**实习报告 - 基础图形绘制软件使用设计模式实现**

姓名：张涵之 学号：191220154 日期：2022/5/22

1. **引言**

**开发任务简介 - 功能需求：**

1. 设计良好的图形用户界面，有三角形、方框、圆形、椭圆、连接线五种元素可供用户选择后，通过拖拽鼠标，绘制到画布上。
2. 绘制图形时，各有五种填充颜色、线条粗细和背景透明度供用户选择。
3. 有文字描述和阴影两种装饰可供用户选择后，点击图形添加，其中文字描述经文本框输入，阴影可选左上、左下、右上、右下四种不同的方向。
4. 采用键盘快捷键和鼠标操作组合的方式为用户提供以下功能：单个图形的选中和拖拽改变位置；批量图形的选中、复制、删除和组合；组合后的图形和普通图形一样支持拖拽、选中、复制、删除、添加装饰操作。
5. 支持撤销和重做多步操作，支持一键清空画布。
6. 支持画布文件的新建、打开、保存、另存为，还可导出JPG格式。

**开发任务简介 - 其他要求：**

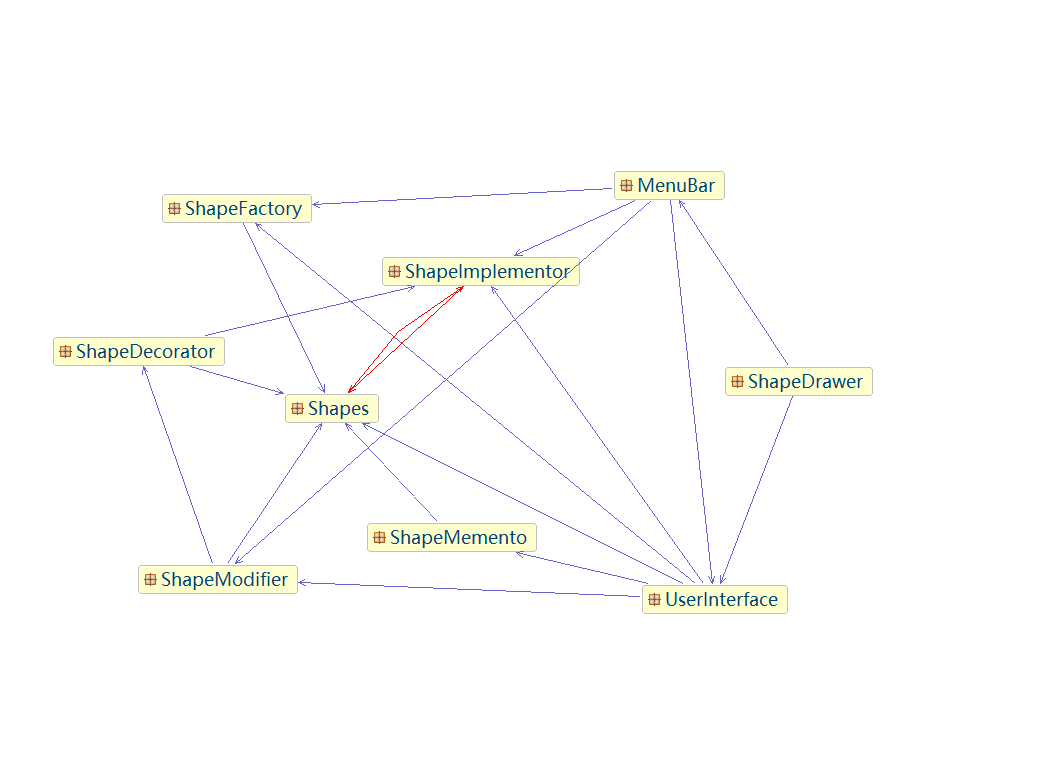
通过面向对象的语言实现，最终通过图形界面与用户进行交互。在设计和实现的过程中要求使用设计模式（不少于4种），在具体实现中应有所体现。

**核心技术简介：**

采用Java语言编写代码，图形界面主要通过java.awt和java.swing提供的方法实现。设计和实现涉及的模式包括工厂方法、装饰模式、原型模式、组合模式、桥接模式、备忘录模式以及不完全的状态模式（借用了部分思想）。

