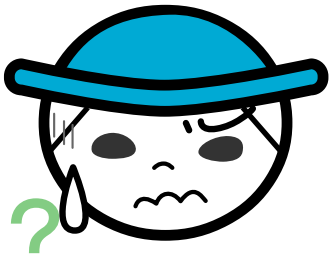
ぐらみん株式会社

Java の抽象クラス・インタフェース・無名クラスを理解しよう

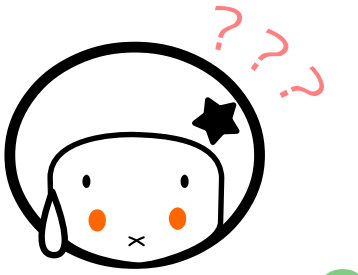
Java の継承を深く理解しよう

抽象クラス、インタフェース、無名クラスの関係を理解しよう



インタフェース

抽象クラス



無名クラス

資料の紹介

Gra

min

この資料はスマートフォンアプリ開発クラス向けに作成

小学生〜中学生を対象としたプログラミング教室 Gramin

Java 基本構文は理解しクラスの理解に苦しむ方が対象

抽象クラス・インタフェースの書き方は知っている方が対象

本来は授業中にこの資料に近いものを元に口頭で説明

作成した背景

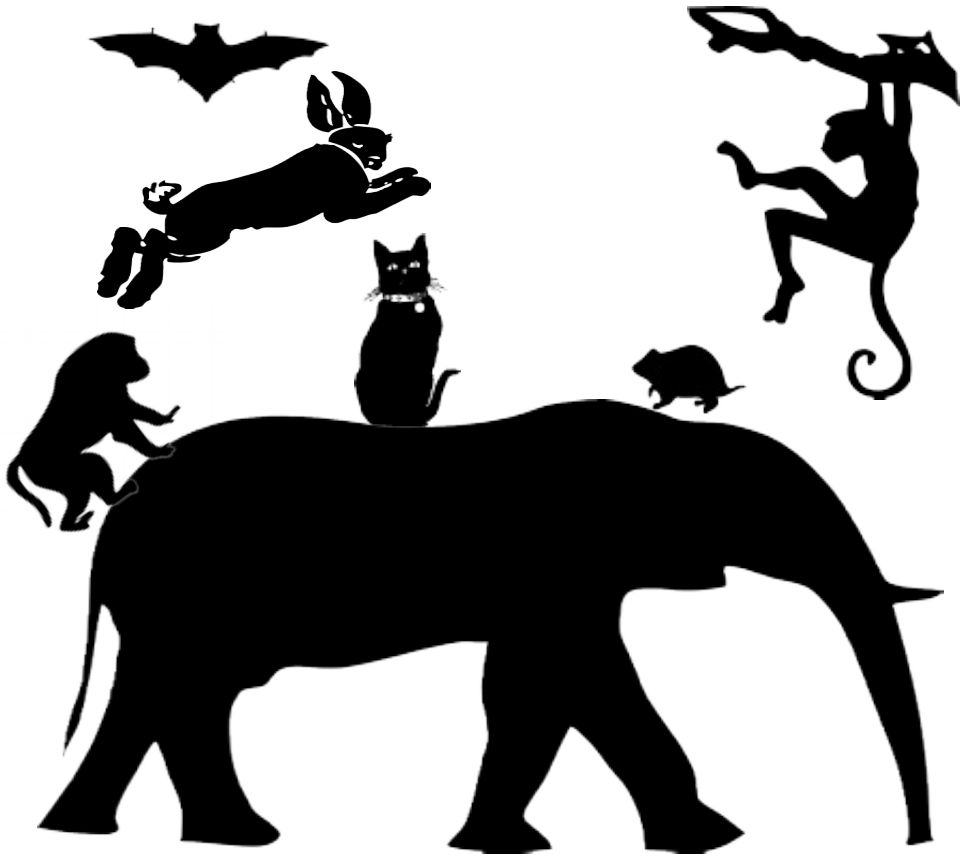
なぜインタフェースがあるか理解してもらえない。。。

