软件设计原则

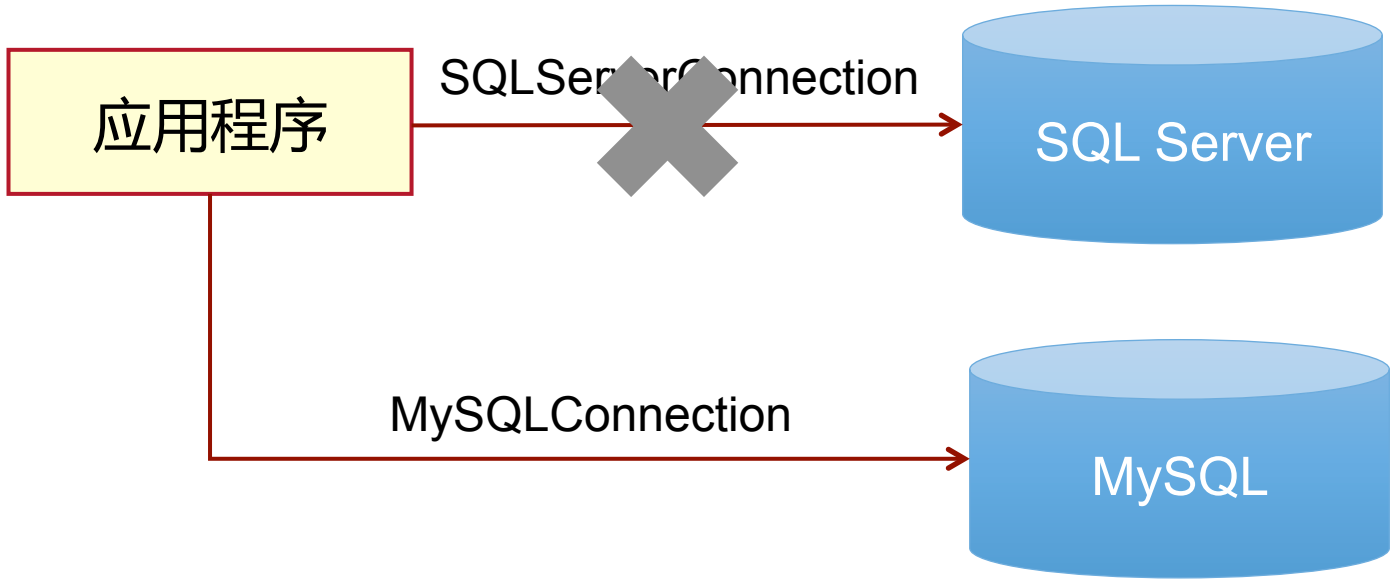
清华大学软件学院 刘强

软件设计原则

设计原则是系统分解和模块设计的基本标准，应用这些原则可以使代码更加灵活、易于维护和扩展。

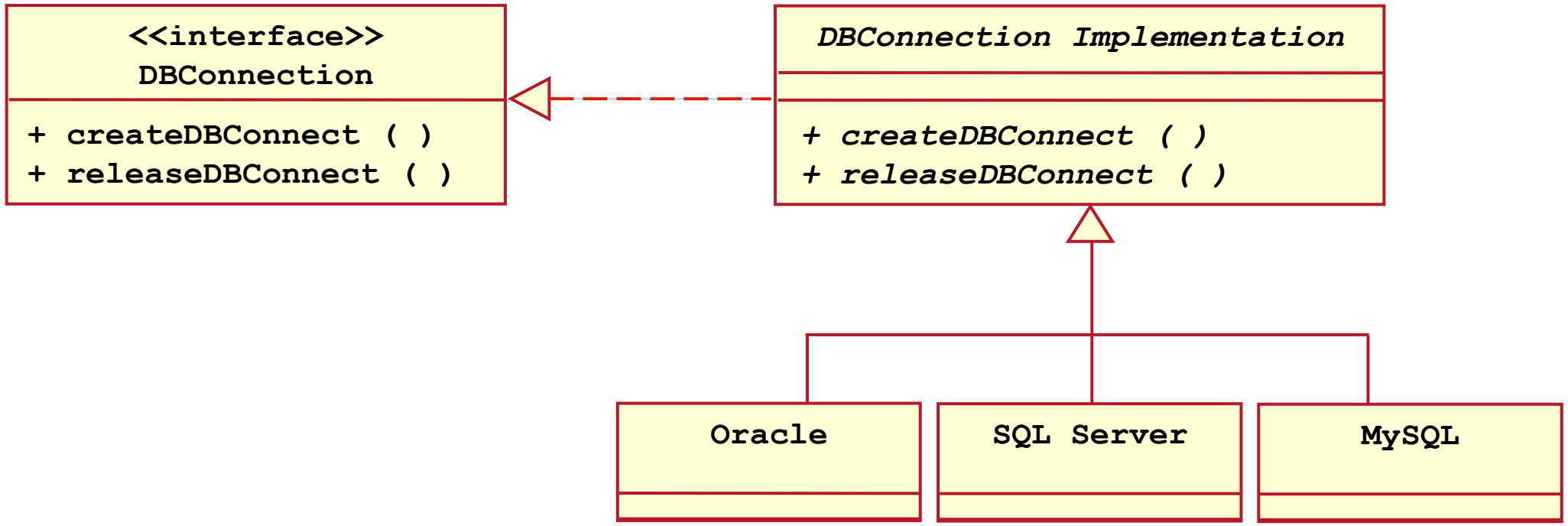
抽象

抽象是关注事物中与问题相关部分而忽略其他无关部分的一种思考方法。



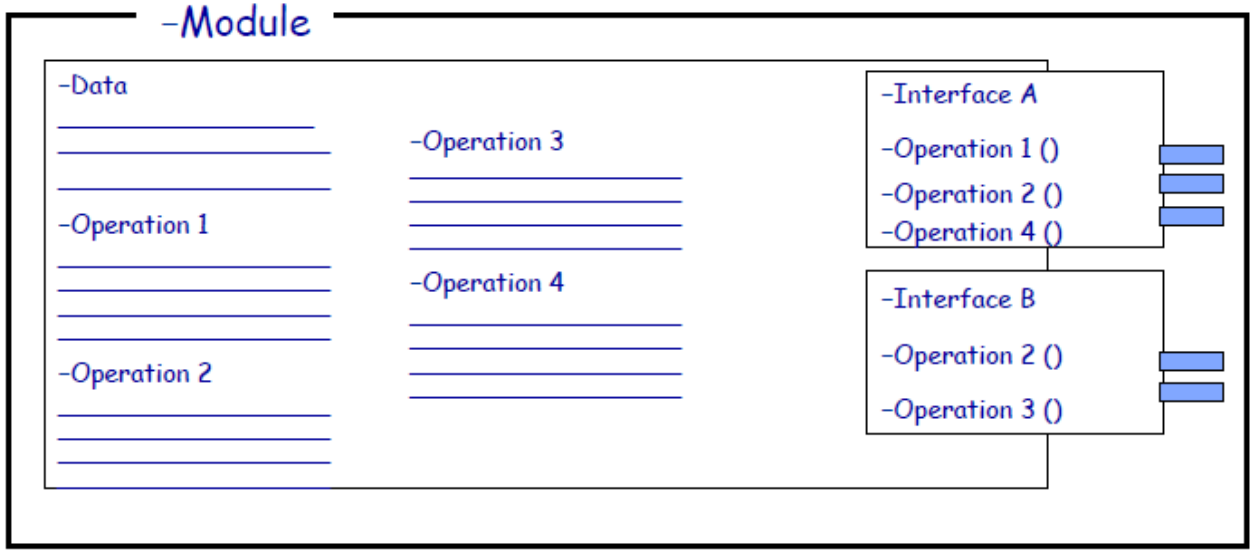