1. **对目标系统的分析和设计**

**UML建模**



Package的设计如上，class的UML图将在具体实现方案中展示。

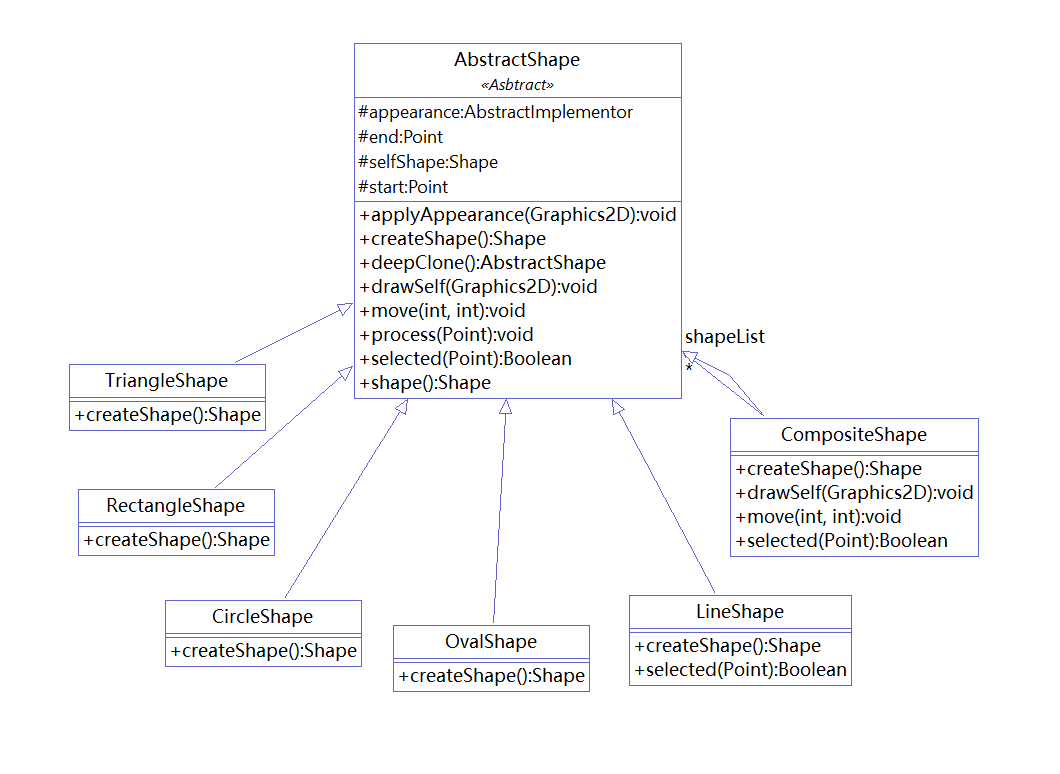
**设计模式**

* 工厂方法：在不同图形的创建（绘制）过程中使用工厂方法模式，从而将图形的选择（菜单点选）和绘制（鼠标事件的交互）解耦，负责绘制的模块不需要知道具体绘制什么图形，只需要调用传入的工厂即可。
* 原型模式：用于各种图形（简单和组合）的拷贝复制。
* 桥接模式：使图形的颜色、粗细和透明度这几个维度可以独立变化。
* 装饰模式：扩展图形功能，可以添加文字描述和阴影装饰。
* 组合模式：提供一种容器对象，使简单图形可以递归地组成复杂图形，且复杂图形在调用移动、复制、删除等方法时可以一视同仁。
* 备忘录模式：提供对多步undo和redo操作的支持。
* 状态模式（\*借用其思想，具体实现有一定差异）：为同一个交互界面（画布）提供拖拽绘制、点击装饰、配合键盘移动/复制/删除/组合这三种不同的状态，不同状态下对相同鼠标和键盘事件的响应也有所区别。

1. **实现方案**

下面将具体介绍每个package中类的设计和实现思路

Shapes: abstract shape, concrete shapes, composite shape



Shapes中定义了抽象图形类、五种基础图形和组合图形类。

其中start和end表示绘制时的对角线两个顶点，决定图形的大小，后续拖拽移动时也用于标记位置；selfShape是java.awt中定义的图形类，用于具体的绘制。appearance是外观（颜色/粗细/透明度）控制接口，后面还会提到。

