JAD mini CAD

xxx xxxxxxxxxx

总体设计

- 遵从MVC模式（model、view、control）
- model模块
  - 主要由*Model类*实现
  - 管理图形数据，包括：
    - 一个存放已经放置的所有图形的容器（放置指的是图形已经绘制完毕）
    - 一个Shape对象，保存正在绘制中的图形
  - 提供接口供control模块修改model模块中存储的数据
- control模块
  - 主要由*CanvasPanel类*和*ToolPanel类*实现
  - 这两个类首先绘制了基本的图形用户界面
  - 这两个类添加相应的响应函数，实现对用户输入的响应
  - 在响应函数中，调用model模块的接口函数修改model模块中存放的数据
- view模块
  - 主要由CanvasPanel类中的*paintComponent*函数实现
  - 在该函数中，调用model模块的接口函数从而获取待绘制的图形信息，并绘制
  - 由于每次control修改了model模块的数据后都需要重新绘制，因此在响应函数修改完model模块中的数据后便直接调用view模块，实现重绘

实现功能

- 实现的功能具体如下：
  - 简单的绘图工具，能放置直线、矩形、圆和文字
  - 能够选中图形，修改参数（如颜色、粗细、大小）
    - 选中时图形会变为红色
    - 绘图前选好颜色，便可以绘制相应颜色的图形
    - 选中图形后，可以点击调色板改变选中图形的颜色（图形进入非选中状态后才会变色）
    - 选中图形后，可以按"[", "]"调整粗细（无论是否按下shift都可）
    - 选中图形后，可以按"-", "+"调整大小（无论是否按下shift都可）
    - 选中图形后，可以按R键删除
    - 可以按E键删除所有图形
  - 图形选中后可以拖动
  - 可以保存和恢复
- 最终效果如下：

模块实现

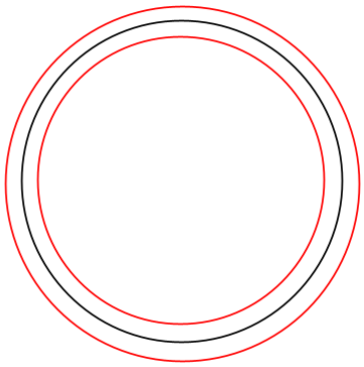
- miniCAD共三个源代码文件，分别为

- Main.java, Model.java, Shape.java

Shape类及其子类

- 在Shape.java中实现
- Shape类为抽象类，实现了它的四个子类：
- Line, Rectangle, Circle, Text
- 首先Shape类实现了Serializable接口，使得文件保存和读取时可序列化
- 对于所有图形，Shape（父类）中存储了它们的所有数据
  - 除了Text子类还需要自己存储它的字符串、字体数据
- Shape父类存储了：
  - ```
    protected int sx, sy, fx, fy;
    ```
    - 对于这四种图形，都只需要两个点就可以确定其位置
  - ```
    protected Color color = Color.black;
    protected float strokeValue = 1.0f;
    ```
    - 存储了颜色和粗细数据
  - ```
    public boolean isSelected = false;
    ```
    - 记录当前图形是否被选中
- Shape父类实现/声明了：（由于大部分函数直接看函数名就可以知道含义，就不详细一一列出）
  - ```
    public void increase();
    public void decrease();
    ```
    - 增大/减小图形的大小
    - 除了Line子类外，其他图形的大小控制都只需要控制对角线的两个点简单移动一下即可
      - 线段的两个端点不能简单的加减坐标来控制大小，需要考虑斜率
      - 因此在Line子类中需要覆盖这两个函数
    - 需要控制图形缩小时不会反转或消失
  - ```
    public abstract void draw(Graphics2D g2d);
    public abstract boolean testSelected(int x, int y);
    ```
    - 对于每个子类，都需要实现draw()，从而实现图形的绘制
      - 在view模块中会调用每个图形的draw方法
      - 对于画圆和画文本，其画图函数的实现需要分类讨论一下，使得绘制出来的图形其固定点都是起始按下的坐标点，而不是最终释放时候的坐标点
    - 对于每个子类，都需要实现testSelected()，从而判断某个鼠标的坐标点是否能选中该图形
      - 对于直线，用点到直线距离来判定
        - 需要注意倘若坐标点和线段两端的两点形成钝角三角形，则是未选中（不能只用点到直线距离）
      - 对于矩形，在下图红色框线范围内算选中：
    ```

- 对于圆形，用点到圆心距离来判断，如图中环状区域：



- 对于文本，需要用到字体相关的函数来判断字体区域：

```
FontMetrics fm;
public boolean testSelected(int x, int y) {
    if(fy > sy) {
        return (x >= sx) && (x <= sx + fm.stringWidth(s))
&&
               (y <= fy) && (y >= fy - fm.getHeight());
    } else {
        return (x >= sx) && (x <= sx + fm.stringWidth(s))
&&
               (y >= fy) && (y <= fy + fm.getHeight());
    }
}
```

Model类

- Model类负责存储和操作数据
-
```
private static List<Shape> shapeList = new ArrayList<>();
public static Shape staticShape = null;
```

  - shapeList存储所有已经放置的图形，staticShape对应当前正在画的图形
- 对shapeList的操作都通过接口来做：

