1. RUP

软件开发过程描述了构造、部署以及维护软件的方式。统一过程（UP）已经成为一种流行的构造面向对象系统可执行规格说明。RUP：是对统一过程的详细精化，并且已经被广泛采纳。

包含六项最佳实践：迭代式软件开发、需求管理、基于构建的架构应用、建立可视化的软件模型、软件质量验证、软件变更控制。

2. association class 举例（关联类）

关联类允许将关联本身作为类，并且使用属性、操作和其他特性对其建模。例如，如果 Company 雇佣了许多 Person，建模时使用了 Employ 关联，则可以将关联本身建模为 Empolyment 类，并拥有 StartDate 这样的属性。

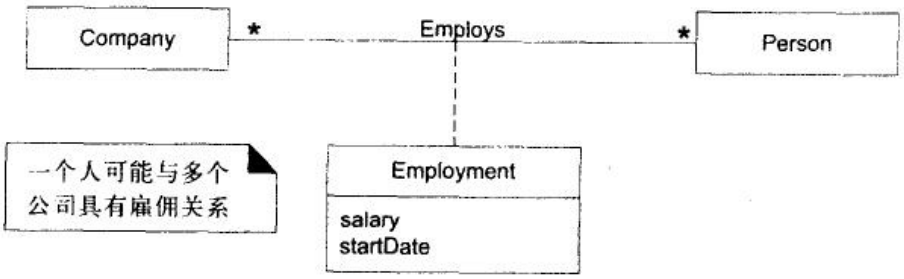


图16-16 UML中的关联类

3. Give simple examples to explain qualified-association

限定关联（qualified-association）具有限定符，限定符用于从规模较大的相关对象集合中，依据限定符的键选择一个或多个对象。一般来说，在软件透视图中，限定关联暗示了基于键对事物进行查找，例如 HashMap 中的对象。

例如，如果 ProductCatalog 中含有许多 ProductDecription，并且每个 ProductDescription 都能够通过 itemID 来选择，那么 UML 表示法可以对此进行描述。

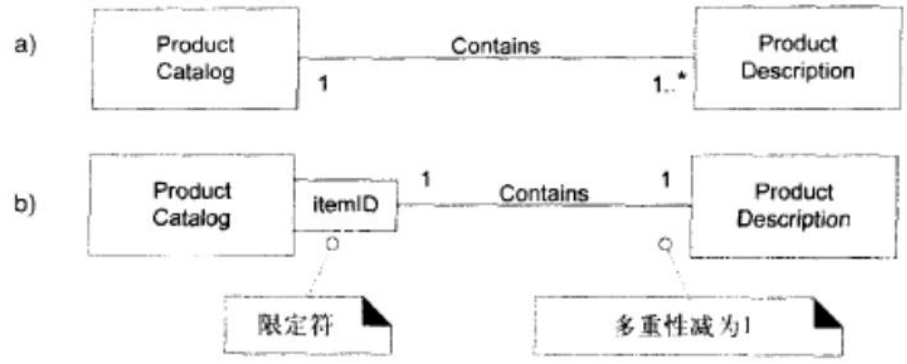


图16-15 UML中的限定关联

4. GRASP

创建者、信息专家、控制器、高内聚、低耦合、间接性、多态性、防止变异、纯虚构。

5. 什么是模式?

有经验的 OO 开发者（以及其他的软件开发者）建立了既有通用原则又有惯用方案的指令系统来指导它们编制软件。如果以结构化形式对这些问题、解决方案和命名进行描述使其系统化，那么这些原则和习惯用法就可以成为模式。

6. FURPS

- 功能性(Functional)：特性、功能、安全性。
- 可用性(Usability)：人性化因素、帮助、文档。
- 可靠性(Reliability)：故障频率、可恢复性、可预测性。
- 性能(Performance):响应时间、吞吐量、准确性、有效性、资源利用率。
- 可支持性(Supportability)：适应性、可维护性、国际化、可配置性。

