面向对象设计与构造第十三次作业

写在前面：请勿提交官方包代码，仅提交自己实现的类。更不要将官方包的 JML 或代码粘贴到自己的类中，否则以作弊、抄袭论处。

第一部分：训练目标

实现一个 UML 解析器，使其支持对 UML 类图的分析，可以通过输入相应的指令来进行相关查询。

第二部分：预备知识

同学们需要掌握 UML 图的相关知识，以及 StarUML 的使用方法。在这里给出一些相关参考资料：

- 第四单元手册

第三部分：基本概念

此处给出一些后文中将会用到的定义。所有用到的定义都会用**_加粗倾斜体_**标注。

如果有需要，你可以在后文遇到**_加粗倾斜体_**的定义时，再回来查阅对应的概念说明。

传入参数与返回值

UML 中操作参数元素 `UMLParameter` 的 `direction` 属性决定了它是传入参数还是返回值：

- `in`：传入参数
- `return`：返回值

在所有的数据中，`direction` 只会有上述两种类型。

NamedType

对于属性和操作中的传入参数而言，`NamedType` 包括

- Java 语言的八大基本类型（`byte`/`short`/`int`/`long`/`float`/`double`/`char`/`boolean`）
- `String`

其余一律视为错误类型。

对于属性和操作中的返回值而言，`NamedType` 包括

- Java 语言的八大基本类型（`byte`/`short`/`int`/`long`/`float`/`double`/`char`/`boolean`）
- `String`
- `void`（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）

其余一律视为错误类型。

ReferenceType

对于属性和操作中的传入参数、返回值而言，`ReferenceType` 指向已定义的**类**或**接口**，类型名为对应的**类名**或**接口名**。在本单元中，我们保证它是合法的，即 `ReferenceType` 不可能为错误类型。

类型相同

当两个类型对应的 `NameableType` 对象（参见官方包中的相关接口）在 `Objects.equals()` 下为真时，则这两个类型相同。

参数列表相同

对于任意两个操作，若它们具有相同数量的**传入参数**，且这两组传入参数之间存在某**一一映射**使得相对应的参数**_类型相同_**，则这两个操作的参数列表相同。例如 `int op(int, Class1, double)` 和 `String op(double, int, Class1)` 的参数列表相同。

重复操作

对于任意两个操作，如果它们的**名称相同**，且操作的**_参数列表相同_**，则它们互为重复操作。

耦合度

OO 度量指标中的类耦合度（Coupling Between Objects, CBO）最早由 Chidamber 与 Kemerer 于 1994 年提出，并被后人不断完善。在这里，我们采用 Visual Studio 2019 中 类耦合度指标 的约定，给出本单元中操作的耦合度与属性的耦合度的相关计算规则。

操作的耦合度：对于一个类的某个操作，考虑其所有类型为 `ReferenceType` 的操作参数类型（即元素 `UMLParameter` 的 `type` 属性，包括传入参数与返回值）：

- 若 `ReferenceType` 引用的类或接口是当前查询的类，则耦合度不增加，否则耦合度增加 1；
- 若存在多个 `ReferenceType` 引用的类或接口相同，则耦合度只会被计算一次，不会重复计算。

属性的耦合度：对于一个类，考虑其所有类型为 `ReferenceType` 的属性类型（即元素 `UMLAttribute` 的 `type` 属性）：

- 若 `ReferenceType` 引用的类或接口是当前查询的类，则耦合度不增加，否则耦合度增加 1；
- 若存在多个 `ReferenceType` 引用的类或接口相同，则耦合度只会被计算一次，不会重复计算。

继承深度

对于类 X，若 Y 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 Z 是 Y 的父类；

则称 Y 是 X 的顶级父类。这里的继承关系只考虑 UML 中存在的 `UmlGeneralization` 定义的继承关系，而不考虑所有类都默认继承自 `Object` 这个情况。

定义继承深度为某类到其顶级父类之间，经过的直接继承次数。例如：若只有两个类 A 与 B，且 A 直接继承 B 时，A 的继承深度为 1；若只有一个类 A ，则 A 的继承深度为 0。

多继承

对于多继承问题，我们保证在类图数据中，类一定没有多继承，但是接口可能有多继承。

第四部分：题目描述

一、作业要求

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，具体形式参考本次作业官方包目录下的 `README.md`。

代码结构说明

`UserApi` 的具体接口规格见官方包的 jar 文件，此处不加赘述。

除此之外，`UserApi` 类必须实现一个构造方法

```
public class MyImplementation implements UserApi {
    public MyImplementation(UmlElement[] elements);
}
```

