第二次作业

2017221105003 黄文杰

1.面向对象程序设计的三个特性是什么？

①封装性；

②多态性；

③继承性。

2.如何用类的成员来表示对象的属性和行为？

①用类的成员变量表示对象的属性；

②用类的成员方法表示对象的行为；

③例如，“狗”类包含”身长“、”体重“、”毛色“等多种属性，也包含”行走“、”吠“、”吃“等行为；

|  |
| --- |
| 代码示例 |
| class Dog{  float len; //身长  float height; //体重  String color; //毛色  walk(){ //行走  …  }  bark(){ //吠  …  }  eat(){ //吃  …  }  } |

3.类的成员变量是如何获得初始化的值的？

在类的成员变量初始化中，可分为三个步骤：

①首先是创建成员变量分配存储空间时，赋上一个缺省的初值；

|  |  |
| --- | --- |
| 类型 | 值 |
| byte | 0 |
| short | 0 |
| int | 0 |
| long | 0L |
| float | 0.0f |
| double | 0.0d |
| char | ‘\u0000’ |
| boolean | false |
| 引用类型 | null |

②在成员变量声明是赋的初值；

③在创建示例时，调用构造方法中的语句所赋值。

4.方法的数据传递方式有几种，它们各有什么特点，举例说明。

①形式参数是简单类型。特点：在方法调用时，实际参数将其存储单元的数据赋值给形式参数。当方法调用结束返回时，无论形式参数的值发生什么的变化都不会影响实际参数的值。

②形式参数是引用类型。特点：在方法调用时，同样实际参数将其数据传递给形式参数。但是，这个数据不再是简单的数值而是一个存储单元的地址，即传递的是实际参数的引用。当方法调用结束返回时，无论如何改变形式参数的引用，都是不会改变形式参数的引用，但是是否会影响实际参数引用空间中变量的值，则要分成两种情况来讨论。

情况一：在方法中，引用类型的参数没有发生引用的改变，则形式参数对引用中的变量值改变自然会影响到实际参数引用中变量的值。

情况二：在方法中，引用类型的参数发生了引用改变，它就不再对实际参数引用空间变量进行操作，则该引用类型的参数无论再做任何处理都不会再对实际参数引用空间的变量产生影响。

举例：

|  |
| --- |
| dataDeliverTest.java |
| public class dataDeliverTest {  public static void main(String args[]) {  //实例化得到3个Point类对象  Point p = new Point(); //p对象用于充当实参  Point test = new Point(); //test对象用于测试  Point result = new Point();//result对象用于充当实参    //测试对象p的初始值  System.out.println("Object p's value:");  System.out.println("p.x=" + p.x + "\np.y=" + p.y);    int xVal = -1; int yVal = -1;    //验证参数为简单类型时的传递效果，表明简单类型实参不会被影响  p.getData(xVal, yVal);  System.out.println("Pass by Value:");  System.out.println("xVal=" + xVal + "\nyVal=" + yVal);    //验证参数为引用类型时的传递效果，表明引用类型实参所引用的空间中的数据会被改变  Location loc = new Location();  p.getLocation(loc);  System.out.println("Pass by reference:");  System.out.println("xVal=" + loc.x + "\nyVal=" + loc.y);    //给result对象赋初值  result.x = 1;  result.y = 2;    //该示例中参数p引用没有被改变，因此改变x，y值会使实参result受影响  test.setData(result);  System.out.println("Call the setData()");  System.out.println("result.x=" + result.x);  System.out.println("result.y=" + result.y);  //该示例中参数p引用被改变，因此改变x，y值，实参result不受影响  test.setRefer(result);  System.out.println("Call the setRefer()");  System.out.println("result.x=" + result.x);  System.out.println("result.y=" + result.y);  }  }  class Location{  int x, y;  }  class Point {  int x, y;  Point(){  x = 20;  y = 30;  }  void getData(int xValue, int yValue) {  xValue = 100;  yValue = 200;  }    void getLocation(Location locRef) {  locRef.x = x;  locRef.y = y;  }    void setData(Point p) {  p.x = 10;  p.y = 20;  }    void setRefer(Point p) {  p = new Point();  p.x = 100;  p.y = 200;  }  } |
|  |

5.方法过载是指什么，创建方法时如何才能方法过载？举例说明。

方法过载也可以称为方法重载。它是指在一个类中有多个方法同名。方法过载是Java面向对象程序设计的多态性的一种表现。

方法过载的方法具有相同的方法名，但是它们的参数必须不同。对于方法参数的不同是从两个方面来区分的：

①参数的数量。当两个方法的参数数量不同时，它们可以构成方法过载；

②参数的对应数据类型。当方法参数数量相同时，则必须相对应的参数数据类型中至少有一个不同，它们可以构成方法过载。

举例：

|  |
| --- |
| void method(int x, int y)  void method() //参数数量不同  void method(float x, int y)//参数类型不同 |

