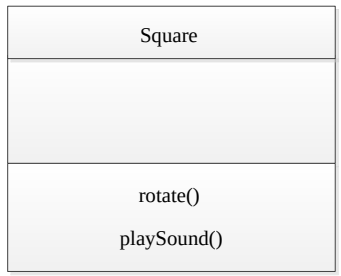
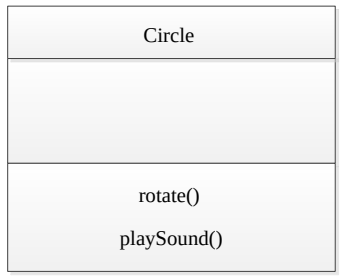
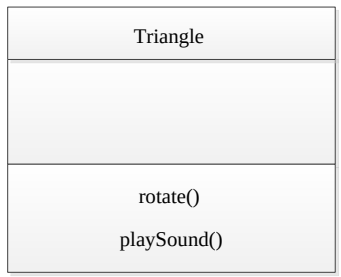
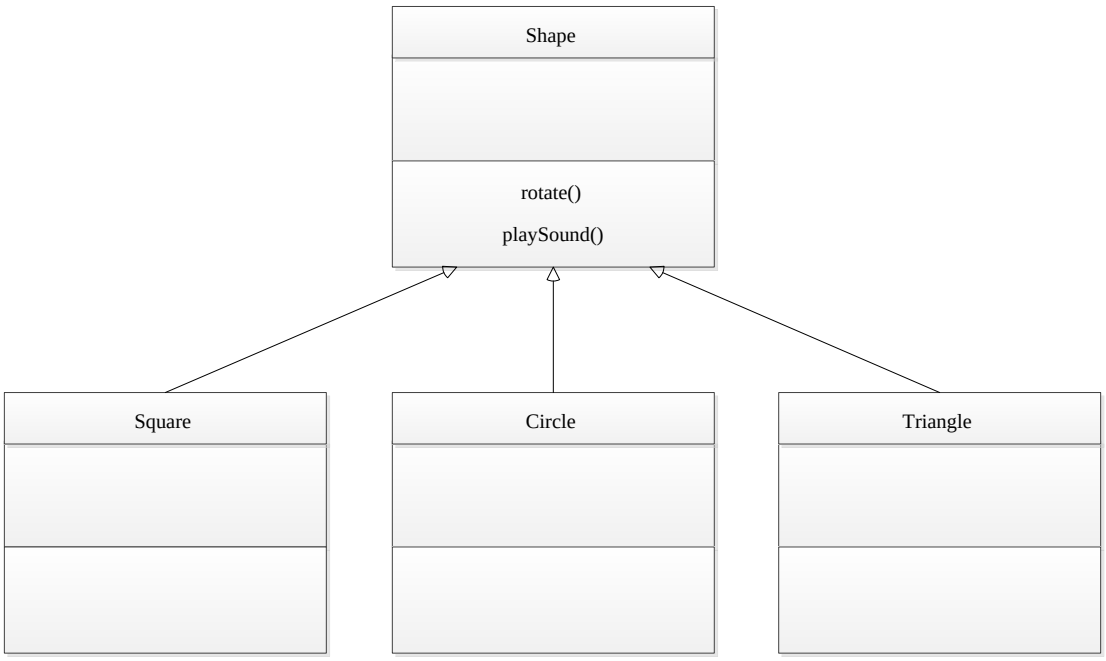
# 第 5 章 子类与继承

