面向对象设计方法大作业报告

201250182, 郑义, 201250182@smail.nju.edu.cn

2022 年 5 月 25 日

目录

1 引言

1.1 开发需求简介

南京大学 2022 春季学期计算机科学与技术系面向对象设计方法课程大作业的开发任务主要是通过一种面向对象的语言来使用 GoF23 种设计模式实现一个类似 PPT 画图功能的画图工具，该项目包括的基本需求有以下几个：

1. 设计良好的图形用户界面，界面中要求至少有默认大小的三角形、方框、圆形、椭圆、连接线等五种元素可供用户选择后，绘制到画布上。

2. 允许用户添加文字描述

3. 单击可以选中图形，并允许对图形的拷贝复制

4. 多个图形可以组合，组合后的图形同样有拷贝复制功能

5. 支持撤销上一步操作的功能

除此之外，本课程的大作业项目也包括了一些在基础需求之上的扩展需求，在实现了基本功能的基础上可以尝试来扩展系统并实现下述的一些扩展功能，需求中的扩展需求主要包括：

1. 右键点击图形可通过文本框对图形的尺寸进行调整

2. 支持个性化界面，包括设置界面字体大小、界面皮肤等

3. 支持图形（包括组合图形）的拖拽调整图形大小

4. 支持撤销多步的功能

5. 设计一种硬盘文件存储格式可以保存用户绘制的图形，并可以加载

1.2 核心技术简介

本项目主要采用了 Java 作为开发语言，ubuntu20.04 + JDK1.8 作为开发环境，主要使用的核心技术为 Java 自带的用户图形界面库 AWT 和 Swing。同时使用 Maven 来构建项目和进行对项目依赖的管理，本项目主要采用的第三方依赖包括：

1. lombok: 一款简化开发的插件，通过注解能够提供 getter、setter 和 toString 等便捷实现

2. Junit5：一款优秀的测试框架，便于在实现功能之后对业务逻辑进行一定的单元测试

AWT 是一个抽象的窗口工具包，它提供了各种组件类，如 Label、Button、TextField 等，用于在屏幕上显示窗口组件。其封装好了很多 GUI 组件，能够快捷方便的在容器上对组件进行快速布局和创建 GUI 应用程序。

Swing 是建立在 AWT 之上并完全用 Java 编写的 JFC（Java 基础类）的一部分。javax.swing API 提供了所有组件类，如 JButton、JTextField、JCheckbox、JMenu 等。很多组件都是在 AWT 基础上进行了扩充与封装，丰富了原本组件的功能，使得开发者能够将不同的容器和组件进行组合实现自己的预期效果。

这里主要想要介绍一下在 AWT 和 Swing 中原本就用到的设计模式，其用到了包括适配器模式、观察者模式等设计模式，为开发者提供了一套方便快捷且强大的图形界面库。

2 设计介绍

2.1 设计包图

本项目中的包主要分为 draw、handler、window、factory、enums、memento、utils，其中最为重要的两个包是 draw 和 window, 分别负责的是图形类的建立和用户界面的建立。其

他几个包主要是为了实现某个功能而被 window 所依赖，handler 包中主要存放的是观察者模式下的被观察对象，存储了画图工具中的一些全局共享的信息，factory 包中则是创建了几个工厂用以实现简单工厂模式来创建图形，具体会在后面的章节述说。enums 类中定义了几个枚举类，用以增强代码的可读性，memento 包是为了实现备忘录模式实现撤销功能，具体的开发包图如下图所示:

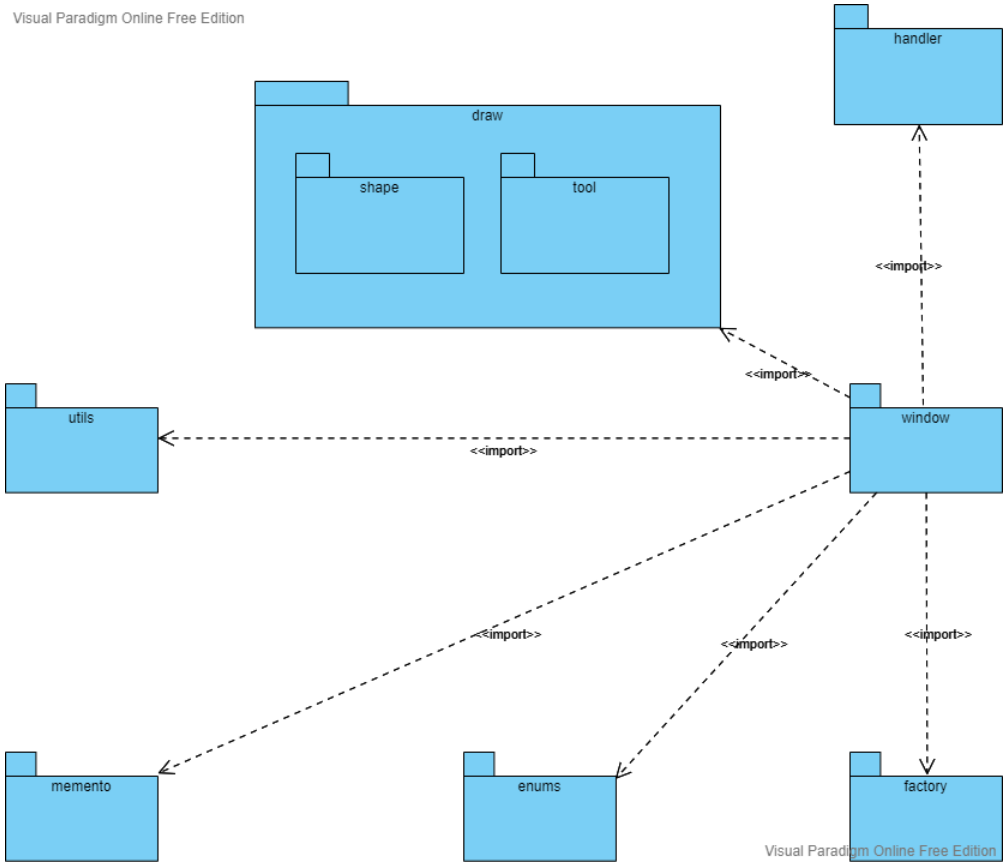


图 1: 设计包图

可以看到，window 包是本项目的重点，其依赖了其他所有的包，window 包中实现了画图工具的界面以及绘制图形等功能，可以说承担了项目中大部分的职责。

2.2 设计类图

下面简要说明一下本项目中的类的设计，为了实现对接口和对抽象编程，以及为了满足面向对象设计的七大原则，本项目在设计中定义了一个有关内容绘制的基类 AbstractContent，该类中定义了所有能够操作的内容的共有定义，比如说图形的绘制起始点和起始终止点和图形的颜色等，所有能够操作的图形和画笔之类的都需要继承该父类。

在 AbstractContent 类下又有 AbstractShape 和 AbstractPaintTool 两个类继承了 AbstractContent，分别代表了抽象的图形类和抽象的画图工具类。AbstractShape 中定义了图形类应该有的一些方法和 draw 方法的公有实现，在该基类下实现了各种具体的图形，包括三角形、圆形、矩形等，AbstractPaintTool 下实现了各种画图的工具（如画笔、橡皮擦、笔

刷等)。同时定义了 Text 类和 Image 类继承了 AbstractContent，Text 类中定义了文本所拥有的一些属性，比如是否是斜体，是否是粗体，字体大小与字体等。该部分的继承结构如下图所示：

