1、

1. 在某绘图软件中提供了多种大小不同的画笔(Pen),并且可以给画笔指定不同颜色,某设计人员针对画笔的结构设计了如图 1-1 所示的初始类图。

通过仔细分析,设计人员发现该类图存在非常严重的问题,即如果需要增加一种新的大小或颜色的笔,就需要增加很多子类,例如增加一种绿色的笔,则对应每一种大小的笔都需要增加一支绿色笔,系统中类的个数急剧增加。

试根据依赖倒转原则和合成复用原则对该设计方案进行重构,使得增加新的大小或颜色的笔都较为方便,请绘制重构之后的结构图(类图)。

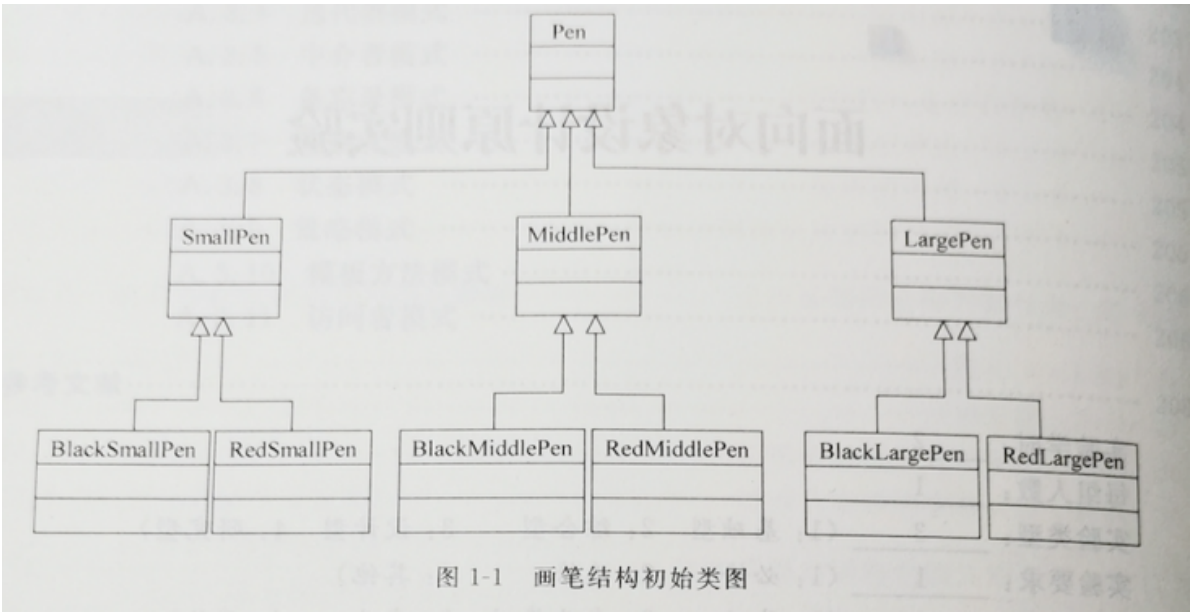
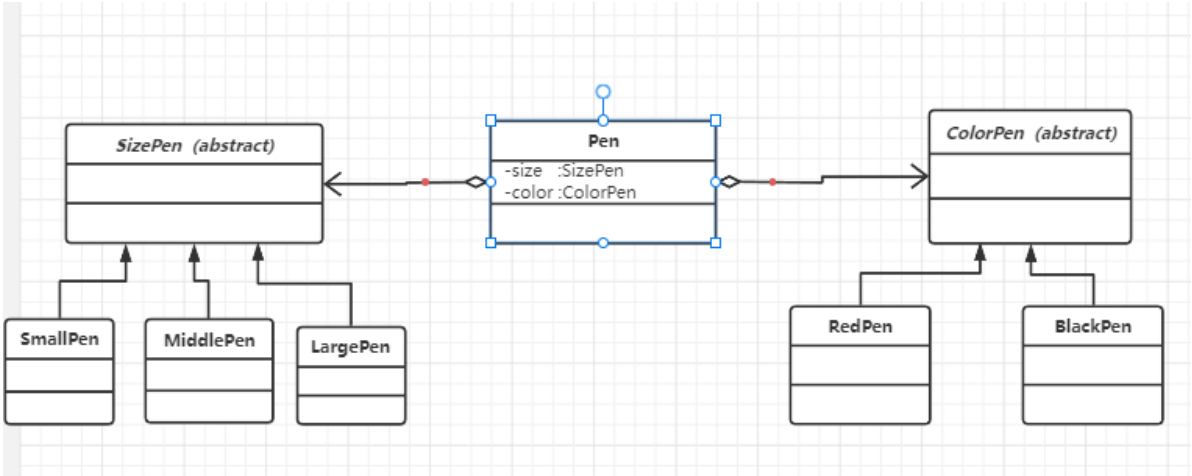


