多 态

多 态

- 多态（Polymorphism）按字面的意思就是“多种状态”，是面向对象的程序设计语言最核心的特征。
- 从一定角度来看，封装和继承几乎都是为多态而准备的。

多 态

- 现实中，关于多态的例子不胜枚举。
  - 动物们都有吃东西，跑，跳，叫的方法，不同的动物表现方式不同
  - 按下 F1 键这个动作，针对当前工作窗口，显示不同工具的帮助文档

多态的分类

- 1、编译时多态（设计时多态）：方法重载。
- 2、运行时多态：JAVA运行时系统根据调用该方法的实例的类型来决定选择调用哪个方法则被称为运行时多态。
- 我们平时说得多态，多指运行时多态

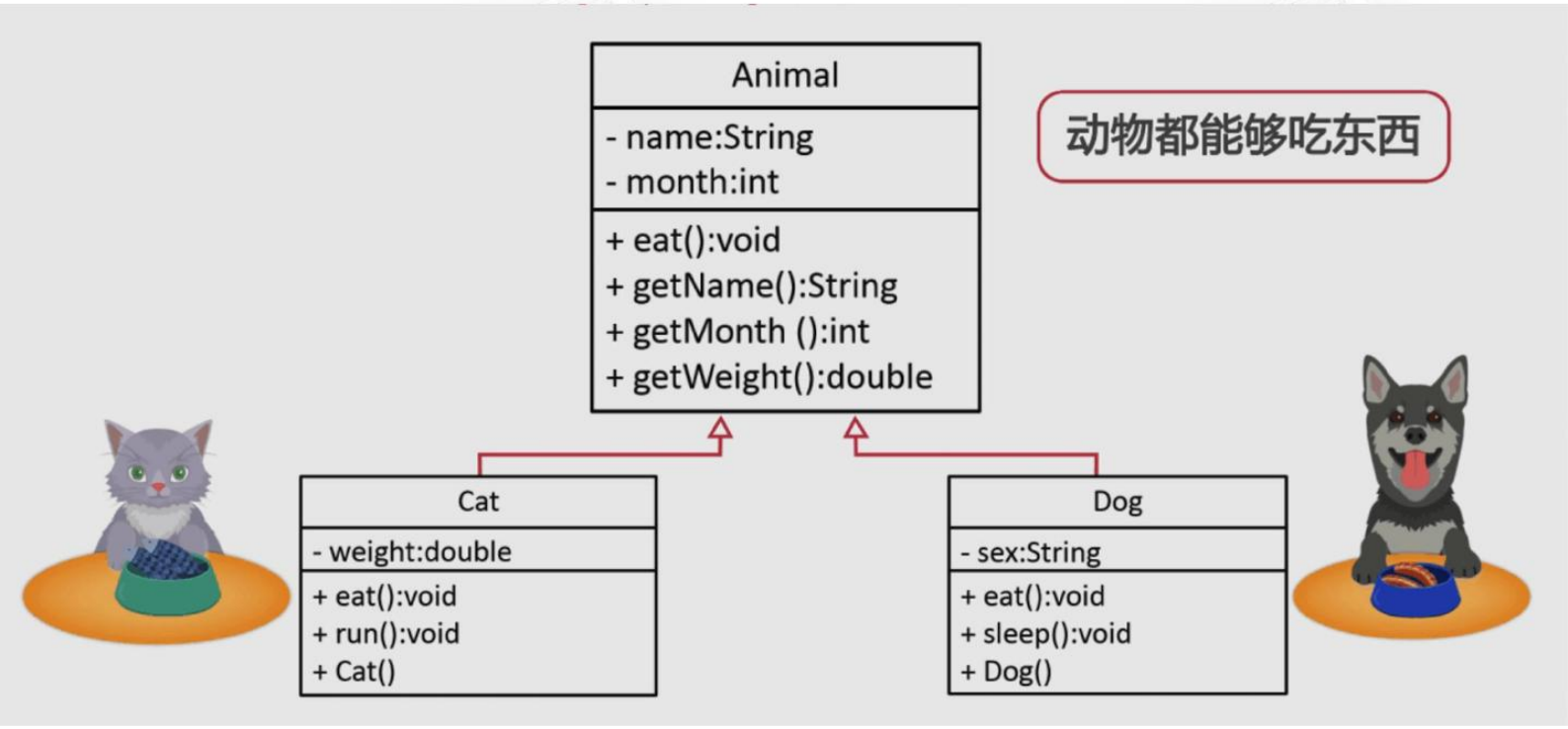
多 态

- 一个引用变量倒底会指向哪个类的实例对象，该引用变量发出的方法调用到底是哪个类中实现的方法，必须在由程序运行期间才能决定。

多 态

- 必要条件：
  - 满足继承关系
  - 父类引用指向子类对象

程序中的继承



多 态

- **向上类型转换（Upcast）：将子类型转换为父类型。**
  - **隐式/自动类型转换，是小类型到大类型的转换**
  - **对于向上的类型转换，不需要显示指定，即不需要加上前面的小括号和父类类型名**

多 态

- 向下类型转换（Downcast）：将父类型转换为子类型。
  - 将一个指向子类对象的父类引用赋值给一个子类的引用
  - 强制类型转换，是大类型到小类型
  - 父类型的引用必须指向转型的子类的对象，即指向谁才能转换成谁。不然也会编译出错
- 通过instanceof运算符，来解决引用对象的类型，避免类型转换的安全性问题，提高代码的强壮性。

多 态

注意：

1、父类引用指向子类实例时，可以调用子类重写父类的方法以及直接继承父类的方法，无法调用子类特有的方法。

2、静态static方法属于特殊情况，静态方法只能继承，不能重写。调用的时候用谁的引用，则调用谁的版本。

程序中的继承

- 以下代码有什么问题？

```
Animal pet = new Animal ("花花",2);
pet.eat();
```

- 语法没问题，但实例化Pet没有意义

抽象类

- Java中使用抽象类，限制实例化

```
public abstract class Animal {
}
```

抽象类

应用场景：

某个父类只是知道其子类应该包含怎样的方法，但无法准确知道这些子类如何实现这些方法。

抽象类

- **以下代码有什么问题？**

```
public abstract class Animal{
        public void eat() {
                //...
        }
}
```

- **每个子类中的具体方法的实现不同**

抽象方法

- abstract也可用于方法——抽象方法
  - 抽象方法没有方法体
  - 抽象方法必须在抽象类里
  - 抽象方法必须在子类中被实现，除非子类是抽象类

```
public abstract void eat();
```

抽象类 & 抽象方法

使用规则

1. abstract定义抽象类
2. 抽象类不能直接实例化，只能被继承，可以通过向上转型完成对象实例
3. abstract定义抽象方法，不需要具体实现
4. 包含抽象方法的类是抽象类

抽象类 & 抽象方法

使用规则

5. 抽象类中可以没有抽象方法
6. 子类如果没有重写父类所有的抽象方法，则也要定义为抽象类
7. abstract 不能与static、final、private共存
8. 抽象方法在子类实现时访问权限必须大于等于父类方法