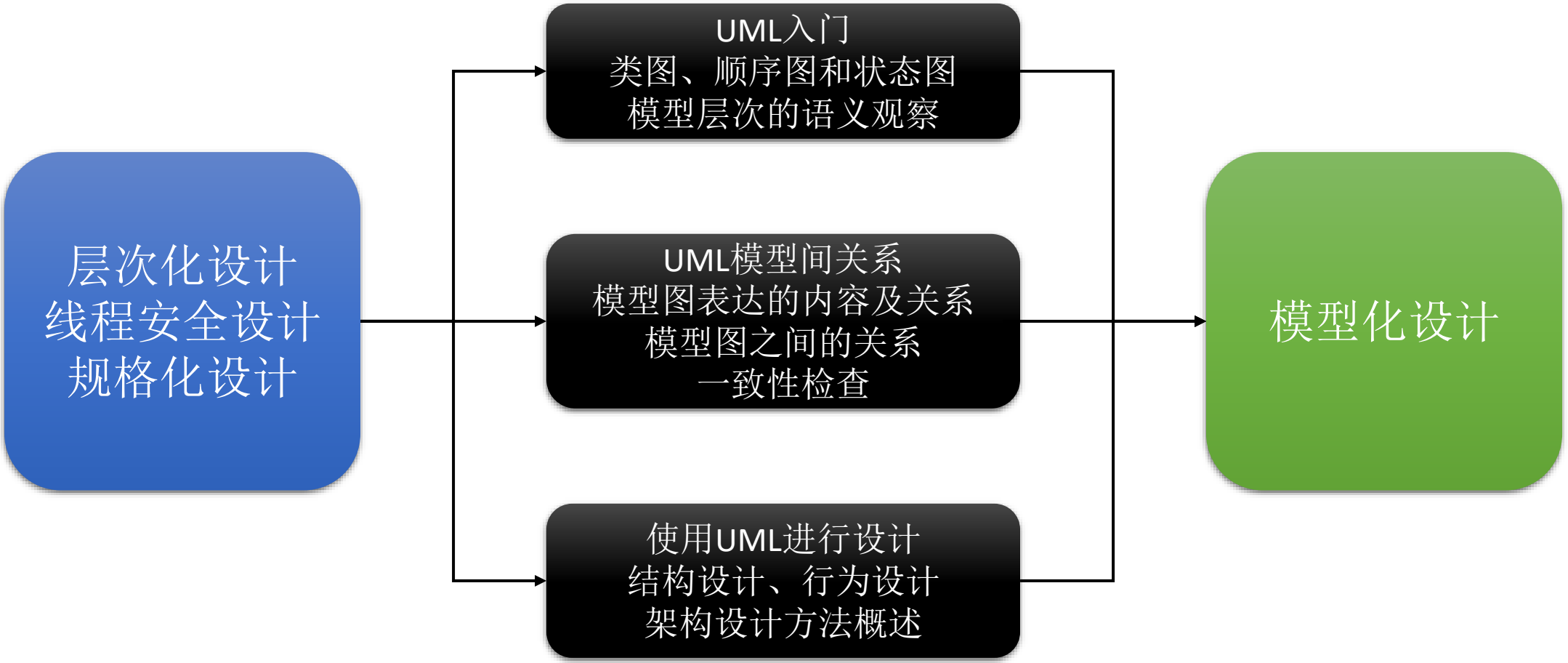
第十三讲：UML入门

OO2019课程组

计算机学院

第四单元内容总览


层次化设计
线程安全设计
规格化设计
UML入门
类图、顺序图和状态图
模型层次的语义观察
UML模型间关系
模型图表达的内容及关系
模型图之间的关系
一致性检查
使用UML进行设计
结构设计、行为设计
架构设计方法概述
模型化设计

摘要

- 为什么要学习一门新语言
- 如何理解一个程序系统
- 如何表示程序系统
- UML类图
  - 类图的几个关键标签类型
- UML顺序图
  - 顺序图的几个关键标签类型
- UML状态图
  - 状态图的几个关键标签类型
- 作业分析

为什么要学习一门新语言

- 语言是一套符号系统，用以表达人的思维
  - 词汇：表示方式和表达的含义
  - 词汇连接：表示方式和表达的含义
- 设计语言的目的
  - 可以更准确的表达思维
  - 可以更直观的表达思维
  - 可以更简单的表达思维
- 语言是沟通的桥梁
  - 表示者通过语言来表达自己的观点[思维]
  - 接受者通过语言来感知和理解对方的观点

学习新语言是为了：
更好的表达自己
更好匹配对方的偏好
更好融入共同体

OO其实是一套语言系统

- 语言一般都会提供两个基本构造机制
  - 词汇构造机制：用以针对一个存在(being)的特征来构造相关词汇，从而描述其相应特征
  - 连接构造机制：用以针对所构造的词汇，构造相应的连接(甚至是连接规则)来组合词汇，从而表达对存在的理解
- 面向对象本质上定义了一套抽象语言系统
  - 词汇：对象、属性、操作、活动、流程、状态、...
  - 连接规则：对象间连接、对象与数据间连接、对象与操作间连接、属性与操作间连接、属性与活动间连接、活动与流程间连接、操作与状态间连接、...

如何理解程序系统

- 程序系统是一种存在
  - 由无到有
  - 有生存状态
  - 能够对外界激励做出响应
  - 甚至可自行演化
- 程序系统是一种人造物
  - 可控
  - 可预测
  - 可配置

如何理解程序系统

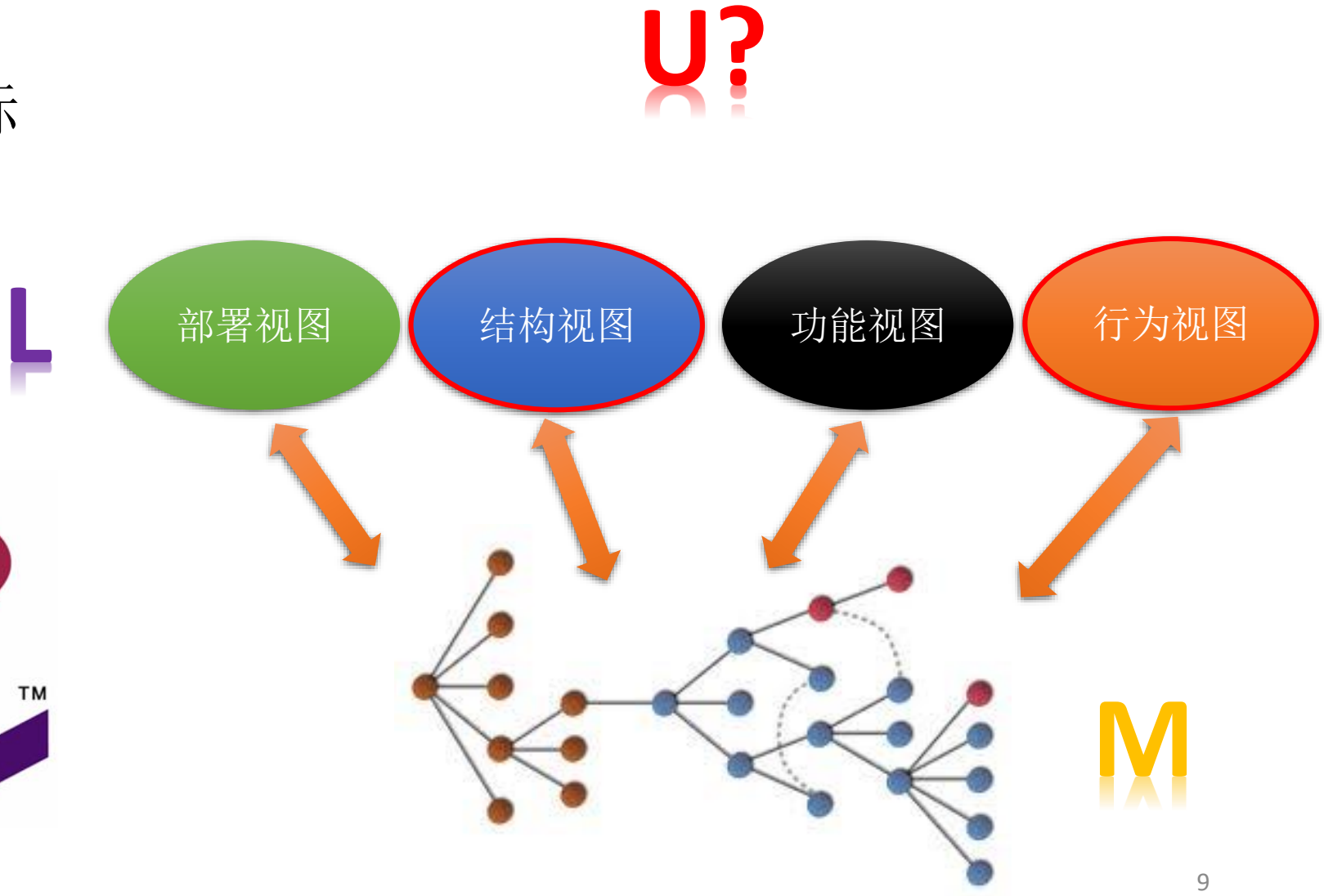
- 结构线
  - 需求层次的结构：数据及其关系、功能及其关系
  - 设计层次的结构：类、接口及相互关系，规格
  - 实现层次的结构：类、结构及相互关系，数据结构
- 行为线
  - 需求层次的行为：功能流程（用户与系统的交互流程）
  - 设计层次的行为：类之间的协作行为、类的状态行为
  - 实现层次的行为：类之间的协作行为、方法控制行为、算法流程

如何描述程序系统

- 自然语言、JML、Java都可以描述程序系统
  - 自然语言不提供专门描述结构和行为的成分
    - 需要大量的脑力来从中识别和理解结构与行为
  - JML可以描述结构和行为，整合的方式
    - 需要一定的脑力来分离其中的结构和行为
  - Java可以描述结构和行为，整合的方式
    - 需要相当的脑力从中分离结构和行为，并逐步建立抽象层次
- 我们希望有一种语言，直接提供针对性、分离的结构与行为描述手段，而且可以在后台把描述元素**整合**起来
- UML就是这样的语言

UML语言简介

- UML的语言设计目标
- UML的建模理念
- UML模型组成



UML语言设计目标

- 提供一种面向对象式的抽象又直观的描述逻辑
  - 抽象：把系统抽象表示为类和类之间的协同
  - 直观：通过可视化的模型图来描述和展示系统功能、结构和行为
- UML经过了二十多年的发展(UML 2.x)
  - 绘画式语言→仅用于人之间的交流
  - 描述性建模语言→机器能够理解模型的部分含义
  - 可执行建模语言→机器能够理解和执行模型的准确语义

UML建模理念

- 语法明确、语义清晰的可视化语言
- 多种描述视角
  - 功能视角：系统或子系统要提供哪些功能(use case)？
  - 结构视角：系统有哪些组件(component)/类(class)/接口(interface)，相互间有什么关系(relation)？
  - 行为视角：组件/类能够做什么？组件/类之间如何协同？
  - 部署视角：组件/类如何分配到不同的可安装软件模块？
- 每个视角可以通过若干UML图来描述
  - 每个图有明确的主题
  - 控制每个图的规模

UML建模理念

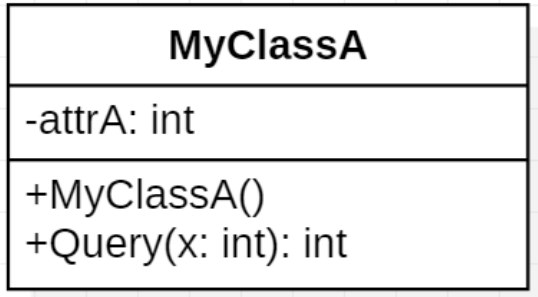
- 用例模型定义系统的需求
  - 使用可视化图来形象展示系统功能整体
  - 基于模板来描述每个用例(需求)的规格
    - 输入/输出，处理流程，异常情况，前置条件和后置条件
- 类模型定义系统的解决方案：使用“这些类”来实现相应的需求
  - 使用可视化图来形象展示系统解决方案的整体（类、类之间的关系）
  - 基于模板(属性、操作、约束条件)定义类的结构规格
- 状态模型定义类的行为机制：“这个类”将按照这样的行为逻辑运行
  - 使用可视化图来形象展示一个类受到关注的状态空间
  - 基于模板(状态行为、迁移行为)来定义类的行为规格→方法规格
- 交互模型定义类之间的协作机制：“这些类”在一起完成“这个业务”
  - 使用可视化图来形象展示类之间的交互序列
  - 基于模板(消息、消息时序控制)来定义类之间的交互规格→方法规格

