

# 上級プログラミング1(第5回)

---

工学部 情報工学科  
木村昌臣

# 今日のテーマ

---

## □ Javaプログラミング入門(3)

- オブジェクト・クラス
  - メソッド・フィールド
  - コンストラクタ
  - オーバーロード
-

# Javaプログラミング入門(3)

---

# クラスとオブジェクト(復習)

---

- データ構造と処理をまとめたモノ
- クラスとオブジェクトは「型」と「変数」の関係
  - クラスは、オブジェクトを定義
  - オブジェクトは、クラスの内容の「実体」(インスタンス)

クラス

オブジェクト

Person kimura=new Person();

... Personクラスのオブジェクトkimuraを宣言

```
class Person{
    String name; int steps=0;
    Person(){
        name="Default";
    }
    Person(String s){
        name=s;
    }
    void walk(){    steps++;    }
    void walk(int i){    step=step+i;    }
    int getSteps(){ return steps;    }
}
```

# メソッドとフィールド(復習)

---

## □ メソッド≡関数

- オブジェクトに対する働きかけを定義

## □ フィールド≡変数

- オブジェクトの情報を保持

## □ これらをクラスのメンバと呼ぶことがある

## □ オブジェクトが持つ情報(フィールドの値)をメソッドをつかって加工して処理をしていく

- 「.」(ドット)を使ってメソッドやフィールドにアクセスする(アクセス演算子)
-

# プログラムの例

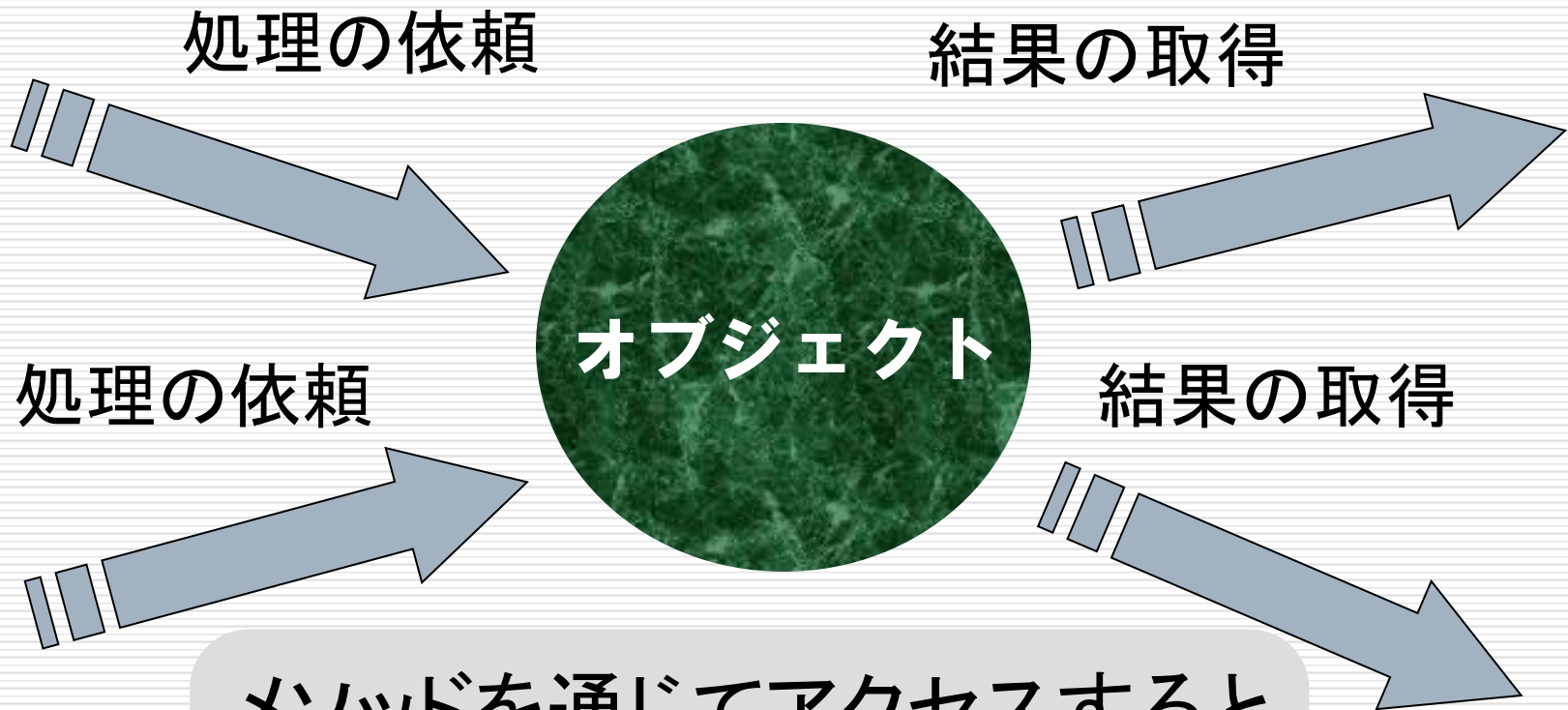
```
class Test{  
    public static void main(String[] args){  
        Person kimura=new Person("Kimura");  
        for(int i=0; i<10000; i++){  
            kimura.walk();  
        }  
        System.out.println(kimura.getSteps()+"歩");  
    }  
}
```

