图形编辑器
详细设计说明书

[V1.0(版本号)]

作 者：况　聪

Email：　congkuang@gmail.com

[二零一一年六月二十四日]

目录

1. 引言

1.1 编写目的

该文档给出了整个图形编辑器的详细设计说明。首先，文档给出了软件的功能描述和整体结构设计。然后给出了各模块的描述，主要描述模块的功能，类图及用到的数据结构和算法，以及用到的设计模式。

1.2 背景

开发者：况聪 congkuang@gmail.com

1.3 定义

此图形编辑器使用 Java 语言开发，JDK1.6 上调试运行。
开发工具：Eclipse Helios;
运行环境：事先安装 JRE6，或者 JDK1.6。

1.4 参考资料

《设计模式》Erich Gamma,Richard Helm,Ralph Johnson,John Vlissides, 机械工业出版社 2009
《Java 核心技术：基础知识》 Cay S.Horstmann, Gary Cornell,机械工业出版社，2006
《Java 核心技术：高级特性》 Cay S.Horstmann, Gary Cornell,机械工业出版社，2006

2. 系统的结构

2.1 功能结构

2.1.1 软件功能结构图

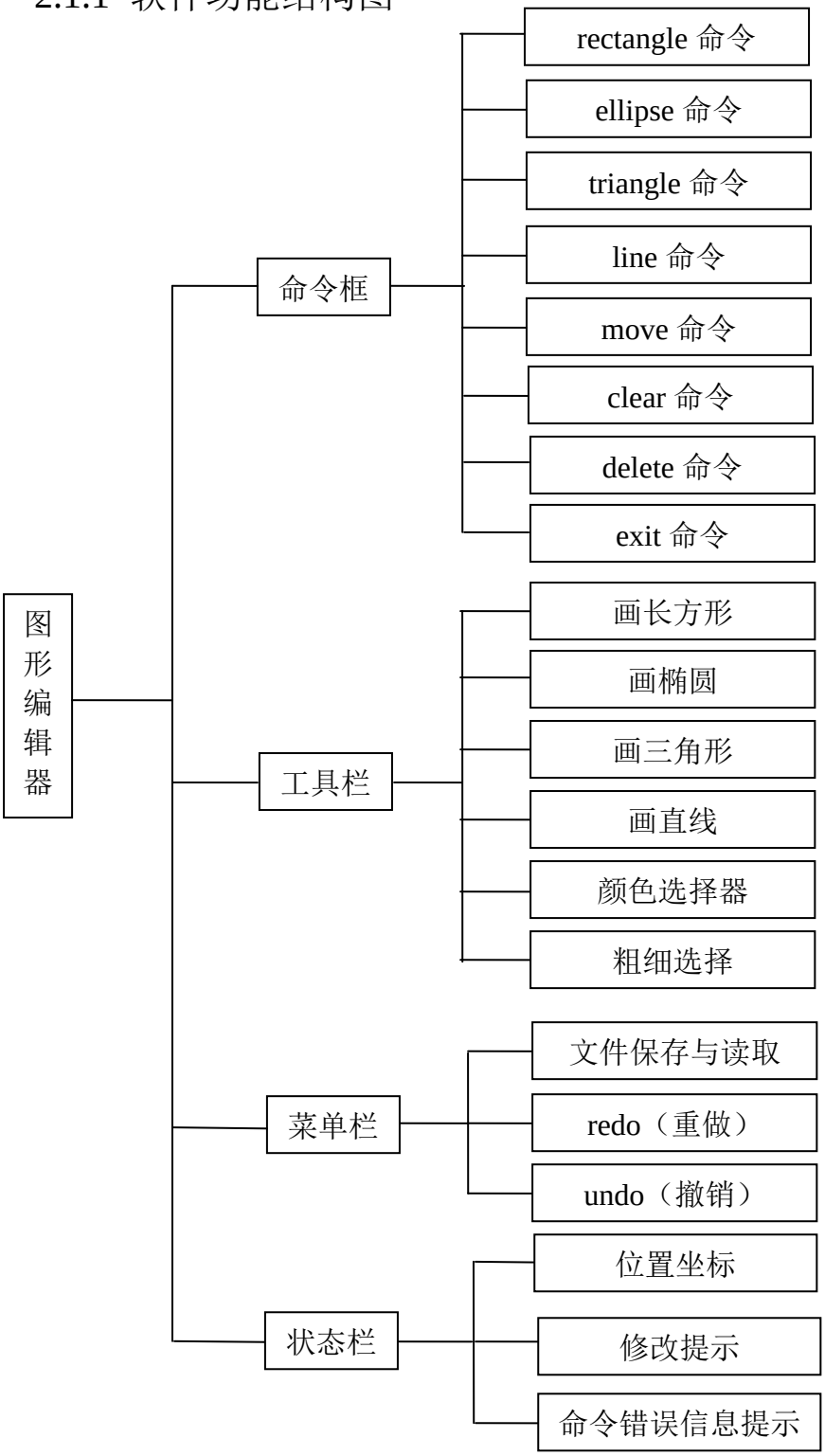


图 1 功能结构图

2.1.2 功能（操作）说明

1. 画长方形：单击工具栏长方形图标，点击画图区域某位置并拖动；
2. 画椭圆：单击工具栏椭圆形图标，点击画图区域某位置并拖动；
3. 画三角形：单击工具栏三角形图标，单击画图区域某三个位置后画出三角形；
4. 画直线：单击工具栏直线图标，点击画图区域某位置并拖动；
5. 选中图形：鼠标移动至画布上图形内部某位置时会变成十字形，此时单击图形表示选中此图形，选中的图形将高亮显示；单击空白将取消选中图形，但系统会记住最后一次被选中的图形，便于改变颜色和线条粗细时实时观察颜色和线条粗细的变化；
6. 着色：画图前单击工具栏颜色选择图标选择颜色，或者选中某图形后单击工具栏颜色选择图标，选择颜色后图形颜色会实时变化；
7. 线条粗细：画图前单击工具栏粗细选择框选择粗细，或者选中某图形后单击工具栏粗细选择框，选择粗细后图形线条粗细会实时变化；
8. 拖动：鼠标移动至画布上某图形内部某位置时会变成十字形，此时单击并拖动鼠标，图形会随着鼠标移动；
9. 删除：鼠标变成十字形时双击，此图形将被删除；
10. 撤销：点击菜单栏 Edit 项下的 undo 项或者使用 Ctrl-z 组合键将撤销操作；
11. 重做：点击菜单栏 Edit 项下的 redo 项或者使用 Ctrl-y 组合键将重做操作；
12. 保存：保存文件时文件后缀名没有限制；
13. 工具栏：可以将工具栏拖动至任意位置，以方便作图；
14. 命令窗口：支持命令输入（大小写敏感），对输入的错误命令有简单提示功能，命令参数均为正数，目前支持的命令有:

