Project1 miniCAD

1 实验内容

用 Java 的 awt 和 swing 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。功能请参考视频演示。

2 构建方法

本项目使用了 **jdk-17.0.5**，在 Ubuntu22.04 上，本项目的构建方法如下：

```
$ mkdir out
$ javac View/*.java Controller/*.java Shapes/*.java -d out
$ cd out
$ touch MAINFEST.MF
```

向 `MAINFEST.MF` 中写入以下内容：

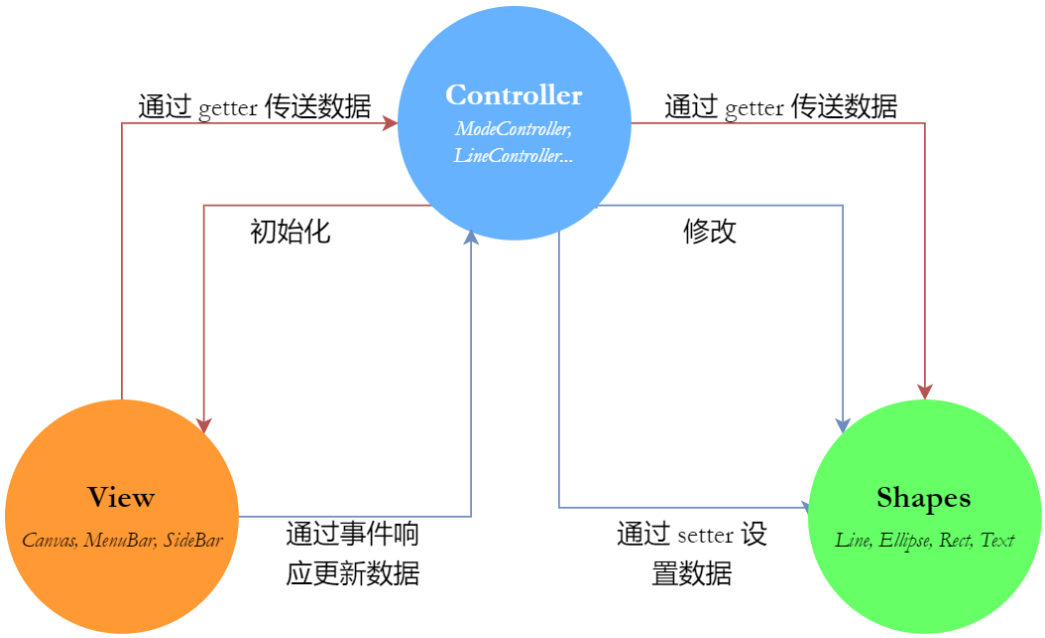
```
Manifest-Version: 1.0
Main-Class: View.Frame
```

继续在 `out` 目录下执行以下命令：

```
$ jar cfvm CAD.jar MAINFEST.MF ./Controller/*.class ./Shapes/*.class ./View/*.class
$ java -jar CAD.jar
```

