

面向对象 JML 系列第三次代码作业指导书

题目描述

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。

基本目标

本次作业最终需要实现一个社交关系模拟系统。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：

- 通过实现官方提供的接口 `Person`、`Network` 和 `Group`，来实现自己的 `Person`、`Network` 和 `Group` 类。
- 通过实现官方提供的接口 `Message`、`EmojiMessage`、`NoticeMessage` 和 `RedEnvelopeMessage`，来实现自己的 `Message`、`EmojiMessage`、`NoticeMessage` 和 `RedEnvelopeMessage` 类。
- 阅读指导书中关于异常类行为的描述，通过继承官方提供的各抽象异常类，实现自己的异常类。

`Person`、`Network` 和 `Group` 类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，各位同学需要新建三个类 `MyPerson`、`MyNetwork` 和 `MyGroup`（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。

继承自 `Message` 的接口有 `RedEnvelopeMessage`、`noticeMessage`、`emojiMessage`，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，同样需要各位同学准确理解 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。

抽象异常类已在官方包内给出，这一部分没有提供 JML 规格，各位同学需要仔细阅读指导书中关于异常类的详细描述，结合样例理解其行为，然后继承这些抽象类实现自己的异常类，使其 `print()` 方法能够正确输出指定的信息。

当然，还需要同学们在主类中通过调用官方包的 `Runner` 类，并载入自己实现的 `Person`、`Network`、`Group` 和各个消息类，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

Person类

`Person` 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，`Person` 类必须实现一个构造方法

```
1 public class MyPerson implements Person {
2     public MyPerson(int id, String name, int age);
3 }
```

构造函数的逻辑为完成以下操作并生成一个 `Person` 对象。

`person` 的属性：

`id`：独一无二的 `id`

`name`：姓名

`age`：年龄

`socialValue`：社交值，初始值为 `0`

`money`：钱数，初始值为 `0`

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

Network类

`Network` 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，`Network` 类必须实现一个构造方法

```
1 public class MyNetwork implements Network {
2     public MyNetwork();
3 }
```

构造函数的逻辑为生成一个 `Network` 对象。

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

Group类

`Group` 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，`Group` 类必须实现一个构造方法

```
1 public class MyGroup implements Group {
2     public MyGroup(int id);
3 }
```

构造函数的逻辑为完成以下操作并生成一个 `Group` 对象。

`Group` 的属性：

`id`：独一无二的 `id`

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

Message类

`Message` 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，`Message` 类必须实现两个构造方法，逻辑为接收消息的属性并生成一个 `Message` 对象。

```
1 public class MyMessage implements Message {
2
3     /*@ ensures type == 0;
4         @ ensures group == null;
5         @ ensures id == messageId;
6         @ ensures socialValue == messageSocialValue;
7         @ ensures person1 == messagePerson1;
8         @ ensures person2 == messagePerson2;
9         @*/
10    public MyMessage(int messageId, int messageSocialValue, Person
        messagePerson1, Person messagePerson2);
11
12    /*@ ensures type == 1;
13        @ ensures person2 == null;
14        @ ensures id == messageId;
15        @ ensures socialValue == messageSocialValue;
16        @ ensures person1 == messagePerson1;
17        @ ensures group == messageGroup;
18        @*/
19    public MyMessage(int messageId, int messageSocialValue, Person
        messagePerson1, Group messageGroup);
20 }
```

`Message` 的属性：

`id`：独一无二的 id

`socialValue`：消息的社交值

`type`：消息的种类，有 0 和 1 两个取值

`person1`：消息的发送者

`person2`：消息的接收者

`group`：消息的接收组

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

RedEnvelopeMessage类

`RedEnvelopeMessage` 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，`RedEnvelopeMessage` 类必须实现两个构造方法，逻辑为接收消息的属性并生成一个 `RedEnvelopeMessage` 对象。

```
1 public class MyRedEnvelopeMessage implements RedEnvelopeMessage {
2
3     /*@ ensures type == 0;
4         @ ensures group == null;
5         @ ensures id == messageId;
```

```

6      @ ensures person1 == messagePerson1;
7      @ ensures person2 == messagePerson2;
8      @ ensures money == luckyMoney;
9      @*/
10     public MyRedEnvelopeMessage(int messageId, int luckyMoney, Person
messagePerson1, Person messagePerson2);
11
12     /*@ ensures type == 1;
13         @ ensures person2 == null;
14         @ ensures id == messageId;
15         @ ensures person1 == messagePerson1;
16         @ ensures group == messageGroup;
17         @ ensures money == luckyMoney;
18         @*/
19     public MyRedEnvelopeMessage(int messageId, int luckyMoney, Person
messagePerson1, Group messageGroup);
20 }

