LightHouseIT

2.2 继承与多态

- 继承的基本概念
- 继承的语法
- 多态
- 接口

目录

Example

我们想设计一个动物园系统。在动物园里有各种各样的动物，如猫、狗、狮子和老虎。使用面向对象的设计思维，我们可以编写猫类、狗类、狮子类和老虎类，再分别创建它们的对象。

- 想想以下问题:
  1. 猫、狗等每个动物类是否具有一些相同的性质（属性或行为）？
  2. 在编写不同的动物类时，是否会编写重复的代码?
  3. 目前为止学过的语法，能够解决这些问题吗?

继承

- 我们发现，猫、狗等类虽然确实有一些不同的性质，但同时也具有一些共通的性质：比如都有身高、性别和年龄等属性；都有进食、睡眠等行为。

- 我们可以这么总结：每一个动物都有身高、性别和年龄等属性；有进食、睡眠等行为。并且，猫和狗都**是一种**动物。

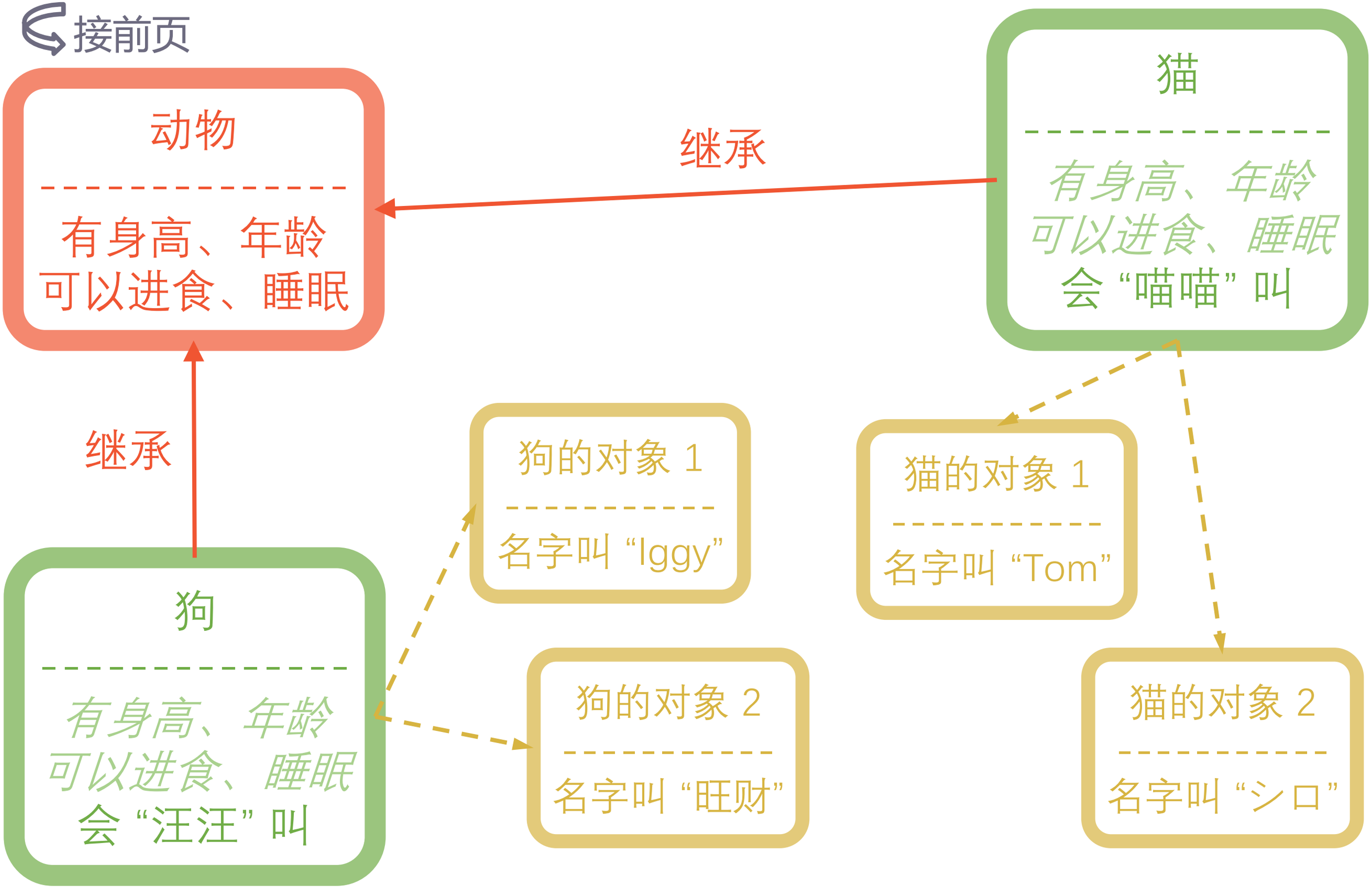
- 继承[繼承]就是指这样的“**是一种**”的关系。我们可以先定义一个动物类，再让猫和狗继承这个动物类。猫和狗会自动具有动物类的相关性质。

