UML notes 01 Intro of UML

collected by wxb

1 Object-oriented Tech 面向对象思想

Q1: Software Engineering 软件工程的定义

- Engineering 什么是工程?

  > - acquiring and applying scientific, mathematical, economic, social, and practical knowledge
  >
  >   获取和应用科学、数学、经济、社会和实践知识
  > - To design and build structures, machines, devices, systems, materials and processes
  >
  >   设计和建造结构、机器、设备、系统、材料和工艺
  > - That safely realize solutions to the needs of society.
  >
  >   安全地实现满足社会需求的解决方案

- Engineering 的主要分支
  - Mechanical engineering 机械工业
  - Electrical engineering 电气工程
  - Civil engineering 土木工程
  - Chemical engineering 化学工程
  - Management engineering 管理工程
- Software Engineering

  Design, implement, modify software => Higher quality, More affordable, Maintainable, Faster to build

  设计、实现、修改软件 => 质量更高，成本更低，可维护性更强，更易构建

  目的：在时间、开销有限的条件下，开发高质量、满足用户需求的软件

Q2: Software Crisis 软件危机

- **Characteristics of Software crisis 软件危机的特征**
  - low efficiency 低效率
  - low quality 低质量
  - hard to maintain 难维护
  - hard to replant 难补种
  - over-high development-cost 开发成本过高
  - long development cycle 开发周期过长

Q3: Characteristics of Software 软件的特征

- **Characteristics of Software 软件的特征**
  - Form: unseen, invisible, untouchable, hard to decide whether good or not

    形式：不可见，透明，不可触，难说好坏
  - production mode: logical product, related to the designers' thought mode & Intelligence, communication matters

    生产方式：逻辑产品，与设计师的思维方式和智力有关。沟通很重要
  - product requirement: cannot have any error. need higher quality insurance systems

    产品要求：不能有任何错误，需要更高质量的保险系统
  - maintenance mode: hard, no spare parts. Bug patch will induce new Bug.

    维护方式：没有备用件，修复bug会导致新bug；维护很艰辛
- **Inherent complexity of software 软件的固有复杂性**
  - Simple hardware and complex software

    硬件简单，但软件复杂
  - Intellectual activity, hard to describe

    软件是逻辑、智力活动，难描述
  - Interaction, geometrically increasing

    交互几何式增长
  - Change, change, change, ……

    需求持续变化，软件生命周期比硬件长，影响了软件地需求
- General process of software development 软件开发的大致流程
  - Technical-Level: Process divide and control 技术层面：过程分解和控制
    - software requirement 软件需求
    - system-level design 系统层面的设计
    - sub-system-level design 子系统设计
    - detailed design 细节设计
    - coding 代码实现
    - test 测试
    - delivery 打包上传
    - maintenance 维护
    - re-engineering 再加工
  - Management-Level: Source management 管理层面：源控制
    - hardware resource 硬件资源
    - software resource 软件资源
    - human resource 人力资源
- **Development method 开发方法**
  - Waterfall model 瀑布模型
  - Rapid Prototype model 快速原型模型/敏捷
  - Spiral model 螺旋模型
  - Evolutionary model 进化模型
  - Incremental model 增量模型
  - Fountain model 喷泉模型

Q4: Control the complexity 控制复杂性

- Decomposition 解构
- Abstraction 抽象
  - Process abstraction 流程抽象
  - data abstraction 数据抽象
- Modularity 模块化
  - high inner cohesion 高内聚
  - low coupling 低耦合
- Encapsulation 封装
- Modeling 建模：简化现实问题

  软件要求功能正确，并且开发尽可能少的软件（复用）

  Function of Modeling 建模的功能：
  - Visualize 可视化
  - Specify　明确
  - Guide　　指导
  - Document 文档
  - Analyze　分析
  - Divide-and-Conquer 分而治之

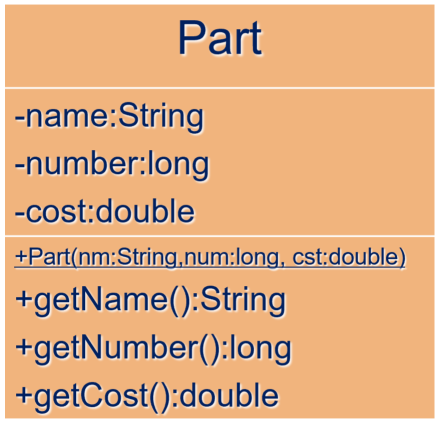
Q5: Object-oriented Tech 面向对象思想

decomposition, abstraction，modularization，encapsulation.