```
public static List<Shape> getShapeList();
public static void setShapeList(List<Shape> ls);
public static void addShape(Shape s);
public static void removeShape(Shape s);
// 判断该坐标是否选中了shapeList中的某个图形
public static boolean testSelected(int x, int y);
// 清除所有图形的选中状态为false
public static void clearSelected();
// 调整被选中图形的位置
```

```
public static void dragSelectedShape(int dx, int dy);
// 调整被选中图形的颜色
public static void changeColorOfSelectedShape(Color colorType);
// 放大被选中图形
public static void increaseSelectedShape();
// 删除被选中图形
public static void deleteSelectedShape();
// 加粗被选中图形
public static void increaseStrokeOfSelectedShape();
```

  - 其中deleteSelectedShape()，需要对shapeList进行修改，但同时view模块又需要遍历shapeList进行绘制，因此会有线程同步问题
    - 在deleteSelectedShape()中不用迭代器的方式遍历
- 对staticShape，主要有如下操作：

```
// 得到staticShape对象
public static Shape getStaticShape();
// 根据control模块传入的数据，创建staticShape对象
public static void newStaticShape(ShapeType shapeType, Color colorType, String s);
// 当用户鼠标释放后，说明当前正在画的图形已经放置
// 此时staticShape应当被添加至shapeList列表中
public static void releaseStaticShape();
// 倘若control模块中认为正在画的图形不应加入列表，则将staticShape设为null
// 具体流程和原因在control模块实现中讨论
public static void abandonStaticShape();
```

CanvasPanel类

- JPanel类的子类
- 拥有成员变量：

```
static boolean isSelected = false;
Point s,f;
```

  - isSelected记录当前整个画布是否有图形被选中
  - 点s, f是为了帮助记录在鼠标拖动时的临时坐标数据
- 覆盖JPanel的paintComponent(Graphics g)函数

```
public void paintComponent(Graphics g) {
    super.paintComponent(g);

    List<Shape> shapeList = Model.getShapeList();
    for(Shape s : shapeList) {
        if(s != null && s.drawable())
            s.draw((Graphics2D)g);
    }
    System.out.println("paintComponent");
    Shape staticShape = Model.getStaticShape();
    if(staticShape != null && staticShape.drawable())
        staticShape.draw((Graphics2D)g);
}
```

  - 该函数扮演了view模块的角色，负责从model模块中取出数据并绘制

    - 需要从shapeList中取出数据，同时也需要绘制出*"正在画的那个图形"*
- 添加MouseMotionListener事件响应
    - 在该事件中，对鼠标拖动进行处理

```
public void mouseDragged(MouseEvent e) {
    requestFocus();
    if(isSelected) {
        s.x = f.x;
        s.y = f.y;
        f.x = e.getX();
        f.y = e.getY();
        Model.dragSelectedShape(f.x - s.x, f.y - s.y);
    } else {
        System.out.println("mouseDragged");
        Shape staticShape = Model.getStaticShape();
        if(staticShape != null) {
            staticShape.setFPoint(e.getX(), e.getY());
        }
    }
    repaint();
}
```

        - 如果当前有图形被选中，则调用Model类的函数，移动被选中的图形
        - 如果当前没有图形被选中，则是在画*"正在画的那个图形"*，需要改变Model类中的staticShape
            - 需要修改其F点（每个图形有S、F两个点，确定两个端点）
- 添加MouseListener事件响应
    - 对鼠标按下，鼠标点击，鼠标释放三个事件进行处理
    - 鼠标按下分为两种情况：

```
public void mousePressed(MouseEvent e) {
    requestFocus();
    System.out.println("mousePressed");
    if(isSelected) {
        int x, y;
        x = e.getX();
        y = e.getY();
        f.x = x;
        f.y = y;
        isSelected = Model.testSelected(x, y);
        System.out.println(x+" "+y);
        if(!isSelected) {
            Model.clearSelected();
        }
    } else {
        Model.newStaticShape(ToolPanel.shapeType,
ToolPanel.colorType, ToolPanel.s);
        Model.getStaticShape().setSPoint(e.getX(), e.getY());
        Model.getStaticShape().setFPoint(e.getX(), e.getY());
    }
    repaint();
}
```

        - 如果当前有图形被选中，则判断鼠标按下的地方是否能够选中另一个图形
            - 若能，则选中另一个图形
            - 若不能，则清除全局的选中状态和所有图形的选中状态
        - 如果当前没有图形被选中，则证明用户即将要画*"正在画的那个图形"*
            - 需要调用Model类函数，传给它当前选中的图形类型和颜色，新建staticShape，并初始化其S、F点
    - 鼠标点击，只有一种情况，和鼠标按下的情况一一致：

