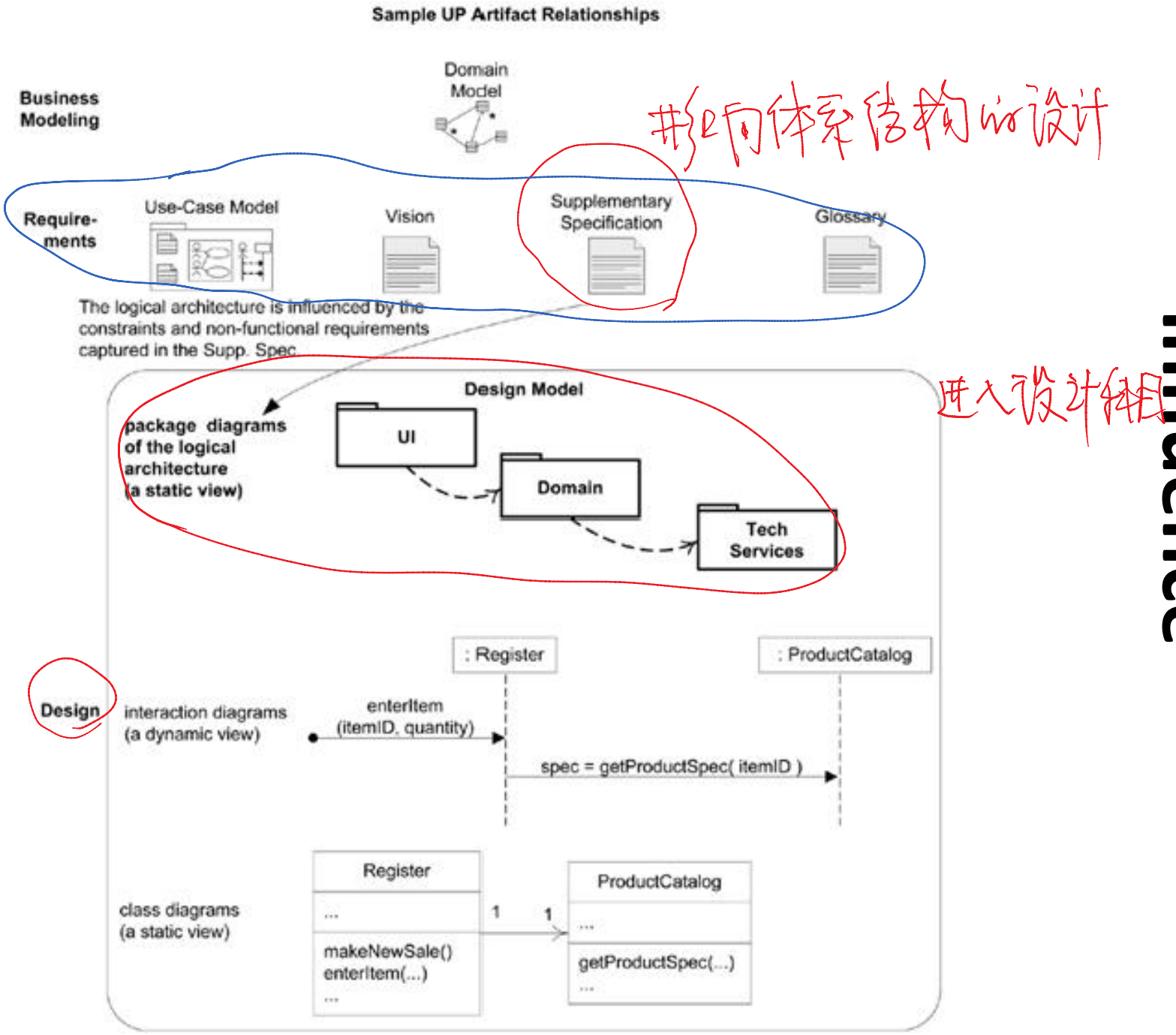
System Analysis and Design

L13. Logical Architecture

Topics

- What is the Logical Architecture?
- Layered Logical Architecture
- Design with Layers
- Model-View Separation
- UML Package Diagrams

Sample UP artifact influence



软件体系结构

Software Architecture

- Decisions about the organization of a software system 决定软件系统的组成
- Selection of **structural elements** and their **interfaces**, along with their **behavior** 选择结构元素及其接口 行为
- **Organization** of elements into progressively larger systems 结构元素组成了更大的系统

结构性元素组成更大的系统！

软件设计首先要考虑软件的逻辑体系结构.

Finally… Software Design

What is the Logical Architecture?

- The logical architecture is the large-scale organization of the classes into packages (namespaces), subsystems, and layers.
- No decision about how they're deployed

逻辑组成: 包、子系统、层

但没有讨论具体的部署方法.

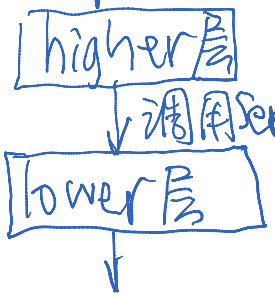
结构性元素: 类、包、子系统、层、…

逻辑架构: 结构性元素的组织结构. (不涉及部署)

Layered Logical Architecture

分层的逻辑架构

- A **layer** is a very coarse-grained grouping of classes, packages, or subsystems that has cohesive responsibility for a major aspect of the system. 一个层次是一个在一个侧面内强职责的组成（职责的划分）
- Also, layers are organized such that "higher" layers (such as the UI layer) call upon services of "lower" layers, but not normally vice versa.

