计算机科学技术学院
School of Computer Science and Technology

Java 程序设计

第 5 章　子类与继承

导读

主要内容

- 子类与父类、子类的继承性、子类与对象
- 成员变量的隐藏和方法重写
- super 关键字、 final 关键字
- 对象的上转型对象、继承与多态
- abstract 类与 abstract 方法
- 面向抽象编程、开 - 闭原则

重点和难点

- 重点：类的继承性、上转型对象和多态技术
- 难点：理解上转型对象和多态技术的理解和运用

5.1 子类与父类

- 继承是一种由已有的类创建新类的机制。利用继承，可以先创建一个共有属性的一般类，根据该一般类再创建具有特殊属性的新类，新类继承一般类的状态和行为，并根据需要增加新的状态和行为
- 继承得到的类称为子类，被继承的类称为父类（超类）

Java 不支持多重继承（子类只能有一个父类）

5.1.1 声明子类

- 使用关键字 extends 来定义一个类的子类，格式如下：

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

- 例如：

```
class Student extends People {
        ...
}
```

5.1.2 类的树形结构

- Java 的类按继承关系形成树形结构这个树形结构中，根节点是 Object 类（ Object 是 java.lang 包中的类），即 Object 是所有类的祖先类。
- 除了 Object 类，每个类都有且仅有一个父类，一个类可以有多个或零个子类。如果一个类（除了 Object 类）的声明中没有使用 extends 关键字，这个类被系统默认为是 Object 的子类，即类声明“ class A”与“ class A extends Object”是等同的。

5.2 子类的继承性

- 类可以有两种重要的成员：成员变量和方法。子类的成员中有一部分是子类**自己声明定义**的，另一部分是从它的**父类继承**的
  - 所谓子类继承父类的成员变量就是把继承来的变量作为自己的一个成员变量
  - 所谓子类继承父类的方法就是把继承来的方法作为子类中的一个方法

5.2 子类的继承性

```
class Father {
float weight,height;
    String head;
    void speak(String s) {

System.out.println(s);
    }
}
```

```
class Son extends Father
 {
    String hand,foot;
}

Son s=new Son();
```

5.2.1 子类和父类在同一包中的继承性

- 如果子类和父类在同一个包中，那么，子类自然地继承了其父类中不是 private 的成员变量作为自己的成员变量，并且也自然地继承了父类中不是 private 的方法作为自己的方法，继承的成员变量或方法的访问权限保持不变

5.2.1 子类和父类在同一包中的继承性

- 如果子类和父类不在同一个包中，那么，子类继承了父类的 protected 、 public 成员做为子类的成员，继承的成员访问权限保持不变

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

```
public class Student extends People {
   int number;
   void tellMessage() {
      showPeopleMess();
   }
}
```

5.2.3 继承关系的 UML 图

- UML 通过使用一个实线连接两个类的 UML 图来表示二者之间的继承关系，实线的起始端是子类的 UML 图，终点端是父类的 UML 图，但终点端使用一个**空心的三角形**表示实线的结束

