

**2021年春季学期  
计算学部《软件构造》课程**

**Lab 2实验报告**

|  |  |
| --- | --- |
| 姓名 | 熊皓 |
| 学号 | 1190501809 |
| 班号 | 1903004 |
| 电子邮件 | 1439188741@qq.com |
| 手机号码 | 18323246430 |

**目录**

[1 实验目标概述 1](#_Toc72249750)

[2 实验环境配置 1](#_Toc72249751)

[3 实验过程 1](#_Toc72249752)

[3.1 Poetic Walks 1](#_Toc72249753)

[3.1.1 Get the code and prepare Git repository 1](#_Toc72249754)

[3.1.2 Problem 1: Test Graph <String> 1](#_Toc72249755)

[3.1.3 Problem 2: Implement Graph <String> 1](#_Toc72249756)

[3.1.3.1 Implement ConcreteEdgesGraph 2](#_Toc72249757)

[3.1.3.2 Implement ConcreteVerticesGraph 2](#_Toc72249758)

[3.1.4 Problem 3: Implement generic Graph<L> 2](#_Toc72249759)

[3.1.4.1 Make the implementations generic 2](#_Toc72249760)

[3.1.4.2 Implement Graph.empty() 2](#_Toc72249761)

[3.1.5 Problem 4: Poetic walks 2](#_Toc72249762)

[3.1.5.1 Test GraphPoet 2](#_Toc72249763)

[3.1.5.2 Implement GraphPoet 2](#_Toc72249764)

[3.1.5.3 Graph poetry slam 2](#_Toc72249765)

[3.1.6 使用Eclemma检查测试的代码覆盖度 2](#_Toc72249766)

[3.1.7 Before you’re done 2](#_Toc72249767)

[3.2 Re-implement the Social Network in Lab1 2](#_Toc72249768)

[3.2.1 FriendshipGraph类 2](#_Toc72249769)

[3.2.2 Person类 3](#_Toc72249770)

[3.2.3 客户端main() 3](#_Toc72249771)

[3.2.4 测试用例 3](#_Toc72249772)

[3.2.5 提交至Git仓库 3](#_Toc72249773)

[4 实验进度记录 3](#_Toc72249774)

[5 实验过程中遇到的困难与解决途径 3](#_Toc72249775)

[6 实验过程中收获的经验、教训、感想 4](#_Toc72249776)

[6.1 实验过程中收获的经验和教训 4](#_Toc72249777)

[6.2 针对以下方面的感受 4](#_Toc72249778)

# 实验目标概述

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；

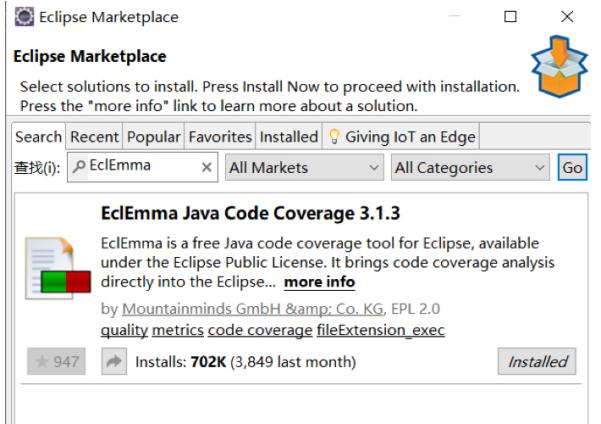
在测试代码中，能够写出 testing strategy 并据此设计测试用例。

# 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。

本次需要安装EclEmma，下面是安装过程：



无异常，成功安装。

在这里给出你的GitHub Lab2仓库的URL地址（Lab2-学号）。

https://github.com/ComputerScienceHIT/HIT-Lab2-1190501809

# 实验过程

## Poetic Walks

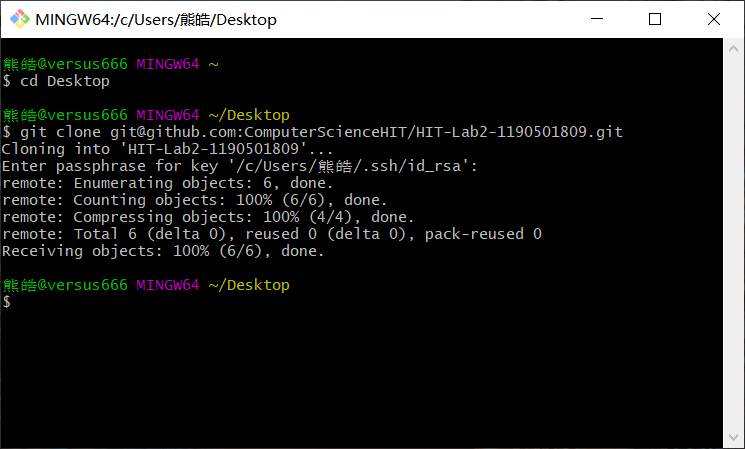
该任务的主要是实验一个图的模块，同时实现以下要求：

1. 完善Graph接口类，并运用泛型的思想，将String拓展为泛型L类；
2. 实行Graph类的方法：add，set，remove，vertices，sources，targets;
3. 利用实现的Graph类及图的思想，实现GraphPoet类，如果输入的文本的两个单词之间存在桥接词，则插入该桥接词；若存在多个单一桥接词，则选取边权重较大者。