```
public void mouseClicked(MouseEvent e) {
    requestFocus();
    System.out.println("mouseClicked");
    int x, y;
    x = e.getX();
    y = e.getY();
    isSelected = Model.testSelected(x, y);
    System.out.println(isSelected);
    if(!isSelected) {
        Model.clearSelected();
    }
    repaint();
}
```

    - 鼠标释放，说明用户画完了*"正在画的那个图形"*，需要将其存入Model类的shapeList中

```
public void mouseReleased(MouseEvent e) {
    requestFocus();
    if(!isSelected) {
        System.out.println("mouseReleased");
        Shape s = Model.getStaticShape();
        double distance = 0;
        if(s != null)
            distance = Math.pow(s.sx-s.fx, 2) + Math.pow(s.sy-s.fy,
2);
        if(distance >= 20) {
            System.out.println("Adding a new Shape!");
            Model.releaseStaticShape();
        } else {
            System.out.println("Abandon the drawing Shape!");
            Model.abandonStaticShape();
        }
    }
    repaint();
}
```

        - **优化考虑**：由于有些时候，用户可能不小心按下鼠标并拖动了一小段肉眼几乎不可见的距离，按照原本的逻辑这个图形也会被加入到shapeList中。然而用户大概率是不需要这个图形的，且由于图形太小很难选中，难以选中后删除。因此存入之前，判断它的大小，如果太小，就丢弃掉*"正在画的那个图形"*
- 添加KeyListener事件响应
    - 根据按下的key，执行Model类的相应函数，实现对数据的操作

```
public void keyTyped(KeyEvent e) {
    requestFocus();
    System.out.println("keyTyped1");
```

```
    char key = e.getKeyChar();
    if(isSelected || key == 'e') {
        System.out.println("keyTyped2");
        switch(key) {
            case '=': case '+':
                Model.increaseSelectedShape();
                break;
            case '-': case '_':
                Model.decreaseSelectedShape();
                break;
            case 'r':
                Model.deleteSelectedShape();
                break;
            case 'e':
                Model.removeAllShape();
                break;
            case ']': case '}':
                Model.increaseStrokeOfSelectedShape();
                break;
            case '[': case '{':
                Model.decreaseStrokeOfSelectedShape();
                break;
            default:
                break;
        }
    }
    repaint();
}
```

ToolPanel类

- JPanel的子类
- 实现用户界面工具栏的绘制
- 添加相应的按钮响应函数，并调用Model类的函数对数据进行操作
- 由于ToolPanel中的响应函数逻辑比较简单，和CanvasPanel类似，不再一一说明

Main类

- JFrame的子类
- ```
  ToolPanel tp = new ToolPanel();
  CanvasPanel cp = new CanvasPanel();
  ```
  - 创建tp, cp两个对象，并加入到Main (JFrame子类) 中
- 添加菜单，实现文件打开和保存的入口

```
// create menu
JMenuBar menuBar = new JMenuBar();
setJMenuBar(menuBar);
JMenu fileMenu = new JMenu("File");
menuBar.add(fileMenu);
JMenuItem openItem = new JMenuItem("Open");
fileMenu.add(openItem);
JMenuItem saveItem = new JMenuItem("Save");
fileMenu.add(saveItem);
```
```

- 添加菜单按键的响应函数
  - 对于保存：

```
saveItem.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        System.out.println("here in save.");
        try {
            FileDialog fd = createFileDialog("Save");
            fd.setVisible(true);
            FileOutputStream fos;
            ObjectOutputStream oos;
            if(fd.getDirectory() != null && fd.getFile() != null) {
                fos = new FileOutputStream(fd.getDirectory() +
 fd.getFile());
                oos = new ObjectOutputStream(fos);
                oos.writeObject(Model.getShapeList());
                fos.flush();
            }
        } catch(Exception exception) {
            System.out.println("exception in save.");
            exception.printStackTrace();
        }
    }
});
```

    - 首先创建文件对话框，得到文件目录和路径后，创建出fos对象
    - 根据fos对象创建oos对象
    - 调用oos对象的writeObject方法，将Model类中的shapeList存入文件中，从而完成数据的存储
    - 需要Shape类实现序列化
  - 对于打开：

```
openItem.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        System.out.println("here in open.");
        try {
            FileDialog fd = createFileDialog("Open");
            fd.setVisible(true);
            FileInputStream fis;
            ObjectInputStream ois;
            if(fd.getDirectory() != null && fd.getFile() != null) {
                fis = new FileInputStream(fd.getDirectory() +
 fd.getFile());
                ois = new ObjectInputStream(fis);
                System.out.println("opening file " +
fd.getDirectory() +  fd.getFile());
                List<Shape> ls = (List<Shape>) ois.readObject();
                Model.setShapeList(ls);
                cp.repaint();
            }
        } catch(Exception exception) {
            System.out.println("exception in open.");
            exception.printStackTrace();
        }
    }
```

```
22 });
```

- IO操作与保存类似
- 在读取完新的shapeList后需要重新绘制

效果演示

- 绘制四种基本图形
- 改变颜色
- 拖动
- 删除
- 变大
- 变粗
- 保存后随意绘制
- 打开刚刚保存的文件