MiniCAD 实验报告

作者：鲍奕帆

学号：3180103499

日期：2020年11月16日

系统：

JRE:

```
evan@EvandeMacBook-Pro myCAD % java --version
java 14.0.2 2020-07-14
Java(TM) SE Runtime Environment (build 14.0.2+12-46)
Java HotSpot(TM) 64-Bit Server VM (build 14.0.2+12-46, mixed mode, sharing)
```

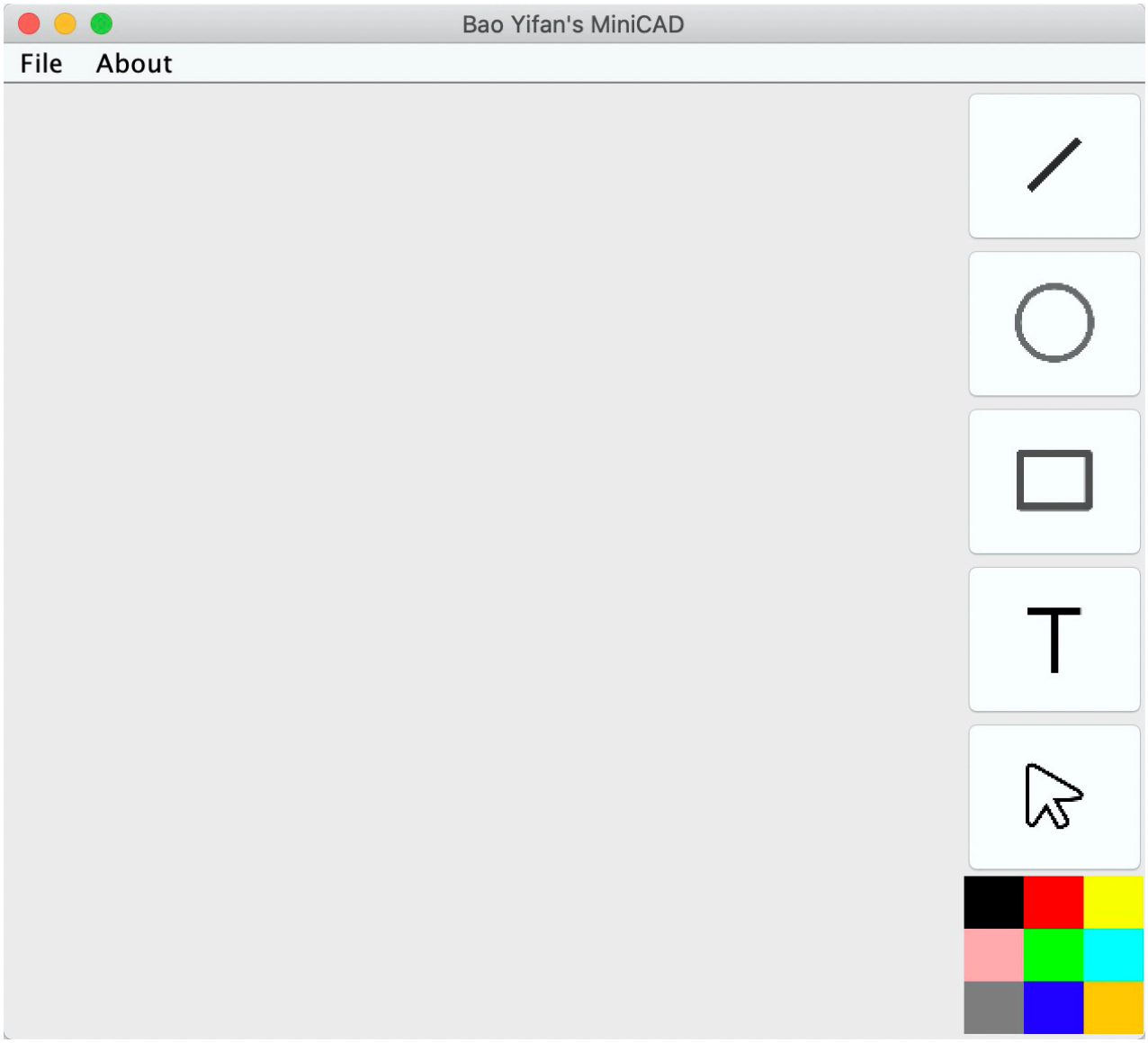
有些内容比如，按钮颜色的显示，Mac与Windows上有些不同，因此不清楚Windows上跑的结果如何。

