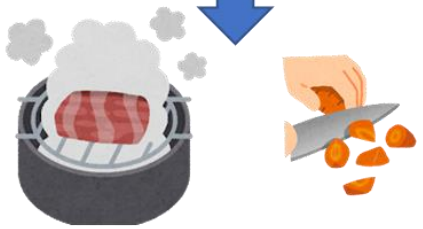
手続き型プログラミングはページの上から順番にプログラムを実行する

オブジェクト指向は色々なファイルから役割をもったもの（オブジェクト）を呼び出して実行していく。

手続き型プログラミング

実行者は一人

実行者が材料を集めて



実行者が料理する

実行者がお客様に運ぶ

オブジェクト指向型

実行者が多数、管理者一人

管理者は命令するだけ

管理者が材料を注文する

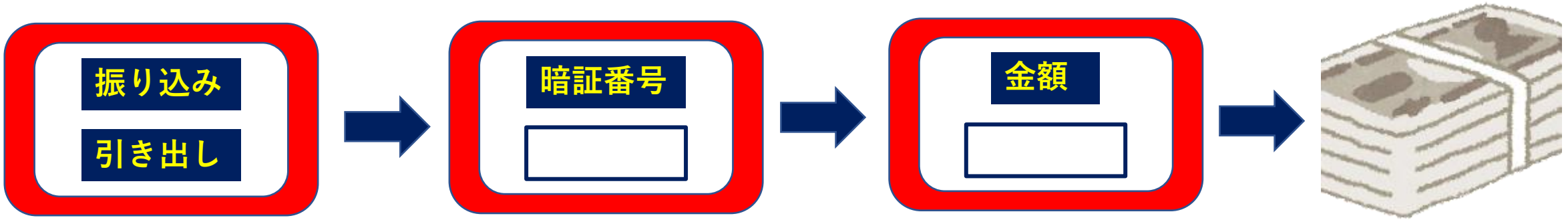
シェフに調理してもらう

ウェイターに料理を運んでもらう

お客様に料理が届く

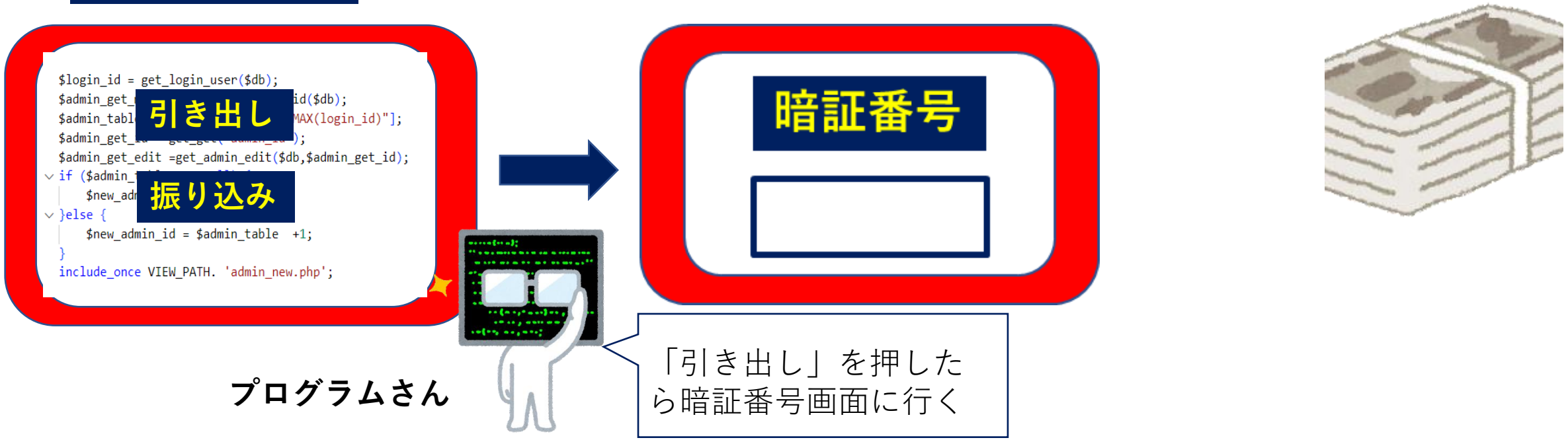
レストランを例にすると

外部オブジェクト　現実に存在するシステム（イメージ）



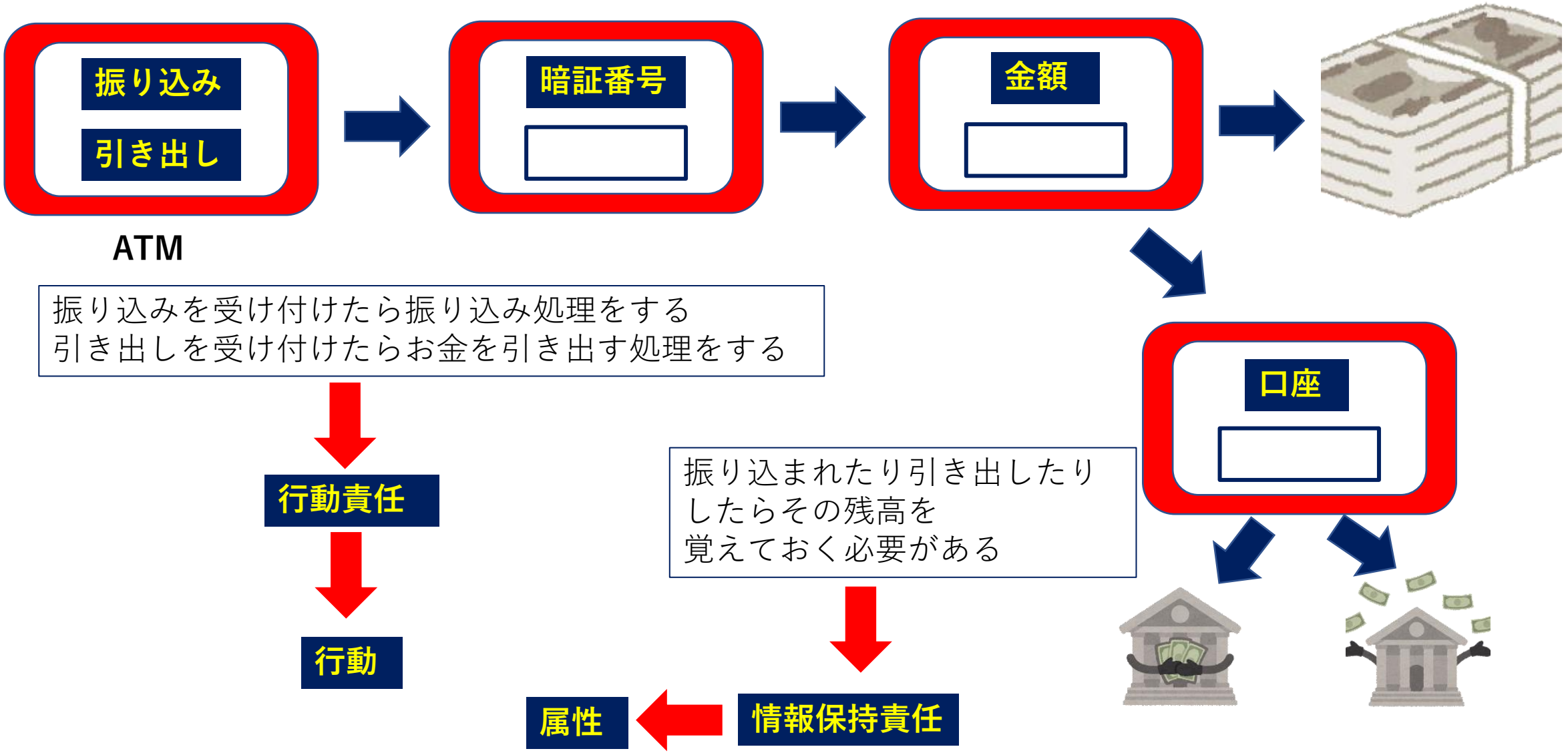
ATM

内部オブジェクト　システムを動かすための裏方役（イメージ）



プログラムさん

行動責任と情報保持責任

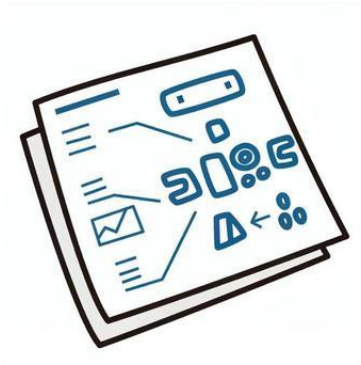


クラスとは！？

プログラムを作る「設計図」のこと！

猫のクラス

猫の設計書



スコティッシュフォールド

メインクーン

インスタンスとは！？

クラスを基に作られた「プログラム」のこと！

猫のクラス

猫の設計書

クラス

インスタンス化

インスタンス化

スコティッシュフォールド

インスタンス

メインクーン

インスタンス

オブジェクトを生み出すまで

クラス

・プロパティ（属性）
・メソッド（ふるまい）

インスタンス化

インスタンス

インスタンス

クラスの書き方

```
class 設計書の名前 {
```
クラスの開始

属性の設定

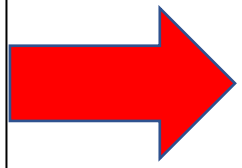
```
データ型　変数名;
```
メンバ変数

フィールド

メソッドの設定

```
void メソッド名() {