UML模型组成

- 在UML建模工具中建立的各种图都是对模型的一种观察
- UML模型在哪儿？
  - 内存中、文件中
  - 由一组数据结构来定义
- 一组数据结构：UML元模型
- 在UML建模工具看来，各种图中的每个要素都是一个对象
  - 用例、类、属性、操作、关联、继承、消息、迁移…
  - 通过一系列数据结构来管理这些对象
- 所建立的UML模型实际上就是UML元模型的实例化结果
  - 通过复杂的图数据结构来管理

UML模型是一棵树

- UMLModel容器管理着模型的所有元素
  - ownedElements：UMLClassDiagram和模型元素(UMLClass)
- UMLClass对应用户所画出来的‘类’
  - 属性由UMLAttribute标记
  - 操作由UMLOperation标记
- 在uml diagram中画出来的元素都是模型组成部分
- UML模型是把各个diagram中的内容按照逻辑关系整合起来的结果
  - 有可能连接不起来

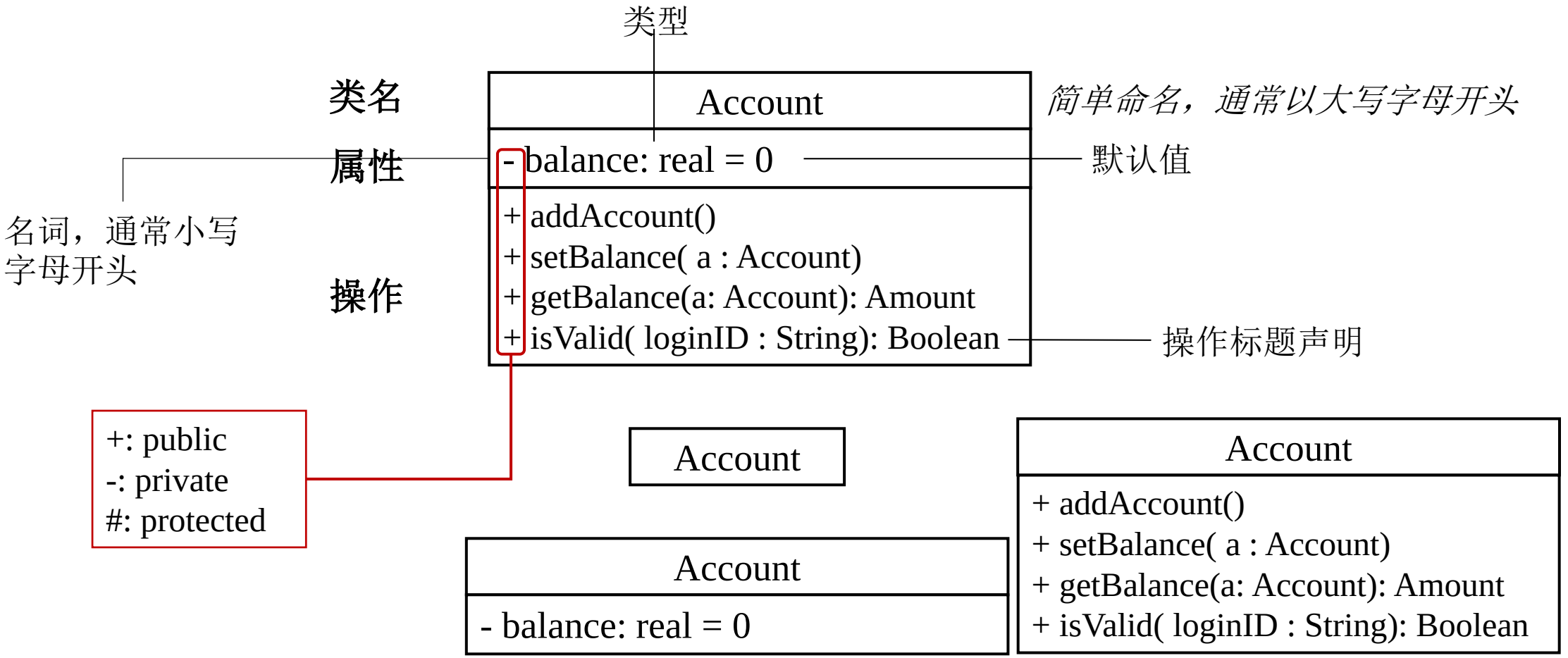


UML类图---对象建模的根本

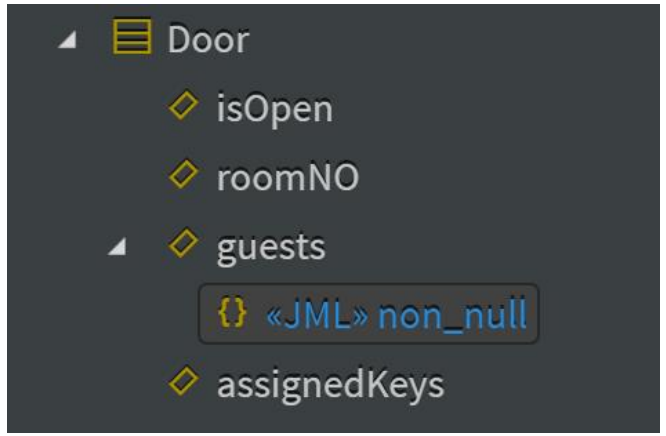
- 最常使用的UML模型图
- 围绕一个具体主题，展示相关的类、接口，它们之间的关系（依赖dependency、继承generalization、关联association、实现realization），以及必要的注释说明
- 三个层次的描述抽象
  - 概念层描述：用来分析**问题域描述**中可看到的类（分析模型）
  - 规格层描述：关注类的规格和接口
  - 实现层描述：可直接映射到代码细节的类

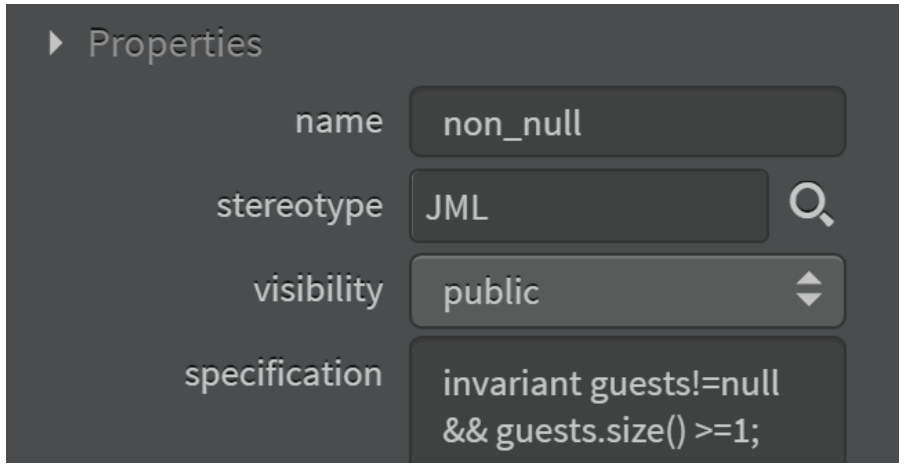
Most users of OO methods take an implementation perspective, which is a shame because the other perspectives are often more useful. -- Martin Fowler

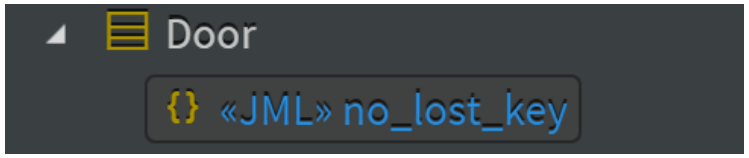
类的表示语法

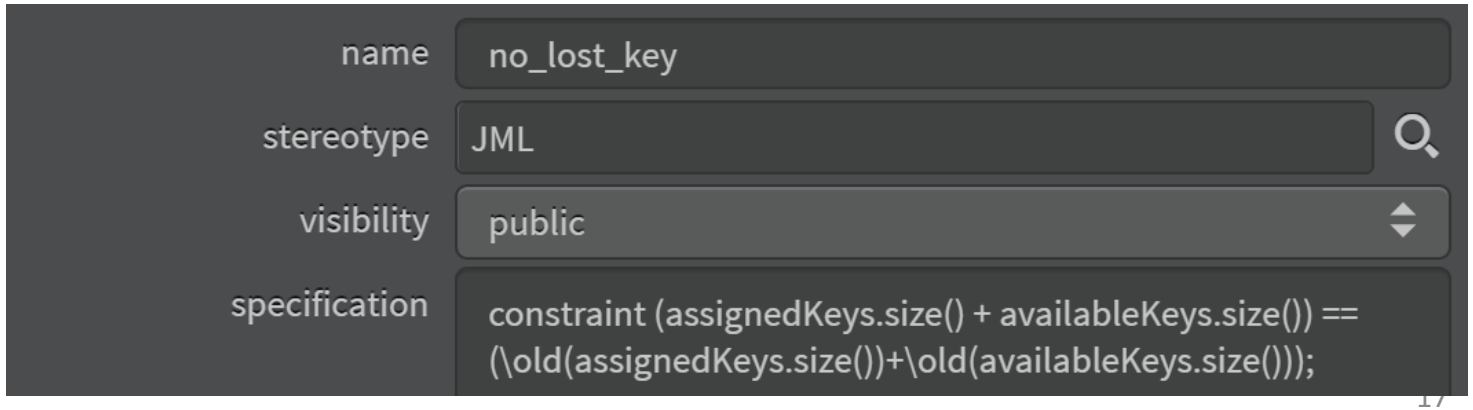

类型
类名
属性
操作
Account
- balance: real = 0
+ addAccount()
+ setBalance(a : Account)
+ getBalance(a: Account): Amount
+ isValid(loginID : String): Boolean
简单命名，通常以大写字母开头
默认值
操作标题声明
名词，通常小写字母开头
+: public
-: private
#: protected
Account
Account
- balance: real = 0
Account
+ addAccount()
+ setBalance(a : Account)
+ getBalance(a: Account): Amount
+ isValid(loginID : String): Boolean

