4．编程题（参考例子 7～9）

用类描述计算机中 CPU 的速度和硬盘的容量。要求 Java 应用程序有 4 个类，名字分别是 PC、CPU、HardDisk 和 Test，其中 Test 是主类。

- PC 类与 CPU 和 HardDisk 类关联的 UML 图（见图 4.34）

其中，CPU 类要求 getSpeed()返回 speed 的值，要求 setSpeed(int *m*)方法将参数 *m* 的值赋值给 speed；HardDisk 类要求 getAmount()返回 amount 的值，要求 setAmount(int *m*)方法将参数 *m* 的值赋值给 amount；PC 类要求 setCPU(CPU *c*) 将参数 *c* 的值赋值给 CPU，要求 setHardDisk (HardDisk *h*)方法将参数 *h* 的值赋值给 HD，要求 show()方法能显示 CPU 的速度和硬盘的容量。

- 主类 Test 的要求

（1）main 方法中创建一个 CPU 对象 cpu，cpu 将自己的 speed 设置为 2200。

（2）main 方法中创建一个 HardDisk 对象 disk，disk 将自己的 amount 设置为 200。

（3）main 方法中创建一个 PC 对象 pc。

（4）pc 调用 setCPU(CPU c)方法，调用时实参是 cpu。

（5）pc 调用 setHardDisk (HardDisk h)方法，调用时实参是 disk。

（6）pc 调用 show()方法。

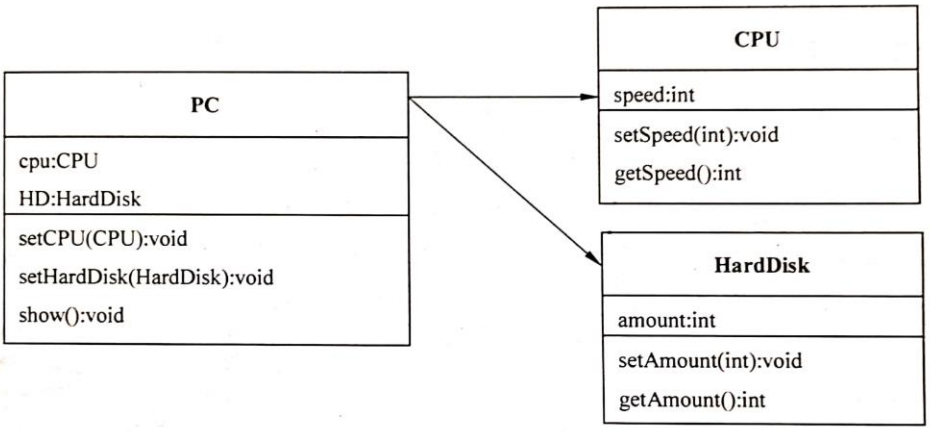

图 4.34　PC 与 CPU 和 HardDisk 关联 UML 图

▶ 4.5.3 引用类型参数的传值

Java 的引用型数据包括前面学习的数组、刚刚学习的对象以及后面要学习的接口。当参数是引用类型时，“传值”传递的是变量中存放的“引用”，而不是变量所引用的实体。

需要注意的是，对于两个相同类型的引用型变量，如果具有同样的引用，就会用同样的实体，因此，如果改变参数变量所引用的实体，就会导致原变量的实体发生同样的变化；但是，改变参数中存放的“引用”不会影响向其传值的变量中存放的“引用”，反之亦然，如图 4.13 所示。

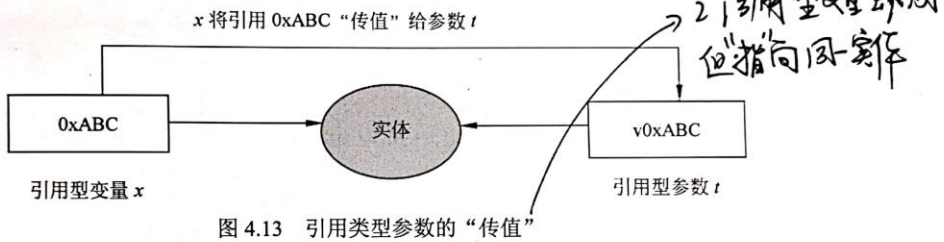

图 4.13 引用类型参数的“传值”

所以在学习对象时，一定要记住：一个类声明的两个对象如果具有相同的引用，二者就具有完全相同的变量（见 4.3.4 节）。

下面的例子 7 模拟收音机使用电池。例子 7 中使用的主要类如下。

- Radio 类负责创建一个“收音机”对象（Radio 类在 Radio.java 中）。
- Battery 类负责创建“电池”对象（Battery 类在 Battery.java 中）。
- Radio 类创建的“收音机”对象调用 openRadio(Battery battery)方法时，需要将一个 Battery 类创建的“电池”对象传递给该方法的参数 battery，即模拟收音机使用电池。
- 在主类中将 Battery 类创建的“电池”对象 nanfu 传递给 openRadio(Battery battery)方法的参数 battery，该方法消耗了 battery 的储电量（打开收音机会消耗电池的储电量），那么 nanfu 的储电量就发生了同样的变化。

例子 7 中收音机使用电池的示意图以及程序的运行效果如图 4.14 所示。

（a）收音机使用电池　　（b）收音机消耗电池的电量

图 4.14 收音机模拟

例子 7

Battery.java

```
public class Battery {
    int electricityAmount;
    Battery(int amount){
        electricityAmount = amount;
    }
}
```

