目录

第 1 章 系统开发基础

一、软件工程概述（2 星）

1、CMM 模型

初始级：杂乱无章，甚至混乱，几乎没有明确定义的步骤，项目的成功完全依赖个人的努力和英雄式核心人物的作用。

可重复级：建立了基本的项目管理过程和实践来跟踪项目费用、进度和功能特性，有必要的过程准则来重复以前在同类项目中的成功。

已定义级：管理和工程两方面的软件过程已经文档化、标准化，并综合成整个软件开发组织的标准过程。

已管理级：制定了软件过程和产品质量的详细度量标准。

优化级：加强了定量分析，通过来自过程质量反馈和来自新观念、新技术的反馈使过程能不断持续地改进。

2、CMMI 阶段式模型

初始的：过程不可预测且缺乏控制。

已管理的：过程为项目服务。

已定义的：过程为组织服务。

定量管理的：过程已度量和控制。

优化的：集中于过程改进。

3、CMMI 连续式模型

CL0（未完成的）：过程域未执行或未得到 CL1 中定义的所有目标。

CL1（已执行的）：其共性目标是过程将可标识的输入工作产品转换成可标识的输出工作产品，以实现支持过程域的特定目标。

CL2（已管理的）：其共性目标是集中于已管理的过程的制度化。

CL3（已定义级的）：其共性目标集中于已定义的过程的制度化。

CL4（定量管理的）：其共性目标集中于可定量管理的过程的制度化。

CL5（优化的）：使用量化（统计学）手段改变和优化过程域，以满足客户的改变和持续改进计划中的过程域的功效。

二、软件开发方法（3 星）

1、结构化开发方法：用户至上，严格区分工作阶段，每阶段有任务和结果，强调系统开发过程

的整体性和全局性，系统开发过程工程化，文档资料标准化，自顶向下，逐步分解（求精）。

2、原型开发方法：适用于需求不明确的情况。

3、面向对象开发方法：更好的复用性，关键在于建立一个全面、合理、统一的模型，分析、设计、实现三个阶段，界限不明确。

4、面向服务开发方法：面向对象更高标准的抽象。

三、软件开发模型（5 星）

1、瀑布模型：瀑布模型是将软件生存周期中的各个活动规定为依线性顺序连接的若干阶段的模型，包括需求分析、设计、编码、运行与维护。

瀑布模型的特点是容易理解，管理成本低，每个阶段都有对应的成果产物，各个阶段有明显的界限划分和顺序要求，一旦发生错误，整个项目推倒重新开始。

适用于需求明确的项目，一般表述为需求明确或二次开发，或者对于数据处理类型的项目

2、V 模型：强调测试贯穿项目始终，而不是集中在测试阶段。是一种测试的开发模型。

3、喷泉模型：典型的面向对象的模型。特点是迭代、无间隙。会将软件开发划分为多个阶段，但各个阶段无明显界限，并且可以迭代交叉。

4、原型模型：典型的原型开发方法模型。适用于需求不明确的场景，可以帮助用户明确需求。

5、增量模型：融合了瀑布模型的基本成分和原型实现的迭代特征，可以有多个可用版本的发布，核心功能往往最先完成，在此基础上，每轮迭代会有新的增量发布，核心功能可以得到充分测试。强调每一个增量均发布一个可操作的产品。

6、螺旋模型：典型特点是引入了风险分析。结合了瀑布模型和演化模型的优点，最主要的特点在于加入了风险分析。它是由制定计划、风险分析、实施工程、客户评估这一循环组成的，它最初从概念项目开始第一个螺旋。属于面向对象开发模型，强调风险引入。

7、统一过程（在软件设计师考试中 UP、RUP 都指统一过程）：典型特点是用例驱动、以架构为中心、迭代和增量。统一过程把一个项目分为四个不同的阶段：

构思阶段：包括用户沟通和计划活动两个方面，强调定义和细化用例，并将其作为主要模型。

细化阶段：包括用户沟通和建模活动，重点是创建分析和设计模型，强调类的定义和体系结构的表示。

构建阶段：将设计转化为实现，并进行集成和测试。

移交阶段：将产品发布给用户进行测试评价，并收集用户的意见，之后再次进行迭代修改产品使之完善

8、敏捷开发：是一种以人为核心、迭代、循序渐进的开发方法，适用于小团队和小项目，具有小步快跑的思想。常见的敏捷开发方法有极限编程法、水晶法、并列争球法和自适应软件开发方法。

敏捷方法	特点
极限编程 XP	4 大价值观、5 个原则、12 个最佳实践。
水晶法（Crystal）	认为每一个不同的项目都需要一套不同的策略、约定和方法论（以项目为本），认为人对软件质量有重要的影响（以人为本），因此随着项目质量和开发人员素质的提高，项目和过程的质量也随之提高。通过更好地交流和经常性交付，软件生产力得到提高。
开放式源码	程序开发人员在地域上分布广。
并列争球法（SCRUM）	橄榄球中的“并列争球”，多个成员站在一排，都同时冲刺着去争抢球权。软件开发中，把每 30 天一次的迭代称为一个“冲刺周期”，并按需求的优先级来实现产品。多个自组织和自治的小组并行地递增实现产品。
功用驱动开发方法 FDD	开发成员人尽其用，把功能快速开发好。首席程序员和“类”程序员。
自适应软件开发 ASD	核心是三个非线性的、重叠的开发阶段：猜测、合作与学习。ASD 有 6 个基本的原则：有一个使命作为指导；特征被视为客户价值的关键点；过程中等待是很重要的，因此“重做”与“做”同样关键；变化不被视为改正，而是被视为对软件开发实际情况的调整；确定的交付时间迫使开发人员认真考虑每一个生产的版本的关键需求；风险也包含其中。
敏捷统一过程 AUP	采用经典的 UP 阶段性（初始、精化、构建、转换），采用 “小型迭代”和 “大型连续”原理来构建软件系统。

极限编程 XP		
4 大价值观	5 大原则	12 大最佳实践
沟通 简单 反馈 勇气	快速反馈 简单性假设 逐步修改 提倡更改 优质工作	计划游戏：快速制定计划、随着细节的不断变化而完善 小型发布：系统的设计要能够尽可能早地交付 隐喻：找到合适的比喻传达信息 简单设计：只处理当前的需求，使设计保持简单 测试先行：先写测试代码，然后再编写程序 重构：重新审视需求和设计，重新明确地描述它们以符合新的和现有的需求 结对编程 集体代码所有制 持续集成：可以按日甚至按小时为客户提供可运行的版本 每周工作 40 小时 现场顾客：系统最终用户代表应该全程配合 XP 团队 编码标准

四、系统分析（5 星）

1、需求分析的任务是解决做什么的问题。

2、需求的分类：

（1）功能需求：考虑系统要做什么，在何时做，在何时以及如何修改或升级。

（2）非功能需求：考虑软件开发的技术性指标，例如存储容量限制、执行速度、响应时间及吞吐率等。

（3）设计约束：除了功能需求和非功能需求以外的需求，例如操作系统限制、开发语言限制等。

3、需求分析的工具有：判定表、判定树、数据流图和数据字典。

4、需求分析的产物有：需求规格说明书（SRS）。

五、系统设计（5 星）

1、软件设计的任务是解决怎么做的问题。

软件设计包括体系结构设计、接口设计、数据设计和过程设计。

过程设计：系统结构部件转换成软件的过程描述。

结构设计：定义软件系统各主要部件之间的关系。

接口设计（人机界面设计）：软件内部、软件和操作系统间以及软件和人之间如何通信。

数据设计：将模型转换成数据结构的定义。好的数据设计将改善程序结构和模块划分，降低过程复杂性。

2、系统设计的基本任务大体上可以分为概要设计和详细设计两个步骤。

（1）概要设计

设计软件系统总体结构：基本任务还是采用某种设计方法，将一个复杂的系统按功能划分成模块；确定每个模块的功能；确定模块之间的调用关系；确定模块之间的接口，即模块之间传递的信息；评价模块结构的质量。

数据结构及数据库设计：在需求分析阶段对数据的组成、操作约束和数据之间的关系进行了描述，概要设计阶段要加以细化，详细设计阶段则规定具体的实现细节。

编写概要设计文档：概要设计说明书、数据库设计说明书、用户手册以及修订测试计划。

评审：对设计部分是否完整地实现了需求中规定的功能、性能等要求，设计的可行性，关键的处理以及外部接口定义的正确性、有效性、各部分之间的一致性等都一一进行评审。

（2）详细设计

对每个模块进行详细的算法设计，用某种图形、表格和语言等工具将每个模块处理过程的详细算法描述出来。

对模块内的数据结构进行设计。

对数据库进行物理设计，即确定数据库的物理结构。

其他设计：根据软件系统的类型，还可能需要进行代码设计、输入/输出格式设计，用户界面设计等。

编写详细设计说明书。

评审：对处理过程的算法和数据库的物理结构都要评审。

3、软件设计的原则：**高内聚、低耦合**

（内聚性）

偶然聚合：模块完成的动作之间没有任何关系，或者仅仅是一种非常松散的关系。

逻辑聚合：模块内部的各个组成在逻辑上具有相似的处理动作，但功能用途上彼此无关。

时间聚合：模块内部的各个组成部分所包含的处理动作必须在同一时间内执行。

过程聚合：模块内部各个组成部分所要完成的动作虽然没有关系，但必须按特定的次序执行。

通信聚合：模块的各个组成部分所完成的动作都使用了同一个数据或产生同一输出数据。

顺序聚合：模块内部的各个部分，前一部分处理动作的最后输出是后一部分处理动作的输入。

功能聚合：模块内部各个部分全部属于一个整体，并执行同一功能，且各部分对实现该功能都必不可少。

（耦合性）

非直接耦合：两个模块之间没有直接关系，它们的联系完全是通过主模块的控制和调用来实现的。

数据耦合：两个模块彼此间通过数据参数交换信息。

标记耦合：一组模块通过参数表传递记录信息，这个记录是某一个数据结构的子结构，而不是简单变量。

控制耦合：两个模块彼此间传递的信息中有控制信息。

外部耦合：一组模块都访问同一全局简单变量而不是同一全局数据结构，而且不是通过参数表传递该全局变量的信息。

公共耦合：两个模块之间通过一个公共的数据区域传递信息。

内容耦合：一个模块需要涉及到另一个模块的内部信息。

六、系统测试（5 星）

1、常见的软件测试方法分类：

（1）静态测试：桌前检查、代码走查、代码审查。

（2）动态测试

黑盒测试：等价类划分、边界值分析、错误推测、因果图。

白盒测试：语句覆盖、判定覆盖、条件覆盖、条件/判定覆盖、路径覆盖。

2、常见的黑盒测试方法：

等价类划分：确定无效与有效等价类，设计用例尽可能多的覆盖有效类，设计用例只覆盖一个无效类。

边界值分析：处理边界情况时最容易出错，选取的测试数据应该恰好等于、稍小于或稍大于边界值。

3、常见的白盒测试方法：

	定义	特点
语句覆盖	被测试程序中的每条语句至少执行一次。	对执行逻辑覆盖很低，一般认为是很弱的逻辑覆盖。
判定覆盖（分支覆盖）	被测程序每个判定表达式至少获得一次“真”值和“假”值（或者程序中每一个判定取“真”分支和取“假”分支至少通过一次。）	判定覆盖比语句覆盖更强一些。判定可以是 1 个条件，也可以是多个条件的组合。

