

上次课程复习

- 面向对象编程
- 类的概念、定义与使用
- 对象的创建、使用
- 参数传值（基本型、引用型）
- 方法重载（单个类中多个同名方法）
- static关键字（变量、方法）
- this关键字（当前调用方法的对象）
- 包和import语句（定义位置、引入类）
- Java类中的访问权限（四个修饰符）



子类与继承



邹国兵

上海大学

计算机学院



第5章 子类与继承

- 1. 子类与父类
 - 2. 子类的继承性
 - 3. 子类对象的特点
 - 4. 成员变量的隐藏和方法重写
 - 5. super关键字
 - 6. final关键字
 - 7. 对象的上转型对象
 - 8. 继承与多态
 - 9. abstract类与abstract方法
 - 10. 面向抽象编程
 - 11. 开-闭原则
- 

1. 子类与父类

➤ 当编写一个类时，若发现某个类已经有了我们所需要的成员变量和方法，要想复用这个类中的成员变量和方法，可以将要编写的类声明为这个类的子类

➤ 在类的声明中，通过使用关键字**extends**来声明一个类的子类，格式如下：

```
class 子类名 extends 父类名 {
```

```
    ...
```

```
}
```

■■■ 1. 子类与父类

- 子类继承父类的成员变量和方法作为子类的成员变量和方法，就像在子类中直接声明一样，可被子类定义的实例方法操作
- 子类不仅可以从父类继承成员变量和方法，而且根据需要还可以声明它自己的新成员变量、定义新的方法



1. 子类与父类

```

1 package shu.ces.java.chap5;
2
3 public class FatherClass {
4     String name;
5     char gender;
6     protected int age;
7     public String school;
8
9     String getName() {
10        return name;
11    }
12
13    private void setName(String name) {
14        this.name = name;
15    }
16
17    char isGender() {
18        return gender;
19    }
20
21    void setGender(char gender) {
22        this.gender = gender;
23    }

```

```

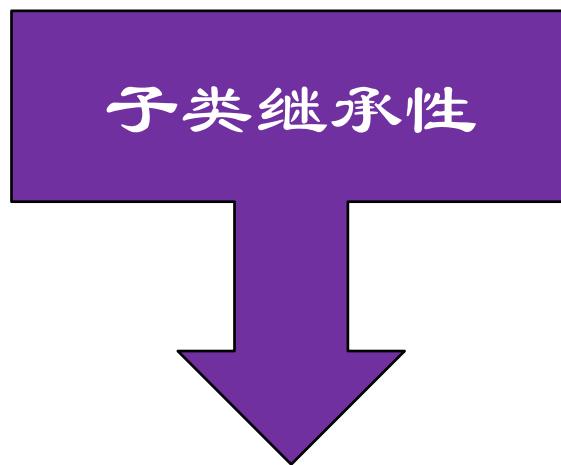
1 package shu.ces.java.chap5;
2
3 public class SubClass extends FatherClass {
4     private String address;
5
6     private String getAddress() {
7         return address;
8     }
9
10    public void setAddress(String address) {
11        this.address = address;
12    }
13
14    public static void main(String[] args){
15        SubClass sc = new SubClass();
16
17        sc.name="Guobing Zou";
18        sc.gender='男';
19        sc.age=39;
20        sc.school="Shanghai University";
21        sc.address="99 Shangda Road, Baoshan District, Shanghai, China";
22
23        System.out.println(sc.getName());
24        System.out.println(sc.isGender());

```

父类哪些变量和方法能够被子类继承呢？

2. 子类的继承性

- Java只支持单继承，不允许多继承：一个子类只能有一个父类，一个父类可以有多个子类
- 访问限制修饰符：private、protected、public、友好，它们不仅限制了对象对自己成员变量操作和方法调用，也限制了子类的继承性



子类的继承性（修饰符）

private

任何子类
不能继承

友好

同包子类
可以继承

protected

不同包子类
也可以继承

public

不同包子类
也可以继承

■■■ 2.1 同一包的继承性

➤ 当子类和父类在同一个包中时：

1. 子类继承父类中的除private访问权限以外的其他成员变量作为子类的成员变量；
2. 子类继承父类中的除private访问权限以外的其他方法作为子类的方法。

➤ 如果子类中定义的实例方法不能操作父类的某个成员变量或方法，那么该成员变量或方法就没有被子类继承



```

1. People.java
2. public class People {
3.     double height=170, weight=67.9;
4.     protected void tellHeightAndWeight() {
5.         System.out.printf("我的体重和身高:%.2fkg
6.                           ,%.2fcm\n", weight, height);
7.     }
8. }
9. Student.java
10. public class Student extends People {
11.     int number;
12.     void tellNumber() {
13.         System.out.println("我的学号是:" + number);
14.     }
15.     int add(int x, int y) {
16.         return x+y;
17.     }
18.     int sub(int x, int y) {
19.         return x-y;
20.     }
21. }
22. UniverStudent.java
23. public class UniverStudent extends Student {
24.     int multi(int x, int y) {
25.         return x*y;
26.     }
27.     double div(double x, double y) {
28.         return x/y;
29.     }

