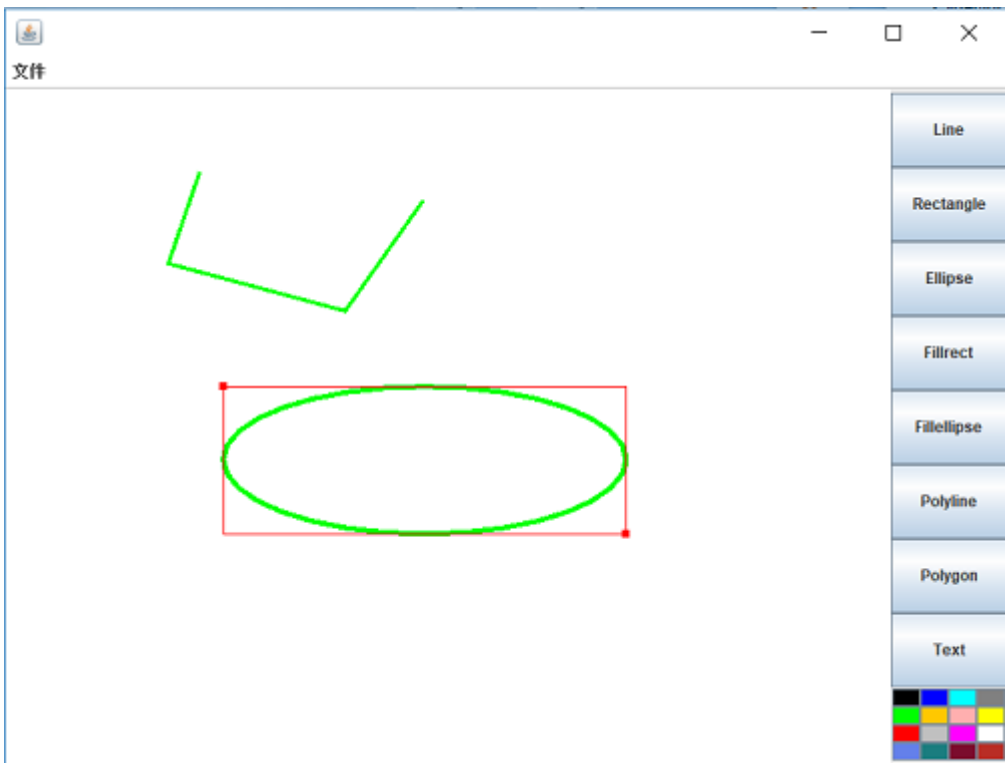
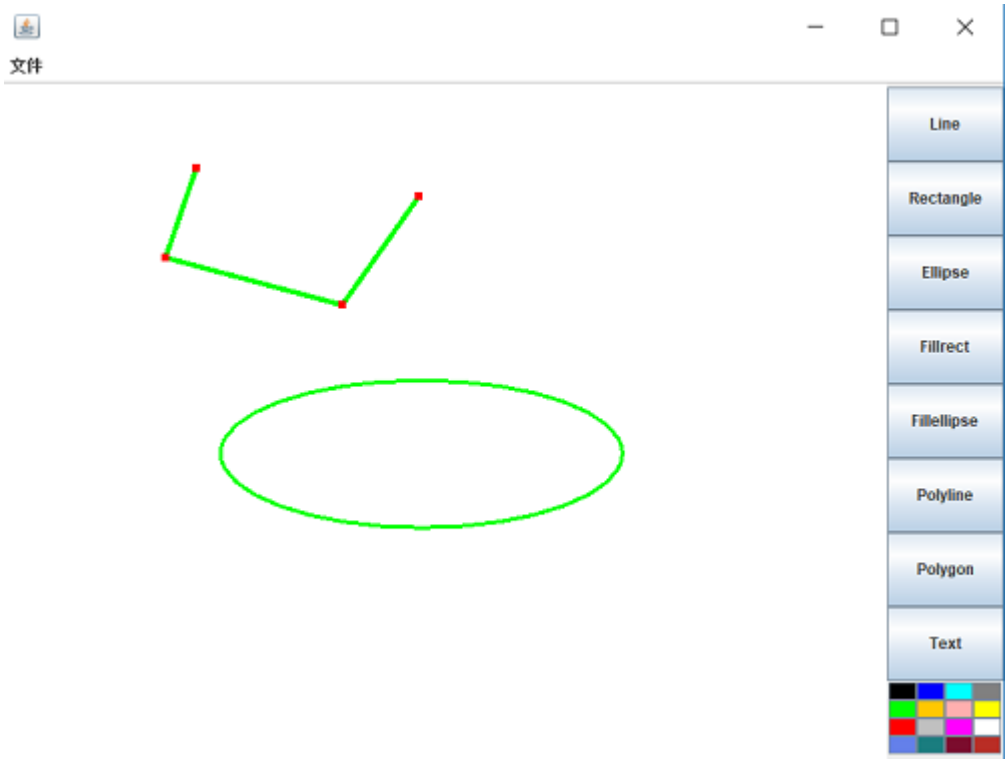
MiniCAD实验报告

