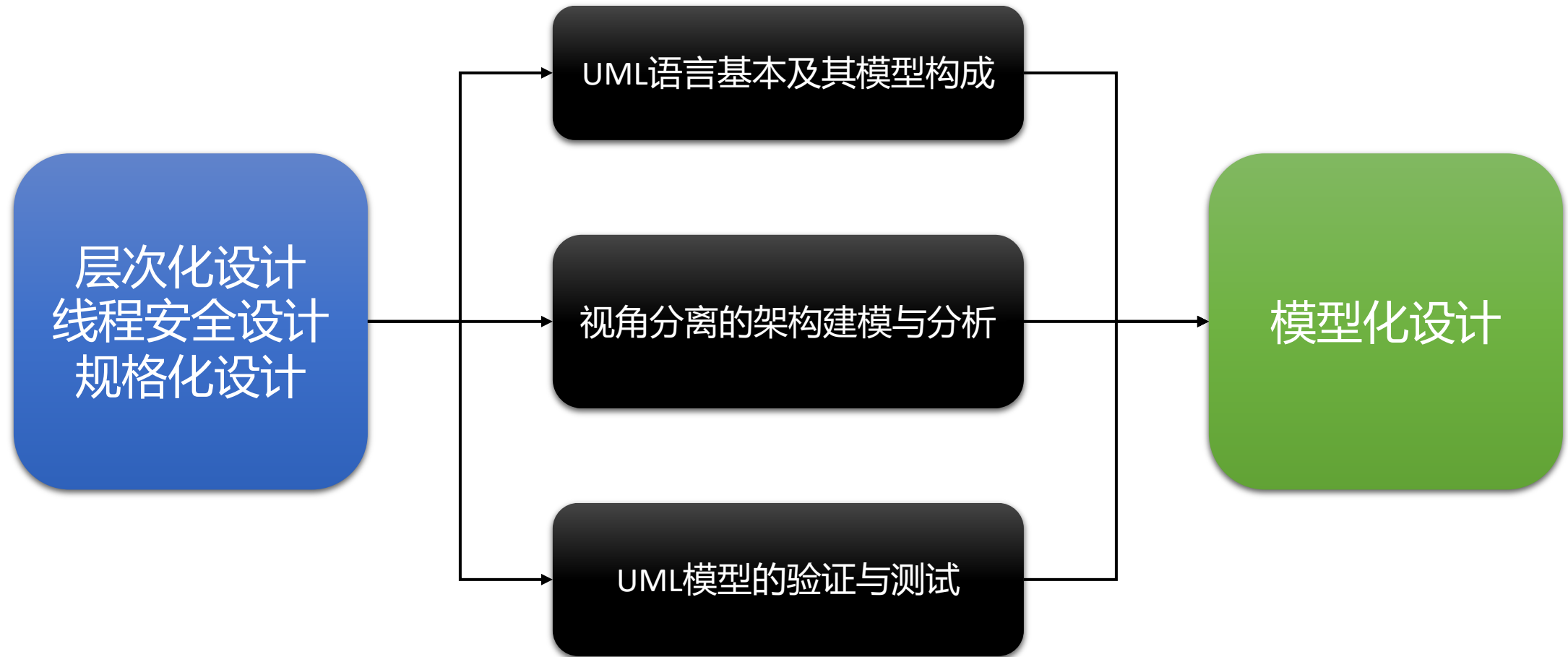


《面向对象设计与构造》

Lec13-UML语言及其模型

面向对象设计与构造课程组2023
计算机学院

课程回顾及本单元内容总览



摘要

- 语言的基本概念
- 对程序系统的理解和描述
- UML语言简介
- UML类图：基本概念及几个关键标签类型
- UML顺序图：基本概念及几个关键标签类型
- UML状态图：基本概念及几个关键标签类型
- 作业分析

语言的基本概念

- 语言：是一套使用**概念**、**关系**和**规则**来表达思维的指令集合
- 设计一种新语言的目的：**获得更好的表达效果**
 - 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：**人与人、人与机器**
 - 表示者通过语言来表达自己的观点/思维
 - 接收者通过语言来感知和理解对方的观点



面向对象是一个语言系统

- 语言一般都会提供**两类基本要素**
 - 词汇：一种语言里所有的词和词的固定搭配的总和
 - 语法：词汇的连接规则，语言是由词汇按一定语法连接构成的表义系统
- 面向对象本质上定义了一个**抽象语言系统**
 - 词汇：对象、属性、操作、活动、流程、状态.....
 - 语法：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、操作与状态间连接.....

对程序系统的理解和描述

- 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

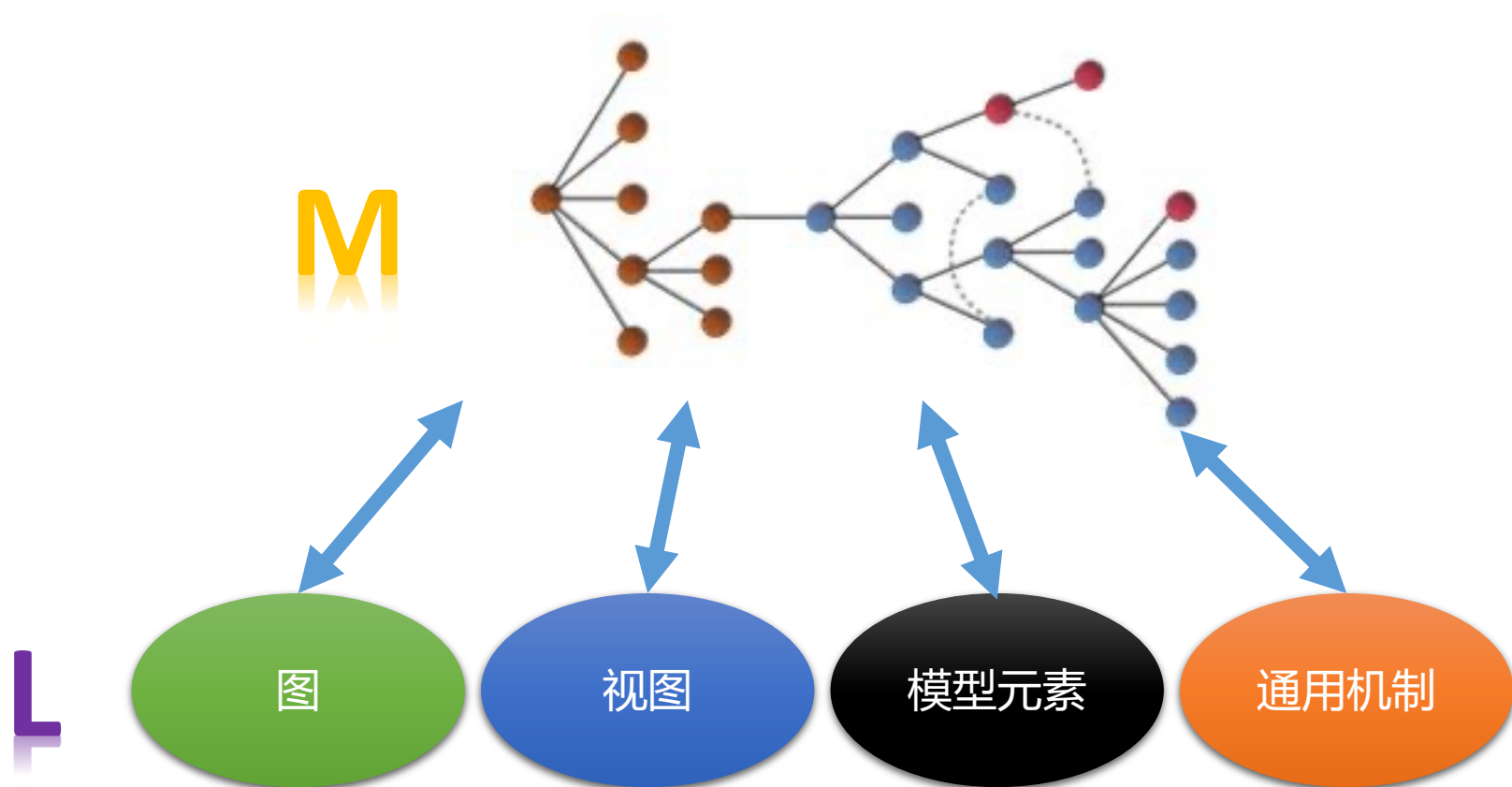
- 自然语言、JML、Java都可以描述程序系统
 - 自然语言不提供专用于描述结构和行为的成分
 - 需要大量的脑力来从中识别和理解结构与行为，容易产生歧义
 - JML可以描述规格层次的结构和行为，整合的方式
 - 需要一定的脑力来分离结构和行为，且形式化描述不直观
 - Java可以描述实现层次的结构和行为，整合的方式
 - 需要相当的脑力从中分离结构和行为，并逐步建立抽象层次
- 需要一种语言提供针对性、分离的结构与行为描述手段，而且可以将描述元素**整合**起来
- UML就是这样的语言

UML语言简介

- UML的语言设计目标
- UML的建模理念
- UML模型组成

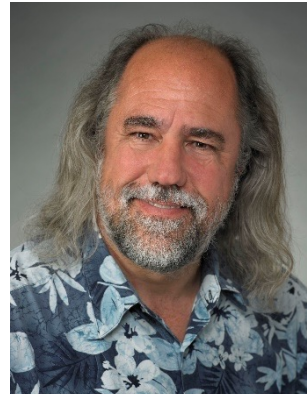


U : 建模语言集大成者



UML语言设计目标

- 面向对象式、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经过了近三十年的发展
 - 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



