面向对象JML系列第一次代码作业指导书

摘要

本次作业，需要完成的任务为实现两个容器类`Path`和`PathContainer`，学习目标为**JML规格入门级的理解和代码实现**。

问题

基本目标

本次作业最终需要实现一个路径管理系统。可以通过各类输入指令来进行数据的增删查改等交互。

基本任务

本次作业的程序主干逻辑我们均已经实现，只需要同学们完成剩下的部分，即：

- **通过继承`Path`、`PathContainer`两个官方提供的接口，来实现自己的`Path`、`PathContainer`容器。**

每个模块的**接口定义源代码和对应的JML规格**都已在接口源代码文件中给出，各位同学需要**准确理解JML规格**，然后使用Java来实现相应的接口，并保证**代码实现严格符合对应的JML规格**。

具体来说，各位同学需要新建两个类`MyPath`和`MyPathContainer`（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通过调用官方包的`AppRunner`类，并载入自己实现的`Path`、`PathContainer`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

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

类规格

Path类

`Path`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，`Path`类必须实现一个构造方法

```
class MyPath implements Path {
    public MyPath(int[] nodeList);
}
```

或者

```
class MyPath implements Path {
    public MyPath(int... nodeList);
}
```

构造函数的逻辑为按照`nodeList`数组内的结点序列生成一个`Path`对象。

请确保构造函数正确实现，且为`public`。`AppRunner`内将自动获取此构造函数进行`Path`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`）

PathContainer类

`PathContainer`类的接口规格见官方包的开源代码，此处不加赘述。

除此之外，`PathContainer`类必须实现构造方法

```
class MyPathContainer implements PathContainer {
    public MyPathContainer();
}
```

构造函数的逻辑为生成一个空的`PathContainer`对象。

请确保构造函数正确实现，且为`public`。`AppRunner`内将自动获取此构造函数进行`PathContainer`实例的生成。

输入输出

本次作业将会下发输入输出接口和全局测试调用程序，前者用于输入输出的解析和处理，后者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，输出一律在标准输出。
- 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。

指令相关概念

- **结点**：一个结点是一个int范围（32位有符号）的整数
- **结点序列**：一个结点序列由n个结点组成，每个整数之间以一个空格分隔，其中 $2 <= n <= 1000$
- **路径id**：任意int范围（32位有符号）整数（每一个成功加入容器的路径都有一个唯一的id，且按

照加入的先后顺序从1开始递增）

指令格式一览

添加路径

输入指令格式：`PATH_ADD 结点序列`

举例：`PATH_ADD 1 2 3`

输出：

- `Ok, path id is x.` x是调用返回的路径id，如果容器中已有相同的路径，则返回已有的路径id，否则返回添加后的路径id

删除路径

输入指令格式：`PATH_REMOVE 结点序列`

举例：`PATH_REMOVE 1 2 3`

输出：

- `Failed, path not exist.` 容器中不存在结点序列对应的路径
- `Ok, path id is x.` 容器中存在路径，x是被删除的路径id

根据路径id删除路径

输入指令格式：`PATH_REMOVE_BY_ID 路径id`

举例：`PATH_REMOVE_BY_ID 3`

输出：

- `Failed, path not exist.` 容器中不存在路径id对应的路径
- `Ok, path removed.` 容器中存在路径，x是被删除的路径id

查询路径id

输入指令格式：`PATH_GET_ID 结点序列`

举例：`PATH_GET_ID 1 2 3`

输出：

- `Failed, path not exist.` 容器中不存在结点序列对应的路径
- `Ok, path id is x.` 容器中存在路径，x是被查询的路径id

根据路径id获得路径

输入指令格式：`PATH_GET_BY_ID 路径id`

举例：`PATH_GET_BY_ID`

输出：

- `Failed, path not exist.` 容器中不存在路径id对应的路径
- `Ok, path is x.` 容器中存在路径，x是用英文圆括号包起来的结点序列，例如：`Ok, path is (1 2 3).`

获得容器内总路径数

输入指令格式：`PATH_COUNT`

举例：`PATH_COUNT`

输出：`Total count is x.` x是容器内总路径数

根据路径id获得其大小

输入指令格式：`PATH_SIZE 路径id`

举例：`PATH_SIZE 3`

输出：

- `Failed, path not exist.` 容器中不存在路径id对应的路径
- `Ok, path size is x.` 容器中存在路径，x是该路径的大小