可以通过Property来描述数据规格

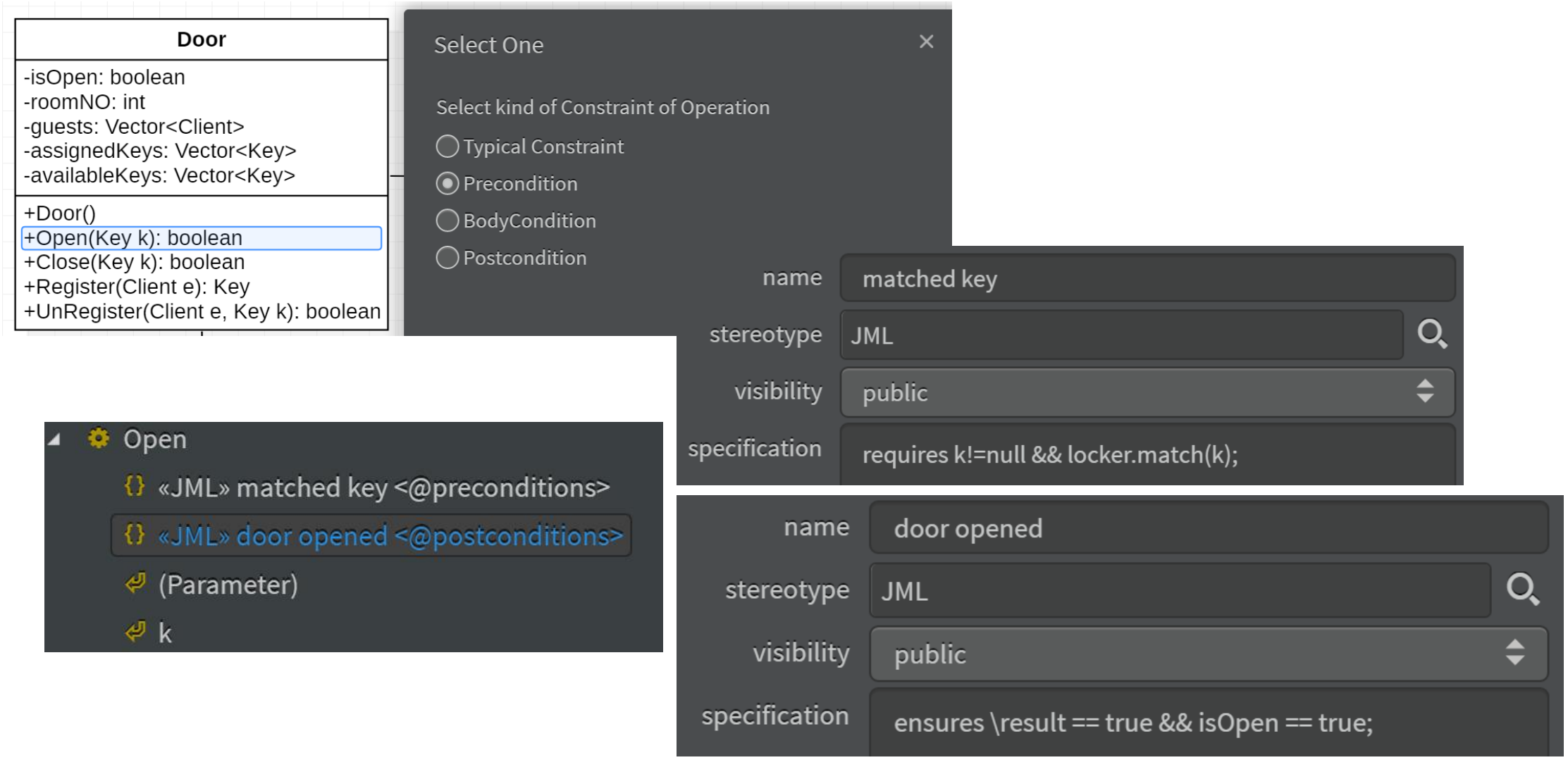






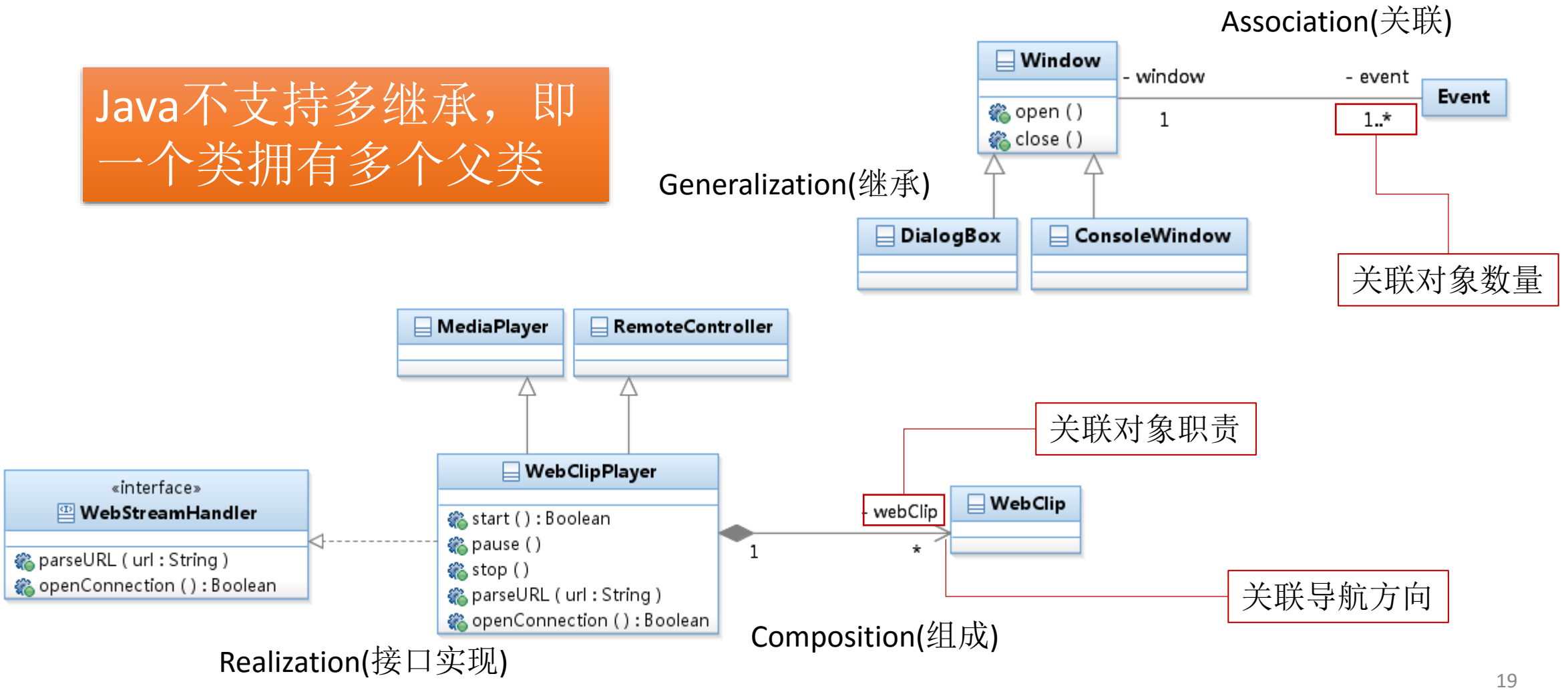


可以通过Property来描述方法规格



类之间的关系

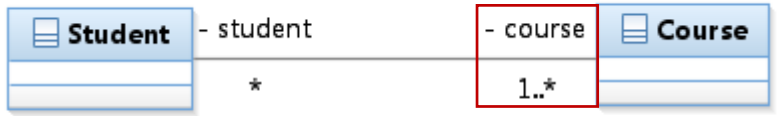
Java不支持多继承，即一个类拥有多个父类



类之间的关联关系

- 一个类需要另一个类的协助才能完成自己的工作
  - 用来管理相关信息
  - 需要获得一些信息
  - 需要协助做一些处理
  - 需要通知对方自己的状态变化
- 从对象的角度来理解关联
  - 从当前对象顺着关联方向可以找到相关联的对象
  - 注意关联对象的数目
    - *：表示为0到多个对象
    - 1..*：表示为1到多个对象
    - m..n：表示为m到n个对象
    - n：表示n个对象

```
public class Course{
  …
  private Vector<Student> student;
  …
}
```



course != null && course.size() >= 1

```
public class Student{
  …
  private Vector<Course> course;
  …
}
```

类之间的继承关系

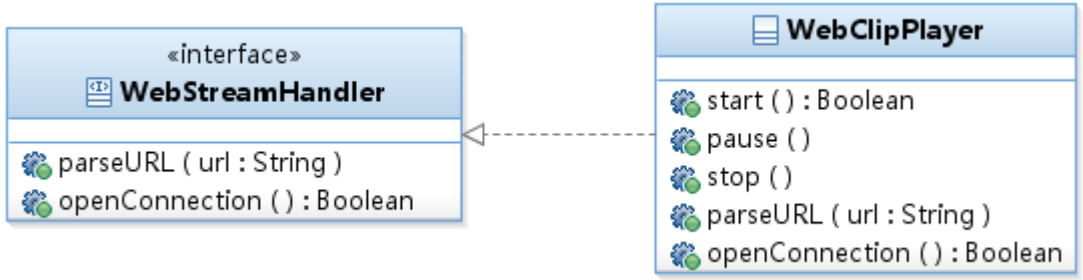
- 父类与子类
  - 父类概括子类
  - 子类扩展父类
- UML支持灵活的多继承
  - Java不支持
  - 建议在使用UML时不用多继承
- 一旦建立继承关系，子类将自动拥有父类的所有属性和操作
  - 设计层次和实现层次
  - Note：不要在子类中重复定义父类已经定义的内容

```
public class OOCourse extends Course{
  …
  …
}
```

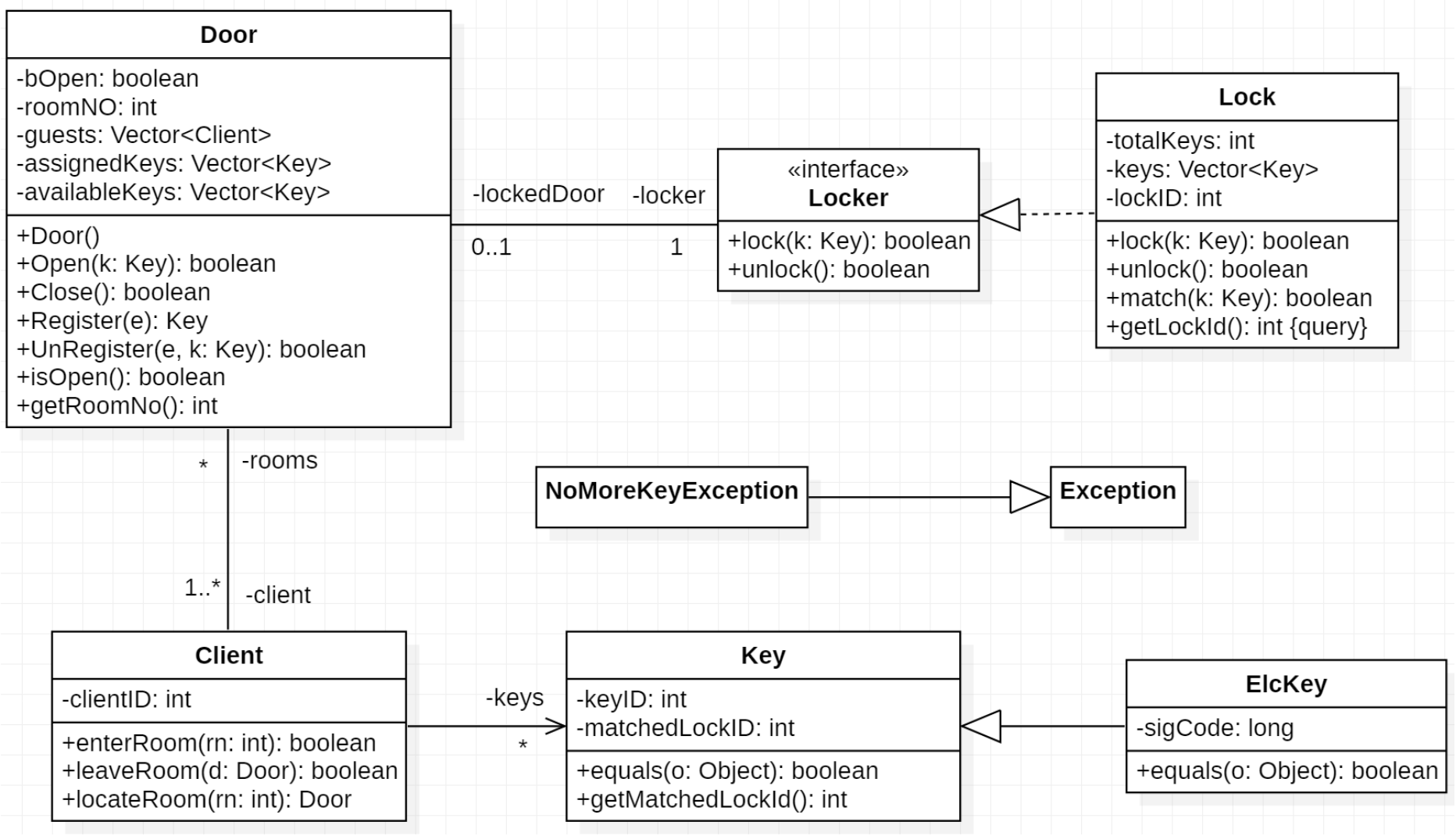
类对接口的实现关系

- UML与Java具有一致性
  - 一个非抽象类必须实现接口中定义但未实现的所有操作
- 接口是UML语言预定义的一种特殊的类
- 一个类可以实现多个接口
- 实现类需要显式列出要实现的操作
  - 和继承机制不同！

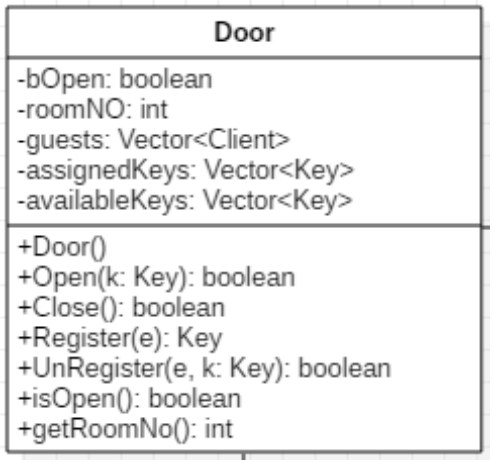
```
public class A extends B implements C,D,E{
   …
   …
}
```