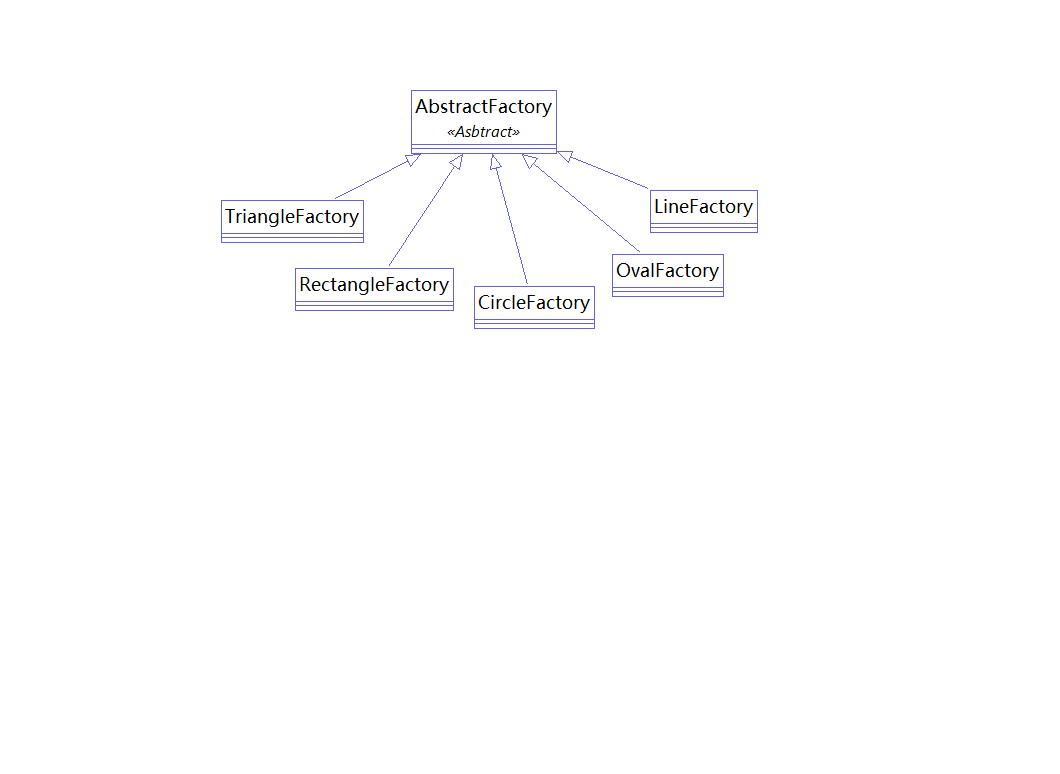
Abstract Shape的接口有：applyAppearance（设置外观）、createShape（用于创建/更新图形）、deepClone（用于图形拷贝复制）、drawSelf（绘制）、move（移动）、process（用于拖拽绘制过程中不断刷新图形位置大小）、selected（用于判断鼠标点击是否选中图形）、shape（返回selfShape）。

每个具体/组合图形都对createShape进行了override，Line和Composite还重载了selected，因为抽象图形中默认的选中是判断鼠标点击处是否落在图形范围内，连接线是判断点击处和线段的距离是否满足一定范围，而对于组合图形递归调用其component的selected方法，只要有一个被选中即返回true。

组合图形还重载了drawSelf和move，均以递归处理component实现。

这个package内的class主要用到了原型模式（模仿课程示例代码进行深拷贝以实现复制）和组合模式（Composite维护一个Abstart类型的shapeList）。

