题型：

简答，选择，填空，大题

第一章导论

1.软件工程的目的：书 P1

对成本、进度和软件质量的控制

2..软件的本质特性：

复杂性、一致性、可变性、不可见性

3..软件的特性（3 个）书 P1-2

重点：书 P3 的图 1.1，理解修改点对失效率的影响

4.软件的类型：书 P3

a.从功能：**系统软件和应用软件**

b．从服务对象：**通用软件和定制软件**

5．软件质量特性

评价标准书 P4-6 图 1.2 表 1.1 表 1.2 以及软件质量要素 Fj 的评价公式（第六页开头）

5.软件危机在概念提出前就存在

6.软件工程在 IEEE 下的定义及其三要素（书 p7）

7 软件过程模型的定义以及典型模型（书 p9）

8 软件工程方法主要有**结构化方法和面向对象的方法**（书 p10）

9 结构化分析方法的基本原则是**功能的分解和抽象**（P10）

10 结构化设计的实质（p11）

11 面向对象软件工程方法设计的四个方面（P11）

12．P12 图 1.6 分析和设计的关系

13CASE 工具的三类及三者的区分（P14）

14 软件工程体系的 10 个知识领域（p15）

第二章软件过程

1 软件过程的定义（书 P18 导言里的第一句）

2 软件过程的三类：（书 P18）

A．**基本过程类**包括需求获取和定义过程、设计过程、实现过程、验证过程和维护过程；

B．**支持过程类**包括文档过程、配置管理过程、质量保证过程、联合评审过程、审计过程等；

C．**组织过程类**包括基础设施过程、改进过程以及培训过程。

3.软件过程模型（6 个，了解其模型图，使用范围以及优缺点）

A．瀑布模型：

模型图（P19）

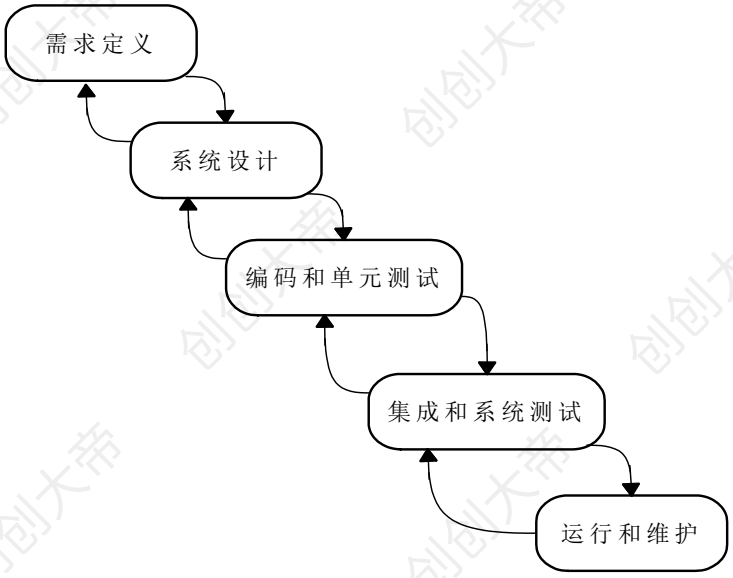


图 2.1 瀑布模型

适用范围：

在开发的早期阶段软件需求被完整确定

优点：

线性是一种简洁，简洁就是美，应活用线性模型。

1“线性”是人们最容易掌握并能熟练应用的思想方法。当人们碰到一个复杂的“非线性”问题时，总是千方百计地将其分解或转化为一系列简单的线性问题，然后逐个解决。

2 一个软件系统的整体可能是复杂的，而单个子程序总是简单的，可以用线性的方式来实现。

缺点：

1.各个阶段的划分完全固定，阶段之间产生大量的文档，极大地增加了工作量

2 用户只有等到整个过程的末期才能见到开发成果，从而增加了开发的风险

3 开发过程中很难响应客户的变更要求

4 早期的错误可能要等到开发后期的测试阶段才能发现，进而带来严重的后果

5.线性模型太理想化，太单纯，已不再适合现代的软件开发模式，几乎被业界抛弃。

例题：

- 某公司计划开发二维 CAD 软件
  - 软件功能需求
    - 基本功能与国外 AutoCAD 产品一致
    - 新增加功能三个功能：智能画线、智能标注、读取 .dwg 文件
- 问题：该软件开发适于采用什么过程模型？原因？

B．演化式开发模型

模型图：

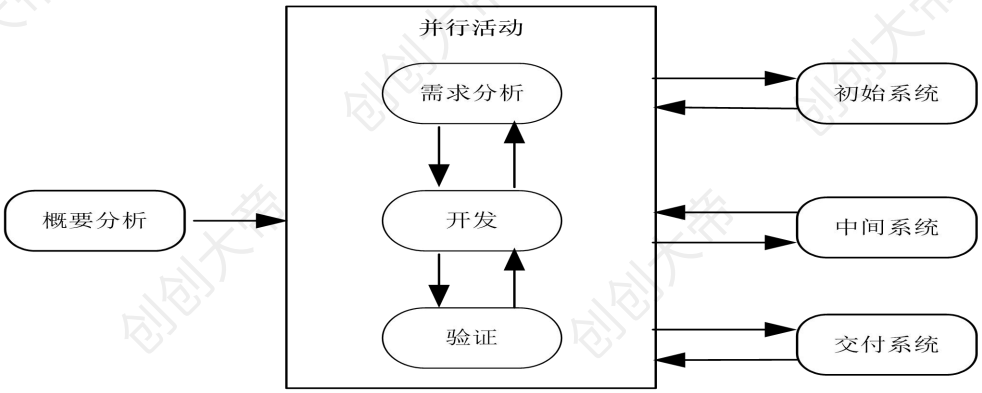


图2.2 演化式开发模型

分为探索式开发和抛弃式开发

适用范围：

适用于生命周期较短、中小规模的系统。当然在大型系统中，视具体情况部分采用演化式开

发也是可行的。
优点：
1 在及时响应用户需求改变方面比瀑布模型更为有效，软件改变的代价较小
缺陷：
1 系统的结构通常不好
2 软件过程可见性不好
3 开发过程通常需要特殊的工具和技术支持
例题：

- 北京市交通设施设计与漫游系统
  - 用户的要求
    - （1）完成交通设施的二维图纸设计
    - （2）在二维设计的基础上模拟显示设计的效果
  - 现实情况
    - （1）二维设计部分已有工作基础，新功能开发量不大
    - （2）设计仿真与漫游显示部分需求不确定
    - （3）用户要求开发方提出可行的方案
- 问题：采用什么方法开发仿真与漫游子系统？原因？

C．形式化变换模型
模型图：

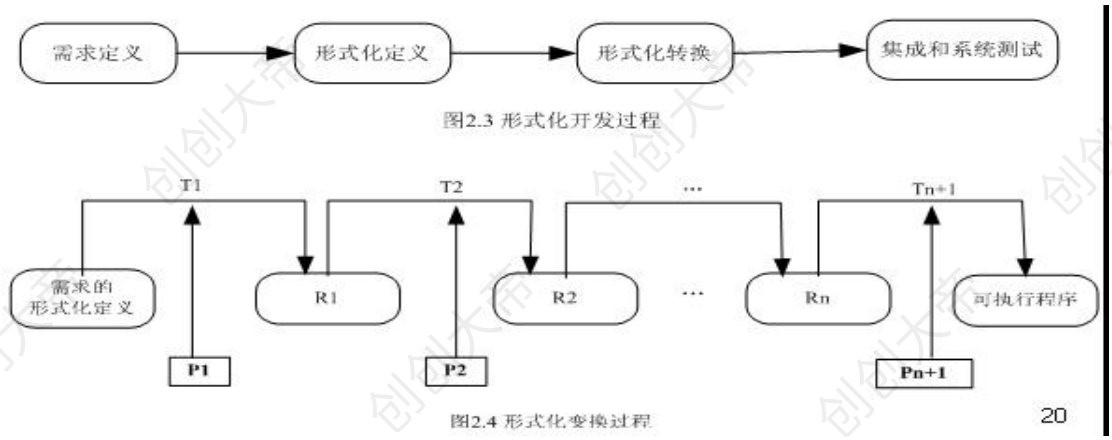


图2.3 形式化开发过程
图2.4 形式化变换过程
20

适用范围：
特别适合于那些对安全性、可靠性和保密性要求极高的软件系统，这些系统需要在投入运行前进行验证
优点
– 由于数学方法具有严密性和准确性，形式化方法开发过程所交付的软件系统具有较少的缺陷和较高的安全性
缺点
– 开发人员需要具备一定技能并经过特殊训练
– 形式化描述和转换是一项费时费力的工作
– 现实应用的系统大多数是交互性强的软件，但是这些系统难以用形式化方法进行描述
例题：

空中交通控制系统

– 在系统运行之前需要进行安全性和可靠性的检验

问题：该软件开发适于采用什么过程模型？

D．面向复用的开发模型

模型图：

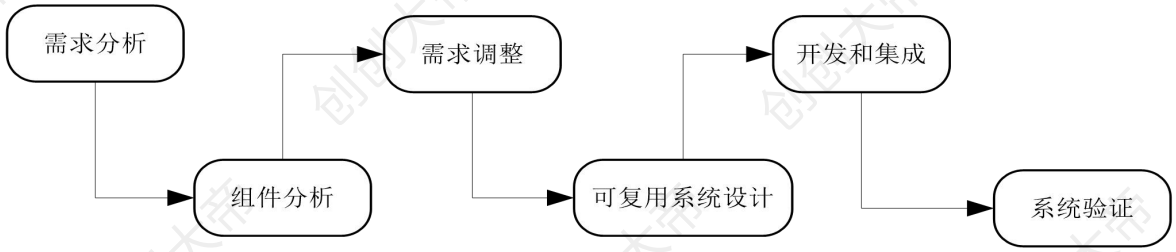


图2.5 面向复用的开发模式

适用范围：

主要建立在已有大量可利用的软件组件和组件的集成框架基础上，可以有效提高开发效率和质量。

优点

– 充分体现软件复用的思想
– 实现快速交付软件

缺点

– 商业组件的修改受到限制，影响系统的演化

例题：

- 开发学生选课系统
  - 某大学计划开发一个新的学生选课系统，以替换原有的选课系统
  - 原有的课程信息数据库将继续使用
  - 新的选课系统允许学生在网上选课和查询课程成绩，教师可以在网上确认所教课程、查询选课学生并登记成绩
- 问题：该软件开发适于采用什么过程模型？

E．增量开发模型

模型图：（书 p22 图 2.6）

适用范围：

按优先级划分的系统

优点：

– 整个产品被分解成若干个构件逐步交付，用户可以不断地看到所开发软件的可运行中间版本

– 将早期增量作为原型有助于明确后期增量的需求

– 降低开发风险

– 重要功能被首先交付，从而使其得到最多的测试

• 缺点：

– 需要软件具备开放式的体系结构

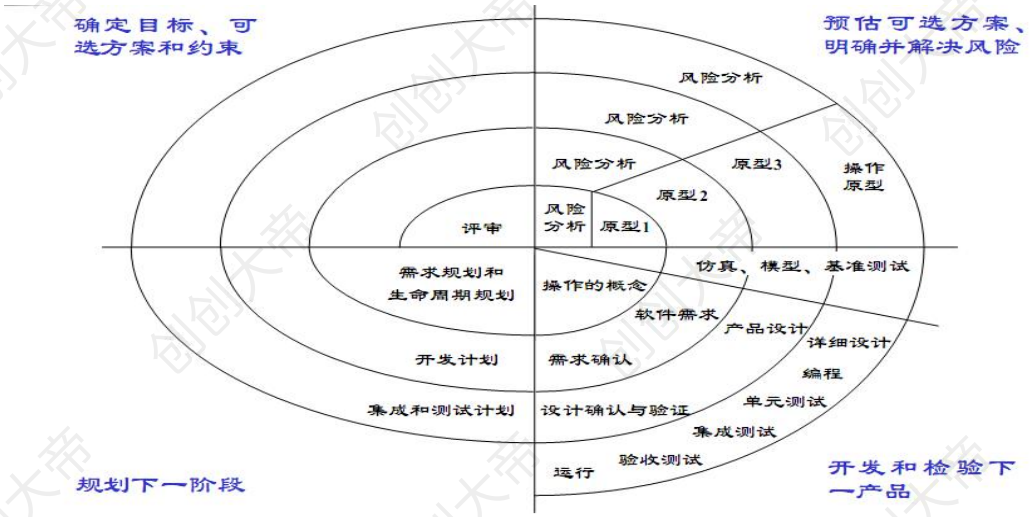
– 需求难以在增量实现之前详细定义，因此增量与需求的准确映射以及所有增量的有效集成可能会比较困难

– 容易退化为边做边改方式，使软件过程的控制失去整体性

例题：

F．螺旋模型

模型图：



• 适用范围：

大型、复杂的软件系统

- 优点：

– 关注软件的重用
– 关注早期错误的消除
– 将质量目标放在首位
– 适合大型软件项目

缺点：

– 需要风险评估的经验

— 螺旋模型强调风险分析，但要求许多客户接受和相信这种分析，并做出相关反应是不容易的，毕竟影响软件的因素复杂。另外风险分析得需要一定的代价，如果软件预算不充裕，较难实行

