

**2021年春季学期  
计算学部《软件构造》课程**

**Lab 2实验报告**

|  |  |
| --- | --- |
| 姓名 | 周凡 |
| 学号 | 1190201008 |
| 班号 | 1903011 |
| 电子邮件 | [434696317@qq.com](mailto:434696317@qq.com) |
| 手机号码 | 18217207114 |

**目录**

[1 实验目标概述 1](#_Toc72247573)

[2 实验环境配置 1](#_Toc72247574)

[3 实验过程 2](#_Toc72247575)

[3.1 Poetic Walks 2](#_Toc72247576)

[3.1.1 Get the code and prepare Git repository 2](#_Toc72247577)

[3.1.2 Problem 1: Test Graph <String> 3](#_Toc72247578)

[3.1.3 Problem 2: Implement Graph <String> 4](#_Toc72247579)

[3.1.3.1 Implement ConcreteEdgesGraph 4](#_Toc72247580)

[3.1.3.2 Implement ConcreteVerticesGraph 7](#_Toc72247581)

[3.1.4 Problem 3: Implement generic Graph<L> 11](#_Toc72247582)

[3.1.4.1 Make the implementations generic 11](#_Toc72247583)

[3.1.4.2 Implement Graph.empty() 12](#_Toc72247584)

[3.1.5 Problem 4: Poetic walks 13](#_Toc72247585)

[3.1.5.1 Test GraphPoet 13](#_Toc72247586)

[3.1.5.2 Implement GraphPoet 14](#_Toc72247587)

[3.1.5.3 Graph poetry slam 14](#_Toc72247588)

[3.1.6 Before you’re done 15](#_Toc72247589)

[3.2 Re-implement the Social Network in Lab1 16](#_Toc72247590)

[3.2.1 FriendshipGraph类 16](#_Toc72247591)

[3.2.2 Person类 17](#_Toc72247592)

[3.2.3 客户端main() 17](#_Toc72247593)

[3.2.4 测试用例 18](#_Toc72247594)

[3.2.5 提交至Git仓库 19](#_Toc72247595)

[4 实验进度记录 20](#_Toc72247596)

[5 实验过程中遇到的困难与解决途径 20](#_Toc72247597)

[6 实验过程中收获的经验、教训、感想 21](#_Toc72247598)

[6.1 实验过程中收获的经验和教训 21](#_Toc72247599)

[6.2 针对以下方面的感受 21](#_Toc72247600)

# 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：

（1）针对给定的应用问题，从问题描述中识别所需的 ADT；

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

（3）根据 ADT 的规约设计测试用例；

（4）ADT 的泛型化；

（5）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）；

（6）测试 ADT 的实现并评估测试的覆盖度；

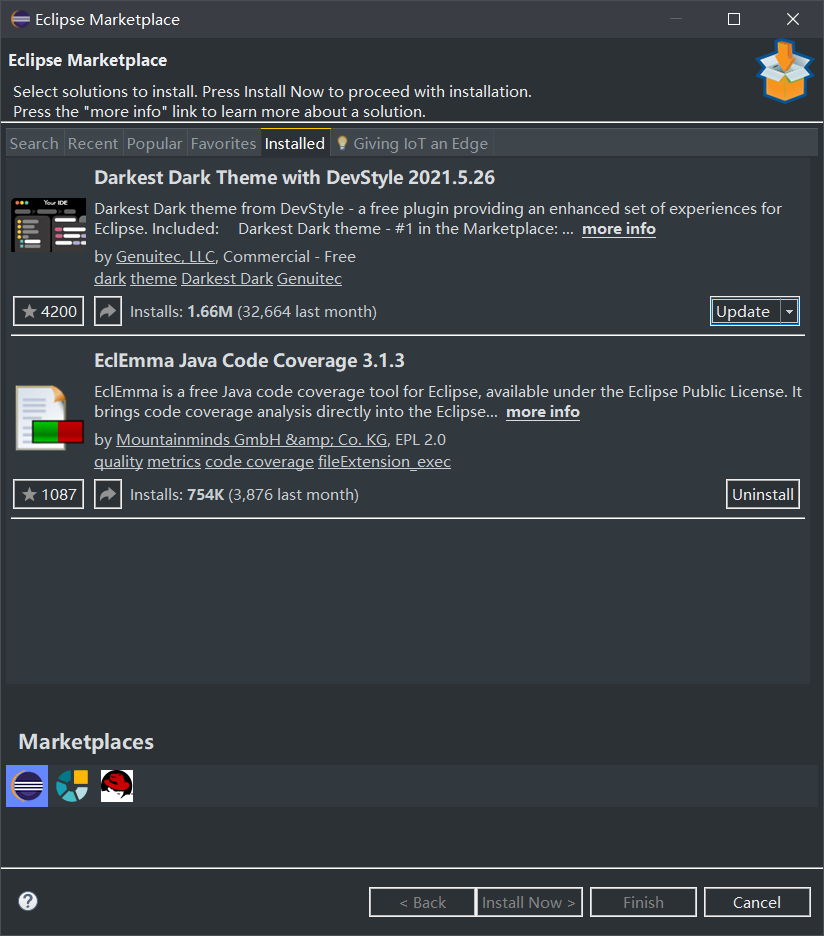
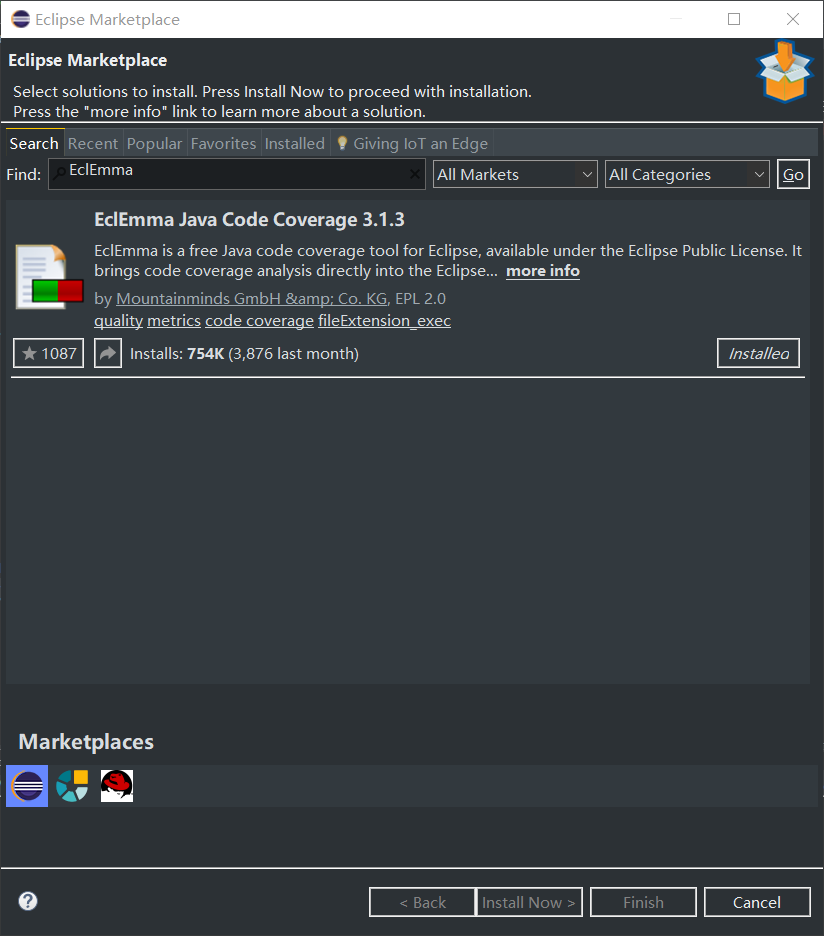
（7）使用 ADT 及其实现，为应用问题开发程序；