条件覆盖	每一个判定语句中每个逻辑条件的各种可能的值至少满足一次。	条件覆盖和判断覆盖没有包含关系。
判断/条件覆盖	判定中每个条件的所有可能取值（真/假）至少出现一次，并使每个判定本身的判定结果（真/假）也至少出现一次。	同时满足判定覆盖和条件覆盖。
条件组合覆盖	每个判定中的各种可能值的组合都至少出现一次。	同时满足判定覆盖、条件覆盖、判定/条件覆盖。
路径覆盖	覆盖被测试程序中所有可能的路径。	可以对程序进行彻底的测试，比语句覆盖、条件覆盖、判定覆盖、条件判定覆盖及条件组合覆盖的覆盖面都广。

4、各测试阶段的任务：

（1）**验收测试**：有效性测试、软件配置审查、验收测试。

（2）**系统测试**：恢复测试、安全性测试、强度测试、性能测试、可靠性测试和安装测试。

（3）**集成测试**：模块间的接口和通信。

（4）**单元测试**：模块接口、局部数据结构、边界条件、独立的路径、错误处理。

（5）其他测试：**回归测试**（修改软件后进行的测试，防止引入新的错误）。负载测试（对软件负载能力的测试）。**压力测试**（对软件超负荷条件下运行情况的测试）。

5、测试的基本原则

尽早、不断地进行测试；

程序员避免测试自己设计的程序；

既要选择有效、合理的数据，也要选择无效、不合理的数据；

修改后应进行回归测试；

尚未发现的错误数量与该程序已发现错误数成正比。

6、McCabe 复杂度计算

（1）McCabe 复杂度计算公式：V（G）=m-n+2，其中 m 是有向弧的条数，n 是结点数。

（2）对于伪代码可以先转换为程序流程图，对程序流程图可以最终转换为结点图处理，转换时注意将交点的地方标注为新的结点，以最终的结点图带入公式计算其 McCabe 复杂度。

七、软件维护（5 星）

1、更正性维护：针对真实存在并已经发生的错误进行的维护行为。

2、预防性维护：针对真实存在但还未发生的错误进行的维护。

3、适应性维护：指使应用软件适应信息技术变化和管理需求变化而进行的修改。企业的外部市场环境和管理需求的不断变化也使得各级管理人员不断提出新的信息需求。

4、完善性维护：扩充功能和改善性能而进行的修改。对已有的软件系统增加一些在系统分析和设计阶段中没有规定的功能与性能特征。

八、软件质量保证（3 星）

1、功能性：适合性、准确性、互操作性、安全保密性。

2、可靠性：成熟性、容错性、易恢复性。

3、易用性：易理解性、易学性、易操作性、吸引性。

4、效率：时间特性、资源利用性。

5、维护性：易分析性、稳定性、易测试性、易改变性。

6、可移植性：适应性、易安装性、共存性、易替换性。

九、软件文档（2 星）

1、开发文档（开发人员）

（1）可行性研究和项目任务书

（2）需求规格说明

（3）功能规格说明

（4）设计规格说明（包括程序和数据规格说明）

（5）开发计划

（6）软件集成和测试计划

（7）质量保证计划、标准、进度

（8）安全和测试信息

2、产品文档（用户）

（1）培训手册

（2）参考手册和用户指南

（3）软件支持手册

（4）产品手册和信息广告

3、管理文档（负责人）

（1）开发过程的每个阶段的进度和进度变更的记录

（2）软件变更情况的记录

（3）相对于开发的判定记录

（4）职责定义

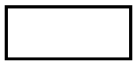
第 2 章【软件设计】数据流图解题技巧

一、理论基础

参照第 4 章《面向对象技术》第二节《UML》相关内容。

二、解题技巧

1、数据流图图示

元素	说明	图元
数据流 （名词）	由一组固定成分的数据组成，表示数据的流向。每个数据流通常有一个合适的名词，反映数据流的含义（数据流必须和加工相关，即从加工流向加工、数据源流向加工或加工流向数据源。）	
加工 （动名词）	加工描述了输入数据流到输出数据流之间的变换，也就是输入数据流做了什么处理后变成了输出数据流。	
数据存储 （文件）	用来表示暂时存储的数据，每个文件都有名字。流向文件的数据流表示写文件，流出的表示读文件。	
外部实体	指存在于软件系统外的人员或组织。	

2、填空技巧

（1）补充实体

实体可能是：

人物角色：如客户、管理员、主管、经理、老师、学生

组织机构：如银行、供应商、募捐机构

外部系统：如银行系统、工资系统、后台数据库（当要开发的是中间件时）

（2）补充存储

存储的文字方面特征：“**文件”“**表”“**库”“**清单”“**档案”本考点的基本考法是与内存地址计算、IP 地址计算结合考查。

（3）补充加工名

加工是用于处理数据流的，所以要补充加工名，可以把该加工涉及到的数据流，在说明中标识出来，再在数据流名称所在的句子中，找“动词+名词”的结构，分析是否可作为加工。

“动词+名词”如：生成报告，发出通知，批改作业，记录分数，当然这只是普遍情况，也有例外，如物流跟踪、用户管理。

（4）补充数据流

数据平衡原则：

顶层图与 0 层图对比，是否有顶层图有，但 0 层图无的数据流，或反之。

检查图中每个加工，是否存在只有入没有出，或只有出没有入，或根据输入的数据无法产生对应的输出的情况。

按题目说明与图进行匹配：

说明中的每一句话，都能与图中有对应关系，当把说明中的实体与数据流标识出来之后，容易缩小对应范围，找出纰漏。

3、主观题分析

（1）数据字典

符　　号	含　　义	举 例 说 明
=	被定义为	x=3，表示 x 被定义为 3
+	与	x=a+b，表示 x 由 a 和 b 组成
[…，…]或[…|…]	或	x=[a，b]，x=[a|b]，表示 x 由 a 或由 b 组成
{…}	重复	x={a}，表示 x 由 0 个或多个 a 组成
(…)	可选	x=（a），表示 a 可在 x 中出现，也可以不出现

（2）数据流图常见的 3 种错误

加工只有输入没有输出，称之为“黑洞”；

加工只有输出没有输入，称之为“奇迹”；

加工中输入不足以产生输出，称之为“灰洞”。

（3）子图与父图保持平衡

父图与子图之间平衡是指任何一张 DFD 子图边界上的输入/输出数据流必须与其父图对应加工的输入/输出数据流保持一致。如果父图中某个加工的一条数据流对应于子图中的几条数据流，而子图中组成这些数据流的数据项全体正好等于父图中的这条数据流，那么它们仍然是平衡的。

（4）对加工进行结构化描述

结构化语言是一种介于自然语言和形式化语言之间的半形式化语言，是自然语言的一个受限子集。

外层：用来描述控制结构，采用顺序、选择和重复 3 种基本结构。

①顺序结构，一组祈使语句、选择语句、重复语句的顺序排列。

②选择结构，一般用 IF-THEN-ELSE-ENDIF、CASE-OF-ENDCASE 等关键词。

③重复结构，一般用 DO-WHILE-ENDDO、REPEAT-UNTIL 等关键词。

内层：一般采用祈使语句的自然语言短语，使用数据字典中的名词和游戏的自定义词，其动词含义要具体，尽量不用形容词和副词来修饰，还可使用一些简单的算法运算和逻辑运算符号。

第 3 章 项目管理

一、进度管理（5 星）

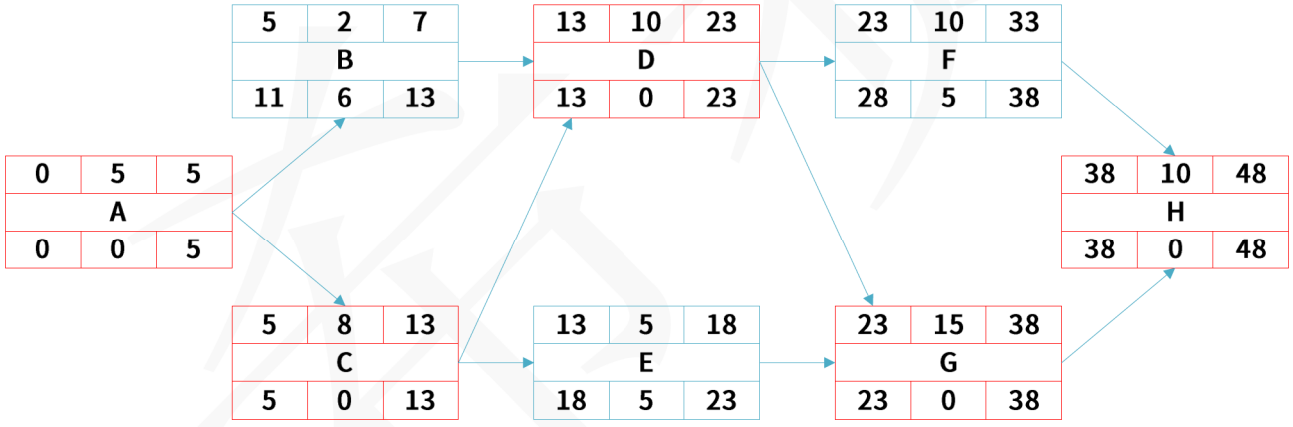
1、甘特图能够清晰描述每个任务的开始/结束时间及各任务之间的并行性，也可以动态地反映项目的开发进展情况，但难以反映多个任务之间存在的逻辑关系；PERT 利用项目的网络图和各活动所需时间的估计值（通过加权平均得到的）去计算项目总时间，强调任务之间的先后关系，但不能反映任务之间的并行性，以及项目的当前进展情况。

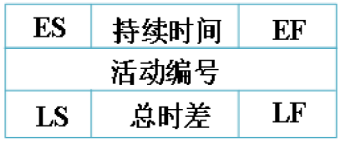
2、关键路径法是图中源点至汇点的最长路径，关键路径的时间称之为项目工期，也表述为项目完成所需的最少时间。