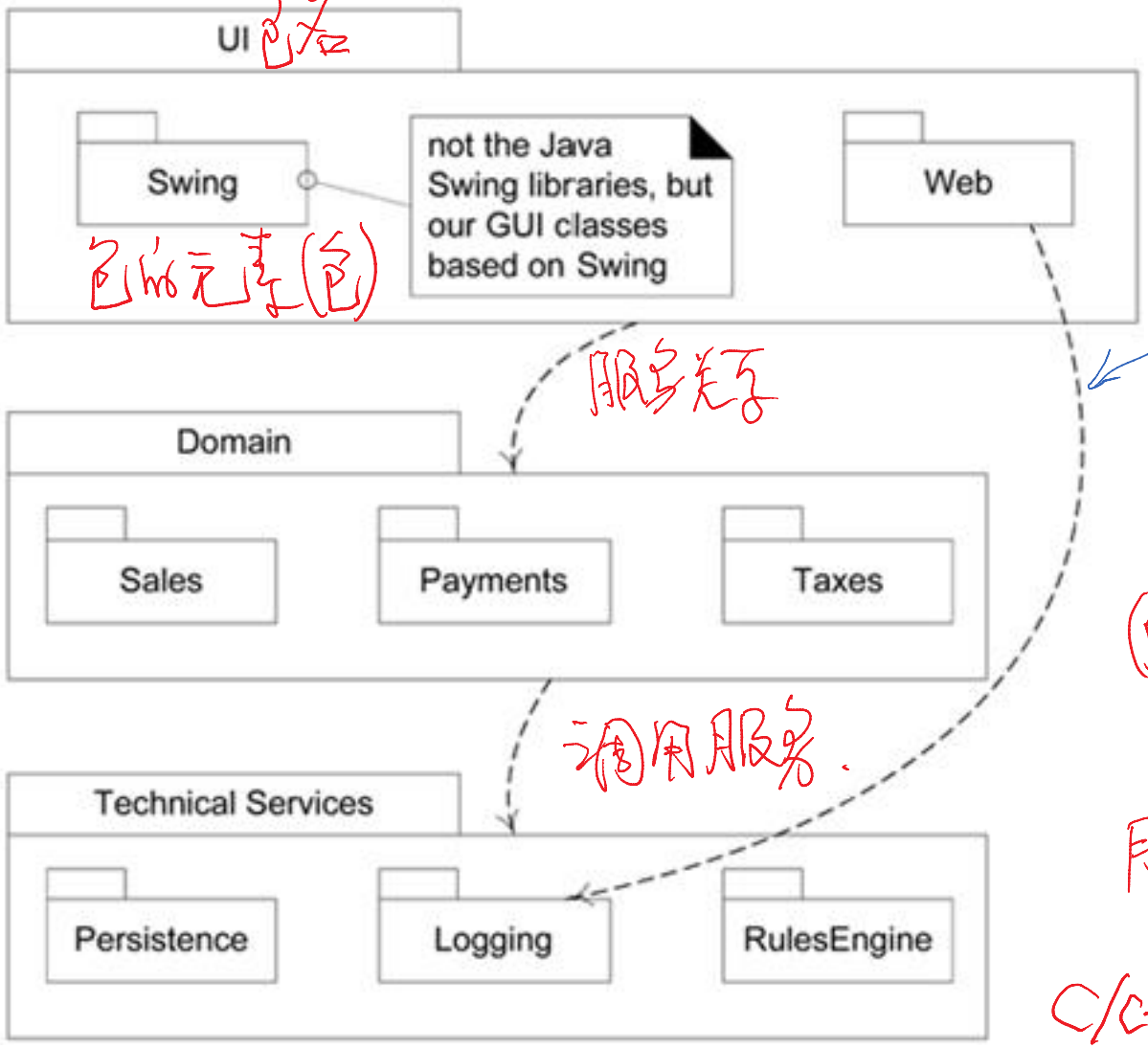


层间的服务关系（单向）（简单的层次结构）

Typically Layers in an OO System

- **User Interface**.
- **Application Logic and Domain Objects** software objects representing domain concepts (for example, a software class *Sale*) that fulfill application requirements, such as calculating a sale total.
- **Technical Services** general purpose objects and subsystems that provide supporting technical services, such as interfacing with a database or error logging. These services are usually application-independent and reusable across several systems.

Layers Shown with UML Package Diagram Notation



Strict and Relaxed Layered Architectures

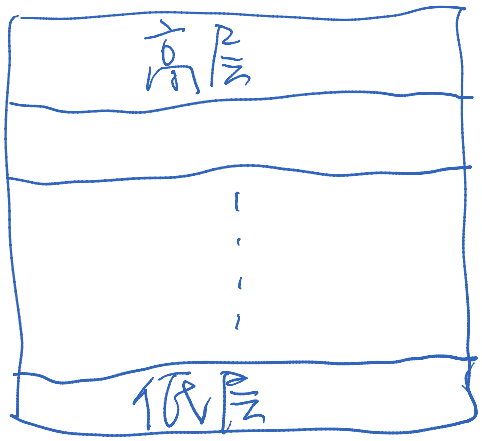
- **Strict layered architecture:** a layer only calls upon the services of the layer directly below it.
  - This design is common in network protocol stacks, but not in information systems
- **Relaxed layered architecture:** a higher layer calls upon several lower layers.
  - For example, the UI layer may call upon its directly subordinate application logic layer, and also upon elements of a lower technical service layer, for logging and so forth.
- A logical architecture doesn't have to be organized in layers. But it's *very* common.

