

**2019年春季学期  
计算机学院《软件构造》课程**

**Lab 2实验报告**

|  |  |
| --- | --- |
| 姓名 | 强文杰 |
| 学号 | 1171000410 |
| 班号 | 1703005 |
| 电子邮件 | [672334335@qq.com](mailto:672334335@qq.com) |
| 手机号码 | 18800421389 |

**目录**

[1 实验目标概述 1](#_Toc1988214)

[2 实验环境配置 1](#_Toc1988215)

[3 实验过程 1](#_Toc1988216)

[3.1 Poetic Walks 1](#_Toc1988217)

[3.1.1 Get the code and prepare Git repository 1](#_Toc1988218)

[3.1.2 Problem 1: Test Graph <String> 1](#_Toc1988219)

[3.1.3 Problem 2: Implement Graph <String> 1](#_Toc1988220)

[3.1.3.1 Implement ConcreteEdgesGraph 2](#_Toc1988221)

[3.1.3.2 Implement ConcreteVerticesGraph 2](#_Toc1988222)

[3.1.4 Problem 3: Implement generic Graph<L> 2](#_Toc1988223)

[3.1.4.1 Make the implementations generic 2](#_Toc1988224)

[3.1.4.2 Implement Graph.empty() 2](#_Toc1988225)

[3.1.5 Problem 4: Poetic walks 2](#_Toc1988226)

[3.1.5.1 Test GraphPoet 2](#_Toc1988227)

[3.1.5.2 Implement GraphPoet 2](#_Toc1988228)

[3.1.5.3 Graph poetry slam 2](#_Toc1988229)

[3.1.6 Before you’re done 2](#_Toc1988230)

[3.2 Re-implement the Social Network in Lab1 2](#_Toc1988231)

[3.2.1 FriendshipGraph类 2](#_Toc1988232)

[3.2.2 Person类 3](#_Toc1988233)

[3.2.3 客户端main() 3](#_Toc1988234)

[3.2.4 测试用例 3](#_Toc1988235)

[3.2.5 提交至Git仓库 3](#_Toc1988236)

[3.3 Playing Chess 3](#_Toc1988237)

[3.3.1 ADT设计/实现方案 3](#_Toc1988238)

[3.3.2 主程序ChessGame设计/实现方案 3](#_Toc1988239)

[3.3.3 ADT和主程序的测试方案 3](#_Toc1988240)

[3.4 Multi-Startup Set (MIT) 4](#_Toc1988241)

[4 实验进度记录 4](#_Toc1988242)

[5 实验过程中遇到的困难与解决途径 4](#_Toc1988243)

[6 实验过程中收获的经验、教训、感想 4](#_Toc1988244)

[6.1 实验过程中收获的经验和教训 4](#_Toc1988245)

[6.2 针对以下方面的感受 4](#_Toc1988246)

# 实验目标概述

根据实验手册简要撰写。

本次实验训练抽象数据类型（ADT）的设计、规约、测试，并使用面向对象 编程（OOP）技术实现 ADT。

具体来说： 针对给定的应用问题，从问题描述中识别所需的 ADT；

设计 ADT 规约（pre-condition、post-condition）并评估规约的质量；

根据 ADT 的规约设计测试用例；

ADT 的泛型化；

根据规约设计 ADT 的多种不同的实现；针对每种实现，设计其表示 （representation）、表示不变性（rep invariant）、抽象过程（abstraction function）

使用 OOP 实现 ADT，并判定表示不变性是否违反、各实现是否存在表 示泄露（rep exposure）；

测试 ADT 的实现并评估测试的覆盖度；

使用 ADT 及其实现，为应用问题开发程序；

在测试代码中，能够写出 testing strategy 并据此设计测试用例。

# 实验环境配置

简要陈述你配置本次实验所需环境的过程，必要时可以给出屏幕截图。

特别是要记录配置过程中遇到的问题和困难，以及如何解决的。

1. 根据助教在群里发的链接，完成Eclipse和JDK11的安装，然后去网上查询Java运行环境的配置，配置计算机的环境变量。

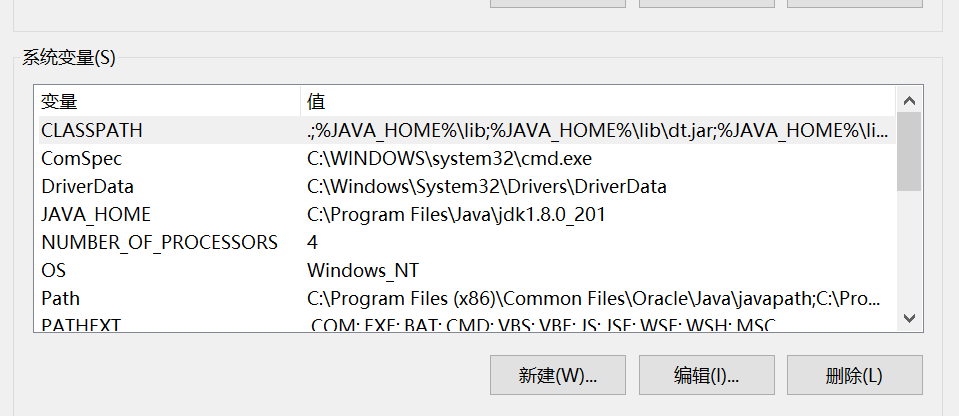


图2-1 环境变量配置

2.注册GitHub账号，并且授权GitHub classroom，获取lab2的GitHub classroom的URL地址。

3. Git代码配置管理。首先安装最新版Git，获取个人GitHub的URL地址，通过在Git bash输入指令建立管理本地仓库，将文件push到远程仓库。

git push origin master

**遇到的困难：**

1.在配置环境时，最早的参考教程是针对Mac OS，并且本地路径的设置出错，导致走了很多弯路。

2. 在安装Git之后，未第一时间在Git的安装目录注册user.name和user.email，导致后面输入命令时bash显示commend not found。

在这里给出你的GitHub Lab2仓库的URL地址（Lab2-学号）。

URL地址：<https://github.com/ComputerScienceHIT/Lab2-1171000410.git>

# 实验过程

请仔细对照实验手册，针对三个问题中的每一项任务，在下面各节中记录你的实验过程、阐述你的设计思路和问题求解思路，可辅之以示意图或关键源代码加以说明（但千万不要把你的源代码全部粘贴过来！）。

## Poetic Walks

在这里简要概述你对该任务的理解。

我们需要构造一个Graph，实现spec中的基本功能，并且以边和点两种方式实现接口，并且需要将ADT泛型化。

编写测试文件的时候，需要从实现的基本功能出发，有良好的测试覆盖率。

对于problem4，我们则需要根据文件中的输入，构造poet，再由input和图中的映射关系，得出最后的poem。

### Get the code and prepare Git repository

如何从GitHub获取该任务的代码、在本地创建git仓库、使用git管理本地开发。

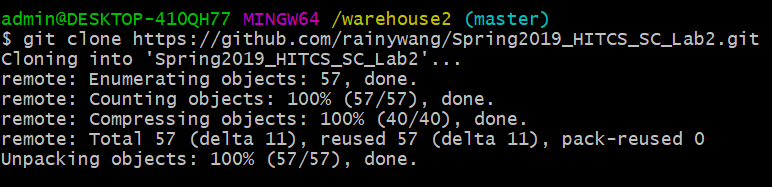
创建本地仓库，添加远程仓库：

git init

git remote add origin <https://github.com/ComputerScienceHIT/Lab2-1171000410.git>

