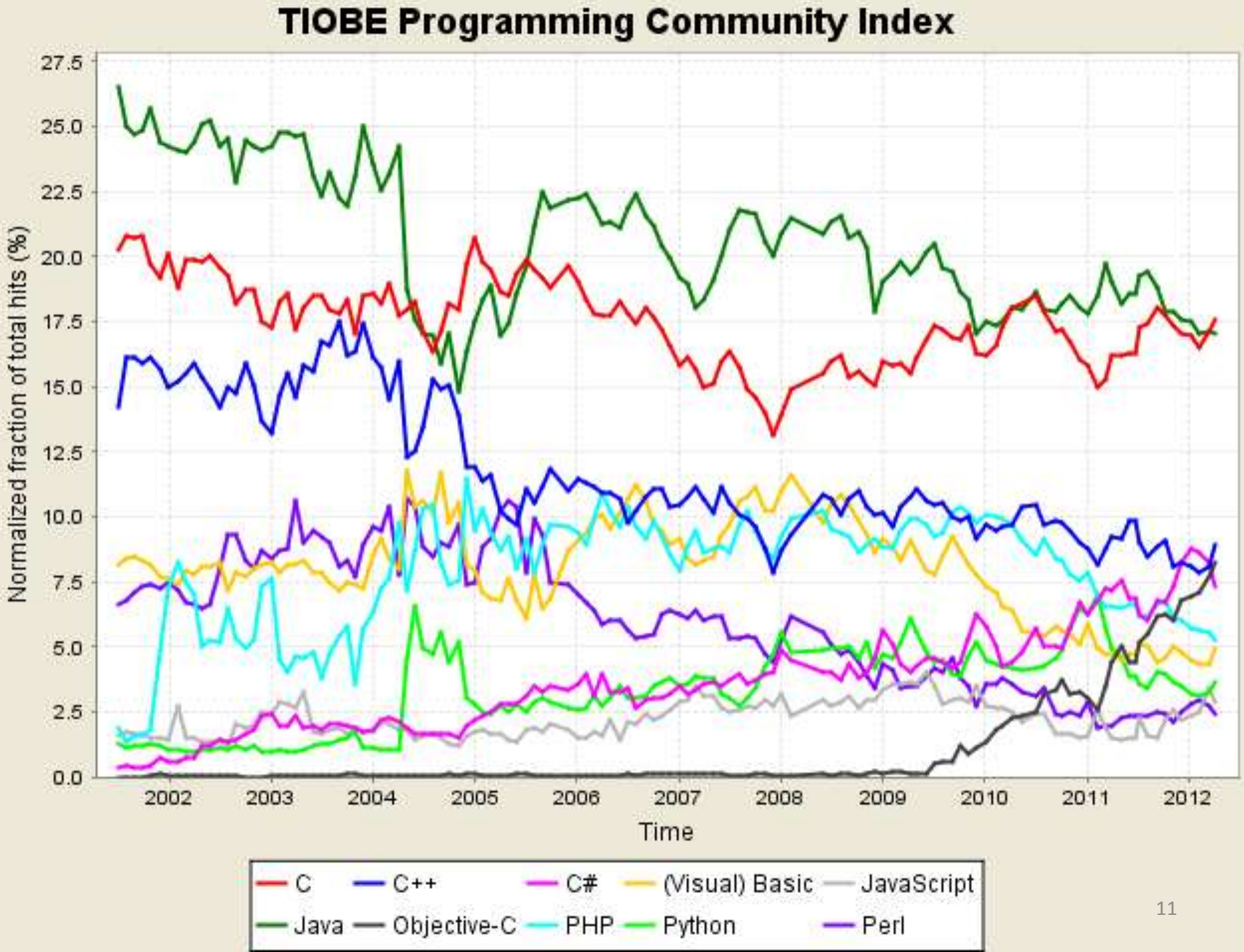
**サイバーエージェントは、エンジニアの新卒採用で、プログラミングスキルのみ・面接なしで内定を決定する「コード採用」を開始した。**

新卒採用のエンジニア職で実施される「コード採用」は、Javaでプログラムを組み、ソースコードを提出するだけで、面接なし、プログラミングスキルのみで合否を決定する。おもに2014年卒の学生を対象に7月から募集を開始する。

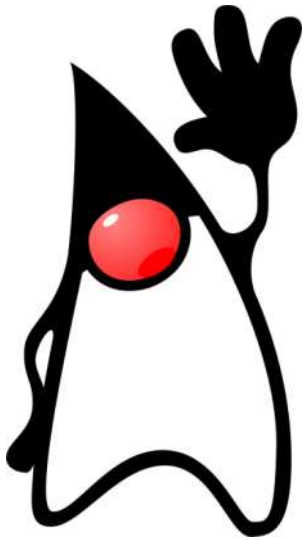
サイバーエージェントでは、「Ameba」のスマートフォンプラットフォームおよびスマートフォン向けサービスの立ち上げ・拡充を強化しており、新たに「コード採用」を実施することで、高いプログラミングスキルをもつ学生の採用を目指す。

# TIOBE Programming Community Index
# （プログラム言語のメジャー度ランキング）


| Position Feb 2013 | Position Feb 2012 | Delta in Position | Programming Language | Ratings Feb 2013 | Delta Feb 2012 | Status |
|---|---|---|---|---|---|---|
| 1 | 1 | = | Java | 18.387% | +1.34% | A |
| 2 | 2 | = | C | 17.080% | +0.56% | A |
| 3 | 5 | ↑↑ | Objective-C | 9.803% | +2.74% | A |
| 4 | 4 | = | C++ | 8.758% | +0.91% | A |
| 5 | 3 | ↓↓ | C# | 6.680% | -1.97% | A |
| 6 | 6 | = | PHP | 5.074% | -0.57% | A |
| 7 | 8 | ↑ | Python | 4.949% | +1.80% | A |
| 8 | 7 | ↓ | (Visual) Basic | 4.648% | +0.33% | A |
| 9 | 9 | = | Perl | 2.252% | -0.68% | A |
| 10 | 12 | ↑↑ | Ruby | 1.752% | +0.19% | A |
| 11 | 10 | ↓ | JavaScript | 1.423% | -1.04% | A |
| 12 | 16 | ↑↑↑↑ | Visual Basic .NET | 1.007% | +0.21% | A |
| 13 | 13 | = | Lisp | 0.943% | +0.04% | A |
| 14 | 15 | ↑ | Pascal | 0.932% | +0.12% | A |
| 15 | 11 | ↓↓↓↓ | Delphi/Object Pascal | 0.886% | -1.08% | A |
| 16 | 14 | ↓↓ | Transact-SQL | 0.773% | -0.07% | A-- |
| 17 | 75 | ↑↑↑↑↑↑↑↑↑↑ | Bash | 0.741% | +0.61% | A-- |
| 18 | 26 | ↑↑↑↑↑↑↑↑ | MATLAB | 0.648% | +0.15% | B |
| 19 | 24 | ↑↑↑↑↑ | Assembly | 0.640% | +0.12% | B |
| 20 | 19 | ↓ | Ada | 0.631% | 0.00% | B |


TIOBE Programming Community Index
Normalized fraction of total hits (%)
27.5
25.0
22.5
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
Time
C
C++
C#
(Visual) Basic
JavaScript
Java
Objective-C
PHP
Python
Perl


Java = ジャワ島のジャワ
アイコンはジャワコーヒー



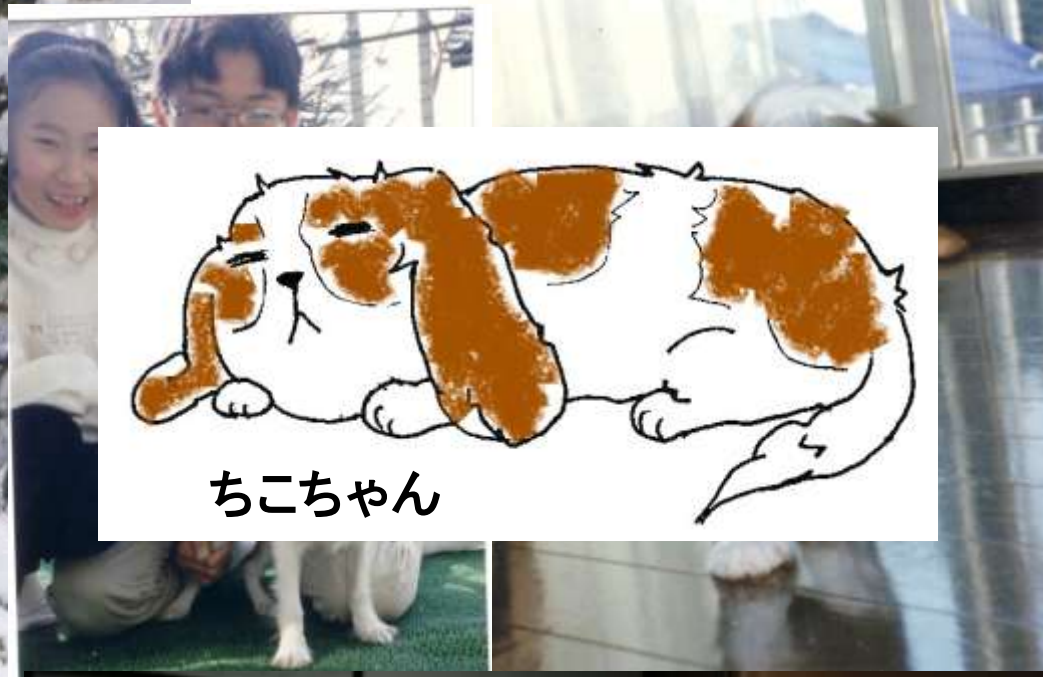
マスコットDuke
あんまかわいくない

# Why Java? その1

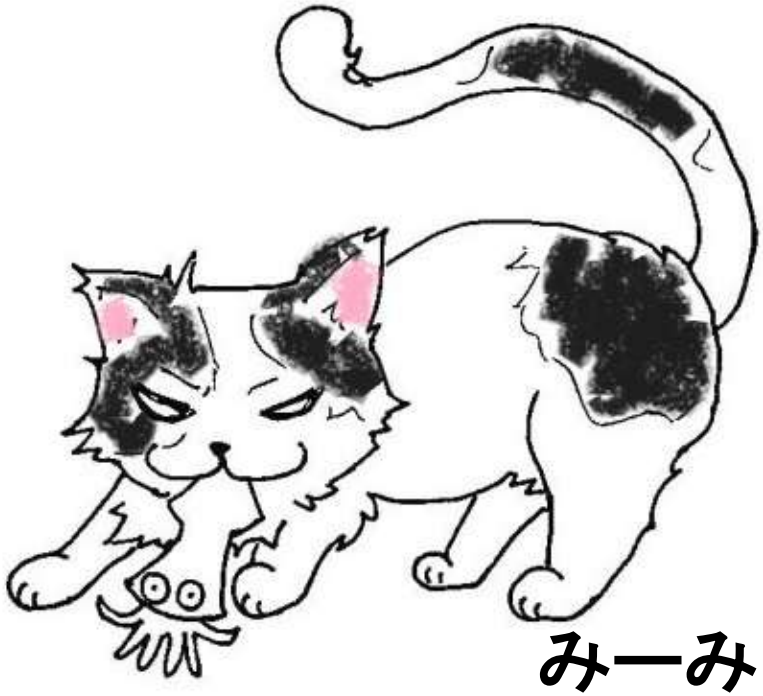
- プラットフォームに依存しない
  - 同じプログラムがWindowsでもMacでもLinuxでも動く
- 無料
  - しかもEclipseという優れた統合開発環境も無料
- 自動化されたメモリ管理 (ガーベージ・コレクション)
  - C言語は自分でメモリの割り当てと開放をしなくてはいけなかった
  - バグが飛躍的に少なくなる