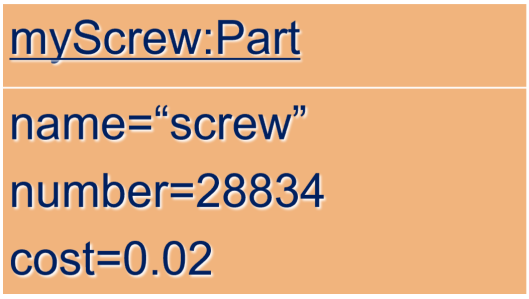
- Advantage of OOT 优势
  - stable 稳定
    - 小的需求不会引起系统结构的变更
  - easily understood 易理解
  - adaptable 适应性强
    - 更好地适应用户需求变更
    - 更能适应大型系统建构
  - reliable 可靠
    - The same concept and representation used for analysis and design

      分析和设计时使用的概念和表示相同
- Basic concept of OOT 基本概念
  - Class 类
    - 一组拥有相同方法和属性的对象的集合
    - Class is static, objects are dynamic

      类是静态的，对象是动态的

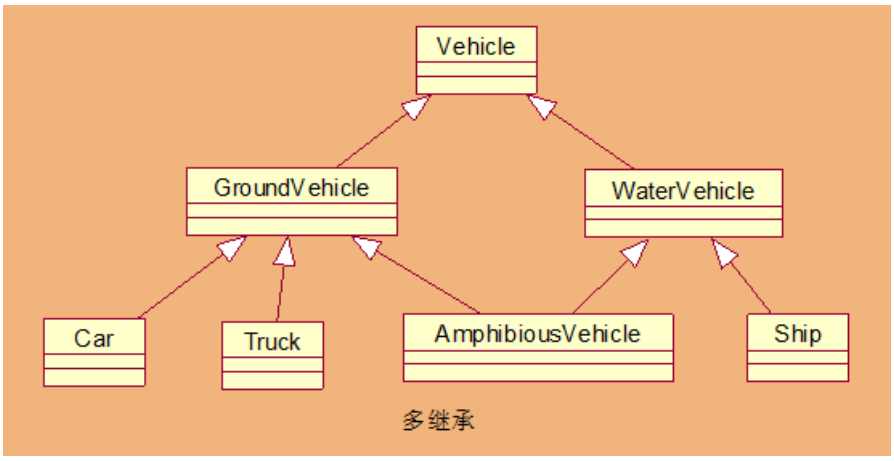


- Object & Instance
  - 对象由一系列属性和方法组成
  - 类的一个实例就是一个对象



- Encapsulation 封装
  - interface & implementation 接口和实现
  - 防止用户改变内部细节，内部实现的改变也不会影响客户端
- Inheritance 继承
  - children class 子类/parent class 父类
  - special class/general class
  - derivation class 衍生类/basic class 基类

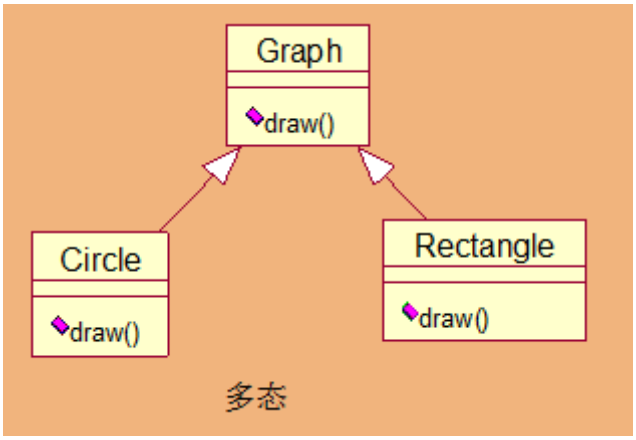
    空心箭头，从 子类/衍生类 指向 父类/基类



  - Override & Overload

```java
@Override
public class A {
    String name;
    public String getInfo(){
        return name;
    }
}
public class B extends A{
    String address;
    public Stirng getInfo() {
        return name + address;
    }
}
```