或者

```
public class MyImplementation implements UserApi {
    public MyImplementation(UmlElement... elements);
}
```

构造函数的逻辑为将 `elements` 数组内的各个 UML 类图元素传入 `MyImplementation` 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 `UserApi` 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

- 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；
- 官方接口通过调用方法的返回值，自动生成对应的输出文本。

二、关于类图的查询指令

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。

指令 1：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.`
  - 其中 `x` 为模型中类的总数。

指令 2：类的子类数量

输入指令格式：`CLASS_SUBCLASS_COUNT classname`

举例：`CLASS_SUBCLASS_COUNT Elevator`

输出：

- `Ok, subclass count of class "classname" is x.`
  - 其中 `x` 为直接继承类 `classname` 的子类数量；
- `Failed, class "classname" not found.`
  - 不存在名为 `classname` 的类时，输出上述内容；
- `Failed, duplicated class "classname".`
  - 存在多个名为 `classname` 的类时，输出上述内容。

指令 3：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.`
  - 其中 `x` 为类 `classname` 中的操作个数；
- `Failed, class "classname" not found.`
  - 不存在名为 `classname` 的类时，输出上述内容；
- `Failed, duplicated class "classname".`
  - 存在多个名为 `classname` 的类时，输出上述内容。

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；
- 本指令中统计的无需考虑***重复操作*带来的影响。若有多个操作为*重复操作***，则这些操作都需要分别计入答案。

指令 4：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: www, protected: xxx, private: yyy, package-private: zzz.`
  - 其中 `www`/`xxx`/`yyy`/`zzz` 分别表示类 `classname` 中，名为 `methodname` 且实际可见性分别为 `public`/`protected`/`private`/`package-private` 的操作个数；
  - 如果类中不存在名为 `methodname` 的操作，则 `www`/`xxx`/`yyy`/`zzz` 全部设置为 0；
- `Failed, class "classname" not found.`
  - 不存在名为 `classname` 的类时，输出上述内容；
- `Failed, duplicated class "classname".`
  - 存在多个名为 `classname` 的类时，输出上述内容。

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；
- 在上一条的前提下，需要统计出全部的名为 `methodname` 的操作的可见性信息。
- 本指令中统计的无需考虑***重复操作*带来的影响。若有多个操作为*重复操作***，则这些操作都需要分别计入答案。

指令 5：类的操作的耦合度

输入指令格式：`CLASS_OPERATION_COUPLING_DEGREE classname methodname`

举例：`CLASS_OPERATION_COUPLING_DEGREE Taxi setStatus`

输出：

- `Ok, method "methodname" in class "classname" has coupling degree: coupling_degree_1, coupling_degree_2, coupling_degree_3.`
  - 此例中，类中名为 `methodname` 的操作共有 3 个，它们的***操作的耦合度*分别为 `coupling_degree_1`、`coupling_degree_2`、`coupling_degree_3`，且这些操作中不存在*重复操作***；
  - 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）；
  - 如果类中不存在名为 `methodname` 的操作，则传出一个空列表；
- `Failed, class "classname" not found.`
  - 不存在名为 `classname` 的类时，输出上述内容；
- `Failed, duplicated class "classname".`
  - 存在多个名为 `classname` 的类时，输出上述内容；
- `Failed, wrong type of parameters or return value in method "methodname" of class "classname".`
  - 类 `classname` 中所有名为 `methodname` 的操作存在***错误类型***时，输出上述内容；
- `Failed, duplicated method "methodname" in class "classname".`
  - 类 `classname` 中所有名为 `methodname` 的操作存在***重复操作***时，输出上述内容。

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；
- 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。

指令 6：类的属性的耦合度

输入指令格式：`CLASS_ATTR_COUPLING_DEGREE classname`

举例：`CLASS_ATTR_COUPLING_DEGREE Taxi`

输出：

- `Ok, attributes in class "classname" has coupling degree x.`
  - 其中 `x` 为类 `classname` 的**_属性的耦合度_**
- `Failed, class "classname" not found.`
  - 不存在名为 `classname` 的类时，输出上述内容；
- `Failed, duplicated class "classname".`
  - 存在多个名为 `classname` 的类时，输出上述内容。

说明：

- 本指令的查询需要考虑继承自其各级父类所定义的属性，但**不需要**考虑实现的接口所定义的属性（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）；
- 本查询中忽略属性名称相同的错误。