3、总时差是在不延误总工期的前提下，该活动的机动时间。一般在图中，以最晚结束时间减去最早结束时间求取，或以最晚开始时间减去最早开始时间求取。

4、对于网络图我们一般采用关键路径分析法处理，关键路径分析法是利用进度模型时使用的一种进度网络分析技术。沿着项目进度网络路线进行正向与反向分析，从而计算出所有计划活动理论上的最早开始与完成日期、最迟开始与完成日期，不考虑任何资源限制。

5、单代号网络图，结点表示活动，箭线表示活动与活动间的依赖关系。





Earliest 最早的

ES：最早开始时间

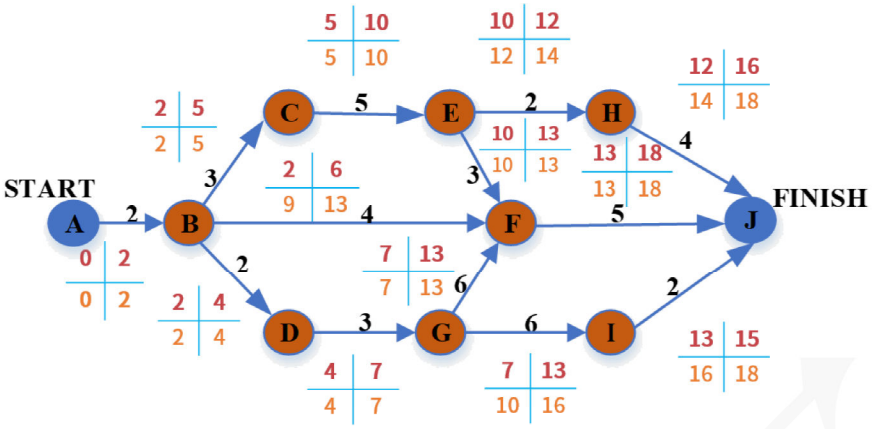
LS：最迟开始时间

Last 最晚的

EF：最早完成时间

LF：最迟完成时间

6、双代号网络图中，结点表示里程碑，箭线表示活动。



二、风险管理（3 星）

【考法分析】

本知识点的考查形式，主要有：对风险相关概念描述判断正误；计算风险曝光度。

【要点分析】

1、风险的特性：具有不确定性，可能会造成损失。

2、风险的类别：项目风险涉及到各种形式的预算、进度、人员、资源以及客户相关的问题，并且可能导致项目损失。技术风险涉及到技术相关的可能会导致项目损失的问题。商业风险与市场因素相关。社会风险涉及到政策、法规等因素。

3、风险暴露：又称风险曝光度，测量的是资产的整个安全性风险，它将表示实际损失的可能性与表示大量可能损失的资讯结合到单一数字评估中。在形式最简单的定量性风险分析中，风险曝光度可通过将风险可能性及影响相乘算出。

风险曝光度（RiskExposure）=错误出现率（风险出现率）×错误造成损失（风险损失）。

三、成本管理（2 星）

COCOMO II 是一种层次结构的估算模型，被分为 3 个阶段性模型。

应用组装模型。在软件工程的前期阶段使用，这时用户界面的原型开发、对软件和系统交互的考虑、性能的评估以及技术成熟度的评价是最重要的。

早期设计界面模型。在需求已经稳定并且基本的软件体系结构已经建立时使用。

体系结构阶段模型。在软件的构造过程中使用。

规模估算选择：对象点，功能点，代码行

应用组装模型使用的是对象点；早期设计阶段模型使用的是功能点，功能点可以转换为代码行。

四、沟通管理（1 星）

1、有主程序员：n 个成员小组，1 个主程序员，普通程序员只需要与主程序员沟通。

沟通路径：n-1。

2、无主程序员：n 个成员的项目小组，相互之间都可以沟通。

沟通路径：n（n-1）/2。

第 4 章 面向对象技术

一、面向对象基础（5 星）

1、基本概念

对象：属性（数据）+方法（操作）+对象 ID

封装：隐藏对象的属性和实现细节，仅对外公开接口（信息隐藏技术）。

类（实体类/控制类/边界类）

接口：一种特殊的类，他只有方法定义没有实现。

继承与泛化：复用机制。

重置/覆盖（Overriding）：在子类中重新定义父类中已经定义的方法。

动态绑定：根据接收对象的具体情况将请求的操作与实现的方法进行连接（运行时绑定）。

多态：不同对象收到同样的消息产生不同的结果，多态实质上是将子类的指针对象或者引用对象传递给父类指针对象后，通过这个父类指针对象调用的函数（此函数在父类中声明为虚函数，且在各个子类中重写这个函数），不是父类中定义的，而是传递进来的子类对象中重写的函数。（软设考试中对于多态分类只出现过过载多态-过载多态：同一个名字在不同的上下文中所代表的含义不同。）。

重载：一个类可以有多个同名而参数类型不同的方法。

类属类：类的模板。

消息和消息通信：对象之间进行通信的一种构造叫作消息。消息是异步通信的（消息传递：接收到信息的对象经过解释，然后予以响应）

【面向对象设计原则】

（1）**单一职责原则**：设计目的单一的类。

（2）**开放-封闭原则**：对扩展开放，对修改封闭。

（3）李氏（Liskov）替换原则：子类可以替换父类。

（4）**依赖倒置原则**：要依赖于抽象，而不是具体实现；针对接口编程，不要针对实现编程。

（5）**接口隔离原则**：使用多个专门的接口比使用单一的总接口要好。

（6）**组合重用原则**：要尽量使用组合，而不是继承关系达到重用目的。

（7）**迪米特（Demeter）原则（最少知识法则）**：一个对象应当对其他对象有尽可能少的了解。

【面向对象其他设计原则】

（1）重用发布等价原则：重用的粒度就是发布的粒度。

（2）共同封闭原则：包中的所有类对于同一性质的变化应该是共同封闭的。一个变化若对一个包产生影响，则将对该包里的所有类产生影响，而对于其他的包不造成任何影响。

（3）共同重用原则：一个包里的所有类应该是共同重用的。如果重用了包里的一个类，那么就要重用包中的所有类。

（4）无环依赖原则：在包的依赖关系图中不允许存在环，即包之间的结构必须是一个直接的无环图形。

（5）稳定依赖原则：朝着稳定的方向进行依赖。

（6）稳定抽象原则：包的抽象程度应该和其稳定程度一致。

2、面向对象开发过程

面向对象分析：认定对象（名词）；组织对象（抽象成类）；对象间的相互作用；基于对象的操作。

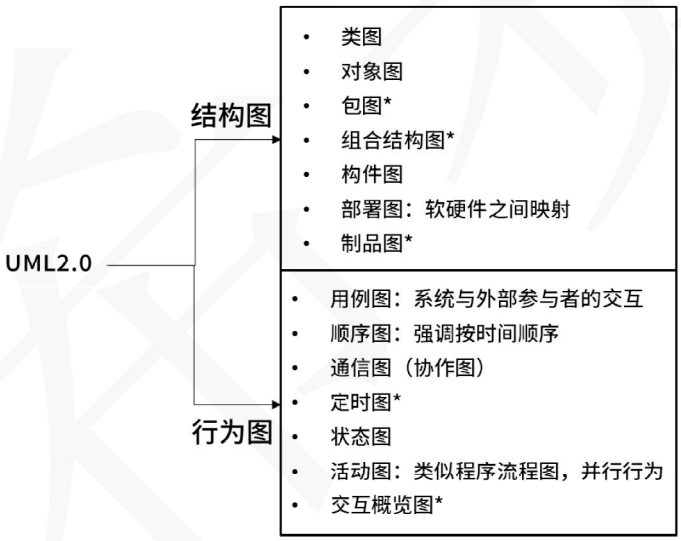
面向对象设计：识别类及对象；定义属性；定义服务；识别关系；识别包。

面向对象开发：程序设计范型；选择一种 OOPL。

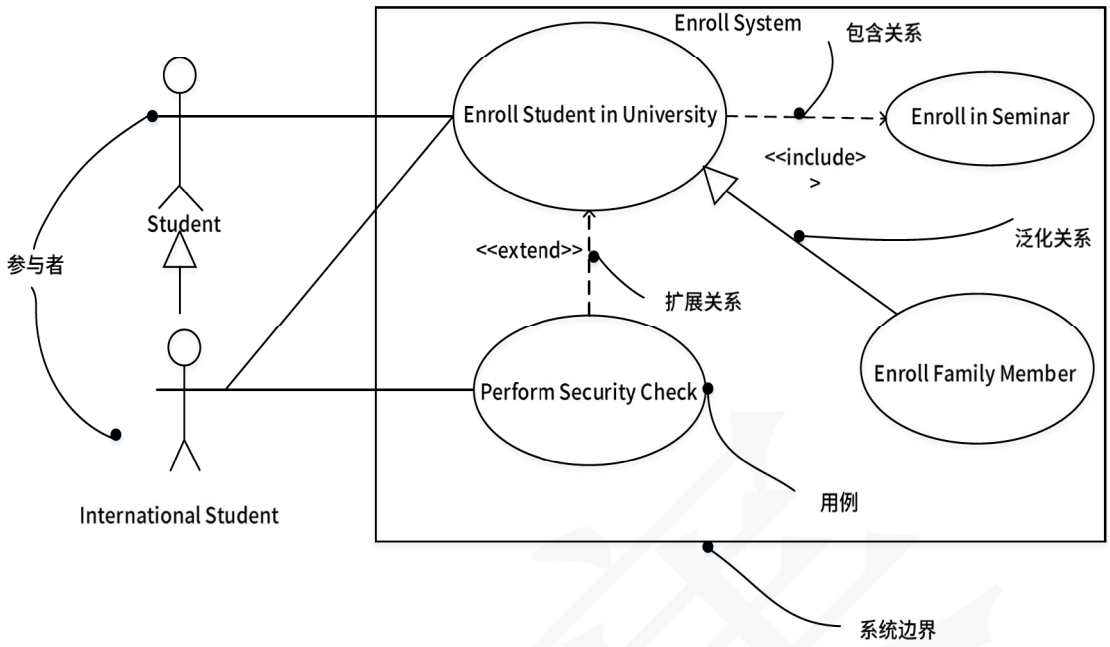
面向对象测试：算法层；类层；模板层；系统层。

二、UML（5 星）

1、UML 图分类：



2、用例图：用例图描述一组用例、参与者及它们之间的关系。



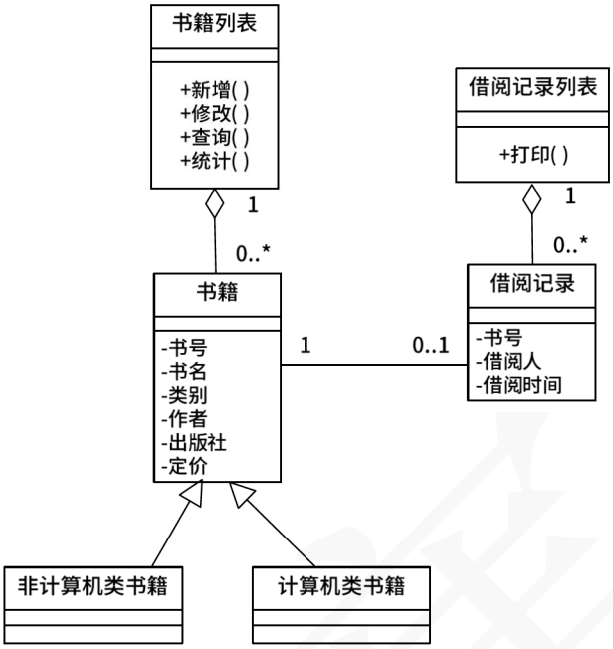
用例之间的关系：

包含关系：其中这个提取出来的公共用例称为抽象用例，而把原始用例称为基本用例或基础用例，当可以从两个或两个以上的用例中提取公共行为时，应该使用包含关系来表示它们。

扩展关系：如果一个用例明显地混合了两种或两种以上的不同场景，即根据情况可能发生多种分支，则可以将这个用例分为一个基本用例和一个或多个扩展用例，这样使描述可能更加清晰。