（8）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

1、安装配置Eclipse编译环境、JDK以及Git。

2、在Eclipse的marketplace中查找并安装EclEmma工具。



3、GitHub Lab2仓库的URL地址（Lab2-学号）：

https://github.com/ComputerScienceHIT/HIT-Lab2-Zz-sev-point.git

# 实验过程

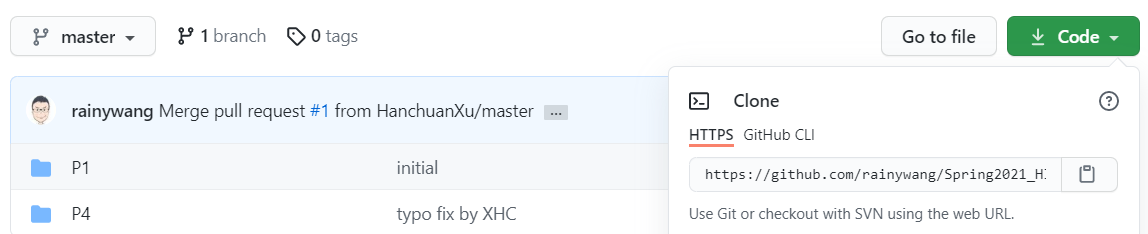
## Poetic Walks

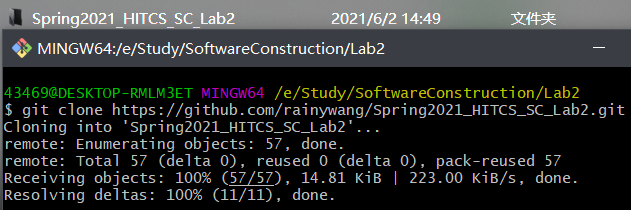
通过建立ConcreteEdgesGraph和ConcreteVerticesGraph两个类来实现graph接口。其中需要实现添加节点、添加边、移除节点、获得节点集合、获得某节点的目标集合、获得某节点的源集合、将图信息转换为字符串这几个功能。

在实现了以上两个类的基础上，将其应该与词句当中：根据一句句子构造一个图，并利用这张图对其他句子按照一定条件（目标句子中相邻两单词在图中两者之间正好通过另一个节点可达，则将该节点插入这两个单词中间，若有多个，则选取权值和最大者）进行扩充。

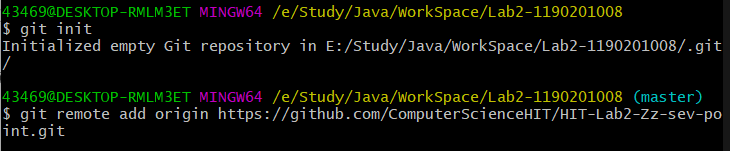
### Get the code and prepare Git repository