Case Study: Open&Close



在UML模型层次来理解类

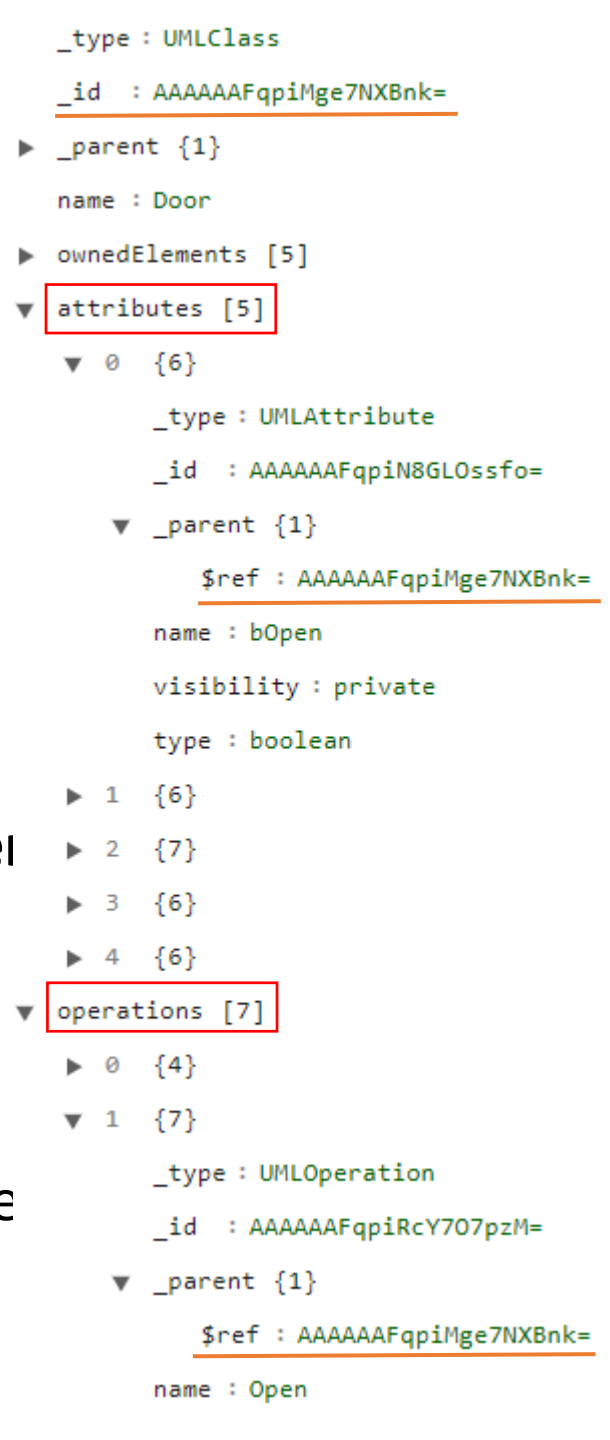


UMLModelElement propery:
name
visibility

_** field: for UML or starUML type immutable
** field: for user defined type mutable

- Door是一个类
  - UML: Door is an object of **UMLClass**.
  - **UMLClass** is a kind of **UMLModelElement**
- bOpen是Door的一个属性
  - UML: bOpen is an object of **UMLAttribute**
  - **UMLAtrribute** is a kind of **UMLModelEleme**
  - bOpen is a member of Door
- Open是Door的一个操作
  - UML: Open is an object of **UMLOperation**
  - **UMLOperation** is a kind of UMLModelEleme
  - O

Door is a container object (typed as UMLClass) of 5 attribute objects (typed as UMLAttribute), and 7 operation objects (typed as UMLOperation).



在UML模型层次理解类

- UMLAttribute对象
  - name
  - type
  - multiplicity
  - defaultValue
  - isUnique
  - specification

```
_type : UMLAttribute
_id   : AAAAAAFqp0ZAqWCp/yc=
▼ _parent {1}
    $ref : AAAAAAFqpiMge7NXBnk=
name : guests
▼ ownedElements [1]
  ▼ 0  {6}
      _type : UMLConstraint
      _id   : AAAAAAFqp26huGyajnk=
    ▶ _parent {1}
      name : non_null
      stereotype : JML
      specification : invariant guests!=null &&
                      guests.size() >=1;
```

```
_type : UMLClass
_id   : AAAAAAFqpiMge7NXBnk=
▶ _parent {1}
  name : Door
▶ ownedElements [5]
▶ attributes [5]
▶ operations [7]
```

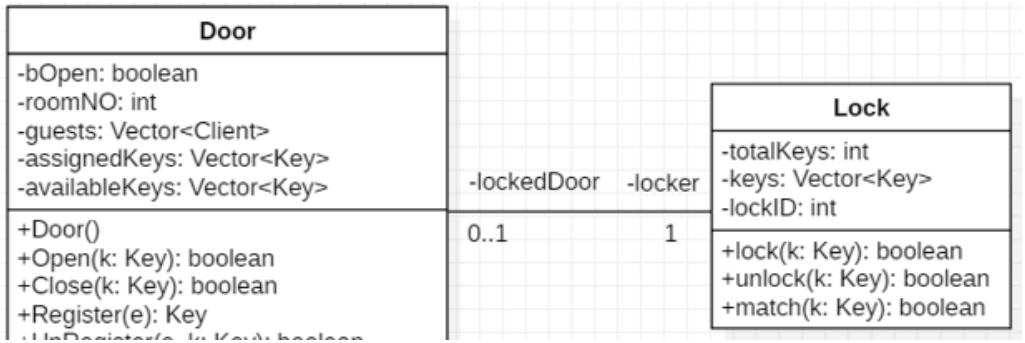
- UMLOperation对象
  - name
  - return type
  - parameters
  - rasiedExceptions
  - specification
- UMLParameter对象
  - name
  - type
  - direction
    - in, inout, out, return
  - defaultValue

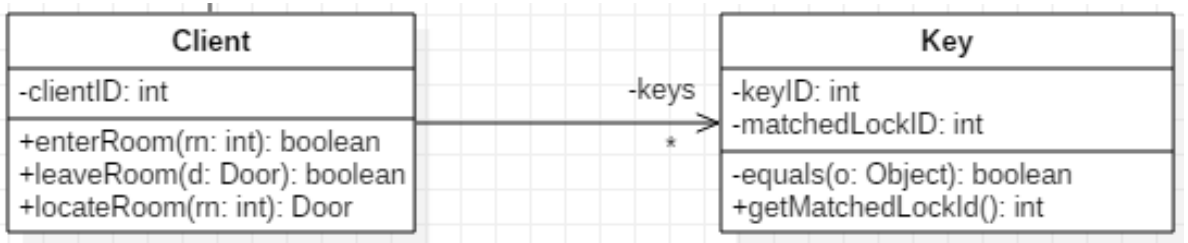
```
_type : UMLOperation
_id   : AAAAAAFqpiRcY7O7pzM=
▼ _parent {1}
    $ref : AAAAAAFqpiMge7NXBnk=
name : Open
▼ parameters [2]
  ▼ 0  {5}
      _type : UMLParameter
      _id   : AAAAAAFqpim3MbPYrBA=
    ▶ _parent {1}
      type : boolean
      direction : return
  ▼ 1  {5}
      _type : UMLParameter
      _id   : AAAAAAFqpz3cy1dqvuQ=
    ▶ _parent {1}
      name : k
    ▶ type {1}
▶ preconditions [1]
▶ postconditions [1]
```

_parent不是面向对象抽象层次的parent-child关系，而是指管理层次树中的层次关系

在UML模型层次看待类关联关系

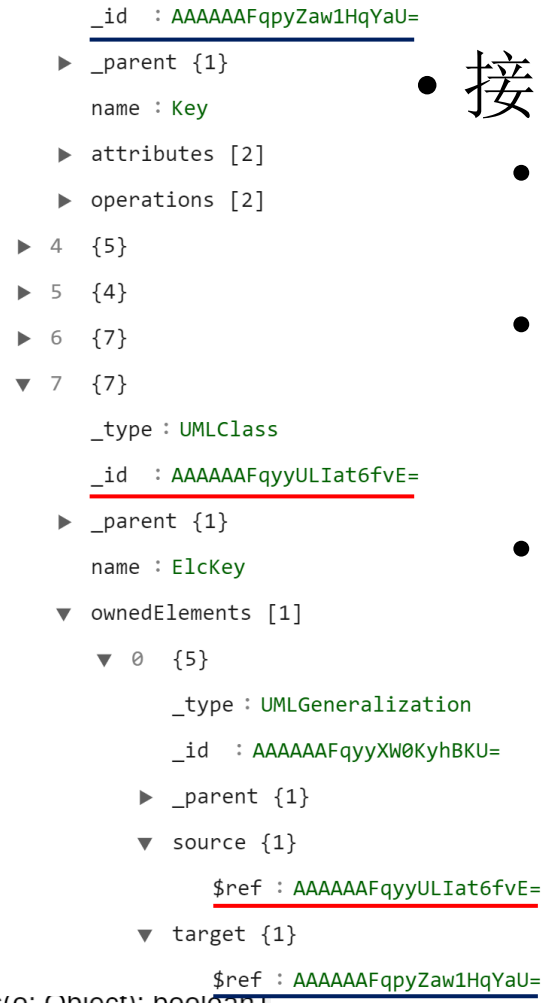
- 关联关系
  - <Door, Lock> is an object of **UMLAssociation**
  - UMLAssociation is a kind of UMLModelElement
  - UMLAssociation has two objects typed as **UMLAssociationEnd**
    - end1: {*name*:lockedDoor, *visibility*:private, *multiplicity*:0..1, *reference*:Door}
    - end2: {*name*:locker, *visibility*:private, *multiplicity*:1, *reference*:Lock}
  - 如果不关心某一端引用的对象，相应end的特性可以缺省
  - navigable：关联访问方向
  - aggregation: {none, shared, composite}

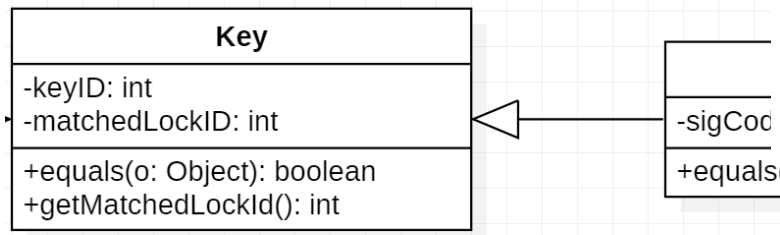




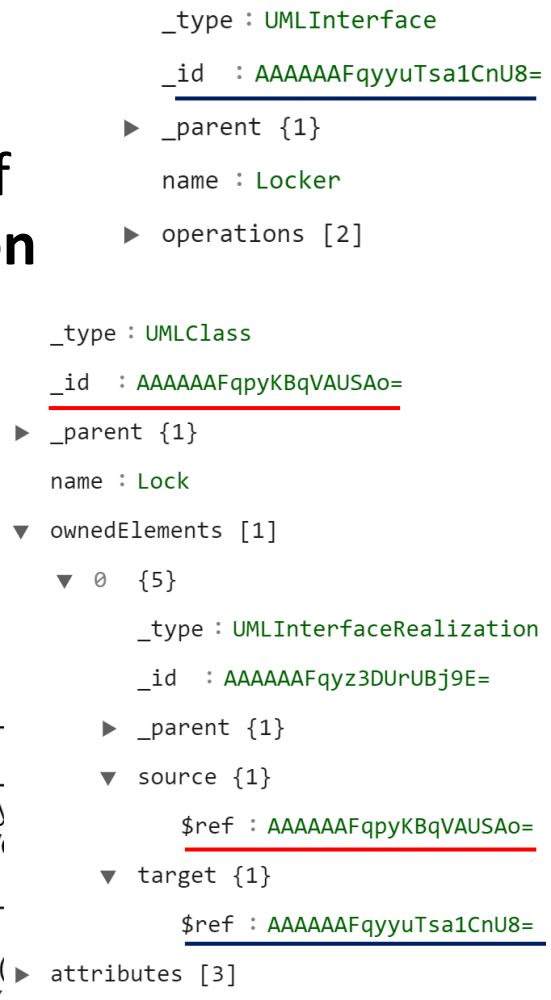
在UML模型层次来理解类抽象层次

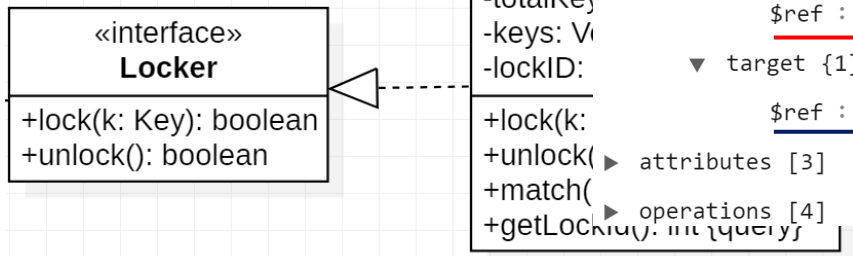
- 继承层次
  - <ElcKey, Key>: object of **UMLGeneralization**
  - source: ElcKey
    - object of **UMLClass**
    - Role: subclass
  - target: Key
    - object of **UMLClass**
    - Role: superclass