```

```

1. Example5_1.java
2. public class Example5_1 {
3.     public static void main(String args[]) {
4.         int x=12, y=18;
5.         Student zhang = new Student();
6.         zhang.weight=73.8;
7.         zhang.height=177;
8.         zhang.number=100101;
9.         zhang.tellHeightAndWeight();
10.        zhang.tellNumber();
11.        System.out.print("zhang会做加减: ");
12.        int result=zhang.add(x, y);
13.        System.out.printf("%d+%d=%d\t", x, y, result);
14.        result=zhang.sub(x, y);
15.        System.out.printf("%d-%d=%d\n", x, y, result);
16.        UniverStudent geng = new UniverStudent();
17.        geng.number=6609;
18.        geng.tellHeightAndWeight();
19.        geng.tellNumber();
20.        System.out.print("geng会做加减乘除: ");
21.        result=geng.add(x, y);
22.        System.out.printf("%d×%d=%d\t", x, y, result);
23.        result=geng.sub(x, y);
24.        System.out.printf("%d÷%d=%d\t", x, y, result);
25.        result=geng.multi(x, y);
26.        System.out.printf("%d×%d=%d\t", x, y, result);
27.        double re=geng.div(x, y);
28.        System.out.printf("%d÷%d=%f\n", x, y, re);
29.    }
30. }

```

我的体重和身高:73.80kg,177.00cm

我的学号是:100101

zhang会做加减: 12+18=30 12-18=-6

我的体重和身高:67.90kg,170.00cm

我的学号是:6609

geng会做加减乘除: 12+18=30 12-18=-6

Eclipse演示

12×18=216

12÷18=0.666667

2.2 不同包的继承性

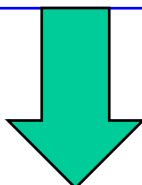
- 当子类和父类不在同一个包中时，private和友好访问权限的成员变量不会被继承：
 1. 子类只继承父类中的protected和public访问权限的成员变量作为子类的成员变量；
 2. 子类只继承父类中的protected和public访问权限的方法作为子类的方法。

2.3 protected的说明

- 一个类A中的protected成员变量和方法可以被它的直接子类和间接子类继承：
- 如B是A的子类，C是B的子类，那么B和C类都继承了A类的protected成员变量和方法。
 1. 如果用C类在C中创建了一个对象，那么该对象总是可以通过“.”运算符访问继承的或者自己定义的protected变量和protected方法

2.3 protected的说明

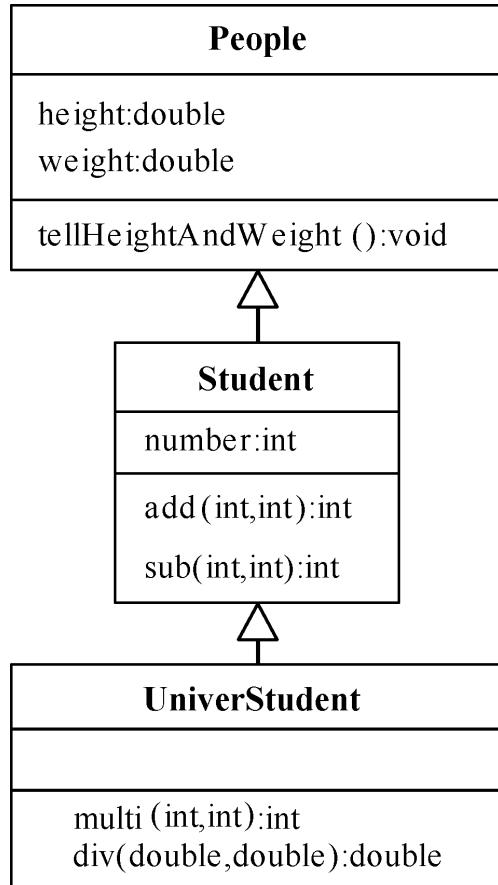
2. 但是，如果在另外一个类(如Other类)中用C类创建了一个对象object，该对象通过“.”运算符访问protected变量和protected方法的权限如何？



Eclipse演示

- (1) 对于C中声明的protected成员变量和方法，如果要访问这些protected成员变量和方法，则Other类和C类需在同一个包中。
- (2) 如果C的对象的protected成员变量或protected方法是从父类继承的，那么就要一直追溯到该protected成员变量或方法的“来源”类，即定义类，如果Other类和“来源”类在同一个包中，那么能访问继承的protected变量和protected方法。

2.4 继承关系的UML图



- 如果一个类是另一个类的子类，那么UML通过使用一个实线连接两个类的UML图来表示具有继承关系的两个类之间的泛化关系(Generalization)；
- 实线的起始端是子类的UML图，终点端是父类的UML图，且在终点端使用一个空心三角形表示实线的结束。

■■■ 2.5 instanceof运算符

- instanceof是双目运算符，其左面的操作元是对象，右面的操作元是类：
- 当左面操作元是右面的类或子类创建的对象，instanceof运算结果是true，否则是false

例： Student wang = new Student();

wang instanceof Student 是true

wang instanceof People 是true

wang instanceof UniverStudent 是false

Eclipse演示



3 子类对象的特点

- 当用子类的构造方法创建一个子类的对象时，不仅子类中声明的成员变量被分配了内存，而且父类的成员变量也都分配了内存空间，但只将其中一部分（**子类继承的那部分**）作为分配给子类对象的变量。
- 父类中的private成员变量尽管分配了内存空间，也不作为子类对象的变量。
- 如果子类和父类不在同一包中，尽管父类的友好成员变量分配了内存空间，但也不作为子类的成员变量。

子类中还有一部分方法是从父类继承的，
这部分方法却可以操作未被继承的变量。



3 子类对象的特点

```
1. A.java
2. public class A {
3.     private int x;
4.     public void setX(int x) {
5.         this.x=x;
6.     }
7.     public int getX() {
8.         return x;
9.     }
10.}
```

```
11. B.java
```

```
12. public class B extends A {
13.     double y=12;
14.     public void setY(int y)
15.     {
16.         //非法，子类没有继承x
17.         //this.y=y+x;
18.         this.y=y;
19.     }
20.     public double getY() {
21.         return y;
22.     }
23. }
```

```
1. Example5_2.java
2. public class Example5_2 {
3.     public static void main(String args[]) {
4.         B b=new B();
5.         b.setX(888);
6.         System.out.println("子类对象未继承的
x的值是:"+b.getX());
7.         b.y=12.678;
8.         System.out.println("子类对象的实例变
量y的值是:"+b.getY());
9.     }
10.}
```