（1）从github中获得文件的url，2在本地文件夹内使用git bash的clone指令获得资料。





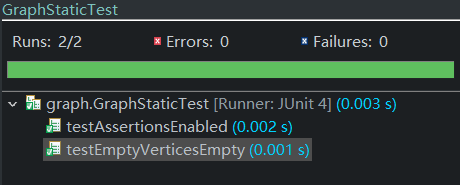
（2）创建本地仓库，连接自己的url。





### Problem 1: Test Graph <String>

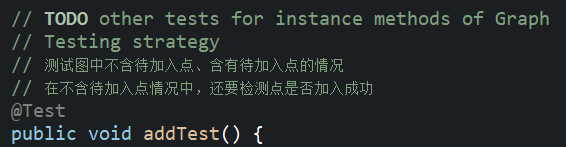
1. Static Test



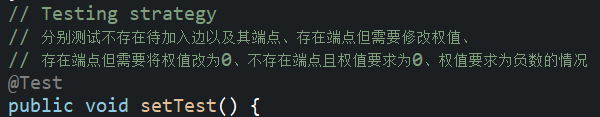
2、Instance Test

对于Graph中的每个方法进行逐个测试：

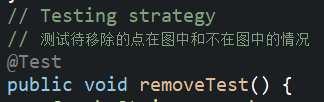
（1）addTest



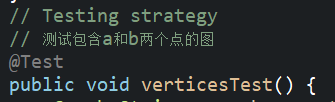
（2）setTest



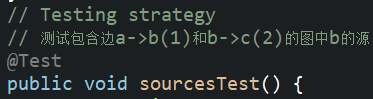
（3）removeTest



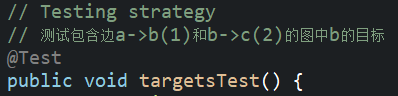
（4）verticesTest



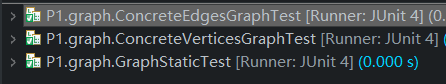
（5）sourcesTest

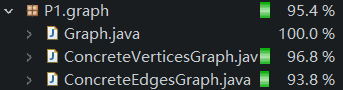


（6）targetsTest



测试结果：

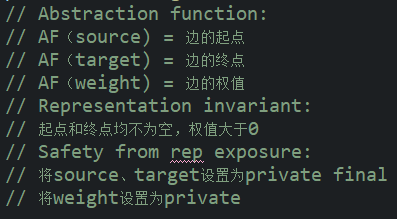




### Problem 2: Implement Graph <String>

#### Implement ConcreteEdgesGraph

1、Edge类的实现



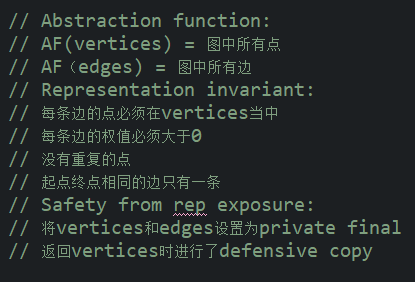
（1）在Edge中申明的三个私有类source、target、weight，分别为边的起点、终点、权重。



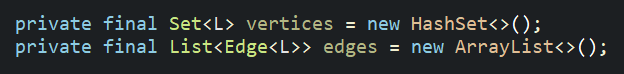
（2）Edge的方法

|  |  |
| --- | --- |
| Edge | 构造方法，初始化起点、终点以及权值。 |
| checkRep | 检查表示不变性，起点终点不为空且权值大于0。 |
| getSource | 返回起点，此处默认source类型L为不可变类型。 |
| getTarget | 返回终点，此处默认target类型L为 不可变类型。 |
| getWeight | 返回权重。 |
| equals | 此处对equals进行重写，使用instanceof判断object是否为Edge，是则强转为Edge类型，然后判断边是否相同，即起点、终点、权重全部相同。 |
| toString | 将边的信息以字符串形式返回，形式为：source+”->”+target+”:”+ weight+”\n”。 |