项目要求

做一个简单的绘图工具，以CAD的方式操作，能**放置直线、矩形、圆和文字**，能**选中**图形，**修改**参数，如**颜色**等，能**拖动图形**和**调整大小**，可以保存和恢复。

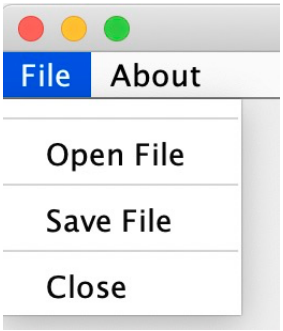
功能介绍

初始界面



主要有4个部分。最上方为菜单栏，中间是画板，右边是一个图像按钮panel，以及调色板panel。

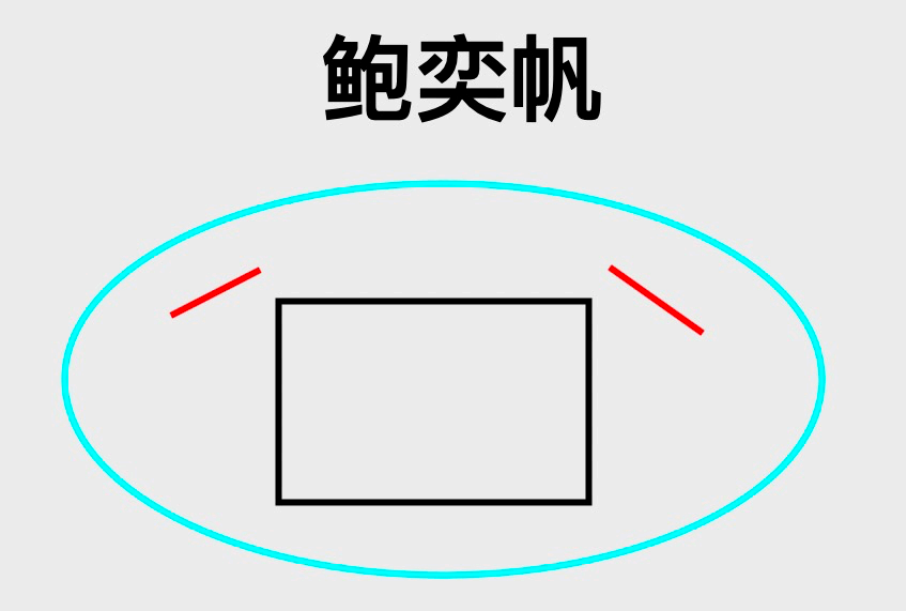
1. 菜单栏



File菜单栏可以用于加载文件和保存文件，以及退出。

About菜单栏用于显示相关的文件信息，包括图形大小和粗细的快捷键，以及程序相关信息。

2. 图像按钮面板。一共5个按钮，分别为直线、椭圆、矩形、文本以及光标选择，各个功能顾名思义。
3. 调色板。根据指示的颜色修改选中的图形信息，同时保持该颜色为以后使用。
4. 画板。



