4.6 对象的组合

1. 对象组合与方法复用

一个类的成员变量可以是指向另一个类或同一个类（递归）的对象的对象名。

p. 72 代码 1

```
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;
    }
}
```

pp. 72-73 代码 2

```
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);
    }
}
```

“Circular”最好改为“CircularCone”或“Cone”,“getVolme”最好改为“getVolume”。

p. 73 代码 2

```
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);
   }
}
```

p. 75 代码 1

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

p. 76 代码 1

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

p. 76 代码 2

```
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());
   }
}
```

2. 类的关联关系和依赖关系的 UML 图

4.7 类成员与实例成员

1. 在“类成员与实例成员”中，“成员”指成员变量和（成员）方法，“类”指只要类来了（类被加载到内存后）就能用了，“实例”指要等对象来了（对象被创建后）才能用。

2. 类变量和实例变量的声明

类变量也称静态变量，前面用 static 修饰，而实例变量前面没有。

3. 类变量和实例变量的区别

（1）一个类的不同对象所拥有的同名类变量是相同的变量（共有），而所拥有的同名实例变量是不同的变量（私有）。

（2）一个对象所拥有的类变量只要该对象所属的类被加载到内存后就出现了，而所拥有的实例变量要等该对象被创建后才出现。

（3）类变量还可通过“类名.类变量名”的形式使用。

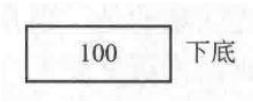
p.78 代码 1

```
public class Lader {
   double 上底,高;            //实例变量
   static double 下底;        //类变量
   void 设置上底(double a) {
      上底 = a;
   }
   void 设置下底(double b) {
      下底 = b;
   }
   double 获取上底() {
      return 上底;
   }
   double 获取下底() {
      return 下底;
   }
}
```

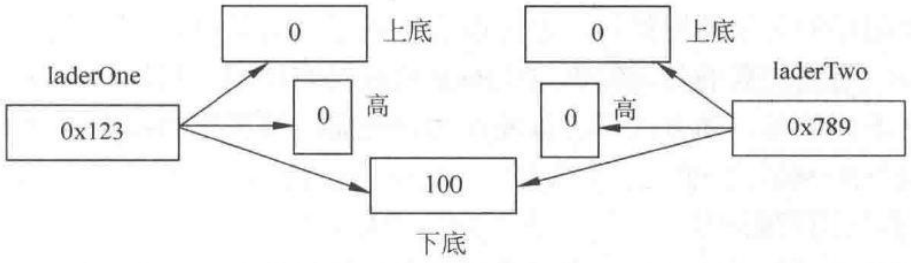
p.78 代码 2

```
public class Example4_10 {
   public static void main(String args[]) {
      Lader.下底 = 100;     //Lader 的字节码被加载到内存,通过类名操作类变量
      Lader laderOne = new Lader();
      Lader laderTwo = new Lader();
      laderOne.设置上底(28);
      laderTwo.设置上底(66);
      System.out.println("laderOne 的上底:"+laderOne.获取上底());
      System.out.println("laderOne 的下底:"+laderOne.获取下底());
      System.out.println("laderTwo 的上底:"+laderTwo.获取上底());
      System.out.println("laderTwo 的下底:"+laderTwo.获取下底());
   }
}
```

Lader.下底 = 100.0;



Lader laderOne = new Lader();
Lader laderTwo = new Lader();



