知识回顾

2022年7月7日 8:39

一、复习知识点

1. 继承：面向对象的三大特征之一，子类 extends 父类，可以继承父类的公共属性以及方法；继承只能单继承、继承提升了代码的重用性
2. 方法重写：子类继承了父类的方法之后，可以根据自身的业务需求重新定义继承到的方法；称为方法的重写，保留方法签名但是方法体中的代码块是子类自己的。 重写发生在父子类之间
3. super关键字：super代表着父类对象，可以在子类中通过super来调用父类的属性或方法（构造方法）
4. 构造方法：Type name = new Type(); 构造方法定义：与类同名、访问权限为public、无返回值类型的，例如public Type(){}；构造方法可以重载，可以存在若干个传入不同参数列表的构造方法；构造方法在类被实例化时被调用，通常需要在实例化时就被初始化的属性会在构造方法中完成初始化工作
5. 方法重载：发生在同一个类当中，可以定义若干个同名的方法，但是方法的参数列表需要不相同（参数个数、参数类型）
6. Debug模式：
   a. 第一步：打断点，toggle breakpoint；在需要查验的代码行处打断点
   b. 第二步：右键通过Debug runAs来运行程序，会进入到Debug视图中
   c. 第三步：通过variables窗口来查看变量值，是否符合需求预期
   d. 第四步：通过step over(F6)来执行断点，一行一行的执行代码
7. 循环：for & while，循环变量初始化、循环变量条件判断、循环变量的改变
8. 随机数：Random类、Math.random()

二、项目涉及的新知识

1. 线程：
   a. 首先通过implements Runnable接口，将类声明为一个线程类，实现其中的run()方法定义当前线程要执行的业务逻辑
   b. 然后通过Thread t = new Thread(线程对象)包装线程类，处理为一个等待抢占CPU执行权的线程
   c. 最后调用t.start()方法，执行线程
2. 异常处理：
   a. try{ //代码块 }catch(Excecption e){ //异常处理}
3. 数组：
   a. 引用数据类型：类、接口、数组、String等
   b. 声明并初始化，例如：int [] arr = new int[n]; 或者先声明int [] arr; 再初始化arr= new int[n];
   c. 通过下标来获取数组中的数据：arr[index]; index的取值范围在[0,n-1]

案例5：反弹的小球

2022年7月7日 10:27

一、案例需求：

1. 需求：创建窗体，设置画布，在窗体的画布上绘制一个黑色的实心小球；然后让这个小球沿着某个方向运行，当碰到四周的墙壁之后反弹，继续运行，一直在窗体中移动
2. 分析：
   a. 创建窗体，设置画布，绘制一个实心的黑色小球
   b. 要使得小球运行起来，需要通过修改小球的坐标来实现；
      i. 上下移动：修改Y轴坐标
      ii. 左右移动：修改X轴坐标
      iii. 斜着沿着和水平线程45°直线移动：同时修改X轴、Y轴的坐标值，并且修改的增量或减量相同

二、案例实现：

1. 创建窗体：

```
public class MoveBallFrame extends JFrame{

    public static void main(String[] args) {
        MoveBallFrame frame = new MoveBallFrame();
        frame.setSize(1024, 768);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setTitle("反弹的小球");
        frame.setVisible(true);
    }
}
```

2. 创建画布，在画布中绘制一个实心黑色小球：

```
public class MoveBallPanel extends JPanel{
//     设置小球的初始坐标
    private int bx = 20;
    private int by = 20;

    public void paint(Graphics g) {
        super.paint(g);
        this.setBackground(Color.pink);
        g.setColor(Color.BLACK);
//            根据坐标画绘制一个黑色实心小球
        g.fillOval(bx, by, 30, 30);
    }
}
```

在Frame的main中将画布添加到窗体上，实现绘制：

```
MoveBallPanel panel = new MoveBallPanel();
```

frame.add(panel);