### Get the code and prepare Git repository

首先找到本地的一个文件夹，然后输入命令：

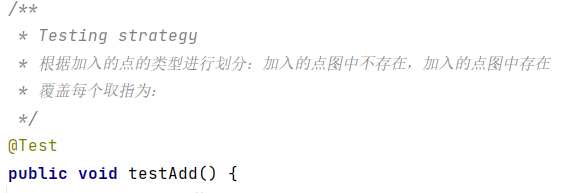
git clone [git@github.com:ComputerScienceHIT/HIT-Lab2-1190501809.git](mailto:git@github.com:ComputerScienceHIT/HIT-Lab2-1190501809.git)

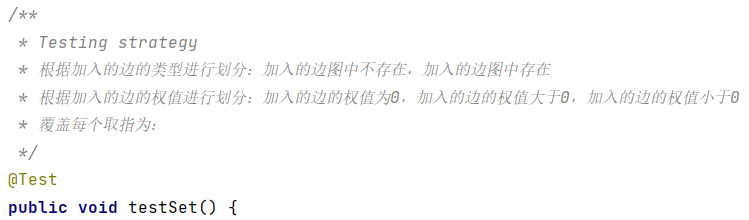


### Problem 1: Test Graph <String>

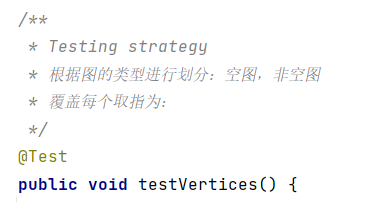
思路：针对Graph类中的每个方法设计测试策略，进行等价类划分的测试。

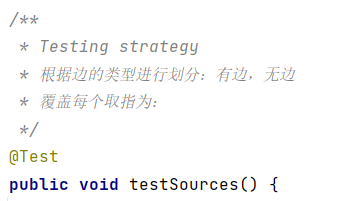
测试策略：

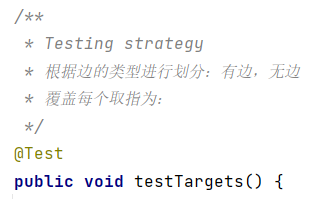












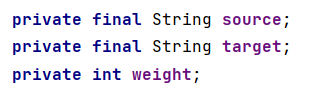
测试结果：



### Problem 2: Implement Graph <String>

#### Implement ConcreteEdgesGraph

1. 实现Edge类
2. 定义私有类型的变量：



1. 实现的方法

**Edge:** 初始化构造方法，初始化新边的两个点和边权值；

**CheckRep:** 检查rep invariant是否为真；

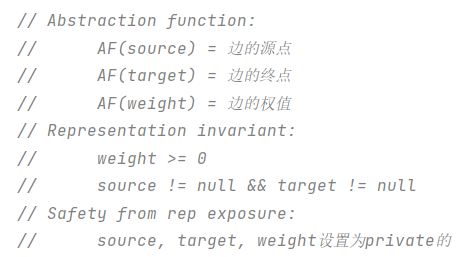
**getSource:** 返回边的source点；

**getTarget**: 返回边的target点；

**getWeight:** 返回边的权值；

**toString:** 返回一条边的字符串，形式为“source -> target : weight”

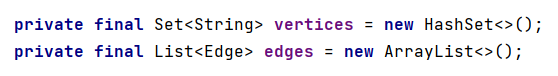
1. AF，RI和Safety from rep exposure



1. 测试策略



1. 实现ConcreteEdgeGraph类
2. 定义的私有类型



1. 实现的方法

**ConcreteEdgesGraph:** 构造方法；

**checkRep:** 检查rep Invariant是否为true；

**Add:** 顶点不为空的时候，添加一个顶点进入顶点表中；

**Set:** 输入source，target, weight; 若weight为负，则返回-1 ;若weight为0且新边已经存在，则删去原边 ;若weight为正整数，且新边不存在，则直接添加新边 ;只要改变了原边的权值，都返回原边的权值，没有权值则返回0;