抽象クラス･インタフェースがかぶる。。。

Android では無名クラスをよく使うのでそれも説明したい

資料は構造化（目次とか）すると内容が大きくなり
口頭で説明しにくくなるのでフラットにしています

哺乳類がいます



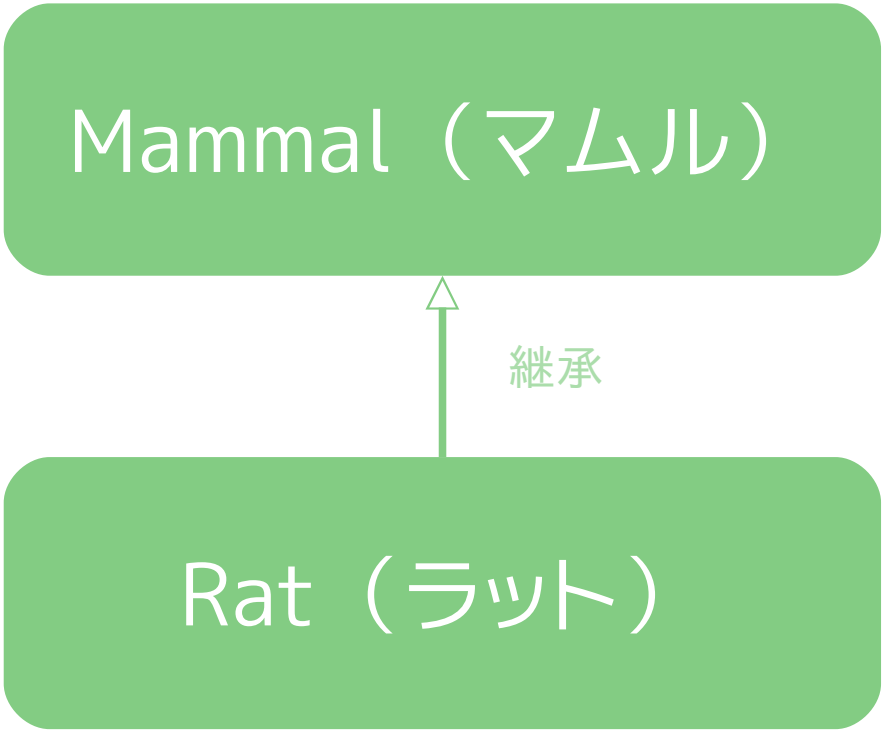
Mammal（マムル）

哺乳類にはネズミがいます

Rat（ラット）

ネズミは哺乳類です

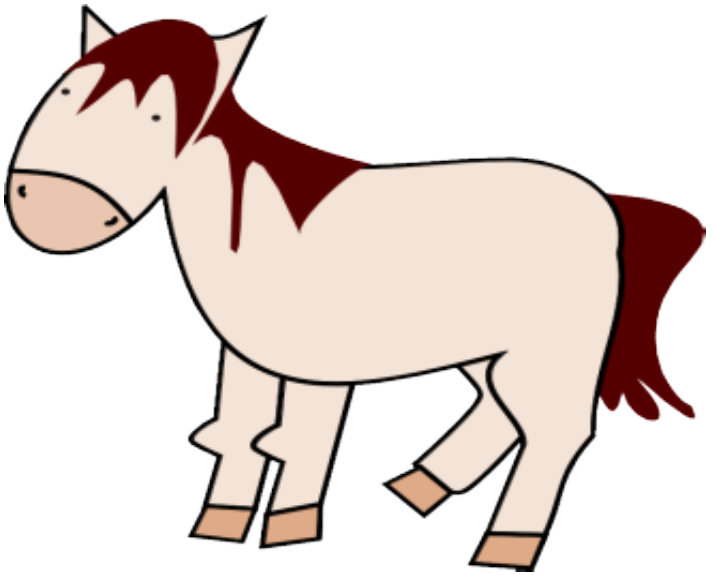
ネズミは哺乳類を“継承”しています



↑ 継承関係はこんな矢印で示す（親に矢印が向く）

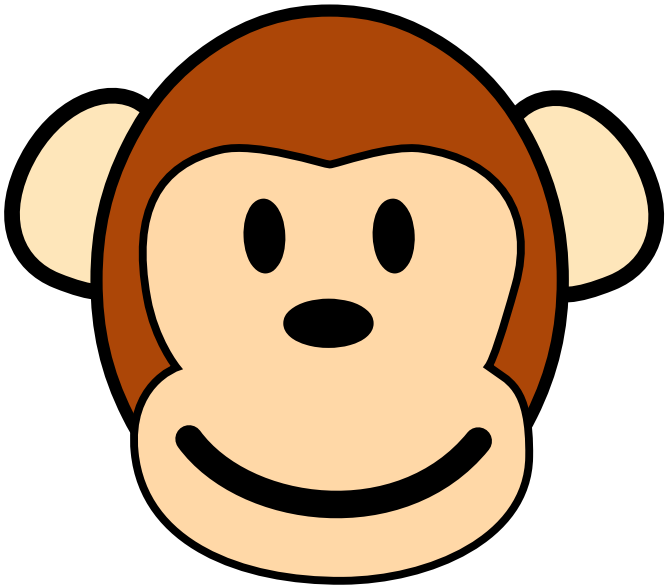
哺乳類には馬もいます

馬は哺乳類を“継承”しています

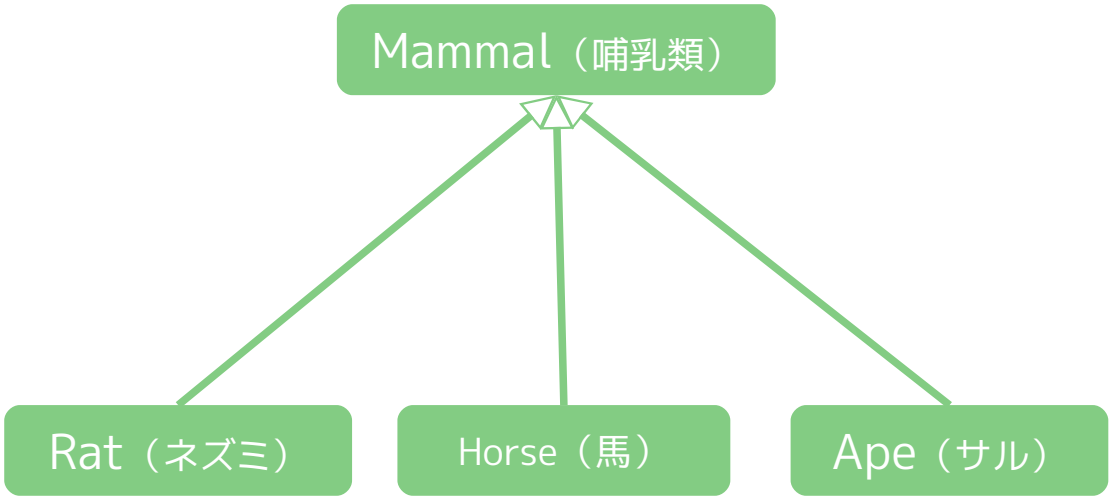


哺乳類にはサルもいます