1）rectangle x y width height or rectangle x y width height wid
说明：画长方形，x，y 为左上角坐标，width，height 为长方形的长和宽，wid 为线条宽度。

2）ellipse x y width height or ellipse x y width height wid
说明：画椭圆，同画长方形的命令。

3）triangle x1 y1 x2 y2 x3 y3 or triangle x1 y1 x2 y2 x3 y3 wid
说明：画三角形，x1 y1 x2 y2 x3 y3 分别为三角形三个顶点的坐标，wid 为线条的宽度。

4）line x1 y1 x2 y2 or line x1 y1 x2 y2 wid
说明：画直线，x1 y1 x2 y2 分别为直线两端点的坐标，wid 为线条宽度。

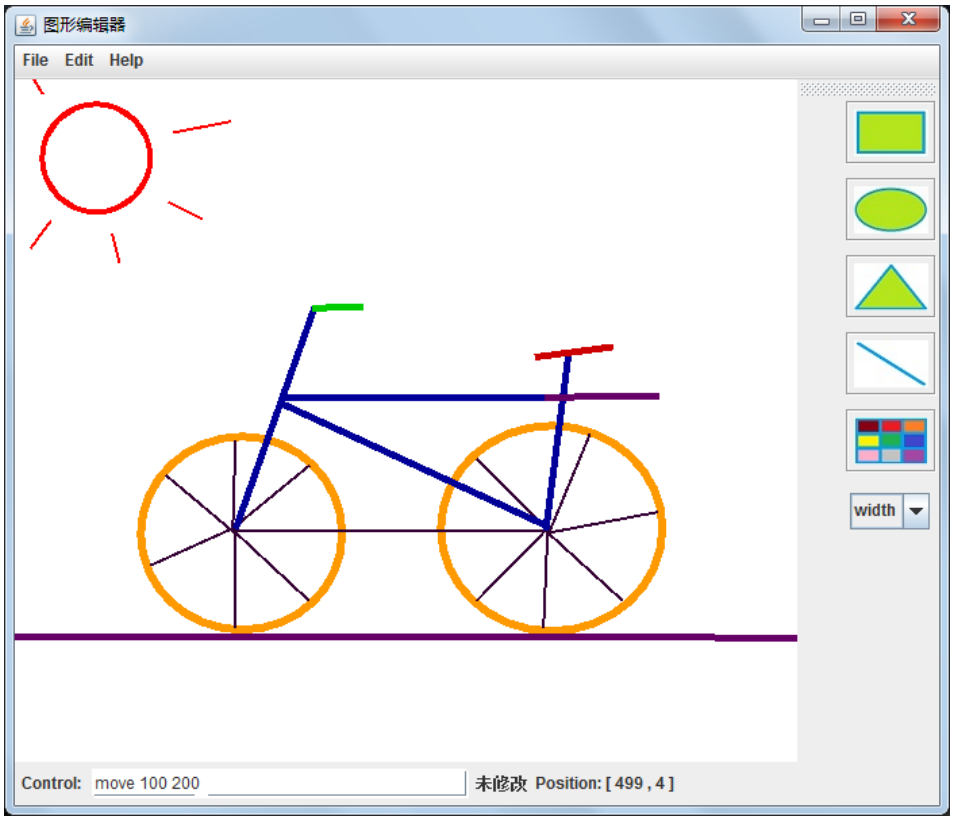
5）clear 说明：清空画布。

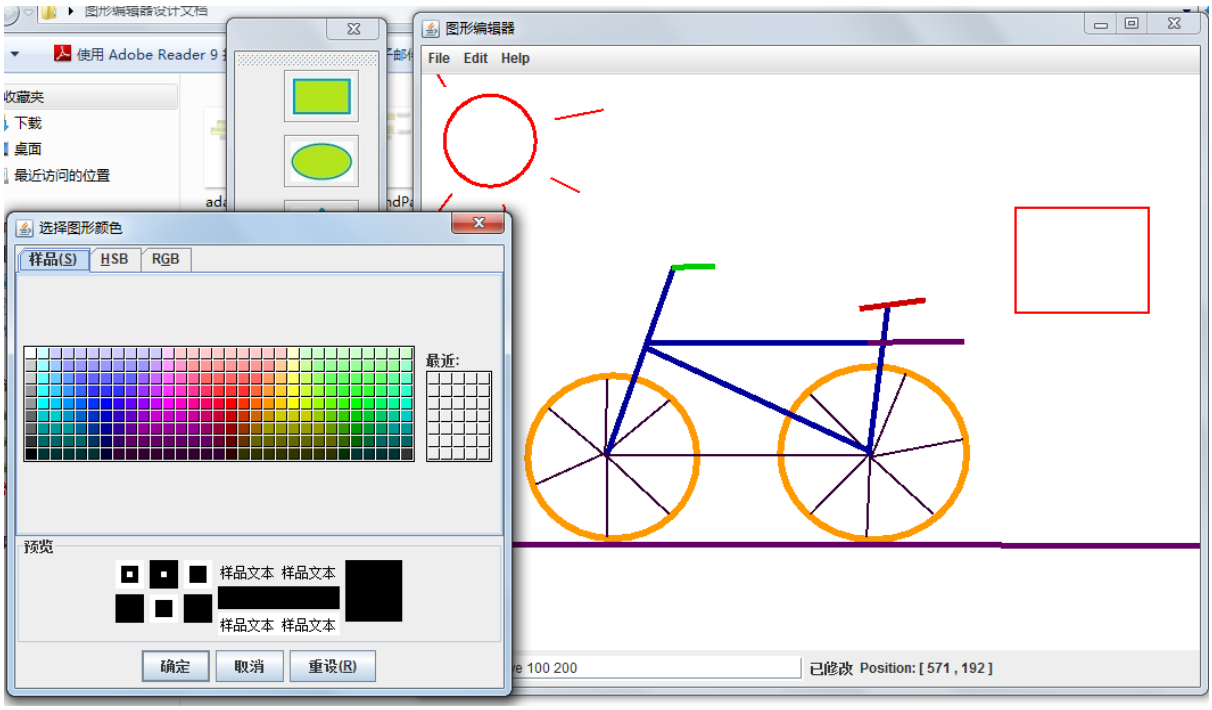
6）move x y
说明：移动选中图形到指定位置，x y 为选中图形要移动到的中心坐标。

