**实验十二 设计模块（一 ）**

**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）**

**二、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。**

**2. 阅读下面DI资料****（或查阅其它相关资料），学习依赖注入技术。**

[**Dependency injection - Wikipedia**](https://en.wikipedia.org/wiki/Dependency_injection#:~:text=In%20software%20engineering%2C%20dependency%20injection,object%20is%20called%20a%20service.)

**Dependency Injection-A Practical Introduction.pdf**

依赖注入（Dependency Injection，简称DI）是一种软件设计模式，用于实现控制反转（Inversion of Control，IoC）。这种模式允许你在运行时选择组件，而不是在编译时。依赖注入的主要目的是减少代码之间的耦合度，从而提高代码的可扩展性、可维护性和可测试性。

依赖注入的核心思想是：

（1）松耦合：组件之间的依赖关系通过抽象（接口或抽象类）来定义，而不是直接依赖于具体的实现类。这样做可以让各个组件保持独立，便于替换和更新。

（2）控制反转：在传统的程序设计中，组件通常会自己创建或直接获取它所依赖的对象。而依赖注入则是将这种控制权反转，由外部（比如依赖注入框架或容器）来提供所需的依赖对象。

（3）依赖传递：依赖注入容器会在组件需要时，动态地将依赖对象传递给该组件。这通常通过构造函数、属性或方法来实现。

依赖注入通常有以下几种实现方式：

（1）构造函数注入（Constructor Injection）：通过类的构造函数来传递依赖

（2）属性注入（Property Injection）：通过类的属性来设置依赖。

（3）方法注入（Method Injection）：通过类的方法来接收依赖。

（4）环境上下文（Ambient Context）：依赖可以在环境中获取，不需要显式注入。

（5）服务定位器（Service Locator）：通过一个服务定位器来获取依赖对象，但这种方式通常不推荐，因为它隐藏了依赖关系并可能导致代码难以测试。

依赖注入在现代软件开发中非常流行，特别是在企业级应用、大型项目和微服务架构中。它与许多设计原则和模式相辅相成，如SOLID原则，能够帮助开发者编写出更加灵活、可维护的代码。

项目跟踪，建立能反映项目及小组每个人工作的进度、里程碑、工作量的跟踪图或表，将其保存到每个小组选定的协作开发平台上，每周更新。