

final

2018年6月20日 12:47

final可以修饰类、方法、变量

```
/*
```

由于继承中有一种现象：方法重写

父类的功能会被子类覆盖掉

如果不想要子类覆盖掉父类的功能，java提供了关键字final

```
*/
```

```
class Fu{  
    public final void show(){  
        System.out.println("不能修改");  
    }  
}
```

```
class Zi extends Fu{  
    //Zi中的show()无法覆盖Fu中的show()  
    public void show(){  
        System.out.println("这是一堆垃圾");  
    }  
}
```

```
class ZiDemo{  
    public static void main(String[] args){  
        Zi z = new Zi();  
        z.show();  
    }  
}
```

final的特点

2018年6月22日 18:57

类

//final class Fu 无法从最终Fu进行继承

该类不能被继承（最底层类）

方法

被final修饰的方法不能被重写

变量

该变量不能被重新赋值，这个变量是常量

常量

1. 字面值常量 “hello” 4

2. 自定义常量: final int x = 10;

//final class Fu 无法从最终Fu进行继承

```
class Fu{
    public int num = 10;
    public final int num2 = 20;
}
```

```
class Zi extends Fu{
    //Zi中的show()无法覆盖Fu中的show()
    public void show(){
        num = 100;
        //num2 = 200
        System.out.println(num);
        System.out.println(num2);
    }
}
```

```
class FinalDemo{
    public static void main(String[] args){
        Zi z = new Zi();
        z.show();
    }
}
```

final修饰局部变量

2018年6月22日 19:44

基本类型：基本类型的值不能发生改变

引用类型：引用类型的地址值不能发生改变，但对象的堆内存值可以改变

/*

面试题：final修饰局部变量的问题

*/

```
class Student{
    int age = 10;
}
```

```
class FinalTest{
    public static void main(String[] args){
        //局部变量是基本数据类型
        int x = 10;
        x = 100;
        System.out.println(x);
        final int y = 10;
        //y = 100; 无法为最终变量y分配值
        System.out.println(y);

        //局部变量是引用数据类型
        Student s = new Student();
        System.out.println(s.age);
        s.age = 100;
        System.out.println(s.age);

        final Student ss = new Student();
        System.out.println(ss.age);
        ss.age = 100;
        System.out.println(ss.age);//与上面输出结果一样

        //重新分配内存空间,不可以
        ss = new Student();
    }
}
```

final修饰变量初始化时机

2018年6月22日 19:46

- 1.被final修饰的变量只能赋值一次
- 2.在构造方法完毕前（非静态的常量）

```
class Demo{
    int num;
    final int num2;
    {
        //num2 = 10;
    }
    public Demo(){
        num = 100;
        num2 = 200;
    }
}
```

```
class FinalTest2{
    public static void main(String[] args){
        Demo d = new Demo();
    }
}
```

多态概述

2018年6月23日 12:56

多态：某一个（对象）事物在不同时刻表现出不同的状态

猫 m=new 猫 ()

动物 d = new 猫 ()

多态的前提：1.要有继承关系

2. 要有方法重写（其实没有也是可以的，但如果没有这个就没有意义）

3.要有父类引用指向子类

父 f = new 子 ()

/*

多态中的成员访问特点：

1.成员变量：编译看左边，运行看左边

2.构造方法：创建子类对象时，访问父类的构造方法，对父类的数据进行初始化

3.成员方法：编译看左边，运行看右边

4.静态方法;编译看左边，运行看左边(静态与类相关，算不上重写)

由于成员方法存在重写，因此运行看右边

*/

```
class Fu{
    public int num = 100;
    public void show(){
        System.out.println("show Fu");
    }
    public static void function(){
        System.out.println("function Fu");
    }
}
```

```
class Zi extends Fu{
    public int num = 1000;
    public int num2 = 200;
    //方法重写
    public void show(){
        System.out.println("show Zi");
    }
    public void method(){
        System.out.println("method Zi");
    }
}
```

```

    }
    public static void function(){
        System.out.println("function Zi");
    }
}

class DuoTaiDemo{
    public static void main(String[] args){
        Fu f =new Zi();
        System.out.println(f.num);
        //System.out.println(f.num2);找不到符号
        f.show();
        //f.method();找不到符号
        f.function();
    }
}

