

2.1 Java オブジェクト指向基礎

- オブジェクト指向
- Java オブジェクト指向文法
- クラス変数とクラスメソッド

目次

- 1 **オブジェクト指向**
- 2 **Java オブジェクト指向文法**
- 3 **クラス変数とクラスメソッド**

手続き型プログラミング

- これまで学んできたのは、**手続き型プログラミング** [Procedure Oriented Programming] の書き方です。

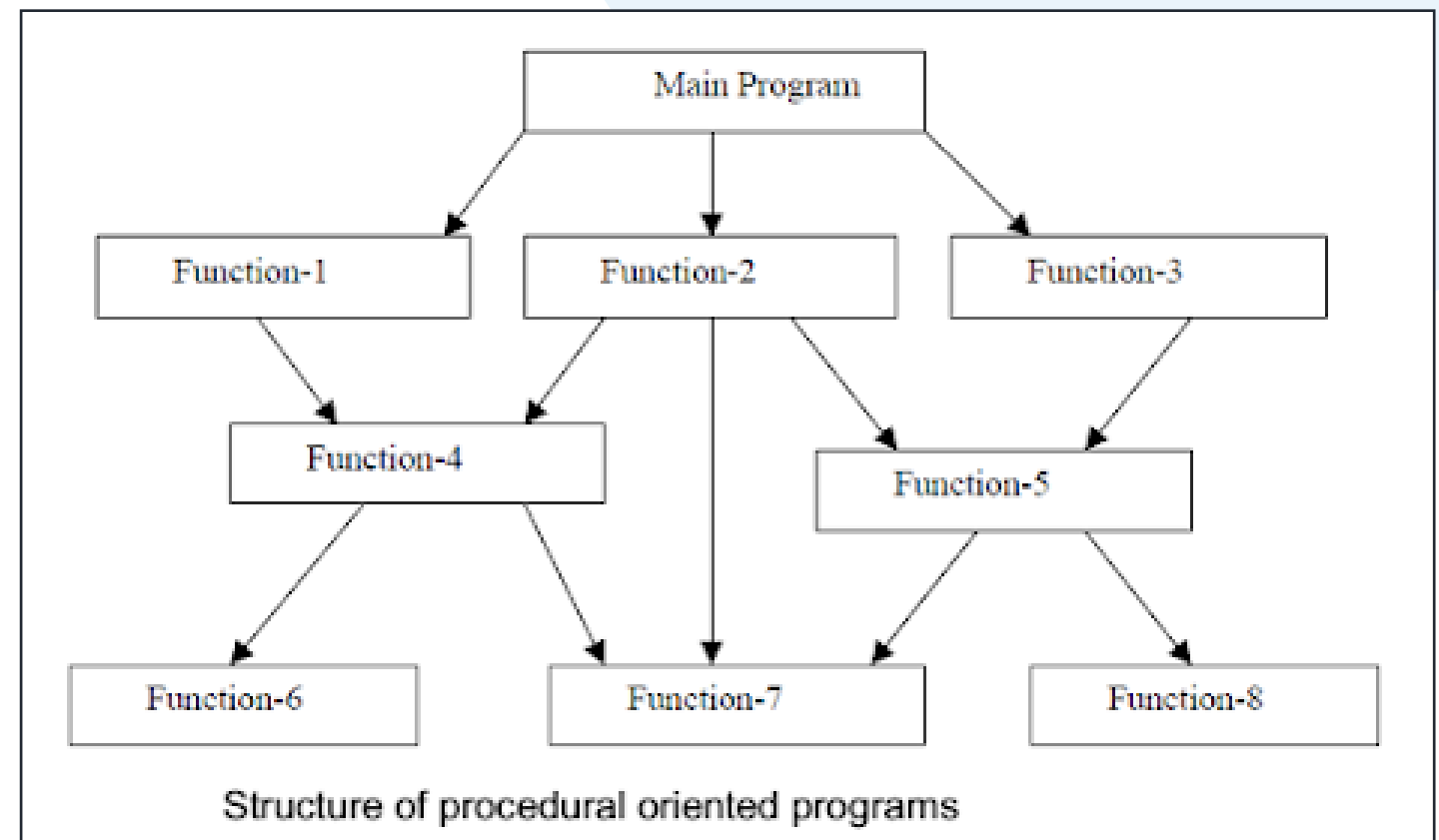
- 色々なメソッドを定義：

```
method1()
method2()
method3()
...
```