- 接口实现层次
  - <Lock,Locker>: object of **UMLInterfaceRealization**
  - source: Lock
    - object of **UMLClass**
    - Role: impl provider
  - target: Locker
    - object of **UMLInterface**
    - Role: interface



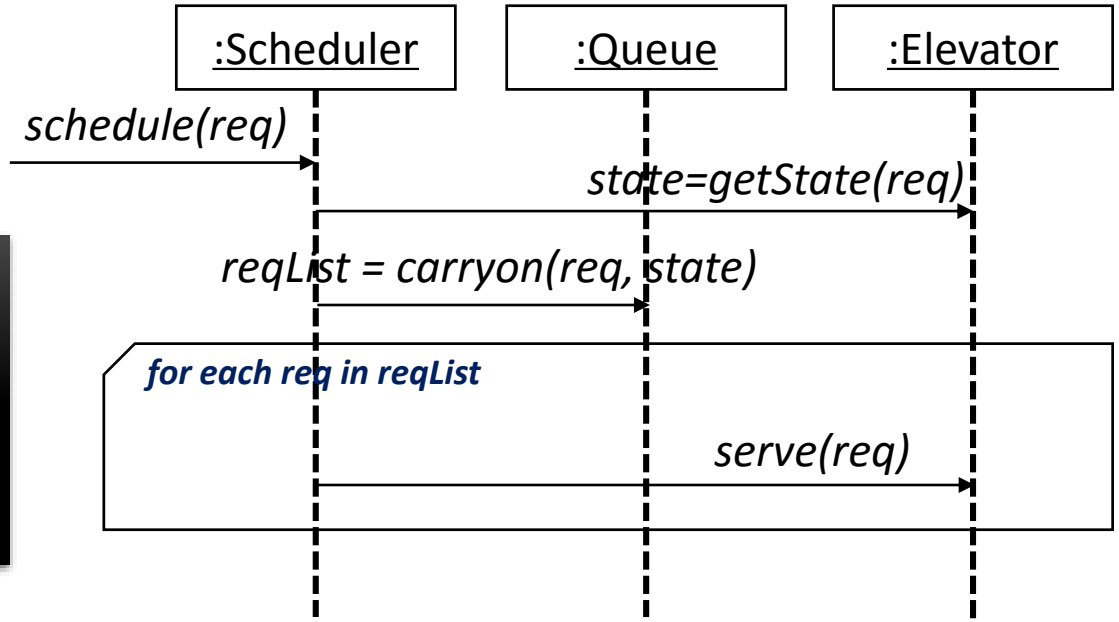


面向对象程序行为的UML表示

- 单个类视角下的行为
  - 观察行为：不改变对象状态
  - 控制行为：会改变对象状态
- 两个类之间的交互行为
  - 方法调用
  - 数据共享（线程交互）
- 多个类之间的组合控制行为
  - 流程控制

UML顺序图模型

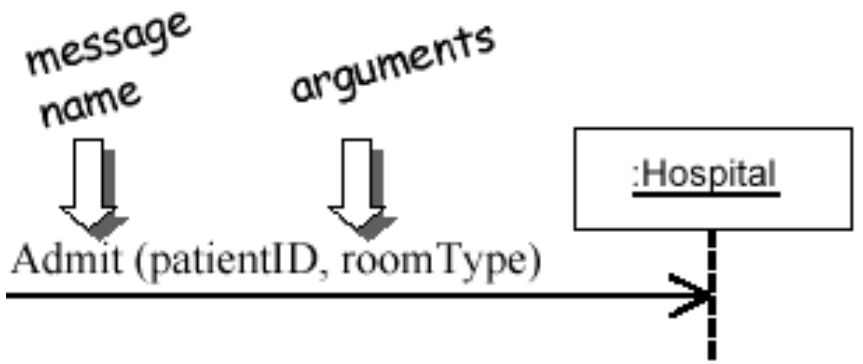
- 顺序图(sequence diagram)来自于通信领域，表示通信实体之间的通信关系
  - 参与对象(participant)：参与交互的对象
  - 消息：对象间的交互
  - 对象生命线：描述对象的存活生命期



顺序图具有典型的笛卡尔坐标图性质:
水平坐标：排列参与交互的对象
垂直坐标：消息时序和时间信息(时间从上往下增长)

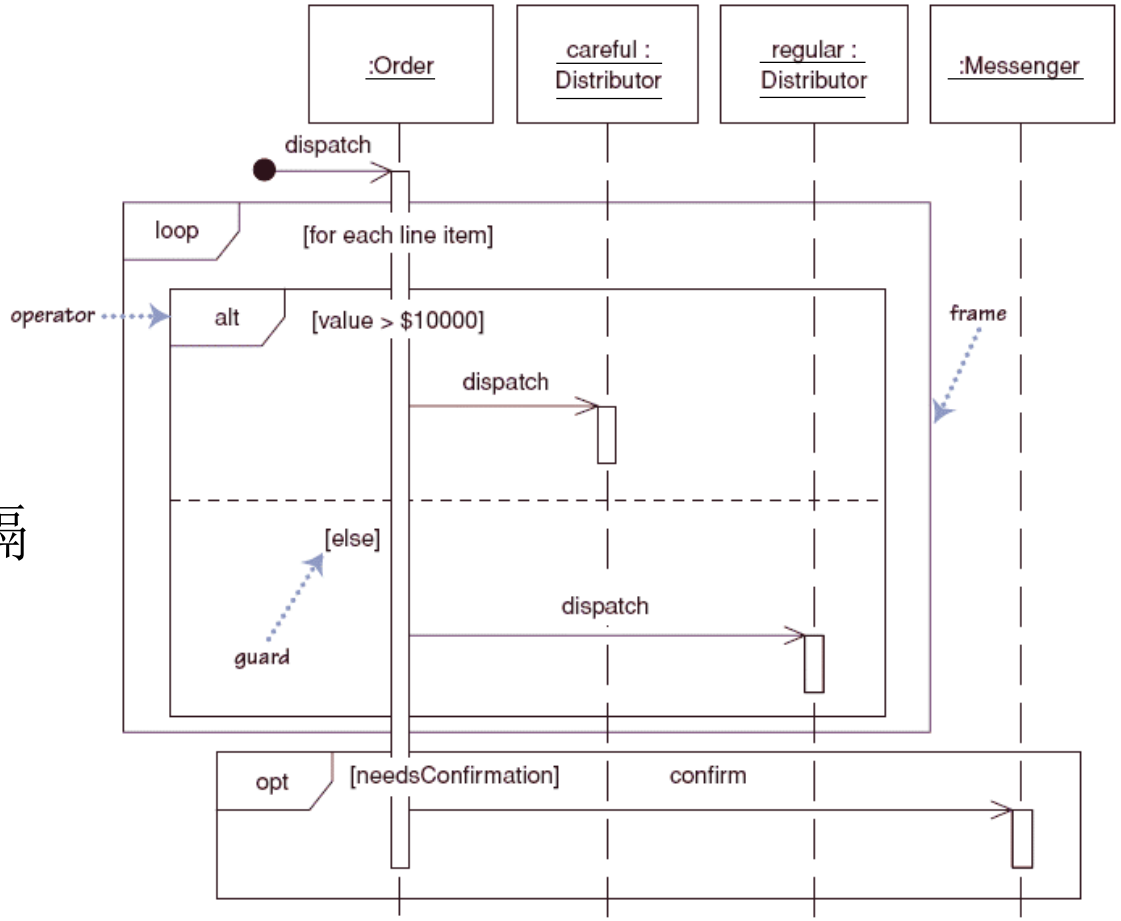
UML顺序图模型

- 对象生命线(UMLLifeline)
  - 矩形框，名称:关联的对象类型名
  - 名称有时可省略
  - 每个对象生命线都应关联到到一个对象
- 消息(UMLMessage)
  - [var=]消息名([消息参数])
  - 与对象连接
- 消息连接意味合作关系
  - 发送者: 请求接受者的服务 / 通知接受者相关状态的变化
  - 接受者: 发送者在请求服务 / 发送者在通知我关心的信息



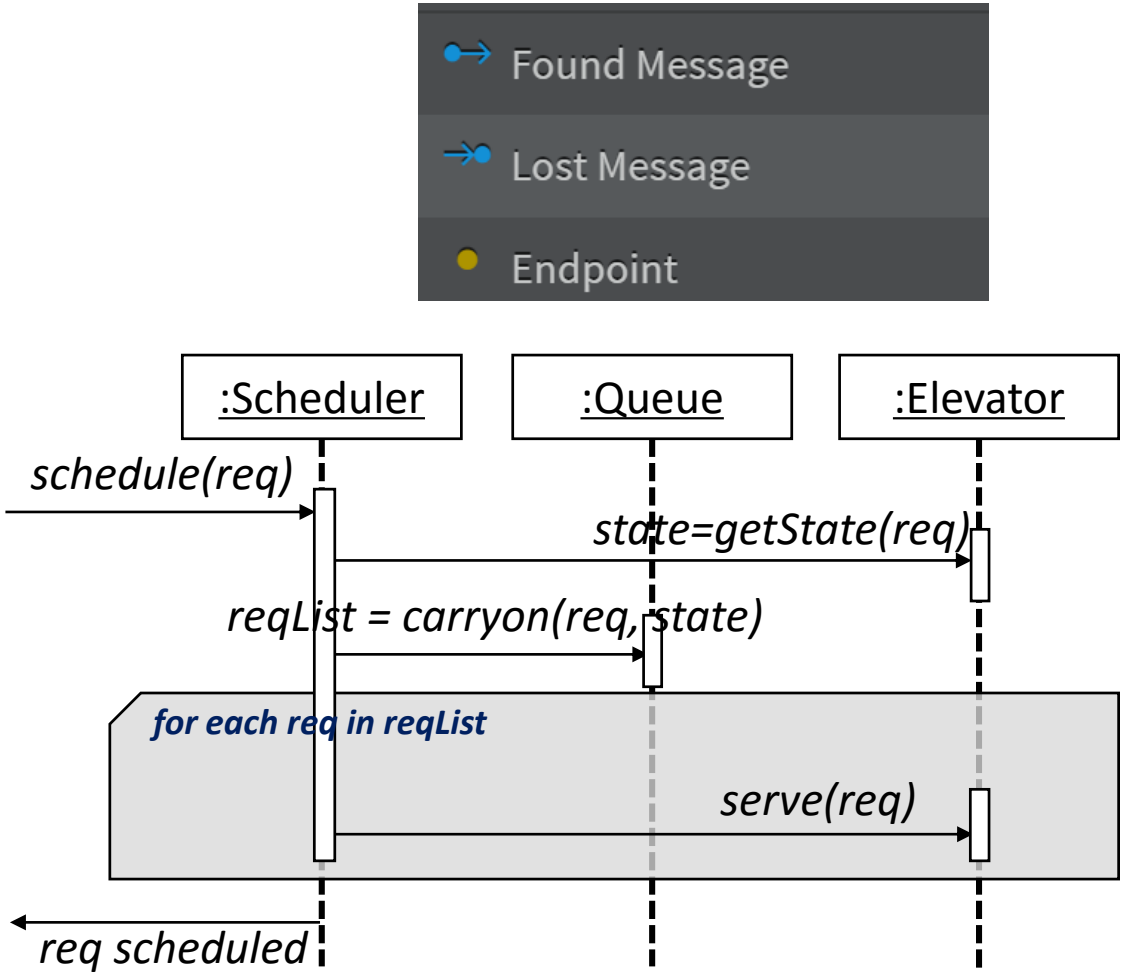
UML顺序图模型

- 基于消息块的交互流程控制
  - UMLCombinedFragment
- if控制-> 可选消息快
  - (opt) [控制条件]
- if/else-> 多分支消息块
  - (alt) [控制条件], 通过水平虚线来分隔多个分支控制
- loop -> 循环消息快
  - (loop) [循环控制条件或循环事项]



