UML课本知识点总结 by hrmm

（标注p+数字的部分是需要看书上图的，部分是没啥用给个链接自己看的）

设计强调的是满足需求的概念上的解决方案

实现则表达了真实完整的设计

面向对象分析OOA强调在问题领域内发现与描述对象

面向对象设计OOD强调定义软件对象以及他们如何协作

解释一下OO OOA OOD与OOP

OO（面向对象）：

面向对象（OO）指的是基于“对象”概念的编程范式，对象可以包含数据（属性）和代码（方法）。对象是类的实例，类定义了对象的结构（属性）和行为（方法）。

OOA（面向对象分析）：

面向对象分析（OOA）是检查问题领域、识别对象（实体）并定义它们的关系和行为的过程。它专注于理解将要开发的系统的需求和约束。

OOP（面向对象编程）：

面向对象编程（OOP）是紧随OOA之后的实施阶段。它涉及在支持对象和类的编程语言中编写代码。OOP允许编写模块化和可重用的代码，以及更准确地建模复杂的现实系统。

OOD（面向对象设计）：

面向对象设计（OOD）是介于OOA和OOP之间的中间步骤。它涉及根据OOA的结果设计软件系统的架构。这包括定义类、它们的属性、方法和关系，同时考虑封装、继承和多态等原则。

总之，OOA专注于理解问题领域，OOD将分析转化为软件实现的设计，而OOP使用对象和类将这一设计转化为可执行代码。

解释一下RUP Rational unified process

RUP（rational统一过程）强调使用迭代和增量开发方法，旨在通过一系列预定义的阶段来管理软件项目的各个方面，包括需求、设计、编码和测试。

RUP的主要特点包括：

迭代性：项目被分成多个迭代周期（称为“阶段”），每个阶段都专注于特定的任务，如需求收集、设计、实现和测试。

增量性：在每个迭代周期内，开发团队会完成一部分工作，并逐步向最终产品靠近。

可视化：RUP使用各种图表和模型来描述软件项目的不同方面，如用例图、类图和序列图。

文档化：RUP强调文档的重要性，要求团队在项目的各个阶段都生成相应的文档，以确保沟通和理解的一致性。

灵活性：RUP允许项目团队根据项目的具体需求调整过程的某些方面，而不是强制使用一个固定的流程。

RUP通常与UML（统一建模语言）结合使用，因为UML提供了一套标准化的图形表示法，可以帮助团队更好地理解和沟通软件的设计和结构。

尽管RUP在2000年代初期非常流行，但随着敏捷开发方法（如Scrum和XP）的兴起，它的使用逐渐减少。敏捷方法强调快速响应变化、短周期的迭代和高度灵活的团队工作，与RUP的某些方面形成对比。然而，RUP的一些理念和实践仍然被现代软件开发方法所借鉴。

面向对象的分析结果可以表示为领域模型描述的是真实类p6

顺序图是描述协作的p7

设计类图表示类的属性与方法描述软件类p7

概念类：rational unified process RUP

up核心思想：短时间定量迭代进化和可适应性开发

up阶段：初始inception细化elaboration构造construction移交transition

需求就是系统必须提供的能力和必须遵从的条件

需求按照FURPS+模型分类

furps+模型：功能性functional可用性usablity可靠性reliablity性能performance可支持性supportablity+实现接口操作包装授权