## 5.1 子类与父类



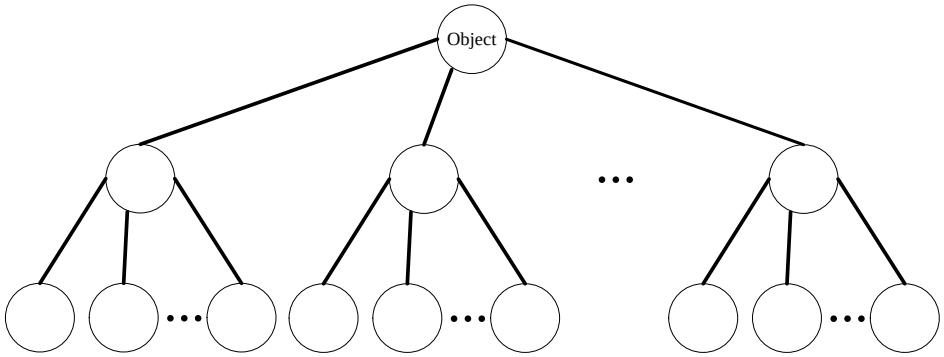
1. 可以在旧类（父类、超类）的基础上声明新类（子类）。

2. 一个父类可以有多个子类，但一个子类只可以有一个父类，不支持多重继承（与 C++不同）。

3. 子类声明

```
class 子类名 extends 父类名 {
    子类体
}
```

4. 类的树形结构




祖先类、子孙（后代）类

5. 声明类时，若缺省“extends 父类名”，则默认为“extends Object”。

6. Object 类在 java.lang 包中，是所有其他类的老祖宗。

## 5.2 子类的继承性

1. 子类和父类（祖先类）在同一个包中

子类继承父类（祖先类）的访问权限为共有、受保护和友好的成员变量和方法。

p.114 代码 1

```
public class People {
    int age,leg = 2,hand = 2;
    protected void showPeopleMess() {
       System.out.printf("%d 岁,%d 只脚,%d 只手\t",age,leg,hand);
    }
}
```

p.114 代码 2

```
public class Student extends People {
   int number;
   void tellNumber() {
       System.out.printf("学号:%d\t",number);
   }
   int add(int x,int y) {
      return x+y;
   }
}
```

p.114 代码 3

```
public class UniverStudent extends Student {
   int multi(int x,int y) {
      return x*y;
   }
}
```

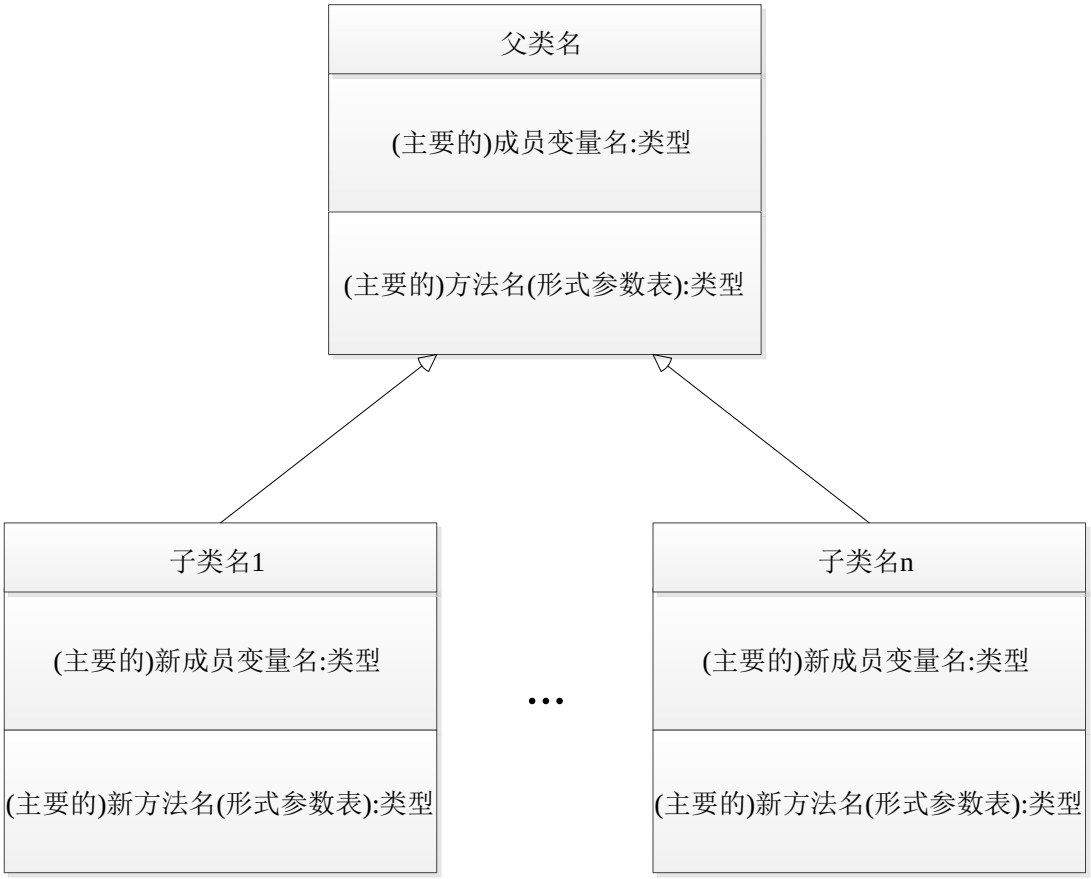
pp.114-115 代码 4

```
public class Example5_1 {
   public static void main(String args[]) {
      Student zhang = new Student();
      zhang.age = 17;              //访问继承的成员变量
      zhang.number=100101;
      zhang.showPeopleMess();      //调用继承的方法
      zhang.tellNumber();
      int x=9,y=29;
      System.out.print("会做加法:");
      int result=zhang.add(x,y);
      System.out.printf("%d+%d=%d\n",x,y,result);
      UniverStudent geng = new UniverStudent();
      geng.age = 21;               //访问继承的成员变量
      geng.number=6609;            //访问继承的成员变量
      geng.showPeopleMess();       //调用继承的方法
      geng.tellNumber();           //调用继承的方法
      System.out.print("会做加法:");
      result=geng.add(x,y);        //调用继承的方法
      System.out.printf("%d+%d=%d\t",x,y,result);
      System.out.print("会做乘法:");
      result=geng.multi(x,y);
      System.out.printf("%d×%d=%d\n",x,y,result);
   }
}
```