# Why Java? その２

- 厳密なオブジェクト指向
- 論理的な構文
  - Cのように多重継承や演算子オーバーロードなし
  - Cに比べて柔軟性は劣るが、カオスにもなりにくい
- グラフィックスを容易に扱える
- ドキュメントを自動作成してくれる (Javadoc)
- 簡単にWebアプリケーション（アプレット）を作れる
- Androidアプリのプログラミング言語




ちこちゃん



みーみ

# Why Java? その２

- **厳密なオブジェクト指向**
- 論理的な構文
  - Cのように多重継承や演算子オーバーロードなし
  - Cに比べて柔軟性は劣るが、カオスにもなりにくい
- グラフィックスを容易に扱える
- ドキュメントを自動作成してくれる (Javadoc)
- 簡単にWebアプリケーション（アプレット）を作れる
- Androidアプリのプログラミング言語

# おしながき

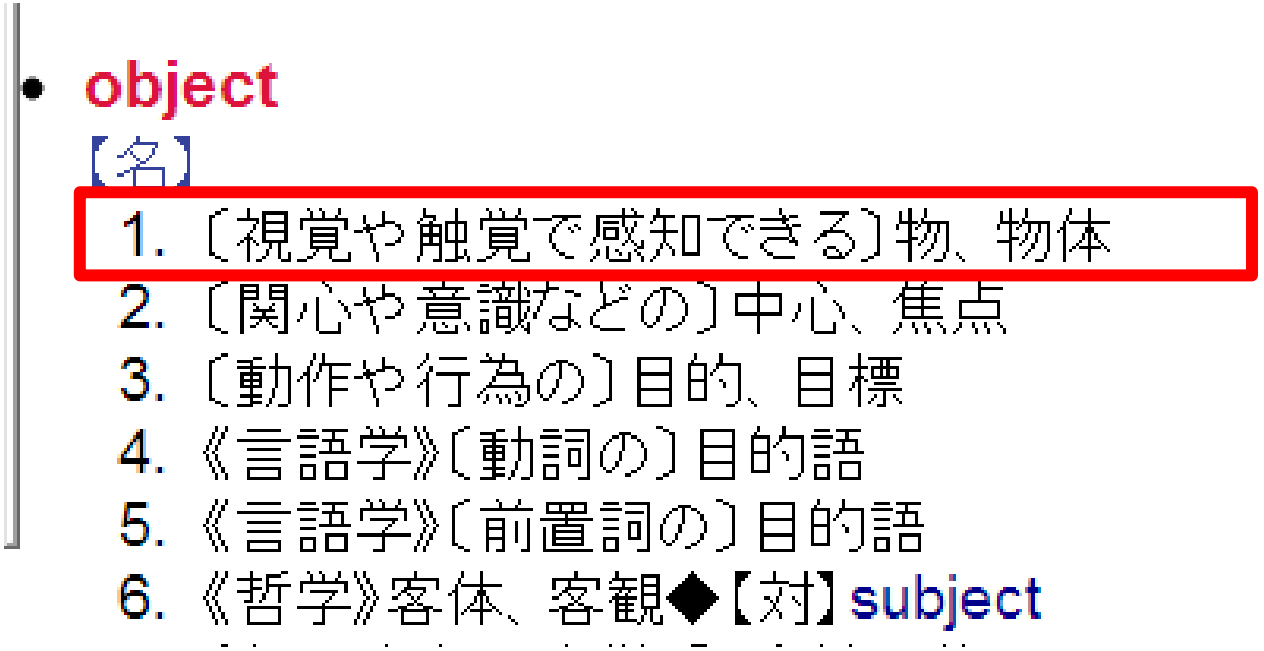
- What is Java? Why Java?
- **オブジェクト指向入門**
  - **クラスとインスタンス**
  - カプセル化
  - クラスの継承
  - ポリモーフィズム
- オセロ大会でのポリモーフィズムの応用
- プログラミング上達のためのコツ

# オブジェクト指向言語

| Position Feb 2013 | Position Feb 2012 | Delta in Position | Programming Language | Ratings Feb 2013 | Delta Feb 2012 | Status |
|---|---|---|---|---|---|---|
| 1 | 1 | = | Java | 18.387% | +1.34% | A |
| 2 | 2 | = | C | 17.080% | +0.56% | A |
| 3 | 5 | ↑↑ | Objective-C | 9.803% | +2.74% | A |
| 4 | 4 | = | C++ | 8.758% | +0.91% | A |
| 5 | 3 | ↓↓ | C# | 6.680% | -1.97% | A |
| 6 | 6 | = | PHP | 5.074% | -0.57% | A |
| 7 | 8 | ↑ | Python | 4.949% | +1.80% | A |
| 8 | 7 | ↓ | (Visual) Basic | 4.648% | +0.33% | A |
| 9 | 9 | = | Perl | 2.252% | -0.68% | A |
| 10 | 12 | ↑↑ | Ruby | 1.752% | +0.19% | A |
| 11 | 10 | ↓ | JavaScript | 1.423% | -1.04% | A |
| 12 | 16 | ↑↑↑↑ | Visual Basic .NET | 1.007% | +0.21% | A |
| 13 | 13 | = | Lisp | 0.943% | +0.04% | A |
| 14 | 15 | ↑ | Pascal | 0.932% | +0.12% | A |
| 15 | 11 | ↓↓↓↓ | Delphi/Object Pascal | 0.886% | -1.08% | A |
| 16 | 14 | ↓↓ | Transact-SQL | 0.773% | -0.07% | A-- |
| 17 | 75 | ↑↑↑↑↑↑↑↑↑↑ | Bash | 0.741% | +0.61% | A-- |
| 18 | 26 | ↑↑↑↑↑↑↑↑ | MATLAB | 0.648% | +0.15% | B |
| 19 | 24 | ↑↑↑↑↑ | Assembly | 0.640% | +0.12% | B |
| 20 | 19 | ↓ | Ada | 0.631% | 0.00% | B |

# オブジェクト指向？

# オブジェクト　＝　もの



- **object**

【名】

1. 〔視覚や触覚で感知できる〕物、物体
2. 〔関心や意識などの〕中心、焦点
3. 〔動作や行為の〕目的、目標
4. 《言語学》〔動詞の〕目的語
5. 《言語学》〔前置詞の〕目的語
6. 《哲学》客体、客観◆【対】subject

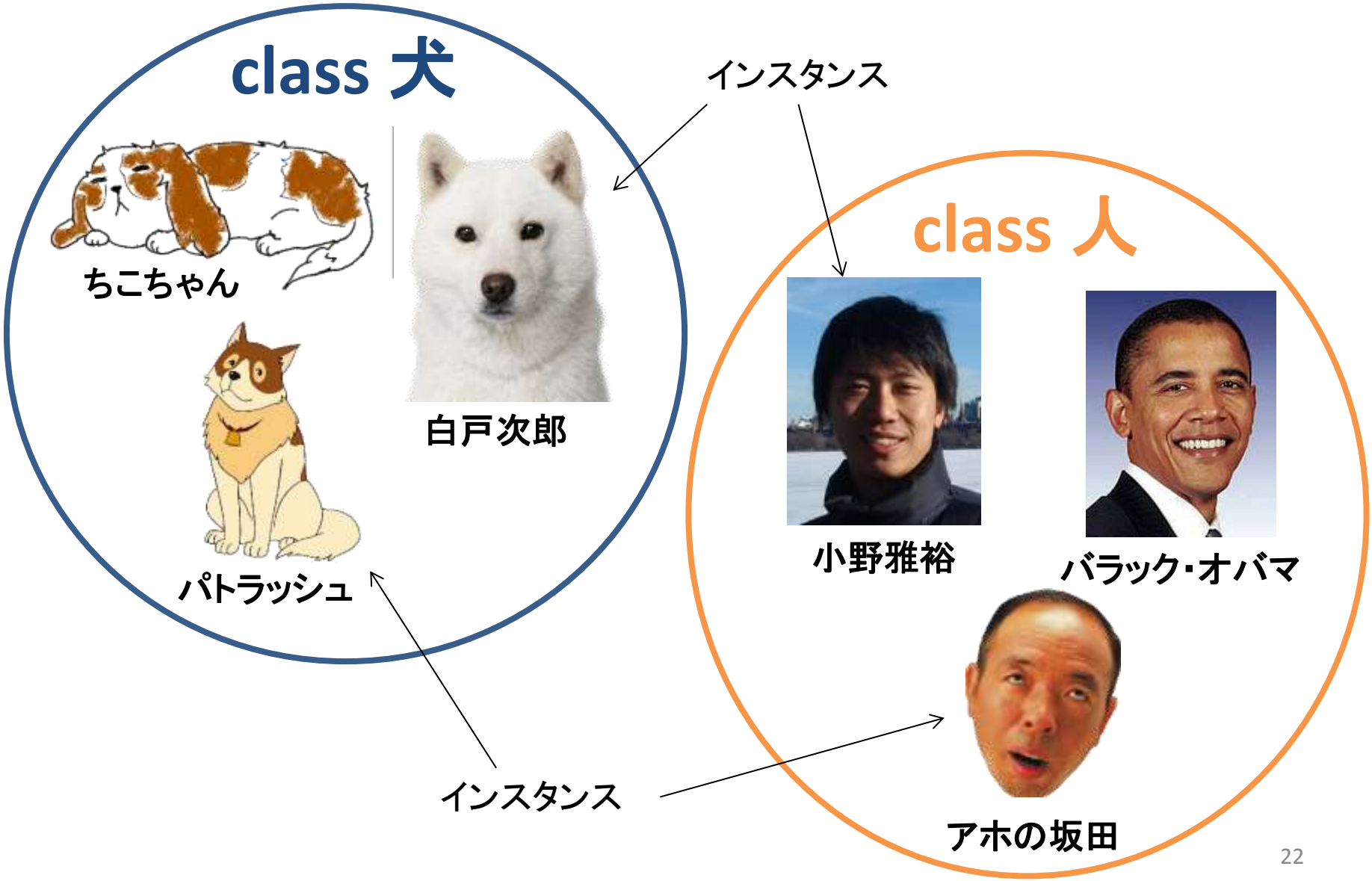
# オブジェクト指向？

# オブジェクト　＝　もの

「もの」の例：

- 僕の車、足立先生の車、新庄剛志の車・・・
- 僕の家、足立先生の家、新庄剛志の家・・・
- 三角形、四角形、五角形・・・・
- 小野雅裕、丸田一郎、高村英雅・・・