Grady Booch



Allan Rumbaugh



Ivar Jacobson

UML建模理念

- 语法明确、语义清晰的可视化图示语言
- 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模型组成

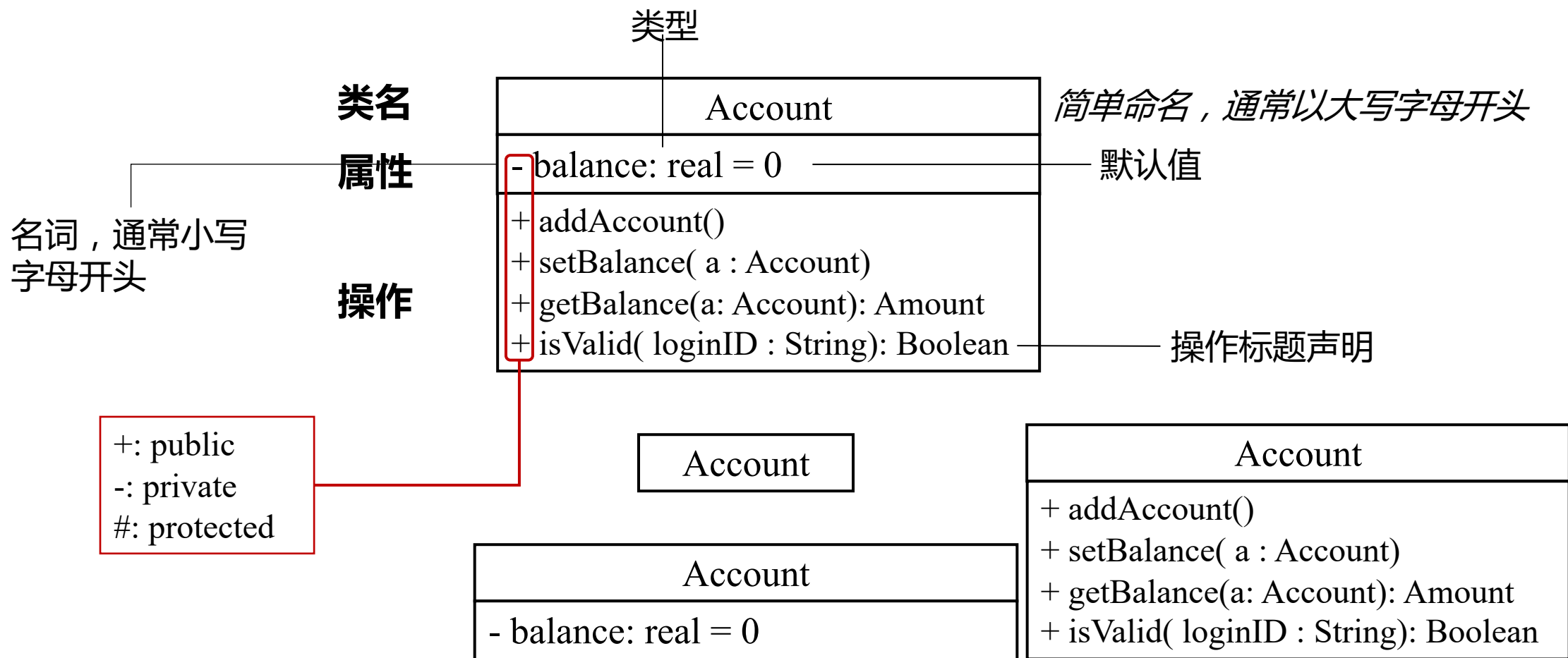
- UML是一个纯OO式的语言
 - 图中的每个要素都是一个实例对象
 - 对象类型：用例、类、属性、操作、关联、继承、消息、迁移...
 - 所有图中的**所有对象**在一起称为用户所建立的**UML模型**
- 这些对象类型由一组**数据结构**来规范定义：**UML元模型**
- UML模型实际上就是UML元模型的实例化结果
 - 所有对象共同形成了复杂的**图结构**：管理层次关系、抽象层次关系

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;  
    ...  
}
```



```
public class Student{  
    course != null && course.size() >= 1  
    private Vector<Course> course;  
    ...  
}
```

类之间的继承关系

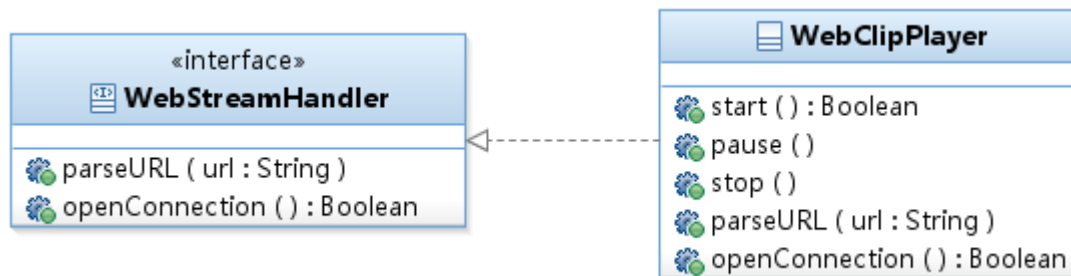
- 父类与子类
 - 父类概括子类
 - 子类扩展父类
- UML支持灵活的多继承
 - Java不支持
 - 建议在使用UML时不用多继承
- 一旦建立继承关系，子类将自动拥有父类的所有属性和操作
 - 设计层次和实现层次
 - Note：不要在子类中重复定义父类已经定义的内容

```
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！

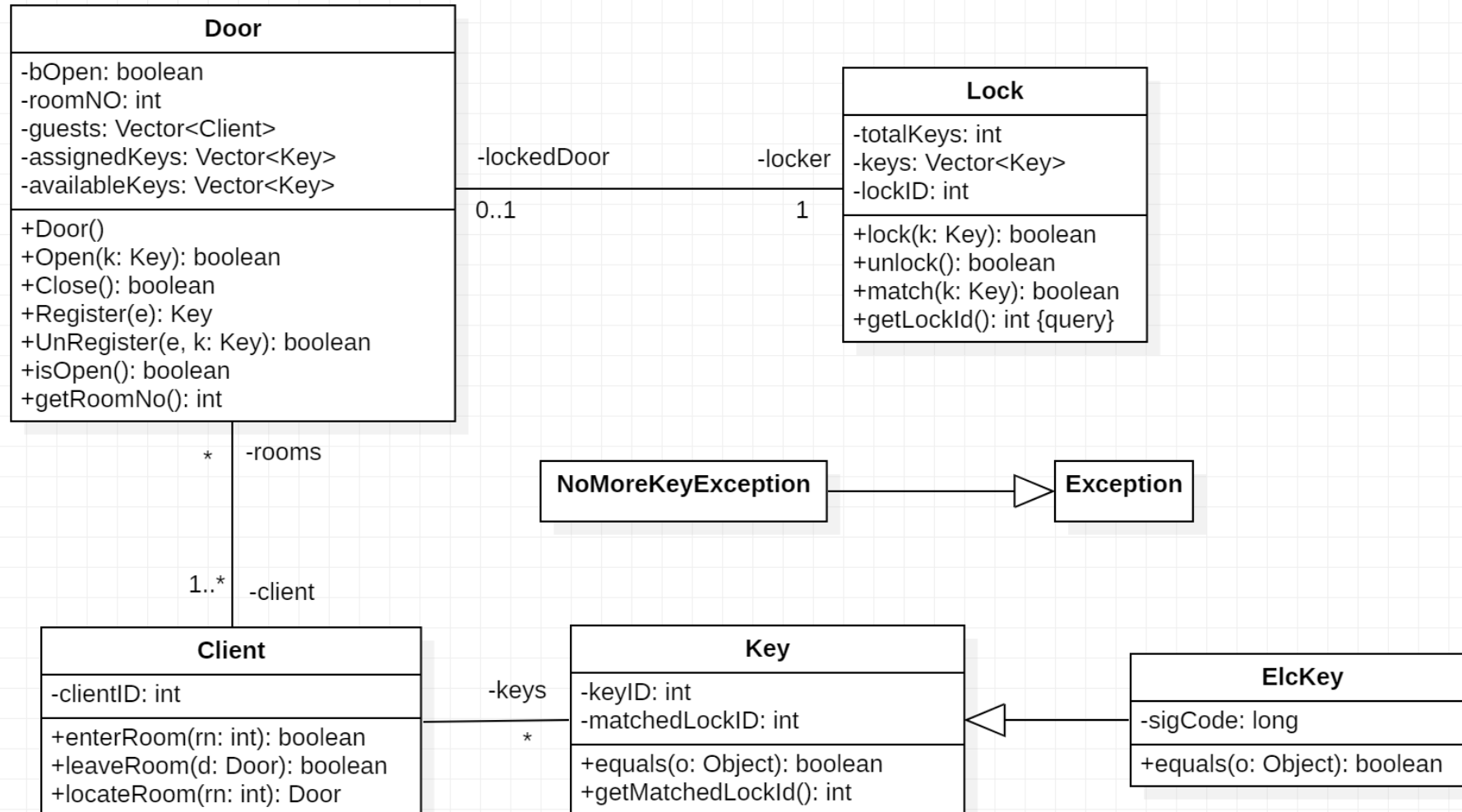
```
public class A extends B implements C,D,E{  
    ...  
    ...  
}
```