kimuraに  
1歩 歩かせる

Kimuraが歩いた  
歩数を返す

# カプセル化

---



メソッドを通じてアクセスすると  
オブジェクトの内部構造を  
隠すことができる



```
class Person{  
    String name;  
    int steps=0, cal=0;  
    Person(){    name="Default";    }  
    Person(String s){    name=s;    }  
    void walk(){  
        steps++;  
        if(steps/15==0){  
            cal++;  
        }  
    }  
    . . .  
}
```

単に歩数  
だけでなく  
カロリー計算  
を機能追加

# カプセル化のメリット

```
class Test{  
    public static void main(String[] args){  
        Person kimura=new Person("Kimura");  
        for(int i=0; i<10000; i++){  
            kimura.walk();  
        }  
  
        System.out.println(kimura.getSteps()+"歩");  
    }  
}
```

機能追加しても、  
利用する側のプログラムの変更不要

# オーバーロード(1)

---

- 同じクラス内に引数が異なる同一名のメソッドが複数存在しても良い

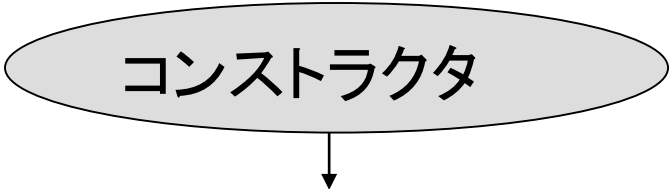
```
class Person{  
    . . .  
    int steps=0;  
    . . .  
    void walk(){  
        steps++;  
    }  
    void walk(int i){  
        step=step+i;  
    }  
    . . .  
}
```

# コンストラクタ

---

- オブジェクトの初期化を行うための処理
- クラス名と同じ名前を持つ
- デフォルトは引数なし
  - クラスにコンストラクタを定義しなければ引数なしのコンストラクタがデフォルトで定義される
  - 定義すれば引数によって処理を変えることも可能