7）delete
说明：删除选中的图形。

8）exit
说明：关闭图形编辑器。

2.1.3 图形编辑器截图


图形编辑器
File Edit Help
width
Control: move 100 200
未修改 Position: [499 , 4]


图形编辑器
File Edit Help
选择图形颜色
样品(S) HSB RGB
最近:
预览
样品文本 样品文本
样品文本 样品文本
确定 取消 重设(R)
已修改 Position: [571 , 192]

2.2 软件系统结构图

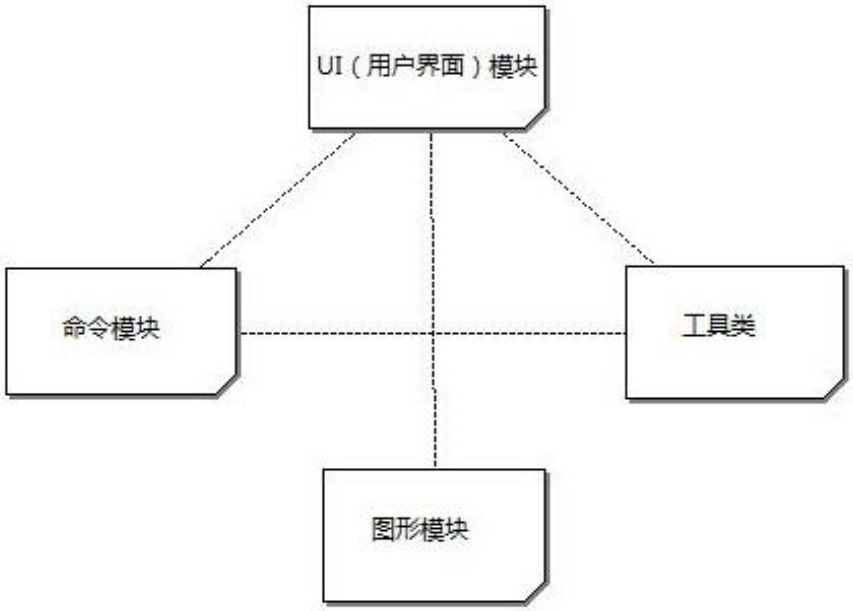


图 2 软件系统结构图

2.3 用到的设计模式

2.3.1 单例模式（Singleton）

PaintPanel 类是画板类，充当中介者的角色，因此只能有一个对象。设计了一个静态的自身对象：

```
public static PaintPanel pPanel = null;
```

一个私有的构造方法：

```
private PaintPanel(){ ... }
```

一个静态公有的获取自身对象的方法：

```
public static PaintPanel getPaintPanel(){
    if(pPanel == null)
        pPanel = new PaintPanel();
    return pPanel;
}
```

以此构成了单例模式，当其他类需要用到 PaintPanel 类对象时，通过 getPaintPanel（）方法获取。单例模式结构图如下：

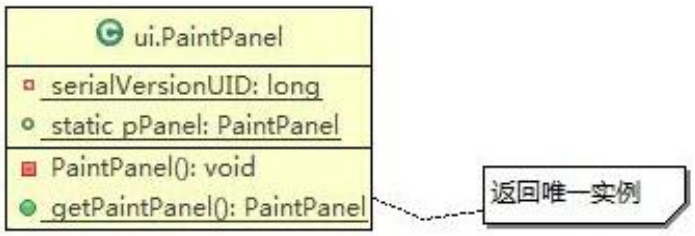


图 3 单例模式结构图

2.3.2 工厂方法（Factory Method）

CommandTools 类是命令的工具类，属性包含支持的命令集合 cmds，生成的命令对象的引用 cmd，以及命令出错信息 errorMsg。关键的方法是 check（）方法，这是一个工厂方法，解析并检查用户输入的命令字符串，并生成对应的命令对象，Client 通过 getCmd（）方法得到命令对象，通过命令对象的 execute（）方法执行对应命令（动态绑定）：

```
if(!cmdtool.check(command)){ //如果命令有错
    labelState.setText(cmdtool.getErrorMsg());
}
else{                                    //如果命令正确，则执行
    cmd = cmdtool.getCmd();
    cmd.execute();
}
```

工厂方法的结构图如下所示：

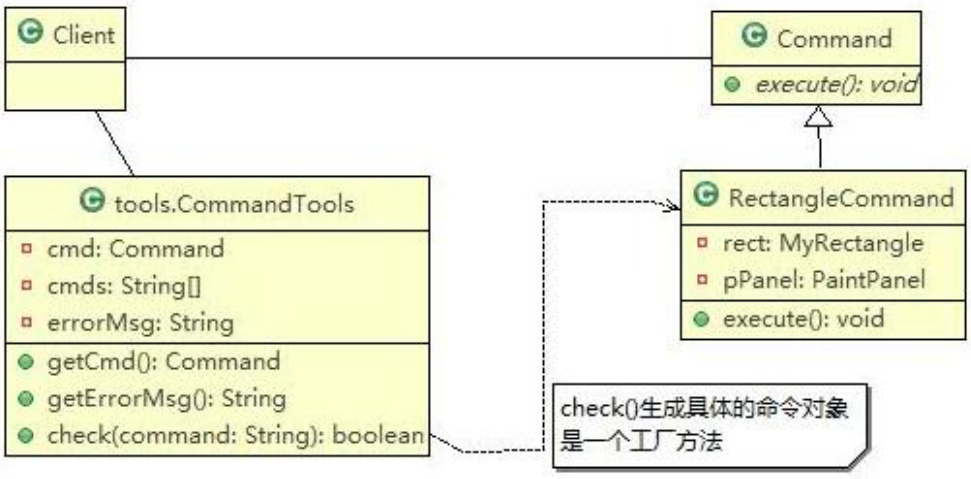


图 4 工厂方法结构图

如果添加新的命令，则需要修改 check（）方法，使其能生成新的命令对象。

2.3.3 命令模式（Command）

用户可以在命令窗口输入命令来执行一些操作，所有的命令类继承自一个抽象的父类 Command，它只有一个抽象方法 execute()，使在 Client 执行命令时可以使用多态性，从而执行具体的命令子类。每个命令子类实现自己的 execute()方法，可以在 execute()方法内部实现具体操作，也可以在 execute()方法内部调用其它方法。但在 Client 始终都只有一种调用格式：

```
cmd.execute();  //cmd 是 Command 型引用变量
```

