面向对象规格化设计系列第三次代码作业指导书

第零部分：提交要求

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，否则可能以作弊、抄袭论处。

请保证提交项目的顶层目录至少存在两个文件夹：`src` 和 `test`（命名需严格与此保持一致），请将作业的**功能代码**文件存放于 `src` 文件夹下，同时将相关 Junit **测试代码**文件存放于 `test` 文件夹下，以保证评测的正常进行（评测时**只会**针对 `src` 目录下的文件进行**程序功能**的评测以及**代码风格**检测，也就是说，`test` 目录下的 Junit 测试代码风格**不会**被检测）。参考目录结构如下：

```
|-src
  |- MainClass.java
  |- ...
|-test
  |- Test.java
  |- ...
```

注意：为了通过 Junit 测试的编译，请大家实现课程组提供的接口时不要分包（在 src 下创建子目录），而是将所有实现接口的类都直接放在 src 目录下，否则本地运行正常的 Junit 测试类代码在评测机上会无法找到课程组提供的待测试类文件。

第一部分：训练目标

本次作业中，需要完成的任务为实现简单社交关系的模拟和查询；学习目标为对规格化开发（以入门级 JML 规格为例）的理解与相应的代码实现。

第二部分：预备知识

需要同学们了解基本的 JML 语法和语义，以及具备根据 JML 给出的规格编写 Java 代码的能力。JML 教程可参考仓库内 JML Level 0 使用手册。

注意：为了简化代码，便于同学们阅读，我们对 JML 进行了一定程度的拓展。对于标注了 `safe` 的方法，只可以保证 JML 描述内容，可能有 side effect，伴随容器长度的增减或者对象的修改，但是不可以有如下的 side effect，具体体现为：

1. 不可在任何容器或对象中增加 JML 没有要求加入的对象。
2. 不可在任何容器或对象中删除 JML 没有要求删除的对象。
3. 不可对 JML 描述中涉及之外的对象或涉及对象中的非涉及属性进行内容的修改，即 JML 涉及之外的对象或属性的 object representation（对应二进制序列）应该前后一致。

第三部分：题目描述

一、作业基本要求

本次作业要求同学们升级已有的社交网络，完成对消息收发的模拟。

- 社交网络的整体框架官方已经给出了 JML 表述并提供了相应接口。同学们需要**阅读 JML 规格，依据规格实现自己的类和方法。**

  具体来说，各位同学需要在上一次作业的基础上，一方面继续维护已有的 `Person`、`Network`、`Tag`、`OfficialAccount` 类，根据官方包的调整新增一些查询方法；另一方面新建一个 `Message` 类及其三个子类 `EmojiMessage`、`RedEnvelopeMessage`、`ForwardMessage`，并分别实现官方包中提供的 `MessageInterface`、`EmojiMessageInterface`、`RedEnvelopeMessageInterface`、`ForwardMessageInterface` 接口，最终类中每个方法的代码实现都需要**严格满足**接口中给出的 JML 规格定义。

- 阅读指导书中关于**异常**行为的描述，结合官方包中提供的异常类的 **javadoc**，体会异常处理的流程。

  异常类已在官方包内给出，这一部分**没有提供 JML 规格**，而是提供了一些相对标准的文档注释来向大家介绍类或方法的功能和参数的含义。各位同学需要仔细阅读指导书中关于异常类的详细描述，恰当地使用这些异常类，正确处理我们规定的各种异常情况，并保证所有的 print() 方法能够正确输出指定的信息。

此外，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`、`Tag`、`OfficialAccount`、`Message`、`EmojiMessage`、`RedEnvelopeMessage`、`ForwardMessage` 类，来使得程序完整可运行，具体形式下文中有提示。

二、类规格要求

注意：

- 与前两次作业一样，同学们需要保证实现**每一个**官方包接口的类命名为**接口名去掉 Interface**，比如实现 `MessageInterface` 接口的类应命名为 `Message`。
- JUnit 评测时，课程组提供的评测代码同样满足上述命名规则。

所有类的具体接口规格见官方包的代码，此处不加赘述。

请确保各个类的**构造方法**正确实现，且类和构造方法均定义为 `public`。`Runner` 内将自动获取**符合下方说明的**构造方法来构造各个类的实例。