**Remove:** 除去某个点及与它相连的边；

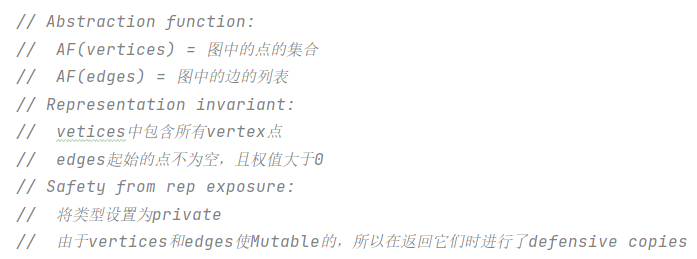
**Vertices:** 返回所有点集；

**Sources:** 给定终点，返回所有与它相连的边的起点和权值；

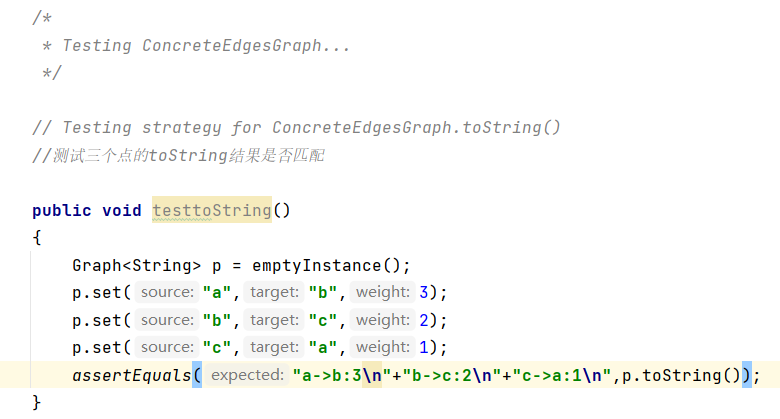
**Target:** 给定起点，返回所有与它相连的边的终点与权值；

**toString:** 将整个图以字符串的形式输出.

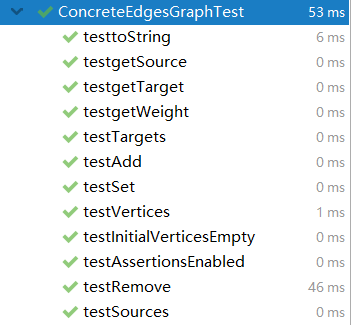
1. AF,RI和Safety from rep exposure



1. 测试策略



1. 测试结果





#### Implement ConcreteVerticesGraph

1. 实现Vertices类
2. 定义的私有类型



1. 实现的方法

**Vertex:** 初始化构造

**checkRep:** 检查表示不变性

**getMark:** 获得该顶点的名字

**getSource:** 获得该顶点所有源点及相连边的权值的map

**getTarget:** 获得该顶点所有终点及相连边的权值的map

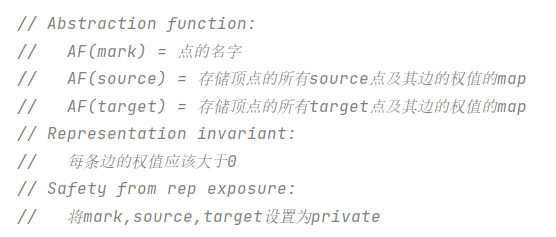
**addSource:** 若weight不为0，则将其加入source中(若源点已存在，则更新其weight，并返回原weight，不存在则直接构建新点并返回0)；若weight为0，则移除源点(不存在返回0，存在返回原weight)

**addTarget:**  若weight不为0，则将其加入target中(若终点已存在，则更新其weight，并返回原weight，不存在则直接构建新点并返回0)；若weight为0，则移除终点(不存在返回0，存在返回原weight)

**removeSource:** 在源点表中删除某源点，并返回旧边的权值（源点不存在则返回0）

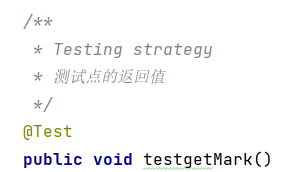
**removeTarget:** 在终点表中删除某终点，并返回旧边的权值（终点不存在则返回0）