Problems @ Javadoc Declaration Console X Outline

<terminated> Example5_2 [Java Application] /Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (2021年12月30日下午4:14:10 - 下午4:14:10)

子类对象未继承的x的值是:888

子类对象的实例变量y的值是:12.678

Eclipse演示

■■■ 4 成员变量的隐藏和方法重写

4.1 成员变量的隐藏

- 在编写子类时，如果子类声明的成员变量和从父类继承的成员变量同名（**声明的类型可以不同**），子类就会隐藏掉继承的成员变量。
- 即子类对象以及子类自己定义的方法操作与父类同名的成员变量是指**子类重新声明定义的这个成员变量**。

需要注意的是：子类对象仍然可以调用从父类继承的方法，操作被隐藏的成员变量



4.1 成员变量的隐藏

Goods.java

```
public class Goods {  
    public double weight;  
    public void oldSetWeight(double w) {  
        weight=w;  
        System.out.println("double型的weight="+weight);  
    }  
    public double oldGetPrice() {  
        double price = weight*10;  
        return price;  
    }  
}
```

CheapGoods.java

```
public class CheapGoods extends Goods {  
    public int weight;  
    public void newSetWeight(int w) {  
        weight=w;  
        System.out.println("int型的weight="+weight);  
    }  
    public double newGetPrice() {  
        double price = weight*10;  
        return price;  
    }  
}
```

4.1 成员变量的隐藏

Example5_3.java

```
public class Example5_3 {  
    public static void main(String args[]) {  
        CheapGoods cheapGoods=new CheapGoods();  
        cheapGoods.newSetWeight(198);  
        System.out.println("对象cheapGoods的weight的值是:"+cheapGoods.weight);  
        System.out.println("cheapGoods用子类新增的优惠方法计算价格: "+  
        cheapGoods.newGetPrice());  
        cheapGoods.oldSetWeight(198.987); //子类对象调用继承的方法  
        System.out.println("cheapGoods使用继承的方法(无优惠)计算价格: "+  
        cheapGoods.oldGetPrice());  
    }  
}
```

```
<terminated> Example5_3 [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (2014-4-21 上午07:40:37)  
int型的weight=198  
对象cheapGoods的weight的值是:198  
cheapGoods用子类新增的优惠方法计算价格: 1980.0  
double型的weight=198.987  
cheapGoods使用继承的方法(无优惠)计算价格: 1989.87
```

Eclipse演示

■■■ 4.2 方法重写

1. 重写的语法规则

如果子类可以继承父类的某个实例方法，那么子类就有权利重写这个方法。

方法重写是指：子类中定义一个方法，这个方法和父类的方法具有**相同的方法名、参数列表、返回类型**。



■■■ 4.2 方法重写

2. 重写的目的

子类通过方法的重写可以隐藏继承的方法，把父类的状态和行为改变为自身的状态和行为。

重写方法既可以操作继承的成员变量、继承的方法，也可以操作子类新声明的成员变量和新定义的方法。



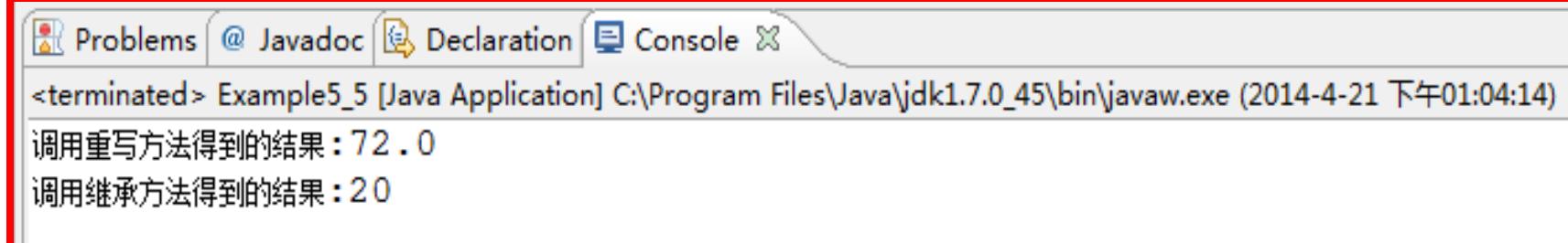
■■■ 4.2 方法重写

Example5_5.java

```
class A {  
    float computer(float x, float y)  
    {  
        return x+y;  
    }  
    public int g(int x, int y) {  
        return x+y;  
    }  
}  
  
class B extends A {  
    float computer(float x, float y)  
    {  
        return x*y;  
    }  
}
```

```
public class Example5_5 {  
    public static void main(String args[]) {  
        B b=new B();  
        double result=b.computer(8, 9);  
        System.out.println("调用重写方法得到  
        的结果:"+result);  
        int m=b.g(12, 8);  
        System.out.println("调用继承方法得到  
        的结果:"+m);  
    }  
}
```