用分层进行设计

Design with Layers

- Organize the logical structure into distinct layers with related responsibilities, clean separation from lower layers. 清晰的分层，各层有自己职责
- Higher layers are more application-specific

高层更依赖于应用



①按职责清晰分层（*** 按职责进行进行分层）
与高层更与依赖有关

Using Layers Helps Address Several Problems

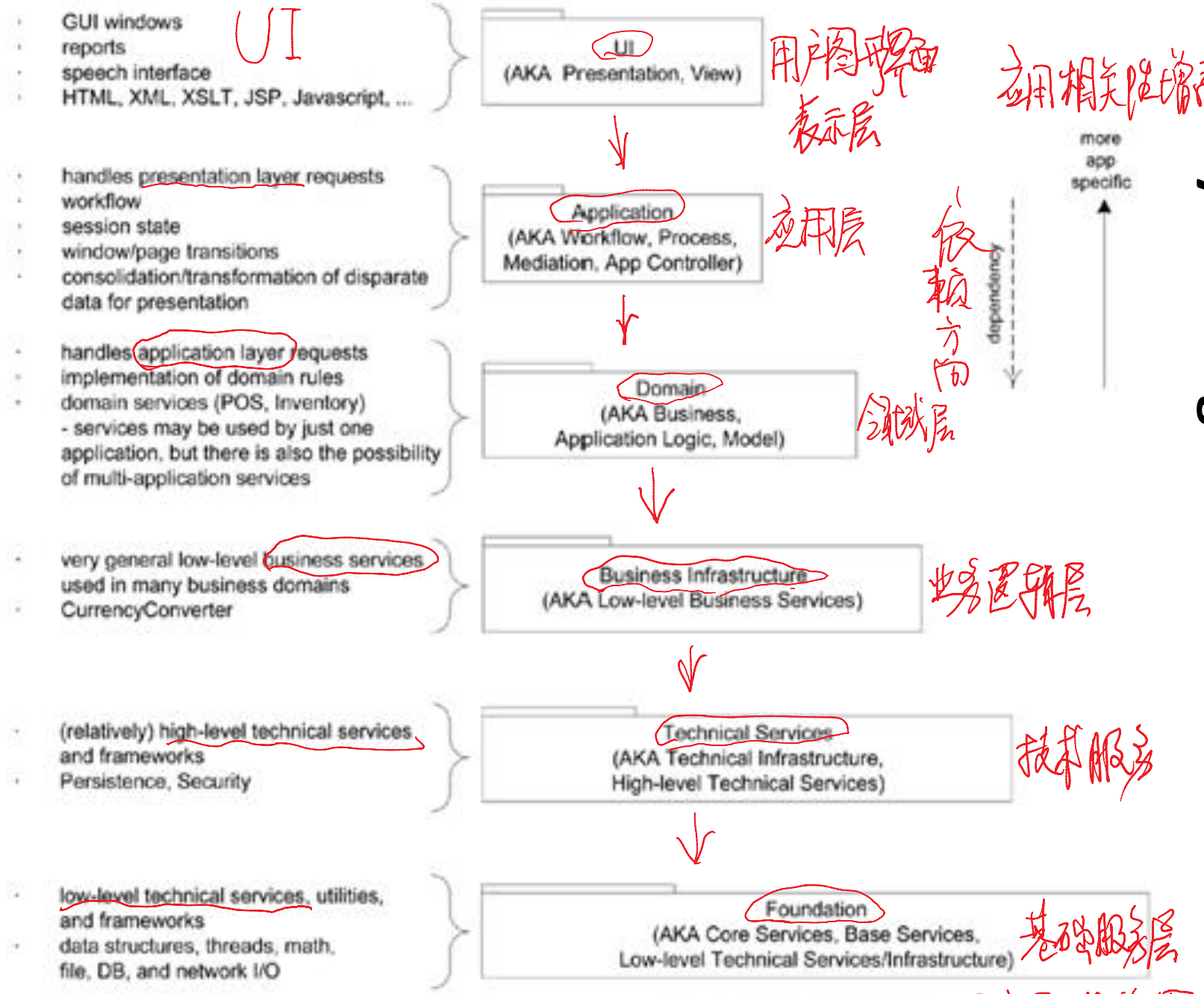
分层有利于如下问题的解决

- Source code changes ripple through the entire system 源码的改变会影响到整个系统
- Application logic can't be reused because it is intertwined with the UI 因与界面关联在一起，应用逻辑不能重用
- General business logic is intertwined with application-specific logic 一般性(通用性)逻辑与应用专门逻辑交织在一起
- High coupling across different areas 不同部分高度耦合

Common Layers in an Information System Logical Architecture

- **GUI**
- **Application**: handles presentation layer requests, workflow, session state, window/page transitions
- **Domain**: app layer requests, domain rules, domain services
- **Business infrastructure**
- **Tech services**
- **Foundation**: threads, math, network I/O

Common layers in an information system logical architecture



Benefits of Using Layers

- Separation of high from low-level services.
  - reduces coupling and dependencies,
  - improves cohesion,
  - increases reuse potential, and
  - increases clarity.
- Related complexity is encapsulated and decomposable.
- Some layers can be replaced with new implementations.
  - This is generally not possible for lower-level Technical Service or Foundation layers (e.g., *java.util*),
  - but may be possible for UI, Application, and Domain layers.
- Lower layers contain reusable functions.
- Some layers (primarily Domain and Services) can be distributed.
- Development by teams is aided because of the logical segmentation.