Person 类 【Modify】

构造方法的要求与上次作业保持一致，需要新增以下属性：

- socialValue：社交值，初始值为 0
- money：钱数，初始值为 0，后续可以是负数

此外，还需要需要新增一个属性数组 `messages`，具体表述参见 `PersonInterface` 接口。

Network 类 【Modify】

构造方法的要求与上次作业保持一致，仅需要新增三个属性数组 `messages`、`emojiIdList` 和 `emojiHeatList`，具体表述参见 `NetworkInterface` 接口。

Message 类

构造方法，用以生成和初始化一个 Message 对象，注意 `Message` 类及其所有子类都需要提供**两个构造方法**供不同情况调用（请按照下面的 JML 正确实现）：

```
public class Message implements MessageInterface {

    /*@ ensures type == 0;
      @ ensures tag == null;
      @ ensures id == messageId;
      @ ensures socialValue == messageSocialValue;
      @ ensures person1 == messagePerson1;
      @ ensures person2 == messagePerson2;
      @*/
    public Message(int messageId, int messageSocialValue, PersonInterface
messagePerson1, PersonInterface messagePerson2);

    /*@ ensures type == 1;
      @ ensures person2 == null;
      @ ensures id == messageId;
      @ ensures socialValue == messageSocialValue;
      @ ensures person1 == messagePerson1;
      @ ensures tag == messageTag;
      @*/
    public Message(int messageId, int messageSocialValue, PersonInterface
messagePerson1, TagInterface messageTag);
}
```

属性:

- id：在整个程序运行过程中的所有时刻，在当前 `Network` 中出现过的所有 `Message` 对象中独一无二的 `id`
- socialValue：消息的社交值
- type：消息的种类（取值为 0 或 1）
- person1：消息的发送者
- person2：消息的接收者（可为空）
- tag：消息的接收标签（可为空）

EmojiMessage 类

构造方法，用以生成和初始化一个 EmojiMessage 对象，注意 `EmojiMessage` 类需要提供**两个构造方法**供不同情况调用（请按照下面的 JML 正确实现）：

```
public class EmojiMessage implements EmojiMessageInterface {

    /*@ ensures type == 0;
      @ ensures tag == null;
      @ ensures id == messageId;
      @ ensures person1 == messagePerson1;
      @ ensures person2 == messagePerson2;
      @ ensures emojiId == emojiNumber;
      @*/
    public EmojiMessage(int messageId, int emojiNumber, PersonInterface
messagePerson1, PersonInterface messagePerson2);

    /*@ ensures type == 1;
```

```
      @ ensures person2 == null;
      @ ensures id == messageId;
      @ ensures person1 == messagePerson1;
      @ ensures tag == messageTag;
      @ ensures emojiId == emojiNumber;
      @*/
    public EmojiMessage(int messageId, int emojiNumber, PersonInterface
messagePerson1, TagInterface messageTag);
}
```

属性:

- id：在整个程序运行过程中的所有时刻，在当前 `Network` 中出现过的所有 `Message` 对象中独一无二的 `id`
- emojiId：表情编号
- type：消息的种类（取值为 0 或 1）
- person1：消息的发送者
- person2：消息的接收者（可为空）
- tag：消息的接收标签（可为空）

RedEnvelopeMessage 类

构造方法，用以生成和初始化一个 RedEnvelopeMessage 对象，注意 `RedEnvelopeMessage` 类需要提供**两个构造方法**供不同情况调用（请按照下面的 JML 正确实现）：

```
public class RedEnvelopeMessage implements RedEnvelopeMessageInterface {

    /*@ ensures type == 0;
      @ ensures tag == null;
      @ ensures id == messageId;
      @ ensures person1 == messagePerson1;
      @ ensures person2 == messagePerson2;
      @ ensures money == luckyMoney;
      @*/
    public RedEnvelopeMessage(int messageId, int luckyMoney, PersonInterface
messagePerson1, PersonInterface messagePerson2);

    /*@ ensures type == 1;
      @ ensures person2 == null;
      @ ensures id == messageId;
      @ ensures person1 == messagePerson1;
      @ ensures tag == messageTag;
      @ ensures money == luckyMoney;
      @*/
    public RedEnvelopeMessage(int messageId, int luckyMoney, PersonInterface
messagePerson1, TagInterface messageTag);
}
```

