

**2018年春季学期  
计算机学院大二软件构造课程**

**Lab 2实验报告**

|  |  |
| --- | --- |
| 姓名 | 穆添愉 |
| 学号 | 1160301008 |
| 班号 | 1603010 |
| 电子邮件 | 1417553133@qq.com |
| 手机号码 | 15636094072 |

**目录**

[1 实验目标概述 1](#_Toc510993493)

[2 实验环境配置 1](#_Toc510993494)

[3 实验过程 2](#_Toc510993495)

[3.1 Poetic Walks 2](#_Toc510993496)

[3.1.1 Get the code and prepare Git repository 3](#_Toc510993497)

[3.1.2 Problem 1: Test Graph <String> 3](#_Toc510993498)

[3.1.3 Problem 2: Implement Graph <String> 6](#_Toc510993499)

[3.1.3.1 Implement ConcreteEdgesGraph 6](#_Toc510993500)

[3.1.3.2 Implement ConcreteVerticesGraph 11](#_Toc510993501)

[3.1.4 Problem 3: Implement generic Graph<L> 17](#_Toc510993502)

[3.1.4.1 Make the implementations generic 17](#_Toc510993503)

[3.1.4.2 Implement Graph.empty() 17](#_Toc510993504)

[3.1.5 Problem 4: Poetic walks 18](#_Toc510993505)

[3.1.5.1 Test GraphPoet 18](#_Toc510993506)

[3.1.5.2 Implement GraphPoet 19](#_Toc510993507)

[3.1.5.3 Graph poetry slam 20](#_Toc510993508)

[3.1.6 Before you’re done 21](#_Toc510993509)

[3.2 Re-implement the Social Network in Lab1 22](#_Toc510993510)

[3.2.1 FriendshipGraph类 22](#_Toc510993511)

[3.2.2 Person类 24](#_Toc510993512)

[3.2.3 客户端main() 24](#_Toc510993513)

[3.2.4 测试用例 25](#_Toc510993514)

[3.2.5 提交至Git仓库 29](#_Toc510993515)

[3.3 The Transit Route Planner（选做，额外给分） 29](#_Toc510993516)

[P3要求根据输入文本建立一个交通网络图，然后输入起始站和终点站，从而规划出一条最短路。由于每条边带有权重，所以不能使用简单的BFS，而是使用Dikjstra算法来进行最短路的求取。 29](#_Toc510993517)

[3.3.1 Stop 设计 29](#_Toc510993518)

[3.3.2 Itinerary类设计 30](#_Toc510993519)

[3.3.3 TrafficGraph类设计 31](#_Toc510993520)

[4 实验进度记录 34](#_Toc510993521)

[5 实验过程中遇到的困难与解决途径 34](#_Toc510993522)

[6 实验过程中收获的经验、教训、感想 35](#_Toc510993523)

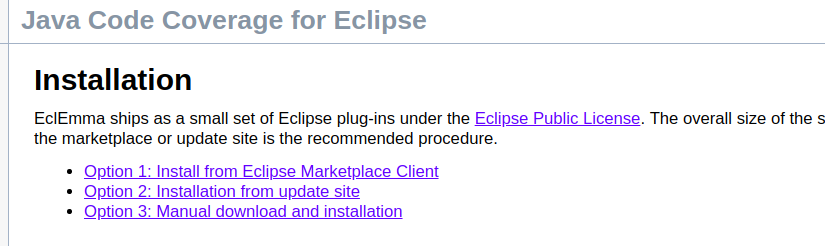
# 实验目标概述

本次试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：

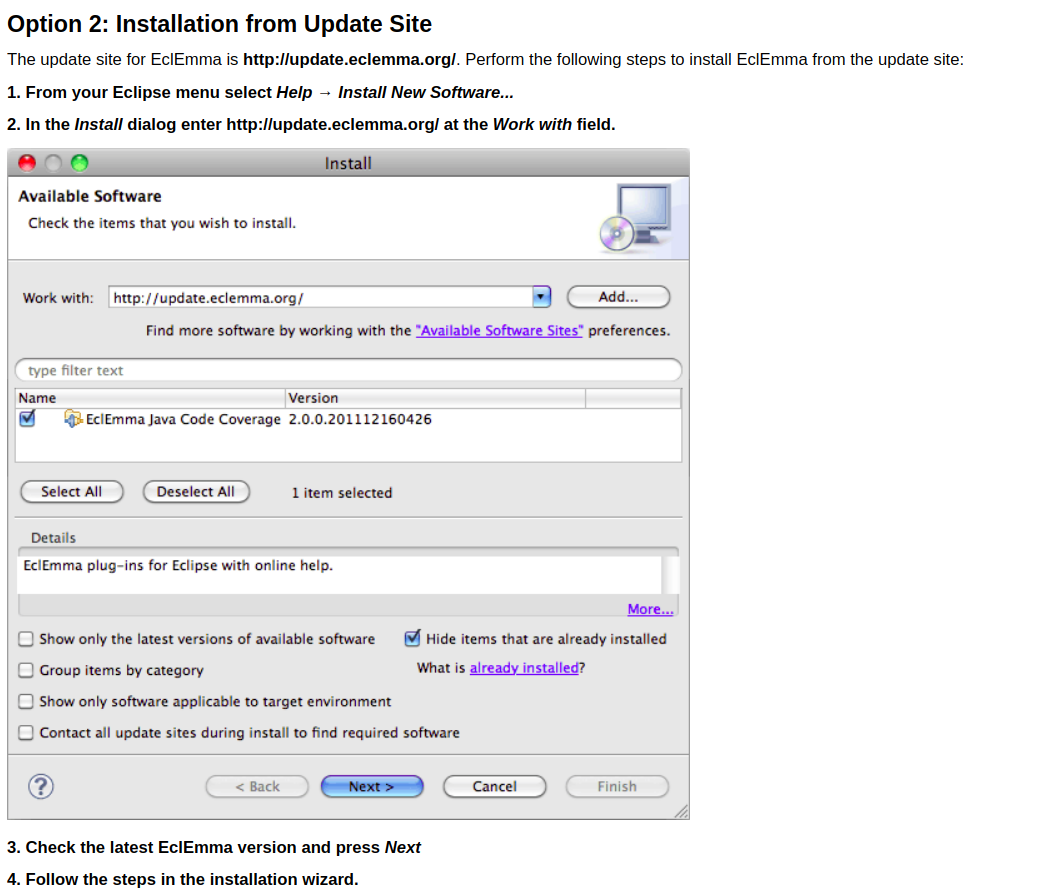
1. 针对给定的应用问题，从问题中描述识别所需的ADT；
2. 设计ADT规约，并评估质量
3. 根据ADT规约设计测试用例
4. 根据规约设计ADT的多种实现，针对每种实现，设计其表示、表示不变性、抽象过程
5. 使用OOP实现ADT，并判定表示不变性是否违反、各实现是否存在表示泄露

# 实验环境配置

本实验要求配置插件EclEmma，官网提供了三种安装方法，任选其一即可



我选择了第二种安装方法，也就是从他给的网址下载压缩包，然后按照指引一路导入到Eclipse中即可，比较简单，并不是很难，安装流程官网也已经给出，按照指引一步一步去操作就好