ShapeFactory: abstract shape factory, concrete shape factory

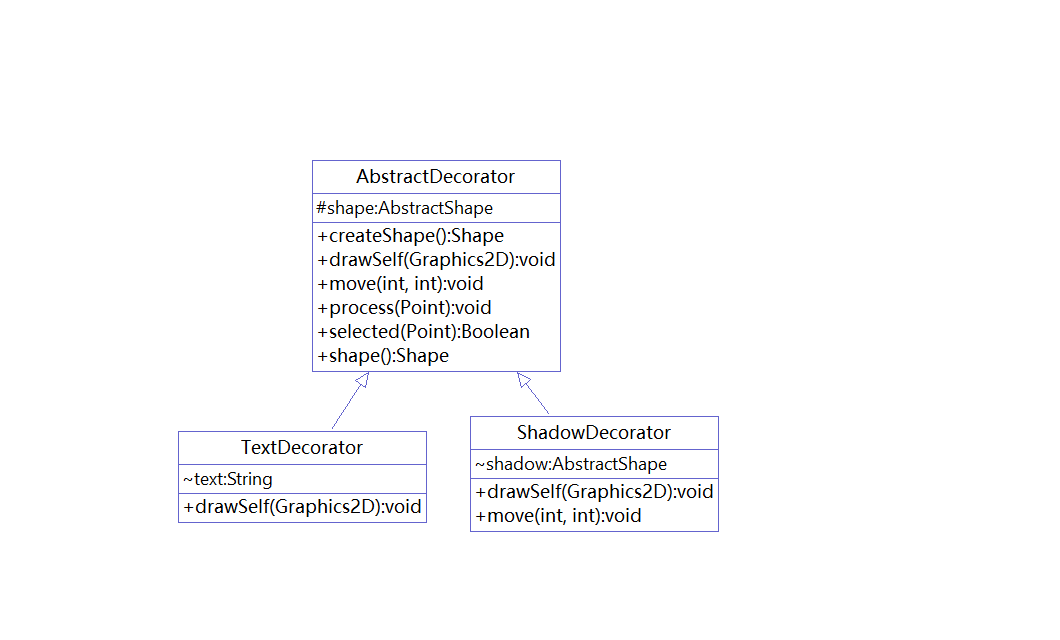


ShapeFactory中定义了抽象图形类和五种基础图形的工厂。

每个具体工厂调用自己对于图形的getShape方法。由于这些工厂只在绘制阶段使用，故只定义了五种基础图形的工厂，而不需要提供组合图形的。

MenuBar的ShapeMenu可以让用户选择自己接下来想要绘制的图形，并把对应的具体工厂传到UserInterface的DrawInterface中。DrawInterface维护一个抽象工厂类，并总是在响应鼠标操作时调用该工厂的getShape创建图形。这里用到了工厂方法模式，使ShapeMenu和DrawInterface可以分开处理“通过窗体下拉菜单选中图形”和“通过鼠标拖拽在画布上绘制图形”两件事。

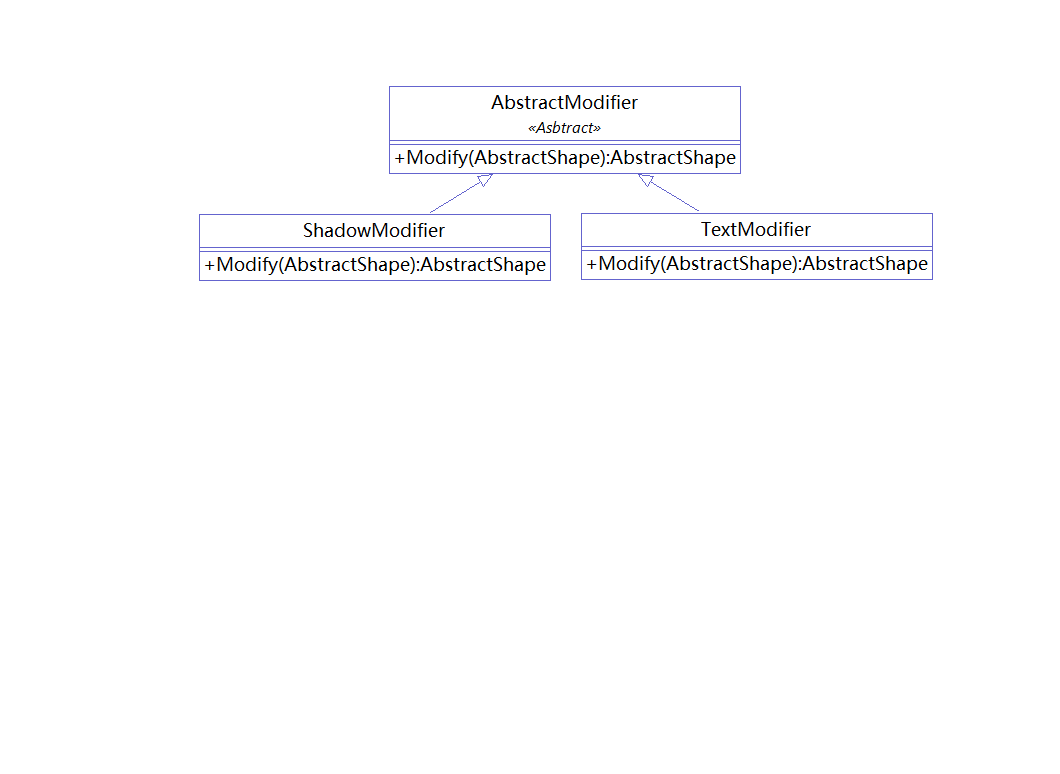
ShapeDecorator: abstract decorator, text decorator, shadow decorator