属性:

- id：在整个程序运行过程中的所有时刻，在当前 `Network` 中出现过的所有 `Message` 对象中独一无二的 `id`
- money：红包的金额
- type：消息的种类（取值为 0 或 1）
- person1：消息的发送者
- person2：消息的接收者（可为空）

- tag：消息的接收标签（可为空）

ForwardMessage 类

构造方法，用以生成和初始化一个 ForwardMessage 对象，注意 `ForwardMessage` 类需要提供**两个构造方法**供不同情况调用（请按照下面的 JML 正确实现）：

```
public class ForwardMessage implements MessageInterface {

    /*@ ensures type == 1;
      @ ensures person2 == null;
      @ ensures id == messageId;
      @ ensures person1 == messagePerson1;
      @ ensures tag == messageTag;
      @ ensures articleId == article;
      @*/
    public ForwardMessage(int messageId, int article, PersonInterface
messagePerson1, PersonInterface messagePerson2);

    /*@ ensures type == 1;
      @ ensures person2 == null;
      @ ensures id == messageId;
      @ ensures person1 == messagePerson1;
      @ ensures tag == messageTag;
      @ ensures articleId == article;
      @*/
    public ForwardMessage(int messageId, int article, PersonInterface
messagePerson1, TagInterface messageTag);
}
```

属性：

- id：在整个程序运行过程中的所有时刻，在当前 `Network` 中出现过的所有 `Message` 对象中独一无二的 `id`
- articleId：转发的文章 ID
- type：消息的种类（取值为 0 或 1）
- person1：消息的发送者
- person2：消息的接收者（可为空）
- tag：消息的接收标签（可为空）

异常类

本次作业的官方包中新增了 **4** 个异常类，同学们可以直接调用，官方包会捕获这些异常，并处理异常输出（具体逻辑见官方包中的 Runner 类）。每个异常类的具体说明在官方包中由 javadoc 给出，在此不作赘述，请同学们在实现过程中仔细阅读官方包中的相关代码与说明，正确传递参数，同时 **将官方包中对类和方法的自然语言描述与各个接口中的 JML 进行对比**，思考各自的优劣。

三、需要编写 Junit 单元测试的方法

本次作业中，需要同学们为 `Network` 类中的 `deleteColdEmoji` 方法编写 Junit 单元测试。

> 在单元测试中，你需要对 JML 的全部内容进行检查，除了检验 requires 和 ensures，还有 pure、assignable 语句等等。例如，对于一个 pure 方法，调用方法前后的状态应该一致，如果前后状态不一致，那么我们认为这不符合给定的 JML。

评测时我们会使用若干正确代码与错误代码进行测试，保证错误代码仅 **deleteColdEmoji** 出现错误，且**保证不涉及** `Tag` **类、** `OfficialAccount` **和** `Person` **类有关的错误**，其余官方包要求方法均正确实现，需要同学们编写的单元测试正确判断代码的 `deleteColdEmoji` 方法是否出现错误。

注意：

在 JUnit 测评时给出的样例中，`Network` 类会提供 `MessageInterface[] getMessages()`、`int[] getEmojiIdList()` 和 `int[] getEmojiHeatList()` 方法供同学调用，其中：

- `getMessages()` 返回一个 `MessageInterface` 数组，该数组是 `Network` 中 `messages` 数组的浅拷贝。
- `getEmojiIdList()` 返回一个 `int` 数组，该数组是 `Network` 中 `emojiIdList` 数组的浅拷贝。
- `getEmojiHeatList()` 返回一个 `int` 数组，该数组是 `Network` 中 `emojiHeatList` 数组的浅拷贝。

补充说明：

- 对于同一个**合法的** `index`，记作 `i`，`getEmojiIdList()[i]` 和 `getEmojiHeatList()[i]` 是对应的。
- 如果要比较两个 `MessageInterface` 实例对象是否相等，需要比较其所有属性是否相等，其中**基本类型**属性使用 `==` 比较是否相等，对于**对象类型**属性使用 `equals()` 方法比较是否相等。
- 本次作业的 JUnit 测评样例中，`Network` 类**没有给出**获取 `persons`，`accounts`，`articles`，`articleContributors` 四个容器的方法，同时保证所有样例**没有**涉及这四个容器的错误，但是推荐大家自行设计测试时能够尽可能全面地测试。