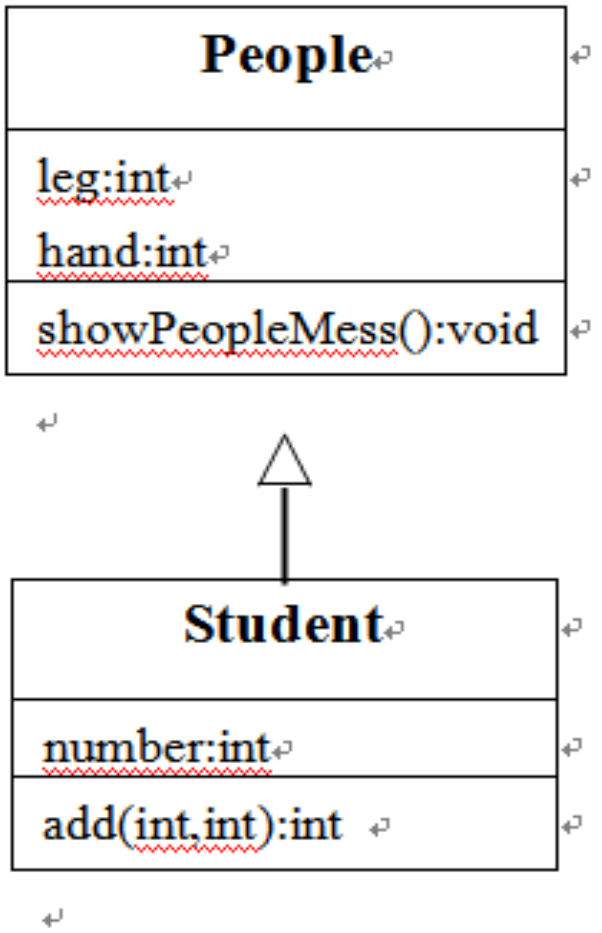


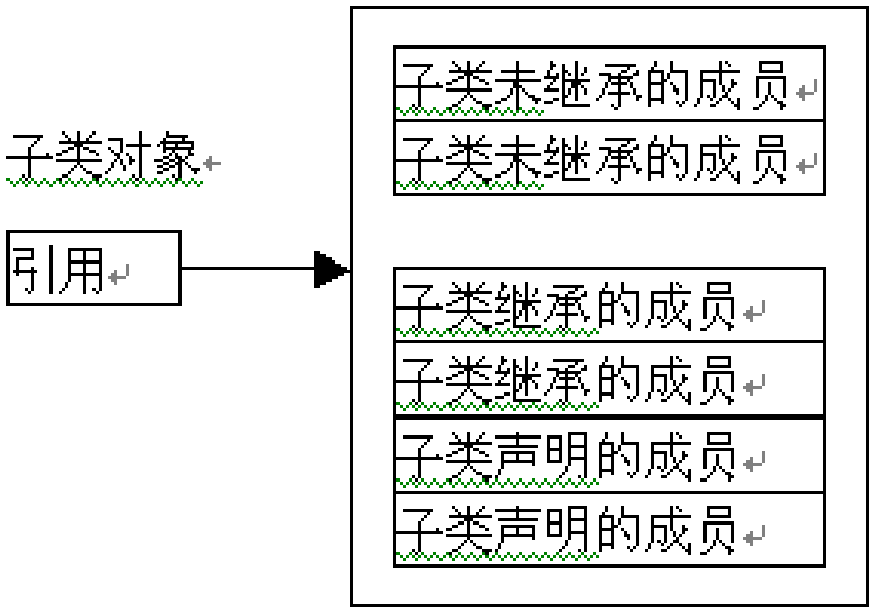
图 5.2 继承关系的 UML 图

5.3 子类与对象

- 类继承了父类的很多东西，那么子类在创建对象的时候，是怎么生成自己的对象的呢。子类生产的对象会有哪些东西呢。

```
例题：子类的继承
class Father
{  float weight,height;
   String head;
   void speak(String s)
   {  System.out.println(s);
   }
}
class Son extends Father
{   String hand,foot;
}

   Son s=new Son();
```



5.3.1 子类对象的生成

- 子类创建对象时，先调用父类的某个构造方法，完成父类部分的创建；再调用子类自己的构造方法
- 子类的构造方法默认调用父类的无参构造方法
- 创建子类对象时，子类中声明的成员变量被分配了内存，父类的所有的成员变量也都分配了内存空间，但子类只能操作继承的那部分成员变量
- 子类通过继承的方法操作未继承的父类变量和方法

5.3.1 子类对象的生成

```
class Base {
    private int age = 20;
    String name = "张三";
    int getAge() {
        return age;
    }
}
```

```
class Sub extends Base {
    void showName() {
        System.out.println(name);
    }
    void showAge() {
        System.out.println(getAge());
    }
}
```

5.3.1 子类对象的生成

结论：

① 创建子类对象时，子类总是按层次结构从上到下的顺序调用所有超类的构造函数。如果继承和组合联用，要先构造基类的构造函数，然后调用组合对象的构造函数（组合按照声明的顺序调用）

② 如果父类没有不带参数的构造方法，则在子类的构造方法中必须明确的告诉调用父类的某个带参数的构造方法，通过 super 关键字，这条语句还必须出现在构造方法的第一句

5.3.2 关于 instanceof 运算符

- instanceof 运算符是 Java 独有的双目运算符，其左面的操作元是对象，右面的操作元是类，当左面的操作元是右面的类或其子类所创建的对象时， instanceof 运算的结果是 true ，否则是 false

```
class Test
{   String s;
    Test()
    {   stu  s=new String();
        if(s instanceof String)
        { System.out.println("YES");
         }
    }
}
```

