案例2：画星星

2022年7月6日 8:37

一、案例实现

1. 方式二：绘制实心圆和直线，实现画星星

```
//          绘制黄色实心圆
            g.fillOval(200, 200, 20, 20);
//          在圆的周围绘制一些黄色的直线
//          水平方向上的线
            g.drawLine(190, 210, 230, 210);
//          垂直方向上的线
            g.drawLine(210, 190, 210, 230);
            int a = (int) (210-Math.sqrt(20*20/2));//195
            int b = (int) (210+Math.sqrt(20*20/2));//224
//          左上到右下
            g.drawLine(a, a, b, b);
            g.drawLine(a, b, b, a);
```

2. 方式三：绘制图片

```
//          绘制图片：注意路径，图片位于工程下直接star.jpeg ，位于src下 src/star.jpeg
            Image img = new ImageIcon("src/night.jpeg").getImage();
//          x&y表示左上角的坐标，height&weight表示图片的大小
            g.drawImage(img, 0,0, 600, 600, this);
```

3. 总结：
   a. 类与类之间的关系：继承、父子类
   b. super关键字：表示父类对象
   c. 方法重写：覆盖继承父类方法，改为调用自己重写后的方法
   d. static关键字：静态
   e. 画布&画笔类

案例3：画满天星

2022年7月6日 9:51

一、案例需求

1. 需求：创建一个图形化界面，将其显示在屏幕上，设置其大小、标题、默认关闭等属性，在窗口上添加一个画布，绘制出黑色的天空中出现满天繁星；使用案例2中的方式一绘制*字符串的方式随机在画布上显示出300颗星星。
2. 分析：
   a. 创建窗体
   b. 创建画布，添加画笔，完成300颗星星的绘制：先画出一颗，然后循环执行
   c. 将画布添加到窗体上，并显示出窗体

二、案例实现

1. 先创建窗口：

```
public class StarSkyFrame{

    private JFrame jframe;

    public void showMe() {
        jframe = new JFrame();
        jframe.setSize(600, 800);
        jframe.setTitle("满天星");
        jframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jframe.setVisible(true);
    }

    public static void main(String[] args) {
        StarSkyFrame ssf = new StarSkyFrame();
        ssf.showMe();
    }

}
```

2. 创建画布，完成一颗星星的绘制：

```
public class StarSkyPanel extends JPanel{

    @Override
    public void paint(Graphics g) {
//            调用父类的paint绘制一张空画布，并设置背景颜色为黑色
      i.  super.paint(g);
        this.setBackground(Color.black);
```

```
//              字符串形式绘制一颗黄色的星星
            g.setColor(Color.YELLOW);
            Font font = new Font("宋体",Font.BOLD,20);
            g.setFont(font);
            g.drawString("*", 200, 210);
        }
    }
```

添加画布到窗体上：

```
        StarSkyPanel ssp = new StarSkyPanel();
    //              在静态方法中不能直接调用非静态的变量或方法，需要通过对象调用
        ssf.jframe.add(ssp);
```

3. 随机绘制出300颗星星：

```
    //              画300颗星星
            for(int i = 0;i < 300 ; i++) {
    //                  获取double类型的随机数，并且强制转换成int类型
                int x = (int) (Math.random()*1024);//0~1024之间的随机数
                int y = (int) (Math.random()*768);//0~768之间的随机数
                g.drawString("*", x, y);
            }
```

4. 知识点总结：
   a. 随机数：Math.random()生成[0,1]之间的随机双精度小数，Math.random()*n表示生成[0,n]之间的随机双精度小数
   b. 强制类型转换：由类型大的数（如double类型）转换（赋值）成类型小的数（如int类型），需要使用强制类型转换实现：小类型变量 = (小类型)大类型变量
   c. for循环：三个要素，变量初始化表达式（执行一次）、变量条件判断表达式（true则执行循环代码块、false则退出循环）、变量修改表达式（通常是递增++或递减--）