- 如果执行风险分析将大大影响项目的利润，那么进行风险分析毫无意义，因此，螺旋模型只适合于大规模软件项目。
- 软件开发人员应该擅长寻找可能的风险，准确地分析风险，否则将会带来更大的风险

4. Rational 统一过程的缩写为 **RUP**，**强调**：开发和维护模型，UML 的使用使其具有语义丰富的软件系统表达。

5Rational 统一过程中生命周期的四个阶段及区分：

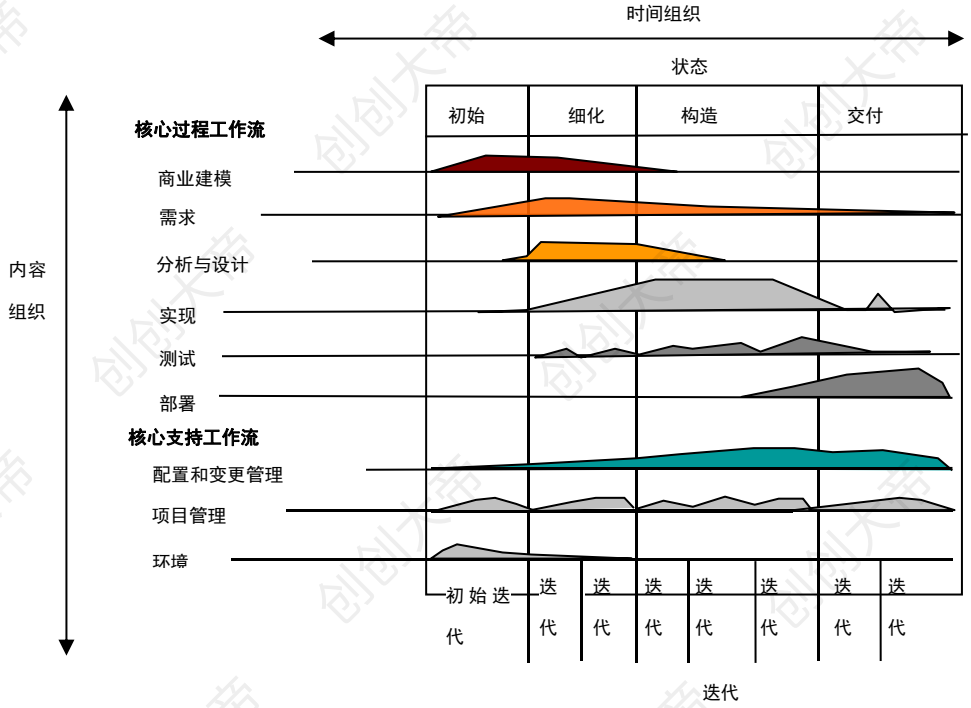


图2.9 Rational统一过程模型

- 初始阶段
  - 目标：为系统建立业务模型并确定项目的边界。
  - 任务：必须识别所有与系统交互的外部实体，在较高层次上定义交互的特性。
  - 关注焦点：是整个项目进行中的业务和需求方面的主要风险。
    - 对于建立在原有系统基础上的开发项目来讲，初始阶段可能很短。
  - 初始阶段结束是第一个重要的里程碑：生命周期目标（Lifecycle Objective）里程碑。
    - 生命周期目标里程碑评价项目基本的生存能力。

- 细化阶段
  - 目标：分析问题领域，建立健全的体系结构基础，编制项目计划，淘汰项目中最高风险的元素。
  - 任务：必须在理解整个系统的基础上，对体系结构做出决策，
    - 包括其范围、主要功能和性能等非功能需求，同时为项目建立支持环境，包括创建开发案例，创建模板、准则并准备工具。
  - 细化阶段结束是第二个重要的里程碑：生命周期体系结构（Lifecycle Architecture）里程碑。
    - 生命周期体系结构里程碑为系统的结构建立了管理基准，并使项目小组能够在构造阶段进行评价。
- 构造阶段
  - 目标：构造出软件产品。
  - 任务：所有组件和功能被开发并集成为产品，并被详细测试。
    - 从某种意义上说，构造阶段是一个制造过程，其重点放在管理资源及控制运作以优化成本、进度和质量。
  - 构造阶段结束是第三个重要的里程碑：初始运行（Initial Operational）里程碑。
    - 初始运行里程碑决定了产品是否可以在测试环境中进行部署。此时的产品版本也常被称为“beta”版。
- 交付阶段
  - 目标：确保软件对最终用户是可用的。
  - 任务：交付阶段可以跨越几次迭代，包括为发布做准备的产品测试，基于用户反馈的少量的调整。在生命周期的这一点上，用户反馈应主要集中在产品调整，设置、安装和可用性问题。
  - 交付阶段的终点是第四个里程碑：产品发布（Product Release）里程碑。

6 敏捷开发过程

强调快速工作、响应变化能力

第 3 章 面向对象系统建模

1. UML 关系：

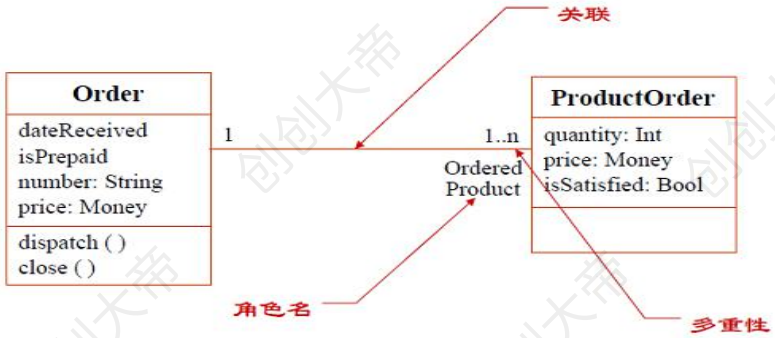
A 关联：

是一种结构关系，它描述了一组对象之间的连接。反映了对象之间的特定对应关系比如：你和你的朋友之间的关系

– 关联两端的类可以某种**角色**参与关联
- 角色是关联中靠近它的一端的类对另一端的类呈现的职责；
- 如果关联上没有标出角色名，则隐含地用类的名称作为角色名。

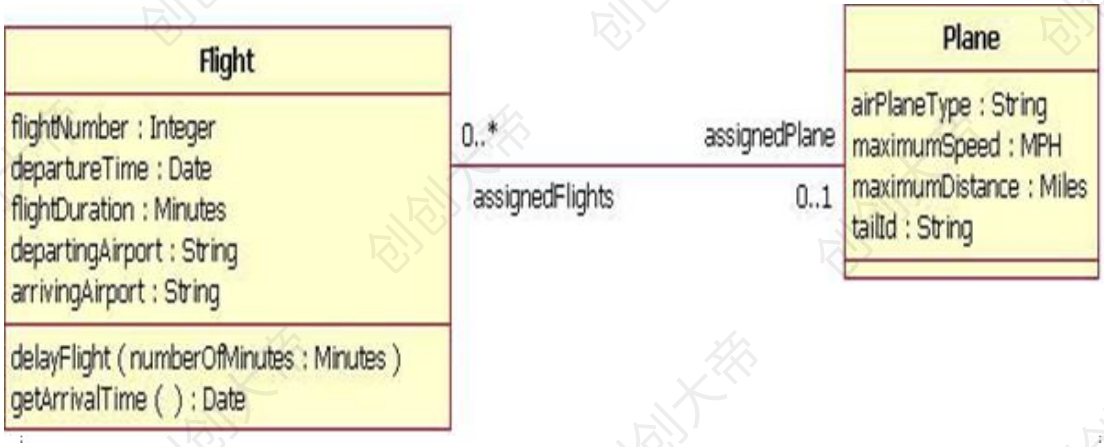
– 关联具有**多重性**
- 多重性表示可以有多少个对象参与该关联
- 固定值：**3**
- 许多值：**n** 或者 ***
- 区间：**0..1** 或者 **3..n**
- 集合：**2,4,8**

UML 图：



单向关联

双向关联



限定关联：

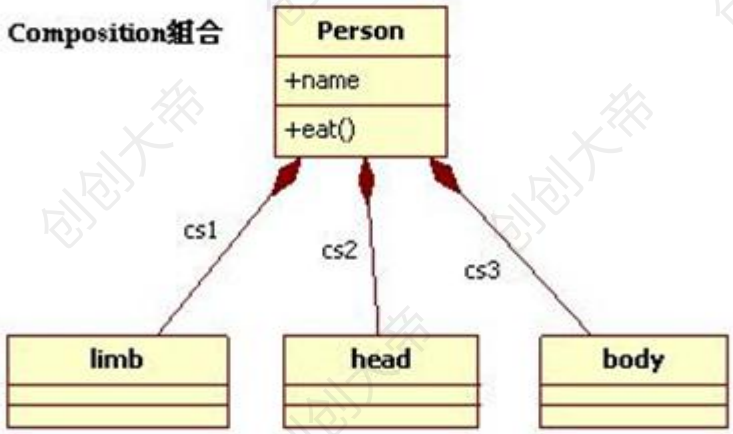
B 组合和聚合：

- 聚合（***Aggregation***）是一种特殊形式的关联，它表示类之间的整体与部分的关系。（比如：汽车和轮胎之间的关系）**整体与部分的生命周期可以不同**
- 组合（***Composition***）是一种特殊形式的聚集，组合关系中的整体与部分**具有同样的生存期。**（比如：你和你的心脏之间的关系）

UML 图：

聚合：

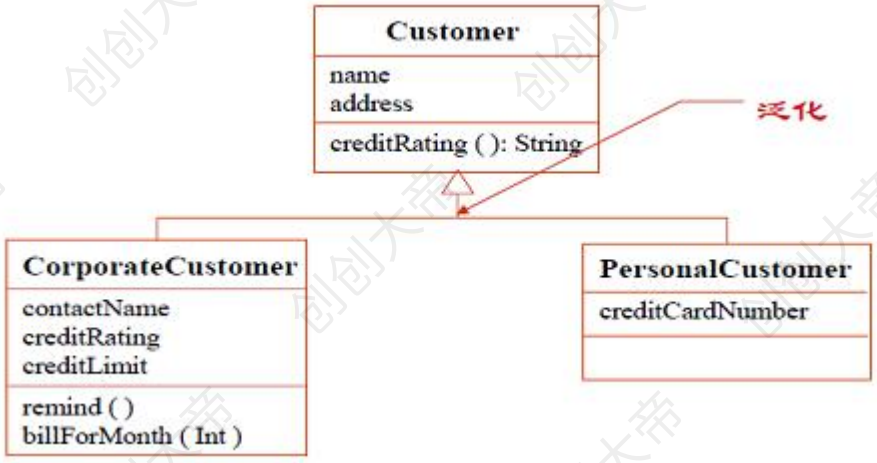
组合：



C 泛化：

泛化是一种特殊/一般的关系。（继承也算范化）

UML 图：



D 依赖关系：

依赖是一种使用关系，它说明一个事物规格说明的变化可能影响到使用它的另一个事物

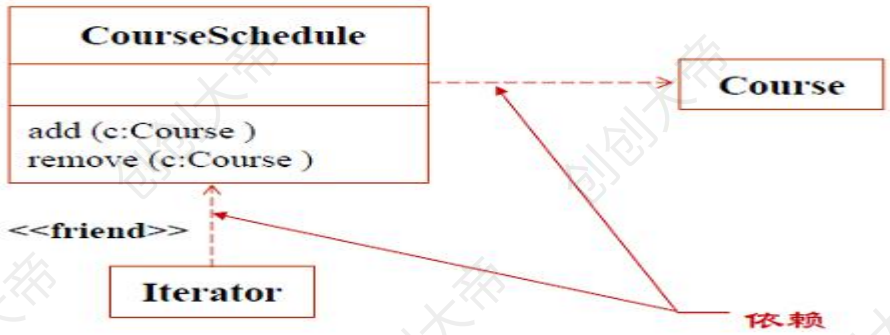
类的依赖可能由各种原因引起

– 一个类向另一个类发消息

– 一个类是另一个类的某个操作参数

• 说明：依赖关系不只是限于类之间。

UML 图：



依赖与关联的区别：

- 类之间的依赖关系：

```
class A{
    public void Function(B b)
    {}
}
class B {
}
```

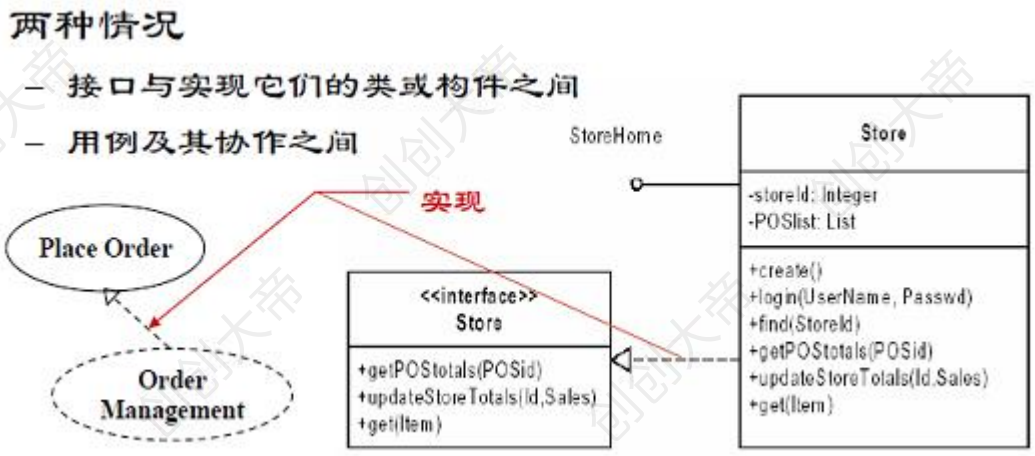
- 类之间的关联关系：

```
class A{
    B b = new B();
}
class B {
}
```