案例分析：门禁管理

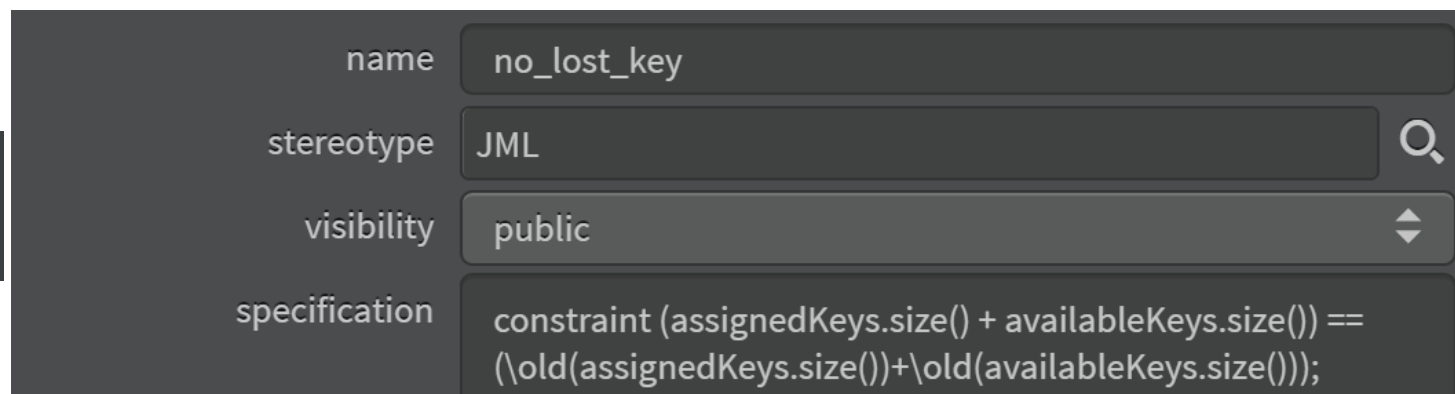
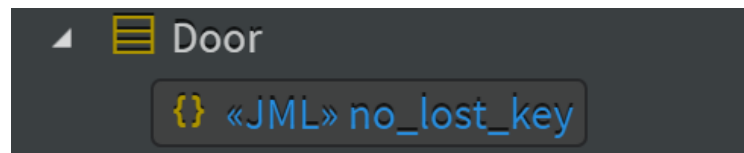
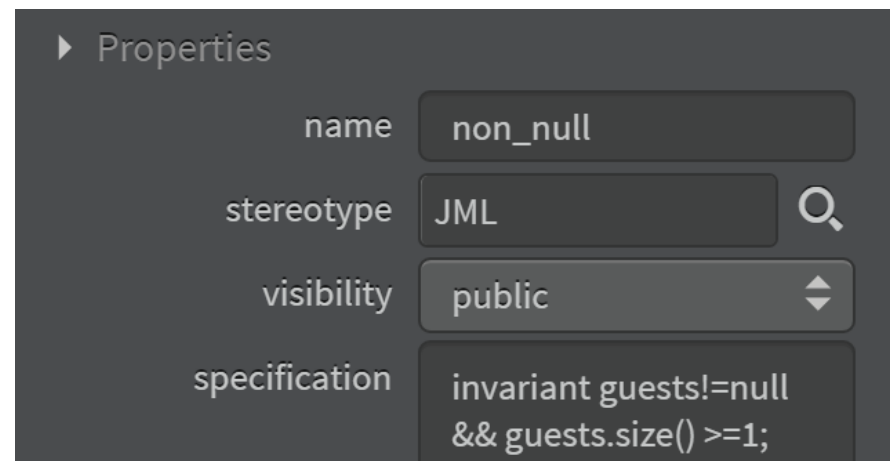
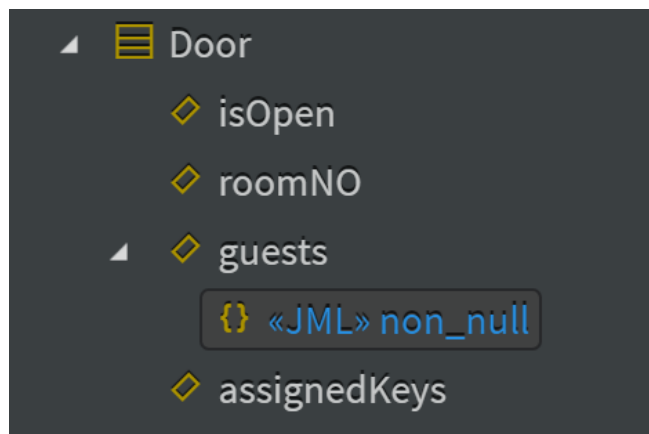
- 背景：一座楼里有多个房间，每个房间设置了1到多个门，每个门配有门禁锁，每个锁配有若干把钥匙。
- 请分析如下三段话的逻辑，使用UML类图加以描述
 - 每个门禁锁都提供上锁和开锁功能，一个门禁锁配置有数量固定的钥匙，可以是传统的机械钥匙，也可以电子式钥匙。
 - 门提供了开门、关门和判断门是否已开的功能。一旦在相应的门注册某个客户，就会自动为客户分配一把匹配的钥匙，从而可以进入房间。也可以取消注册以收回钥匙。
 - 客户可以注册多个房间，并管理所分配的钥匙。客户可以尝试进入或离开一个指定房间。

Case Study: Open&Close



可以通过Property来描述数据规格

Door
-isOpen: boolean
-roomNO: int
-guests: Vector<Client>
-assignedKeys: Vector<Key>
-availableKeys: Vector<Key>
+Door()
+Open(Key k): boolean
+Close(Key k): boolean
+Register(Client e): Key
+UnRegister(Client e, Key k): boolean



可以通过Property来描述方法规格

Door

-isOpen: boolean
-roomNO: int
-guests: Vector<Client>
-assignedKeys: Vector<Key>
-availableKeys: Vector<Key>

+Door()
+Open(Key k): boolean
+Close(Key k): boolean
+Register(Client e): Key
+UnRegister(Client e, Key k): boolean

Select One

Select kind of Constraint of Operation

☐ Typical Constraint

☒ Precondition

☐ BodyCondition

☐ Postcondition

name

matched key

stereotype

JML

visibility

public

specification

requires k!=null && locker.match(k);

Open

{ } «JML» matched key <@preconditions>

{ } «JML» door opened <@postconditions>

(Parameter)

k

name

door opened

stereotype

JML

visibility

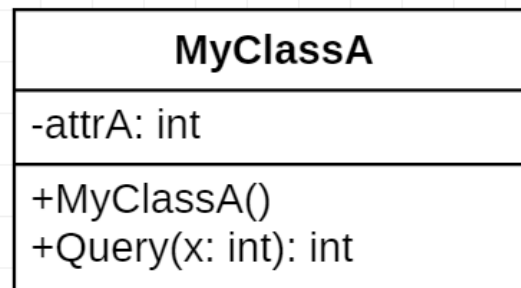
public

specification

ensures \result == true && isOpen == true;

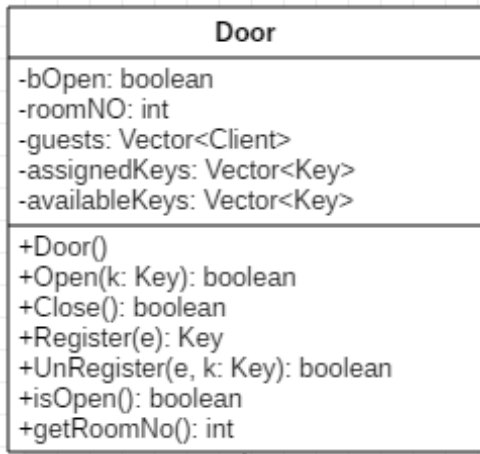
UML模型的存储结构是一棵树

- 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
- UML图中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



```
▼ ownedElements [1]
  ▼ 0 {5}
    _type : UMLModel
    _id : AAAAAAFF+qBWK6M3Z8Y=
    ► _parent {1}
      name : Model
  ▼ ownedElements [2]
    ▼ 0 {6}
      _type : UMLClassDiagram
      _id : AAAAAAFF+qBtyKM79qY=
      ► _parent {1}
        name : Main
        defaultDiagram : ☒ true
        ► ownedViews [1]
    ▼ 1 {6}
      _type : UMLClass
      _id : AAAAAAFqpiMge7NXBnk=
      ► _parent {1}
        name : MyClassA
      ► attributes [1]
      ► operations [2]
```

在UML模型层次来理解类



UMLModelElement property:

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**
 - **UMLAttribute** is a kind of **UMLModelElement**
 - bOpen is a member of Door
- Open是Door的一个操作
 - UML: Open is an object of **UMLOperation**
 - Door is a container object (typed as UMLClass) of 5 attribute objects (typed as UMLAttribute), and 7 operation objects (typed as UMLOperation).

```
_type : UMLClass
_id : AAAAAAFqpiMge7NXBnk=
▶ _parent {1}
  name : Door
▶ ownedElements [5]
▼ attributes [5]
  ▼ 0 {6}
    _type : UMLAttribute
    _id : AAAAAAFqpiN8GLOssfo=
    ▼ _parent {1}
      $ref : AAAAAAFqpiMge7NXBnk=
      name : bOpen
      visibility : private
      type : boolean
  ▶ 1 {6}
  ▶ 2 {7}
  ▶ 3 {6}
  ▶ 4 {6}
▼ operations [7]
  ▶ 0 {4}
  ▼ 1 {7}
    _type : UMLOperation
    _id : AAAAAAFqpiRcY707pzM=
    ▼ _parent {1}
      $ref : AAAAAAFqpiMge7NXBnk=
      name : Open
```