up需求制品：用例模型 supplementary补充性说明词汇表glossary设想vision业务规则

用例是文本形式的情节描述p46

参与者是具有行为的食物

场景是参与者和系统之间的一系列特定的活动和交互也称为用例实例

用例是一组成功和失败的场景集合用来描述参与者如何使用系统来实现目标

用例是文本文档而不是图形 用例建模主要是编写文本的活动而不是制图p50-p67

uml用例图p68

uml领域模型p100需要真是类与属性 不是软件制品 有类有属性还有方法的是软件类 属性应为数字与文本等非实体的

领域模型构建：寻找概念类 绘制uml图 添加关联与属性

关联在uml内用实线表示 可零～多对零～多（多重性）

关联是类的实例之间的关系p113

系统顺序图SSD是用来阐述与所讨论系统相关的输入与输出事件p128

对用例的主成功场景，以及频繁发生的或者复杂的替代场景绘制SSD

SSD与SD的区别 SSD主要注重操作与行为 SD主要注重消息

操作契约是对系统行为进行更加详细和精确的描述p134

操作契约有操作 交叉引用 前置条件与后置条件

uml包图p147p148

逻辑架构是软件类的宏观组织结构

层是对类 包或子系统的分组 具有对系统主要方面加以内聚的职责

oo系统通常包含的层有：用户界面 应用逻辑和领域对象 技术服务

uml交互图描述对象间通过信息的交互 用于动态对象建模

交互图有两类 顺序图和通信图p162

顺序图与通信图优缺点p163大致就是顺序图画图的时候需要不断在右端加新对象比较麻烦，通信图随便加比较灵活，顺序图可以表示消息的顺序二通信图对顺序表示不是很清晰

p165交互中参与者的生命线框图

顺序图的表示方法与通信图基本表示方法p166~179

方框是执行规格条 表示控制期

uml使用类图表示类，接口及其关联 用于静态对象建模p181

利用属性文本与关联线来表示类的属性 关联线：→这中箭头

领域模型中关联用的是实直线类图中用的是导航箭头如上

准则规定对数据类型使用属性文本表示，其余对象用关联线

数据类型指的是其唯一标识不具有意义的对象如布尔 时间 数字 字符 字符串 日期 地址等

关联端点也可以附加多重性值 前文有提到过

关键字表p187

泛化用由子类到超类的实线和空心三角箭头表示

可以这样解释泛化 在领域模型中表示超集到子集的继承 在类图中则是超类到子类的继承

依赖线是虚线箭头线表示----→类似这个

依赖可以视为耦合

依赖有以下几种作用：拥有提供者类型的属性 向提供者发送信息 接收提供者类型的参数 提供者是超类或接口

若有关联则不需要依赖

依赖标签有call和create

接口实现：插座线表示法 虚线空心三角形线 依赖线 棒棒糖表示法p191（一堆不知道有啥用的东西）

聚合：作者都说没用

组合：组成集合 用带有实心菱形箭头的关联线表示 箭头指向组成类

组合代表了拥有者与部分的关系 组成类为一个整体 组合的另一段则是整体的一个部分 组成类拥有其

约束p192（看不出来有什么用）

限定关联 暗示了基于键对事物进行查找 键就是限定符的内容 类似于图上多一个小方块 然后关联线从小方块上引出 类似于限定tag的东西p193

关联类p193（也是不知道有什么用的）

单实例类 右上角角标1 表示仅能创建一个实例

模板类主动类p194

重点的GRASP（前面写了这么多感觉脑子里一点痕迹都没留下来QAQ）

OO设计原则：GRASP和GoF（四人帮）设计模式（虽然不知道为啥这么叫 书上是这么写的）

OO设计建模基于职责驱动设计RDD所代表的内在含义是考虑怎样给协作中的对象分配职责

职责即其所作所为的抽象 就是某个对象类负责创建～ 负责认知～

GRASP对基本的职责分配原则进行命名与描述 GRASP是使用职责进行OO设计的学习工具

职责与方法相关

模式是对问题 解决方案和命名进行描述使其系统化的原则与习惯用法就是模式 模式具有名称 是对问题与解决方案的已命名描述（作者感觉是车轱辘话讲多了）（模式大致就是对一类问题给的某些一般性的解决方案）

p203~210应用模式的实例 建议多看看

GRASP九种模式：创建者 creator 控制器 controller 纯虚构 pure fabrication

信息专家 information expert 高内聚 high cohesion 间接性 indirection

低耦合 low coupling 多态性 polomorphism 防止变异 protected variations

主要复习 控制器 高内聚 低耦合

创建者模式：解决谁来创建实例的问题 应当寻找聚合或包含该实例的类来创建实例 支持低耦合

信息专家模式：解决给对象分配职责的基本原则 专业的事情交给专业的人去做 将职责分配给有实现这个职责所必须的信息的专家 支持低耦合 高内聚

低耦合：解决如何降低依赖性减少变化带来的影响 提高重用性 就是去掉那些高度依赖于其他类的类 分配其职责降低耦合性 高耦合不是问题 问题是与不稳定元素的高耦合带来的不稳定性与不确定性

控制器：解决在UI层之上首先接受和协调控制系统操作的第一个对象是什么（层次结构在前文有提到）系统操作主要指系统的输入事件类似于外部给系统的指令 通过将职责分配给代表整个系统 根对象 与子系统或者代表用例场景，在该场景中发生系统事件的类来解决 （同一用例场景的所有系统事件均使用相同的控制器类）

