第8章 包图
(Package diagram)

学习目标

- 学习完本章节，要求达到以下状态：
  - 能够说明包图的表示方法和使用方法。
  - 能够读懂包图并理解其中的含义。
  - 能够用包图来对UML元素进行整理分类。

包

- 定义：A **package** is a general-purpose mechanism for organizing elements into groups.
- 包的作用：
  - 用于组织模型中的元素以便更容易理解；
  - 对包中元素进行可见性控制。

说明：

- 包名分simple name和path name两种形式。

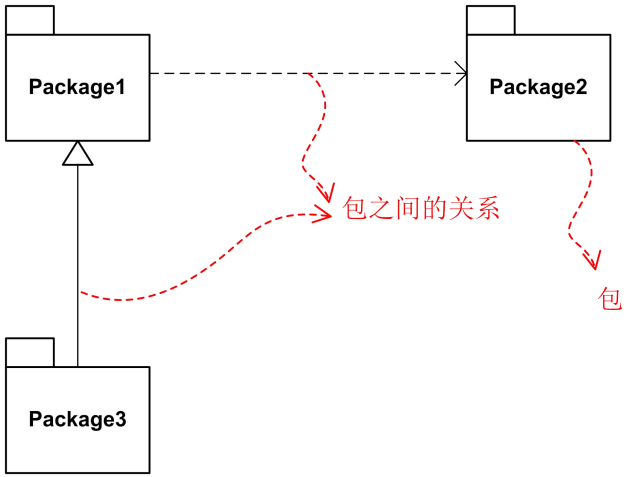
例：

  - Camera
  - Sensors::Vision::Camera

- 包中可以包含其它建模元素，如class, interface, component, node, use case, package, … , 等。
- 包可以嵌套，但嵌套层次不要过深。
- 包没有实例，即在系统运行时见不到包。

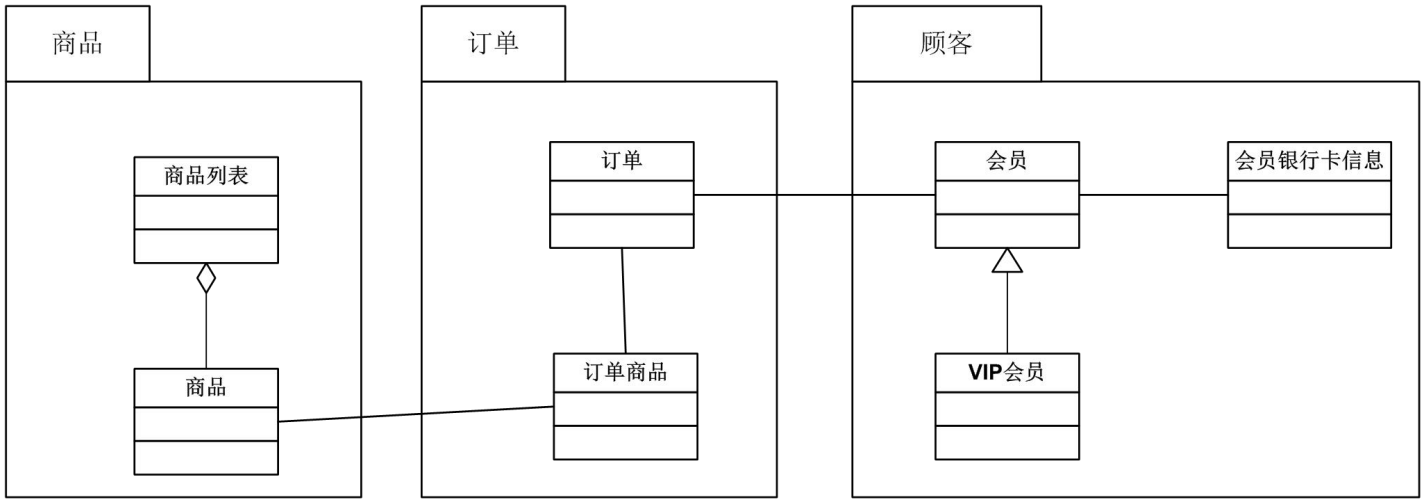
包图的概要

- 对UML的模型元素按照意义，功能等进行分组整理的时候使用。
- 通过对UML的模型元素进行分组整理，可以提高模型的可读性和可维护性。
- 包图可适用于UML的所有模型元素。
- 包图内部还可以再含有包。
- 包图的组成元素：
  - 包(package)
  - 包之间的关系(依赖，泛化)