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 &&
```

```
_type : UMLClass
```

```
_id : AAAAAAFqpiMge7NXBnk=
```

- ▶ _parent {1}
- name : Door
- ▶ ownedElements [5]
- ▶ attributes [5]
- ▶ operations [7]

• UMLOperation对象

- name
- return type
- parameters
- raisedExceptions
- specification

• UMLParameter对象

- name
- type
- direction
 - in, inout, out, return
- defaultValue

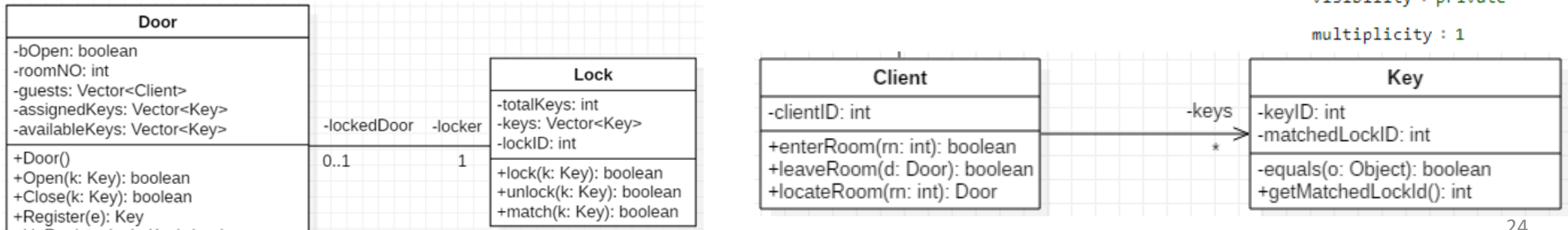
```
_type : UMLOperation
_id : AAAAAAFqpiRcY707pzM=
_parent {1}
  $ref : AAAAAAFqpiMge7NXBnk=
  name : Open
_parameters [2]
  0 {5}
    _type : UMLParameter
    _id : AAAAAAFqpiM3MbPYrBA=
    _parent {1}
      type : boolean
      direction : return
  1 {5}
    _type : UMLParameter
    _id : AAAAAAFqpz3cy1dqvuQ=
    _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}

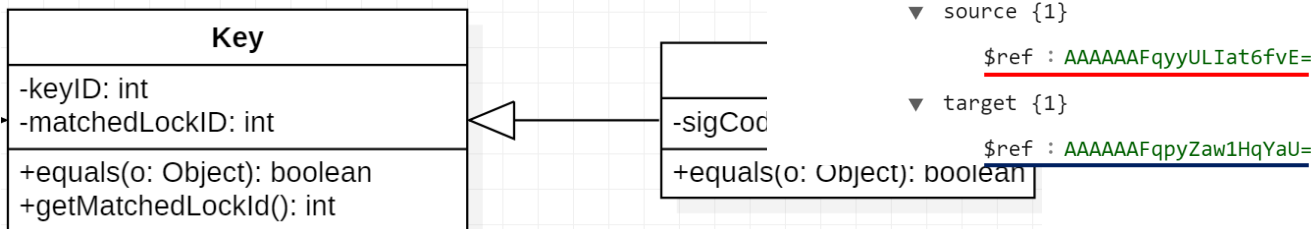


```
_type : UMLAssociation
_id : AAAAAAFqpyLHQ1A/uHQ=
▶ _parent {1}
▼ end1 {7}
    _type : UMLAssociationEnd
    _id : AAAAAAFqpyLHQ1BA8jU=
    ▶ _parent {1}
        name : lockedDoor
    ▼ reference {1}
        $ref : AAAAAFqpiMge7NXBnk=
        visibility : private
        multiplicity : 0..1
▼ end2 {7}
    _type : UMLAssociationEnd
    _id : AAAAAAFqpyLHQ1BBCwQ=
    ▶ _parent {1}
        name : locker
    ▶ reference {1}
        visibility : private
        multiplicity : 1
```


在UML模型层次来理解类抽象层次

• 继承层次

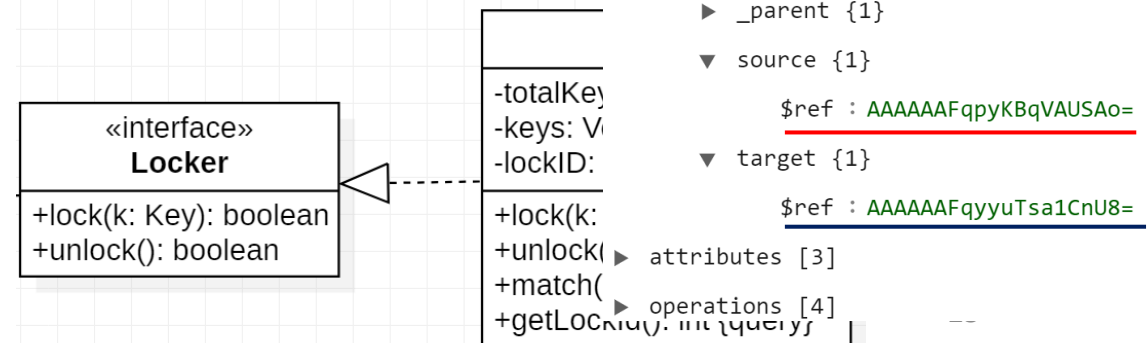
- <ElcKey, Key>: object of **UMLGeneralization**
- source: ElcKey
 - object of **UMLClass**
 - Role: subclass
- target: Key
 - object of **UMLClass**
 - Role: superclass



```
_id : AAAAAAFqpyZaw1HqYaU=  
▶ _parent {1}  
  name : Key  
▶ attributes [2]  
▶ operations [2]  
4 {5}  
5 {4}  
6 {7}  
▼ 7 {7}  
  _type : UMLClass  
  _id : AAAAAAFqyyULIat6fvE=  
▶ _parent {1}  
  name : ElcKey  
▼ ownedElements [1]  
  ▼ 0 {5}  
    _type : UMLGeneralization  
    _id : AAAAAAFqyyXW0KyhBKU=  
▶ _parent {1}  
▼ source {1}  
  $ref : AAAAAAFqyyULIat6fvE=  
▼ target {1}  
  $ref : AAAAAAFqpyZaw1HqYaU=
```

• 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



```
_type : UMLInterface  
_id : AAAAAAFqyyuTsa1CnU8=  
▶ _parent {1}  
  name : Locker  
▶ operations [2]
```

```
_type : UMLClass  
_id : AAAAAAFqpyKBqVAUSAo=  
▶ _parent {1}  
  name : Lock  
▼ ownedElements [1]  
  ▼ 0 {5}  
    _type : UMLInterfaceRealization  
    _id : AAAAAAFqyz3DUrUBj9E=  
▶ _parent {1}  
▼ source {1}  
  $ref : AAAAAAFqpyKBqVAUSAo=  
▼ target {1}  
  $ref : AAAAAAFqyyuTsa1CnU8=  
▶ attributes [3]  
▶ operations [4]
```

