实验十二　设计模块（一 ）

实验目的：
1. 培养设计原则实践的能力
2. 学习依赖注入（dependency injection）

实验内容：

1. 参考教材 6.2，结合项目的进程和开发历程，从设计原则的几个方面，组员对负责设计的模块进行评估，思考存在的问题和解决方案。

设计原则是指把系统功能和行为分解成模块的指导方针。它从两种角度明确了我们应该使用的标准:系统分解，以及确定在模块中将要提供哪些信息(和隐蔽哪些信息)。设计原则在进行创新性设计时十分有用，此外它们还有其他用武之地，特别是在设计公约、设计模式和体系结构风格的设计建议基础的形成过程中。设计原则主要包含一下几个方面——模块化、接口、信息隐藏、增量式开发、抽象、通用性。

模块化

模块化(modularity)，也称作关注点分离(separation of concerm),是一种把系统中各不相关的 部分进行分离的原则，以便于各部分能够独立研究(Dijkstra 1982)。关注点(concem)可以是功能、数据、特征、任务、性质或者我们想要定义或详细理解的需求以及设计的任何部分。为了实现模块化设计，我们通过辨析系统不相关的关注点来分解系统，并且把它们放置于各自的模块中。如果该原则运用得当，每个模块都有自己的唯一目的，并且相对独立于其他模块。使用这种方法，每个模块理解和开发将会更加简单。同时，模块独立也将使得故障的定位和系统的修改更加简单(因为对于每一个故障，可疑的模块会减少，且个模块的变动所影响的其他模块会减少)。为了确定一个设计是否很好地分离了关注点，使用以下两个概念度量模块的独立程度：

（1）耦合度： 包括紧密耦合（模块之间存在大量依赖关系）、松散耦合（模块之间相互连接比较弱）和非耦合（模块之间相互独立）。耦合分为很多类型，例如：内容耦合、公共耦合、控制耦合、标记耦合、数据耦合。在设计过程中，要尽量实现低耦合，因此，在划分模块时，要充分考虑各个模块之间的相互关系。

（2）内聚度： 是指模块内部各个元素（数据、功能、内部模块）的粘合程度。聚合包括巧合内聚、逻辑内聚、时态内聚、过程内聚、通信内聚、功能内聚、信息内聚，它们的内聚度是逐渐升高的。我们的目的是尽可能地使模块高内聚，这样各个模块之间能易于理解，减少更改。

因此为了实现高内聚，可采用面向对象的设计，当对象和动作有着一个共同并且明确的目标值，将它们放到一个模块内。

接口

接口(interface) 为系统其余部分定义了该软件单元提供的服务， 以及如何获取这些服务。一个对象的接口是该对象所有公共操作以及这些操作的签名(signature)的集合，指定了操作名称、参数和可能的返回值。更全面地讲，依据服务或假设，接口还必须定义该单元所必需的信息，以能确保该单元能够正确工作。例如，在上.述对象的接口中，对象的操作可能要使用程序库、调用外部服务，或者只有上下文环境符合一定条件时才能被调用，而一旦不满足或违背这些条件中的任何一个，操作将不能如预期那样提供应有的功能。因此，软件单元的接口描述 了它向环境提供哪些服务，以及对环境的要求。

每个软件单元有一个边界将它和系统的其余部分分开，以及一个接口来和其它软件单元进行交互。接口是系统其余部分定义了该软件单元提供的服务，以及如何获取这些服务。一个对象的接口是该对象所有公共操作以及这些操作的签名的结合，指定了操作名称、参数和可能的返回值。

接口是一种设计结构，它对其它开发人员封装和隐藏了软件单元的设计和实现细节。

对于我们的项目，可能由于对接口的规格说明不到位而影响到实际工作的开展。因此接口的规格说明描述必须要准确详尽，要向其它开发人员传达正确应用某个软件单元的所有信息。除了单元的访问功能和它们的签名，还应该包括：