泛化关系：当多个用例共同拥有一种类似的结构和行为的时候，可以将它们的共性抽象成为父用例，其他的用例作为泛化关系中的子用例。在用例的泛化关系中，子用例是父用例的一种特殊形式，子用例继承了父用例所有的结构、行为和关系。

3、类图（class diagram）：类图描述一组类、接口、协作和它们之间的关系。在 OO 系统的建模中，最常见的图就是类图。类图给出了系统的静态设计视图，活动类的类图给出了系统的静态进程视图。对象图（object diagram）：对象图描述一组对象及它们之间的关系。对象图描述了在类图中所建立的事物实例的静态快照。和类图一样，这些图给出系统的静态设计视图或静态进程视图，但它们是从真实案例或原型案例的角度建立的。



类之间的关系：

依赖关系：一个事物发生变化影响另一个事物。

泛化关系：特殊/一般关系。

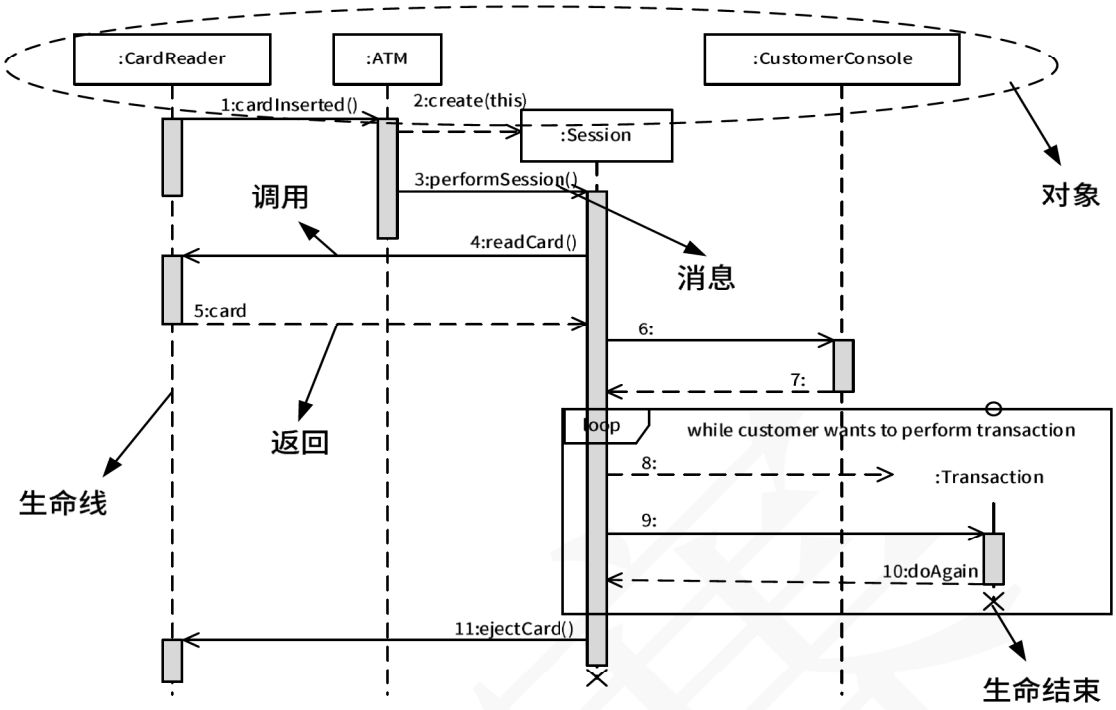
关联关系：描述了一组链，链是对象之间的连接。

聚合关系：整体与部分生命周期不同。

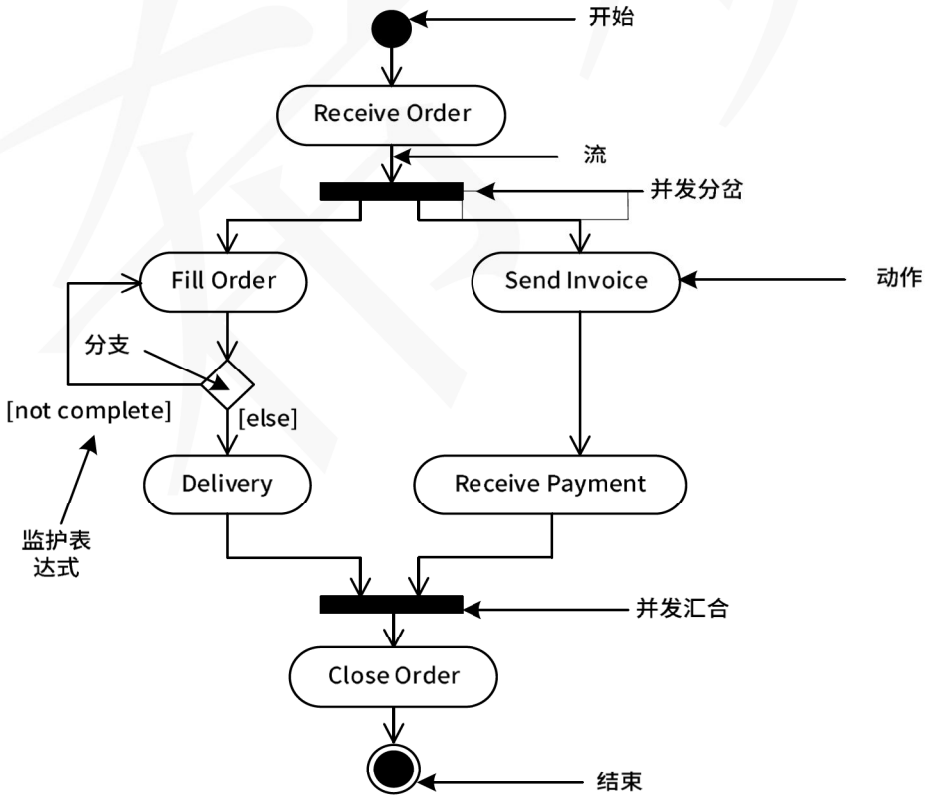
组合关系：整体与部分生命周期相同。

实现关系：接口与类之间的关系。

4、顺序图（sequence diagram，序列图）：顺序图是一种交互图（interaction diagram），交互图展现了一种交互，它由一组对象或参与者以及它们之间可能发送的消息构成。交互图专注于系统的动态视图。顺序图是强调消息的时间次序的交互图。

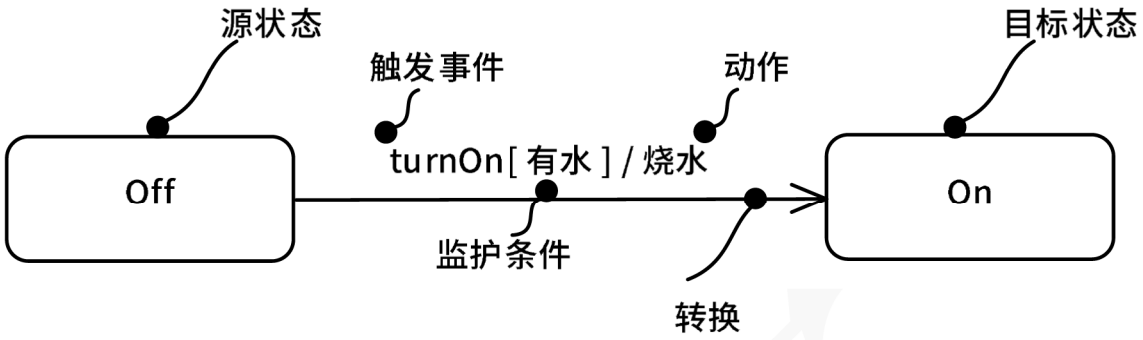


5、活动图（activity diagram）：活动图将进程或其他计算结构展示为计算内部一步步的控制流和数据流。活动图专注于系统的动态视图。它对系统的功能建模和业务流程建模特别重要，并强调对象间的控制流程。