从GitHub上获取该任务的代码：

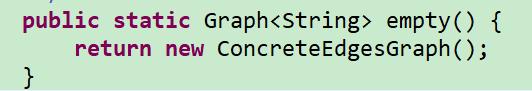
git clone



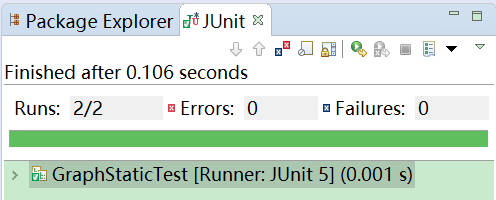
### Problem 1: Test Graph <String>

以下各部分，请按照MIT页面上相应部分的要求，逐项列出你的设计和实现思路/过程/结果。

我选用ConcreteEdgesGraph作为Graph<String>的具体实现，如需要测试，将Graph里原empty()方法修改为：



此时运行**GraphStaticTest**即可得到测试结果：



**GraphInstanceTest.java**：

针对这个测试，我们可使用抽象类，针对边和顶点两种实现的测试类，并且我们所有的测试都与具体的实现无关。

在测试中，使用emptyInstance()方法来获取新的空图，所有的测试策略都根据需要实现的功能来设计。

具体的strategy已在test测试文件中标注，如下：

// 使用抽象类

// 针对两个测试类

// 此test与具体实现无关

// Testing strategy

//

// 测试Graph.add()方法：

// 如果已经存在该顶点，修改graph，顶点数加1

// 否则图不变

// 通过顶点个数的变化来测试

//

// 测试Graph.remove()方法：

// 如果不存在该点，图不变

// 否则删除该点

// 相邻的边也要删除

// 通过顶点个数变化来测试

//

// 测试Graph.set()方法：

// 分开测试weight的情况 : 0 , > 0

// weight为0检查边是否被删除

// weight > 0 , 检查边是否被添加或更新

// 观察source , target 的映射关系变化

//

// 测试Graph.vertices()方法：

// graph为空

// graph添加顶点后

// 观察顶点的数量

//

// 测试Graph.sources()方法

// graph为空

// set点和边

// 观察source数量 , 对应关系是否正确

//

// 测试Graph.targets()方法

// graph为空

// set点和边

// 观察targets数量 , 对应关系是否正确

### Problem 2: Implement Graph <String>

以下各部分，请按照MIT页面上相应部分的要求，逐项列出你的设计和实现思路/过程/结果。

#### Implement ConcreteEdgesGraph

**一. ConcreteEdgesGraph部分：**

**(先介绍ConcreteEdgesGraph，后Edge类，最后测试类)**

// Abstraction function:

// 通过对graph中边和点的抽象

// 边的类中包含了source,target,weight

// 带方向的边

// 构成有向图

//

// Representation invariant:

// 不变量在于隐含的数学关系

// n个点，最多构成n\*(n-1)条有向边

//

// Safety from rep exposure:

// 在可操作的情况下，所有的变量都定义为private , final

// 顶点和边是mutable类型, 因此多处使用:

// 防御式拷贝

// 使用Collections.unmodifiableSet等方法

**构造函数：public** ConcreteEdgesGraph() {

}

**private void checkRep():**

思路：n个点，最多构成n\*(n-1)条有向边，因此存在这种不可变的数学关系

**public** **boolean** add(String vertex)

思路：如果在vertices的Set集合中成功添加了顶点string，则返回true

**private** **int** findEdge(String source, String target)

思路：My method findEdge() ，为了找到list是否存在一条指定的边，该边的source和target必须与传入的参数相等，返回该边的index

**public** **int** set(String source, String target, **int** weight)

思路：根据parameter找到指定边，并调用findEdge()返回index。

当weight>0，如果index<0，没找到指定边，则添加顶点和边；如果index>0，找到了指定边，则update这条边。

当weight>0且index<0时，找到了指定边，将其删除

最后checkRep()。

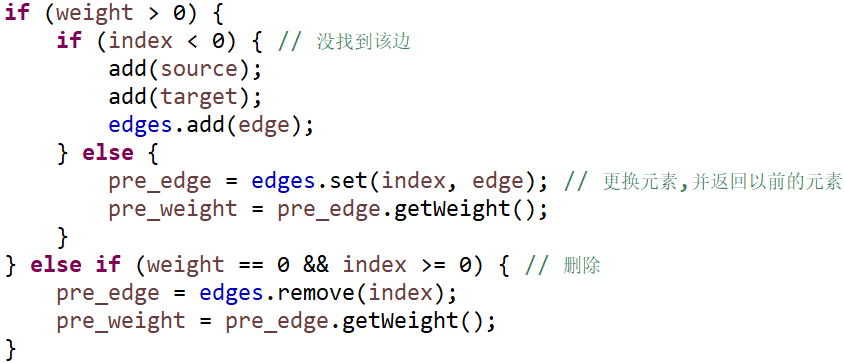


图3.1.1 set主要部分

**public** **boolean** remove(String vertex)

思路：如果不含该点，返回false。否则遍历edges，如果某个edge的source或是target与vertex相等，则删除该边。最后删除vertex点。并checkRep。

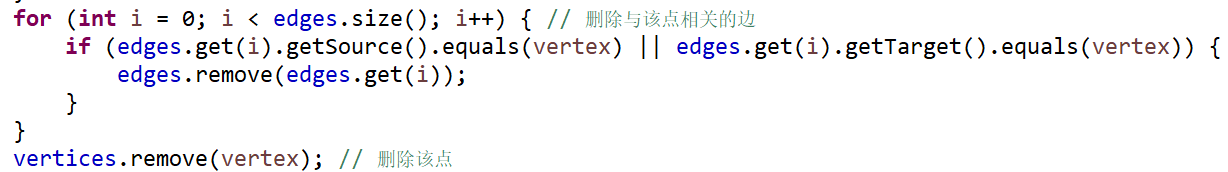


图3.3.2 remove主要部分

**public** Set<String> vertices()

思路：返回vertices的Set，注意做到safety from rep exposure ，使用Collections.unmodifiableSet()方法。



**public** Map<String, Integer> sources(String target)

思路：建立一个map，遍历edges，如果某个edge的edge.getTarget()和传入参数target相等，则将该边的source和weight存入map中。

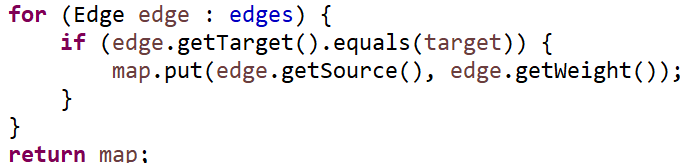


图3.3.3 sources主要部分

**public** Map<String, Integer> targets(String source)

思路：建立一个map，遍历edges，如果某个edge的edge.getSource()和传入参数source相等，则将该边的target和weight存入map中。

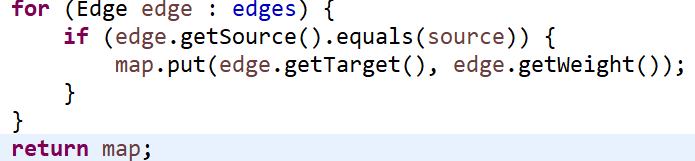


图3.3.4 targets主要部分

**public** String toString()

思路：graph空，则返回“This is An Empty Graph”.

graph不为空，则将每个边的toString连接起来，调用String.concat()方法。

注意连接后为了可读性，有“\n”.

