第3章 面向对象设计基础——抽象和封装

3.1 实验目的

- 理解对象和类的基本概念，并初步理解面向对象设计原则中的抽象、封装，掌握 get 方法和 set 方法的设计，掌握 toString()方法和 equals()方法的设计技巧，掌握方法重载的概念和实现技巧。
- 理解 Java 中数组的概念和使用技巧，并掌握基本类型变量和引用变量的区别。

3.2 相关知识

面向对象程序设计是使用类和对象将现实世界中真实的实体或抽象的概念在程序中建立起相应的模型，这一过程本身就是抽象，是人类特有的一种不断训练和强化的能力。还要重点理解类和对象的关系，类是创建对象的代码模板，对象是用类创建的实例。在设计类时，要采用封装的思想，使用 private 关键字将数据和方法对外隐藏，用 public 关键字提供对象和外部进行信息交换的接口，并在这些接口方法中，提供合理的代码设计用来过滤传入和传出数据，也就是说封装不是简单用 private 关键字私有化某些成员，而是保证被封装的对象是一个有机的整体，不能因为传入“坏”的数据导致对象出问题。在 Java 中数组被看成是对象，它有一个属性 length 用来指明此数组的元素的个数，通过下标使用数组元素。

3.3 实验内容


视频讲解

3.3.1 验证实验

(1) 理解抽象和封装。在第一个验证实验中，抽象一个简单的 Person 类，只抽象了“人”的三个最基本的属性：年龄(age)、姓名(name)和性别(sex)。对它们进行了封装，并提供了相应的 get 方法和 set 方法，同时在类中也提供了两个构造方法，并给出了 equals()和 toString()方法。

① 用记事本或 Ultraedit 输入以下程序并以 Person.java 存盘。

② 用 javac 编译，用 java 执行，然后填空。

```
public class Person{
    private int age = 0;
```

```
    private String name = "noname";
    private char sex = 'M';
    public Person(){}
    public Person(String n, int a,char s){
       name = n;
       if(a > = 0&&a < 140) age = a;              //数据过滤
       else age = 0;
       if(s == 'M') sex = s;                      //数据过滤
       else sex = 'F';
   }
  public void introduceme() {
     System.out.println("my name is: " + name + "\tmy age is: " + age);
     if(sex == 'M') System.out.println("I am man!");
     else System.out.println("I am woman!");
     }
     public String getName(){return name;}
     public void setName(String n){name = n;}
     public int getAge(){return age;}
     public void setAge(int a){//注意数据过滤
           if(a > = 0&&a < 140) age = a;
           else age = 0;
     }
     public char getSex(){return sex;}
     public void setSex(char s){//注意数据过滤
         if(s == 'M') sex = 'M';
         else sex = 'F';
     }
     public boolean equals(Person a){
         if(this.name.equals(a.name)&&this.age == a.age
&&this.sex == a.sex)
             return true;
         else
             return false;
     }
     public String toString(){
          return name + "," + sex + "," + age;
     }
}
class PersonTest{
    public static void main(String args[]) {
        Person p1,p2;
        p1 = new Person("张三",28, 'M');
        p2 = new Person ();
        p2.setName("陈红");p2.setAge(38);p2.setSex('F');
        p1.introduceme();
        p2.introduceme();
     }
}
```

封装的意思是_____。

```
p1 = new Person("张三",28, 'M');
```

这条语句的含义和作用是_____。

p2.setName("陈红");的用处是_____。

（2）数组测试。数组是将相同类型的数据放在连续的存储区域，通过数组名和下标来使用每一个数组元素，注意Java中的数组名仅是一个引用变量名，并且Java认为数组是一个对象，有一个length属性用来表示数组元素个数，通过下标使用数组元素。请输入以下代码并以ArrayTest.java为文件名保存，然后编译运行，并回答问题。

```
public class ArrayTest {
   public static void main(String[] args){
     int[] a;
     Person[] b;
     a = new int[10];
     b = new Person[3];
     for(int i = 0;i < 10;i++){
       a[i] = (int)(100 * Math.random());
     }
     b[0] = new Person("张三",28,'M');
     b[1] = new Person("李四",20,'M');
     b[2] = new Person();
     b[2].setName("葛优");
     b[2].setAge(46);
     b[2].setSex('F');
     for(int i = 0;i < 10;i++) {
       System.out.println("a[" + i + "] = " + a[i]);
     }
     System.out.println(b[0] + "\n" + b[1] + "\n" + b[2]);
     System.out.println("a 中元素个数: " + a.length);
     System.out.println("b 中元素个数: " + b.length);
   }
}
```