抽象

抽象是关注事物中与问题相关部分而忽略其他无关部分的一种思考方法。



封装

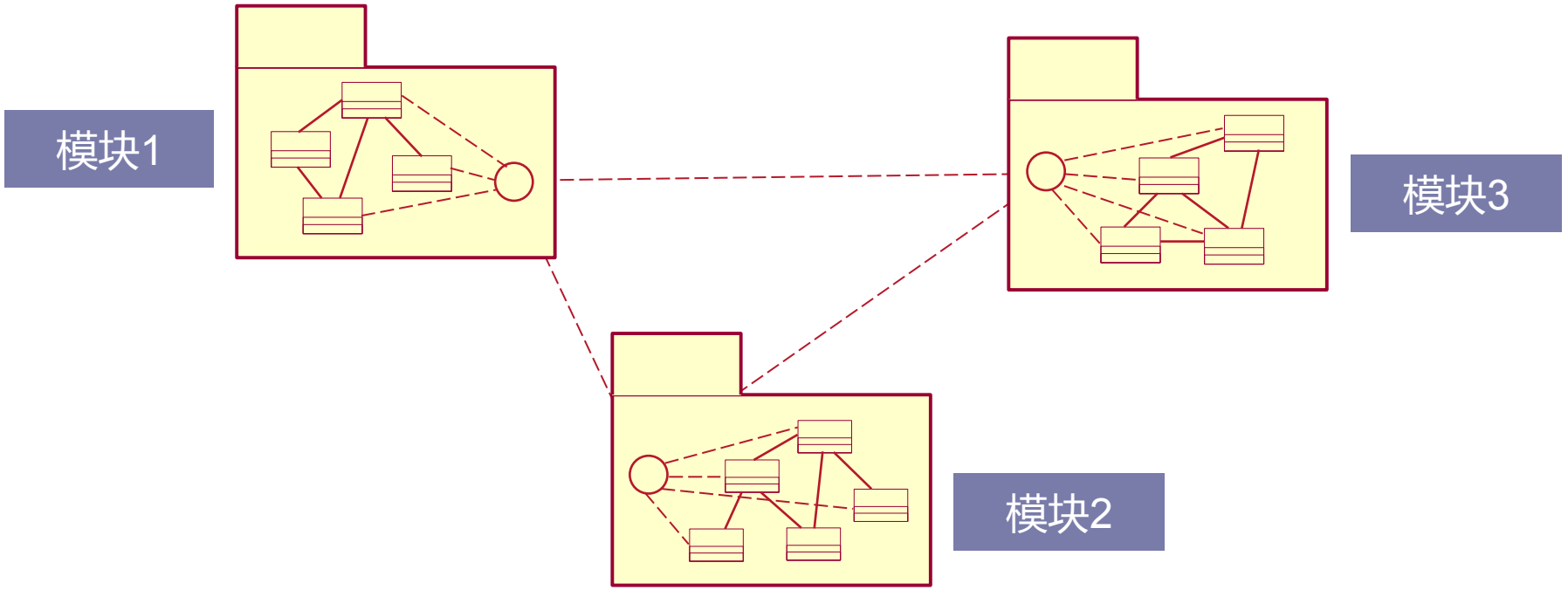
封装和信息隐藏是指每个软件单元对其他所有单元都隐藏自己的设计决策，各个单元的特性通过其外部可见的接口来描述。



要求：应将单元接口设计得尽可能简单，并将单元对于环境的假设和要求降至最低。

模块化