1 功能简介

- 支持绘制多种图形
  - 线段
  - 矩形
  - 椭圆
  - 填充的矩形
  - 填充的椭圆
  - 多点折线
  - 多边形
  - 文字块
- 可以用鼠标选中已经绘制的图形
  - 在选中图形时，线段，矩形，填充的矩形多点折线，多边形和文字块会将其角点描红，同时并加粗选中图形的线条宽度。
  - 对椭圆金和填充的椭圆，会绘制出椭圆的外切矩形边缘，以示选中。





    - 在执行选中操作时，将鼠标箭头一直图形边缘，此时图形会对是否此时点击鼠标左键可以选中图形作出响应，如果响应可以选中，点击左键就可以选中图形。
- 可以移动选中的图形，点击选中对应图形后，可以使用鼠标拖动图形移动其位置。
- 可以修改选中的图形的颜色、大小、线条粗细和文字内容。
    - 选中图形后，单击颜色选择按钮，即可切换对应图形的颜色。
    - 选中图形后，按下“向上箭头”即可放大图形，按下“向下箭头”即可缩小图形。
    - 选中图形后，按下“1”键即可加粗线条，按下“2“键即可减小线条。
    - 选中文字内容后，按下“向右箭头“即可进行文字内容的修改操作。
- 可以删除选中的图形
    - 选中图形后，按下“退格键”即可删除对应图形。
- 可以将多绘制的图形保存在文件中。
    - 点击文件，保存即可将所绘制的图形保存在文件中。
- 可以将保存的文件中的图形加载到当前的图形中
    - 点击文件，打开选择对应的"jag"文件即可加载对当前的图形中。
- 可以对选中图形进行旋转操作
    - 选中图形后，按下"R键"可以实现当前选中图形绕其中心顺时针旋转的功能。

NOTE:

- 对图形绘制的时候，
    - `Line`，`Rectangle`，`Ellipse`，`Fillrect`，`Fillellipse`可直接在绘制面板上进行拖拽绘制。

- `Polyline`，`Polygon`,需要先用鼠标左键在面板上点击选择多边折线或多边形的点，然后点击鼠标右键进行绘制。
- `Text`，需要在面板上先点击选择文字块位置，然后输入相应内容。

2 主要原理

2.1 Model部分

该部分负责维护所有图形元素的基本数据。

在该部分定义了`Shape`基类，以该类为基础，实现了`Line`，`Rectangle`，`Ellipse`，`Poly`，`Text`等子类，同时`Model`负责存储所有图形的信息，并提供添加、删除图形的对应接口。

```
public abstract class Shape implements Serializable
{
    private static final long serialVersionUID = 1L;
    public Point pb;
    public Point pe;
    Color color;
    float thickness;
    final static double PI = 3.1415926;

    public int status;
    public abstract void draw(Graphics g);
    public abstract boolean isSelect(Point p);
    public boolean fullfill(boolean flag){
    }
    public Point resize(double sx, double sy, Point p, Point base){
    }
    public Point rotate(double theta, Point p, Point base){
    }
    public abstract void move(Point start, Point end);
    public abstract void changeAngle(double angle);
    public abstract void changeSize(double sx, double sy);
    public void setcolor(Color color){
    }
    public void setthickness(int flag){
    }
    public void setcontent(String s) {
    }
}
```