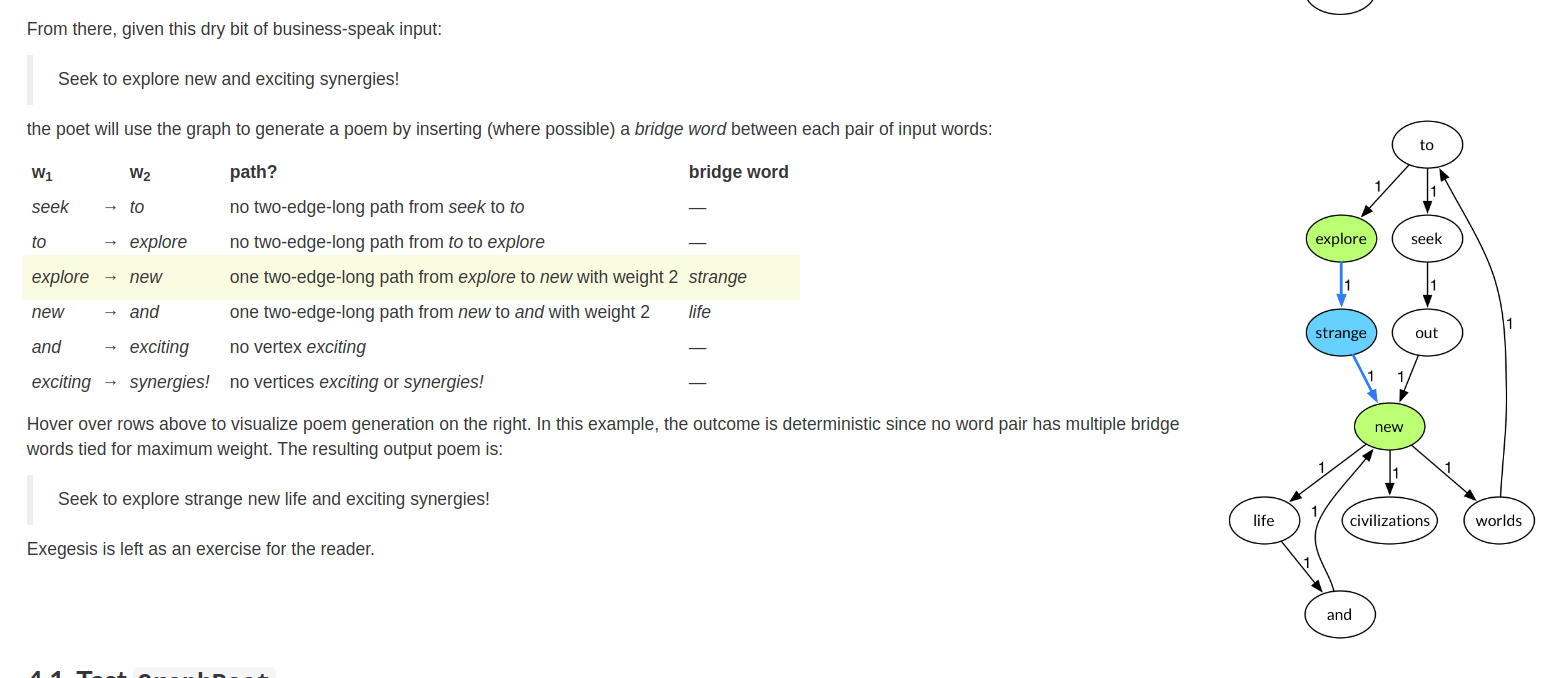
下面是我仓库的网址：

https://github.com/ComputerScienceHIT/Lab2-1160301008

# 实验过程

## Poetic Walks

首先要实现concreteVerticesGraph和concreteEdgesGraph两种生成图的方法，对于图中顶点或边的操作都是继承自Graph接口然后利用他们来生成我们想要的传统意义的图，然后，参照Poetic Walk中的描述来进行一个类似于扩句的操作，达到所谓的“创作出更美妙的诗篇”。



### Get the code and prepare Git repository

使用指令:

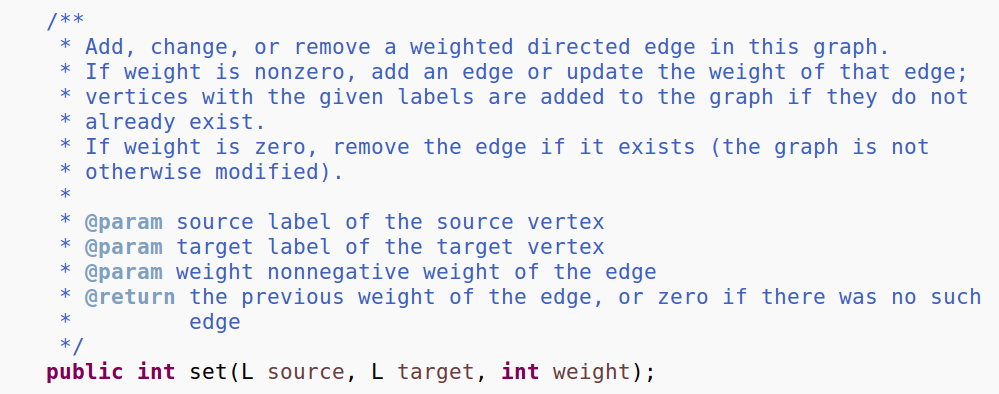
git clone https://github.com/rainywang/Spring2018\_HITCS\_SC\_Lab2.git

将源码复制到本地仓库.

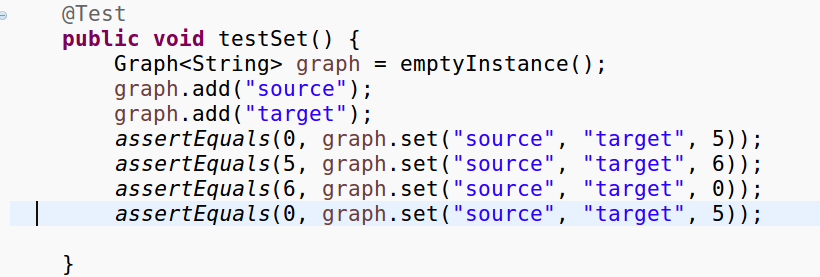
### Problem 1: Test Graph <String>

首先，根据要求，要完善GraphInstanceTest操作，来达到实验要求，每个测试方法都要先按照Graph接口中每个方法所给出的spec来进行测试。

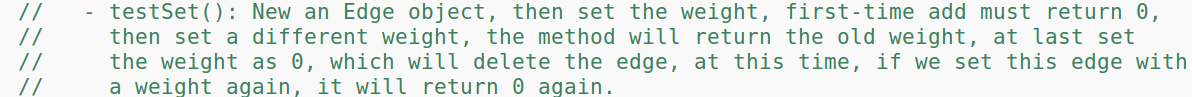
（1）set方法测试：



根据set函数的要求，要求如果输入的边不存在的话，则新建一条边加入到图中，并且返回0，如果存在这条边，那么修改这条边的值，并返回接旧的权重值，如果传进去的参数为0，那么相当于删除掉这条边，于是我建立了下面的测试文件：

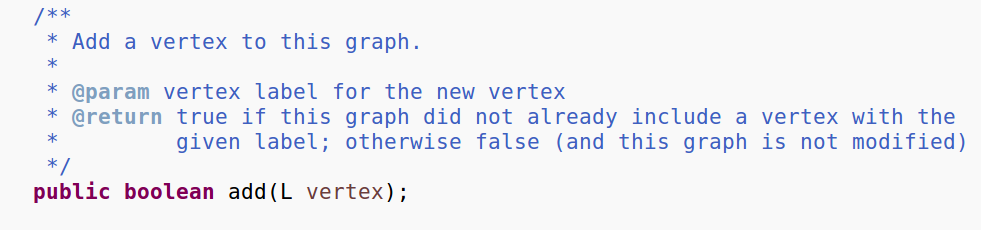


和如下的test strategy



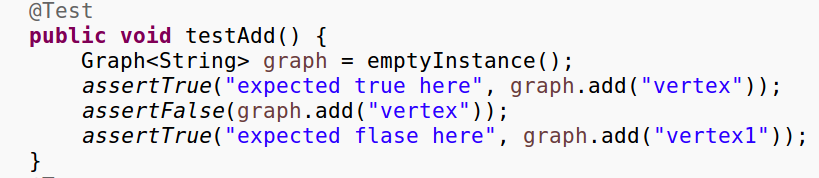
新建一条边，然后加入到图中，测试是不是返回0，然后修改这条边的权重，然后修改这条边的权重，观察是不是返回旧的权重值，然后删除这条边，判断是否加入成功。至此，这个方法就测试完毕。