2、ConcreteEdgeGraph类的实现



（1）申明两个类vertices和edges，分别存储图中所有的点和边。

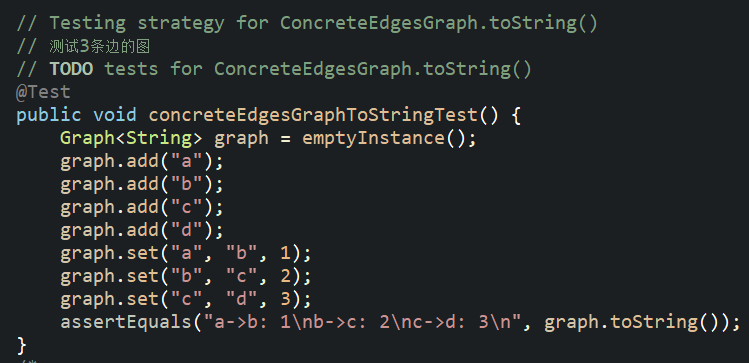


（2）ConcreteEdgeGraph的方法

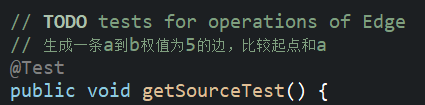
|  |  |
| --- | --- |
| ConcreteEdgeGraph | 构造函数，什么也不执行。 |
| checkRep | 检查表示不变性，edges中每条边的点都在vertices中且权值全为正。 |
| add | 添加一个点到vertices中，若传入为空或节点名已经存在，返回false，否则加入点返回true。 |
| set | 传入参数为source、target、weight。若weight为负，返回-1。否则，若source或target原本不存在图中，则添加未在图中的点，并添加source到target的边，权值设为weight，返回0；若两点均存在，但原本不存在边，则添加边，权值设为weight，返回0；若该边原本就存在，删除该边，重新添加边，并把权值设为weight，返回原本该边的权值。 |
| remove | 移除某一节点以及以该节点为某一端点的所有边。首先判断图中是否存在该点，若不存在，则返回false结束。否则，在vertices中删除该点，遍历edges，将以该点作为端点的边删除，最后返回true。 |
| vertices | 返回图中节点的集合。由于vertices为Set属于可变类型，此处采取defensive copy的策略，复制一个vertices再返回之。 |
| sourecs | 返回一个节点到整型的Map，含义是某节点的所有源点以及它们对应边的权值。申明一个Map，然后遍历egdes将符合要求的节点加入Map即可。 |
| targets | 返回一个节点到整型的Map，含义是某节点的所有目标点以及它们对应边的权值。申明一个Map，然后遍历egdes将符合要求的节点加入Map即可。 |
| toString | 将图的信息转化为字符串类型。实现方法为遍历edges并逐个调用对应边的toString方法。 |

3、测试

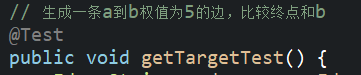
（1）对ConcreteEdgeGraph类toString方法的测试



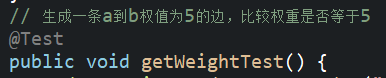
（2）对Edge类getSource方法的测试



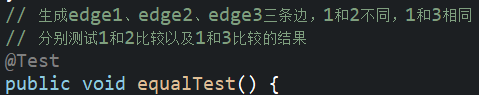
（3）对Edge类getTarget方法的测试



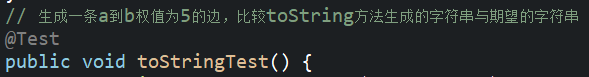
（4）对Edge类getWeight方法的测试



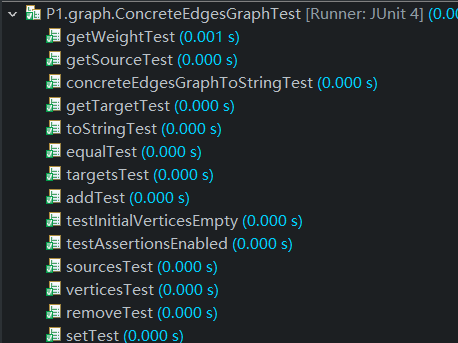
（5）对Edge类equal方法的测试

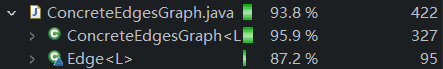


（6）对Edge类toString方法的测试



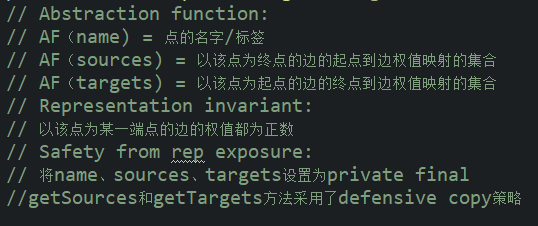
测试结果：



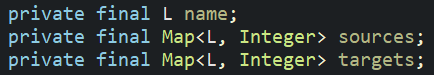


#### Implement ConcreteVerticesGraph

1、Vertex类的实现



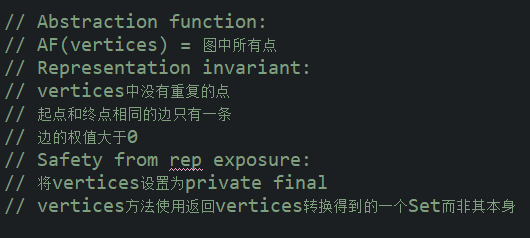
（1）在Vertex中申明的三个私有类name、sources、targets，分别为点的名字/标志、指向该点的边的起点到边权值的映射的集合、以该点为起点的边的终点到边权值的映射的集合。



（2）Vertex的方法

|  |  |
| --- | --- |
| Vertex | 构造方法，初始化点的名字/标志，新建两个空map来初始化sources和targets。 |
| checkRep | 检查表示不变性，起点终点不为空且权值大于0。 |
| getName | 返回点的名字/标签。 |
| getSources | 返回以该点为终点的边的起点集合到边权值的映射集合，并采取defensive copy。 |
| getTargets | 返回以该点为起点的边的终点集合到边权值的映射集合，并采取defensive copy。 |
| addSource | 传入参数为节点名source和权重weight。若weight为负，返回-1退出；若weight为0，则调用removeSource从sources中溢出source项，返回removeSource即原本source到该点边的权值；若weight为正，使用map.put添加，并将其返回值返回（为null时返回0）。 |
| addTarget | 传入参数为节点名target和权重weight。若weight为负，返回-1退出；若weight为0，则调用removeTarget从sources中溢出source项，返回removeTarget即原本target到该点边的权值；若weight为正，使用map.put添加，并将其返回值返回（为null时返回0）。 |
| removeSource | 使用map的remove方法将sources中目标点移除，返回该方法返回值（为null时返回0）。 |
| removeTarget | 使用map的remove方法将targets中目标点移除，返回该方法返回值（为null时返回0）。 |
| toString | 将边的信息以字符串形式返回，形式为：name+”\nsource:\n”+sources+”\ntargets\n”+targets。 |