コンストラクタ

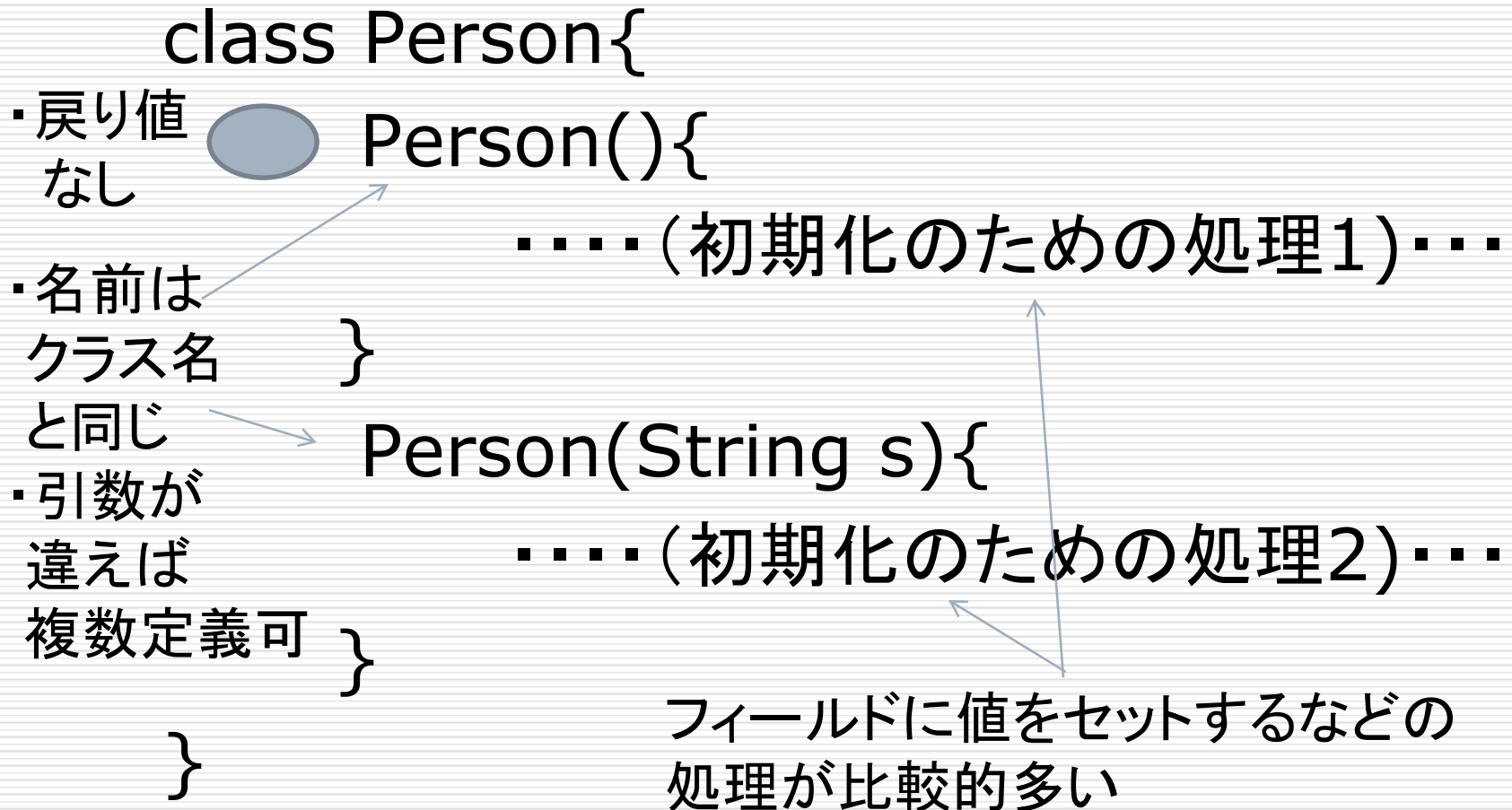


```
graph TD; A([コンストラクタ]) --> B[Person tanaka = new Person(“田中”);];
```

```
Person tanaka = new Person(“田中”);
```

# コンストラクタの定義

---



# オーバーロード(2)

## □ コンストラクタもオーバーロード可能

- コンストラクタは、クラスと同じ名前がつけられる。  
引数に複数の変数のとり方が可能なため、多様な初期化が可能

```
class Person{  
    String name; ...  
    Person(){    name="Default";    }  
    Person(String s){    name=s;    }  
    ...  
}
```

デフォルトは  
Defaultで初期化

名前を指定する場合の  
書記か

# ポリモルフィズム(多態性)

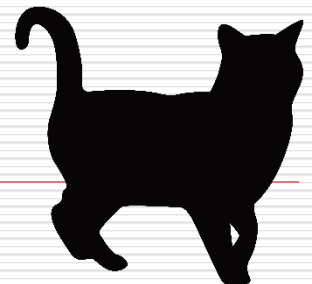
---

- 「歩く」のは人間ばかりではない。猫も歩く。異なるクラスであっても類似の概念による動作をするなら同じ名前を付けることができる

```
class Person{  
    ...  
    void walk(){  
        ...  
    } ...  
}
```



```
class Cat{  
    ...  
    void walk(){  
        ...  
    } ...  
}
```



# 注意！

## オブジェクトと「普通の」変数との違い

---

- 「普通の」変数のことを基本データ型と呼ぶ。
  - 基本データ型の変数への代入は「値渡し」
- クラス型の「変数」のことをオブジェクトと呼ぶ。
  - **オブジェクトへの代入は「参照渡し」**
  - ポインタのようなものと考えてよい（厳密には違う）。

```
int i=1;  
int j=i;  
i=2;
```



jの値は？

```
Person kimura=new Person();  
Person kimura2=kimura;  
Kimura.walk(10);
```