```

多态的好处

2018年6月23日 13:34

- 1.提高了代码的维护性 (继承保证)
- 2.提高了代码的扩展性 (由多态保证)

//猫狗案例

```
class Animal{
    public void eat(){
        System.out.println("eat");
    }
    public void sleep(){
        System.out.println("sleep");
    }
}

class Dog extends Animal{
    public void eat(){
        System.out.println("dog eat meat");
    }
    public void sleep(){
        System.out.println("dog sleep");
    }
}

class Cat extends Animal{
    public void eat(){
        System.out.println("cat eat fish");
    }
    public void sleep(){
        System.out.println("cat sleep");
    }
}

//针对动物操作的工具类
class AnimalTool{
    private AnimalTool(){}
    /*
    public static void useCat(Cat c){
        c.eat();
        c.sleep();
    }
    public static void useDog(Dog d){
```

```


        d.eat();
        d.sleep();
    }
    */
    public static void useAnimal(Animal a){
        a.eat();
        a.sleep();
    }
}

```

```

class DuoTaiTest{
    public static void main(String[] args){
        Cat c1 = new Cat();
        c1.eat();
        c1.sleep();
        Cat c2 = new Cat();
        c2.eat();
        c2.sleep();
        Dog d1 = new Dog();
        System.out.println("-----");
        //用方法改进
        AnimalTool.useAnimal(c1);
        AnimalTool.useAnimal(c2);
        AnimalTool.useAnimal(d1);
    }
}

```

//我喜欢宠物，定义猪类，继承自动物，提供对应的方法重写，**添加工具方法**  无需修改代码

```

}

```


多态的弊端

2018年6月23日 14:17

不能使用子类的特有功能

解决方法:

1.创建子类对象,调用方法即可 (不合理,占用内存)

2.把父类的引用强制转换为子类的引用 (向下转型)