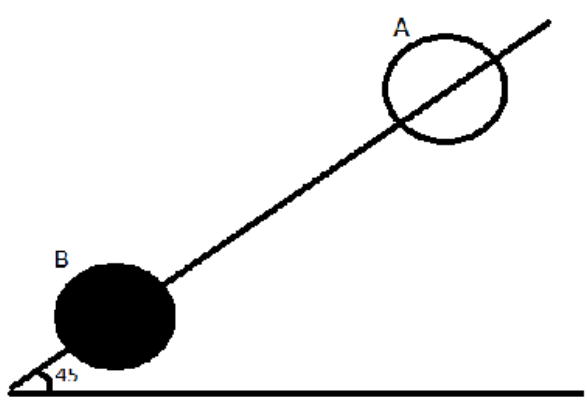
```
for(初始化；条件；改变){
}
变形为
初始化;
for(;条件;){
    //循环体
    //循环变量的改变
}
```

案例4：月食

2022年7月6日 11:00

一、案例需求

1. 需求：创建一个图形化窗口，显示并设置大小、标题、默认关闭等，在窗体的画布上实现夜空中的满天繁星；再此基础上绘制一轮明亮的圆月，然后动态的显示月食的全部过程
2. 分析：
   a. 在窗口的画布上绘制出漫天星星，然后在画布的右上角绘制一轮明亮的月亮A
   b. 再绘制一个和圆月A一样大小的实心圆B，让实心圆B在窗口中实现移动
   c. 让实心圆B在月亮A的左下方沿着右上45°移动，慢慢的经过月亮A，然后将实心圆B的颜色改为黑色



二、案例实现

1. 在窗口中绘制出满天星，然后在窗体的右上方绘制一轮圆月A：

```
public class MoonPanel extends JPanel{

    public void paint(Graphics g) {
        super.paint(g);
        this.setBackground(Color.BLACK);
        g.setColor(Color.YELLOW);
        Font font = new Font("宋体",Font.BOLD,10);
        g.setFont(font);
//          画星星
        for(int i = 0;i < 300 ; i++) {
//              获取double类型的随机数，并且强制转换成int类型
            int x = (int) (Math.random()*1024);//0~1024之间的随机数
            int y = (int) (Math.random()*768);//0~768之间的随机数
            g.drawString("*", x, y);
        }
//          画圆月A：绘制一轮直径为80的圆月
        g.fillOval(580, 100, 80, 80);
    }
```

```
}
```

2. 再绘制一个和圆月A相同大小的实心圆B，在窗体中实现移动：
   a. 绘制：g.fillOval(280, 500, 80, 80);
   b. 移动：频繁的改变坐标：
      i. 45°向右上方移动：

```
int x =280 ;
        int y =400 ;
        for (int i = 0; i < 300; i++) {//x&y不断的移动，不是好的解决方案
            g.fillOval(x, y, 80, 80);
            x += 50;
            y -= 50;
        }
```

3. 利用线程的知识，解决：抹去原有的实心圆，重画一个新的。
   a. 概念：线程是通过利用CPU的轮转，让程序中不同的代码段同时执行的机制。
   b. 实现：
      i. 在MoonPanel类中添加一个run方法：

```
public void run() {


}
```

表示有一个新的线程需要在合适的时机抢占CPU执行权；也即run方法是这个程序的入口方法，想要让run方法成为操作系统可以识别到的线程的入口方法，则需要将所在的类声明为一个线程类；

      ii. 让当前MoonPanel类实现线程提供的Runnable接口，从而表示MoonPanel是一个线程机制的类

```
public class MoonPanel extends JPanel implements Runnable{
    //    线程的入口方法：
        public void run() {


        }
}
```

从而让操作系统知道此类中包含一个线程的入口方法run()

      iii. 需要通知操作系统，此处有一个线程需要获取CPU的执行权；需要在MoonFrame类的main方法中添加以下代码：

```
        MoonPanel ssp = new MoonPanel();
//          告知操作系统此时MoonPanel是一个线程类
        Thread t = new Thread(ssp);
        t.start();
```

Thread是线程类！！用Thread类包装一下应用||软件||程序，此时这个程序就是一个Thread的对象，ssp是MoonPanel对象的应用；也即将MoonPanel包装成了Thread对象，当此Thread对象t调用start()跑起来的时候，发现MoonPanel类里面有一个run()

      iv. 需要在run方法中实现画图，此时run方法中无法实现画图因为没有Graphics g，此时有g变量是paint方法的局部变量，只在paint方法中有效，依旧在paint中绘制初始的实心圆B
在MoonPanel类中定义：

```
//     声明并初始化实心圆B(黑色)的起始坐标
    private int x = 280;
    private int y = 400;
```

在paint方法中：

```
//              再绘制一个和圆月A相同大小的实心圆B(黑色)
            g.setColor(Color.BLACK);
            g.fillOval(x, y, 80, 80);
```