```

`RedEnvelopeMessage` (红包消息) 的属性:

`id`: 独一无二的消息 id

`money`: 红包的金额

`type`: 消息的种类, 有 0 和 1 两个取值

`person1`: 消息发送者

`person2`: 消息接收者

`group`: 消息的接收组

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

NoticeMessage类

`NoticeMessage` 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, `NoticeMessage` 类必须实现两个构造方法, 逻辑为接收消息的属性并生成一个 `NoticeMessage` 对象。

```

1  public class MyNoticeMessage implements NoticeMessage {
2
3      /*@ ensures type == 0;
4          @ ensures group == null;
5          @ ensures id == messageId;
6          @ ensures person1 == messagePerson1;
7          @ ensures person2 == messagePerson2;
8          @ ensures string == noticeString;
9          @*/
10     public MyNoticeMessage(int messageId, String noticeString, Person
messagePerson1, Person messagePerson2);
11
12     /*@ ensures type == 1;
13         @ ensures person2 == null;
14         @ ensures id == messageId;
15         @ ensures person1 == messagePerson1;

```

```

16         @ ensures group == messageGroup;
17         @ ensures string == noticeString;
18         @*/
19         public MyNoticeMessage(int messageId, String noticeString, Person
messagePerson1, Group messageGroup);
20     }

```

`NoticeMessage`（通知消息）的属性：

`id`：独一无二的消息 id

`string`：通知的内容

`type`：消息的种类，有 0 和 1 两个取值

`person1`：消息发送者

`person2`：消息接收者

`group`：消息的接收组

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

EmojiMessage类

`EmojiMessage` 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，`EmojiMessage` 类必须实现两个构造方法，逻辑为接收消息的属性并生成一个 `EmojiMessage` 对象。

```

1     public class MyEmojiMessage implements EmojiMessage {
2
3         /*@ ensures type == 0;
4             @ ensures group == null;
5             @ ensures id == messageId;
6             @ ensures person1 == messagePerson1;
7             @ ensures person2 == messagePerson2;
8             @ ensures emojiId == emojiNumber;
9             @*/
10        public MyEmojiMessage(int messageId, int emojiNumber, Person
messagePerson1, Person messagePerson2);
11
12        /*@ ensures type == 1;
13            @ ensures person2 == null;
14            @ ensures id == messageId;
15            @ ensures person1 == messagePerson1;
16            @ ensures group == messageGroup;
17            @ ensures emojiId == emojiNumber;
18            @*/
19        public MyEmojiMessage(int messageId, int emojiNumber, Person
messagePerson1, Group messageGroup);
20    }

```

`EmojiMessage`（表情消息）的属性：

`id`：独一无二的消息 id

`emojiId`：表情编号

`type`：消息的种类，有 0 和 1 两个取值

`person1`：消息发送者

`person2`：消息接收者

`group`：消息的接收组

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

异常类要求

同学们需要通过继承官方包提供的抽象异常类，来实现 10 个具有计数功能的异常类。

每个异常类必须正确实现指定参数的构造方法。

除此之外，还需要实现一个无参的 `print()` 方法。`print()` 方法需将包含计数结果的指定信息输出到标准输出中，`Runner` 类会自动调用该方法。为实现计数功能，同学们可以在异常类中自定义其他属性、方法（例如：可以构造一个计数器类，其实例作为每个异常类的 `static` 属性，管理该类型异常的计数）。

具体要求如下：

- **PersonIdNotFoundException：**

```
1 public class MyPersonIdNotFoundException extends
  PersonIdNotFoundException {
2     public MyPersonIdNotFoundException(int id);
3 }
```

- 输出格式：`pinf-x, id-y`，`x` 为此类异常发生的总次数，`y` 为该 `Person.id` 触发此类异常的次数
- 当 `network` 类某方法中有多个参数都会触发此异常时，只以第一个触发此异常的参数抛出一异常

- **EqualPersonIdException：**

```
1 public class MyEqualPersonIdException extends EqualPersonIdException {
2     public MyEqualPersonIdException(int id);
3 }
```

- 输出格式：`epi-x, id-y`，`x` 为此类异常发生的总次数，`y` 为该 `Person.id` 触发此类异常的次数

- **RelationNotFoundException：**

```
1 public class MyRelationNotFoundException extends
  RelationNotFoundException {
2     public MyRelationNotFoundException(int id1, int id2);
3 }
```

- 输出格式：`rnf-x, id1-y, id2-z`，`x` 为此类异常发生的总次数，`y` 为 `id1` 触发此类异常的次数，`z` 为 `id2` 触发此类异常的次数
- `id1`，`id2` 按数值大小排序，由小到大输出