5.4 成员变量的隐藏和方法重写

5.4.1 成员变量的隐藏

- 对于子类可以从父类继承的成员变量，只要子类中声明的成员变量和父类中的成员变量同名时，子类就隐藏了继承的成员变量
- 在子类中要操作这个与父类同名的成员变量时，子类操作的是子类重新声明的这个成员变量。而不是被隐藏掉的

5.4 成员变量的隐藏和方法重写

```
class Person {
    String name = " 张三 ";
    Person() {}
    Person(String name) {
        name = name;
    }
    public String toString() {
        return name;
    }
}
```

5.4.2 方法重写（Override）

子类通过重写可以隐藏已继承的实例方法

1．重写的语法规则

- 如果子类继承了父类的实例方法，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，这个方法的类型和父类的方法的类型一致或是父类方法的类型的子类型，且这个方法的名字、参数个数、参数的类型和父类的方法**完全相同**

5.4.2 方法重写（Override）

2．重写的目的

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

3. 重写后方法的调用

- 子类创建的一个对象，如果子类重写了父类的方法，则运行时系统调用的是子类重写的方法；
- 子类创建的一个对象，如果子类未重写父类的方法，则运行时系统调用的是子类继承的方法；

5.4.2 方法重写（Override）

4. 重写的注意事项

- 重写父类的方法时，**不允许降低**方法的访问权限，但**可以提高访问权限**（访问限制修饰符按访问权限从高到低的排列顺序是：public、protected、private）

例题

```
class A {
    void show() {
        System.out.println("This is A");
    }
}
class B extends A {
    void show() {
        System.out.println("This is B");
    }
}
```

5.5 super 关键字

① 用 super 操作被隐藏的成员变量和方法

子类可以隐藏从父类继承的成员变量和方法，如果在子类中想使用被子类隐藏的成员变量或方法就可以使用关键字 super 。比如 **super.x** 、 **super.play()**

② 使用 super 调用父类的构造方法

子类如果想使用父类的构造方法，必须在子类的构造方法中使用，并且必须使用关键字 super 来表示，而且 super 必须是子类构造方法中的头一条语句

5.6 final 关键字

- final 关键字可以修饰类、成员变量和方法中的局部变量
  - 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
  - 如果成员变量或局部变量被修饰为 final，就是常量