在run方法中改变实心圆B的坐标：

通过while循环实现实心圆B移动，结束循环设置一个条件判断坐标超过了月亮的坐标时则结束掉方法的执行（通过return;语句表示结束方法调用）

```
    while(true) {
        x++;
        y--;
    //反复执行月食的现象：x = 580;y = 400;
        if(x>=910 || y <= 80) {
            return;
        }
    }
```

在while循环中添加一个线程休眠的语句Thread.sleep(300); 此语句会出现编译期问题，按照提示通过try..catch抛出异常即可

```
try {
    Thread.sleep(300);
} catch (InterruptedException e) {
    e.printStackTrace();
}
```

最后在try..catch的后面写上重画的方法：

```
//                      重画方法
repaint();
```

v. 此时由于执行重画方法时，会重新调用paint()，其中的满天星的绘制也会重新执行，bug：满天星的坐标每次都会重新生成一对随机数，导致月食的同时星星也在移动；修改为在MoonPanel类的构造方法中完成初始化满天星的随机坐标

在MoonPanel类中定义：

```
//     声明满天星的随机坐标：用数组保存300颗星星的x-y坐标值
    private int [] sx;
    private int [] sy;
```

在其构造方法中初始化：

```
//     在构造方法中完成满天星坐标的初始化
    public MoonPanel() {
        sx = new int[300];
        sy = new int[300];
        Random ran = new Random();
```

```
            for (int i = 0; i < 300; i++) {
        //                  通过数组名[下标值]的方式来操作（添加||获取||删除）数组中的元素；下标从0开
        始
                    sx[i] = ran.nextInt(1024);//0~1024之间的随机数
                    sy[i] = ran.nextInt(768);//0~768之间的随机数
            }
        }
        在paint方法中绘制满天星：
        //          画星星
            for(int i = 0;i < 300 ; i++) {
                g.drawString("*", sx[i], sy[i]);
            }
```

vi. 至此，完成了月食的全过程！

三、案例总结

1. while循环：

```
循环变量初始化表达式; int 1 =0;
while(条件判断 i <100){
    //循环体
    1、循环代码块：例如坐标移动 sysout();
    2、循环变量：改变 i++;
}
特殊写法：
while(true){
    //需要在循环体中通过if判断，来决定循环何时结束
    if(条件){
        //true
        退出循环：break;、return;
    }
    //否则此while循环会是一个死循环
}
```

2. 随机数Random：java类库所提供的随机数的类，可以通过其提供的方法来获取整型、浮点型等随机数，例如nextInt(n)表示获取0-n的随机整数
3. 线程：
   a. 定义：通过利用CPU的轮转让程序中不同的代码段同时执行的机制
   b. 实现：
      i. 当前类要通过implements关键字实现Runnable接口，在当前类中实现run()方法作为线程的入口方法
      ii. 使用Thread类来包装实现了Runnable接口的：Thread t = new Thread(类对象);
      iii. 调用t.start()开启线程，从而执行线程的run方法
   c. 此处不使用继承Thread的形式，是由于继承只支持单继承，此时画布类已经继承过了JPanel，声明为了一个系统可以感知到的画布，就不能再同样的通过继承的形式来声明为了一个系统可以感知到的线程

类；转为采用实现接口的形式！

4. 接口：通过interface关键字声明的，称为接口；可以理解为java中一种特殊的类，不能被实例化。
5. 异常处理格式：
   a. 编译期间，编译器提示某行代码飘红，称为编译时异常
   b. 可以按照开发工具的提示来使用

```
try{
    //尝试执行可能会出现异常的代码
    Thread.sleep(30);
} catch(Exception e){
    //代码出现异常，会被catch捕获，进行异常的处理：例如打印异常信息
    e.printStackTrace();
}
```

6. 数组：
   a. 定义：
      i. 用于存储和管理一组相同类型数据的组合
      ii. 数组属于引用数据类型，通过new关键字创建对象
      iii. 数组在初始化的时候，一定要指明长度；也即最多能存储多少个数据
   b. 声明与初始化：
      i. 声明：type [] name; 例如int [] arr;
      ii. 初始化：name = new int[length]; 例如 arr = new int[300];
      iii. 可以声明的同时并初始化：int [] arr01= new int[100];
   c. 操作数组中的元素，通过数组名[下标]; 下标从0开始，到length-1结束