Example

我们发现狮子或老虎也**是一种**动物，因此它们也可以继承动物类。狮子、老虎也会自动具有上述的属性和行为。

接次页

接前页



父类和子类

- 如果 A 类继承了 B 类，我们就称 A 是 B 的**子类**[サブクラス]；相对的，B 就是 A 的**父类**[スーパークラス]。

- 判断继承关系的简单方法：如果 A **是一种（is-a）** B，那么 A 就是 B 的子类，即 A 应该继承 B。

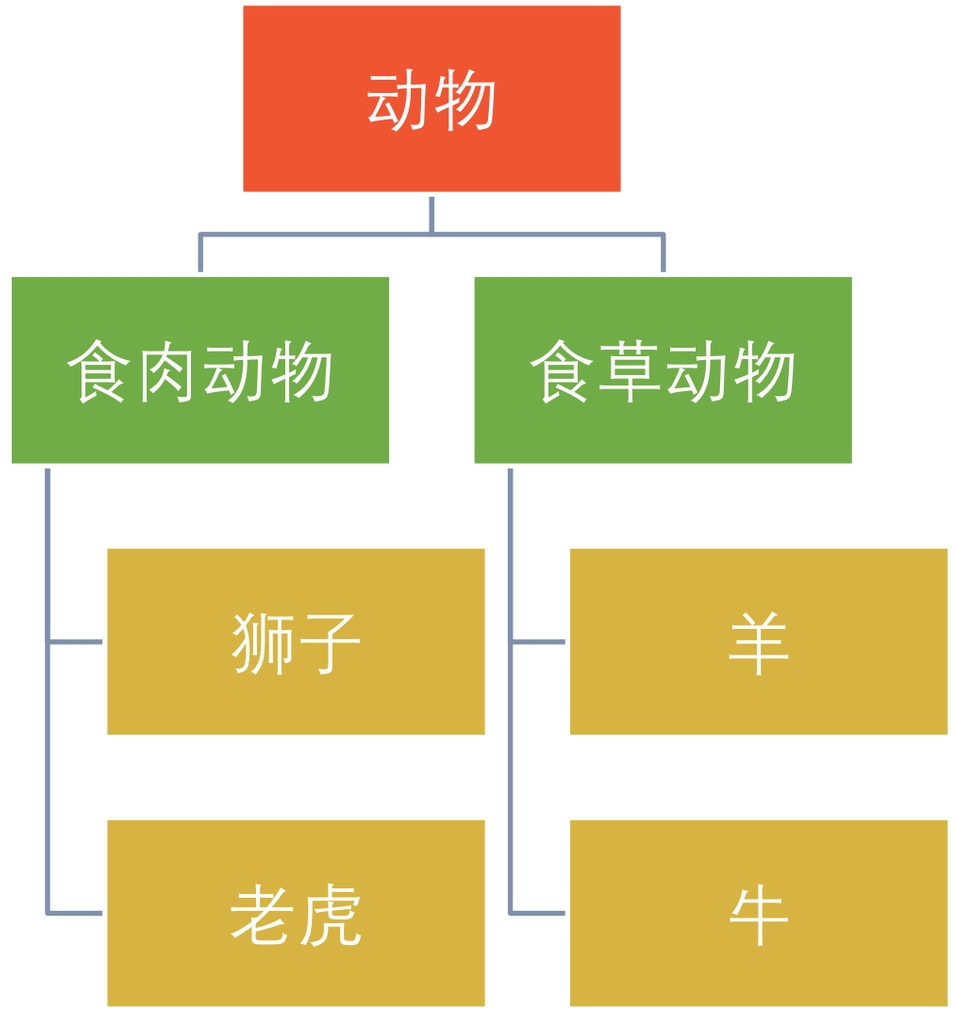
- 想一想，以下每组两个类之间应该有怎样的继承关系？
  1. 学生 和 大学生；
  2. 哈士奇 和 狗；
  3. 正方形 和 长方形；
  4. 生物 和 食草动物；
  5. 人 和 植物。