# クラスとインスタンス


class 犬
インスタンス
ちこちゃん
白戸次郎
パトラッシュ
class 人
小野雅裕
バラック・オバマ
インスタンス
アホの坂田

# クラス：犬

- 定数
  - 名前：
  - 色：
- 変数
  - 現在位置：
- 機能
  - 歩く
  - 吠える
  - …

# インスタンス:チコちゃん


- 定数
  - 名前:**チコちゃん**
  - 色:**茶色**
- 変数
  - 現在位置:**小野家のコタツ**
- 機能
  - 歩く
  - 吠える
  - …

# インスタンス:お父さん


- 定数
  - 名前:**白戸次郎**
  - 色:**白**
- 変数
  - 現在位置:**白戸家の食卓**
- 機能
  - 歩く
  - 吠える
  - …

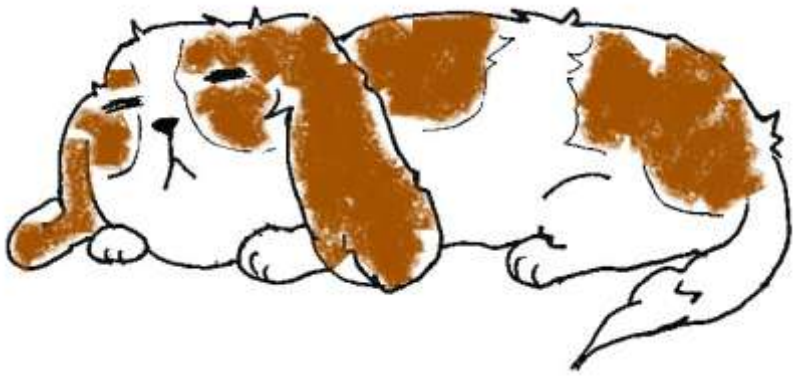
# クラス：犬　　Dog.java

- 定数
  - 名前：　　`String name;`
  - 色：
- 変数
  - 現在位置：
```
int x;
int y;
```
- 機能
  - 歩く　　`public void moveRight()`
  - 吠える　`public void makeSound()`
  - …

Dog.java

```
/
public class Dog extends Animal {

    private String[] messages = {"うるさいなぁ、寝てるん
    private Random rnd = new Random(); //乱数発生

    public Dog(Color animalColor, String name){
        super(name);
        stepSize = 2; //犬の歩幅を指定
```

# インスタンスの作り方



- 定数
  - 名前：チコちゃん
  - 色：Brown
- 変数
  - 現在位置：
- 機能
  - 歩く
  - 吠える
  - …

クラス
Dog型

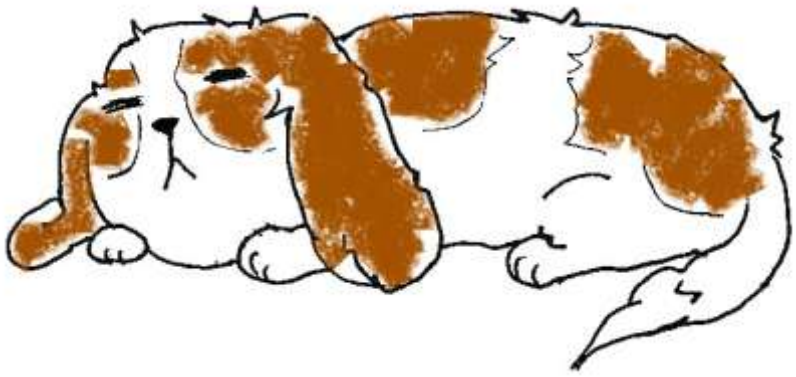
チコちゃんを表すインスタンス

```
Dog dog1 = new Dog(BROWN, "チコちゃん");
```

(Main.java 37行目)

**コンストラクタ**：新しいインスタンスを作る関数
（関数名は必ずクラス名と同じ名前）

# インスタンスの作り方



- 定数
  - 名前：チコちゃん
  - 色：Brown
- 変数
  - 現在位置：
- 機能
  - 歩く
  - 吠える
  - …

クラス
Dog型

チコちゃんを表すインスタンス

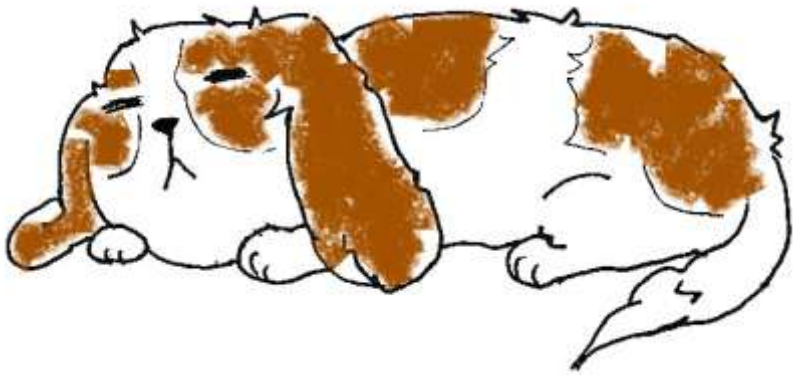
```
Dog dog1 = new Dog(BROWN, "チコちゃん");
```

(Main.java 37行目)

**練習問題**

- **チコちゃんの色と名前を変えてください**

# チコちゃんの動かし方



- 定数
  - 名前：チコちゃん
  - 色：Brown
- 変数
  - 現在位置：
- 機能（メソッド）
  - 歩く　　`public void moveRight()`
  - 吠える　`public void makeSound()`
  - …

チコちゃんを表すインスタンス

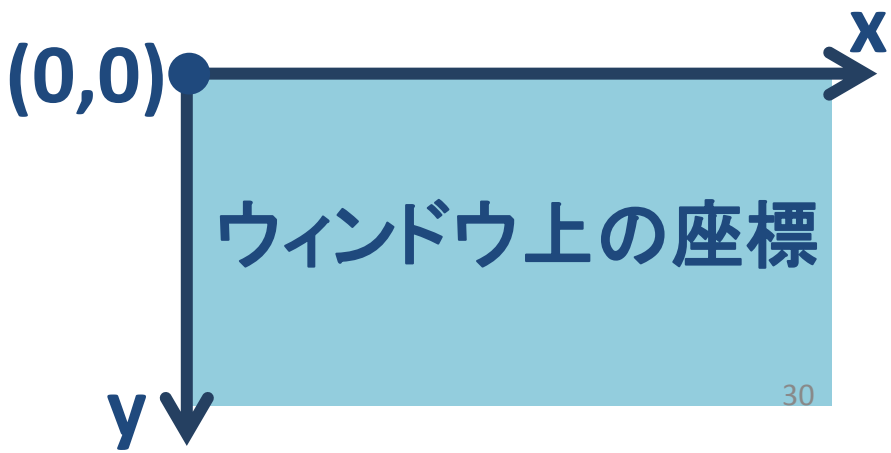
```
Dog dog1 = new Dog(BROWN, "チコちゃん");
```

(Main.java 37行目)

```
dog1.moveRight(); //右に動かす
dog1.makeSound(); //吠えさせる
```

# class Dog

- メソッド
  - makeSound() 鳴く
  - setPosition(int x, int y) (x, y)に移動する
  - moveUp() 上に移動する
  - moveDown() 下に移動する
  - moveUp() 左に移動する
  - moveRight() 右に移動する

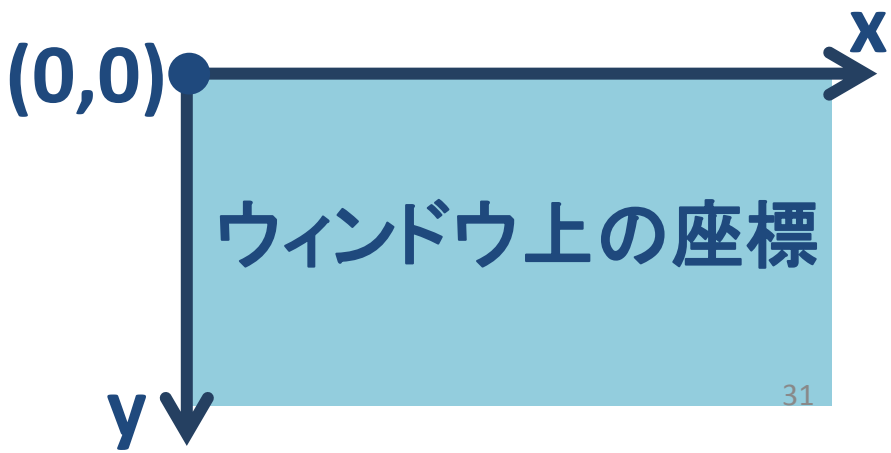


# class Cat

- メソッド
  - makeSound() 鳴く
  - setPosition(int x, int y) (x, y)に移動する
  - moveUp() 上に移動する
  - moveDown() 下に移動する
  - moveUp() 左に移動する
  - moveRight() 右に移動する

**練習問題**

- チコちゃんを1秒に一回吠えさせてください
- ミーミをジグザグに動くようにしてください



# クラスの定義の仕方

- 基本的にひとつのクラスにひとつのファイル
  - Dogクラスを定義するファイルはDog.java

Dog.java

```
public class Dog{

        （クラスの中身をここに書く）

}
```

# クラスの定義の仕方

- 基本的にひとつのクラスにひとつのファイル
  - Dogクラスを定義するファイルはDog.java

Dog.java

```
public class Dog{
        定数、変数の定義
        コンストラクタの定義
        メソッド（関数）の定義
}
```

注：コンストラクタもメソッドですが、特殊なメソッドです。

```java
import java.awt.Color; //色を表すクラス Colorを使うためのおまじない

public class Dog {

    /***************定数たち**************/
    public final String name; //犬の名前
    public final Color color; //犬の色

    /***************変数たち**************/
    private double x, y;

    /***************メソッド(関数)たち**************/
    //コンストラクタ
    public Dog(String _name, Color _color, double initX, double initY){//引数は、名前
        name = _name;
        color = _color;
        x = initX;
        y = initY;
    }

    //右に動かす
    public void moveRight(){
        x = x + 10;
    }

    //左に動かす
    public void moveUp(){
        y = y - 10;
    }
}
```