具体使用方式

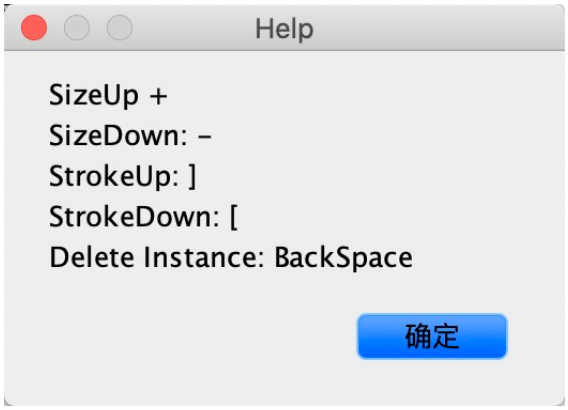
1. 图形绘制

初始状态为**选择**状态。通过点击右侧的图标按钮，选择需要绘制的图形。

而后再画板上**拖拽**即可。对于文本的显示，需要点击面板，而后在对话框上输入

2. 大小和粗细修改以及删除实例

   按住"+"号能够增大图形;按住"-"号，能够减小图形。按住"]"能够加粗，按住"["能够变细。按住回车键即可删除对应的实例



3. 图形选择

点击光标，选择对应图形即可。图形的选择是实心的，也就是内部可以选择。

4. 图形位置修改

在选择图形后，拖拽图形即可更改图形位置。

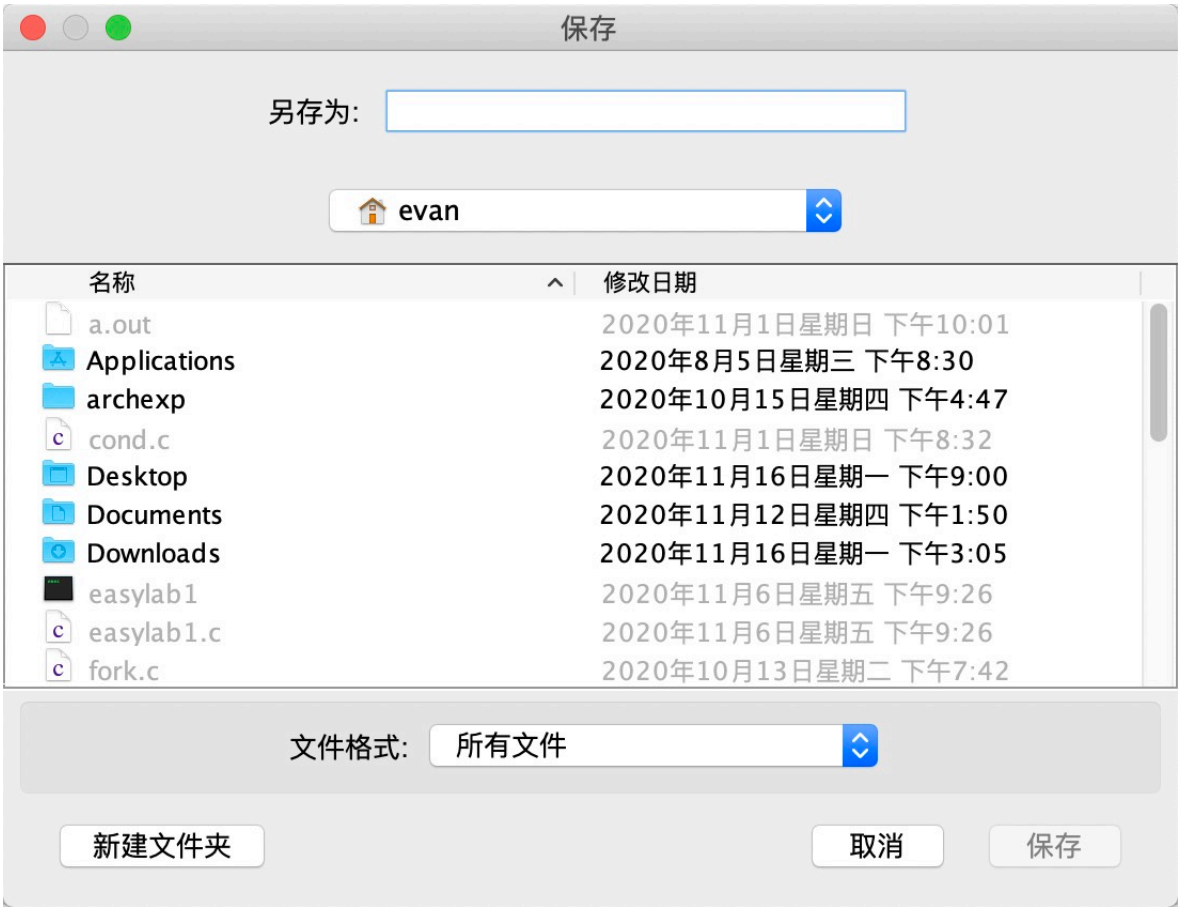
