1.软件工程：

（1）应用系统、规范、可度量的方法来开发、运行和维护软件，即将工程应用到软件。

（2）对（1）中方法的研究。

2.团队结构有哪几种？

1）主程序员团队：一名技术能力出色的成员做为主程序员，负责领导团队完成任务。工作效率高，保证一致性。缺点是项目复杂，主程序员能力不足那么主程序员成为瓶颈。适合于把握性大，时间要求紧的情况。

2）民主团队：没有集中的瓶颈，成员发挥能动性，工作效率降低，冲突解决。敏捷+较有挑战性的项目

3）开放团队：为了创新而存在的。黑箱管理，问题在于项目进展没有可视度。

3.从1950s—2000s之间的特点（简答）

1950s：科学计算；以机器为中心进行编程；像生产硬件一样生产软件。

1960s：业务应用（批量数据处理和事物计算）；软件不同于硬件；用软件工艺的方式生产软件。

1970s：结构化方法；瀑布模型；强调规则和纪律。它们奠定了软件工程的基础，是后续年代软件工程发展的支撑。

1980s：追求生产力最大化；现代结构化方法/面向对象编程广泛应用；重视过程的作用。

1990s：企业为中心的大规模软件系统开发；追求快速开发、可变更性和用户价值；web应用出现。

2000s：大规模web应用；大量面向大众的web产品；追求快速开发、可变更性、用户价值和创新。

4.质量保障有哪些措施

1）评审：有作者之外的其他人来检查产品问题，使用评审检查表，发现问题，返工并跟踪。项目中每个阶段都进行了评审，的确可以发现问题。缺点是可能会准备不充分导致效率低下。

2）测试：项目中包括集成测试，单元测试，系统测试。

3）度量：基于只有对可以度量的东西才可以控制。项目中进行了需求度量，代码度量和测试度量。

6.名词解释：需求

1）用户为了解决问题或者达到某些目标所需要的条件或能力

2）系统或系统部件为了满足合同，标准，规范或其他正式文档所规定的要求二需要具备的条件或能力。

3）对1）或2）中的一个条件或一种能力的一种文档化表述

7.区分需求的三个层次

业务需求，用户需求，系统需求

1）业务需求解决方案和系统特性，系统开发的战略出发点，高层次的需求，描述为什么要开发系统 例：在系统使用三个月后，销售额度应该提高20%

2）用户需求问题域知识：执行具体任务的用户对系统所能完成任务的期望，描述了系统能够帮用户做什么。Note：需要补充问题域知识 例：系统要帮助收银员完成销售处理

3）系统需求需求分析模型：用户对系统行为的期望，每一个系统及需求反映了一次外界与系统间的交互行为或者一个系统的实现细节。收银员输入购买商品的标识和数量时，系统显示该商品的描述、学分、数量和总价。

8.掌握需求的类型

1）功能需求：软件系统需求中最常见主要重要的。在不考虑物理约束的情况下，用户希望系统能执行的活动。

注意UR（用户需求）和SR（系统需求）都可以是功能需求。

2）性能需求：定义了系统必须多好多块的完成专门的功能。如

速度：所有用户查询都必须在10秒内完成

容量：系统所能存储的数据量至少10万个

吞吐量：解释器每分钟至少应该解析5000条没有错误的语句

负载：系统应该允许50太营业服务器通识从集中服务器上进行数据的上传或下载。（可以限定性能目标的同时给出一定的灵活性）

PR7：（最低标准）在200用户并发时，系统不能崩溃

（一般标准）在200用户并发时，系统应该在80%内正常工作

（理想标准）在200用户并发时，系统应该能保持正常的工作状态

实时性：检测到病人的异常后，监控器必须在0.5秒内发出警告

3）质量需求（显式+隐式）

可靠性：QA1：在客户端和服务器端通信时，如果网络故障，系统不能出现故障。

可用性：QA2：系统的可用性要达到98%

安全性：QA3：收银员只能查看不能修改、删除会员信息

可维护性：QA4：如果系统要增加新的特价类型，要能够在2个人月内完成

可移植性：QA5：服务器要能够在一个人月内从windows7操作系统更换到××系统

易用性：QA6使用系统一个月的收银员进行销售处理的效率要达到10商品/min

4）对外接口：用户界面，硬件接口，软件接口，网络通信接口

5）约束：进行系统构造是需要遵守的约定，例如编程语言、硬件设备等

6）数据需求:需要在数据库、文件或者其他介质中存储的数据描述

每个功能使用的数据信息，使用频率，可访问性要求，数据实体及其关系，完整性约束，数据保持要求。

DR1：系统需要存储的数据实体及其关系为图。。所示的内容

DR2：系统需要存储1年的销售记录和退货记录

9. 用例图，类图，顺序图，需要描述建模过程

1）用例图建立过程

① 进行目标分析与确定解决方向：确定项目的目标，定义高层次解决方案的系统特性。

② 寻找参与者：连锁商店管理系统中有总经理，客户经理，收银员和管理员

③ 寻找用例：每一个参与者与系统特性相符的一个任务就是一个用例。将系统所有的用例表示在一个用例图中，该图就是系统用例图。描述为：在连锁商店管理系统中，总经理的目标有产品调整，特价策略制定，赠送策略制定，库存分析；客户经理的目标有会员管理，库存管理，库存分析；收银员的目标有销售处理和退货；管理员的目标有用户管理。