2. 子类和父类（祖先类）不在同一个包中

   子类继承父类（祖先类）的访问权限为共有和受保护的成员变量和方法。

3. 继承关系的 UML 图




4. 访问限制符总结

| | 当前类 | 同一包 | 子孙类 | 其他包 |
|---|---|---|---|---|
| 共有 | √ | √ | √ | √ |
| 受保护 | √ | √ | √ | × |
| 友好 | √ | √ | × | × |
| 私有 | √ | × | × | × |

## 5.3 子类与对象

1. 子类对象的特点

（1）子类实际上还有父类（祖先类）的所有成员变量，但子类对象不可以直接使用子类未能继承的父类（祖先类）的成员变量。

（2）子类对象调用子类所继承的父类（祖先类）的方法时，可以使用子类未能继承的父类（祖先类）的成员变量。（这就是子类要有子类未能继承的父类（祖先类）的成员变量的原因）

p. 116 代码 1

```
class People {
    private int averHeight = 166;
    public int getAverHeight() {
       return averHeight;
    }
}
class ChinaPeople extends People {
     int height;
     public void setHeight(int h) {
         //height = h+averHeight;    //非法，子类没有继承 averHeight
         height = h;
     }
     public int getHeight() {
        return height;
     }
}
public class Example5_2 {
  public static void main(String args[]) {
      ChinaPeople zhangSan = new ChinaPeople();
      System.out.println("子类对象未继承的 averageHeight 的值是:"+zhangSan.
          getAverHeight());
      zhangSan.setHeight(178);
      System.out.println("子类对象的实例变量 height 的值是:"+zhangSan.getHeight());
  }
}
```

2. instanceof 运算符

对象名 instanceof 类名

若该对象名所指对象是该类或其子孙类的，则该表达式的值为 true，否则为 false。

## 5.4 成员变量隐藏和方法重写

1. 成员变量隐藏

（1）若子类的某成员变量和子类所继承的父类（祖先类）的某成员变量重名，则子类隐藏父类（祖先类）的该成员变量。

（2）子类对象调用子类所继承的父类（祖先类）的方法时，可以使用被子类隐藏的父类（祖先类）的成员变量。

pp. 117-118 代码 1

```
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;
    }
}
```

p. 118 代码 2

```
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;
    }
}
```

p. 118 代码 3

```
public class Example5_3 {
  public static void main(String args[]) {
     CheapGoods cheapGoods = new CheapGoods();
     //cheapGoods.weight=198.98; 是非法的，因为子类对象的 weight 变量已经是 int 型
     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操作隐藏的 double
                                       //型变量 weight
     System.out.println("cheapGoods 使用继承的方法（无优惠）计算价格："+
                         cheapGoods.oldGetPrice());
  }
}
```

2. 方法重写

（1）子类可以重写子类所继承的父类（祖先类）的方法，重写方法的返回值类型必须是原方法的返回值类型或其子类型。

（2）子类可以通过方法重写隐藏子类所继承的父类（祖先类）的方法。

（3）重写方法不可以直接使用被子类隐藏的父类（祖先类）的成员变量和直接调用被子类隐藏的父类（祖先类）的方法。

（4）重写方法的访问权限可以升但不可以降。

访问权限：共有>受保护>友好>私有

p. 119 代码 1

```
public class University {
    void enterRule(double math,double english,double chinese) {
        double total = math+english+chinese;
        if(total >= 180)
          System.out.println(total+"分数达到大学录取线");
        else
          System.out.println(total+"分数未达到大学录取线");
    }
}
```