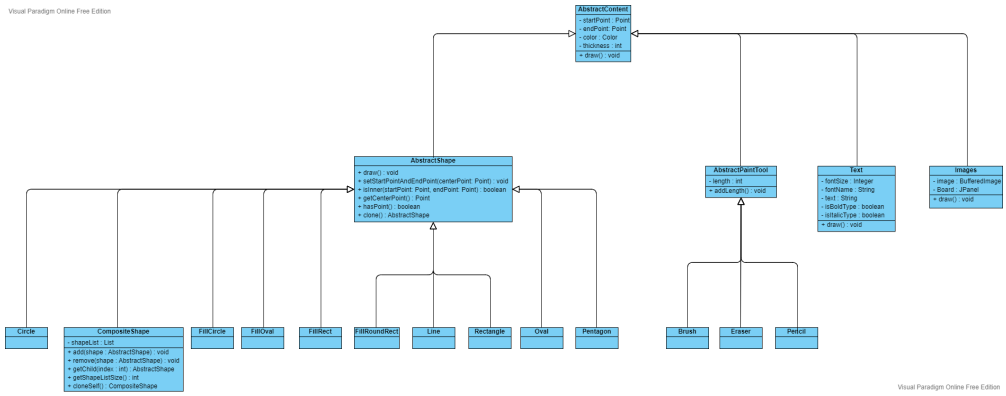


图 2: draw 包设计类图

window 包下包括四个类，分别是 ColorPanel、DrawPanel、MainFrame、PaintMenu 和 PaintToolBar。分别是代表画图工具的颜色选择栏，画布区，主要的窗体（用以组织其他的组件）、菜单栏和工具栏。主窗体是存放其他四个组件的容器，四个组件都放在主窗体之上，同时主窗体也用来设置一些界面的信息，比如该应用程序的标题和图标等。

ColorPanel 主要使用的是 Swing 中的 JColorChooser 组件，通过该组件可以实现对颜色的选择，具体会在后面的实现方案中介绍。DrawPanel 中则是组合了上述的抽象内容类 AbstractContent，可以绘制具体的图形，并实现撤销组合等功能。PaintMenu 是程序的菜单栏，可以通过菜单栏将当前的画布上的内容保存至图片等功能，PaintToolBar 是工具选择栏，可以选择所要绘制的图形或者调整文字的字体和大小等。window 包的设计类图如下所示：

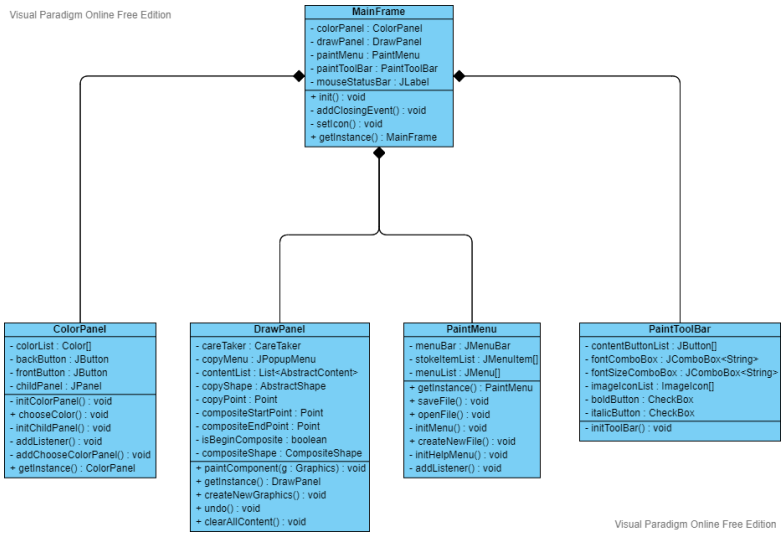


图 3: window 类设计包图

需要注意的是，window 包下的五个类都是使用 Java 中的静态内部类单例模式实现的，即可以直接通过 getInstance() 方法来获得某一个类的唯一实例，因此其实部分的两个类之间是有聚合关系的，即在某个方法内可能引用了另一个类，但更强力的组合关系只有 MaiFrame 中组合了另外四个类，因为 MainFrame 是另外四个类的容器，是一种包含的关系。