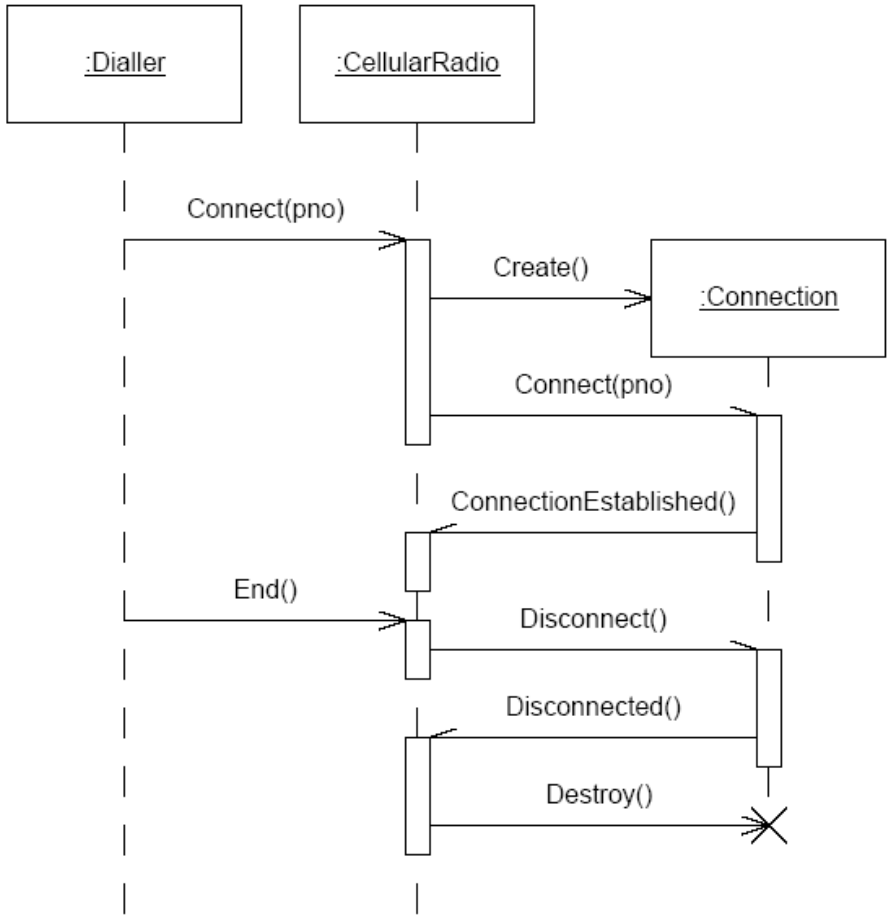
UML顺序图模型

- 有时不关心消息来自于哪个对象，只关心收到的消息
  - Found Message(staruml)
  - 如”*schedule(req)*”
  - UMLEndpoint --> Receiver
- 有时不关心消息发给谁，只关心发出去消息
  - Lost Message(staruml)
  - 如” *req scheduled*”
  - Sender --> UMLEndpoint

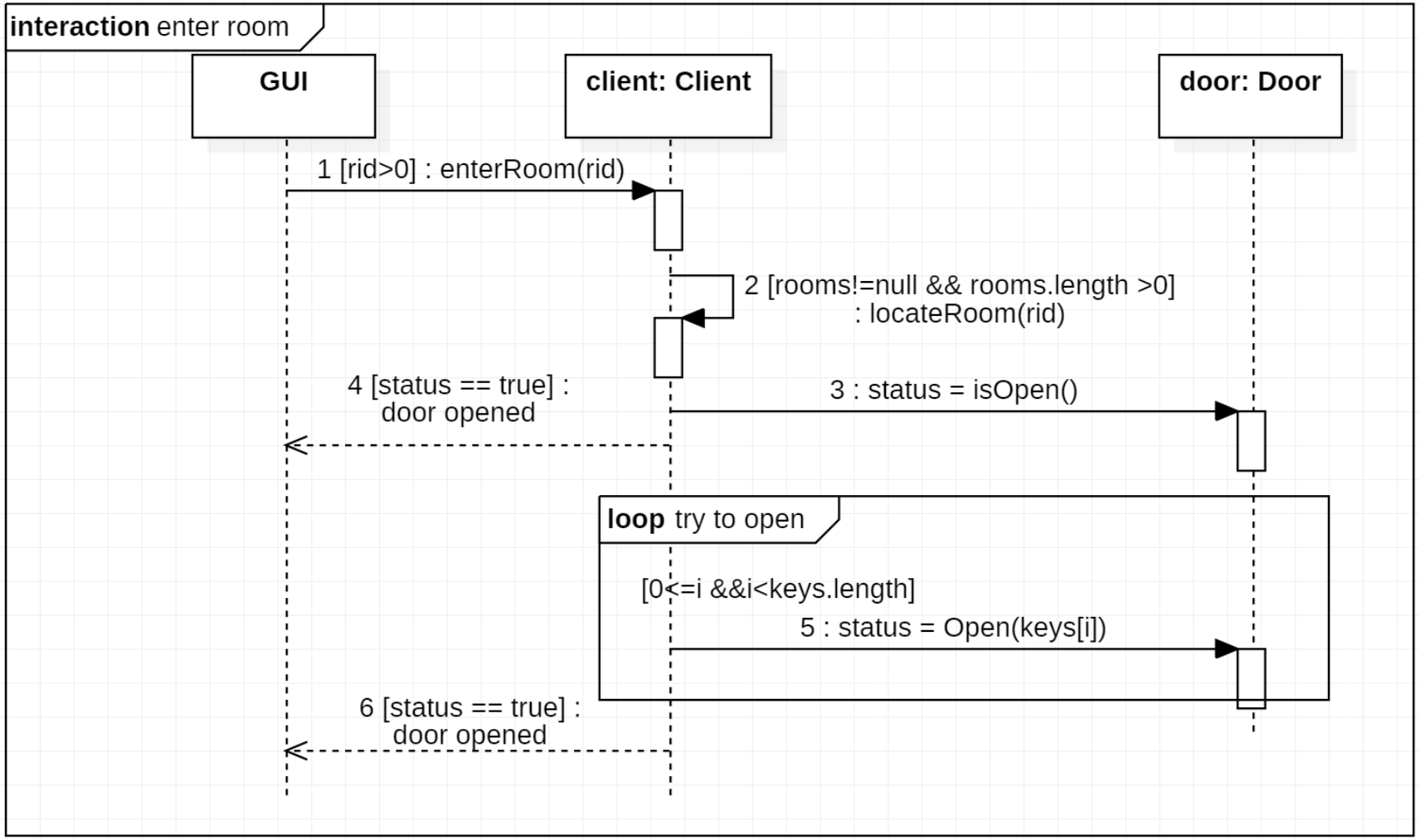


UML顺序图模型

- 顺序图描述多个类之间如何协作来完成一个具体功能
  - 架构设计的一部分
- 每个顺序图都应该有一个明确的行为主题
  - 建模主题反应建模者意图
- 这个顺序图有什么问题？

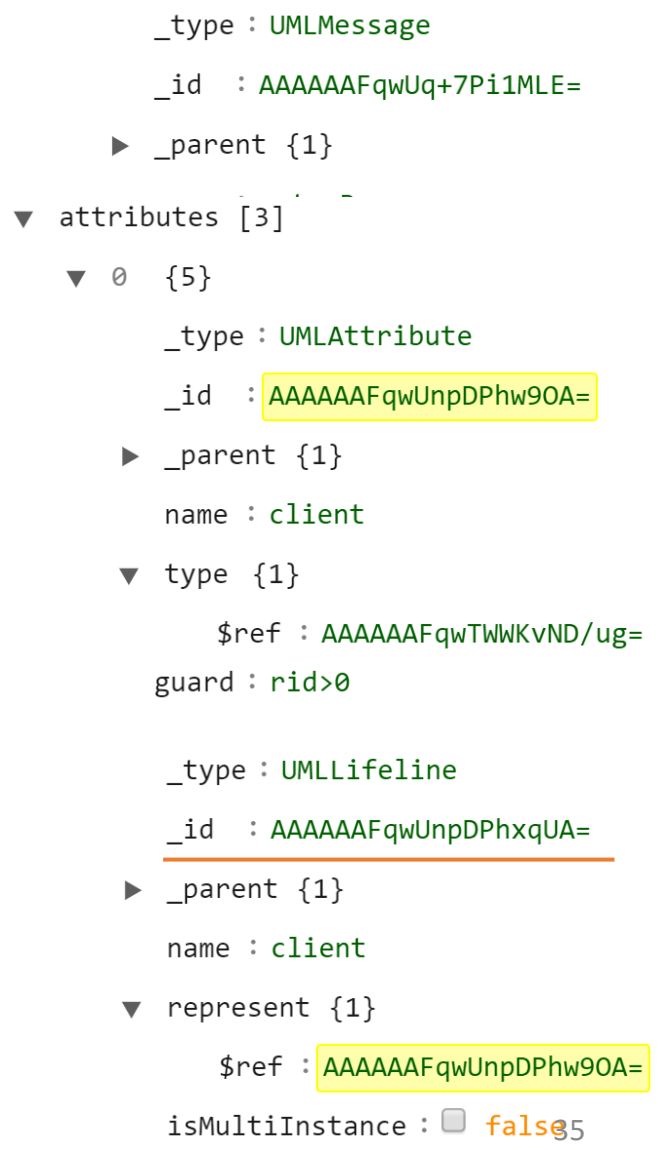


Case Study: Open&Close



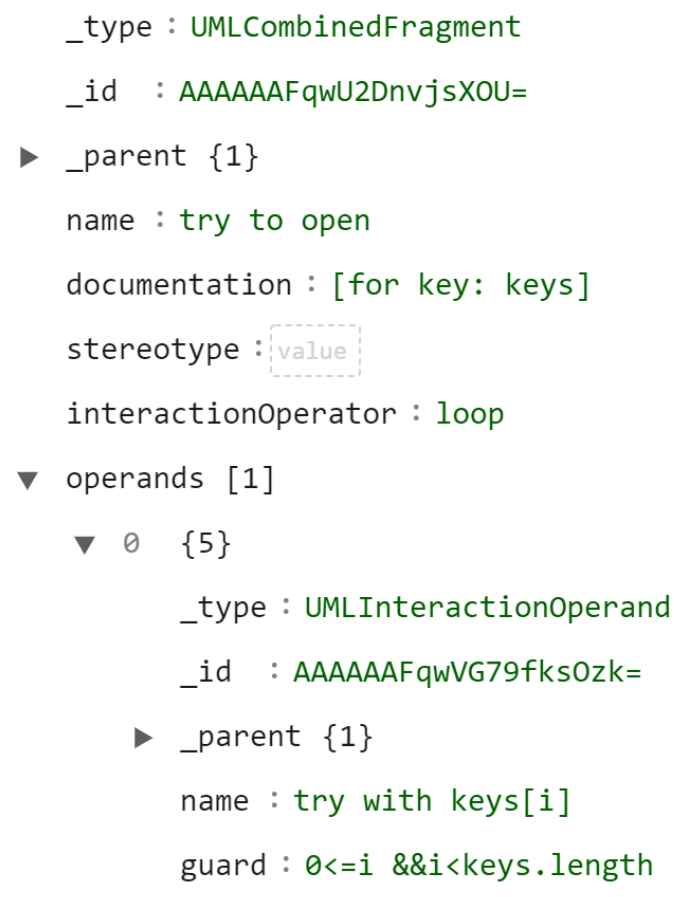
在UML模型层次理解消息

- enterRoom(rid) is an object of **UMLMessage**
  - GUI(the sender) is an object of **UMLLifeline**
  - client(the receiver) is an object of **UMLLifeline**
- **UMLMessage** is a kind of **UMLModelElement**
  - name, source, target, signature, arguments, guard…
- UMLLifeline is a kind of UMLModelElement
  - name, **represent**, isMultiInstance
  - UMLLifeline represents an object of UMLAttribute