3 设计架构

本 miniCAD 的设计采用 MVC 架构，即：



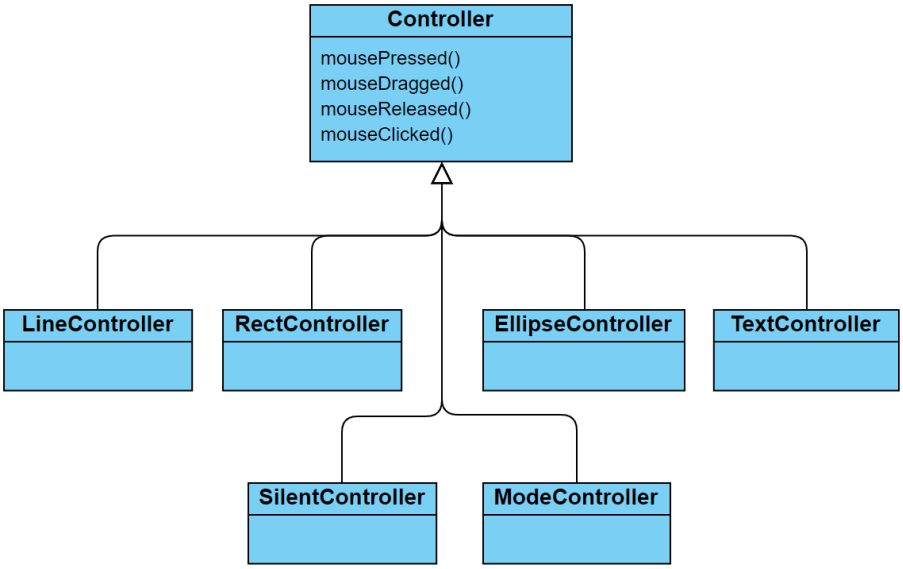
本项目工程结构按照 MVC 架构分成了三个 package，如下图所示：

```
kenshin@VM-4-16-ubuntu ~/CAD
$ tree
.
├── Controller
│   ├── Controller.java
│   ├── EllipseController.java
│   ├── LineController.java
│   ├── ModeController.java
│   ├── RectController.java
│   ├── SilentController.java
│   └── TextController.java
├── MAINFEST.MF
├── Shapes
│   ├── Ellipse.java
│   ├── Line.java
│   ├── Rect.java
│   ├── Shape.java
│   └── Text.java
└── View
    ├── Canvas.java
    ├── Frame.java
    ├── MenuBar.java
    ├── Mode.java
    └── SideBar.java

3 directories, 18 files
```

3.1 Controller

Controller 包中包含了所有事件处理函数的实现，对应 MVC 中的 Controller。类间的继承关系如图：



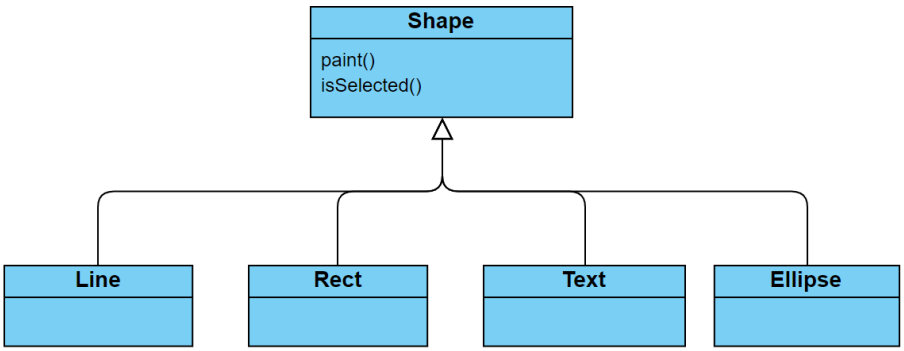
其中，`LineController` `RectController` `EllipseController` `TextController` 分别对基类中定义的四种鼠标事件进行了按需实现。`SilentController` 是在没有指定任何 Controller 时的默认选项（不做任何事）。`ModeController` 对三种模式（选择（含移动）、删除、缩放）对应的事件进行了实现。

- `LineController`：实现了鼠标按下、鼠标松开、鼠标拖动
- `RectController`：实现了鼠标按下、鼠标松开、鼠标拖动
- `EllipseController`：实现了鼠标按下、鼠标松开、鼠标拖动

- `TextController`：实现了鼠标点击

3.2 Shapes

Shapes 包中包含了所有图形类的实现，对应 MVC 中的 Model。类间的继承关系如下（类方法部分缺省，缺省的部分是 getter 和 setter）：



其中，`paint` 函数是各个图形类真正被绘制的地方，`isSelected` 函数用来判断图形是否被选中，选中的算法方便起见，在覆盖图形的最小矩形框内即判定为选中。

除此之外，`Shape` 类实现了 `Serializeable` 接口，用于后续保存和恢复。

3.3 View

View 包中包含了所有面板组件的实现，同时在这里完成图形的绘制，对应 MVC 中的 View。

3.3.1 MenuBar

`MenuBar` 类中包含了包含了保存文件和打开文件的两个按钮，同时对这两个按钮进行了监听。对保存按钮，将所有的图形序列化形成二进制文件；对于读取文件，从二进制文件中将所有的图形反序列化并构造 `Shape` 对象。