除了这两个包的类图之外，其他包内的类图较为简单，memento 包中包含的类包括 Memento、Originator、CareTaker 三个类，其中只有 CareTaker 类与 DrawPanel 之间有聚合关系。在 handler 包中的 GlobalStateHanlder 的作用是类似观察者模式中的被观察者，存储着一些画图工具的共有变量，比如说当前选择的颜色，当前选择的图形等，与 window 包下的多个类有聚合关系。这些将在后面的实现方案中具体描述如何实现。

3 实现方案

下面简要介绍本项目的实现方案，主要根据需求来描述

3.1 图形绘制

图形绘制主要是使用了 Swing 自带的 paintComponent() 方法，在上面我们提到了我们定义了基类 AbstractContent 来对所有可绘制的内容进行抽象，在该类中定义了一个抽象方法 draw(Graphics2D g)，该方法是用来让不同的内容实现不一样的绘制过程。在 AbstractShape 中，定义了图形绘制共有的 draw 方法的部分，如下所示：

```
@Override
public void draw(Graphics2D g) {
        // 设置颜色
        g.setPaint(color);
        // 设置线宽
        g.setStroke(new BasicStroke(thickness));
        // 设置渲染算法
        g.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
        RenderingHints.VALUE_ANTIALIAS_ON);
}
```

这部分代码设置了图形的颜色，绘制的边缘线的宽度和渲染的抗锯齿算法，能够让图形的边缘更加平滑，对 AbstractShape 的子类来说，在实现自己的 draw() 方法的时候，只需要调 super.draw(g) 就可以复用父类中已经实现的部分 draw 方法的代码，然后再实现自己的即可，下面我们以绘制圆为例子简单的叙述一下整个图形绘制的过程，在 Circle 类当中，它的 draw 方法如下所示：

```
@Override
public void draw(Graphics2D g) {
        super.draw(g);
        drawOval(g);
```

```
}

private void drawOval(Graphics2D g) {
        int radius = getRadius();
        g.drawOval(PointUtil.getMinPointX(startPoint,
            endPoint), PointUtil.getMinPointY(startPoint,
            endPoint), radius, radius);
}

private int getRadius() {
        return Math.max(Math.abs(startPoint.getX() -
            endPoint.getX()), Math.abs(startPoint.getY() -
            endPoint.getY()));
}
```

对于不同的图形来说，只有最后这一步 draw() 方法的实现是不同的，前面的设置颜色、线宽和抗锯齿算法等都是相同的，因此我们可以在父类中定义并重用即可，对于不同的绘制来说，Swing 提供了很方便的画笔绘制功能，如上述所示的 g.drawOval() 便可以通过一个圆的起始点、结束点和半径便绘制出一个精美的圆。

然后我们来看看在 DrawPanel 中重写的 paintComponent() 方法，在上面类图中提到我们在 DrawPanel 中存储了一个绘制图形的列表，在 paintComponent() 方法中，我们只需要将这个列表中的所有内容绘制出来即可，Java Swing 提供了 repaint() 方法，该方法的实现实际上就是调用一次 paintComponent() 方法用以刷新整个画布，DrawPanel 中的 paintComponent() 方法部分实现如下:

```
@Override
public void paintComponent(Graphics g) {
        super.paintComponent(g);
        // 定义画板
        Graphics2D g2d = (Graphics2D) g;
        for (AbstractContent content : contentList) {
                draw(g2d, content);
        }
}

private void draw(Graphics2D g2d, AbstractContent content) {
        // 将画笔传入到各个子类中，用来完成各自的绘图
        content.draw(g2d);
}
```

可以看到实际上该方法的实现就是遍历整个 contentList 列表，并将其中的内容逐一通过我们上述所说的 draw() 方法来绘制出来，因为不同的子类实现了不同的 draw() 方法，因