4. 类方法和实例方法的定义

类方法也称静态方法，前面用 static 修饰，而实例方法前面没有。

5. 类方法和实例方法的区别

（1）一个类的类方法只要该类被加载到内存后就被分配入口地址了，而实例方法要等该类的第一个对象被创建后才被分配入口地址。

（2）类方法中只能使用类变量，而实例方法中既能使用类变量又能使用实例变量。

（3）类方法中只能调用类方法（包括自身），而实例方法中既能调用类方法又能调用实例方法（包括自身）。（方法的嵌套调用、递归调用）

（4）类方法还可通过“类名.类方法名(实际参数表)”的形式调用。

（5）可以将无需使用实例变量的方法定义为类方法。（如判断正整数是否为质数）

pp.80-81 代码 1

```
import java.util.*;
public class Example4_11 {
   public static void main(String args[]) {
      Scanner scanner = new Scanner(System.in);
      int [] a = {12,34,9,23,45,6,45,90,123,19,34};
      Arrays.sort(a);
      System.out.println(Arrays.toString(a));
      System.out.println("输入整数，程序判断该整数是否在数组中:");
      int number = scanner.nextInt();
      int index=Arrays.binarySearch(a,number);
      if(index>=0)
         System.out.println(number+"和数组中索引为"+index+"的元素值相同");
      else
         System.out.println(number+"不与数组中任何元素值相同");
    }
}
```

“import java.util.*”表示引入 java.util 包中所有的类。

4.8 方法重载

1. 方法重载指在一个类中定义了多个名字一样但形式参数表互不相同的方法（不涉及返回值类型）。形式参数表互不相同指把形式参数名去除后互不相同。

pp.81-82 代码 2

```
class People {
   float hello(int a,int b) {
      return a+b;
   }
   float hello(long a,int b) {
      return a-b;
   }
   double hello(double a,int b) {
      return a*b;
   }
}
public class Example4_12 {
    public static void main(String args[]) {
      People tom = new People();
      System.out.println(tom.hello(10,20));
      System.out.println(tom.hello(10L,20));
      System.out.println(tom.hello(10.0,20));
   }
}
```

p.82 代码 2

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

p.82 代码 3

```
public class Tixing {
   double above,bottom,height;
   Tixing(double a,double b,double h) {
      above = a;
      bottom = b;
      height = h;
   }
   double getArea() {
      return (above+bottom)*height/2;
   }
}
```

Circle 类的 getArea 方法和 Tixing 类的 getArea 方法，跟重载无关。

pp.82-83 代码 4

```
public class Student {
   double computerArea(Circle c) {  //是重载方法
      double area = c.getArea();
      return area;
   }
   double computerArea(Tixing t) {  //是重载方法
      double area = t.getArea();
      return area;
   }
}
```

p.83 代码 2

```
public class Example4_13 {
  public static void main(String args[]) {
     Circle circle = new Circle();
     circle.setRadius(196.87);
     Tixing lader = new Tixing(3,21,9);
     Student zhang = new Student();
     System.out.println("zhang 计算圆的面积：");
     double result = zhang.computerArea(circle);
     System.out.println(result);
     System.out.println("zhang 计算梯形的面积：");
     result = zhang.computerArea(lader);
     System.out.println(result);
   }
}
```

2. 要注意调用时实际参数的表示，避免重载出现歧义。

4.9 this 关键字

1. 在构造方法中

（1）“this”指代指向调用该构造方法所创建的当前对象的对象名。

（2）使用实例变量时，通常省去“this.”。只有当实例变量和局部变量（包括形式参数）同名时，才在实例变量前面加上“this.”。

（3）使用类变量时，通常省去“类名.”，只有当类变量和局部变量（包括形式参数）同名时，才在类变量前面加上“类名.”。

（4）调用实例方法时，通常省去“this.”。（方法的嵌套调用）

（5）调用类方法时，通常省去“类名.”。（方法的嵌套调用）

p.84 代码 1

```
public class People{
    int leg,hand;
    String name;
    People(String s){
        name = s;
        this.init();   //可以省略 this，即将“this.init()”;写成“init();”
    }
    void init(){
       leg = 2;
       hand = 2;
       System.out.println(name+"有"+hand+"只手"+leg+"条腿");
    }
    public static void main(String args[]){
      People boshi = new People("布什");
                                  //创建 boshi 时，构造方法中的 this 就是对象 boshi
    }
}
```