Eclipse演示



■■■ 4.2 方法重写

3. JDK 1.5对重写的改进

在JDK 1.5版本之后，允许重写方法的类型可以是父类方法的类型的子类型，即不必完全一致。

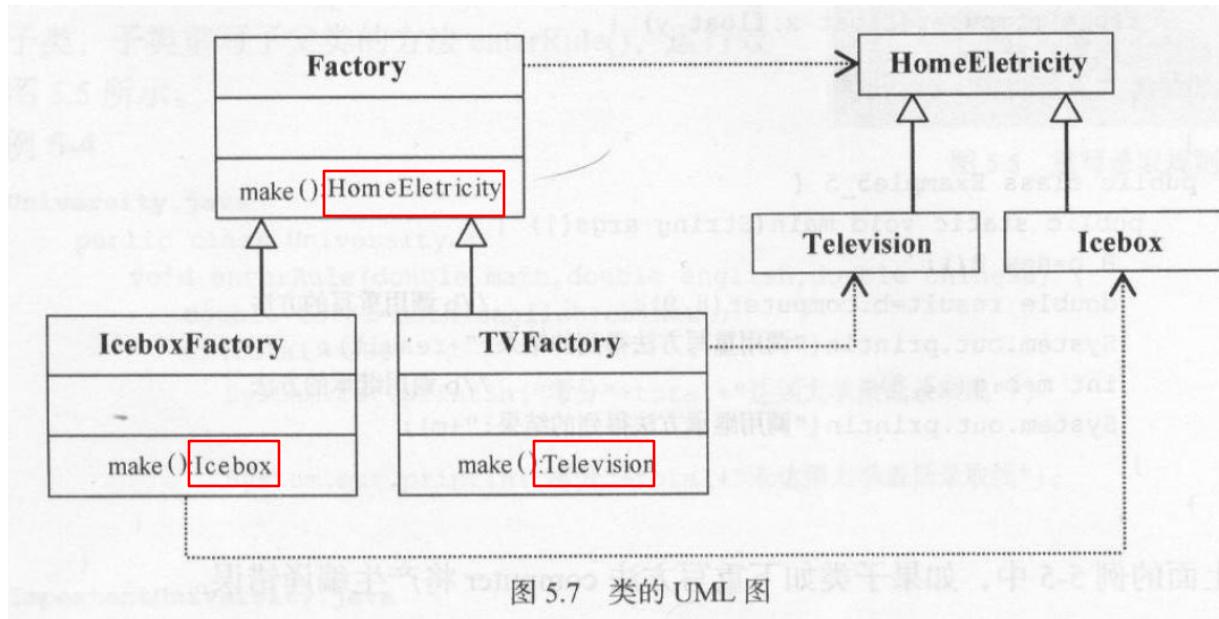


图 5.7 类的 UML 图

■■■ 4.2 方法重写

4. 重写的注意事项

重写父类方法时，不可以降低方法访问权限和继承权限

```
class A {  
    protected float f(float x, float y) {  
        return x-y;  
    }  
}  
  
class B extends A {  
    float f(float x, float y) {  
        return x+y ;  
    }  
}  
  
class C extends A {  
    public float f(float x, float y) {  
        return x*y ;  
    }  
}
```