用包对类图进行分组整理的例子

- 看一个用包对类图中的类进行分组整理的例子：

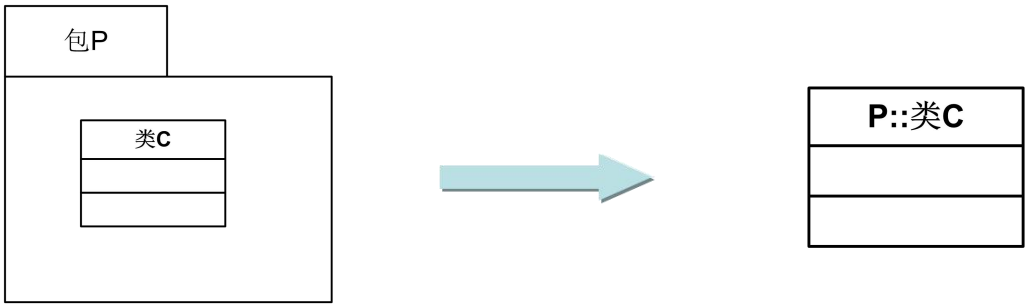


把类图中的类根据其用途划分为三个包–“顾客”包，“订单”包和“商品”包。

用包名来修饰类名

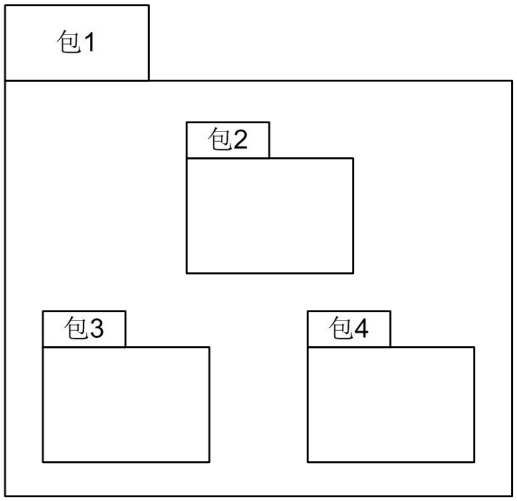
◆ 当需要明确一个类和其所属的包之间的所属关系时，可用如下方式来对类名进行修饰：

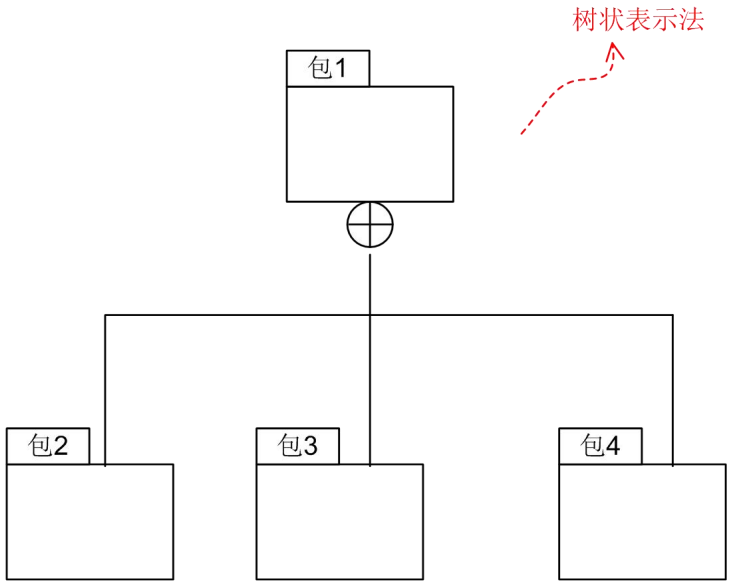
包名::类名



包的层次化

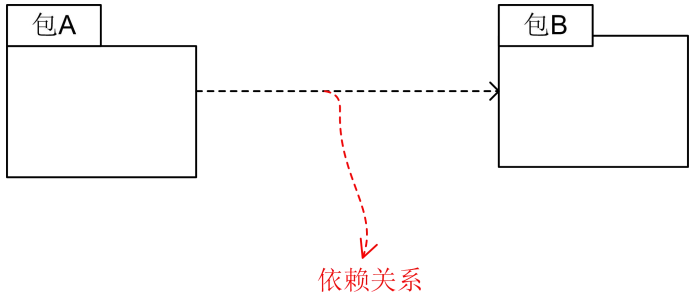
- 包中还可以含有包，把这种情况称为包的层次化。
- 包的层次化表现有两种方式：