サルは哺乳類を“継承”しています

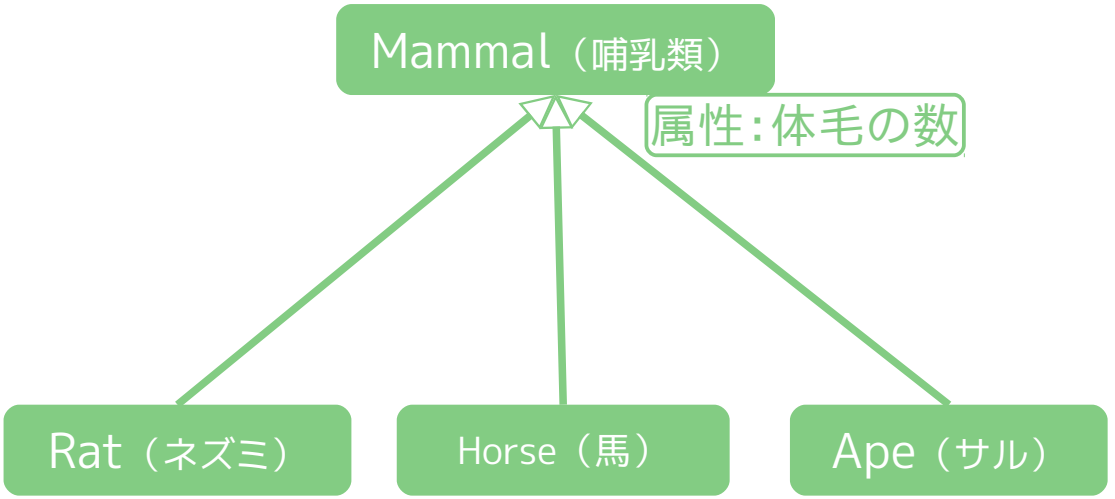


今の時点の継承関係



哺乳類は体毛の数を持ちます

オブジェクト志向で”体毛の数”は “属性” として表現できる



継承すると属性も継承される

哺乳類は実際にいる生物？

“哺乳類”という生き物は存在する？

＝

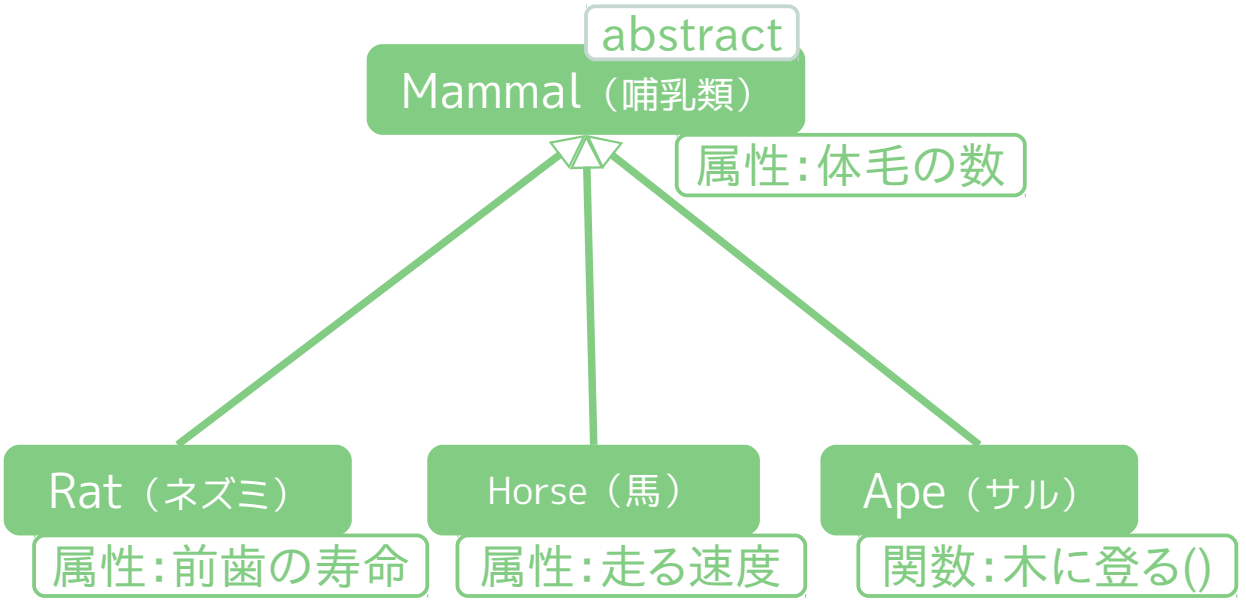
new できる？

“哺乳類”という単位で new するとおかしくなる（実際に存在せず”分類”でしかないから）

じゃあ、抽象クラス（abstract）にしよう！

今の時点の継承関係

ついでにネズミ・馬・サルに属性と関数をつけてみました



爬虫類もいます

Reptilia（レプティリア）

爬虫類にはトカゲがいます

トカゲは爬虫類を“継承”しています

爬虫類は実際にいる生物？

new??