此我们就可以通过多态来得到各种各样精美的图形，可以看到这边使用了 AbstractContent 作为方法的参数，这种面向抽象编程的方法可以很好的将程序抽象化，不必关心传过来的具体内容是什么，而只要知道该内容肯定有属于自己的 draw() 方法即可。

在这个基础上我们只需要在画布上添加鼠标的监听事件，监听鼠标的移动就可以获得用户想要绘制圆的起始坐标和结束坐标，然后通过这两个坐标就能够算出圆的半径来进行绘制，上述代码的 startPoint 和 endPoint 便代表了用户在使用鼠标起始绘制和结束绘制的像素点的坐标，这里使用了 适配器模式，在 DrawPanel 中实现了一个内部类，该内部类继承了 MouseAdapter 来对 MouseListener 进行适配，MouseAdapter 实现了 MouseListener 的接口提供了默认的空实现，这样就可以不用实现 MouseListener 中的全部接口，而只要选择自己想要实现的接口即可，对于本项目中的绘制操作来说，最为重要的监听接口有两个，一个是 mousePressed() 方法，另一个是 mouseReleased() 方法。下面展示了部分 mousePressed() 方法的内容:

```
AbstractContent content = getCurContent();
content.setStartPoint(new Point(e.getX(), e.getY()));
content.setEndPoint(new Point(e.getX(), e.getY()));
```

可以看到在用户使用鼠标点击的时候，会获得用户点击的事件 MouseEvent e，然后可以获得点击的坐标，并将该坐标设成当前选择的绘制内容的起始坐标，mouseReleased() 同理，只是获得的是绘制内容的结束坐标（鼠标释放即代表绘制结束），之后只需要调用 repaint() 就可以对画布进行一次刷新，绘制出图形。

当然，上述的实现只能够在释放鼠标的时候才能够看到绘制的效果，那么我们该如何实现可以在鼠标拖动的过程中 动态 的看到图形大小的变化呢？其实原理十分简单，通过监听鼠标的拖动来不断更新当前图形的 endPoint 并不断重新刷新画布即可。为了实现这个效果本项目在 DrawPanel 实现了另一个内部类来继承 MouseMotionAdapter，与上述相同，该类也是采用适配器模式来减少了我们要实现的接口数量，通过实现该类中的 mouseDragged() 即可实现上面我们想要的方法，该方法负责监听鼠标的拖动，因此我们只需要在鼠标拖动的过程中不断更新当前图形的坐标并重新绘制画布，就可以获得通过鼠标拖拽时实时变化图形大小的效果。下面展示了部分的 mouseDragged() 方法

```
@Override
public void mouseDragged(MouseEvent e) {
        AbstractContent content = getCurContent();
        ContentType curContentType = GlobalStateHandler.
                                getCurContentType();
        content.setEndPoint(new Point(e.getX(), e.getY()));
        repaint();
}
```

主要逻辑就是获得当前用户选择的图形，然后更新该图形的 endPoint，即可达到实时更新图形大小的效果。

还有一个需要提及的点是在图形绘制中使用到了简单工厂模式，用户选择对应图形时我们只设置了 GlobalStateHandler 中的成员变量 curContentType，创建图形的职责则是

交给了 ContentFactory 类来进行创建，该类中的静态方法 createContent() 能够根据参数 curContentType 来产生具体的 AbstractContent 类并返回，这样就做到了解耦和职责的分离，DrawPanel 并不需要知道如何创建对象，只需要根据 curContentType 传参给 ContentFactory，就可以获得创建好的对象并进行绘制。

总结一下，关于图形绘制的需求主要使用到了适配器模式和简单工厂模式来实现，同时通过抽象父类的定义，做到了面向抽象编程，同时采用了源-监听器模式来监听鼠标事件并进行对应的操作。

3.2 文字描述添加

文字描述的添加其实在实现中是与图形绘制差不多的，文字也是继承了 AbstractContent 类，差异点可能在于文字的绘制只需要起始的 startPoint 其实就足够了，以及文字可以设置字体、字体大小、斜体、粗体等，其他与上面的绘制图形基本类似。