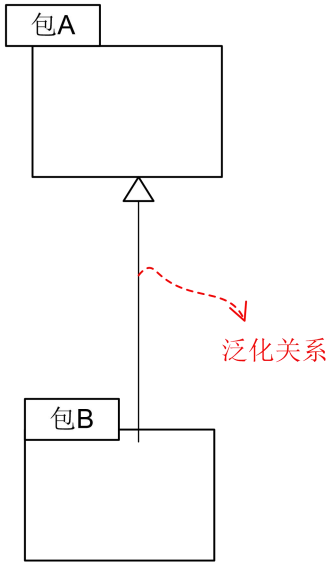




包之间的关系

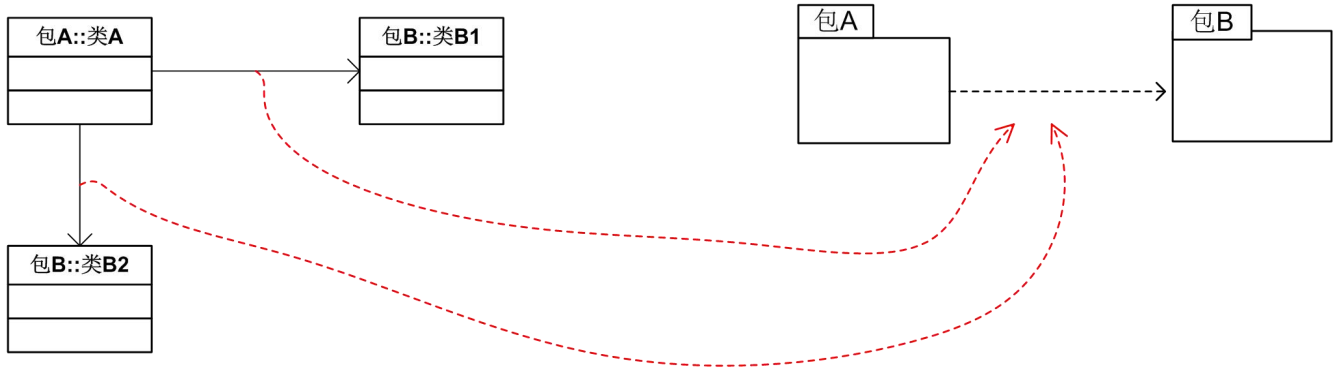
- 包之间可以具有依赖关系或者泛化关系。
- 包之间的关系取决于包内的元素之间的关系。
  - 两个包内的元素见具有使用上关系时，这两个包之间具有依赖关系。
  - 两个包内的元素具有泛化关系时，这两个包之间具有泛化关系。



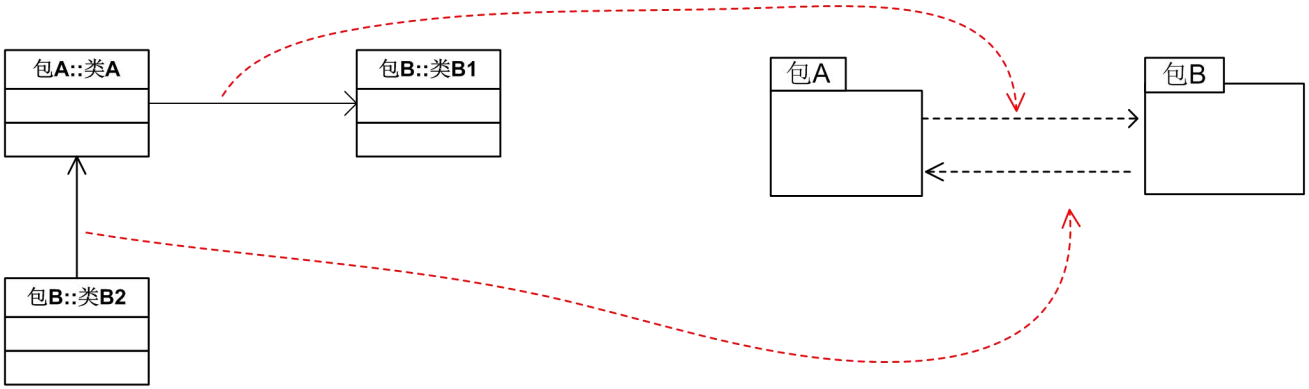


包之间依赖关系(单向)