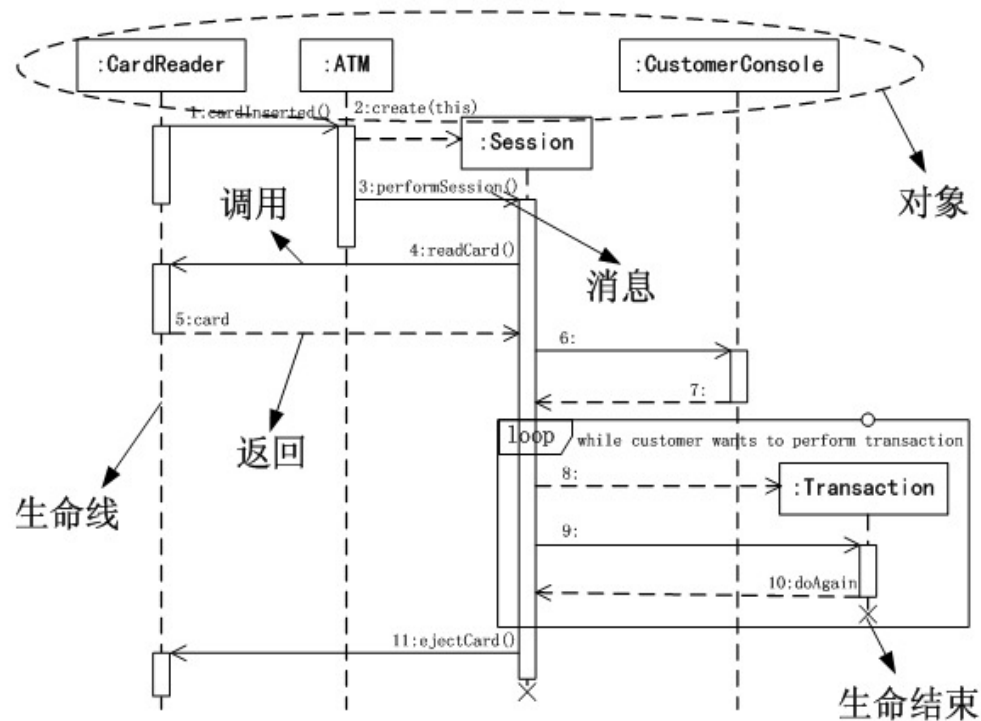
面向对象程序行为

- 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

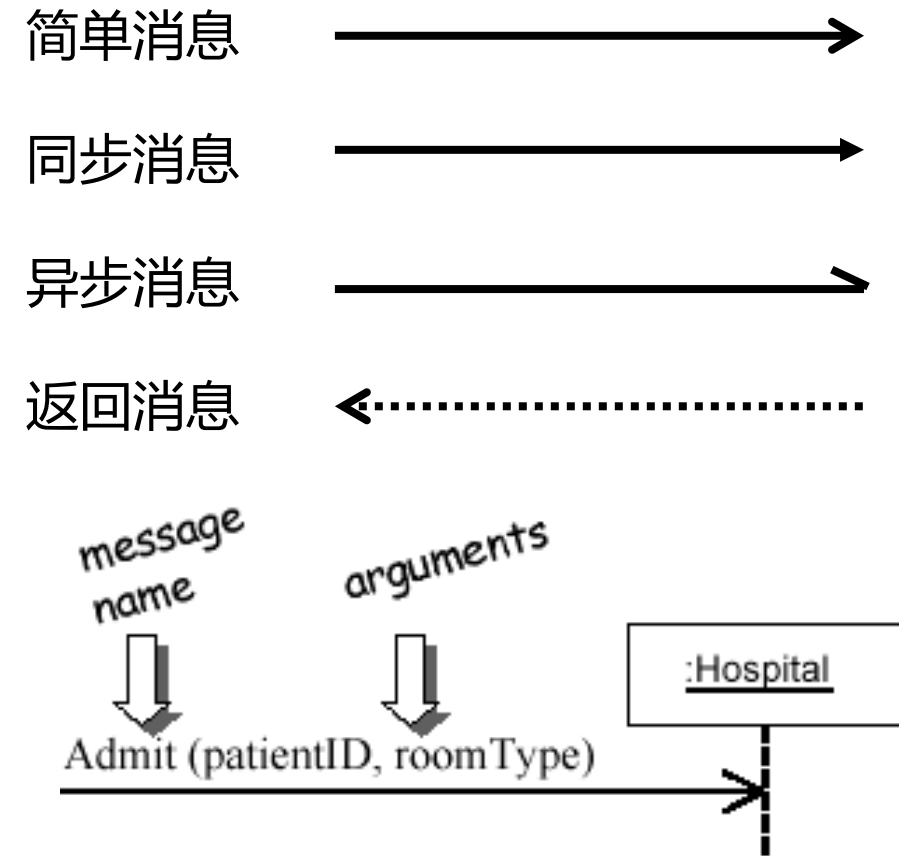
- 顺序图(sequence diagram)来自于通信领域，表示通信实体之间的通信关系
 - 参与对象(participant)：参与交互的对象
 - 消息：对象间的交互
 - 对象生命线：描述对象的存活生命期

顺序图具有典型的平面坐标系性质：
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)



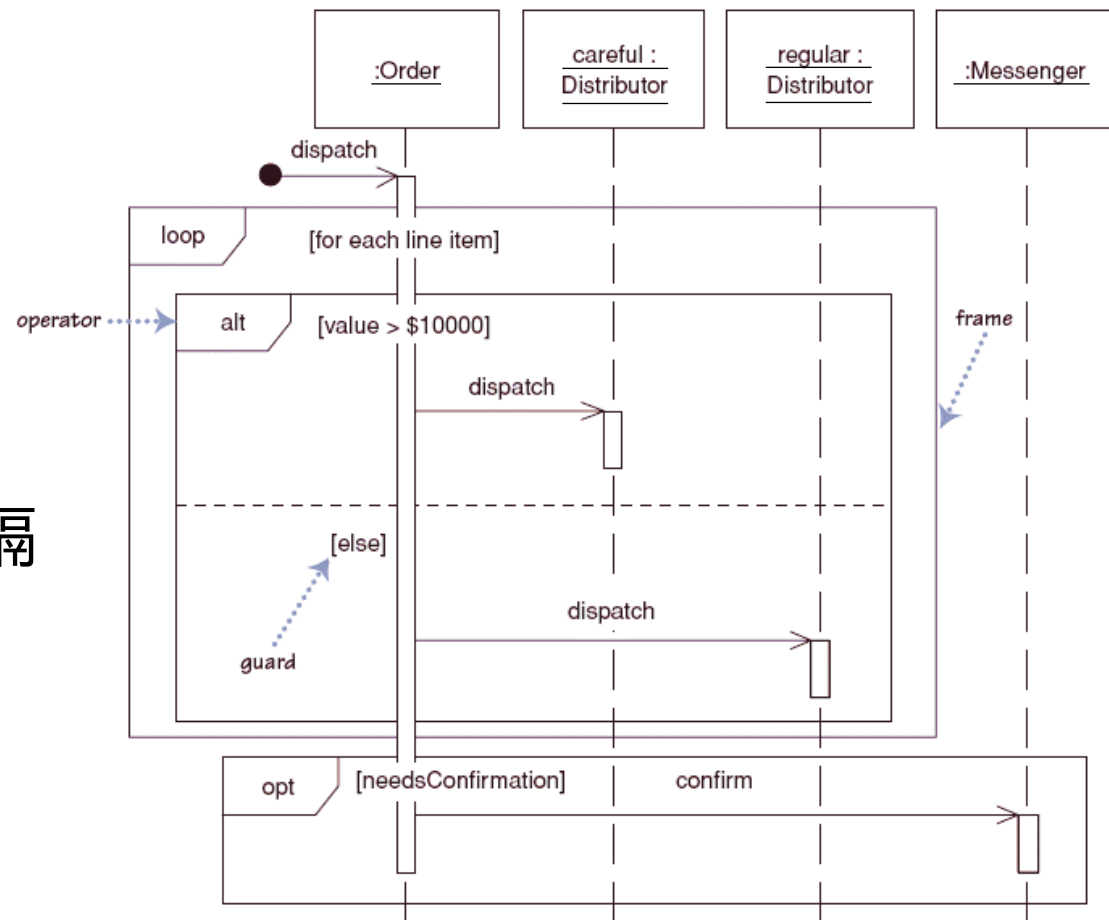
UML顺序图模型

- 对象生命线(UML Lifeline)
 - 矩形框，名称:关联的对象类型名
 - 名称或类型名有时可省略
 - 每个对象生命线都应关联到一个对象
- 消息(UML Message)
 - [guard]:[var=]消息名([消息参数])
 - 与对象生命线连接
- 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



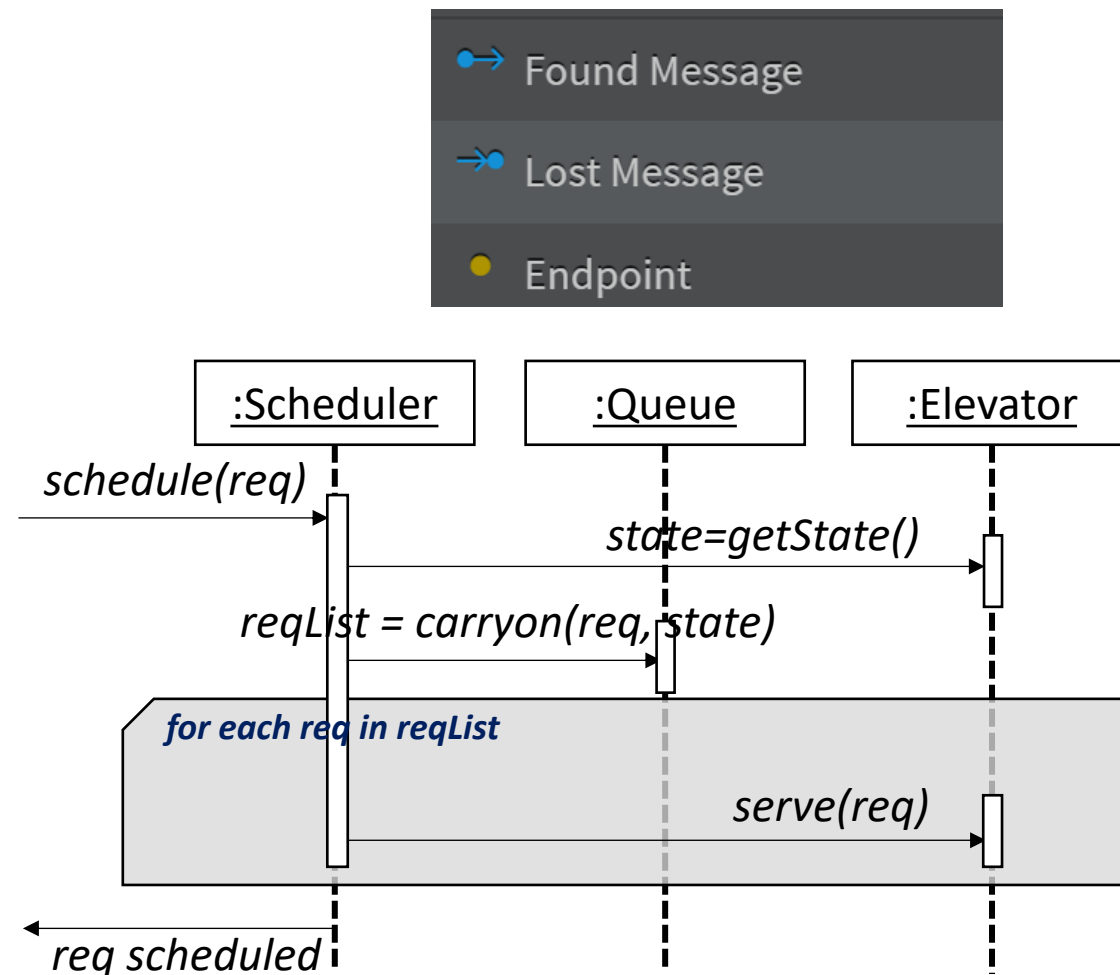
UML顺序图模型

- 基于消息块的交互流程控制
 - UMLCombinedFragment
- if控制-> 可选消息块
 - (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-> 循环消息块
 - (loop) [循环控制条件或循环事项]