5. 图形颜色修改

在选择图形后，点击对应调色板的颜色即可修改颜色。注意，之后的颜色都为选定的颜色。

6. 文件读入与保存

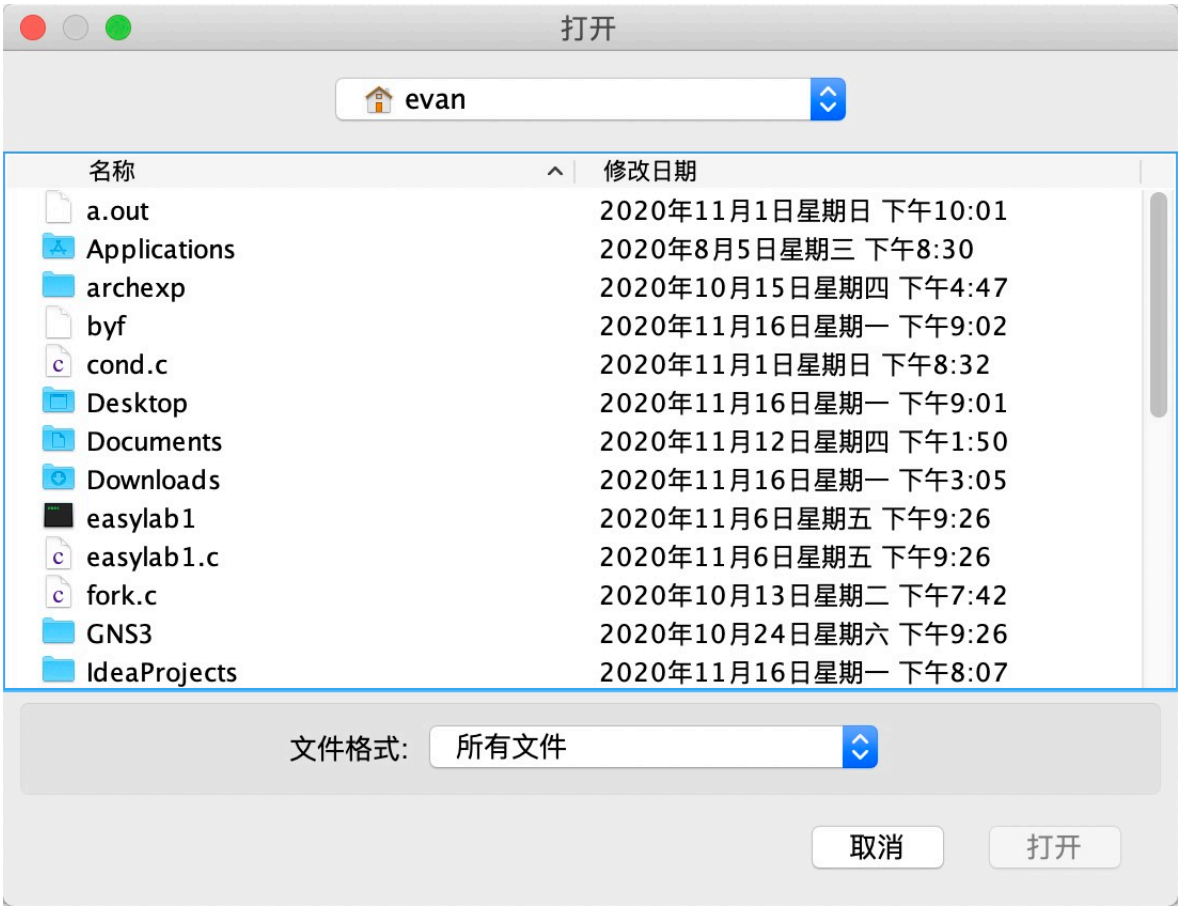
保存通过另存为的形式实现。

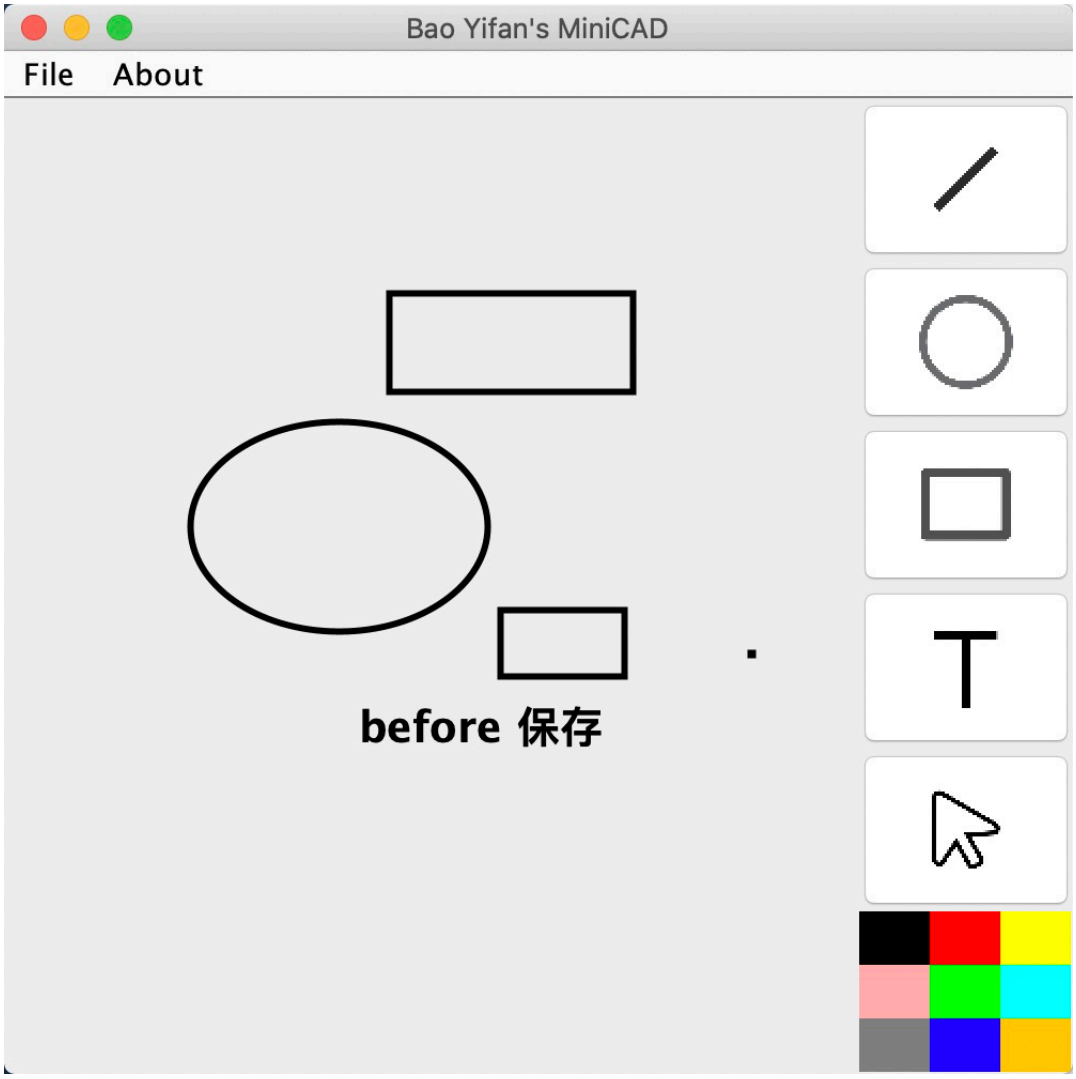
保存按钮在菜单栏的文件处



名字可以任意命名。

打开文件，只能打开之前自己保存过的内容。

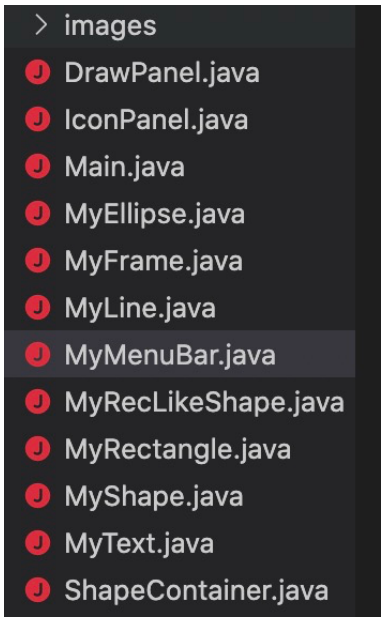

打开
evan
名称
修改日期
a.out
2020年11月1日星期日 下午10:01
Applications
2020年8月5日星期三 下午8:30
archexp
2020年10月15日星期四 下午4:47
byf
2020年11月16日星期一 下午9:02
cond.c
2020年11月1日星期日 下午8:32
Desktop
2020年11月16日星期一 下午9:01
Documents
2020年11月12日星期四 下午1:50
Downloads
2020年11月16日星期一 下午3:05
easylab1
2020年11月6日星期五 下午9:26
easylab1.c
2020年11月6日星期五 下午9:26
fork.c
2020年10月13日星期二 下午7:42
GNS3
2020年10月24日星期六 下午9:26
IdeaProjects
2020年11月16日星期一 下午8:07
文件格式:
所有文件
取消
打开