5.7 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，称对象 a 是对象 b 的上转型对象

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

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

5.7 对象的上转型对象

上转型对象的使用

① 上转型对象不能操作/调用子类**新增的成员**变量和方法

② 上转型对象可以访问子类**继承或隐藏的成员变量**，也可调用子类**继承的方法或子类重写的实例方法**

③ 如果子类重写了父类的某个实例方法后，当用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**

5.7 对象的上转型对象

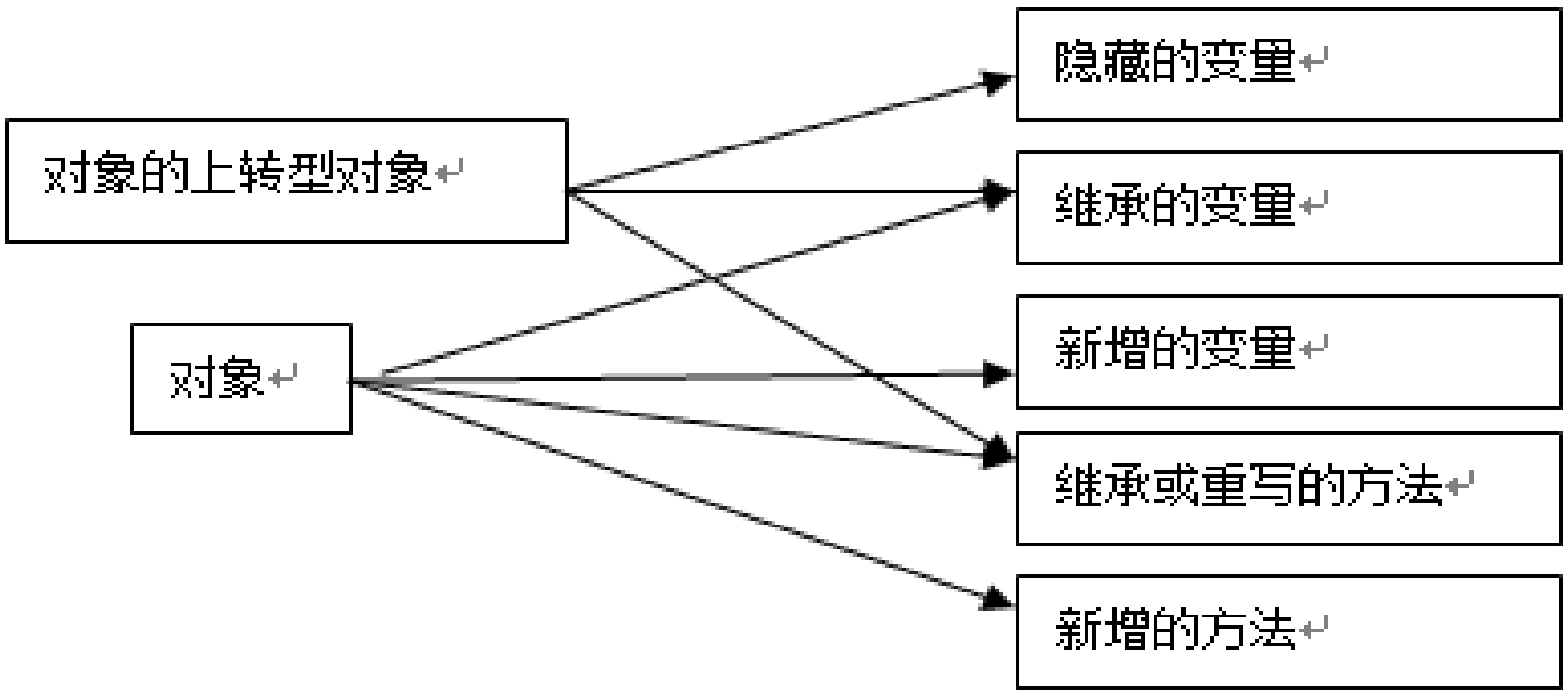


图 5.9 上转型对象示意图

5.7 对象的上转型对象

```
class A {
    void show() {
        System.out.println("This is A");
    }
    void showA() {
        System.out.println("This is A_1");
    }
}
                class B extends A {
                    void show() {
                        System.out.println("This is B");
                    }
                    void show1() {
                        System.out.println("This is B_1");
                    }
                }
```

5.8 继承与多态

- 多态性就是指父类的某个方法被其子类重写时，可以各自产生自己的功能行为

5.9 abstract 类和 abstract() 方法

- 用关键字 abstract 修饰的类称为 **abstract 类** (抽象类)

```
abstract class A {
                    … …
}
```

- 用关键字 abstract 修饰的方法称为 **abstract 方法** (抽象方法)

例如： `abstract int min(int x,int y);`

5.9 abstract 类和 abstract() 方法

- abstract 类特点
  - 和普通的类相比， abstract 类里可以有 abstract 方法。也可以没有。对于 abstract 方法，只允许声明，不允许实现，而且不允许使用 final 修饰 abstract 方法
  - 对于 abstract 类，不能使用 new 运算符创建该类的对象，只能产生其子类，由子类创建对象
  - 如果一个类是 abstract 类的子类，它必须具体实现父类的所有的 abstract 方法

理解抽象类

理解的关键点是：

①抽象类可以抽象出重要的**行为标准**，该行为标准用**抽象方法**来表示。即抽象类封装了子类必需要有的行为标准。

②抽象类声明的对象可以成为其子类的对象的**上转型对象**，调用子类重写的方法，即体现子类根据抽象类里的行为标准给出的**具体行为**。

开发者可以把主要精力放在一个应用中需要那些行为标准（不用关心行为的细节），不仅节省时间，而且非常有利于设计出易维护、易扩展的程序

5.10 面向抽象编程

- 设计程序时，可以先声明一个 abstract 类，声明若干 abstract 方法，方法体的内容细节由它的非 abstract 子类去完成
- 然后利用多态实现编程。使用多态进行程序设计的核心技术是使用方法重写和上转型对象，即将 abstract 类声明对象作为其子类的上转型对象，那么这个上转型对象就可以调用子类重写的方法
- 所谓面向抽象编程，是指当设计某种重要的类时，不让该类面向具体的类，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，而不是具体类声明的对象。