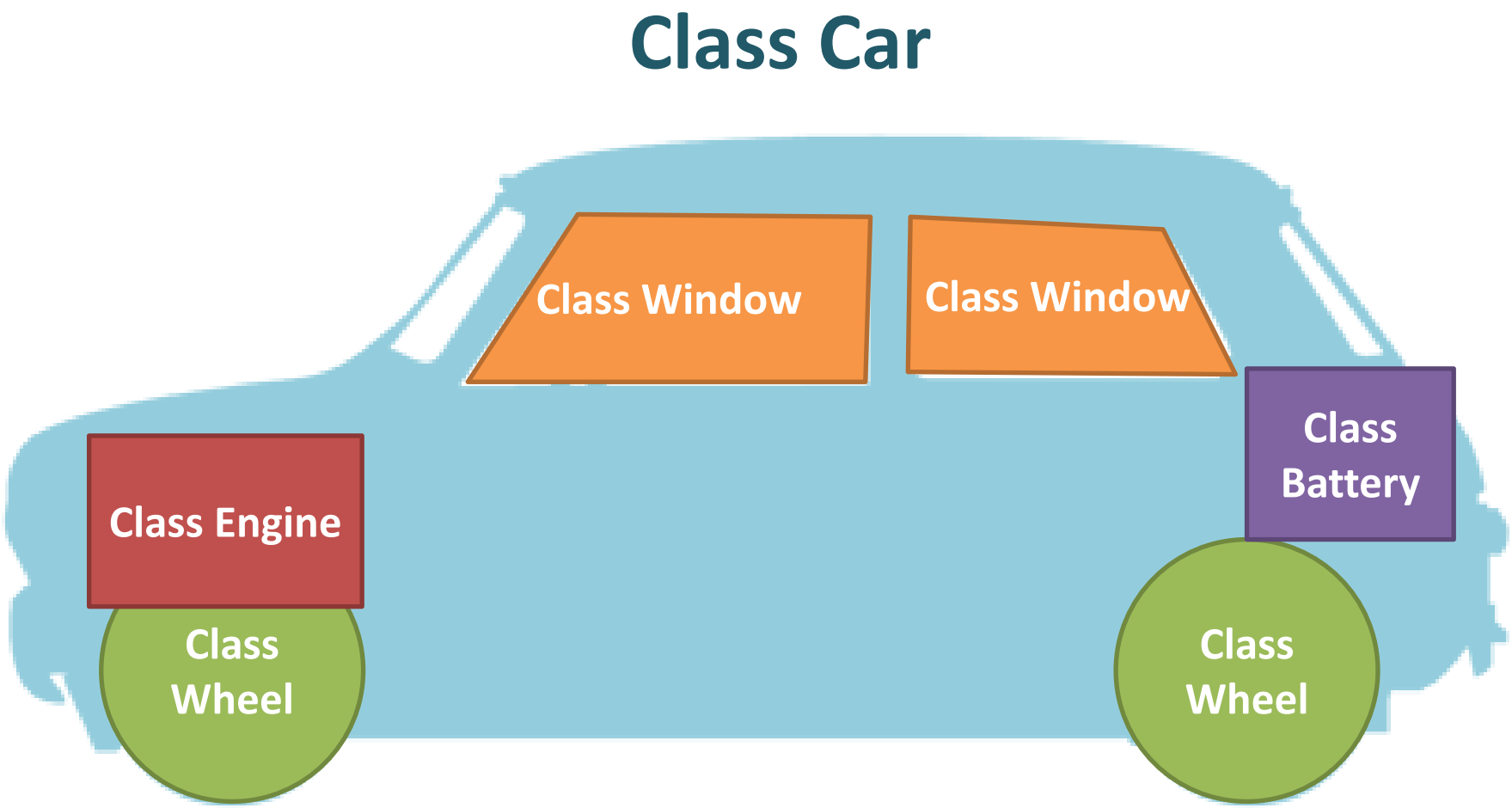
注：みなさんにお配りしたDog.javaは実際にはこのようにはかかれていません。殆どの変数やメソッドの定義はAnimal.javaに書かれています。その理由は後述します。

# クラスは型として使える

- つまり、クラスを定義するとは、新しい型を定義することとも捉えられる。

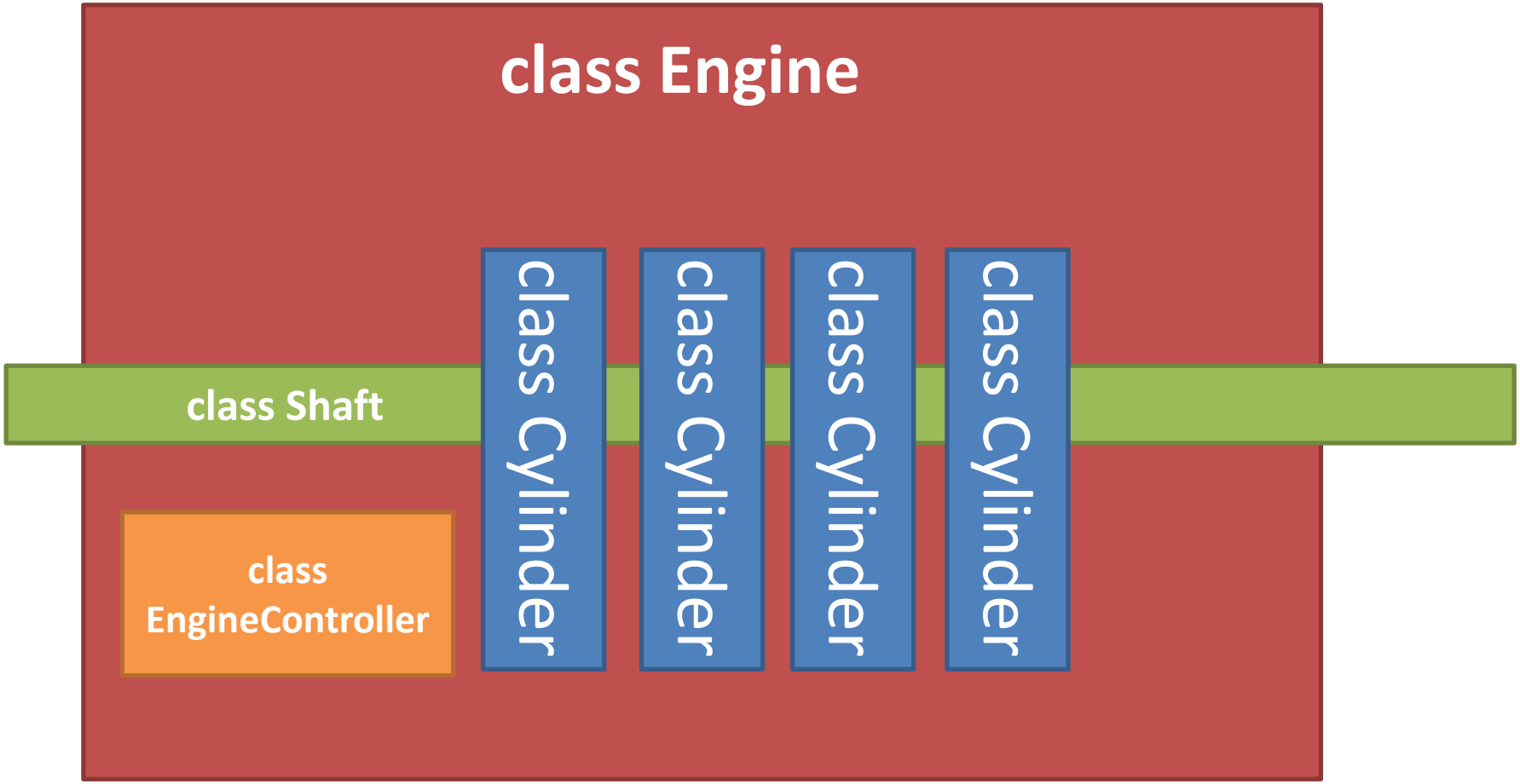
```
//int型変数の定義
int n = 50;

//Dog型インスタンスの定義
Dog pochi = new Dog(Color.BLACK, "Pochi");
```

# クラスは大規模なプログラムを分かりやすく書くのに適している




メソッド： drive(), stop(), turn(), ….

# クラスは大規模なプログラムを分かりやすく書くのに適している



# おしながき

- What is Java? Why Java?
- オブジェクト指向入門
  - クラスとインスタンス
  - **カプセル化**
  - クラスの継承
  - ポリモーフィズム
- オセロ大会でのポリモーフィズムの応用
- プログラミング上達のためのコツ

```java
import java.awt.Color; //色を表すクラス Colorを使うためのおまじない

public class Dog {

    /***************定数たち***************/
    public final String name; //犬の名前
    public final Color color; //犬の色

    /***************変数たち***************/
    private double x, y;

    /***************メソッド(関数)たち***************/
    //コンストラクタ
    public Dog(String _name, Color _color, double initX, double initY){//引数は、名前
        name = _name;
        color = _color;
        x = initX;
        y = initY;
    }

    //右に動かす
    public void moveRight(){
        x = x + 10;
    }

    //左に動かす
    public void moveUp(){
        y = y - 10;
    }
}
```

ナニコレ？

```java
import java.awt.Color; //色を表すクラス Colorを使うためのおまじない

public class Dog {

    /***************定数たち***************/
    public final String name; //犬の名前
    public final Color color; //犬の色

    /***************変数たち***************/
    private double x, y;

    /***************メソッド(関数)たち***************/
    //コンストラクタ
    public Dog(String _name, Color _color, double initX, double initY){//引数は、名前
        name = _name;
        color = _color;
        x = initX;
        y = initY;
    }

    //右に動かす
    public void moveRight(){
        x = x + 10;
    }

    //左に動かす
    public void moveUp(){
        y = y - 10;
    }
}
```

## アクセス修飾子

- publicな変数は、このクラスの外から読み書きできる
- privateな変数は、このクラスの中からしか読み書きできない
- publicなメソッドは、このクラスの外から呼び出せる
- privateなメソッドは、このクラスの中からしか呼び出せない

# アクセス修飾子の使い方

Dog.java

```
public class Dog {

    private double x, y;

    /**************メソッド(関数)たち*************/
    ...
    //コンストラクタ
    public Dog(String _name, Color _color, double initX, double initY){//引数は、名前
        name = _name;
        color = _color;
        x = initX;
        y = initY;
    }
```

現在位置x, yはprivateなので・・・・

このようにDogクラスの中からは
読み書きできるが

Main.java

```
public class Main {
    public static void main(String[] args) {
        Dog pochi = new Dog("Pochi", Color.BLACK, 50, 60);
        pochi.x = 20;
        pochi.y = 30;
    }
}
```

このように他のクラスから書き
込もうとするとエラーが出る

# アクセス修飾子の使い方

Dog.java

```
public class Dog {

    public double x, y;

    /***************メソッド(関数)たち**************/
    //コンストラクタ
    public Dog(String _name, Color _color, double initX, double initY){//引数は、名前
        name = _name;
        color = _color;
        x = initX;
        y = initY;
    }
```