Eclipse演示



■■■ 5 super关键字

5.1 用super操作被隐藏的成员变量和方法

子类一旦隐藏了继承的成员变量，那么子类创建的对象就不再拥有该变量，该变量将归关键字super所有。

同样子类一旦重写了继承的方法，那么子类创建的对象就不能调用被重写的方法，该方法调用由关键字super负责。

因此，如果在子类中想使用被子类隐藏的成员变量或被重写的方法，可以使用关键字super。



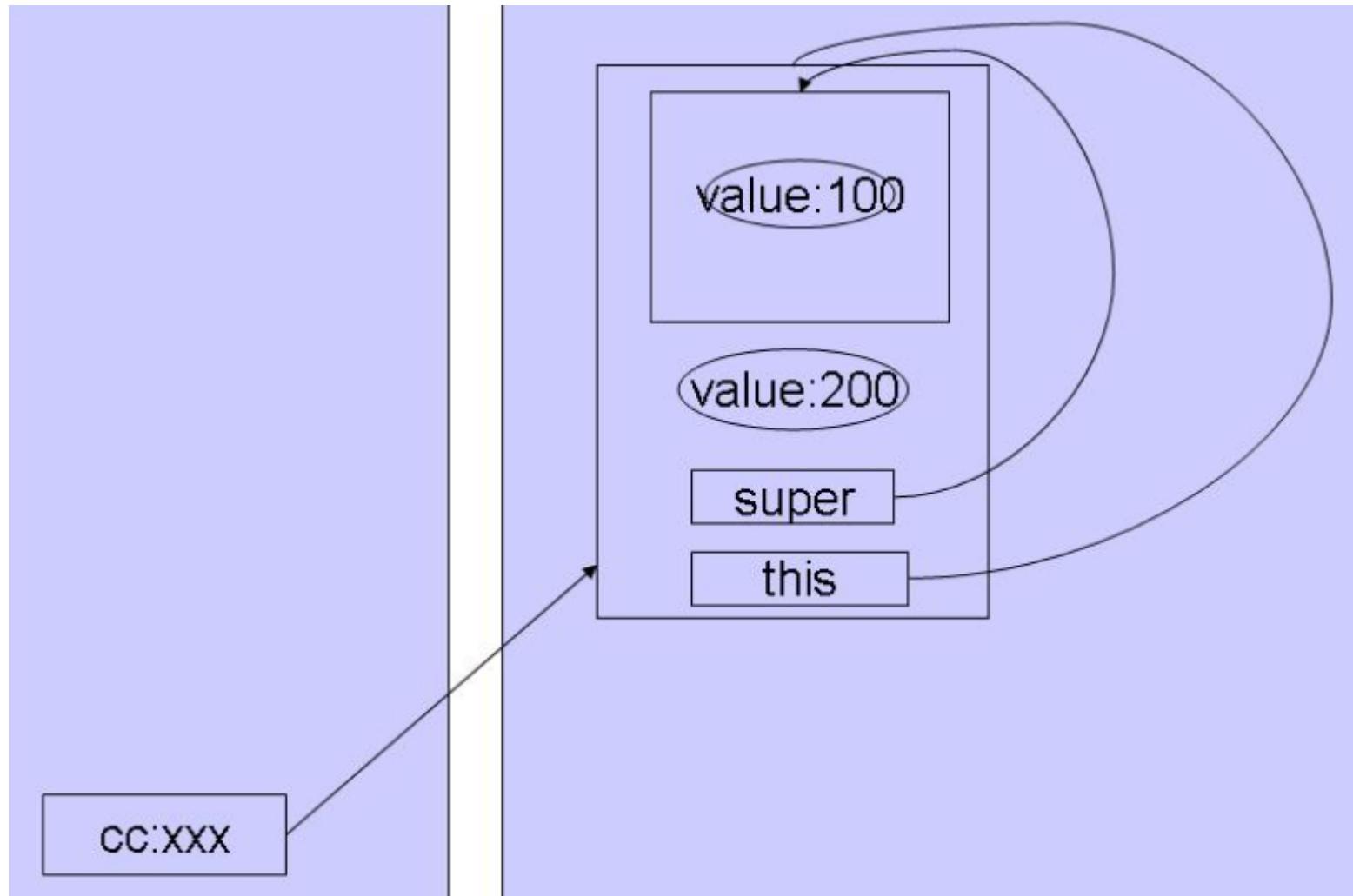
■■■ 5.1 用super操作被隐藏的成员变量和方法

➤ 在Java类中使用**super**来引用基类的成分；例如：

```
class FatherClass {  
    public int value;  
    public void f(){  
        value = 100;  
        System.out.println  
            ("FatherClass.value="+value);  
    }  
}  
class ChildClass extends FatherClass {  
    public int value;  
    public void f() {  
        super.f();  
        value = 200;  
        System.out.println  
            ("ChildClass.value="+value);  
        System.out.println(value);  
        System.out.println(super.value);  
    }  
}
```

运行结果讨论

■■■ 5.1 用super操作被隐藏的成员变量和方法



■■■ 5.2 用super调用父类的构造方法

- ❖ 子类的构造的过程中**必须**调用其基类的构造方法。
- ❖ 子类可以在自己的构造方法中使用**super(argument_list)**调用基类的构造方法。
 - ❖ 使用**this(argument_list)**调用本类的另外的构造方法
 - ❖ 如果调用**super**, 必须写在子类构造方法的第一行
- ❖ 如果子类的构造方法中没有显示地调用基类构造方法, 则系统默认调用基类无参数的构造方法。
- ❖ 如果子类构造方法中既没有显式调用基类构造方法, 而基类中又没有无参的构造方法, 则编译出错。



5.2 用super调用父类的构造方法

```
class Person {  
    private String name;  
    private String location;  
  
    Person(String name) {  
        this.name = name;  
        location = "beijing";  
    }  
    Person(String name, String location) {  
        this.name = name;  
        this.location = location;  
    }  
    public String info() {  
        return  
            "name: "+name+  
            " location: "+location;  
    }  
}
```

```
class Student extends Person {  
    private String school;  
    Student(String name,  
            String school) {  
        this(name,"beijing", school);  
    }  
    Student(String n, String l  
            , String school) {  
        super(n,l);  
        this.school = school;  
    }  
    public String info() {  
        return super.info() +  
            " school: "+school;  
    }  
}
```

■■■ 5.2 用super调用父类的构造方法

```
public class Test {  
    public static void main(String[] args) {  
        Person p1 = new Person("A");  
        Person p2 = new Person("B", "shanghai");  
        Student s1 = new Student("C", "S1");  
        Student s2 =  
            new Student("C", "shanghai", "S2");  
        System.out.println(p1.info());  
        System.out.println(p2.info());  
        System.out.println(s1.info());  
        System.out.println(s2.info());  
    }  
}
```

Eclipse演示（讨论）



6 final关键字

6.1 final类

可以使用final将类声明为final类。final类不能被继承，即不能有子类。如：

```
final class A {  
    ...  
}
```

A就是一个final类，将不允许任何类声明成A的子类

出于安全性的考虑，将一些类修饰为final类。

例如，Java提供的String类，它对于编译器和解释器的正常运行有很重要的作用，对它不能轻易改变，它被修饰为final类。

■■■ 6 final关键字

6.2 final方法

如果用final修饰父类中的一个方法，那么这个方法不允许子类重写。也就是说，不允许子类隐藏可以继承的final方法。

6.3 常量

如果成员变量或局部变量被修饰为final，那么它就是常量。常量在声明时没有默认值，所以在声明常量时必须指定该常量的值，而且不能再发生变化。



■■■ 7 对象的上转型对象

- 假设，A类是B类的父类，当用子类创建一个对象，并把这个对象的引用放到父类的对象中时，如：

```
A a;  
a=new B();
```

或

```
A a;  
B b=new B();  
a=b;
```

称对象a是对象b的上转型对象。

对象的上转型对象的实体是子类负责创建的，但上转型对象会失去原对象的一些属性和功能（上转型对象相当于子类对象的一个“简化”对象）。

7 对象的上转型对象

➤ 上转型对象具有如下特点

- (1) 上转型对象不能操作子类新增成员变量（失掉了一些属性）；
- (2) 不能调用子类新增的方法（失掉了一些功能）。
- (3) 上转型对象可以访问子类继承的成员变量，也可以调用子类继承的方法或子类重写的方法。