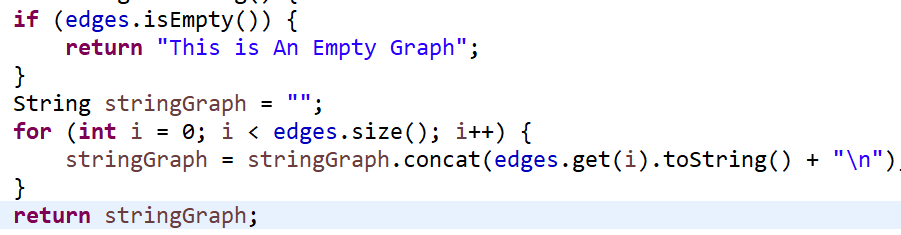


图3.3.5 toString主要部分

**实现 clss Edge ：**

1. fields如下：

**private** String source;

**private** String target;

**private** **int** weight;

// Abstraction function:

// 设置了边有向边必备的参数

// source , target , 和权值weight

//

// Representation invariant:

// source和target不能是null

// weight >= 0

//

// Safety from rep exposure:

// 尽量使用private和final来定义内部属性

// 使用immutable数据类型

2. 构造器 constructor

**public** Edge(String source, String target, **int** weight) {

**this**.source = source;

**this**.target = target;

**this**.weight = weight;

checkRep();

}

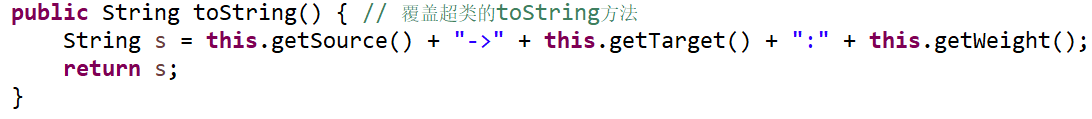
3. checkRep()：

source和target不能是null，并且 weight >= 0

4. 三个field的get函数

5. toString()方法：

输出形式：V1->V2:weight

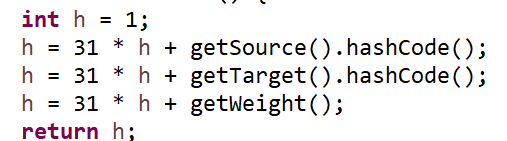


6. equals()方法：

思路：判断两条边是否相等 ，先判断一个对象是否是另一个对象的实例 ，如果是再判断source ,target ,weight等是否都相等.

7. hashCode()方法

复写hashCode方法，@return hash address



**测试类主要思路**：

// Testing strategy for ConcreteEdgesGraph.toString()

// 当Graph为空的时候

// 当Graph不为空的时候

// 匹配string，判断是否相等

// Testing strategy for Edge

// 根据实现的功能来制定相应的test

//

// 测试getSource()方法:

// 建立特定edge

// 返回source

// 判断是否匹配

//

// 测试getTarget()方法:

// 建立特定edge

// 返回target

// 判断是否匹配

//

// 测试getWeight()方法:

// 建立特定edge

// 返回weight

// 判断是否匹配

//

// 测试Edge里的toString()方法：

// 设置指定的边

// 判断字符串是否与指定字符串相等

//

// 测试equals()方法：

// 设置多条边

// 相等：和自己，和其他边

// 不等：source,target,weight存在不相等

//

// 测试hashCode()方法：

// 相等：this和that相等

// 不相等：this和that不相等

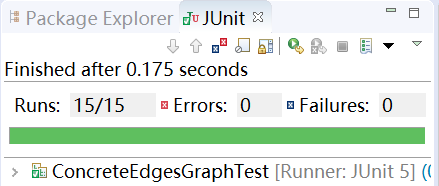


图3.3.6 ConcreteEdgesGraphTest测试结果

#### Implement ConcreteVerticesGraph

**（先介绍ConcreteVerticesGraph类，再Vertex类，最后测试类）**

// Abstraction function:

// 将有向加权图描述为多个顶点

// 点之间的映射关系为边

// 边有权值weight

//

// Representation invariant:

// 每个顶点只能存在一个实例

// 因此顶点个数vertices()的大小相等

//

// Safety from rep exposure:

// 变量尽可能定义为private和final

// 防御式编程

**构造函数：**

**public** ConcreteVerticesGraph() {

}

**public** **void** checkRep()

思路：每个顶点只能存在一个实例，因此顶点个数vertices()的大小相等

**public** **boolean** add(String vertex)

思路：若vertices()中已包含vertex，返回false，否则新建一个顶点将其加入vertices即可。

**private** **int** findVertex(String str)

思路：My method findVertex , 找到对应str的vertex所在的位置，返回index。

**public** **int** set(String source, String target, **int** weight)

思路：如果存在vertices()中找到source，使用findVertex返回对应index，从而找到源点，否则以source为string创建一个新的源点，并将它添加进vertices。对于target操作同理。

得到源点和目标点后，分别对目标点调用setSource，源点调用setTarget即可。

最后checkRep并返回previous weight。

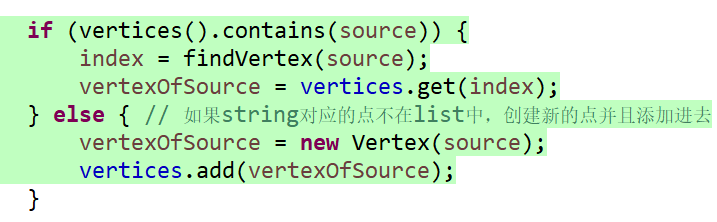


图3.3.7 找到vertexOfSource

**public** **boolean** remove(String vertex)

思路：如果vertices()不包含vertex，返回false。否则遍历所有点，如果某点和vertex存在映射关系，则将这种关系删除。最后将vertex对应的点从vertices中删除即可。

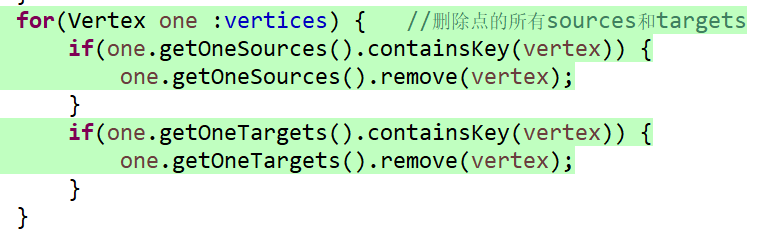


图3.3.8 删除点的所有sources和targets

**public** Set<String> vertices()

思路：遍历vertices，找到每个点对应的string，添加进set即可。

**public** Map<String, Integer> sources(String target)

思路：如果找不到target对应的点，返回Collections.emptyMap()。否则调用getOneSource返回target对应的源点图。

**public** Map<String, Integer> targets(String source)

思路：如果找不到source对应的点，返回Collections.emptyMap()。否则调用getOneTarget返回source对应的目标点图。

**public** String toString()

思路：graph空，则返回“This is An Empty Graph”.

graph不为空，则将每个点的toString连接起来，调用String.concat()方法。

注意连接后为了可读性，有“\n”.

**实现class Vertex：**

// Abstraction function:

// 顶点类刻画图的关键要素

// 使用HashMap存取映射关系

// key为该点的source或target , value为该点的weight

//

// Representation invariant:

// 每个顶点的source或target不能是自身

// HashMap中的values必须不小于0

//

// Safety from rep exposure:

// 所有fields是private final

// String是imutable类型

// 防御性编程

1.fields

**private** String str;

**private** Map<String, Integer> oneSources = **new** HashMap<String, Integer>();

**private** Map<String, Integer> oneTargets = **new** HashMap<String, Integer>();

2. /\*构造函数\*/

**public** Vertex(String str) {

**this**.str = str;

}

3.checkRep()

思路：检查不变量，保证该点不会是自己的source或者target，weight不小于0即可。

4.fields的get函数

思路：使用this返回相应的field即可。

5. **public** **int** setSource(String source, **int** weight)

思路：设置到该点的source，处理同ConcreteVerticesGraph中set方法

\* weight = 0 , source存在 ：移除

\* weight > 0 , source存在 ：更新

\* weight > 0 , source不存在 ：添加

6. **public** **int** setTarget(String target, **int** weight)

思路：设置到该点的target，处理同ConcreteVerticesGraph中set方法

\* weight = 0 , target存在 ：移除

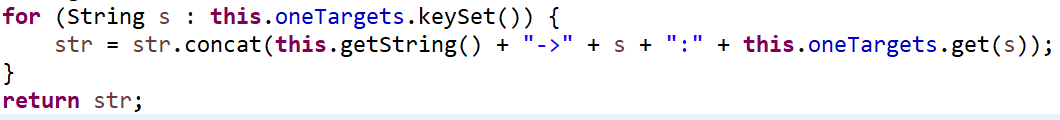
\* weight > 0 , target存在 ：更新

\* weight > 0 , target不存在 ：添加

7. **public** String toString()

思路：遍历该点oneTargets的keyset，输出格式：

V1->V2:weight 。再连接到string并输出。



**测试类主要思路：**

// Testing strategy for ConcreteVerticesGraph.toString()

