

# 面向对象UML系列第二次作业指导书

---

## 摘要

---

本次作业，在上次作业基础上，扩展解析器，使得能够支持对UML顺序图和UML状态图的解析，并能够支持几个基本规则的验证。

## 问题

---

### 基本目标

1. 扩展类图解析器，使得可以支持对UML状态图和顺序图的解析，并可以通过输入相应的指令来进行相关查询。
2. 在所解析的UML模型基础上，按照规定的规则检查模型中是否存在违背规则的情况，并输出相应信息。

### 基本任务

本次作业的程序主干逻辑（包括解析 `mdj` 格式的文件为关键数据）我们均已实现，只需要同学们完成剩下的部分，即：

- 通过实现官方提供的接口，来实现自己的UML解析器

官方的**接口定义源代码**都已在接口源代码文件中给出（暂缓开放），各位同学需要实现相应的官方接口，并保证**代码实现功能正确**。

具体来说，各位同学需要新建一个类，并实现相应的接口方法。

当然，还需要同学们在主类中调用官方包的 `AppRunner` 类，并载入自己实现的UML解析器类，来使得程序完整可运行，具体形式下文中有提示。

### 测试模式

本次作业继续**不设置互测环节**。针对本次作业提交的代码实现，课程将使用公测+bug修复的黑箱测试模式，具体测试规则参见下文。

### 类说明

---

- 本部分将暂缓开放

### 输入输出

---

本次作业将会下发 `mdj` 文件解析工具、输入输出接口（实际上为二合一的工具，接口文档会详细说明）和全局测试调用程序

- 解析工具用于将 `mdj` 格式文件解析为包含了文件内模型中所有关键信息的元素字典表
- 输入输出接口用于对元素字典表的解析和处理、对查询指令的解析和处理以及对输出信息的处理
- 全局测试调用程序会实例化同学们实现的类，并根据输入接口解析内容进行测试，并把测试结果通过输出接口进行输出

输入输出接口的具体字符格式已在接口内部定义好，各位同学可以阅读相关代码，这里我们只给出程序黑箱的字符串输入输出。

## 规则

- 输入一律在标准输入中进行，输出一律向标准输出中输出
- 输入内容以指令的形式输入，一条指令占一行，输出以提示语句的形式输出，一句输出占一行
- 输入使用官方提供的输入接口，输出使用官方提供的输出接口
- 输入的整体格式如下：
  - 由 `mdj` 文件解析而来的关键元素表
  - `END_OF_MODEL` 分隔开行
  - 指令序列，每条指令一行

## 关于类图的查询指令

~~（本段并无修改，和上次保持一致，仅有少量的补充说明以保证不存在歧义）~~

### 模型中共有多少个类

输入指令格式： `CLASS_COUNT`

举例： `CLASS_COUNT`

输出：

- `Total class count is x.` x为模型中类的总数

### 类中的操作有多少个

输入指令格式： `CLASS_OPERATION_COUNT classname mode`

举例： `CLASS_OPERATION_COUNT Elevator NON_RETURN`

输出：

- `Ok, operation count of class "classname" is x.` x为模型中参数个数（不包括return）大于等于1的操作个数
- `Failed, class "classname" not found.` 类不存在
- `Failed, duplicated class "classname".` 类存在多个

说明：

- `mode` 表示查询类型，数据类型为 `OperationQueryType`，取值为：
  - `NON_RETURN` 无返回值操作数量
  - `RETURN` 有返回值操作数量
  - `NON_PARAM` 无传入参数操作数量
  - `PARAM` 有传入参数操作数量
  - `ALL` 全部操作数量
- 此外，本指令中统计的一律为此类自己定义的操作，不包含其各级父类所定义的操作

### 类中的属性有多少个

输入指令格式: `CLASS_ATTR_COUNT classname mode`

举例: `CLASS_ATTR_COUNT Elevator SELF_ONLY`

输出:

- `Ok, attribute count of class "classname" is x.` x 为类中属性的个数
- `Failed, class "classname" not found.` 类不存在
- `Failed, duplicated class "classname".` 类存在多个

说明:

- `mode` 表示查询的模式, 数据类型为 `AttributeQueryType`, 取值为:
  - `ALL` 全部属性数量 (包括各级父类定义的属性)
  - `SELF_ONLY` 此类自身定义的属性数量