如果子类重写了父类的某个实例方法后，当对象的上转型对象调用这个实例方法时一定是调用了子类重写的实例方法！

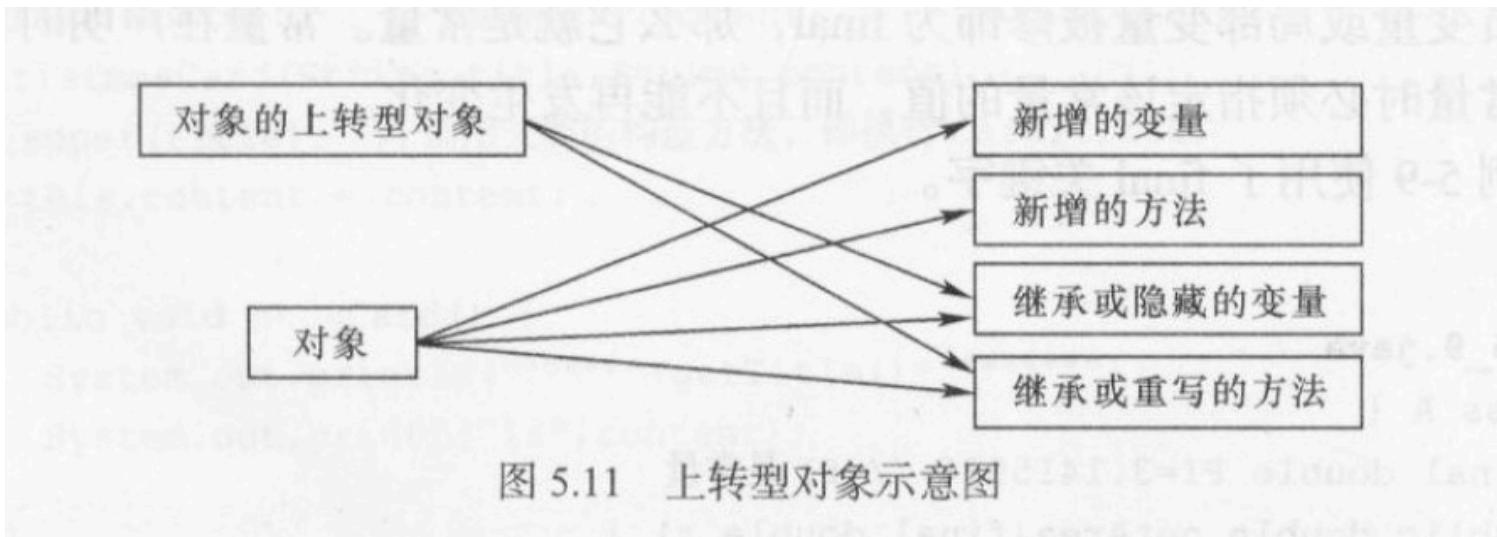


图 5.11 上转型对象示意图

7 对象的上转型对象

```
1. People.java
2. public class People {
3.     int height;
4.     double weight;
5.     void showBodyMess() {
6.         System.out.printf("*****\n");
7.     }
8.     void mustDoingThing() {
9.         System.out.println("吃饭、睡觉... ... 饮水");
10.    }
11. }

12. American.java
13. public class American extends People {
14.     void showBodyMess() {
15.         System.out.println("bodyHeight:"+height+"cm"+
16.             "bodyWeight:"+weight+"kg");
17.     }
18.     void speakEnglish() {
19.         System.out.println("I am Amerian");
20.     }
}
```

```
1. Chinese.java
2. public class Chinese extends People {
3.     void showBodyMess() {
4.         System.out.printf("身高:%5dcm\t体重:%3.2fkg\n",
5.             height, weight);
6.     }
7.     void speakChinese() {
8.         System.out.println("我是中国人");
9.     }
}
```

7 对象的上转型对象

```
1. Example5_10.java
2. public class Example5_10 {
3.     public static void main(String args[]) {
4.         People people=null;
5.
6.         American Johnson = new American();
7.         people = Johnson;
8.         people.height = 187;
9.         people.weight = 78.67;
10.        people.showBodyMess();
11.        //people.speakEnglish();
12.        people.mustDoingThing();
13.
14.        Chinese zhang = new Chinese();
15.        people = zhang;
16.        people.height = 177;
17.        people.weight = 68.59;
18.        people.showBodyMess();
19.        //people.speakChinese();
20.        people.mustDoingThing();
```

```
Problems @ Javadoc Declaration Console
<terminated> Example5_10 [Java Application] C:\Program Files\Java\jdk1.7.0_45\bin\javaw.exe (2014-4-21 下午01:50:16)
bodyHeight:187cm bodyWeight:78.67kg
吃饭、睡觉....饮水
身高: 177cm 体重:68.59kg
吃饭、睡觉....饮水
我是中国人
```

Eclipse演示

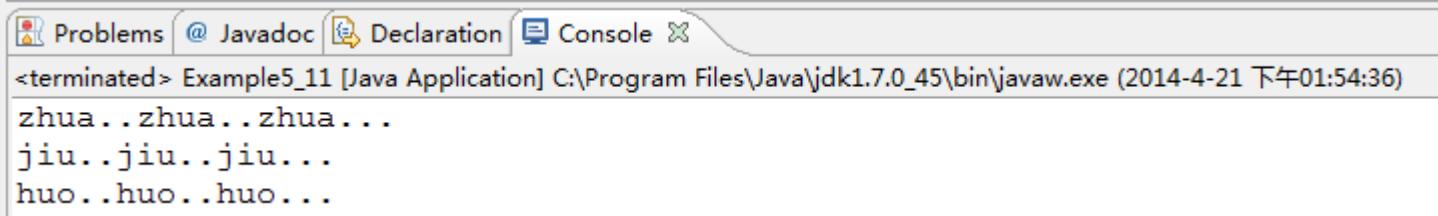
8 继承与多态

- 多态性就是指父类的某个实例方法被其子类重写时，可以各自产生自己的功能行为。

```
class EspecialCar {  
    void cautionSound() {  
    }  
}  
class PoliceCar extends EspecialCar {  
    void cautionSound() {  
        System.out.println("zhua..zhua..zhua...");  
    }  
}  
class AmbulanceCar extends EspecialCar {  
    void cautionSound() {  
        System.out.println("jiu..jiu..jiu...");  
    }  
}  
class FireCar extends EspecialCar {  
    void cautionSound() {  
        System.out.println("huo..huo..huo...");  
    }  
}
```

```
public class Example5_11 {  
    public static void main(String args[]) {  
        //car是警车的上转型对象  
        EspecialCar car=new PoliceCar();  
        car.cautionSound();  
        //car是救护车的上转型对象  
        car=new AmbulanceCar();  
        car.cautionSound();  
        //car是消防车的上转型对象  
        car=new FireCar();  
        car.cautionSound();  
    }  
}
```

