案例6：下雪

2022年7月8日 8:39

一、案例实现

1. 让雪花落下：通过线程和接口实现雪花下落

   实现Runnable接口，重写run方法，在其中先执行重画画布

```
public class SnowPanel extends JPanel implements Runnable{
    @Override
    public void run() {
        while(true) {
            try {
                Thread.sleep(10);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
//                  重画画布
            repaint();
        }
    }
}
```

   实现雪花的下落则需要修改y轴的坐标值，所有的300片雪花纵坐标递增，在while循环里再写一个for循环：

```
for (int i = 0; i < sy.length; i++) {
    sy[i]++;
    //雪花不停的下落
    if(sy[i]>768) {
        sy[i]=0;
    }
}
```

二、案例总结

1. 数组：存储大量相同类型的数据；是一个批量声明变量的工具。也即类型一旦固定是不能改变
   a. 获取数组的长度，通过属性值length来获取即可：例如arr.length
   b. 可以尽量避免出现数组下标越界的问题！

案例7：碰撞的小球

2022年7月8日 9:21

一、案例需求

1. 需求：创建窗口，创建一个画布，然后在上面绘制两个小球，让两个小球在窗口中分别沿着某个方向运动，遇到墙壁之后按照一定的反射角度反弹（45°），一直运动到两个小球碰撞为止，并且碰撞之后两个小球其中一个小时，另一个变大，然后停止运动。
2. 分析：窗体的画布上有两个小球，然后分别让两个小球运动起来，并且在碰撞到球之后能够符合物理规律继续运动，直到两个球碰撞后合并到一起
   a. 窗体上的画布
   b. 画布上绘制一个小球
      i. 让小球斜着飞起来
      ii. 遇到墙反弹
   c. 再绘制一个小球，起点和飞行方向与前一个不同，其他的实现基本一致
   d. 可以考虑将小球抽象成一个类，将坐标 和飞行方向抽象为属性，将飞行和碰撞抽象为方法；实现两个小球则可以创建两个属性值不同的对象即可
   e. 当两个小球碰撞到一起，其中一个小球消失，另一个小球变大

二、案例实现步骤与代码：

1. 首先实现一个球斜着飞，碰到墙之后反弹，分解成四步：窗口、画布上的小球、小球斜着飞、遇墙反弹；此步骤的实现思路与代码参考案例5即可。目前只实现了一个小球的飞行，想实现两个小球的飞行，可以重新定义一次小球的信息，再画一个

```
//     第一个小球：坐标和飞行方向
    private int x = 20;
    private int y = 20;
    private int att = 0;
//     第二个小球：坐标和飞行方向
    private int x1 = 400;
    private int y1 = 400;
    private int att1 = 3;
```

2. 定义Ball类：

   此时两个小球的飞行代码是除了数值之外完全一致的，需要实现3个、4个、若干个小球的发行，如果还采用重新定义的方式，则实现了很多冗余的繁琐的代码；因此为了避免代码的冗余提供重用性，可以考虑将特征相同、行为相同的小球定义到一个类当中（Ball类），从而在需要小球的时候创建一个小球Ball类对象即可，开发Ball类：

   a. 属性：小球初始位置坐标x\y、小球的半径、运动方向、运动速度、颜色、画布

```
public class Ball {
//     半径
      private int r;
//     坐标
```

```
        private int x,y;
    //    颜色
        private Color color;
    //    运动方向：0表示左上、1表示左下、2表示右上、3表示右下
        private int orientation;
    //    运动速度
        private int speed;
    //    小球所在的画布
        private JPanel panel;
    //    当属性私有化之后，需要提供公共的set&get方法：alt+shift+s，选择构建set&get即可
    }
```

此时运动方法需要使用的具体运动方向，入如果仅仅通过数字0\1\2\3表示，在类的外部调用的时候容易混乱，可以参考Color.PINK等颜色的设置方式：

```
        public final static Color pink      = new Color(255, 175, 175);//RGB设置无法区分具体颜色的
        public final static Color PINK = pink;//常量所指向的值就是粉色的RGB坐标
```

通过java提供的常量的概念来定义具体的运动方向值

格式：public final static 类型 常量名称=常量值 //通常常量名称需要大写，后续使用“类名.常量名”的方式调用常量值，在Ball类中添加：

```
    //    具体运动方向的常量：修饰的东西不能被修改；修饰属性一定要声明的同时被初始化
        public final static int LEFT_UP = 0;
        public final static int LEFT_DOWN = 1;
        public final static int RIGHT_UP = 2;
        public final static int RIGHT_DOWN = 3;
```

b. 方法：

   i. 小球属性初始化的构造方法：

```
    //    无参构造
        public Ball() {    }
    //    初始化小球部分属性的有参构造
        public Ball(int r, int x, int y, Color color, int orientation) {
            this.r = r;
            this.x = x;
            this.y = y;
            this.color = color;
            this.orientation = orientation;
        }
```

   ii. 绘制小球的方法：

```
    //    绘制小球的方法：
        public void draw(Graphics g) {
            g.setColor(color);
            g.fillArc(x, y, 2*r, 2*r, 0, 360);
```

```
}
```

iii. 小球移动的方法：

1) 根据创建小球对象时所传入的移动方向，来判断不同的小球应该向哪个方向飞行

```
switch(orientation){
    case left_up:
        x--；
        y--;
        //TODO 反弹
        break;
    ……
}
```