对象间的转型问题: 向上转型 `Fu f = new Zi();`

向下转型 `Zi z = (Zi) f;` 要求f必须能转换成Zi的

```
class Fu{
    public void show(){
        System.out.println("show Fu");
    }
}

class Zi extends Fu{
    public void show(){
        System.out.println("show Zi");
    }
    public void method(){
        System.out.println("method Zi");
    }
}

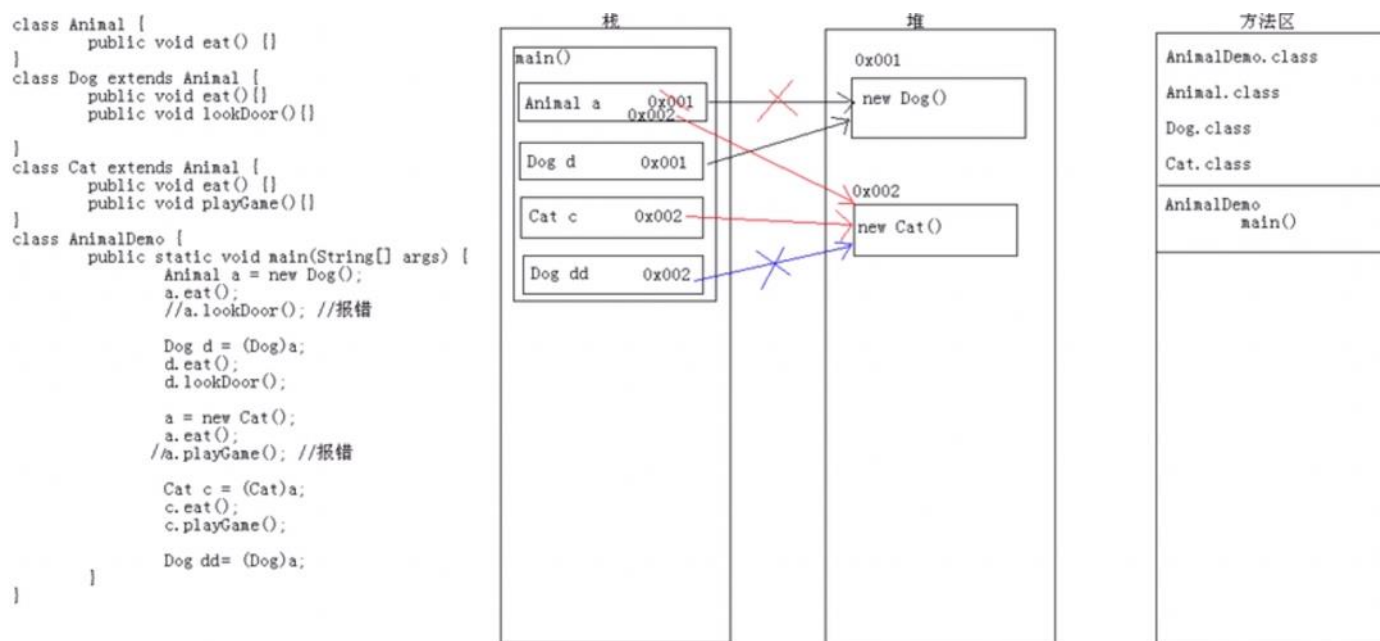
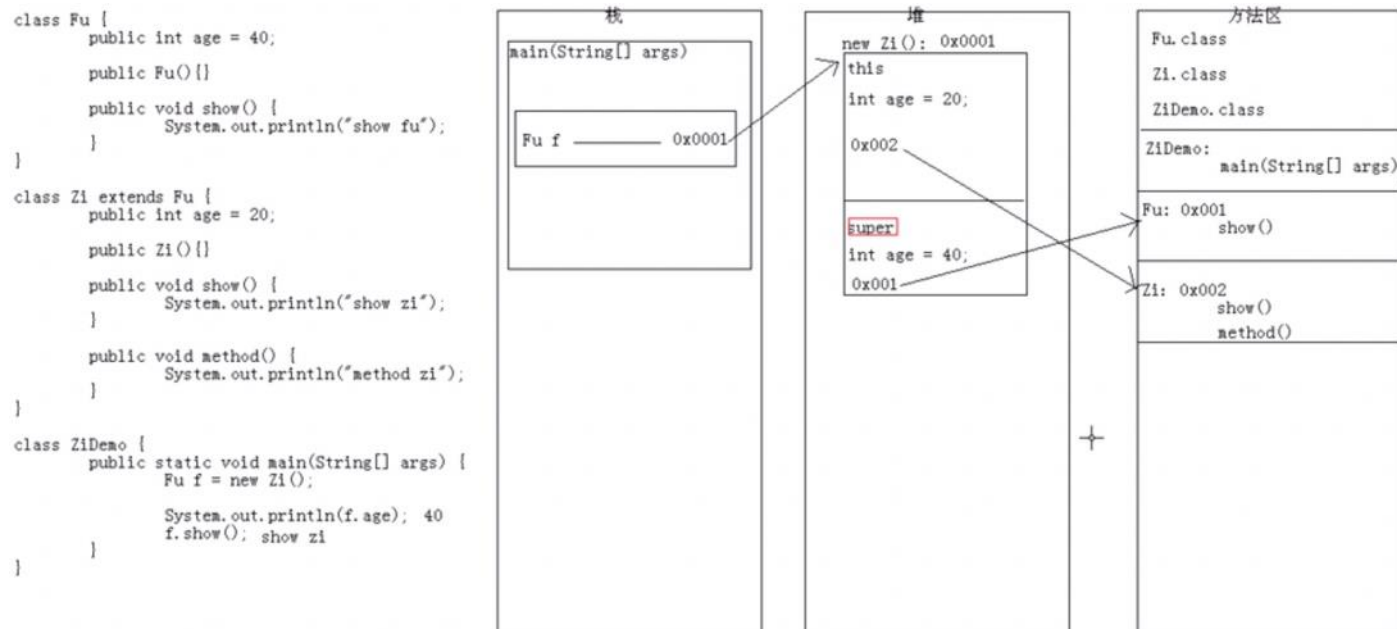
class DuoTaiDemo{
    public static void main(String[] args){
        Fu f =new Zi();
        f.show();
        //f.method();找不到符号
        /*
        创建子类对象
        Zi z = new Zi();
        z.show();
        z.method();
        */

        //把父的引用赋值给子的引用 强制转换
        Zi z = (Zi) f;
        z.show();
        z.method();
    }
}
```