3.3.2 Canvas

`Canvas` 是图形元素被绘制的地方，因此这个类中记录了一些必要的信息用于判断当前的状态，如已绘制的图形列表、临时图形、当前的模式、画笔的颜色和粗细等。

本类中注册了四种鼠标事件监听函数，分别对应 `Controller` 中的四个监听函数，用于监听鼠标事件。

本类中重写了 `paint(Graphics)` 函数，用来绘制所有图形元素。

3.3.3 SideBar

`SideBar` 是所有受支持操作按钮的列表，采用了 GridBagLayout，支持在窗口缩放时自适应大小。为了简化，我将所有模式（直线、椭圆、矩形、文本、拖动、删除、放缩）都做成了按钮，改变颜色也做成了按钮，有 9 种颜色可供选择，改变画笔的粗细我做成了滑块的形式，可以连续调节粗细。

类似地，我为每一个按钮以及滑块注册了事件监听函数，并将控制流交由 Controller 中的对应类进行处理。

- 直线模式：由 `LineController` 处理
- 椭圆模式：由 `EllipseController` 处理
- 矩形模式：由 `RectController` 处理

- 文本模式：由 `TextController` 处理
- 拖动、缩放、删除：由 `ModeController` 处理

改变颜色和粗细，事件处理函数直接以 lambda 表达式的形式写在了 SideBar 里。

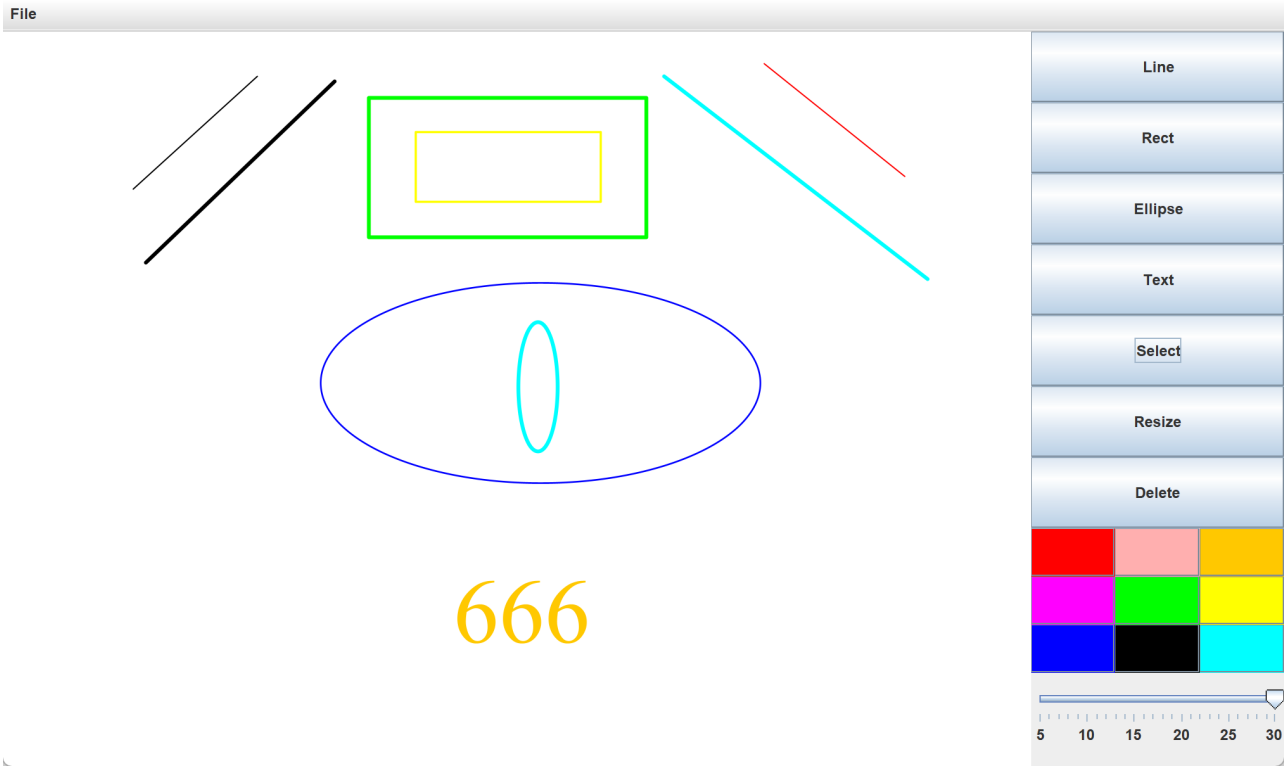
