# Java 画图工具程序代码说明文档

同正南(2016551439)

## 目录



# 图索引

# 1. 该系统的结构(界面组成)可用图和文字描述:

## 1.1. 主界面及总体组成结构

MainFrame 类采用 BorderLayout 布局,分别设置了 MenuBar 菜单栏,工具栏 ToolBar 布置在主界面的北部,附加工具栏(AdditionalTool)布置在主界面的西部,画板 DrawBoard 类布置在主界面的中部,画笔位置坐标栏 PositionLabel 类布置在主界面的南部。

下面是我画图程序的主界面:

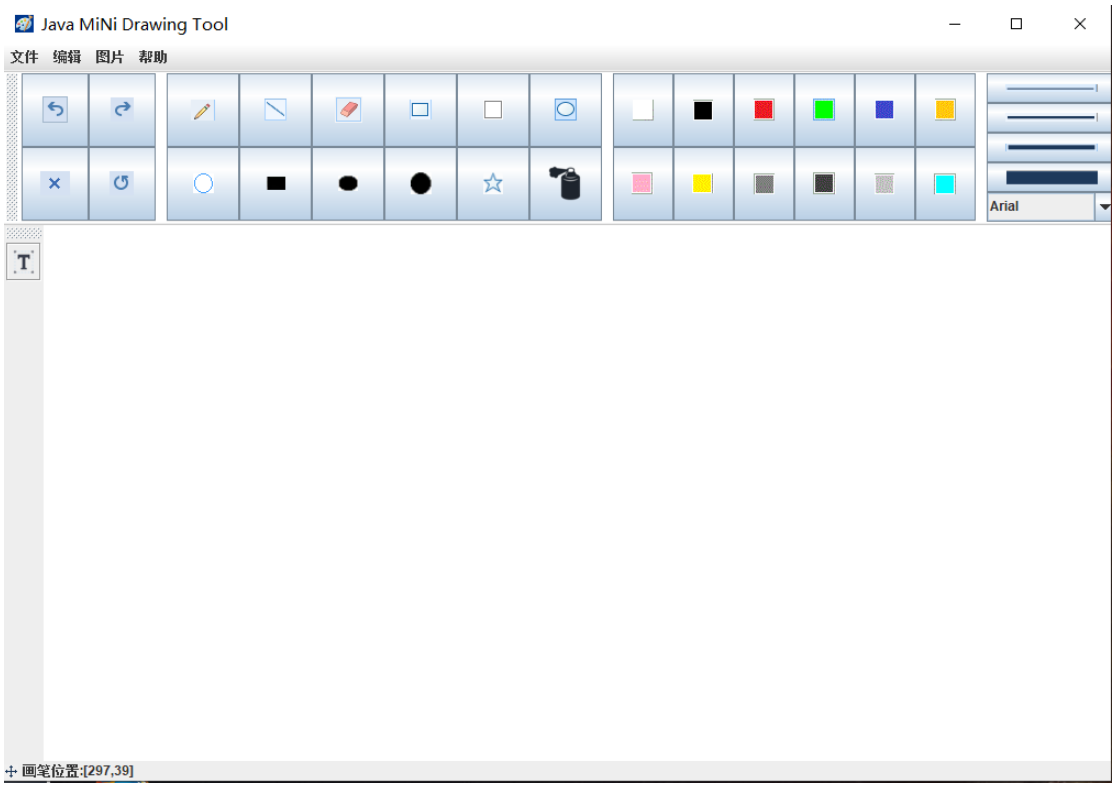


图 1.1-1 画图程序主界面

下面是我项目的目录树结构:

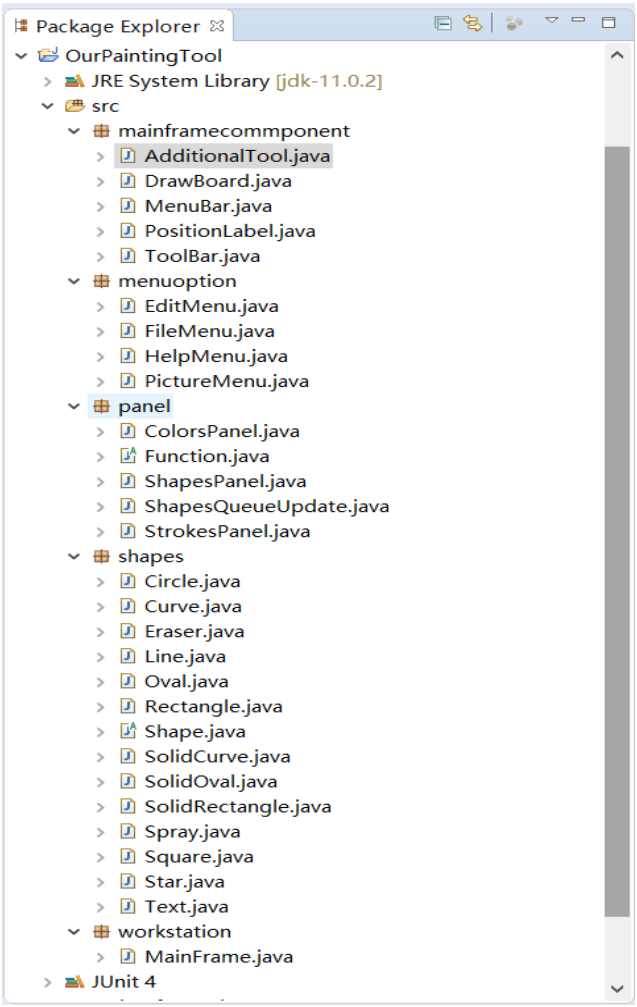


图 1.1-2 项目的目录树结构

下面是程序的结构图

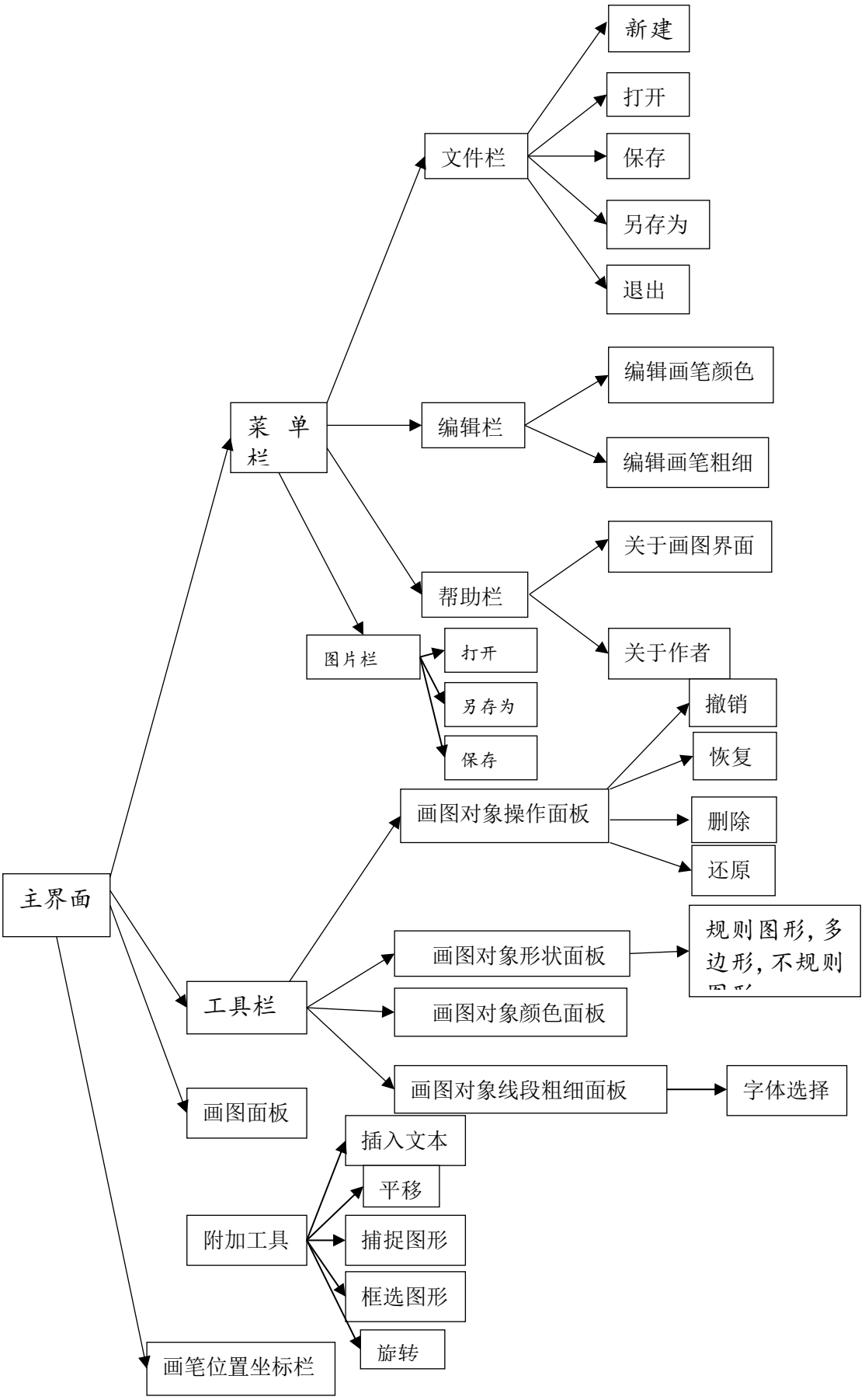
主界面
菜 单 栏
文件栏
新建
打开
保存
另存为
退出
编辑栏
编辑画笔颜色
编辑画笔粗细
帮助栏
关于画图界面
关于作者
图片栏
打开
另存为
保存
工具栏
画图对象操作面板
撤销
恢复
删除
还原
画图对象形状面板
规则图形，多边形，不规则
画图对象颜色面板
画图对象线段粗细面板
字体选择
画图面板
附加工具
插入文本
平移
捕捉图形
框选图形
旋转
画笔位置坐标栏

图 1.1-3 程序系统结构图

## 1.2. Panel(工具组成面板 ToolBar):

包括 ColorsPanel(画图对象颜色面板)、ShapesPanel(画图对象形状面板)、StrokesPanel(画图对象线段粗细面板)、ShapesQueueUpdate(画图对象操作面板),这里还有一个 Function 抽象类是作为一个功能函数去使用的,里面是一些画图过程中会使用到的静态方法,其中 ColorsPanel、ShapesPanel、StrokesPanel, ShapesQueueUpdate 都继承 Jpanel，他们都静态导入 Function 类以用来使用其中的静态方法。 这 4 个 Jpanel 都采用了网格布局。Jpanel 中的 Button 按钮都添加了选择监听器，一经触发，就会改变 Function 类中的对应的 Choice 变量属性，用以改变画图对象及其颜色、粗细，这里以 ShapesPanel 为例:

采用网格布局 GridLayout()

```
public ShapesPanel() {
    setLayout(new GridLayout(2,6));
    addShapeButtons();
}
```

一经触发，改变了 shapeChoice 属性。

```
@Override
    public void actionPerformed(ActionEvent e) {
        for(int i=0;i<shapeButtons.length;i++) {
            if(e.getSource()==shapeButtons[i]) {
                shapeChoice=i;
            }
        }
    }
```

关于 Function 类，主要是 creatAction()、newColor()、newStroke()、queueUpdate()
其中 creatAction()、newColor()、newStroke()都是在用户触发 button 按钮时根据改变的 choice 的值来选择当前所要创建的图形对象以及图形的颜色和画笔的粗细，而 queueUpdate()主要用来维护图形队列用来支撑图形的撤销、恢复、删除、复原功能，大体上是将图形队列中的对象移位覆盖，调整它在图形队列中的位置。

这里以 创建画笔粗细 newStroke()为展示，其它是类似的

```
protected static void newStroke() {
        switch(strokeChoice) {
            case 0:stroke=width[0];break;
            case 1:stroke=width[1];break;
            case 2:stroke=width[2];break;
            case 3:stroke=width[3];break;
        }
    }

    //对ShapesQueue的更新维护
public static void queueUpdate(int pastIndex,int newIndex) {

    if(shapesQueue[pastIndex].getClass()==Curve.class||shapesQueue[pastInde
x].getClass()==Eraser.class||shapesQueue[pastIndex].getClass()==Spray.class
) {
            if(shapesQueue[pastIndex].left==pastIndex)
                shapesQueue[pastIndex].left=newIndex;
            else if(shapesQueue[pastIndex].right==pastIndex)
                shapesQueue[pastIndex].right=newIndex;
        }
        shapesQueue[newIndex]=shapesQueue[pastIndex];
    }
```

### 1.3. AdditionalTool(附加工具栏)

这里因为时间关系暂时只把添加文字的功能加了进去，本来计划是要把图形的捕捉、框选以及对应的平移、旋转、放缩放置在这个附加工具栏的，以后有时间再把这部分功能需求拓展吧。绘制文本的方法采用了 Java Graphics2D 中提供的 drawString()方法。

### 1.4. 画图面板(DrawBoard)：

画板的实现主要是通过鼠标监听器再借用 AWT 重型组件的重绘来完成的画图。同样继承了 Jpanel。

这里重绘是覆盖了 paintComponent()方法

用户所绘制的所有图形都会按照顺序存放在 ShapeQueues 里

绘制的原则是如果有打开的图片 Image 有对象句柄就先在画板上绘制图片

(drawImage())，之后再根据用户实际在画板上所绘制的图形放置在图形队列 ShapeQueues 里的顺序，一个一个地绘制在画板上。

```
@Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);//重绘原来固有组件,只有这样才能替换刷新成白板,把原先画过的用白板覆盖
        Graphics2D p=(Graphics2D)g;
        if(image!=null)//如果有image就先绘制image，用于编辑图片 observer观察者是个不好的设计最好设置为null,这里也因为能够一次读取玩image
            p.drawImage(image, 0, 0, this.getWidth(), this.getHeight(), null);
        for(int i=0;i<=index;i++)//重绘之前ShapesQueue中的图形对象
            shapesQueue[i].draw(p);
    }
```

