|  |  |
| --- | --- |
| 一、实验内容  一个软件团队开发绘图系统，设计了圆对象(Circle) 、矩形对象(Rectangle)，线对象(Line)，这三者都具备Draw()方法（绘制图形）。现在使用装饰器模式对这三个图形对象进行扩展，使系统能够实现对象的边框和填充行为，即，针对一个图形对象来说，其可以进一步支持有颜色的边框ColorEdge和带颜色的填充ColorFill两种行为。  请设计对应的类图，并给出实现代码。 | |
| 二、实验过程记录 | |
| 类图 | |  | | --- | |  | |
| 实验过程记 录 | 按照装饰模式的设计思路，在本例中Shape充当抽象图形类，它的作用是使客户端以一致的方式处理未被装饰和装饰之后的对象，实现客户端的透明操作；其子类Circle、Rectangle、Line充当具体图形类，实现了抽象图形类中声明的方法；ShapeColor作为Shape的另一个子类，充当抽象装饰类，同时与抽象图形类存在一个关联关系，维持了对抽象图形类型对象的引用，也注入了抽象图形类型的对象；而ColorFill带颜色填充类和ColorEdge有颜色的边框类继承了抽象装饰类，充当具体装饰类。客户端代码则通过创建一个具体图形对象，将它作为构造函数的参数注入到具体装饰类，调用具体装饰类中的display()方法就能得到装饰后的对象。 |
| 总结 | 装饰模式以对客户透明的方式动态地给一个对象附加上更多的责任，可以在不需要创建更多子类的情况下让对象的功能得到扩展，有效地降低了系统的耦合度。在设计中，我更加理解了半透明装饰模式和透明装饰模式的区别，并思考了在本例中运用哪一种模式更加合适。透明装饰模式中客户端完全针对抽象编程，将具体图形和具体装饰对象全部声明为抽象构件类型，方便客户端透明地使用装饰之前和装饰以后的对象，还可以对已经装饰过的对象进行多次装饰。半透明装饰模式则为了能够单独调用新增的业务方法，只能用具体装饰类型定义装饰之后的对象，它给系统带来更多的灵活性的同时，又有不能实现对同一个对象进行多次装饰的缺点，而且需要客户端区别对待装饰前后的对象，因此在本例中采用透明装饰模式。我觉得在实际问题中还可以考虑用XML配置文件存储具体装饰类的类名和具体图形类的类名，利用反射机制创建对象，使系统更符合开闭原则。 |
| 备注 | 在客户端代码中创建一个Circle类型的具体图形对象shape，再将shape作为构造函数的参数注入到具体装饰类ColorEdge中，得到一个装饰之后的对象shapeCE；如果还想叠加装饰，可以把装饰了一次的shapeCE对象作为参数注入另一个装饰类ColorFill中，得到装饰两次的对象shapeCF，然后调用shapeCF的draw()方法就能得到既有彩色边框又有颜色填充的圆形。 |