Note

子类的一个对象同时也是父类的一个对象。比如猫的一个对象（一只猫）当然也是动物类的一个对象（一个动物）。

层次化的继承

- 很多时候，系统内的各个类别之间可以形成一种层次化的结构。比如狮子或老虎**是一种**食肉动物；羊或牛**是一种**食草动物；食草动物或食肉动物**是一种**动物：
- 它们所对应的继承也会是层次化的。
- 注意到，层次化的继承满足*传递性*：狮子或羊也**是一种**动物。
- 想一想，图中哪些类是哪些类的父类？哪些类是哪些类的子类？



Question and answer

目录

Java 中的继承

- 在 Java 中定义类时，可以使用 **extends** 关键字指定它的父类：

```
1 public class Cat extends Animal {
2     codes;
3 }
```

- 父类中的大部分属性和方法，子类都可以直接使用：

Animal.java：

```
1 public class Animal {
2     String name = "";
3 
4     void eat(String food) {
5         System.out.println(name + " eat " + food);
6     }
7 }
```

Cat.java：

```
1 public class Cat extends Animal { }
```

Main.java：

```
1 public class Main {
2     public static void main(String[] args) {
3         Cat kitty = new Cat();
4         kitty.name = "Kitty";
5         kitty.eat("fish"); // => "Kitty eat fish"
6     }
7 }
```

接次页

接前页

- 你也可以在子类中定义它特有的成员变量或方法：

Cat.java：

```java
public class Cat extends Animal {
    void meow() {
        System.out.println("meow~");
    }
}
```

Main.java：

```java
public class Main {
    public static void main(String[] args) {
        Cat kitty = new Cat();
        kitty.name = "Kitty";
        kitty.meow(); // => meow~
    }
}
```

Note

在 Java 中，一个类只能有一个父类（想想这是为什么）。

子类的构造方法

- 如果我们要为子类定义构造方法，则可以在其第一行使用 **super** 关键字调用父类的构造方法。如果不写，将会自动调用父类的**无参构造方法**（”super();”）。

Animal.java：

```
1 public class Animal {
2     String name;
3
4     Animal(String name_) {
5         name = name_;
6     }
7 }
```

Cat.java：

```
1 public class Cat extends Animal {
2     Cat(String name_) {
3         super(name_);
4     }
5 }
```

Main.java：

```
Cat kitty = new Cat("Kitty");
System.out.println(kitty.name); // => Kitty
```

Try 01011 11010 01011

animals 包

子类的默认构造方法

- 我们说过（§ 2.1.2）：如果我们没有为一个类定义构造方法，Java 会自动为我们创建一个**默认构造方法**。
- 默认构造方法也会自动在第一行调用父类的**无参构造方法**（”super();”）。
- 如果父类没有无参构造方法呢?

