

面向对象UML系列第三次作业指导书

注意

请不要提交官方包代码！！！其代码仅供参考，请使用已经编译打包好的jar文件。

因提交官方包源代码导致的任何问题，请自行承担后果！！

摘要

本次作业，在上次作业基础上，对模型进行**有效性检查**。

问题

基本目标

扩展类图解析器，使得可以支持对UML状态图和顺序图的分析，可以通过输入相应的指令来进行相关查询。

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）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分析器**

官方的**接口定义源代码**都已在接口源代码文件中给出，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，具体形式参考[官方包目录](#)下的README.md。

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。

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)

各个指令对应的方法名和参数的表示方法详见官方接口说明。

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

举例：CLASS_OPERATION_COUNT Elevator

输出：

- Ok, operation count of class "classname" is x. x为类中的操作个数
- Failed, class "classname" not found. 类不存在
- Failed, duplicated class "classname". 类存在多个

说明：

- 本指令中统计的一律为此类自己定义的操作，不包含继承自其各级父类所定义的操作
- 本指令不检查操作的合法性，所有操作均计入总数。如有重复操作分别计入总数。

类中的属性有多少个

输入指令格式：CLASS_ATTR_COUNT classname

举例：CLASS_ATTR_COUNT Elevator

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：

- 本指令的查询均需要考虑属性的继承关系，即需包括各级父类定义的属性。
- 不需要考虑所实现的接口的属性。

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
- 如果不存在对应的方法, 则全部置0

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 "attrname" not found in class "classname".` 类中没有该属性
- `Failed, duplicated attribute "attrname" in class "classname".` 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。
- 不需要考虑所实现的接口的属性。

类的属性类型

输入指令格式: `CLASS_ATTR_TYPE classname attrname`

举例: `CLASS_ATTR_TYPE Taxi id`

输出:

- `Ok, the type of attribute "attrname" in class "classname" is typeA.` 该属性的类型
- `Failed, class "classname" not found.` 类不存在
- `Failed, duplicated class "classname".` 类存在多个
- `Failed, attribute "attrname" not found in class "classname".` 类中没有该属性
- `Failed, duplicated attribute "attrname" in class "classname".` 类中属性存在多个同名
- `Failed, wrong type of attribute "attrname" in class "classname".` 属性类型错误

说明:

- 类型 `type` 的数据类型有两种，分别为 `ReferenceType` 和 `NamedType`。对于这两种情形，
 - `NamedType` 为命名型类别，只考虑 JAVA 语言八大基本类型(`byte`, `short`, `int`, `long`, `float`, `double`, `char`, `boolean`)和 `String`，其余一律视为错误类型。
 - `ReferenceType` 为依赖型类别，其指向已定义的类或接口，类型名为对应的类名或接口名。
- 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。
- 不需要考虑所实现的接口的属性。

类的操作的参数类型

输入指令格式：`CLASS_OPERATION_PARAM_TYPE classname methodname`

举例：`CLASS_OPERATION_PARAM_TYPE Taxi setStatus`

输出：

- `Ok, method "methodname" in class "classname" has parameter tables and return value: (type1, return: type2), (type3, return: type4), (type5, type6, no return)`。该操作有三种参数表和返回值的搭配，其中
 - 传出列表时可以乱序，内部参数类型也可以乱序，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**）
 - 如果不存在该命名的操作，则不输出任何搭配。
- `Failed, class "classname" not found`。类不存在
- `Failed, duplicated class "classname"`。类存在多个
- `Failed, wrong type of parameters or return value in method "methodname" of class "classname"`。存在错误类型
- `Failed, duplicated method "methodname" in class "classname"`。存在重复操作