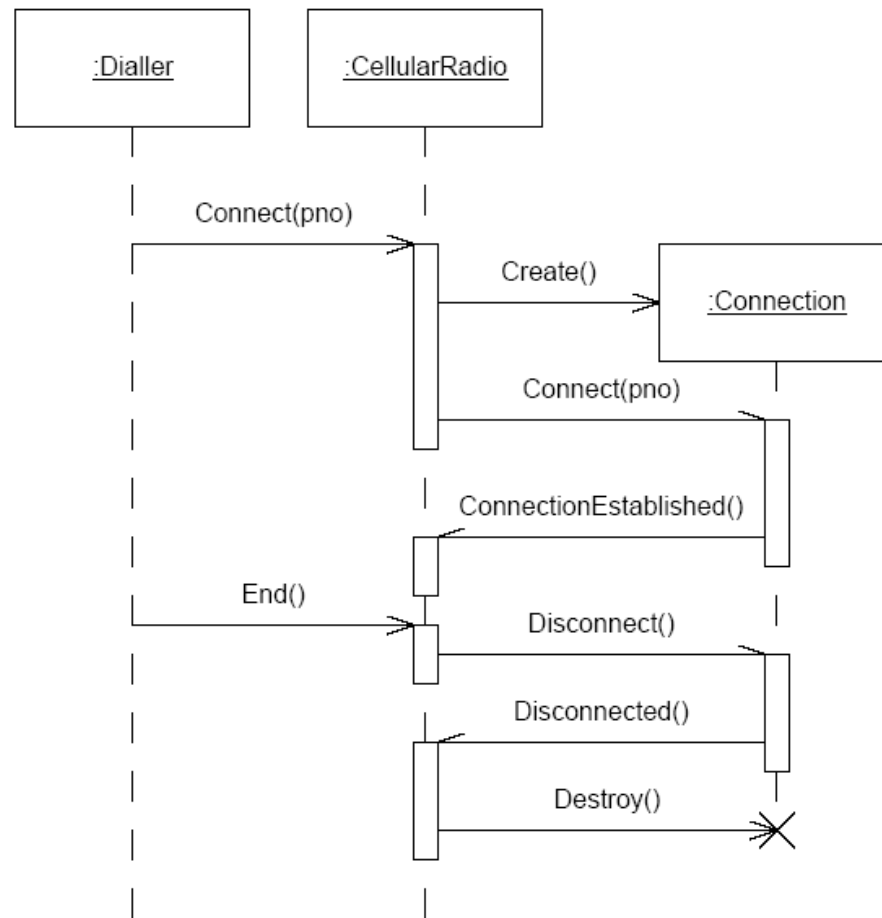
UML顺序图模型

- 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



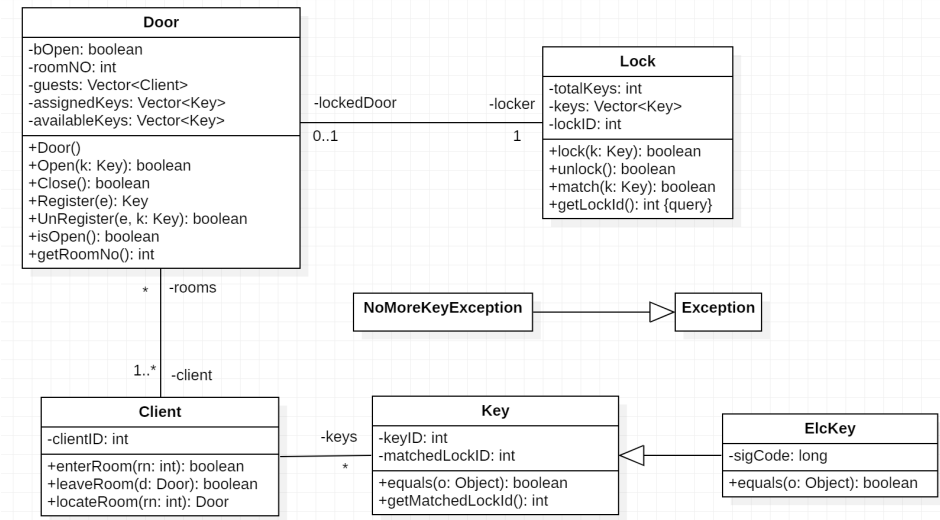
UML顺序图模型

- 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？

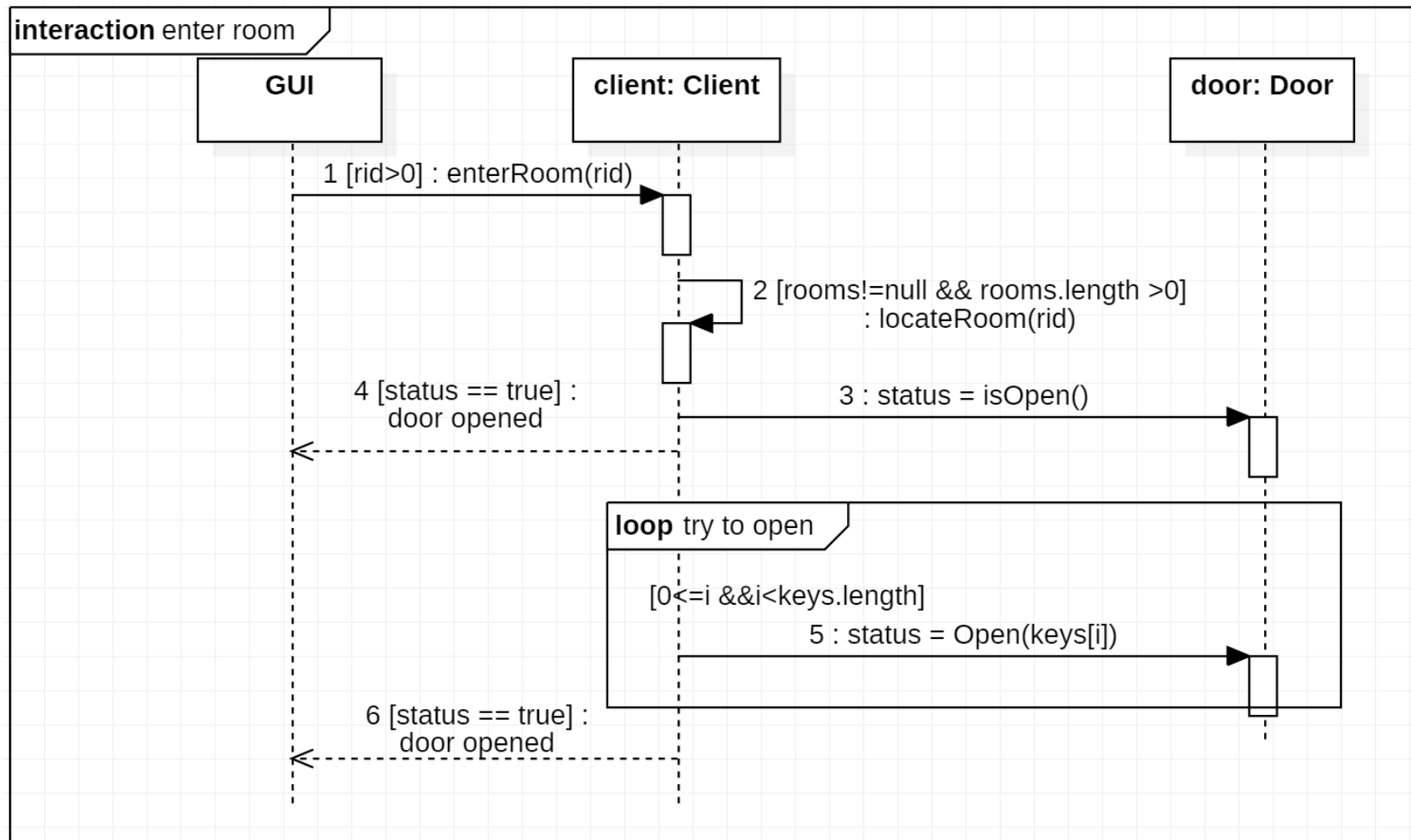


案例分析

- 请基于所建立的UML类图，使用顺序图描述如下场景：
 - 假设已有某个管理系统，用户在GUI界面提供某个房间号rid，系统能自动通过对应该客户的client对象来完成进入相应房间的行为
 - 提示1：如何通过rid找到相应的room对象，进而找到相应的door对象？
 - 提示2：如果一个door对象状态本身是开着的，还需要执行open操作吗？
 - 提示3：如何找到能打开相应door的钥匙，然后开门？