上述为对应基类`base`的相应数据结构及对应操作。通过上述基类，我们可以调用里面的函数实现图形的颜色，大小，粗细的调整操作等。

缩放操作实现

运用变换矩阵，先将选中图形移动至原点，在原点进行缩放后，再将选中图形移动至原来位置。最后实现选中图形可以以自己的中心为基准，进行缩放。

```
public Point resize(double sx, double sy, Point p, Point base)
{
    Point res = new Point((int)(p.getX() * sx + base.getX() * (1 - sx)),
(int)(p.getY() * sy + base.getY() * (1 - sy)));
    return res;
}
```

上述函数实现了对应点的变换，输入缩放比例，以及缩放基准点和要缩放图形的端点，返回缩放操作后图形的新端点所对应的位置。

颜色变换实现

对于颜色的变换，我们通过改变`Shape`类中的颜色的值来实现对此图形绘制时的颜色的选择，通过设置相应颜色，在重绘时就可体现出颜色变换的效果。

粗细调整实现

对于粗细的调整，我们通过改变`Shape`类中的画笔宽度的值来实现对此图形绘制时画笔的选择，通过设置相应粗细，在重绘时就可体现出图形粗细的相应结果。

选中操作实现

选中操作时，我们通过遍历所有图形的集合，来判断鼠标位置是否在某个图形的边界上，若是在边界上，即可选中图形。对于不同的图形选中操作的实现原理是不尽相同的。

- 线段，通过点到直线的距离，以及选段长度来控制对应区域实现选中。
- 矩形，通过判断矩形的边的位置，进而将可选中图形的位置，限制在四条边的狭小区域内。
- 椭圆，通过椭圆公式，来判断是否选中图形。
- 多点折线和多边形，以线段选中为基础，分别判断不同的线段即可。

删除操作实现

当选中图形时，通过移除该图形在`model`中对应的数据存储结构，实现删除操作，再重绘过程中，就可显示出删除的效果。

2.2 Control部分

Control部分设计的初衷，是希望通过Control部分可以实现`View`与`Model`的交互控制，即当`View`部分进行了一种操作，例如对底层数据结构进行了更改，便可以通过`Control`实现对`Model`数据结构的修改，当`Model`底层数据结构修改之后，通过`Control`可以通知`View`进行视图的更新。

但是在实现过程中，发现部分操作检测到`View`改变之后，如果通过`Control`对`Model`进行更新，会需要重复传较多参数，故部分操作，直接触发修改了`Model`数据。

该部分负责维护，`Model`部分数据的添加，例如当添加一个图形时，通过该部分实现添加操作；`Model`部分数据的删除，当执行删除操作时，通过该部分删除底层数据结构中的数据；当读取文件时，通过该部分进行相应数据结构的拷贝复制。

2.3 View部分

该部分主要负责对用户输入的读取，根据读取用户的相应操作实现对底层数据的修改；该部分还负责底层数据的显示，每次操作之后，通过重绘底层数据来显示用户所做的操作变化。因为刷新频率足够快，所以人的视力基本看不出刷新变化。

对不同的图形，我们采用的读取方式不太相同，具体可分类两部分，我们将线段，矩形，椭圆，填充矩形，填充椭圆归为一类，将多点折线，多边形，文字块归为一类。

第一类读取方式

对于第一类，进行绘制时，需要实现动态绘制的操作，即鼠标每拖动一点，便需要绘制一个新的图形出来。故我们选择实现`mousePressed()`函数，当所绘制的图形，属于此类时，我们根据鼠标位置确定第一个点，然后根据鼠标拖动时选取第二个点，当拖动时，进行动态绘制，拖动结束后所要添加的图形的属性也全部确定，此时将对应数据添加到底层数据结构。