// 测试toString()方法：

// 当Graph为空的时候

// 当Graph不为空的时候

// 匹配string，判断是否相等

// Testing strategy for Vertex

// 测试getString()方法：

// 设置特定点和其字符串

// 判断该点获取到的字符串和以上字符串是否相等

//

// 测试getOneSources()方法：

// 设置特定点

// 点的source为空

// 点的source不为空：sources个数，sources字符

// 判断是否相等

//

// 测试getOneTargets()方法：

// 设置特定点

// 点的target为空

// 点的target不为空：targets个数，targets字符

// 判断是否相等

//

// 测试setSource()方法：

// 分开测试weight的情况 : 0 , > 0

// weight为0检查source是否被删除

// weight > 0 , 检查边是否被添加或更新

// 若更新则边的权值会变化

// 包括previous weight的测试

//

// 测试setTarget()方法：

// 分开测试weight的情况 : 0 , > 0

// weight为0检查target是否被删除

// weight > 0 , 检查边是否被添加或更新

// 若更新则边的权值会变化

// 包括previous weight的测试

//

// 测试Vertex中toStiring()方法

// 设置source,target,weight

// 判断输出字符串与指定字符串是否相等

### Problem 3: Implement generic Graph<L>

#### Make the implementations generic

将具体类的声明更改为：

public class ConcreteEdgesGraph<L> implements Graph<L> { ... }

class Edge<L> { ... }

和：

public class ConcreteVerticesGraph<L> implements Graph<L> { ... }

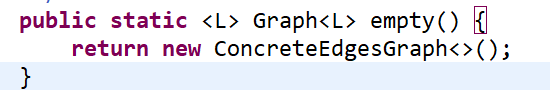
class Vertex<L> { ... }

更新两个实现以支持任何类型的顶点标签，使用占位符L代替String。

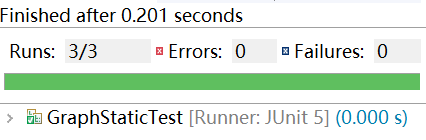
以前，可能已声明类型为Edge或的变量List<Edge>。那些将需要成为Edge<L>和List<Edge<L>>。

同样，可能已经调用了类似new ConcreteEdgesGraph()或的构造函数new Edge()。例如，那些将需要成为new ConcreteEdgesGraph<String>()和new Edge<L>()。

#### Implement Graph.empty()

**选择ConcreteEdgesGraph来使用和实施Graph.empty()**

为了确信Graph确实支持不同类型的标签，我在GraphStaticTest增加了标签类型Integer。其中调用的Graph中所提供的各种函数，全部测试通过。



### Problem 4: Poetic walks

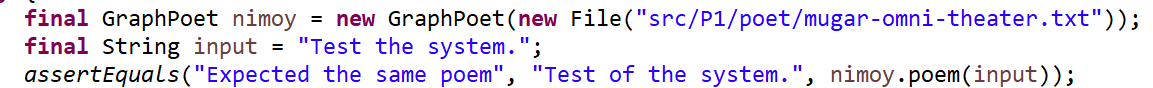
#### Test GraphPoet

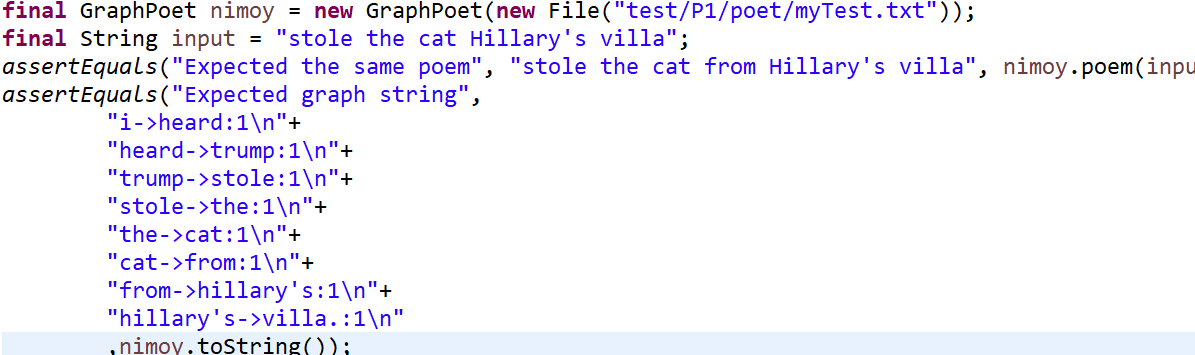
// Testing strategy

// 给定一个input。从文件中读取poet，调用Graph.poem()后

// 观察输出与预期是否相等

对mugar-omni-theater.txt和自定义的test/P1/poet/myTest.txt测试分别如下：





#### Implement GraphPoet

fields：**private** **final** Graph<String> graph = Graph.*empty*(); //有向图

**private** **final** List<String> words = **new** ArrayList<String>(); //文本的单词

// Abstraction function:

// 将输入的文本单词提取作为顶点

// 构建有向图

// 转化为poem

//

// Representation invariant:

// 输入的文本words不为空

// 有向图不为null

//

// Safety from rep exposure:

// 所有fields都是private final

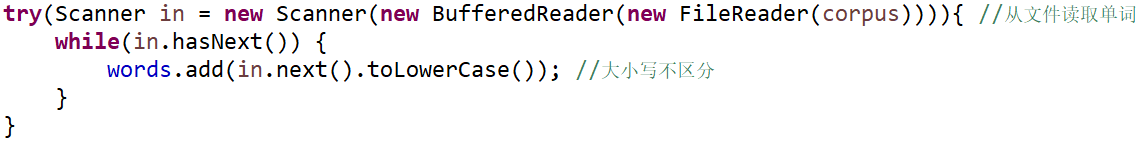
// 防御式编程

方法：

1. **public** GraphPoet(File corpus) **throws** IOException

从语料库的图形中创建一个新的poet。

先读文件，并把文件中的单词存在words中。使用BufferedReader读取文本文件中的数据，类Scanner用于将输入的文本分解成多个部分。



然后调用Graph类中的方法，将单词转化为图，添加顶点，set边，其中权值全部设置为1.

2. **public** **void** checkRep()

检查图和words不为空。

3. **public** String poem(String input)

String[] newWords = input.split("\\s"); //空格回车换行等空白符

StringBuilder poem = **new** StringBuilder(input); //为了方便后面高效插入字符串

遍历input中所有单词，调用Graph.targets()和Graph.sources()方法。如果该单词的targets和后面一个单词的sources有交集，则添加一条bridge，并且在两个单词的bridge中随机选择一个插入到字符串中。

4. **public** String toString()

调用Graph.tostring()方法。

#### Graph poetry slam

### Before you’re done

请按照[http://web.mit.edu/6.031/www/sp17/psets/ps2/#before\_youre\_done](http://web.mit.edu/6.031/www/sp17/psets/ps2/" \l "before_youre_done)的说明，检查你的程序。

git add Lab2\_1171000410

git commit -m “第一次提交”

git push origin master

在这里给出你的项目的目录结构树状示意图。

项目名称： Lab2\_1171000410

src

P1

graph

….java

poet

… .java

… .txt

test

P1

graph

…Test.java

poet

… Test.java

… .txt

## Re-implement the Social Network in Lab1

在这里简要概述你对该任务的理解。

这个实验是基于在Poetic Walks中定义的Graph及其两种实现，重新实现Lab1中的 FriendshipGraph类。我们需要尽可能复用ConcreteEdgesGraph或 ConcreteVerticesGraph中已经实现的add()和set()方法，而不是从零开始。另外基于所选定的 ConcreteEdgesGraph 或 ConcreteVerticesGraph的rep来实现，而不能修改父类的rep。