kimura2.stepsの値は？

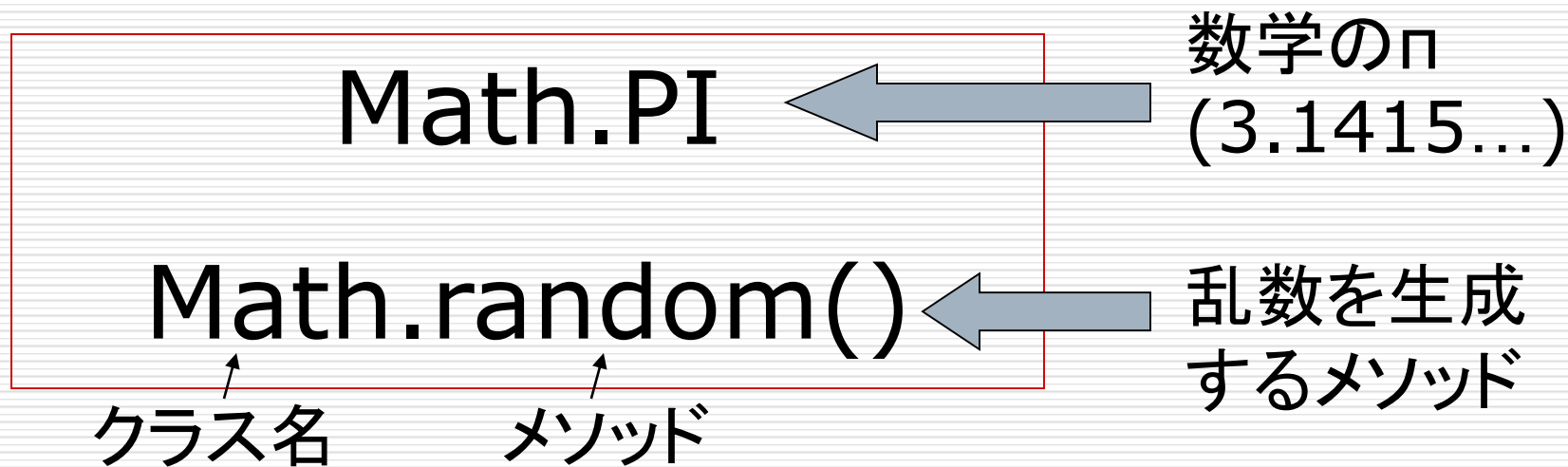


# クラス変数・クラスメソッド

---

## クラスに固有なメンバ(クラス変数・メソッド)

- オブジェクトを生成しないでも、フィールドやメソッドを利用することができる。



どちらもMathという**クラス**に属するメンバであり、オブジェクトを生成せず直接利用できる

実は、もうすでにクラスメンバを使っている

---

```
System.out.println("残高:" + i)
```

- フィールドoutはSystemクラスのクラスフィールドになっている。
  - outは、出力に関するクラス(PrintStream)のオブジェクトであり、printlnメソッドはそのオブジェクトがもつメソッド。
-

# クラス変数・クラスメソッドの作り方

```
class Example0205{
    public static void main(String[] args){
        System.out.println("哺乳類:"+Dog.mamal);
        System.out.println("鳴き声:"+Dog.bark());
    }
}

class Dog{
    static boolean mamal=true;
    static void bark(){
        return "ワンワン！";
    }
}
```