说明：

- 对于参数类型
 - `NamedType` 只考虑JAVA 语言八大基本类型(`byte`, `short`, `int`, `long`, `float`, `double`, `char`, `boolean`)和 `String`，其余一律视为错误类型。
 - `ReferenceType` 指向已定义的类或接口，类型名为对应的类名或接口名。
- 对于返回值类型
 - `NamedType` 只考虑JAVA 语言八大基本类型(`byte`, `short`, `int`, `long`, `float`, `double`, `char`, `boolean`)和 `String`, `void`，其余一律视为错误类型。（实际上，`void`也算是一种类型，C/C++/Java对于这件事也都是这样的定义）。`void`不等同于无返回值。
 - `ReferenceType` 指向已定义的类或接口，类型名为对应的类名或接口名。
- 参数之间不分次序，即 `op(int,int,double)` 和 `op(double,int,int)` 视为具有相同参数类型，但参数和返回值之间是有区别的，且保证最多只有一个返回值，无返回值时相应位置返回**关键字** `null`。
- 如果两个操作的**操作名相同**，且**参数和返回值的类型也相同**，视为重复操作。
- 如果同时存在错误类型和重复操作两种异常，按错误类型异常处理。
- 本指令中统计的一律为此类自己定义的操作，不包含继承自其各级父类所定义的操作。

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; 特别的, 对于同名但id不同的类, 如果结果同时包含多个的话, 需要在列表中返回对应数量个类名)
 - 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

注意:

- 同上一条, Association关系需要考虑父类的继承。即, 假如X类继承了Y类, 那么Y的Association对端节点也属于X。
- 不考虑由属性和操作参数类型引起的关联。

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

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)
- 不需要考虑所实现的接口的属性。

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

说明:

- Initial State 和 Final State均算作状态。

给定状态机模型和其中的一个状态, 有多少个不同的可达状态

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：

- Initial State 和 Final State均算作状态。
- 可达状态不仅包括直接可达状态，也包括间接可达状态。

给定状态机模型中和其中两个状态，引起状态迁移的所有触发事件

输入指令格式： `TRANSITION_TRIGGER statemachine_name statename1 statename2`

举例： `TRANSITION_TRIGGER door_sm open close`

输出：

- `Ok, triggers of transition from state "open" to state "close" in statemachine "door_sm" are (A, B, C).` A、B、C为应状态机模型door_sm中引起状态open迁移到状态close的触发事件，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）
 - 保证所有的触发事件名称都不相同，且不为空
- `Failed, statemachine "door_sm" not found.` 未找到状态机模型door_sm
- `Failed, duplicated statemachine "door_sm".` 存在多个状态机模型door_sm
- `Failed, state "open" in statemachine "door_sm" not found.` 在状态机模型door_sm中未找到状态open
- `Failed, duplicated state "open" in statemachine "door_sm".` 在状态机模型door_sm中存在多个open状态
- `Failed, transition from state "open" to state "close" in statemachine "door_sm" not found.` 在状态机模型door_sm中未找到从状态open到状态close的迁移

说明：

- 该询问考虑的迁移为状态间的**直接迁移**。
- 检测状态与迁移异常时，先检测起点状态是否存在异常，再检测终点状态是否存在异常，最后检查是否存在相应的迁移。
- 若存在相应迁移，但无触发事件，应输出空列表。

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

给定UML顺序图和参与对象，发出了多少条消息

输入指令格式: `SENT_MESSAGE_COUNT umlinteraction_name lifeline_name messagesort`

举例: `SENT_MESSAGE_COUNT normal door SYNCH_CALL`

输出:

- `Ok, sent message count of lifeline "door" of umlinteraction "normal" is x.` x为顺序图模型normal (UMLInteraction) 中 door 发送的类别为 `SYNCH_CALL` 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

模型有效性检查

此部分为新增部分。

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 均不能有重名
- 输出:
 - 如未发现此问题，则不需要任何输出
 - `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。
 - 此规则无需考虑继承关系。

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

```
interface A extends B {  
    // something here  
}  
  
interface B extends A {  
    // something here  
}
```

这里就构成了一组最简单的循环继承。

- 输出：
 - 如未发现此问题，则不需要任何输出
 - 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类或接口(UML007)

- 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，包括直接继承或间接继承。
 - 例如下面的场景

```
interface A {  
    // something here  
}  
  
interface B extends A {  
    // something here  
}  
  
interface C extends A, B {  
    // something here  
}
```

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）

- 输出：
 - 如未发现此问题，则不需要任何输出
 - Failed when check R003, class/interface (A, B, C, D) have duplicate generalizations. 列出所有带有重复继承的类/接口名
- 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。