2) 完整的飞行方法：

```
//    小球飞行的方法：
    public void move() {
        switch(orientation) {
//          初始方向为左上时：
        case LEFT_UP:
            x--;
            y--;
//              反弹
            if(x <= 0) {//碰到左边的墙壁：弹向右上
                this.setOrientation(RIGHT_UP);
            }
            if(y <= 0) {//碰到上边的墙壁：弹向左下
                this.setOrientation(LEFT_DOWN);
            }
//              中断，重新switch
            break;

//          初始方向为左下时：
        case LEFT_DOWN:
            x--;
            y++;
            if(x <= 0) {
                this.setOrientation(RIGHT_DOWN);
            }
//              y轴的判断：需要获取小球所在的画布高度减去小球的直径
            if(y >= panel.getHeight()- 2*r) {
                this.setOrientation(LEFT_UP);
            }
```

```
                    break;

            //          初始方向为右上时：
                case RIGHT_UP:
                    x++;
                    y--;
            //              x轴的判断：需要获取小球所在的画布宽度减去小球的直径
                    if(x >= panel.getWidth()-2*r) {
                        this.setOrientation(LEFT_UP);
                    }
                    if(y <= 0) {
                        this.setOrientation(RIGHT_DOWN);
                    }
                    break;

            //          初始方向为右下时：
                case RIGHT_DOWN:
                    x++;
                    y++;
                    if(x >= panel.getWidth()-2*r) {
                        this.setOrientation(LEFT_DOWN);
                    }
                    if(y >= panel.getHeight()- 2*r) {
                        this.setOrientation(RIGHT_UP);
                    }
                    break;
                }
            }
```

3. CrashBallFrame类：显示窗口以及内容、

```
public class CrashBallFrame extends JFrame{
    public static void main(String[] args) {
        CrashBallFrame frame = new CrashBallFrame();
        frame.setSize(1024, 768);
        frame.setTitle("碰撞的小球");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
        CrashBallPanel panel = new CrashBallPanel();
        frame.add(panel);
        Thread t = new Thread(panel);
        t.start();
```

```
        }
    }
```

4. CrashBallPanel类：绘制小球的类，完成了两个小球的运动和反弹
    a. 属性：传入一个小球类的数组（用于存储若干个小球对象）以及颜色数组

```
    //    小球类数组
        private Ball [] allBalls;
    //    小球颜色的数组
        private Color [] colors;
    //    随机数
        private Random ran;
```

    b. 构造方法：完成初始化

```
    //    构造方法中完成属性初始化
        public CrashBallPanel() {
            colors = new Color[2];
            colors[0] = Color.PINK;
            colors[1] = Color.YELLOW;
            /*
             *    小球初始化（两种方法：new Ball()-set或者new Ball(参数)）：
             *    1、小球数组的初始化，
             *    2、然后通过循环创建两个小球对象，半径r一致为20
             *    3、分别为两个小球对象设置随机初始x\y坐标
             *    4、颜色从颜色数组中按顺序取出并赋值
             *    5、初始的运动方向：0 | 1
             *    6、小球的运动速度：两个小球的运动速度不相同，在0-55毫秒之间
             *    7、小球所在的画布：this
             */
            allBalls = new Ball[2];
            ran = new Random();
            for (int i = 0; i < allBalls.length; i++) {
    //                方式一：无参构造
                Ball ball = new Ball();
                ball.setR(20);
                ball.setX(ran.nextInt(1024 - 2*ball.getR()));
                ball.setY(ran.nextInt(768 - 2*ball.getR()));
                ball.setColor(colors[i]);
    //                在0-100之内随机生成数字取余2，结果只有0或1
                ball.setOrientation(ran.nextInt(100) % 2);
    //                设置小球在运动中重画的速度：传给Thread.sleep(speed);
                ball.setSpeed(ran.nextInt(50)+5);
                ball.setPanel(this);
```

```
//                  方式二：有参构造
                Ball ball2 =
                        new Ball(20,ran.nextInt(1024 - 2*ball.getR()),
                        ran.nextInt(768 - 2*ball.getR()),colors[i],ran.nextInt(100) % 2,
                        ran.nextInt(55),this);
//                  将生成的小球对象存储到小球数组中
                allBalls[i] = ball;
            }
        }
```