Eclipse演示



9 abstract类和abstract方法

- 用关键字abstract修饰的类称为abstract类（抽象类）。如：

```
abstract class A {  
    ...  
}
```

- 用关键字abstract修饰的方法称为abstract方法（抽象方法），对于abstract方法，只允许声明，不允许实现，而且不允许使用final和abstract同时修饰一个方法。

9 abstract类和abstract方法

1. abstract类中可以有abstract方法

和普通的类相比， abstract类可有abstract方法（抽象方法），也可以有非abstract方法。

```
1 package shu.ces.java.test;  
2  
3 public abstract class A {  
4     abstract int min(int x, int y);  
5  
6     int max(int x, int y){  
7         return (x>y?x:y);  
8     }  
9 }  
10
```

Eclipse演示

9 abstract类和abstract方法

2. abstract类的继承关系

- 如果一个非抽象类是某个抽象类的子类，那么它必须重写父类的抽象方法，给出方法体。这就是为什么不允许使用final和abstract同时修饰一个方法的原因。
- 如果一个abstract类是另外一个abstract类的子类，它可以重写父类的abstract方法，也可以继承这个abstract方法。

9 abstract类和abstract方法

```
1. abstract class A {  
2.     abstract int sum(int x, int y);  
3.     int sub(int x, int y) {  
4.         return x-y;  
5.     }  
6. }  
7. class B extends A {  
8.     int sum(int x, int y) {  
9.         return x+y;  
10.    }  
11. }
```

```
1. public class Example5_12 {  
2.     public static void main(String args[]) {  
3.         B b=new B();  
4.         int sum=b.sum(30, 20);  
5.         int sub=b.sub(30, 20);  
6.         System.out.println("sum="+sum);  
7.         System.out.println("sub="+sub);  
8.     }  
9. }
```

Eclipse演示

10 面向抽象编程

- 在设计程序时，经常会使用abstract类，避免设计者把大量的时间和精力花费在具体算法上。
- 例如：在设计地图时，首先考虑地图最重要的轮廓，不必考虑诸如城市中的街道牌号等细节，细节应当由抽象类的非抽象子类去实现，这些子类可以给出具体的实例，来完成程序功能的具体实现。

10 面向抽象编程

➤ 在设计一个程序时，可以通过在abstract类中声明若干个abstract方法，表明这些方法在整个系统设计中的重要性，方法体的内容细节由它的非abstract子类去完成。

➤ 所谓面向抽象编程：是指当设计一个类时，不让该类面向具体类，而是面向抽象类，即所设计类中的重要数据是抽象类声明的对象，而不是具体类声明对象。

➤ 所采用的核心技术之一是使用上转型对象，也就是将abstract类声明对象作为其子类的上转型对象，这个上转型对象就可调用子类重写的方法。

10 面向抽象编程

Pillar.java

```
public class Pillar {  
    Circle bottom;          //bottom是用具体类 Circle 声明的对象  
    double height;  
    Pillar (Circle bottom,double height) {  
        this.bottom=bottom;this.height=height;  
    }  
    public double getVolume() {  
        return bottom.getArea()*height;  
    }  
}
```

Geometry.java

```
public abstract class Geometry {  
    public abstract double getArea();  
}
```

10 面向抽象编程

Pillar.java

```
public class Pillar {  
    Geometry bottom; //bottom是抽象类 Geometry 声明的对象  
    double height;  
    Pillar (Geometry bottom,double height) {  
        this.bottom=bottom; this.height=height;  
    }  
    public double getVolume() {  
        return bottom.getArea()*height; //bottom可以调用子类重写的 getArea 方法  
    }  
}
```