2、ConcreteVerticesGraph类的实现



（1）申明一个类vertices图中所有的点。

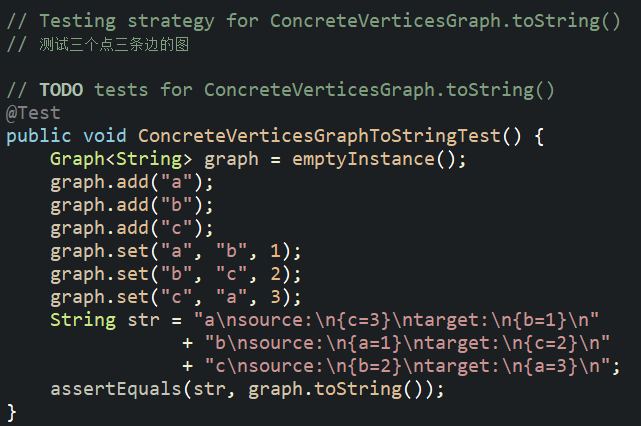


（2）ConcreteEdgeGraph的方法

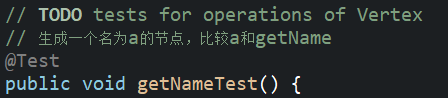
|  |  |
| --- | --- |
| ConcreteEdgeGraph | 构造函数，什么也不执行。 |
| checkRep | 检查表示不变性，检查vertices中是否有重复的点。 |
| add | 添加一个点到vertices中，若待加入节点名为空或vertices中已经存在节点名与待加入节点重复，返回false，否则加入节点并返回true。 |
| set | 传入参数为source、target、weight。首先调用add将source和taregt加入图的点集中，然后遍历vertices，调用Vertex的addTarget和addSource方法将source和target分别加入target的sources和source的taregts中，将addSource的返回值返回。 |
| remove | 移除某一节点以及以该节点为某一端点的所有边。首先遍历vertices查看是否存在目标节点，不存在返回false；否则，找到以该点为端点的边的其他端点，分别删除他们的sources和targets中该点对应的项。 |
| vertices | 返回图中节点的集合。将vertices转化为Set并返回。 |
| sourecs | 返回一个节点到整型的Map，含义是某节点的所有源点以及它们对应边的权值。申明一个Map，然后遍历vertices将符合要求的节点加入Map即可。 |
| targets | 返回一个节点到整型的Map，含义是某节点的所有目标点以及它们对应边的权值。申明一个Map，然后遍历vertices将符合要求的节点加入Map即可。 |
| toString | 将图的信息转化为字符串类型。实现方法为遍历vertices并逐个调用对应点的toString方法。 |