图 1-1 画笔结构初始类图

笔的大小、颜色更应该成为笔本身具有的属性，于是将这两种属性抽象出来形成两个抽象类。具体的SmallPen、MiddlePen、RedPen等应该继承这个抽象类。最后的笔去拥有不同属性、大小的笔



2、

2. 在某公司财务系统的初始设计方案中存在如图 1-2 所示的 Employee 类,该类包含员工编号(ID)、姓名(name)、年龄(age)、性别(gender)、薪水(salary)、每月工作时数(workHoursPerMonth)、每月请假天数(leaveDaysPerMonth)等属性。该公司的员工包括全职和兼职两类,其中每月工作时数用于存储兼职员工每个月工作的小时数,每月请假天数

用于存储全职员工每个月请假的天数。系统中两类员工计算工资的方法也不一样，全职员工按照工作日数计算工资，兼职员工按照工作时数计算工资，因此在 Employee 类中提供了两个方法 calculateSalaryByDays()和 calculateSalaryByHours()，分别用于按照天数和时数计算工资，此外，还提供了方法 displaySalary()用于显示工资。

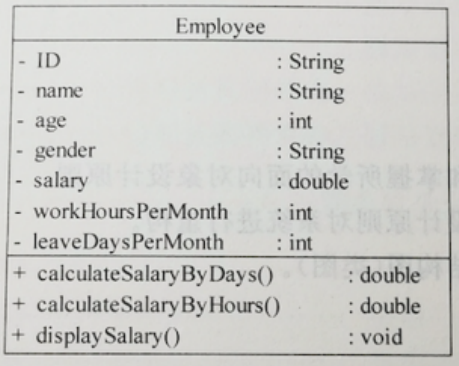
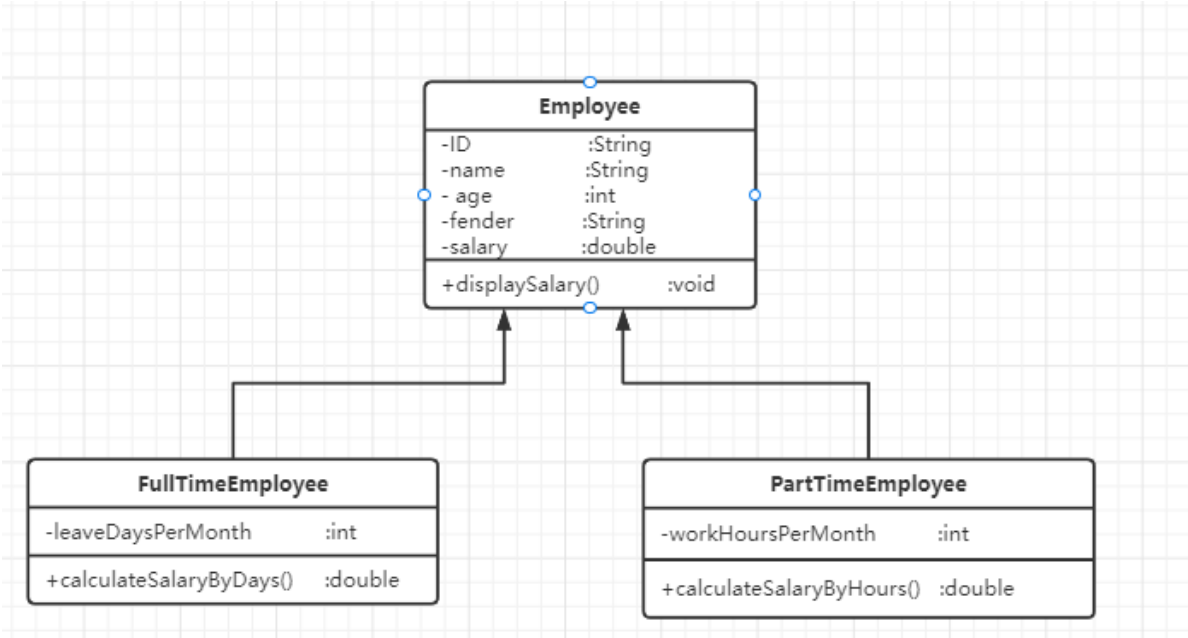