- 当两个包内的元素只有单向的使用关系时，两个包之间具有单向依赖关系。



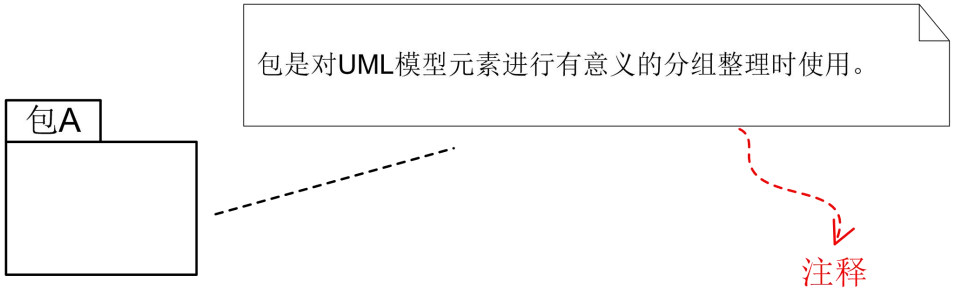
- 当两个包内的元素具有互相使用的关系时，两个包之间具有双向依赖关系。



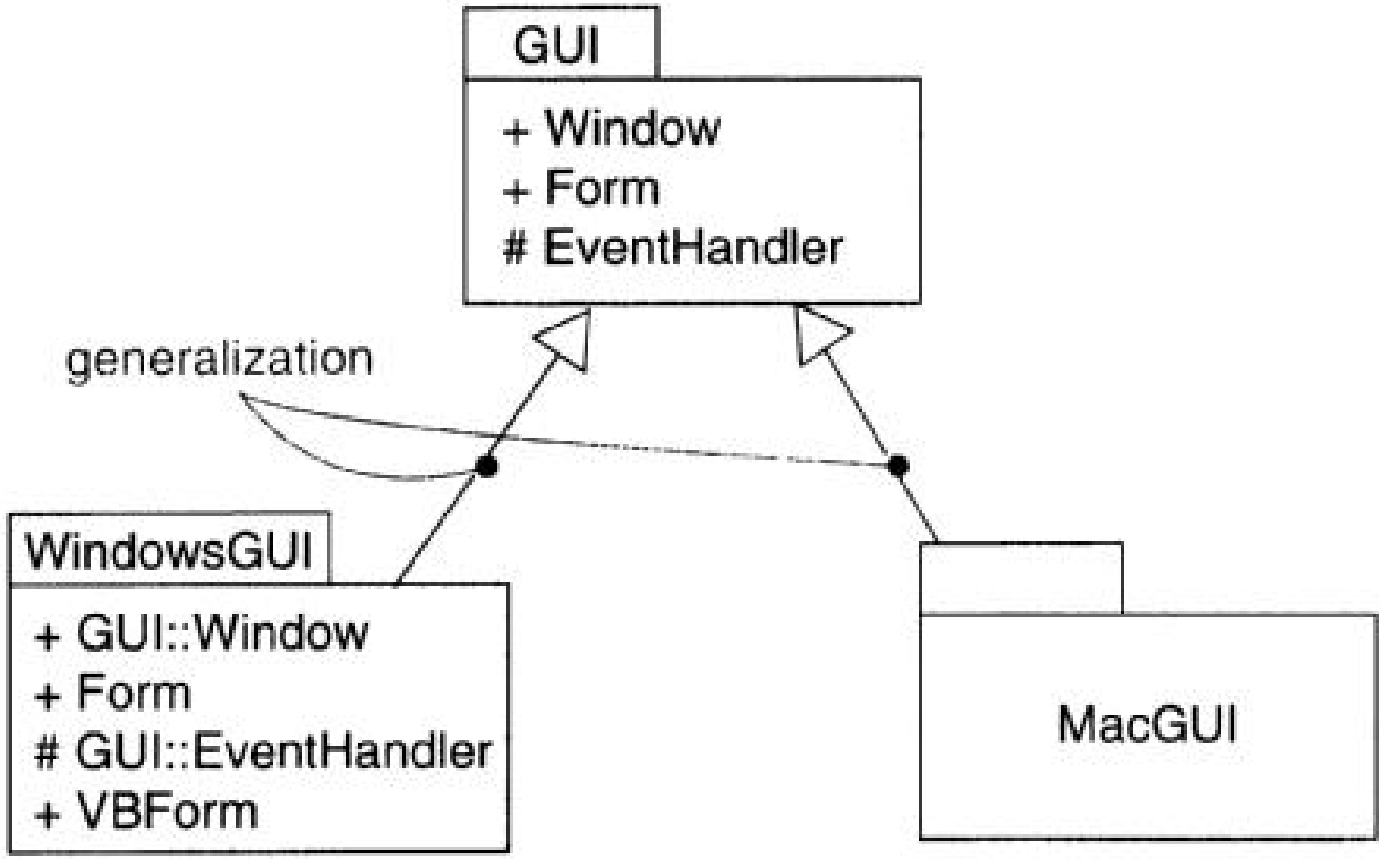
包之间的双向依赖关系会造成包维护修改上的循环影响，在划分包的时候应尽可能避免。

注释(Note)

◆ 注释(Note)，是对UML的模型元素进行解释，约束，或者添加补充信息时使用。
◆ 注释(Note)和包(package)在UML1.5中都是作为UML的公用机制出现的，顾名思义，即可用于UML的所有模型元素上。
◆ 注释(Note)用带折角的矩形框来表示。



- 子系统概念：版型为<<subsystem>>的包。
- 包之间的泛化关系。例：



说明：包之间的泛化关系很少用到，Rose 2003中也不支持。

包的应用

- 对建模元素进行分组。
- 在Rose中，包可以提供一些特殊的功能，如
  - 在数据建模中，用包表示模式和域包；在数据模型和对象模型之间的转换是以包为单位进行的；
  - 在Web建模中，包可以表示某一虚拟目录(virtual directory)，在该目录下的所有web元素都在这个包中；
  - 包在Rose中还可以作为控制单元(controlled unit)，以方便团队开发和配置管理。

设计包的原则

- 在建立包时，应把概念上和语义上相近的模型元素纳入一个包。好的设计要求体现高内聚、低耦合的特性。在设计包时，应遵循以下一些原则：
  - 重用等价原则
  - 共同闭包原则
  - 共同重用原则
  - 非循环依赖原则