“name = s;”中的“name”前面省去了“this.”。

2. 在实例方法中

（1）“this”指代指向调用该实例方法的当前对象的对象名。

（2）使用实例变量时，通常省去“this.”，只有当实例变量和局部变量（包括形式参数）同名时，才在实例变量前面加上“this.”。

（3）使用类变量时，通常省去“类名.”，只有当类变量和局部变量（包括形式参数）同名时，才在类变量前面加上“类名.”。

pp.84-85 代码 2

```
class A {
```

```
    int x;
    static int y;
    void f() {
      this.x = 100;
      A.y = 200;
    }
}
```

x 是实例变量，y 是类变量。f 是实例方法。

p.85 代码 2

```
class A {
    int x;
    static int y;
    void f() {
      x = 100;
      y = 200;
    }
}
```

（4）调用实例方法（包括自身）时，通常省去“this.”。（方法的嵌套调用、递归调用）

（5）调用类方法时，通常省去“类名.”。（方法的嵌套调用）

p.85 代码 3

```
class B {
    void f() {
      this.g();
       B.h();
    }
    void g() {
       System.out.println("ok");
    }
    static void h() {
       System.out.println("hello");
    }
}
```

f 和 g 是实例方法，h 是类方法。

p.86 代码 1

```
class B {
  void f() {
       g();      //省略 this
       h();      //省略类名
    }
    void g() {
       System.out.println("ok");
    }
    static void h() {
       System.out.println("hello");
    }
}
```

3. 在类方法中不能使用“this”。

4.10 包

1. 可通过将同名的类放在不同的包中来区分它们。

2. package 语句
   package 包名;

3. 无名包

4.11 import 语句

1. 用于在一个类中引入不在同一个包中的类。

2. import 语句放在 package 语句和类声明之间。

3. 引入类库中的类
   使用 java.lang 包时可省去 import 语句。

4. 引入自定义包中的类

4.12 访问权限

1. 访问权限指能否创建对象以及所创建的对象能否使用成员变量和调用方法。

2. 共有类和友好类
（1)使用 public 访问限制符修饰的类称为共有类,不使用访问限制符修饰的类称为友好类。
（2）若 A 类是共有类，则在 B 类中可以创建 A 类的对象。
（3）若 A 类是友好类，且 B 类和 A 类在同一个包中，则在 B 类中可以创建 A 类的对象。

3. 在类体中，定义实例方法时可以使用所有成员变量和调用所有方法（包括自身），不受成员变量和方法前面的访问限制符的影响,定义类方法时可以使用所有类变量和调用所有类方法（包括自身），不受类变量和类方法前面的访问限制符的影响。

4. 私有成员变量和私有方法
（1）使用 private 访问限制符修饰的成员变量和方法分别称为私有成员变量和私有方法。
（2）在 B 类中创建 A 类的 a 对象后，a 对象不可以使用 A 类的私有成员变量和调用 A 类的私有方法。
（3）在 B 类中不可以通过 A 类的类名使用 A 类的私有类变量和调用 A 类的私有类方法。
   p.92 代码 1

```
class Tom {
   private float weight;                    //weight 是 private 的 float 型变量
    private float f(float a,float b) {   //方法 f 是 private 方法
       return a+b;
    }
}
```

p.92 代码 2

```
class Jerry {
   void g() {
      Tom cat = new Tom();
      cat.weight = 23f;          //非法
      float sum = cat.f(3,4);    //非法
    }
}
```

p.92 代码 3

```
public class Student {
   private int age;
   public void setAge(int age) {
      if(age>=7&&age<=28) {
         this.age = age;
      }
   }
   public int getAge() {
      return age;
   }
}
```

p.93 代码 1

```
public class Example4_19 {
   public static void main(String args[]) {
      Student zhang = new Student();
      Student geng = new Student();
      zhang.setAge(23);
      System.out.println("zhang 的年龄："+zhang.getAge());
      geng.setAge(25);
      //zhang.age = 23;或 geng.age = 25;都是非法的，因为 zhang 和 geng 已经不在
      //Student 类中
      System.out.println("geng 的年龄："+geng.getAge());
    }
}
```