“爬虫類”という生き物は存在する？

＝

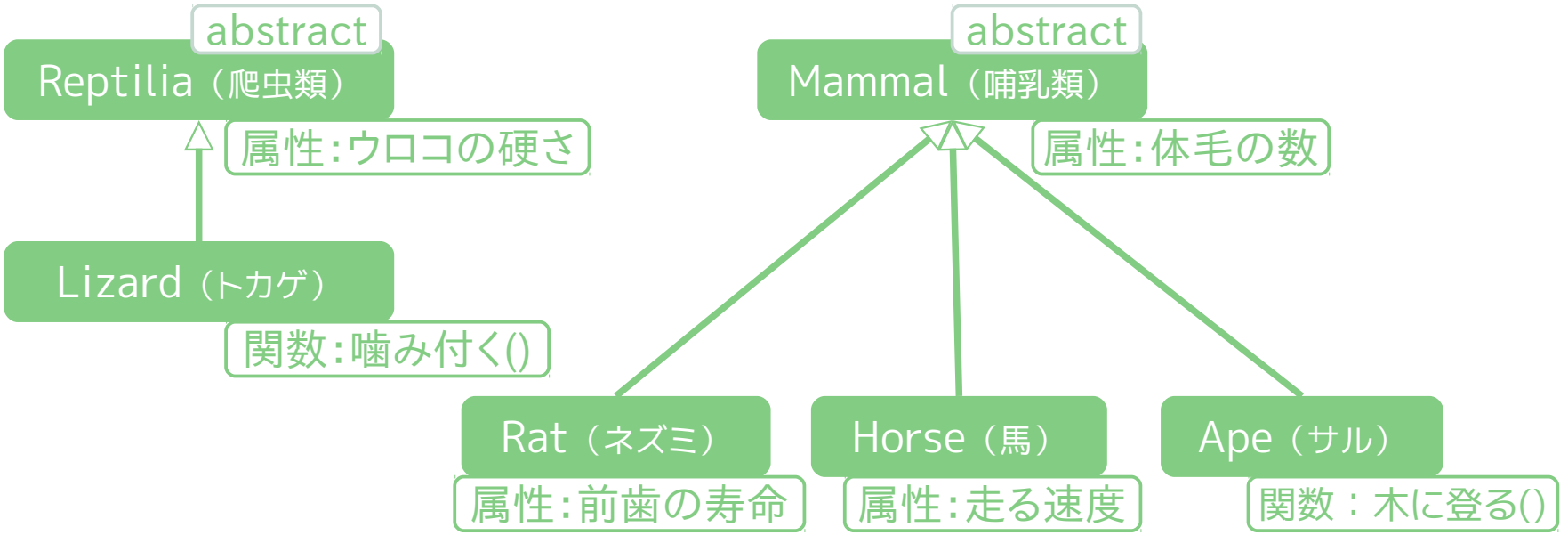
new できる？

“爬虫類”という単位で new するとおかしくなる（実際に存在せず”分類”でしかないから）

じゃあ、爬虫類も抽象クラス（abstract）にしよう！

今の時点の継承関係

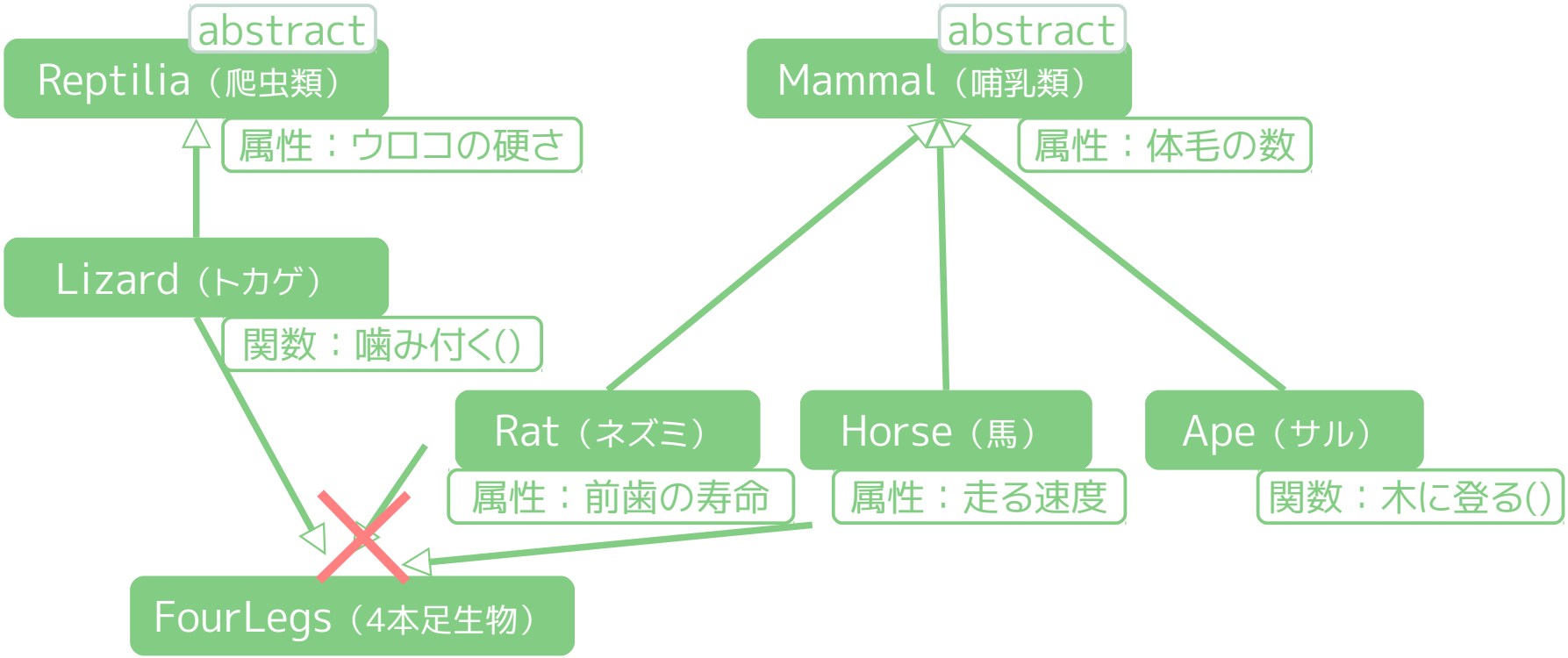
爬虫類にウロコの硬さ属性とトカゲに噛み付くもつけてみました



だんだん大きくなってきた。。。