5.11 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。
- 在设计系统时，用户需求的变化设计为对扩展开放，设计的核心部分是经过精心考虑之后确定下来的基本结构，这部分应当是对修改关闭的，不必去修改系统中的核心模块。

5.11 开－闭原则

开 - 闭原则

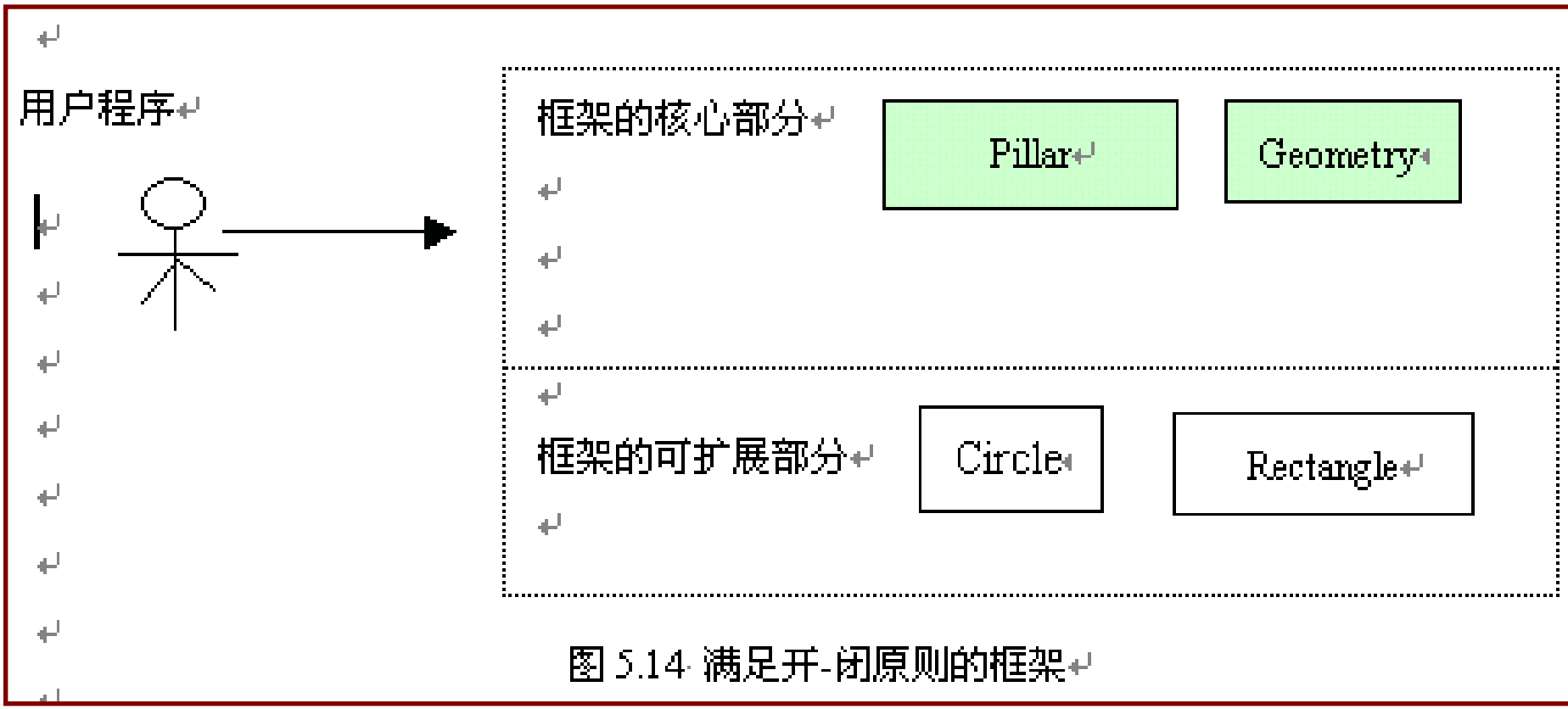


图 5.14 满足开-闭原则的框架

5.12 应用举例