1. add方法测试：

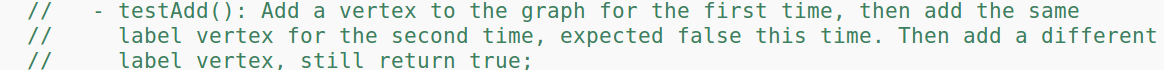


加入一个点，若图中已存在这个点，那么就返回false，否则就将这个点加入到图中，至此，这是add（）方法的功能实现。

所以我写了如下的测试文件：

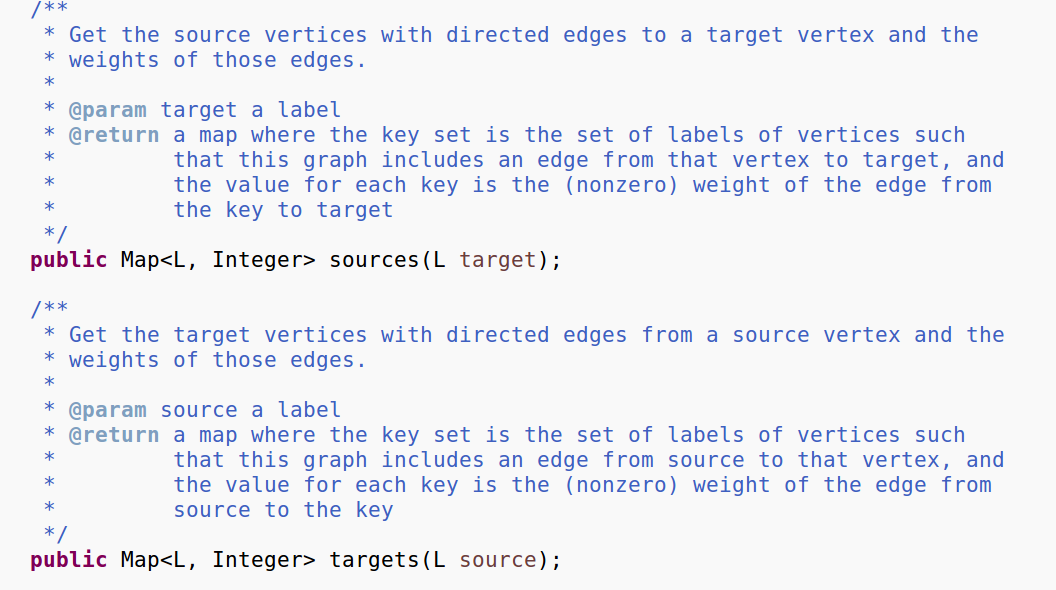


和如下的test strategy



只需要加入一个存在的点判断返回值为false，再加入一个新的点判断返回true即可。

1. source（）/target（）方法测试：



这两个方法类似，可以说是互补方法，分别是返回所有以这个点为起点的边的另一个节点，和所有以这条边为终点的边的另一个节点。

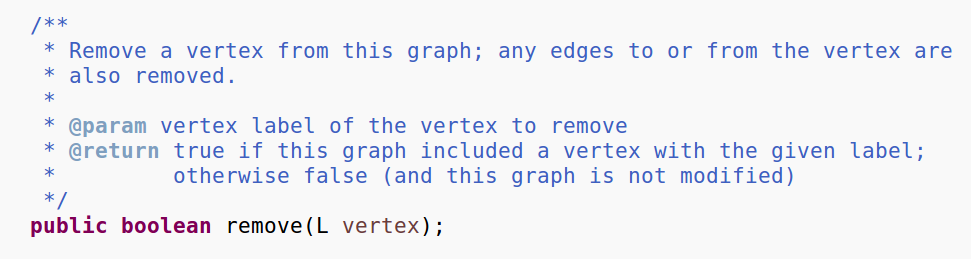
所以我有如下的测试文件：



和如下的test strategy

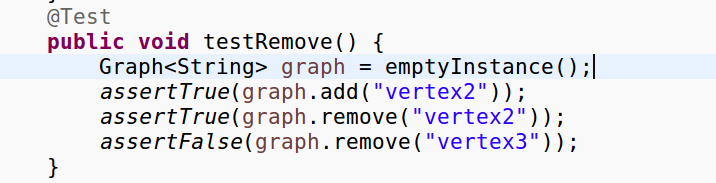


1. remove（）方法测试



这个方法就是要首先从顶点集中删除掉这个节点，然后再删除掉所有和这个节点相关的边，无论是以这条边为起点的边，还是以这条边为终点的边，都要删除掉，这样就可以来进行测试。

有如下的测试文件：



并有如下的test strategy

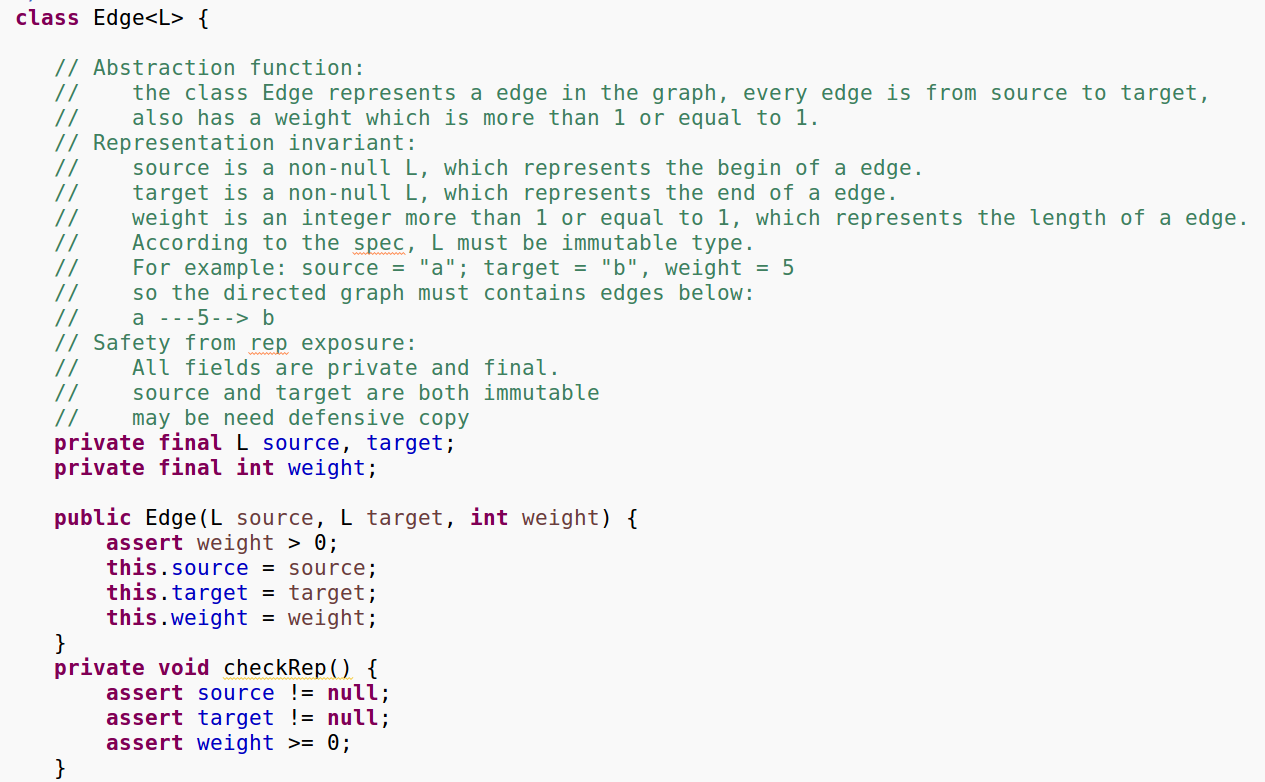


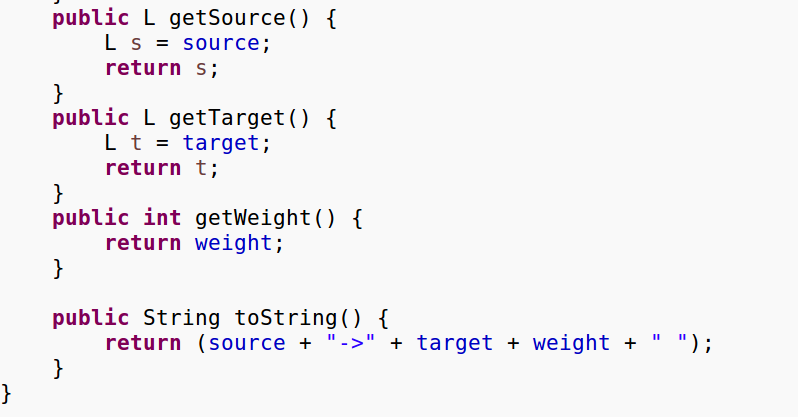
### Problem 2: Implement Graph <String>

这里要针对两种生成图的方法来进行图的实现，并编写AF、RI、checkrep（）等

#### Implement ConcreteEdgesGraph

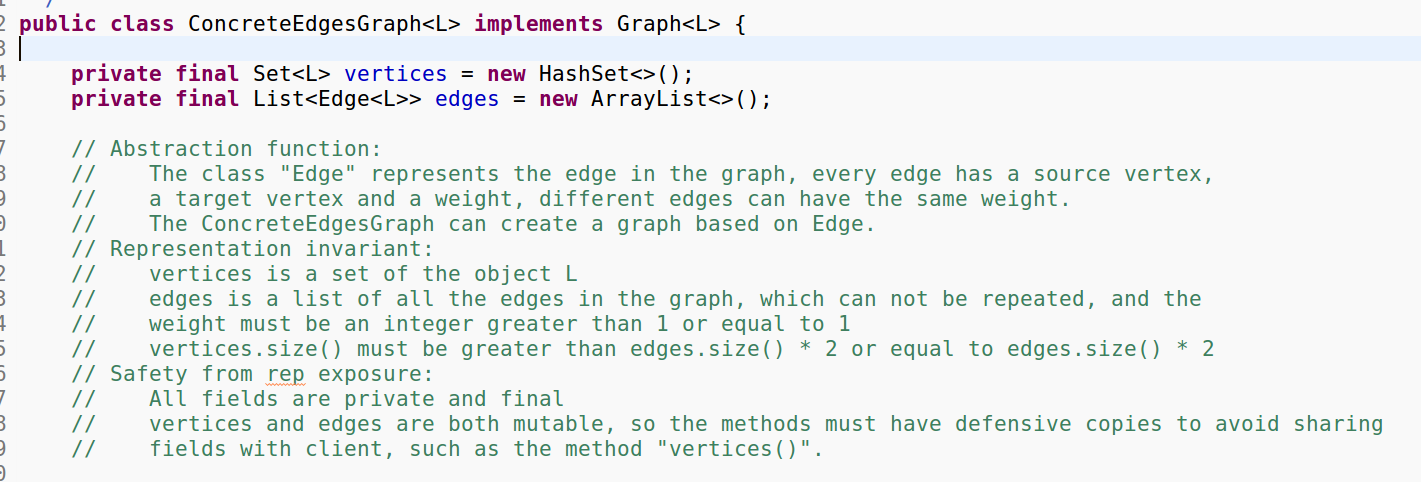
首先是对Edge类进行编写，实现如下：





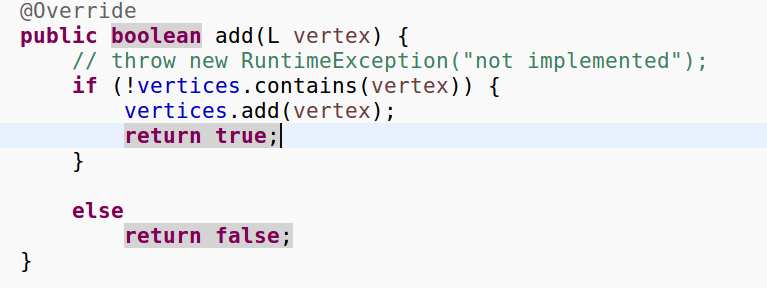
Edge类的编写很多是针对于Graph的操作，Edge类中的成员变量有：L source，L target， int weight，这样有了起点、终点、每条边的权重就可以唯一确定一条边。