ShapeDecorator中定义了文字描述和阴影两种装饰（装饰模式）。

装饰器中包含一个shape，即被它装饰的图形的引用，Text额外保存了文字描述内容，Shadow则有一个原图形的副本，其位置信息与原图形略有区别（取决于用户设定阴影位于原图形的左上/左下/右上/右下），其他信息都一样，移动时需与原图形一同移动，两个装饰器都重写了drawSelf方法，调用被装饰图形的drawSelf同时Text还要在图形旁边（本来想实现为正中间，但是要根据图形大小、形状和文字内容长度来调整居中，很难做到效果美观，故直接在图形start point旁边的角落输出文字描述）；Shadow要用预先设定的一种较高透明度画一遍副本图形，再画原图形，以实现一种影子的效果。

ShapeModifier: abstract modifier, text modifier, shadow modifier



ShapeModifier中定义了对应Decorator的Modifier，其实类似工厂方法，调用对应的装饰器对图形进行装饰。此外，TextModifier会产生一个弹窗供用户输入或修改文字描述，ShadowModifier则通过弹窗供用户选中阴影的方向。

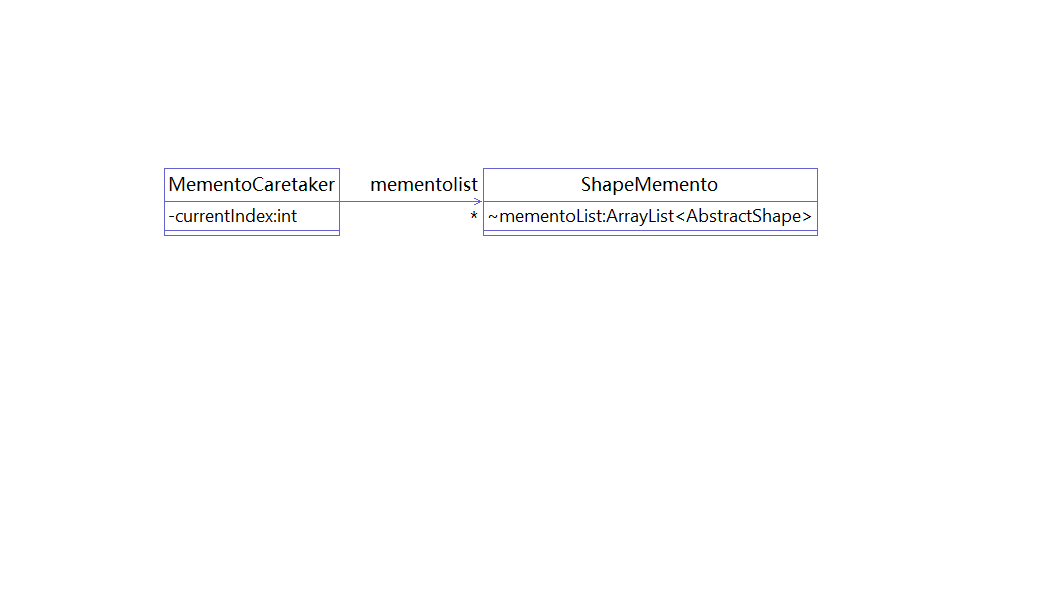
MenuBar中的ModifiyMenu可以将用户选定的Modifier传入UserInterface的ModifyInterface，这个Interface用于判断用户是否点击选中了哪个图形，然后对这个图形调用传入Modifier的Modify方法即可实现图形的装饰。

ShapeImplementor: abstract implementor, appearance implementor



ShapeImplementor中只定义了一个具体类AppearanceImplementor，维护了颜色color、线条粗细width、填充透明度transparency三个维度的变化。程序使用java.awt中的Graphics2D画图，这个Implementor提供applyAppearance方法对传入的Graphic2D进行setPaint、setStroke、setComposite操作。上面提到的各个Shape类维护一个AppearanceImplementor的引用，在drawSelf中调用applyAppearance方法。这个引用有默认值，此外也可以通过MenuBar中的ColorMenu、WidthMenu和TransparencyMenu设定全局Appearance，设置成功后，新画的图形都会以新的颜色、粗细和透明度绘制（桥接模式）。