④ 细化用例：用例粒度的判断标准是：为应对一个业务事件，由一个用户发起，并在一个连续的时间段内完成，可以增加业务价值的任务。 例如：上述特价策略制定，赠送策略制定两个用例的业务目的发起源和过程基本相同，仅仅业务数据不同，可以合并为一个用例“销售策略制定”；会员管理有两个不同的业务事件，细化为发展会员和礼品赠送；库存管理细化为出库，入库和库存分析。

5 注意事项：不要将用例细化为单个操作：如增删改查应该体现一个业务价值；不要将同一个业务目标细化为不同用例；不要将没有业务价值的内容做为用例。

2）类图：概念类图又称为领域模型关注系统与外界的交互

① 识别候选类：发现软件系统与外界交互是可能涉及的类和对象（行为分析，名词分析，CRC）

② 确定概念类：根据系统的需求，该类对象的实例的状态与行为是否全部必要。

③ 识别关联:分析用例文本描述，发现协作；分析和补充问题域内的关系；去除冗余关联和导出关联。

④ 识别重要属性：协作的必要信息，通过分析用例的描述，补充问题域信息发现。

3）系统顺序图：系统顺序图是将系统整个看做一个黑箱的对象而描述的简单顺序图的形式，强调外部参与者和系统的交互行为，重点展示系统级事件。

① 确定上下文环境：
② 发现交互对象
③ 根据用例描述中的流程，逐步添加消息

4）建立状态图

① 确定上下文环境，搞清楚状态的主体（常见的状态主体有类用以多个用例和整个系统）用例UC1销售处理
② 识别状态用例UC1销售处理可能存在的状态：空闲状态，销售开始状态，会员信息显示状态，错误提示状态，列表显示状态，账单处理状态，销售结束状态。
③ 建立状态转换：

补充详细信息：在已经识别的状态和转换的基础上，添加详细的信息说明。

11. 对给定的需求示例，判断并纠正其错误

技术文档注意要点：

① 简洁：动词名词+辅助词，不要使用复杂长句，形容词和副词。
② 精确：不能产生歧义或者无法理解。

歧义词汇	改进方法
可接受的、足够的	具体定义可接受的内容，说明系统怎样判断"可接受"或"足够"
依赖	描述依赖的原因，数据依赖？服务依赖？还是资源依赖？等等
有效的	明确"有效"所意味的具体实际情况
快的、迅速的	明确指定系统在时间或速度上可接受的最小值
灵活的	描述系统为了响应条件变化或需求变化而可能发生的变更方式
改进的、更好的、优越的	定量说明在一个专门的功能领域内，充分改进的程度和效果
包括但不限于、等等、诸如	应该列举所有的可能性，否则就无法进行设计和测试
最大化、最小化、最优	说明对某些参数所能接受的最大值和最小值
一般情况下、理想情况下	需要增加描述系统在异常和非理想情况下的行为
可选择地	具体说明是系统选择、用户选择还是开发人员选择
合理的、必要的、适当	明确怎样判断合理、必要和适当
健壮的	显式定义系统如何处理异常和如何响应预料之外的操作
不应该	试着以肯定的方式陈述需求，描述系统应该做什么
最新技术水平的	定义其具体含义，即"最新技术水平"意味什么
充分的	说明"充分"具体包括哪些内容
支持、允许	精确地定义系统的功能，这些功能组合起来支持某些能力
用户友好的、简单的、容易的	描述系统特性，用这些特性说明词汇所代表的用户期望的实质

③ 易读（查询）有效使用引言目录和索引；使用系统化的方式组织内容信息，提高文档内容的可读性。
④ 易修改：使用相同的语句格式组织相关联或相似的信息；使用列表组织独立、并列的信息；使用编号表达繁杂信息之间的关系。

需求书写要点

① 使用用户术语：不要使用“计算机术语”