接下来是对ConcreteEdgesGraph类的编写，根据Graph中的spec来进行编写



首先明确，vertices是一个存着所有节点的集合，edges是一个存着所有边的list，这样子，一个图就可以这样被表示出来。下面是各个方法的具体实现：

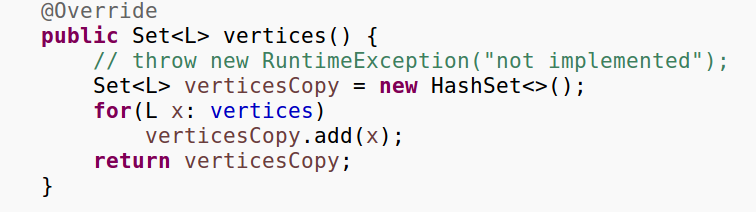
1. add（）方法就只要像vertices中来进行加顶点就好,只需要在加入定点之前来进行判断一下就好，若图中已经存在这个点，那么就返回false，否则就可以加入，其实运用Hashset中的元素不重复的性质就可以实现。



1. set方法首先就要判断图中是不是已经存在这条边，如果存在的话， 那么就找到这条边并修改边的权重，并返回旧的权重值。若不存在这条边并且weight参数不为0，那么就在图中新建这条边，并返回0。如果weight参数为0，那么将该边删除。



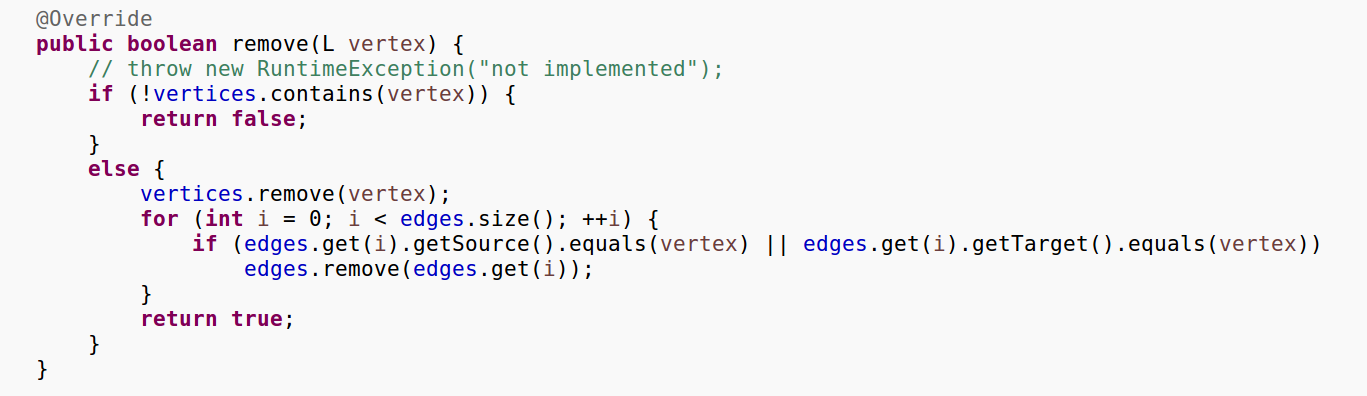
1. vertices方法在这个类中比较好写，只要将成员变量vertices进行返回即可，注意set为mutable型数据结构，所以需要防止表示暴露，进行defensive copy。



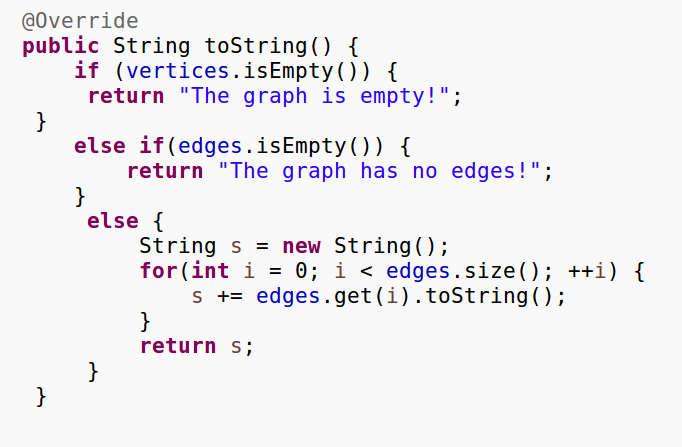
1. sources方法/targets方法，只需要遍历edges list，来进行寻找所有以该点为起点的边或以该点为终点的边就好，这两个方法算是互补方法。