ShapeMemento: shape memento, memento caretaker



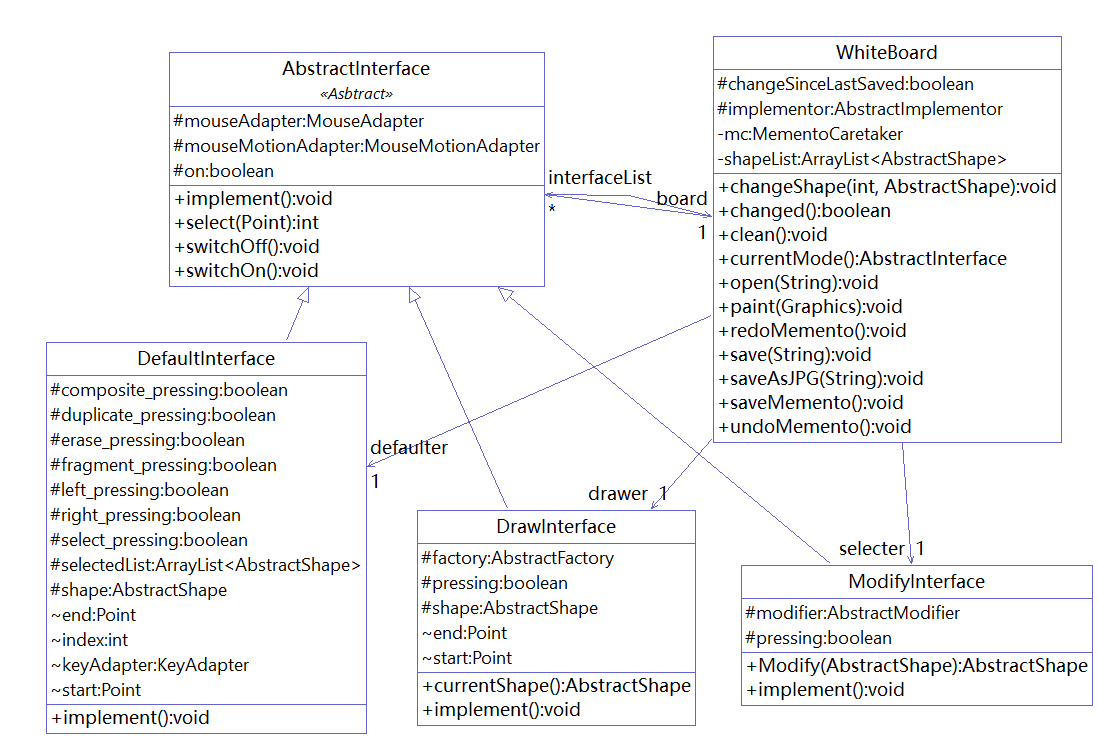
ShapeMemento定义了memento和caretaker，其中memento保存当前画布上所有图形的ArrayList的一个副本。caretaker保存一个memento的列表，还要记录当前列表所读到的index下标，提供addMemento方法清除index后面的项目、向列表添加记录并更新index，提供getLeftMemento、getRightMemento分别向左和向右读列表，从而实现undo和redo操作（备忘录模式）。

MenuBar: abstract, file/edit/shape/modify/color/width/transparency menu

MenuBar定义了许多不同的MenuBar，FileMenu负责文件的新建、打开、保存、另存、导出；Edit Menu提供撤销、重做、清空三个操作；在ShapeMenu选中一个图形可以在画布上进行一次拖拽绘制；在ModifyMenu选中一种装饰可以点击一个画布上的图案添加装饰；Color、Width、Transparency设置全局画笔属性。这些Menu最后都会展示在窗口的顶部，依次排列。

这个package里的class比较简单，也没用到什么设计模式，就不赘述了。

UserInterface: abstract, default/draw/modify interface, whiteboard



UserInterface定义了三种与用户交互的接口和一个“白板”类。这里用到了类似状态模式的思想，白板维护一个判断上次保存后是否进行了新的操作的布尔值，一个全局Implementor，一个memento caretaker和一个shape list，是最综合的模块，此外还有Default、Draw和Modify三个Interface的引用。这三个与用户交互的接口都能响应鼠标和键盘的事件，状态转换由WhiteBoard来操控，保证同一时刻只能有一个在工作。但在具体的实现上，其实是每个引用维护了一个自己的状态值on，根据这个状态值来控制各自的mouse或key event listener是否工作，并提供switchOn和switchOff方法供白板调用。这样和课件中讲到的状态模式实现应该还是有一定的出入，由于我对java.awt里面的各种event使用不怎么熟练，故而暂时没有想到更好的方法。