- 色々なデータを定義：

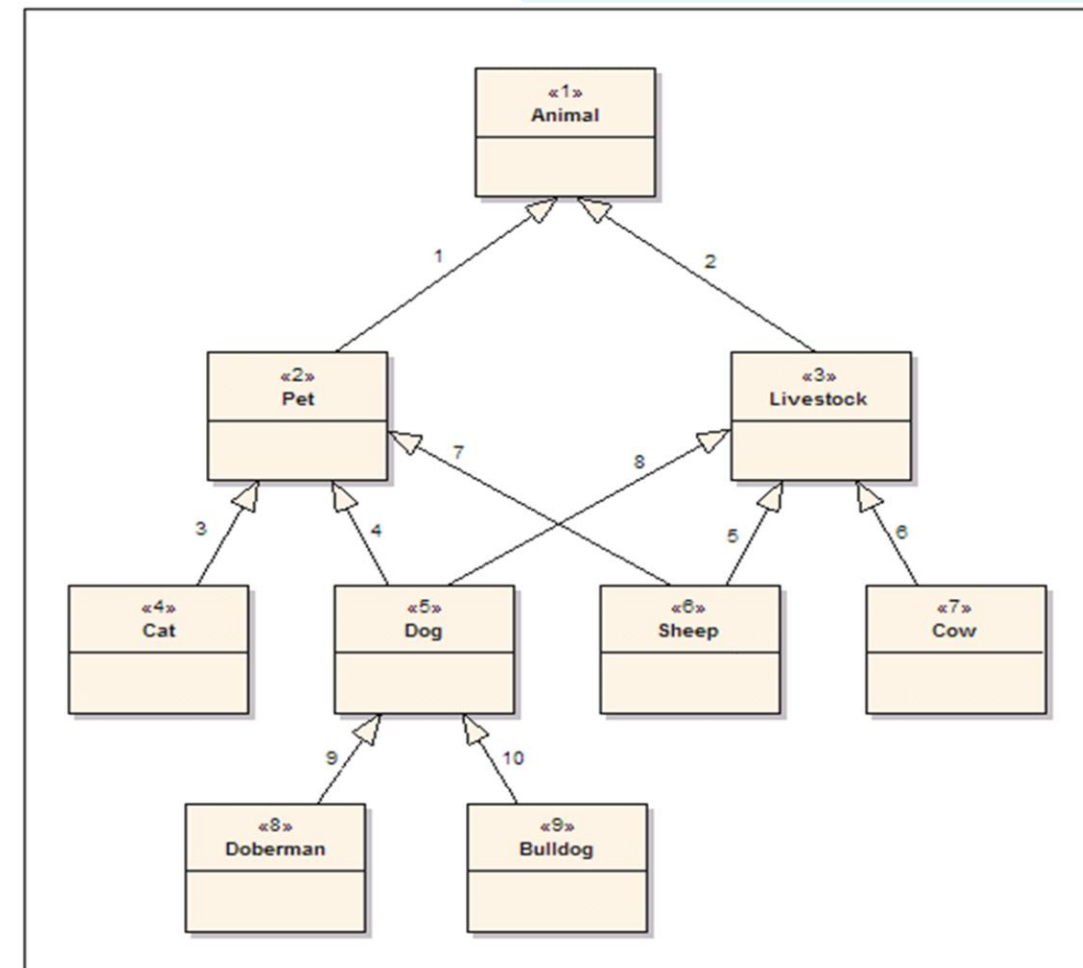
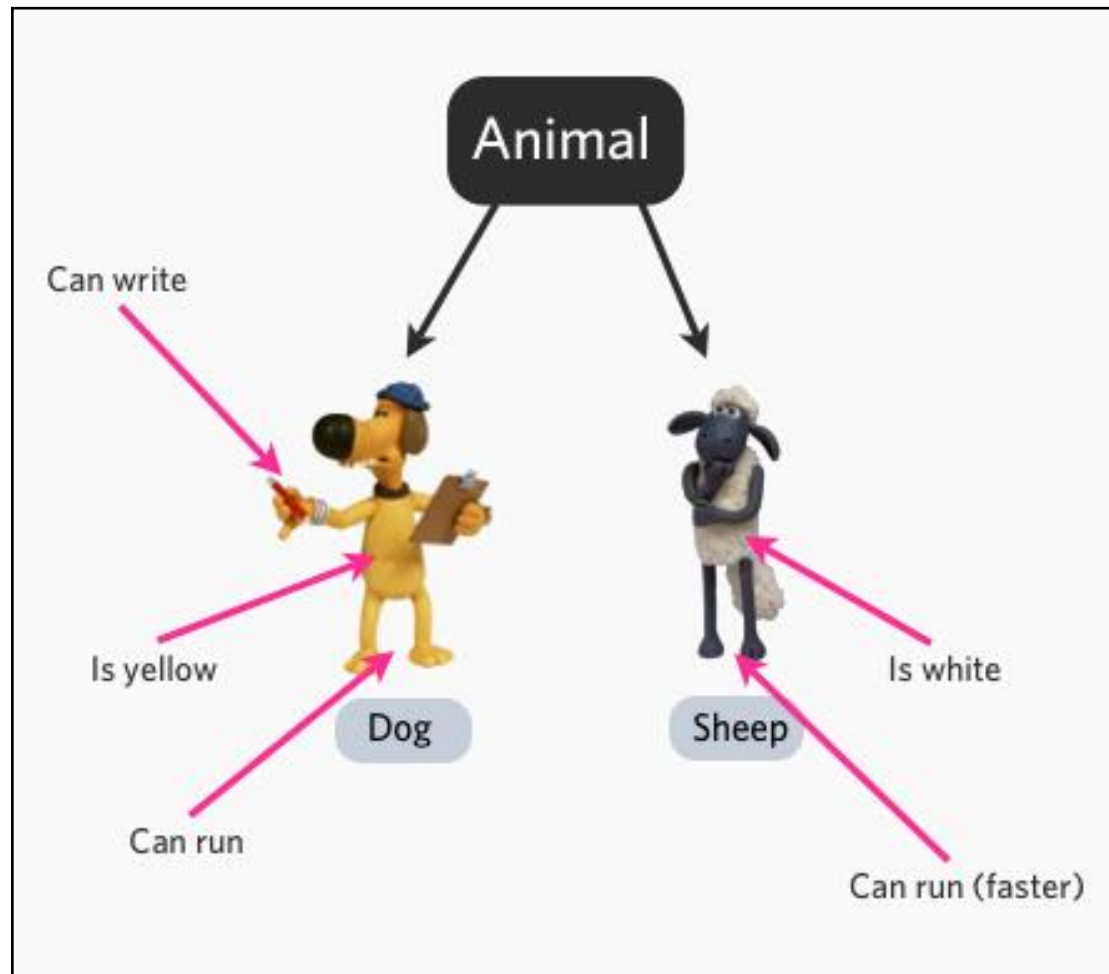
```
int a;
float b;
char c;
String d;
...
```