指令 7：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).`
  - 此例中，类 `classname` 实现了 `A`、`B`、`C` 这 3 个接口；
  - 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；
  - 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）；
  - 如果类 `classname` 没有实现任何接口，则传出一个空列表；
- `Failed, class "classname" not found.`
  - 不存在名为 `classname` 的类时，输出上述内容；
- `Failed, duplicated class "classname".`
  - 存在多个名为 `classname` 的类时，输出上述内容。

指令 8：类的继承深度

输入指令格式：`CLASS_DEPTH_OF_INHERITANCE classname`

举例：`CLASS_DEPTH_OF_INHERITANCE AdvancedTaxi`

输出：

- `Ok, depth of inheritance of class "classname" is x.`
  - 其中 `x` 为类 `classname` 的继承深度；
- `Failed, class "classname" not found.`
  - 不存在名为 `classname` 的类时，输出上述内容；
- `Failed, duplicated class "classname".`
  - 存在多个名为 `classname` 的类时，输出上述内容。

第五部分：设计建议

- 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
  - **好好复习下本次的 ppt，并理清楚各个 `UmlElement` 数据模型的结构与关系。**

第六部分：输入/输出说明

本次作业将会下发 mdj 文件解析工具、`UserApi` 类、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，其中输入输出接口、全局测试程序均在官方接口中。

- 解析工具用于将 mdj 格式文件解析为输入输出接口可以识别的格式，该格式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调用对应方法，并把返回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，这里只给出程序黑箱的字符串输入输出。

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。

样例输入

```
{"_parent":"AAAAAAF8pZCdQzQZZWk=","visibility":"package","name":"Class0","_type":"
UMLClass","_id":"AAAAAAF8pZCdTTQaucw="}
{"_parent":"AAAAAAF8pZCdTTQaucw=","visibility":"private","name":"Operation0","_typ
e":"UMLOperation","_id":"AAAAAAF8pZCdTTQbmVc="}
{"_parent":"AAAAAAF8pZCdTTQbmVc=","name":null,"_type":"UMLParameter","_id":"AAAAAA
F8pZCdTTQcAQc=","type":"byte","direction":"return"}
{"_parent":"AAAAAAF8pZCdQzQZZWk=","visibility":"package","name":"Class1","_type":"
UMLClass","_id":"AAAAAAF8pZCdTTQdhjs="}
{"_parent":"AAAAAAF8pZCdTTQdhjs=","name":null,"_type":"UMLGeneralization","_id":"A
AAAAAF8pZCdTjQifT8=","source":"AAAAAAF8pZCdTTQdhjs=","target":"AAAAAAF8pZCdTTQaucw
="}
{"_parent":"AAAAAAF8pZCdTTQdhjs=","visibility":"protected","name":"Attribute1","_t
ype":"UMLAttribute","_id":"AAAAAAF8pZCdTjQkPlM=","type":"byte"}
{"_parent":"AAAAAAF8pZCdQzQZZWk=","visibility":"package","name":"Class2","_type":"
UMLClass","_id":"AAAAAAF8pZCdTTQeyHs="}
{"_parent":"AAAAAAF8pZCdTTQeyHs=","visibility":"private","name":"Operation1","_typ
e":"UMLOperation","_id":"AAAAAAF8pZCdTTQf1NU="}
{"_parent":"AAAAAAF8pZCdTTQf1NU=","name":null,"_type":"UMLParameter","_id":"AAAAAA
F8pZCdTTQg0v8=","type":"boolean","direction":"return"}
{"_parent":"AAAAAAF8pZCdTTQeyHs=","name":null,"_type":"UMLGeneralization","_id":"A
AAAAAF8pZCdTjQjm3k=","source":"AAAAAAF8pZCdTTQeyHs=","target":"AAAAAAF8pZCdTTQh6N0
="}
{"_parent":"AAAAAAF8pZCdQzQZZWk=","visibility":"package","name":"Class3","_type":"
UMLClass","_id":"AAAAAAF8pZCdTTQh6N0="}
{"_parent":"AAAAAAF8pZCdTTQh6N0=","name":null,"_type":"UMLAssociation","end2":"AAA
AAAF8pZCdTjQn110=","end1":"AAAAAAF8pZCdTjQm1zQ=","_id":"AAAAAAF8pZCdTjQl2RU="}
{"reference":"AAAAAAF8pZCdTTQdhjs=","multiplicity":"","_parent":"AAAAAAF8pZCdTjQl2
RU=","visibility":"public","name":"Attribute0","_type":"UMLAssociationEnd","aggreg
ation":"none","_id":"AAAAAAF8pZCdTjQn110="}
{"reference":"AAAAAAF8pZCdTTQh6N0=","multiplicity":"","_parent":"AAAAAAF8pZCdTjQl2
RU=","visibility":"package","name":null,"_type":"UMLAssociationEnd","aggregation":
"none","_id":"AAAAAAF8pZCdTjQm1zQ="}
END_OF_MODEL
CLASS_COUNT
CLASS_SUBCLASS_COUNT Class0
CLASS_SUBCLASS_COUNT Class1
CLASS_SUBCLASS_COUNT Class2
CLASS_SUBCLASS_COUNT Class3
CLASS_OPERATION_COUPLING_DEGREE Class0 Operation0
CLASS_OPERATION_COUPLING_DEGREE Class2 Operation1
```