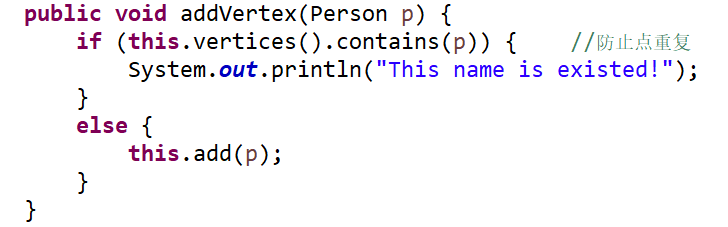
### FriendshipGraph类

**设计和实现思路：**继承ConcreteEdgesGraph<Person>，根据实验指导的提示，类中需要增加一些非法情况的判断，例如人已经存在，或者边已经存在等。

**过程：**

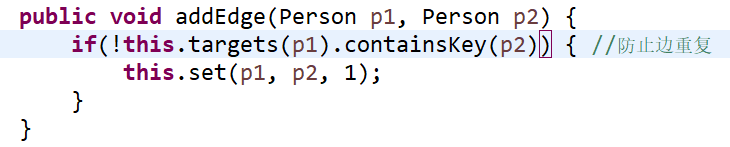
1. **public** **void** addVertex(Person p)

这个函数是为把参数添加到图中，作为图的一个顶点，直接调用父类的this.add()即可。



2. **public** **void** addEdge(Person p1, Person p2)

构建图的要素，添加了点后就该添加边了。先调用 this.targets()方法来判断边是否已经存在，然后调用this.set()方法设置边即可。

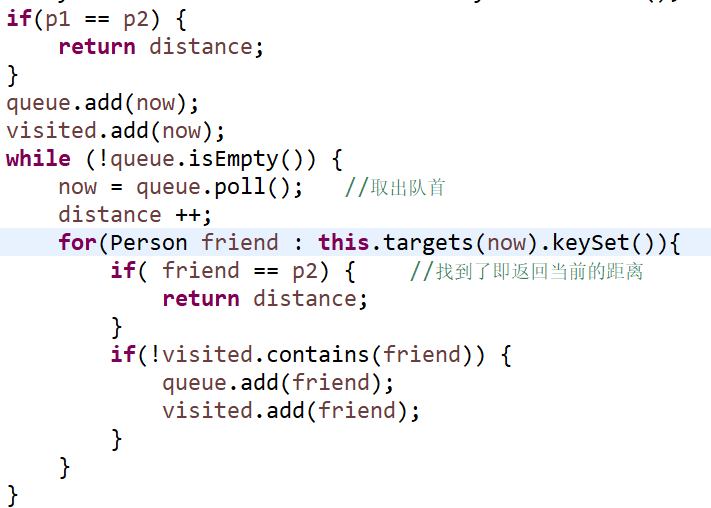


3. **public** **int** getDistance(Person p1, Person p2)

最主要的就是找两个人距离的函数，因为根据题设要求最短距离，因此采用广度遍历的方式，此处需要用到Queue的数据结构，并且设置了一个List来存放已经访问过的person。

广度遍历的主要思路是：将P1进队，设置访问过，然后出队，依次调用this.targets

()方法访问其相邻点，并且把它们进队，然后P1访问完再取队列第一个元素访问，直到某一相邻的点为P2则返回最短距离。如果所有点都访问完仍无最短距离，说明P1和P2没有关系，此时返回-1 。



**测试结果：**

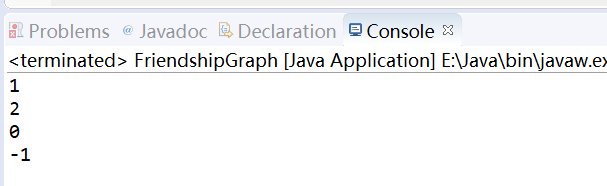


图1 正常输出的测试结果

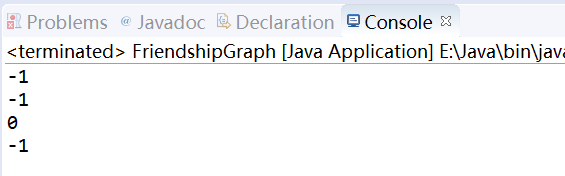


图2 注释掉rachel->ross后的输出结果

### Person类

给出你的设计和实现思路/过程/结果。

Person类较为简单，主要是根据FriendshipGraph类的需求编写的。它用于描述每个成员的性质，主要是实例化姓名的构造方法，getName()方法，判断姓名是否重复的nameSameWith方法。

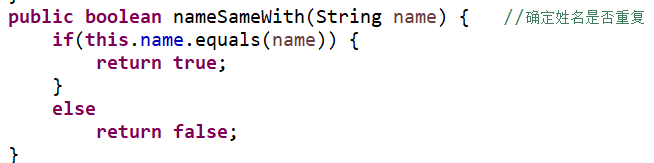


图3.3.2-1 nameSameWith方法

### 客户端main()

给出你的设计和实现思路/过程/结果。

main()函数主体部分就是实验指导上的要求：先new一个FriendshipGraph类的对象，然后添加顶点，添加边。

不同之处在于我增加了一个测试，如果Person的对象姓名字符串相同，违反了“Each person has a unique name”，将会提示出错并结束程序运行。

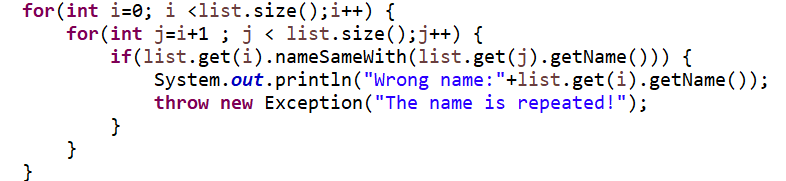


图3.3.3-1约束条件主体部分

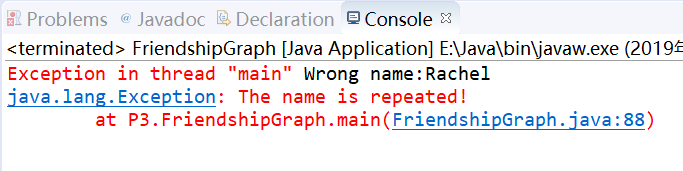


图3.3.3-2 违反“Each person has a unique name”

### 测试用例

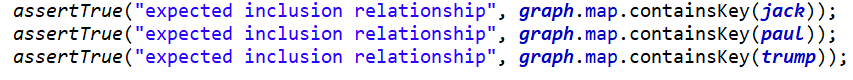
给出你的设计和实现思路/过程/结果。

**设计和实现思路：**在测试包Test中创建了一个 FriendshipGrapgTest.java文件后，通过注解的方式@Test表示这是一个测试方法。

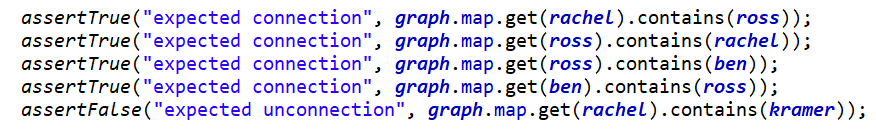
主要使用的方法：assertEquals、assertTrue和assertFalse方法。都是通过判断预期值和实际值是否相等来返回true或false。

**过程：**

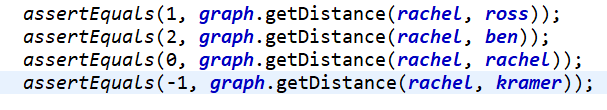
对于addVertex()方法，主要测试了添加的点是否存在。



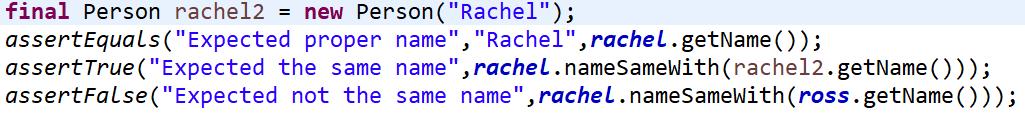
对于addEdge()方法，测试了边是否存在和是否有超出unconnected边。