    処理内容を記載
}
```

```
}
```
クラスの終わり



```
class Cat {

    String name;
    int age;
    String color;

/*食べる*/

 void eat(String food) {
 System.out.print(name + " は " + food + "を食べます ");
  meow();
}
    /**
     * 鳴く
     */
    void meow() {
        System.out.println("meow~");
    }

}
```

定義されたものよりも
前に使用できる

インスタンス化

クラス（設計書）

```
class 設計書の名前 {   クラスの開始

属性の設定

データ型　変数名;      フィールド
メンバ変数

メソッドの設定

void メソッド名() {
    処理内容を記載
}

}   クラスの終わり
```

インスタンス化

```
public class Main {

public static void main(String[] args) {

  インスタンスの生成
  クラス名 変数名= new クラス名();

  フィールドへの値の代入
  変数名.フィールドの変数名＝値;

  インスタンス内容を表示
  System.out.print(変数名.フィールドの変数名);


  変数名. メソッド名(引数);
  }
}
```

```
public class Main {

public static void main(String[] args) {
Cat alice = new Cat();
alice.name= “アリス” ;
alice.age = 5;
alice.color =”茶色”;
System.out.println(alice.name);
System.out.println(alice.age);
System.out.println(alice.color);
alice.eat(”キャットフード”);
}
}
```

プログラムの実行

Cat.java

```
class Cat {

   String name;
    int age;
    String color;


/*食べる*/

 void eat(String food) {
 System.out.print(name + " は " + food + "を食べます ");
  meow();
}
  /* 鳴く*/
    void meow() {
        System.out.println("meow~");
    }

}
```

設計図を基に
インスタンス化

インスタンスを
作成

```
public class Main {

public static void main(String[] args) {
Cat alice = new Cat();
alice.name =″アリス″;
alice.age = 5;
alice.color =″茶色″;

System.out.println(alice.name);
System.out.println(alice.age);
System.out.println(alice.color);
alice.eat(″キャットフード″);}

}
```

実行結果

```
アリス
5
茶色
アリス は キャットフードを食べます meow~
```

名前(name):アリス
年齢(age):5
色（color):茶色

クラス（設計書）

```
class 設計書の名前 {
```
クラスの開始

Cat

属性の設定

```
データ型　変数名;
メンバ変数
```
フィールド

```
String name;
```

メソッドの設定

```
void メソッド名() {
    処理内容を記載
}
/*食べる*/

void eat(String food) {
System.out.print(name + " は " + food + "を食べます ");
 meow();
}
}
```
クラスの終わり