试解释Java中数组和C语言中数组的区别。

试解释b=new Person[3];语句和b[0]=new Person("张三",28,'M');语句的作用，以及它们之间的区别和关系。

（3）Java方法的参数传递用法。Java方法的参数传递是传值操作。对于基本数据类型（如int、char类型）变量作为参数，方法内对参数的操作实质是对参数复制变量的操作，不会改变原变量的值；对于引用类型（如数组、字符串）变量作为参数，方法内对该变量的操作是对引用变量所指向对象的操作，会改变原对象的数据。下面示例演示了方法参数调用的传递情况。

```
public class MethodParameter {
       public static void main(String[] args) {
           int a = 6;
           char[] str = new char[] { 'H', 'e', 'l', 'l', 'o' };
           StringBuffer sb = new StringBuffer("TOM");
           changeAddr(str, sb);
           System.out.println(str);
           System.out.println(sb.toString());
           changeValue(a, str, sb);
           System.out.println(a);
```

```
        System.out.println(str);
        System.out.println(sb.toString());
    }

    private static void changeAddr( char[] c, StringBuffer sb) {
        c = new char[] { 'Y', 'e', 'l', 'l', 'o' };
        sb = new StringBuffer("SawYer");
    }

    private static void changeValue(int a, char[] c, StringBuffer sb) {
        a = 8;
        c[0] = 'Y';
        sb.append(" Sawya");
    }
}
```

程序的运行结果是_____。

(4) 重载方法演示。在同一个类中,有多个同名的方法,但方法参数列表不同,执行代码也不同,称为方法重载。请输入以下程序代码进行分析,学习方法重载。

```
public class DemoOverloading {
    public void disp(char c)       {
            System.out.println(c);
        }
    public void disp(char c, int num){
            for(int i = 1;i <= num;i++) System.out.print(c);
            System.out.println();
        }
    public void disp(String s){
        System.out.println(s.toUpperCase().charAt(1));
        }
    public void disp(String s,int num) {
        for(int i = 1;i <= num;i++) System.out.print(s + " ");
        System.out.println();
        }
    public static void main(String[] args){
        DemoOverloading obj = new DemoOverloading();
        obj.disp('*');
        obj.disp('=',10);
        obj.disp("abcdefg");
        obj.disp("abcdefg",10);
        }
}
```

试解释方法重载的实现机制,即编译器是如何识别不同的方法。

3.3.2　填空实验

(1) 理解抽象和封装。把主教材 3.2.1 节中的屏幕抽象和矩形抽象示例实验一下:假设要在抽象屏幕上用“*”打印矩形,可以把此矩形看成一个对象,用面向对象的思维来进行分析和抽象,所有的矩形都有宽(w)和高(h),并且在屏幕上有一个位置,而位置是由形如