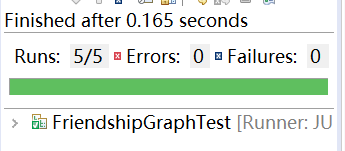
对于getDistance方法，测试了距离实际值是否和预期值相等。



测试person类，测试了getName()和nameSameWith()方法。



**结果：**



### 提交至Git仓库

git add Lab2\_1171000410

git commit -m “第二次提交”

git push origin master

在这里给出你的项目的目录结构树状示意图。

项目名称： Lab2\_1171000410

src

P2

FriendshipGraph.java

Person.java ...

test

P2

FriendshipGraphTest.java

## Playing Chess

### ADT设计/实现方案

设计了哪些ADT（接口、类），各自的rep和实现，各自的mutability/ immutability说明、AF、RI、safety from rep exposure。

必要时请使用UML class diagram（请自学）描述你设计的各ADT间的关系。

**一. Piece类（可变）**

**fields：**

**private** String pieceName; //棋子名称

**private** **int** pieceState; //0未放置，1已放置。-1被remove

**private** **int** pieceX; //横坐标

**private** **int** pieceY; //纵坐标

// Abstraction function:

// Piece代表棋盘上的棋子 ，pieceName代表棋子的名称，pieceState代表棋子的放置状态，

// pieceX代表棋子的横坐标，pieceY代表棋子的纵坐标

// Representation invariant:

// piece不能映射为空

//

// Safety from rep exposure:

// 所有fields都是 private

// 使用immutable数据类型

**/\*\*Constructor\*/**

**public** Piece(String name,**int** state, **int** x ,**int** y) 构造函数

**方法：**

1.所有fields的get函数和set函数。分别是返回Piece的fileds和通过传递参数设置fields。

2. **public** **void** remove()

将一个点从棋盘中移除。分别设置pieceState、pieceX、pieceY为 -1 。

**二. Position类（不可变）**

fields：

**private final int x; //横坐标**

**private final int y; //纵坐标**

// Abstraction function:

// Position代表棋子在棋盘上的位置，x代表位置的横坐标，y代表位置的纵坐标

// Representation invariant:

// position不能映射为空

//

// Safety from rep exposure:

// 所有fields都是 private and final

// 使用immutable数据类型

/\*\*Constructor\*/

**public** Position(**int** positionX, **int** positionY) 构造函数

方法：

1. 所有fields的get函数，分别返回该position 的横坐标和纵坐标。

2. @Override

**public** **boolean** equals(Object that)

判断两个position是否相等。如果position的x、y坐标值都相等，则两个位置相等，返回true，否则返回false。

**三. Player类（不可变）**

**fields：**

private String playerName; //棋手名

private Set<Piece> remaining = new HashSet<Piece>(); //玩家剩余棋子

private String history = new String(); //操作历史

// Abstraction function:

// Player映射操作pieces的玩家，playerName为玩家的名称，

// remaining代表玩家在棋盘上的棋子，history的字符串为玩家历史

// Representation invariant:

// Player不能映射为空

//

// Safety from rep exposure:

// 所有fields都是 private and final

// 使用immutable数据类型

// 使用Collections.unmodifiableSet不可变类型

/\*\*Constructor\*/

**public** Player(String p) 构造函数

**方法：**

1. 所有fields的get函数，返回Player的playerName，remaining，history

2.playerName的set函数，根据传入的name设置this.playername

3. **public** Piece getPiece(String pieceName)

这个是针对围棋go写的方法，根据传入的pieceName，从player的剩余棋子remaining中返回一个未放置的棋子。如果没有，返回null。

4. **public** **boolean** addPieces(Piece piece)

如果该棋手remaining有这个棋子，则返回false；否则在remaining中添加该piece并且返回true。

5. **public** **void** addHistory(String oneHistory)

传入的参数是每个操作后需要添加的玩家历史，判断其为非空。再调用String.concat函数将传入的历史连接在this.history后面即可。

6. **public** **boolean** judgeOwnPiece(Piece piece)

参数是棋子piece，判断remaining中是否由该棋子，有返回true，否则返回false。

7. **public** **int** countQuantityOfPieceInBoard()

计算player在棋盘上的棋子总数，初始化num = 0，遍历remaining，调用piece.getPieceState()方法，如果棋子状态为1，num++，最后返回num。

**四. Board类（可变）**

fields：

**private** **int** boardSize; //棋盘大小

**private** Piece[][] boardPosition; //棋子在棋盘的位置

// Abstraction function:

// Board 代表棋子所处的棋盘，boardSize代表了棋盘的边长，

// boardPosition代表pieces在棋盘上的位置

// Representation invariant:

// Board不能映射为空

//

// Safety from rep exposure:

// 所有fields都是 private and final

// 使用immutable数据类型

**方法：**

1.所有fields的get、set函数，分别返回和设置棋盘的边长、pieces在棋盘上的位置。

2. **public** **boolean** check(**int** x ,**int** y)

检查输入数据大小的合理性，如果参数x、y的大小超出了棋盘范围，返回false，否则返回true。

3. **public** Piece getBoardPiece(**int** x , **int** y)**throws** Exception

获得指定坐标的棋子。如果参数坐标超出棋盘范围，抛出异常。

4. **public** **void** setBoardPosition(Piece piece, **int** x , **int** y)**throws** Exception

将棋子放置在指定的位置，如果参数x、y超出棋盘范围，或者该棋盘的此位置有棋子，则抛出异常。

5. **public** **void** setBoardPositionState(**int** x , **int** y ,**int** newState)**throws** Exception

改变指定位置棋子的pieceState。如果参数x、y超出棋盘范围，抛出异常。

**五. Action类（不可变）**

field：

**private** final Board chessBoard = **new** Board();

// Abstraction function:

// Action映射为棋手的动作，chessBoard是一个Board的对象

// Representation invariant:

// Action和Board不能映射为空

//

// Safety from rep exposure:

// 所有fields都是 private and final

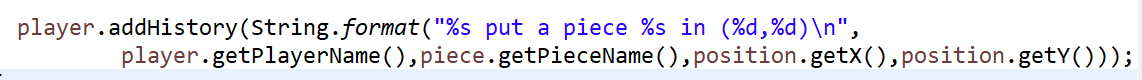
// 使用immutable数据类型

**方法：**

1. field的get和set函数。分别返回和设置this.chessBoard

2. **public** **void** putPiece(Player player , Piece piece , Position position)**throws** Exception

将棋子放置在指定的位置。先调用Board.getBoardPosition获得该position处的棋子，设置棋子的坐标，设置pieceState为1，将棋盘的position处棋子设置为piece。添加玩家历史。



如果piece并非属于player、position超出棋盘的范围、position处已有棋子、piece已经在棋盘上，抛出异常。

3. **public** **void** movePiece(Player player , Position oldPosition , Position newPosition)**throws** Exception

将棋子移动到指定的位置。先获取oldPosition处的棋子，判断棋子归属正确，存在性等，new一个Piece对象piece，复制原棋子参数，将原位置棋子的pieceState设置为0，将newPosition处棋子设置为piece。添加玩家remaining，添加玩家操作历史。

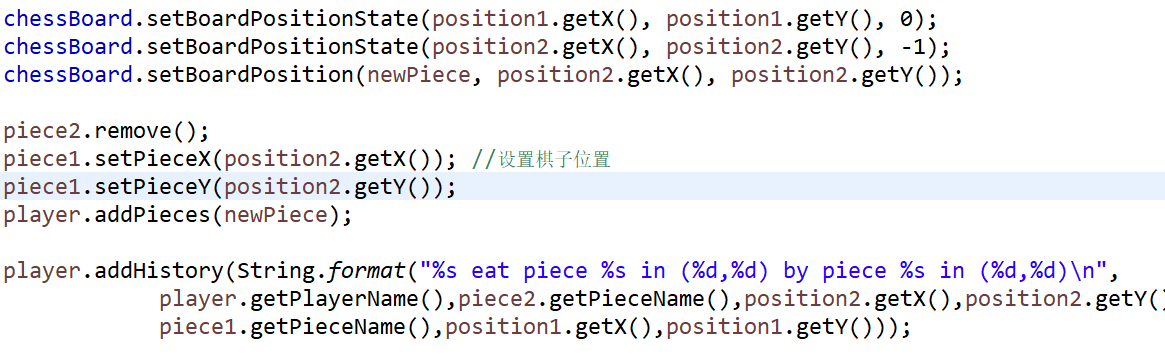
4. **public** **void** removePiece(Player player , Position position) **throws** Exception