6.实例成员和类成员有什么不同，它们如何使用？

①类成员是指由static修饰的成员，也可以称为静态成员，它属于类而不属于对象。

类成员的访问由类名直接实施，其形式为：

<类名>.<类成员名>

类成员也是可以通过该类的对象来访问类成员的。

②实例成员是指没有用static修饰的成员，它是由对象引用类的实例的成员，它属于对象，为每一个对象所拥有。

实例成员的访问由实例引用的对象名或者实例访问，其形式为：

<对象名>.<实例成员名>

实例成员不能由该类的类名访问。

7.对象与类的实例是什么关系，对象的引用是指什么？举例说明。

对象是类的实例，对象变量是对类的实例引用，类为对象的创建提供了“模板”。

对象实例的引用可以分为三个阶段，即创建、使用和回收。

一个声明的对象必须对实例引用，实例的创建是通过使用new运算符调用构造方法得到。对象对实例的引用称为对象的实例化。

8.对象是如何创建和使用的，又是什么情况下回收的？

①一个声明的对象必须对实例引用，实例的创建是通过使用new运算符调用构造方法得到。对象对实例的引用称为对象的实例化。声明一个对象就是创建一个变量，其标识符的存储单元保存对类的实例引用的地址，以通过对象的操作实现访问实例化的成员。多个对象通过消息传递进行交互，实现任务处理。所谓消息传递是指激活指定的某个对象的方法，即对象调用其方法，以改变它的状态或使其产生一定的动作。

②Java的JVM会自动地对不再被对象引用的实例存储空间进行内存垃圾搜集和清除，而为系统节省开销。

9.创建一个日期类MyDate

|  |
| --- |
| 1）源程序  public class S4E9{  public static void main(String args[]){  MyDate md = new MyDate();  md.plusDate("2018", "10", "11");  md.plusDate("2018", "10", "12");  md.plusDate("2018", "10", "13");  md.showDate();  System.out.println("改变其中一个日期:");  md.setDate(0,"2017", "10", "11");  md.showDate();  }  }  class MyDate{  final int MAXSIZE = 100;  public String[] dates = new String[MAXSIZE];  MyDate(){  for(int i = 0; i < MAXSIZE; i++){  dates[i] = "-1";  }  }    void setDate(int n, String y, String m, String d){ //设置第 n 个日期  if(dates[n] == "-1"){  System.out.println("Is Empty");  }  else{  dates[n] = (y + "-" + m + "-" + d);  }  }    void plusDate(String y, String m, String d){ //增加日期  for(int i = 0; i < MAXSIZE; i++){  if(dates[i] == "-1"){  dates[i] = (y + "-" + m + "-" + d);  return;  }  }  System.out.println("Is Full");  }    void showDate(){ //展示日期  for(int i = 0; i < MAXSIZE; i++){  if(dates[i] == "-1"){  break;  }  System.out.println(dates[i]);  }  }  } |
| 2）运行结果：   |  | | --- | | 2018-10-11  2018-10-12  2018-10-13  改变其中一个日期:  2017-10-11  2018-10-12  2018-10-13 | |
| 3）实验结论：测试正确 |

10.编写一个类ArraySort

|  |
| --- |
| 1）源程序  public class TestArray{  public static void main(String args[]){  int[] array = {3, 1, 4, 7, 5};  ArraySort as = new ArraySort(array);  as.setOrder();  for(int i=0; i<as.sim.length; i++){  System.out.print(as.sim[i] + " ");  }  }  }  class ArraySort{  int[] sim = new int[5];    ArraySort(int[] array){  for(int i=0; i<sim.length; i++){  sim[i] = array[i];  }  }    void setOrder(){  for(int i = 0; i < sim.length - 1; i++){  for(int j = 0; j <sim.length - i -1; j++){  if(sim[j] > sim[j+1]){  int temp = sim[j];  sim[j] = sim[j+1];  sim[j+1] = temp;  }  }  }  }  } |
| 2）运行结果：   |  | | --- | | PS C:\Users\knsugit\Desktop\javaWork\expReport\exp2\S4E10> java TestArray  1 3 4 5 7 | |
| 3）实验结论：测试正确 |

11.创建一个类Point

|  |
| --- |
| 1）源程序  public class TestPoint{  public static void main(String args[]){  Point p = new Point();  p.getPoint();  p.movePoint(10, 20);  p.getPoint();  }  }  class Point{  int x, y;  Point(){  x=0;  y=0;  }    void setX(int x){  this.x = x;  }    void setY(int y){  this.y = y;  }    void getPoint(){  System.out.println("(" + x + "," + y + ")");  }    void movePoint(int x, int y){  this.x += x;  this.y += y;  }    } |
| 2）运行结果：   |  | | --- | | PS C:\Users\knsugit\Desktop\javaWork\expReport\exp2\S4E11> java TestPoint  (0,0)  (10,20) | |
| 3）实验结论：测试正确 |