根据路径id获取其不同的结点个数

输入指令格式：`PATH_DISTINCT_NODE_COUNT 路径id`

举例：`PATH_DISTINCT_NODE_COUNT 5`

输出：

- `Failed, path not exist.` 容器中不存在路径id对应的路径
- `Ok, distinct node count of path is x.` 容器中存在路径，x是路径中不同的结点个数

根据结点序列判断容器是否包含路径

输入指令格式：`CONTAINS_PATH 结点序列`

举例：`CONTAINS_PATH 1 2 3`

输出：

- `Yes.` 如果容器中包含该路径
- `No.` 如果容器中不包含该路径

根据路径id判断容器是否包含路径

输入指令格式：`CONTAINS_PATH_ID 路径id`

举例：`CONTAINS_PATH_ID 3`

输出：

- `Yes.` 如果容器中包含该路径
- `No.` 如果容器中不包含该路径

容器内不同结点个数

输入指令格式：`DISTINCT_NODE_COUNT`

举例：`DISTINCT_NODE_COUNT`

输出：`Ok, distinct node count is x.` x是容器内不同结点的个数

根据字典序比较两个路径的大小关系

输入指令格式：`COMPARE_PATHS 路径id 路径id`

举例：`COMPARE_PATHS 3 5`

输出：

- `Ok, path x is less than to y.` 字典序x小于y，x是输入的第一个路径id，y是第二个路径id
- `Ok, path x is greater than to y.` 字典序x大于y，x是输入的第一个路径id，y是第二个路径id

路径中是否包含某个结点

输入指令格式：`PATH_CONTAINS_NODE 路径id 结点`

举例：`PATH_CONTAINS_NODE 3 5`

输出：

- `Failed, path not exist.` 容器中不存在路径id对应的路径
- `Yes.` 容器中存在该路径，且包含此结点
- `No.` 容器中存在该路径，但不包含此结点

样例

#	标准输入	标准输出
1	PATH_ADD 1 2 2 3 PATH_ADD 3 PATH_REMOVE 1 2 3 CONTAINS_PATH 1 2 2 3 CONTAINS_PATH_ID 1 PATH_REMOVE_BY_ID 1 CONTAINS_PATH_ID 1	Ok, path id is 1. Failed, invalid path. Failed, path not exist. Yes. Yes. Ok, path removed. No.
2	PATH_ADD -3 0 5 PATH_ADD -2 0 PATH_REMOVE 0 -2 PATH_COUNT PATH_SIZE COMPARE_PATHS 1 2 PATH_CONTAINS_NODE 1 -3	Ok, path id is 1. Ok, path id is 2. Failed, path not exist. Total count is 2. Ok, path size is 3. Ok, path 1 is less than to 2. Yes.
3	PATH_ADD 1 2 3 5 PATH_ADD 1 2 4 3 4 PATH_ADD 1 2 3 5 PATH_DISTINCT_NODE_COUNT 2 DISTINCT_NODE_COUNT CONTAINS_PATH 1 2 3 CONTAINS_PATH 1 2 3 5 PATH_REMOVE 1 2 4 3 4 DISTINCT_NODE_COUNT	Ok, path id is 1. Ok, path id is 2. Ok, path id is 1. Ok, distinct node count of path is 4. Ok, distinct node count is 5. No. Yes. Ok, path id is 2. Ok, distinct node count is 4.

关于判定

数据基本限制

- 输入指令数
  - 总上限为3000条
  - 其中，指令
    - `PATH_ADD`
    - `PATH_REMOVE`
    - `PATH_REMOVE_BY_ID`
    - `PATH_GET_ID`
    - `PATH_GET_BY_ID`
    - `CONTAINS_PATH`
    - `COMPARE_PATHS`
  - 的指令条数之和不超过300
- 输入指令满足标准格式。
- 输入结点序列最长为1000，最短为2，其中结点编号可重复。

- 输入的路径id必须是int范围（32位有符号）的整数。
- 满足以上基本限制的输入数据都是合法数据。也即互测数据需要满足的基本条件。

测试模式

公测和互测都将使用指令的形式模拟容器的各种状态，从而测试各个接口的实现正确性，即是否满足JML规格的定义。**可以认为，只要代码实现严格满足JML，就能保证正确性**。

任何满足规则的输入，程序都应该保证不会异常退出，如果出现问题即视为未通过该测试点。

程序的最大运行cpu时间为5s，虽然保证强测数据有梯度，但是还是请注意时间复杂度的控制。

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。只需要保证两个类的继承与实现即可。
- **关于本次作业容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`。
- **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，在互测环节中，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**。