3. 移动小球：让MoveBallPanel实现Runnable接口，重写run方法，并在run方法中修改小球的x\y坐标值
   让小球沿着水平线的45°∠的右下方移动：bx++、by++

```
public class MoveBallPanel extends JPanel implements Runnable{
    @Override
        public void run() {
            while(true) {
                bx++;
                by++;
                try {
                    Thread.sleep(30);
                } catch (InterruptedException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
                repaint();
            }
        }
}
```

   在Frame的main方法中添加Thread：

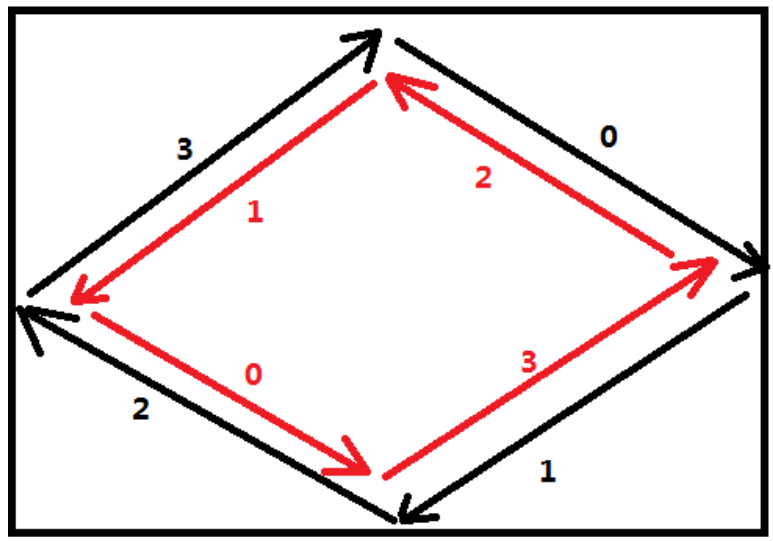
```
Thread t = new Thread(panel);
t.start();
```

4. 此时的小球碰到墙壁之后未反弹，需要实现让小球撞墙之后回头，在窗口一直运动下去。
   a. 要判断四周的墙壁在哪里
   b. 碰到墙之后 ，小球应该反弹回去；
      假如小球由右下运动碰到了下方的墙壁，则需要往右上移动：bx++ by--

```
//                  碰到下方的墙壁，by=768
                if(by>768) {
                    bx++;
                    by--;
                }
```

   c. 发现并没有成功实现反弹！思考解决方案
      i. 方案一：
         1) 小球的坐标是左上角的坐标，判断by和768比较，必须是小球都已经进入到了下方的墙壁才能够符合条件
         2) 设置窗口的大小是1024*768，此值是窗口整体的大小，而窗口的粉色区域要比这个值小，大概是1007*729（此处横向的窗口边框大约是17px、纵向大约是39px）
         3) 因此修正之后判断墙壁的值应该是：by>(729-小球直径)
         4) 此方案存在bug：y--之后立马又被循环++回去了，并且此时x++了两次，小球依旧没有被弹回去，而是沿着墙壁从右滑落消失了
      ii. 方案二：

1) 小球会碰撞四面墙体，可以通过判断小球碰到了哪个墙壁来处理小球反弹



2) 添加对四面墙壁接触的方位变量的控制，并通过判断墙壁来移动小球

```
//     设置四面墙壁接触的方位：0：右下、1：左下、2：左上、3：右上
      private int att = 0;
在run方法中定义
//            45°∠分别向 0：右下、1：左下、2：左上、3：右上移动
            while(true) {
                if(att == 0) {
                    bx++;
                    by++;
                }
                if(att == 1) {
                    bx--;
                    by++;
                }
                if(att == 2) {
                    bx--;
                    by--;
                }
                if(att == 3) {
                    bx++;
                    by--;
                }
            }
```