② 可验证：不可验证的需求往往是过于抽象或者描述模糊。具体化，避免程度词的使用。

R1：用户界面的查询应该友好 bad

R2：用户完成任何一个查询任务时的鼠标点击数都不能超过5次。Good

③ 可行性：在运行环境的已知条件和约束下实现。

R3：系统必须持续可用，即每周7天，每天24小时都是可用的。 Bad

Note:PPT上有例题（第七章P18-…）

12. 对给定的需求示例，设计功能测试用例

1）以需求为线索，开发测试用例套件

2）使用测试技术确定输入输出

13. 名词解释：软件设计

软件设计是指关于软件对象的设计，是一种设计活动。软件设计既指软件对象实现的规格说明，又指这个规格说明产生的过程。

软件设计活动以需求开发的制品（需求规格说明和分析模型）为基础，构建软件设计方案描述和原型，为后期的构造活动提供规划或蓝图。

软件设计兼具工程性和艺术性，由于软件系统的可变性，软件设计具有演化性，也因为软件设计的过程实际上就是一系列决策发生的过程，软件设计具有决策性。

17. 体系结构风格的优缺点（*）（要会画图）

① 主程序/子程序风格：

将系统组织成层次结构，包括一个主程序和一系列子程序。主程序是系统的控制器，负责调度各个子程序的执行。各子程序又是一个局部的控制器，负责调度其子程序的执行。

A) 优点：流程清晰，易于理解（符合分解和分治的思想）；强控制性（很容易保证正确性）；

B) 缺点：程序调用是一种强耦合的连接方式，难以修改和复用；可能会使得不同部件之间使用隐含的共享数据，出现不必要的公共耦合。

② 面向对象式：

借鉴面向对象思想组织整个系统的高层结构。基于封装，对象之间通过协作完成系统任务。

A) 优点：内部实现的可修改性（隐藏内部实现）；易开发、易理解、易复用的结构组织（契合模块化思想）

B) 缺点：接口的耦合性（由于方法调用机制，接口的耦合性无法消除）；标识的耦合性（一个对象要和其他对象交互，必须知道标识）；副作用（难以实现程序的“正确性”）

③ 分层：根据不同的抽象层次，将系统组织为层次式结构。每个层次被建立为一个部件，不同部件之间通常用程序调用方式连接。连接件被建立为程序调用机制。

A).设计机制清晰，易于理解（抽象层次分离，隔离复杂度）；支持并行开发（层次之间遵守成熟稳定的接口）；更好的可复用性和内部可修改性（接口的稳定性，不同层次的部件能够互相替换）

B).交互协议难以修改（可能需要改变所有的层次，接口具有强耦合性）；性能损失（禁止跨层调用）；难以确定层次数量和粒度。

④ MVC：模型-视图-控制

以程序调用为连接件。将系统功能组织为模型、视图和控制三个部件。模型封装了系统的数据和状态信息，实现业务逻辑，对外提供数据服务和执行业务逻辑。视图封装了用户交互，提供业务展现，接受用户行为。控制封装了系统的控制逻辑，根据用户行为调用需要执行的业务逻辑和数据更新，并且根据执行后的系统状态决定后续的业务展现。

A).易开发性（分别抽象了业务逻辑，表现和控制机制清晰，易于开发）；视图和控制的可修改性；适宜于网络系统的开发特征（一个模型可以同时建立并支持多个视图）

B).复杂性；模型修改困难（视图和控制均依赖于模型）；

20.体系结构构件之间的接口（能够根据用例写出某层的接口，需求分配的过程，重点）

22. 名词解释：可用性

人机交互中一个重要概念，多维度的定义

1）易学性：新手用户容易学习，能够很快使用系统。

2）易记性：使用过软件系统的用户，能够有效记忆或快速重新学会使用该系统。

3）效率：数量用户使用系统完成任务的速度

4）主观满意度：让用户有良好的体验。

23. 能够列出至少5个人机交互原则进行解释（例子违反了哪些界面设计原则）

1）简洁设计：（7+-2原则）不要使用太大的菜单，不要再一个窗口中表现过多的信息类别，不要再一个表单中使用太多的颜色和字体做为线索。

2）一致性设计：①一个系统中相似的任务具有一致的交互机制，用户一致的精神模型②还有用户使用类似系统的习惯等。

3）低出错率设计：帮助人们避免犯错，尽可能设计不让用户犯严重错误的系统，具体可以是将不适当的菜单选项功能以灰色的显示屏蔽，禁止数值输入域内出现字母等。

4）易记性设计：减少短期记忆负担，使用逐层方式显现信息，使用直观的快捷方式，设置有意义的默认值。

5）可视化设计要点

25. 导航反馈协作式设计

1）导航：

导航的目的就是为用户提供一个很好的完成任务的入口，好的导航会让这个入口符合人的精神模型。

a) 全局导航：按照任务模型将软件产品的功能组织起来，并区分不同的重要性和主题提供给不同的用户。常常包括窗口，菜单，列表，快捷方式，热键等。全局结构的设计主要以功能分层和任务交互为依据。

b) 局部结构通过安排界面布局细节，制造视觉上的线索来给用户提供导航。常用的控件包括可视化控件布局和组合、按钮设计、文本颜色、字体大小。主要以用户关注的任务细节为依据。

2）反馈：好的人机交互设计需要对用户行为进行反馈，让用户能够意识到行为的结果。
目的是提示用户交互的结果，但不能打断用户工作的意识流。
对用户思考和反应时间的把握。

3）协作式设计：

人和计算机是人机交互的两方，其中人的因素是比较固定的，一定时期内不会发生大的变化，所以要让二者交互顺畅，就需要让计算机更多地适应人的因素，这也是人机交互设计以用户为中心的根本原因。

这种调整计算机因素以更好地适应并帮助用户的设计方式被称为协作式设计

27. 职责分配：

1）职责是持有某项数据或者表现某种职责的义务；数据职责主要由属性满足，行为职责主要由方法满足；可能会包含类间协作
2）职责可以从不同的抽象层次开始；职责是可以被分解的；职责分解可以是分解组件的基础
3）职责分配可以帮助实现高内聚低耦合，确保不同模块职责没有重合，当且仅当数据和方法有助于实现模块职责时将其添加给模块。
4）GRASP原则（通用职责分配软件模式—将职责分配原则总结为模式）：
① Expert专家：将一个职责分配给专家—掌握了履行职责所必需的信息的类。
② Creator创建者：