对于用户在画板上鼠标的操作主要通过设置鼠标监听器，mousePressed(MouseEvente)、mouseDragged(MouseEvente)、mouseReleased(MouseEvente)、mouseClicked(MouseEvent e)，主要的思路还是用户触发鼠标监听事件后，会获取当前鼠标的坐标点，来对它所选择图形属性赋值，然后再调用画板的重绘方法来绘制已画的一系列图形。

### 1.5. 菜单条(MenuBar)：

菜单条 MenuBar 这里由于时间关系只设置了 4 个 JMenu,分别为文件(FileMenu)、编辑(EditMenu)、图片(PictureMenu)、帮助(HelpMenu)。这里着重做了文件(FileMenu)、和图片(PictureMenu);

对于 FileMenu,这里的打开、保存、另存为对二进制 IO 文件操作，而 PictureMenu 的打开、保存、另存为是对真正的图片格式的文件操作。

### 1.6. 画笔位置坐标栏(PositionLabel)

PositionLabel 类只是简单的 Jlabel 通过画板部分的鼠标监听器显示不同鼠标操作时的坐标状态。

## 2. 功能描述(我所做的模块，熟悉的模块)：

### 2.1. 绘制图形对象

Shape 抽象类作为公共父类，定义了抽象方法 draw(),其它实体图形类都继承 Shape

```
public abstract class Shape implements Serializable{
    private static final long serialVersionUID = 1L;
    public int x1,y1,x2,y2;
    public int left=-1,right=-1;//作为curve和eraser的起始和结束标志
    public Color color;
    public float thickness;
    public boolean EOF=true;
    abstract public void draw(Graphics2D p);
}
```

其中 left、right 是作为多对象连续图形边界的一个判断，在画板鼠标监听的时候，会对用户所选择的图形进行特判，如果是曲线、橡皮擦、喷漆，就会把 left、right 赋值为当前它再图形队列中的序号 index 的值，而其他图形 left、right 都是作为-1 去处理的。

我的画笔可画曲线、直线、橡皮擦、矩形、正方形、椭圆、圆、对应我的实心图形、五角星(多边形)、喷漆、文本。基本上都是覆盖 Shape 这个抽象类重写 draw() 方法来实现。实心图形只不过对应着使用了 Java API 接口中的填充方法 fillLine() 等方法。

在这些图形的绘制中，除了曲线、橡皮擦、五角星、喷漆需要手写实现 draw()方法外，其它图形如直线、矩形、圆 Java 的 Graphics 的 API 接口中已经实现它的 draw()方法，我所做的只是需要在画板 DrawBoard 部分为它的坐标赋值，然后圆和正方形只是矩形、椭圆的变种，在获取坐标的时候取了起点和终点的最小点，对应的边长都取了最大值。

eg:

画直线

```
@Override
public void draw(Graphics2D p) {
    p.setColor(color);
    p.setStroke(new BasicStroke(thickness));
    p.drawLine(x1, y1, x2, y2);
```

```
    }
    画矩形
    @Override
    public void draw(Graphics2D p) {
        p.setColor(color);
        p.setStroke(new BasicStroke(thickness));
        p.drawRect(min(x1, x2), min(y1, y2), abs(x2-x1), abs(y2-y1));
    }

    画正方形
    @Override
    public void draw(Graphics2D p) {
        p.setColor(color);
        p.setStroke(new BasicStroke(thickness));
        p.drawRect(min(x1,x2), min(y1,y2), max(abs(x1-x2),abs(y1-y2)),
max(abs(x1-x2),abs(y1-y2)));
    }
```

这里着重介绍下画曲线、喷漆、五角星

曲线的绘制本质上还是绘制连续的直线，只不过通过设置鼠标按下移动释放的监听器，不断将画出的细微直线段开始和结束的坐标拼接起来,类似与数学里把曲线微分成细小的直线段那个原理，不断重绘画板来实现，橡皮擦和此原理相同，只不过它重绘的颜色为白色。关键代码:这是 mouseDragged()监听器中的

0 和 2 对应曲线和橡皮擦，绘制所调用的方法依然是 drawLine(),只不过会随鼠标拖动不断创建出新的直线对象。

```
if(shapeChoice==0||shapeChoice==2) {
        index++;
        if(index<=maxShapesCount) {//当现在要画的图形会覆盖队列后面的图形时,将
队列右移拓宽
            for(int i=maxShapesCount+1;i>=index+1;i--)
                queueUpdate(i-1,i);
            maxShapesCount++;
        }
        creatAction();//放在前面,这样不会多余在左上角多画出一个点

    shapesQueue[index1].x2=shapesQueue[index].x2=shapesQueue[index].x1=e.ge
tX();
    shapesQueue[index1].y2=shapesQueue[index].y2=shapesQueue[index].y1=e.ge
tY();
}
```

对于其它单对象图形只会在在拖动时创建一次图形对象，只不过拖动时再实时改

变其终点坐标，在不断重绘，这样用户看到的情形就是随着鼠标的拖动，图形的大小再不断自适应变化。
这是单对象图形鼠标拖动，监听器处理代码：

```
else if(shapeChoice!=12){//只要不是文字,其它单对象图形
            shapesQueue[index].x2=e.getX();
            shapesQueue[index].y2=e.getY();
}
if(shapeChoice!=12)//排除插入文本拖动的情形
    repaint();
```

对于喷漆而言，实际上所绘制的是某一个半径范围内随机的点，所采用得绘制 draw 方法为 drawLine()，只不过起点和终点是一样的，所以就绘制成了一个点。
这是关键的 draw() 方法：

```
@Override
    public void draw(Graphics2D p) {
        p.setColor(color);
        p.setStroke(new BasicStroke(thickness));
        for(int i=0;i<40;i++) {//每次拖动画出40个随机点分布
            if(!flag) {
                xChange[i]=rand.nextInt(20);//定义坐标半径变化范围在20以内
                yChange[i]=rand.nextInt(20);
            }
            p.drawLine(x1+xChange[i], y1+yChange[i], x1+xChange[i],
y1+yChange[i]);
        }
        flag=true;
    }
```

这是画板鼠标监听器拖动时的坐标获取

```
    else if(shapeChoice==11) {//喷漆
          index++;
          creatAction();
          shapesQueue[index].x1=e.getX();
          shapesQueue[index].y1=e.getY();
    }
```

关于五角星的绘制，这里是用了 Graphics 中 drawPolygon(int[], int[], int)，因为五角星是复杂的十边形，实现的基本思想还是根据用户在画板上点击的起点，和终点确定的十边形的边长，计算这个 10 个点的坐标。一下是我的主要思想：用到了一些三角函数、正弦、余弦、和做辅助线构造直角三角形来计算辅助边长。
下面是我做出的辅助线计算图解：

