Base & Bonus Project

*注意：因为两个 **Project** 有很大的关联性，所以在一个程序中实现，其中 **Base Project** 是 **Bonus Project** 的一个子步骤。*

一. Base Project 多边形区域填充

1. 项目需求

本项目的要求为可输入一个多边形区域，程序自动对此区域进行填充实现该项目的时候只能用**基本的画点算法**，不准用程序语言中**自带的画线和填充算法**。

2. 项目分析

2.1 步骤说明

为了实现多边形区域填充，要分如下几个步骤：

1) 获取输入的顶点
2) 根据顶点画出多边形的边（这里要用到直线扫描转换的算法）
3) 根据多边形的边进行区域填充（这里要用到区域填充算法）

2.2 算法说明

2.2.1 直线扫描转换算法

采用最朴素也最简单高效的数值微分法 DDA(Digital Differential Analyzer), 设过端点 P0(X0,Y0)、P1(X1,Y1)的直线段为 L(P0,P1)。

斜率 k=(Y1-Y0)/(X1-X0)

L 的起点 P0 的横坐标 X0 向 L 的终点 P1 的横坐标 X1 步进，取步长=1 个像素，用 L 的直线方程 y=kx+b 计算相应的 y 坐标，并取像素点(X,round(Y))作为当前点的坐标。

要注意的是，上述的算法讨论的 k 在[0,1]之间的情况，下面讨论其他几种情况算法应该如何处理。

a. k 在[-1,0]之间，此时只要把 P0 和 P1 交换一下位置就可以了
b. k 在(1,正无穷)之间，此时要根据 Y 坐标计算，即 P0 的纵坐标 Y0 向 L 的终点 P1 的纵坐标 Y1 步进
c. k 在(负无穷,-1)之间，此时也要根据 Y 坐标计算，只不过跟情况 b 相反，即 P1 的纵坐标 Y1 向 L 的终点 P0 的纵坐标 Y0 步进

2.2.2 多边形域填充算法

本算法采用的思想是书本上介绍的边填充算法的思想，**但又不完全一样，有特别的地方。**现将该算法描述如下：

设多边形顶点依次为 S0,S1,…,Sn. 依次扫描多边形的每条边 S0S1,S1S2,S2S3,…… 对于每次扫描的边，设边顶点为 AB，**把 S0 A B 这三个点构成的三角形内部的像素值取反**。扫描完成后，区域填充完毕。边的顺序可随意处理。

注意，每次把三角形内部的像素值取反，这个操作用到的是种子填充(Floodfill)算法，具体过程相对简单，这里不再赘述。

3. 项目实现

项目采用基于 JAVA 平台的图像化界面实现，由于整个项目是流程化的，所以采用过程化实现，核心代码如下：

1) 直线扫描转换

```
void drawLines(int x1,int y1,int x,int y,int[][] color,int canshu)
    {
    //画线   数值微分法
        float dx,dy,ee;
        float k,yy,xx;
        dx=x-x1;
        dy=y-y1;
        yy=y1;
        xx=x1;

        if (Math.abs(dx)>Math.abs(dy))
        {
            k=dy/dx;
            if (dx>=0) ee=1;else ee=-1;
            for (int i=0;i<=Math.abs(dx)+0.5;i++)
            {
                color[(int)(xx+0.5)][(int)(yy+0.5)]=canshu;
                yy=yy+k*ee;
                xx=xx+ee;
            }
        }
        if (Math.abs(dx)<Math.abs(dy))
        {
            k=dx/dy;
            if (dy>=0) ee=1;else ee=-1;
            for (int i=0;i<=Math.abs(dy)+0.5;i++)
            {
                color[(int)(xx+0.5)][(int)(yy+0.5)]=canshu;
                xx=xx+k*ee;
```

```
                    yy=yy+ee;
                }
            }
}
```

2) 多边形区域填充

```
    //画直线、进行实时填充

    public void myRepaint(){
        drawLines(x1,y1,x,y,color,10);
        for (int i=0;i<300;i++)
         for (int j=0;j<300;j++)
              mark[i][j]=0;
        drawLines(x1,y1,x,y,mark,10);
        drawLines(x1,y1,x0,y0,mark,20);
        drawLines(x,y,x0,y0,mark,10);

        floodFill();

        repaint();

    }
```