- Polymorphism 多态
  - Override & Dynamic binding
  - 让系统适应性更强



- Message 通信
  - 对象间的通信，不同的调用函数的方式
  - synchronous message and synchromous message 同步和异步通信

Conclusion

- software crisis 软件危机
- crisis origin 危机来源

  intrinsic, complexity of software 软件固有的复杂性.....
- solution idea

  control complexity 控制复杂性：解构 抽象 模块化 封装
- solution method: OOT 解决方法就是面向对象

2 Intro of UML

2.1 Definition of UML 定义

- a **standard language** for writing software blueprints 绘制软件蓝图的**标准语言**
- used to: **Visualize, Specify, Construct, Document** the artifactes of a software-intensive system

  用于：**可视化、明确、构建、记录**软件密集型系统的各个方面（模型、代码、测试样例……）

2.2 UML is a Language 特征

- vocabulary +rules 词汇+语法
- modeling language 建模语言
- can: how to create and read well-formed models

  能告诉你怎么建模
- cannot: what models should create and when to create

  不能告诉你要建啥模和啥时候整

Visualizing

如果没有UML，可能会导致以下三个问题

- error-prone communication 沟通容易出错
- hard to understand with only texutual words 纯文本难理解

  => 一图胜千言
- conceptual knowledge lost 概念性知识丢失

  => 好记性不如烂笔头

Specifying

- **building precise, unambiguous and complete models**
- analysis => design => implementation

  分析 => 设计 => 实现

Constructing

- not a visual programming language 不是一种可视化编程语言
- forward engineering 正向工程：将UML描述的模型转化为编程语言描述
- reverse engineering 逆向工程 也是可以的
- Direct execution of models, the simulation of systems, the instrumentation of running systems

  模型的直接执行，系统仿真，运行系统的检测

Documenting

requirements, architecture, design, source code, project plans, test case, prototypes, release ...

需求，架构，设计，源码，项目计划，测试用例，原型，发布版本……

Usage

- software-intensive systems 软件密集型系统

  企业信息系统、金融服务、国防军事、零售、分布式网络服务……
- non-software systems 非软件系统

  法律系统中的工作流、硬件设计、系统机制……

历史：略

Charateristics 特征

- uniform-standard 统一标准
- Object-oriented 面向对象
- visualization & strong representation 可视化 + 强表达力
- independent of modeling process 独立于建模过程
- easy to understand 好理解

  explicit concept, concise modeling, clear structure

  概念明确，建模简洁，结构清晰

2.3 UML的一些介绍

things in UML

structural things 结构性事物

- UML 模型的名词，是模型的静态部分，表示概念/物理元素
- 被称作 classifiers 分类器
- Class, Interface, use case, collaboration, active class, component, artifact, node

  类 接口 用例 合作 活动类 组件 部件 节点

behavioral things 行为性事物

- UML模型的动词，是模型的动态部分，表示跨越时空的行为
- 三种基本类型：interaction, state machine, activity 交互 状态机 活动
  - interaction: 带有特定目的的、对象或角色之间的信息交换；包括messages, actions, connectors
  - state machine: 表示一个对象或交互对外部事件响应的生命周期；包括states, transitions, events, activities
  - activity: 表示一个计算型进程的执行序列，每一步被称作一个action

grouping things 分组事物

- 与组件不同
- 是UML模型中有组织的部分
- package

annotational things 注释性事物

- UML模型中的注释部分

relationships in UML

dependency 依赖关系

一个元素的更改可能会影响另一个元素

association 联系

描述关系的集合

- aggregation
- composition: 组成

  例：一个老板雇佣多个员工，一对多关系

generalization

子元素由父元素特化而来

子->父 generalization

realization 实现关系

diagrams in UML

- class diagram ★ 类图：最常用
- component diagrams 组件图: 类图的变体
- object diagram 对象图: 展示了一系列对象以及他们的关系
- use case diagram 用例图：表示一系列用例，用例执行者以及它们之间的关系；在行为建模时非常重要

以上👆描述的都是系统的静态视图

- sequence diagram 时序图：交互图的一种，侧重于时序 time-ordering
- communication diagram 通讯图：交互图的一种，侧重于发送或接收消息的对象或角色的结构组织

- statte diagram 状态图：展现了状态机，由状态、转变、事件和活动组成，侧重于对象的事件顺序 event order
- activity diagram 活动图：将进程表示为一步步的控制流和数据流，功能建模时很重要

