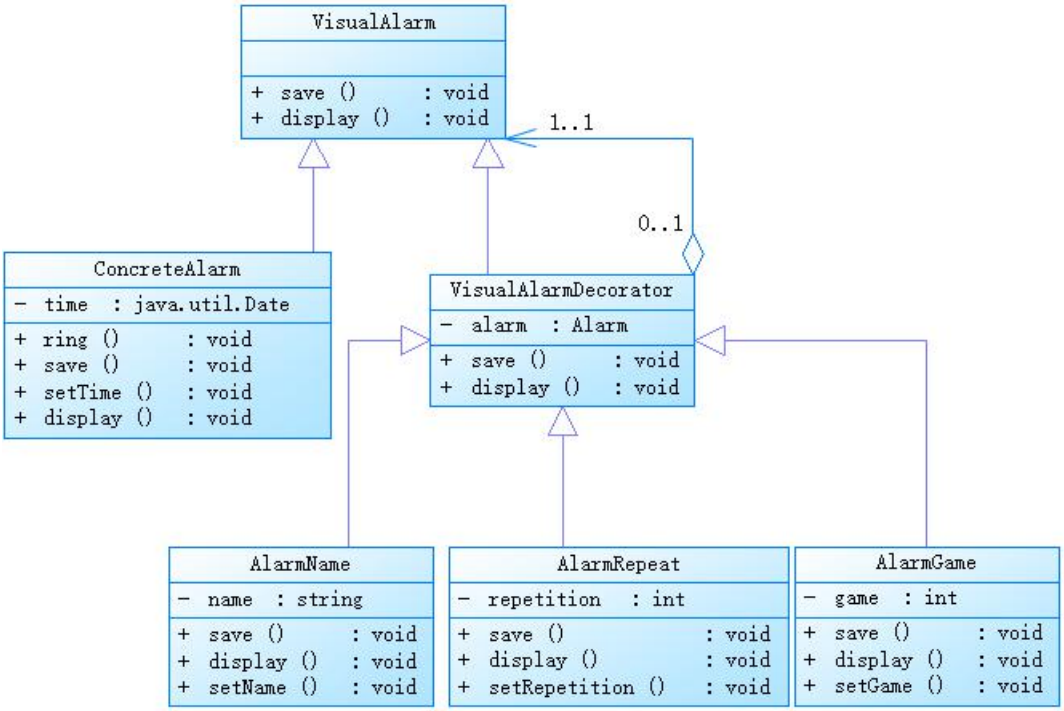


Decorator 模式：

角色有：

VisualAlarm：给出抽象接口，是装饰类和 ConcreteAlarm 的公共父类

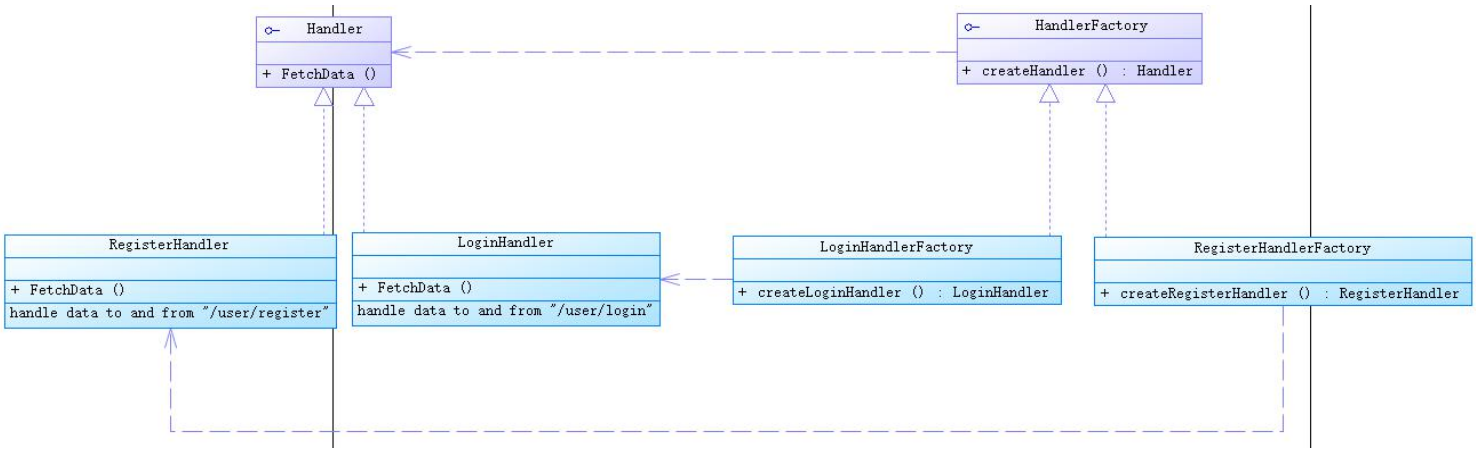
ConcreteAlarm：用来定义具体的闹钟，主要属性是闹钟响起时间

VisualAlarmDecorator：抽象装饰类，持有一个闹钟实例

AlarmGame：实现了 VisualAlarmDecorator，负责给 Alarm 添加新的装饰

Decorator 模式以对客户端透明的方式拓展对象的功能，是继承关系的一种替代方案。

在闹钟类的设计中，我们采用了 Decorator 模式，用户可以只设置闹钟的时间，也可以设置闹钟的游戏、名称、重复、游戏等，允许用户根据自己对功能的需求，动态的决定贴上一个需要的装饰（使闹钟更加个性化），或者除掉一个不需要的装饰（使闹钟更加简洁明了），创造出很多不同的行为组合。



工厂模式:

在前端与后端通讯的时候，很多部分逻辑是重复的，比如创建一个新的 OkHttpClient 对象用于发送 Http 请求，需要修改只是 URL、参数和数据处理，于是我们抽象出了一个 HandlerFactory 对象，可以相应创建出用以处理登录的 LoginHandlerFactory 和用以处理注册的 RegisterHandlerFactory 对象，他们有着不同的 handler（LoginHandler 和 RegisterHandler），可以分别对返回数据进行不同的处理，但代码整体框架逻辑相似，进行这样的抽象可有效提高重用率和可读性。

（注：实际代码命名略有不同，为了使解释更清晰所以有一定修改）