E 实现

实现是类元之间的语义关系，其中的一个类元指定了由另一 个类元保证执行的契约。

UML 图：



2..UML 的构成：

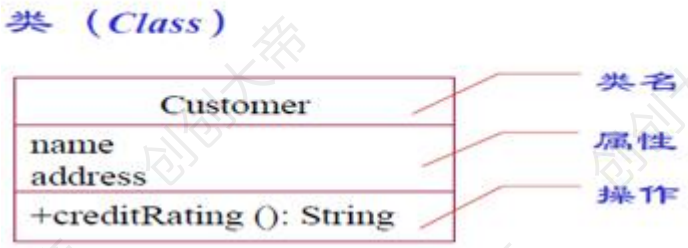
重点掌握基本构造块

3.UML 事物：

A **结构事物**

UML 符号：

1



2 接口：

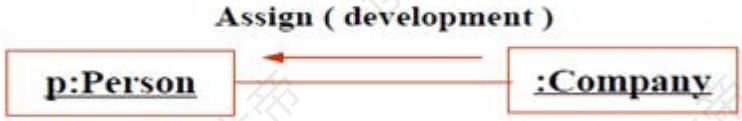
3

协作

B 行为事物

UMl 符号：

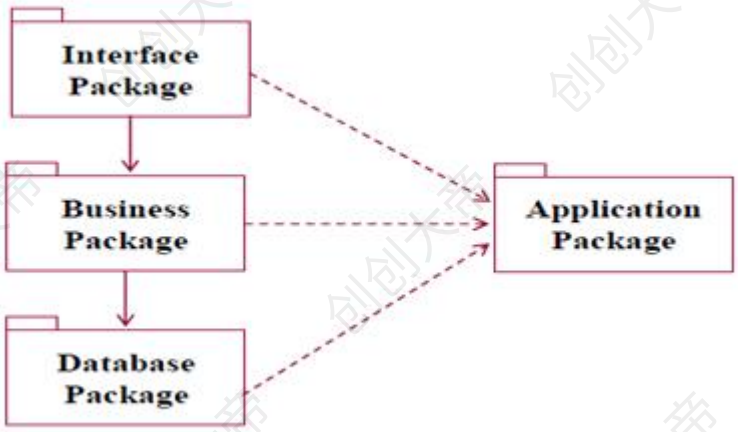
交互：



C 分组事物

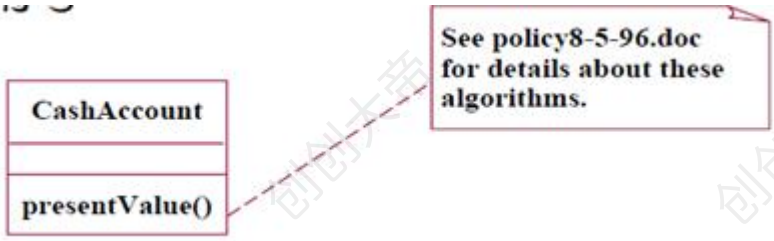
UMl 符号：

包：



D 注释事物

UMl 符号



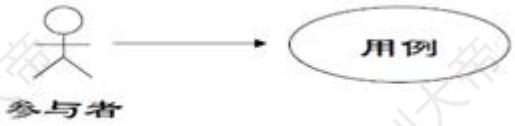
4.UML 图：（9 种图及其作用）

A．用例图

用例图定义了系统的功能需求，它完全是从系统外部观看系统功能，并不描述系统内部对功能的具体实现。

用例图表示了用例、参与者及其它们之间的关系。

UML 符号：



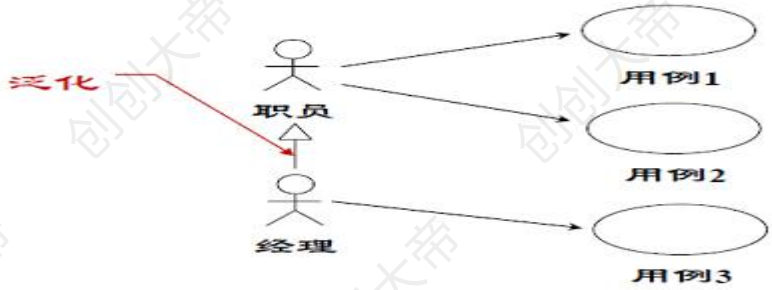
■ 参与者（***Actor***）

– 参与者是与系统交互的外部实体。

– 参与者既可以是使用该系统的用户，也可以是与系统交互的 其他外部系统、硬件设备或组织机构。

■ 参与者的泛化关系

– 参与者之间可以存在泛化的关系，类似的参与者可以利用泛化关系组成一般与特殊的层次结构。



■ 用例（***Use Case***）

– 用例是从用户角度描述系统的行为，它将系统的一个功能描述成一系列动作序列，这些动作最终对参与者产生有价值的可观测结果。

– 用例可以促进与用户沟通，理解正确的需求，同时也可以用 来划分系统与外部实体的界限。

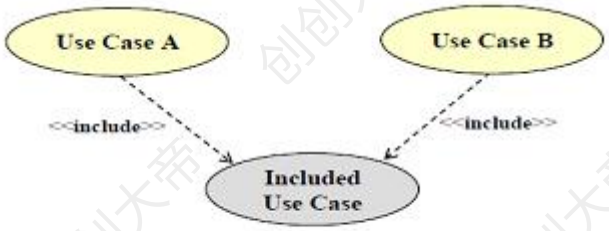
• 用例之间的关系（与书中不同）

– 包含关系（***include***）

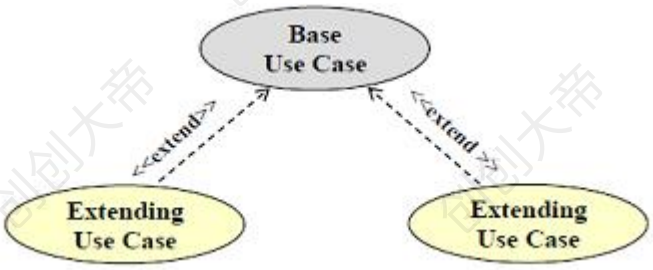
– 扩展关系（***extend***）

泛化关系（***generalization***）

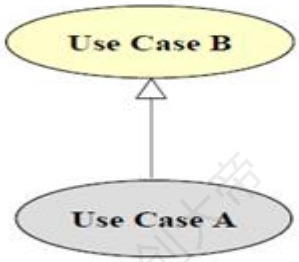
包含关系



扩展关系



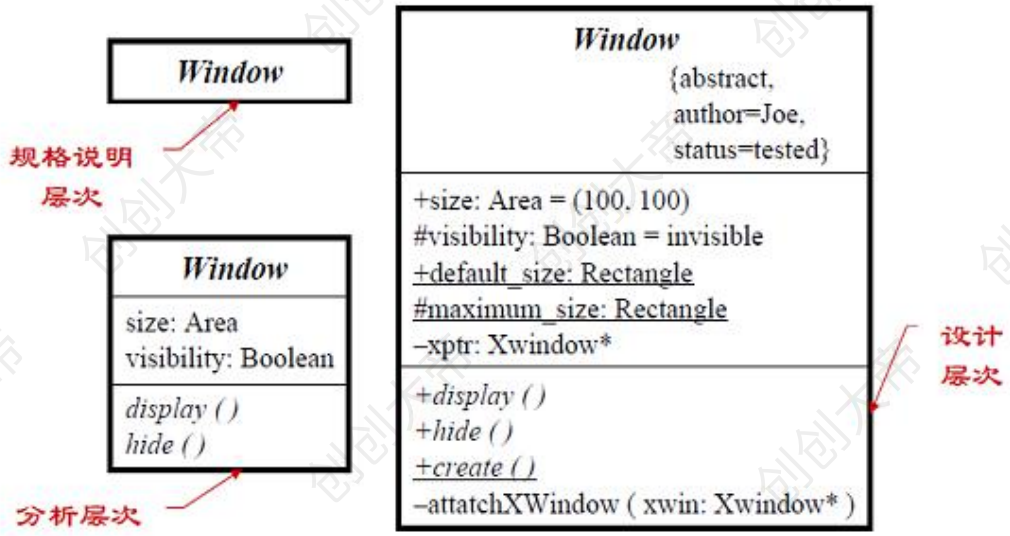
泛化关系



B 类图

类图描述系统的静态结构，表示系统中的类、类与类之间的关系以及类的属性和操作。

- 类在不同层次上的 UML 符号表示



C 包图：

包一般用于较大系统组块的建模

UMl 符号（P４２图３．１３）：

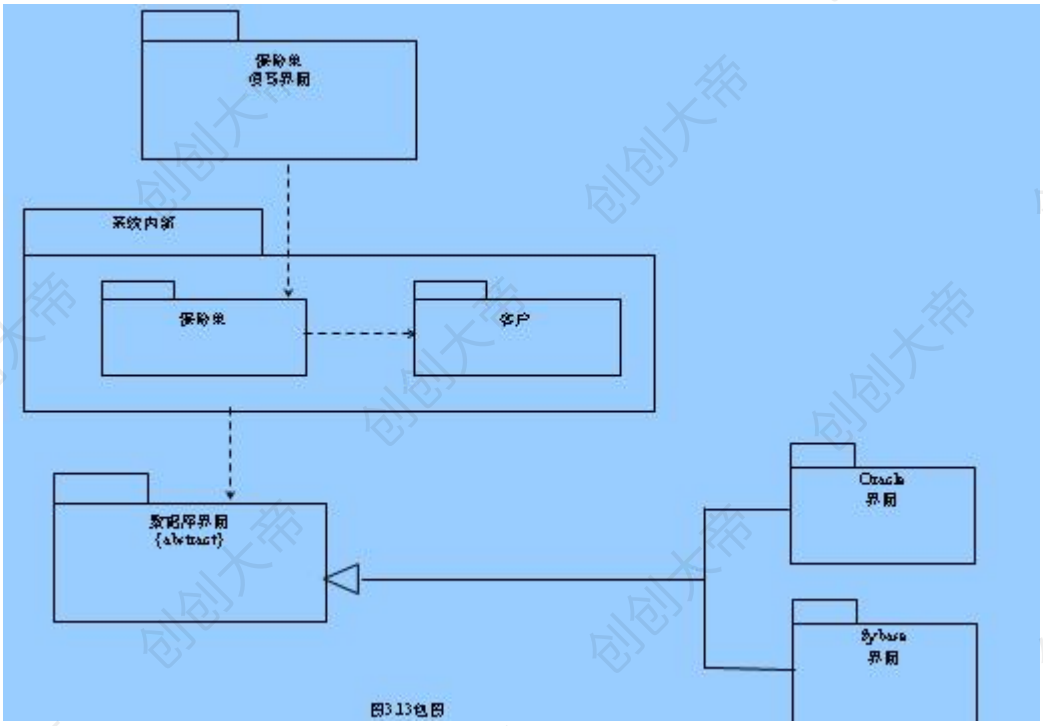


图3.13包图

D　组件图：

组件图描述软件组件及组件之间的依赖关系，显示代码的静态结构。

UMl 符号（P４３图３．１４）

E部署图

- 部署图也称实施图，用于建模处理器、硬件设备和软件组件在运行时的架构
- 它可以用来描述系统中计算结点的拓扑结构、通信路径与结点上运行的软组件等。
- 一个系统模型只有一个部署图，部署图常常用于帮助理解分布式系统。

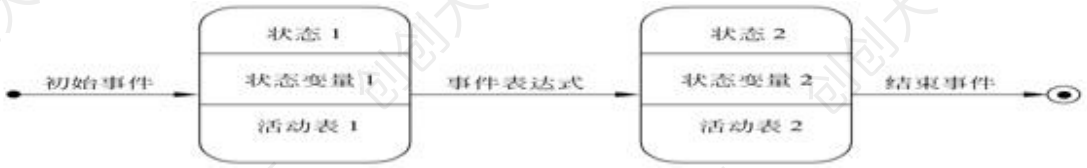
UMl 符号：（ｐ４３图３．１５）

F状态图：

状态图是由状态机扩展而来的，用来描述对象所经过的对外部事件做出相应的状态序列。

状态图针对单个对象建立模型。侧重于描述某个对象在其生命周期中的动态行为

UMl 符号：



G 顺序图：

顺序图描述了一组交互对象间的交互方式，它表示完成某项行为的对象和这些对象之间传递消息的时间顺序。

–　一般情况下，我们使用顺序图描述一个用例的事件流，标识参与这个用例的对象，并以服务的形式将用例的行为分配到对象上。

UMl 符号：

- 顺序图的建立步骤：

1〉 确定交互的上下文。

2〉 找出参与交互的对象类角色，把他们横向排列在顺序图的顶部，最重要的对象安置在最左边，交互密切的对象尽可能相邻。在交互中创建的对象在垂直方向应安置在其被创建的时