内存图

2018年6月23日 15:23

ClassCastException 类型不匹配 (在多态向下转型中容易出现)



多态练习

2018年6月24日 14:41

```
class A{
    public void show(){
        show2();
    }
    public void show2(){
        System.out.println("I");
    }
}

class B extends A{
    public void show2(){
        System.out.println("Love");
    }
}

class C extends B{
    public void show(){
        super.show();
    }
    public void show2(){
        System.out.println("You");
    }
}

public class DuoTaiTest2{
    public static void main(String[] args){
        A a = new B();
        a.show();//Love
        B b = new C();
        b.show();//You
    }
}
```

抽象类

2018年6月24日 14:58

在java中，一个没有方法体的方法定义为抽象方法
而类中没有抽象方法，该类必须定义为抽象类

- 特点：
- 1.抽象类和抽象方法必须用abstract修饰
 - 2.抽象类中不一定有抽象方法，但有抽象方法的类必须为抽象类
 - 3.抽象类不能实例化，因为它不是具体的
抽象类有构造方法，但不能实例化，用于子类访问父类数据的初始化
 - 4.抽象类的子类：
 - a.如果不想重写抽象方法，抽象类的子类是一个抽象类
 - b.重写所有的抽象方法，这时候子类是一个具体的类

```
abstract class Animal{
    //抽象方法
    public abstract void eat();
    //public abstract void eat(){} 空方法体，会报错
    public Animal(){}
}

abstract class Dog extends Animal{}

class Cat extends Animal{
    public void eat(){
        System.out.println("猫吃鱼");
    }
}

class AbstractDemo{
    public static void main(String[] args){
        //通过多态的方法实例化
        Animal a = new Cat();
        a.eat();//猫吃鱼
    }
}
```

抽象类成员特点

2018年6月24日 16:08

/*

抽象类的成员特点：

成员变量：既可以是变量，也可以是常量

构造方法：有，用于子类访问父类数据的初始化

成员方法：既可以是抽象的，也可以是非抽象的

抽象方法特性：1.抽象方法：强制子类做的事

2.非抽象方法：子类继承的事情，提高代码复用性

*/

```
abstract class Animal{
    public int num = 10;
    public final int num2 = 20;
    public Animal(){
    }
    public Animal(String[] args){
    }
    public abstract void show();
    public void method(){
        System.out.println("method");
    }
}
```

```
class Dog extends Animal{
    public void show(){
        System.out.println("dog");
    }
}
```

```
class AbstractDemo2{
    public static void main(String[] args){
        Animal a = new Dog();
        a.num = 100;
        System.out.println(a.num);
        System.out.println(a.num2);
        a.show();
        a.method();
    }
}
```

抽象类小问题

2018年6月28日 14:47

1.一个类没有抽象方法，也可以定义抽象类

目的：不让创建对象

2.abstract不能和以下关键字共存：

Private 冲突:私有不能被继承，更不能被重写

Final 冲突

Static 无意义

接口

2018年6月28日 15:02

接口的特点：

1.接口用关键字interface表示

interface 接口名()

2.类实现接口用implements表示

class 类名 implements 接口名{

3.接口是抽象的，无法实例化

按照多态的方式实例化

4.接口的子类：

可以是抽象类，意义不大

也可以是具体类，重写接口中的所有抽象方法

具体类多态（几乎没有）

抽象类多态（常用）

接口多态（最常用）

接口成员特点

2018年6月28日 15:30

```
/*
```

接口成员变量特点:

成员变量: 只能是常量, 并且是静态。有默认修饰符 public static final

构造方法: 接口没有构造方法 (所有类都继承自一个类object)