第四部分：设计建议

推荐各位同学在课下测试时使用 Junit 单元测试来对自己程序的全部方法进行测试。

第五部分：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。

```
import com.oocourse.spec3.main.Runner;

public class xxx {
    public static void main(String[] args) throws Exception {
        Runner runner = new Runner(Person.class, Network.class, Tag.class,
Message.class,
                EmojiMessage.class, ForwardMessage.class,
RedEnvelopeMessage.class);
        runner.run();
    }
}
```

输入输出格式

输入部分，一行或多行一条指令，形如 `op arg1 arg2 ...`，表示指令类型和参数，部分指令可能跟随若干行额外信息，具体格式见下。

输出部分，每条指令对应一行输出，为指令的执行结果或发生的异常。

输入输出实际由官方包进行处理。

指令格式一览

本次作业新增指令如下（括号内为变量类型）：

```
add_message id(int) social_value(int) type(int) person_id1(int)
person_id2(int)|tag_id(int)
add_emoji_message id(int) emoji_id(int) type(int) person_id1(int)
person_id2(int)|tag_id(int)
add_red_envelope_message id(int) money(int) type(int) person_id1(int)
person_id2(int)|tag_id(int)
add_forward_message id(int) article_id(int) type(int) person_id1(int)
person_id2(int)|tag_id(int)
send_message id(int)
store_emoji_id id(int)
delete_cold_emoji limit(int)

query_social_value id(int)
query_received_messages id(int)
query_popularity id(int)
query_money id(int)
```

上次作业中的指令仍然有效（括号内为变量类型）：

```
add_person id(int) name(String) age(int)
add_relation id1(int) id2(int) value(int)
modify_relation id1(int) id2(int) m_val(int)

add_tag person_id(int) tag_id(int)
del_tag person_id(int) tag_id(int)
add_to_tag person_id1(int) person_id2(int) tag_id(int)
del_from_tag person_id1(int) person_id2(int) tag_id(int)

create_official_account person_id(int) account_id(int) account_name(String)
delete_official_account person_id(int) account_id(int)
contribute_article person_id(int) account_id(int) article_id(int)
article_name(String)
delete_article person_id(int) account_id(int) article_id(int)
follow_official_account person_id(int) account_id(int)

query_value id1(int) id2(int)
query_circle id1(int) id2(int)
query_triple_sum
query_tag_age_var person_id(int) tag_id(int)
query_best_acquaintance id(int)

query_shortest_path id1(int) id2(int)
query_best_contributor account_id(int)
```

```
query_received_articles person_id(int)
query_tag_value_sum person_id(int) tag_id(int)
query_couple_sum

load_network n(int)
load_network_local n(int) file(String)
```

每条指令的执行结果为其对应方法的输出（`void` 方法则为 `Ok`），方法的对应详见官方包。

其中，`load_network` 和 `load_network_local` 指令为复合指令，由官方包实现。

`load_network` 指令后跟随 n+2 行：

- 第一行 n个 id1,id2...idn，第二行 n 个 name：name1,name2...namen，第三行 n 个 age:age1,age2...agen，表示添加 n 个 `Person`，第 i个人对应 idi,namei,agei；
- 接下来n-1行，第 i行 i 个value(value>=0)，其中第j个表示 id{i+1} 和 idj 对应 `Person` 间添加 value 的关系(value>0)或不添加关系 (value=0)。

`load_network_local` 指令与其类似，但仅占一行，而从 `file` 文件中读取所需n+2 行数据。

此二指令会调用 `add_person` 和 `add_relation` 添加 `Person` 和关系，具体实现方式参考官方包。保证复合指令不会出现异常。

其余指令均为一行的简单指令。

注意：

对于 `add_message`、`add_emoji_message`、`add_red_envelope_message`、`add_forward_message` 指令，输入指令可能存在 `person_id1`、`person_id2`、`tag_id` 中某一个在社交网络中**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。