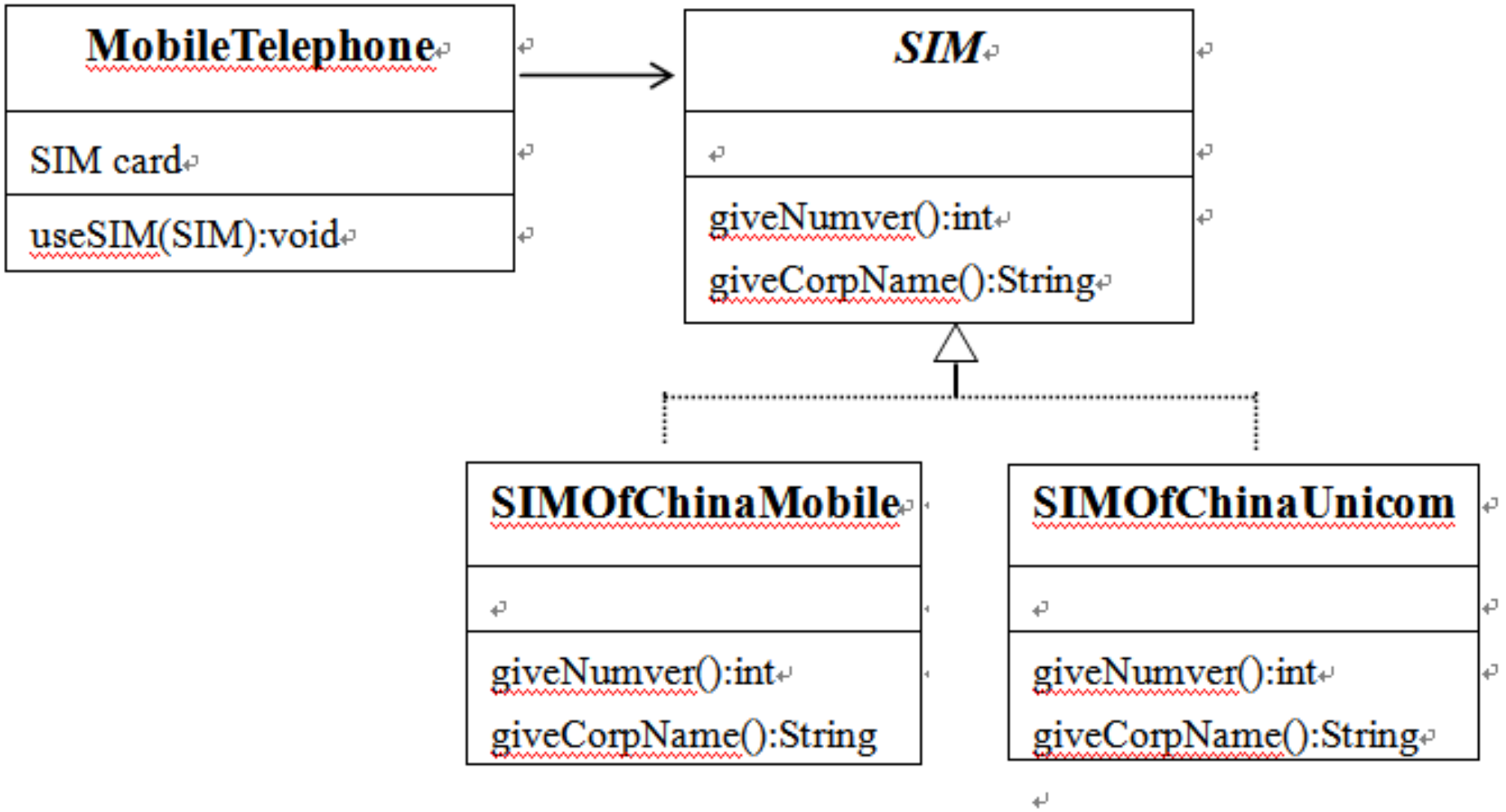


图 5.15 UML 类图

总结

- 继承是一种由已有的类创建新类的机制
- 子类继承父类的成员变量和方法，并可以实现变量隐藏与方法重写
- 上转型对象可以访问子类继承或隐藏的成员变量，也可以调用子类继承的方法或子类重写的实例方法
- 多态是面向对象编程的又一重要特性。子类通过方法的重写可以把父类的状态和行为改变为自身的状态和行为

作业

- 习题 5 ： 3,4