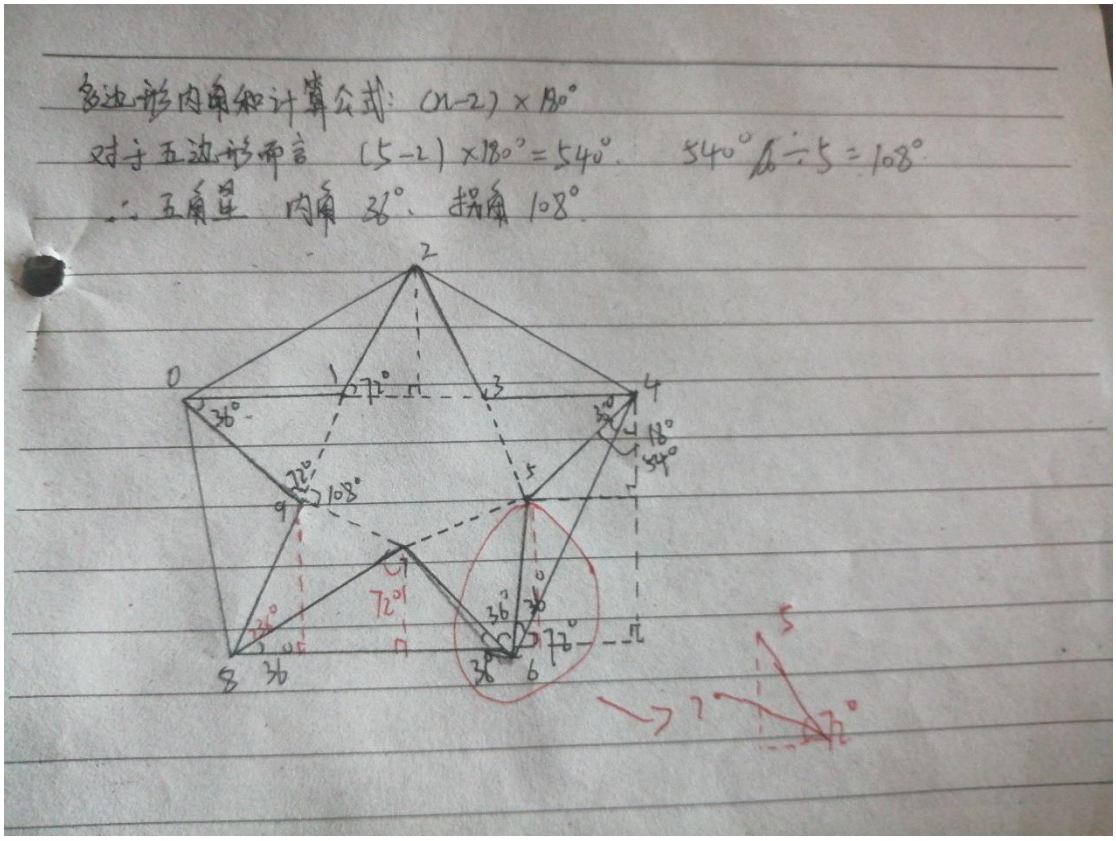


图 2.1-1 五角星图解

下面是主要代码

```
//标准五角星5个内角为36' 拐角为108'
private void initPoints() {
    edge=max(abs(x2-x1)/2, abs(y2-y1)/2);
    xPoints[0]=x1;yPoints[0]=y1;
    xPoints[1]=xPoints[0]+edge;yPoints[1]=yPoints[0];
    //72'=0.4π=1.256

xPoints[2]=(int)(xPoints[1]+edge*cos(1.256));yPoints[2]=(int)(yPoints[1]-
edge*sin(1.256));
        xPoints[3]=(int)(xPoints[2]+edge*cos(1.256));yPoints[3]=yPoints[1];
        xPoints[4]=xPoints[3]+edge;yPoints[4]=yPoints[3];
        //54'=0.3π=0.942

xPoints[5]=(int)(xPoints[4]edge*sin(0.942));yPoints[5]=(int)(yPoints[4]+edg
e*cos(0.942));

xPoints[6]=(int)(xPoints[5]+edge*cos(1.256));yPoints[6]=(int)(yPoints[5]+ed
ge*sin(1.256));

xPoints[7]=(int)(xPoints[6]edge*sin(0.942));yPoints[7]=(int)(yPoints[6]-
```

```
edge*cos(0.942));

xPoints[8]=(int)(xPoints[7]edge*sin(0.942));yPoints[8]=(int)(yPoints[7]+edg
e*cos(0.942));

xPoints[9]=(int)(xPoints[8]+edge*cos(1.256));yPoints[9]=(int)(yPoints[8]-
edge*sin(1.256));
    }

    @Override
    public void draw(Graphics2D p) {
        initPoints();
        p.setColor(color);
        p.setStroke(new BasicStroke(thickness));
        p.drawPolygon(xPoints, yPoints, nPoints);
    }
```


图 2.1-2 所画图形对象

## 2.2. 快捷工具栏

快捷工具栏有撤销、恢复、删除、复原功能
然后在撤销、恢复的时候会对当前队列中的对象做运行时对象识别，用 getClass 方法做判断。
其实主要的思路都是通过将 index 的值减小一个图形对象的数量。

这里是分为两种情形做特判，对于单对象图形如直线、矩形等，实际的 index 只减少 1。

而对于像曲线、橡皮擦、喷漆等图形，因为是连续的图形对象拼凑起来的多对象图形，这里在处理的时候，在 Shape 抽象模型中设置 left、right 属性是作为多对象连续图形边界的一个判断，在画板鼠标监听的时候，会对用户所选择的图形进行特判，如果是曲线、橡皮擦、喷漆，就会把 left、right 赋值为当前它再图形队列中的序号 index 的值，而其他图形 left、right 都是作为-1 去处理的。实际 index 在减少的时候就是连续减少了那个多对象图形在 shapeQueues 中所占满的一个区间的数量。
删除是同样的道理，只是附加了一步，就是把删除后图形所占据的位置用后面的图形对象给它覆盖掉，再把整体用户绘制的最大图形数目 maxShapesCount 减去删除的图形对象数目。