(x、y)的坐标标识出来的，所以最简单的抽象就是通过(w，h，x，y)来定义一个矩形类(Rectangle)，然后提供一个 printme()方法在抽象屏幕上打印出这个矩形。

```
//Rectangle.java
public class Rectangle {
    int x,y,w,h;
    Rectangle() {
        _________________;                    //调用另一个构造方法传递参数(0,0,1,1)
    }
    public Rectangle(int x,int y,int w,int h) {
        this.x = x;
        this.y = y;
        this.w = w;
        this.h = h;
    }

    public void printme(Screen myscreen) {
       myscreen.setY(y);
       for(int i = 1;i <= h;i++)          {
          myscreen.setX(x);
          myscreen.repeat('*',w);
          myscreen.println();
       }
    }
}
//Screen.java
public class Screen {
    int SCREEN_WIDTH;
    int SCREEN_HEIGHT;
    int x,y;
    char[][] data;
    int getX(){
        return x;
    }
    public void setX(int x){
        this.x = x;
    }
    public int getY(){
        return y;
    }
    public void setY(int y){
        this.y = y;
    }
    public Screen(){
        SCREEN_HEIGHT = 50;
        SCREEN_WIDTH = 80;
        data = new char[SCREEN_HEIGHT][SCREEN_WIDTH];
    }
    public Screen(int r,int c) {
        SCREEN_HEIGHT = r;
        SCREEN_WIDTH = c;
        data = new char[SCREEN_HEIGHT][SCREEN_WIDTH];
    }
    public void cls() {
```

```
    for(int i = 0;i < SCREEN_HEIGHT;i++) {
        for(int j = 0;j < SCREEN_WIDTH;j++){
            data[i][j] = ' ';
        }
     }
    }
    public void display() {
    for(int i = 0;i < SCREEN_HEIGHT;i++){
        for(int j = 0;j < SCREEN_WIDTH;j++){
            System.out.print(data[i][j]);
        }
        System.out.println();
     }
    }
    public  void repeat(char ch,int m)  {
        for(int i = 1;i < = m;i++) print(ch);
    }
    public void print(char ch){
        if (y < SCREEN_HEIGHT && x < SCREEN_WIDTH){
            data[y][x] = ch;
            x++;
            if (x == SCREEN_WIDTH){
                y++;
                if (y == SCREEN_HEIGHT){
                    scroll();                    //屏幕上滚一行
                    y = SCREEN_HEIGHT - 1;
                }
                x = 0;
            }
        }else {
            System.out.println("错误: 超出屏幕了!");
        }
    }
    public void println() {
    if(++y == SCREEN_HEIGHT) {
       scroll();
       y = SCREEN_HEIGHT - 1;
     }
    x = 0;
    }
    public void scroll() {
    for(int i = 0;i < data.length - 1;i++){
        data[i] = data[i + 1];
    }
    data[data.length - 1] = new char[SCREEN_WIDTH];
  }
}
```

有了上面的矩形类和屏幕类程序代码,就可以进行测试了,设计一个测试类,提供 main() 方法和测试代码,运行结果如图 3-1 所示。

```
//TestRectangle.java
public class TestRectangle {
   public static void main(String[] args){
```

```
        Screen myscreen = new Screen();
        Rectangle rc1 = _____;   //第 0 行 0 列的 5 行 6 列的矩形
        rc1.printme(_____);      //在屏幕 myscreen 上打印 rc1
        Rectangle rc2 = new Rectangle(32,4,5,7); //第 4 行 32 列的 7 行 5 列矩形
        rc2.printme(myscreen);
        myscreen._____;          //显示屏幕对象
    }
}
```

但是上面的抽象并没有封装，没有封装的对象很容易被非法修改或者破坏，如图 3-2 所示，看以下程序。

```
//TestNoEnCapsulation.java
public class TestNoEnCapsulation {
    public static void main(String[ ] args){
        Screen myscreen = new Screen();
        myscreen.cls();
        Rectangle rc1 = new Rectangle(0,0,6,5);
        rc1.h = 3;                          //数据被任意修改,对象被破坏
        rc1.x = 10;                         //数据被任意修改,对象被破坏
        rc1.printme(myscreen);
        Rectangle rc2 = new Rectangle(32,4,5,7);
        rc2.w = 10;                         //数据被任意修改,对象被破坏
        rc2.printme(myscreen);
        myscreen.data[5][33] = '中';        //数据被非法修改,对象内容被破坏
        myscreen.display();
    }
}
```

图 3-1　屏幕上打印矩形对象

图 3-2　对象内容被破坏

如果修改屏幕对象的数据非法，还有可能出错，例如下面程序中将屏幕对象 myscreen 的宽度修改为-3，则程序就会出错，矩形对象就无法显示了，如图 3-3 所示。

```
//TestNoEnCapsulation1.java
public class TestNoEnCapsulation1 {
    public static void main(String[ ] args){
        Screen myscreen = new Screen();
        myscreen.cls();
        myscreen.width = -3;                //屏幕对象的宽改为 -3,程序报错
        Rectangle rc1 = new Rectangle(0,0,6,5);
        rc1.printme(myscreen);
        Rectangle rc2 = new Rectangle(32,4,5,7);
        rc2.printme(myscreen);
        myscreen.display();
    }
  }
```