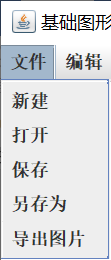
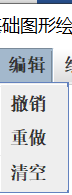
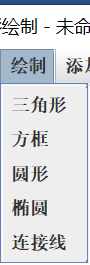
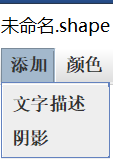
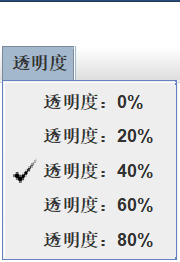
WhiteBoard除了控制状态切换，因为它是当前画布上所有图案shape list的维护者，因此还要处理文件存取、图形绘制（每次有更改时刷新画布），并且要和memento的caretaker进行交互，提供当前shape list进行暂存，或者取出存储的memento进行覆盖。具体到文件存取，虽然手册提出可以“设计一种硬盘文件存储格式保存和加载用户绘制的图形”，但是试了一下用Java内置的ObjectInput/OutputStream，发现效果意外地好，就偷了这个懒。

ShapeDrawer: the shape drawer itself

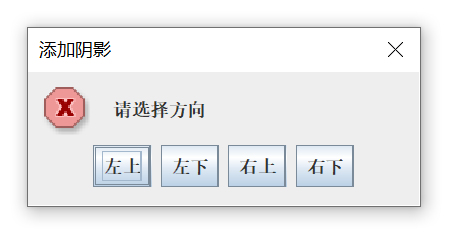
ShapeDrawer定义了整个程序的主类，对JFrame进行extend实现窗口。

1. **实现功能介绍**
2. 有三角形、方框、圆形、椭圆、连接线五种元素可通过鼠标绘制。
3. 有五种填充颜色、线条粗细和背景透明度可进行全局设置。
4. 有文字描述和阴影两种装饰可添加到图形。
5. 单个图形可右键选中拖拽移动位置；批量图形可左键选中，结合键盘快捷键进行复制、删除和组合；组合图形和普通图形一样支持各类操作。
6. 支持撤销和重做多步操作，支持一键清空画布。
7. 支持画布文件的新建、打开、保存、另存，可导出JPG格式。