Try

animals.constructor 包

- 总结：如果父类没有无参构造方法，我们**必须**手动为子类定义构造方法，**并必须**在第一行使用父类的构造方法。

Question and answer

目录

LightHouseIT

方法的重写

- 在子类中定义一个和父类方法同类型（即**名称**，**参数类型**和**返回类型**都一样）的方法，称为对父类方法的**重写**[オーバーライド]。

Example

猫类和狗类都继承了动物类。我们知道动物类有一个方法是表示进食的动作。然而，猫类在进食后会喵喵叫，狗类则会汪汪叫——和动物类里定义的动作是不同的。因此，我们可以重写方法来实现新的动作。

- 注意区分它和**重载**[オーバーロード]：重载必须保证参数列表类型不同。而重写必须保证参数和返回值类型都相同。

重写和多态

- 通过重写方法，当我们调用同一类对象的同一*方法*时，可以有不同的*实际表现*：具体的表现取决于对象属于哪个子类。这种性质被称为**多态**[ポリモーフィズム]。

Example

猫类和狗类都继承了动物类，并用不同方式重写了进食的方法。如果我们创建了一只猫和一只狗，它们都将是动物的对象。此时，这两只动物的进食方法将会有不一样的表现。

接次页

接前页

- 注意我们在 Java 里是怎么体现刚刚的例子的：狗或猫**是一种**动物，那么一只狗或猫的对象也是一个动物的对象。因此，我们应该可以把它存在一个动物类型的变量中：

```
Animal kitty = new Cat("Kitty");
Animal iggy = new Dog("Iggy");
```

- 现在我们能使用 kitty 的哪些方法呢？Java 虽然不知道 kitty 是一只猫，但知道它是一个动物。因此我们可以使用 kitty 继承自动物的方法：

```
kitty.eat("cat food"); // kitty 可以运行动物的方法
kitty.meow(); // 报错：Java 不知道 kitty 是一只猫
```

- 结合对 eat 方法的重写，同样是 Animal 的对象在调用 eat 时就会有不同的表现。

Try 01011 11010 01011

animals.polymorphism 包

多态的作用

- 那么，多态究竟有哪些实际价值呢？

Example

想一想我们正在做的动物园系统：动物园里可能有几十上百种动物，每种动物都会有一些自己的特征，我们需要为它们创建各自的类。
同时，我们希望把所有动物的对象存在一个数组里，我们可以依次遍历每一个动物，执行某些相同的操作。
那么，我们应该用**什么类型的数组**呢？
猫类？狗类？还是……

接次页

接前页

- 由于 Java 可以通过重写实现多态特性，我们可以在将所有对象统一存在动物类型的数组中。当我们想让所有动物依次做某种动作时，只要遍历这个数组即可。

Try 01011 11010 01011

animals.zoo 包

- 想一想，如果没有重写，在存储和使用动物们时会遇到什么麻烦？

重写的注解

- **注解**[アノテーション]是 Java 中的一种特殊语法：它并不被算在 Java 代码中（和注释类似），但是可以提醒 Java 编译器或其他程序在处理代码时进行某些特殊操作。

- 注解都以 “@” 开头。今天我们只需要了解以下这个注解：

```
1 @Override
2 void eat(String food) {
3     System.out.print(name + " ate " + food + ", ");
4     meow();
5 }
```

- “@Override” 注解声明了其下方的方法重写了父类中的方法。如果编译器注意到父类里没有该方法，就会报出语法错误。

- 虽然不加注解也能实现重写，但养成在每个重写方法前添加注解的习惯可以避免重写方法的类型错误，提高可读性。

重写的特别用法：Object.toString

- Java 中，所有的类都隐含地继承了一个基本的类 **Object**。
- 我们可以重写 Object 中定义的一些方法来实现一些特殊功能。比如，重写 toString 方法：

```
@Override
public String toString() {
    return "I'm a cat named " + name;
}
```