图 1-2　Employee 类初始类图

试采用所学面向对象设计原则分析图 1-2 中 Employee 类存在的问题并对其进行重构，绘制重构之后的类图。

原有employee类存在的问题是违背了单一职责原则，employee类同时拥有了全职、兼职员工的属性，粒度太大。于是可以将其拆分。



3、

3. 在某图形界面中存在如下代码片段，组件类之间有较为复杂的相互引用关系：

```
//按钮类
public class Button {
    private List list;
    private ComboBox cb;
    private TextBox tb;
    private Label label;
    //...
    public void change() {
        list.update();
```

```
        public void update() {
            //...
        }
        //...
    }

    //列表框类
    public class List {
        private ComboBox cb;
        private TextBox tb;
            //...
        public void change() {
            cb.update();
            tb.update();
        }
        public void update() {
            //...
        }
        //...
    }

    //组合框类
    public class ComboBox {
        private List list;
        private TextBox tb;
        //...
        public void change() {
            list.update();
            tb.update();
        }
        public void update() {
            //...
        }
        //...
}

//文本框类
public class TextBox {
    private List list;
    private ComboBox cb;
    //...
    public void change() {
        list.update();
        cb.update();
    }
```

```
        public void update() {
            //...
        }
        //...
    }

    //文本标签类
    public class Label {
        //...
        public void update() {
            //...
        }
        //...
    }
```

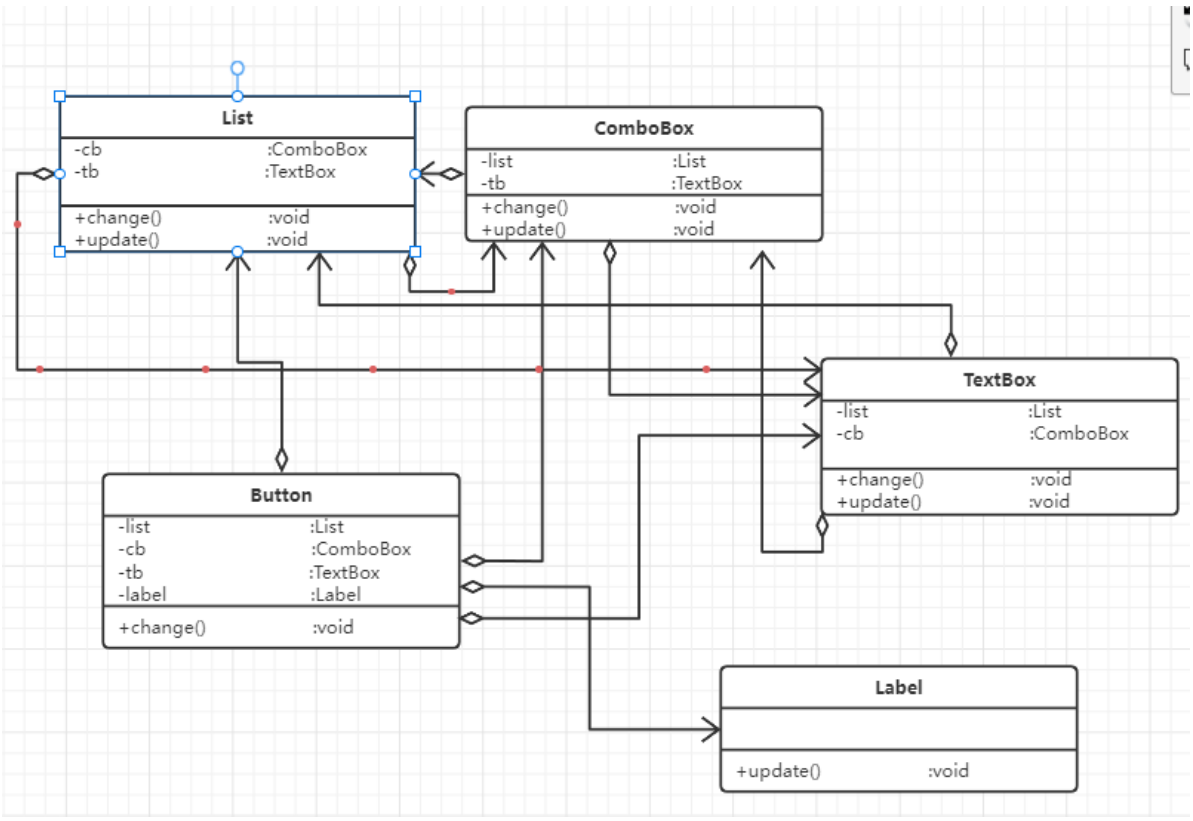
如果在上述系统中增加一个新的组件类，则必须修改与之交互的其他组件类的源代码，将导致多个类的源代码需要修改。