## 类有几个关联

输入指令格式: `CLASS ASSO_COUNT classname`

举例: `CLASS ASSO_COUNT Elevator`

输出:

- `Ok, association count of class "classname" is x.` x 为类关联的个数
  - 如果出现自关联行为的话, 也算在内
- `Failed, class "classname" not found.` 类不存在
- `Failed, duplicated class "classname".` 类存在多个

## 类的关联的对端是哪些类

输入指令格式: `CLASS ASSO_CLASS_LIST classname`

举例: `CLASS ASSO_CLASS_LIST Elevator`

输出:

- `Ok, associated classes of class "classname" are (A, B, C).` A、B、C为类所有关联的对端的类名, 其中
  - 传出列表时可以乱序, 官方接口会自动进行排序 (但是需要编写者自行保证不重不漏)
  - 如果出现自关联的话, 那么自身类也需要加入输出
- `Failed, class "classname" not found.` 类不存在
- `Failed, duplicated class "classname".` 类存在多个

## 类的操作可见性

输入指令格式: `CLASS OPERATION_VISIBILITY classname methodname`

举例: `CLASS OPERATION_VISIBILITY Taxi setStatus`

输出:

- `Ok, operation visibility of method "methodname" in class "classname" is public: xxx, protected: xxx, private: xxx, package-private: xxx.` 该操作的实际可见性统计

- `Failed, class "classname" not found.` 类不存在
- `Failed, duplicated class "classname".` 类存在多个

说明:

- 本指令中统计的一律为此类自己定义的操作, 不包含其各级父类所定义的操作
- 在上一条的前提下, 需要统计出全部的名称为methodname的方法的可见性信息

## 类的属性可见性

输入指令格式: `CLASS_ATTR_VISIBILITY classname attrname`

举例: `CLASS_ATTR_VISIBILITY Taxi id`

输出:

- `Ok, attribute "attrname" in class "classname"'s visibility is public/protected/private/package-private.` 该属性的实际可见性
- `Failed, class "classname" not found.` 类不存在
- `Failed, duplicated class "classname".` 类存在多个
- `Failed, attribute not found.` 类中没有该属性
- `Failed, duplicated attribute.` 类中属性存在多个同名

说明:

- 本指令的查询均需要考虑属性的继承关系。
- 其中对于父类和子类均存在此名称的属性时, 需要按照 `duplicated attribute` 处理。

## 类的顶级父类

输入指令格式: `CLASS_TOP_BASE classname`

举例: `CLASS_TOP_BASE AdvancedTaxi`

输出:

- `Ok, top base class of class "classname" is top_classname.` `top_classname` 为顶级父类
- `Failed, class "classname" not found.` 类不存在
- `Failed, duplicated class "classname".` 类存在多个

说明:

- 具体来说, 对于类 $X$ , 如果 $Y$ 为其顶级父类的话, 则满足
  - $X$ 是 $Y$ 的子类 (此处特别定义,  $X$ 也是 $X$ 的子类)
  - 不存在类 $Z$ , 使得 $Y$ 是 $Z$ 的子类

## 类实现的全部接口

输入指令格式: `CLASS_IMPLEMENT_INTERFACE_LIST classname`

举例: `CLASS_IMPLEMENT_INTERFACE_LIST Taxi`

输出:

- `Ok, implement interfaces of class "classname" are (A, B, C).` A、B、C为继承的各个接口

- 传出列表时可以乱序，官方接口会自动进行排序（但是需要编写者自行保证不重不漏）
- 特别值得注意的是，无论是直接实现还是通过父类或者接口继承等方式间接实现，都算做实现了接口
- `Failed, class "classname" not found.` 类不存在
- `Failed, duplicated class "classname".` 类存在多个

## 类是否违背信息隐藏原则

输入指令格式： `CLASS_INFO_HIDDEN classname`

举例： `CLASS_INFO_HIDDEN Taxi`

输出：

- `Yes, information of class "classname" is hidden.` 满足信息隐藏原则。
- `No, attribute xxx in xxx, xxx in xxx, are not hidden.` 不满足信息隐藏原则。
- `Failed, class "classname" not found.` 类不存在
- `Failed, duplicated class "classname".` 类存在多个