③ 高内聚：分配一个职责的时候要保持类的高聚合度
④ 低耦合：分配一个职责是要保持低耦合度
⑤ Controller：

控制要点：
A) 避免大多数消息由一个类发出
B) 组件相对较小
C) 行为、职责和数据绑定
D) 职责单一

28. 协作：（产生顺序图、状态图）

29. 控制风格：
1）集中式：做决策的只有一个对象，

2）委托式：作出决策的对象不只有一个，职责的分解决定了控制对象的层次。

抽象对象之间的协作

- 1.从小到大,将对象的小职责聚合形成大职责;
- 2.从大到小,将大职责分配给各个小对象。
- 这两种方法,一般是同时运用的,共同来完成对协作的抽象。

3）分散式：无法找到明确的控制对象，每个对象都只承担一个相对较小的职责，完全靠对象自治的方式实现各自的职责。

30. 给定分析类图、系统顺序图和设计因素描述，建立设计类图或者详细顺序图（注意流程）

面向对象设计的过程：能够画出设计类图和顺序图

1） 设计模型的建立：（静态设计模型）①抽象类的职责②抽象类的关系③添加辅助类（动态设计模型）①抽象对象之间的协作②明确对象的创建③选择合适的控制风格。

需要添加的辅助类有：接口类，容器类，实现数据类型的类，控制器类，启动类，记录类。例如：SalesBLService,SalesList&SalesLineItem,?,controller,mainframe,PO

2） 根据设计模式重构：模块化思想高内聚低耦合，信息隐藏隐藏职责与变更，利用设计模式重构。

31. 协作的测试MockObject的书写

32. 名词解释：耦合和内聚

① 耦合描述的是两个模块之间关系的复杂程度：包括内容耦合，公共耦合，重复耦合，控制耦合，印记耦合，数据耦合

② 内聚表达的是一个模块内部的联系的紧密性：包括偶然内聚，逻辑内聚，过程内聚，通信内聚，功能内聚，信息内聚。

33. 对给出例子说明耦合和内聚的情况，并说明理由

34.信息隐藏：

① 基本思想：每个模块都隐藏一个重要的设计决策。每个模块都承担一定的职责，对外表现一定的契约，并且在这份契约下隐藏着只有这个模块知道的设计决策或者秘密，决策实现的细节（特别是容易改变的细节）只有该模块自己知道。

② 两种常见的信息隐藏决策：一是根据需求分配的职责，因为实践表明，需求是经常变化的，频率和幅度都很大；二是内部实现机制，常见的变化主题包括硬件依赖，输入输出形式，非标准语言特征和库，负责的设计和实现，复杂的数据结构，复杂的逻辑，全局变量。数据大小限制等。

③ 对给出的例子，说明信息隐藏的优劣

35. 模块化的原则：

① 全局变量是有害的
② 简洁
③ 不要重复
④ 针对接口编程
⑤ 迪米特法则
⑥ 接口最小化原则
⑦ Liskov替换原则
⑧ 多用组合少用继承
⑨ 单一职责原则

给定实例，发现违反的原则，并纠正

36. 封装

① 将数据和行为同时包含在类中。

② 分离对外接口和内部实现:

A) 封装数据和行为一般情况下，所有数据应该是private的，不要为所有变量提供getter和setter，getter和setter中可以加入约束检查和数据装换。

B) 封装内部结构：实现中的复杂数据结构重点封装。信息隐藏要求我们不能暴露数据结构的实现决策。Public Position[] getPosition(){}暴露了存储决策，更坏的是getPositionArray()public Position getPosition(int index)这样较好

C) 封装其他对象的引用：上例中实现不应该是 return this.position[index]而是

```
Public Position getPosition(int index){
    Return new Position(positions.get(index));//避免外部对内部数据的修改
}
```

D) 封装类型信息：
在多种子类型对象因为具备一些共性而被视作一种类型加以使用，应该隐藏具体子类型的类别，只知道其共性类别。 LSP,&&some design pattern

E) 封装潜在的变更：
如果预计类的实现中有特定
地方会发生变更，就应该将其作为单独的类或方法，为单独的类或方法建立稳定的接口，并在原类中使用稳定的接口以屏蔽潜在变更的影响。

37. OCP：开闭原则：好的设计应该对扩展开放，对修改关闭

在发生变更时，好的设计只需要添加新的代码而不需要修改原有的代码，就能够实现变更。

使用多态实现OCP：

• 对于新增加的需求，可以将其实现代码组织为一个新类型，并将新类型与程序中某个原有类型联合起来建立多态机制。（Strategy可以是一种实现方式，command也可）