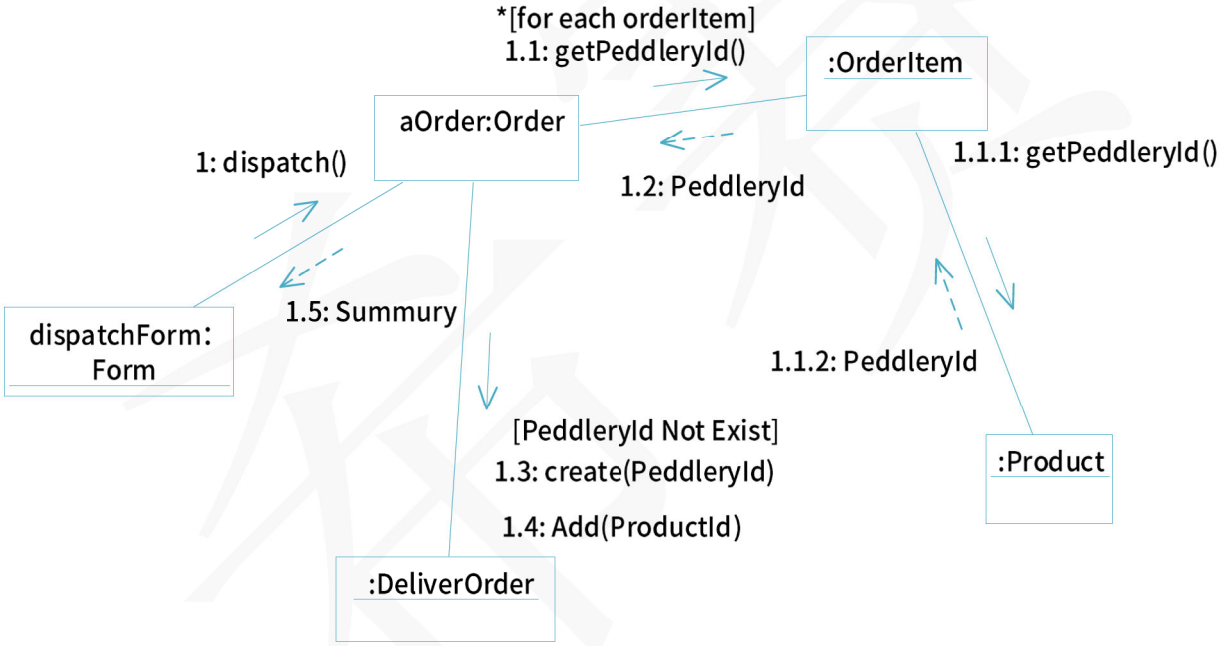


6、状态图（state diagram）：状态图描述一个状态机，它由状态、转移、事件和活动组成。状

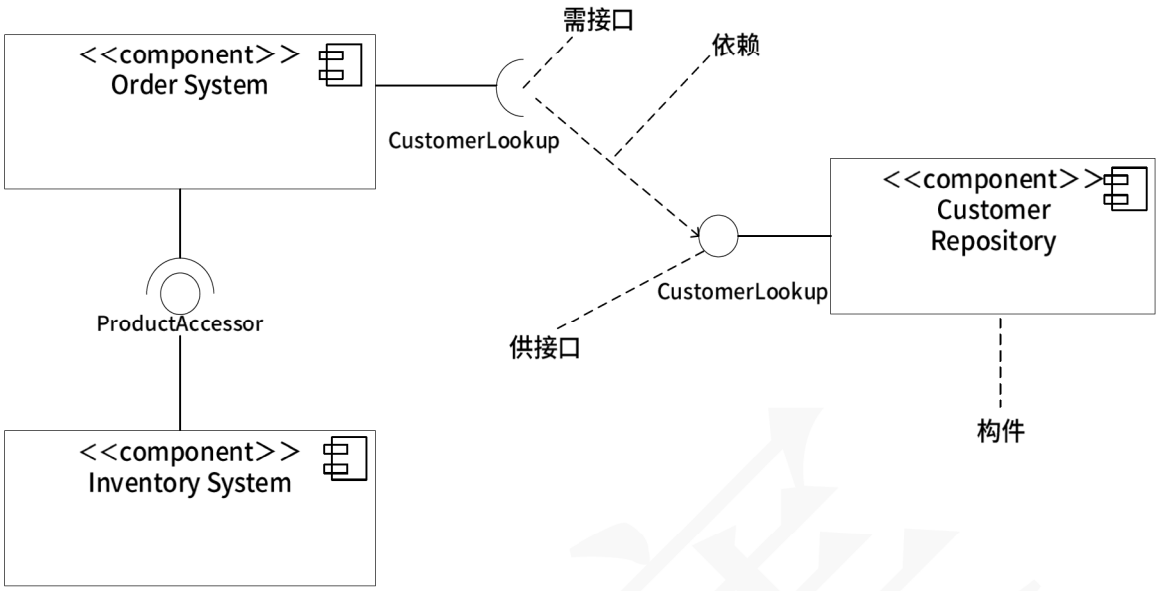
态图给出了对象的动态视图。它对于接口、类或协作的行为建模尤为重要，而且它强调事件导致的对象行为，这非常有助于对反应式系统建模。



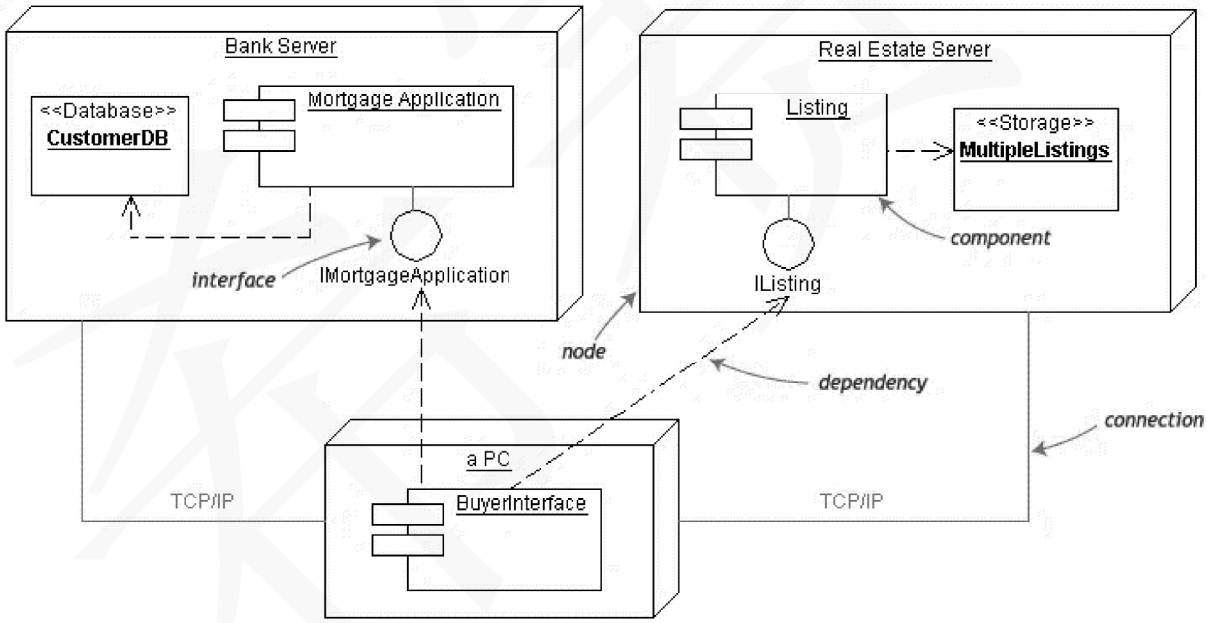
7、通信图（communication diagram）：通信图也是一种交互图，它强调收发消息的对象或参与者的结构组织。顺序图和通信图表达了类似的基本概念，但它们所强调的概念不同，顺序图强调的是时序，通信图强调的是对象之间的组织结构（关系）。



8、构件图（component diagram）：构件图描述一个封装的类和它的接口、端口，以及由内嵌的构件和连接件构成的内部结构。构件图用于表示系统的静态设计实现视图。对于由小的部件构建大的系统来说，构件图是很重要的。构件图是类图的变体。



9、部署图（deployment diagram）：部署图描述对运行时的处理节点及在其中生存的构件的配置。部署图给出了架构的静态部署视图，通常一个节点包含一个或多个部署图。



三、设计模式（5 星）

1、设计模式的分类：

	创建型	结构型	行为型	
类	factory method 工厂方法模式	adapter 适配器模式（类和对象）	template method 模板方法模式	interpreter 解释器模式
对象	abstract factory 抽象工厂模式 prototype 原型模式 singleton 单例模式 builder 构建器模式	bridge 桥接模式 composite 组合模式 decorator 装饰模式 facade 外观模式 flyweight 享元模式 proxy 代理模式	chain of responsibility 职责链模式 command 命令模式 iterator 迭代器模式 mediator 中介者模式	memento 备忘录模式 observer 观察者模式 state 状态模式 strategy 策略模式 visitor 访问者模式

2、设计模式应用场景和记忆关键字：

（1）创建型模式

设计模式名称	简要说明	速记关键字
Factory Method 工厂方法模式	定义一个创建对象的接口，但由子类决定需要实例化哪一个类。 工厂方法使得子类实例化的过程推迟	动态生产对象
Abstract Factory 抽象工厂模式	提供一个接口，可以创建一系列相关或相互依赖的对象，而无需指定它们具体的类	生产成系列对象
Builder 构建器模式	将一个复杂类的表示与其构造相分离，使得相同的构建过程能够得出不同的表示	复杂对象构造
Prototype 原型模式	用原型实例指定创建对象的类型，并且通过拷贝这个原型来创建新的对象	克隆对象
Singleton 单例模式	保证一个类只有一个实例，并提供一个访问它的全局访问点	单实例

（2）结构型模式

设计模式名称	简要说明	速记关键字
Adapter 适配器模式	将一个类的接口转换成用户希望得到的另一种接口。它使原本不相容的接口得以协同工作	转换接口
Bridge 桥接模式	将类的抽象部分和它的实现部分分离开来，使它们可以独立地变化	继承树拆分
Composite 组合模式	将对象组合成树型结构以表示“整体-部分”的层次结构，使得用户对单个对象和组合对象的使用具有一致性	树形目录结构
Decorator 装饰模式	动态地给一个对象添加一些额外的职责。它提供了用子类扩展功能的一个灵活的替代，比派生一个子类更加灵活	动态附加职责
Facade 外观模式	定义一个高层接口，为子系统中的一组接口提供一个一致的外观，从而简化了该子系统的使用	对外统一接口
Flyweight 享元模式	提供支持大量细粒度对象共享的有效方法	汉字编码
Proxy 代理模式	为其他对象提供一种代理以控制这个对象的访问	快捷方式

（3）行为型模式

设计模式名称	简要说明	速记关键字
Chain of Responsibility 职责链模式	通过给多个对象处理请求的机会，减少请求的发送者与接收者之间的耦合。将接收对象链接起来，在链中传递请求，直到有一个对象处理这个请求	传递职责
Command 命令模式	将一个请求封装为一个对象，从而可用不同的请求对客户进行参数化，将请求排队或记录请求日志，支持可撤销的操作	日志记录，可撤销
Interpreter 解释器模式	给定一种语言，定义它的文法表示，并定义一个解释器，该解释器用来根据文法表示来解释语言中的句子	虚拟机的机制
Iterator 迭代器模式	提供一种方法来顺序访问一个聚合对象中的各个元素，而不需要暴露该对象的内部表示	数据集
Mediator 中介者模式	用一个中介对象来封装一系列的对象交互。它使各对象不需要显式地相互调用，从而达到低耦合，还可以独立地改变对象间的交互	不直接引用
Memento 备忘录模式	在不破坏封装性的前提下，捕获一个对象的内部状态，并在该对象之外保存这个状态，从而可以在以后将该对象恢复到原先保存的状态	游戏存档
Observer 观察者模式	定义对象间的一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并自动更新	联动
State 状态模式	允许一个对象在其内部状态改变时改变它的行为	状态变成类
Strategy 策略模式	定义一系列算法，把它们一个个封装起来，并且使它们之间可互相替换，从而让算法可以独立于使用它的用户而变化	多方案切换
Template Method 模板方法模式	定义一个操作中的算法骨架，而将一些步骤延迟到子类中，使得子类可以不改变一个算法的结构即可重新定义算法的某些特定步骤	框架
Visitor 访问者模式	表示一个作用于某对象结构中的各元素的操作，使得在不改变各元素的类的前提下定义作用于这些元素的新操作	数据与操作分离