3、测试

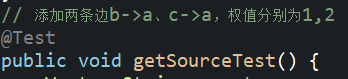
（1）对ConcreteVerticesGraph类toString方法的测试



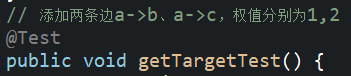
（2）对Vertex类getName方法的测试



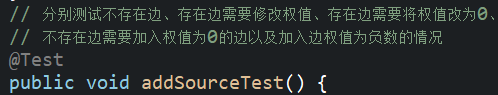
（3）对Vertex类getSource方法的测试



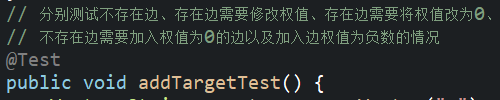
（4）对Vertex类getTarget方法的测试



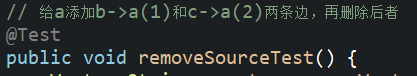
（5）对Vertex类addSource方法的测试



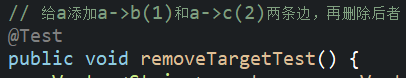
（6）对Vertex类addTarget方法的测试



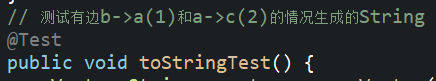
（7）对Vertex类removeSource方法的测试



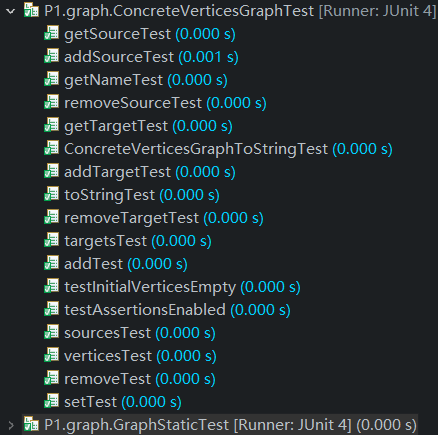
（8）对Vertex类removeTarget方法的测试

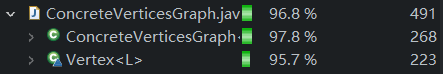


（9）对Vertex类toString方法的测试



测试结果：

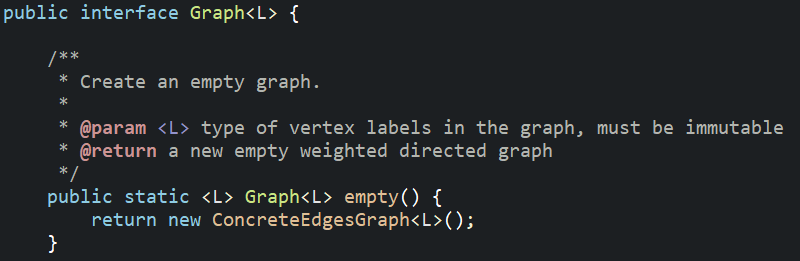


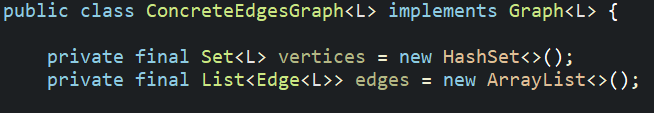


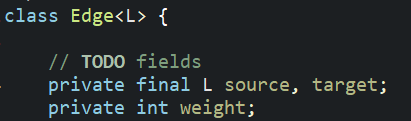
### Problem 3: Implement generic Graph<L>

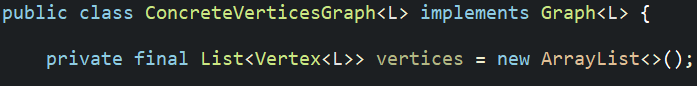
#### Make the implementations generic

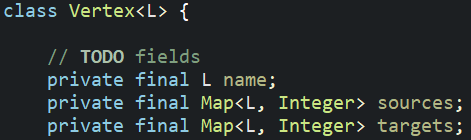
使用泛型编程实现即可，即将String修改为L。





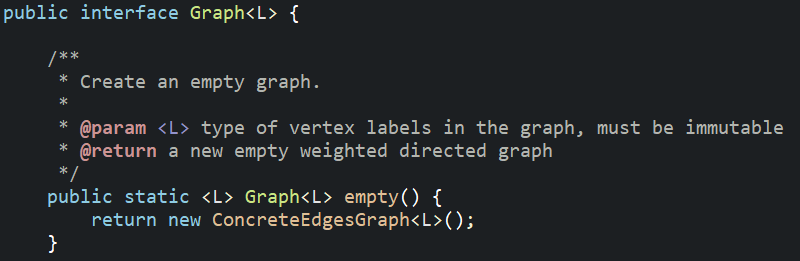






#### Implement Graph.empty()

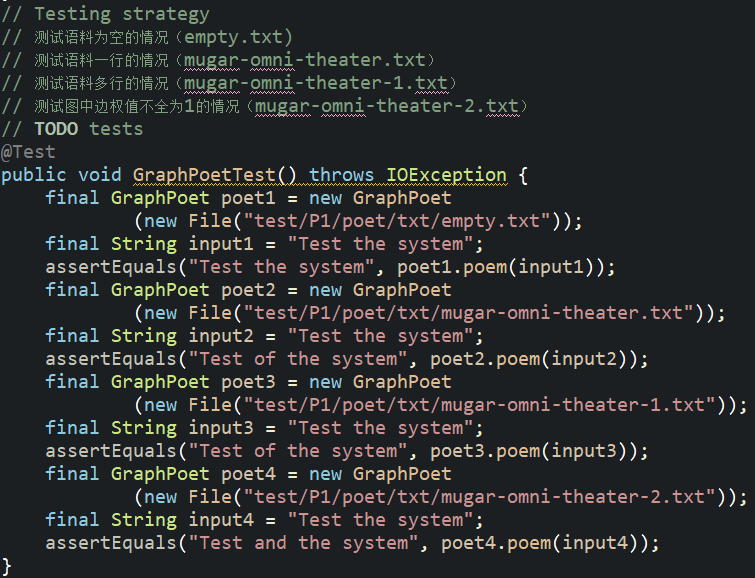
调用一个具体实例即可。



### Problem 4: Poetic walks

#### Test GraphPoet

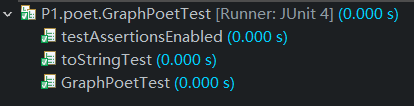
1、对GraphPoet类的测试，即对其中的GraphPoet构造函数以及poem方法进行测试



2、对GraphPoet类中toString方法进行测试



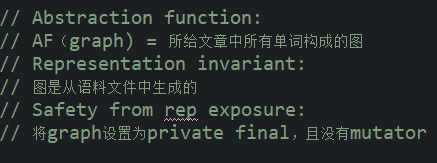
3、测试结果





#### Implement GraphPoet

（1）AF,RI和Safety from rep exposure



（2）如图定义一个私有Graph<String>类

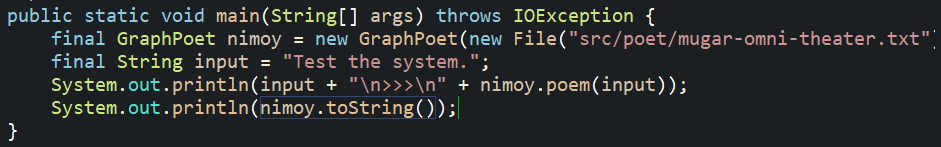


（3）GraphPoet类中方法的实现

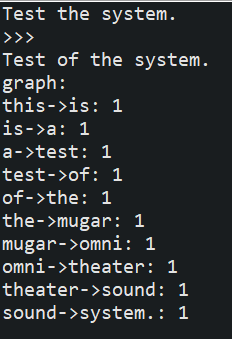
|  |  |
| --- | --- |
| GraphPoet | 构造函数，一行一行地读入，并存储在List当中，每次取相邻的两个单词构成一条边加入图中。 |
| checkRep | 检查表示不变性 |
| poem | 传入需要扩充的字符串input，通过split将其转化为字符串数组按顺序存入List当中，申明一个StringBuilder用于存储输出。每次取List中第i个单词加入输出当中，然后在图中查找是否有以它为源且以第i+1个单词为目标的节点，有则取对应两边权值和最大者加入输出。最后再加入最后一个单词。 |
| toString | 调用graph中的toString方法 |