- 这可以方便我们把复杂的类型用统一的形式转换字符串用于输出。实际上，System.out.println 方法就使用了 toString 方法。

Question and answer

目录

多态实现中的问题

Example

通过多态，我们成功将所有动物存在了同一个数组内统一处理。现在，我们想把动物园系统进一步扩张，使它除了动物之外，还能处理动物园里的人、汽车等等其他对象。
这里，我们想再次运用多态：很多动物、汽车和人都可以“**跑**”。我们想把这样可以跑的类型统一保存起来，在发生意外时利用它们的“跑”方法进行疏散。
该怎么把动物、汽车和人保存在同一个数组中呢？

多态实现中的问题

Example

方法一：定义一个类表示“会跑的东西”，让会跑的动物、汽车和人继承这个类。
问题：Java 无法*多重继承*，狗不能同时继承动物和“会跑的东西”。

方法二：定义一个类表示“会跑的东西”，让动物类（Animal）、汽车和人继承这个类。
问题：鱼？

- 我们可以使用接口来解决这个问题。

接口

- 接口[インターフェイス]可以理解成是一种特殊的 Java 类，它和一般 Java 类的区别在于：
  1. 接口没有**成员变量**；
  2. 接口的方法只有**类型定义**，没有**具体实现**的代码（**没有 “{}”**）；
  3. 接口无法被**实例化**（创建对象）。

- 要定义接口，使用 interface 关键字：

```
1 public interface Addable {
2     int add(int a, int b);
3 }
```

- 可以看到，我们这里只定义了 add 方法的参数列表和返回值，并没有说明具体怎么实现这个方法。

接口的实装

- 在定义类时，可以使用 **implements** 关键字，注明它**实装**[实装]了某个接口。这有点类似类的*继承*（extends）：这个类会自动具有接口的所有方法的定义：

```
public class Adder implements Addable { }
```

- 然而，接口的方法都是**没有**实现的。在实际使用这个类前，我们需要先实现这些方法，即写上这些方法的具体实行代码。

```
public class Adder implements Addable {
    @Override
    public int add(int a, int b) {
        return a + b;
    }
}
```

- 这里 add 方法前的 public 我们会在之后说明。

接口的使用

- 之所以说接口的*实装*类似类的*继承*，是因为对 Java 来说，实装了接口 A 的类 B 也**是一种** A。换句话说，我们可以用接口 A 的变量来保存类 B 的对象：

```
1 Addable addable = new Adder();
2 System.out.println(addable.add(1, 2)); // => 3
```

- 接口还有一个与类不同的最大的特点：与继承不同，一个类可以实装**任意多个**接口！继承了别的类的类也可以任意实装接口。

接次页

接前页

- 实装多个接口的语法如下:

```
public class Cat extends Animal implements Runnable, Eatable { }
```