Cohesive Responsibilities
Maintain a Separation of Concerns

- The responsibilities of the objects
  - in a layer should be strongly related to each other and
  - should not be mixed with responsibilities of other layers.

Mapping Code Organization to Layers and UML Packages (Java例)

代码按层与包组织

Com.mycompany.nextgen.ui.swing

Com.mycompany.nextgen.ui.web

} UI

Com.mycompany.nextgen.domain.sales

Com.mycompany.nextgen.domain.payments

} Domain

Com.mycompany.service.persistence

} Tech service

Org.apache.log4j

Com.mycompany.util

Foundation

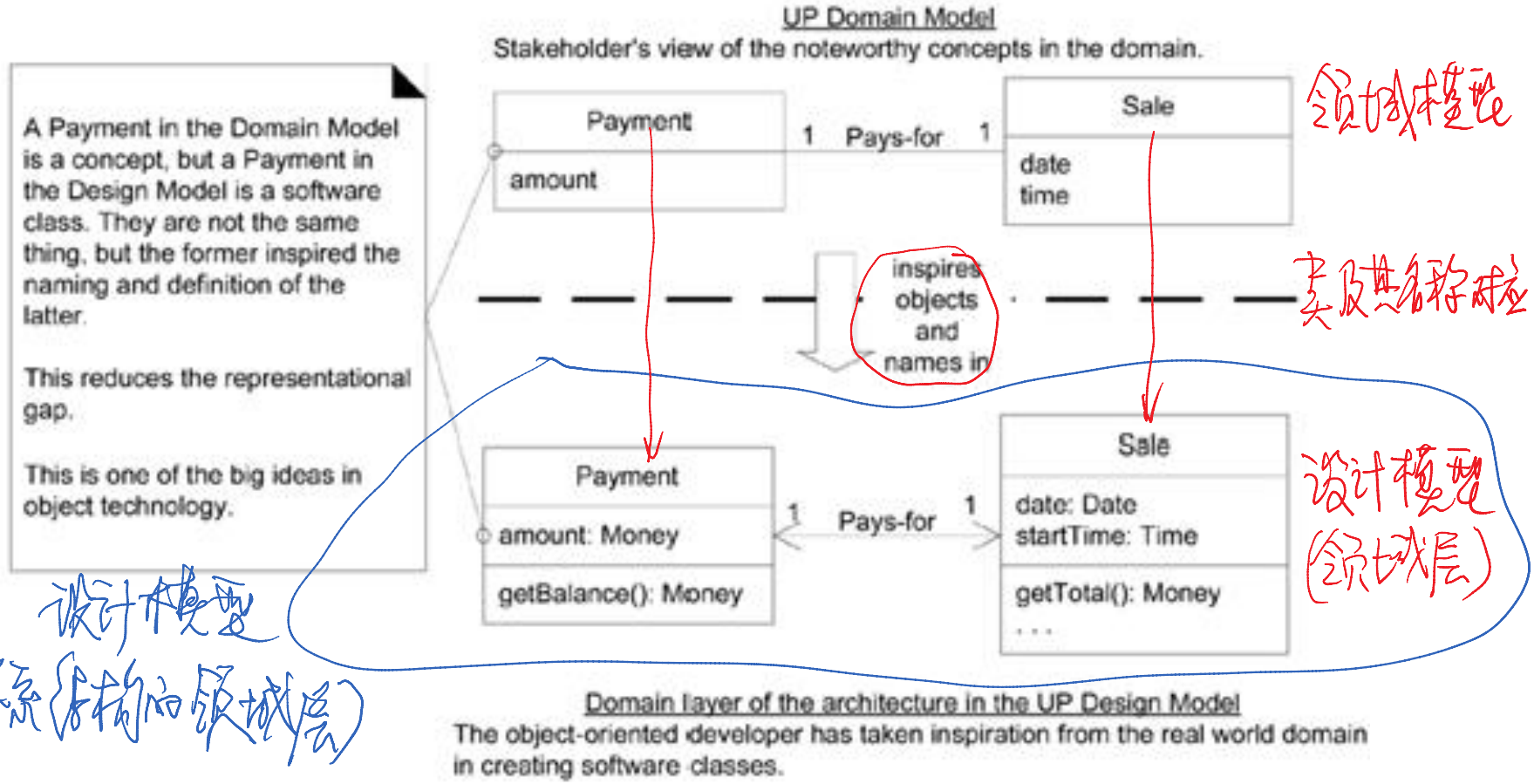
提供重用，不要加入包含！

C/C++ use namespace !