オブジェクトを作成

```
public class Main {

public static void main(String[] args) {
```

インスタンスの生成

```
クラス名 変数名= new クラス名();  Cat alice = new Cat();
```

フィールドへの値の代入

```
変数名.フィールドの変数名＝値;      alice.name=“アリス”;
```

インスタンス内容を表示

```
System.out.print(変数名.フィールドの変数名);
System.out.println(alice.name);
変数名. メソッド名(引数);  alice.eat(″キャットフード″);
}
}
```

プログラムの実行

```
class Cat {

    String name;
    int age;
    String color;


    /**
     * 鳴く
     */
    void meow() {
        System.out.println("meow~");
        →「meow~」を表示させる
    }

}
```

設計図を基に
インスタンス化

インスタンスを
作成

```
public class Main {

public static void main(String[] args) {

Cat alice = new Cat();
System.out.println(alice.name); ①
System.out.println(alice.age); ②
System.out.println(alice.color); ③
alice.meow();④
}

}
```

実行結果

```
null ①
0    ②
null ③
meow~ ④
```

どういうこと！？

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

New Cat()

String name
int age
String color

・name:null
・age:0
・color:null

作成したてのCatクラスは、名前も色もない
「実際の猫は、作成した瞬間に色等が存在しない」のは考えずらい

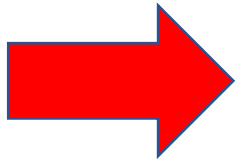
作った際には、色や名前が決まっていることがほとんど

Javaに置き換えると、
インスタンス化と同時にcolorの等の値が決まればOK!

コンストラクタ　　インスタンス化の時に初期値を決める特別なメソッド

コンストラクタの書き方

```
クラス名(引数) {
    初期化の内容
}
```



```
Cat() {
    this.color = red;
}
```

コンストラクタのポイント

- 戻り値は記載しない　→　記載しちゃうと普通のメソッドと変わらない
- メソッド名はクラス名と同じものにする

※　thisは「自分自身の」という意味

コンストラクタ追加プログラムの実行

```
class Cat {

  String name;
  int age;
  String color;


  /*コンストラクタ*/
    Cat() {

        this.color = “red”;
    }
}
```

設計図を基に
インスタンス化

インスタンスを
作成

```
public class Main {

public static void main(String[] args) {

Cat alice = new Cat();
Cat bob = new Cat();
System.out.println(alice.color);    ①
System.out.println(bob.color);

}

}
```

実行結果

```
red
red
```

①aliceの色(color)

② bobの色(color)

コンストラクタで
値を代入すると、
この先、猫を作った時に同じ体重になっちゃう

色（color):赤

色（color):赤

コンストラクタに引数を追加

```
class Cat {

   String name;
    int age;
    int height;


    /*コンストラクタ*/
Cat(String name, int age,String color) {
        this.name = name;
        this.age = age;
        this.color = color;
    }
}
```

設計図を基にインスタンス化

インスタンスを作成

```
public class Main {

public static void main(String[] args) {

Cat alice = new Cat( “Alice” , 5,” 茶色” );
Cat bob = new Cat( “Bob” , 6,” 紺色” );
System.out.println(alice.name);
System.out.println(alice.age);
System.out.println(alice.color);
System.out.println(bob.name);
System.out.println(bob.age);
System.out.println(bob.color);
}

}
```

実行結果

```
Alice
5
茶色
Bob
6
紺色
```

①aliceの名前(name)
②aliceの年齢(age)
③aliceの色(color)
④bobの名前(name)
⑤bobの年齢(age)
⑥ bobの色(color)

引数を使うことで、
違う情報を持った猫ちゃんができた！

名前(name):alice
年齢(age):5
色 (color):茶色

名前(name):bob
年齢(age):6
色(color):紺色

猫ちゃんに餌をあげよう！

```
class Cat {

   String name;
    int age;
    int color;

    /*コンストラクタ*/

 Cat(String name_, int age_,String color_)
      {
       name = name_;
       age = age_;
       color = color_;
    }

/* 食べる*/

 void eat(String food) {
 System.out.print(name + " は " + food + "
を食べます ");
       meow();
}

/* 鳴く*/
     void meow() {
       System.out.println("meow~");
    }
}
```