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 80
        at myjava2019.chap3.test0.Screen.init(Screen.java:39)
        at myjava2019.chap3.test0.Test.main(Test.java:7)
```

图 3-3　没有封装数据被其他类修改后出错

为了防止数据被非法修改，需要使用封装技术。在 Java 中，实现封装的关键字是 private，提供公有接口的关键字是 public。要实现封装需要以下两步。

第一步，将对象内部的属性数据用 private 修饰，这样其他对象就无法直接访问和修改了，并且有些属性在对象创建后再也不允许修改，则此类属性应该定义为常量。

第二步，对于需要访问的属性提供读值方法 getter，并需要特定代码对数据进行处理，根据安全需要可隐藏某些数据；对于需要修改的属性提供写值方法 setter，并且在方法中提供约束和过滤代码，保证合法数据进入，阻挡非法数据进入。

屏幕类的封装代码如下：

```
//Screen.java
public class Screen {
    private final int SCREEN_WIDTH;
    private final int SCREEN_HEIGHT;
    private int x;
    private int y;
    private char[][] data;
    public int getX(){
        return x;
    }
    public void setX(int x){
        if (x < SCREEN_WIDTH)  this.x = x;
    }
    public int getY(){
        return y;
    }
    public void setY(int y) {
        if (y < SCREEN_HEIGHT)  this.y = y;
    }
    public Screen(){
        SCREEN_HEIGHT = 50;
        SCREEN_WIDTH = 80;
        data = new char[SCREEN_HEIGHT][SCREEN_WIDTH];
    }
    public Screen(int r, int c){                    //通过判断对输入的数据进行过滤
        if (r >= 1 && r <= 1000)
            SCREEN_HEIGHT = r;
        else
            SCREEN_HEIGHT = 50;
        if (c >= 1 && c <= 1000)
            SCREEN_WIDTH = c;
        else
            SCREEN_WIDTH = 80;
        data = new char[SCREEN_HEIGHT][SCREEN_WIDTH];
    }

    public void cls() {
```

```
        for (int i = 0; i < SCREEN_HEIGHT; i++){
            for(int j = 0; j < SCREEN_WIDTH; j++) {
                data[i][j] = ' ';
            }
        }
    }
    public void display(){
        for (int i = 0; i < SCREEN_HEIGHT; i++){
            for (int j = 0; j < SCREEN_WIDTH; j++){
                System.out.print(data[i][j]);
            }
            System.out.println();
        }
    }

    public void repeat(char ch, int m){
        for (int i = 1; i <= m; i++)
            print(ch);
    }
    public void print(char ch){
        if (y < SCREEN_HEIGHT && x < SCREEN_WIDTH){
            data[y][x] = ch;
            x++;
            if (x == SCREEN_WIDTH){
                y++;
                if (y == SCREEN_HEIGHT) {
                    scroll();
                    y = SCREEN_HEIGHT - 1;
                }
                x = 0;
            }
        } else {
            System.out.println("错误：超出屏幕了!");
        }
    }
    public void println(){
        y++;
        if (y == SCREEN_HEIGHT){
            scroll();
            y = SCREEN_HEIGHT - 1;
        }
        x = 0;
    }
    public void scroll(){
        for (int i = 0; i < data.length - 1; i++){
            data[i] = data[i + 1];
        }
        data[data.length - 1] = new char[SCREEN_WIDTH];
    }
}
```

修改后，屏幕(Screen)类对象的内部数据外部就无法修改了，并对相应方法的代码也做了过滤处理，非法数据无法进入。读者可以用前面例子中的测试类进行测试，看看效果。

请读者将矩形类 Rectangle 进行封装，然后再使用下面的测试程序进行测试，看看封装是否成功。