5. 共有成员变量和共有方法

（1）使用 public 访问限制符修饰的成员变量和方法分别称为共有成员变量和共有方法。

（2）在 B 类中创建 A 类的 a 对象后，a 对象可以使用 A 类的共有成员变量和调用 A 类的共有方法。

（3）在 B 类中可以通过 A 类的类名使用 A 类的共有类变量和调用 A 类的共有类方法。

p.93 代码 2

```
class Tom {
   public float weight;                    //weight 是 public 的 float 型变量
    public float f(float a,float b) {      //方法 f 是 public 方法
       return a+b;
    }
}
```

p. 93 代码 3

```
class Jerry {
  void g() {
       Tom cat = new Tom();
        cat.weight = 23f;          //合法
        float sum = cat.f(3,4); //合法
    }
}
```

6. 友好成员变量和友好方法

（1）不使用访问限制符修饰的成员变量和方法分别称为友好成员变量和友好方法。

（2）若 B 类和 A 类在同一个包中，则在 B 类中创建 A 类的 a 对象后，a 对象可以使用 A 类的友好成员变量和调用 A 类的友好方法。

（3）若 B 类和 A 类在同一个包中，则在 B 类中可以通过 A 类的类名使用 A 类的友好类变量和调用 A 类的友好类方法。

pp. 93-94 代码 4

```
class Tom {
   float weight;                       //weight 是友好的 float 型变量
    float f(float a,float b) {         //方法 f 是友好方法
       return a+b;
    }
}
```

Jerry 类和 Tom 类在同一个包中。

p. 94 代码 2

```
class Jerry {
   void g() {
       Tom cat = new Tom();
       cat.weight = 23f;          //合法
       float sum = cat.f(3,4);   //合法
    }
}
```

7. 受保护成员变量和受保护方法

（1）使用 protected 访问限制符修饰的成员变量和方法分别称为受保护成员变量和受保护方法。

（2）受保护成员变量和受保护方法分别好似友好成员变量和友好方法，但在子类中有差异。

p. 94 代码 3

```
class Tom {
    protected float weight;                    //weight 是 protected 的 float 型变量
    protected float f(float a,float b) { //方法 f 是 protected 方法
       return a+b;
    }
}
```

Jerry 类和 Tom 类在同一个包中。

p. 94 代码 4

```
class Jerry {
   void g() {
      Tom cat = new Tom();
      cat.weight = 23f;          //合法
      float sum = cat.f(3,4);    //合法
    }
}
```

4.13 基本类型的类封装

与基本类型相比，增加了方法。

p. 96 代码 1

```
public class Example4_20 {
   public static void main(String args[ ]) {
      char a[] = {'a','b','c','D','E','F'};
      for(int i =0;i<a.length;i++) {
        if(Character.isLowerCase(a[i]))
           a[i] = Character.toUpperCase(a[i]);
        else if(Character.isUpperCase(a[i]))
           a[i] = Character.toLowerCase(a[i]);
       }
      for(int i=0;i<a.length;i++)
        System.out.print(" "+a[i]);
    }
}
```

4.14 对象名数组

对象名数组的元素是一系列指向同一个类的若干对象的对象名。

对象名数组的能力比链表（后续课程中涉及）的弱得多。

pp. 96-97 代码 5

```
class Student{
   int number;
}
public class Example4_21 {
   public static void main(String args[ ]) {
```

```
        Student stu[] = new Student[10];   //创建对象数组 stu
        for(int i=0;i<stu.length;i++) {
          stu[i] = new Student();          //创建 Student 对象 stu[i]
          stu[i].number = 101+i;
        }
        for(int i=0;i<stu.length;i++) {
           System.out.println(stu[i].number);
        }
    }
}
```