指令的简称

实际评测使用的输入为简称，但官方包提供简称和全称的识别方式，在测试时同学们可以自由选择简称与全称。

指令	简称
add_message	am
add_red_envelope_message	arem
add_forward_message	afm
add_emoji_message	aem
send_message	sm
store_emoji_id	sei
delete_cold_emoji	dce
query_social_value	qsv
query_received_messages	qrm
query_popularity	qp
query_money	qm
create_official_account	coa
delete_official_account	doa
contribute_article	ca
delete_article	da
follow_official_account	foa
query_shortest_path	qsp
query_best_contributor	qbc
query_received_articles	qra
query_tag_value_sum	qtvs
query_couple_sum	qcs
add_person	ap
add_relation	ar
modify_relation	mr
add_tag	at
del_tag	dt
add_to_tag	att
del_from_tag	dft

指令	简称
query_value	qv
query_circle	qci
query_triple_sum	qts
query_tag_age_var	qtav
query_best_acquaintance	qba
load_network	ln
load_network_local	lnl

样例

Case 1

标准输入

```
ap 0 NagasakiSoyo 16
ap 1 ChihayaAnon 16
ap 3 TogawaSakiko 16
ar 1 0 99
ar 0 3 9
am 1 9 0 1 0
am 2 99 0 1 0
am 3 6 0 1 0
sm 1
sm 2
sm 3
arem 4 99 0 0 3
arem 5 13 0 0 3
arem 6 14 0 0 3
sm 4
sm 5
sm 6
qsv 0
qsv 1
qsv 3
qrm 3
qm 0
qm 3
```

标准输出

```
Ok
Ok
Ok
Ok
Ok
Ok
Ok
Ok
Ok
```

```
Ok
Ok
Ok
Ok
Ok
Ok
Ok
Ok
744
114
630
RedEnvelope: 14; RedEnvelope: 13; RedEnvelope: 99
-126
126
```

Case 2

标准输入

```
ap 0 NagasakiSoyo 16
ap 1 ChihayaAnon 16
ap 3 TogawaSakiko 16
ar 1 0 99
ar 0 3 9
sei 6
sei 7
aem 1 6 0 1 0
aem 2 6 0 1 0
aem 3 7 0 1 0
sm 1
sm 2
sm 3
aem 4 6 0 0 3
aem 5 7 0 0 3
sm 4
sm 5
dce 3
qp 6
qp 7
qrm 0
qrm 1
qrm 3
```

标准输出

```
Ok
Ok
Ok
Ok
Ok
Ok
Ok
Ok
Ok
Ok
Ok
```

```
Ok
Ok
Ok
Ok
Ok
Ok
1
3
einf-1, 7-1
Emoji: 7; Emoji: 6; Emoji: 6
None
Emoji: 7; Emoji: 6
```

Case 3

标准输入

```
ln 5
-2 -1 0 1 2
NagasakiSoyo ChihayaAnon KanameRana TakamatsuTomori ShiinaTaki
16 16 16 16 16
10
10 10
10 10 10
10 10 10 10
coa -1 1 MyGo!!!!!
foa 1 1
foa -2 1
foa 2 1
ca -1 1 0 HelloEveryOne
at 1 5
att -2 1 5
att -1 1 5
att 0 1 5
att 1 1 5
att 2 1 5
afm 7 0 1 1 5
sm 7
qra -2
qra 0
```

标准输出

```
Ok
Ok
Ok
Ok
Ok
Ok
Ok
Ok
Ok
Ok
epi-1, 1-1
Ok
Ok
```

```
ok
0 0
0
```

注意

以上样例对本次作业中涉及到的指令进行了较全面的覆盖，但是并不足以测试到所有情况，请大家尽量**全面分析、充分测试。**

本单元的训练栏目中为大家提供了一些测试方法的参考，欢迎大家自主学习、积极思考（不强制要求完成），如果有相关想法也欢迎在作业讨论区分享。

数据范围

公测数据限制

- 指令条数不多于 10000 条。
- 所有 id在 `int` 范围内。
- name 为字符串，长度 |name| 满足不小于 1≤ name ≤100
- 0 < value,age<2000 且为整数。
- −200≤mval≤2000 且为整数。
- 1≤n≤300 且为整数。
- 若出现 `load_network` 指令，则其一定为测试点中的第一条指令，且其中的 id不重。
- 不出现 `load_network_local` 指令。
- $-1000 \leq social\_value \leq 1000$ 且为整数。
- $0 \leq money \leq 200$且为整数。
- $type \in \{0, 1\}$
- limit在 `int` 范围内。