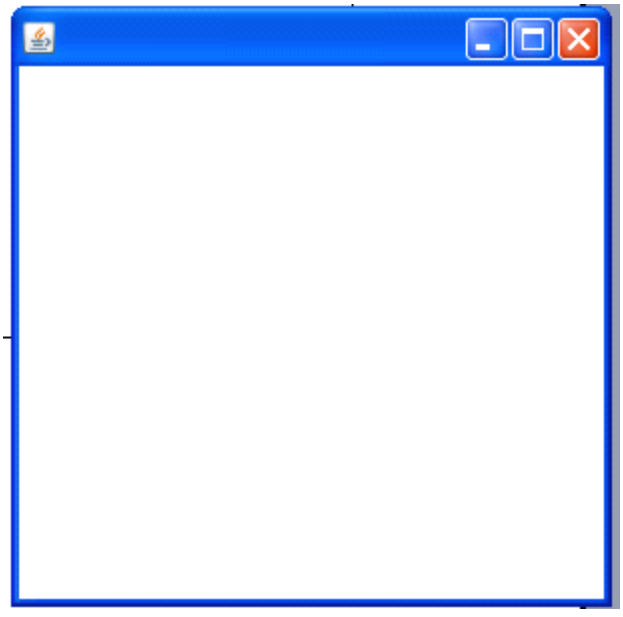
3) 种子填充

```
void floodFill()
    {
        int x=0,y=0;
        int head=0;
        int tail=0;
        int queuex[]=new int[100000];
        int queuey[]=new int[100000];
        while (head<=tail)
        {
        if (x<300) if (mark[x+1][y]==0)
            {
            tail++;
            queuex[tail]=x+1;
            queuey[tail]=y;
            mark[x+1][y]=1;
        }
        if (x>0) if (mark[x-1][y]==0)
        {
            tail++;
            queuex[tail]=x-1;
```

```
            queuey[tail]=y;
            mark[x-1][y]=1;
        }

        if (y<300) if (mark[x][y+1]==0)
        {
            tail++;
            queuex[tail]=x;
            queuey[tail]=y+1;
            mark[x][y+1]=1;
        }

        if (y>0) if (mark[x][y-1]==0)
        {
            tail++;
            queuex[tail]=x;
            queuey[tail]=y-1;
            mark[x][y-1]=1;
        }
        head++;
        x=queuex[head];
        y=queuey[head];
        }
    }
```

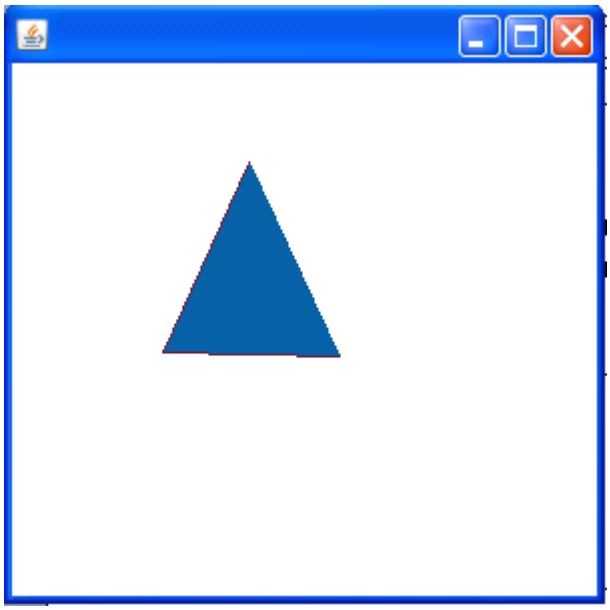
4. 项目说明

打开项目，出现如下空白画面：

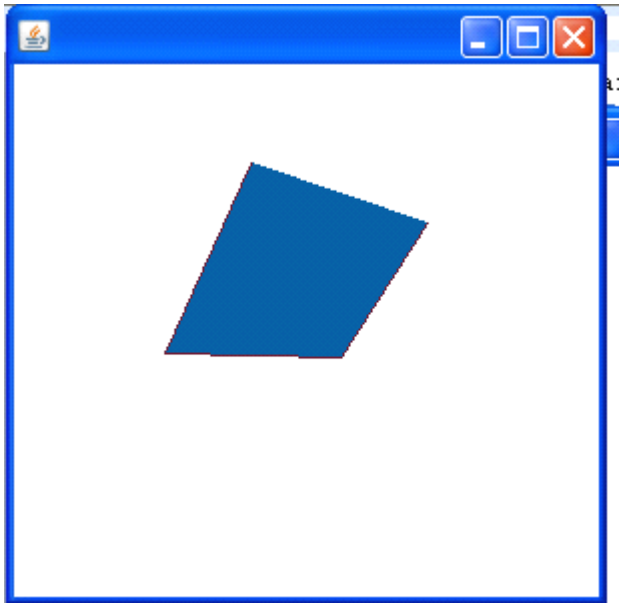


按照多边行顶点顺序，依次用鼠标点击各个顶点，每点击一个新点就会出现一个填充的多边形，(这里默认用鼠标点击的最后一个顶点和鼠标点击的第一个顶点围成多边形)。
如下面的图示：