- 重用等价原则(Reuse Equivalency Principle, REP) 指的是把类放入包中时，应考虑把包作为可重用的单元。
- 这种设计原则和用户的使用心理有关，对于可重用的类，其开发可能比较快，开发人员会不断地推出这些可重用类的升级版本，但对于可重用类的使用者来说，不会随着可重用类的每次升级而修改自己的系统，不过，在需要升级的时候又会要求很容易地用新版本的可重用类替换旧版本的可重用类，因此设计包的一个原则是把类放在包中时要方便重用，方便对这个包的各个版本的管理。

- 共同闭包原则(Common Closure Principle, CCP) 指的是把那些需要同时改变的类放在一个包中。
- 例如，如果一个类的行为和/或结构的改变要求另一个类作相应的改变，则这两个类应放在一个包中；或者在删除了一个类后，另一个类变成多余的，则这两个类应放在一个包中；或者两个类之间有大量的消息发送，则这两个类也应放在一个包中。
- 共同闭包原则就是要提高包的内聚性、降低包与包之间的耦合度，希望在改动或升级一个包的时候要尽量少影响别的包。

- 共同重用原则(Common Reuse Principle, CRP)指的是不会一起使用的类不要放在同一包中。
- 这个原则和包的依赖关系有关。如果元素A依赖于包P中的某个元素，则表示A会依赖于P中的所有元素。也就是说，包P中的任何一个元素做了修改，则都要检查元素A是否还能使用包P，即便所修改的元素和A完全没有关系。
- 修改了一个对用户实际上毫无影响的类，却逼得用户不得不重新检查是否还可以同样方式使用新的包是不合理的。

说明：

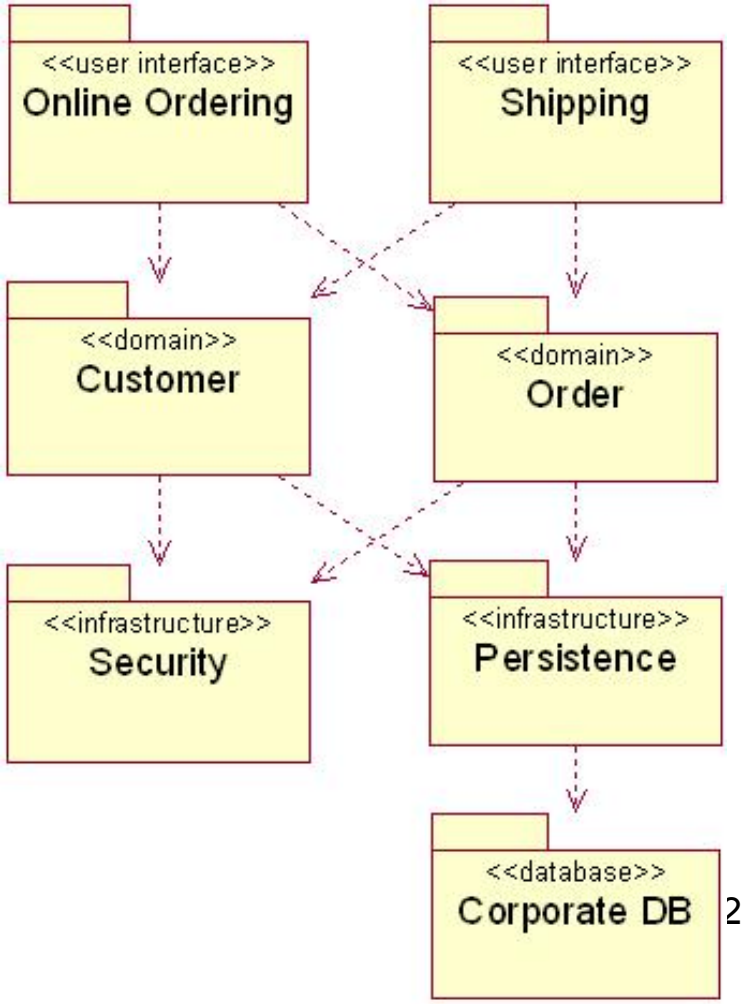
- 重用等价原则、共同闭包原则、共同重用原则这3个原则事实上是相互排斥的，不可能同时被满足，它们是从不同使用者的角度提出的，重用等价原则和共同重用原则是从重用人员的角度考虑的，而共同闭包原则是从维护人员的角度考虑的。
- 共同闭包原则希望包越大越好，而共同重用原则却要求包越小越好。
- 在开发的早期，可以共同闭包原则为主，而当系统稳定后，可以对包做一些重构，这时要以重用等价原则和共同重用原则为主。