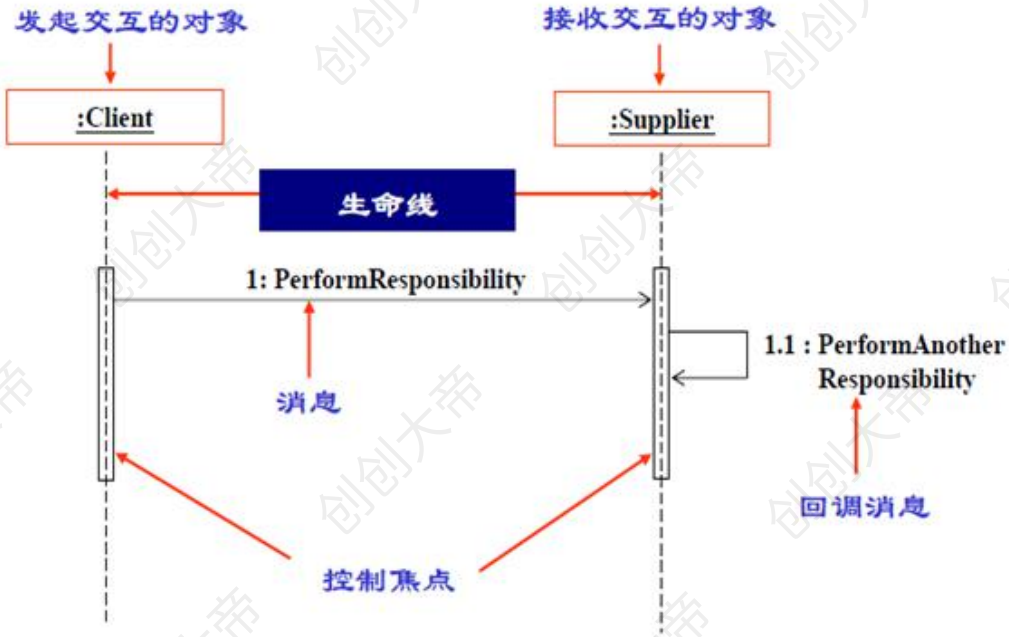
间点处。

3〉 对每一个对象设置一条垂直的向下的生命线。

4〉 从初始化交互的信息开始，自顶向下在对象的生命线之间安置信息。注意用箭头的形式区别同步消息和异步消息。根据顺序图是属于说明层还是属于实例层，给出消息标签的内容，以及必要的构造型与约束。

5〉 在生命线上绘出对象的激活期，以及对象创建或销毁的构造型和标记。

6〉 根据消息之间的关系，确定循环结构及循环参数和出口条件。



■ 顺序图的建立步骤：

1〉 确定交互的上下文。

2〉 找出参与交互的对象类角色，把他们横向排列在顺序图的顶部，最重要的对象安置在最左边，交互密切的对象尽可能相邻。在交互中创建的对象在垂直方向应安置在其被创建的时间点处。

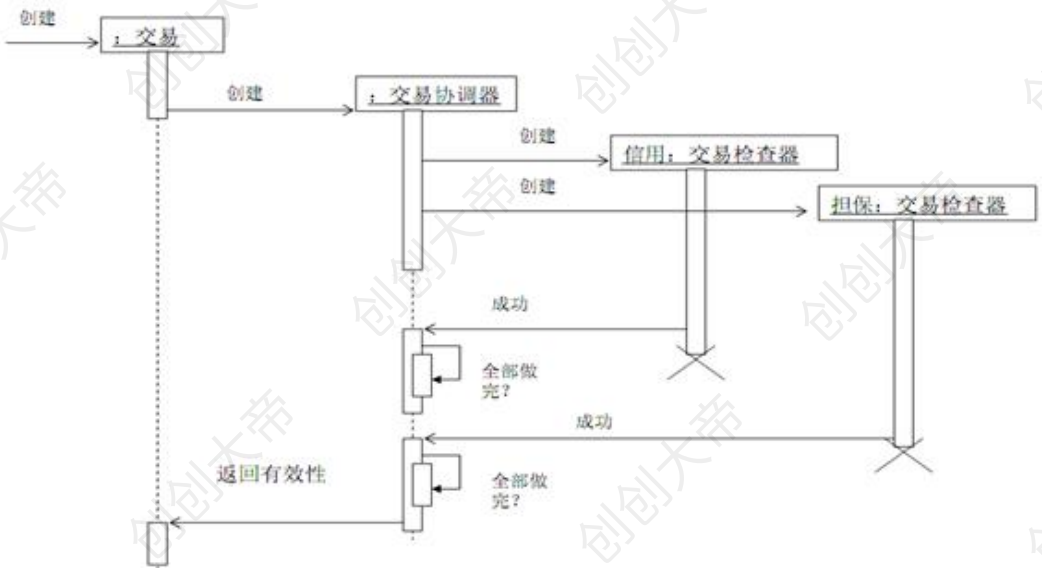
3〉 对每一个对象设置一条垂直的向下的生命线。

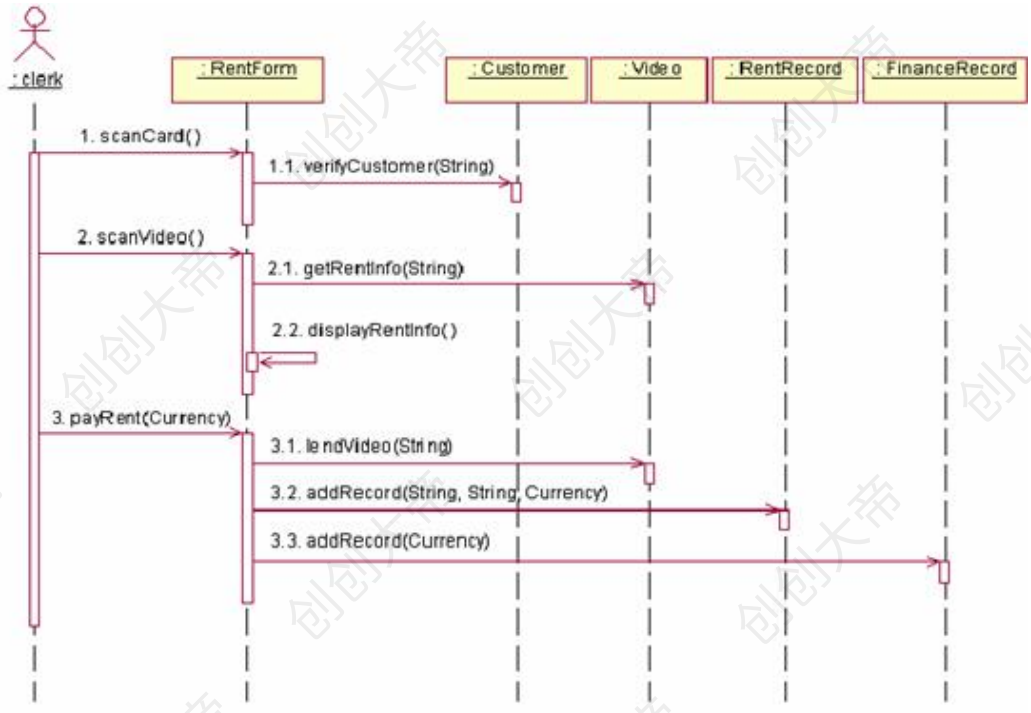
4〉 从初始化交互的信息开始，自顶向下在对象的生命线之间安置信息。注意用箭头的形式区别同步消息和异步消息。根据顺序图是属于说明层还是属于实例层，给出消息标签的内容，以及必要的构造型与约束。

5〉 在生命线上绘出对象的激活期，以及对象创建或销毁的构造型和标记。

6〉 根据消息之间的关系，确定循环结构及循环参数和出口条件。

例题 银行系统的交易验证

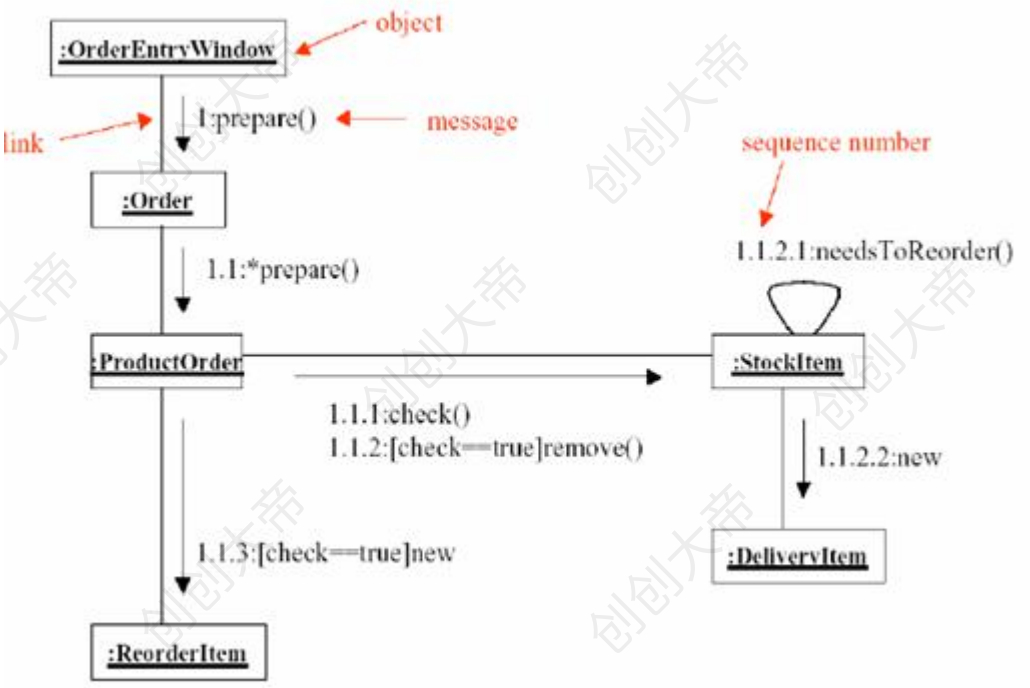




H 协作图：

协作图反映收发消息的对象的结构组织，用于描述系统的行为是如何由系统的成分协作实现的。

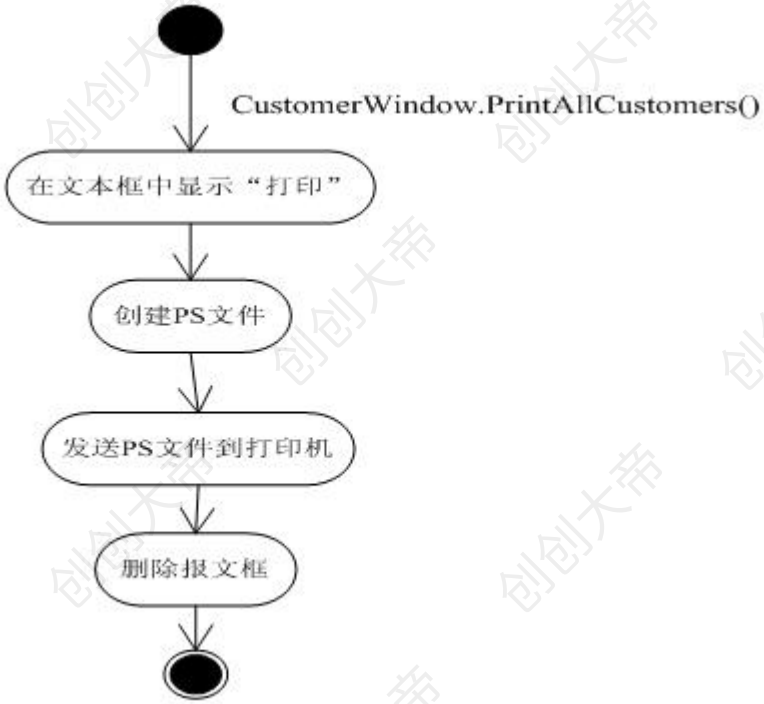
UMl 符号：



I 活动图：

- 活动图用于对系统工作流程建模，即对系统的业务过程建模，也可以对具体的操作建模，以描述计算过程的细节。
- 例如：算法流程、软件过程活动

UML 符号：



第 4 章 需求工程

1 软件需求分类：业务需求、用户需求、系统需求、功能需求、非功能需求

2．5 个需求的定义：

（1）业务需求是组织或客户对于系统的高层次目标要求， 定义了项目的远景和范围，即确定软件产品的发展方向、功能范围、目标客户和价值来源。

(2) 用户需求是从用户角度描述的系统功能需求和非功能需求，通常只涉及系统的外部行为，而不涉及系统的内部特性。

（3）系统需求是更加详细地描述系统应该做什么，通常包括许多不同的分析模型，诸如对象模型、数据模型、 状态模型等。

（4）功能需求

– 描述系统应该提供的功能或服务，通常涉及用户或外部系统 与该系统之间的交互，一般不考虑系统的实现细节。

（5）非功能需求

从各个角度对系统的约束和限制，反映了应用对软件系统质量和特性的额外要求，例如响应时间、数据精度、可靠性、 开发过程的标准等。

3 需求工程过程：

对需求的收集、分析、定义和验证

4 需求获取技术/方式：

A 户交流

B 用例的需求获取

需求获取的步骤：

确定参与者（Actor）

确定用例

建立用例模型

描述用例

C 原型化方法

5 需求验证围绕的质量特性

- 围绕正确性、二义性、完整性、可验证性、一致性、可修改性、可跟踪性、实现性检验和需求的可验证性展开（P71）

第 5 章 面向对象的分析

1. 面向对象的分析的过程是找出分析类、分析包，构建它们之间的静态关系和动态协作模型的过程

2 三个分析类的定义及作用：

实体类：表示系统存储和管理的永久信息

边界类：表示参与者与系统之间的交互

控制类：表示系统在运行过程中的业务控制逻辑

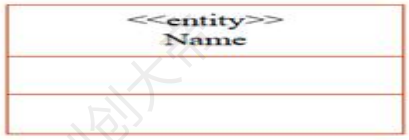
实体类

描述必须存贮的信息及其相关行为

实体类的主要职责是存储和管理系统内部的信息，它也可以有行为，甚至很复杂的行为，但这些行为必须与它所代表的实体对象密切相关

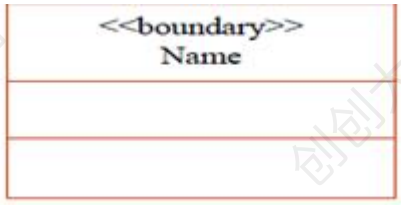
通常对应现实世界中的“事物”，如人，银行账号等

UML 表示：



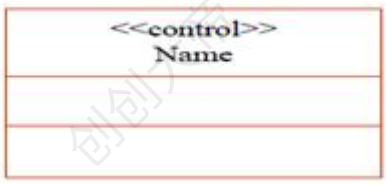
简写：

边界类：
描述外部的参与者与系统之间的交互
这种交互通常包括来自用户和外部系统的信息与请求，以及将信息与请求提交到用户和外部系统
每个边界类至少应该与一个参与者有关
类型：用户界面、系统接口、设备接口
UML 表示