“FURPS+”中"+”是指一些辅助性的和次要的因素，比如：

- 实现(Implementation):资源限制、语言和工具、硬件等。
- 接口 （Interface）:强加于外部系统接口之上的约束。
- 操作（Operation）：对其操作设置的系统管理。
- 包装（Packaging）：例如物理的包装盒。
- 授权（Legal）：许可证或其他方式。

7. OOD 解决了什么问题?

OOD 强调的是定义软件对象以及如何协作以实现需求。强调的是如何做正确的事。解决了如何为对象类分配职责、对象之间应该如何协作、什么样的类应该做什么样的事情等问题。

OOD 的制品

静态模型： 类图、包图

动态模型： 顺序图、通信图

8. What solved in OOA?

面向对象分析的制品是领域模型，在领域模型中展示了重要的领域概念或对象。需要注意的是，领域模型并不是对软件对象的描述，而是真实事件领域内的概念和对象的可视化。

OOA 是面向对象分析，在面向对象分析中，强调了问题领域所描述的对象或者概念，由属性，关系和概念组成。面向对象分析的结果可以表示为领域模型，在领域模型中展示重要的领域对象或者概念。但是领域模型并不是对软件对象的描述，只是对真实世界领域中的对象和概念的可视化。

OOA 过程

首先需要识别出系统中的用例并且建立用例模型，发现用例中的概念类，定义基本属性，创建出基本的动态模型。

OOA 的制品

领域模型：现实世界中对象的概念透视图

用例模型：描述功能需求，一组使用系统的典型场景

补充性规格说明：描述其他需求，主要是非功能行需求

词汇表：关键领域术语和数据字典

9. OOA 制品

OOA 是面向对象分析，在面向对象分析中，强调了在问题领域内发现和描述对象或者概念。

制品（artifact）是对所有工作产品的统称，如代码、Web 图形、数据库模式、文本文档、图、模型等。

OOA 制品包含：

领域模型：现实世界中对象的概念透视图

用例模型：描述功能需求，一组使用系统的典型场景

补充性规格说明：描述其他需求，主要是非功能行需求

词汇表：关键领域术语和数据字典

设想

业务规则

10. operation contract

1) 操作契约使用前置和后置条件的形式，描述领域模型里对象的详细变化，并作为系统操作的结果。
2) 操作契约可以视为 UP 用例模型的一部分，因为它对用例指出的系统操作的效用提供了更详细的分析。
3) 操作契约有哪些部分
   - 操作：操作的名称和参数
   - 交叉引用：会发生此操作的用例
   - 前置条件：执行操作之前，对系统或领域模型对象状态的重要假设。
   - 后置条件：最重要的部分。完成操作后，领域模型对象的状态。
4) 系统操作是作为黑盒构件的系统在其公共接口中提供的操作。系统操作可以在绘制 SSD 草图时确定。
5) 后置条件描述了领域模型内对象状态变化。领域模型状态变化包括创建实例、形成或消除关联以及改变属性。后置条件有三种类型创建或删除实例、形成或消除关联、属性值的变化。
6) 在 UP 中，用例是项目需求的主要知识库。用例可以为设计提供大部分或全部所需细节。这中情况下，契约就没什么用处。
7) 如何创建和编写契约
   - 从 SSD 图中确定系统操作

- 如果系统操作复杂，其结果可能不明显，或者在用例中不清楚，则可以为其构造契约
- 使用以下几种类表来描述后置条件：
  a) 创建或删除实例
  b) 属性值的变化
  c) 形成或消除关联（UML 连接）

11.Why SSD(绘制 SSD 的好处)

系统顺序图(SSD)是为阐述与所讨论系统相关的输入和输出事件而快速、简单地创建的制品。它们是操作契约和（最重要的）对象设计的输入。

（1）绘制系统时序图的好处：

SSD 使事件变得更加具体和明确；

（2）**绘制的动机**

基本上，软件系统要对以下三种事件进行响应：1）来自于参与者（人或计算机）的外部事件，2）时间事件，3）错误或异常（通常源于外部）。

因此，需要准确地知道，什么是外部输入的事件，即系统事件。这些事件是系统行为分析的重要部分。

你可能对如何识别进入软件对象的消息非常熟悉。而这种概念同样适用于更高阶的构件， 包括把整个系统（抽象地）视为一个事物或对象。

在对软件应用将如何工作进行详细设计之前，最好将其行为作为“黑盒”来调查和定义。 系统行为（system behavior）描述的是系统做什么，而无需解释如何做。这种描述的一部分就是系统顺序图。