第二类读取方式

对于多点折线，多边形，采取的绘制方式是先通过描点，确定折线或多边形的各个顶点，当点确定之后，通过点击鼠标右键实现绘制。在绘制过程中，为了将所绘制的点描绘出来，往底层数据结构添加了多个相应的点直线(即绘制起点终点相同的直线)，当确定折线和多边形的属性之后，我们将所添加的点依次移除。

对于文字块，我们通过鼠标点击，选中文字块的位置，即可进行绘制。

故对第二类，采用实现`mouseClicked()`函数的方式。

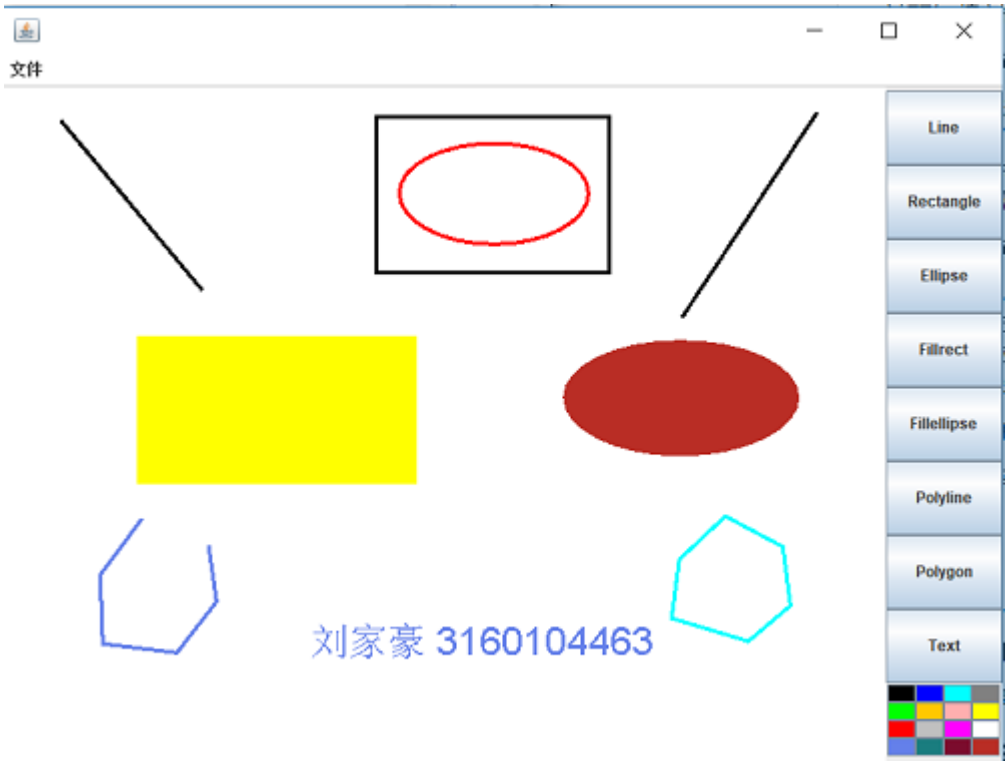
选中操作

在`View`部分，读取鼠标点的位置，然后遍历底层数据结构，根据点到不同图形的距离判断是否有图形能被选中，若此时有图形能被选中，单击左键即可选中对应图形。

颜色，粗细，删除操作

这类操作，均是点击相应按键时，触发`Model`中的各类对应操作实现相应功能。

3 实验结果



通过测试，可以实现作业要求的所有功能。与此同时自己添加了可以旋转选中图形的相应操作。具体如何进行旋转操作及其实现方式，将在另一个单独的文档中进行讲解。

为了便于说明相应的功能，会在相应文件夹里添加操作的相应GIF演示视频。

4 实验总结

通过此次实验自己对java gui的相应编程方式有了进一步的了解，与此同时，对MVS模式的编程有了相应的了解，在此次实验中，自己尽力使用MVC模式进行构造相应代码，虽然最后对MVS的真正实现仍有很大差距，但是这在一定程度上锻炼了自己的能力。