トカゲもネズミも馬も４本足です

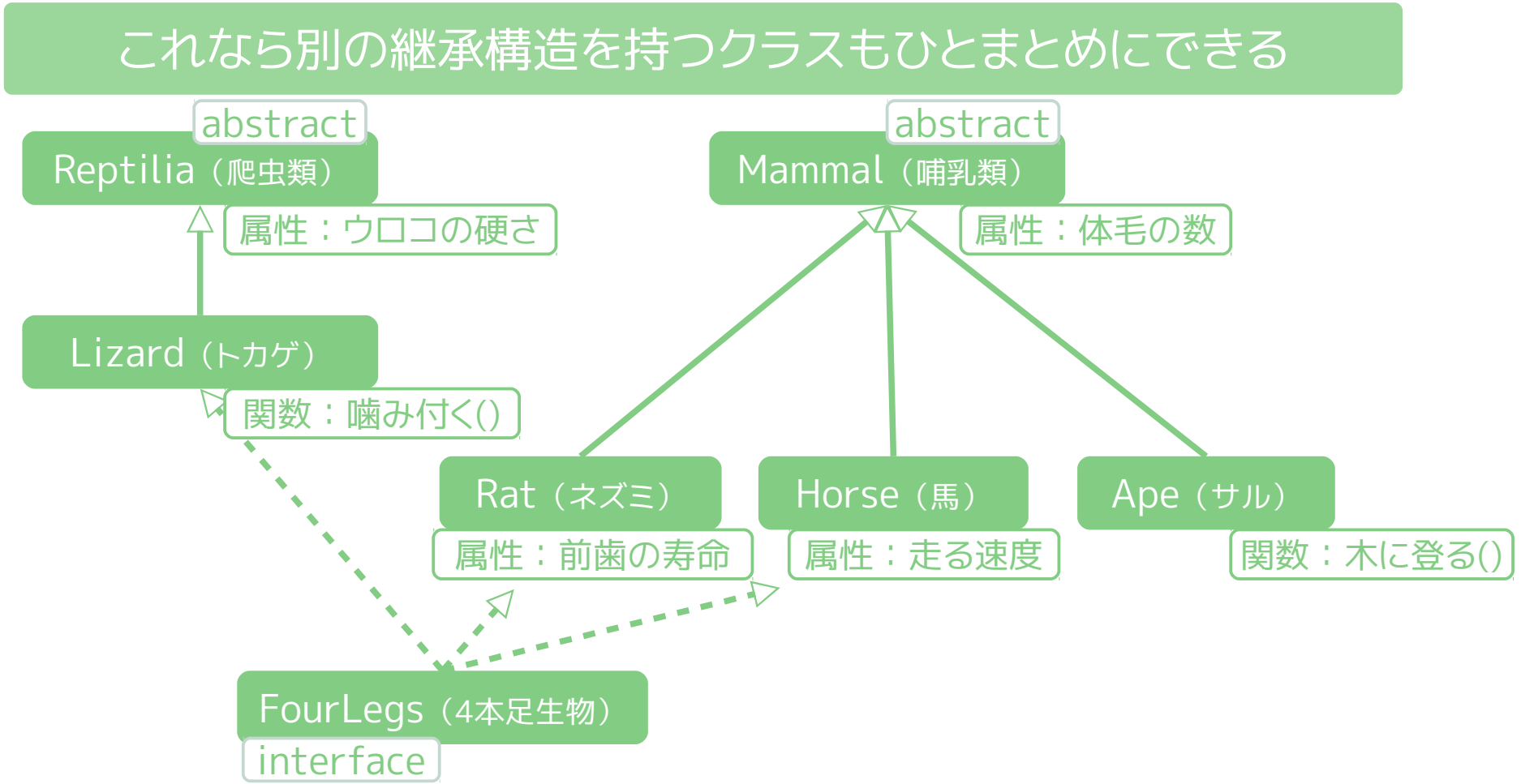
こんな時どうやって４本足生物の継承関係を表現する？



多くのオブジェクト志向言語では親を二人以上持つことは許されない（多重継承）

C++ではOK

４本足はインタフェースにします



インタフェースはいくつでも実装可能（継承ではなく実装）

４本足生物（インタフェース）は前足で蹴ります

中身はよく分からなくても「前足で蹴る」生物がいることが分かる

FourLegs（4本足生物）

関数：前足で蹴る()