Domain Vs. Application Logic Layer

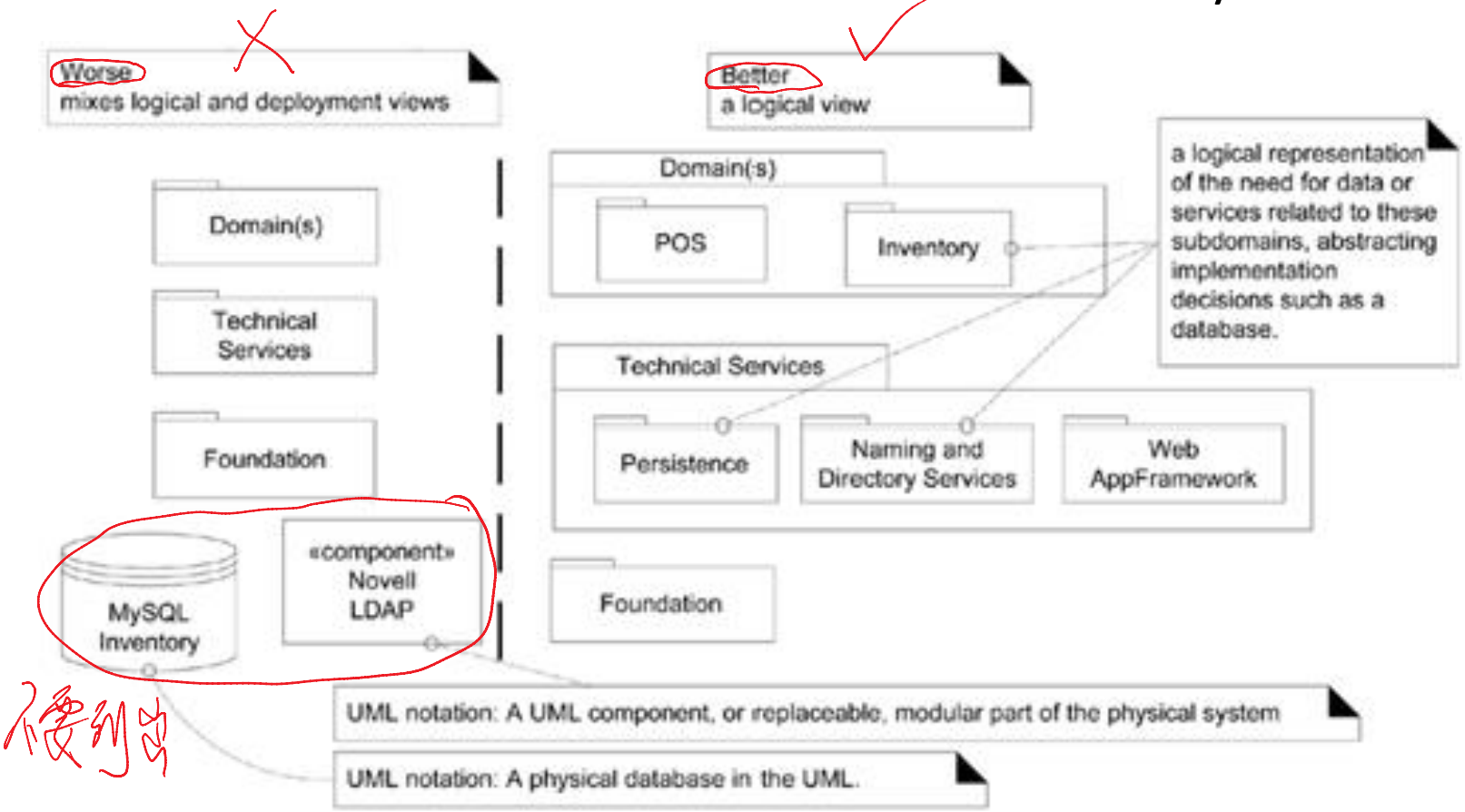
- How do we design the application logic?
  - To create software objects with names and information similar to the real-world domain, and
  - assign application logic responsibilities to them.
  - E.g. sales, payments, claims, etc. These are domain objects
- This kind of software object is called a **domain object**.
  - It represents a thing in the problem domain space, and
  - has related application or business logic,
  - for example, a *Sale* object being able to calculate its total.
- Designing objects this way leads to the application logic layer being more accurately called the **domain layer** of the architecture,
  - the layer that contains domain objects to handle application logic work.

Domain Layer and Domain Model Relationship



Guideline

- Most systems rely on external resources or services. These are *physical* implementation components, not a layer in the *logical* architecture.
- Don't show external resources as the bottom layer.