#### Graph poetry slam

在main函数中添加一行输出nimoy的toString。



运行结果如下：



### 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。

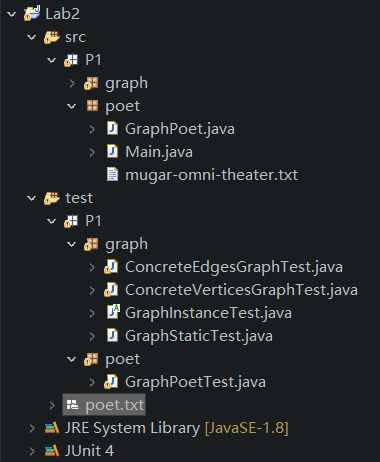
通过Git提交当前版本到GitHub上你的Lab2仓库。







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

该实验要求我们使用Poetic Walks中定义的Graph<L>接口重新实现Lab1中的FriendshipGraph类，并且尽可能使用Graph<L>中的方法来实现FriendshipGraph类中要求实现的方法。

### FriendshipGraph类

（1）这里采取ConcreteEdgeGraph类来实现，声明一个私有类graph。



（2）FriendshipGraph方法实现

|  |  |
| --- | --- |
| FriendshipGraph | 构造函数，new一个ConcreteEdgeGraph给私有类graph。 |
| addVertex | 调用graph.add实现，并将该方法返回值返回。 |
| addEdge | 调用graph.set实现。 |
| getDistance | 与Lab1中一样采取广度优先的策略进行搜索，不同的是搜索中对某个Person朋友的查找不使用Person本身的私有类Friends，而是使用graph.targets方法。最后返回距离值，若两人不可达，则返回-1。具体实现如下： |
| getGraph | 由于graph可能会很复杂，这里不使用defensive copy，直接返回私有类graph。 |

### Person类

（1）由于使用ConcreteEdgeGraph类来实现FriendshipGraph，其中的私有类edges已经存储了每个Person的所有friends，所此处Person不在申明私有类Friends存储friends，只申明Person的name。

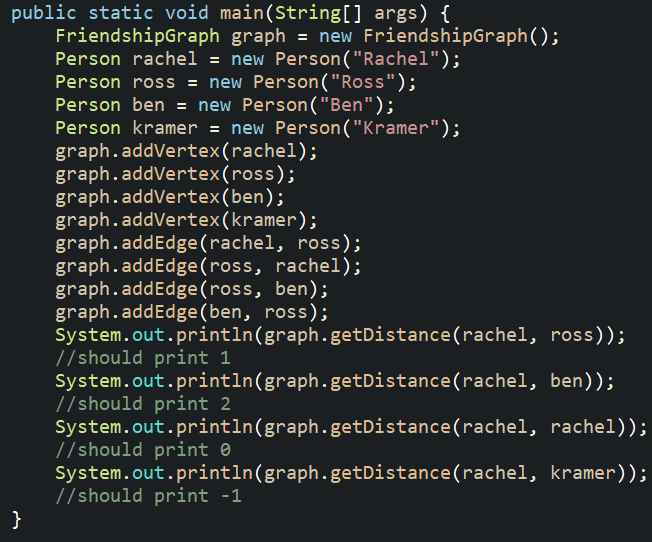


（2）Person方法的实现

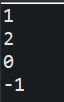
|  |  |
| --- | --- |
| Person | 构造函数，将传入参数作为Person的name。 |
| getName | 直接返回name。 |