- メソッド、データ間の操作、演算順番を定義。



オブジェクト指向プログラム

- 今日は、もう一つのプログラミングの書き方を学ぶ：**オブジェクト指向プログラミング** [Object Oriented Programming, **OOP**]。



手続き型 vs オブジェクト指向

- 手続き型のプログラミングではコンピュータでの問題解決の全ての**ステップ**を考え、**一つずつ列挙**してプログラムとして記述してきます。
- オブジェクト指向プログラミングになれば、システムの中にどんな**もの**があってどんな**属性**と**機能**を持っているのか、を考えることから始めます。そして、それらの属性や機能をどのように実現して問題解決に利用するかを決めます。
- 現実の複雑なシステムの問題を解決する際、オブジェクト指向プログラミングを用いると、コードがよりシンプルになり、開発の流れは人間の思考と一致するだけではなく、コード自体の可読性も向上させることができる。

手続き型とオブジェクト指向の流れ

Example



問題：クラス全員の名前と生徒番号を出力したい。

手続き型プログラミング：

1. 全生徒の名前と学籍番号の入った配列を 2 つ作成する。
2. 変数 i を「0」から「生徒数-1」まで変化させ、各配列の i 番目の要素を出力する。

オブジェクト指向プログラミング：

1. 各生徒は名前と学籍番号の属性と、自分の情報を出力する機能を持つ。
2. 全生徒のデータを含む生徒の配列を作成する。
3. 配列の要素を順番にあげ、それらの機能を使って情報を出力します。