样例输出

```
Total class count is 4.
Ok, subclass count of class "Class0" is 1.
Ok, subclass count of class "Class1" is 0.
Ok, subclass count of class "Class2" is 0.
Ok, subclass count of class "Class3" is 1.
```

```
Ok, method "Operation0" in class "Class0" has coupling degree: 0.
Ok, method "Operation1" in class "Class2" has coupling degree: 0.
```

一、公测说明

`.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；
- `.mdj` 各元素字段值限制见**第八部分：附录**；
- 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。

测试数据限制：

- 所有公测数据不会对接口中定义的方法、类属性（static attribute）、类方法（static method）做任何测试要求。

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 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 CPU 时间为 2s，保证强测数据有一定梯度。

二、互测说明

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。

第七部分：提示与警示

一、提示

- 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 `UserApi` 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`。
- **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

二、警示

- **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 `18374457@buaa.edu.cn` 或使用微信向助教进行举报，**经核实后，将直接作为无效作业处理。**

第八部分：附录

标准输入限制说明

下面给出了标准输入在经过官方接口转换后生成的各种对象中各种字段的格式限制。

首先，对所有 `UmlElement` 子类以及所有实现 `NameableType` 接口的类的以下字段有通用规定，不再在后面赘述：

- 所有的 `id` 字段都是不为 `null` 的字符串，且在主函数中传入的 `UmlElement` 列表中唯一。
- 除额外说明，所有的 `name` 字段**都不作任何保证**，即均可能为 `null`，或空字符串 `""`，或有其他内容的字符串。
- 所有的 `visibility` 字段都是不为 `null` 的枚举对象。
- 若字段 X 的格式说明是 "***None***"：**对该字段不作任何保证**，可能为 `null` 或者该类型的对象。

有某些字段是代表指向其他 `UmlElement` 的字符串格式的 ID，如 `parentId`、`end1`、`source` 等等，为了简洁地说明特做出以下规定：

- 若某 `String` 类型字段 S 的格式说明形如 "***UmlOperation***" : 则 S 不为 `null`，且在传入的 `UmlElement` 列表中存在唯一一个 `id` 等于 S 的对象，且该对象的类型是 `UmlOperation`。
- 若某 `String` 类型字段 S 的格式说明形如 "***UmlClass* | *UmlInterface***"：则 S 不为 `null`，且在传入的 `UmlElement` 列表中存在唯一一个 `id` 等于 S 的值的对象，且该对象的类型是 `UmlClass` **或者** `UmlInterface`。

UML 类图相关元素格式限制

UmlClass

- parentId：***None***

UmlOperation

- parentId：***UmlClass***

UmlAttribute

- parentId：***UmlClass* | *UmlInterface***
- type：NotNull

UmlParameter

- parentId：***UmlOperation***
- type：NotNull

- direction： IN | RETURN

ReferenceType

- referenceId：***UmlClass* | *UmlInterface***

UmlAssociation

- parentId：***None***
- end2：***UmlAssociationEnd***
- end1：***UmlAssociationEnd***

UmlAssociaitonEnd

- parentId：***UmlAssociation***
- reference：***UmlClass* | *UmlInterface***
- multiplicity：NotNull
- aggregation：NotNull

UmlInterface

- parentId：***None***

UmlInterfaceRealization

- parentId：***None***
- source：***UmlClass***
- target：***UmlInterface***

UmlGeneralization

- parentId：***None***
- source：***UmlClass* | *UmlInterface***
- target：***UmlClass* | *UmlInterface***
- 注：source 和 target 指向的元素类型一定相同。