案例分析：Open&Close

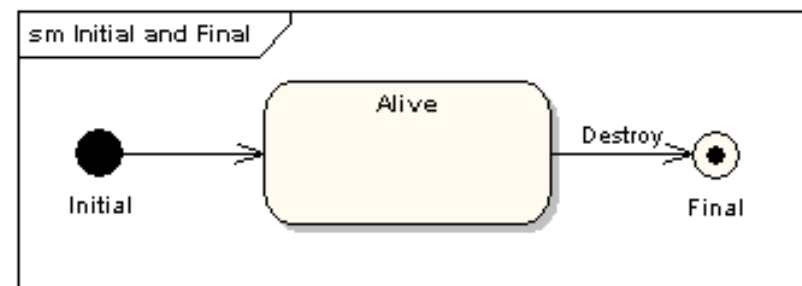
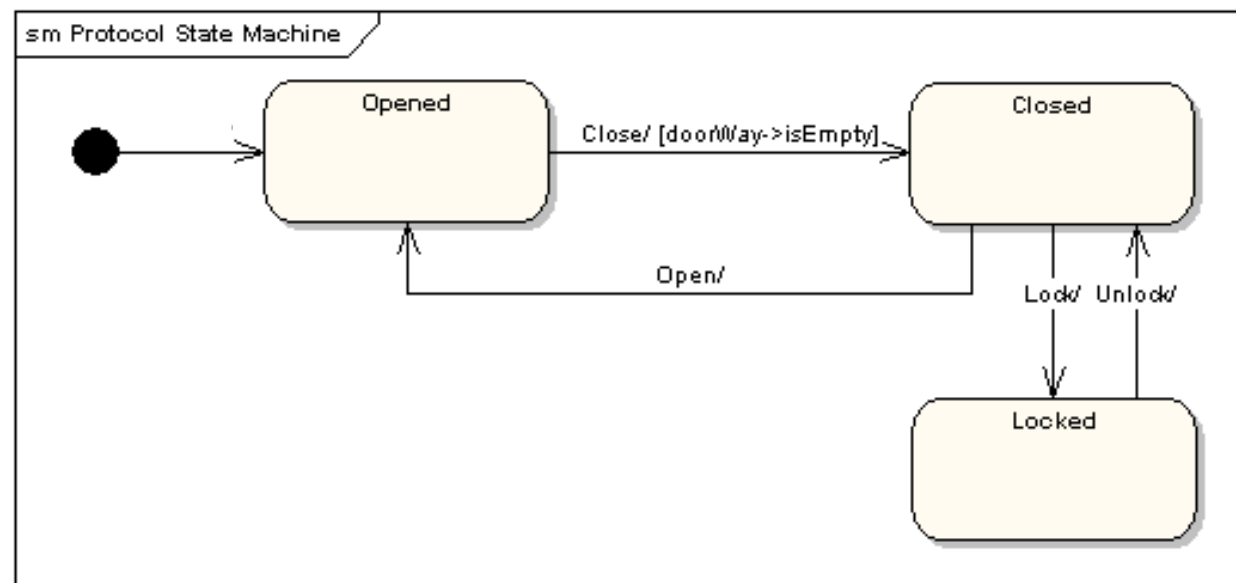


UML状态图模型

- 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**

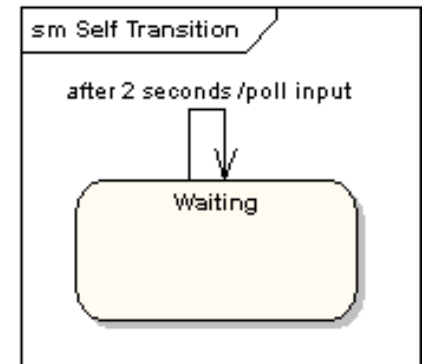
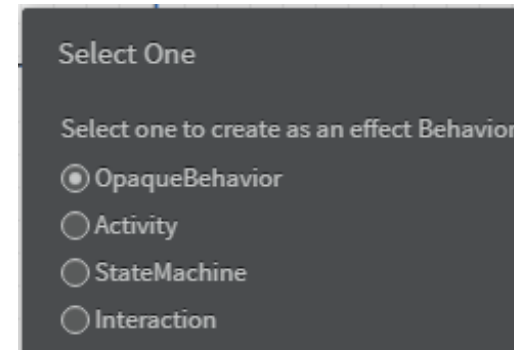
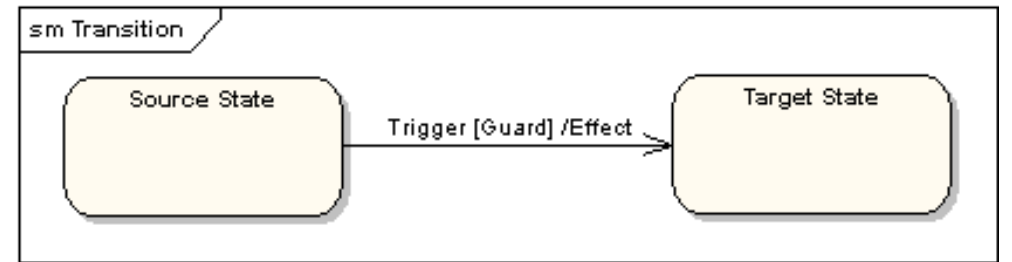
UML状态图模型

- 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)



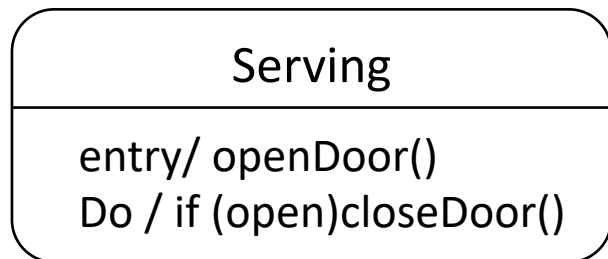
UML状态图模型

- 迁移的定义(UMLTransition)
 - Name: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
- 迁移发生的后置条件
 - Effect满足 && 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: UMLOpaqueBehavior
- 初始状态出来的迁移不能有trigger和guard