オブジェクトの定義

- オブジェクト指向プログラミングでは、コードの処理対象が具体的なデータから**オブジェクト**^[Object]に変わります。オブジェクトは実在する物体や生物でもいいし、理論的な概念でも構いません。

Example



車はオブジェクトである。
人間はオブジェクトである。
正方形はオブジェクトである。
学校はオブジェクトである。

外部と内部のオブジェクト

- オブジェクトは、以下の 2 つの概念を指すことがあります：
 - **外部オブジェクト**：システムに関わる現実の物理的な対象や概念。
 - **内部オブジェクト**：プログラム言語で実装されたコンピュータ上での表現。
- 外部オブジェクトから内部オブジェクトを作るまでの流れ：
 1. 分析ステージ：問題に関わる外部オブジェクトへの認識。
 2. 設計ステージ：内部オブジェクトの表現への変換。
 3. 実現ステージ：内部オブジェクトをプログラムで表現。

次へ 

Example

例えば、動物園を管理するシステムのプログラム作りたい：

1. システムを作るために、動物オブジェクトが必要だとわかった。
2. それぞれの動物には「文字列である名前」「数字である年齢」、その他色々な属性や行動を設計する。
3. 上記の設計をしたがって、実際に Java でコードを書いて実装する。

オブジェクト設計の例

Example



動物園では、多くの猫のオブジェクトが必要です。
猫に必要な特性はどんなものがありますか？



属性と振る舞い

- **属性**または**プロパティ** [Property] : 猫の性質や状態。
 - 性質 : 毛の長さ、毛の色、名前……
 - 状態 : 食事したか、寝ているか、ネズミを捕ったか……
- **振る舞い** [Behavior] : 猫ができる行動や機能。
 - 行動 : 食べる、鳴く、ネズミを捕る……
 - 機能 : 名前を知る、寝ているかを知る……

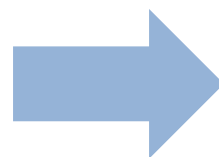
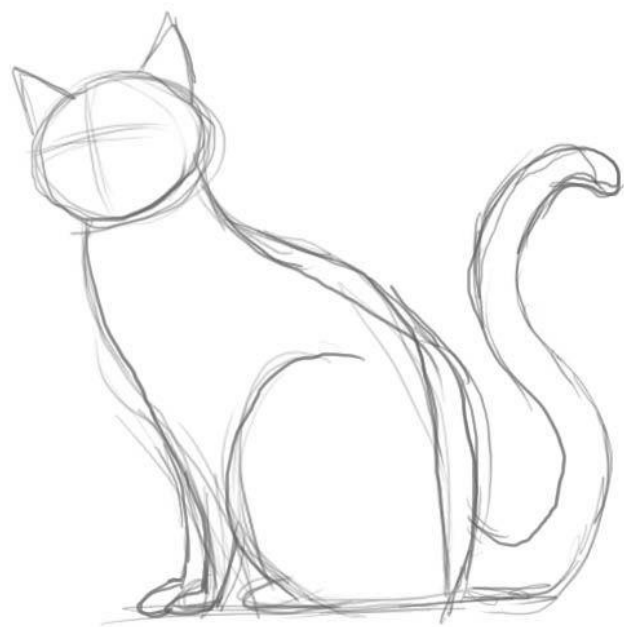


クラス

- これらのオブジェクトはこのような特徴を持っています：属性は常に同じ型であり、振る舞いもまた同じです。異なるのは、**属性の具体的な値**だけです。
- これらの共通の性質をまとめるものは、**クラス**[Class]といます。クラスをテンプレートや、設計図に相当するものだと考えれば結構です。
- クラスがあれば、猫のオブジェクトを作るときは、どのような属性や振る舞いがあるのか、猫クラスから直接判断することができます。あとは、属性に適切な値を設定するだけ。

次へ 

- あるクラスから、属性に適切な値を設定することによって、新しいオブジェクトを作ること、**インスタンス化**
[Instantiation] といいます。作られたオブジェクトは、そのクラスの**インスタンス** [Instance] ともいいます。