Bao Yifan's MiniCAD
File
About
before 保存

对于不是之前保存的内容，不会产生什么效果。但终端可能会有些异常输出。

最后是文件关闭的时候的保存信息提示。

代码实现



代码结构较为简单。由于自己也不是特别熟悉是使用IDE，因此仍然是直接使用的VSCode编写。

关键主要分为几个

1. 形状相关

根据题目要求，形状相关的类主要有MyShape类、MyRecLikeShape类、MyEllipse类、MyLine类、MyRectangle类，以及MyText类

其中MyRecLikeShape类、MyLine类和MyText类继承自MyShape类，MyEllipse类与MyRectangle类继承自MyRecLikeShape类。

与此同时，**MyEllipse类、MyRectangle类、MyLine类中封装有java.awt.geom中的集合形状实例，用以使用他们提供的区域包含功能**。和MyText也封装了java.awt.geom.Rectangle2D。

抽象父类MyShape代码如下

```
enum ShapeCate {
    UNDEFINED, LINE, ELLIPSE, RECT, TEXT
```

```
}
public abstract class MyShape implements Serializable {
    private static final long serialVersionUID = 1L;
    protected static final double INCREASE = 1.05;
    protected static final double DECREASE = 0.95;
    private static final double STROKEMAX = 20;
    protected ShapeCate theShape = ShapeCate.UNDEFINED;
    private Color color = Color.BLACK;
    private int stroke = 2;
    public ShapeCate getShapeCato() {
        return theShape;
    }
    public Color getColor() {
        return color;
    }
    public void setColor(Color color) {
        this.color = color;
    }
    public int getStroke() {
        return stroke;
    }
    public boolean strokeUp() {
        if (stroke < STROKEMAX * 2) {
            stroke++;
            return true;
        }
        return false;
    }
    public boolean strokeDown() {
        if(stroke > 1) {
            stroke--;
            return true;
        }
        return false;
    }
    public void Draw(Graphics2D g) {
        g.setColor(color);
        g.setStroke(new BasicStroke(stroke));
        drawShape(g);

    }
    abstract public void drawShape(Graphics2D g);
    abstract public void move(int dx, int dy);
    abstract public boolean sizeUp();
    abstract public boolean sizeDown();
    abstract public void shapeDragged(int x1, int y1, int x2, int y2);
    public MyShape(ShapeCate category, Color color) {
        this.theShape = category;
        this.color = color;
```

```
    }
    public MyShape(ShapeCate category, Color color, int stroke) {
        this.theShape = category;
        this.color = color;
        this.stroke = stroke;
    }
    abstract boolean contains(int x, int y);
}
```

MyShape中已经实现的都是一些基本的gets/sets，而抽象方法是实现miniCAD画图的最重要的部分。

drawShape函数根据对应形状进行画图。move函数移动图形。sizeUp与sizeDown修改图形大小。shapeDragged设置一开始图形拖动绘图。contains函数用于判断点是否包含在图形中。

MyShape子类MyRecLikeShape是椭圆和矩形的共有子类，封装了基本相同的接口。同时实现了MyShape除了contains外的所有函数。

MyRectangle类继承了MyRecLikeShape，其封装了java.awt.geom.Rectangle2D，实现了contains方法

```
public class MyRectangle extends MyRecLikeShape   {
    private static final long serialVersionUID = 1L;
    private Rectangle2D innerRectangle;
    public MyRectangle(int x, int y, Color color) {
        super(x, y, color, ShapeCate.RECT);
        innerRectangle = new Rectangle2D.Double(x, y, 1, 1);
    }
    public MyRectangle(int x, int y, int width, int height, Color color, int
stroke) {
        super(x, y, width, height, color, stroke, ShapeCate.RECT);
        innerRectangle = new Rectangle2D.Double(x, y, width, height);
    }
    public boolean contains(int x, int y) {
        innerRectangle.setFrame(this.x, this.y, this.width, this.height);
        return innerRectangle.contains(x, y);
    }
}
```

其contains方法内部调用Rectangle2D实例的contains方法。

MyEllipse类的实现与MyRectangle类似。

MyLine类继承了MyShape类，实现了其基本的方法。但是他的contains函数始终返回false，这是因为直线理论上不占面积的。这里提供了一个额外的nearLine()方法来判断直线附近的点。同样是通过封装java.awt.geom.Line2D类实现的。

```
public boolean nearLine(int x, int y) {
        innerLine.setLine(x1, y1, x2, y2);
        return innerLine.ptLineDist(x ,y) < 5;
       // 距离足够小就返回
    }
    public boolean contains(int x, int y){
        return false;//no use now
    }
```