在项目实现中，我们把字体、字体大小、斜体、粗体都存放在了 GlobalStateHandler 中作为共有的变量进行管理，且对字体和字体大小的选择采用了 Swing 中的 JComboBox 类来进行实现，该类是一个下拉选择框，通过将所有可能的选项存放在列表里可以很方便的让用户来下拉选择对应的字体和字大小。在 Text 的 draw() 方法中，是通过如下方式来实现的：

```
@Override
public void draw(Graphics2D g) {
        g.setColor(color);
        g.setFont(new Font(fontName, (isBoldType ?
            Font.BOLD : Font.PLAIN) + (isItalicType ?
            Font.ITALIC : Font.PLAIN), fontSize));
        if (text != null) {
                g.drawString(text, startPoint.getX(),
                    startPoint.getY());
        }
}
```

可以看到 Swing 很方便的提供了设置粗体和斜体的接口，我们只需要记录用户的操作即可，项目中使用了勾选框来记录用户是否选择斜体或者粗体，然后存到 GlobalStateHandler 中以便其他观察者类进行访问获得和更新

3.3 拷贝复制

拷贝复制的实现其实就是用了 原型模式 来对图形进行浅拷贝（因为图形的定义中的成员变量实际上都并不是对象，不需要深拷贝来创建新的对象）。在 AbstractShape 中实现了 Cloneable 接口并重写了 Clone 接口，如下所示：

```
@Override
public AbstractShape clone() {
        try {
```

```
            return (AbstractShape) super.clone();
        } catch (CloneNotSupportedException e) {
            throw new AssertionError();
        }
    }
```

通过原型模式可以方便的获得一个新的拷贝对象，下面要解决的问题就是我们要如何判断当前鼠标点击的时候是要复制哪一个对象了，这里采用的是右键弹出框的形式，Java Swing 中提供了 JPopUpMenu 来实现弹出框，通过监听鼠标右键点击来弹出弹出框，同时会获得鼠标点击的坐标，项目中实现判断哪个图形是根据点击点是否在图形内部来判断的（在图形重叠时，只会判断到比较早画的图形），这里就体现了在前面定义了抽象类 AbstractShape 的好处，在该类中我定义了抽象方法 hasPoint(Point point) 来判断该图形是否包含某个点，具体的图形类可以根据自己的图形特征来实现，比如圆形的实现就是判断该点到圆心的距离是否超过半径来判断该点是否在圆内。通过判断能够获得用户所点击的地方所在的图形，然后对该图形进行拷贝并暂存即可。

在粘贴的时候的难点主要是，AbstractShape 是通过 startPoint 和 endPoint 来确定一个图形的位置的，那么在粘贴的时候我们该如何确定新的图形的位置呢？我的实现是通过新旧位置的平移来确定，即在粘贴的时候监听鼠标事件，获取鼠标坐标，以该坐标作为粘贴图形的中心点来平移图形的 startPoint 和 endPoint 即可。大概的实现代码如下所示：

```
private void addPasteListener(JMenuItem item) {
        item.addActionListener(e -> {
            // 绘制图形
            if (copyShape != null) {
                // 往列表中添加一个新的图形即可
                // 需要克隆
                AbstractShape shape = copyShape.clone();
                // 设置新的坐标
                shape.setStartPointAndEndPoint(copyPoint);
                contentList.add(shape);
                repaint();
            }
            isBeginComposite = false;
        });
    }
```

通过给 item 加上监听事件，就可以很方便的通过点击按钮来实现粘贴图形的功能了。

3.4 组合与拷贝复制

组合主要采用的是组合模式，上面所说的 AbstarctShape 实际上就是 Component 类，用来抽象化叶子和组合类，同时定义了一个 CompositeShape 来作为组合图形类，这里采用的是安全组合模式，即在 AbstractShape 中没有定义 CompositeShape 中的 add()、remove()