12.Give simple examples to explain static modeling

静态建模有助于设计包、类名、属性和方法特征标记（但不是方法体）的定义，例如 UML 类图。UML 中支持静态建模的其他制品包括包图和部署图。

13.动态建模

动态建模有助于设计逻辑、代码行为或方法体，例如 UML 交互图（顺序图或通信图）。

14.DCD

DCD 是设计类图，同一种 UML 可以用于多种透视图，在概念透视图里，类图可以用于将领域模型可视化，在 UP 中，所有 DCD 的集合形成了设计模型的一部分。

DCD 中的操作问题：

Create 操作

访问属性的操作

15.GRASP 最重要的三个模式及举例

名称：创建者
问题：谁应该负责创建某类的新实例？
解决方案：如果以下的条件之一（越多越好）为真时，将创建类 A 实例的职责分配给类 B：

① B“包含”或组成聚集 A
② B 记录 A
③ B 直接使用 A
④ B 具有 A 的初始化数据，并且在创建 A 时会将这些数据传递给 A。

举例：在 NextGenPos 应用，Sale 包含许多 SaleLineItem 对象，故根据创建者模式，Sale 是具有创建 SaleLineItem 实例职责的良好候选者。

名称：信息专家模式
问题：给对象分配职责的基本原则是什么？
解决方案：把职责分配给信息专家，它具有完成该职责所必需的信息。

举例：在 NextGenPos 应用中某个类需要知道销售总额。销售总额需要销售的所有 SaleLineItem 实例及其小计之和。Sale 实例包含了上述信息，故根据信息专家建议的准则，Sale 是适合这一项工作的信息专家。

名称：控制器模式
问题：在 UI 层之上首先接收和协调（控制）系统操作的第一个对象是什么？
解决方案：把职责分配给能代表以下选择之一的类：

① 代表整个“系统”、“根对象”、运行软件的设备或主要子系统，这些事外观控制器的所有变体。
② 代表用例场景。

举例：在 NextGenPos 中，对于 enterItem 和 endSale 这样的系统事件，应该使用谁作为控制器？根据控制器模式，下面给出一些选择：

代表整个“系统”、“根对象”、装置或子系统：Register，POSSystem
代表用例场景中所有系统事件的接收者或处理者：ProcessSaleHandler,ProcessSaleSession