目标： 为每个访问函数的功能性建立充分详细的文档，以帮助其他开发入员找出最符合他们需要的访问函数。

前置条件： 列出所有假设，又称为前置条件（如，输入参数的值、全局资源的状态，或者存在哪些程序库及软件单元），以帮助其他开发人员了解在何种情况下该软件单元才能正确工作。

协议： 协议的信息包括访问函数的调用顺序、两个构件交换信息的模式。比如，一个模块进行调用共享资源之前需要被授权允许。

后置条件： 将可见的影响称为后置条件。 我们为每个访问函数的后置条件编写文档，包括返回值、引发的异常以及公共变量（如输出文件）的变化，这样，调用它的代码才能对函数的输出作出适当的反映。

质量属性： 质量属性（性能、可靠性）是对开发人员和用户可见的。

此外，结构可能导致一些耦合的发生，为了实现低耦合，要尽量将接口设计的简单。

信息隐藏

信息隐藏的目标是是的软件系统更加易于维护。信息隐藏是将单元的设计决策隐藏，每个软件单元都封装了一个将来可以改变的独立的设计决策，然后我们根据外部可见的性质，在接口和接口规格说明的帮助下描述各个软件单元。信息隐藏的一个很大好处是使得软件单元具有低耦合度。

实现信息隐藏的方法： 在面向对象设计中，可将一个系统分解成对象和它们的抽象类型，换言之，每个对象（模块）都是抽象数据类型的示例。从这个角度上讲，每个对象对其他对象隐藏了它的数据表示，其他对象访问给定的一个对象的唯一途径是通过这个对象接口中声明的访问函数。

增量式开发

增量式开发是软件工程当中，一种常用的软件开发过程思想。其中增量是指在软件开发过程中，先开发主要功能模块，再开发次要功能模块，逐步完善，最终开发出符合需求的软件产品。实际上，我们在初期的设计中，已考虑增量式开发，分别分为三个模块：信息录入和修改模块 、查询信息模块、学生论坛模块。

抽象

抽象是一种忽略细节来关注其他细节的模型或表示。在我们的项目开发中，我们将功能模块化，对各个细化功能用类和函数封装使用，起到了抽象的效果。

通用性

在开发软件单元时，使它尽可能地能够成为通用的软件，来加强它在将来某个系统中能够被使用的可能性。

问题 1：

某些功能可能跨越了多个模块，导致模块间耦合度过高。

部分模块的功能划分不够清晰，职责重叠。

解决方案：

重新梳理系统需求，确保每个模块的功能职责明确，避免功能重叠。

使用接口或消息队列等方式降低模块间的耦合度，确保模块间的通信清晰且易于管理。

问题 2：

接口设计可能过于复杂，导致使用和维护困难。

接口文档可能不够详细，缺乏必要的说明和示例。

解决方案：
简化接口设计，确保接口功能明确、参数清晰。
完善接口文档，提供详细的说明和示例，帮助开发者理解和使用接口。
问题 3：
某些敏感数据可能被不当地暴露给了外部模块或用户。
缺乏对内部数据和算法的访问控制机制。
解决方案：
审查系统设计和实现，确保敏感数据得到妥善保护，不被非法访问和篡改。
引入访问控制机制，如身份验证、权限管理等，限制对内部数据和算法的访问。
问题 4：
增量版本之间的依赖关系可能过于复杂，导致后期维护困难。
某些增量版本可能未经充分测试就被部署到生产环境。
解决方案：
确保增量版本之间的依赖关系清晰，避免过度依赖和复杂的交互。
加强测试和验证流程，确保每个增量版本都经过了充分的测试，符合预期要求。
问题 5：
系统中可能存在过度抽象或抽象不足的情况。
某些设计可能违反了面向对象的设计原则。
解决方案：
审查系统设计和实现，确保抽象的使用恰到好处，避免过度或不足的抽象。
学习和应用面向对象的设计原则，如单一职责、开闭原则等，提升系统的可维护性和可扩展性。