等接口，CompositeShape 存放了一个组合起来的图形的列表 List<AbstractShape>，并在 draw() 方法中实现了遍历列表并调用所有 AbstractShape 的 draw() 方法。如下所示：

```
@Override
public void draw(Graphics2D g) {
        for (AbstractShape shape : shapeList) {
                shape.draw(g);
        }
}
```

组合图形的拷贝复制与上述的拷贝复制基本同理，唯一不同的地方是需要深拷贝，即 CompositeShape 中所存储的 Shape 都需要拷贝一份，实现同样简单，只需要遍历列表调用列表中每一项的 clone() 方法即可，如下所示：

```
public CompositeShape cloneSelf() {
        CompositeShape cloneCompositeShape =
                                new CompositeShape();
        for (int i = 0; i < getShapeListSize(); i++) {
                AbstractShape shape = getChild(i).clone();
                // 通过平移的距离来获得中心的点
                cloneCompositeShape.add(shape);
        }

        return cloneCompositeShape;
}
```

组合边界的确定也是通过绘制图形中所描述的两个鼠标事件监听器，分别监听获得组合框的起始点 compositeStartPoint 和结束点 compositeEndPoint，然后遍历所有已经绘制的图形，看哪个图形的起始点 startPoint 和 endPoint 在这两个点所形成的矩形范围内，如果在的话就加入 compositeShape 的列表中。

3.5 撤销

撤销主要采用的是 备忘录模式，定义了 Memento 类、Originator 类和 CareTaker 类来负责存储每一步绘制的过程，撤销则是从 CareTaker 的列表中移除 Memento 即可。

```
/**
 * @author Zyi
 */
public class CareTaker {

        /**
         * 存储绘制的图形列表
         */
```

```
        private List<Memento> contentList;

        private CareTaker() {
                contentList = new ArrayList<>();
        }

        public static CareTaker getInstance() {
                return InnerClass.INSTANCE;
        }

        public Memento getMemento(int index) {
                return contentList.get(index);
        }

        public void removeAllMemento() {
                contentList = new ArrayList<>();
        }

        public int getListSize() {
                return contentList.size();
        }

        public void addMemento(Memento memento) {
                contentList.add(memento);
        }

        public Memento removeMemento(int index) {
                if (index >= 0 && index < getListSize()) {
                        return contentList.remove(index);
                } else {
                        return null;
                }
        }

        private static class InnerClass {
                private static final CareTaker INSTANCE = new CareTaker();
        }
}
```

3.6 存储图形

存储图形主要是用了 Java Swing 的 JFileChooser 和 ImageIO 类下的 writeImage 方法，实际上就是将画布上的内容复制粘贴一份写入文件之中。具体方法实现如下：

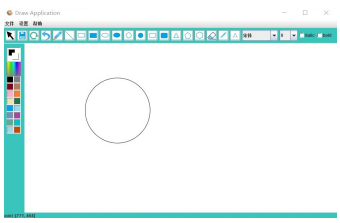
```
public void saveFile() {
        // swing自带的文件选择器
        JFileChooser fileChooser = getFileChooser();
        int result = fileChooser.showSaveDialog(MainFrame.getInstance());
        if (result == JFileChooser.CANCEL_OPTION) {
                // 如果取消了
                return;
        }

        File file = fileChooser.getSelectedFile();
        if (!file.getName().endsWith(fileChooser.getFileFilter().
                getDescription())) {
                // 判断是否后缀和选择的类型相同
                String fileName = file.getPath() +
                fileChooser.getFileFilter().getDescription();
                file = new File(fileName);
        }
        if (!file.canWrite()) {
            JOptionPane.showMessageDialog(fileChooser, "该文件不可写",
            "该文件不可写", JOptionPane.ERROR_MESSAGE);
        }
        if ("".equals(file.getName())) {
                JOptionPane.showMessageDialog(fileChooser, "无效的文件名",
                "无效的文件名", JOptionPane.ERROR_MESSAGE);
        }

        BufferedImage image = createImage(DrawPanel.getInstance());

        try {
                ImageIO.write(image, "png", file);
        } catch (IOException e) {
                e.printStackTrace();
        }
}
```