模块化是在逻辑和物理上将整个系统分解成多个更小的部分，其实质是“分而治之”，即将一个复杂问题分解成若干个简单问题，然后逐个解决。



系统分解原则

系统分解的目标：高内聚、低耦合

内聚性是一个模块或子系统内部的依赖程度。

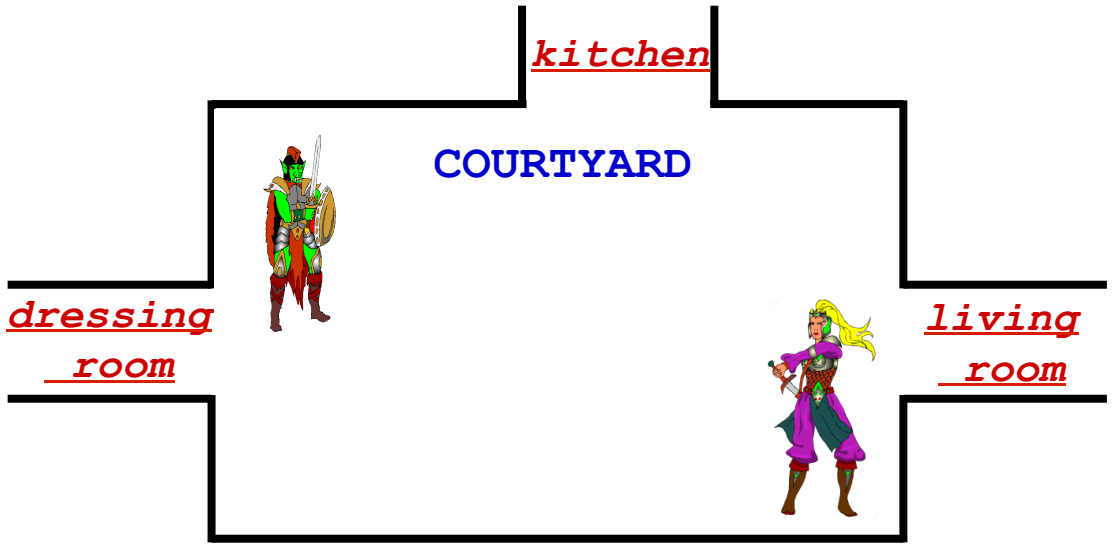
- 如果一个模块或子系统含有许多彼此相关的元素，并且它们执行类似任务，那么其内聚性比较高；如果一个模块或子系统含有许多彼此不相关的元素，其内聚性就比较低。