インスタンス化



クラスとオブジェクトの例

Example

猫クラス：

猫には、名前（文字列）、年齢（数字）、色（文字列）などの属性があります。

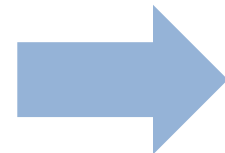
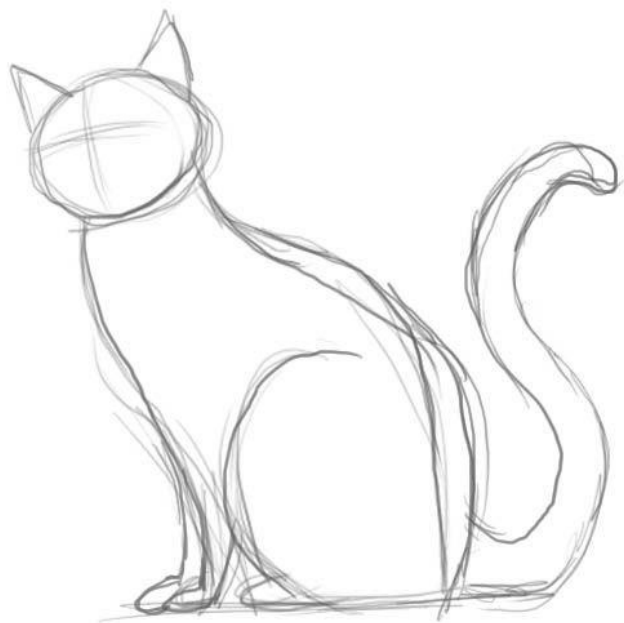
猫には食べる、ネズミを捕るなどの機能（振る舞い）があります。

猫のオブジェ 1：名前は Alice、6 歳、白。

猫のオブジェ 2：名前は Bob、8 歳、青。

猫クラス

猫の持つ属性や振る舞いを定義



インスタンス化

猫のオブジェクト 1



猫のオブジェクト 2



猫のオブジェクト 3



猫のオブジェクト 4



Q&A

目次

- 1 オブジェクト指向
- 2 **Java オブジェクト指向文法**
- 3 クラス変数とクラスメソッド

クラスの作成

- クラスを作成するにはキーワード **class** を使います：

```
1 public class Cat {
2     // 类的定义
3 }
```

- クラスの定義にはクラスの属性と振る舞いが含まれる。
Java では、クラスの属性を**メンバ変数**^[Instance Variable]（または**インスタンス変数**、**フィールド**^[Field]）、振る舞いは**メソッド**^[Method]と呼びます。
- 1 つの Java ファイルに複数のクラスを作成することもできますが、基本的にクラスごと 1 つのファイルを作ります。

メンバ変数の定義

- メンバ変数はローカル変数と同じような文法で定義できます。ただし、メソッド内じゃなくて**直接にクラスの中で**定義することが必要です：

```
1 public class Cat {
2     String name;
3     int age;
4 }
```

- 定義する時に変数を初期化することも可能です：

```
1 public class Cat {
2     String name = "Unkown";
3     int age = 0;
4 }
```

メソッドの定義

- メソッドについてはすでに学びましたが、一般的なクラスメソッドの定義は前とは若干異なります：

```
1 class Cat {  
2     void meow() {  
3         System.out.println("meow~");  
4     }  
5 }
```

- 実は、メソッド定義は名前の前に**戻り値の型だけ**を書いていいです。以前使った `static` キーワードは、後ほど説明します。

オブジェクトの作成

- Cat というクラスを作成し、それを使ってオブジェクトを作成すること（インスタンス化）ができます。
- オブジェクトを作るにはキーワード **new** を使います：

```
new Cat();
```

- 作成されたオブジェクトは変数に格納することができます。この変数は、Cat のオブジェクトを格納する必要があるため、型を「Cat」にしました：

```
Cat alice = new Cat();
```

オブジェクトの使用

- オブジェクトができたなら、そのオブジェクトのメンバ変数とメソッドが使えます。
- オブジェクトのメンバ変数やメソッドを使用するには、「.」 演算子を使用します：

```
1 Cat alice = new Cat();  
2 System.out.println(alice.name); // 使用成员变量的值  
3 alice.age = 5; // 改变成员变量的值  
4 alice.eat("catfood"); // 使用方法
```