1. remove方法主要就是先将该点从vertices中移除掉，然后将所有与该点相关联的边也都删除掉。



1. toString方法就是将该图以String的形式输出，这样就可以更加直观的显示这张图，

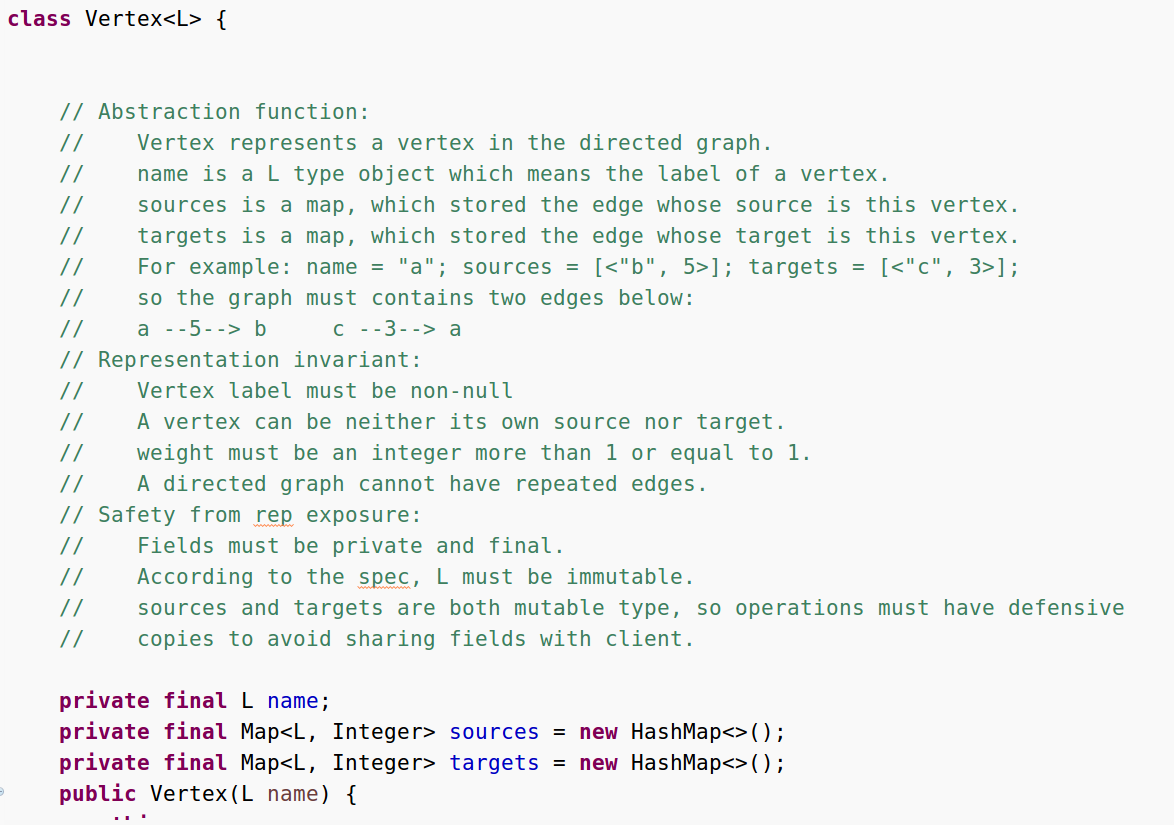


这里我为每个节点写了一个toString方法，用来输出每个节点信息，在这里可以循环调用。

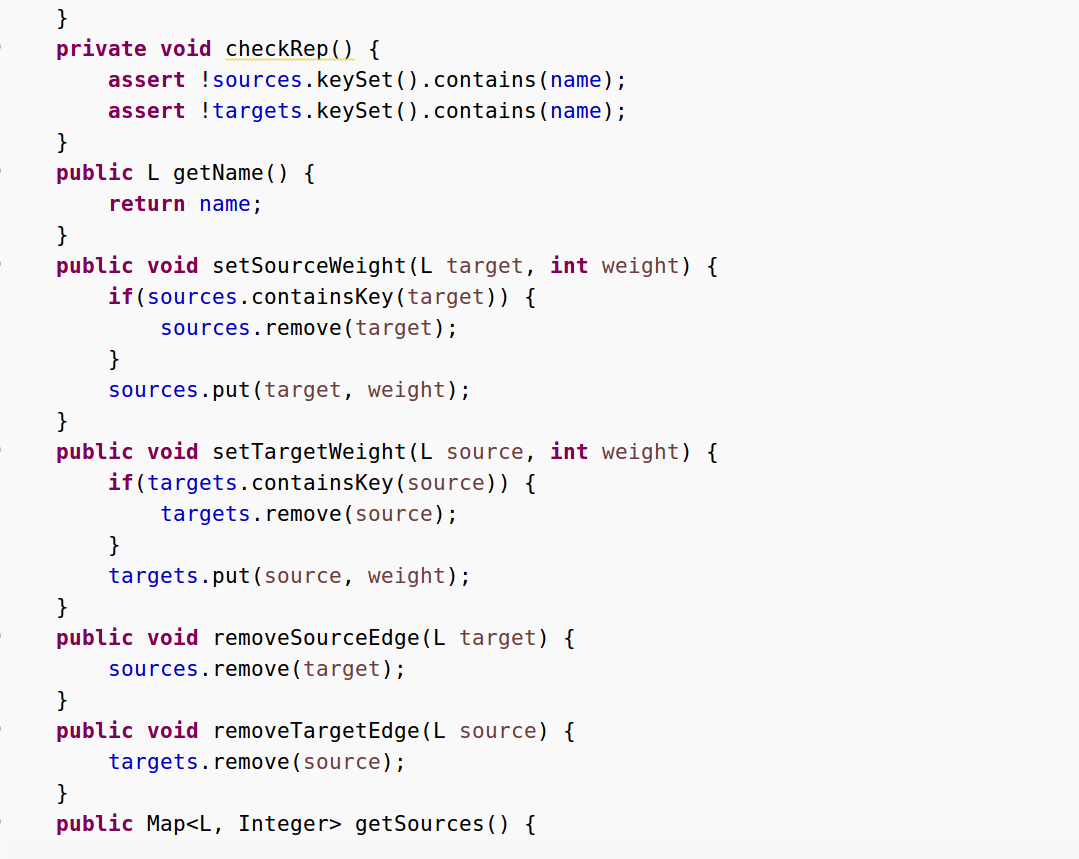
#### Implement ConcreteVerticesGraph

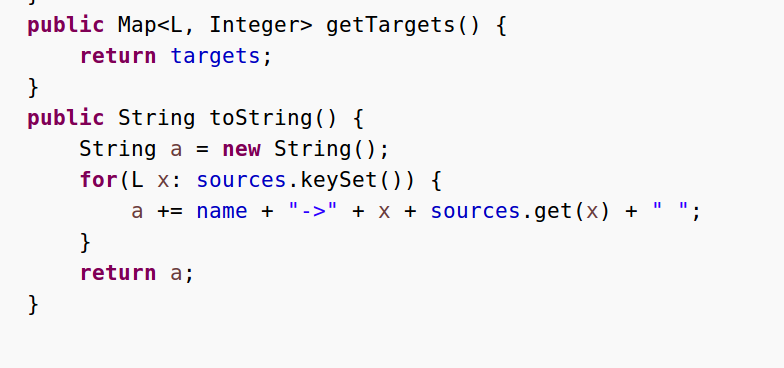
实现过ConcreteEdgesGraph之后，这个类和那个类似，思路相对来说比较明确，

首先还是Vertex类的编写

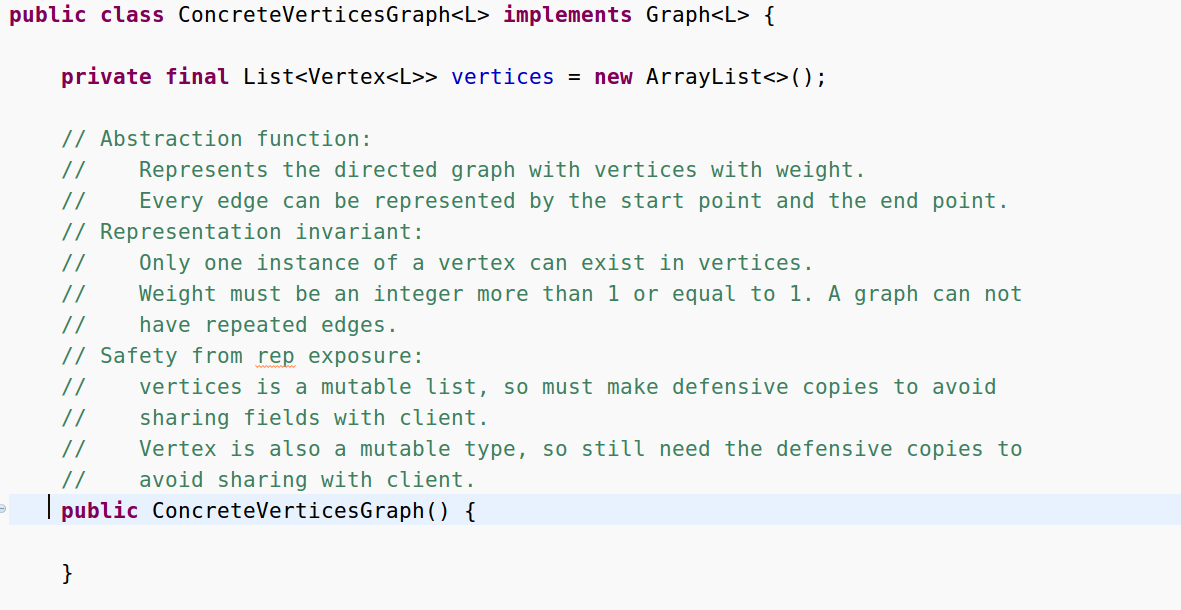


每个Vertex类中，都有L name， 来代表这个节点的标签，然后有一个sources和targets的map来相应的存取边，分别代表以该边为起点的边和以该边为终点的边。





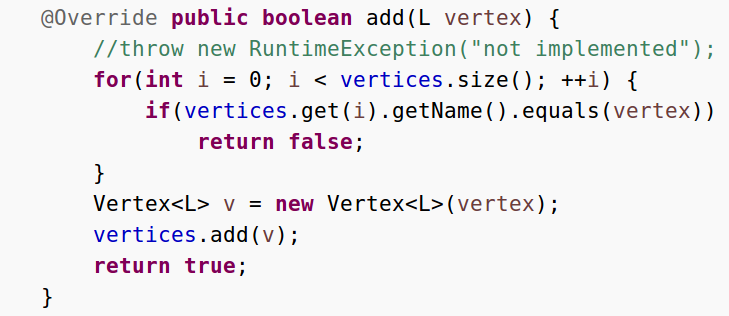
其次是对ConcreteVerticesGraph类的实现：



首先是AF 、RI 、rep的编写，以下是各个方法的编写：

上面已经实现了对ConcreteEdgesGraph的编写，这样每个方法都已经熟悉了思路，只需要针对不同的数据结构，也就是Vertex类来进行编写，难度不是特别的高。

1. add（）方法实现：



和之前一样，加点之前先遍历一遍所有的点集，如果之前没加过该点，那么将其加入到点集中，并返回true，否则返回false。

（2）





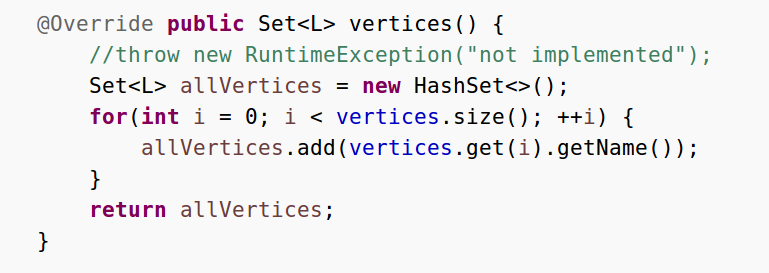
Set方法和之前的一样，这里不再细说

1. remove方法实现



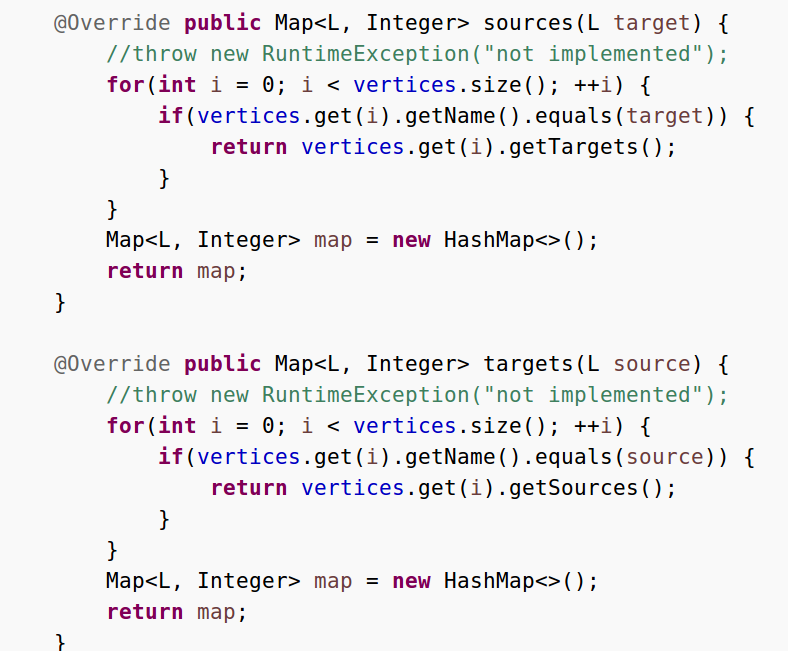
这里的remove方法相对来说比较长，是因为数据结构的改变，造成了遍历的时候不太方便，所以代码会变长。