設計図を基に
インスタンス化

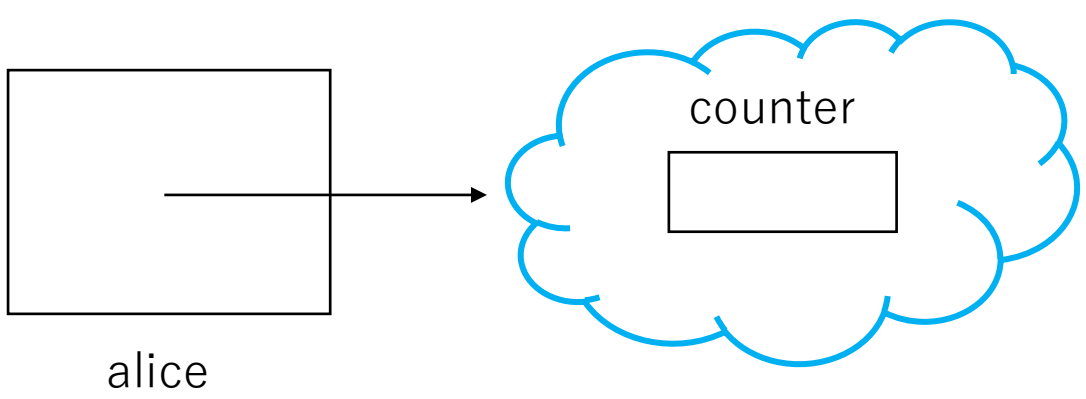
インスタンスを作成

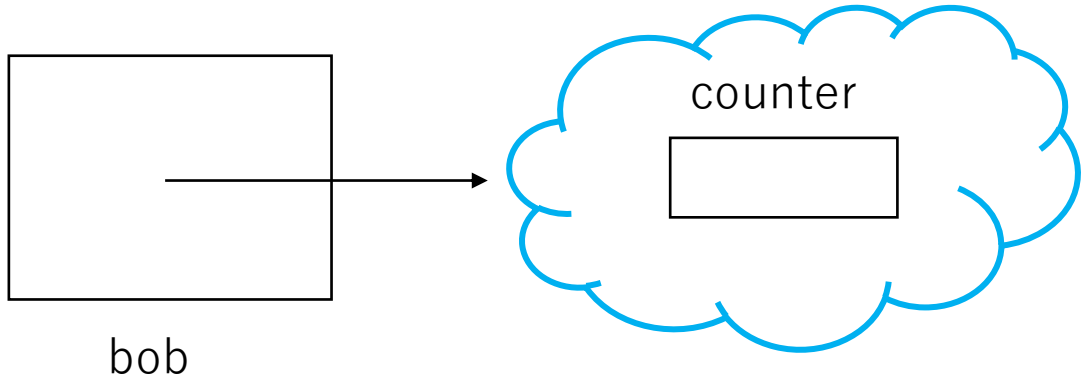
```
public class Main {

public static void main(String[] args) {

Cat alice = new Cat( “Alice” , 5,” 茶色” );
Cat bob = new Cat( “Bob” , 6,” 紺色” );
Cat alice = new Cat("Alice", 5,"茶色");
Cat bob = new Cat("Bob", 6,"紺色");
System.out.println(alice.name);
System.out.println(alice.age);
System.out.println(alice.color);
System.out.println(bob.name);
System.out.println(bob.age);
System.out.println(bob.color);
alice.eat("ネズミ");
bob.eat("キャットフード"); }

}
```