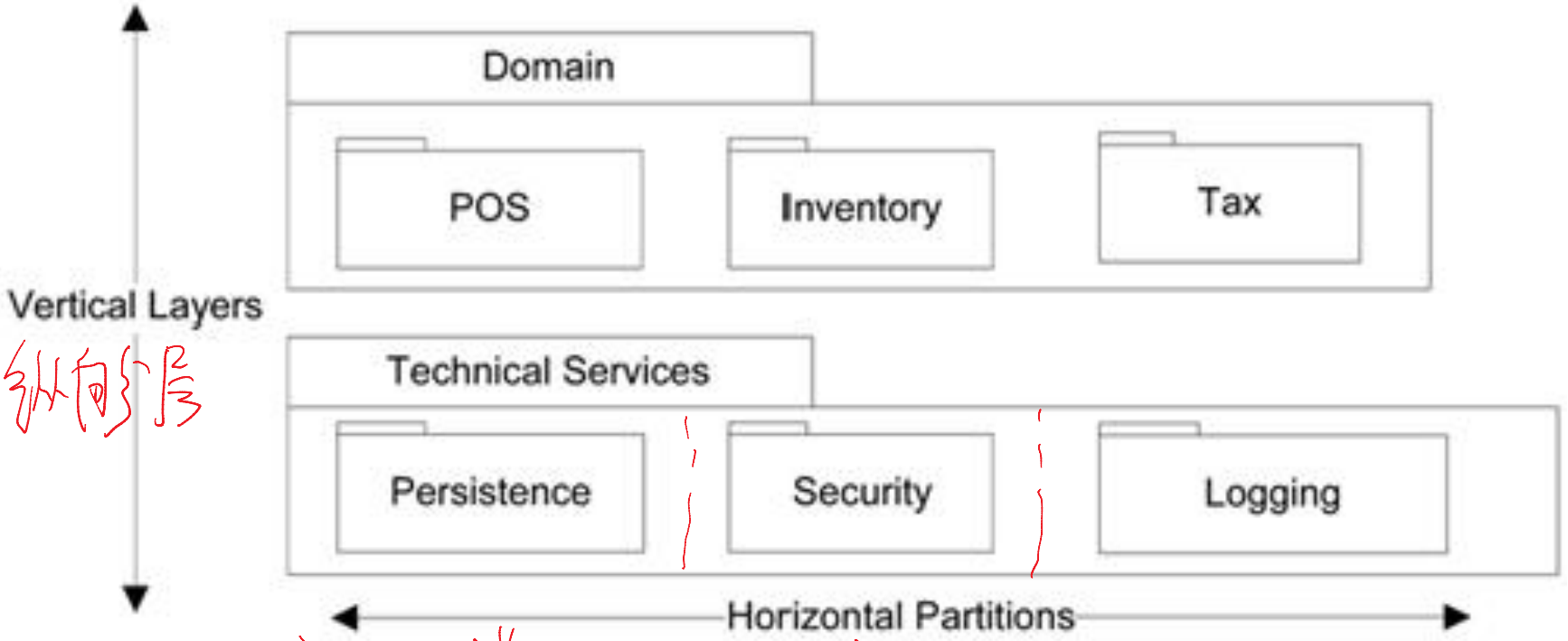


分层与分区(划分)

Layers and Partitions

- **Layers** are vertical slices, while **partitions** are horizontal divisions of subsystems within a layer. 分层是纵向分层，划分是横向分割(在一个纵向层内)
  - E.g. tech services may contain Security and Reporting
- **Tiers** were originally logical layers, but now the term has come to mean physical nodes.
  - Client Tier in Service Oriented Architecture

Layers and Partitions



层与区均用包图表示

Model-View Architecture

- UI layer can be understood as **View** 视图子系统
- Domain layer contains application logics as **Model** 模型子系统
- What kind of visibility should packages (in Model) have to the UI (View) layer? 可视性?
- How should non-window classes (in Model) communicate with windows (in View)? 如何交互?
- We usually have a **Model-View Separation Principle** 设计必须遵照分离原则

Model-View Separation Principle

- Don't connect non-UI objects directly to UI objects.
  - Don't let a Sale object reference directly a Jframe
- Don't put application logic, such as tax calculation, in a UI object's methods.
- UI objects should only
  - initialize the elements,
  - receive UI events, and
  - delegate requests for application logic to non-UI objects

观察者模式(最典型的图形界面模式)

Observer Pattern

实现Model与View的分离

- Domain objects send messages to UI objects viewed only in terms of an interface such as PropertyListener. (observers称为Listeners,定义的接口)
- Domain classes encapsulate the information and behavior related to the application logic. Subject
- Windows (UI) classes are thin. 这样UI对象必须很独立,而小。

关于设计模式,后续课还会讨论.

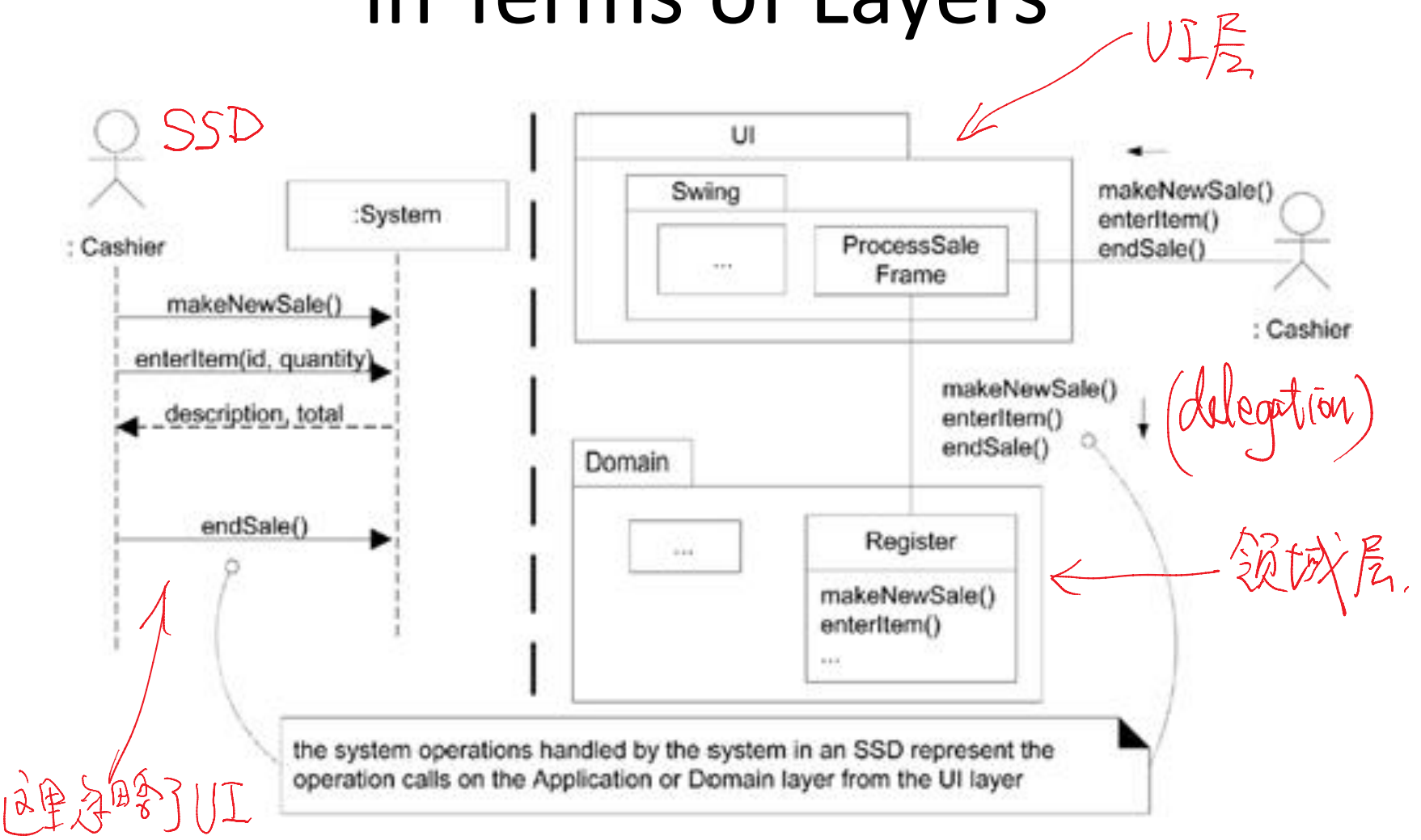
Why Model-View Separation?