16.GoF 模式

1、 **名称：适配器（GoF）：**

问题：如何解决不相容的借口问题，或者如何为具有不同接口的类似构件提供稳定的借口？

解决方案：通过中介适配器对象，将构件的原有接口转换为其他接口。

有一种解决方案是：增加一层间接性对象，通过这些对象将不同的外部接口

调整为在应用程序内使用的一致接口

例子：NextGen Pos 系统需要支持多种第三方外部服务，其中包括税金计算器、信用卡授权服务、库存系统和账务系统。它们都具有不同的 API，而且还无法改变。

2、名称：工厂（Factory）

问题：当有特殊考虑时，应该由谁来负责创建对象？

解决方案：创建称为工厂的纯虚构对象来处理这些创建职责。

3、名称：单实例类

问题：只有唯一实例的类即为“单实例类”。对象需要全局可见性和单点访问

解决方案：对类定义静态方法用以返回单实例。

4、名称：策略

问题：如何设计变化但相关的算法或政策？如何设计才能使这些算法或政策具有可变更能力？

解决方案：在单独的类中分别定义每种算法/政策/策略，并且使其具有共同接口

5、名称：组合

问题：如何能够像处理非组织（原子）对象一样，（多态地）处理一组对象或具有组合解耦股的对象呢？

解决方案：定义组合和原子对象的类，使它们实现相同的接口。

6、名称：外观

问题：对一组完全不同的实现或接口需要公共、统一的接口。可能会与子系统内部的大量事物产生耦合，或者子系统的实现可能会改变，怎么办？

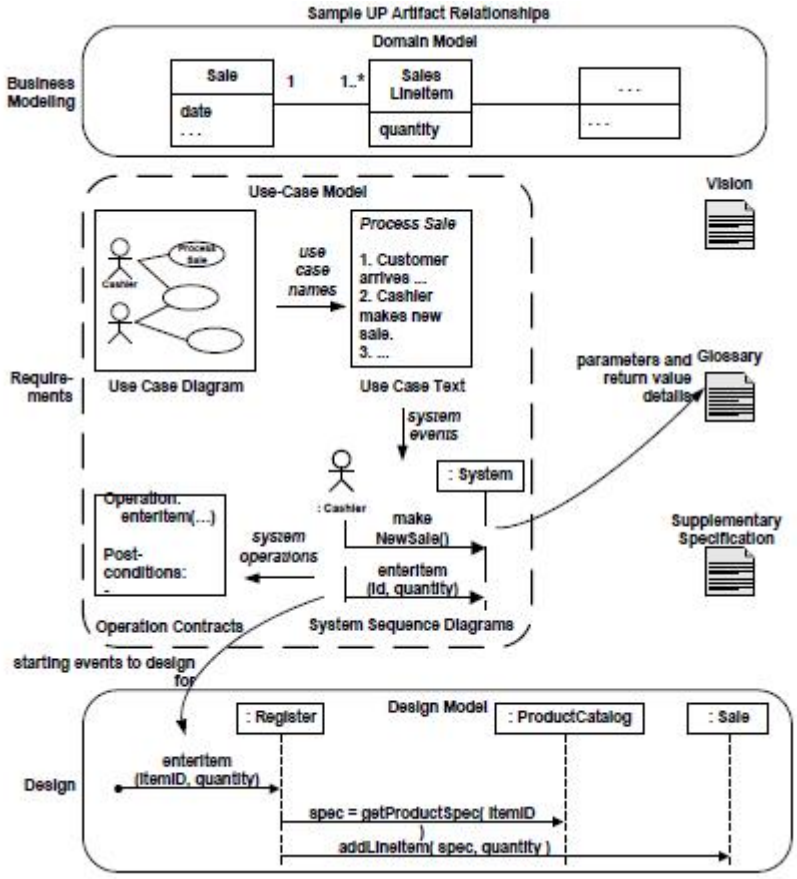
解决方案：对子系统定义惟一的接触点—使用外观对象封装子系统。该外观对象提供了惟一和统一的接口，并负责与子系统构件进行协作

7、名称：观察者（发布—订阅）

问题：不同类型的订阅者对象关注于发布者对象的状态变化或事件，并且想要在发布者产生事件时以自己独特的方式作出反应。此外，发布者想要保持与订阅者的低耦合。如何对此进行设计呢？

解决方案：定义“订阅者”和“监听器”接口。订阅者实现此接口。发布者可以动态注册关注某事件的订阅者，并在事件发生时通知它们

17. Explain the following graph 课本 P46 页原图的英文版



UP 描述了**科目（discipline）**中的工作活动。
科目是在一个主题域中的一组活动（及相关制品）。
在 UP 中，**制品（artifact）**是对所有工作产品的统称，如代码、Web 图形、数据库模式、文本文档、图、模型等。

UP 中有几个科目，本书只关注以下三个科目中的制品：

- 业务建模：领域模型制品，使用领域中的重要概念的可视化。
- 需求：用以捕获功能需求和非功能需求的用例模型及其补充性的规格说明制品。
- 设计：设计模型制品，用于对软件对象进行设计。