Example

我们可以定义一个 Runnable 接口代表 “会跑的东西”，让会跑的动物，汽车和人*实装*这个接口。动物，汽车和人可以*继承*各自的父类。

Try

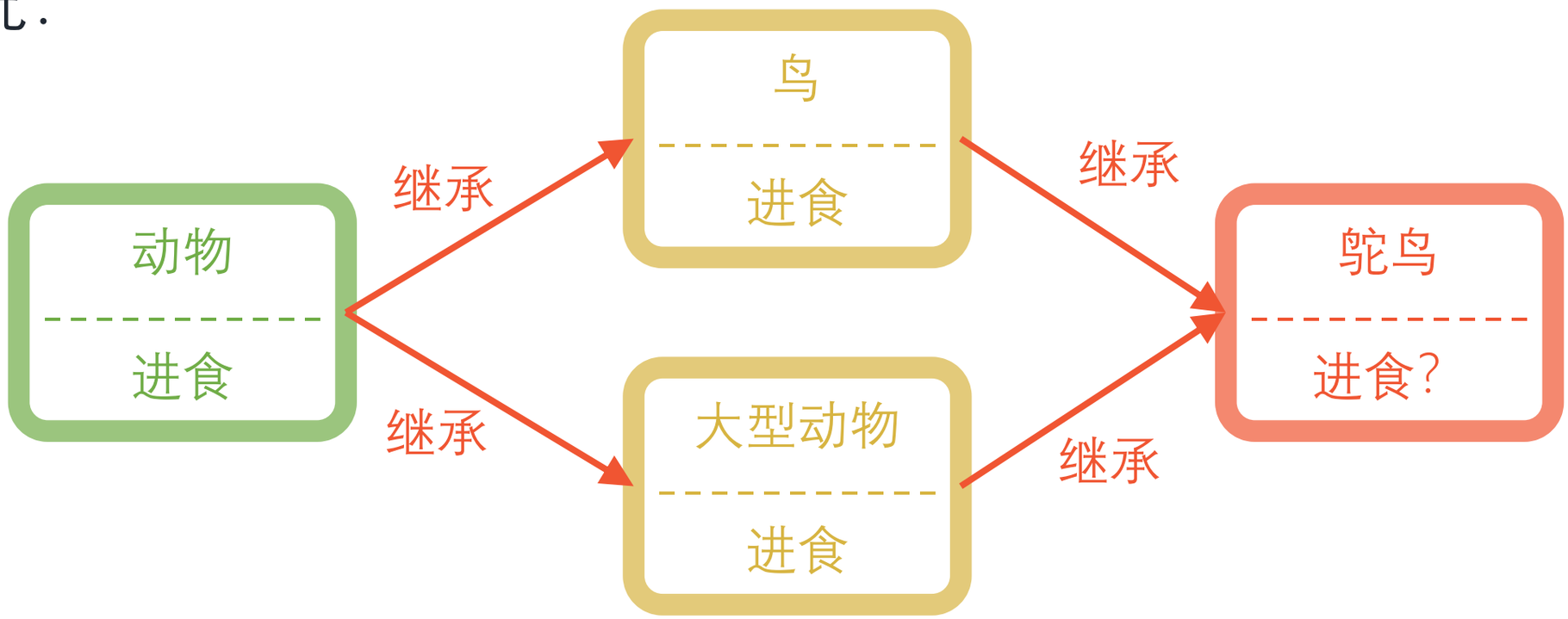
animals.bigzoo 包

Question and answer

Coffee Break

多重继承与钻石继承问题（1）

为什么 Java 不支持多重继承机制呢？考虑下面这种情况：



这里，鸟类和大型动物类可以通过不同的代码重写动物的进食方法实现多态。当 Java 想调用一个鸵鸟类对象的进食方法时，应该执行哪个类中的代码呢？

Coffee Break

多重继承与钻石继承问题（2）

这个问题被称作**钻石继承**问题，因其类之间的关系图形似钻石而得名（也译作菱形继承问题）。

在一些支持多重继承的编程语言如 C++ 中，钻石问题有时会导致代码的逻辑结构的混乱和可读性的降低，影响开发效率；同时，这些语言的编译器也会更难开发，使得语法变得更复杂。

在 Java 中，采用接口实现多重 “继承” 的效果，变相解决了这一问题。这当然不是解决问题的唯一方案，但在在面向对象编程的实践中它在开发效率和可读性上取得了出色的平衡，因此很多其他语言也都借用了这一机制。

总结

Sum Up

1. 继承的基本概念。

2. Java 的继承语法。
   ① 子类的定义和使用。
   ② 子类的构造方法。

3. 多态：
   ① 多态的基本概念。
   ② 实现多态的方法：重写。
   ③ 接口。

LightHouseIT

THANK YOU!