所以，当需要加入新的命令时，只需让新命令继承自 Command 类，编写新命令类的 execute()方法，而无需修改 Client 的代码，Client 也无需知道命令的接收者或执行的步骤，从而把 Client 与具体命令解耦，增加了可扩展性。

本应用使用的命令模式结构图如下：

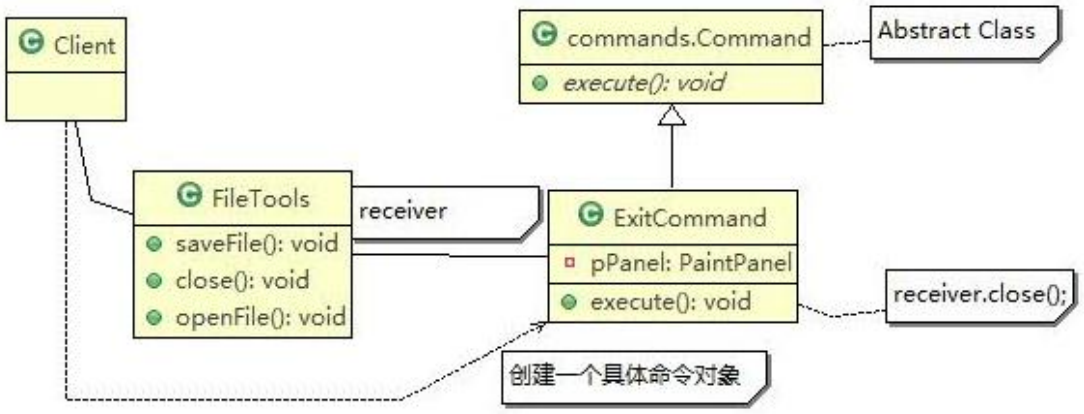


图 5 命令模式结构图

2.3.4 中介者模式（Mediator）

PaintPanel 类作为中介者，所以它的属性非常多。其它对象与它交互并修改或读取其属性值，从而到达其它对象间互相通信的目的。中介者使各对象不需要显示地相互引用，从而使其耦合松散，而且可以独立地改变它们之间的交互。

例如，状态信息，当对当前画板有修改动作时，包括添加，删除，移动，改变颜色，粗细等，状态信息会显示“已修改”，发出这些动作的对象可能是工具栏，可能是命令窗口，可能是鼠标。而一旦发现修改动作，当新建文件，打开新文件或者关闭文件时系统都要提示用户是否保存当前文件。如果没有中介者，让各个对象自己去通信，这种对象间的相互引用将是十分复杂的。把状态信息作为属性放在中介者（PaintPanel）里时，各对象只需读取、修改中介者中的状态信息，就可以达到相同的目的。

下图是中介者模式结构图：

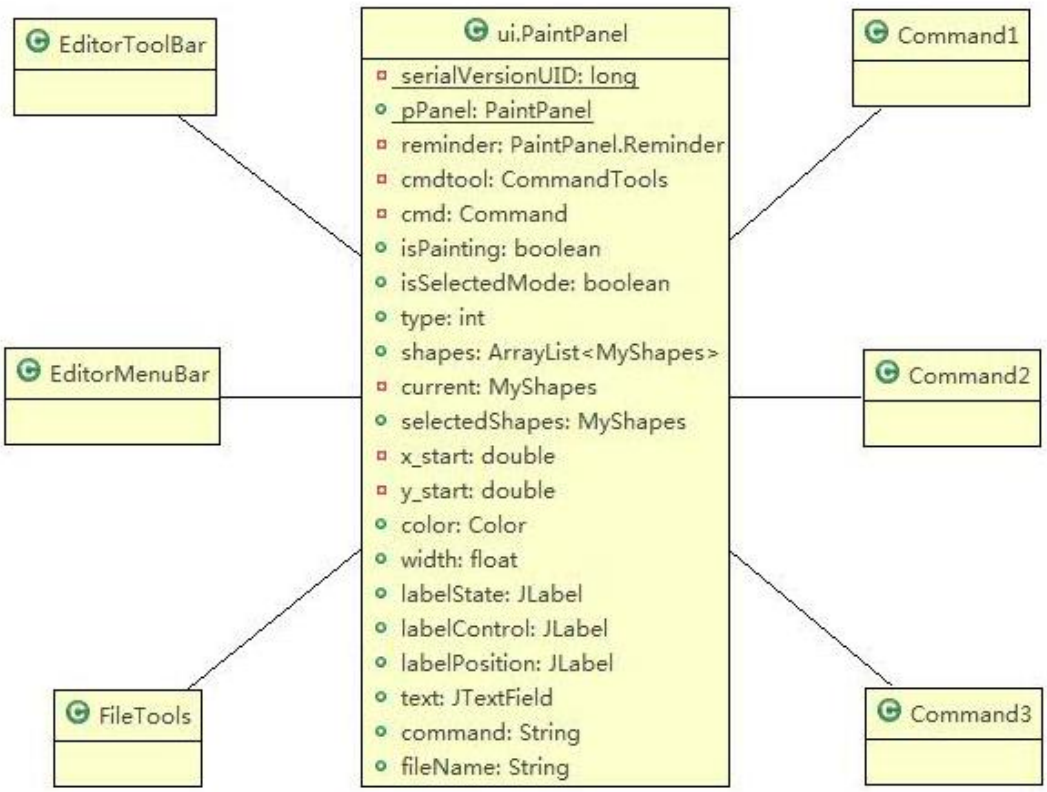


图 6 中介者模式结构图

2.3.5 适配器模式（Adapter）

三角形类画自身的 draw()方法需要一次画三条线段，以便画出一个三角形。而画直线的方法直接调用 Graphics2D 类的 drawLine()方法，所以 MyTriangle 对象作为一个 Adapter，Graphics2D 作为 Adaptee。