これがインタフェースの役立つところ

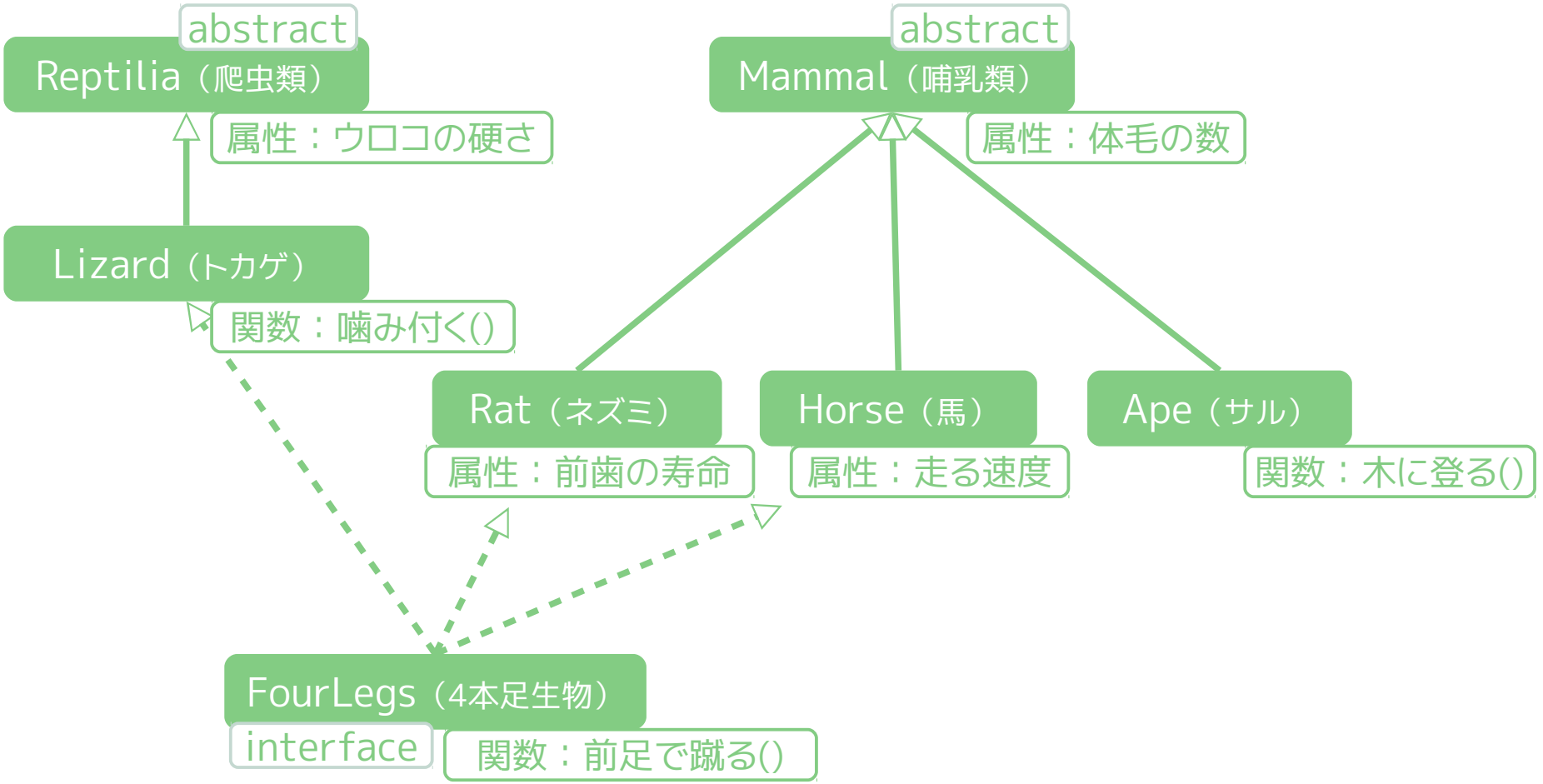
4本足生物の使い道は？

前足で蹴ることを知っているだけでも敵を攻撃することはできます

誰が中身か知らないけど
蹴ってくれます

敵を攻撃する目的はそれだ
けでも十分果たしています

今の時点の継承関係


abstract
Reptilia（爬虫類）
属性：ウロコの硬さ
Lizard（トカゲ）
関数：噛み付く()
abstract
Mammal（哺乳類）
属性：体毛の数
Rat（ネズミ）
属性：前歯の寿命
Horse（馬）
属性：走る速度
Ape（サル）
関数：木に登る()
FourLegs（4本足生物）
interface
関数：前足で蹴る()

４本足生物は知らないとダメ？

前足で蹴るってことだけ知っていればいいんじゃない？

Java の場合”前足で蹴る“
関数の中身だけ定義してあれば使える

↓

これが無名クラス

普通のインタフェースの実装

インタフェースの実装 implements を使って関数を Override する

```
public interface FourLegs {

    /** 前足で蹴る **/
    public void kickWithFourFoot();
}
```

4本足生物クラス

```
public class Rat extends Mammal implements FourLegs {

    /** 前歯の寿命 **/
    public int frontTeethLifeSpan = 3;

    @Override
    public void kickWithFourFoot(){
        System.out.println("ネズミのキック");
    }
}
```

ネズミクラス

4本足生物は「前足で蹴る」生物

↓

継承したクラスでは必ず「前足で蹴る」関数をもっていなければならない（Override）

インタフェースを使う時

インタフェースの実装 implements を使って関数を Override する

```
// 4本足生物の宣言
FourLegs fourLegs;

// ネズミが前足で蹴る
fourLegs = new Rat();
fourLegs.kickWIthFourFoot();

// 馬が前足で蹴る
fourLegs = new Horse();
fourLegs.kickWithFourFoot();

// トカゲが前足で蹴る
fourLegs = new Lizard();
fourLegs.kickWithFourFoot();
```

KickWithFourFoot … 前足で蹴る

new した中身（インスタンス）が何であろうとも
fourLegs.kickWithFourFoot(); 関数を実行することには変わらない

インタフェースは「接続部分」の意味

インタフェースを毎回作るのは大変

手っ取り早く前足で蹴りたいだけの時もある

1. インタフェースを実装したクラスを作る
2. 作ったクラスを new する
3. 実行する

```
class Kick implements FourLegs {
    @Override
    public void kickWithFourFoot(){
        // 前足を上げる処理
        ...
        // 敵に向かって伸ばす処理
        ...
    }
}
```

```
FourLegs kick = new Kick();
```

```
kick.kickWithFourFoot();
```

特に１の作業はファイルも分けたりと面倒くさい。。。

無名（名前が無い）関数なら。。。

前足で蹴るだけならクラス名なんて無くてもいいんじゃない？

1. インタフェースを実装した無名を作って変数に格納
2. 実行する

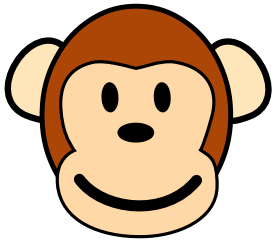
```
FourLegs kick = new FourLegs(){
    @Override
    public void kickWithFourFoot(){
        // 前足を上げる処理
        ...
        // 敵に向かって伸ばす処理
        ...
    }
}
```