• 对于已有需求的变更，可以将变更后需求的实现代码组织为一个新类型，并将类型与其元类型联合起来建立多态机制。

38. DIP：依赖倒置原则：

① 抽象不应该依赖于细节，细节应该依赖于抽象。因为抽象是稳定的，细节是不稳定的。

② 高层模块不应该依赖于低层模块，而是双方都依赖于抽象，因为抽象是稳定的，高层低层模块都可能是不稳定的。

39. 策略模式：

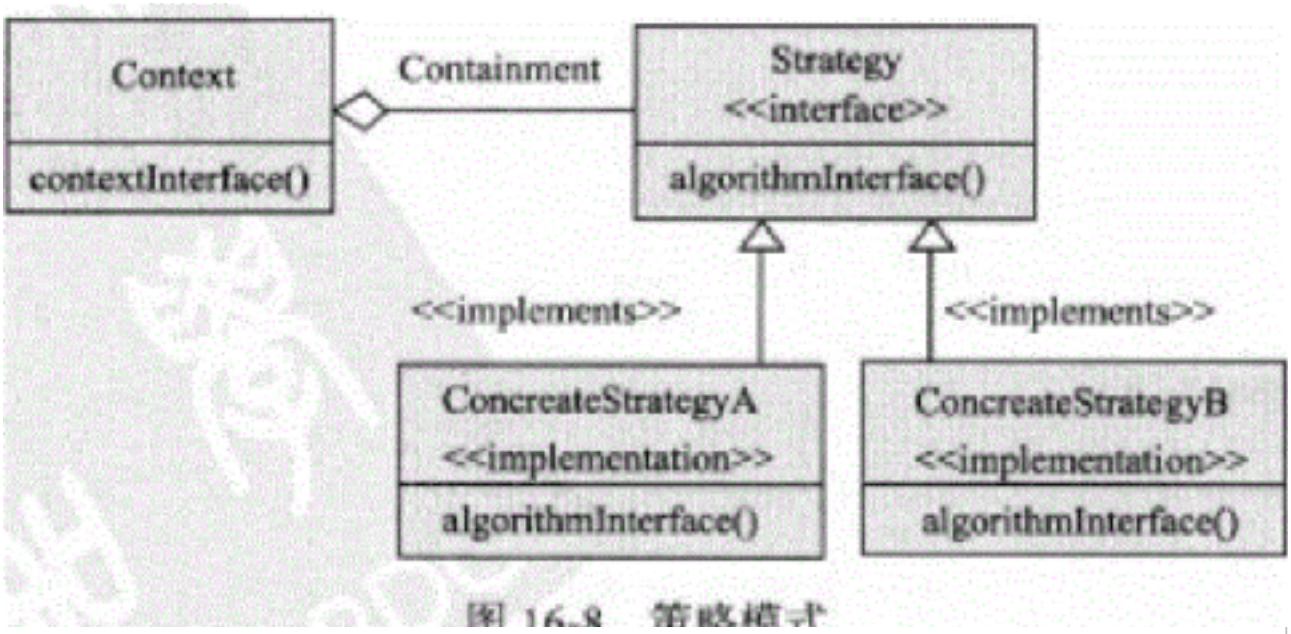


图 16-8 策略模式

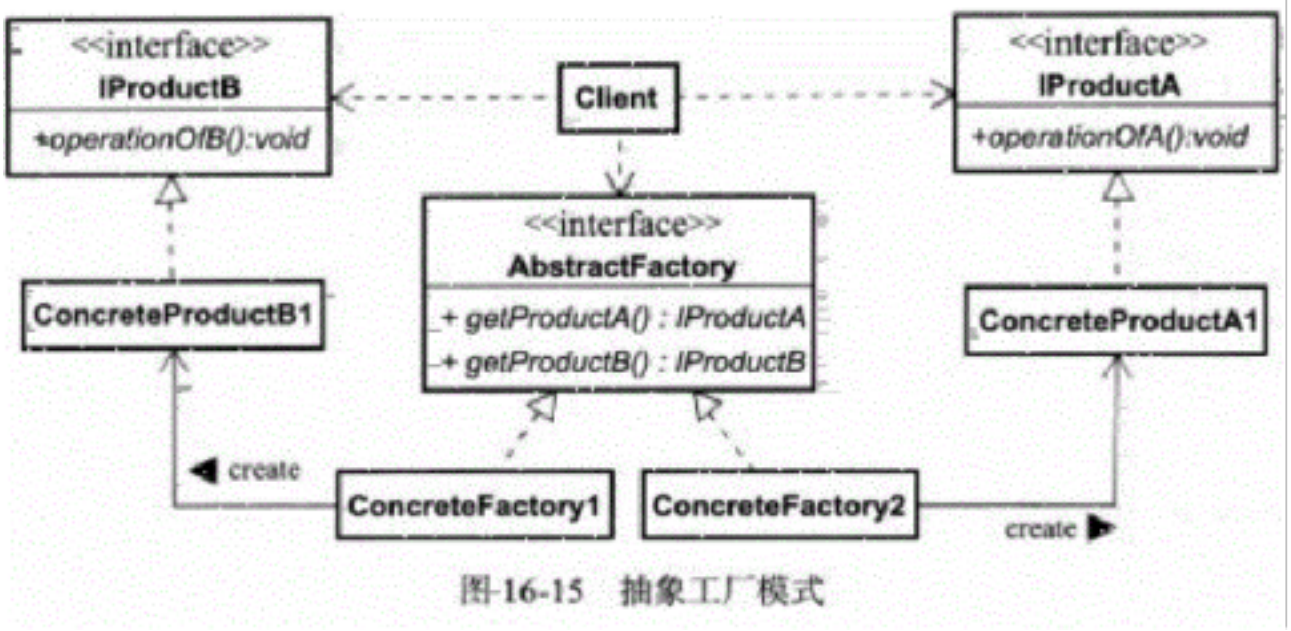
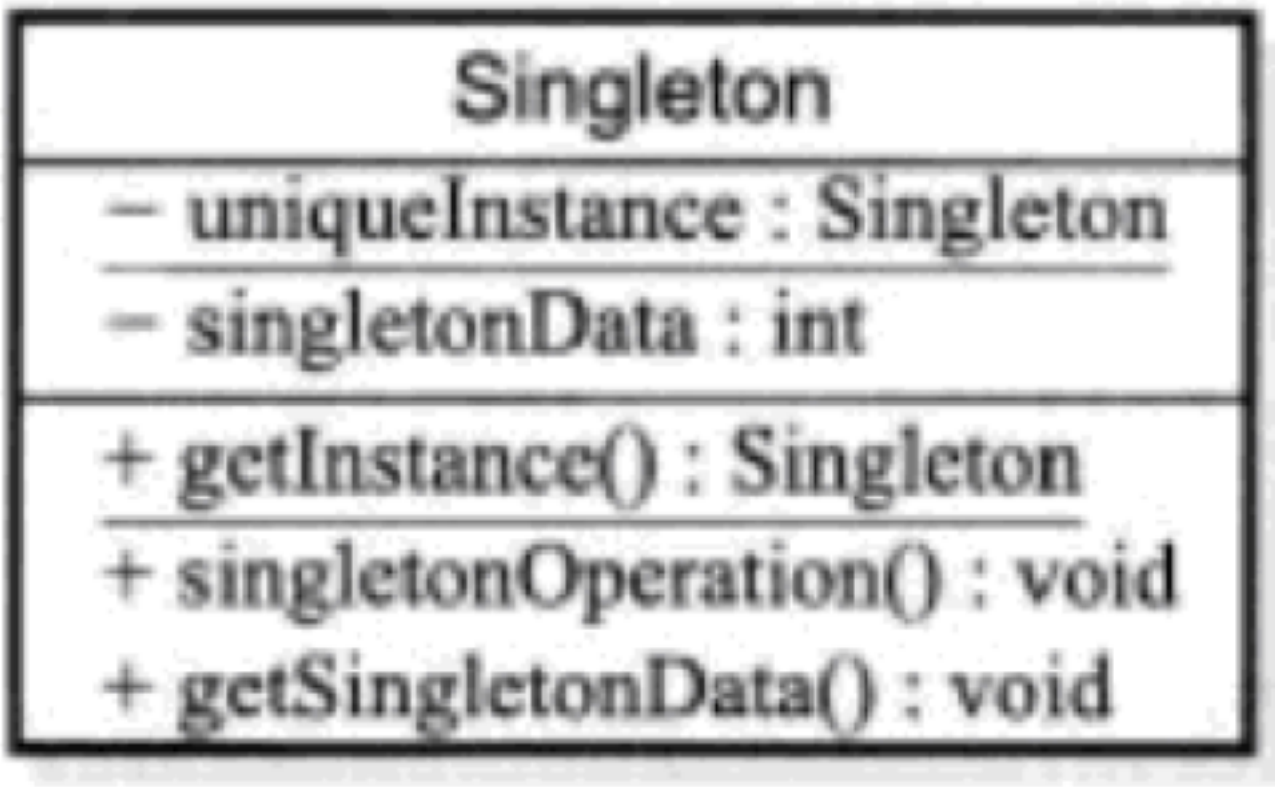
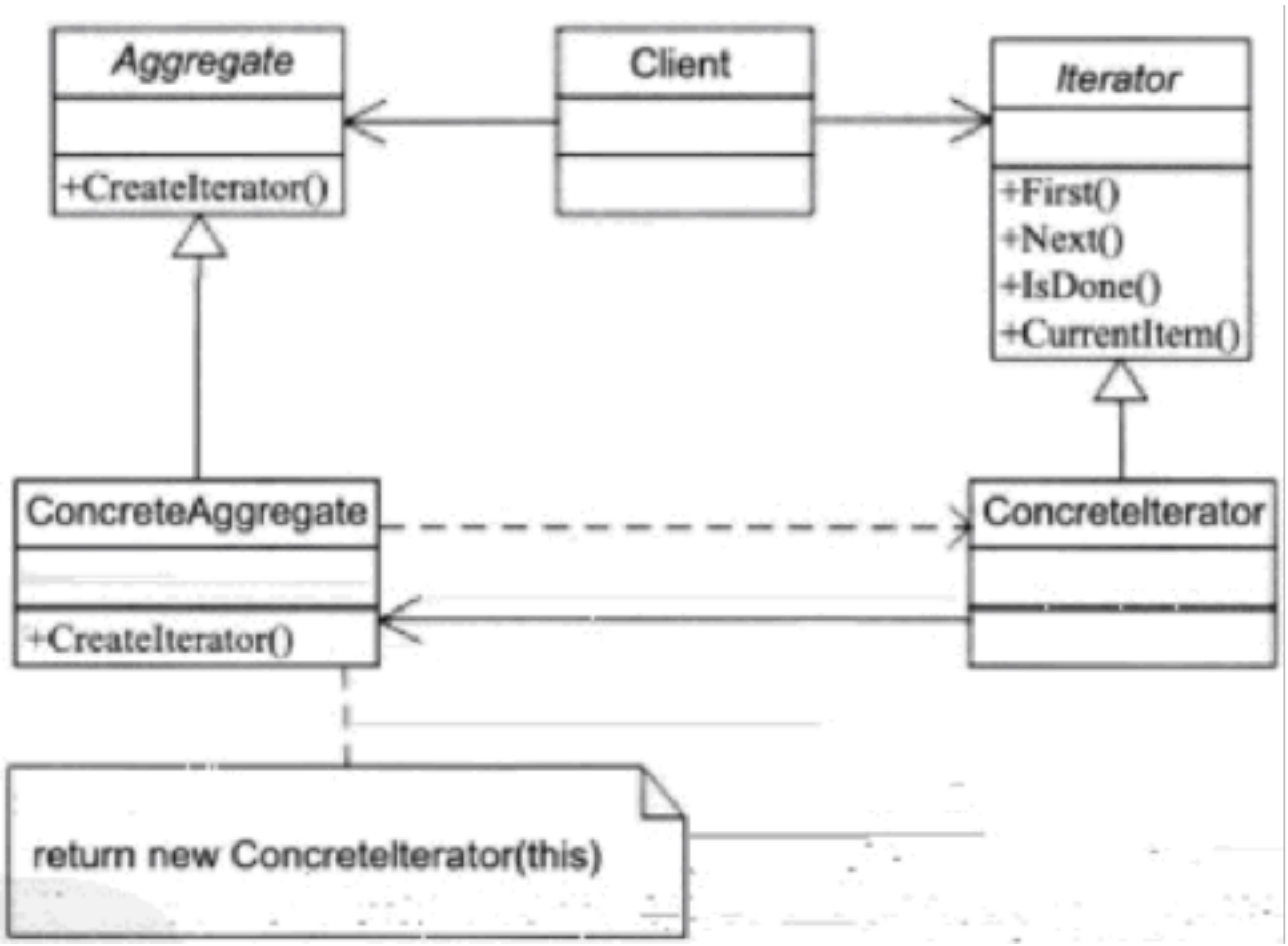