本クラスのメンバ変数とメソッドの使用

- クラスのメソッドは、このクラスのメンバ変数とメソッドを直接使用できます：

```

1 class Cat {
2     // ...
3
4     void eat(String food) {
5         System.out.println(name + " eat " + food);
6         meow( ); // 发出猫叫
7     }
8 }

```

次へ 

- 実際の実行時にはそのオブジェクト自身のメンバ変数とメソッドが使用されます：

```
1 Cat alice = new Cat();  
2 alice.name = "Alice";  
3 alice.eat("cat food"); // => Alice eat cat food, meow~
```

Note

メンバ変数やメソッドは、それらを定義するコードの前に使用することができます。つまり、コードの書く順番は関係ありません。

コンストラクタ

- **コンストラクタ** [Constructor] とは、オブジェクトが生成されるときに、自動的に呼び出されるメソッドです。オブジェクトを生成するたびに実行する必要がある**初期化操作**があれば、それらをコンストラクタに記述しましょう。
- コンストラクタを定義するためには、クラス名と同じ名前のメソッドを定義し、その戻り値を定義**しません**：

```
1 class Cat {  
2     // ...  
3  
4     Cat() {  
5         name = "Default Name";  
6         age = 1;  
7     }  
8 }
```

コンストラクタの引数

- コンストラクタは任意の数の引数を持つことができます。いくつかの引数を受け取って、メンバ変数の初期値を設定することができます：

```
1 Cat(String name_, int age_) {  
2     name = name_;  
3     age = age_;  
4 }
```

- オブジェクトを作成する際には、対応する引数を入れることが必要です：

```
1 Cat alice = new Cat("Alice", 5);  
2 System.out.println(alice.name); // => Alice  
3 System.out.println(alice.age); // => 5
```

- コンストラクタをオーバーロードすることもできます。

デフォルトコンストラクタ

- コンストラクタを定義しない場合、Java は引数なしの**デフォルトコンストラクタ** [Default Constructor] を自動的に定義します。
- ただし、コンストラクタを手動で定義したら、この**デフォルトコンストラクタ**がなくなります！

```
1 Cat alice = new Cat("Alice", 5); // Cat 类定义了一个有参数的构造器
2 Cat bob = new Cat(); // 语法错误: Cat 没有无参构造器
```



Q&A

目次

- 1 オブジェクト指向
- 2 Java オブジェクト指向文法
- 3 クラス変数とクラスメソッド

クラス変数とクラスメソッド

- メソッド定義する際に使った static キーワードを覚えていますか？「static」は「**静的**」のことです。Java では、変数もメソッドも静的にすることが可能です。静的とは、変数やメソッドが特定のオブジェクトではなく、クラスに属します。したがって、これらの変数とメソッドは、**クラス変数** [Class Variable] と **クラスメソッド** [Class Method] といいます。
- これらを定義するためには、static キーワードを使います：

```
1 class A {  
2     static int staticVariable = 10;  
3     static int staticMethod(int a) {  
4         return a + staticVariable;  
5     }  
6 }
```

- クラス変数やクラスメソッドを使用する時も普通の変数やメソッドと同じように、「.」演算子を使用できます。ただし、「.」の前に書かれるのは**クラス名**であります：

```
1 System.out.println(A.staticVariable); // => 10
2 A.staticVariable = 5;
3 System.out.println(A.staticMethod(10)); // => 15
```

- 同じクラスのメソッドではそのクラスのクラスメソッドやクラス変数を直接使用することができます：

```
1 class A {  
2     static int staticVariable = 10;  
3     static int staticMethod(int a) {  
4         return a + staticVariable;  
5     }  
6  
7     void method() {  
8         System.out.println(staticVariable);  
9         staticVariable = staticMethod(10);  
10    }  
11 }
```



- そのクラスの任意のオブジェクトに「.」を使ってクラス変数やクラスメソッドを呼び出すこともできますが、推奨されません：

```
1 A a = new A();  
2 a.staticVariable = 5;  
3 System.out.println(a.staticMethod(10)); // => 15
```