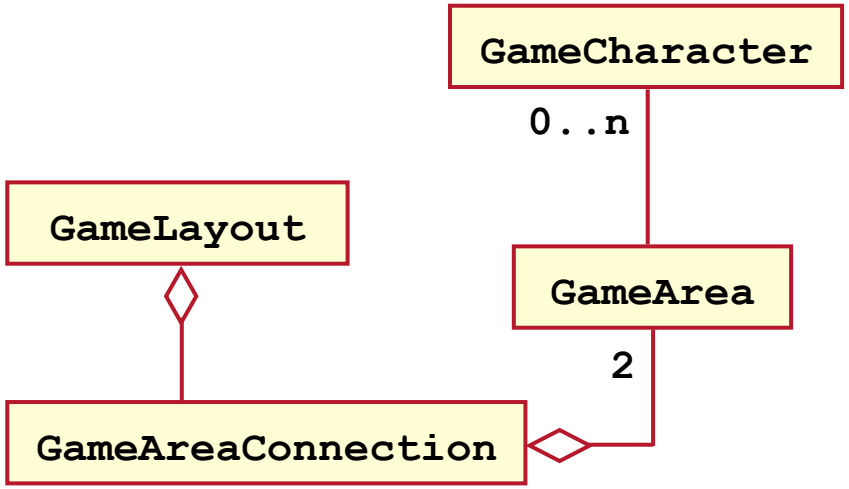
耦合性是两个模块或子系统之间依赖关系的强度。

- 如果两个模块或子系统是松散耦合的，二者相互独立，那么当其中一个发生变化时对另一个产生的影响就很小；如果两个模块或子系统是紧密耦合的，其中一个发生变化就可能对另一个产生较大影响。

