经典案例二的评价与改进

> 2021240205班 严晏来 2021902228

一、案例分析

1.1对每个类的理解与解释

1. **CreationTool（创建工具）**：
   - 用于创建图形元素的工具。
   - 属性包括定位起点、定位终点、起点坐标等。
   - 方法包括按下、释放、绘制元素等。
   - 通过构造函数与Diagram（图表）关联。
2. **Diagram（图表）**：
   - 包含图形元素的类。
   - 属性包括元素的Vector。
   - 方法包括绘制、缩放、查找、添加、删除元素等。
3. **DiagramEditor（图表编辑器）**：
   - 负责管理图表编辑的主要类。
   - 属性包括颜色、工具名称数组、工具类型等。
   - 方法包括设置控件、删除元素、处理鼠标事件、创建新图表等。
   - 通过构造函数与DiagramEditorControls（图表编辑器控制）关联。
4. **DiagramEditorApplet（图表编辑器Applet）**：
   - Applet类，用于在Web页面中显示图表编辑器。
   - 包含初始化方法。
5. **DiagramEditorControls（图表编辑器控制）**：
   - 包含在图表编辑器中的控制元素，如工具选择、放大缩小按钮等。
   - 方法包括处理事件等。
   - 通过构造函数与DiagramEditor关联。
6. **Element（元素）**：
   - 图形元素的基类。
   - 包含颜色、边界框等属性。
   - 方法包括移动、缩放、绘制、包含点等。

7. **Ellipse（椭圆）**：
    - 继承自Element，表示椭圆。
    - 包含绘制方法。
8. **EllipseTool（椭圆工具）**：
    - 继承自CreationTool，用于创建椭圆。
9. **Line（直线）**：
    - 继承自Element，表示直线。
    - 包含绘制方法。
10. **LineTool（直线工具）**：
    - 继承自CreationTool，用于创建直线。
11. **SelectionTool（选择工具）**：
    - 继承自Tool，用于选择图形元素。
    - 包含移动、缩放、删除等方法。
12. **Tool（工具）**：
    - 图形编辑器中所有工具的基类。
    - 包含状态、当前坐标等属性。
    - 包含绘制、按下、释放、移动等方法。
13. **Wrectangle（可调整边界矩形）**：
    - 继承自Element，表示可调整边界的矩形。
14. **WrectangleTool（可调整边界矩形工具）**：
    - 继承自CreationTool，用于创建可调整边界的矩形。

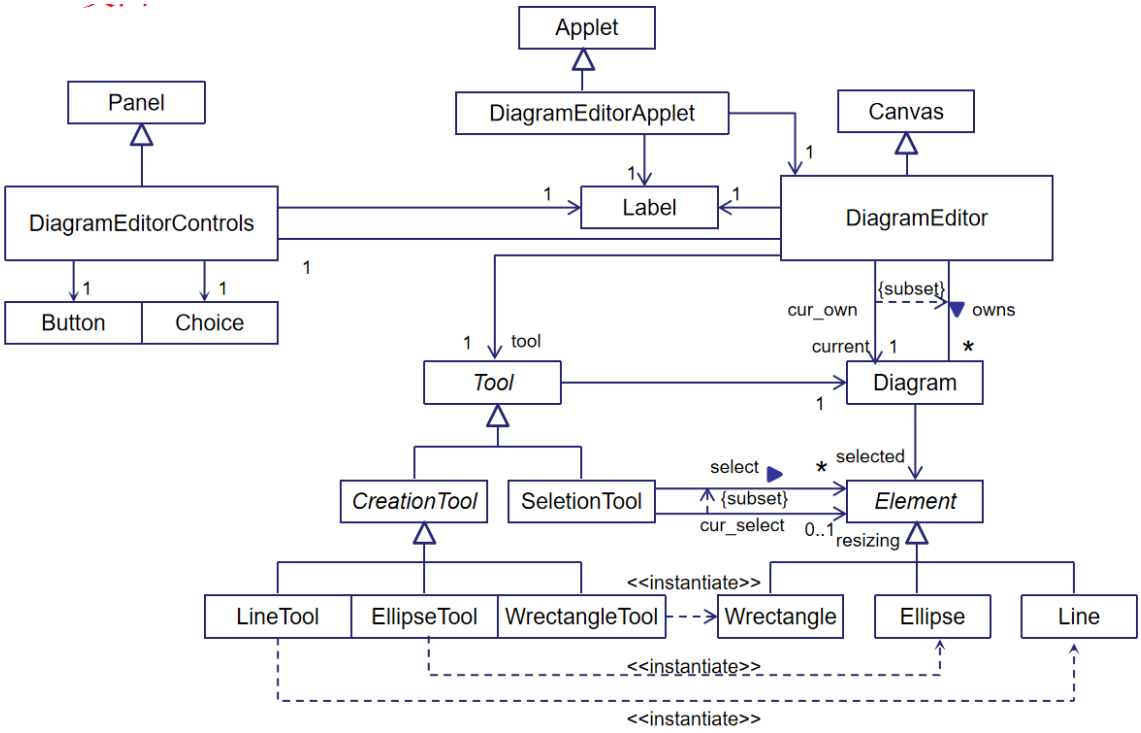
1.2优点分析

这个架构使用了面向对象的设计模式，通过类的继承和关联关系来实现图形编辑器的功能。其中，不同的工具类用于创建不同类型的图形元素，而图形元素类包含了基本的操作方法，如移动、缩放、绘制等。 DiagramEditor 类作为主要的控制器，通过与其他类的关联来管理整个编辑器的状态和操作。