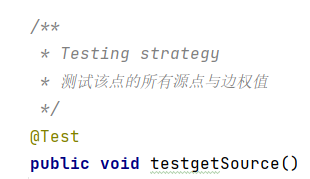
（3）AF,RI和Safety from rep exposure

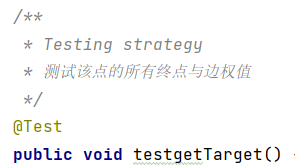


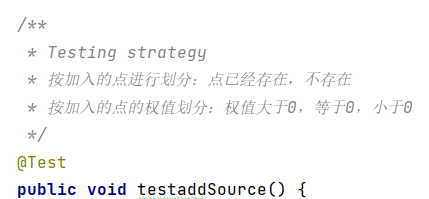
（4）测试策略

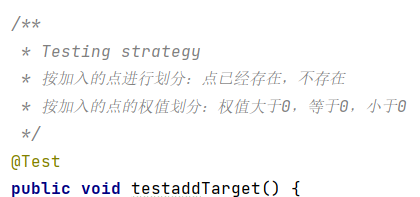










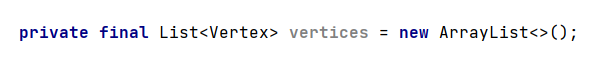






2.实现ConcreteVerticesGraph

1. 定义的私有类型



1. 实现的方法

**ConcreteVerticesGraph:** 构造方法

**checkRep:** 检查rep invariant是否为true

**Add:** 顶点不为空时，向点表中添加一个顶点

**Set:** 输入source，target, weight; 若weight为负，则返回-1 ;若weight为0且新边已经存在，则删去原边 ;若weight为正整数，且新边不存在，则直接添加新边 ;只要改变了原边的权值，都返回原边的权值，没有权值则返回0;

**Remove:** 除去某个点及与它相连的边；

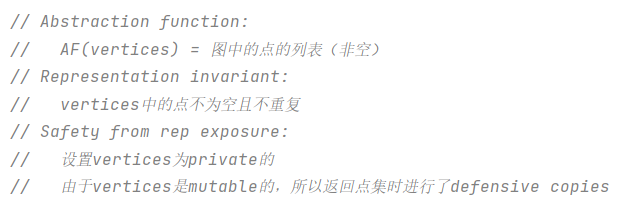
**Vertices:** 返回所有点集；

**Sources:** 给定终点，返回所有与它相连的边的起点和权值；

**Target:** 给定起点，返回所有与它相连的边的终点与权值；

**toString:** 将整个图以字符串的形式输出.

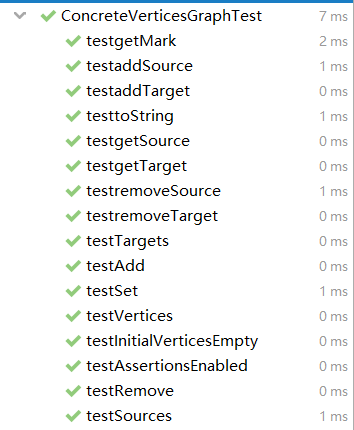
1. AF,RI和Safety from rep exposure



1. 测试策略



1. 测试结果

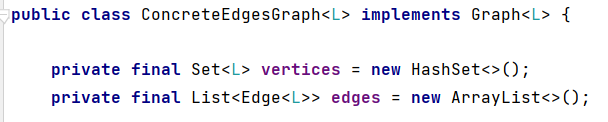


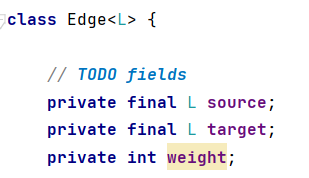


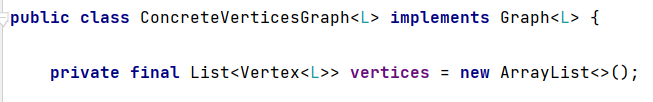
### Problem 3: Implement generic Graph<L>