高内聚：解决如何保持对象是有重点可理解可管理并且能支持低耦合 内聚是对职责相关性和集中度的度量 内聚性低体现在某些类要做许多互不相关的工作 低内聚是由职责分配不合理导致的

模式并不难理解 主要的内容我都给你们归纳了 重要的是要看例子 根据具体事例学会如何应用该模式 或者是嘤嘤怪你是改进设计图

用例实现（感觉也是没啥用的东西）p232

可见性 属性可见性 参数可见性 局部可见性 全局可见性

若对象a能向对象b发送信息 则对于a而言b必须是可见的

若b是a的属性 则存在由a到b的属性可见性

若b作为参数传递给a的方法 则存在由a到b的参数可见性

若b被声明为a的方法内的局部变量 则存在由a到b的局部可见性

当b对于a是全局时 则存在由a到b的全局可见性

继续上面的GRASP模式

多态：解决如何处理基于类型的选择 如何创建可插拔的软件构件 当相关选择或行为随类型不同时，使用多态操作为变化的行为类型分配职责（简单来说就是根据不同类型的设备更改不同的职责）打个不恰当的比方 假如你有很多个mm 他们的共性就是都love you 然后根据共性执行一下多态化操作 针对每个人的性格延伸出不一样的类 类的方法就是共性

纯虚构：解决不违背高内聚低耦合的前提 同时又不符合专家模式该如何选择对象承担职责的问题 可以虚构一种类 然后正常画图即可 支持高内聚 增加复用性

间接性：解决避免两个或多个事物直接耦合来分配职责 或者使对象解耦合来支持低耦合并提高复用性的问题 将职责分配给中介对象 将其作为其他构件或服务之间的媒介

防止变异：解决如何设计对象与系统 使其内部变化或者不稳定性不会对其他元素产生不良影响的问题 可以识别预计变化或者不稳定的地方 分配职责用以在这些变化之外创建稳定接口

GoF设计模式（更加细化的设计准则）

重点是适配器 策略 工厂 单例

适配器：解决接口不相容 或者如何为具有不同接口的类似构件提供稳定接口的问题 可以通过中介器适配对象 将原有接口转换为其他接口

工厂：分为具体工厂和抽象工厂 具体工厂不是GoF设计模式但是是GoF设计模式里面抽象工厂的简化版（md搞出这么多东西出来干嘛）

工厂：解决当为了改良内聚而分离创建职责时应该由谁来负责创建对象的问题 可以创建称为工厂的纯虚构对象来处理创建职责（有点类似GRASP的纯虚构 一种是改良现有设计 一种是在原有规则不适合的情况下的选择）

单实例类：解决单实例类的对象全局可见性和单点访问 可以对类定义静态方法以返回单实例 类似私有变量通过定义静态方法返回变量值

策略：解决设计变化但相关的算法或政策 可以在单独的类中分别定义每种算法或政策策略 差不多就是多态+细化方法和类 （工厂+策略p326）

组合：解决处理一组对象或具有组合结构的对象 可以定义组合和原子对象的类 使他们实现相同的接口

外观：解决对一组完全不同的实现或接口需要公共统一接口的问题 可以使用外观对象封装子系统 外观对象提供唯一和统一的接口 并负责与子系统构件进行协作

观察者（发布-订阅）：p337（懒得整理了 没讲过 开摆了）

uml活动图 表示一个过程中多个顺序活动和并行活动p347

数据流图DFDp348

耙子符号 决策符号 合并符号p349

uml状态机图 描述事物的事件和状态的表示法p352

抛出异常p428

异步消息使用刺形箭头 就是→的直线穿过了折线的箭头（很抽象对吧）p429

继续GoF设计模式（可以看出来编写这本书的人有多ex了 全tm都是混乱的 有几章内容前面都讲过了 搬到后面重新讲一遍 你人还怪好的嘞 给学生复习 混在一坨跟狗屎一样）

代理：p432

抽象工厂：解决如何创造实现相同接口的一族相关的类 可以定义一个工厂接口 为每一族要创建的事物定义一个具体工厂类

部署图p468

构件图p469

（知识点差不多就到这里了 啃完了这一坨屎 肝了五个多小时 搞到半夜三点半😭😭😭）