- 非循环依赖原则(Acyclic Dependencies Principle, ADP) 指的是包之间的依赖关系不要形成循环。也就是说不要有包A依赖于包B，包B依赖于包C，而包C又依赖于包A这样的情况出现，如果确实无法避免出现包之间的循环依赖，则可以把这些有循环依赖关系的包放在一个更大的包中，以消除这种循环依赖关系。

包图建模风格

- 建模风格1：使包具有内聚性。
  - 放在包中的任何东西，当和包中的其余内容一起考虑时应该有意义。
  - 一个好的判断包是否具有内聚性的方法是看能否给包起一个简短的、描述性的名字，如果不能，则很可能把几个不相关的东西放在一个包中了。

- 建模风格2：垂直分层类包图。
  - 包位置的放置反映了系统体系结构的逻辑分层。按照惯例，一般是采用自上而下的方式画分层结构。
  - 例：用户界面和领域类交互，而领域类又会使用基础结构类，某些基础结构类又会存取数据库。



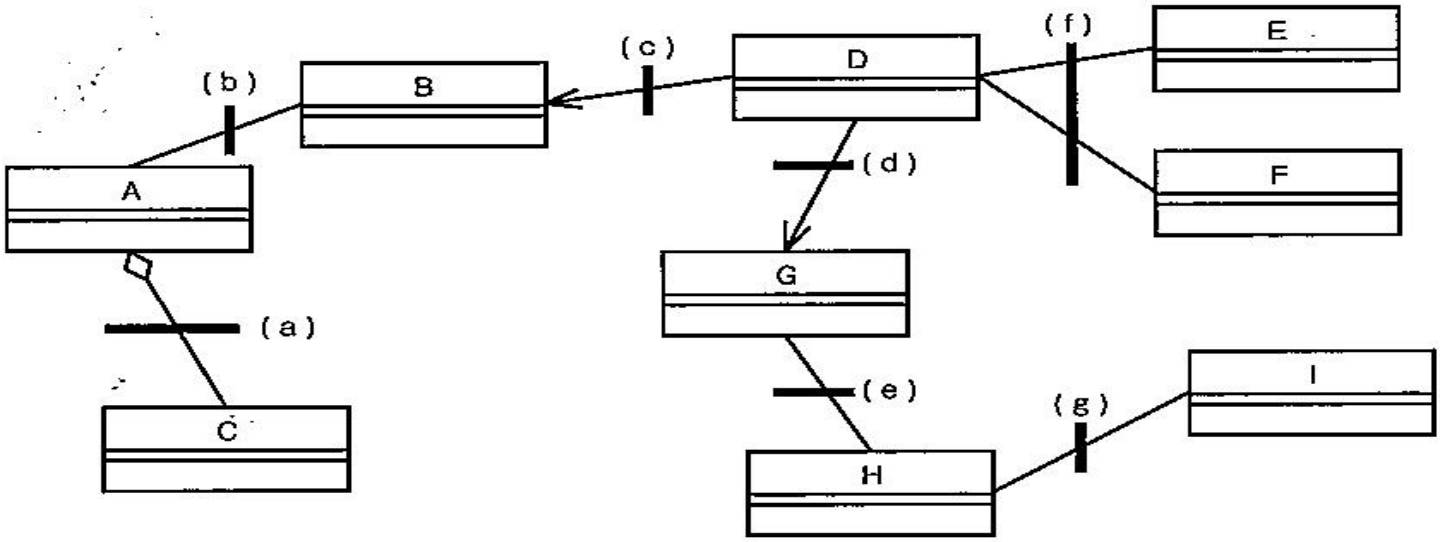
- 建模风格3：在包上用版型指明体系结构的层次。
  - 把设计按照体系结构的层次进行组织是非常常见的，比如用户界面层、业务/领域层、持久/数据层、基础设施/系统层等进行组织。
  - 例：可以用版型<<user interface>>、<<domain>>、<<infrastructure>>和<<database>>等表示不同层的包。

总结

◆ 包图是用来对UML模型元素进行有意义的分组整理时使用，可提高模型的可读性和可维护性。
◆ 注释是对UML模型元素进行解释，添加补充信息时使用。同包一样，属于UML中的共有机制，可使用在任何UML元素上。
◆ 包和包之间的关系具有依赖关系和泛化关系两种。取决于包内元素之间的关系。
◆ 在进行分包的时候，要尽量避免包之间的双向依赖关系。

