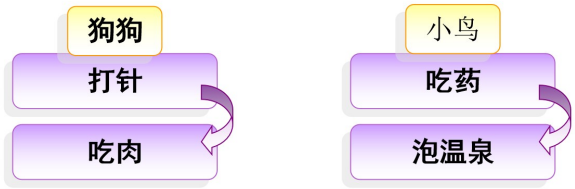
为什么使用多态

问题

- 宠物生病了，需要主人给宠物看病
  - 不同宠物看病过程不一样



  - 不同宠物恢复后体力值不一样

为什么使用多态

分析

- 编写主人类
  - 编写给狗狗看病的方法
  - 编写给小鸟看病的方法
- 编写测试方法
  - 调用主人类给狗狗看病的方法
  - 调用主人类给小鸟看病的方法

为什么使用多态

- **编码实现**

主人类

```
public class Master {
    public void Cure(Dog dog) {
        if (dog.getHealth() < 50) {
            dog.setHealth(60);
            System.out.println("打针、吃药");
        }
    }
    public void Cure(Bird bird){
        if (bird.getHealth() < 50)
            bird.setHealth(70);
            System.out.println("吃药、疗养");
    }
}
```

测试方法

```
… …
Master master = new Master();
master.Cure(dog);
master.Cure(bird);
… …
```

为什么使用多态

- **如果又需要给XXX看病，怎么办?**
  - **添加XXX类，继承Pet类**
  - **修改Master类，添加给XXX看病的方法**

频繁修改代码，代码可扩展性、可维护性差

使用多态优化设计

什么是多态

- **生活中的多态**
  - **你能列举出一个多态的生活示例吗?**

提问

同一种事物，由于条件不同，产生的结果也不同

- **程序中的多态**

多态：相同的父类引用，不同的子类实例，执行相同的行为，产生不同的结果

父类引用，子类对象

如何实现多态3-1

问题

- **用多态实现打印机**
  - **分为黑白打印机和彩色打印机**
  - **不同类型的打印机打印效果不同**

如何实现多态3-2

计算机可以连接各种打印机

无论连接何种打印机打印方法都相同

根据连接打印机不同，效果也不同

- **使用多态实现思路**
  - **编写父类**
  - **编写子类，子类重写父类方法**
  - **运行时，使用父类的类型，子类的对象**

如何实现多态3-3

- **编码实现**

父类

```
abstract class Printer(){
    print(String str);
}
```

子类

```
class ColorPrinter (){
    print(String str) {
        System.out.println("输出彩色的"+str);
    }
}
```

```
class BlackPrinter (){
    print(String str) {
        System.out.println("输出黑白的"+str
```

只能调用父类已经定义的方法

同一种操作方式，不同的操作对象

运行

```
public static void main(String[] ar    {
    Printer p = new ColorPrinter()
    p.print();
    p = new BlackPrinter();
    p.print();
}
```

实现多态的两个要素：
1. 方法重写
2. 使用父类类型

方法重写

- 方法重写的规则
  - 在继承关系的子类中
  - 重写的方法名、参数、返回值类型必须与父类相同
  - 私有方法不能继承因而也无法重写

方法重写 VS 方法重载

	位置	方法名	参数表	返回值	访问修饰符
方法重写	子类	相同	相同	相同	不能比父类更严格
方法重载	同类	相同	不同	无关	无关

instanceof运算符


```
/**
 * 测试instanceof运算符的使用。
 */
public void cure(Pet pet){
  if(pet.getHealth()<50){
     pet.toHospital();
    pet.setHealth(60);
  if(pet instanceof Bird){
      Bird bird=(Bird)pet;
     bird.run();
}else if(pet instanceof Dog){
   Dog dog=(Dog)pet;
  dog.run();
}
}
}
```

- 该运算符用来 一个接口，结果为true或false
- 在强制类型转 真实类型，可以避免类型转换

如何实现多态

- 使用多态实现思路
  - 编写具有继承关系的父类和子类
  - 子类重写父类方法
  - 使用父类的引用指向子类的对象
    - 向上转型

实现多态的三个要素

```
Pet pet = new Dog();
```

自动类型转换

- 实现多态的两种形式
  - 使用父类作为方法形参实现多态
  - 使用父类作为方法返回值实现多态

为什么使用多态

- 使用多态优化后的代码

Dog类

```
public class Dog extends Pet {
    public void
        this.setH
        System.
    }
}
```

主人类

```
public class Master {
                    t pet) {
                    < 50)
                );
```

3

又要给XXX看病时，只需：
1. 编写XXX类继承Pet类（旧方案也需要）
2. 创建XXX类对象（旧方案也需要）
3. 其他代码不变（不用修改Master类）

Bird类

```
public class Bird extends Pet {
    public void toHospital() {
        this.setHealth(70);
        System.out.println("吃药、疗养");
    }
}
```

2

测试方法

```
… …
Pet pet = new Dog();
Master master = new Master();
master.Cure(pet);
… …
```

4

小结

- 什么是多态?
- 使用的多态有什么好处?

多　　态