依赖倒转原则（Dependency Inversion Principle, DIP）是面向对象设计中的一个重要原则，其核心理念在于减少模块间的耦合性，提高系统的稳定性和可维护性。以下是关于依赖倒转原则的详细解释：

**一、定义**

依赖倒转原则要求高层模块不应该依赖于低层模块，它们都应该依赖于抽象。抽象不应该依赖于细节，细节应该依赖于抽象。这意味着在代码设计时，我们应该尽量将具体的实现细节与高层逻辑分离，通过抽象接口或类来实现两者之间的交互。

**二、核心思想**

1. **面向接口编程**：依赖倒转原则的中心思想是面向接口编程，而不是面向实现编程。这意味着我们应该通过接口或抽象类来定义高层模块和低层模块之间的交互方式，而不是直接依赖于具体的实现类。
2. **抽象稳定性**：相对于细节的多变性，抽象的东西要稳定的多。因此，以抽象为基础搭建的架构比以细节为基础的架构要稳定的多。这有助于减少因低层模块变动而对高层模块产生的影响。

**三、应用实例**

以数据读取为例，我们可以定义一个抽象的数据读取接口（如Reader），然后创建多个具体的数据读取实现类（如FileReader、DatabaseReader等）。高层模块（如DataProcessor）可以通过依赖于Reader接口来与不同的数据读取实现类进行交互。当需要添加新的数据读取方式时，只需实现一个新的数据读取类，而无需修改高层模块的代码。

**四、优点**

1. **降低耦合性**：通过抽象接口或类来实现模块间的交互，降低了模块间的耦合性，使得系统更加灵活和可扩展。
2. **提高稳定性**：由于抽象的东西相对稳定，因此基于抽象搭建的架构也相对稳定。这有助于减少因低层模块变动而对高层模块产生的影响。
3. **提高可维护性**：当需要修改或扩展系统功能时，只需修改或添加新的实现类，而无需修改高层模块的代码。这大大提高了系统的可维护性。

**五、总结**

依赖倒转原则是一种重要的面向对象设计原则，它要求我们在代码设计时尽量将具体的实现细节与高层逻辑分离，通过抽象接口或类来实现两者之间的交互。这有助于降低模块间的耦合性、提高系统的稳定性和可维护性。在实际开发中，我们应该积极运用依赖倒转原则来构建更加健壮、灵活和可扩展的系统。

我们所做的实验使用依赖倒转原则，所使用的高层逻辑与抽象接口之前的交互，使用几个类完成。例如：我们使用了相同名称的student id,在不同的类中使用，作为每个类的主键。