- **EqualRelationException：**

```

1 public class MyEqualRelationException extends EqualRelationException {
2     public MyEqualRelationException(int id1, int id2);
3 }

```

- 输出格式: `er-x, id1-y, id2-z`, `x` 为此类异常发生的总次数, `y` 为 `id1` 触发此类异常的次数, `z` 为 `id2` 触发此类异常的次数
- `id1`, `id2` 按数值大小排序, 由小到大输出
- `id1` 与 `id2` 相等时, 视为该 `id` 触发了一次此类异常

- **GroupIdNotFoundException:**

```

1 public class MyGroupIdNotFoundException extends GroupIdNotFoundException
2 {
3     public MyGroupIdNotFoundException(int id);
4 }

```

- 输出格式: `ginf-x, id-y`, `x` 为此类异常发生的总次数, `y` 为该 `Group.id` 触发此类异常的次数

- **EqualGroupIdException:**

```

1 public class MyEqualGroupIdException extends EqualGroupIdException {
2     public MyEqualGroupIdException(int id);
3 }

```

- 输出格式: `egi-x, id-y`, `x` 为此类异常发生的总次数, `y` 为该 `Group.id` 触发此类异常的次数

- **EqualMessageIdException:**

```

1 public class MyEqualMessageIdException extends EqualMessageIdException {
2     public MyEqualMessageIdException(int id);
3 }

```

- 输出格式: `emi-x, id-y`, `x` 为此类异常发生的总次数, `y` 为该 `Message.id` 触发此类异常的次数

- **MessageIdNotFoundException:**

```

1 public class MyMessageIdNotFoundException extends
2     MessageIdNotFoundException {
3     public MyMessageIdNotFoundException(int id);
4 }

```

- 输出格式: `minf-x, id-y`, `x` 为此类异常发生的总次数, `y` 为该 `Message.id` 触发此类异常的次数

- **EmojiIdNotFoundException:**

```

1 public class MyEmojiIdNotFoundException extends EmojiIdNotFoundException
2 {
3     public MyEmojiIdNotFoundException(int id);
4 }

```

- 输出格式: `einfx, id-y`, `x` 为此类异常发生的总次数, `y` 为该 `emojiId` 触发此类异常的次数

- **EqualEmojiIdException:**

```
1 public class MyEqualEmojiIdException extends EqualEmojiIdException {
2     public MyEqualEmojiIdException(int id);
3 }
```

- 输出格式: `eei-x, id-y`, x 为此类异常发生的总次数, y 为该 `emojiId` 触发此类异常的次数

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。

关于main函数内对于 `Runner` 的调用, 参见以下写法。

```
1 package xxx;
2
3 import com.oocourse.spec3.main.Runner;
4
5 public class xxx {
6     public static void main(String[] args) throws Exception {
7         Runner runner = new Runner(MyPerson.class, MyNetwork.class,
8         MyGroup.class, MyMessage.class, MyEmojiMessage.class, MyNoticeMessage.class,
9         MyRedEnvelopeMessage.class);
10        runner.run();
11    }
12 }
```

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。

指令格式一览(括号内为变量类型)

- 基本格式: `指令字符串 参数1 参数2 ...`

本次作业涉及指令如下:

实际上为了减小输入量, 真实输入为简写

```
1 add_person id(int) name(String) age(int)
2 add_relation id(int) id(int) value(int)
3 query_value id(int) id(int)
4 compare_name id(int) id(int)
5 query_name_rank id(int)
6 query_people_sum
7 query_circle id(int) id(int)
8 query_block_sum
9
```



```
10 add_group id(int)
11 add_to_group id(int), id(int)

12 query_group_sum
13 query_group_people_sum id(int)
14 query_group_value_sum id(int)
15 query_group_age_mean id(int)
16 query_group_age_var id(int)
17 del_from_group id(int) id(int)
18 add_message id(int) socialValue(int) type(int)
19     person_id1(int) person_id2(int)|group_id(int)
20 send_message id(int)
21 query_social_value id(int)
22 query_received_messages id(int)
23
24 add_red_envelope_message id(int) money(int) type(int)
25     person_id1(int) person_id2(int)|group_id(int)
26 add_notice_message id(int) string(String) type(int)
27     person_id1(int) person_id2(int)|group_id(int)
28 add_emoji_message id(int) emoji_id(int) type(int)
29     person_id1(int) person_id2(int)|group_id(int)
30 store_emoji_id id(int)
31 query_popularity id(int)
32 delete_cold_emoji limit(int)
33 query_money id(int)
34 send_indirect_message id(int)
```