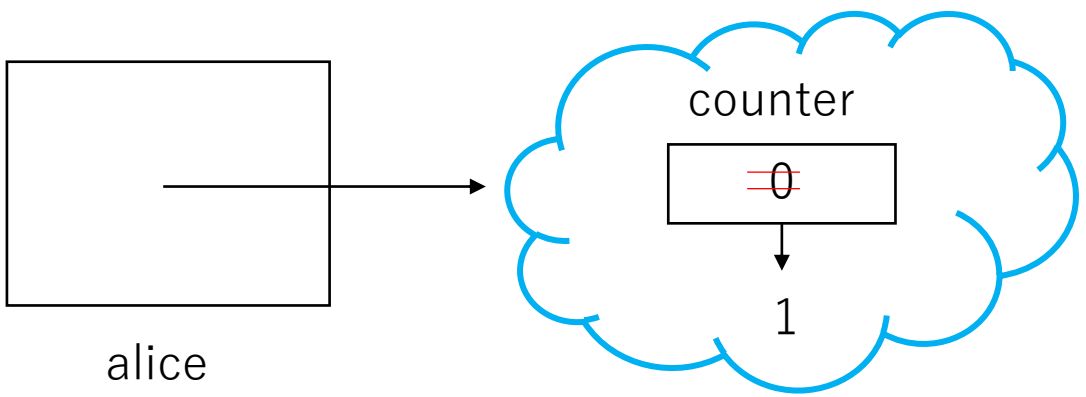
実行結果

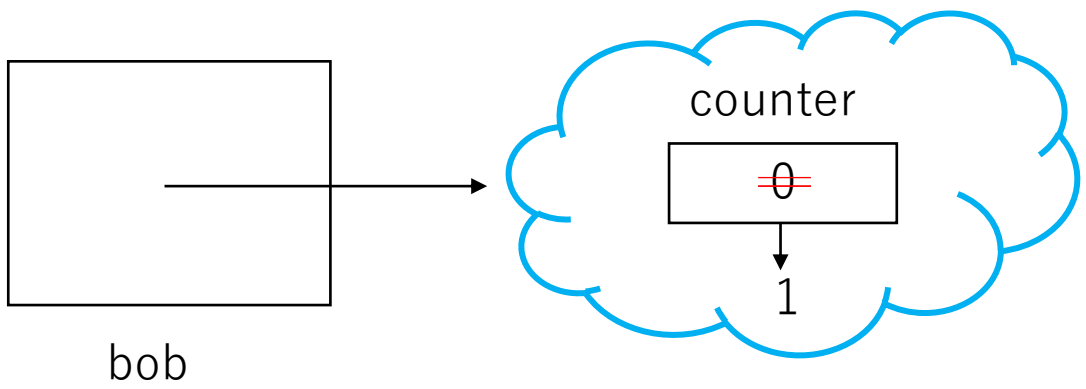
```
Alice
5
茶色
Bob
6
紺色
Alice は ネズミを食べます meow~
Bob は キャットフードを食べます meow~
```