MyText类也继承了MyShape类，内部封装了java.awt.geom.Rectangle2D，用矩形了框定文本的范围。而后使用其contains方法。注意，这里没有实现拖动画文本的方式，也就是说只能在固定的位置绘制出固定大小的文本。这里我尚未想出设置文本边界定位的方式。

```
public void drawShape(Graphics2D g) {
        Font font = new Font(null, Font.BOLD, size);
        g.setFont(font);
    FontRenderContext context = g.getFontRenderContext();
    Rectangle2D stringBounds = font.getStringBounds(text, context);
    height = stringBounds.getHeight() / 2;
    width = stringBounds.getWidth();
    g.drawString(text, x, y + (int) height);
    }
public boolean contains(int x, int y) {
        innerRectangle = new Rectangle2D.Double(x, y, width, height);
        return innerRectangle.contains(x, y);
    }
```

修改颜色，大小，笔触粗细等实现较为简单，这里就不一一表明了。

2. 图形界面设计以及消息监听

最顶层的Main类中的main函数，就是创建了一个MyFrame窗口

```
public class Main {
    private static void createGUI() {
        MyFrame frame = new MyFrame("myFrame");
        frame.setSize(new Dimension(500,500));
    }
    public static void main(String[] args){
        createGUI();
    }
}
```

Myframe类

主要包含了三个部分，MyMenuBar类的成员，IconPanel的成员以及drawPanel的成员。最终的图形界面就是本文 ”功能介绍“->"初始界面"的样子

这部分的绘制相对比较简单。下面展开对应面板的实现，以及消息监听。

IconPanel类

这里面包含了一个内部类ColorPanel，也就是调色板，其内部还有colorButton类，辅助用。

```
private class ColorPanel extends JPanel{
        private static final long serialVersionUID = 1L;
        private final Color[] colorArray = {
            Color.BLACK, Color.RED, Color.YELLOW, Color.PINK,
            Color.GREEN, Color.CYAN, Color.GRAY, Color.BLUE, Color.ORANGE
        };
        private final int COLORNUM = colorArray.length;
        private colorButton[] colorButtons = new colorButton[COLORNUM];
        ColorPanel() {
            setPreferredSize(new Dimension(100,100));
            int row = (int)Math.sqrt(COLORNUM); //这里就是可开方的形式
            setLayout(new GridLayout(row,row));
            for(int i = 0; i < COLORNUM; i++) {
                colorButtons[i] = new colorButton(colorArray[i]);
                colorButtons[i].addActionListener(new ActionListener() {
                    public void actionPerformed(ActionEvent e) {
                        Color color = ((JButton)
e.getSource()).getBackground();
                        drawPanel.setColor(color);
                    }
                });
                add(colorButtons[i]);
            }
        }
        private class colorButton extends JButton {
            private static final long serialVersionUID = 1L;
            colorButton(Color color) {
            setBackground(color);
            setOpaque(true);
            setBorderPainted(false);
            setVisible(true);
            }
        }
```

调色板采用GridLayout，按照课程要求视频的形式绘制。里面包含了各种颜色的按钮(colorButtons数组)。其基本的事件监听就是修改绘画板以及选中对象的当前颜色。