复原是将 index 恢复到了 maxShapesCount 的最大数目。
最后在最后一步去 repaint()重绘，画板上的图形数目就会变化。
核心代码：

对画图对象 Shape 的撤销、恢复、删除、复原核心代码

```
        //撤销当前队尾的图形
    private void revoke() {
        if(index>-1) {

    if(shapesQueue[index].getClass()!=Curve.class&&shapesQueue[index].getCl
ass()!=Eraser.class&&shapesQueue[index].getClass()!=Spray.class)//判断最后画
上的图形的形状
                index--;
            else {
                index--;//如果是曲线或橡皮擦或喷漆,那么末端的端点index和right一
定相等,这里可以把本身是点的跳过
                while(shapesQueue[index].left!=index)//直到遇到非曲线或橡皮擦
或者这条要撤销曲线的起点左端点结束
                    index--;
                index--;//只绘制到曲线或橡皮擦前一个图形的末端点
            }
            drawPanel.repaint();
        }
    }

    //恢复上一步的图形
    private void recover() {
        if(index<maxShapesCount) {

    if(shapesQueue[index+1].getClass()!=Curve.class&&shapesQueue[index+1].g
```

```
etClass()!=Eraser.class&&shapesQueue[index+1].getClass()!=Spray.class)//判
断后一个图形的形状
                index++;
            else {
                if(shapesQueue[index+1].left!=index+1) {//说明这条曲线可能部分
已经被别的图形替代,对于这条剪切的曲线的处理
                    int left=index+1,right;
                    while(shapesQueue[left].right!=left)
                        left++;
                    left++;//从这条不要的曲线的下个图形开始左移队列

    for(right=left,left=index+1;right<=maxShapesCount;right++,left++)
                        queueUpdate(right,left);
                    maxShapesCount=left-1;//重新设置最大图形数目
                }
                if(maxShapesCount>index) {//只有这条被替代的曲线不是最后一个图
形时才执行
                    if(shapesQueue[index+1].left==index+1) {//有可能这个不要
的曲线的下一个图形并不是曲线,而是一个对象的图形
                        while(shapesQueue[index+1].right!=index+1)//遇到非曲
线或橡皮擦时直接结束,否则到曲线或橡皮擦右端点结束
                            index++;
                    }
                    index++;//将曲线末端或后一个图形绘制出来
                }
            }
            drawPanel.repaint();
        }
    }

    //将当前队尾的图形从队列中删除
    private void delete() {
        if(index>-1) {

    if(shapesQueue[index].getClass()!=Curve.class&&shapesQueue[index].getCl
ass()!=Eraser.class&&shapesQueue[index].getClass()!=Spray.class) {
                for(int i=index;i<=maxShapesCount-1;i++)//左移队列
                    queueUpdate(i+1,i);
                maxShapesCount--;
            }
            else {
                int left,right=index+1;
                while(shapesQueue[index].left!=index)
                    index--;
```

```
                for(left=index;right<=maxShapesCount;right++,left++)
                    queueUpdate(right,left);
                maxShapesCount=left-1;
            }
            index--;//只绘制到要删除图形的前一个图形的末端
            drawPanel.repaint();
        }
    }

    //将图形复原为最大图形数
    private void restore() {
        recover();
        index=maxShapesCount;
    }
```

## 2.3. 文件菜单栏

新建实际是把指向 ShapeQueues 图形对象的序列 index 置为-1 再调用重绘方法，此时画板被清空后，由于 index 为-1 所以没有对象被绘制到画板上，画板即被清空。