  以上👆描述的都是系统的动态试图

- artifact diagram
- package diagram
- timing diagram

rules in UML

name, scope, visibility, integrity, execution

common mechanism in UML

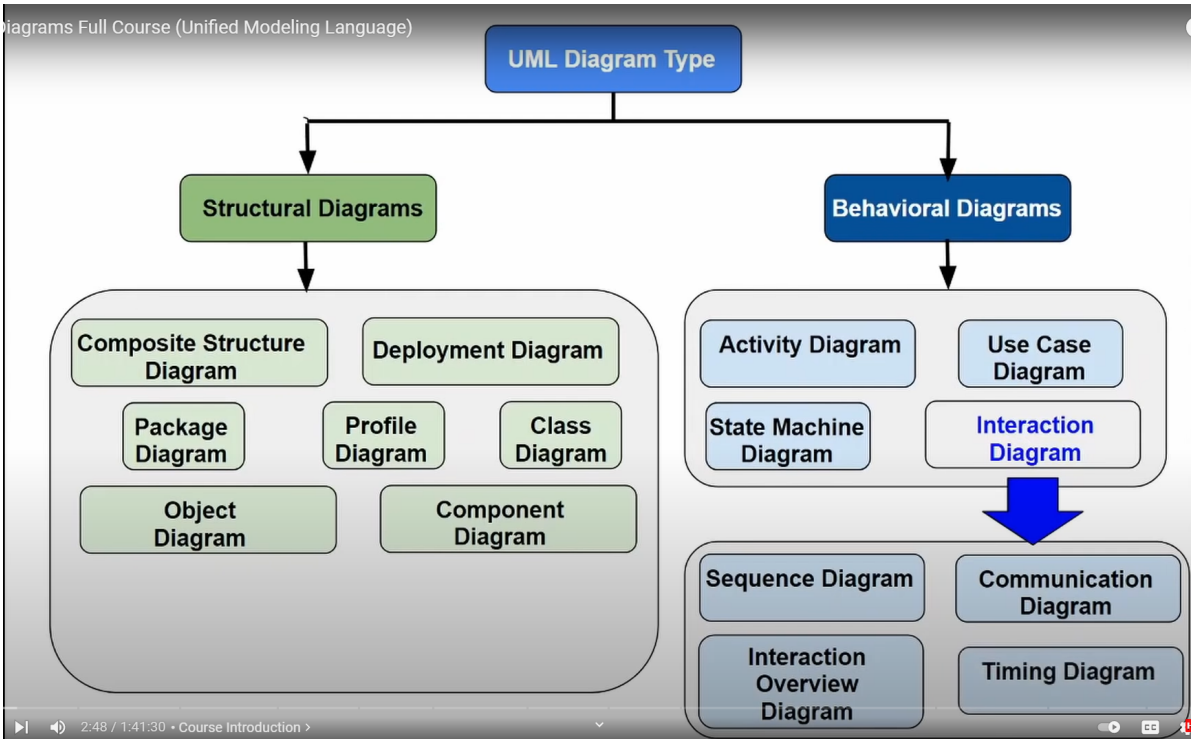
- specification: 全面的语义模板
- adornments: 包含细节
- divisions: 类和对象的分解；接口和实现的分离；类型和角色分类
- extensibility
  - stereotype 范式: 扩展UML词汇
  - tagged values: 扩展UML的属性
  - constraints: 扩展UML的语义

2.4 Architecture

- use case view: 终端用户看到的系统行为
- design view: 系统提供给终端用户的功能性需求（逻辑视图）
- process view: 各部分的控制流，包括并发和同步
- implementation view: 用于组装和发布物理系统的工件
- deployment view: 构成系统硬件的拓扑节点图

Apendix 附录

https://www.youtube.com/watch?v=WnMQ8HlmeXc&t=29s


iagrams Full Course (Unified Modeling Language)
UML Diagram Type
Structural Diagrams
Behavioral Diagrams
Composite Structure Diagram
Deployment Diagram
Package Diagram
Profile Diagram
Class Diagram
Object Diagram
Component Diagram
Activity Diagram
Use Case Diagram
State Machine Diagram
Interaction Diagram
Sequence Diagram
Communication Diagram
Interaction Overview Diagram
Timing Diagram
2:48 / 1:41:30 • Course Introduction ›