先判断移除的是对方的且在棋盘上的棋子。将棋盘上对应的棋子pieceState设置为-1，调用Piece.remove，增加玩家历史。

如果position超出棋盘的范围、position处无棋子可提、所提piece不是对方棋子，则抛出异常。

5. **public** **void** eatPiece(Player player , Position position1 , Position position2) **throws** Exception

通过position1和position2获得要吃子piece1和被吃子piece2。判断棋子归属正确，存在性，将棋盘上position1处棋子pieceState设置为0，position2处pieceState设置为-1，并且在position2处新添加一个piece1的对象。移除piece2，添加newPiece到玩家的remaining中，添加玩家历史。



如果position1、position2超出棋盘的范围、position1上无棋子、position2上无棋子、两个位置相同、position1上的棋子不是player的棋子、第二个位置上的棋子不是对方棋子，则抛出异常。

**六. Game类（不可变）**

fields：

private final String gameName; //游戏名

private final Board gameBoard = new Board(); //棋盘

private final Action gameAction = new Action(); //动作

private final Player player1; //棋手1

private final Player player2; //棋手2

// Abstraction function:

// Game代表了一局游戏，gameBoard映射为game中的棋盘，棋盘的操作这是player1和player2.

// gameAction为棋手的操作

// Representation invariant:

// Game不能映射为空，gameBoard不能映射为空，Action不能映射为空

//

// Safety from rep exposure:

// 所有fields都是 private and final

// 使用immutable数据类型

**方法：**

1.所有fields的get函数

2.player1、player2、gameName的set函数

3. **public** **void** putPiece(Player player, Piece piece, Position position) **throws** Exception

**public** **void** movePiece(Player player, Position oldPosition, Position newPosition)

**public** **void** removePiece(Player player, Position position) **throws** Exception

**public** **void** eatPiece(Player player, Position position1, Position position2) **throws** Exception

以上调用Action类中的方法即可。

4. **public** Piece getOccupationOfPosition(Position position) **throws** Exception

在指定的position处获得一个棋子。调用Board.getBoardPiece()即可。

5. **public** **void** initByChess(String name1, String name2)

设置国际象棋chess棋盘。设置棋盘的gameName，通过参数设置players名称，初始化棋盘，设置32个pieces的pieceName、pieceState、坐标为(-1,-1)。将棋子添加到棋盘上，增加玩家棋子。

6. **public** **void** initByGo(String name1, String name2)

设置围棋go棋盘。设置棋盘的gameName，通过参数设置players名称，初始化棋盘，增加玩家棋子，设置 pieces的pieceName、pieceState、坐标为(-1,-1)。最后将棋子添加到棋盘上。

7. **public** **void** printBoard()

打印棋盘。遍历gameBoard即可，需要注意的是二元数组的行和可视坐标系中的行恰好相反，因此每行的棋子倒序打印。

### 主程序MyChessAndGoGame设计/实现方案

辅之以执行过程的截图，介绍主程序的设计和实现方案，特别是如何将用户在命令行输入的指令映射到各ADT的具体方法的执行。

fields:

**public** Game game = **new** Game(); //新建游戏

**public** String player1Name; //玩家1名称

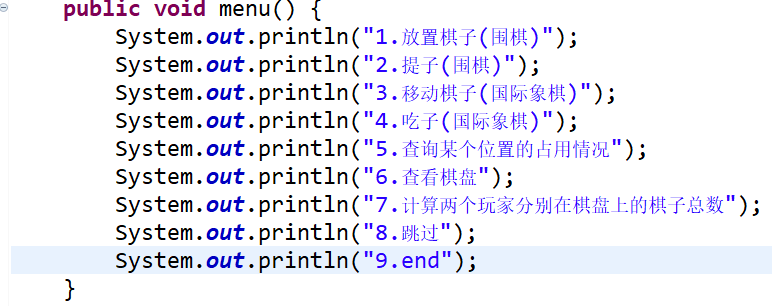
**public** String player2Name; //玩家2名称

**public** ArrayList<Player> players = **new** ArrayList<Player>(); //玩家链表

备注：设计玩家链表是为了后面while循环能通过索引值的变化，实现两个玩家的操作的切换。turn = (turn +1) % 2

为了调用非静态方法，在main函数中new了MyChessAndGoGame对象，并且调用自己创建的myMain()方法。其中主体部分在myMain之中。

打印菜单（展示了实现的功能）：



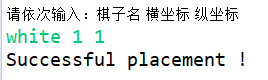
**public** **void** myMain() **throws** Exception：

1. BufferedReader提供通用的缓冲文本读取，readLine()读取一个文本行。InputStreamReader将字节流转化字符流。

使用以上方式读取用户的输入。

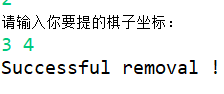
2.让用户输入棋类名称chess or go，来确定调用Game类中哪种初始化方法，再读取用户输入的两个棋手的名称，游戏即可开始。

3.放置棋子（围棋）



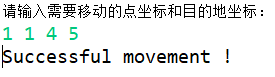
读取line，调用String.split()将字符串分开。white映射到棋子名称，数字1和1映射到棋盘上的位置，player则通过对players索引获得。最后调用Game.putPiece()即可。其中异常情况的处理在Game类中已介绍。

4.提子（围棋）



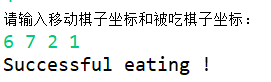
读取line，调用String.split()将字符串分开。数字3和4映射到棋盘上的位置，player通过对players索引获得。最后调用Game.removePiece()即可。其中异常情况的处理在Game类中已介绍。

5.移动棋子（国际象棋）



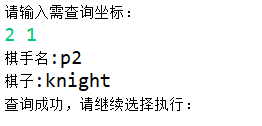
读取line，调用String.split()将字符串分开。数字1和1映射需要移动的棋子点坐标，4和5映射目的地坐标，player通过对players索引获得。最后调用Game.movePiece()即可。其中异常情况的处理在Game类中已介绍。

6.吃子（国际象棋）



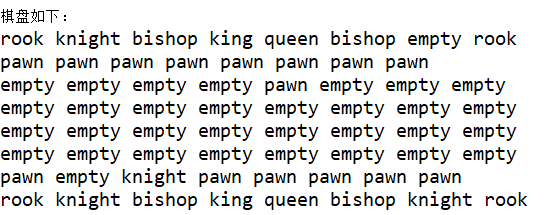
读取line，调用String.split()将字符串分开。数字6和7映射需要主动吃子的棋子点坐标，2和1映射被吃棋子坐标，player通过对players索引获得。最后调用Game.eatPiece()即可。其中异常情况的处理在Game类中已介绍。

7.查询某个位置的占用情况



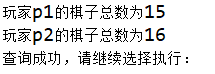
读取line，调用String.split()将字符串分开。数字2和1映射需要待查位置坐标，player通过对players索引获得，该棋子通过Game.getOccupationOfPosition()方法得到，棋手名players.get(i).getPlayerName()，棋子名piece.getPieceName()。其中异常情况的处理在Game类中已介绍。

8. 查看棋盘



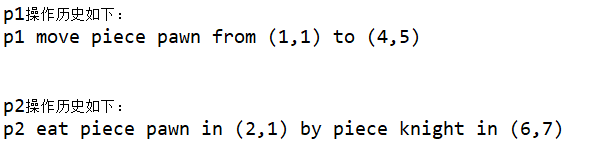
调用Game.printBoard()方法打印棋盘。

9.计算两个玩家分别在棋盘上的棋子总数



调用Player.getPlayerName()和Player.countQuantityOfPieceInBoard()即可。

10.结束查看玩家历史



调用Player.getPlayerName()和Player.getHistory()来打印对应玩家的操作历史。其中每次历史中包括玩家名、棋子名和棋子移动的坐标。