第 5 章【软件设计】UML 建模解题技巧

一、理论基础

参照第 4 章《面向对象技术》第二节《UML》相关内容。

二、解题技巧

1、用例图常见考查形式

（1）补充参与者：参与者一般为外部实体，可以是人，也可以是外部系统。

（2）补充用例：根据题干描述补充缺失的用例，在补充过程中需要参照用例图中用例间的关系。（用例名从题干中直接获取或提炼）

（3）分析用例间的关系

（4）其他：可能会出现其他题型，比如添加新的用例等。

2、类图/对象图常见考查形式（常与其他 UML 图结合考查）

（1）补充类名/对象名：一般为名词，来源于题干描述。

（2）补充多重度：多重度体现类与类或对象与对象之间的对应关系，有 0…1，1..1，1…*/n，*…*/m…n。

（3）分析类与类或对象与对象之间的关系。

3、常见考查形式

（1）补充类名/对象名

（2）补充消息

4、常见考查形式

（1）补充动作名称

（2）补充监护表达式

（3）分析并发关系

（4）活动图与状态图对比，活动图与进程图关系。

5、状态图常见考查形式

（1）补充状态

（2）补充触发事件、监护条件、动作

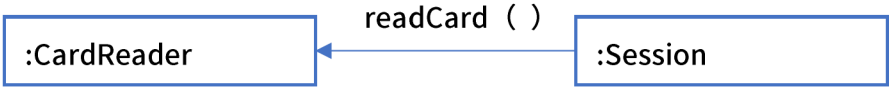
（3）与状态模式结合考查

6、通信图常见考查形式

（1）补充对象

（2）补充消息

7、消息的所属判断：箭头上的消息属于箭头指向的对象/类。




```
class   CardReader{
    /*代码省略*/
    readCard（   ）  {/*代码省略*/}
}
class   Session{
    /*代码省略*/
    function（CardReader CR）{
        CR.readCard（   ）；
    }
    }
```

第 6 章【软件设计】面向对象程序设计（JAVA）解题技巧

一、理论基础

参照第 4 章《面向对象技术》第三节《设计模式》相关内容。

二、解题技巧

1、了解类与类（接口）之间的关系——接口是一种特殊的类

（1）接口的概念：接口只是说明类应该做什么，但并不指定应该如何去做。在实际开发过程中，通过类来实现接口。接口一般只有方法名，没有方法体。实现接口就是让其既有方法名又有方法体。

（2）接口的声明：接口用关键字“interface”来声明。接口的形式跟类很相似。例 1 格式如下：

```
interface [接口名]{
        …
}
```

（3）接口的实现

实现接口的类表示例 2 如下：（接例 1 接口的声明）

```
class  [类名]  implements  [接口名]{
        …
}
```

（4）特殊的接口方法：接口中的方法只有方法名没有方法体。

（注意：没有花括号）例 3 如下：

```
interface [接口名]{
        [返回类型 void/String/…] [方法名] （参数列表）；
}
```

（5）接口类与其实现类的对应关系：接口的用处就是让类通过实现它来执行一定的功能。因此实现接口的类，要实现接口的类。对于实现类或接口类有方法或属性的缺失，则可以根据二者的对应关系进行补充。一般实现类的方法是对接口类方法的具体实现。

2、继承

（1）继承的概念：继承，就是在已有类（父类）的基础上进行扩展，从而产生新的类（子类）。父类拥有自己的属性和方法，这些子类都可以继承。子类除了拥有父类的属性和方法外，还可以有自己的特性。

（2）继承的表示：继承通过关键字“extends”表示。结构例 4 如下：

```
class [子类名] extends [父类名]{
        …
}
```

（3）父类与子类之间的对应关系：子类的属性和方法有部分是从父类继承而来，一般而言，父类和子类之间同名的属性和方法具有一定的对应关系，子类也可以对父类的方法进行重写。

（4）super 和 this 的使用

在继承关系中为了区分父类和子类，会用 super 代替父类指针，this 代替子类指针。举例如下：

```
class [父类名] {
        [属性 1];
        [父类名]（[属性 1]）{  属性 1=1;  }
}
class [子类名]{
        [属性 2];
        [子类名]（[属性 1],[属性 2]）{
        super（[属性 1]）;          //这里的 super（[属性 1]）时父类带属性 1 参数的构造函数
        this.[属性 2]=[属性 2];     //这里的 this 是当前类指针
        //加上 this 和 super 可以区分同名的属性或方法。子类可以利用 super 指针调用父类的属性和方法。
}
}
```

3、了解类的结构——属性和构造函数

（1）默认构造函数和非默认的构造函数：对类实例化创建对象时，系统会调用构造函数对其所属成员进行初始化。

在 JAVA 中，一般以与类名同名的方法作为构造函数。在实际开发过程中，系统会默认不带参数的构造函数，如果需要带参数的构造函数，需要明确写出该构造函数。构造函数结构例 5 如下：

```
class [类 1]{
        [属性 1];
        //[类 1]（）{}  //默认无参构造函数
        [类 1]（属性 1  形参 1）{
            this.属性 1=形参 1;    //一般这里的形参命名与属性名一致
}
```

```
}
```

（2）继承中的构造函数：实际上，在创建子类对象时，会先执行其父类的构造函数，然后执行子类的构造函数，最后完成对象的创建。即创建子类对象时，先调用父类构造函数，初始化继承自父类的成员，随后调用子类构造函数，初始化子类的成员。例 6 如下：

```
class [父类名]{
        [属性 1];
    [父类名]（）{
            [属性 1]=10;
            System.out.println（“这是父类构造函数”）；
}
}
class   [子类名]{
        [属性 2];
        [子类名]（）{
            [属性 2]=20;
            System.out.println（“这是子类构造函数”）；
}
}
//主函数测试
public class test{
        public static void main（String [] args）
        {
            [子类名] [子类对象名]= new [子类名]  （）；
System.out.println（[子类对象名] .[属性 1]+“ ”+[子类对象名] .[属性 2]）；
}
}
//这里的输出结构应该是：
//    这是父类构造函数
//    这是子类构造函数
//    10 20
```

（3）get 方法和 set 方法

在 JAVA 中会有对类中属性的 get 和 set 方法。在实际开发过程中，会对类的属性设定私有

private 限定，只有本类中的方法可以访问，拒绝其他类的访问。同时，会在本类中设定 get（）和 set（）方法，对相关属性进行操作。这些在开发工具中，有时候会默认自动给出。结构如下所示：

```
class [类名]{
        private [数据类型] [属性 1];
        public [数据返回类型] get 属性 1（）{
                return [属性 1];
}
public void set 属性 1（[数据类型] [形参]）{
                this.[属性 1]=[形参]；  //这里的形参名一般与属性名一致
}
}
```

（4）其他函数：类中函数成员格式如下所示：

[访问控制符] [返回类型] [方法名] （形参 1 类型 形参名 1，形参 2 类型 形参名 2，…）{ … }

访问控制符可以为：private/public/default/protected/abstract

返回类型为该方法返回数据的数据类型。

返回语句 return [返回参数]

如果返回类型为 void，即返回类型为空时，不需要 return 语句。

4、访问控制符/类修饰符

private

（1）用 private 修饰的成员变量与成员方法只能在类的内部被访问，类的外部不能访问。

public

（1）用 public 修饰数据表示所有的类都可以访问。

（2）用 public 修饰类，也表示所有类中可以访问。

default

（1）用 default 修饰成员数据表示只有用一个包里的类才能访问。

（2）用 default 修饰类也表示只有一个包里的类才能访问。当在类前不加任何控制符时，默认就是 default。

protected

（1）用 protected 修饰成员数据表示不仅同一个包中的类可以访问，位于其他包中的子类也可以访问。

abstract

（1）abstract 可以用来修饰方法或类。具有一个或多个抽象方法的类，本身需要被定义为抽象

类。抽象类不仅可以有抽象方法，还可以有具体方法。

（2）抽象类可以被继承，如果其子类没有实现抽象类全部的抽象方法，则子类也是抽象类，需要用 abstract 修饰。

（3）含有抽象方法的一定是抽象类，但抽象类可以没有抽象方法。抽象类不能实例化，不能 new 一个对象，但可以声明一个抽象类的变量指向具体的子类对象。

5、了解对象的实例化

（1）new 一个对象

在主函数中，需要对类进行实例化才能够调用。而我们对类的实例化，一般用的是 new。结构如下：

[类名]　[对象名] = new [类名]（参数列表）；

（2）与超类相关的实例化

//创建子类对象并转到父类

[父类名] [对象名] = new [子类名]（参数列表）；

//此时以对象名调用某方法（子类从父类直接继承而来的方法），虽然将父类对象句柄指向了子类对象，实际上操作的还是子类对象，只不过将对象句柄声明为父类的数据类型，此时编译器根据实际情况选择了子类的函数。

6、了解方法调用的格式

[对象名] . [方法名] ;

7、了解特殊的数据结构接口

List（表单）

（1）将元素放到指定集合的结尾

[表单名]. add（[元素名]）；

（2）将集合 2 放到指定集合 1 的指定位置

[表单名 1].addAll（[位置序号]，[表单名 2]）；

（3）清除集合

[表单名].clear（）；

Iterator（迭代器）--用迭代器访问表单

第 7 章 数据结构

一、线性结构（5 星）

1、顺序表和链表的对比：

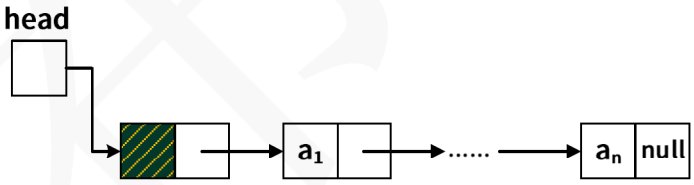
性能类别	具体项目	顺序存储	链式存储
空间性能	存储密度	=1，更优	<1
	容量分配	事先确定	动态改变，更优
时间性能	查找运算	O（n）	O（n）
	读运算	O（1），更优	O（n），最好情况为 1，最坏情况为 n
	插入运算	O（n），最好情况为 0，最坏情况为 n	O（1），更优
	删除运算	O（n）	O（1），更优

2、**顺序表**：线性表顺序存储，即用一组地址连续的存储单元依次存储线性表中的数据元素，从而使得逻辑上相邻的两个元素，在物理上也相邻。在存储之前，先根据线性表的长度分配连续的物理空间，因此后续不方便扩展。只需要存储数据元素，不需要存储元素的逻辑关系因此存储密度为 1。