3.3.4 Frame

前面所述的三个组件展示在 Frame 中，采用 BorderLayout。其中 MenuBar 位于北边，Canvas 位于中央，SideBar 位于东边。

4 成果展示

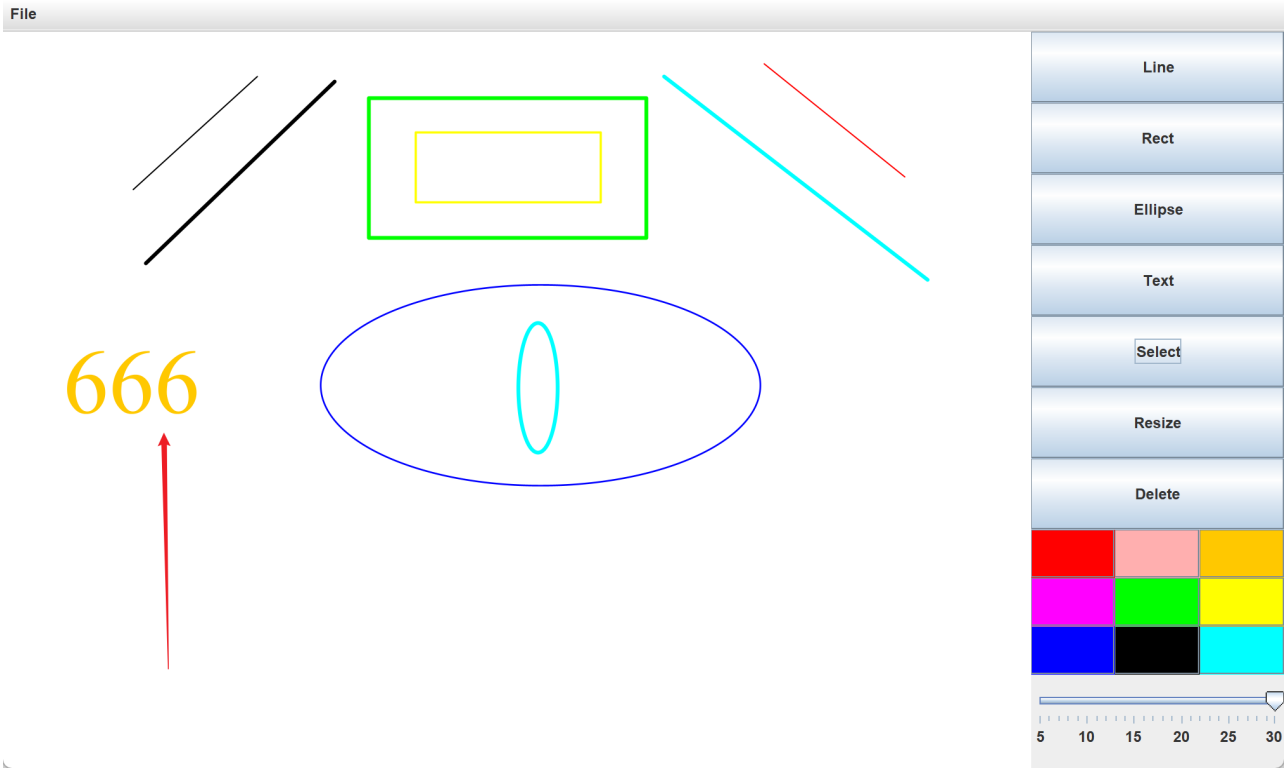
4.1 新建图形

包含了所有图形元素的创建，粗细的修改和颜色的修改。



4.2 拖动

对比 3.1 中的 `666`，我们可以发现其的位置发生了改变。

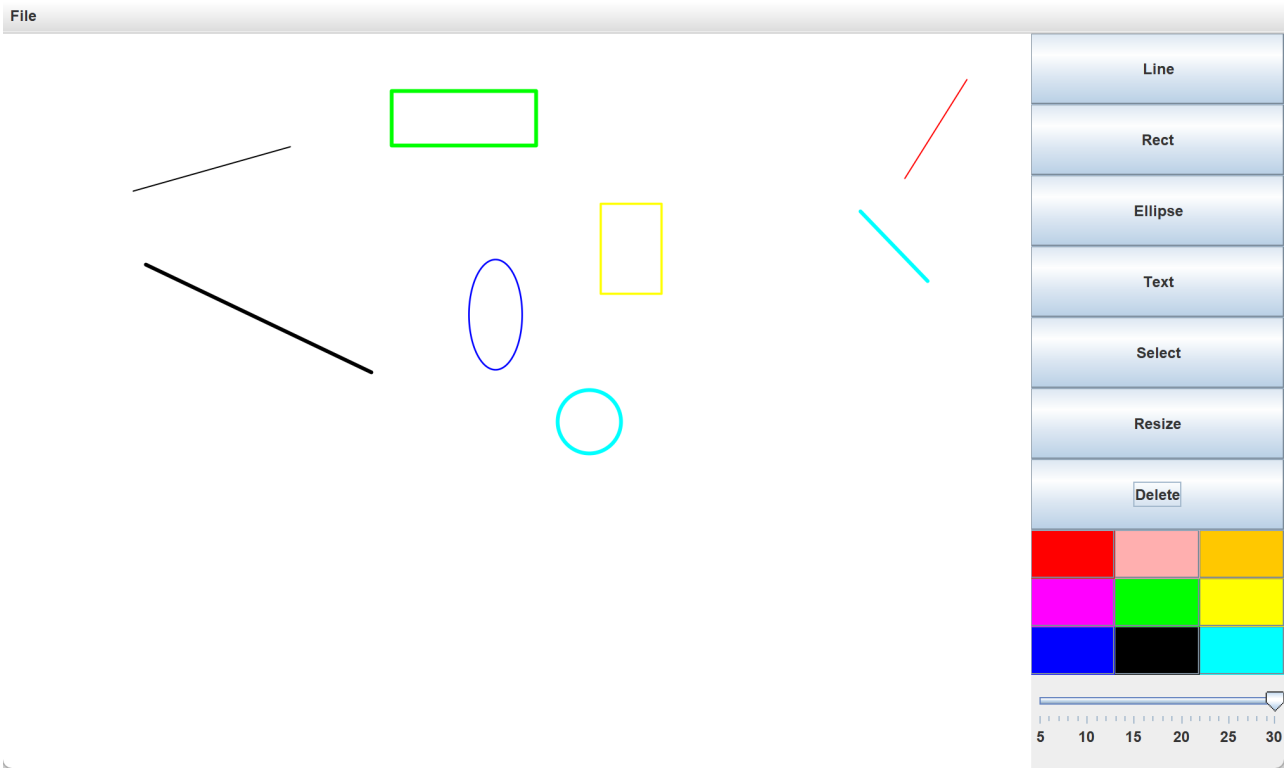


4.3 缩放

对比 3.1 中的所有图形，我们可以发现图形的大小均发生了变化。