3) 每个墙有两个可能的小球飞来的方向，也存在两个反弹出去的方向，所以反弹的代码需要对飞来的方向进行判断：

在小球坐标修改的判断语句之后添加以下代码：

```
//判断小球碰上了哪边墙壁，并且改变小球反弹的方向（两个方向）
if(bx>987){
```

```
                if(att == 0){
                    att = 1;
                }else{
                    att = 2;
                }
            }
            if (by > 709) {
                if (att == 1) {
                    att = 2;
                } else {
                    att = 3;
                }
            }
            if(bx < 0) {
                if(att == 2) {
                    att = 3;
                }else {
                    att = 0;
                }
            }
            if(by < 0) {
                if(att == 3) {
                    att = 0;
                }else {
                    att = 1;
                }
            }
```

三、知识点总结

1. if条件判断常见的格式：

```
if(条件表达式 | 布尔值){
    //true：代码块
}

if(条件表达式){
    //true：代码块
}
else{
    //false：代码块
}
if(条件表达式1){
    //true：代码块
```

```
}else  if(条件表达式2){
      //true：代码块
}else{
      //false：代码块
}
```

2. =表示赋值，==表示比较两个值是否相等

案例6：下雪

2022年7月7日 15:53

一、案例需求：

1. 需求：创建并显示窗体；添加一个绘制了漫天雪花（白色的*字符串）的画布，然后想办法让雪花纷纷落下，实现下雪的效果
2. 分析：
   a. 第一步实现漫天雪花：自行实现
   b. 第二步雪花不停下落：实现漫天雪花的基础上，添加线程让雪花下落，再加上循环让雪花不停的产生并下落，考虑雪花缓慢下落，重画雪花。

二、案例实现：

1. 画300片雪花，定义好SnowPanel，然后在其中生成300片雪花，初始坐标随机生成并保存在数组中

```
//     定义数组保存雪花的初始坐标
    private int [] sx = new int[300];
    private int [] sy = new int[300];
```

定义构造方法，并在其中初始化随机坐标值，添加到数组当中

```
public SnowPanel() {
    Random ran = new Random();
    for (int i = 0; i < sx.length; i++) {
        sx[i] = ran.nextInt(1024);
        sy[i] = ran.nextInt(768);
    }
}
```

在paint方法中绘制：

```
//          设置画笔的颜色为白色
        g.setColor(Color.white);
//          设置画笔的字体
        Font font = new Font("宋体",Font.BOLD,10);
        g.setFont(font);
//          绘制
        for (int i = 0; i < sx.length; i++) {
            g.drawString("*", sx[i], sy[i]);
        }
```

此时得到的300片雪花的大小都是一样的，但是实际上应该是各有不同，可以通过数组的形式定义若干个不同大小的画笔的字体Font，在绘制雪花的时候，从字体数组中随机抽取一个，这样就可以得到大小不一致的雪花

在sy数组的下方添加字体数组的定义：

```
//     画笔的字体数组：当前数组可以存储四个Font类型的对象
    private Font [] allFonts = new Font[4];
```

在SonwPanel构造方法中添加字体数组的初始化

```
//     字体：创建四个字体对象，然后保存到字体数组allFonts
    allFonts[0] = new Font("宋体",Font.BOLD,10);
```

```
        allFonts[1] = new Font("宋体",Font.BOLD,16);
        allFonts[2] = new Font("宋体",Font.BOLD,22);
        allFonts[3] = new Font("宋体",Font.BOLD,28);
```

将原先paint方法中的单个Font对象删除，修改为在for循环中从字体数组中随机获取一个字体对象：

```
        for (int i = 0; i < sx.length; i++) {
//              设置画笔的字体：从字体数组中随机获取
//              bound:一定要是正整数，是上限（此上限会被排除）
            g.setFont(allFonts[ran.nextInt(4)]);
            g.drawString("*", sx[i], sy[i]);
        }
```

2. 让雪花落下