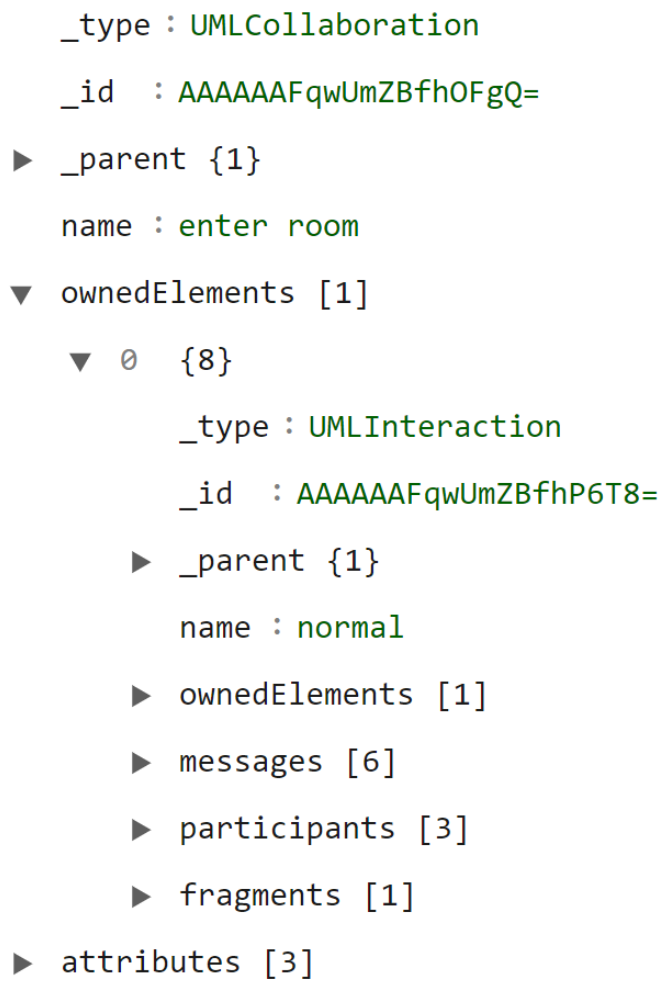
在UML模型层次理解消息控制

- “try to open” is an object of **UMLCombinedFragment**
  - operator + operand
  - interationOperator: {loop, alt, opt, …}
  - operand: try with keys[i]
- 消息在**UMLLifeline**之间传递
  - UMLLifeline关联到对象
- 消息也可以在**UMLEndpoint**与**UMLLifeline**之间传递
  - UMLEndpoint --> UMLLifeline: found message
  - UMLLifeline --> UMLEndpoint: lost message



在UML模型层次理解对象协同

- 顺序图定义了对象协同(**UMLCollaboration**)
  - ownedElements
  - attributes: 来完成协同行为的属性成员(对象)
    - UMLAttribute
- 协同行为: **UMLInteraction**
  - 可以建立多个UMLInteraction，描述特定主题下的交互行为
  - messages
  - participants
    - UMLLifeline
    - UMLEndpoint
  - fragments

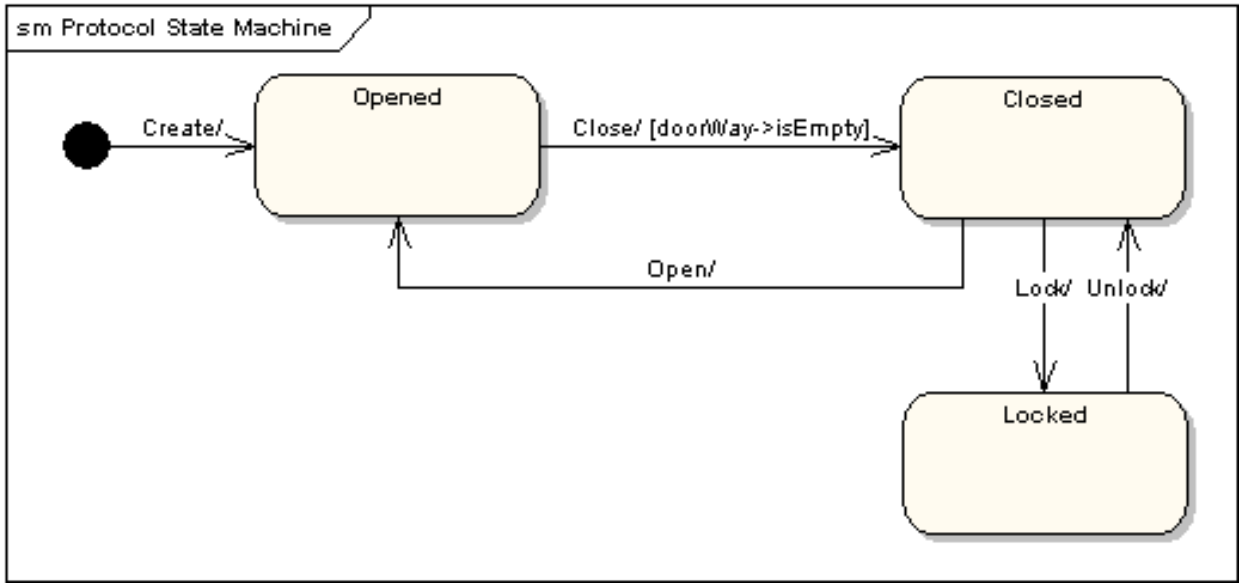


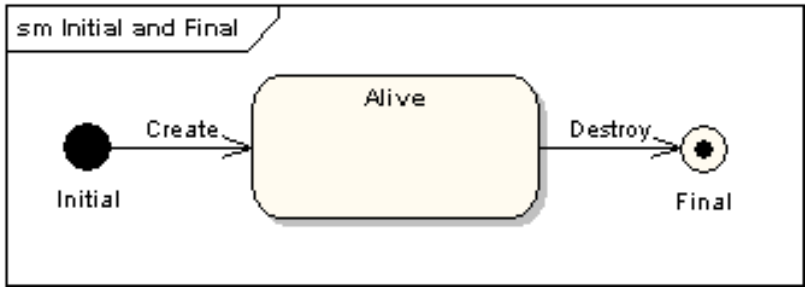
UML状态图模型

- 对象是一种状态化的存在
  - 状态由数据定义
  - 外部可见状态、内部细节状态
- 对象行为引发状态变化
  - 状态迁移
- 使用UML状态图来描述外部可见的状态
  - 类的行为规格设计
- **本课程强调针对一个类来建立其状态模型**

UML状态图模型

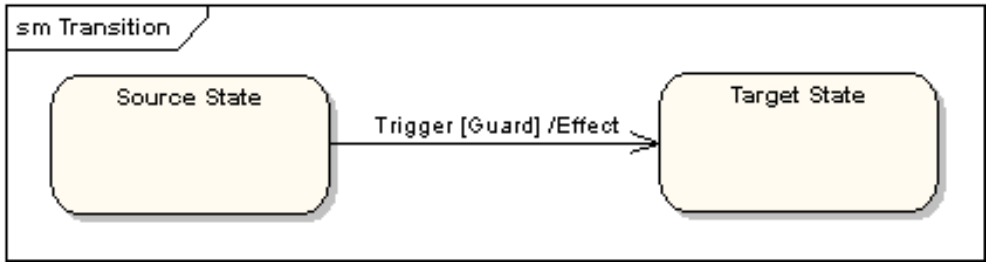
- 只用来描述一个对象的行为
  - 不能跨越“边界”
- 状态使用圆角矩形框表示
  - 初始状态
  - 终止状态(可能没有)
- 迁移使用带箭头的线表示
  - 一个迁移只能连接一个源状态、一个目标状态
  - 任何一个状态都必须从初始状态可达
  - 任何一个状态都能够迁移到终止状态(如果有)

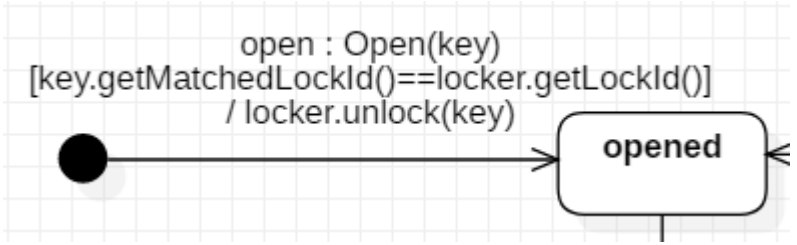


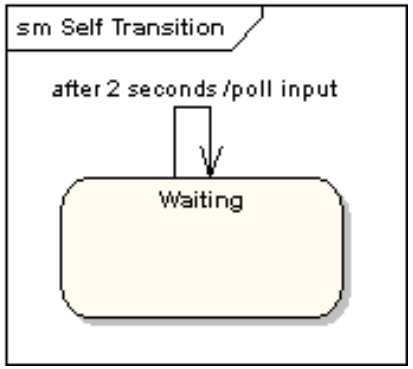


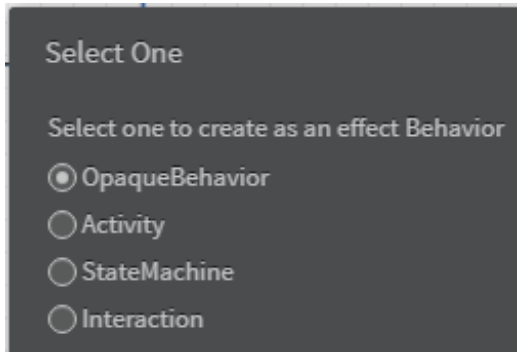
UML状态图模型

- 迁移的定义(UMLTransition)
  - trigger[guard]/effect
- Trigger是引起迁移的原因
  - UMLEvent
- Guard是迁移能够发生的前置条件
- Effect是迁移发生的后置条件之一
  - Effect
  - 对象状态改变为迁移的目标状态
- 目前我们规定只使用简单情形下的Effect
  - OpaqueBehavior: UMLOpaueBehavior



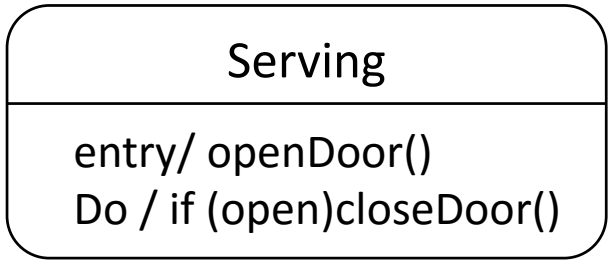






UML状态图模型

- 对象在某些状态下可完成一定的动作
  - 进入状态动作entry activity：在进入状态时执行
  - 退出状态动作exit activity：在退出状态时执行
  - 处于状态中的动作do activity：进入状态后执行
- 可以为一个状态构造任意数目的这三种类型动作