成员方法: 只能是抽象方法, 默认为public abstract

```
*/
```

```
interface Inter{  
    public int num = 10;  
    public final int num2 = 20;  
    //public Inter(){}  
    void show();//默认为public abstract  
}
```

//接口名+Impl这种格式是接口的实现类格式

```
class InterImpl extends Object implements Inter{  
    public InterImpl(){  
        super();  
    }  
    public void show(){};  
}
```

```
class InterfaceDemo2{  
    public static void main(String[] args){  
        Inter i = new InterImpl();  
        System.out.println(i.num);//10  
        System.out.println(i.num2);//20  
        //i.num = 100;  
        //i.num2 = 200;不能修改 (final)  
        //System.out.println(i.num);  
        //System.out.println(i.num2);  
        System.out.println(Inter.num);  
        System.out.println(InterImpl.num2);//10 默认是静态  
        System.out.println("-----");//20  
    }  
}
```


接口与类的关系

2018年6月28日 15:53

/*

类与类：继承关系，只能单继承，可以多层继承

类与接口：实现关系,可以单实现，也可以多实现，并且可以继承一个类的同时实现多个接口

接口与接口：继承关系，可以单继承，也可以多继承

*/

```
interface Father{  
    public abstract void show();  
}
```

```
interface Mother{  
    public abstract void show2();  
}
```

```
interface Sister extends Father,Mother{
```

```
class Son implements Father,Mother{  
    public void show(){  
        System.out.println("show son");  
    }  
    public void show2(){  
        System.out.println("show2 son");  
    }  
}
```

```
class InterfaceDemo3{  
    public static void main(String[] args){  
        Father f = new Son();  
        f.show();  
        Mother m = new Son();  
        m.show2();  
    }  
}
```

抽象类与接口的区别

2018年6月28日 16:11

● 成员区别

- 抽象类 变量,常量;有抽象方法;抽象方法,非抽象方法
- 接口 常量;抽象方法

● 关系区别 I

- 类与类 继承, 单继承
- 类与接口 实现, 单实现, 多实现
- 接口与接口 继承, 单继承, 多继承

● 设计理念区别

- 抽象类 被继承体现的是: "is a"的关系。共性功能
- 接口 被实现体现的是: "like a"的关系。扩展功能

猫狗案例加入跳高功能

2018年6月28日 16:16

//定义调高接口

```
interface Jumping{  
    public abstract void jump();  
}
```

//动物抽象类

```
abstract class Animal{  
    private String name;  
    private int age;  
    public Animal(){  
    public Animal(String name,int age){  
        this.name = name;  
        this.age = age;  
    }  
    public String getName(){  
        return name;  
    }  
    public int getAge(){  
        return age;  
    }  
    public void setName(String name){  
        this.name = name;  
    }  
    public void setAge(int age){  
        this.age = age;  
    }  
    public abstract void eat();  
    public void sleep(){  
        System.out.println("睡觉");  
    }  
}
```

//具体猫类

```
class Cat extends Animal{  
    public Cat(){  
    public Cat(String name,int age){
```

```

        super(name,age);
    }
    public void eat(){
        System.out.println("猫吃鱼");
    }
}

```

//具体狗类

```

class Dog extends Animal{
    public Dog(){}
    public Dog(String name,int age){
        super(name,age);
    }
    public void eat(){
        System.out.println("狗吃肉");
    }
}

```

//有调高功能的猫

```

class JumpCat extends Cat implements Jumping{
    public JumpCat(){}
    public JumpCat(String name,int age){
        super(name,age);
    }
    public void jump(){
        System.out.println("跳高猫");
    }
}

```

//有调高功能的狗

```

class JumpDog extends Dog implements Jumping{
    public JumpDog(){}
    public JumpDog(String name,int age){
        super(name,age);
    }
    public void jump(){
        System.out.println("跳高狗");
    }
}

```

```
class InterfaceTest{
    public static void main(String[] args){
        JumpCat jc = new JumpCat();
        jc.setName("Amy");
        jc.setAge(3);
        System.out.println(jc.getName()+"---"+jc.getAge());
        jc.eat();
        jc.sleep();
        jc.jump();
    }
}
```