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

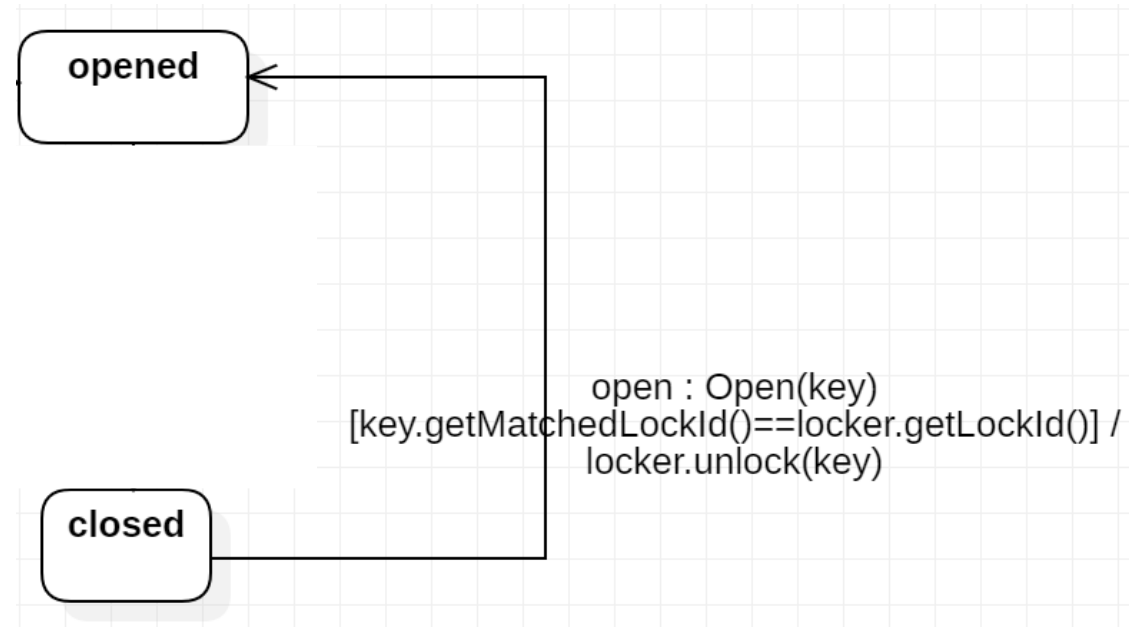
- Trigger：方法调用或者事件通知
 - Guard：调用时的相关检查

- 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

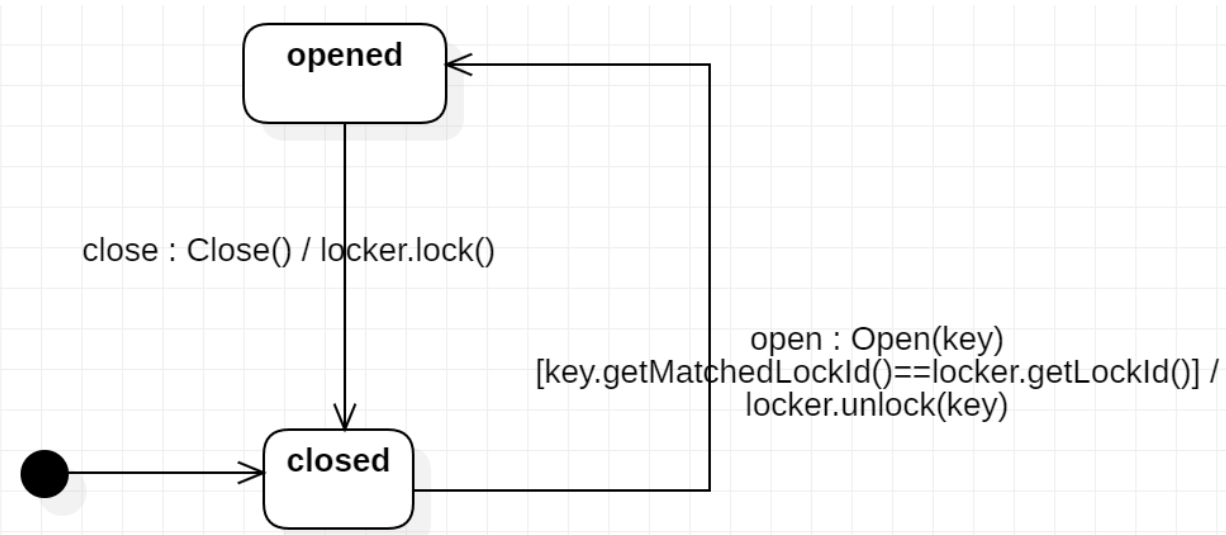
- 状态动作与程序代码的对应

- 通常外部用户不会调用，对象为了满足相关规格而实现的内部方法
 - 要求不改变对象状态



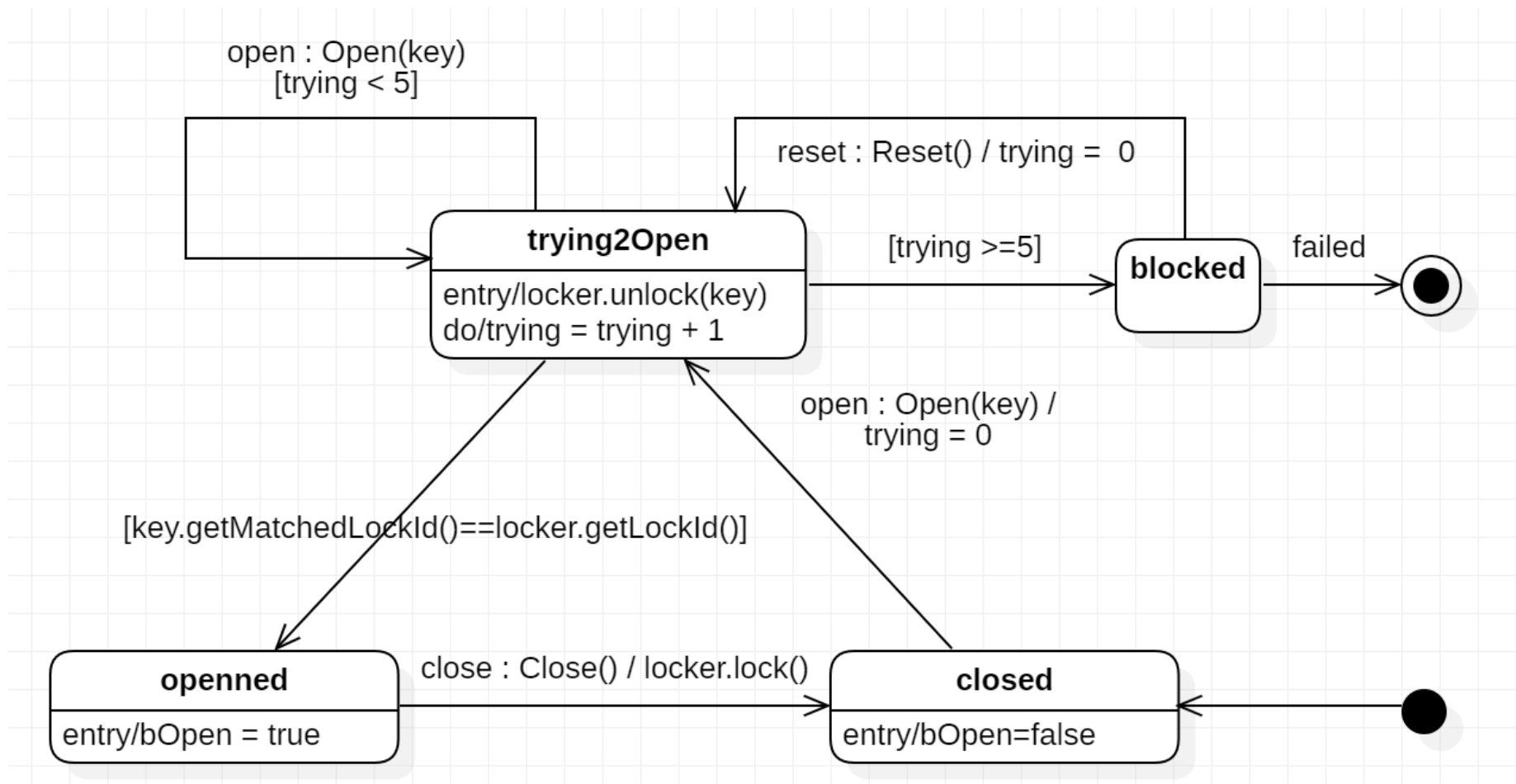
讨论：Door状态模型

- 顶层状态
 - opened, closed



1. 如果要求Door能够记录尝试开锁的次数（ Door对象构造方法把尝试次数置为0 ），如果超过5次未打开，就限制不让再开锁，直到进行重置操作，应该如何修改状态图？
2. 从opened→closed的状态迁移，是否可以直接使用bOpen==false来判断？

讨论：Door状态模型



类图评测规则

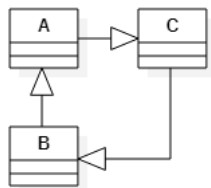
- 类图本身应符合UML规范
 - 类图中除特定元素外（具体见作业指导书），其他所有元素的**name字段**不能为空
 - 类图中不能含有重名的类
 - 类中不能有重名的属性
 - 不能有循环继承
 - 任何一个类或接口不能重复继承另外一个类或接口
 - 接口中所有方法的可见性均为public
- 类图与程序应保持一致
 - 类图与程序应能互相找到名字相同的类
 - 类图中的**继承、实现、关联**关系应和代码实现保持一致，即同时存在或同时不存在
 - 类图中的每个**类(class)**拥有的**属性**和**方法**应与程序中对应类的属性和方法一致。
 - 属性：考察它们的**名字、可见性、类型**是否一致
 - 方法：考察它们的**名字、可见性、返回值类型和参数类型**是否一致
 - 类图中所有枚举类与代码中的枚举类需要一一对应
 - 所定义的**枚举项**要求与代码中相应枚举类所定义枚举项名称相同

类图评测规则

- 对于所提交的UML类图和所提交程序
 - 设类图中的设计单元集合为MC，程序中的实现单元集合为CC
 - “单元”包括class和interface
- 对于MC中任意一个元素Ma
 - Ma中属性个数和方法个数均不少于2
 - 例外1：主类的属性个数和方法个数可以少于2
 - 例外2：接口中没有属性，故属性数量约束不适用
 - CC中一定存在一个元素Ca与Ma相对应，且Ma中属性数目不少于Ca中属性数目的60%，同时Ma中方法数目不少于Ca中方法数目的60%
- 对于CC中的任意一个元素Ca
 - MC中必然存在一个元素Ma相对应
- 对于MC中任意两个元素Ma和Mb的关系Rab
 - CC中必然有对应的两个元素Ca和Cb，且Ca和Cb同样具有相同的关系Rab，反之亦然

简单示例

①



错误：出现了循环继承

②



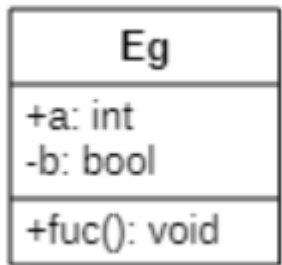
错误：出现了重名的类

③

```
{
  "_type": "UMLClass",
  "_id": "AAAAAAGILkkTWQwmRdY=",
  "_parent": {
    "$ref": "AAAAAAFF+qBWK6M3Z8Y="
  },
  "name": ""
}
```

错误：元素UMLClass的name字段为空

④

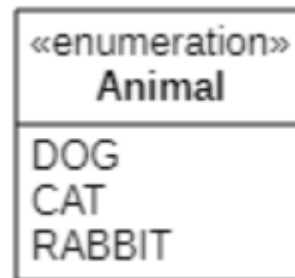


```
public class Eg {
    private int a;
    private boolean b;

    public void func() {
    }
}
```

- 错误1：属性a可见性不一致
- 错误2：属性b类型不一致
- 错误3：方法名func不一致（图中找不到方法func）

⑤



```
public enum Animal {
    DOG,
    CAT,
    RABBIT,
    TIGER
}
```

错误：类图中Animal缺少枚举项TIGER

作业分析

- 实现一个图书馆模拟系统
 - 读者、各类管理员
 - 三类借阅和预约规则不同的书
 - 基于UML的架构设计能力和正向实现能力
- 引入新的评测模式
 - 目标：引导大家练习基于模型的设计
 - UML图评测（本次作业仅检查类图）
 - 提交UML类图+代码实现