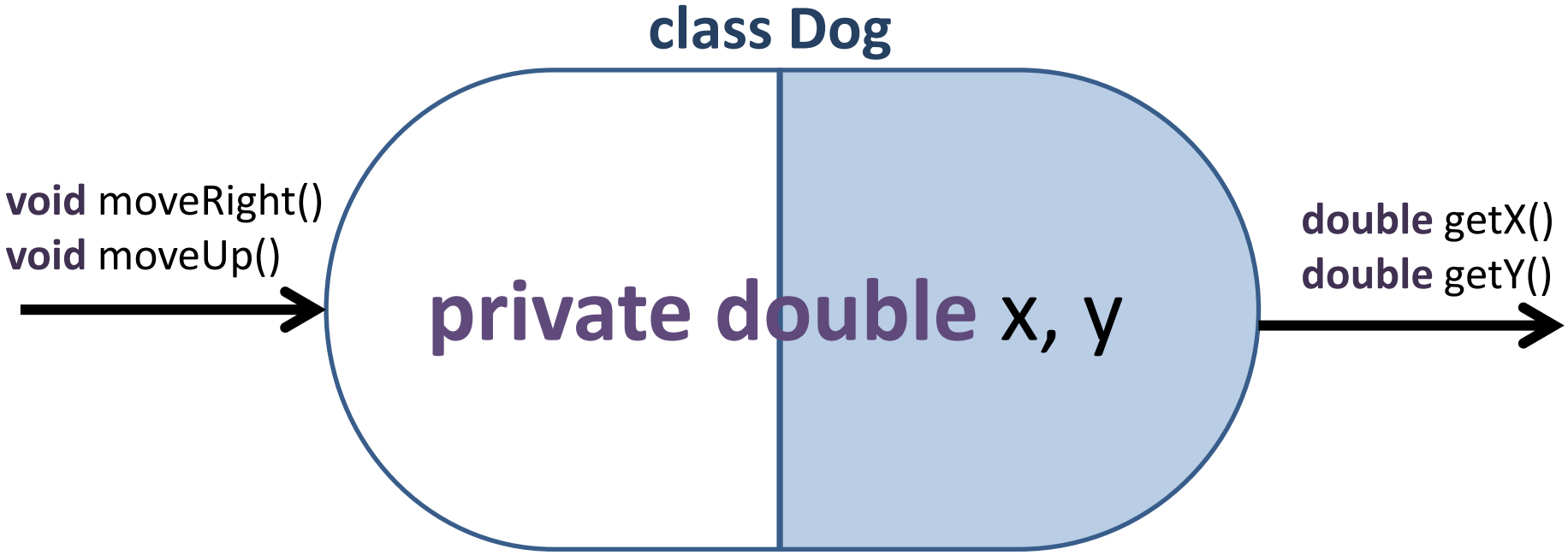
ここをpublicにすれば…

Main.iava

```
public class Main {
    public static void main(String[] args) {
        Dog pochi = new Dog("Pochi", Color.BLACK, 50, 60);
        pochi.x = 20;
        pochi.y = 30;
    }
}
```

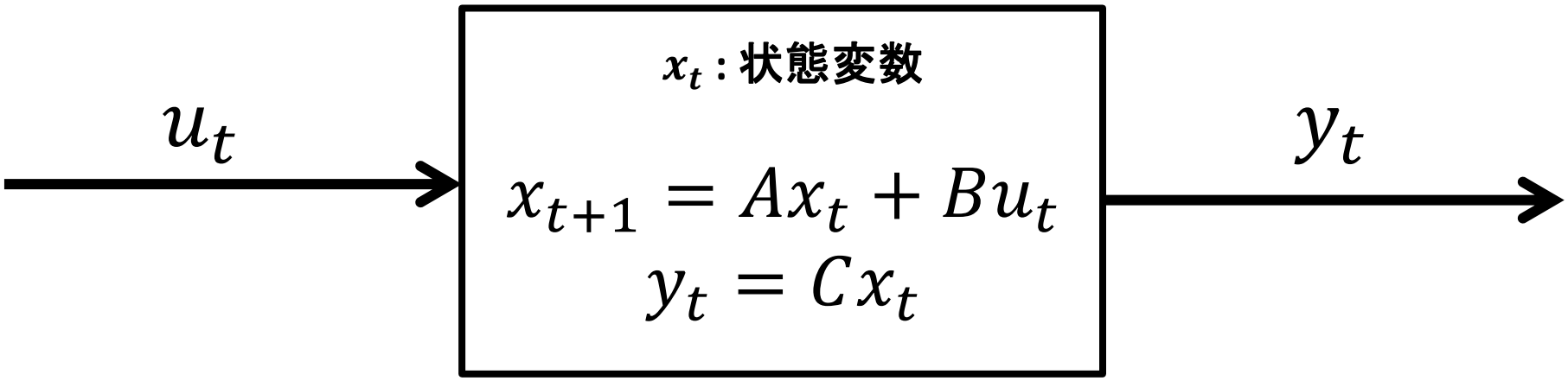
エラーが消える

# カプセル化

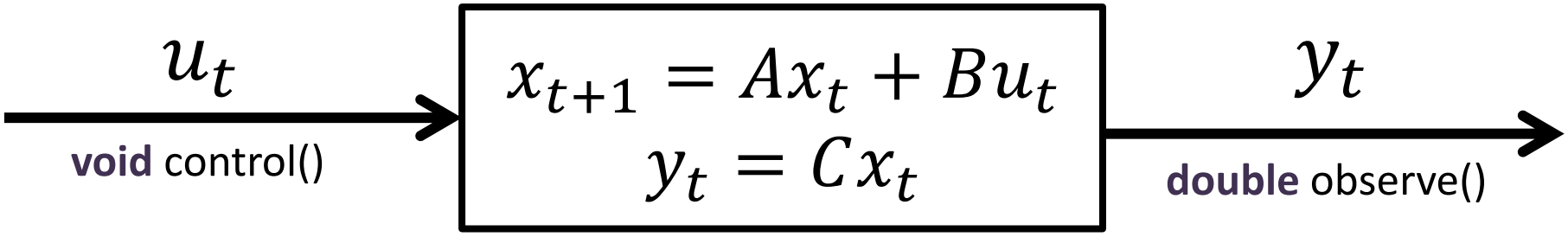



- privateの変数は直接操作することも見ることも出来ない
- 操作はpublicメソッドを通して行う
- 変数を読むのもpublicメソッドを通して行う
  - getなんとか、という名前のメソッドがあればそれ

# 制御の状態空間モデルと似た概念




- 状態変数は直接操作することも見ることも出来ない
- 操作は制御入力$u_t$ を通して行う
- 状態推定は観測$y_t$を通して行う




```
public class LTIsystem {

    private final double A, B, C;
    private double x;

    public LTIsystem(double _A, double _B, double _C, double x0){
        A = _A; B = _B; C = _C; //システムの係数を設定
        x = x0; //xの初期値を代入
    }

    public void control(double u){ //制御入力を受け取り、xを1ステップ進める
        x = A*x + B*u;
    }

    public double observe(){ //観測
        return C*x;
    }
}
```

注：簡単のため、ここではx, u, yをスカラーとした。ベクトルを扱うには、VectorやMatrixといったクラスを定義し、掛け算や足し算などの演算をメソッドとして定義してやればよい。

# おしながき

- What is Java? Why Java?
- オブジェクト指向入門
  - クラスとインスタンス
  - カプセル化
  - **クラスの継承**
  - ポリモーフィズム
- オセロ大会でのポリモーフィズムの応用
- プログラミング上達のためのコツ