### 客户端main()

采用Lab1中所给的main函数。



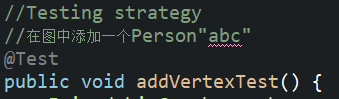
运行结果如下：



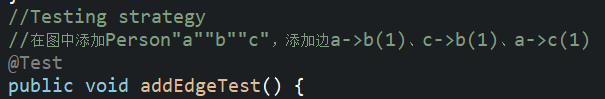
### 测试用例

采取与Lab1相同的测试策略。

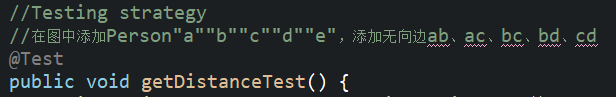
（1）addVertexTest

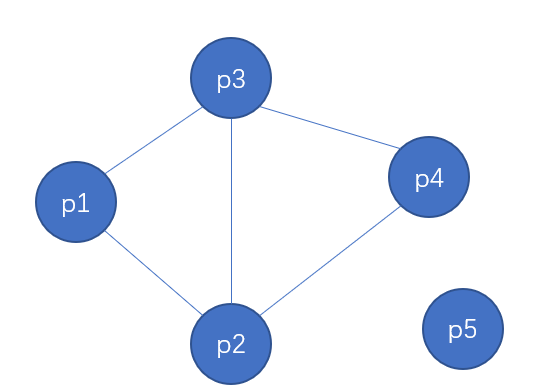


（2）addEdgeTestTest



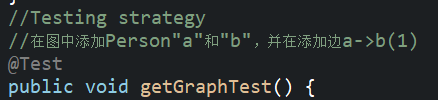
（3）getDistanceTest



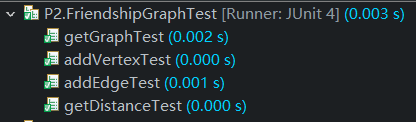


测试用例

（4）getGraphTest（使用toString方法输出图进行比较）



（5）测试结果

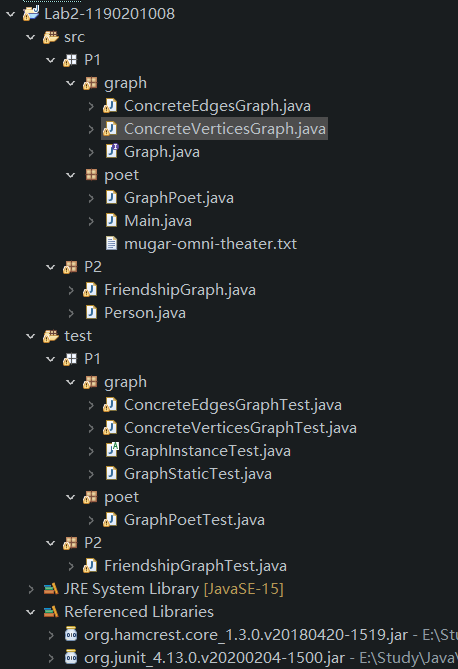




### 提交至Git仓库

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。

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 计划任务 | 实际完成情况 |
| 5.25 | 18:30-22:00 | 完成P1中的ConcreteEdgeGraph类 | 未完成 |
| 5.27 | 18:30-21:00 | 完成P1中的ConcreteEdgeGraph类 | 完成 |
| 5.28 | 14:00-17:00 | 完成P1中的ConcreteVerticesGraph类 | 未完成 |
| 5.28 | 19:00-21:00 | 完成P1中的ConcreteVerticesGraph类 | 完成 |
| 5.29 | 9:00-11:30 | 完成P1中对ConcreteEdgeGraph类的测试以及对ConcreteEdgeGraph类的修改 | 未完成 |
| 5.30 | 19:30-21:00 | 完成P1中对ConcreteEdgeGraph类的测试以及对ConcreteEdgeGraph类的修改 | 完成 |
| 6.1 | 9:00-11:30 | 完成P1中对ConcreteVerticesGraph类的测试以及对ConcreteVerticesGraph类的修改 | 完成 |
| 6.1 | 18:00-21:30 | 完成P1中的GraphPoet类及其测试 | 完成 |
| 6.2 | 19:00-21:30 | 完成P2及其测试 | 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 |
| 起初做P1时对AF, RI, rep exposure, checkRep等一些术语比较陌生，在编程时比较困惑，对编程目标不是特别明确。 | 后来通过老师的讲解以及自己课下对相关资料的查阅学习，解决了这些困惑。 |
| 在写P1的GraphPoet时，对题意不太理解，不知道类中的方法需要实现什么。 | 通过阅读所给链接以及代码框架中所给的注释最终理解了。 |
| 编写测试代码时，很难将各种特殊情况都全部考虑。 | 通过使用EclEmma工具查看测试代码的覆盖率，通过查找未覆盖的代码得到为考虑的情况，在将这些情况加入到测试代码中。 |

# 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

对自行设计类、方法以及泛型编程有了更加深刻的认识，对AF, RI, rep exposure, checkRep等概念也有了一定程度的理解。但是在编程时，有时考虑的还是不够全面，对于检查RI、defensive copy等操作有时还是会忽略。对于Java仍然需要多加练习，提高熟练度。

## 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：面向应用场景通常是将一个完整的过程分割开来，然后逐个实现；而面向ADT的编程则是以一个对象为核心进行展开。在编程思路上有很大的区别。

1. 使用泛型和不使用泛型的编程，对你来说有何差异？

答：具体编程过程中感觉差异不大，不过泛型编程需要考虑到不可使用数组，但它在使用中更加灵活方便。

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：优势是可以及时更正代码，降低不必要的时间浪费，而且在多人编程中可以实现代码编写和测试用例编写并行，从而进一步提高效率。

不太适应，因为编写测试用例时可能会出现个别特殊情况未考虑的情况。

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：提高代码利用率，降低了不必要的重复工作量，节省时间又降低成本。

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：意义在于为编程理清思路，同时帮助提高编写出的代码的健壮性。以后编程中我会坚持这么做。

1. 关于本实验的工作量、难度、deadline。

答：本实验工作量适中，在理解题目、编写注释上花费了比较多的时间，难度适中，deadling很合适。

1. 《软件构造》课程进展到目前，你对该课程有何体会和建议？

体会：该课程帮助我对Java有了一个由浅到深的理解，对我的编程能力提高有很大的帮助，同时也让我对软件开发流程有了一个比较清晰的认识。

建议：希望老师对开设一些习题课，通过一些具体案例的较为详细的分析帮助我们更好地学习和理解。