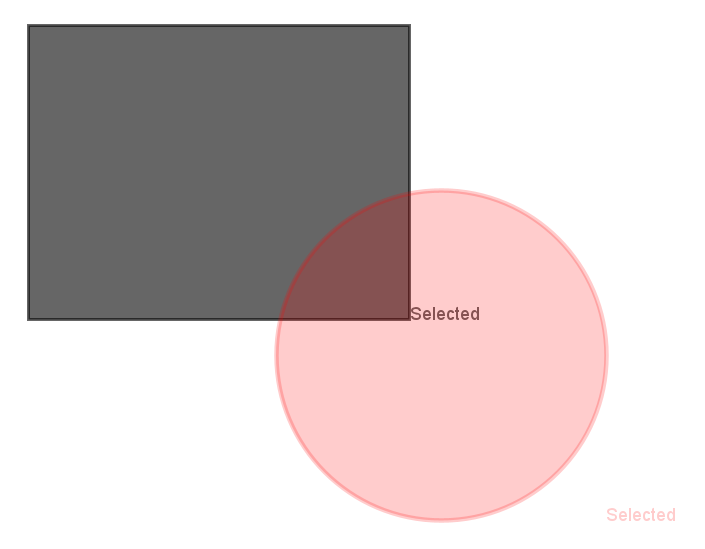
界面展示

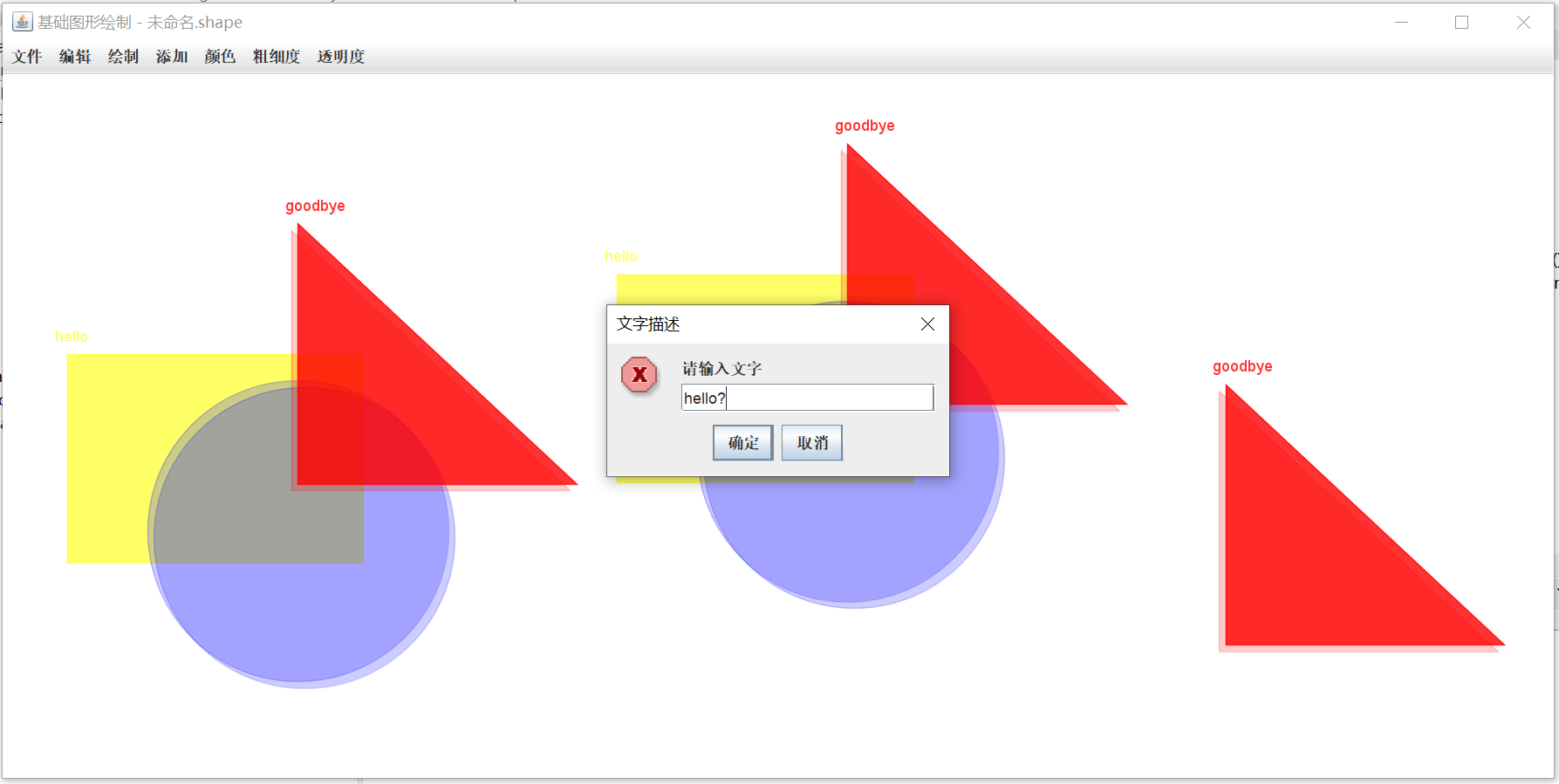
界面顶部工具栏如图。其中“颜色”、“粗细度”、“透明度”为全局设置，默认为黑色、粗细度2和透明度40%。在“绘制”下点选一种图形后，可以通过点击拖拽（不分左右键）在画布上绘制一个对应的图案。在“添加”下点选一种装饰后，可以点击画布上的一个图形进行装饰，其中“文字描述”会提供弹窗编辑或修改文字描述，“阴影”则弹窗选择阴影方向。

除了点选绘制或添加后等待用户操作的状态，画布大多数时间都处于默认模式。该模式下鼠标右键点击拖拽实现单个图形（含组合图形）的移动。提供五种键盘快捷键S(select)、X(cross)、D(duplicate)、C(composite)、F(fragment)和E(erase)。长按S（类似ctrl键使用逻辑）可通过鼠标左键依次单击，对画布上的图案进行多选，选中的图形右下角会出现Selected标识（如下图），按X可取消当前所有选中。对于选中的图形，按D批量复制，按C组合，按F取消组合（对选中的所有组合图形，取消最顶层的组合），按E批量删除。



注：出于某些不为我知的原因，使用快捷键时必须切成英文输入法，如果中文输入法的取词框出现了，那么这个时候按下任何快捷键都是没有用的。



1. **小结**

本次实验中最大的困难有两个，一是将多个设计模式结合使用时，类的设计和不同类之间的通信。我感觉我在使用工厂方法、组合模式、备忘录模式这几个设计模式时，对其优缺点和适用环境的把握相对更到位一些，而对装饰模式和桥接模式的使用比较生硬，没有较好的融合到整个项目中去。

二是这门课是我第一次接触Java这一编程语言。由于感觉纯粹的面向对象语言很有趣，也比较适合应用各种设计模式，就选择了用Java来完成实验。然而我之前编程，尤其是写图形界面的经验主要都在C++，对Java内置的各种窗口控件如何使用、如何响应鼠标键盘事件、如何绘图等都不是很了解，需要在短期内进行学习，主要靠写到的时候再去搜索引擎或者查手册。