1. vertices（）方法实现



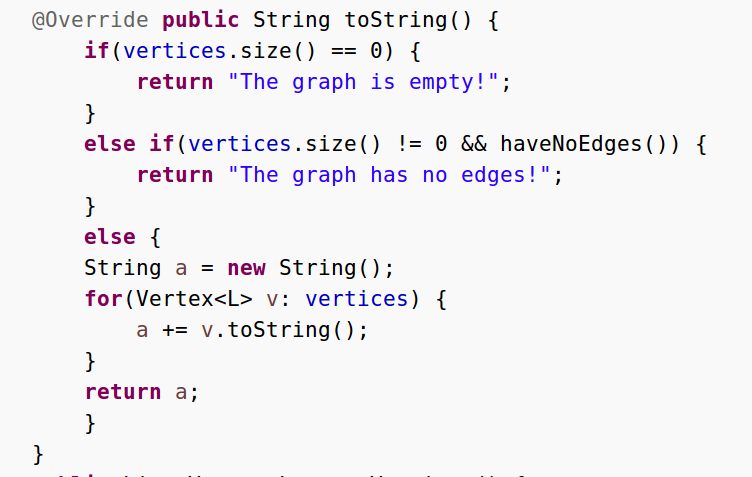
直接返回成员变量vertices即可，注意Set是mutable变量，所以为了防止表示暴露，要进行defensive copy。

1. source（）/target（）方法实现



同样的是互补方法

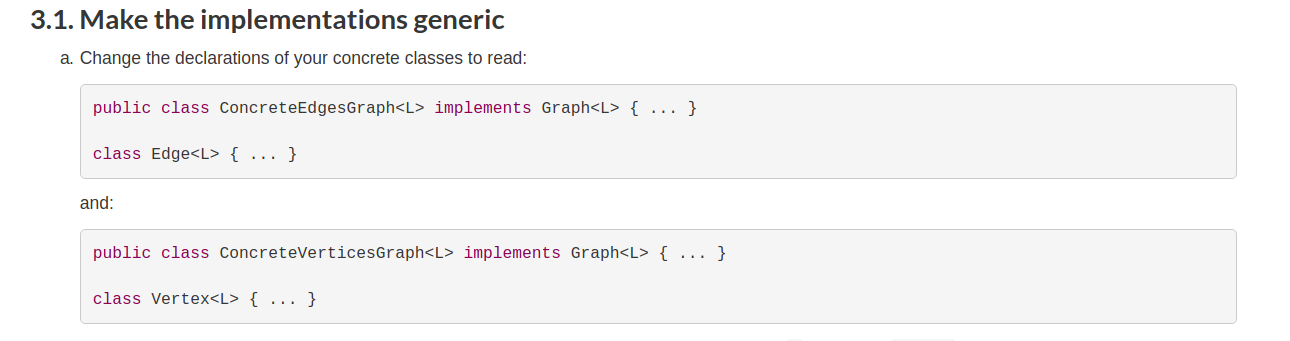
1. toString方法



这里和上面的toString方法思路一样，都是调用Vertex类里面的toString，然后空图和没有边的时候会进行特判，从而特殊输出。

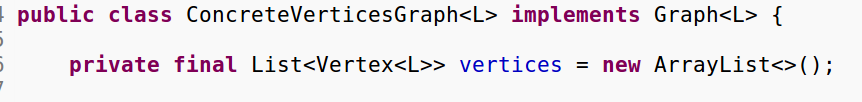
### Problem 3: Implement generic Graph<L>

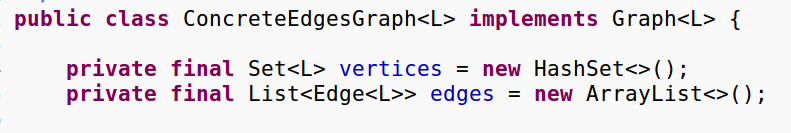
这里要实现泛型，也就是不能只用String来表示一个点的标签，也可以换成别的数据结构，比如Integer 等。具体操作网站上已经给的很详细



#### Make the implementations generic

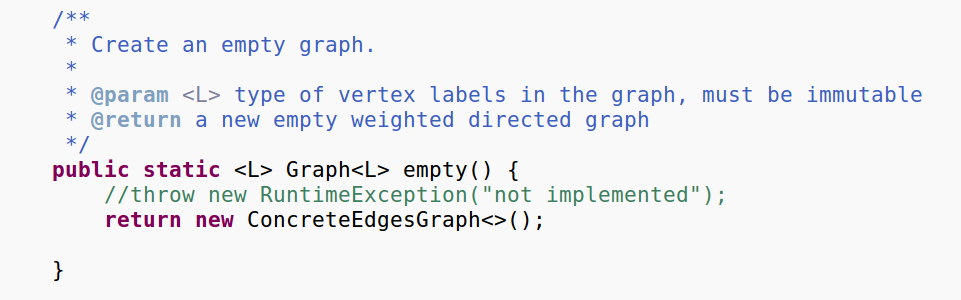
就像MIT网站给的样例一样，只需要将String这个标签换成L就可以实现泛型





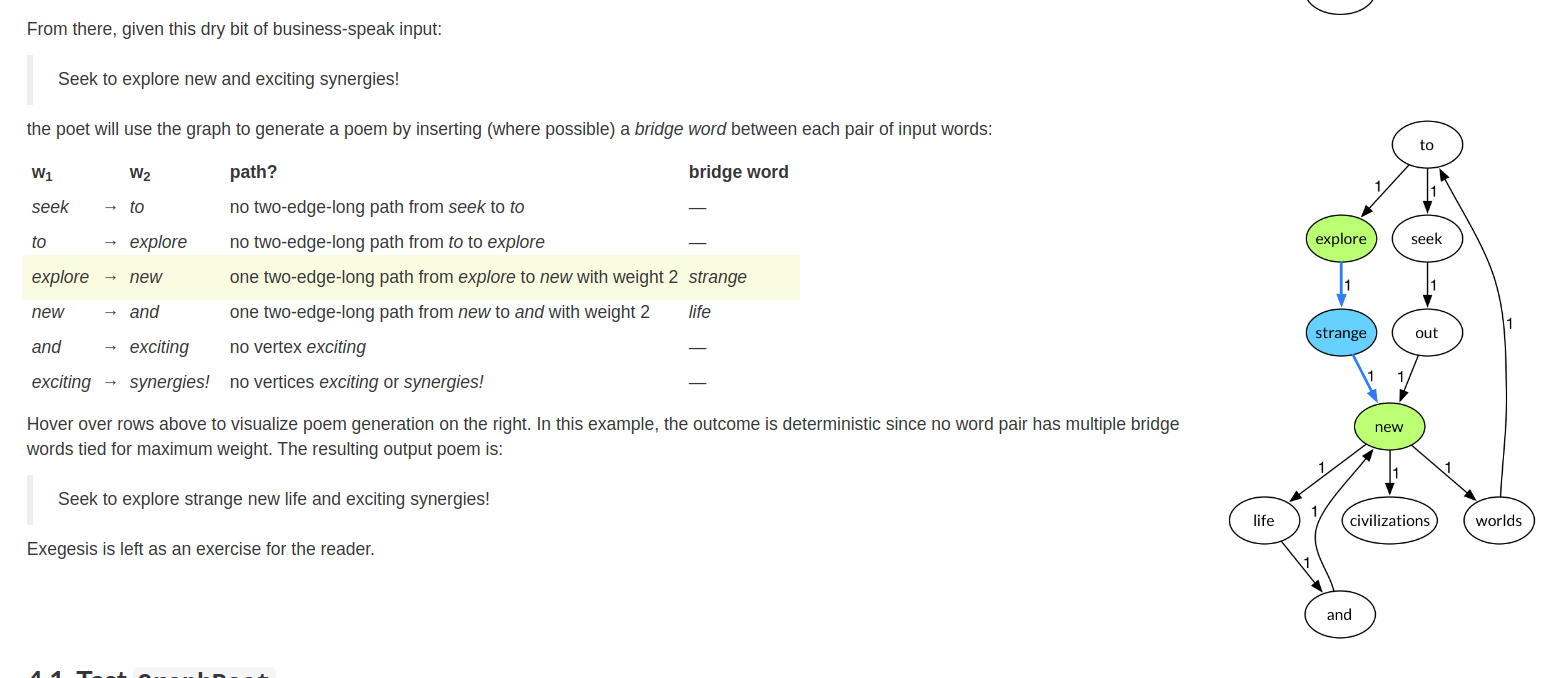
#### Implement Graph.empty()

这里可以利用我们已经写好的类来进行实现，我选择了ConcreteEdgesGraph类



### Problem 4: Poetic walks

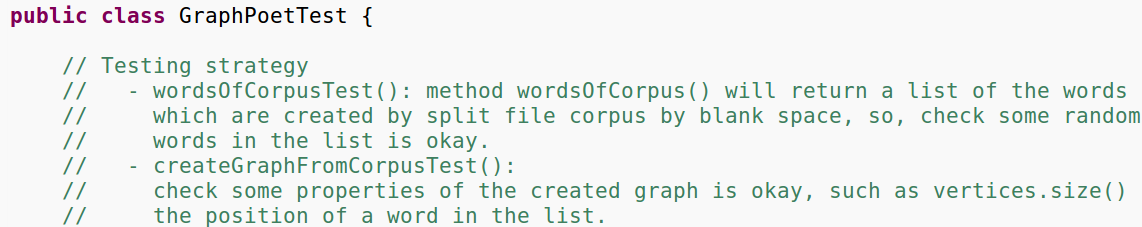
实现创造”优美诗句“，也就是类似于扩句一样的功能，也就是已经有一个建好的图，然后输入一句话或者一段话，然后挨着的两个单词只要是在图中有一条距离为2的单相路，就将中间的那个元素插入到两个单词之间，具体如下图