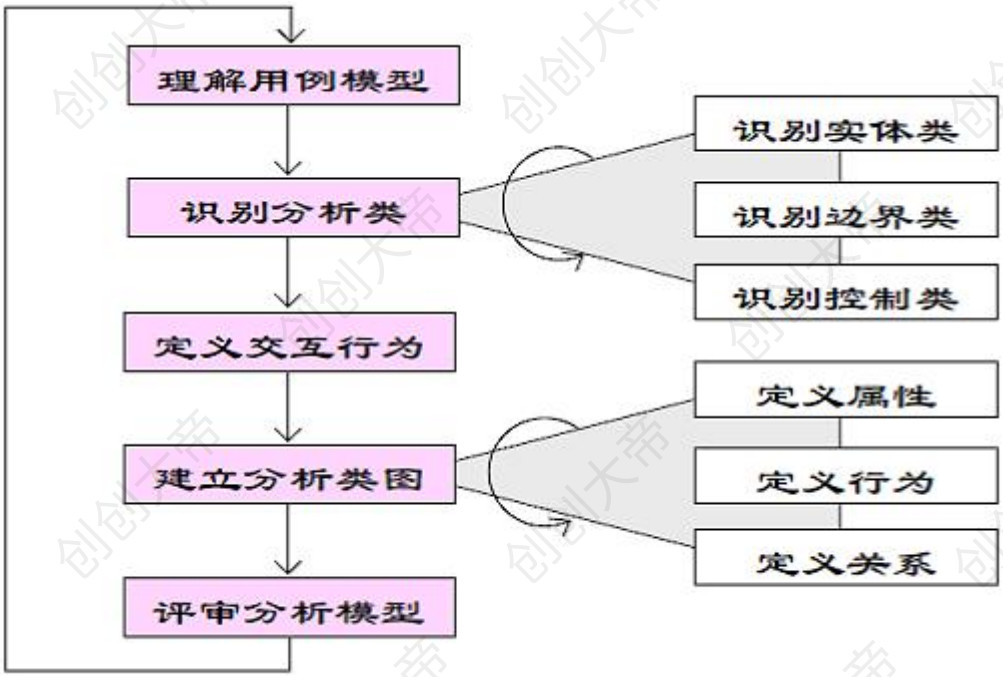
简写

控制类
描述一个用例所具有的事件流控制行为，控制一个用例中的事件顺序 。
实现对用例行为的封装，将用例的执行逻辑与边界和实体进行隔离
UML 表示



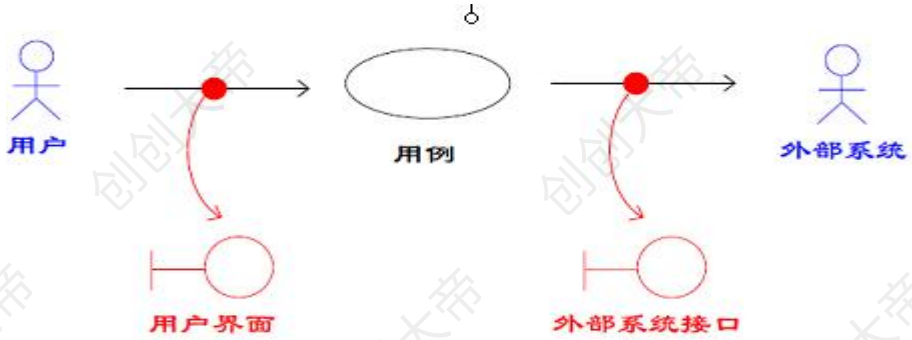
简写：

3. 基于 UML 的需求分析的步骤



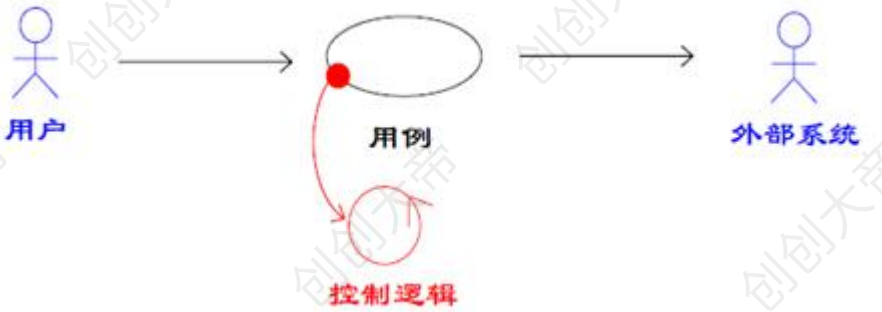
4. a 识别边界类：

– 通常，一个参与者与一个用例之间的交互或通信关联对应一个边界类。



B 识别控制类

– 控制类负责协调边界类和实体类，通常在现实世界中没有对应的事物。
– 一般来说，一个用例对应一个控制类。
控制类在一个用例开始时被创建，用例结束时就不再存在



C 识别实体类

– 实体类通常是用例中的参与对象，对应着现实世界中的“事物”,一般通过分析用例描述和词汇表等发现备选的实体类。

例：如何识别 ***MiniLibrary*** 的实体

实体类	说 明
BorrowerInfo	普通读者的基本信息
Loan	普通读者的借书记录
Reservation	普通读者的预订信息
Title	图书资料的基本信息
Item	书目
（由于图书资料中包括书籍和杂志等类型，因此可以进一步划分子类）	
BookItem	书籍的基本信息
MagazineItem	杂志的基本信息

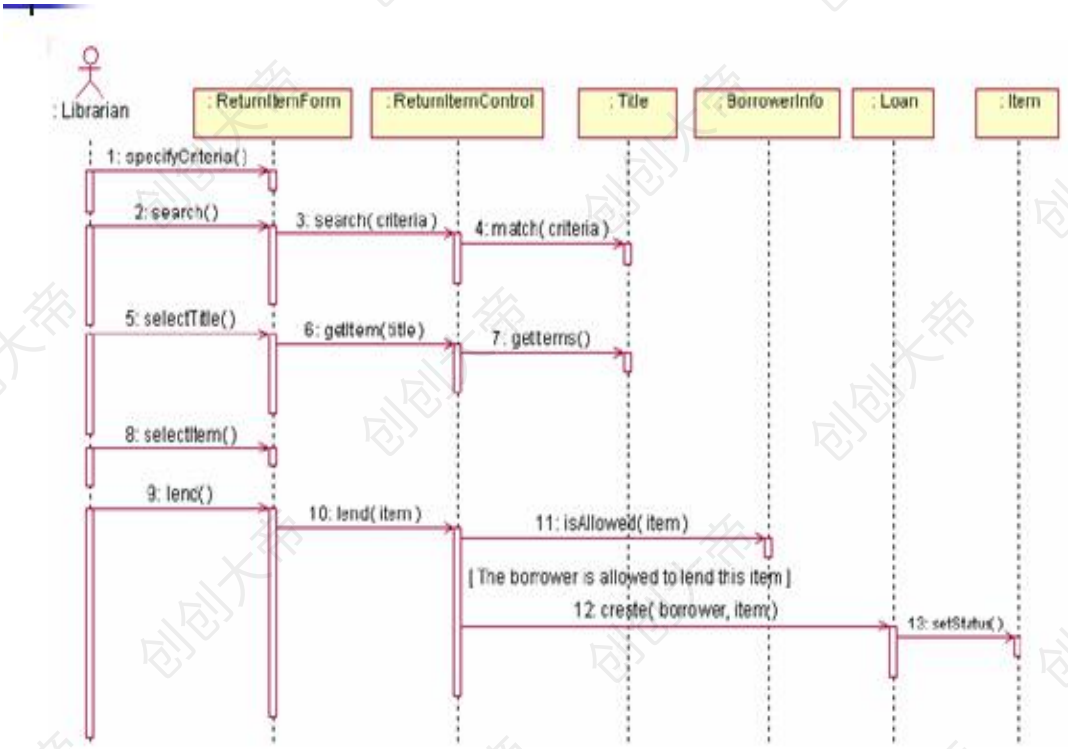
此章会有一道大题

课堂作业（一定要掌握顺序图的绘制方法）

以登记借书为例绘制顺序图

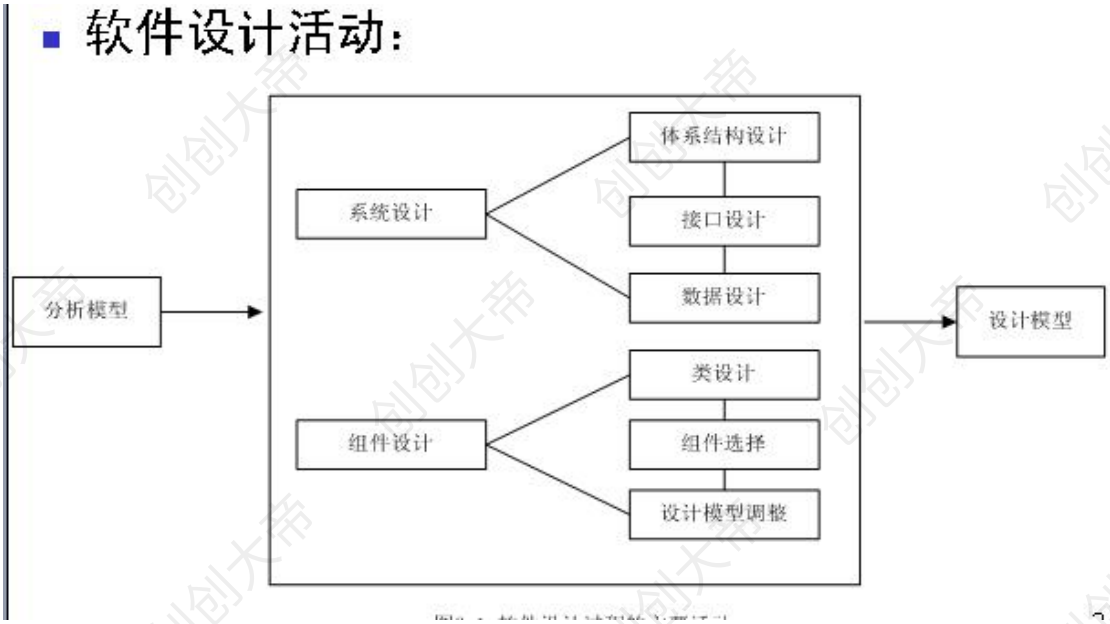
1. 列出启动该用例的参与者： 图书管理员
2. 列出该用例使用的边界对象： 登记借书界面
3. 列出该用例使用的实体类： 图书资料的基本信息，读者的基本信息，读者的借书记录，书目
4. 列出管理该用例的控制对象： 登记借书控制
5. 根据用例描述的流程，按时间顺序列出分析对象之间进行消息访问的序列