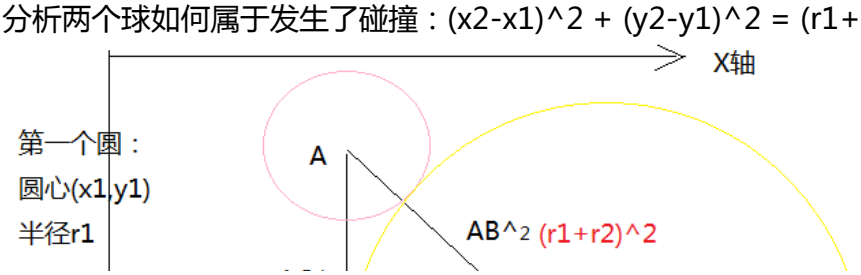
c. paint方法：完成两次小球的绘制，调用小球类中的draw方法即可

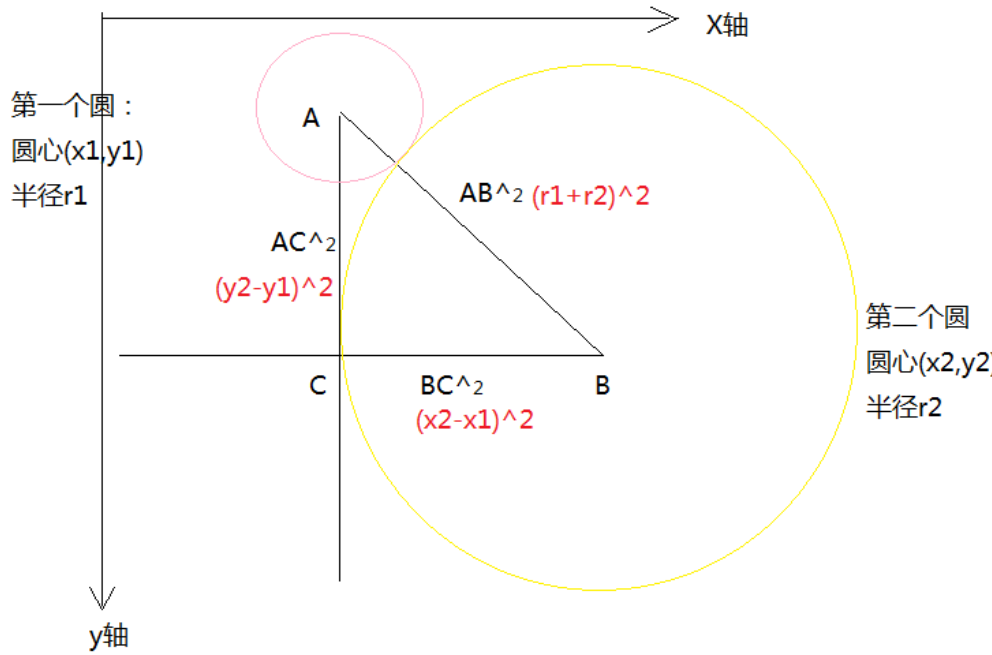
```
@Override
public void paint(Graphics g) {
    super.paint(g);
    this.setBackground(Color.DARK_GRAY);
    for (int i = 0; i < allBalls.length; i++) {
        allBalls[i].draw(g);
    }
}
```

d. run方法：完成两个小球的飞行移动，调用小球类的move方法即可，并且设置不同的速度

```
@Override
    public void run() {
        while (true) {
            for (int i = 0; i < allBalls.length; i++) {
                allBalls[i].move();
                try {
                    Thread.sleep(allBalls[i].getSpeed());
                } catch (InterruptedException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
            repaint();
        }
    }
```

5. 实现：当两个运动中的小球相撞之后，让其中一个变大，另一个变小
   a. 分析两个球如何属于发生了碰撞：(x2-x1)^2 + (y2-y1)^2 = (r1+r2)^2





b. 在CrashBallPanel中添加一个判断两个小球是否相撞的方法：

(x2-x1)^2 + (y2-y1)^2 = (r1+r2)^2 转换为判断 (x平方+y平方)开方之后的值是否小于等于r

```
//     判断两个小球是否碰撞：true表示碰撞、false表示未碰撞
    public boolean twoBallsHit(Ball ballOne,Ball ballTwo) {
//          定义一个标记位：未相撞
        boolean isHit = false;
        int bxot = ballOne.getX() - ballTwo.getX();
        int byot = ballOne.getY() - ballTwo.getY();
        int rot = ballOne.getR() + ballTwo.getR();
//          判断小球碰撞
        double z = Math.sqrt(bxot*bxot+byot*byot);
        if(z <= rot) {
            isHit = true;//此时小球相撞
        }
        return isHit;
    }
```

此处注意判断条件不写成if(z==rot)，此时两个小球的移动是跳跃式的，不是连续的；如果使用相等来判断会导致有时候即使两个小球相撞到了一起，也返回不了true

c. 在run方法的for循环中try..catch后调用判断小球是否相撞的方法，传入数组中的两个小球：

```
//调用判断两个球相撞的方法
boolean isHit = twoBallsHit(allBalls[0],allBalls[1]);
```

d. 在run方法的判断相撞之后，根据相撞的结果是否为true，来设置小球变大变小

```
if(isHit) {
//碰撞之后将一个小球的半径变为0，另一个小球的半径加10(或者另一个小球的半径)
    allBalls[1].setR(allBalls[1].getR()+allBalls[0].getR());
    allBalls[0].setR(0);
    allBalls[0].setX(0);
```

```
        allBalls[0].setY(0);
    }
```