

面向对象UML系列第一次作业指导书

摘要

本次作业，需要完成的任务为实现一个UML类图解析器 `UmlInteraction`，学习目标为UML入门级的理解、UML类图的构成要素及其解析方法

问题

基本目标

本次作业最终需要实现一个UML类图解析器，可以通过输入各种指令来进行类图有关信息的查询。

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）我们均已实现，只需要同学们完成剩下的部分，即：

- 通过实现 `UmlInteraction` 这个官方提供的接口，来实现自己的 `UmlInteraction` 解析器

`UmlInteraction` 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 `MyUmlInteraction`（仅举例，具体类名可自行定义并配置），并实现相应的接口方法，每个方法的代码实现均需要严格满足给出的JML规格定义。

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 `MyUmlInteraction` 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**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

UmlInteraction类

`UmlInteraction` 的具体接口规格见官方包的开源代码，此处不加赘述。

除此之外，`UmlInteraction` 类必须实现一个构造方法

```
1 public class MyUmlInteraction implements UmlInteraction {
2     public MyUmlInteraction(UmlElement[] elements);
3 }
```

或者

```
1 public class MyUmlInteraction implements UmlInteraction {
2     public MyUmlInteraction(UmlElement... elements);
3 }
```

构造函数的逻辑为将 `elements` 数组内的各个UML类图元素传入 `UmlInteraction` 类，以备后续解析。

请确保构造函数正确实现，且类和构造函数均定义为 `public`。 `AppRunner` 内将自动获取此构造函数进行 `UmlInteraction` 实例的生成。

（注：这两个构造方法实际本质上等价，不过后者实际测试使用时的体验会更好，想要了解更多的话可以百度一下，关键词：`Java 变长参数`）

交互模式

交互的模式为：

- 调用上述构造函数，生成一个实例，并将UML模型元素传入。
- 之后将调用此实例的各个接口方法，以实现基于之前传入的UML模型元素的各类查询操作。

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

指令格式一览

模型中一共有多少个类

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为模型中指定类型的操作个数
- `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

注意：

- Associate关系需要考虑父类的继承。即，如果X类继承了Y类，那么Y的Association也属于X。

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

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。

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
 - 简单来说, 假如把继承关系比作上下级关系的话 (被继承者为上级), 那么顶级父类就相当于顶头上司

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

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）

样例

样例一

输入文本（模型部分导出自 mdj 文件：[传送门](#)，导出方法见官方包开源文档）

```

1  {"_parent": "AAAAAAfQ3tvYM76UevI=", "visibility": "public", "name": "key", "_type": "UMLClass",
   "_id": "AAAAAAfQ7weIMsb5xqQ="}
2  {"_parent": "AAAAAAfQ7weIMsb5xqQ=", "visibility": "public", "name": "equals", "_type": "UMLOperation",
   "_id": "AAAAAAfQ7weIMsb8qxc="}
3  {"_parent": "AAAAAAfQ7weIMsb8qxc=", "name": "o", "_type": "UMLParameter", "_id": "AAAAAAfQ7weIMsb9G0k=",
   "type": "Object", "direction": "in"}
4  {"_parent": "AAAAAAfQ7weIMsb8qxc=", "name": null, "_type": "UMLParameter", "_id": "AAAAAAfQ7weIMsb+Au4=",
   "type": "boolean", "direction": "return"}
5  {"_parent": "AAAAAAfQ7weIMsb5xqQ=", "visibility": "public", "name": "getMatchedLockId", "_type": "UMLOperation",
   "_id": "AAAAAAfQ7weIMsb\\6gM="}
6  {"_parent": "AAAAAAfQ7weIMsb\\6gM=", "name": null, "_type": "UMLParameter", "_id": "AAAAAAfQ7weIMsCAoOk=",
   "type": "int", "direction": "return"}
7  {"_parent": "AAAAAAfQ7weIMsb5xqQ=", "visibility": "public", "name": "Operation1", "_type": "UMLOperation",
   "_id": "AAAAAAfQ7w1zLCePjRI="}
8  {"_parent": "AAAAAAfQ7w1zLCePjRI=", "name": "Parameter1", "_type": "UMLParameter", "_id": "AAAAAAfQ7w2dZCev4K8=",
   "type": {"$ref": "AAAAAAfQ7weIMsb5xqQ="}, "direction": "return"}
9  {"_parent": "AAAAAAfQ7weIMsb5xqQ=", "name": null, "_type": "UMLGeneralization", "_id": "AAAAAAfQ7wfnvyd+GgY=",
   "source": "AAAAAAfQ7weIMsb5xqQ=", "target": "AAAAAAfQ7weqoCCQE7I="}
10 {"_parent": "AAAAAAfQ7weIMsb5xqQ=", "visibility": "private", "name": "keyID", "_type": "UMLAttribute",
   "_id": "AAAAAAfQ7weIMsb6+v8=", "type": "int"}
11 {"_parent": "AAAAAAfQ7weIMsb5xqQ=", "visibility": "private", "name": "matchedLockID", "_type": "UMLAttribute",
   "_id": "AAAAAAfQ7weIMsb7oPM=", "type": "int"}
12 {"_parent": "AAAAAAfQ3tvYM76UevI=", "visibility": "public", "name": "Elckey", "_type": "UMLClass",
   "_id": "AAAAAAfQ7weqoCCQE7I="}
13 {"_parent": "AAAAAAfQ7weqoCCQE7I=", "visibility": "public", "name": "equals", "_type": "UMLOperation",
   "_id": "AAAAAAfQ7weqoCCtngY="}
14 {"_parent": "AAAAAAfQ7weqoCCtngY=", "name": "o", "_type": "UMLParameter", "_id": "AAAAAAfQ7weqoCCUI6g=",
   "type": "Object", "direction": "in"}
15 {"_parent": "AAAAAAfQ7weqoCCtngY=", "name": null, "_type": "UMLParameter", "_id": "AAAAAAfQ7weqoCCVxIO=",
   "type": "boolean", "direction": "return"}
16 {"_parent": "AAAAAAfQ7weqoCCQE7I=", "name": "sdfsdf", "_type": "UMLGeneralization", "_id": "AAAAAAfQ7weqoCCRDg8=",
   "source": "AAAAAAfQ7weqoCCQE7I=", "target": "AAAAAAfQpyZaw1HqYaU="}
17 {"_parent": "AAAAAAfQ7weqoCCQE7I=", "visibility": "private", "name": "sigCode", "_type": "UMLAttribute",
   "_id": "AAAAAAfQ7weqoCCsu1Q=", "type": "long"}
18 END_OF_MODEL
19 CLASS_COUNT
20 CLASS_TOP_BASE Key

```

输出文本

```
1 Total class count is 2.
2 Ok, top base class of class "Key" is ElcKey.
```

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
 - 包含且仅包含类图，并在 `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）
 - **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 300 个元素
 - **此外为了方便本次的情况处理，保证所建模的类图模型，均可以在Oracle Java 8中正常实现出来**
- 输入指令限制
 - 最多不超过200条指令
 - 输入指令满足标准格式
- 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类图各种信息的正确性，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。

程序的最大运行时间为 2s，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
- 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：[传送门](#)
- 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
 - **去好好复习下本次的ppt（[传送门](#)），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进行举报，**经核实后，将直接作为无效作业处理。**