R004: 任何一个类不能重复实现同一个接口(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

```
interface A {  
    // something here  
}  
  
interface B extends A {  
    // something here  
}  
  
class D implements A,B {  
    // something here  
}
```

类D就重复实现了接口A（通过接口B间接继承）

- 输出：
 - 如未发现此问题，则不需要任何输出
 - Failed when check R004, class (A, B, C, D) have duplicate realizations. 列出所有带有重复实现的类名
- 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。
 - 该错误可能会和R003同时出现，需优先检测R003，并算作R003错误。

R005: 类图元素名字不能为空(UML 001)

- 规则解释
 - 目前类图已有的元素中，除 direction 为 return 的 UMLParameter、UMLGeneralization、UMLAssociation、UMLInterfaceRealization、UMLAssociationEnd 以外，其余元素的 name 字段均不能为空。
- 输出
 - 如未发现此问题，则不需要任何输出
 - Failed when check R005, a certain element doesn't have name.

R006: 接口的所有属性均需要为public(UML 011)

- 规则解释
 - 接口的attribute的可见性需要为 public
- 输出
 - 如未发现此问题，则不需要任何输出
 - Failed when check R006, All attributes and operations of interface must be public.

- 说明
 - 在我们的模型中，接口不会有方法，因此只检测属性

R007: lifeline必须表示在同一sequence diagram中定义的attribute

- 规则解释
 - `lifeline` 的 `represent` 属性必须与一个在同一 `sequence diagram` 中定义的 `attribute` 的 `id` 对应
- 输出
 - 如未发现此问题，则不需要任何输出
 - Failed when check R007, each lifeline must represent an attribute.

R008: Initial state最多只能有一个状态迁出且这个迁出不能有trigger和guard(UML 021&&UML 022)

- 规则解释
 - `Initial state` 最多只能有一个状态迁出且这个迁出不能有 `trigger` 和 `guard`
- 输出
 - 如未发现此问题，则不需要任何输出
 - Failed when check R008, An initial vertex has more than one outgoing transition or the outgoing transition from an initial vertex has a trigger or guard.
- 说明
 - 如果有多个状态迁移的尾端是同一个状态，依然算作多个状态迁出

样例

由于内容过多，请参考随指导书下发的8个mdj，并自行使用接口导出。[传送门](#)

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，保证所建模的模型类图，如果通过了检查，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，如果未通过检查，除了需要检查的错误，其余均符合java 8标准**
 - 每个数据，均可按照前述条件，找到唯一一个错误。
- 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
- 为了确保测试数据的合理性，测试数据有如下限制
 - 所有公测数据不会对

- 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。

- 类图相关

- 关于继承和实现：数据中只会出现类继承类、类实现接口、接口继承接口
- 不会出现类的多继承

- 状态图相关

- 确保每个State Machine中有且仅有一个Region；
- 确保每个State Machine中最多只有一个Initial State，最多只有一个Final State；
- 确保每个State Machine中Initial State和Final State的name均为null，查询指令中给出的状态也不会为Initial State或Final State；
- 确保每个State Machine中，除初始状态出发的转移外，从某个状态到另一个状态的直接迁移均具有不同的Event或Guard，即从某个状态到另一个状态的直接迁移若有多个，则这些迁移一定互不相同；
- 确保每个State Machine中，Initial State没有状态迁入，Final State没有状态迁出；
- 确保每个State Machine中不包含复合状态。

- 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

公测均通过标准输入输出进行。

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

- 本次作业的有效性检测基于StarUML的有效性检测，但出于易实现性和作业难度的考虑，对原规则进行了一定的修改。因此，在staruml中检测出的Validation Results仅为参考。
- StarUML官方文档中关于有效性检查的部分：[传送门](#)
- 关于标准输入输出
 - 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，请使用搜索引擎
- 关于作业中的一些问题，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。**

- 如果同时满足多个异常，在查询上层模型发生“异常”后，我们自然不该再去查询这个“异常层次”的下层次模型。

- [开源库地址](#)
- 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，请加助教微信进行举报，**经核实后，将直接作为无效作业处理。**