# class 哺乳類

## class 犬

ちこちゃん

白戸次郎

パトラッシュ

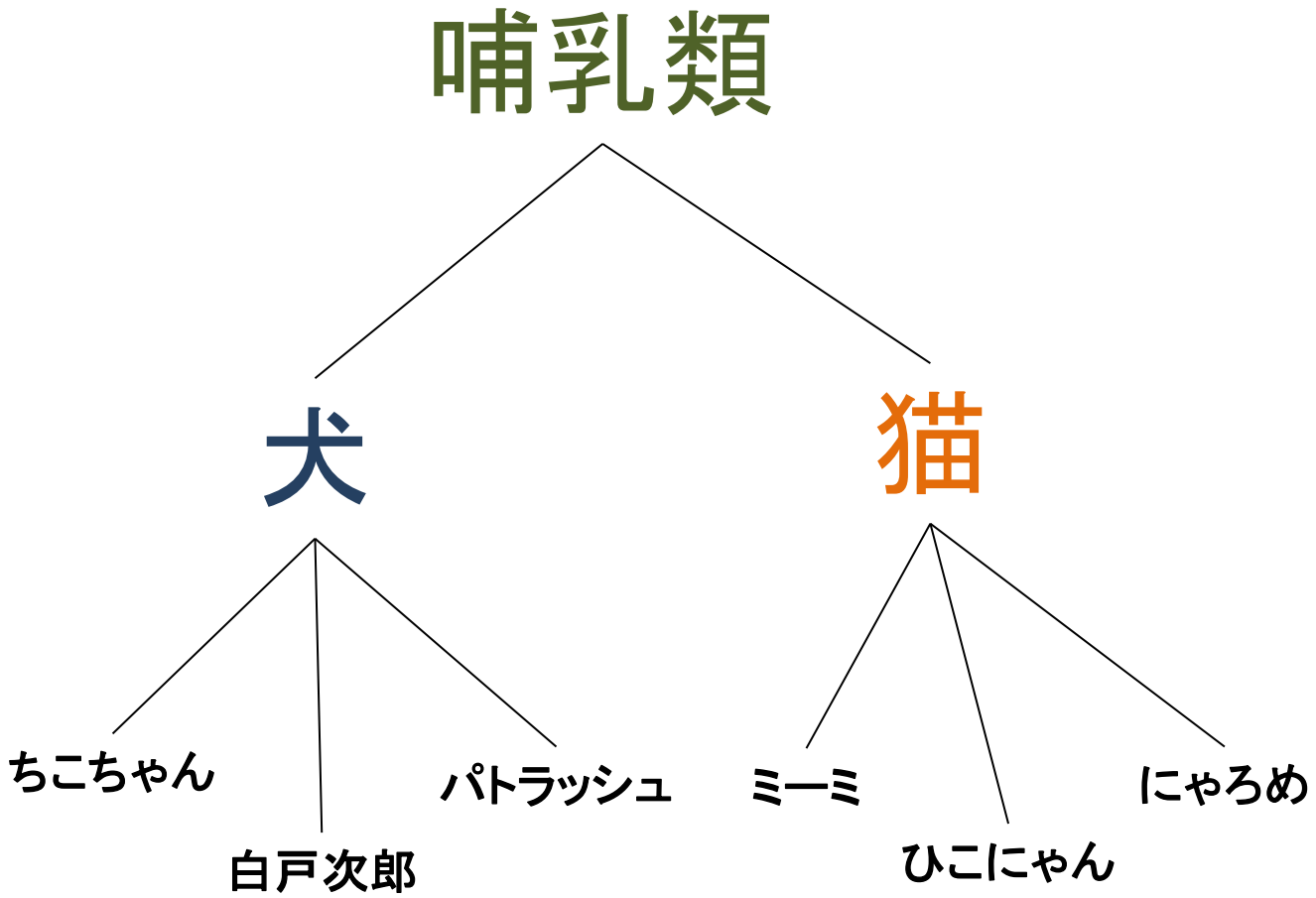
## class 猫

みーみ

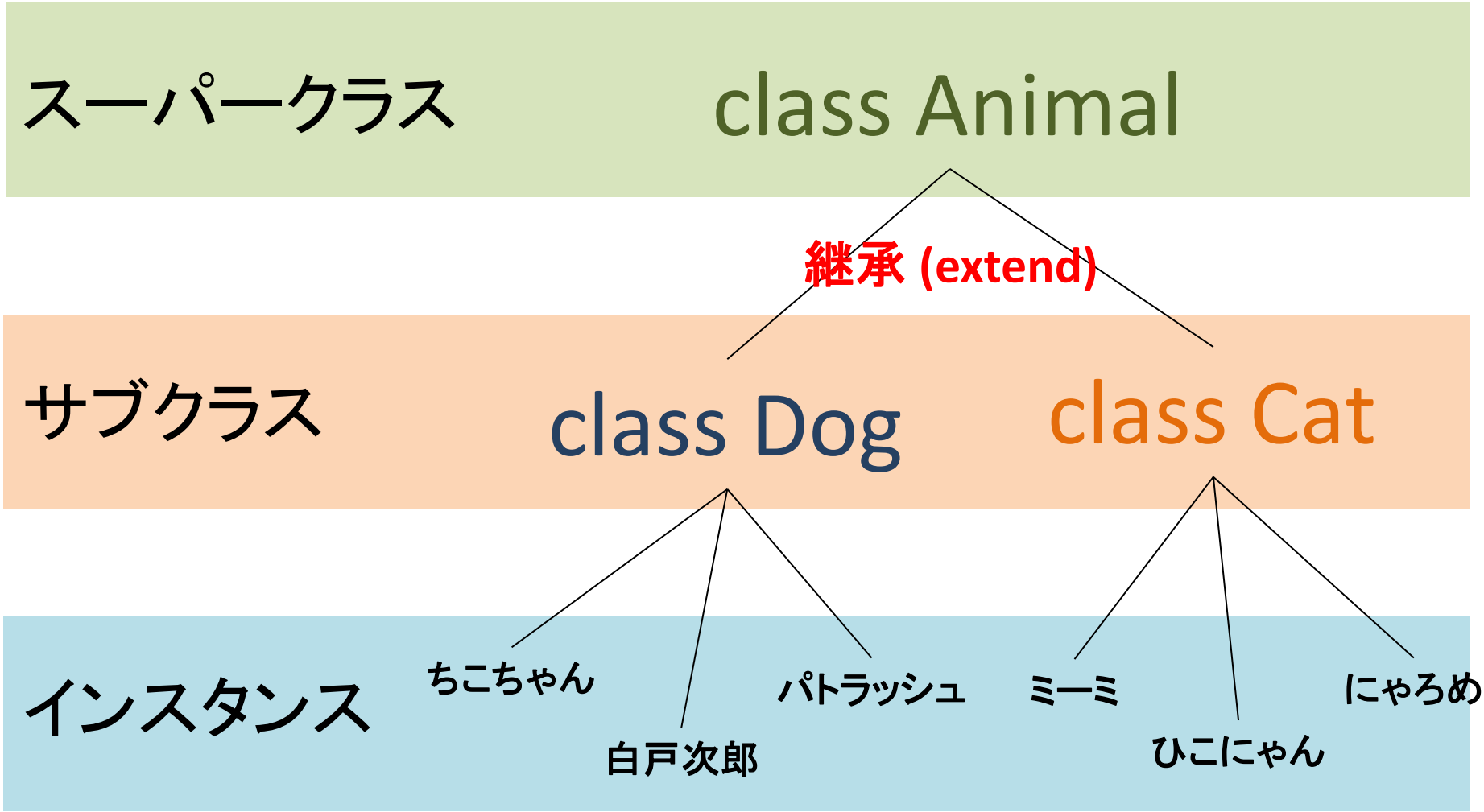
ニャロメ

ひこにゃん

# クラスの階層構造



# オブジェクト指向のクラスの階層構造



# Javaのクラスの継承

```
class Animal{
....
}

class Dog extends Animal{
....
}

class Cat extends Animal{
....
}
```

- class B extends A と宣言すれば、クラスAを継承したクラスBを作成できる
- Q. クラスを継承できて、なにが嬉しいの？？
- A. 共通の機能を二度書かずに済む
  - 他にもいろんな嬉しいことがあるが、後述する。

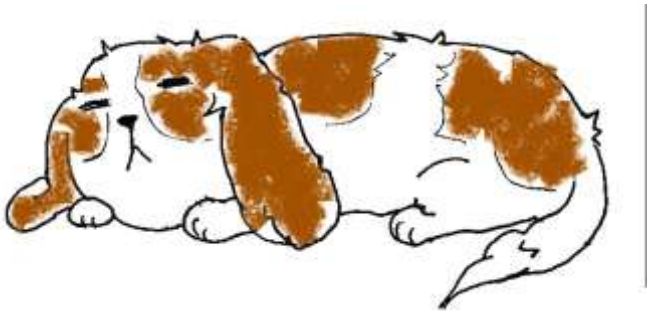
# クラスの階層構造：共通性と差異

- Q. なぜ犬と猫はともに同じ「哺乳類」に分類されるか？
- A. 共通する特徴や機能を持っているから
  - 特徴（定数、変数）：目が二つ、足が四本
  - 機能（メソッド）：子供を産む、授乳する、肺呼吸する

- Q. なぜ犬と猫は同じ種ではないか
- A. 異なる特徴や機能もあるから
  - 特徴：猫はお腹に毛が生えてる、犬は生えてない
  - 機能：猫の高いジャンプ力

# class Dog

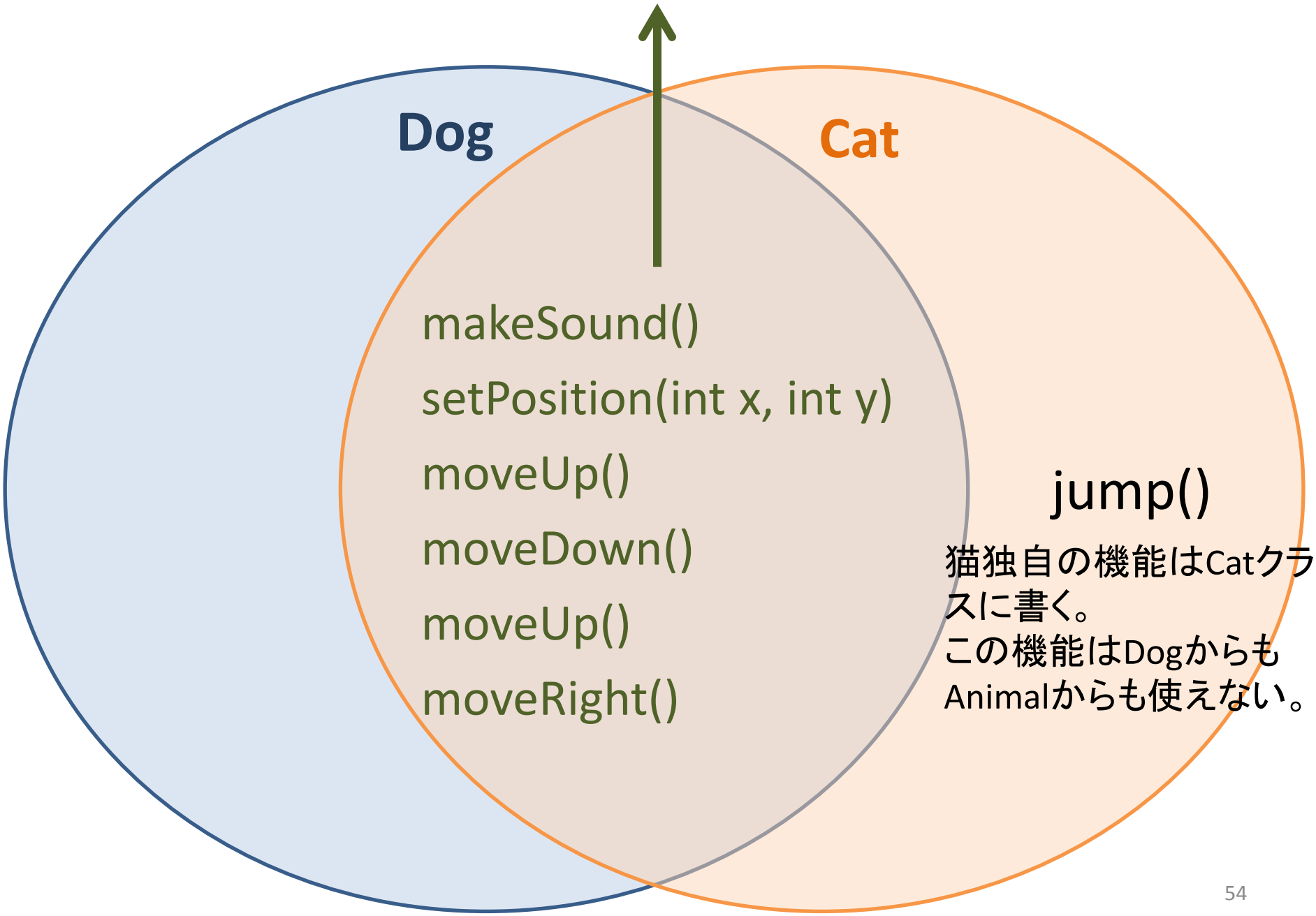
- 変数・定数
  - 名前、色
  - 現在位置
- メソッド
  - makeSound() 鳴く
  - setPosition(int x, int y) (x, y)に移動する
  - moveUp() 上に移動する
  - moveDown() 下に移動する
  - moveUp() 左に移動する
  - moveRight() 右に移動する



# class Cat

- 変数・定数
  - 名前、色
  - 現在位置
- メソッド
  - makeSound() 鳴く
  - setPosition(int x, int y) (x, y)に移動する
  - moveUp() 上に移動する
  - moveDown() 下に移動する
  - moveUp() 左に移動する
  - moveRight() 右に移動する
  - jump() ジャンプする

共通の機能はAnimalクラスに書く。これらの機能はDog, Catからも使える



# クラス継承で嬉しいこと　その1

- 複数のクラスで重複する機能は、スーパークラスにまとめれば、一度しか書かなくて済む。

# ソースコードを見てみよう

- Animal.javaをみてみよう
  - moveRight(), makeSound()などが実装されている
- 次に、Cat.javaをみてみよう
  - moveRight(), makeSound()などはここには書かれてない
  - jump()だけここにある
- でも、Animalのメソッドはcatオブジェクトから使える。

```
Dog pochi = new Dog(Color.WHITE, "Pochi");
Cat tama  = new Cat(Color.BLACK, "Tama");

pochi.moveRight(); //Animalで定義されたメソッドをDogクラスのインスタンスから呼び出せる。
tama.moveRight();  //Animalで定義されたメソッドをCatクラスのインスタンスから呼び出せる。

tama.jump();       //Catクラスで定義されたメソッドは当然タマは使えるが、
pochi.jump();      //ポチが使おうとしたらエラーが出る。
```