在explore和new之间有一条长度为2的路，所以就可以在两个单词之间加入strange，seek 和 to 之间就没有这样的路，所以就不用加

#### Test GraphPoet

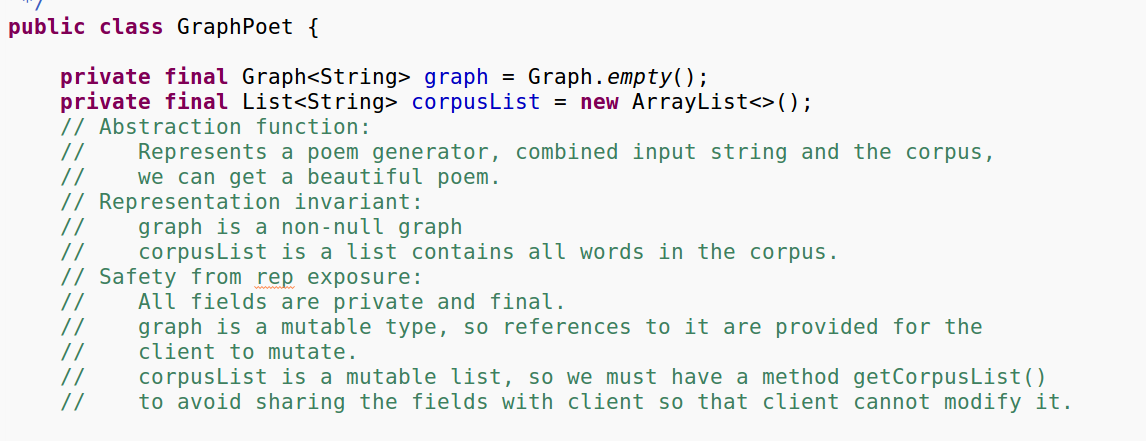
有如下的test strategy



方法实现如下：



#### Implement GraphPoet



有如上的AF、RI、rep，corpusList是用来存已知文本，来进行建图



以上两个方法用于根据已有的文本来进行图的生成。建图的时候注意，如果有两个单词 连续出现多次，那么出现几次，两个单词之间的边权重就为几。例如出现三次，那么权重就为3.

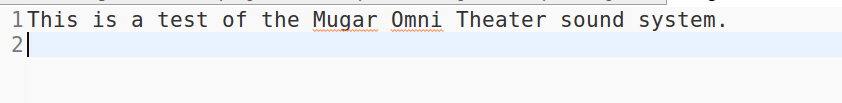


这里用来进行“诗意漫步”，也就是循环比较来进行单词的扩充

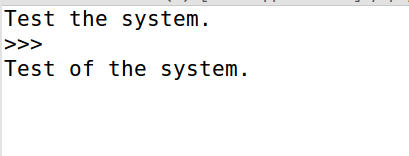
#### Graph poetry slam

（1）

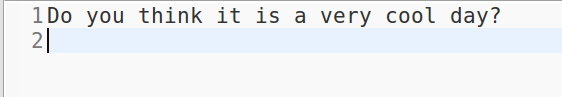
corpus：



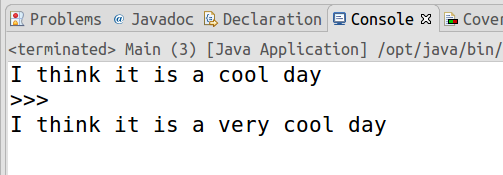
输入及输出



1. corpus：

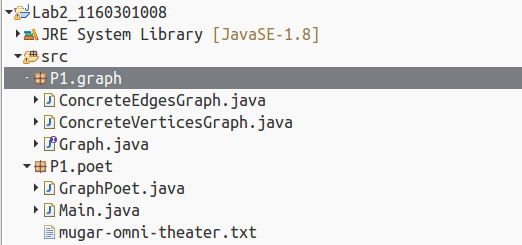


输入及输出



### Before you’re done

请按照<http://web.mit.edu/6.031/www/sp17/psets/ps2/#before_youre_done>的说明，检查你的程序。

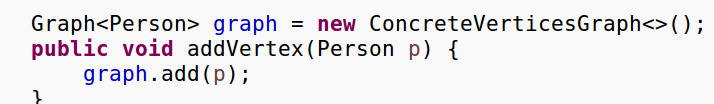


## Re-implement the Social Network in Lab1

实验的这一部分主要是想要利用我们在Poetic Walk中实现的图来进行建图，重写实验一中Social Network部分

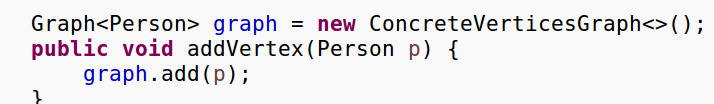
### FriendshipGraph类

首先，在FriendGraph类中有一个利用P1中曾经写好的ConcreteVerticesGraph类生成的图，如下图



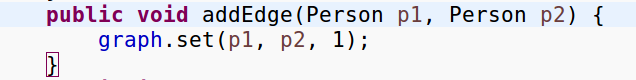
利用泛型，可以将Graph中的元素类型换成Person类。

1. addVertex方法：



由于graph中有已经实现好的add方法，直接调用即可。

1. addEdge方法：



同样可以利用实现好的Graph类中的方法，直接调用graph.set（）即可。

1. getDistance方法：

和lab1中的思路一样，采取广度优先搜索即可搜索到最短路径，如果搜到最后还是搜不到最短路径，那么返回 -1，只不过遍历的时候更多的借助于Graph中实现过的方法，比如说add方法，可以说P1中的图由于内置方法设计的很好，在这里实现一些功能十分方便，也体现了ADT的便捷。只不过这里和lab1中不一样的一点是，这里的Person类是immutable的，所以在图中我新添加了两个map，一个来进行存储flag看是否被访问，另一个map来进行存储每个点的dist性质。