*注意：类图部分的查询命令和上次作业的完全相同，这意味着本次的公测仍然可能使用上次作业用到的公测用例，甚至增加一些新的公测用例。同学们务必针对上次作业中没有通过的测试用例，找到bug并修复，要不然可能会导致重复失分。*

说明：

- 信息隐藏原则，指的是**在类属性的定义中，不允许使用private以外的任何可见性修饰**
- 本指令中需要列出全部的非隐藏属性，同时也需要考虑继承自父类的非隐藏属性
- 值得注意的是，父类和子类中，是可以定义同名属性的（甚至还可以不同类型，不同可见性，感兴趣的话可以自己尝试尝试），然而**父类中定义的和子类中定义的实际上并不是同一个属性，需要在输出时进行分别处理**
- 同样的，返回的列表可以乱序，官方接口会进行自动排序（但是依然需要编写者保证不重不漏）

## 关于UML状态图的查询指令

### 给定状态机模型中一共有多少个状态

输入指令格式： `STATE_COUNT statemachine_name`

举例： `STATE_COUNT complex_sm`

输出：

- `Ok, state count of statemachine "complex_sm" is x.` x为应状态机模型complex\_sm的状态总数.
- `Failed, statemachine "complex_sm" not found.` 未找到状态机模型complex\_sm
- `Failed, duplicated statemachine "complex_sm".` 存在多个状态机模型complex\_sm

### 给定状态机模型中一共有多少个迁移

输入指令格式： `TRANSITION_COUNT statemachine_name`

举例： `TRANSITION_COUNT complex_sm`

输出：

- `Ok, transition count of statemachine "complex_sm" is x.` x为状态机模型complex\_sm中的迁移个数.

- Failed, statemachine "complex\_sm" not found. 未找到状态机模型complex\_sm
- Failed, duplicated statemachine "complex\_sm". 存在多个状态机模型complex\_sm

## 给定状态机模型和其中的一个状态，有多少个不同的后继状态

输入指令格式：SUBSEQUENT\_STATE\_COUNT statemachine\_name statename

举例：SUBSEQUENT\_STATE\_COUNT complex\_sm opened

输出：

- Ok, subsequent state count from state "opened" in statemachine "complex\_sm" is x. x为状态机模型complex\_sm中从opened状态可达的不同状态个数
- Failed, statemachine "complex\_sm" not found. 未找到状态机模型complex\_sm
- Failed, duplicated statemachine "complex\_sm". 存在多个状态机模型complex\_sm
- Failed, state "opened" in statemachine "complex\_sm" not found. 在状态机模型complex\_sm中未找到状态opened
- Failed, duplicated state "opened" in statemachine "complex\_sm". 在状态机模型complex\_sm中存在多个opened状态

说明：

- 本次作业给定的状态机模型中不包含复合状态

## 关于UML顺序图的查询指令

### 给定UML顺序图，一共有多少个参与对象

输入指令格式：PTCP\_OBJ\_COUNT umlinteraction\_name

举例：PTCP\_OBJ\_COUNT normal

输出：

- Ok, participant count of umlinteraction "normal" is x. x为顺序图模型normal (UMLInteraction) 中的参与对象个数 (UMLLifetime)
- Failed, umlinteraction "normal" not found. 不存在normal这个顺序图模型
- Failed, duplicated umlinteraction "normal". 存在多个normal顺序图模型

### 给定UML顺序图，一共有多少个交互消息

输入指令格式：MESSAGE\_COUNT umlinteraction\_name

举例：MESSAGE\_COUNT normal

输出：

- Ok, message count of umlinteraction "normal" is x. x为顺序图模型normal (UMLInteraction) 中的消息个数 (UMLMessage, 不考虑消息内容是否相同)
- Failed, umlinteraction "normal" not found. 不存在normal这个顺序图模型
- Failed, duplicated umlinteraction "normal". 存在多个normal顺序图模型

### 给定UML顺序图和参与对象，有多少个incoming消息

输入指令格式：INCOMING\_MSG\_COUNT umlinteraction\_name lifeline\_name

举例: `INCOMING_MSG_COUNT normal door`

输出:

- `Ok, incoming message count of lifeline "door" in umlinteraction "normal" is x.` x为顺序图模型normal (UMLInteraction) 中发送给door的消息个数
- `Failed, umlinteraction "normal" not found.` 不存在normal这个顺序图模型
- `Failed, duplicated umlinteraction "normal".` 存在多个normal顺序图模型
- `Failed, lifeline "door" in umlinteraction "normal" not found.` 在顺序图模型normal中未找到参与对象door
- `Failed, duplicated lifeline "door" in umlinteraction "normal".` 在顺序图模型normal中存在多个door参与对象

注意:

- 这里的UMLInteraction指UML所定义的一个类型, 见第14讲ppt的第16页

## 模型有效性检查

模型有效性检查部分, 将在**实例化完毕后自动按序触发执行**, 不通过指令的形式。且一旦**发现不符合规则的情况**, 将**直接退出, 不进行后续有效性检查和指令查询**。

### R001: 针对下面给定的模型元素容器, 不能含有重名的成员(UML002)

- 规则解释:
  - 针对类图中的类 (UMLClass), 其成员属性 (UMLAttribute) 和关联对端所连接的UMLAssociationEnd不能有重名
- 输出:
  - `Ok, model is fine with R001.` 未发现相关的重名
  - `Failed when check R001, "member" in "Container", "member2" in "AnotherContainer" has duplicate name.` 发现重名
- 说明:
  - 如果模型中有多个模型元素违背R001, 则依次输出, 次序不敏感, 接口会在输出前进行排序。

### R002: 不能有循环继承(UML008)

- 规则解释:
  - 该规则只考虑类的继承关系、类和接口之间实现关系, 以及接口之间的继承关系。所谓循环继承, 就是按照继承关系形成了环。
  - 例如下面的场景

```
1 interface A extends B {  
2     // something here  
3 }  
4  
5 interface B extends A {  
6     // something here  
7 }
```

这里就构成了一组最简单的循环继承。

- 输出：
  - `Ok, model is fine with R002.` 未发现重复继承
  - `Failed when check R002, class/interface (A, B, C, D) have circular inheritance.` 列出所有在循环继承链中的类或接口名
- 说明：
  - 输出的集合中需要包含全部继承环上的类、接口。
  - 对于同一个类、接口，只需要输出一次即可。

### R003: 任何一个类或接口不能重复继承另外一个接口(UML009)

- 规则解释：
  - 该规则考虑类之间的继承关系、接口之间的继承关系，以及类对接口的实现关系，包括直接继承或间接继承。
  - 例如下面的场景

```
1 interface A {
2     // something here
3 }
4
5 interface B extends A {
6     // something here
7 }
8
9 interface C extends A, B {
10    // something here
11 }
12
13 class D implements C {
14    // something here
15 }
```

接口C就重复继承了接口A（一次直接继承，一次通过接口B间接继承），同时类D也重复继承了接口A（通过接口C间接继承）

- 输出：
  - `Ok, model is fine with R003.` 未发现重复继承
  - `Failed when check R003, class/interface (A, B, C, D) have duplicate inheritance.` 列出所有带有重复继承的类名
- 说明：
  - 如果存在多个直接或间接重复继承了其他的类或接口的类或接口，则按照任意顺序传出即可，次序不敏感，接口会在输出前进行排序。
  - 值得注意的是，**本次作业的本条限制，同样也禁止了接口的重复继承**。然而接口重复继承在Java 8中实际上是允许的，也就是说，这是UML本身的一条合法性规则，无关语言。请各位判断的时候务必注意这件事。

## 关于判定

### 公测（包括弱测、中测与强测）数据基本限制



- **mdj 文件内容限制**
  - 包含类图，类图在 `UMLModel` 内进行建模，且每个 `UMLModel` 内的元素不会引用当前 `UMLModel` 以外的元素（即关系是一个闭包）
  - 包含顺序图，与 `UMLModel` 同级，可能会引用到 `UMLModel` 中的模型元素
  - 包含状态图，一定处于 `UMLClass` 下面的层次，不会引用 `UMLModel` 中的其他模型元素
  - **原始mdj文件仅通过staruml工具建模生成**（不存在手改json等行为）
  - 原始mdj文件符合 `starUML` 规范，可在 `starUML` 中正常打开和显示
  - `mdj` 文件中最多只包含 400 个元素
  - **此外为了方便本次的情况处理，除了所列出的三个规则，保证所建模的模型均可以在不与前面所述的规定相矛盾的情况下，在Oracle Java 8中正常实现出来**
- 输入指令限制
  - 最多不超过300条指令
  - 输入指令满足标准格式
- 测试数据限制
  - 所有公测数据不会对
    - 接口中定义的属性
    - 类属性（static attribute）
    - 类方法（static method）
    - 做任何测试要求，本次作业不需要对这些情况进行考虑。
- 我们保证，公测中的所有数据均满足以上基本限制。