答案：



第 6 章 软件设计

1 软件是如何进行设计（P90 图 6.1）



数据设计和接口设计（p90）

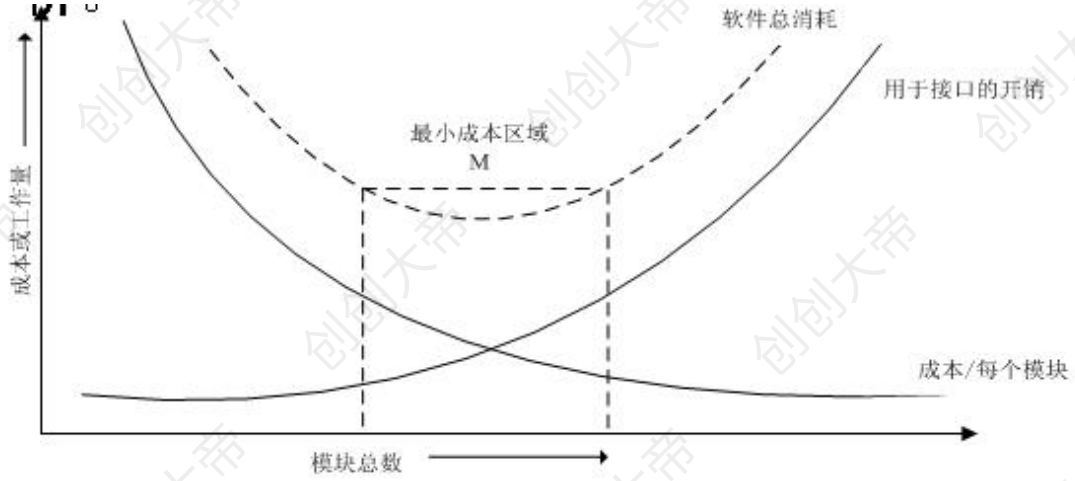
2. 软件设计原则 4 个

A．模块化和信息隐蔽

思考为什么要模块化？是不是模块数量越多越好？

答：1 将一个复杂的大系统模块化（Modularity），即将大系统分解成若干个相对简单的子系统（Subsystem），分解后系统的复杂性和工作量比一个大系统要小，实现更为容易。2 模块的分解并非越多越好，模块之间存在着交互接口，当模块过多时，将会增加接口

的代价。



B 内聚和耦合

内聚：

内聚是子系统内部的相关程度

作用：高内聚的方法做且仅做一件事，这会很容易理解与维护

高内聚的类表示且仅表示一种类型的对象。

- 内聚度标志一个模块内部各成分彼此结合的紧密程度。
- 内聚度按其高低程度可分为 7 级，从低至高分别为
  - 偶然内聚（Coincidental cohesion）
  - 逻辑内聚（Logical cohesion）
  - 时间内聚（Temporal cohesion）
  - 过程内聚（Procedural cohesion）
  - 通信内聚（Communicational cohesion）
  - 顺序内聚（Sequential cohesion）
  - 功能内聚（Functional cohesion）
- 内聚度越高越好

耦合：

耦合表示两个子系统（或类）之间的关联程度。

– 当一个子系统（或类）发生变化时对另一个子系统（或类） 的影响很小，则称它们是松散耦合的；反之，如果变化的影响很大时，则称它们是紧密耦合的。

– 耦合也可分为 7 级，从低至高为：
  - 非直接耦合（Nondirect coupling）
  - 数据耦合（Data coupling）
  - 标记耦合（Stamp coupling）
  - 控制耦合（Control coupling）
  - 外部耦合（External coupling）
  - 公共耦合（common coupling）
  - 内容耦合（Content coupling）

– 耦合度应越低越好。

耦合的原则：

- 一般来说，设计时应尽量使用数据耦合，减少控制耦合，限制外部环境耦合和公共

耦合，绝对禁止内容耦合。

C **抽象和求精（了解）**

- 抽象是对问题的简化和概括，它忽略了问题的某些细节，有助于把握问题的本质。
- 求精（Refine）是抽象的逆过程，是对问题自顶向下逐步分解、细化逐渐至细节的过程。

D．复用

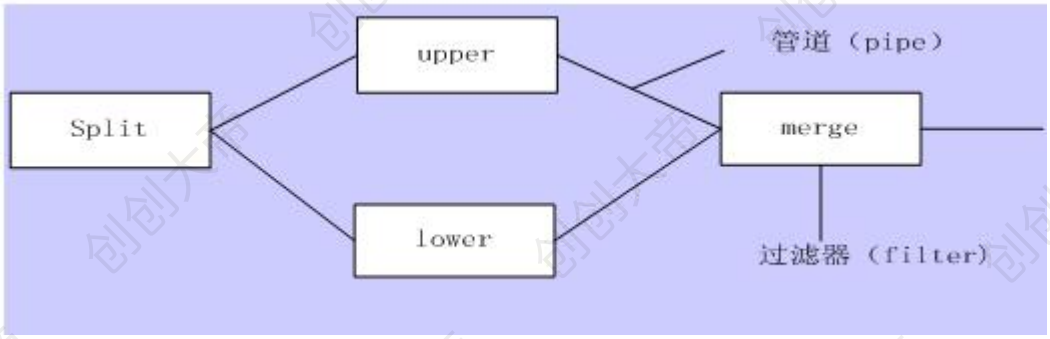
复用策略

- 工具包强调在**类级别**的代码复用，如类库；
- 组件强调**组件级**的复用，如 ***CORBA***、***EJB***、***DCOM*** 等；
- 设计强调**设计级**的复用，如复用设计模式，复用体系结构等

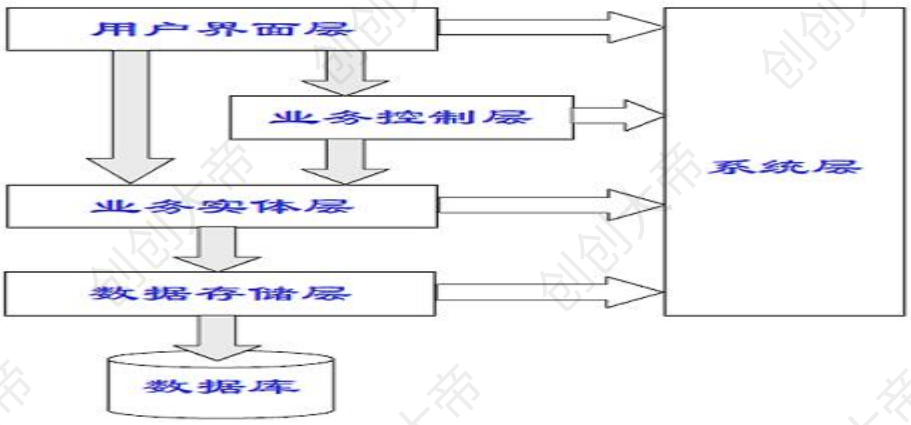
3 五种常见的体系结构及其体系结构图（P99-102）

管道—过滤器结构

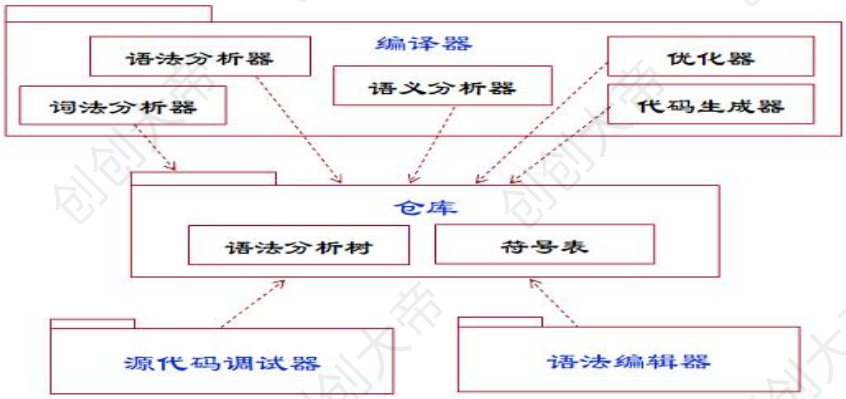
过滤器的活动可以通过以下方法激活：
（1）后续的部件从过滤器中取出数据；
（2）前续的部件向过滤器推入新输入数据；
（3）过滤器处于活跃状态，不断地从前续部件取出并向后续部件推入数据。
前两种情况产生的是被动式过滤器（passive filter），最后的是主动式过滤器（active filter）。



分层体系结构

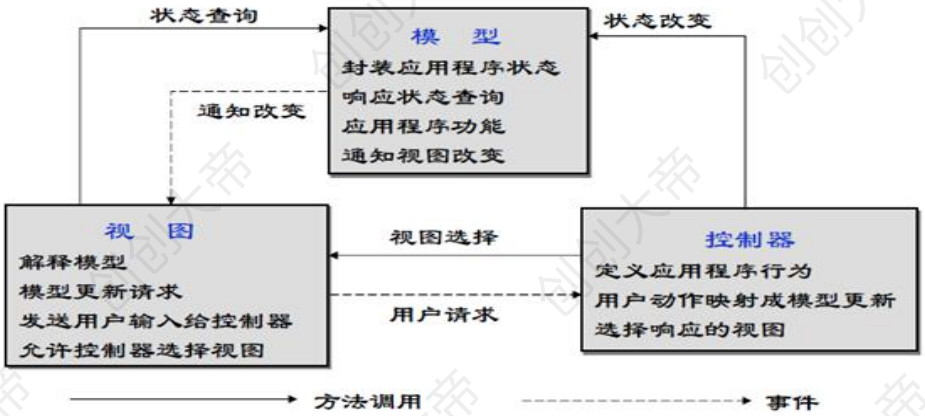


仓库系统结构



客户/服务器模式 （ATM 是胖客户，网站是瘦客户）

MVC 模式 （MVC 的核心）



第七章面向对象的设计

1 设计模型是设计的主要结果，是系统实现的抽象，因而用作系统实现活动的基本输入

2 **识别设计类（p108）**

对于边界类的设计依赖于所采用的特定的接口技术。接口技术不同，设计类不同。

- 实体类用于表示持久信息，因此实体类的设计和特定的数据存储或数据库技术相关。
- 控制类的设计较为复杂，它们封装了控制逻辑、与其它对象的协作以及业务逻辑问题。

3 识别类方法：一般，设计类的方法首先可由分析类的职责得到（需细化），其次，遍历用例实现和画状态图，找遗漏的方法

4 别类属性：设计类的属性可继承自分析类，有时，分析类的属性隐含着设计类需要的一个或多个属性，一个类的属性不能被多个设计对象共享

5 识别类关系：找设计类之间的关联（包括聚合和组合），依赖，泛化关系，可从分析模型中得到

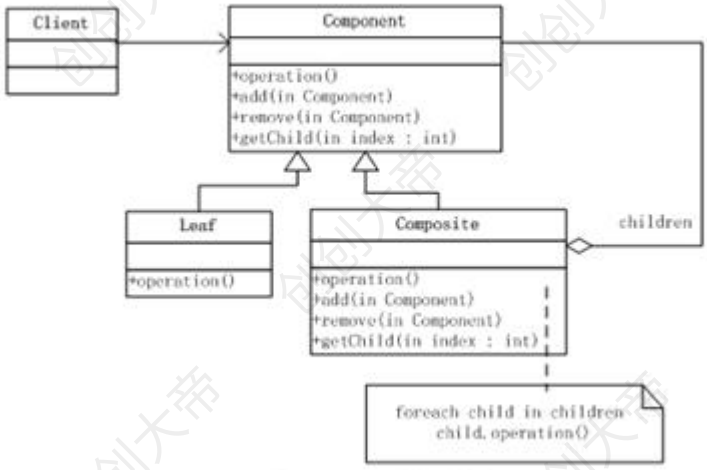
6 识别完设计类后，需要识别设计类的方法，识别类属性，识别类关系，最后建模设计类图，产生面向对象设计结果

第九章

掌握三种设计模式（p147-154）

Composite 模式适用于以下两种情况：

- （1）表示对象的整体-部分层次结构；
- （2）希望可以忽略组合对象和单个对象的不同。
- 针对系统中存在单个对象和组合对象的情况，Composite 模式模糊了单个对象和组合对象的概念，客户程序可以像处理单个对象一样来处理组合对象



Abstract Factory 模式

– 该模式适用于封装具体平台，使应用可在不同平台上运行。

特点：提供一个创建一系列相关或相互依赖对象的接口，而无需指定它们具体的类

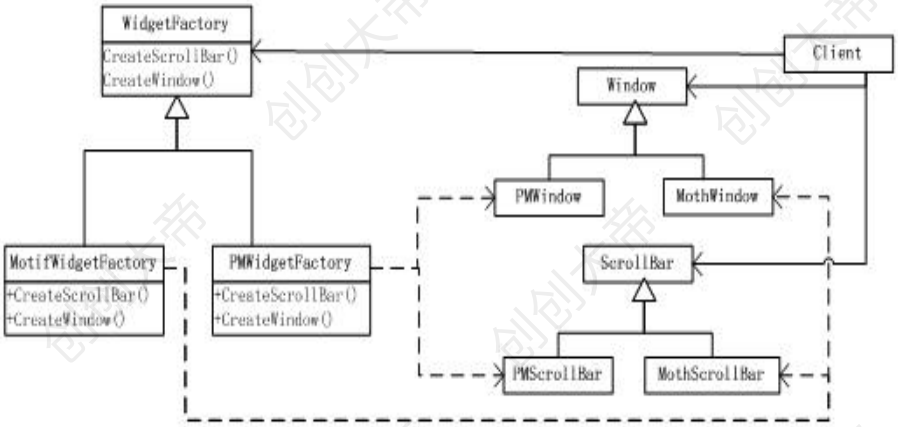
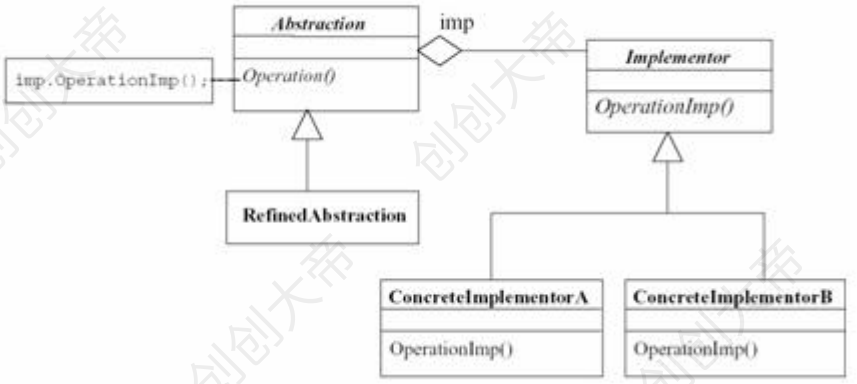


