**实验十四 设计模块（三 ）**

**实验目的：**

学习设计模式，能在项目设计中运用设计模式进行面向对象设计

**实验内容：**

**1. 阅读下面设计模式资料（或查阅其它相关资料），结合项目的进程和开发历程，分析项目采用了那些设计模式**

Design Patterns-Elements of Reusable Object-Oriented Software.pdf

The GoF Design Patterns Reference.pdf

[Design Patterns - Wikipedia](https://en.wikipedia.org/wiki/Design_Patterns)

1. **抽象工厂模式(Abstract Factory)**：用于提供一个接口，创建一系列相关或相互依赖的对象，而无需指定它们具体的类。此模式确保系统能够独立于产品类的具体实现。
2. **装饰器模式(Decorator)**：动态地给对象添加新的责任。此模式允许在不改变对象结构的情况下增加功能，例如在用户界面中添加额外的功能而不影响其基本操作。
3. **外观模式(Facade)**：为子系统中的一组接口提供一个一致的界面，使子系统更容易使用。它通过简化复杂的子系统的接口来提高代码的可读性和可维护性。
4. **享元模式(Flyweight)**：运用共享技术有效地支持大量细粒度的对象。通过共享相同部分的状态，可以显著减少内存使用量，如文档编辑器中字符的表示。
5. **代理模式(Proxy)**：为其他对象提供一个代理或占位符，以控制对这个对象的访问。这可以用于延迟加载、权限检查或远程调用等场景。
6. **访问者模式(Visitor)**：定义一个作用于某对象结构中的各元素的操作。它使得你可以在不改变各元素的类的前提下定义作用于这些元素的新操作。
7. **依赖注入(Dependency Injection)**：将依赖关系从组件内部解耦，通过构造函数、setter方法或接口注入。这样可以降低模块间的耦合度，并使单元测试更简单。
8. **组合模式(Composite)**：将对象组合成树形结构以表示“部分-整体”的层次结构。组合模式使得客户以一致的方式处理单个对象以及组合对象。
9. **策略模式(Strategy)**：定义一系列算法，把它们一个个封装起来，并且使它们可相互替换。策略模式让算法可独立于使用它的客户而变化。
10. **迭代器模式(Iterator)**：提供一种方法顺序访问一个聚合对象中各个元素而又不暴露该对象的内部表示。
11. **中介者模式(Mediator)**：用一个中介对象来封装一系列的对象交互，使得各对象不需要显式地相互引用，从而使其耦合松散，而且可以独立地改变它们之间的交互。
12. **备忘录模式(Memento)**：在不破坏封装性的前提下，捕获一个对象的内部状态，并在该对象之外保存这个状态。这样以后就可将该对象恢复到原先保存的状态。