指令	简写
add_person	ap
add_relation	ar
query_value	qv
compare_name	cn
query_name_rank	qnr
query_people_sum	qps
query_circle	qci
query_block_sum	qbs
add_group	ag
add_to_group	atg
query_group_sum	qgs
query_group_people_sum	qgps
query_group_value_sum	qgvs
query_group_age_mean	qgam
query_group_age_var	qgav
del_from_group	dfg
add_message	am
send_message	sm
query_social_value	qsv
query_received_messages	qrm
add_red_envelope_message	arem
add_notice_message	anm
add_emoji_message	aem
store_emoji_id	sei
query_popularity	qp
delete_cold_emoji	dce
query_money	qm
send_indirect_message	sim

样例

#	标准输入	标准输出
1	ap 1 jack 100 ap 2 mark 100 ar 1 2 100 qv 1 2	Ok Ok Ok 100
2	ap 1 jack 100 ap 2 mark 100 ar 1 2 100 qnr 1 qbs	Ok Ok Ok 1 1
3	ap 1 jack 100 ap 2 mark 100 ar 1 2 100 qps	Ok Ok Ok 2
4	ap 1 jack 100 ap 2 mark 100 ar 1 2 100 ag 1 atg 1 1	Ok Ok Ok Ok Ok
5	ap 1 jack 100 ap 2 mark 100 ar 1 2 100 ag 1 atg 1 1 qgs qgps 1 dfg 1 1 qgps 1	Ok Ok Ok Ok Ok 1 1 Ok 0
6	ap 1 jack 100 ap 2 mark 100 ar 1 2 100 ag 1 atg 1 1 qgvs 1 qgam 1 qgav 1 am 1 100 1 1 1 sm 1 qsv 1	Ok Ok Ok Ok Ok 0 100 0 Ok Ok 100

#	标准输入	标准输出
7	qv 1 2 qv 2 1 ap 1 jack 100 ap 1 mark 100 ap 2 mark 100 qv 1 2 qv 2 1 ar 1 2 100 ar 1 2 200 atg 1 1 ag 1 ag 1 sm 1 am 1 100 0 1 2 am 1 100 0 1 2	pinf-1, 1-1 pinf-2, 2-1 Ok epi-1, 1-1 Ok rnf-1, 1-1, 2-1 rnf-2, 1-2, 2-2 Ok er-1, 1-1, 2-1 ginf-1, 1-1 Ok egi-1, 1-1 minf-1, 1-1 Ok emi-1, 1-1
8	ap 1 jack 100 ap 2 mark 100 ap 3 dark 100 ar 1 2 100 ar 1 3 10 ar 2 3 10 am 1 100 0 1 2 anm 2 str 0 1 2 arem 3 100 0 1 2 sei 1 aem 4 1 0 1 2 sm 1 sm 2 sm 3 sim 4 qsv 1 qrm 2 qp 1 dce 100 qm 1	Ok Ok Ok Ok Ok Ok Ok Ok Ok Ok Ok Ok Ok Ok 20 604 Emoji: 1; RedEnvelope: 100; notice: str; Ordinary message 1 0 -100

关于判定

数据基本限制

指令条数不多于 10000

add_person 指令条数不超过 5000

query_name_rank 指令条数不超过 333

query_circle 指令条数不超过 333

add_group 指令条数不超过 10

age(int) 值在 [0,200] 中

`value(int)` 值在 `[0,1000]` 中

`name(String)` 长度不超过 10

`emoji_id(int)` 值在 `[0,10000]` 中

`add_message`, `add_red_envelope_message`, `add_notice_message`, `add_emoji_message` 指令: 保证 `type` 为 0 或 1, `socialValue(int)` 值在 `[-1000,1000]` 中, `money(int)` 值在 `[0,200]` 中, `string(String)` 长度不超过 100。此外, 输入指令可能存在 `person_id1 | person_id2 | group_id` 在社交网络中不存在的情况, `Runner` 类会检查出这样的指令并且屏蔽。因此, 当类和方法都按照指导书要求和JML规格正确实现时, 无需考虑此类情况。详情见官方包中的 `Runner` 类源码。

互测数据限制

指令条数不多于 5000

`add_person` 指令条数不超过 5000

`query_name_rank` 指令条数不超过 333

`query_circle` 指令条数不超过 333

`add_group` 指令条数不超过 10

`age(int)` 值在 `[0,200]` 中

`value(int)` 值在 `[0,1000]` 中

`name(String)` 长度不超过 10

`emoji_id(int)` 值在 `[0,10000]` 中

`add_message`, `add_red_envelope_message`, `add_notice_message`, `add_emoji_message` 指令满足数据基本限制中的相关约束

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 6s, 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 `Runner` 的代码实现。**
- 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 ahuangjiawei@qq.com 或私聊助教进行举报，**经核实后，将直接作为无效作业处理**。