另存为操作演示：

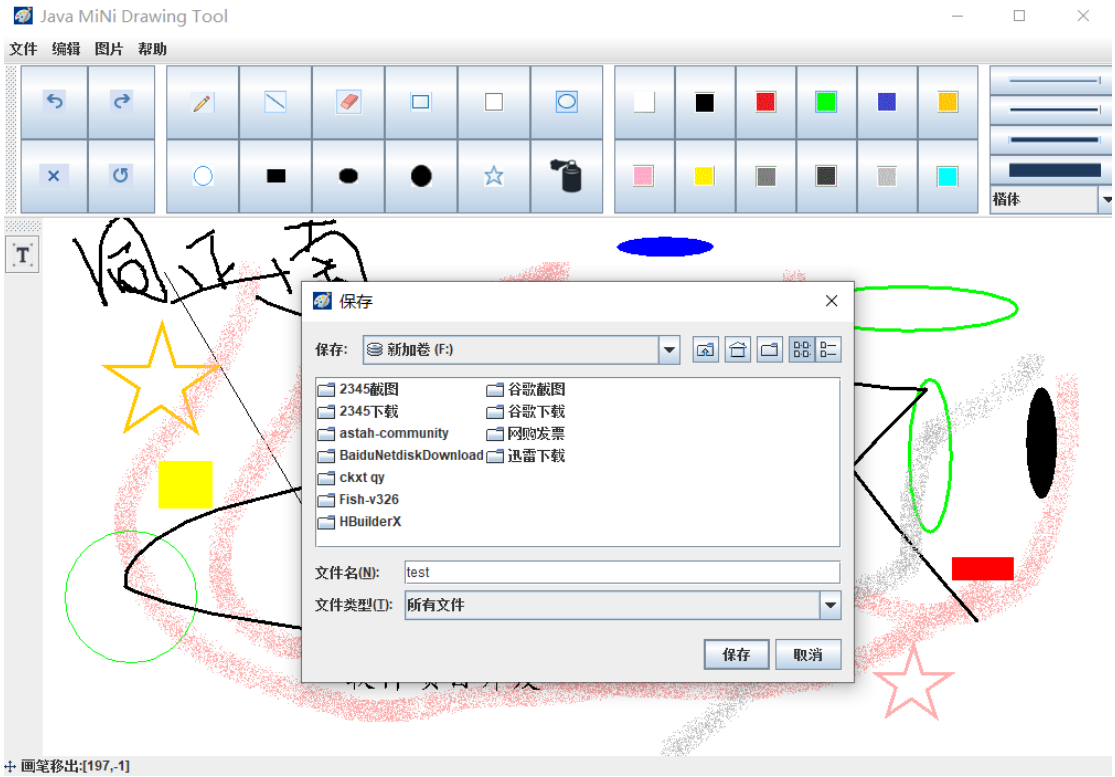


图 2.3-1 保存窗口文件选择器

上图是保存在 F 盘并写命名为 test 的一个二进制 IO 文件

这是保存之后本地文件的情况：

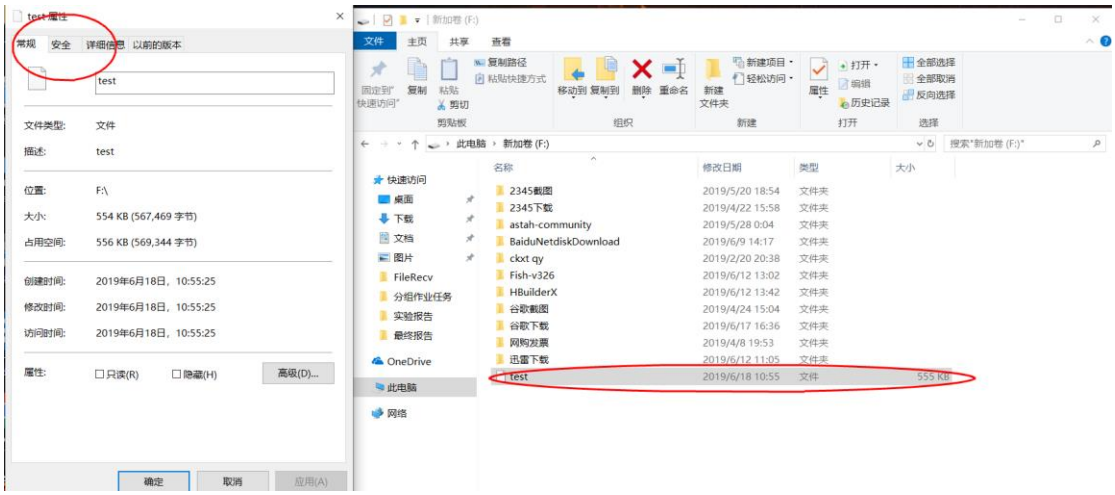

图 2.3-2 保存到本地的 IO 二进制文件详情

对于另存为操作实际上是以对象输出流的形式将画板上的图形对象以 IO 对象流的形式保存在本地，每次读取到最后一个对象时会将 Shape 中的 EOF 标志修改为 false，再下次打开读取这个 IO 文件时就会去判断这个 EOF 标志作为读取结束的标志，避免出现 EOFException 异常。对于保存操作，当用户在原有打开的 IO 文件上再绘制新的图形时，就会把之前 EOF 标志修改为 ture，再把之后的最后一个图形对象的 EOF 标志修改为 false。

保存、另存为到本地的操作的核心代码：

```
ObjectOutputStream ShapeObject=new ObjectOutputStream(new FileOutputStream(src));
        for(int i=0;i<=index;i++) {
            if(i==index)
                shapesQueue[i].EOF=false;
            ShapeObject.writeObject(shapesQueue[i]);
        }
        lastEOF=index;//实现同一文件多次修改,不断将结束符推后
        isChanged=false;
        if(!flag)//另存文件完成时,自动将OpenFile的文件句柄改为src(另存之后的文件)
            openFile=src;
```

打开操作的核心代码：

```
ObjectInputStream ShapeObject=new ObjectInputStream(new FileInputStream(openFile));
        for(index=0;true;index++) {
            shapesQueue[index]=(Shape)ShapeObject.readObject();//子
```

```
类引用无法接受父类对象Object
                    if(!shapesQueue[index].EOF) {
                        maxShapesCount=lastEOF=index;//打开画图图片对象数的上
界设立在此
                        break;
                    }
                }
                drawPanel.repaint();//将打开图像呈现在画板上
```

新建的核心代码:

```
    public void newFile() {
        openFile=null;//新文件还没有打开的图片文件设置为null,方便实现用户错把保
存当另存为
        index=maxShapesCount=lastEOF=-1;
        isChanged=false;
        drawPanel.repaint();
    }
```

## 2.4. 编辑菜单栏

编辑选项是画笔颜色选择器这个是直接调用的 JcolorChooser.showDialog, 还有画笔粗细选择这个是调用的 Joption.showInputDialog 来实现的。本质还是修改 Shape 对象当前的 color 和 stroke

核心代码:

```
    @Override
    public void actionPerformed(ActionEvent e) {
        if(e.getSource()==editItem[0]) {
            color=JColorChooser.showDialog(null,"选择画笔颜色",color);
        }
        else if(e.getSource()==editItem[1]) {
            String input=JOptionPane.showInputDialog("请输入画笔粗细");
            if(input!=null)//防止用户点击取消,造成的空指针异常
                stroke=Float.parseFloat(input);
        }
}
```

## 2.5. 图片菜单栏

这个菜单栏下的新建、保存、另存为 操作的是真正的图片格式的文件。这里设置三种图片格式.png .jpg .gif 图片供用户操作。

另存为图片演示:这里是以另存为一个.png 格式的图片为例,注意这里在弹出的文件选择器需要用户自己输入.png 的图片后缀,

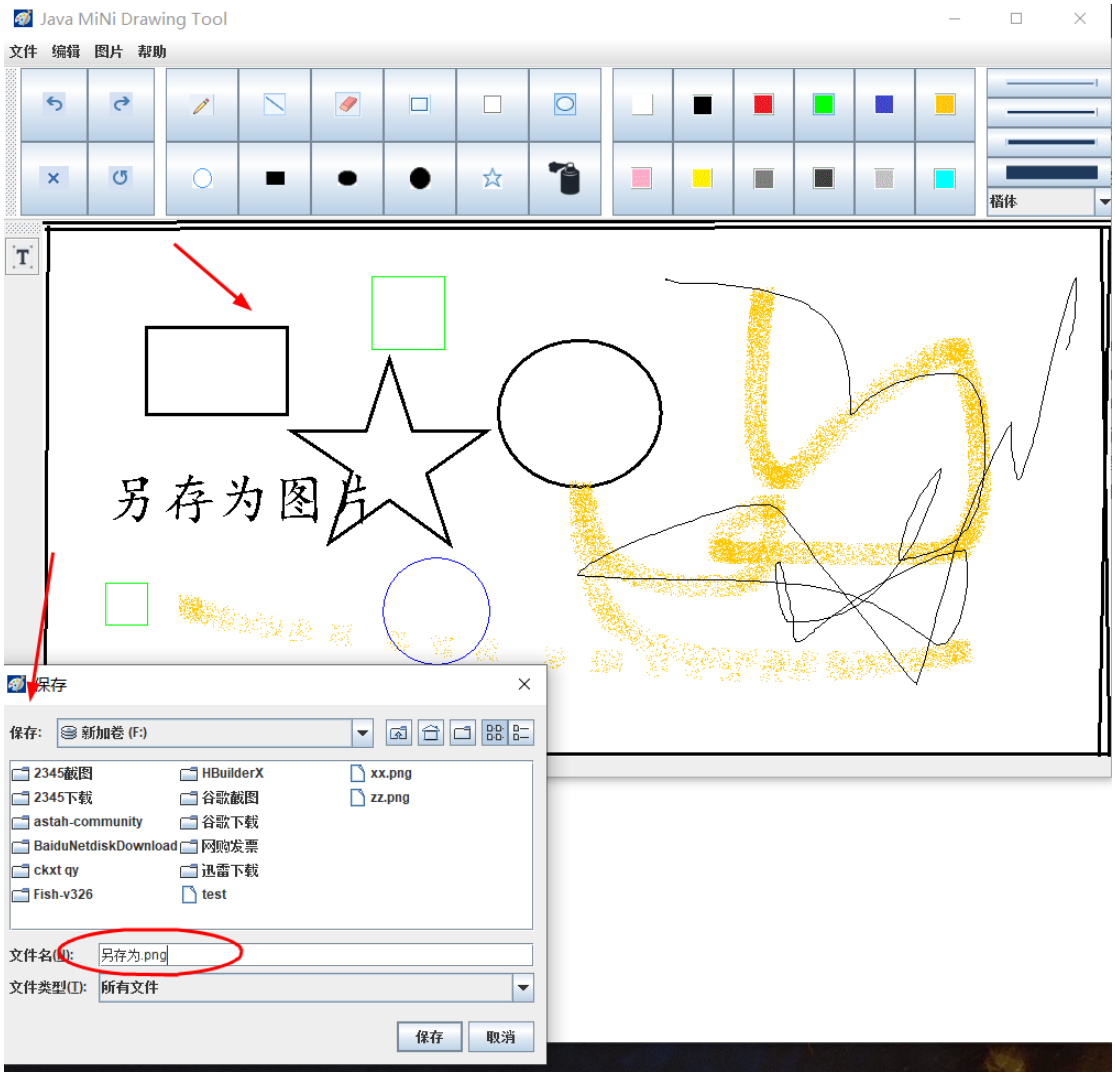


图 2.5-1 另存为.png 格式的图片

这是保存到本地后用图片查看器 2345 看图王打开可以预览的情形:


图 2.5-2 .png 格式的图片被成功保存到本地，并用看图软件能打开预览

对于保存、另存为的实现方式 将画板上的图画 用 Rectangle() 类和 Robot 类对画板做了区域化的截屏将其保存为 BufferedImage 对象。最后再通过 Java 中的 ImageIO 的 write() 方法将 BufferedImage 以指定的图片格式写入本地。

保存、另存为关键代码:

```
//这里Rectangle的纵坐标.exe要比原生项目多10px
image=new Robot().createScreenCapture(new Rectangle(mainWindow.getX()+44,
mainWindow.getY()+193, drawPanel.getWidth(), drawPanel.getHeight()));
ImageIO.write(image, format, src);
```

打开.png 图片演示:左图为 2345 看图王软件打开的格式，右图为我画图工具打开预览的图片。

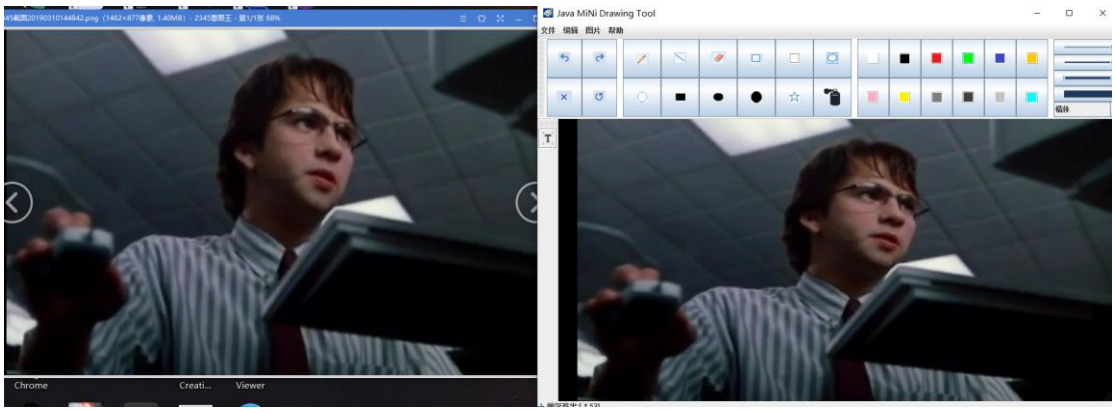


图 2.5-3 打开.png 格式图片预览模式

对于打开还是利用 ImageIO 类 read()读取本地的指定格式的图片文件，最后在画板不符将读出的 BufferedImage 以 drawImage()方法 重新绘制了。
打开关键代码:

```
drawPanel.image=ImageIO.read(openPicture);
fileOption.newFile();
```

```
    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);//重绘原来固有组件,只有这样才能替换刷新成白板,把原先画过的用白板覆盖
        Graphics2D p=(Graphics2D)g;
        if(image!=null)//如果有image就先绘制image，用于编辑图片 observer观察者是个不好的设计最好设置为null,这里也因为能够一次读取玩image
            p.drawImage(image, 0, 0, this.getWidth(), this.getHeight(), null);
        for(int i=0;i<=index;i++)//重绘之前ShapesQueue中的图形对象
            shapesQueue[i].draw(p);
    }
```

## 2.6.帮助菜单栏

帮助选项中有画图工具的一些开发信息,和关于开发团队的一些信息。

就是通过监听器监听菜单条款，再去 IO 读取本地的文本信息加载到 JOptionPane.showMessageDialog 里事
这是关键代码:

```
    private void readMessage() {
        int len;
        byte []tmp=new byte[350];
        for(int i=0;i<message.length;i++) {
            try(BufferedInputStream input=new BufferedInputStream(new FileInputStream("HelpInformation/"+name[i]+".txt"))){
                len=input.read(tmp);
                message[i]=new String(tmp,0,len);
            }catch(FileNotFoundException e) {
                JOptionPane.showMessageDialog(this, "不存在该文件", "打开文件错误提示", JOptionPane.ERROR_MESSAGE);
            }catch(IOException e) {
                e.printStackTrace();
            }
        }
```