下面从这三个科目来着手解释这张图：

（1） 在业务建模阶段：

面向对象分析的结果可以表示为领域模型，在领域模型中展示重要的领域概念或对象。

在此图中，sale 表示了销售交易过程，其中包括日期 date 属性

SaleLineItem 表示具体的购买商品条目，其中包括此商品的 quantity 属性

Sale 与 SaleLineItem 存在一对多的关联关系，即一次销售过程有许多购买条目。

②需求分析阶段：

建立用例模型：

1) 建立用例图：系统与环境通过外框分开，参与者 cashier 有 processSale 的用例行为。
2) 建立用例文本：包括一个 processSale 用例，其中描写了一系列的（主成功或者拓展）场景，如 Customer arrives 等
3) 建立系统顺序图：描述了某场景的系统事件，外部参与者 cashier 向系统发出不带参数的 makenewsale()请求和一个带有参数的 enterItem(id,quantity)请求。
4) 建立操作契约：描述了系统顺序图中某系统操作 enteritem（）的详细操作契约，包括名称，交叉引用，前置条件，后置条件等。
5) Vision 设想：范围、目标、参与者、特性
6) Glossary 词汇表：术语、属性、验证
7) Supplementary XXXX 补充规格说明：非功能性需求、质量属性

③设计阶段：

图中只有系统时序图：外部向 register 对象发送 enteritem（）请求希望输入销售商品条目，register 收到后，调用 productCatalog 的 getProductDescription（）方法并获得返回值用来确定商品的详细信息，接着调用 sale 类的 addlineitem 让其处理此次输入的商品条目并加入销售清单。这样，一次商品添加过程完成。

18.根据下面的通信图写出 enterLineItem 函数和 makeLineItem 函数的 java 代码

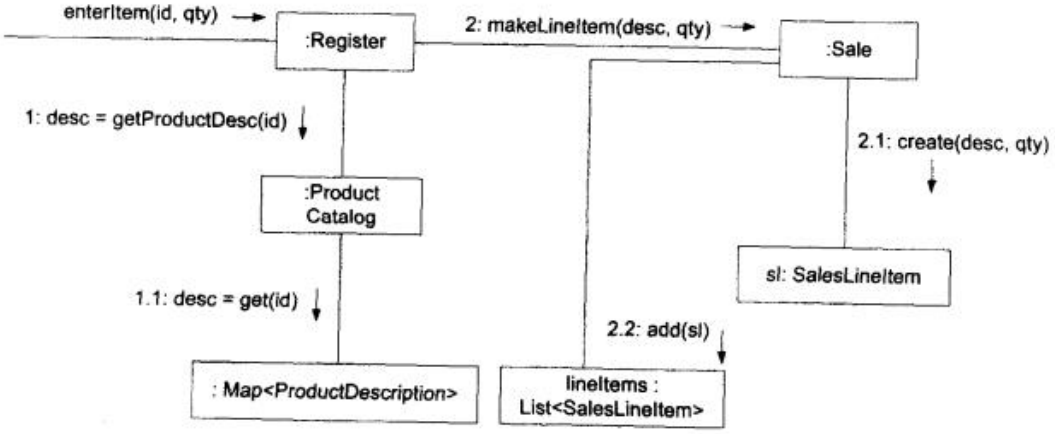


图20-2 enterItem交互图

```
public class Register {
    private ProductCatalog catalog;
    private Sale curreSale;

    public Register(ProductCatalog catalog) {
        this.catalog=catalog;
    }
    public void enterItem(ItemID id,int quantity) {
        ProductDescription desc = catalog.getProductDescription(id);
        curreSale.makeLineItem(desc,quantity);
    }
```

```java
public class ProductCatalog {
    private Map<ItemID,ProductDescription> descriptions
    =new HashMap<ItemID, ProductDescription>();

    public ProductCatalog() {
        ItemID id1=new ItemID(100);
        ItemID id2=new ItemID(200);
        Money price =new Money(2);
        ProductDescription desc;
        desc=new ProductDescription(id1,price,"product 1");
        descriptions.put(id1, desc);
        desc=new ProductDescription(id2,price,"product 2");
        descriptions.put(id2, desc);
    }

    public ProductDescription getProductDescription(ItemID id) {
        return descriptions.get(id);
    }
}
```

```java
public class Sale {
    private ArrayList<SaleLineItem> lineItems= new ArrayList<SaleLineItem>();
    private Date date=new Date(0);
    private boolean isComplete=false;
    private Payment payment;

    public void makeLineItem(ProductDescription desc,int quantity) {
        lineItems.add(new SaleLineItem(desc,quantity));
    }
```