结构图如下所示：

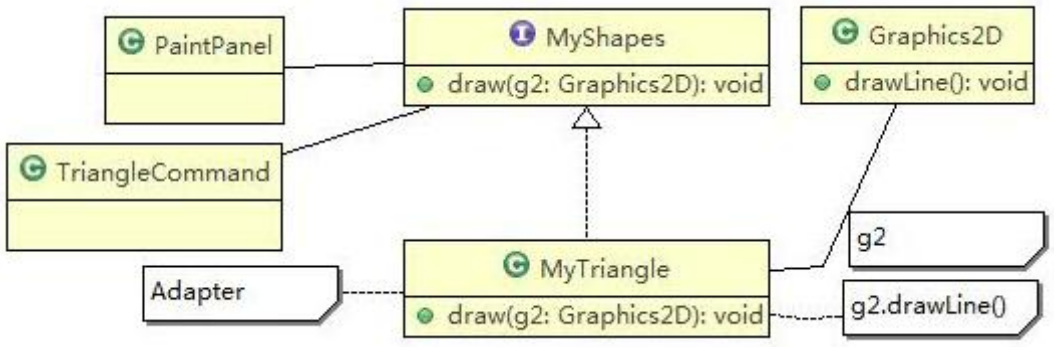


图 7 适配器模式结构图

2.3.6 外观模式（Facade）

EditorMenuBar,EditorToolBar 两个类都使用了外观模式，这两个类为菜单栏和工具栏的构建提供了统一和简单的接口，并封装了变化，隐藏了菜单栏和工具栏的具体实现。在 Client 中，实例化菜单栏和工具栏的代码始终不变：

```
EditorMenuBar menuBar = new EditorMenuBar();
EditorToolBar toolBar = new EditorToolBar();
```

这样使 Client 与菜单栏工具栏子系统之间形成一种松耦合关系。当需要更改菜单栏和工具栏时，在 EditorMenuBar,EditorToolBar 类中修改即可。通过类中的公有方法 getToolBar()可以获得 JToolBar 组件对象。

工具栏外观模式结构如下图所示：

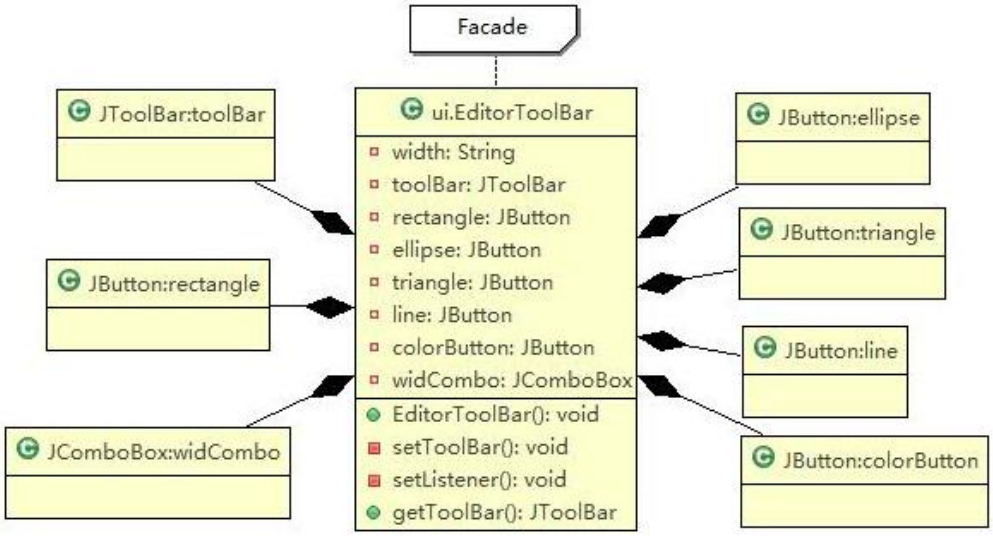


图 8 工具栏的外观模式结构图

3. 模块设计

3.1 UI（用户界面）模块

3.1.1 功能与组成

绘制图形用户界面，为各组件添加动作监听器。实现画板上的鼠标操作（画图，移动，删除，选中）和键盘操作。实现用于记录画板状态的内部类 Reminder，用于撤销与重做。