图-16-15 抽象工厂模式

41. 单件模式



42. 迭代器模式



43. 给定代码示例，发现低层设计级别的问题
    - 简洁性/可维护性
    - 使用数据结构消减复杂判断
    - 控制结构
    - 变量使用
    - 语句处理
    - 防御与错误处理
    - 不可维护的代码
44. 单元测试用例的设计：见上
45. 契约式设计：又称断言式设计，如果一个函数或方法，在前置条件满足的情况下开始执行，完成后能够满足后置条件，那么这个函数就是正确的可靠的
    a) 异常方式：
        i. 代码开始执行的时候，检查前置条件是否满足，如果不满足抛出异常，代码执行完之后，检查后置条件是否满足，不满足也抛出异常。
        ii. Public class Sales extends DominObject{
            Public double getChange(double payment) throws PreException{
            If(payment<=0||payment<total){
            Throw new PreException("Sales.getChange:Payment)
            }
            }
            }
    b) 断言方式：

46. 防御式编程：在一个方法与其他方法，操作系统、硬件等外界环境进行交互时，不能确保外界都是正确的，所以要在外界发生错误时，保护方法内部不受损害。

47. 表驱动编程：复杂逻辑判断决策表表驱动

48. 黑盒白盒测试的主要方法优缺点
    a) 黑盒
        i. 等价类划分：把所有可能的输入数据，即程序的输入域划分为若干部分（子集），从每个子集中选取少量具有代表性的数据作为测试用例。
        ii. 边界值分析：对定价类划分的补充，错误容易发生在各个等价类的边界上，而不是等价类的内部，因此针对边界情况设计测试用例，可以发现更多的缺陷。
        iii. 决策表：为复杂逻辑判断设计测试用例的技术
        iv. 状态转换：针对复杂测试对象的测试技术。该类复杂测试对象对输入数据的反应是多样的，还需要依赖自身的状态才能决定。需要建立状态图，状态装换表。
    b) 白盒
        i. 语句覆盖：确保被测试对象的每一行代码都至少执行一遍。（相对而言，比后两种覆盖率弱）

ii. 条件覆盖：确保程序中没个判断的结果都至少满足一次（保证每个条件都被覆盖了，比语句覆盖覆盖程度强，但不能保证所有执行路径）
iii. 路径覆盖：确保程序中每条独立的执行路径都至少执行一次（覆盖了强，代价大）。

49. 解释并区别白盒测试的主要方法：语句覆盖，分支覆盖，路径覆盖
50. 给出场景写对应：

给出功能需求，则要求写功能测试用例
给出设计图，则要求写集成测试用例，Stub and Driver
给出方法的描述，则要求写单元测试用例，Mock Object
JUnit基本使用方法

51. 软件生命周期模型

a) 构建修复模型-是最早也是最自然产生的软件开发模型。没有规划组织，完全依靠开发人员的个人能力。
   i. 缺点：
      1. 没有分工，随着软件系统复杂度上升，开发活动超出个人控制能力。
      2. 没有分析需求的真实性，给软件开发带来很大的风险
      3. 软件结构的质量随着修改越来越差
      4. 没有考虑可维护性
   ii. 使用范围：
      1. 软件规模很小，只有几百行程序。
      2. 对软件质量要求不高，即使出错也无所谓
      3. 只关注开发，对后期委会要求不高甚至不需维护

b) 瀑布模型：按照软件生命周期模型将软件开发活动组织为需求开发、软件设计、软件实现、软件测试、软件交付和软件维护等基本活动，并且规定了它们自上而下、相互邻接的次序。允许反复和迭代，重点在于每个活动必须进行验证，文档驱动的。
   i. 缺点：
      1. 对文档期待过高
      2. 对开发活动的线性顺序假设是不符合实际情况的
      3. 客户用户参与不够，只有在需求一个阶段用户参与
      4. 里程碑粒度过粗，现在软件系统复杂，每个阶段持续周期较长。
   ii. 优点：
      1. 为软件开发活动定义了清晰的阶段划分（包括了输入/输出、主要工作及其关注点），降低了复杂度。

iii. 使用范围

1. 需求非常成熟稳定。例：开发一个电梯调度系统，需求是明确的。
2. 所需的技术成熟、可靠。例：开发一套自动化办公系统。
3. 复杂度适中，不至于产生太大的文档负担和过粗的里程碑。