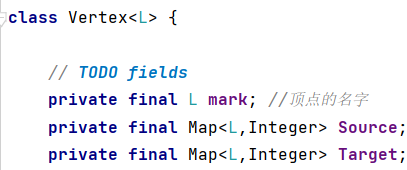
#### Make the implementations generic

将ConcreteEdgesGraph、Edge和ConcreteVerticesGraph、Vertex中的String类型更改为L泛型



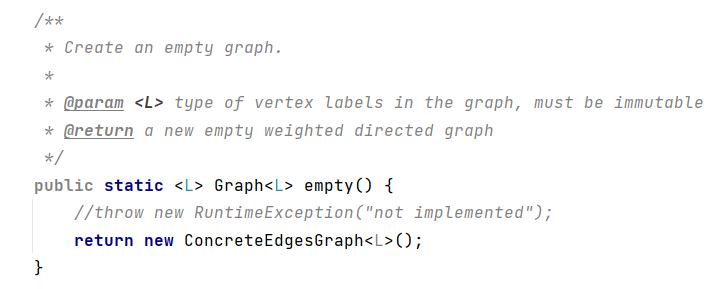




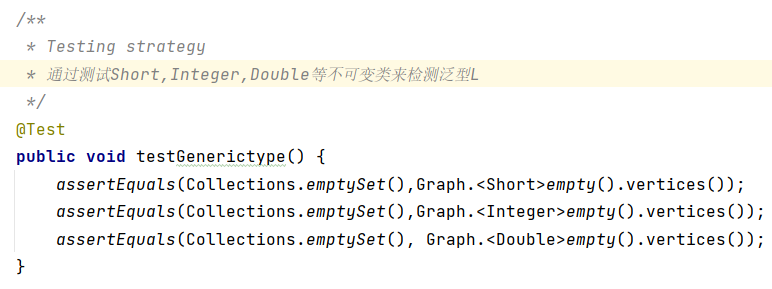


#### Implement Graph.empty()

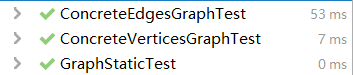
在Graph.empty()中任意实现一个Graph的类

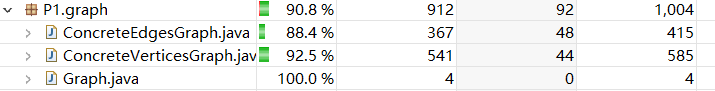


然后在GraphStaticTest.java中进行一些其他的不可变类型的测试



检测整体测试及测试覆盖率



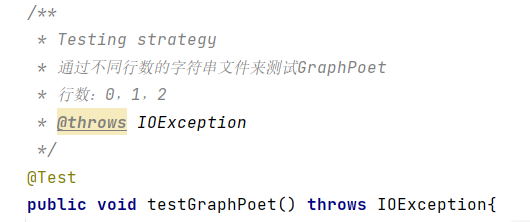


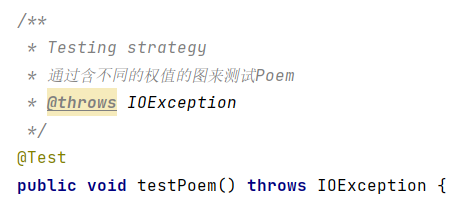
### Problem 4: Poetic walks

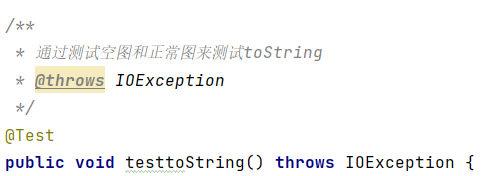
该任务主要要求我们实现一个类，利用之前实现的图结构，将文本文件中的字符内容转换为一个图形结构，然后基于这个图形结构将输入的诗句进行语句扩充。

#### Test GraphPoet

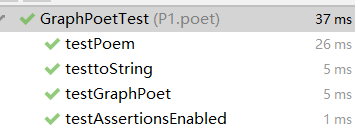
测试策略：







测试结果：





#### Implement GraphPoet

1. 定义的私有类型：



1. 实现的方法：

**GraphPoet:** 读入文件内容，利用graph中的set(source, target, weight)方法构建一个图形结构

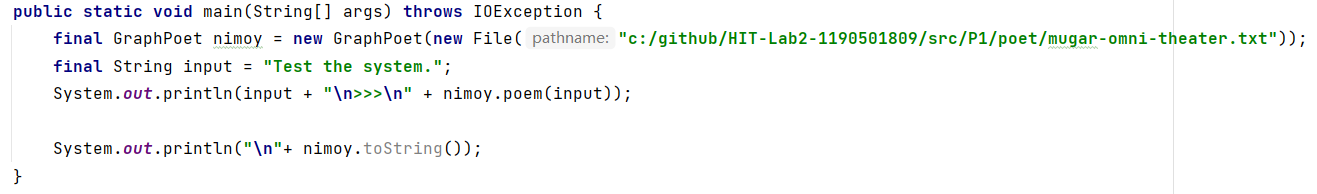
**checkRep:** 通过检测图是否为空来检测图形结构是否构建成功

**Poem:** 首先将输入语句进行分割，然后加入到字符数组s中。然后循环遍历这个字符数组s，每次寻找图中的与s[i]匹配的顶点，然后再遍历这个顶点的target点集，检查是否有target点与s[i+1]匹配，若有则继续遍历s中的下一点；若没有，则再遍历一次这个target点的target点集，若找到，则将前一个target插入语句中，否则就直接break跳出循环，开始遍历下一个s点。最后返回处理完成的语句

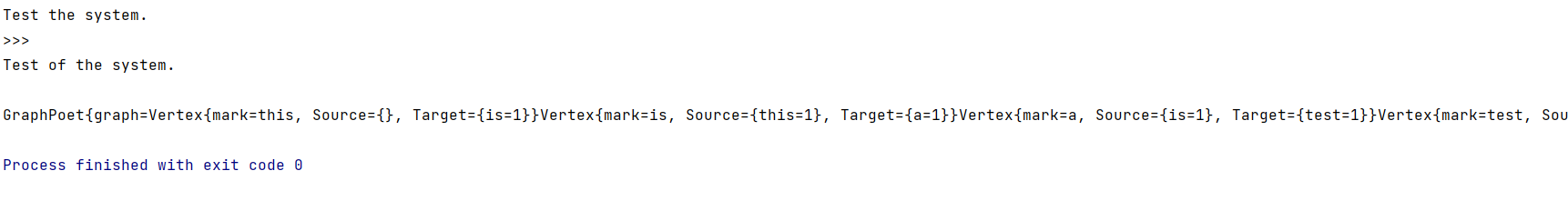
**toString:** 使用ConcreteVerticesGraph中的toString方法，将整个图转换为一条字符串输出