p. 119 代码 2

```
public class ImportantUniversity extends University{
    void enterRule(double math,double english,double chinese) {
        double total = math+english+chinese;
        if(total >= 220)
          System.out.println(total+"分数达到重点大学录取线");
        else
          System.out.println(total+"分数未达到重点大学录取线");
    }
}
```

p. 120 代码 1

```
public class Example5_4 {
    public static void main(String args[]) {
       double math = 62,english = 76.5,chinese = 67;
       ImportantUniversity univer = new ImportantUniversity();
       univer.enterRule(math,english,chinese); //调用重写的方法
       math = 91;
       english = 82;
       chinese = 86;
       univer.enterRule(math,english,chinese); //调用重写的方法
    }
}
```

p. 120 代码 2

```
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);      //b 调用重写的方法
      System.out.println(result);
      int m=b.g(12,8);                    //b 调用继承的方法
      System.out.println(m);
    }
}
```

p. 122 代码 2

```
class A {
   protected  float f(float x,float y) {
       return x-y;
    }
}
class B extends A {
   float f(float x,float y) {          //非法，因为降低了访问权限
       return x+y ;
    }
}
class C extends A {
   public float f(float x,float y) {  //合法，提高了访问权限
       return x*y ;
    }
}
```

## 5.5 super 关键字

1. 用于成员变量和方法

子类对象可以通过 super 关键字，使用被子类隐藏的父类（祖先类）的成员变量和调用被子类隐藏的父类（祖先类）的方法。

pp. 122-123 代码 3

```
class Sum {
```

```
    int n;
    float f() {
       float sum = 0;
       for(int i=1;i<=n;i++)
          sum = sum+i;
       return sum;
    }
}
class Average extends Sum {
    int n;
    float f() {
      float c;
      super.n = n;
      c = super.f();
      return c/n;
    }
   float g() {
       float c;
       c = super.f();
       return c/2;
   }
}
```

```
public class Example5_7 {
   public static void main(String args[]) {
      Average aver = new Average();
      aver.n = 100;
      float resultOne = aver.f();
      float resultTwo = aver.g();
      System.out.println("resultOne="+resultOne);
      System.out.println("resultTwo="+resultTwo);
   }
}
```

2. 用于构造方法

（1）子类不能继承父类的构造方法。

（2）在子类的构造方法的方法体开头，要通过 super 关键字调用父类的构造方法，若缺省，则隐含“super();”。因而，若父类定义构造方法，则最好要定义一个无参构造方法。

p. 124 代码 1

```
class Student {
   int number;String name;
   Student() {
   }
   Student(int number,String name) {
      this.number = number;
      this.name = name;
      System.out.println("我的名字是:"+name+ "学号是:"+number);
   }
 }
class UniverStudent extends Student {
   boolean 婚否;
   UniverStudent(int number,String name,boolean b) {
      super(number,name);
      婚否 = b;
      System.out.println("婚否="+婚否);
   }
}
public class Example5_8 {
   public static void main(String args[]) {
      UniverStudent zhang = new UniverStudent(9901,"何晓林",false);
   }
}
```

## 5.6 final 关键字

1. 用于类

使用 final 关键字修饰的类无子类。

2. 用于方法

父类的使用 final 关键字修饰的方法，子类（子孙类）不可以重写。

3. 用于成员变量和局部变量

使用 final 关键字修饰的成员变量和局部变量实际上就是符号常量，在声明的同时要赋值（形式参数除外）。

pp.125-126 代码 1

```
class A {
  final double PI=3.1415926;// PI 是常量
  public double getArea(final double r) {
     //r = r+1; //非法，不允许对 final 变量进行更新操作
     return PI*r*r;
  }
  public final void speak() {
     System.out.println("您好，How's everything here ?");
  }
}
public class Example5_9 {
```

```
    public static void main(String args[]) {
        A a=new A();
        System.out.println("面积: "+a.getArea(100));
        a.speak();
    }
}
```

## 5.7 对象的上转型对象

1. 设 B 类是 A 类的子类（子孙类）。若声明 A 类的对象名，并赋予所引入的 B 类的对象的引用，则称 A 类的对象名所指对象为 B 类的对象的上转型对象。