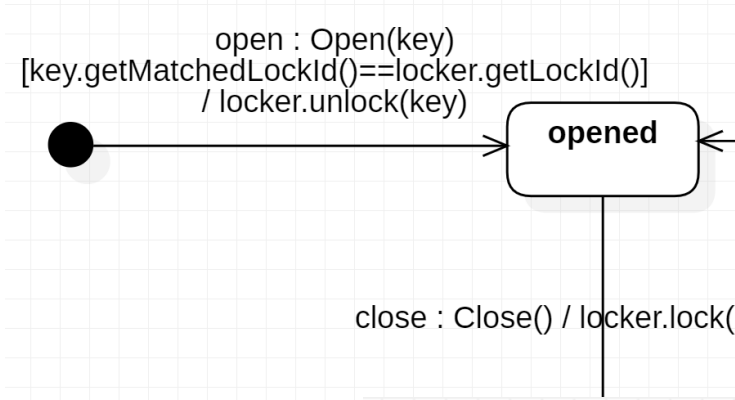
电梯在进入服务状态时打开门
处于服务状态后，如果未关，不断尝试关闭电梯门

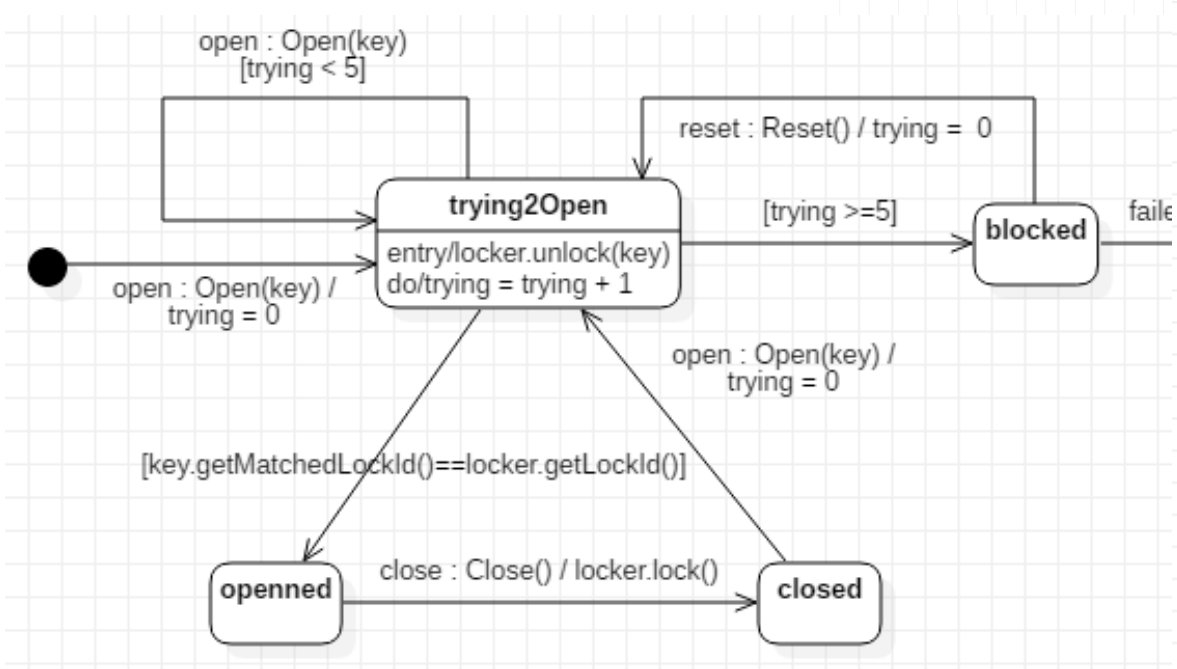
UML状态图模型

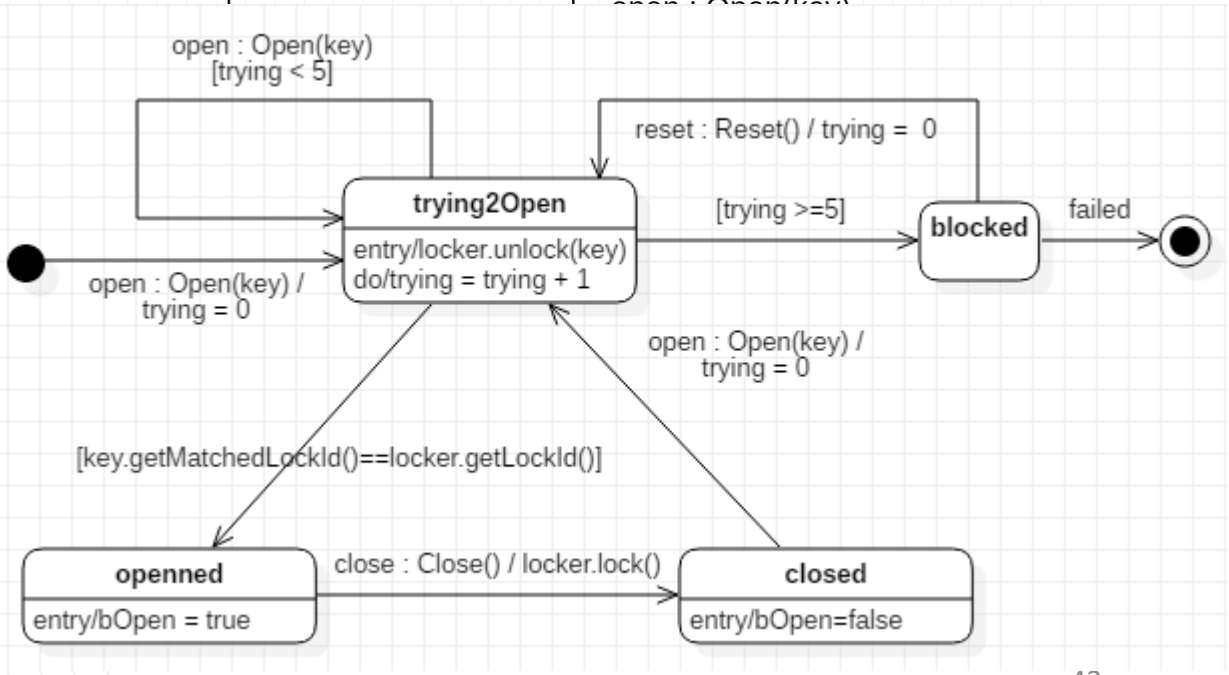
- 状态迁移与程序代码的对应
  - 从源状态来看
    - Trigger：方法调用或者事件通知
    - Guard：调用时的相关检查
  - 从目标状态来看
    - Trigger：方法体
    - Guard：方法入口处的检查
    - Effect：方法执行后的效果，即迁移到目标状态
- 状态动作与程序代码的对应
  - 对应到方法，通常外部用户不会调用，对象为了满足相关规格而实施的行为
  - 要求不改变对象状态

Case Study：Door状态模型

- 顶层状态
  - opened, closed

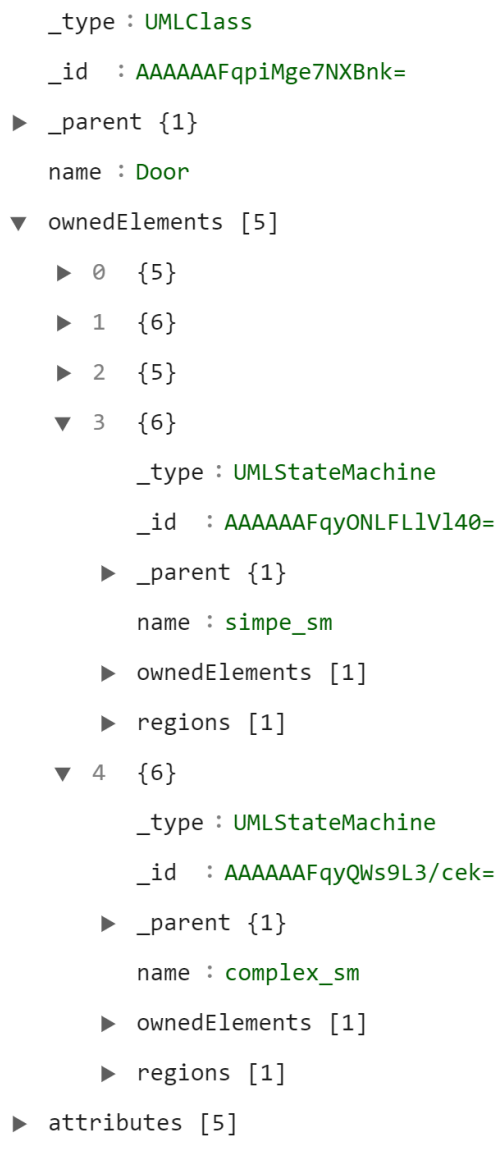






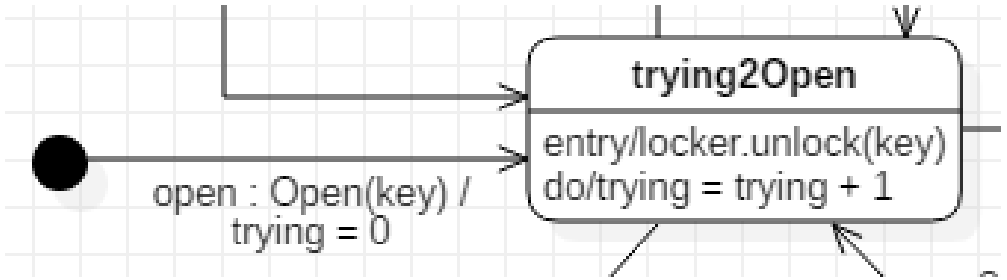
在UML模型层次理解状态机模型

- UMLClass
  - ownedElements
    - UMLAssociation
    - UMLConstraint
    - UMLGeneralization
    - **UMLStateMachine**
  - attributes
  - operations
- UMLStateMachine: object container
  - ownedElements
    - UMLStatechartDiagram
  - regions: {UMLRegion}
    - Vertices: {UMLState}
    - Transitions: {UMLTransition}



在UML模型层次理解状态

- UMLState
  - name
  - entryActivities: {UMLOpqueBehavior}
  - doActivities: {UMLOpqueBehavior}
  - exitActivities: {UMLOpqueBehavior}

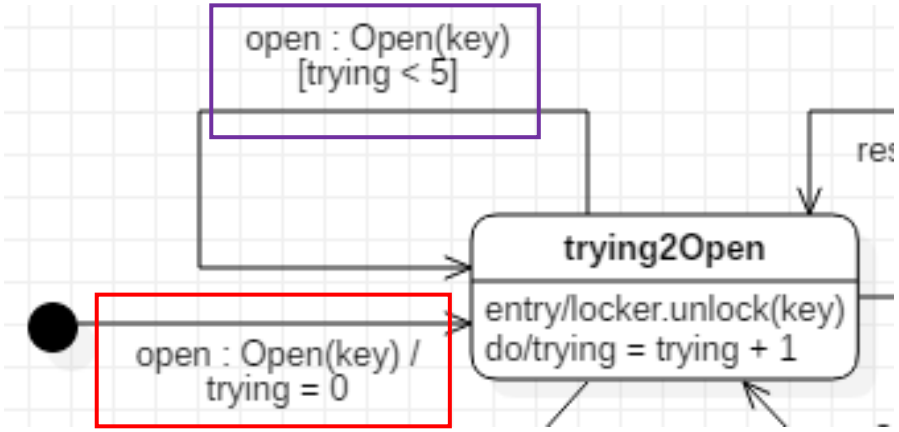



```
_type : UMLPseudostate
_id   : AAAAAAFqyeEMPTDVjII=
▶ _parent {1}
kind : ☐ initial

_type : UMLState
_id   : AAAAAAFqyeFWgDDmGrM=
▶ _parent {1}
name : trying2Open
▶ entryActivities [1]
▶ doActivities [1]
```

在UML模型层次理解状态迁移

- UMLTransition
  - UMLEvent
  - UMLGuard
  - UMLOpqueBehavior




```
_type : UMLTransition
_id : AAAAAAFqyeMDizGdIG4=
▶ _parent {1}
name : open
▼ source {1}
    $ref : AAAAAAFqyeFWgDDmGrM=
▼ target {1}
    $ref : AAAAAAFqyeFWgDDmGrM=
guard : trying < 5
▼ triggers [1]
  ▼ 0 {4}
      _type : UMLEvent
      _id : AAAAAAFqye1oTDJqUtU=
    ▶ _parent {1}
      name : Open(key)
```

```
_type : UMLTransition
_id : AAAAAAFqyeLuBjGMJ9M=
▶ _parent {1}
name : open
▼ source {1}
    $ref : AAAAAAFqyeEMPTDVjII=
▼ target {1}
    $ref : AAAAAAFqyeFWgDDmGrM=
▼ triggers [1]
  ▼ 0 {4}
      _type : UMLEvent
      _id : AAAAAAFqyealLTIrDKQ=
    ▶ _parent {1}
      name : Open(key)
▼ effects [1]
  ▼ 0 {4}
      _type : UMLOpaqueBehavior
      _id : AAAAAAFqyetqrDJRthg=
    ▶ _parent {1}
      name : trying = 0
```

作业分析

- 给定UML模型
  - 类图、顺序图、状态图
- 提供模型文件解析包
  - 从UML模型文件中提取所关注信息，形成标签对象容器
  - 每个标签对象提供的信息：{<property, value>}
  - 标签对象容器提供迭代器
  - 标签对象提供迭代器
- 提供查询接口，要求同学们实现
  - 迭代访问对象容器中的对象和对象信息
  - 按照_id, _parent, _ref建立层次关系和引用关系
  - 按照对象类别恢复模型语义信息