注意:

此处说明的是**输入**的数据限制，部分同名属性经修改可能超出该范围。

互测数据限制

- 指令条数不多于 30003000 条。
- 1≤*n*≤100 且为整数。
- 其他同公测限制相同。

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，从而测试各个类的实现正确性，即是否满足 JML 规格的定义或者指导书描述。

可以认为，只要所要求的所有类的具体实现严格满足 JML，同时异常处理符合指导书和官方包的描述，就能保证正确性，但是不保证满足时间限制。

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 **10s**，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。**

第六部分：补充说明

关于提交代码部分的文件结构：

src 目录下包含主入口类（例如 `MainClass.java`），同学们实现官方包接口的类（`Network`，`Person`，`Tag`，`OfficialAccount`，`Message`，`EmojiMessage`，`RedEnvelopeMessage`，`ForwardMessage`）以及同学们可能自行设计的辅助类。

请注意：提交的时候请不要在 src 目录下包含官方包，同学们自己实现的官方包接口的类的命名还请按照约定命名，并直接放在 src 目录下，不要嵌套子目录，否则和测试程序一起编译的时候会无法通过。

test 目录下设置测试类（例如 Test.java），以及同学们可能自己设计的辅助类。

关于评测机制：

受到系统限制，我们只能统一编译 `src` 文件夹和 `test` 文件夹，同学提交代码之前请确保 `src` 和 `test` 文件夹下本地的静态编译能通过。

对于普通测试点，我们使用同学提交的 `src` 目录中代码运行结果判断正误。

对于 JUnit 测试点，和实验类似，每一个数据点对应一份课程组提供的代码（也就是同学们的 src 文件夹中的部分），在测试时是基于课程组提供的测试代码运行的，可以理解为你的测试类测试的是课程组提供的测试代码。课程组提供的代码中类的命名也按照约定命名，且保证除了需要测试的 `deleteColdEmoji` 之外的方法都正确实现，同学们需要自行构造数据，调用 `deleteColdEmoji` 方法，按照 JML 中的要求进行测试。

注意：使用 JUnit 单元测试的过程中请避免使用 `assert()` 进行逻辑断言，而是使用 JUnit 测试框架下的断言类 `org.junit.Assert` 中的对应方法进行判定。

关于 JUnit 评测限制：

对于 `deleteColdEmoji` 的正确性检查部分，课程组提供的错误测试点的 bug 比较明显，不会出现需要用很刁钻的数据才能覆盖的情况。

再次强调，Network.java 文件一定要**直接**放在 src 目录下，否则本地正常工作的代码在评测机上运行 JUnit 评测时会找不到课程组提供的代码文件。

说明：JUnit评测只在公测存在，互测和强测不存在。

关于编译的说明：

由于评测是 src 文件夹和 test 文件夹统一编译的，同学们如果在 test 文件夹中的测试类使用了课程组提供的 `getMessages`、`getEmojiIdList` 和 `getEmojiHeatList` 方法，请在 src 文件夹下的 `Network` 类中实现这三个方法（不必正确实现方法，写任意能通过编译的内容都可）。以下是一种解决方式的参考：

```
public class Network implements NetworkInterface {
    // ... 同学自己编写过的其他属性和方法，此处省略

    /* 声明方法即可，内容任意，仅仅是为了通过评测时的编译，不会被调用，不过建议大家自己实现以本地测试 junit */
    public MessageInterface[] getMessages() { return null; }
    public int[] getEmojiIdList() { return null; }
    public int[] getEmojiHeatList() { return null; }
}
```

一、提示

- 请同学们参考源码，注意本单元中一切叙述的讨论范围实际限定于全局唯一的 Network 实例中
- 本次作业中可以自行组织代码结构。任意新增 java 代码文件。只需要保证题目要求的几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。**
- 仓库地址：第十一次作业公共仓库

二、警示

- **请勿试图对官方接口进行操作。**此外，在互测环节中，如果发现有人试图 hack 输出接口，请联系助教，经核实后，**将直接作为无效作业处理。**