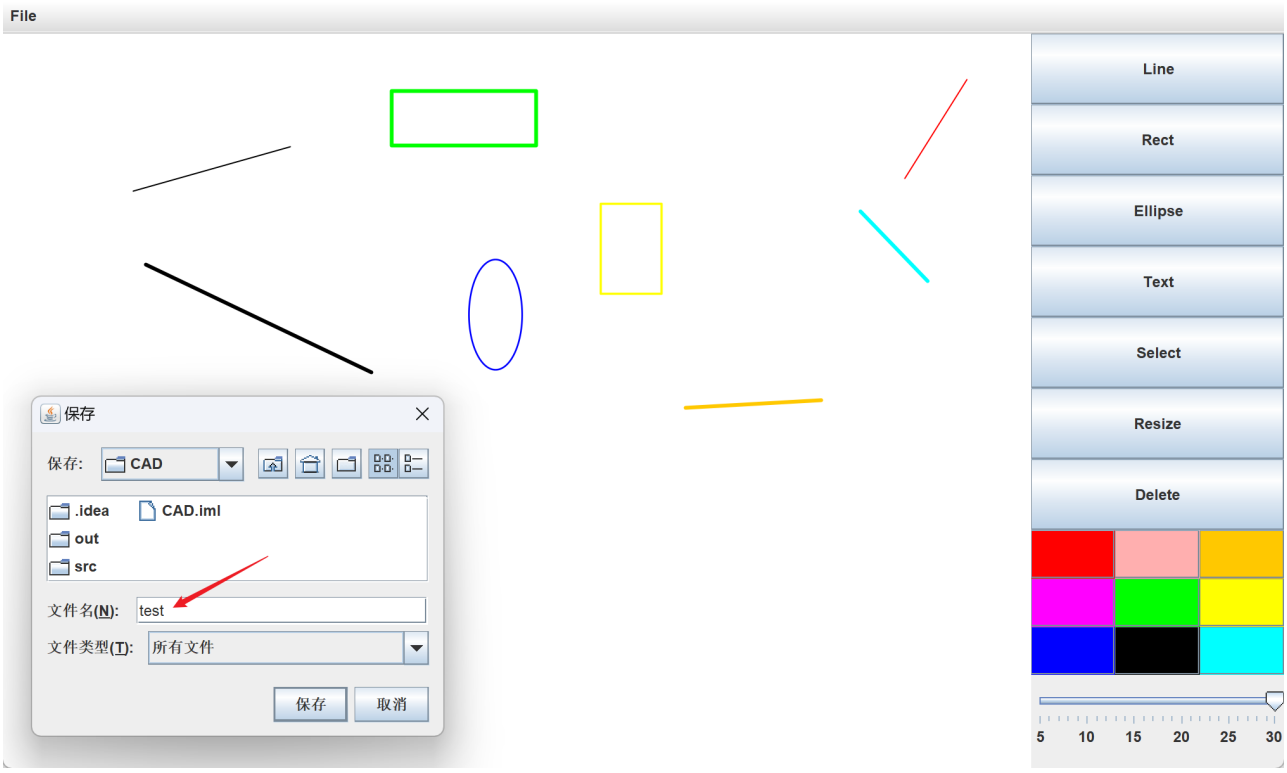
4.4 删除

对比 3.1，我们可以发现 `666` 被删除。



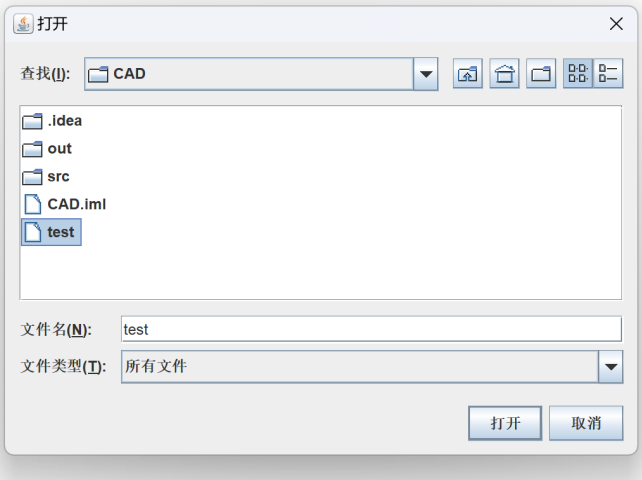
4.5 文件保存、打开

保存文件，命名为 test。



再次打开，可见文件成功恢复：

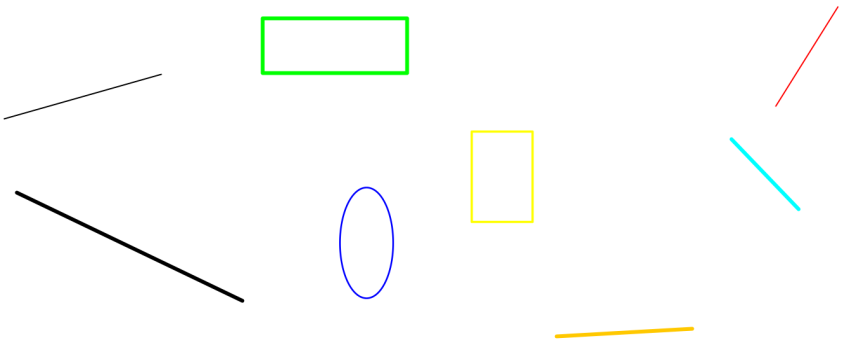
File



Line
Rect
Ellipse
Text
Select
Resize
Delete

5 10 15 20 25 30

File



Line
Rect
Ellipse
Text
Select
Resize
Delete

5 10 15 20 25 30