Java 核心技术

第五章 继承、接口和抽象类

第三节 转型、多态和契约设计

华东师范大学 陈良育

类转型(1)

- 变量支持互相转化，比如 int a = (int) 3.5;
- 类型可以相互转型，但是只限制于有继承关系的类。
  - 子类可以转换成父类，而父类不可以转为子类。
  - 子类继承父类所有的财产，子类可以变成父类(从大变小，即向上转型);从父类直接变成子类(从小变大，即向下转型)则不允许。

```
13    Human obj1 = new Man(); //OK, Man extends Human
14    Man obj2 = new Human(); //illegal, Man is a derived class Human
```

类转型(2)

- 父类转为子类有一种情况例外
  - 就是这个父类本身就是从子类转化过来的。

```
13          Human obj1 = new Man(); //OK, Man extends Human
14          Man obj2 = (Man) obj1;  //OK, because obj1 is born from Man class.
```

多态(1)

- 类型转换，带来的作用就是多态。
- 子类继承父类的所有方法，但子类可以重新定义一个名字、参数和父类一样的方法，这种行为就是重写(覆写，覆盖，overwrite/override, not overload(重载))。
- 子类的方法的优先级高于父类的。

多态(2)


```
public class Man extends Human {
    public void eat() {
        System.out.println("I can eat more");
    }
    public void plough() { }

    public static void main(String[] a) {
        Man obj1 = new Man();
        obj1.eat();    // call Man.eat()
        Human obj2 =  (Human) obj1;
        obj2.eat();    // call Man.eat()
        Man obj3 = (Man) obj2;
        obj3.eat();    // call Man.eat()
    }
}
```

多态(3)

- 多态的作用
  - 以统一的接口来操纵某一类中不同的对象的动态行为
  - 对象之间的解耦
- 参看AnimalTest.java

契约设计(1)

- Java编程设计遵循契约精神
- 契约：规定规范了对象应该包含的行为方法
- 接口定义了方法的名称、参数和返回值，规范了派生类的行为
- 基于接口，利用转型和多态，不影响真正方法的调用，成功地将调用类和被调用类解耦(decoupling)

契约设计(2)

- 被调用类(havaLunch只和Animal有联系)

```
public static void haveLunch(Animal a)
{
    a.eat();
}
```

- 调用类

```
haveLunch(new Cat());
haveLunch(new Dog());
haveLunch(
        new Animal()
        {
            public void eat() {
                System.out.println("I can eat from an anonymous class");
            }
            public void move() {
                System.out.println("I can move from an anonymous class");
            }

        });
```

总结

- 类转型：子类可以转父类，父类不可以转子类(除非父类对象本身就是子类)
- 多态：子类转型为父类后，调用普通方法，依旧是子类的方法
- 契约设计：类不会直接使用另外一个类，而是采用接口的形式，外部可以“空投”这个接口下的任意子类对象

代码(1) Human/Man.java


```java
public class Human {
    int height;
    int weight;

    public void eat()  {
        System.out.println("I can eat!");
    }
}
```

```java
public class Man extends Human {
    public void eat() {
        System.out.println("I can eat more");
    }

    public void plough() { }

    public static void main(String[] a) {
        Man obj1 = new Man();
        obj1.eat();    // call Man.eat()
        Human obj2 =  (Human) obj1;
        obj2.eat();    // call Man.eat()
        Man obj3 = (Man) obj2;
        obj3.eat();    // call Man.eat()
    }
}
```

代码(2) Animal/Cat/Dog.java


```java
public interface Animal {
    public void eat();
    public void move();
}
```

```java
public class Cat implements Animal
{
    public void eat() {
        System.out.println("Cat: I can eat");
    }

    public void move(){
        System.out.println("Cat: I can move");
    }
}
```

```java
public class Dog implements Animal
{
    public void eat() {
        System.out.println("Dog: I can eat");
    }

    public void move() {
        System.out.println("Dog: I can move");
    }
}
```

代码(3) AnimalTest.java


```
public class AnimalTest {

    public static void haveLunch(Animal a)  {
        a.eat();
    }

    public static void main(String[] args) {
        Animal[] as = new Animal[4];
        as[0] = new Cat();
        as[1] = new Dog();
        as[2] = new Cat();
        as[3] = new Dog();

        for(int i=0;i<as.length;i++) {
            as[i].move();  //调用每个元素的自身的move方法
        }
        for(int i=0;i<as.length;i++) {
            haveLunch(as[i]);
        }

        haveLunch(new Cat());  //Animal  a = new Cat();  haveLunch(a);
        haveLunch(new Dog());
        haveLunch(
                new Animal()
                {
                    public void eat() {
                        System.out.println("I can eat from an anonymous class");
                    }
                    public void move() {
                        System.out.println("I can move from an anonymous class");
                    }

                });
    }
}
```

谢 谢！