- Focus on domain processes rather than the UI 关注领域处理
- Allow separate development of View (UI) and the model (domain) 开发任务的分离
- Minimize impact of requirements changes in the UI on the domain layer 在界面需求变化时，最小化对领域层的影响
- Allow views to be connected to existing domain layer
- Multiple views of same domain object UI替换 领域对象可以有多种视图
- Allow porting of UI to another framework UI可以移植
- Allow execution of model independently of UI 应用程序可以独立于UI运行

Connections Between SSDs, System Operations, and Layers

- SSDs illustrate system operations but hide the UI layer. Yet the UI capture the operation requests UI收集了操作请求，SSD展示了系统的操作.
- The UI layer delegates the requests to the domain layer UI将操作请求传递给领域层.
- Thus messages from the UI to the domain are the messages in the SSDs, such as enterItem.
从UI传到领域层的message就是SSD中描述的message.

System Operations in the SSDs and in Terms of Layers



UML Package Diagrams 包图

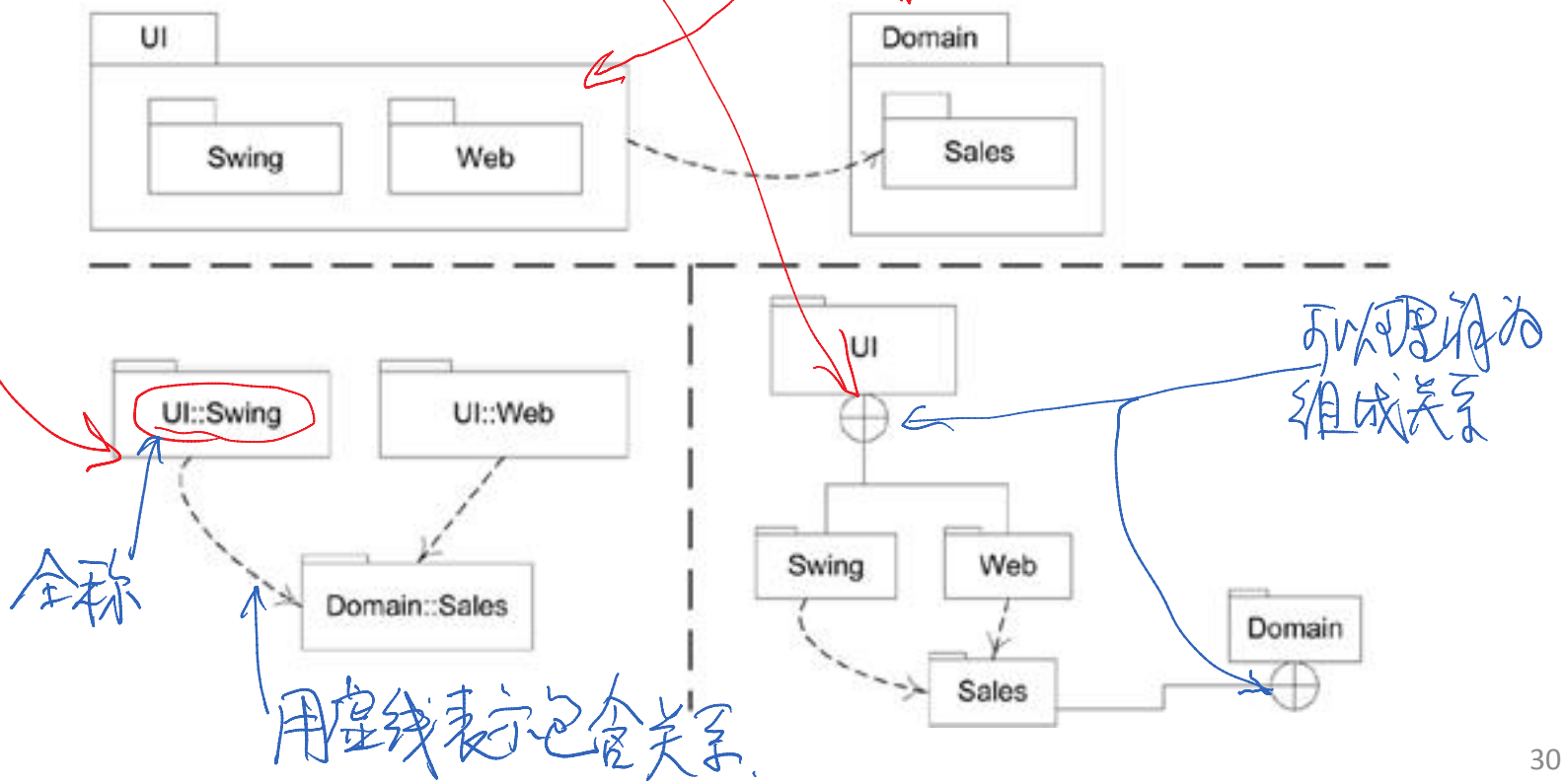
- A UML package diagram provides a way to group elements (anything): classes, other packages, use cases, and so on. 包图可以将其它一些元素打包成组.
- Nesting packages is very common. 允许嵌套
- A UML package is a more general concept than simply a Java package or .NET namespace. UML包图比命名空间更一般
- They are often used to illustrate the logical architecture of a system 通常用于体系结构描述
  - the layers, subsystems, packages, etc. 这些均可以是包的元素,
  - A layer can be modeled as a UML package; 一个层一般建模为一个包
  - for example, the UI layer modeled as a package named UI. 如:UI层可以建模为一个UI包.