1. **模块化设计 (Modular Design):**
    - 各类，如`CreationTool`、`Diagram`、`DiagramEditor`等都代表不同的模块，负责特定的任务，如创建工具、管理图表或处理整个编辑器的功能。这种模块化结构使得更容易理解和独立维护系统的不同方面。
2. **面向对象设计 (Object-Oriented Design):**
    - `Ellipse`、`Line`、`Wrectangle`等类代表不同类型的图形元素，展现了面向对象的设计。每个类都封装了与特定类型图形元素相关的属性和方法。这种设计允许通过添加新类来扩展系统，而无需修改现有代码，符合开闭原则。
3. **可重用性 (Reusability):**
    - `EllipseTool`、`LineTool`、`WrectangleTool`等工具继承自通用基类`CreationTool`，展示了一种促进代码可重用性的设计。这种继承允许工具创建的共享功能用于不同类型的形状，遵循了“不要重复自己”（DRY）原则，避免了代码重复。
4. **分层设计 (Layered Design):**
    - 存在层次结构，如`DiagramEditor`、`CreationTool`和`Element`，表明了分层设计。每个层次都有特定的职责，例如管理编辑器、创建工具、处理元素等。这种设计使得各层次可以相对独立地进行修改和扩展。

1.3 UML 类图

书上的图：



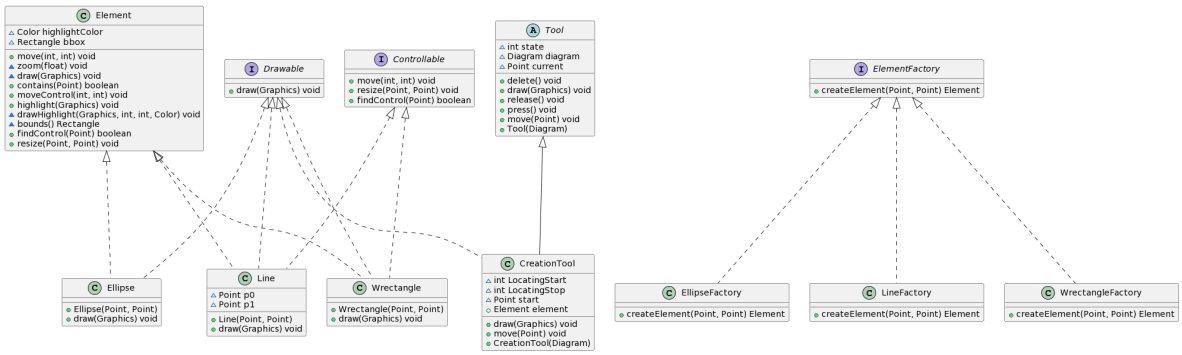
自己构建：

二、案例改进1（工厂方法）

2.1改进后的设计理念

1. **工厂方法模式:** 引入工厂方法模式来创建不同类型的元素对象，以便更容易添加新的元素类型。
2. **抽象工厂模式 :** 使用抽象工厂模式来创建具体工具对象，使得系统更容易扩展以支持新的工具类型。
3. **抽象类和接口 :** 引入抽象类和接口来定义通用的行为和属性，以提高代码的一致性和可扩展性。

2.2改进后的UML类图



2.3改进分析

1. **工厂方法模式：** 引入 `ElementFactory` 接口和具体的工厂类，如 `EllipseFactory`、`LineFactory` 和 `WrectangleFactory`，使得系统更容易添加新的元素类型，而不需要修改现有的创建工具类。
2. **抽象类和接口：** 引入 `Drawable` 接口和 `Controllable` 接口，分别表示可以绘制和可控制的元素。`Ellipse`、`Line` 和 `Wrectangle` 类实现这些接口，提高了系统的一致性。
3. **可扩展性：** 通过引入工厂方法模式和抽象类和接口，系统更容易扩展，支持新的元素类型和工具类型。这样设计使得添加新功能或调整现有功能更加简单，同时保持了代码的一致性和清晰度。

三、案例改进2（观察者）

3.1改进后的设计理念

观察者模式 (Observer Pattern): 引入观察者模式，以便其他部分可以订阅图表元素状态的变化。通过这种方式，可以实现在元素发生变化时通知到相关的组件。

3.2改进后的UML类图

新增：

3.3改进分析

`Element` 类引入了观察者模式，其中 `Observer` 是一个接口，用于表示观察者。具体的图形元素类，如 `Ellipse`、`Line` 和 `Wrectangle` 实现了 `Element` 接口，并可以添加、删除和通知观察者。这样，其他部分可以注册为观察者，以便在图形元素状态变化时得到通知，实现更松散的耦合和更好的系统可维护性。