インスタンスの個数を数えてみよう

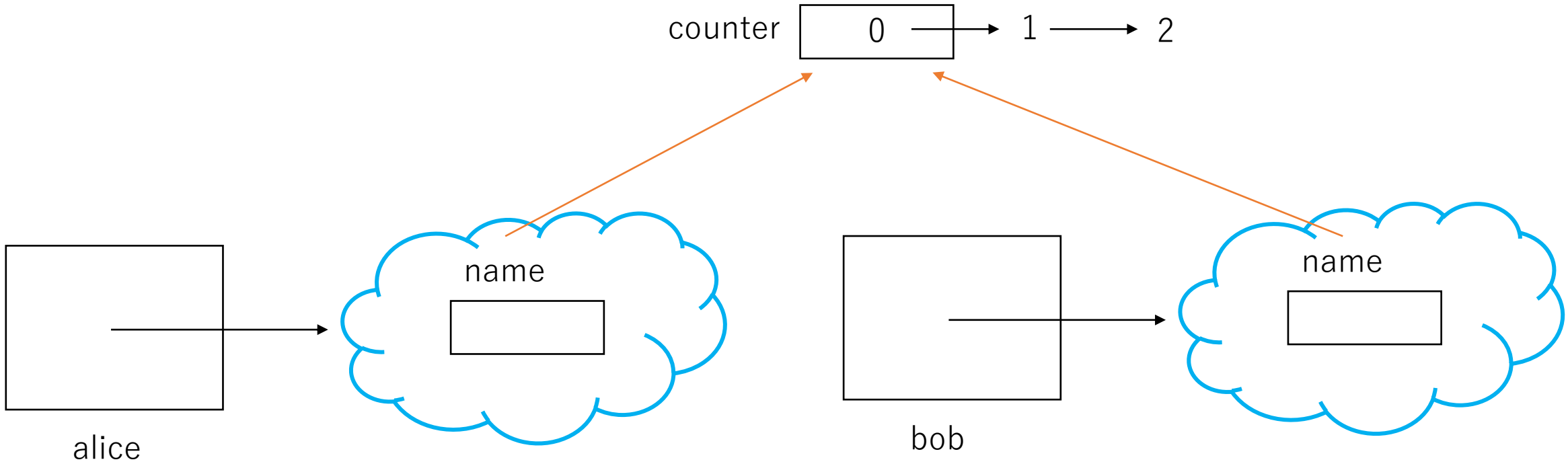




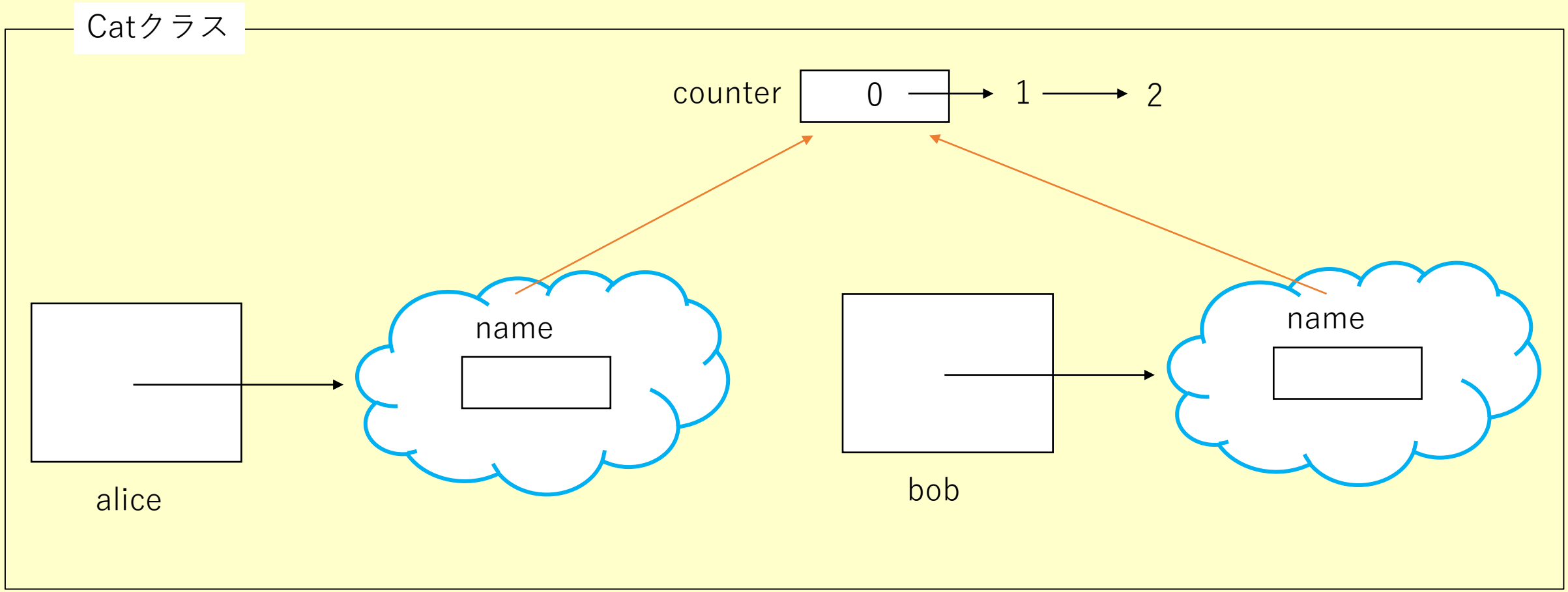




どうやったらインスタンスの個数を数えられる？



Staticでインスタンスの個数を数える



メソッドの補足事項

今までのメソッド（Staticメソッド）

クラスが持つメソッドでクラスが実行する

```
public static 戻り値の型 メソッド名(引数) {
メソッドが実行されたとき動く処理
return　戻り値
}
```

```
public static void メソッド名(引数) {
メソッドが実行されたとき動く処理

}
```