}

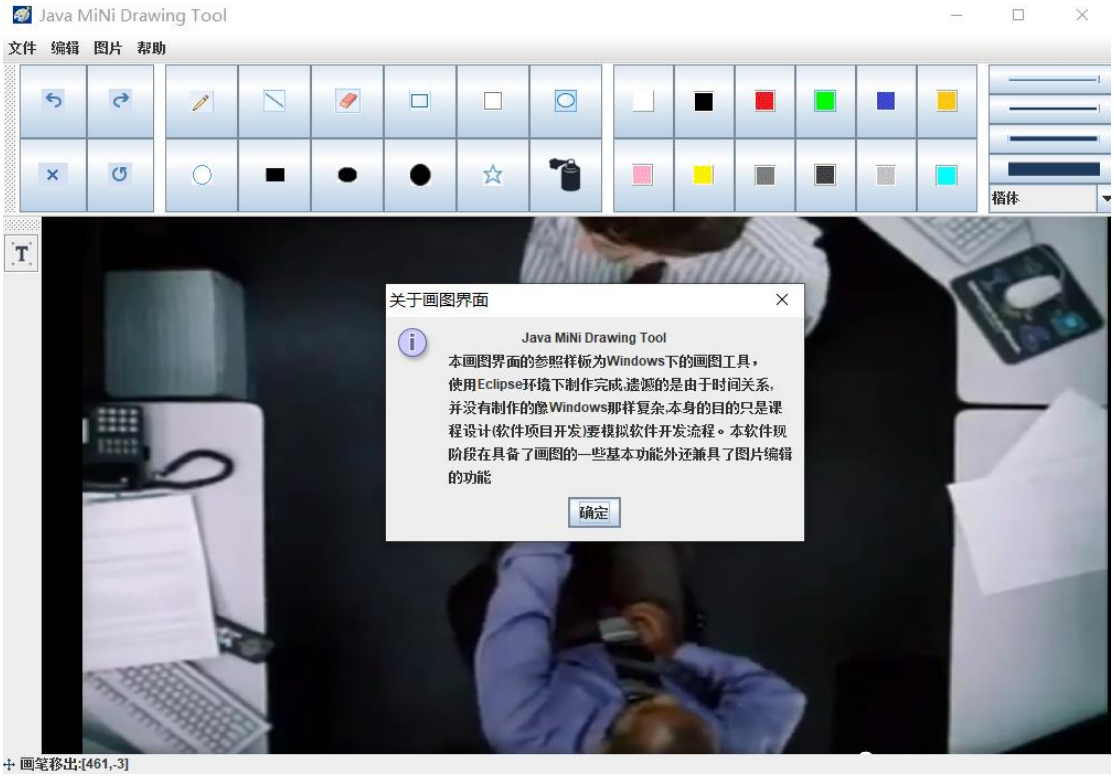


图 2.6-1 关于画图界面系统的制作

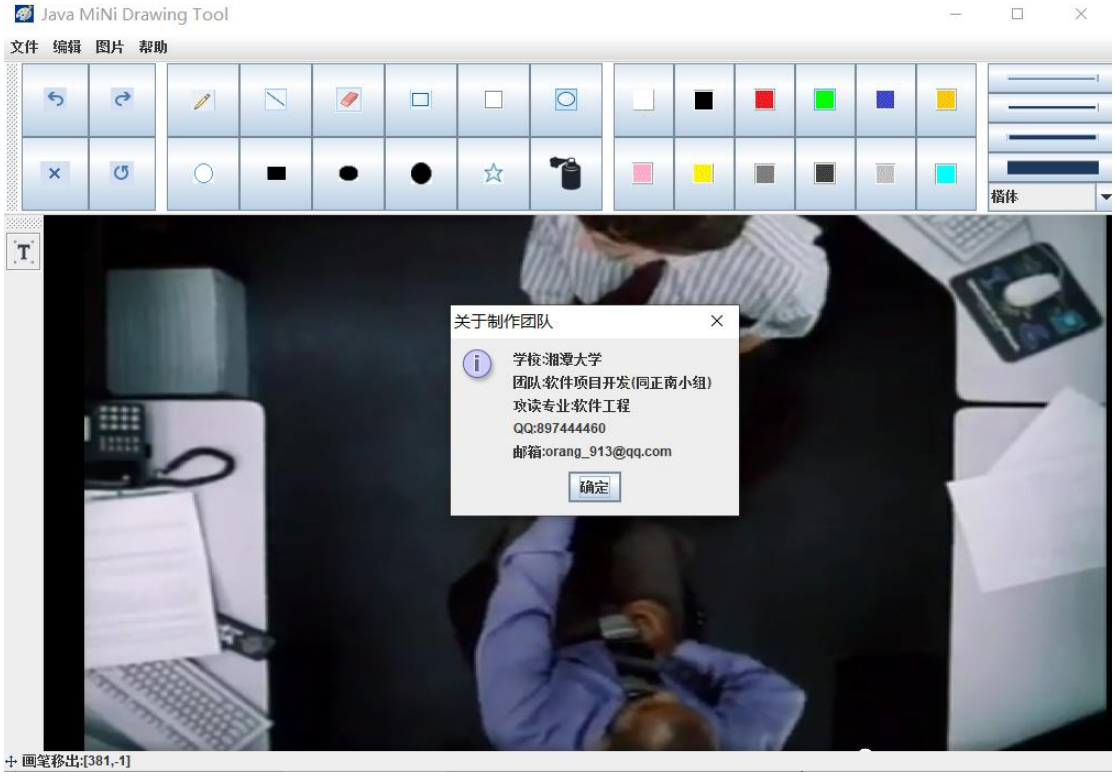


图 2.6-2 关于画图界面系统作者信息

## 2.7. 画笔位置坐标栏

最后一个是画笔状态,画笔进入画板区会在主界面左下角显示器坐标状态。

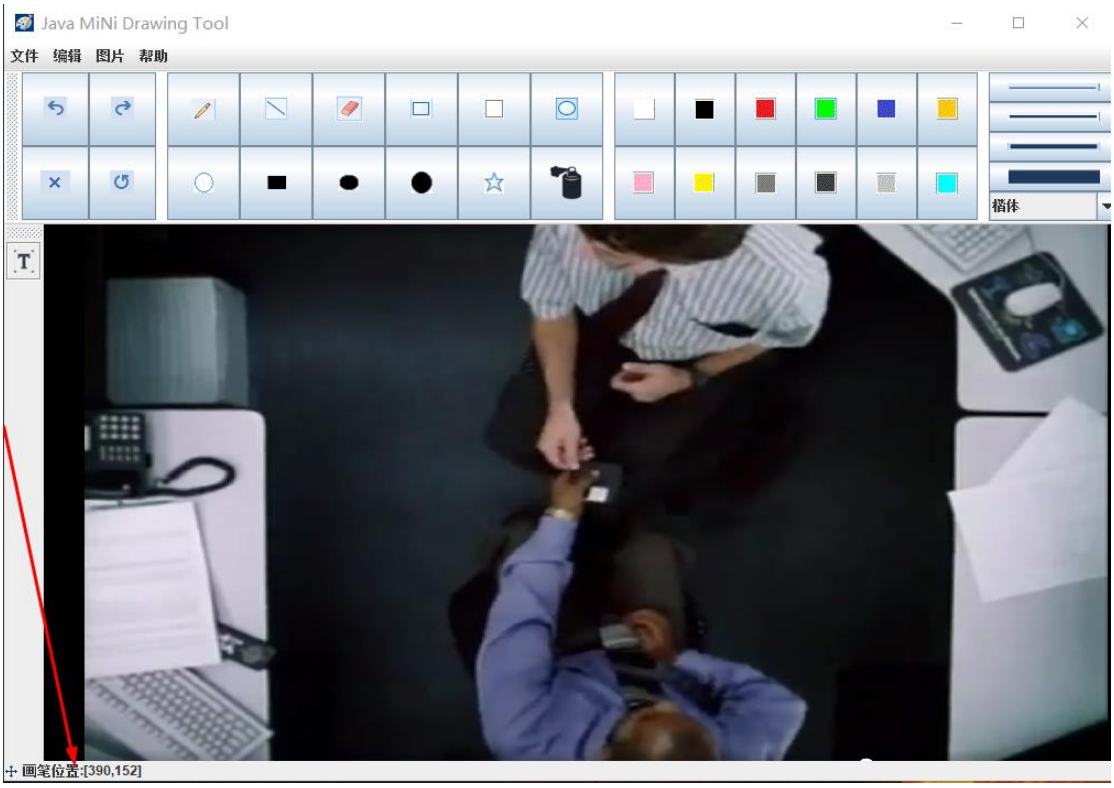


图 2.7-1 画笔位置坐标

代码设置在画板的鼠标进入、退出监听器里

```
@Override
public void mouseEntered(MouseEvent e) {
    mousePostion.setText("画笔进入:"+"["+e.getX()+","+e.getY()+"]");
}

@Override
public void mouseExited(MouseEvent e) {
    mousePostion.setText("画笔移出:"+"["+e.getX()+","+e.getY()+"]");
}
```