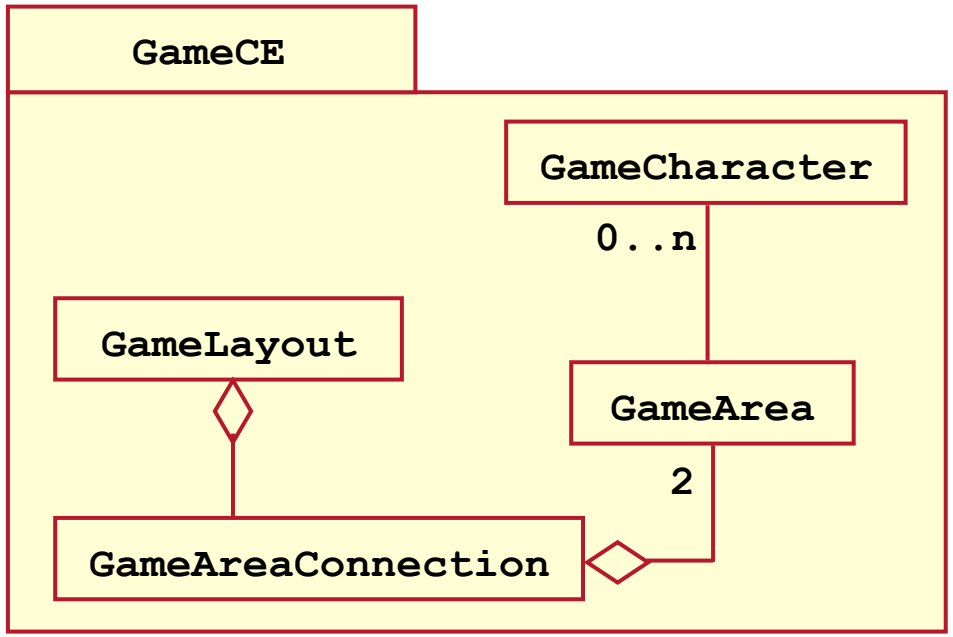
系统分解原则

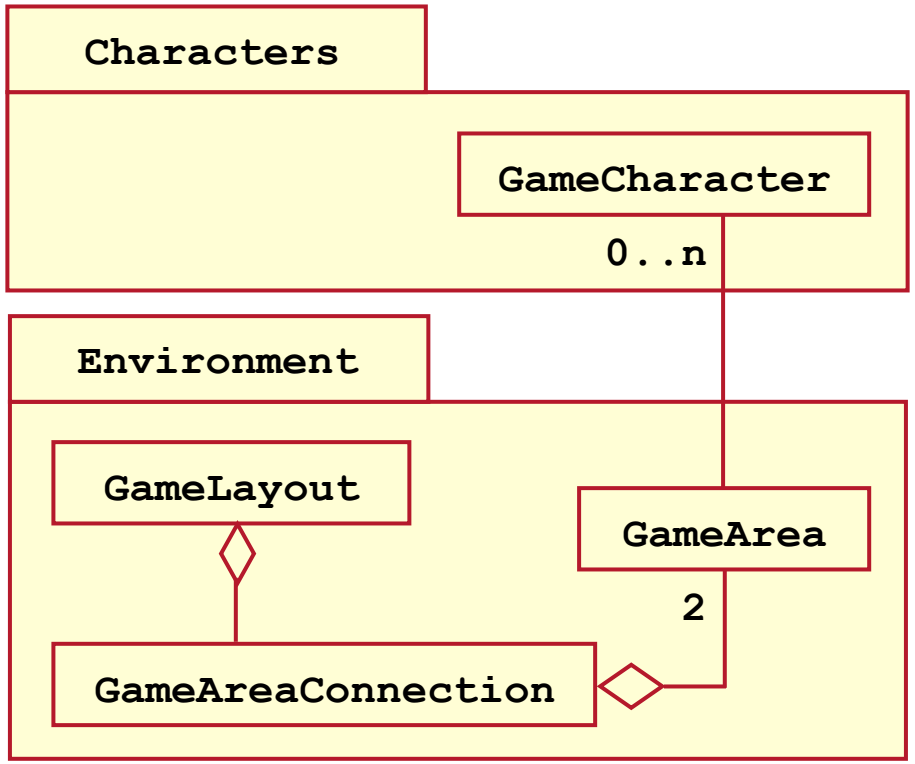
举例：某游戏软件的部分类图如下，其中GameCharacter代表人物，GameLayout代表布局，GameArea 代表一个区域，GameAreaConnection 代表两个区域的连接。





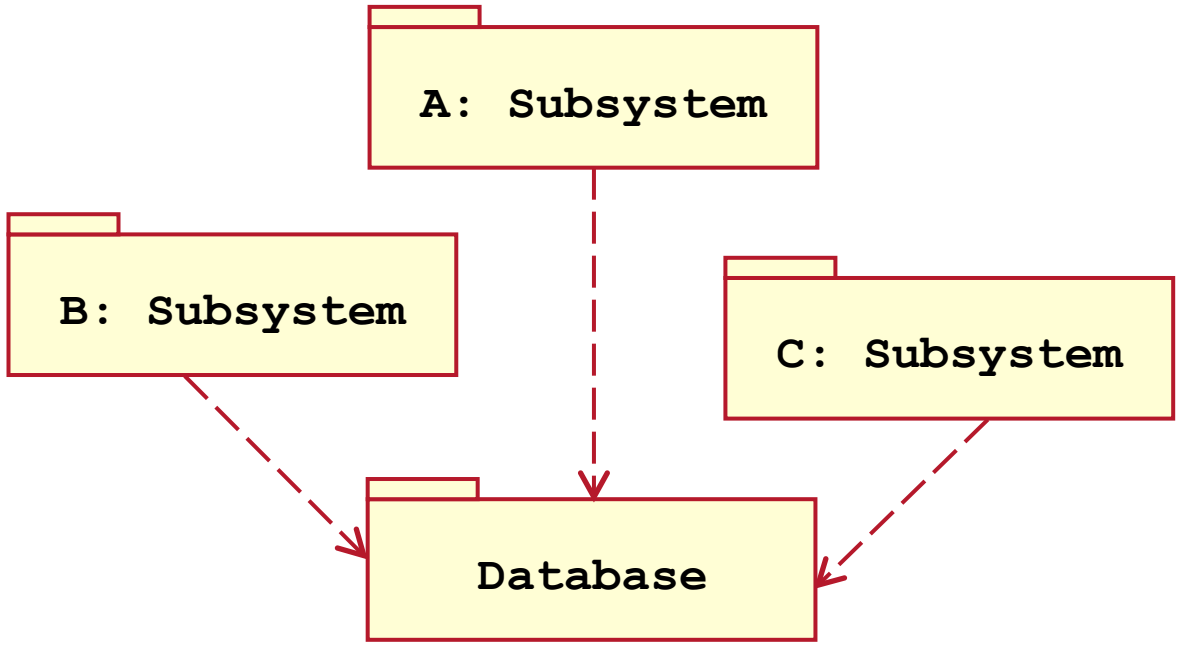
系统分解原则


GameCE
GameCharacter
0..n
GameLayout
GameArea
2
GameAreaConnection


Characters
GameCharacter
0..n
Environment
GameLayout
GameArea
2
GameAreaConnection