基于上述代码，请结合所学知识完成以下两道练习题：

（1）绘制上述代码对应的类图。

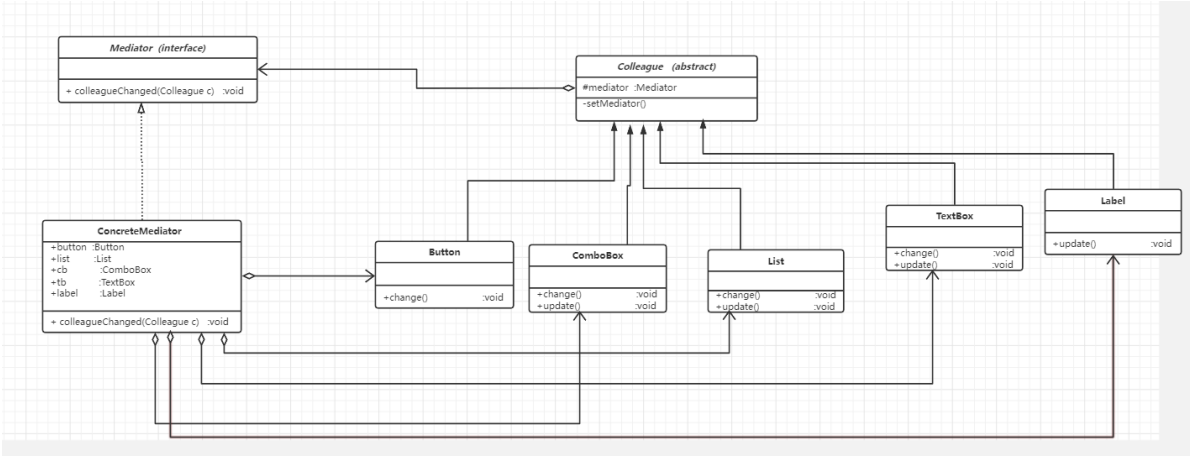
（2）根据迪米特法则对所绘制的类图进行重构，以降低组件之间的耦合度，绘制重构后的类图。

1、



2、最大的问题是关系结构过于复杂、你中有我、我中有你、可以增加一个中介者来解决这一问题。这就是根据迪米特原则设计的中介者模式。各个组件在要调用其他组件时，通过与中介者交互来实现调用。

每个组件都要有一个设置、使用中介者的方法、所以这些方法可以抽象出来成为公共的抽象父类。所有具体组件都继承自这个公共父类。中介者为了实现组件之间的调用，需要拥有这些组件。



4、

4. 在某图形库 API 中提供了多种矢量图模板，用户可以基于这些矢量图创建不同的图形，图形库设计人员设计的初始类图如图 1-3 所示。

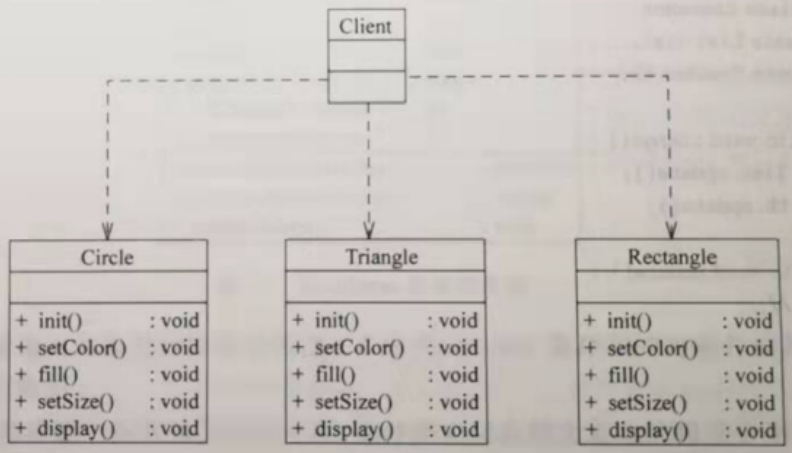


图 1-3 图形库初始类图

在该图形库中，每个图形类（例如 Circle、Triangle 等）的 init()方法用于初始化所创建的图形，setColor()方法用于给图形设置边框颜色，fill()方法用于给图形设置填充颜色，setSize()方法用于设置图形的大小，display()方法用于显示图形。