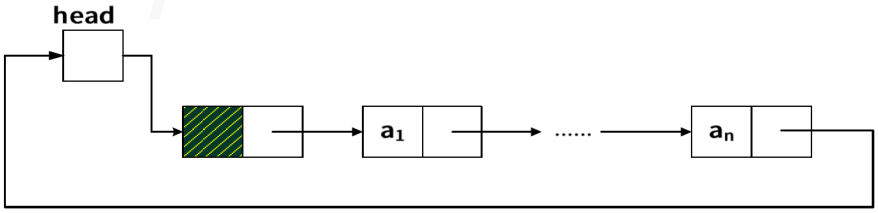
3、**链表**：线性表链式存储，即用通过指针链接起来的结点来存储数据元素，存储各数据元素的结点物理上不要求连续，因此后期扩展方便。因为物理上不连续，需要同时存储各元素之间的逻辑关系，存储密度小于 1。

4、链表的分类：单链表、双链表、循环链表。

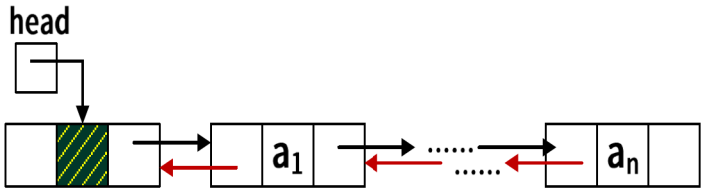
单链表



循环链表



双向链表



5、特殊的线性结构：队列（先进先出）、栈（先进后出）。

循环队列：head=tail 表示队空；（tail+1）%size=head 表示队满。

6、特殊的线性结构：字符串

串的定义：串是仅由字符构成的有限序列，是一种线性表。一般记为 S=“a1a2a3…an”，其中，S 是串名，引号括起来的字符序列是串值。

7、基本概念

（1）空串与空格串

空串：长度为零，不包含任何字符。

空格串：由一个或多个空格组成的串。虽然空格是一个空白字符，但它也是一个字符，在计算串长度时要将其计算在内。

（2）子串与子序列

子串：由串中任意长度的连续字符构成的序列称为子串。含有子串的串称为主串。子串在主串中的位置是指子串首次出现时，该子串的第一个字符在主串中的位置。空串是任意串的子串。

子序列：一个串的“子序列”（subsequence）是将这个串中的一些字符提取出来得到一个新串，并且不改变它们的相对位置关系。

（3）串比较与串相等

串比较：两个串比较大小时以字符的 ASCII 码值（或其他字符编码集合）作为依据。实质上，比较操作从两个的第一个字符开始进行，字符的码值大者所在的串为大；若其中一个串先结束，则以串长较大者为大。

串相等：指两个串长度相等且对应序号的字符也相同。

（4）模式匹配：子串的定位操作通常称为串的模式匹配。（子串也称为模式串）

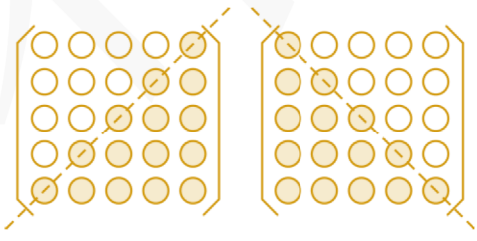
KMP 算法：其改进之处在于——每当匹配过程中出现相比较的字符不相等时，不需要回退到主串的字符位置指针，而是利用已经得到的“部分匹配”结果将模式串向右“滑动”尽可能远的距离，再继续进行比较。在 KMP 算法中，依据模式串的 next 函数值实现子串的滑动。若令 next[j]=k，则 next[j]表示当模式串中的 p_j 与主串中相应字符不相等时，令模式串的 $p_{next[j]}$ 与主串的相应字符进行比较。（j=next[j]）

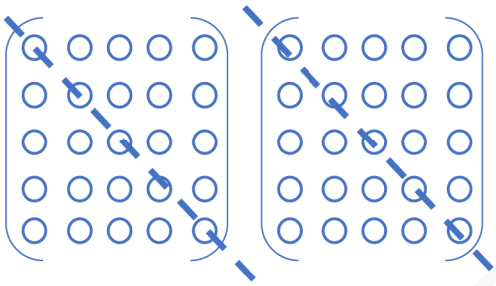
二、数组与矩阵（3 星）

1、对于数组或矩阵，存储时注意存储方式是按行存储还是按列存储，二者结果有区别。

2、对于**存储位置的计算**，可以理解为计算当前位置以要求的存储方式存放时，前面已经存放了多少个元素。

数组类型	存储地址计算
一维数组 a[n]	a[i]的存储地址为：a+i*len
二维数组 a[m][n]	a[i][j]的存储地址（按行存储）为：a+(i*n+j)*len a[i][j]的存储地址（按列存储）为：a+(j*m+i)*len

稀疏矩阵	示 意 图	要 点
上三角矩阵	上三角矩阵示意图 	在矩阵中下标分别为 i 和 j 的元素，对应的一维数组的下标计算公式为： (2n-i+1) × i/2+j
下三角矩阵	下三角矩阵示意图 	在矩阵中下标分别为 i 和 j 的元素，对应的一维数组的下标计算公式为： (i+1) × i/2+j

特殊矩阵	示 意 图	要　　点
对角矩阵	对角矩阵示意图 	矩阵中的非零元素都集中在以主对角线为中心的对称带状区域中。

稀疏矩阵的表示形式：三元组表（行号，列号，元素值），例：（1,1,5），（1,2,10）等。

三元组表的顺序存储结构称为三元组顺序表，常用的三元组表的链式存储结构是十字链表。

三、树（5 星）

1、树与二叉树的概念：

结点（圆形）、分支（结点下面的线）

结点的度：该节点的子树数目；树的度：各结点度的最大值

叶子结点：度为 0 的结点-无分支；分支结点（非终端结点）：度不为 0 的结点-有分支

内部结点：除根以外的分支结点

层次：根为第 1 层，以此类推；树的高度：一棵树的最大层次数-总层数

二叉树：度最大为 2，左右区分很重要，可以为空，可以是单结点数

满二叉树：任意结点，度为 0 或 2（除最后一层以外，所有结点的度都为 2）

完全二叉树：最多只有下面 2 层结点的度可以小于 2，并且最后一层结点集中在该层左侧的若干位置（编号连续）

2、树的存储：【顺序存储】数组；【链式存储】十字链表法

3、二叉树的重要特性：

（1）在二叉树的第 i 层上最多有 2^{i-1} 个结点（i≥1）；

（2）深度为 k 的二叉树最多有 2^k-1 个结点（k≥1）；

（3）对任何一棵二叉树，如果其叶子结点数为 n_0，度为 2 的结点数为 n_2，则 $n_0=n_2+1$；

（4）如果对一棵有 n 个结点的完全二叉树的结点按层序编号（从第 1 层到$\lfloor log_2 n \rfloor + 1$层，每层从左到右），则对任一结点 i（1≤i≤n），有：

如果 i=1，则结点 i 无父结点，是二叉树的根；如果 i>1，则父结点是⌊i/2⌋；

如果 2i>n，则结点 i 为叶子结点，无左子结点；否则，其左子结点是结点 2i；

如果 2i+1>n，则结点 i 无右子叶点，否则，其右子结点是结点 2i+1。

（5）具有 n 个结点的二叉树形态数 $A[N]=\sum_{M=0}^{N-1}(A[M]*A[N-M-1])$

4、特殊的树

平衡二叉树：树中任一结点的左右子树高度之差不超过 1。

查找二叉树：又称之为排序二叉树。任一结点的权值，大于其左孩子结点，小于其右孩子结点。

最优二叉树：又称为哈弗曼树，它是一类带权路径长度最短的树。

路径是从树中一个结点到另一个结点之间的通路，路径上的分支数目称为路径长度。

树的路径长度是从树根到每一个叶子之间的路径长度之和。结点的带权路径长度为从该结点到树根之间的路径长度与该结点权值的乘积。

树的带权路径长度为树中所有叶子结点的带权路径长度之和。

哈弗曼树的构造：（1）根据给定的权值集合，找出最小的两个权值，构造一棵子树将这两个权值作为其孩子结点，二者权值之和作为根结点；（2）在原集合中删除这两个结点的权值，并引入根节点的权值；（3）重复步骤（1）和步骤（2），直到原权值集合为空。——自己操作熟悉

哈夫曼编码：根据哈夫曼树进行边长编码，编码长度与路径长度相关，左侧分支编码为 0（或 1），右侧分支编码为 1（或 0），从根结点到对应叶子结点所有路径分支上的编码记录下来，即为该叶子结点的编码。

5、树的遍历操作：遍历是按某种策略访问树中的每个结点，且仅访问一次的过程。

前序遍历：又称为先序遍历，按根→左→右的顺序进行遍历。

后序遍历：按左→右→根的顺序进行遍历。

中序遍历：按左→根→右的顺序进行遍历。

层次遍历：按层次顺序进行遍历。

四、图（5 星）

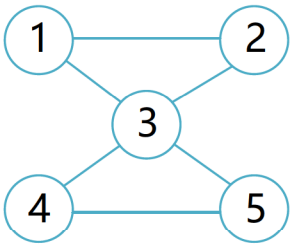
1、完全图

在无向图中，若每对顶点之间都有一条边相连，则称该图为完全图（complete graph）。

在有向图中，若每对顶点之间都有二条有向边相互连接，则称该图为完全图。

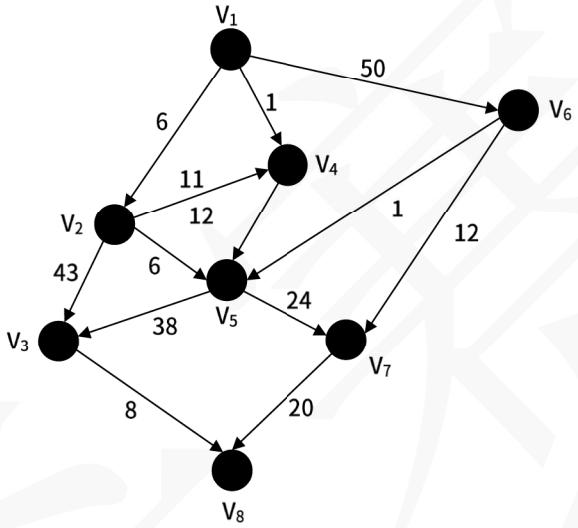
2、图的邻接矩阵表示：用一个 n 阶方阵 R 来存放图中各结点的关联信息，其矩阵元素 R_{ij} 定义为：