系统分解原则

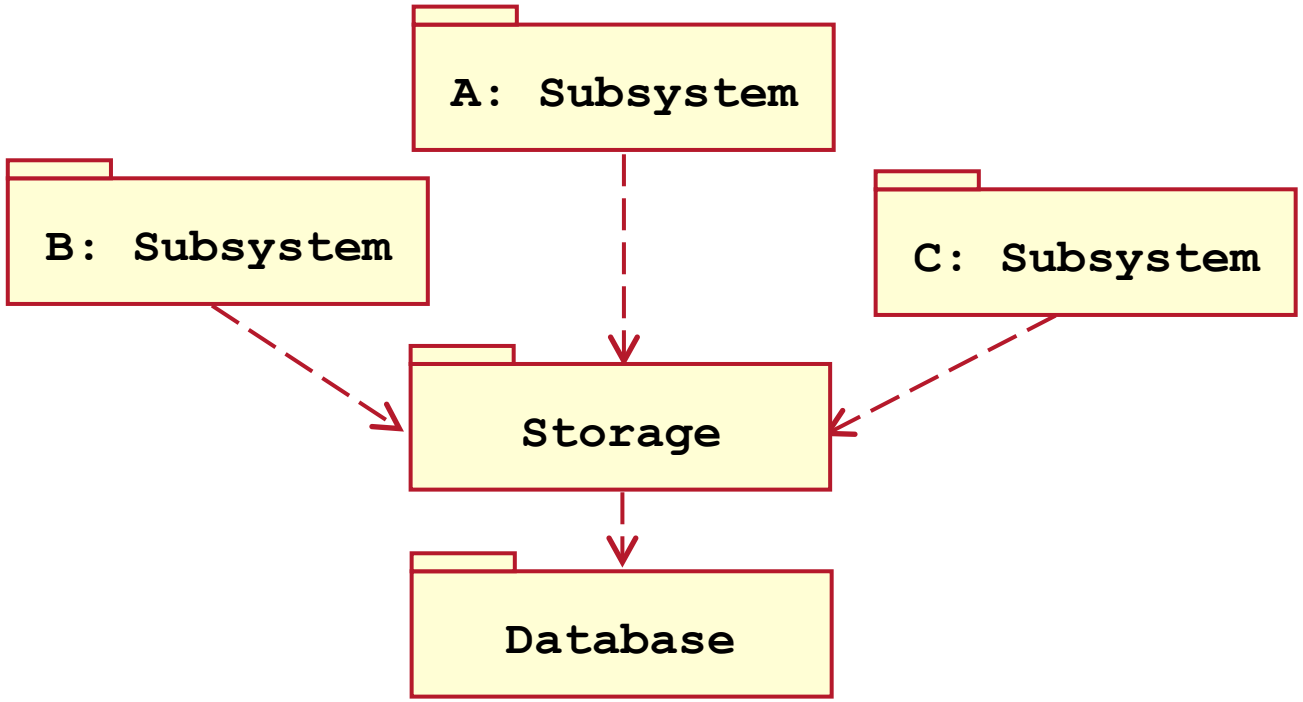
举例：在一个软件系统中，有三个子系统A、B、C都要访问一个关系数据库。



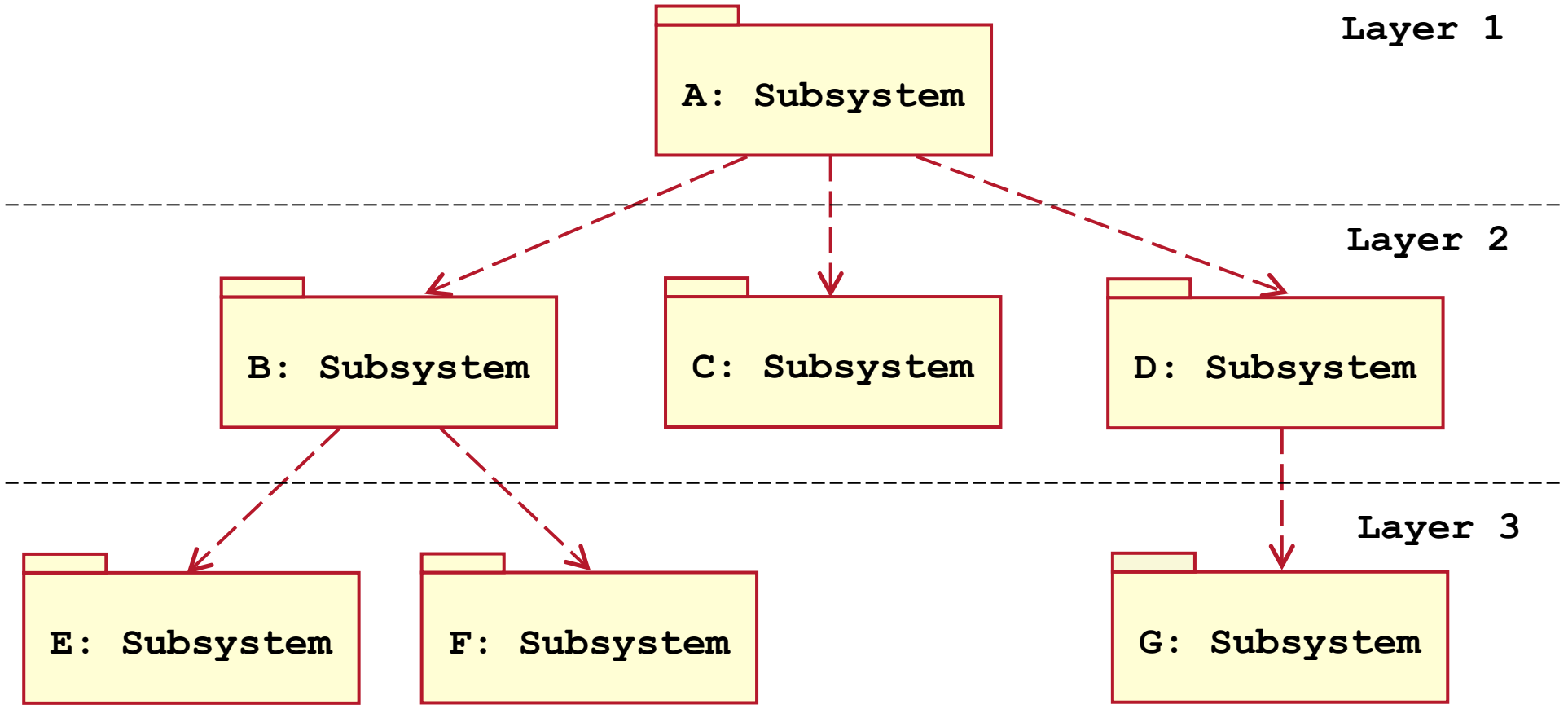
这个方案有什么问题?

系统分解原则

改进：在子系统和数据库之间增加一个存储子系统Storage，从而屏蔽了底层数据库的变化对上层子系统的影响。



层次化


Layer 1
A: Subsystem