由以下类组成：

EditorMain：主类；

EditorFrame：框架类；

EditorPanel：最底层的面板；

EditorMenuBar：菜单栏；

EditorToolBar：工具栏；

PaintPanel：画板；

Reminder：用于备份的内部类；

3.1.2 类图

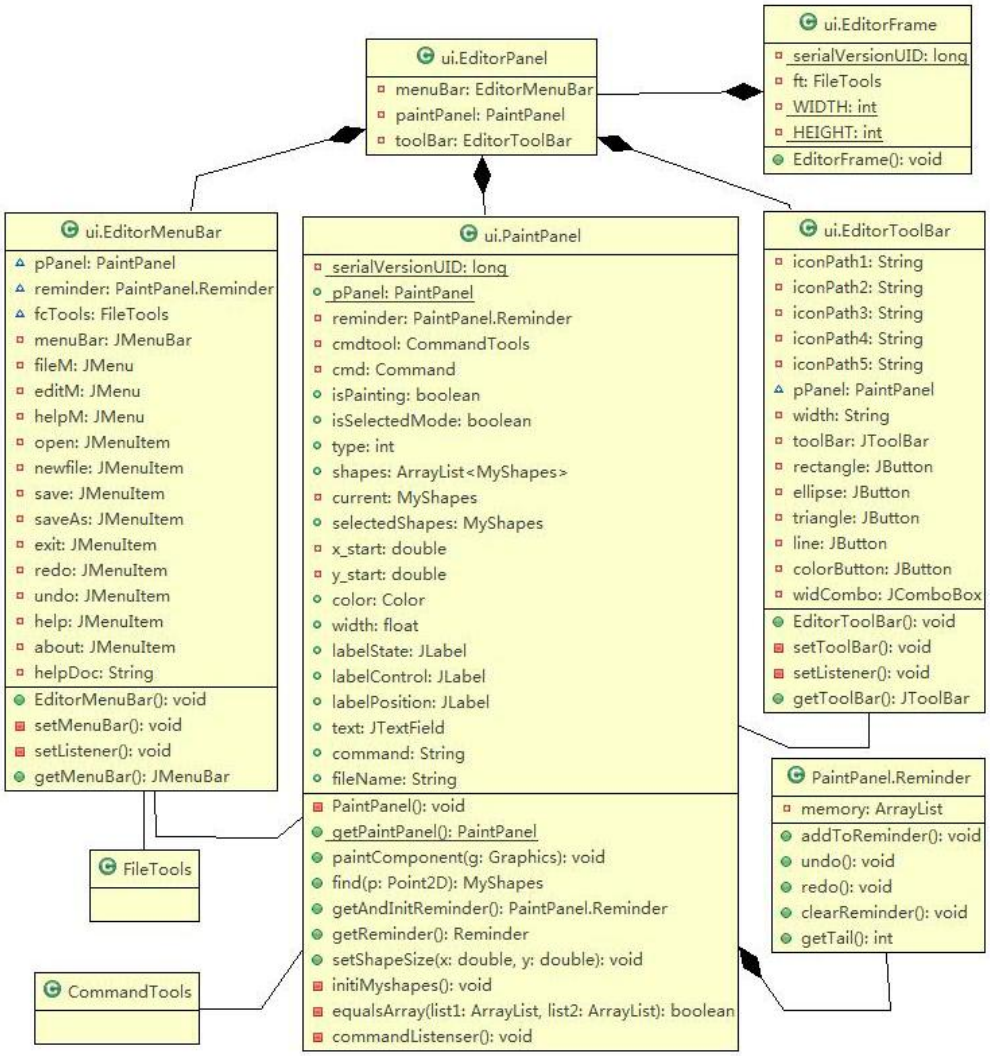


图 9 UI 模块类图

3.1.3 数据结构与算法

1. 图形的存储

画板上的所有图形存储在 ArrayList<MyShapes> shapes 中，ArrayList 是一个可以自动扩容的数组类。

2. 图形的备份

备份图形用于撤销和重做操作。使用 PaintPanel 的内部类 Reminder 实现，图形备份存储在 ArrayList<ArrayList<MyShapes>> memory 中，memory 的一个元素是整个 shapes。为了节约内存只存储当前操作之前的 10 次操作，其余的操作将会被覆盖。

数据结构如下：

```
private int tail = -1,  //指示数组的尾巴，因为这是一个循环数组
            index = 0,  //当前重做或者回滚到的位置
            memorySize = 0;  //memory中元素的最大下标
private int memoryLen = 10;  //备忘操作的最大长度
```

调用addToReminder()方法时将把当前所有图形备份：

```
public void addToReminder(){
memorySize = ++tail;
memorySize = memorySize >= memoryLen ? memoryLen - 1 : memorySize;
tail = tail % memoryLen;
if(tail < memory.size())
    memory.remove(tail); //先清空相应位置的内容
memory.add(tail, (ArrayList<MyShapes>) shapes.clone());
index = tail;
}
```

undo 和 redo 操作就在 memory 数组上移动取值，以达到撤销和重做的效果。

3. 菜单栏上 redo 操作项的禁用

当前操作是最新操作时禁止 redo 操作。分别在 undo 和 redo 事件中加以控制，控制代码为：

```
redo.setEnabled(reminder.getIndex() < reminder.getTail() ||
                reminder.getTail() < reminder.getMemorySize());
```

3.2 shapes（图形）模块

3.2.1 功能与组成

实现自定义图形类，类中封装了图形属性如颜色和宽度，以及画图，移动，高亮显示等方法。每个图形的这些方法实现都不一样，但它们有一个共同的图形接口 MyShapes，从而使扩展自定义图形非常方便。