UML Package Diagrams

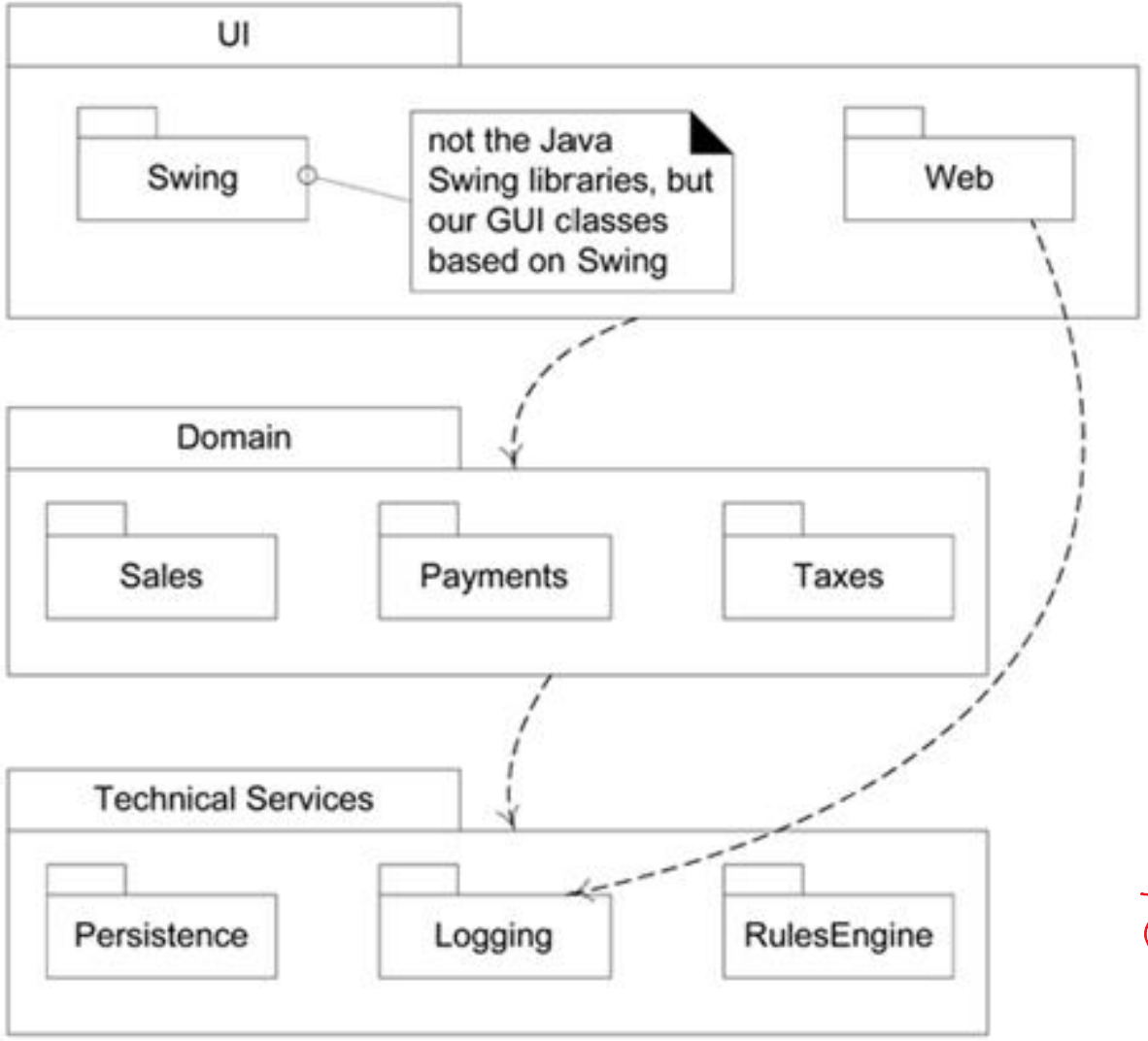
- The package name may be placed on the tab if the package shows inner members, or on the main folder, if not.
- Dependency (a coupling) between packages is shown by the UML **dependency line**,
  - a dashed arrowed line with the arrow pointing towards the depended-on package.
- A UML package represents a **namespace** so that a *Date* class may be defined in two packages.
- You need to provide **fully-qualified names (*java::util::Date)*** in this situation.

Alternate UML Approaches to Show Package Nesting

1. Using embedded packages
2. UML fully-qualified names
3. The circle-cross symbol



NextGen Logical Architecture and Package Diagram (三层体系结构)



图形界面层

领域层

技术服务层