```
//TestEnCapsulation.java
public class TestEnCapsulation {
    public static void main(String[] args){
        Screen myscreen = new Screen();
        myscreen.cls();
        Rectangle rc1 = new Rectangle(0,0,6,5);
        rc1.h = -3;                              //试图直接修改数据,无法通过编译
        rc1.x = 10;                              //试图直接修改数据,无法通过编译
        rc1.printme(myscreen);
        Rectangle rc2 = new Rectangle(32,4,5,7);
        rc2.w = 10;                              //试图直接修改数据,无法通过编译
        rc2.printme(myscreen);
        myscreen.data[5][33] = '中';             //试图直接修改数据
        myscreen.display();
    }
}
```

将封装好的 Rectangle.java 给实验老师检查。

(2) 数组使用。用面向对象方法实现筛法求素数，从面向对象的视角看，筛法求素数需要下列器件：一个数字产生器；能够逐一输出需要判断的数据；另一个是筛子：用于数据的过滤。筛子中有一个过滤器，用于存储素数。每次过滤时，就是判断数据是否被过滤器中的所有数据整除，若无法过滤掉，则将该数据保留在过滤器中，请填空以完成程序。

```
class Shiyan3_2_2 {
   public static void main(String[] args){
      int n = 100;
      ____________________;                  //创建 Sieve 类的对象
      s.executeFilter(n);
      System.out.println("小于" + n + "的素数有: ");
      s.outFilter();
   }
}
class Counter{                               //数字产生器
   private int value;                        //数字产生器的初值
   Counter(int val){value = val;}
   public int getValue(){return value;}
   public void next(){value++;}              //产生下一个数字
}
class Sieve{                                 //筛子
   final int Max = 100;                      //设定过滤器的最大值
   private int filterCount = 0;
   private int[] f;                          //存储过滤器数据的数组
   public Sieve(){f = new int[Max];filterCount = 0;}
   public void executeFilter(int n){         //执行过滤,产生 2～n 素数
      Counter c = new Counter(2);
      for(;c.getValue()< n;c.next()){
          ____________________________;     //实施过滤
      }
   }
```

```
    public void passFilter(Counter c){
        for(int i = 0;i < filterCount/2;i++)
            if(_____________________) return;    //判断若为合数则返回
                _____________________;           //若为素数,则加入过滤器
    }
    private void addElementIntoFilter(int x){
        f[filterCount] = x;
        filterCount++;
    }
    public void outFilter(){
        for(int i = 0;i < filterCount;i++){
            System.out.printf(" % 4d",f[i]);
            if((i + 1) % 10 == 0)System.out.println();
        }
    }
}
```

3.3.3 设计实验

(1) 抽象和封装。从直角坐标系的视角抽象设计一个 Point 类,用来表示平面上的点对象,该类包含两个 double 型成员变量：x、y,一个 Color 类型成员变量 mycolor,请给出此类的三个构造方法(重载的),分别是一个不带参数的、一个带两个参数的、一个带三个参数的构造方法；给出一个计算两点间距离的方法 distance(Point another)。还要给出对应的 get 方法和 set 方法,最后重写 equals()方法和 toString()方法。用下面的 main()方法测试。

提示：import java.awt.Color；才能使用 Color 类。

```
public static void main(String[] args) {
    Point A = new Point();
    Point B = new Point(50,60);
    Point C = new Point(100,200,Color.red);
    System.out.println("B:(" + B.getX() + "," + B.getY() + ")" + "color: " + B.getColor());
    A.setX(100);
    A.setY(200);
    A.setColor(Color.red);
    System.out.println("A == B? " + A.equals(B));
    System.out.println("A→B " + A.distance(B));}
```

(2) 数组使用。Java 的方法返回值可以是基本数据类型的变量,也可以是对象的引用,如矩阵类的加法运算方法的返回值可以用矩阵对象的引用变量。运用面向对象建模思想,抽象建模高等代数中矩阵类 Matrix,用两个大于 0 的整型成员变量表示行数和列数,用一个浮点型二维数组存储该矩阵的数据,提供相应成员变量的 get 方法和 set 方法,提供矩阵的初始化方法 initValue()和矩阵的加减方法 add(Matrix b)和 sub(Matrix c),注意对非法数据的过滤,提供 equals()方法和 toString()方法等,使用以下代码进行测试。

```
class TestMatrix {
    public static void main(String[] args){
        Matrix A = new Matrix(3,4);
        A.initValue();
```

```
        System.out.print("矩阵 A:" + A);
        Matrix B = new Matrix(3,4);
        B.initValue();
        System.out.print("矩阵 B:" + B);
        Matrix C = A.add(B);
        System.out.println("A + B:" + C);
        Matrix D = A.sub(B);
        System.out.println("A - B:" + D);
    }
}
```