オブジェクトを  
作っていない  
ことに注意

キーワードstaticを使ってるのがミソ

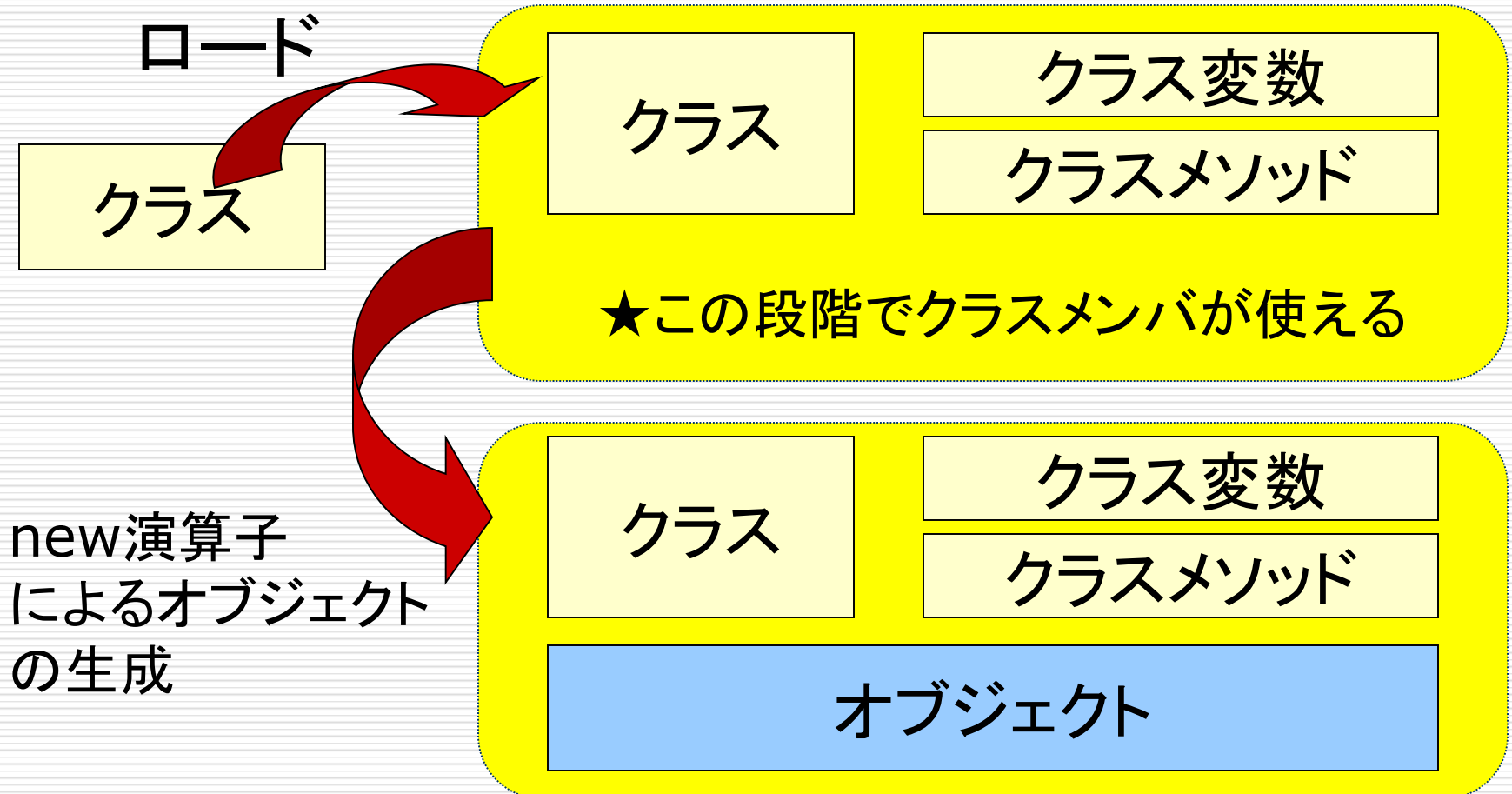
# クラスメンバがあると何がうれしいのか

---

- オブジェクトがなくても情報や処理が利用できる
    - タローでもパトラッシュでも、「犬」であれば哺乳類であり“ワンワン！”と吠える
    - その情報を使いたいときに、特定の犬のオブジェクトを作るのではなく「犬としての情報」を直接利用可能
  - 複数のオブジェクトに共通に使いたい情報を扱うのに便利
    - インスタンスが複数あっても一度しかメモリにロードされないので、メモリの節約になる
-

## 【参考】

# クラスとオブジェクトのロードのタイミング



# 注意

---

- クラス内には、クラス変数・メソッドとインスタンス変数・メソッドは共存できる。
  - ただし、以下はエラーとなる
    - クラス変数の初期化にインスタンス変数・メソッドを利用する。
    - クラスメソッド内にインスタンス変数・メソッドを使う。
    - 理由は、インスタンス変数・メソッドが、クラスメンバに利用されるときにメモリに存在しないため
-

# ラッパークラス

---

- 基本データ型を操作するためのメソッドや付随する情報を提供するフィールドを提供
    - データ型の変換
    - 大文字・小文字の変換 など
  - 例)
    - `int i=Integer.parseInt("-123");`
    - `Integer wi= Integer(10);`  
`float f=wi.floatValue();`
    - `char c=Character.toLowerCase('A');`
-



# ガベージコレクション

---

- C++と違い、デストラクタがない
  - そのかわり、メモリの後始末はJavaが面倒をみってくれる
    - 使用済みでもう使われないメモリ領域を自動的に開放
    - 実行時にCPU時間を多く消費する処理であるため、速度重視のアプリケーションではタイミングに注意
    - `System.gc()` というクラスメソッドを実行することで明示的に実行可能
-