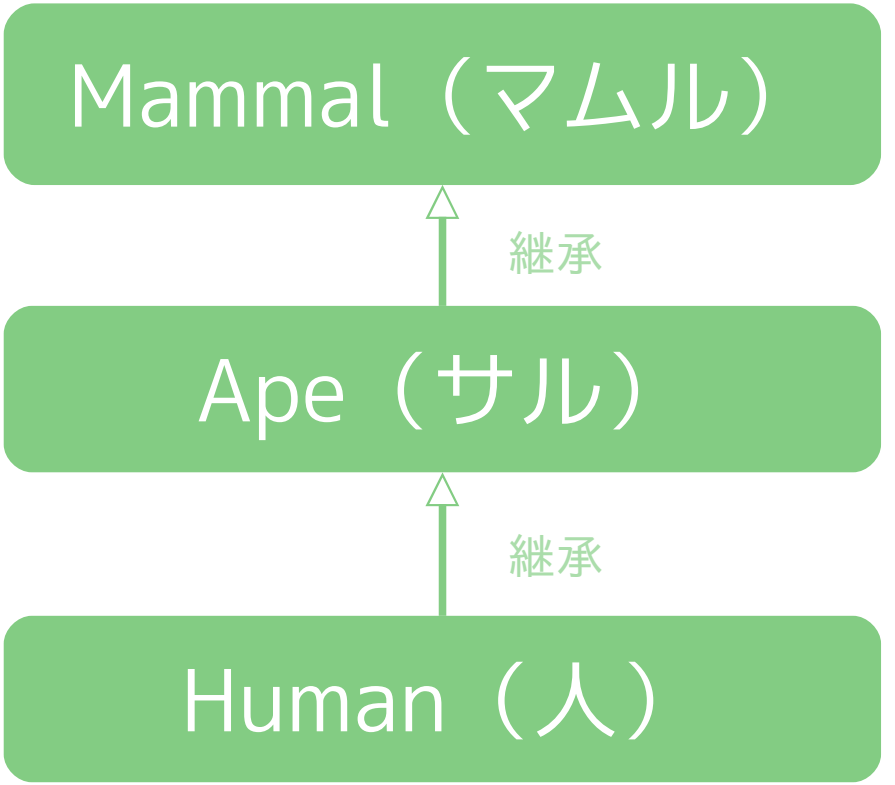
```
kick.kickWithFourFoot();
```

作業が結構減って単純になる

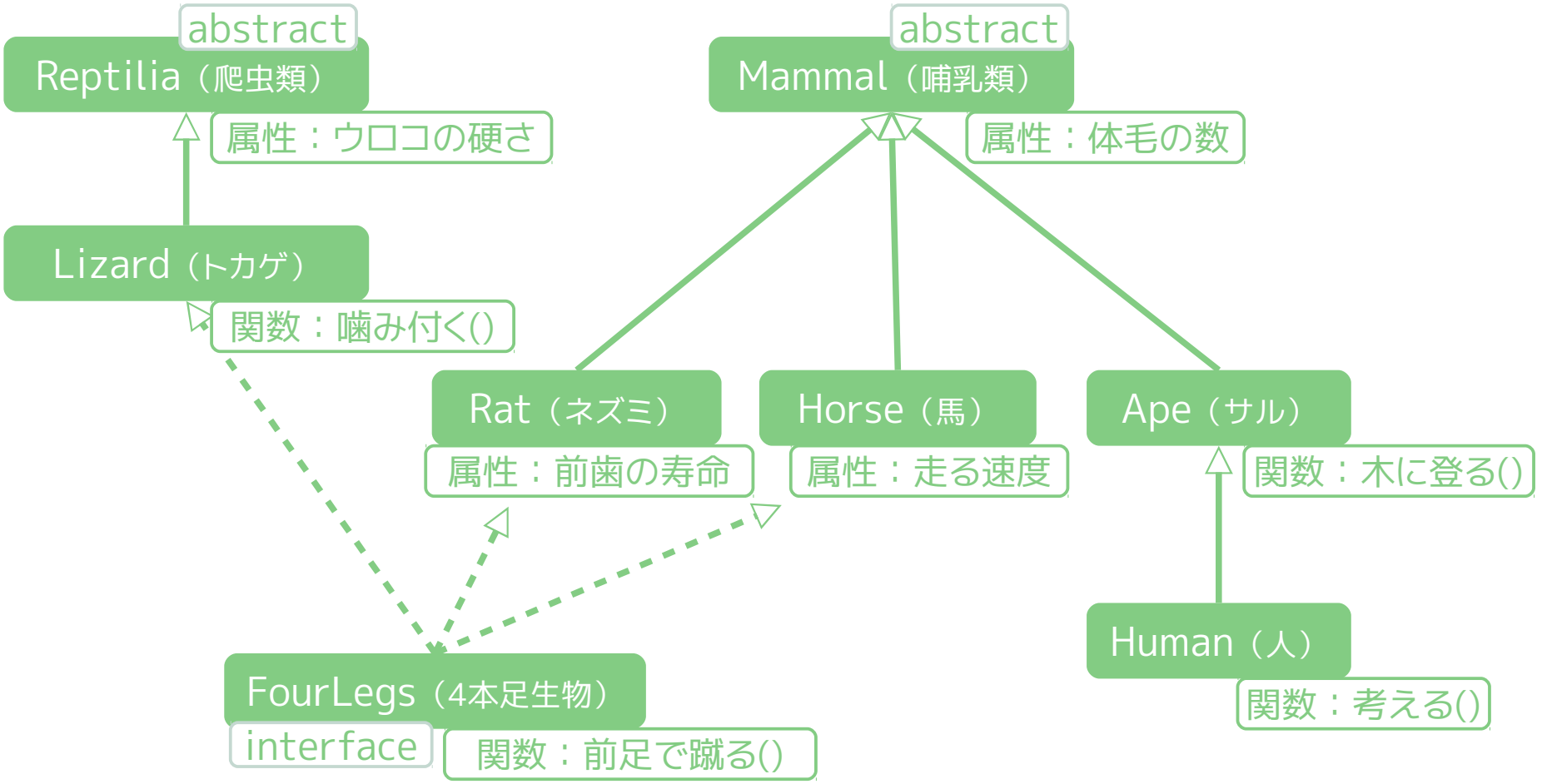
最後にサルは人に進化しました

人はサルを“継承”しています

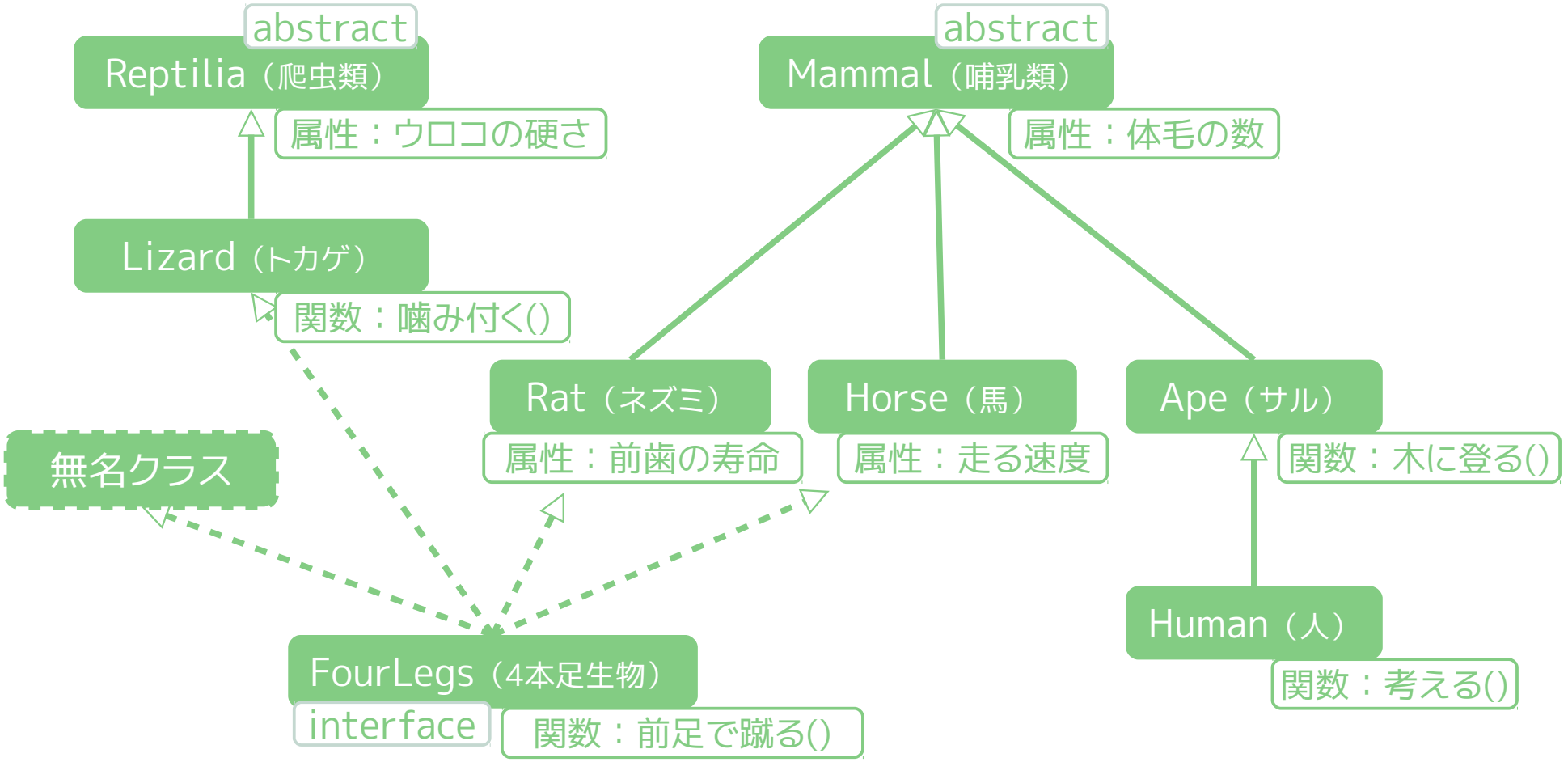




今の時点の継承関係


abstract
Reptilia（爬虫類）
属性：ウロコの硬さ
Lizard（トカゲ）
関数：噛み付く()
abstract
Mammal（哺乳類）
属性：体毛の数
Rat（ネズミ）
属性：前歯の寿命
Horse（馬）
属性：走る速度
Ape（サル）
関数：木に登る()
Human（人）
関数：考える()
FourLegs（4本足生物）
interface
関数：前足で蹴る()

Java の継承を理解する



プログラムにしてみよう

これまでの登場人物を Java で書いてみよう

Reptilia（爬虫類）
ウロコの硬さ
+hardnessOfScale

Mammal（哺乳類）
体毛の数
+bodyHairLength

FourLegs（4本足生物）
前足で蹴る
+kickWithFourFoot()

Lizard（トカゲ）
噛み付く
+bit()

Rat（ネズミ）
前歯の寿命
+frontTeethLifeSpan

Human（人）
考える()
+think()

-（無名クラス）-
前足で蹴る
+kickWithFourFoot()

Horse（馬）
走る速さ
+runSpeed

Ape（サル）
走る速さ
+climbTree()

課題を解いてみよう

※ +記号は public、- は private を意味する

終わり

ありがとうございました