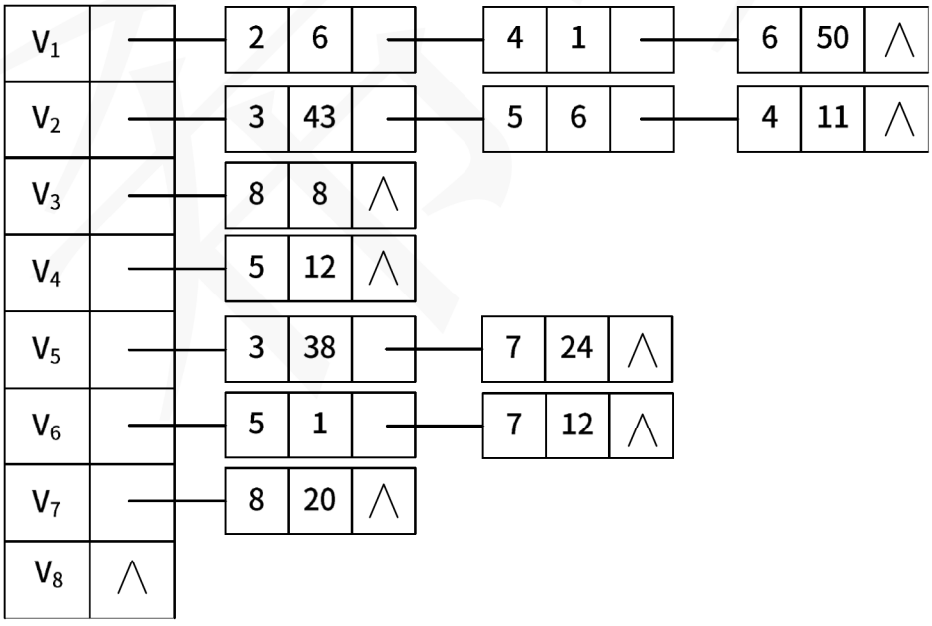
$$R_{ij}=\begin{cases}1 & \text{若顶点 i 到顶点 j 有邻接边}\\ 0 & \text{若顶点 i 到顶点 j 无邻接边}\end{cases}$$



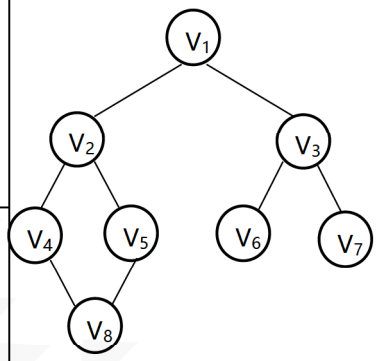
$$R_1 = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

3、图的邻接表表示：首先把每个顶点的邻接顶点用链表示出来，然后用一个一维数组来顺序存储上面每个链表的头指针。





4、图的遍历：

<table>
<tr><th>遍历方法</th><th>说明</th><th>示例</th><th>图例</th></tr>
<tr><td>深度优先
（垂直优先）</td><td>1.首先访问出发顶点 V；
2.依次从 V 出发搜索 V 的任意一个邻接点 W；
3.若 W 未访问过，则从该点出发继续深度优先遍历；
它类似于树的前序遍历。</td><td>V_1，V_2，V_3，V_4，V_5，V_6，V_7</td><td rowspan="2"> 
</td></tr>
<tr><td>广度优先
（水平优先）
【结合队列】</td><td>1.首先访问出发顶点 V；
2.然后访问与顶点 V 邻接的全部未访问顶点 W、X、Y...；
3.然后再依次访问 W、X、Y...邻接的未访问的顶点。</td><td>V_1，V_2，V_3，V_4，V_5，V_6，V_7，V_8</td></tr>
<tr><td colspan="4">注：遍历过程的时间复杂度只与存储结构有关系，无论是深度优先还是广度优先遍历，邻接矩阵存储时它的时间复杂度为 O（n^2），邻接表存储时它的时间复杂度为 O（n+e）其中 n 为邻接顶点规模数，e 为边的规模数。</td></tr>
</table>

5、图的拓扑排序：拓扑排序是将 AOV 网中的所有顶点排成一个线性序列的过程，并且该序列满足：若在 AOV 网点中从顶点 Vi 到 Vj 有一条路径，则在该线性序列中，顶点 Vi 必然在顶点 Vj 之前。

6、最短路径问题：迪杰斯特拉算法，每一次只考虑从上一层节点到当前结点的最短路径。

第 8 章 知识产权与标准化

一、保护对象和保护期限（5 星）

1、保护范围与对象

法律法规名称	保护对象及范围	注意事项
软件著作权法 计算机软件保护条例	软件著作权 软件作品	1、不需要申请，作品完成即开始保护 2、登记制度便于举证（中国版权保护中心登记）
专利法	专利权	需要申请，专利权有效期是从申请日开始计算
商标法	商标权	1、需要申请，核准之日起商标受保护 2、无特殊含义的行政名不能作为商标注册，比如：湖南
反不正当竞争法	商业秘密权	1、商业秘密包括技术与经营两个方面 2、必须有保密措施才能认定商业秘密

软件著作权人享有下列各项权利：

（1）发表权，即决定软件是否公之于众的权利；

（2）署名权，即表明开发者身份，在软件上署名的权利；

（3）修改权，即对软件进行增补、删节，或者改变指令、语句顺序的权利；

（4）复制权，即将软件制作一份或者多份的权利；

（5）发行权，即以出售或者赠与方式向公众提供软件的原件或者复制件的权利；

（6）出租权，即有偿许可他人临时使用软件的权利，但是软件不是出租的主要标的的除外；

（7）信息网络传播权，即以有线或者无线方式向公众提供软件，使公众可以在其个人选定的时间和地点获得软件的权利；

（8）翻译权，即将原软件从一种自然语言文字转换成另一种自然语言文字的权利；【在 IT 行业中，翻译权通常指程序设计语言的转换。建议以程序语言理解。】

（9）应当由软件著作权人享有的其他权利。

软件著作权人可以许可他人行使其软件著作权，并有权获得报酬。

软件著作权人可以全部或者部分转让其软件著作权，并有权获得报酬。

按照被许可使用权的排他性强弱不同，可以将使用许可分为以下三种：

独占使用许可-仅 1 个授权对象可用，著作权人不可用

排他使用许可-仅 1 个授权对象和著作权人可用

普通使用许可-多个授权对象和著作权人可用

2、保护期限

（1）著作权类

作品完成即开始保护。普通著作权中署名权、修改权、保护作品完整权以及软件著作权中署名权、修改权是永久保护没有限制的。其他权利保护期限为作者终身及其死后 50 年。【单位作品只有 50 年期限】

注：著作权除署名权等人身权利以外，可以被继承。

（2）商标权可续注，保护期限可延长。

（3）商业秘密权保护期限不确定，一旦泄密则不再保护。

（4）专利权类

发明专利保护期限 20 年，实用新型和外观设计专利权保护期限 10 年。

二、知识产权人确定（5 星）

情况说明		判断说明	归属
作品	职务作品	利用单位的物质技术条件进行创作，并由单位承担责任的	除署名权外其他著作权归单位
		有合同约定，其著作权属于单位	除署名权外其他著作权归单位
		其他	作者拥有著作权，单位有权在业务范围内优先使用
软件	职务作品	属于本职工作中明确规定的开发目标	单位享有著作权
		属于从事本职工作活动的结果	单位享有著作权
		使用了单位资金、专用设备、未公开的信息等物质、技术条件，并由单位或组织承担责任的软件	单位享有著作权
专利权	职务作品	本职工作中作出的发明创造	单位享有专利
		履行本单位交付的本职工作之外的任务所作出的发明创造	单位享有专利
		离职、退休或调动工作后 1 年内，与原单位工作相关	单位享有专利

情况说明		判断说明	归属
作品软件	委托创作	有合同约定，著作权归委托方	委托方
		合同中未约定著作权归属	创作方
	合作开发	只进行组织、提供咨询意见、物质条件或者进行其他辅助工作	不享有著作权
		共同创作的	共同享有，按人头比例。 成果可分割的，可分开申请。
商标		谁先申请谁拥有（除知名商标的非法抢注） 同时申请，则根据谁先使用（需提供证据） 无法提供证据，协商归属，无效时使用抽签（但不可不确定）	
专利		谁先申请谁拥有 同时申请则协商归属，协商不成则同时驳回双方的专利申请	

三、侵权判断（3 星）

1、中国公民、法人或者其他组织的作品，不论是否发表，都享有著作权。

2、开发软件所用的思想、处理过程、操作方法或者数学概念不受保护

3、著作权法不适用于下列情形：

（1）法律、法规，国家机关的决议、决定、命令和其他具有立法、行政、司法性质的文件，及其官方正式译文；

（2）时事新闻；

（3）历法、通用数表、通用表格和公式。

不侵权	侵权
个人学习、研究或者欣赏； 适当引用； 公开演讲内容 用于教学或科学研究 复制馆藏作品； 免费表演他人作品； 室外公共场所艺术品临摹、绘画、摄影、录像； 将汉语作品译成少数民族语言作品或盲文出版。	未经许可，发表他人作品； 未经合作作者许可，将与他人合作创作的作品当作自己单独创作的作品发表的； 未参加创作，在他人作品署名； 歪曲、篡改他人作品的； 剽窃他人作品的； 使用他人作品，未付报酬； 未经出版者许可，使用其出版的图书、期刊的版式设计的。

注：合理使用不需要通知作者也不需要付报酬，此时原作品仍然受著作权法保护的。

四、标准化（1 星）

1、标准化分类

【标准化基础知识 – 标准的分类】

（1）国际标准：ISO、IEC 等国际标准化组织

（2）国家标准：GB—中国、ANSI—美国、BS—英国、JIS—日本

（3）区域标准：又称为地区标准，如 PASC—太平洋地区标准会议、CEN—欧洲标准委员会、ASAC—亚洲标准咨询委员会、ARSO—非洲地区标准化组织

（4）行业标准：GJB—中国军用标准、MIT-S—美国军用标准、IEEE—美国电气电子工程 师协会

（5）地方标准：国家的地方一级行政机构制订的标准

（6）企业标准

（7）项目规范

2、标准化代号

【标准化基础知识 – 标准的编号】

（1）国际、国外标准代号：标准代号+专业类号+顺序号+年代号

（2）我国国家标准代号：强制性标准代号为 GB、推荐性标准代号为 GB/T、指导性标准代号为 GB/Z、实物标准代号 GSB

（3）行业标准代号：由汉语拼音大写字母组成（如电子行业为 SJ）

（4）地方标准代号：由 DB 加上省级行政区代码的前两位

（5）企业标准代号：由 Q 加上企业代号组成

更多备考资料和学习福利，可扫码添加希赛萧萧老师，申请入群