Radio.java

```
public class Radio {
    void openRadio(Battery battery){
        battery.electricityAmount = battery.electricityAmount - 10;
                                                          //消耗了电量
    }
}
```

Example4_7.java

```
public class Example4_7 {
   public static void main(String args[]) {
      Battery nanfu = new Battery(100);        //创建电池对象
      System.out.println("南孚电池的储电量是:"+nanfu.electricityAmount);
      Radio radio = new Radio();               //创建收音机对象
      System.out.println("收音机开始使用南孚电池");
      radio.openRadio(nanfu);                  //打开收音机
      System.out.println("目前南孚电池的储电量是:"+nanfu.electricityAmount);
   }
}
```

例子 8 展示了圆锥和圆的组合关系（运行效果如图 4.15 所示），圆锥的底是一个圆，即圆锥有一个圆形的底。圆锥对象在计算体积时，首先委托圆锥的底（一个 Circle 对象）bottom 调用 getArea()方法计算底的面积，然后圆锥对象再计算出自身的体积。涉及的类如下。

- Circle 类创建圆对象。
- Circular 类创建圆锥对象，Circular 类将 Circle 类声明的对象作为自己的一个成员。
- 圆锥通过调用方法将某个圆的引用传递给圆锥的 Circle 类型的成员变量。

```
circle的引用:Circle@15db9742
圆锥的bottom的引用:null
circle的引用:Circle@15db9742
圆锥的bottom的引用:Circle@15db9742
圆锥的体积:523.3333333333334
修改circle的半径，bottom的半径同样变化
bottom的半径:20.0
重新创建circle,cirlce的引用将发生变化
circle的引用:Circle@6d06d69c
但是不影响circular的bottom的引用
圆锥的bottom的引用:Circle@15db9742
```

例子 8

Circle.java

```
public class Circle {
    double radius,area;
    void setRadius(double r) {
        radius=r;
    }
    double getRadius() {
        return radius;
    }
    double getArea(){
        area=3.14*radius*radius;
        return area;
    }
}
```

Circular.java

```
public class Circular {
    Circle bottom;
    double height;
```

```
    void setBottom(Circle c) { //设置圆锥的底是一个 Circle 对象
       bottom = c;
    }
    void setHeight(double h) {
       height = h;
    }
    double getVolme() {
       if(bottom == null)
          return -1;
       else
          return bottom.getArea()*height/3.0;
    }
    double getBottomRadius() {
       return bottom.getRadius();
    }
    public void setBottomRadius(double r){
       bottom.setRadius(r);
    }
}
```

Example4_8.java

```
public class Example4_8 {
   public static void main(String args[]) {
      Circle circle = new Circle();          //【代码 1】
      circle.setRadius(10);                  //【代码 2】
      Circular circular = new Circular();    //【代码 3】
      System.out.println("circle 的引用:"+circle);
      System.out.println("圆锥的 bottom 的引用:"+circular.bottom);
      circular.setHeight(5);
      circular.setBottom(circle);            //【代码 4】
      System.out.println("circle 的引用:"+circle);
      System.out.println("圆锥的 bottom 的引用:"+circular.bottom);
      System.out.println("圆锥的体积:"+circular.getVolme());
      System.out.println("修改 circle 的半径，bottom 的半径同样变化");
      circle.setRadius(20);                  //【代码 5】
      System.out.println("bottom 的半径:"+circular.getBottomRadius());
      System.out.println("重新创建 circle,cirlce 的引用将发生变化");
      circle = new Circle(); //重新创建 circle 【代码 6】
      System.out.println("circle 的引用:"+circle);
      System.out.println("但是不影响 circular 的 bottom 的引用");
      System.out.println("圆锥的 bottom 的引用:"+circular.bottom);
   }
}
```

结合程序运行的效果（图 4.15）对重要的代码分析讲解。

（1）执行【代码 1】和【代码 2】：

一个手机可以组合任何的 SIM 卡，下面的例子 9 模拟手机和 SIM 卡的组合关系。涉及的类如下：

- SIM 类负责创建 SIM 卡。
- MobileTelephone 类负责创建手机，手机可以组合一个 SIM 卡，并可以调用 setSIM (SIM card)方法更改其中的 SIM 卡。

程序运行效果如图 4.21 所示。

例子 9

```
手机号码:13889776509
手机号码:15967563567
```

图 4.21　手机组合 SIM 卡

SIM.java

```
public class SIM {
    long number;
    SIM(long number){
        this.number = number;
    }
    long getNumber() {
        return number;
    }
}
```

MobileTelephone.java

```
public class MobileTelephone {
    SIM sim;
    void setSIM(SIM card) {
        sim = card;
    }
    long lookNumber(){
        return sim.getNumber();
    }
}
```

Example4_9.java

```
public class Example4_9 {
   public static void main(String args[]) {
      SIM simOne = new SIM(13889776509L);
      MobileTelephone mobile = new MobileTelephone();
      mobile.setSIM(simOne);
      System.out.println("手机号码:"+mobile.lookNumber());
      SIM simTwo = new SIM(15967563567L);
      mobile.setSIM(simTwo);  //更换 SIM 卡
      System.out.println("手机号码:"+mobile.lookNumber());
   }
}
```