クラスメソッドが使えない場合

- クラスメソッドは特定のオブジェクトに所属しないので、インスタンスメソッドやメンバ変数は、クラスメソッドの内から直接呼び出すことは**できません**：

```

1 class A {
2     void method1() { }
3     int a = 0;
4
5     static void method2() {
6         method1(); // => 报错
7         System.out.println(a); // => 报错
8     }
9 }

```

Try  Student.java

- さて、以前メソッドを定義したとき、static を書いた理由はわかりましたか？

staticの応用例ーインスタンスの個数を数える

```
public class Cat {
    int counter = 0;

    Cat() {
        counter++;
        System.out.println(counter);
    }
}
```

```
public class Main {

    public static void main(String[] args) {

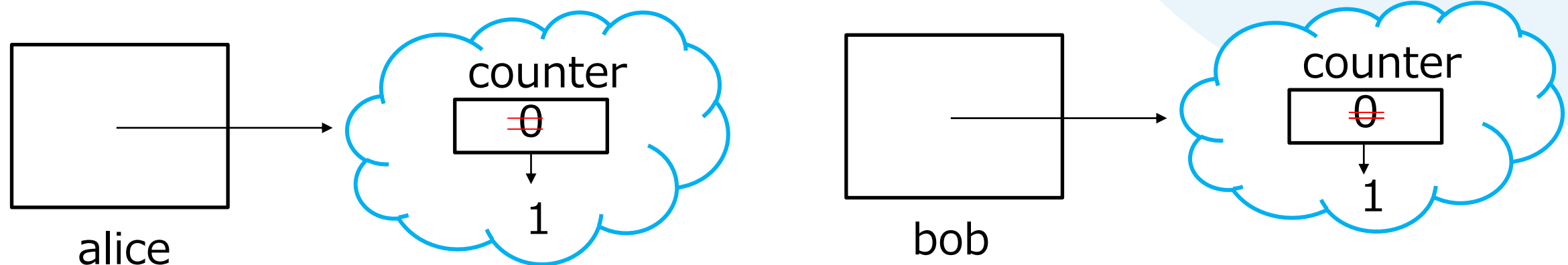
        Cat kitty = new Cat();
        Cat bob = new Cat();
        Cat alice = new Cat();

    }
}
```

- 考えてみよう：
実行結果はどうなるかを予想してみましょう。

次へ 

- 結果は、1, 1, 1 と出力されます。
- 理由は、counterがインスタンスごとに独立した変数であるためです。



- カウントするためには、インスタンスの中の変数だと難しいため、それぞれのインスタンスが共通して使える変数があれば良いと考えます。

static変数とstaticメソッド

- 全インスタンス変数が見えるメンバ変数やメソッドを定義するには、**static (静的)** を指定する。
- 使用する際は、「**クラス名.変数[メソッド] 名**」と記述する

```
public class Cat {
    static int counter = 0;
    String name;
    Cat(){
        counter ++;
    }
}
```