3.2.2 类图

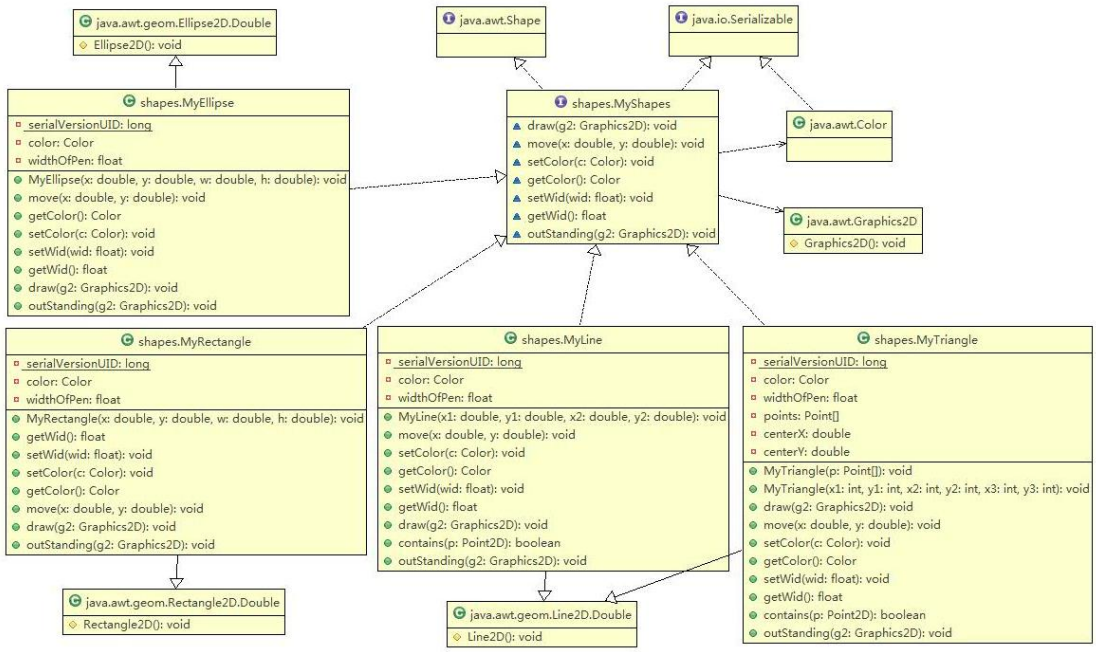


图 10 图形模块类图

3.2.3 数据结构与算法

现有图形类 5 个，MyShapes,MyRectangle，MyEllipse，MyTriangle，MyLine。

1. 三角形移动算法

以上二维图形的移动都为平移，move()方法需要两个参数，即，移动的目的坐标 x 和 y。

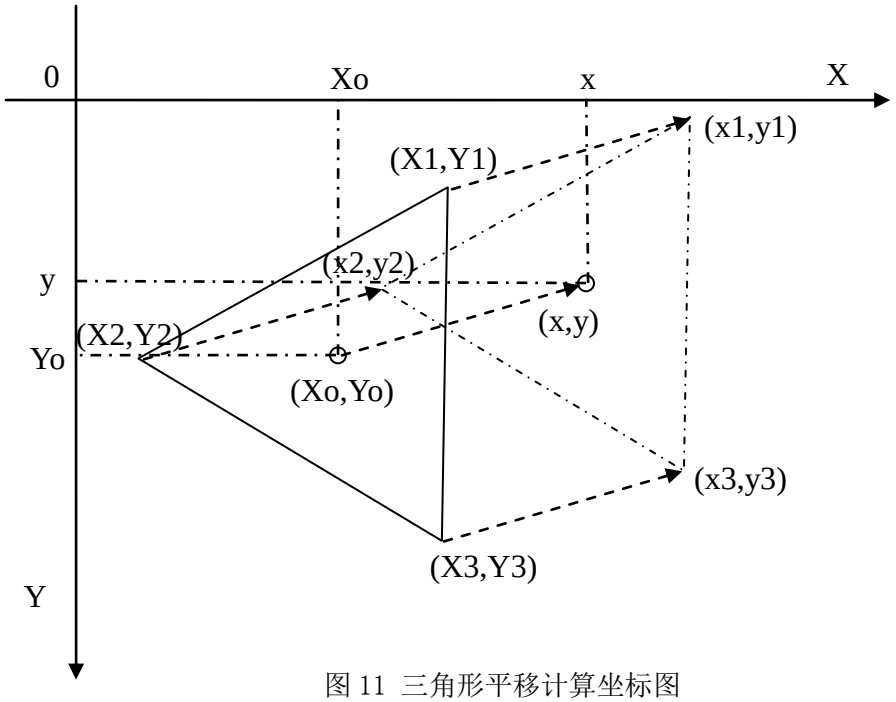


图 11 三角形平移计算坐标图

原始三角形位置为(X1,Y1)(X2,Y2)(X3,Y3)，（Xo,Yo）是中心，

```
Xo = ( X1 + X2 + X3 ) / 3 ,
Yo = ( Y1 + Y2 + Y3 ) / 3;
```

作为参数传入 move()方法的坐标 x 和 y 是新三角形的中心，所以，

```
x1 = X1 + x - Xo ,
x2 = X2 + x - Xo ,
x3 = X3 + x - Xo ;
y1 = Y1 + y - Yo ,
y2 = Y2 + y - Yo ,
y3 = Y3 + y - Yo ;
```

新三角形的坐标为(x1,y1)(x2,y2)(x3,y3)。对于任意多边形平移算法是一样的。

2. 重写 contains()方法

Line2D 原有的 contains()方法始终返回 null，所以使用原有的 contains()方法始终无法捕获到直线和三角形，因此直线类和三角形类需要覆盖 contains()方法。

重写的三角形的 contains()方法：

```
public boolean contains(Point2D p) {
    double dist = p.distance(centerX, centerY);
```

```
        if(dist < 20)    //点到中心的距离小于20个像素时，看作包含
            return true;
        return false;
    }
```

重写的直线的 contains()方法：

```
    public boolean contains(Point2D p) {
        if(ptSegDist(p) < 5) //到直线的距离小于5个像素时，看作包含
            return true;
        return false;
    }
```

3.3 commands（命令）模块

3.3.1 功能与组成

命令模块提供了各种命令的集合。所有命令类均继承自抽象类 Command，Command 类只有一个抽象方法：

```
public abstract void execute();
```

当 Client 调用一个命令时，调用语句始终不变：

```
cmd.execute();  //cmd 是 Command 型引用变量
```

因为 cmd 指向的命令对象不同，利用多态性将自动执行具体的命令。当需要扩展新的命令时，直接继承 Command 类即可，新命令的 execute()方法或使用 Adapter 模式，或自行定义方法过程。