图9.12 Abstract Factory模式示例

Bridge 模式

– 该模式将一个类的接口与具体实现进行分离。适用于存在一个维度以上的变化的情况下

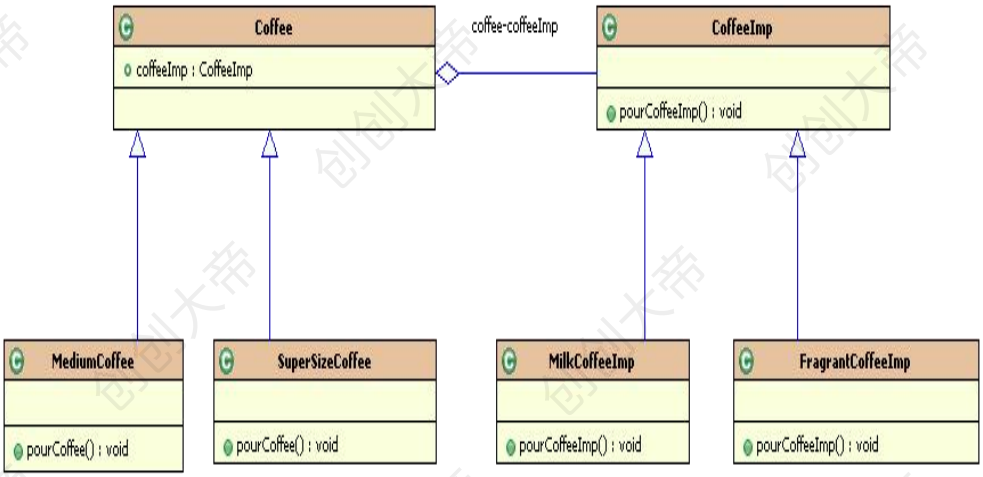


例题：

■ 咖啡分为大杯和中杯两种型号，每种型号又可分为加奶和不加奶两种口味）。

1. 设计咖啡类（咖啡基类为抽象类，用泛化关系实现型号和口味）
2. 用 Bridge 模式优化 1。
3. 如果咖啡型号加一个小杯，重新实现 1，2，了解 Bridge 模式的好处。

利用桥模式的图



第十一章

1. 实现是否等同于编码？

- 正确观点
  - 软件编码是一个复杂而迭代的过程，包括程序设计和程序实现。

2. 在编码过程中强调风格还是规范？

强调规范。编码风格是指编程时表现出来的特点、习惯、逻辑思维等。

软件编码规范

– 与特定语言相关的描写如何编写代码的规则集合

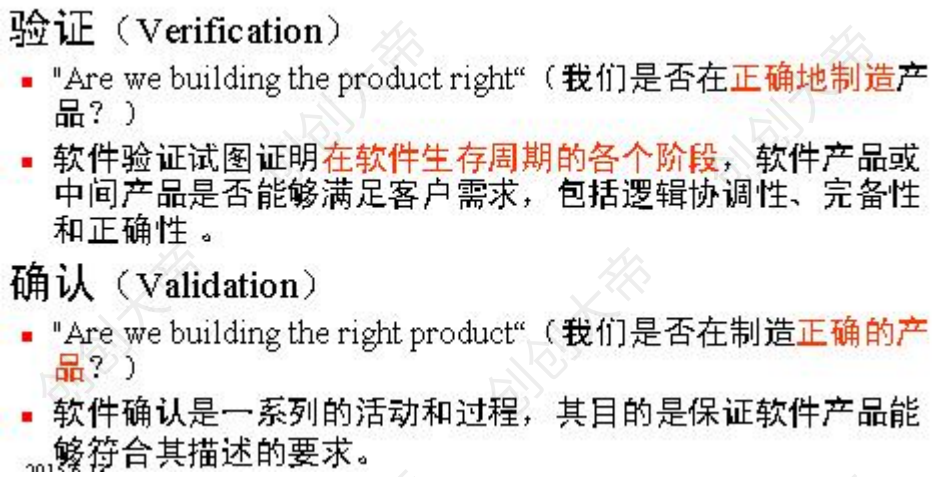
软件编码更多地是一种工程，而不是一种个人艺术。如果不统一编程规范，最终写出的程序，其可读性将较差，这不仅给代码的理解带来障碍，增加维护阶段的工作量，同时不规范的代码隐含错误的可能性也比较大。

第十二章

1，验证和确认（Verification & Validation，简称 V&V）工作是在**整个软件生命周期**中对软件的规范性评估活动，以保证软件开发各个环节的正确性。

2 验证与确认对比：

验证和确认是两个相互独立但却相辅相成的活动，二者很容易混淆



3.

- 验证强调对于过程的检验！
- 确认强调对于结果的检验！

第十四章

1. 软件项目管理是为了使软件项目能够按照预定的成 本、进度、质量顺利完成，而对成本、人员、进度、 质量、风险等进行分析和管理的活动。降低复杂性和控制变化是软件项目管理的关键问题

2 软件项目计划过程：

（1）确定软件范围

（2）估算项目（成本和进度）

（3）编制项目进度表

3 **成本估算**

软件规模估算：

采用代码行 LOC（Line of code）来表示，可以间接采用功能点 FP（Function points）或者对象点 OP（Object points）来测算。

功能点计算

信息域特征	估算值	加权因子				单项总和
		简单	中等	复杂		
外部输入		× 3	4	6	=	
外部输出		× 4	5	7	=	
外部查询		× 3	4	6	=	
内部逻辑文件		× 7	10	15	=	
外部接口		× 5	7	10	=	
未调整功能点总计 UFP						

每种因素的复杂度调整值 Fi：取值 0..5

- 0: 无影响，1: 偶然，2: 适中，3: 普通，4: 重要，5: 极重要

因子	影响	因子	影响
数据通信		联机更新	
分布式处理		复杂处理	
性能		可重用性	
配置负载		易安装	
事务率		易操作	
联机数据登录		多个场所	
最终用户效率		设施变更	

计算公式 $FP = 总计数 \times [0.65 + 0.01 \times \sum F_i]$

功能点的计算例题：

某公司大约有 3000 名员工，准备开发一个简单的工资系统

- 系统生成员工的工资单，并在屏幕上分别显示工资单的收入项和纳税扣除额，工资单的功能复杂度是“复杂”；另外，系统产 生7个报表，每个报表的复杂度是“简单”。
- 系统要求用户从屏幕上输入员工的基本信息（包括员工编号、基本 工资、所在等级、所属部门等）和每月的考勤情况，这两个屏幕输 入的复杂度为“复杂”；另外，还有一个所得税信息的输入，其复杂度 为“中等”。
- 系统提供20个查询，每个查询的复杂度是“简单”。
- 系统内部维护一个员工信息文件，该文件的复杂度是“复杂”。
- 系统引用了3个数据表，包括员工基本信息、部门信息和所在等级，其中员工基本信息的复杂度是“中等”，其他两个的复杂度是“简单”。

1. 计算未调整功能点

	低	平均	高	合计
外部输入	0×3	1×4	2×6	16
外部输出	7×4	0×5	2×7	42
外部查询	20×3	0×4	0×6	60
内部逻辑文件	0×7	0×10	1×15	15
外部接口文件	2×5	1×7	0×10	17
未调整的功能点数：				150

2. 计算调整后的功能点

因子	影响值	因子	影响值
数据通信	0	联机更新	0
分布式处理	0	复杂处理	2
性能	3	可重用性	0
配置负载	4	易安装	5
事务率	0	易操作	5
联机数据登录	2	多个场所	0
最终用户效率	5	设施变更	1

$$FP = 150 \times [0.65 + 0.01 \times \Sigma Fi] = 138$$

0: 无影响，1: 偶然，2: 适中，3: 普通，4: 重要，5: 极重要

4. COCOMO 模型

■ COCOMO 81

定义了“基本的”、“中间的”和“详细的”三种形式的 COCOMO 模型，其两个核心公式为：

其中，*S* 为千行源代码数 *KLOC*，经验常数 *r*，*c*，*a* 和 *b* 取决于项目的总体类型。

工作量（人月 *PM*）：$ED = rS^c$

开发时间（月）：$TD = a(ED)^b$

成本=工作量*开发时间

主要掌握：

开发类型	工作量	进度
半独立型	$ED=3.0S^{1.12}$	$TD=2.5(ED)^{0.35}$

软件项目计划的主要内容包括确定软件范围，确定需要进行哪些活动，明确每项活动的职责，明确这些活动的完成顺序，估算资源、成本和进度，制定项目计划，编排进度等。（p239）
此章会有一道大题应该和上面例题类似

第一章 导论

1. 软件工程关注于开发成本和软件质量问题
2. 软件的 3 个特性，以及补充的四个本质特性
   a) 软件不会被磨损，并不意味着软件容易修改
   b) 所有成功的软件都会发生变更，并不意味着变更越多，软件越成功
3. 软件的类型
   a) 从软件的功能角度分为系统软件和应用软件
   b) 从服务对象的角度分为通用软件和定制软件，能区分什么是通用软件和定制软件
4. 软件质量的定义？
   a) 如何评判软件质量的好坏？
      i. 软件质量通常采用质量模型来建立软件质量特性间的关系
      ii. 常见的有如 Bohem 质量模型，McCall 的质量模型，ISO 的质量模型
      iii. 用 McCall 的质量模型判断软件质量，给出了哪 11 个质量要素，分哪三类？
      iv. 为了让 11 个质量要素定量化，给出了 20 质量评价准则

   “运行正确的软件就是高质量的软件。”正确吗？
5. 软件危机是指在计算机软件的开发和维护过程中遇到的一系列严重问题
6. 人们在 **20** 世纪 **60** 年代末普遍认识到“软件危机 ”的存在。
7. 软件工程的定义（IEEE）？
8. 软件工程至今尚未解决大型复杂软件开发的问题
   a) 软件工程是一个正在兴起的年轻学科
9. 软件工程的三要素？
   a) 软件工程方法提供如何构造软件的技术
   b) 工具提供自动化或半自动化的支持。
   c) 软件工程过程是用合适的方法、语言和工具由软件工程师进行软件活动的集合。即确定软件开发分为哪些过程活动，活动之间的先后次序等。
10. 软件工程的方法：典型的两类？
    a) 结构化分析的基本原则是分解和抽象
    b) 结构化分析的结果是 DFD 图
    c) 结构化设计时将 DFD 转化为系统结构模型（功能模块图）
    d) 面向对象方法：OOA,OOD,OOP, OOT,OOSM
    e) 分析类和设计类是多对多的
11. 工具的集合称之为计算机辅助软件工程，简称 CASE（Computer-Aided Software Engineering，计算机辅助软件工程）
12. CASE 分成三个层次：工具、工作台和集成的软件开发环境
    a) 工具往往完成软件过程活动中的单一任务
    b) 工作台支持软件过程某个阶段的活动
    c) 集成的软件开发环境则是支持整个软件开发过程中所有活动或核心活动
13. 软件工程知识体系（**SWEBOK**），将软件工程知识分解成层次化的 10 个知识域

第二章

1. 软件过程可概括为三类 P18
2. 常见的六种过程模型
   a) 模型图（区分模型图）
   b) 使用范围
   c) 优缺点
3. **瀑布模型**

   适用：在开发的早期阶段软件需求被完整确定

   缺点：

   各个阶段的划分完全固定，阶段之间产生大量的文档，极大地增加了工作量

   用户只有等到整个过程的末期才能见到开发成果，从而增加了开发的风险

   开发过程中很难响应客户的变更要求

   早期的错误可能要等到开发后期的测试阶段才能发现，进而带来严重的后果

4. **演化式开发模型**

适用：软件需求不确定，常用于命周期较短、中小规模的系统。当然在大型系统中，视具体情况部分采用演化式开发也是可行的。

优点

在及时响应用户需求改变方面比瀑布模型更为有效，软件改变的代价较小

缺陷

- 系统的结构通常不好
- 软件过程可见性不好
- 开发过程通常需要特殊的工具和技术支持

5. **形式变换模型**

适用

– 特别适合于那些对安全性、可靠性和保密性要求极高的软件系统，这些系统需要在投入运行前进行验证

• 优点

– 由于数学方法具有严密性和准确性，形式化方法开发过程所交付的软件系统具有较少的缺陷和较高的安全性

由 IBM 建议的净室开发过程（Cleanroom）是一个典型的形式化开发过程

6. **面向复用的开发**

主要建立在已有大量可利用的软件组件和组件的集成框架基础上，可以有效提高开发效率和质量

优点

– 充分体现软件复用的思想

– 实现快速交付软件

• 缺点

– 商业组件的修改受到限制，影响系统的演化

7. **增量开发**

是将系统的需求定义、设计和实现分解成若干增量依次开发和交付，以减少开发过程中的返工，也可以让用户通过体验先期交付的系统更准确、详细地给出需求。

- 优点

– 整个产品被分解成若干个构件逐步交付，用户可以不断地看到所开发软件的可运行中间版本
– 将早期增量作为原型有助于明确后期增量的需求
– 降低开发风险
– 重要功能被首先交付，从而使其得到最多的测试

- 缺点

– 需要软件具备开放式的体系结构
– 需求难以在增量实现之前详细定义，因此增量与需求的准确映射以及所有增量的有效集成可能会比较困难
– 容易退化为边做边改方式，使软件过程的控制失去整体性

8. **螺旋模型**
   a) 将瀑布模型和快速原型模型结合起来，其过程活动不是按顺序进行而是以螺旋式展开，强调了其他模型所忽视的风险分析，特别适合于大型复杂的系统。

9. Rational 统一过程 RUP（Rational unified process）强调开发和维护模型
   其软件生命周期在时间上被分解为四个连续的阶段：分别是？
10. 敏捷开发是由业界专家针对企业现状提出的让软件开发团队具有快速工作、响应变化能力的价值观和原则，

第三章

1. 对象类之间的联系
   - 分类结构：一般与特殊的关系。在面向对象术语中为泛化
   - 组成结构：部分与整体的关系。在面向对象术语中为聚合和组合
   - 实例连接：对象类之间的静态联系。在面向对象术语中为关联
   - 消息连接：对象类之间的通信联系。在面向对象术语中为依赖
2. **UML（ *Unified Modeling Language* ）**

统一建模语言是一种直观化、明确化、构建和文档化软件系统产物的通用可视化建模语言

3.