Eclipse演示

Circle.java

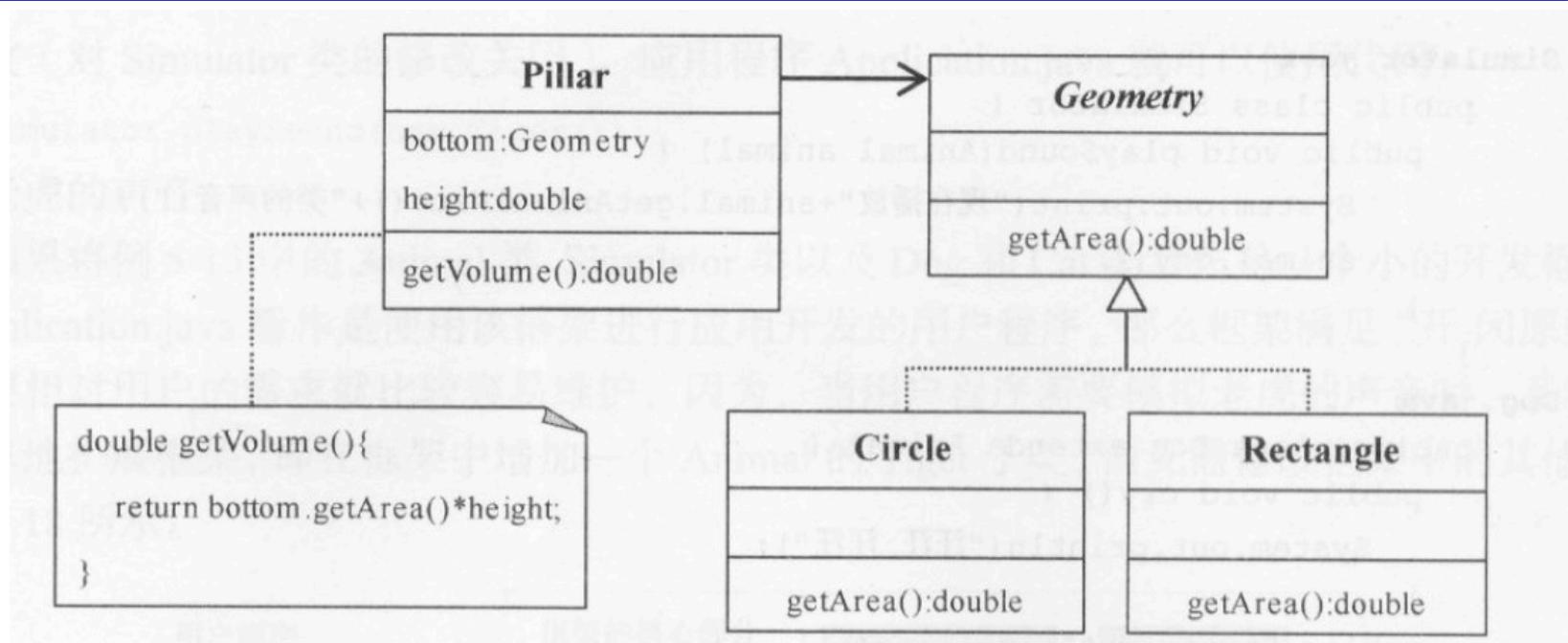
```
public class Circle extends Geometry {  
    double r;  
    Circle(double r) {  
        this.r=r;  
    }  
    public double getArea() {  
        return(3.14*r*r);  
    }  
}
```

Rectangle.java

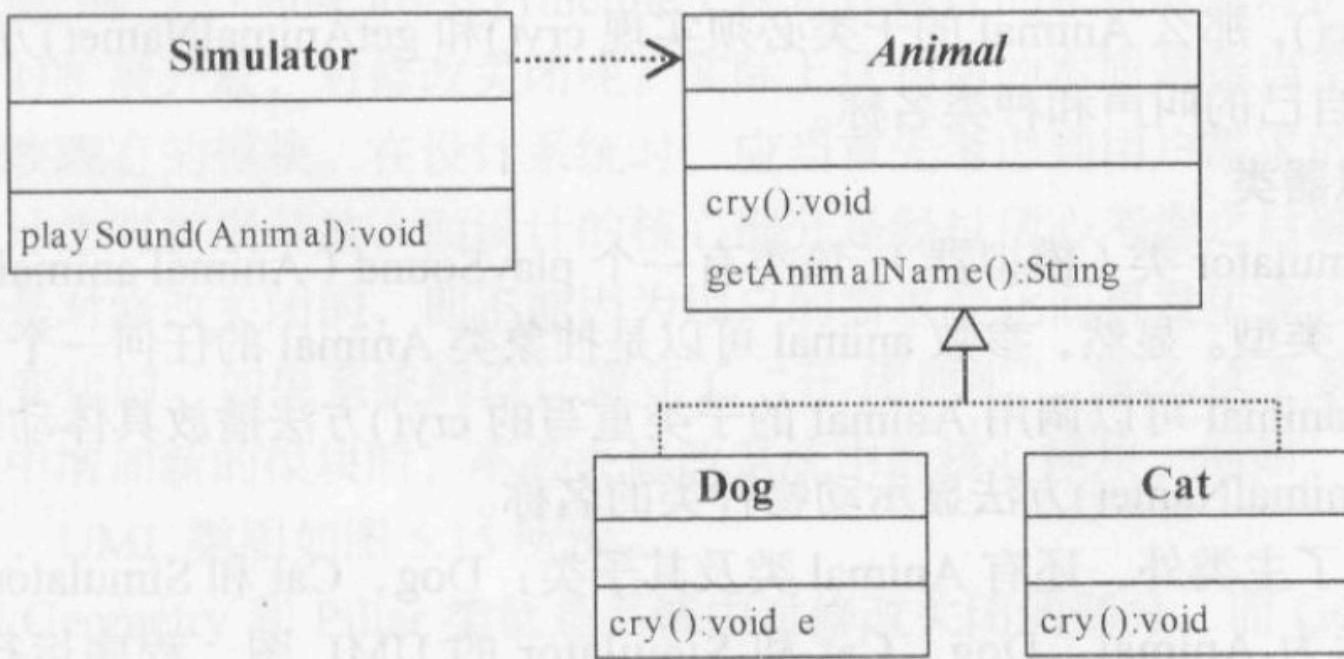
```
public class Rectangle extends Geometry {  
    double a,b;  
    Rectangle(double a,double b) {  
        this.a=a;  
        this.b=b;  
    }  
    public double getArea() {  
        return a*b;  
    }  
}
```

11 开-闭原则

所谓“开-闭原则”（Open-Closed Principle）就是让设计的系统应当对扩展开放，对修改关闭。即：当系统中增加新的模块时，不需要修改现有的模块。



11 开-闭原则



11 开-闭原则





第5章 子类与继承

- 1. 子类与父类
 - 2. 子类的继承性
 - 3. 子类对象的特点
 - 4. 成员变量的隐藏和方法重写
 - 5. super关键字
 - 6. final关键字
 - 7. 对象的上转型对象
 - 8. 继承与多态
 - 9. abstract类与abstract方法
 - 10. 面向抽象编程
 - 11. 开-闭原则
- 