此处需要注意的是，Mac中设置按钮的背景颜色，要增加几句内容（见colorButton(Color color)函数）

对于IconPanel来说，还有一部分内容就是一列的图标按钮。

这里通过按钮的setActionCommand以及事件的getActionCommand进行处理，从而修改绘画板上的对应的工具类型。

需要注意的是，这里由于要和绘画板交互，因此IconPanel有一个字段就是drawPanel，需要构造的时候传入。

MyMenuBar类

这个类的实现较为简单，就是最为基本的添加内容，增加MenuItem的消息响应输出即可。

由于有文件操作，而我们的数据就是通过将容器中的各种形状对象输出和输入获得的。因此这里会通过绘画板间接的调用形状容器，通过形状容器的对象输出流将文件进行数处。

ShapeContainer类

这里说用于实现MVC机制的Model部分和View的接口部分，也就是数据部分和显示接口部分。内部有一个MyShape的ArrayList，用于存储MyShape对象。同时也提供了内部对象的输出输入。其部分代码如下

```
public class ShapeContainer {
    private ArrayList<MyShape> shapeList = new ArrayList<MyShape>();
    public void drawShapes(Graphics2D g) {
        g.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);  //抗锯齿
        for(MyShape shape : shapeList) {
            shape.Draw(g);
        }
    }
    public void readFile() {
        JFileChooser fc = new JFileChooser();
        fc.showOpenDialog(null);
        File f = fc.getSelectedFile();
        try {
            FileInputStream file = new FileInputStream(f);
            ObjectInputStream in = new ObjectInputStream(file);
            Integer shapeNum = (Integer)in.readObject();
            shapeList.clear();
            for(int i=0;i<shapeNum;i++){
                MyShape tmp=(MyShape)in.readObject();
                if(tmp!=null){
                    if(tmp instanceof MyLine){
                        shapeList.add((MyLine)tmp);
                    }
                    else if(tmp instanceof MyRectangle){
                        shapeList.add((MyRectangle)tmp);
                    }
```

```
                    else if(tmp instanceof MyEllipse){
                        shapeList.add((MyEllipse)tmp);
                    }
                    else if(tmp instanceof MyText){
                        shapeList.add((MyText)tmp);
                    }
                    else{
                        System.out.println("ERROR: File is not parsable");
                    }
                }
            }
            in.close();
            file.close();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
    public void saveFile() {
        JFileChooser fc = new JFileChooser();
        fc.showSaveDialog(null);
        File f=fc.getSelectedFile();
        FileOutputStream file;
        try{
            file = new FileOutputStream(f);
            ObjectOutputStream out = new ObjectOutputStream(file);
            out.writeObject(shapeList.size());
            for(MyShape s : shapeList){
                out.writeObject(s);
            }
            out.close();
            file.close();
        }catch(IOException e){
            e.printStackTrace();
        }
    }
}
```

文件的保存和读入采用ObjectOutputStream和ObjectInputStream。仅仅保留对象的内容。对于之前的绘画板的状态不保留。

DrawPanel类

这里实现了MVC中的Control部分，也就是事件监听，同时也有view部分的repaint操作

对于control，主要有两部分，一部分是键盘事件的响应，一部分是鼠标事件的响应。

键盘事件的响应，较为简单。只需要对于自己设定的按键，执行选中的形状的对应方法即可。同时最后repaint

```
addKeyListener(new KeyAdapter() {
            public void keyPressed(KeyEvent e) {
                if(shapeChosen == null) {
                    return;
                }
                boolean shapeChanged;
                switch (e.getKeyCode()) {
                case KeyEvent.VK_MINUS:
                case KeyEvent.VK_D:
                    shapeChanged = shapeChosen.sizeDown();
                    break;
                case KeyEvent.VK_EQUALS:
                case KeyEvent.VK_U:
                    shapeChanged = shapeChosen.sizeUp();
                    break;
                case KeyEvent.VK_OPEN_BRACKET:
                case KeyEvent.VK_W:
                    shapeChanged = shapeChosen.strokeDown();
                    break;
                case KeyEvent.VK_CLOSE_BRACKET:
                case KeyEvent.VK_S:
                    shapeChanged = shapeChosen.strokeUp();
                    break;
                case KeyEvent.VK_BACK_SPACE:
                    shapeContainer.remove(shapeChosen);
                    shapeChosen = null;
                    shapeChanged = true;
                    break;
                default:
                    return;
                }
                if(shapeChanged) {
                    repaint();
                }
            }
        });
```

这里考虑到兼容性。对应的方法实现的时候，也提供了通过西文字母的方式控制。具体来说是D对应sizeDown，U对应sizeUp，S对应strokeDown，W对应strokeUp。回车键进行删除对象。当然，一般情况下，+对应sizeUp，-对应sizeDown，]对应strokeUp，[对应strokeDown

由于对应的形状都已经实现了相应方法，对于鼠标事件的响应也比较简单。因此就简单叙述实现思路，具体可见代码。

对于拖动画图，只需要调用对应形状的shapeDragged()方法即可，传入初始以及拖动时的坐标，不断的repaint。

对于选择形状移动，调用对应形状的move()方法即可，传入初始以及拖动时的坐标，不断repaint。

对于选择形状，遍历所有形状，调用contains方法判断。需要指出的是，我们首先判断的是直线。

结语

至此，整个项目的设计介绍基本结束。本项目难度适中，但需要学习比较多的课程上不讲的API，因此也是一个挑战。自己的coding能力还需要不断的练习与提高，从而适应更高难度的任务。整个代码的组织结构可能不算完美，自己对于IDE的使用也不甚熟练，还需要不断学习，多多进步。