## 测试模式

公测均通过标准输出输出进行。

指令将会通过查询UML各种信息的正确性，从而测试UML解析器各个接口的实现正确性。

对于任何满足基本数据限制的输入，程序都应该保证不会异常退出，如果出现问题则视为未通过该测试点。

程序运行的最大CPU时间为 10s，保证强测数据有一定梯度。

## 提示&说明

- 如果还有人不知道标准输入、标准输出是啥的话，那在这里解释一下
  - 标准输入，直观来说就是屏幕输入
  - 标准输出，直观来说就是屏幕输出
  - 标准异常，直观来说就是报错的时候那堆红字
  - 想更加详细的了解的话，请去百度
- 关于上次作业中发现的一些问题，在此进行统一的补充说明：
  - 对于基于类的查询，**除非明确表示查询类与接口，否则一律只针对类(UMLClass)进行查询。**
  - 对于所有的异常抛出，应该这样去思考：
    - 通过读代码，搞明白相关异常是什么样的意义
    - 通过读代码，搞明白抛出去的异常会被以怎么样的形式进行使用
    - 比如，十分显然的
      - **对于Duplicated一类的异常**，表示且仅表示当根据仅有的输入无法唯一确定需要查询的对象时（即符合条件的超过1个），所需要抛出的异常

- **对于NotFound一类的异常**，表示且仅表示当根据仅有的输入无法找到需要查询的对象时（即符合条件的为0个），所需要抛出的异常
- 以及，异常中所需要传入的类名之类的值，是用来输出的。**所以查询的输入值是什么，就传入什么，以保证和输入信息的对应性。**
  - 关于关联的统计方式，请参见上一次作业讨论区老师发的暖心贴。
- 本次作业中可以自行组织工程结构。任意新增 java 代码文件。只需要保证 UmlInteraction 类的继承与实现即可。
- **关于本次作业解析器类的设计具体细节，本指导书中均不会进行过多描述，请自行去官方包开源仓库中查看接口的规格，并依据规格进行功能的具体实现，必要时也可以查看AppRunner的代码实现。关于官方包的使用方法，可以去查看开源库的 README.md。**
- 开源库地址：[已开放，传送门](#)。现已开放
  - 在官方包发布之前，同学们可以自行构建简单的输出接口进行测试，**或者使用Junit进行单元测试（推荐，单元测试实际上非常适合这种场合的测试）。**
- 推荐各位同学在课下测试时使用Junit单元测试来对自己的程序进行测试
  - Junit是一个单元测试包，**可以通过编写单元测试类和方法，来实现对类和方法实现正确性的快速检查和测试。**还可以查看测试覆盖率以及具体覆盖范围（精确到语句级别），以帮助编程者全面无死角的进行程序功能测试。
  - Junit已在评测机中部署（版本为Junit4.12，一般情况下确保为Junit4即可），所以项目中可以直接包含单元测试类，在评测机上不会有编译问题。
  - 此外，Junit对主流Java IDE（Idea、eclipse等）均有较为完善的支持，可以自行安装相关插件。推荐两篇博客：
    - [Idea下配置Junit](#)
    - [Idea下Junit的简单使用](#)
  - 感兴趣的同学可以自行进行更深入的探索，百度关键字：`Java Junit`。
- 强烈推荐同学们
  - 去阅读本次的源代码
  - **去好好复习下本次和上次的ppt，并理清各个 UmlElement 数据模型的结构与关系。**
- **不要试图通过反射机制来对官方接口进行操作，我们有办法进行筛查。**此外，如果发现有人试图通过反射等手段hack输出接口的话，请邮件[niuyazhe@buaa.edu.cn](mailto:niuyazhe@buaa.edu.cn)进行举报，**经核实后，将直接作为无效作业处理。**