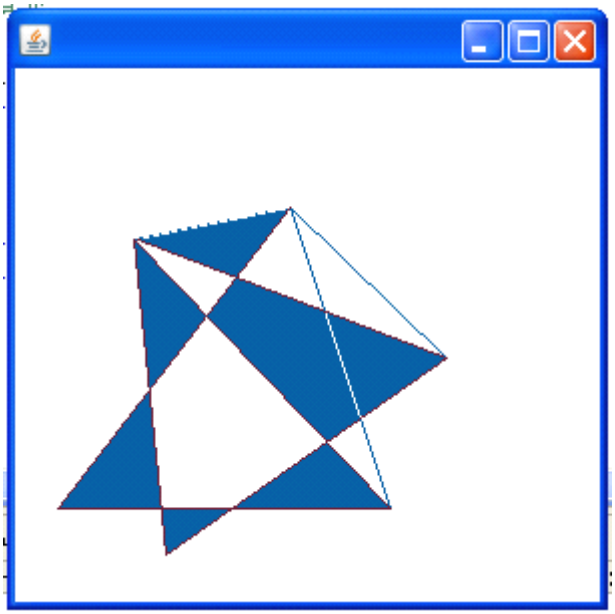
点击三个点时



再增加一个点于右上方



本算法可支持多边形自交，如下图



二．Bonus Project 多边形区域填充

1. 项目需求

任意两个多边形，求出其相交区域。

2. 项目分析

本项目和第一个项目 Base Project 的关系密不可分。
先输入两个多边形，从第一个项目入手，对两个多边形进行区域填充，**对于一个像素，如果被这两个多边形都填充过，那么这个像素就是相交区域像素，用另外一种颜色标出。**
以上便为该项目的核心思想，非常简单可行。

3. 项目实现

直接在第一个项目的基础上实现，输入一个多边形后，**点击鼠标右键**，可输入另一个多边形。
求交的核心代码如下：

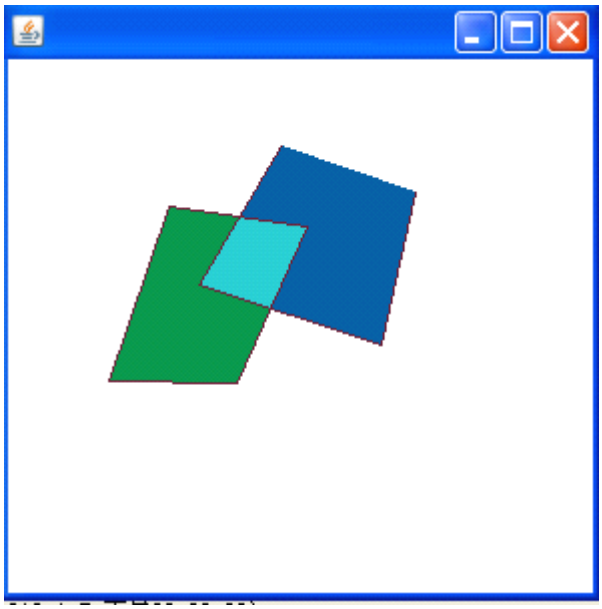
```
public void paint(Graphics g)
    {
```

```
            if (flag)
            {
                flag = false;
                return;
            }
        //  g.drawLine(x,y,x1,y1);
/*          BufferedImage image = new BufferedImage(this.getWidth(), this.getHeight(),
BufferedImage.TYPE_3BYTE_BGR);
            image.setRGB(x, y, 112123);

            g.drawImage(image, 0, 0, this);
      */

            BufferedImage image = new BufferedImage(300,300, BufferedImage.TYPE_3BYTE_BGR);
            for (int i=0;i<300;i++)
                for (int j=0;j<300;j++)
                {
                    image.setRGB(i,j,0XFFFFFF);
                }

            for (int i=0;i<300;i++)
                for (int j=0;j<300;j++)
                {
                    if ((mark[i][j]==0 || mark[i][j]==10)   && color[i][j]<10) color[i][j]=color[i][j]^1;
                    if (color[i][j]==1 && oldcolor[i][j]==0)
                        {
                        image.setRGB(i, j,23456 );
                        //g.drawLine(i,j,i,j);


                        }
                    if   (color[i][j]==0 && oldcolor[i][j]==1) image.setRGB(i, j,234567 );
                    if (color[i][j]==1 && oldcolor[i][j]==1) image.setRGB(i, j, 2345678);
                    if (color[i][j]==10 || oldcolor[i][j]==10) image.setRGB(i, j, 6887987);
                }
            g.drawImage(image, 0, 0, this);
           for (int i=0;i<300;i++)
            for (int j=0;j<300;j++)
                mark[i][j]=0;

      }
```

4. 项目说明

先输入一个多边形，点击鼠标右键可输入另外一个，再次点击右键可把第一个输入的多边形删除，保留第二个，继续求交。效果图如下：

效果图 1



效果图 2