### ADT和主程序的测试方案

介绍针对各ADT的各方法的测试方案和testing strategy。

介绍你如何对该应用进行测试用例的设计，以及具体的测试过程。

测试方案：对于较为简单的类，Piece类、Player类、Position类，可以在testGame中定义对象的时候，辅助进行测试。

由于Action类的方法在Game类中会被调用，因此也不必单独测试。

主要是测试Game类和Board类。

Game类：对于chess、go两种棋类需要分别初始化棋盘，分别进行测试。使用assertEquals()语句判断game类获取的gameName、gamePlayer等是否与预期值一致，使用assertTrue()确定某个判断的真假，assertFalse()则是针对错误情况进行覆盖。

主要的几个方法如：putPiece()，eatPiece()等，测试方法基本一致。逐句测试，判断棋子位置、玩家历史、棋子归属等是否与Expection一致。对于非法情况，需要单独处理和判断，由于错误情况会抛出异常，因此需要根据各个异常情况构造方法单独测试。getOccupationOfPosition()方法则是判断返回值和预期Piece是否相等即可。

事实上，当测试完Game类，Board的测试也基本覆盖到了。特殊的是，Board类中有许多对于超出棋盘范围的check，这些需要举反例单独测试。

// Testing strategy

// 因为类之间的存在的调用关系，

// 先主要测试Game类

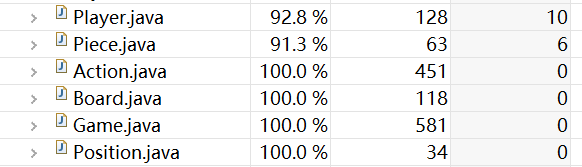
// 然后对其他类做补充

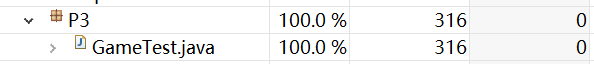
// 合法情况，非法情况，抛出异常

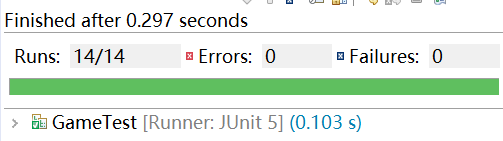
// 使用断言进行检查

// chess和go分开测试

测试覆盖率结果：







## Multi-Startup Set (MIT)

请自行设计目录结构。

注意：该任务为选做，不评判，不计分。

# 实验进度记录

请使用表格方式记录你的进度情况，以超过半小时的连续编程时间为一行。

每次结束编程时，请向该表格中增加一行。不要事后胡乱填写。

不要嫌烦，该表格可帮助你汇总你在每个任务上付出的时间和精力，发现自己不擅长的任务，后续有意识的弥补。

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 计划任务 | 实际完成情况 |
| 2019.3.19 | 14:00-18:00 | 开始p1 | 未完成，p1理解较为困难 |
| 2019.3.20 | 18:30-23:00 | 通读p1内容，完成problem1的测试程序编写 | 按计划完成 |
| 2019.3.21 | 15:30-23:00 | 完成ConcreteEdgeGraph() | 比预计晚2小时 |
| 2019.3.22 | 13:00-18:00 | 完成ConcreteVertexGraph() | 比预计晚1小时 |
| 2019.3.23 | 19:00-23:00 | 完善GraphInstanceTest.java，并且写ConcreteEdgeGraphTest.java | 按计划完成 |
| 2019.3.24 | 14:00-17:30 | 写ConcreteVertexGraphTest.java | 按计划完成 |
| 2019.3.25 | 18:30-22:30 | 完成p2并写完其测试语句 | 按计划完成 |
| 2019.3.26 | 14:00-17:00 | 设计ADT | 提前半小时完成 |
| 2019.3.27 | 17:30-23:00 | 写Piece、Player、Position类 | 按计划完成 |
| 2019.3.28 | 13:30-15:30 | 写Game、Action类 | 未完成 |
| 2019.3.29 | 18:30-23:00 | 完成前一日内容，并构思MyChessAndGoGame | 按计划完成 |
| 2019.3.30 | 10:00-17:30 | 写完MyChessAndGoGame | 比预计晚2小时 |
| 2019.3.31 | 18:30-23:00 | 写完测试文件 | 比预计晚1小时 |
| 2019.4.1 | 18:30-23:00 | 写报告 | 未完成 |
| 2019.4.2 | 18:30-23:00 | 完成报告 | 按计划完成 |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
| 拿到P1，感到无从下手，难以理解题意 | 反复阅读实验指导书和文件注释中的提示，冷静下来考虑解题思路 |
| 对于AF、RI的理解不足 | 在老师课堂讲解的基础上，去MIT网站上找到相关的文献资料进行阅读。 |
| 测试覆盖率不高 | 将类中的方法分开进行测试，考虑各种合法情况和非法情况，提高test文件的质量 |
| 对泛型理解不足 | 翻阅Java编程思想，对于泛型部分的介绍仔细阅读，理解其中的实例 |
| P3棋类ADT设计思路不明确 | 停下了编写代码的工作，仔细设计ADT |
| 在嵌套的调用中，非法情况的处理出现问题 | 研究异常的机制 |
| git提交的时候显示git@github.com: Permission denied (publickey).  fatal: Could not read from remote repository. | 在网上查找教程，重新设置ssh key |
| MyChessAndGoGame运行时bug较多 | 设置断点，单步执行debug |

# 实验过程中收获的经验、教训、感想

## 实验过程中收获的经验和教训

经验：加深了自己对于泛型的理解和认识，提高了代码编写、ADT设计的能力，在找不出代码bug时，要学会设置断点进行debug，编写test测试文件时，有些方法的测试也能覆盖到其他的方法，避免重复测试增加工作量。

教训：在编写代码之前，最重要的是理解题意，这样可以避免走很多弯路

## 针对以下方面的感受

1. 面向ADT的编程和直接面向应用场景编程，你体会到二者有何差异？ 答：我感觉，对于面向对象的编程，类是它的主要特点，程序执行过程中，先由主函数进入，定义一些类，根据需要，执行[类的成员函数](https://www.baidu.com/s?wd=%E7%B1%BB%E7%9A%84%E6%88%90%E5%91%98%E5%87%BD%E6%95%B0&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)，过程的概念被淡化了
2. 使用泛型和不使用泛型的编程，对你来说有何差异？

答：我认为泛型编程可以使代码被很多不同类型的对象所重用，并使代码具有更好的可读性。对于我编写时差异并不大。

1. 在给出ADT的规约后就开始编写测试用例，优势是什么？你是否能够适应这种测试方式？

答：优势是不考虑代码的内部实现，只需考虑是否完成了规约中指定的功能，我认为这种策略很不错。

1. P1设计的ADT在多个应用场景下使用，这种复用带来什么好处？

答:提高了代码的利用率，减轻编程工作量。

1. P3要求你从0开始设计ADT并使用它们完成一个具体应用，你是否已适应从具体应用场景到ADT的“抽象映射”？相比起P1给出了ADT非常明确的rep和方法、ADT之间的逻辑关系，P3要求你自主设计这些内容，你的感受如何？

答：初接触P1，还不太适应，完成P3之后，对于从具体应用场景到ADT的“抽象映射”适应了很多。P3的难度比较大，很具有挑战性。

1. 为ADT撰写specification, invariants, RI, AF，时刻注意ADT是否有rep exposure，这些工作的意义是什么？你是否愿意在以后编程中坚持这么做？

答：使编写的代码更加安全和可读性更强。愿意这么做。

1. 关于本实验的工作量、难度、deadline。

工作量较大，难度适中，deadline设置合理。

1. 《软件构造》课程进展到目前，你对该课程有何体会和建议？

体会：感觉在编写代码之前的设计非常关键，对一个problem的理解程度往往会决定工作效率。

建议：上课时简单的部分可以讲快一点，这样节省时间思考较难的部分。