例题1

要将下面的类图分割成3个包。从下列选项中选择合适的分割位置。

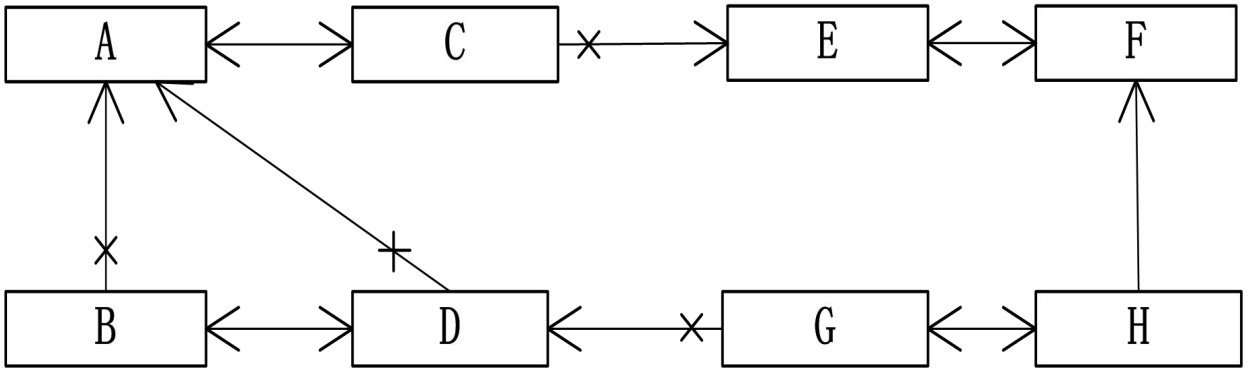


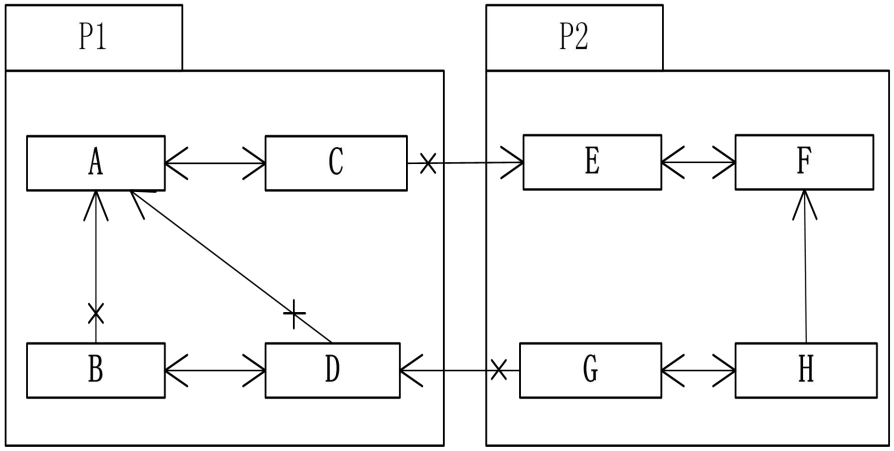
(1) a　(2) b　(3) c　(4) d　(5) e　(6) f　(7) g