用户在客户类(Client)中使用该图形库时发现存在如下问题：

(1) 由于在创建窗口时每次只需要使用图形库中的一种图形，因此在更换图形时需要修改客户类源代码。

(2) 在图形库中增加并使用新的图形时，需要修改客户类源代码。

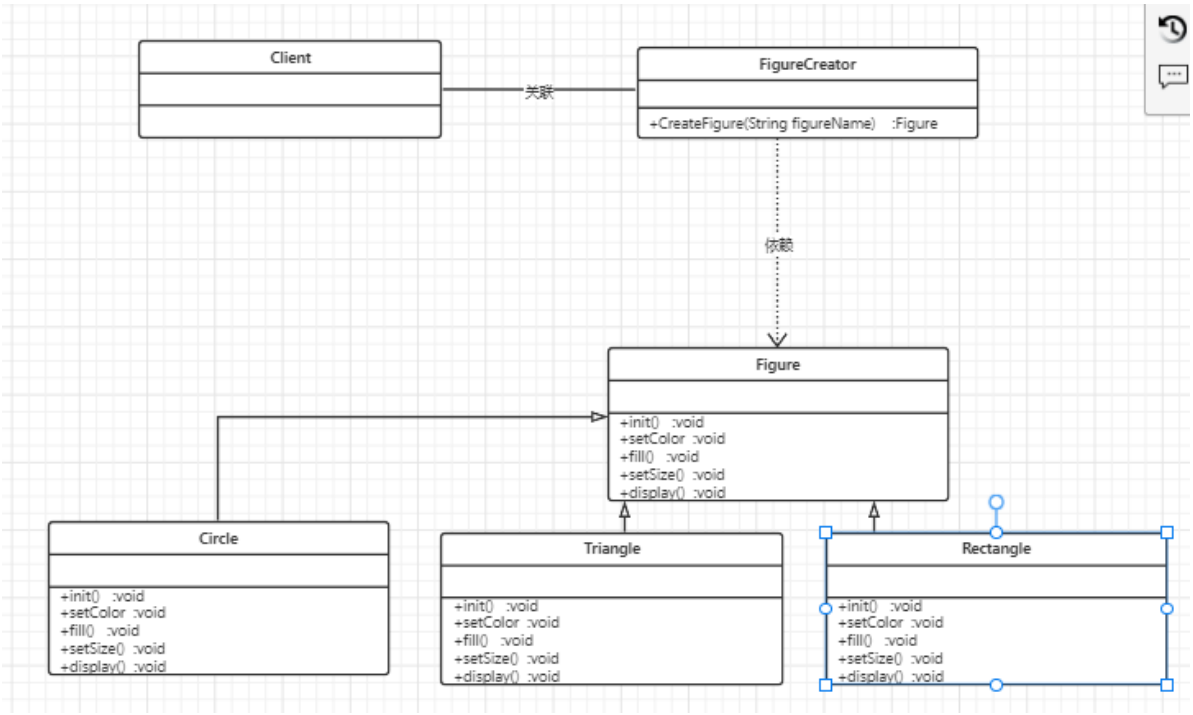
(3) 客户类在每次使用图形对象之前需要先创建图形对象，有些图形的创建过程较为复杂，导致客户类代码冗长且难以维护。

现需要根据面向对象设计原则对该系统进行重构，要求如下：

(1) 隔离图形的创建和使用，将图形的创建过程封装在专门的类中，客户类在使用图形时无须直接创建图形对象，甚至不需要关心具体图形类类名。

(2) 客户类能够方便地更换图形或使用新增图形，无须针对具体图形类编程，符合开闭原则。

请绘制重构后的结构图(类图)。


Client
关联
FigureCreator
+CreateFigure(String figureName) :Figure
依赖
Figure
+init() :void
+setColor :void
+fill() :void
+setSize() :void
+display() :void
Circle
+init() :void
+setColor :void
+fill() :void
+setSize() :void
+display() :void
Triangle
+init() :void
+setColor :void
+fill() :void
+setSize() :void
+display() :void
Rectangle
+init() :void
+setColor :void
+fill() :void
+setSize() :void
+display() :void