2. B 类的对象和它的上转型对象实际上是同一个对象，但场合不同所能使用的成员变量和所能调用的方法有所不同。

3. B 类的对象的上转型对象不可以使用 B 类的新成员变量和调用 B 类的新方法。

4. B 类的对象的上转型对象可以使用 B 类所继承（包括所隐藏）的 A 类的成员变量。

5. B 类的对象的上转型对象可以调用 B 类未重写的 B 类所继承的 A 类的方法，以及 B 类重写的 B 类所继承的 A 类的实例方法（为了多态），但不可以调用 B 类重写的 B 类所继承的 A 类的类方法（可以调用 A 类的原类方法）。

p. 127 代码 1

```
class 类人猿 {
    void crySpeak(String s) {
        System.out.println(s);
    }
}
class People extends 类人猿 {
    void computer(int a,int b) {
        int c=a*b;
        System.out.println(c);
    }
    void crySpeak(String s) {
        System.out.println("***"+s+"***");
    }
}
public class Example5_10 {
    public static void main(String args[]) {
        类人猿 monkey;
        People geng = new People();
        monkey = geng ; //monkey 是 People 对象 geng 的上转型对象
        monkey.crySpeak("I love this game");
//等同于 geng.crySpeak("I love this game");
        People people=(People)monkey; //把上转型对象强制转化为子类的对象
        people.computer(10,10);
    }
}
```

## 5.8 继承与多态

当父类的一个实例方法被多个子类（子孙类）重写后，可以借助上转型对象，让父类的这个实例方法产生不同的行为。这称为多态。

p.128 代码 1

```
class 动物 {
   void cry() {
   }
}
class 狗 extends 动物 {
   void cry() {
      System.out.println("汪汪.....");
   }
}
class 猫 extends 动物 {
   void cry() {
      System.out.println("喵喵.....");
   }
}
public class Example5_11 {
   public static void main(String args[]) {
      动物 animal;
      animal = new 狗();
      animal.cry();
      animal=new 猫();
      animal.cry();
   }
}
```

## 5.9 抽象类和抽象方法

1. 使用 abstract 关键字修饰的类和方法分别称为抽象类和抽象方法。

2. 抽象类可以有抽象方法和非抽象方法，而非抽象类只可以有非抽象方法。

3. 抽象方法无“{方法体}”。

4. 不可以使用 final 关键字修饰抽象类和抽象方法。

5. 抽象方法不可以是类方法。

6. 可以声明抽象类的对象名，但不可以引入抽象类的对象。

pp.130-131 代码 1

```
abstract class GirlFriend {  //抽象类，封装了两个行为标准
   abstract void speak();
   abstract void cooking();
}
class ChinaGirlFriend extends GirlFriend {
   void speak(){
      System.out.println("你好");
   }
   void cooking(){
      System.out.println("水煮鱼");
   }
}
class AmericanGirlFriend extends GirlFriend {
   void speak(){
      System.out.println("hello");
   }
   void cooking(){
      System.out.println("roast beef");
   }
}
class Boy {
   GirlFriend friend;
   void setGirlfriend(GirlFriend f){
       friend = f;
   }
   void showGirlFriend() {
      friend.speak();
      friend.cooking();
   }
}
public class Example5_12 {
   public static void main(String args[]) {
      GirlFriend girl = new ChinaGirlFriend(); //girl 是上转型对象
      Boy boy = new Boy();
      boy.setGirlfriend(girl);
      boy.showGirlFriend();
      girl = new AmericanGirlFriend(); //girl 是上转型对象
      boy.setGirlfriend(girl);
      boy.showGirlFriend();
   }
}
```

## 5.10 面向抽象编程

p. 134 倒数第 3 段

p. 132 代码 3

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

p. 133 代码 2

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

p. 133 代码 3

```
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;
    }
```

p. 133 代码 1

```
public class Pillar {
    Geometry  bottom;        //bottom 是抽象类 Geometry 声明的变量
    double height;
    Pillar (Geometry bottom,double height) {
        this.bottom=bottom; this.height=height;
    }
    public double getVolume() {
        if(bottom==null) {
          System.out.println("没有底,无法计算体积");
          return -1;
        }
        return bottom.getArea()*height; //bottom 可以调用子类重写的 getArea 方法
    }
}
```

p. 134 代码 1

```
public class Application{
    public static void main(String args[]){
        Pillar pillar;
        Geometry bottom =null;
        pillar =new Pillar (bottom,100);  //null 底的柱体
        System.out.println("体积"+pillar.getVolume());
        bottom=new Rectangle(12,22);
        pillar =new Pillar (bottom,58);  //pillar 是具有矩形底的柱体
        System.out.println("体积"+pillar.getVolume());
        bottom=new Circle(10);
        pillar =new Pillar (bottom,58); //pillar 是具有圆形底的柱体
        System.out.println("体积"+pillar.getVolume());
    }
}
```

## 5.11 开-闭原则

软件实体应当对扩展开放,对修改关闭。(Software entities (modules, classes, functions, etc.) should be open for extension, but closed for modification.)