3.3.2 类图

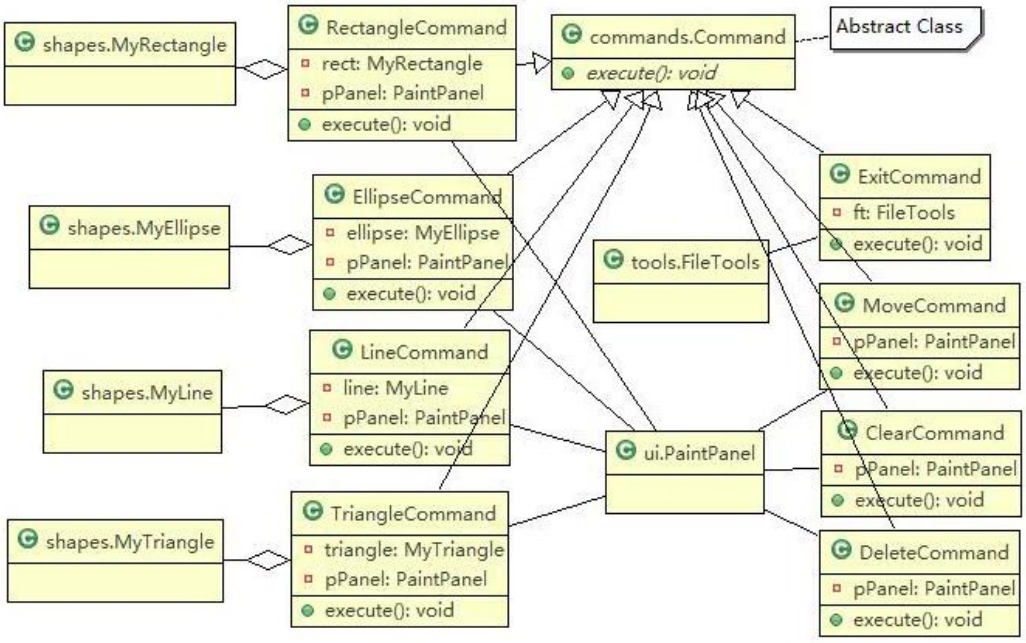


图 12 命令模块的类图

3.3.3 数据结构与算法

1. 删除（delete）命令

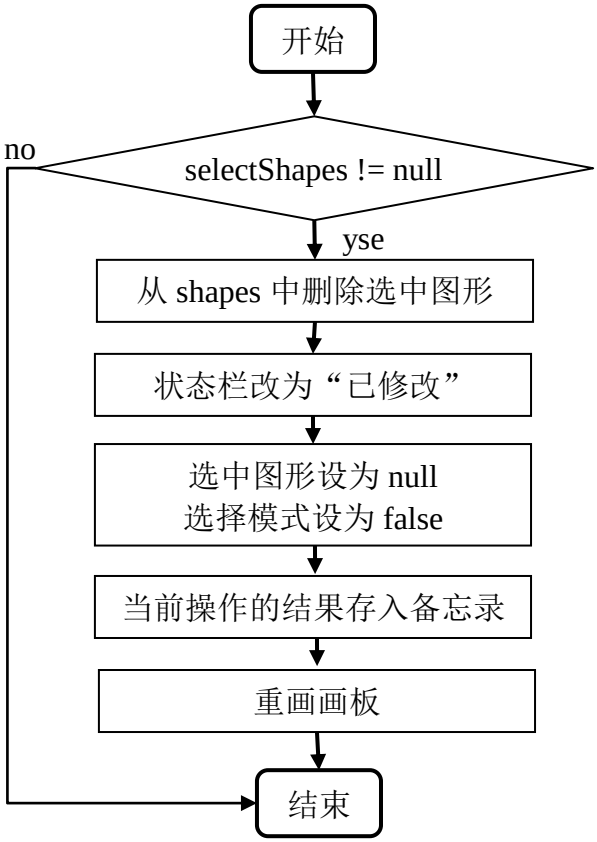


图 13 删除命令算法流程图

2. 清除（clear）命令

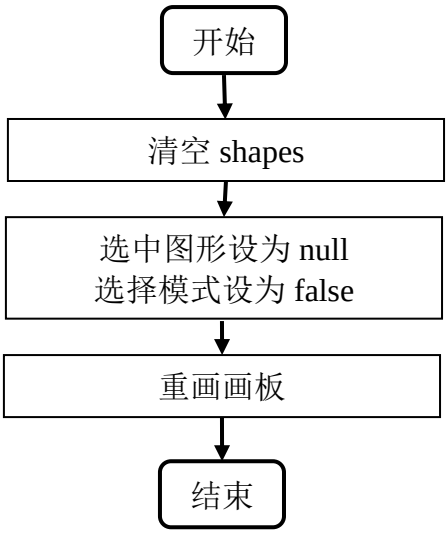


图 14 清除命令的算法流程图

3.4 tools（工具类）模块

3.4.1 功能与组成

提供了其他类用到的一些方法的集合，目的是复用这些方法，而不会导致对象间复杂的引用关系。同时也分离了类的职责。本模块由两个类组成，

CommandTools：提供命令相关的一些方法；

FileTools：提供文件操作相关的一些方法。

3.4.2 类图

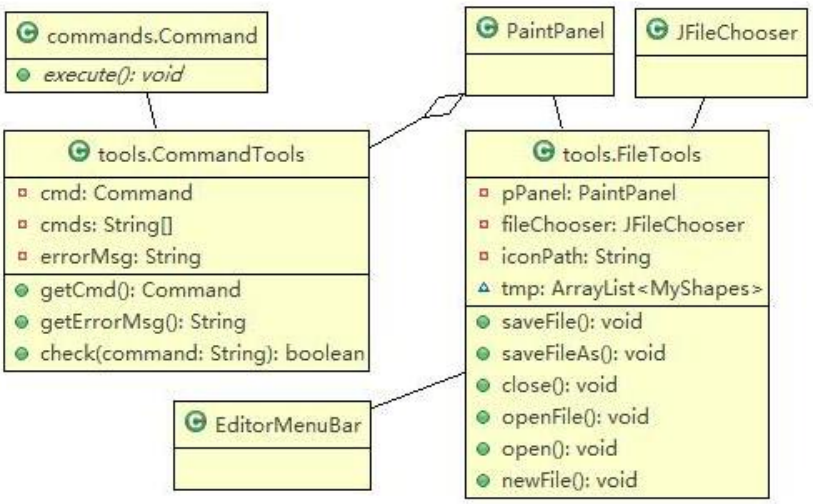


图 15 工具模块的类图

3.4.3 数据结构与算法

1. 命令格式检查

命令格式的检查使用函数 check(),返回值为布尔型。这是一个工厂方法，如果命令格式正确，则返回 true，同时会根据 Client 输入的命令生成具体的命令对象。

命令的参数必须是正实数，用如下的正则表达式检查：

```
[0-9]+[.]?[0-9]+|[0-9]
```

同时，对参数的个数和取值范围也有限制。如果命令格式出现错误，则返回 false，对应的错误信息存入：

```
private String errorMsg ;
```

此信息传递到 Client 并显示在状态栏上，以便用户修改命令。

2. 对象的输入输出（持久化）

图形编辑器编辑完成的图像需要以文件的形式存储到硬盘上，那么下次使用图形编辑器就可以打开以前的图形文件继续编辑。

图形文件的保存使用了对象的持久化技术，保存时把当前的 shapes（保存当前所有图形的 ArrayList）用对象流写入文件中，Java 的对象持久化技术保证了嵌套对象持久化的正确性。主要代码为：

```
//建立对象输出流
ObjectOutputStream objOut=new ObjectOutputStream(new
```

```
                        FileOutputStream(file));
    //把对象写入文件
    objOut.writeObject(pPanel.shapes);
    objOut.flush();
    objOut.close();
```

每个需要持久化的对象分配了一个显式的各不相同的序列化 ID：

```
    private static final long serialVersionUID =
                                -6465778200383354347L;
```

打开文件时，是一个反序列化过程，主要代码如下：

```
    // 创建文件输入流，读取对象输入流
    ObjectInputStream objIn = new ObjectInputStream(
        New FileInputStream(fileChooser.getSelectedFile()));
    pPanel.shapes.clear(); // 清空当前的图形数组
    tmp = (ArrayList<MyShapes>) objIn.readObject();
    pPanel.shapes.addAll(tmp);
```

这样可以达到图形文件保存和读取的目的。