Layer 2
B: Subsystem
C: Subsystem
D: Subsystem
Layer 3
E: Subsystem
F: Subsystem
G: Subsystem

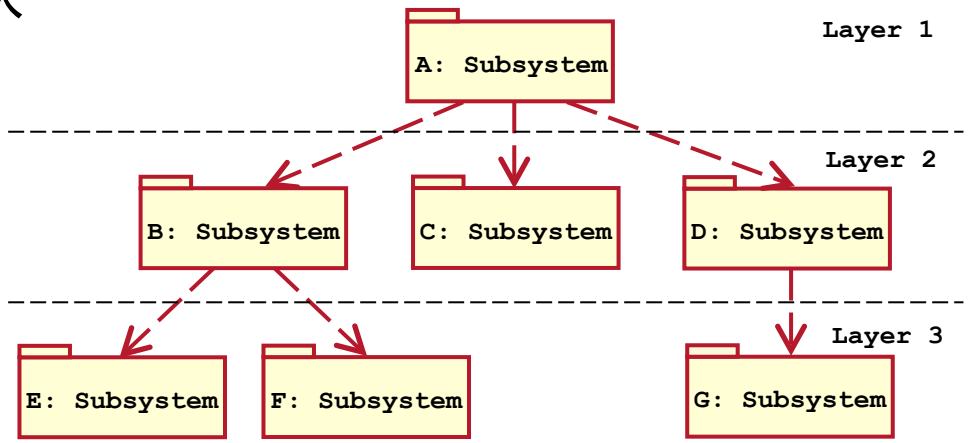
层次化

分层（Layering）

- 每一层可以访问下层，不能访问上层
- 封闭式结构：每一层只能访问与其相邻的下一层
- 开放式结构：每一层还可以访问下面更低的层次
- 层次数目不应超过 7±2 层