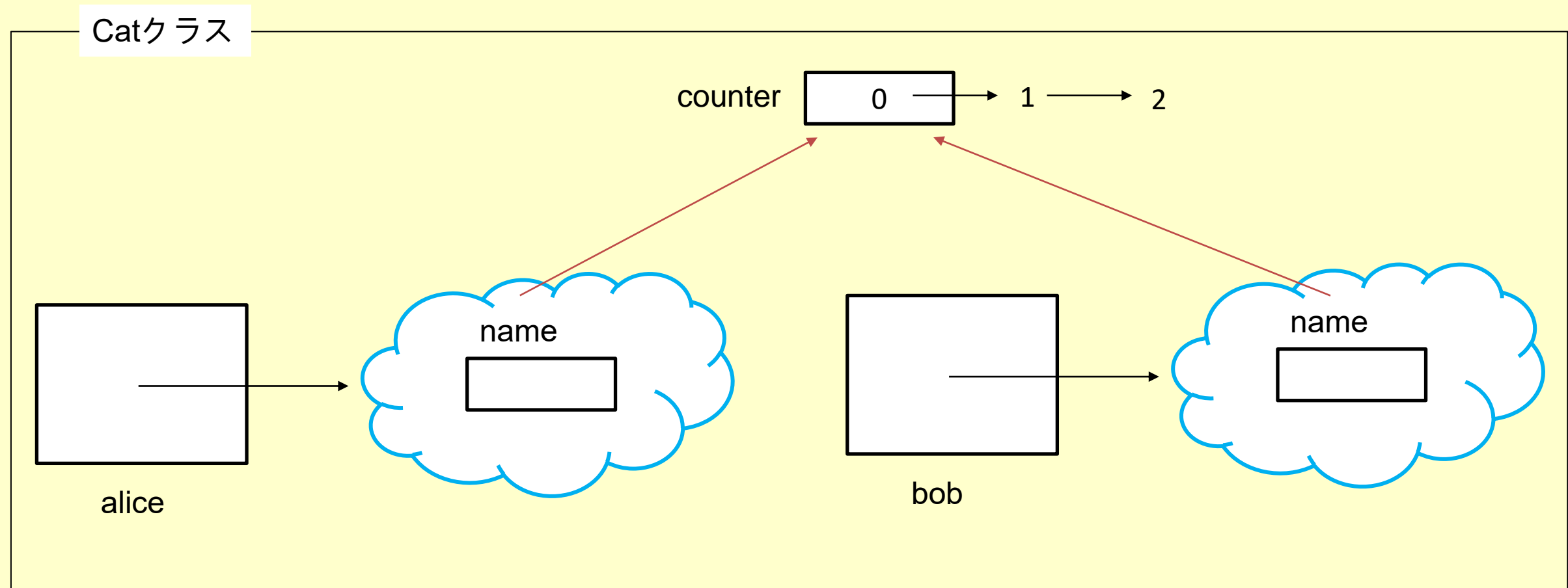
```
public class Main {

    public static void main(String[] args) {
        Cat kitty = new Cat();
        Cat bob = new Cat();
        Cat alice = new Cat();

        System.out.println(Cat.counter);
        //=>3
    }
}
```

前へ

以下がcounterを実行したときのイメージになります。
共通して使える変数を作成したことで、カウントすることができました。



次へ

● オブジェクトを生成しなくても利用できる

staticな変数・メソッドは、インスタンスを作成しなくても使用することができます。

```
public class Cat {  
    static int counter = 0;  
    String name;  
    Cat() {  
        counter ++;  
    }  
}
```

```
public class Main {  
  
    public static void main(String[] args) {  
        System.out.println(Cat.counter);  
        //=>0  
    }  
}
```

- インスタンスメソッドからstaticな変数（メソッド）を参照することはできますが、staticなメソッドからインスタンス変数（メソッド）を参照することはできません。
- 理由は、static視点から考えたときに、インスタンス変数（メソッド）は、そもそもインスタンスが作成されないと呼び出すことができません。そのため、必ず存在するかの保証がないものを参照することはできないため、エラーとなります。


```
public class Cat {
    static int counter = 0;
    String name;
    Cat(){
        counter ++;
    }

    static void display() {
        System.out.println(counter);
    }

    static void eat() {
        display();
        meow();
    }

    void meow() {
        display();
        System.out.println(counter);
    }
}
```

**staticから非staticを
呼び出すことはできない**

```
public class Main {

    public static void main(String[] args) {
        Cat kitty = new Cat();
        Cat bob = new Cat();
        Cat alice = new Cat();
        Cat.eat();
        kitty.eat();
    }
}
```

インスタンス変数からstatic
な変数やメソッドを呼び出す
ことはできるが推奨されない

↓
インスタンス変数視点で考え
ると、staticは各インスタン
スが共通して使えるもので、
自分の一部でもあるから呼び
出すことができる

→ 型 Cat の非 static メソッド meow() を static 参照することはできません

まとめ：メソッドの呼び出し

Sum Up

クラスメソッドと関連する呼び出しの動作を以下のようにまとめることができます：

メソッドの種類	インスタンスメソッド・メンバ変数を使う	クラスメソッド・クラス変数を使う
インスタンスメソッド	可能	可能
クラスメソッド	不可能	可能

Q&A

まとめ

Sum Up



1. オブジェクト指向の基本概念。
2. Java でのクラスとオブジェクト：
 - ① クラス、メンバ変数、メソッドの定義。
 - ② コンストラクタの定義とオブジェクトの生成。
 - ③ メンバ変数やメソッドの使用。
 - ④ クラス変数とクラスメソッドの定義と使用。



Light in Your Career.
THANK YOU!