3.7 图形拖拽确定大小

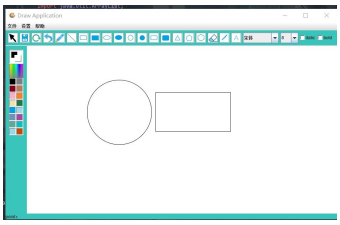
该点在 3.1 图形绘制已经有了介绍

4 实现效果

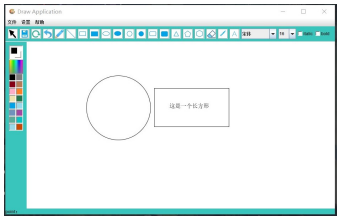
实现效果可以看同一级文件夹中的视频，下面以几张运行时截图作为实现效果的简要展示：



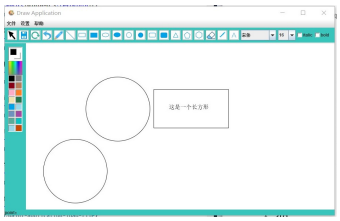
(a) 绘制圆形



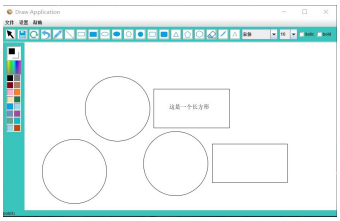
(b) 绘制方形



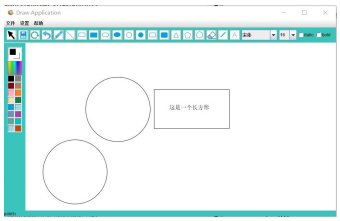
(c) 文字描述



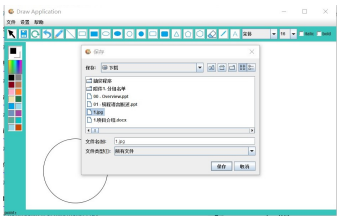
(d) 复制粘贴



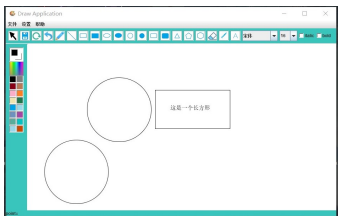
(e) 组合复制粘贴



(f) 撤销



(g) 保存文件



(h) 加载文件

该项目实现了基本要求中的以下几点：

1. 设计良好的图形用户界面，界面中要求至少有默认大小的三角形、方框、圆形、椭圆、连接线等五种元素可供用户选择后，绘制到画布上。
2. 允许用户添加文字描述
3. 单击可以选中图形，并允许对图形的拷贝复制
4. 多个图形可以组合，组合后的图形同样有拷贝复制功能
5. 支持撤销上一步操作的功能

在扩展要求中，该项目实现了以下的几点：

1. 支持图形（包括组合图形）的拖拽调整图形大小

2. 支持撤销多步的功能

3. 设计一种硬盘文件存储格式可以保存用户绘制的图形，并可以加载

5 小结

在本项目中，主要使用到了以下几种设计模式：

1. 简单工厂模式：用以创建绘制的内容

2. 单例模式：用以规范画图工具的各个面板和组件的唯一性

3. 原型模式：用以实现拷贝复制和粘贴

4. 备忘录模式：用以实现撤销

5. 适配器模式：用以更好的使用 Swing 中的某些接口

6. 观察者模式：用以存储一些共享的变量和内容

通过这个项目可以比较熟悉的掌握运用一些面向对象设计中的原则和方法，并运用各种设计模式来设计出符合面向对象设计思想的程序。