4. 事物（***things***）P37 分辨图 3.2~3.5 的模型符号

– 表示系统中的元素，事物是是实体抽象化的最终结果，是 UML 模型中的基本成员

– 包括：结构事物、行为事物、分组事物、注释事物

5. • 关系（***relationships***）

- 表示系统中的元素如何进行连接，即将事物联系在一起的方式 ；包括四种：依赖、关联、泛化、实现
- 四种关系的模型符号，图 3.9

关联中角色多重性的含义

聚合组合是特殊的关联，他们的区别？

6. UML 关系包括关联、聚合、泛化、实现、依赖等 5 种类型，请指出下面关系的类型，并采用 UML 符号表示这些关系。

a) （1） 在学校中，一个学生可以选修多门课程，一门课程可以由多个学生选修，那么学生和课程之间是什么关系？

b) （2） 类 A 的一个操作调用类 B 的一个操作，且这两个类之间不存在其他关系，那么类 A 和类 B 之间是什么关系？

c) （3） 接口及其实现类或构件之间是什么关系？

d) （4） 一个汽车有四个轮子，那么类“汽车”和“轮子”之间是什么关系？

e) （5） 学生与研究生之间是什么关系？

7. • 图（***diagrams***）

– 对整个系统而言，其功能由用例图描述，静态结构由类图和对象图描述，动态行为由状态图、时序图、协作图和活动图 描述，而物理架构则是由组件图和部署图描述。

8. 用例图：用例图定义了系统的功能需求，用例图表示了用例、参与者及其它们之间的关系。

9. 类图（***Class Diagram***）：类图描述系统的静态结构，表示系统中的类、类与类之间的关系以及类的属性和操作

10. 状态图针对单个对象建立模型。侧重于描述某个对象在其生命周期中的动态行为，包括对象在各个不同的状态间的跳转以及触发这些跳转的外部事件，即从状态到状态的控制流。必须有一个初态。

11. 顺序图（***Sequence Diagram***）

– 顺序图描述了一组交互对象间的交互方式，它表示完成某项行为的对象和这些对象之间传递消息的时间顺序。

– 一般情况下，我们使用顺序图描述一个用例的事件流，标识参与这个用例的对象，并以服务的形式将用例的行为分配到对象上。

12. 顺序图和协作图是同构的，即两者之间可以相互转换。

第四章

1. 软件需求并不单指用户需求，分为业务需求，用户需求，系统需求，功能需求和非功能需求
2. 业务需求是组织或客户对于系统的高层次目标要求， 定义了项目的远景和范围，即确定软件产品的发展方向、功能范围、目标客户和价值来源（P61）
3. 用户需求是从用户角度描述的系统功能需求和非功能需求，通常只涉及系统的外部行为，而不涉及系统的内部特性; 用户需求用自然语言表达，容易含糊和不准确
4. 系统需求是更加详细地描述系统应该做什么，通常包括许多不同的分析模型，诸如对象模型、数据模型、 状态模型等
5. 功能需求描述系统应该提供的功能或服务，通常涉及用户或外部系统 与该系统之间的交互，一般不考虑系统的实现细节
6. 非功能需求从各个角度对系统的约束和限制
7. 图 4-1 需求工程过程活动？需求获取 需求分析 需求定义 需求验证
8. 需求工程过程中需求获取需求获取主要涉及到聆听用户的需求，分析和整理所获取的信息，形成文档化的描述
9. 基于用例的需求获取（4.4.2）中四个步骤？
10. 需求定义产生的文档是需求规格说明书 SRS
11. 需求验证对 SRS 进行验证。

第五章

1. 本章讨论需求工程中的需求分析，采用了面向对象方法
2. 面向对象需求分析首先识别分析类，分析类有三种

– 实体类：表示系统存储和管理的永久信息

– 边界类：表示参与者与系统之间的交互

– 控制类：表示系统在运行过程中的业务控制逻辑

3. P79 5.2 基于 UML 的需求分析步骤，哪 3 个步骤？识别分析类 建模分析对象之间的交互 构建分析类图

4. 看一遍 P79 页识别分析类的基本原则，根据用例图和用例描述（需求工程中需求获取活动产生的模型）识别分析类。
5. 5.2.3 构建分析类图时，对每个分析类的职责，可以使用顺序图把用例行为分配到所识别的分析类中

6：书后练习 9

第六章

1. 软件设计的四个通用原则？
2. 系统模块化后复杂性和工作量比原系统小
3. 但模块的分解并非越多越好，模块之间存在着交互接口，当模块过多时，将会增加接口的代价。
4. 内聚是子系统内部的相关程度，当子系统中彼此相关的多个对象执行类似的任务时，则认为 该子系统是高内聚的；反之，当子系统内的多个对象彼此不相关时，则认为是低内聚的。
5. 内聚越高越好
6. 内聚度标志一个模块内部各成分彼此结合的紧密程度。内聚度按其高低程度可分为 7 级
7. 偶然内聚，逻辑内聚，时间内聚是低级内聚，设计时应尽量避免
8. 耦合表示两个子系统（或类）之间的关联程度。

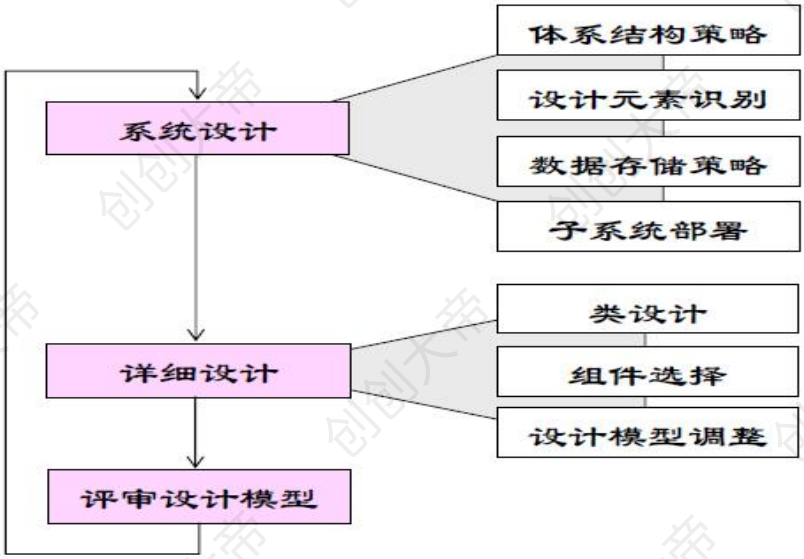
– 当一个子系统（或类）发生变化时对另一个子系统（或类） 的影响很小，则称它们是松散耦合的；反之，如果变化的影响很大时，则称它们是紧密耦合的。

9. 耦合也可分为 7 级，耦合度应越低越好
10. 一般来说，设计时应尽量使用数据耦合，绝对禁止内容耦合
11. 复用（ ***Reuse*** ）
    - 对于建立软件系统而言，所谓复用就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。
    - 复用的好处在于提高生产效率，提高软件质量，改善软件系统的可维护性
12. 常见的体系结构设计策略：

管道—过滤器结构
分层体系结构
仓库系统结构
客户/服务器模式 （ATM 是胖客户，网站是瘦客户）
MVC 模式

第七章

1. 面向对象设计过程



2. 识别设计类的基本原则

– 如果一个“分析类”比较简单，代表着单一的逻辑抽象，那么可 以将其映射为“设计类”。通常，主动参与者对应的边界类、控 制类和一般的实体类都可以直接映射成设计类。

– 如果“分析类”的职责比较复杂，很难由单个“设计类”承担，则 应该将其映射成“子系统接口”或“子系统”。通常，被动参与者对应的边界类被映射成子系统接口

3. 识别完设计类后，需要识别设计类的方法，识别类属性，识别类关系，最后建模设计类图，产生面向对象设计结果
4. 识别类方法：一般，设计类的方法首先可由分析类的职责得到（需细化），其次，遍历用例实现和画状态图，找遗漏的方法
5. 识别类属性：设计类的属性可继承自分析类，有时，分析类的属性隐含着设计类需要的一个或多个属性，一个类的属性不能被多个设计对象共享
6. 识别类关系：找设计类之间的关联（包括聚合和组合），依赖，泛化关系，可从分析模型中得到
7. 设计时会复用设计模式
8. 设计模式描述了系统设计过程中常见问题的解决方案， 它是从大量的成功实践中总结出来的
9. Composite 模式针对系统中存在单个对象和组合对象的情况，Composite 模式模糊了单个对象和组合对象的概念，客户程序可以像处理单个对象一样来处理组合对象
10. Abstract Factory 模式：适用于封装具体平台，使应用可在不同平台上运行
11. *Adaptor* 模式：该模式的目的是封装遗留系统的代码

第 11 章

1. 软件实现并非单纯地将设计模型转换为代码，包括 P176 多种基本活动
2. 编码风格是指编程时表现出来的特点、习惯、逻辑思维等。
3. 软件编码更多地是一种工程，而不是一种个人艺术。如果不统一编程规范，最终写出的程序，其可读性将较差，这不仅给代码的理解带来障碍，增加维护阶段的工作量，同时不规范的代码隐含错误的可能性也比较大。

4. 软件编码规范目的

提高编码质量，避免不必要的程序错误

增强程序代码的可读性、可重用性和可移植性

第十二章

1. 有错是软件的属性，而且是无法改变的。

关键在于如何避免错误的产生和消除已经产生的错误，使程序中的错误密度达到尽可能低的程度。

Solution: 进行验证和确认活动

2. 验证和确认（Verification & Validation，简称 V&V）工作是在整个软件生命周期中对软件的规范性评估活动，以保证软件开发各个环节的正确性。

系统开发完毕后再测试的观念是错误的。

3. 验证和确认是两个相互独立但却相辅相成的活动，二者很容易混淆
4. 验证强调对于过程的检验！
5. 确认强调对于结果的检验！

第十三章

1. Lehman 定律：著名的关于系统变更的定律
2. 对软件变更引起的各种问题，人们通常采用软件维护和软件再工程策略处理。
3. 软件维护活动可以分为三种典型的类别：改正性维护、适应性维护、完善性维护。另外不排除其他类型的一些维护，如预防性维护。
4. 在几种维护活动中，完善性维护所占的比重最大（50%），即大部分维护工作是改变和加强软件，而不是纠错

第十四章

1. 软件项目管理是为了使软件项目能够按照预定的成 本、进度、质量顺利完成，而对成本、人员、进度、 质量、风险等进行分析和管理的活动。降低复杂性和控制变化是软件项目管理的关键问题
2. 软件项目计划管理目的：

是项目管理者对资源、成本和进度作出合理的估算，制定出切实可行的软件项目计划。

3. 软件项目计划过程：

（1）确定软件范围

（2）估算项目

（3）编制项目进度表

4. 软件规模估算：采用代码行 LOC（Line of code）也可采用功能点 FP（Function points）或者对象点表示
5. 功能点技术是依据软件信息域的基本特征和对软件复杂性的估计，估算出软件规模。通常软件信息域的 **5** 个基本特征
6. 软件成本估算一般包括专家判定、类比估算和经验模型等三种技术
7. 结构性成本模型 COCOMO（COnstructive COst MOdel）是一种利用经验模型进行成本估算的方法
8. COCOMO 中两个公式，软件规模的单位？工作量估算的单位？开发时间的单位？怎么换算出成本？

9.