インスタンスでのメソッド

インスタンスが持つメソッドでインスタンスが実行する

```
public void メソッド名(引数) {
メソッドが実行されたとき動く処理

}
```

メソッドの違い

Main.java

```
package method;

public class Main {

public static void main(String[] args) {

Human alice = new Human("Alice", 5);
alice.eat("お饅頭");

Human.introduce();

}

}
```

インスタンスを作らないと呼び出せない

インスタンスを作らずに呼び出せる

Human.java

```
package method;

public class Human {
String name;//属性を設定
   int age;

   /*コンストラクタ*/

   Human(String name_, int age_) {
      name = name_;
      age = age_;
   }

   /*食べる*/

 void eat(String food) {
 System.out.print(name + " は " + food + "を食べます ");

}

public static void introduce(){
      System.out.println("私はダイヤがほしい");
   }

}
```

インスタンスメソッド

staticメソッド

多態性の例

① **Cat** alice ＝ new Cat()

② **Animal** alice ＝ new Cat()

Animal alice ＝ new Cat()

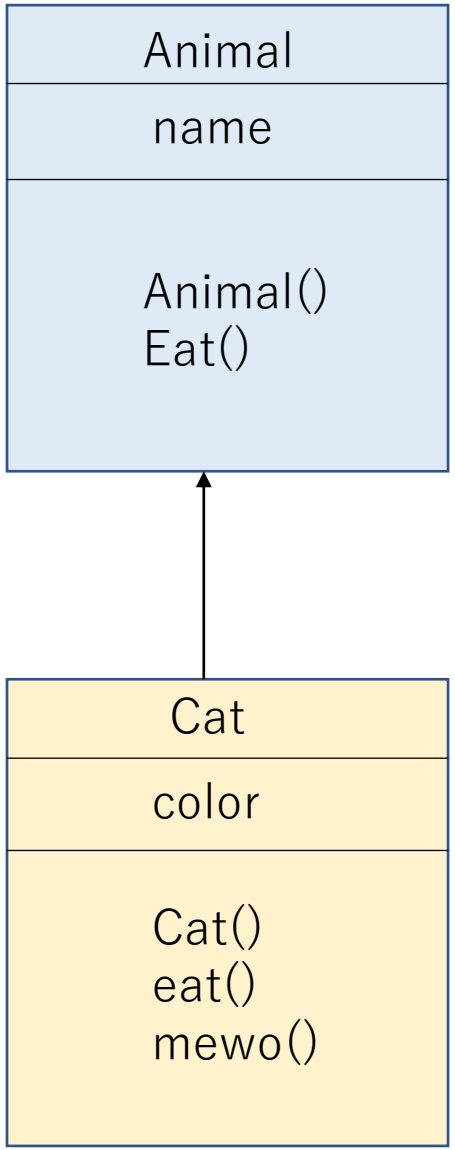
箱の型

中身の型

そのインスタンスを何とみなすか
同じCatインスタンスでもAnimal型など様々な箱に入れ換えることでとらえ方を変えることができる

そのインスタンスが一体何かは、一度newされたら何があっても変わらない

多態性の例




```
Cat alice = new Cat(“Alice");
```

子クラスの内容が親クラスに入っている
Animalクラスの変数→Animalクラスの領域しか見ることができない

```
Animal kitty = new Cat("Kitty");
```

情報がつなっがっていて
オーバーライドした後のeatが呼び出される

```
kitty.eat("キャットフード");
```

サブクラスにしかないメソッド
コンパイルエラー

```
kitty.meow();
```

情報がつなっがっていて
オーバーライドした後のeatが呼び出される