#### Graph poetry slam

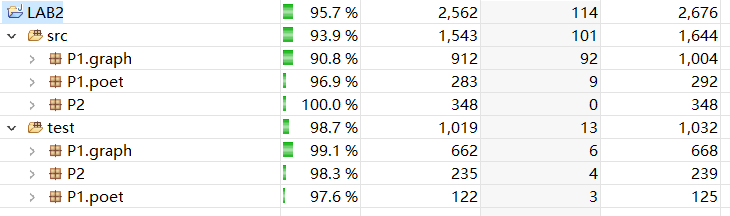
在提供代码的基础上加一个toString的输出



输出为：



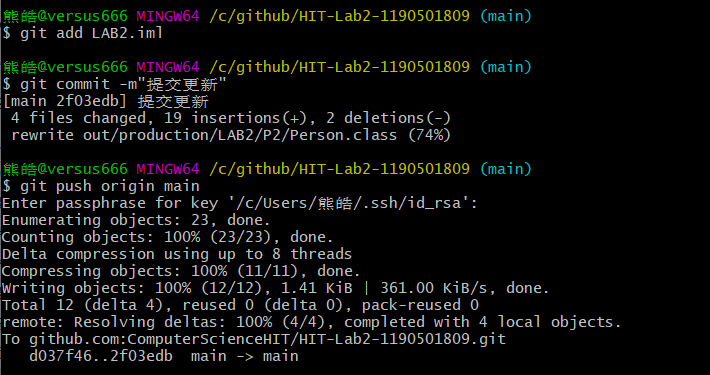
### 使用Eclemma检查测试的代码覆盖度



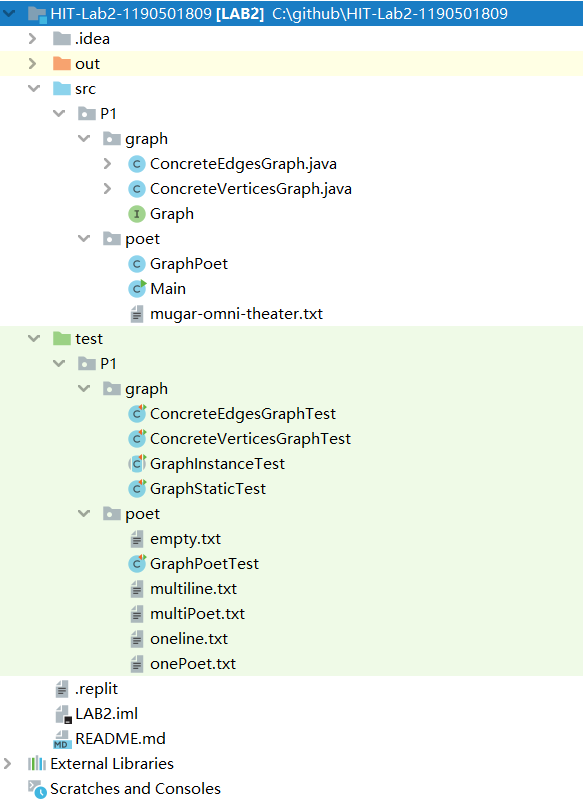
### Before you’re done

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。

如何通过Git提交当前版本到GitHub上你的Lab2仓库。



在这里给出你的项目的目录结构树状示意图。



## Re-implement the Social Network in Lab1

基于P1中的定义的Graph<L>及其两种实现，重新实现FriendshipGraph类

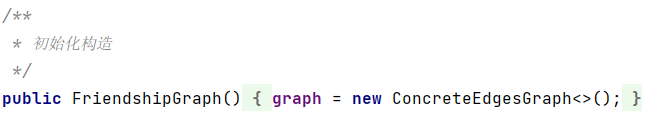
### FriendshipGraph类

1. 思路

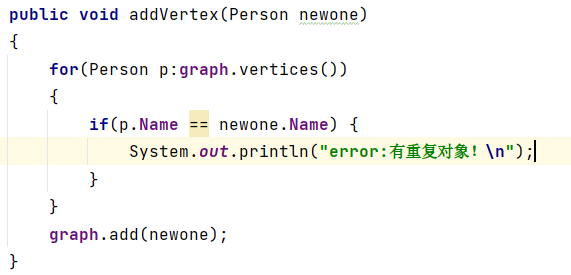
基于已经实现的Graph<L>及其两种实现，L替换为Person，同时FriendshipGraph应该提供三种方法：addVretex(), addEdge(), getDistance()。针对addVertex()和addEdge()，选择复用ConcreteEdgesGraph中的add()和set()方法；针对getDistance()方法，应基于ConcreteEdgesGraph中的rep来实现（不修改rep）。