# おしながき

- What is Java? Why Java?
- オブジェクト指向入門
  - クラスとインスタンス
  - カプセル化
  - クラスの継承
  - ポリモーフィズム
- オセロ大会でのポリモーフィズムの応用
- プログラミング上達のためのコツ

# 三段論法

- ミーミは猫である
- 猫は動物である
- よってミーミは動物である

# オブジェクト指向的三段論法

- mimiはCatクラスのインスタンスである
- CatクラスはAnimalクラスのサブクラスである
- よってmimiはAnimalクラスのインスタンスとしても扱える
  - ただし、使えるメソッド（機能）はAnimalクラスのものに限られる

```
Animal mimi  = new Cat(Color.BLACK, "Mimi"); //ミーミをAnimal型のインスタンスに格納
mimi.makeSound();    //Animalクラスのメソッドを使える
mimi.jump();         //でもCatクラス独自のメソッドは使えない。
                     //なぜならインスタンスmimiはAnimal型として宣言されているから
((Cat)mimi).jump(); //でも、キャスト演算子(<クラス名>)を用いてCat型にキャストすれば、
                     //Cat型独自のメソッドを使えるようになる
```

# クラス継承で嬉しいこと　その2

- 同じクラスを継承した異なるクラスを、一緒くたに扱える
  - とりわけ、ひとつの配列に異なるクラスを入れられるのが便利

**もちろん、Dog型の配列にCat型のインスタンスは入れられないけど・・・**

```
Dog myPets[] = new Dog[2]; //Javaの配列はこうやって宣言する
myPets[0] = new Dog(Color.WHITE,"Chiko");
myPets[1] = new Cat(Color.WHITE,"Mimi");
```

**Animal型の配列には、Dog型インスタンスもCat型インスタンスも入れられる。**

```
Animal myPets[] = new Animal[2]; //Javaの配列はこうやって宣言する
myPets[0] = new Dog(Color.WHITE,"Chiko");
myPets[1] = new Cat(Color.WHITE,"Mimi");
```

# protected?

- private - 自分のクラスからしか参照できない
- protected – 自分自身とそのサブクラスから参照できる
- public – どのクラスからでも参照できる

# 余談：可変長配列

- Javaは標準で可変長配列をサポートしている
  - 平たく言えば、いつでも要素を追加したり削除したりして長さを変えられる配列のこと
  - Interface: List, Set, Mapなど
  - 実装：ArrayList, LinketList, TreeSetなど

```
List<Animal> myPets = new LinkedList<Animal>(); //可変長配列の宣言
myPets.add(new Dog(Color.WHITE,"Chiko"));    //チコちゃんをリストに追加
myPets.add(new Cat(Color.WHITE,"Mimi"));     //ミーミをリストに追加
//好きな数だけ動物を追加できる。
```

*Playground.javaの27行目、45行目にこれが使われています！

# 余談：イタレーター

- While文の中で可変長配列から要素をひとつづつ取り出すのに便利な機能
  - hasNext(), next()のふたつのメソッドから成る

**以下のコードを実行すると、まず「ワン」、次に「ニャー」と音が鳴ります。**

```
List<Animal> myPets = new LinkedList<Animal>(); //可変長配列の宣言
myPets.add(new Dog(Color.WHITE,"Chiko"));    //チコちゃんをリストに追加
myPets.add(new Cat(Color.WHITE,"Mimi"));     //ミーミをリストに追加

Iterator<Animal> itr = myPets.iterator(); //ペットのリストのイタレーターを取得
while(itr.hasNext()){ //hasNext()は、まだ次の要素が存在すればtrueを返す
    itr.next().makeSound(); //next()で次の要素を取り出し、makeSound()で鳴かせる。
}
```

*PlayGround.javaの125行目にイタレーターが使われています。

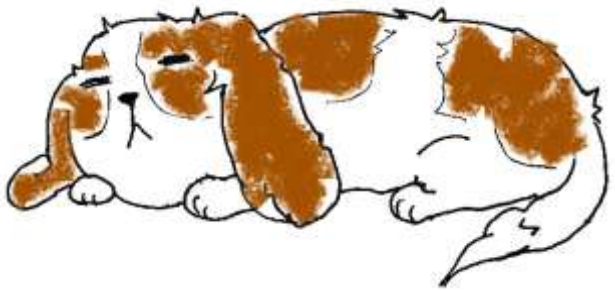
# ついに本題　ポリモーフィズム入門

- Animalクラスのpoke()という関数に注目
  - poke: つっつく、という意味
- この動物がつっつかれた (i.e.,クリックされた)ときの処理を書く関数
  - PlayGround.javaの129行目で使われている

```
@Override
public void mousePressed(MouseEvent e) { //マウスがクリックされたときに呼ばれる関数
    int x = e.getX();
    int y = e.getY();
    Iterator<Animal> itr = allAnimals.iterator(); //ListやSetなど可変長配列に対するwhile文は
    while(itr.hasNext()){ //allAnimalに格納されているすべての動物に対してループ
        Animal nextAnimal = itr.next(); //つぎの動物を配列から取り出す
        if(nextAnimal.isHit(x, y)){//もしクリックされた位置がこの動物の上だったら
            nextAnimal.poke(); //つっつく
        }
    }
}
```

余談：マウスからの入力を処理するには、MouseListnerというインターフェースを継承し、コンストラクタの中で addMouseListener(**this);** というおまじないを書く。PlayGround.java参照

# poke()メソッドの動作は犬と猫で違う



- 「ワン」と吠える
- やる気のないメッセージをウィンドウ上部に出す
- Dog.javaの43行目あたりに書かれている

- 「ニャー」と鳴く
- 素っ気ないメッセージをウィンドウ上部に出す
- ジャンプして逃げる
- Cat.javaの43行目あたりに書かれている

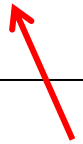
**ポリモーフィズム (Polymorphism)** : ひとつの手続きで、多様な機能を実現すること

# でもちょっとまて

- PlayGround.javaでは、Animal型のインスタンスからpoke()が呼ばれている
- Animal型では犬も猫も同様に扱われるはず
- なのにどうして、犬と猫で違う動作をするの！？

PlayGround.java 126行目付近

```
Animal nextAnimal = itr.next(); //つぎの動物を配列から取り出す
if(nextAnimal.isHit(x, y)){//もしクリックされた位置がこの動物の上だったら
    nextAnimal.poke(); //つっつく。
}
```



nextAnimalはAnimal型のインスタンスなので、ここではAnimalクラスの中にあるpoke()の定義が呼ばれるはず。なのにどうして犬と猫で違う動作をする？

# ポリモーフィズムはオーバーライドという仕組みで実現されている

- オーバーライド＝上書き

Animal.java

```
class Animal{
        ....
        public abstract void poke(); //メソッドpoke()の定義
        ....
}
```

**Animalで定義されたメソッドpoke()が・・・**

Dog.java

```
public class Dog extends Animal {
        ...
        @Override
        public void poke() {
                makeSound(); //まず、鳴く。
                say(messages[rnd.nextInt(messages.length-1)]); //メッセージを表示。
        }
}
```

**Dogで上書きされている**

# 抽象メソッドという概念

- 抽象メソッド＝実態のない、定義だけのメソッド

Animal.java

```
class Animal{
        ....
        public abstract void poke();
        ....
}
```

抽象メソッドは、メソッド（関数）の中身はない

塾生注目！abstractと接頭辞をつけると、抽象メソッドになる

Dog.java

```
public class Dog extends Animal {
        ...
        @Override
        public void poke(){
                makeSound(); //まず、鳴く。
                say(messages[rnd.nextInt(messages.length-1)]); //メッセージを表示。
        }
}
```

サブクラスの中で抽象メソッドの中身が実装される！

# Animal型インスタンスから抽象メソッドpoke()が呼ばれると、

```
Animal nextAnimal = itr.next(); //つぎの動物を配列から取り出す
if(nextAnimal.isHit(x, y)){//もしクリックされた位置がこの動物の上だったら
    nextAnimal.poke(); //つっつく
}
```

nextAnimalの実体がDog型だったらDogクラスのpoke()の実装が呼ばれ、

nextAnimalの実体がCat型だったらCatクラスのpoke()の実装が呼ばれ、

Dog.poke()

- 「ワン」と吠える
- やる気のないメッセージをウィンドウ上部に出す

Cat.poke()

- 「ニャー」と鳴く
- 素っ気ないメッセージをウィンドウ上部に出す
- ジャンプして逃げる

# 用語

- 抽象クラス：抽象メソッドを含むクラス
- インターフェース：抽象メソッドだけしか含まないクラス

- 抽象クラスもインターフェースも、そのままじゃ使えない
  - 抽象メソッドを実装したサブクラスを作らなくては、実体（インスタンス）を作成できない
  - 実世界でも – 犬でも猫でも、いかなる種でもない「哺乳類」は存在しない。

# ポリモーフィズムの威力は複数人で共同開発するときに発揮される！！

Aさん：抽象クラスAnimal.javaを書く。具体的に何をさせるかはおいておいて、マウスでクリックされたときの処理をpoke()に書く、というインターフェースのみ決める

Animal.java

```
class Animal{
        public abstract void poke();
}
```

BさんとCさん：それぞれDog.javaとCat.javaを書く。Aさんの書いたものを一切変更せずに、犬・猫独自の動作を記述できる。

Dog.poke()

Cat.poke()

Dさんが後に熊とか虎とか作っても、A, B, Cさんのコードに変更不要

# 練習問題

Dog.poke()やCat.poke()を自分で変更し、動作がどのように変わるか確かめてみてください

ヒント:

- say(“ほにゃらら”) ： ウィンドウ上部に「ほにゃらら」と表示される
- makeSound() ： 鳴かせる
- setPosition(int x, int y) ： (x,y)にこの動物を移動させる
- moveLeft(), moveRight(), moveUp(), moveDown ：上下左右に移動

以下に公開されているドキュメントも参考にしてください:

http://yoda.appi.keio.ac.jp/Othello/IntroToJava/doc/

# 余談：Javadoc

- 自動でドキュメントを作成してくれるツール

**ソースファイル**

```
public abstract void poke();

/**
 * この動物の色を変更する
 * @param newColor 新しい色
 */
protected void changeColor(Color newColor){
    AnimalColorChanger filter = new AnimalColorChanger(new
    coloredImage = Toolkit.getDefaultToolkit().createImage
}

/**
 * 与えられたメッセージをウィンドウ上部のラベルに表示する
 * @param message
 */
protected void say(String message){
    label.setText(this.name + ":   " + message );
}

/**
 * この動物を右に動かす
 */
public void moveRight(){
    x = x+stepSize;
}

/**
 * この動物を左に動かす
 */
public void moveLeft(){
    x = x-stepSize;
}
```