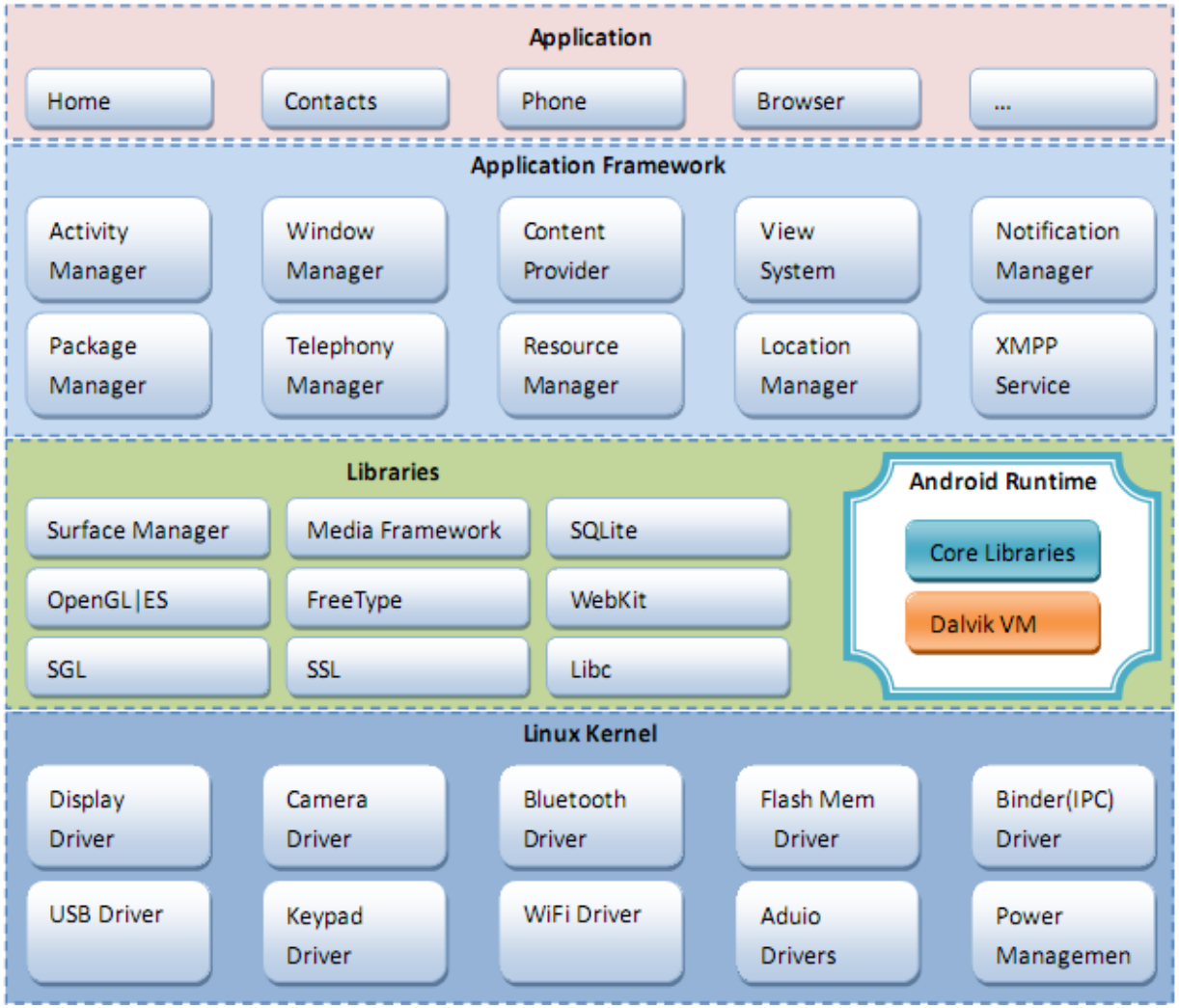
划分（Partitioning）

- 系统被分解成相互对等的若干模块单元
- 每个模块之间依赖较少，可以独立运行



注意：模块单元增加了处理开销，过度分层或划分会增加复杂性。

示例：安卓操作系统层次结构



应用层：运行在虚拟机上的Java应用程序。

应用框架层：支持第三方开发者之间的交互，使其能够通过抽象方式访问所开发的应用程序需要的关键资源。

系统运行库层：为开发者和类似终端设备拥有者提供需要的核心功能。

Linux内核层：提供启动和管理硬件以及Android应用程序的最基本的软件。

复用

复用（Reuse）是利用某些已开发的、对建立新系统有用的软件元素来生成新的软件系统，其好处在于提高生产效率，提高软件质量。

- 源代码复用：对构件库中的源代码构件进行复用
- 软件体系结构复用：对已有的软件体系结构进行复用
- 框架复用：对特定领域中存在的一个公共体系结构及其构件进行复用
- 设计模式：通过为对象协作提供思想和范例来强调方法的复用

谢谢大家！

THANKS