1. 过程

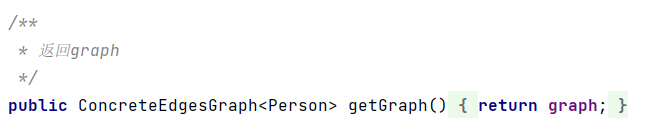
**FreindshipGraph():**



**addVertex():** 针对addVertex()方法，需要增加一个重复对象的判断



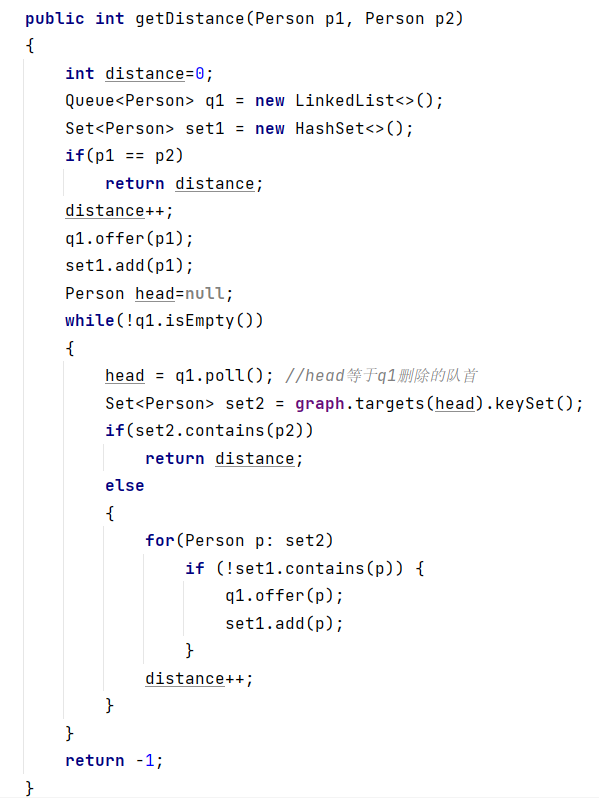
**getGraph():** 新增一个返回graph的方法getGraph()



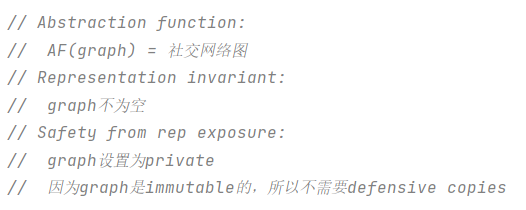
**addEdge():**



**getDistance():** 基于ConcreteEdgesGraph的rep，使用广度优先求最短路径



1. AF，RI和Safety from rep exposure



### Person类

1. 思路

复用Lab1中的Person类

1. 实现过程

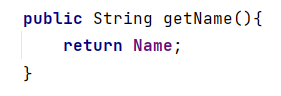
定义私有类型



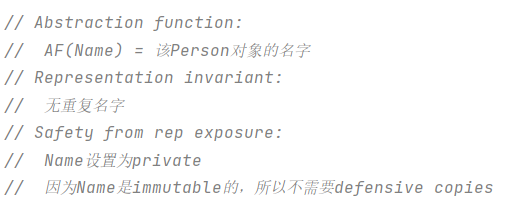
**Person():** 将输入的name赋值给Person的Name



**getName():** 返回Person的名字



1. AF, RI和Safety from exposure

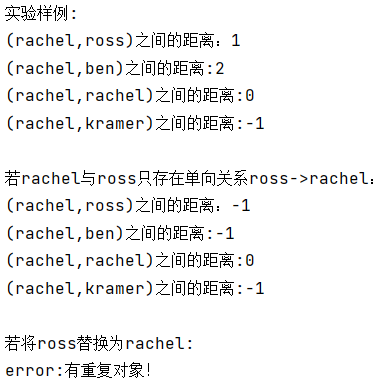


### 客户端main()

复用Lab1的main:



运行后的结果：

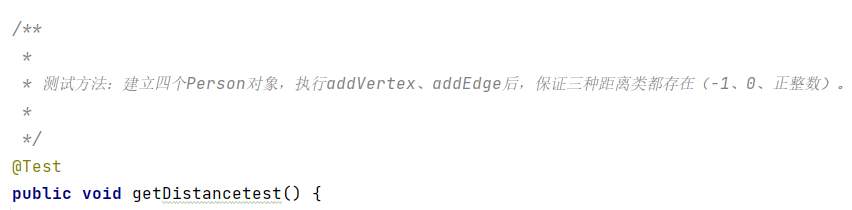


### 测试用例

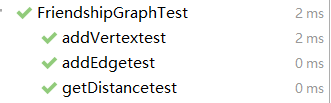
1. 测试策略







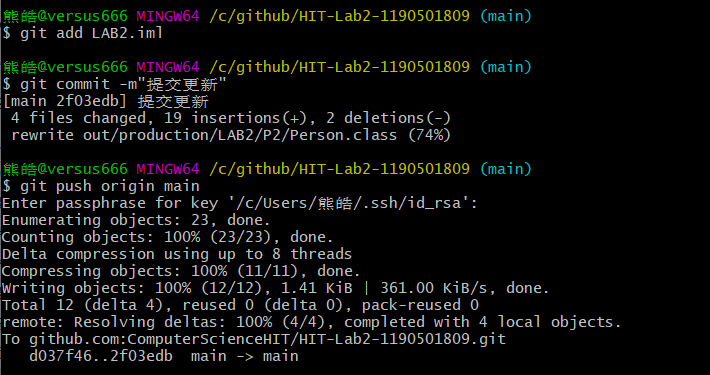
1. 测试结果



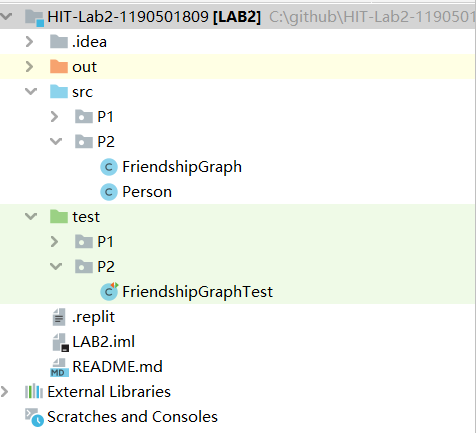


### 提交至Git仓库

如何通过Git提交当前版本到GitHub上你的Lab3仓库。



在这里给出你的项目的目录结构树状示意图。



# 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 |
| 2021.6.28 | 9:30-12:00 | 完成P1的Graph的测试内容 | 按时完成 |
| 2021.6.29 | 14:00-21:25 | 完成P1的ConcreteEdgesGraph的实现与测试 | 代码已完成，但有一些测试仍有一些问题未解决 |
| 2021.6.30 | 10:00-16:45 | 解决上次遗留问题并完成P1的ConcreteVerticesGraph的实现与测试 | P1问题全部解决，ConcreteVerticesGraph的测试未全部完成 |
| 2021.6.3 | 9:00-15:00 | 完成P1的ConcreteVerticesGraph的测试及之后的Poet的实现与测试 | 其他按时完成，Poet的实现有部分有问题，还需修改 |
| 2021.6.4 | 10:00-20:00 | 完成P1的余下部分及P2的全部内容 | 按时完成 |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
| 刚开始写的时候不知道AF,RI及Safety from exposure的含义 | 通过课上老师的讲解及自己查看课件的详细解释 |
| P1中要求将String类型全部改为泛型L，改完后遇到很多  bug | 通过查询CSDN网站同时询问学长，依次处理了bug |
| 完成GraphPoet时, 读取两行的文本文件来构建图模型时出现顶点连接问题 | 首先通过Debug，找到错误来源。然后结合自己的readLine方法，设计了一个机制每次循环处理读入的文本，解决问题 |
| 在测试覆盖度时，遇到受自己test类设计考虑不全而导致覆盖度较低的问题 | 通过分析总结每一种情况，然后再test类中依次实现，最终提高了自己的代码覆盖度 |

# 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

对JAVA的掌握还是不够熟练，在面对自己设计类来实现多种功能时，代码出现了很多不规范的情况，同时在自行设计test类时容易考虑情况不周全，导致测试覆盖度较低。但经过本次实验，我发现自己的java水平提升很多，同时自己在面对很多问题时能够独立思考并解决，即使遭遇了bug，也能一步一步debug的找出问题并解决。

## 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？

面向ADT的编程需要考虑AF,RI及Safety from exposure的问题，且编程主要是基于对象来具体实现的，这与面向过程的编程思路很不相同。

1. 使用泛型和不使用泛型的编程，对你来说有何差异？

使用泛型的编程需要考虑更多的情况，要注意通用性，而不同于不使用泛型编程时只用考虑当前单个对象的特殊性。

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？

能够保证代码的正确性。不是特别适应，因为一开始感觉比较没有头绪。

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？

可提高代码的利用率，减少重复。

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？

本次实验未涉及P3

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？

防止内部变量被外部修改，同时面对包泄露等情况时能够排除程序的本身的一些问题。这项工作很有意义，非常值得坚持完成。

1. 关于本实验的工作量、难度、deadline。

本次实验工作量主要集中在P1，但一步一步有计划的完成，工作量就不显得很大。难度的感觉本次实验正常水平，并没有特别难。截止时间老师给的很宽裕，非常舒适。

1. 《软件构造》课程进展到目前，你对该课程有何体会和建议？

通过这个课程，我最直观的体验是教会了我更规范的使用java进行编程，同时培养了我面向对象的思想，对软件编程程序的规范性有了新的认识与收获。