。

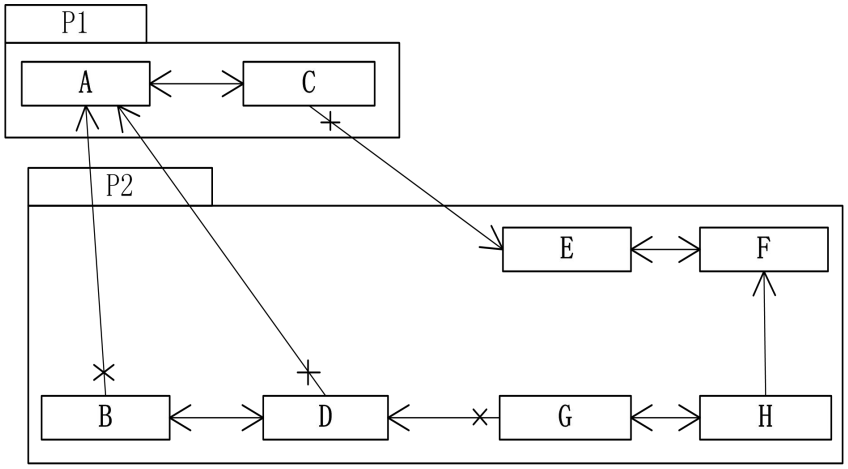
例题2

考虑对下面的类图分割包。请从选项中选出2个不恰当的选项

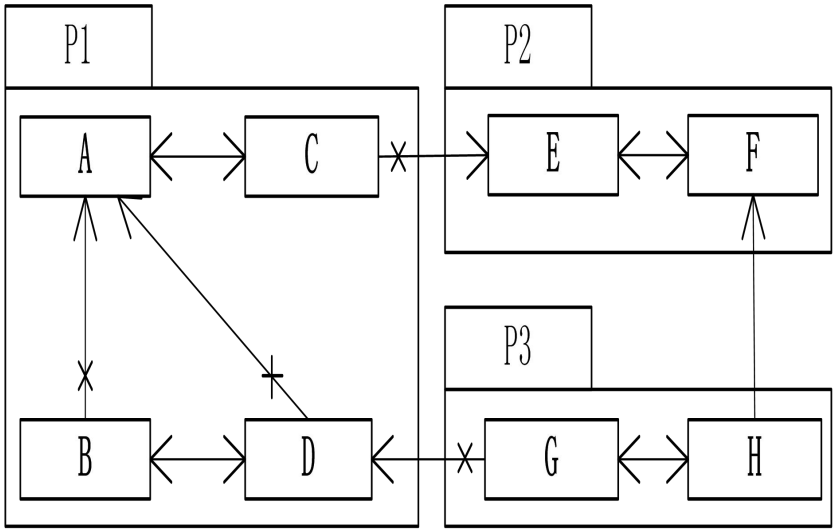




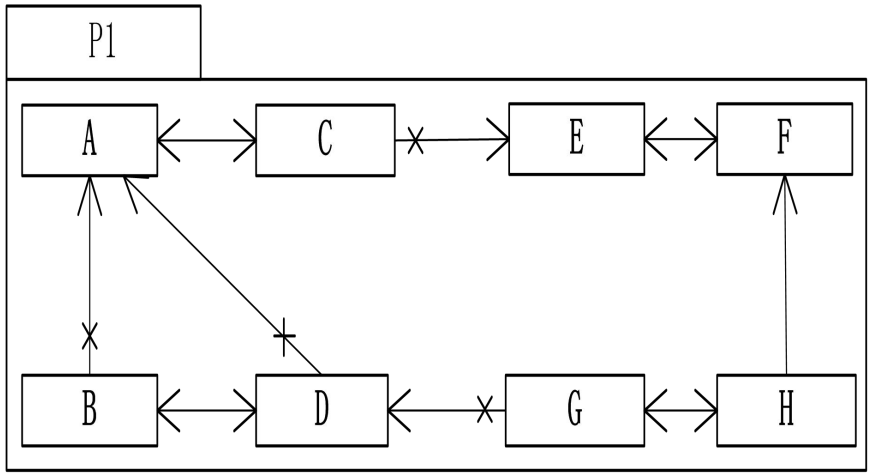
A



B



C



D

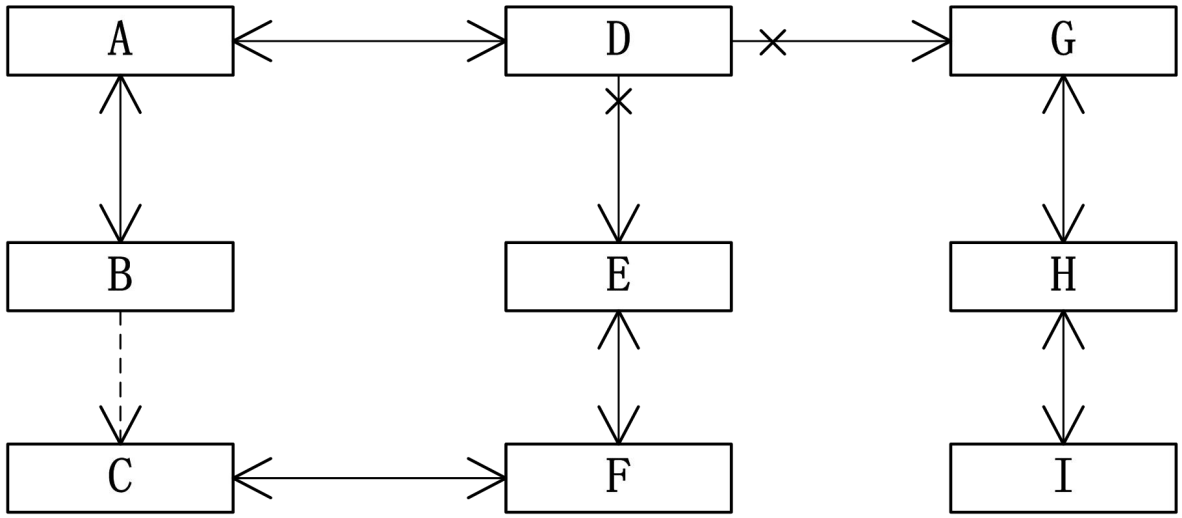
例题2

原则：
分割包的时候，注意以下几点。
1)包与包之间的依赖关系尽可能少
2)避免双向依赖关系
3)包内的类不能过多（100个以下）

A、B中，包之间的依赖关系是双向的，特别是B的依赖关系比A还多，所以是不恰当的。C没有双向依赖关系，包之间的依赖关系只有一个，所以是恰当的。D只有一个包。只有8个类，可以说不必非要分割包。所以，D也是恰当的。

例题3

把下面类图划分到3个包中。请选出最适合的类分组的组合 。



A. （A类、B类、C类、D类）、（E类、F类）、（G类、H类、I类）
B. （A类、B类、D类）、（C类、E类、F类）、（G类、H类、I类）
C. （A类、B类、D类、E类）、（C类、F类）、（G类、H类、I类）
D. （A类、B类、C类）、（D类、E类、F类）、（G类、H类、I类）

例题3

原则：

分割包时，应该尽量从类之间的关系薄弱的地方分开。在关联数少、单向关联的地方划分。