Note：周期过长和渐进演化。时间压力和并行开发

c) 增量迭代模型：项目开始时，通过系统需求的开发和核心体系结构设计活动完成项目前景和范围的商定，然后将后续开发活动组织为多个迭代、并行的瀑布式开发活动。需求驱动的。

i. 优点：

1. 迭代式开发符合软件开发的实际情况，更好的适用性。
2. 并行开发可以缩短产品的开发时间
3. 渐进交付可以加强用户反馈，降低开发风险。

ii. 缺点：

1. 由于各个构件是逐渐并入已有的软件体系结构中的，所以加入构件必须不破坏已构造好的系统部分，要求具有开放式的体系结构。
2. 增量交付模型需要一个完备、清晰的项目前景和范围以进行并发开发规划，但是在一些不稳定的领域，不确定性太多或者需求变化非常频繁，很难在项目开始就确定前景和范围

iii. 适用范围

1. 大规模软件开发，比较成熟和稳定的领域。如：电子商务。CSS选课系统可以考虑。

d) 演化模型：演化模型将软件开发活动组织为多个迭代、并行的瀑布式开发活动。

i. 优点：

1. 迭代开发，尤其适用于需求变更频繁或不确定性较高的系统。
2. 并行开发
3. 渐进交付

ii. 缺点：

1. 无法在项目早期阶段确定项目范围，项目的整体计划，商务协商难以把握。Manager难度
2. 体系结构难做，体系结构几乎确定会发生变更。
3. 容易退化为构建修复方式。

iii. 使用范围：

1. 不稳定领域的大规模软件系统。如：互联网金融，大数据。CSS选课系统不适合，需求不易变更，体系结构难做。

e) 原型模型：在整体安排迭代的情况下，强调“抛弃式原型”的演化模型。抛弃式原型解决对未来知识的局限性产生的不确定性，将未来置于现在进行推敲。

i. 优点：

1. 加强了和用户、客户的交流，提高满意度

2. 适用于新颖的领域。

ii. 缺点：

1. 原型开发带来风险：如成本，耗尽项目时间资金
2. 不舍得抛弃原型，使得低质量的代码进入最终产品导致产品低质量。

iii. 使用范围：

大量不确定的新颖领域。如第一个智能手机。。。

f) 螺旋模型：螺旋模型将软件开发活动组织为风险解决的迭代

i. 优点：降低风险

ii. 缺点：

1. 风险解决使用原型，存在原型的风险
2. 模型过于复杂，不利于管理。

iii. 使用范围：

高风险大大规模软件系统开发。

g) Rational统一过程：RUP总结和借鉴传统上的各种有效经验，建立最佳实践方法的集合，并提供有效的过程定制手段，允许开发者根据特定的需要定制一个有效的过程模型

i. 核心思想：

1. 迭代式开发，这是过去被反复证明的最佳实践方法；
2. 管理需求，重视需求工程中除了需求开发之外的需求管理活动；
3. 使用基于组件的体系结构，它帮助建立一个可维护、易开发、易复用的软件体系结构；
4. 可视化建模，利用UML进行建模；
5. 验证软件质量，尽早和持续地开展验证，以尽早发现缺陷，降低风险和成本；
6. 控制软件变更，适应1990s以后需求变更越来越重要的事实。

ii. 优点：

1. 吸收和借鉴了传统上的最佳实践方法，尤其是其核心的6个实践方法，能够保证软件开发过程的组织是基本有效和合理的。
2. RUP依据其定制机制的不同，可以适用于小型项目，也可以适用于大型项目的开发，适用面广泛。
3. RUP有一套软件工程工具的支持，这可以帮助RUP的有效实施。

iii. 缺点

1. 没有考虑交付之后的软件维护问题；
2. 裁剪和配置工作不是一个简单的任务，无法保证每个项目都能定制一个有效的RUP过程

iv. 适用性：RUP是重量级过程，能够胜任大型软件团队开发大型项目时的活动组织。但RUP经过裁剪和定制，也可以变为轻量级过程，也能够胜任小团队的开发活动组织。

h) 敏捷过程：针对传统过程模型的缺陷和新的形势，人们总结实践中的经验和最佳实践方法，尝试建立轻量级过程方法。

i. 优点：
ii. 缺点：
iii. 适用性：从敏捷联盟声明的思想和原则来看，它们反映了1990s之后软件工程的发展趋势，所以得到了广泛的应用，尤其是能够适应于快速变化或者时间压力较大的项目。

解释比较不同的过程模型（特征，优缺点，场景）