**ソースファイルの /** … */で囲った部分**

Javadoc

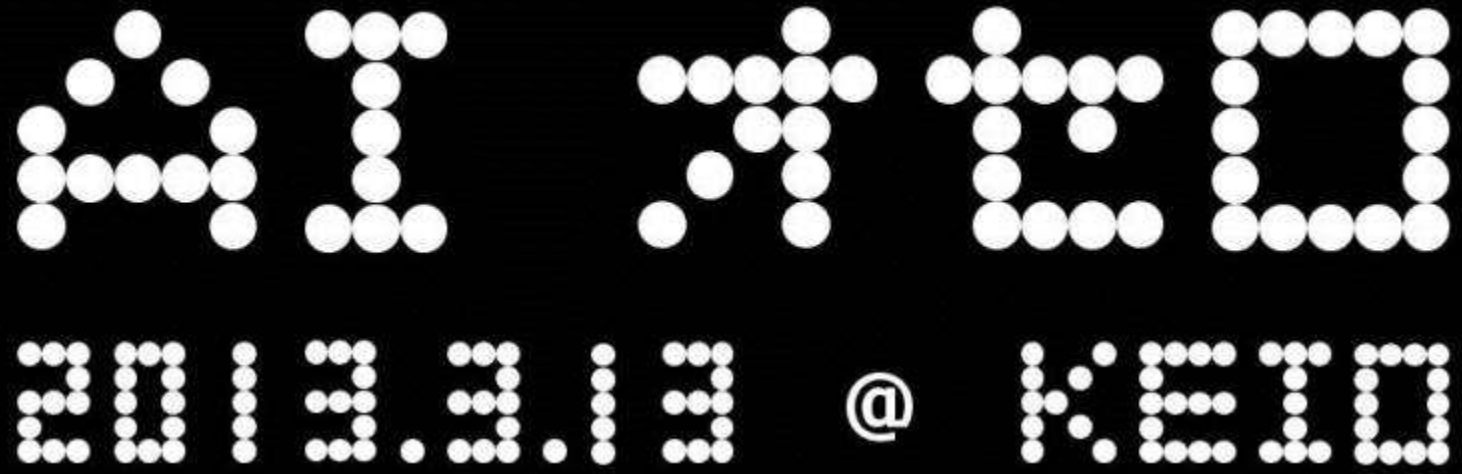
**HTMLドキュメント**

Methods

| Modifier and Type | Method and Description |
|---|---|
| int | **getHeight**()<br>この動物の高さを返す |
| java.awt.Image | **getImage**()<br>この動物の画像オブジェクトを返す |
| javax.swing.JLabel | **getLabel**()<br>メッセージを表示するラベルを表すオブ |
| java.lang.String | **getName**()<br>この動物の名前を返す |
| int | **getWidth**()<br>この動物の横幅を返す |
| int | **getX**()<br>この動物の現在位置のX座標を返す |
| int | **getY**()<br>この動物の現在位置のY座標を返す |
| boolean | **isHit**(int xin, int yin)<br>(x,y)であらわされる画面上の位置が、 |
| void | **makeSound**()<br>この動物を鳴かせる |
| void | **moveDown**()<br>この動物を下に動かす |
| void | **moveLeft**()<br>この動物を左に動かす |
| void | **moveRight**()<br>この動物を右に動かす |

# おしながき

- What is Java? Why Java?
- オブジェクト指向入門
  - クラスとインスタンス
  - カプセル化
  - クラスの継承
  - ポリモーフィズム
- オセロ大会でのポリモーフィズムの応用
- **プログラミング上達のためのコツ**




- オセロ大会
- ただし、人間がプレイするのではなく、人間が作ったプログラム同士が対戦する
- 3月13日　本大会
- ルール:
  - ゲームのルールやグラフィックスはJAVAで実装済みのものを渡す。
  - 戦略をあらわす抽象クラスを各参加者が実装

# オセロ大会のポリモーフィズム

- 各参加者がそれぞれ異なる戦略を実装する
- しかし、全ての戦略は同じインターフェースで動かされる
- ポリモーフィズムを使う！

## 動物の動作のテンプレート

poke() : つっつかれたときに反応する

**抽象メソッド：Animal.poke()**

```
class Animal{
        public abstract void poke();
}
```

**具体的に「どう」反応するかを、サブクラスで実装**

**オーバーライド・実装**

**オーバーライド・実装**

Dog.poke()

- 「ワン」と吠える
- やる気のないメッセージをウィンドウ上部に出す

Cat.poke()

- 「ニャー」と鳴く
- 素っ気ないメッセージをウィンドウ上部に出す
- ジャンプして逃げる

## 戦略のテンプレート

nextMove(…) : 現在の盤面(currentState)と残り時間(remainingTime)を入力とし、次に打つ手(Move型)を返す

**抽象メソッド : game.Strategy. nextMove(…)**

```
class Strategy{
        public abstract Move nextMove(GameState currentState, int remainingTime);
}
```

**具体的に「どう」次の手を決めるかをサブクラスで実装**

**オーバーライド・実装**

SuzukiStrategy.nextMove(…)

鈴木さんの戦略を実装

- 「開放度」の一番低いマスを選ぶ
- Mini-max searchで10手先まで先読みする
- ・・・

SatoStrategy.nextMove()

佐藤さんの戦略を実装

- 「モビリティー」の一番高いマスを選ぶ
- Alpha-beta searchで15手先まで先読みする

# オセロのプログラムの概略

- ①黒の手番なら鈴木さんの戦略を呼び出し
- ②白の手番なら佐藤さんの戦略を呼び出し
- ③これをゲーム終了まで続ける

exe/Othellomain.java （若干簡略化して書いてあります）

```
Strategy blackStrategy = new SuzukiStrategy(); //戦略の定義。黒は鈴木さん
Strategy whiteStrategy = new SatoStrategy();   //戦略の定義。白は佐藤さん

while(game.isInGame()){ //③ゲーム終了まで繰り返し
        if(game.getCurrentPlayer() == Player.Black){ //黒の手番
                //①黒の戦略を呼び出し、次の手を打つ。
                game.putPiece(blackStrategy.nextMove(現在の盤面、残り時間);
        }else{ //白の手番
                //②白の戦略を呼び出し、次の手を打つ。
                game.putPiece(whiteStrategy.nextMove(現在の盤面、残り時間);
        }
}
```

# オセロのプログラムの概略

- 鈴木さん v.s. 五十嵐さんをやりたいなら・・・
  - プログラムをたった一行変更するだけでいい。

exe/Othellomain.java （若干簡略化して書いてあります）

```
Strategy blackStrategy = new SuzukiStrategy(); //戦略の定義。黒は鈴木さん
Strategy whiteStrategy = new IgarashiStrategy();   //戦略の定義。白は五十嵐さん

while(game.isInGame()){ //③ゲーム終了まで繰り返し
        if(game.getCurrentPlayer() == Player.Black){ //黒の手番
                //①黒の戦略を呼び出し、次の手を打つ。
                game.putPiece(blackStrategy.nextMove(現在の盤面、残り時間);
        }else{ //白の手番
                //②白の戦略を呼び出し、次の手を打つ。
                game.putPiece(whiteStrategy.nextMove(現在の盤面、残り時間);
        }
}
```

# おしながき

- What is Java? Why Java?
- オブジェクト指向入門
  - クラスとインスタンス
  - カプセル化
  - クラスの継承
  - ポリモーフィズム
- オセロ大会でのポリモーフィズムの応用
- **プログラミング上達のためのコツ**

# どうやったらプログラミングが上達するか？

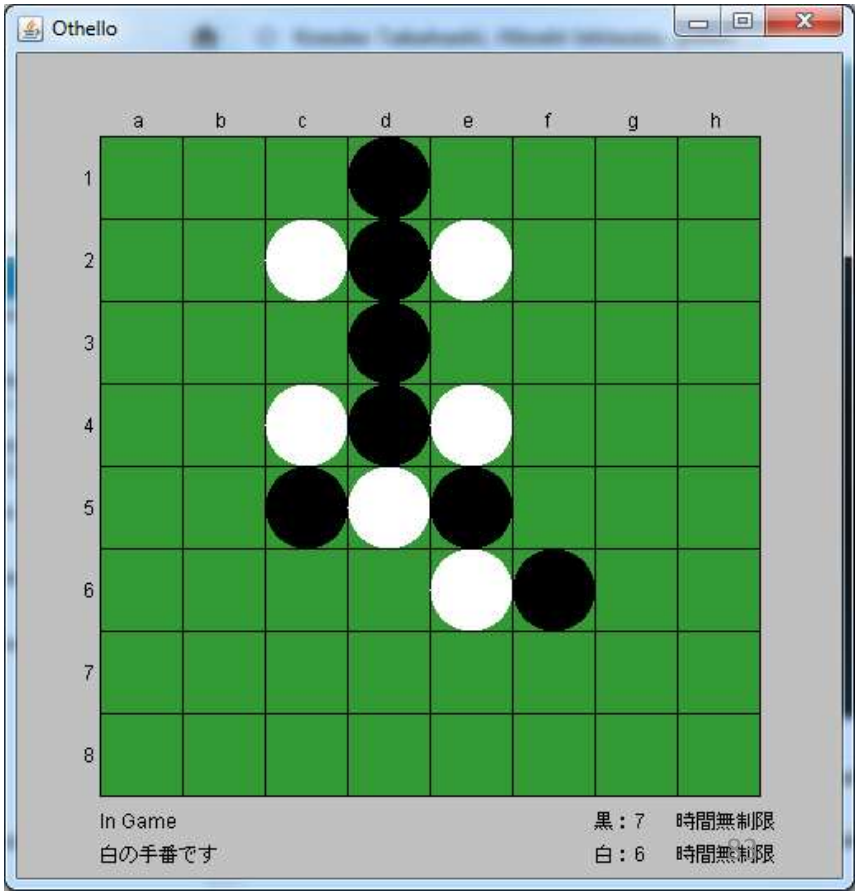
小野の例

- 小6の時にパソコンとC言語の本を父からもらう
- 中学の頃、趣味でいろいろとプログラムを作る
- 主にゲームを作った
  - 数当て
  - 神経衰弱
  - 15パズル

# どうやったらプログラミングが上達するか？

## *好きこそものの上手なれ*

- ゲームを作ろう！
- オセロ大会を良い機会にしてください



# AI Tea

2013.3.13 @ Keio

# AndroidアプリはJava


- 開発キット（SDK）は無償配布
  - http://developer.android.com/sdk/
- 公式に推奨されている開発環境はeclipse
- 25ドルの登録料を一度払えば、アプリをGoogle Play Storeで売れる
  - 開発者は売り上げの7割をもらえる
- ちなみにiOSはObjective-Cを使用
- ぜひ挑戦してみては！？

# まとめ

- チコちゃんとミーミは小野家で飼われてた犬猫
- 現代の主流はJavaなどオブジェクト指向言語
- 学習した概念
  - クラスとインスタンス
  - カプセル化
  - スーパークラスとサブクラスと継承
  - ポリモーフィズム
  - 抽象メソッドのオーバーライド
- *好きこそものの上手なれ！！*