### Person类

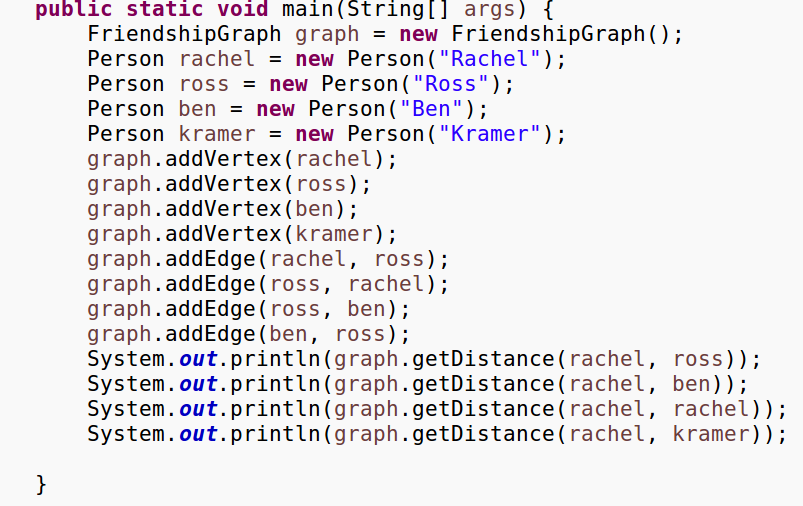


只存储一个名字即可，实现类似于P1中的String功能即可。

### 客户端main()

这里采用了和Lab1中一样的测试用例，

代码如下：



运行结果如下：

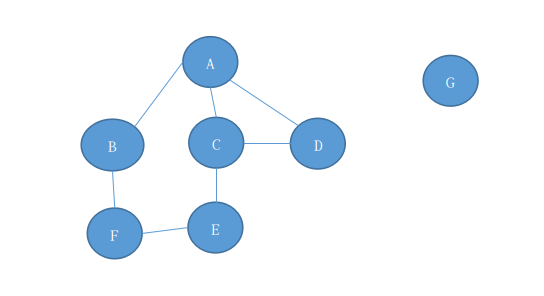


就是新建几个点，然后判断距离即可

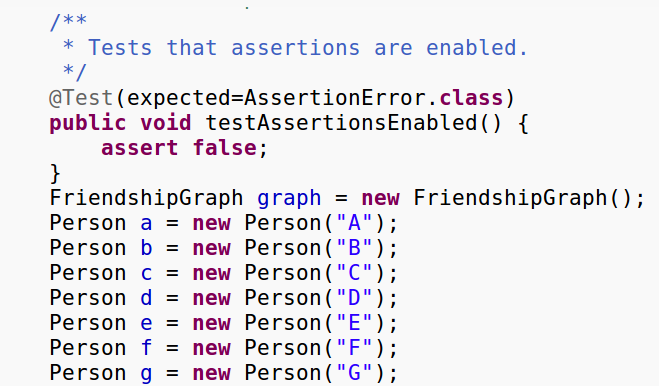
### 测试用例

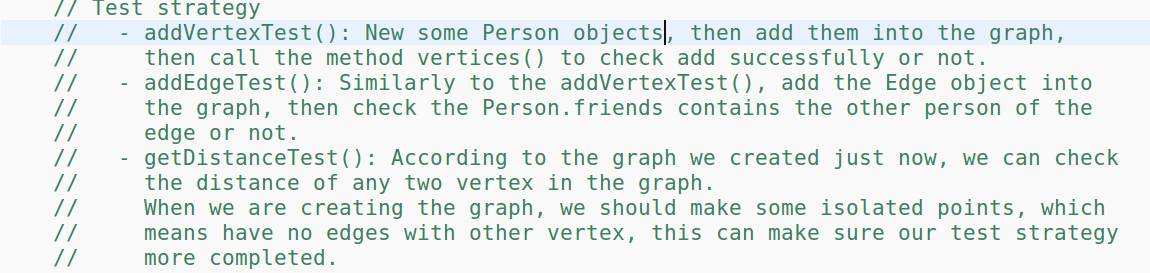
**测试思路：**

初始化一些人，并且做出一个简单的图，其中有孤立于其他的点，来检测不连通的情况。图如下



初始化代码如下：





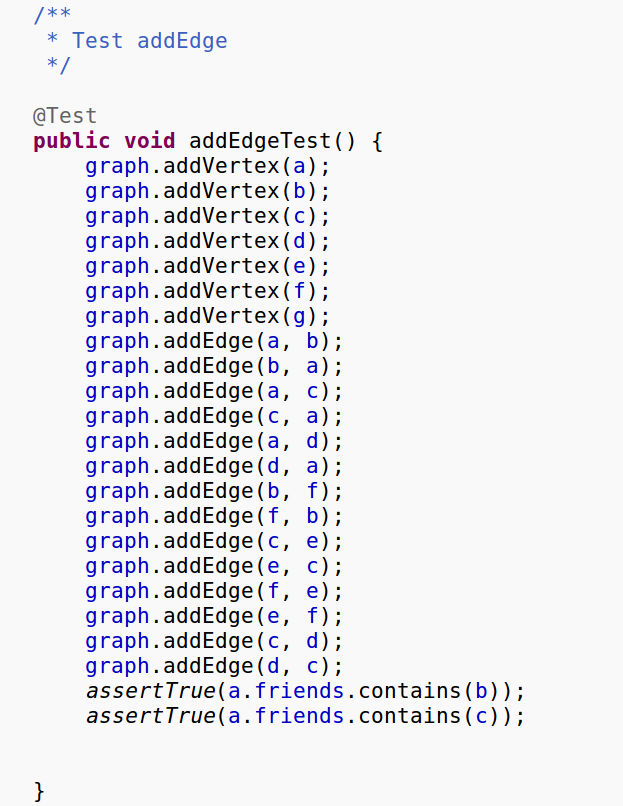
有如上的test strategy

代码如下：

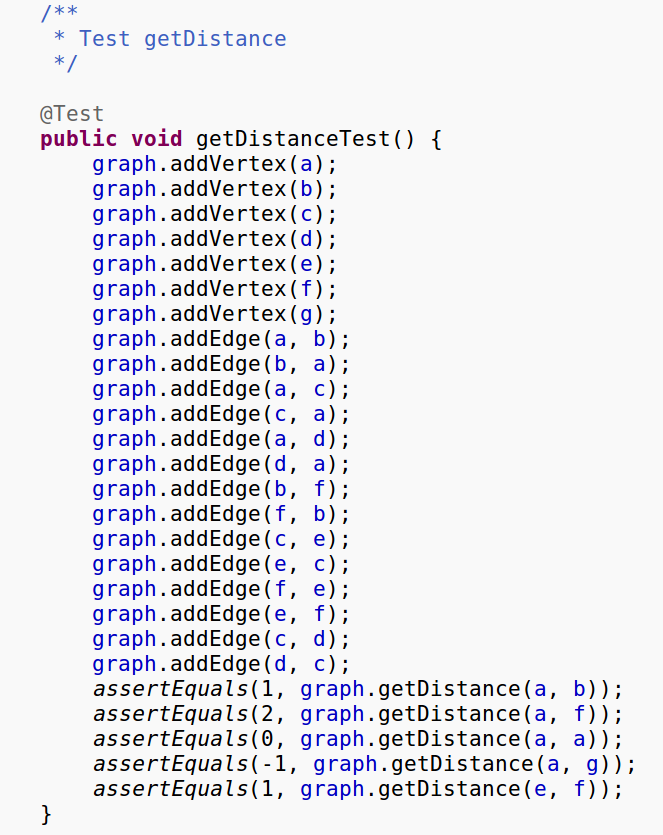
（1）测试addVertex方法：



1. 测试addEdge方法：

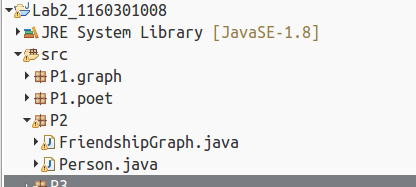


1. getDistance方法测试：



### 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。

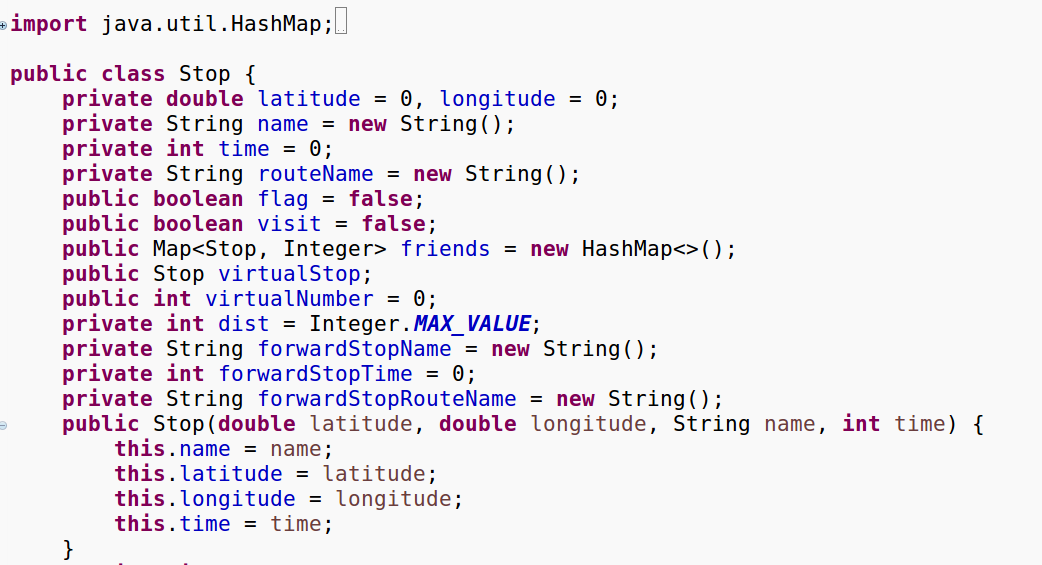


## The Transit Route Planner（选做，额外给分）

# P3要求根据输入文本建立一个交通网络图，然后输入起始站和终点站，从而规划出一条最短路。由于每条边带有权重，所以不能使用简单的BFS，而是使用Dikjstra算法来进行最短路的求取。

# 3.3.1 Stop 设计

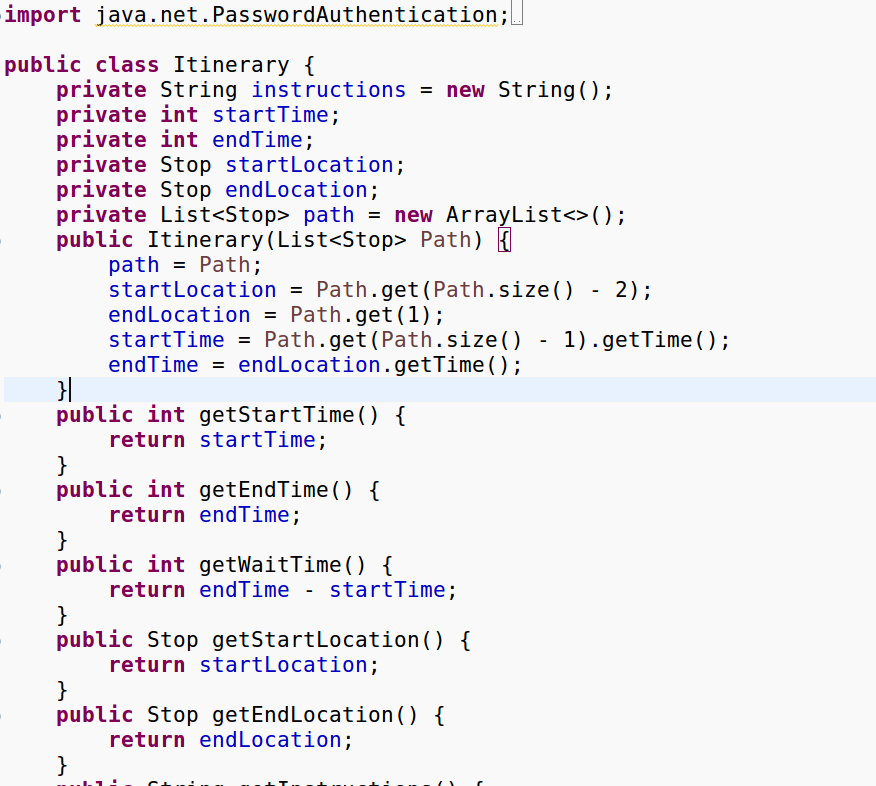
每个Stop要存储很多信息，在这里我把Stop类直接设置为交通图中的点，所以，和大多数人不一样的一点就是我在Stop中存储了时间，也就是如果一个站有多条线路经过的话，就建立不同的Stop用以区分，区分标志就是RouteName和Time，也就是说，同一个车站可能会有很多的点在图中呈现。设计如下：

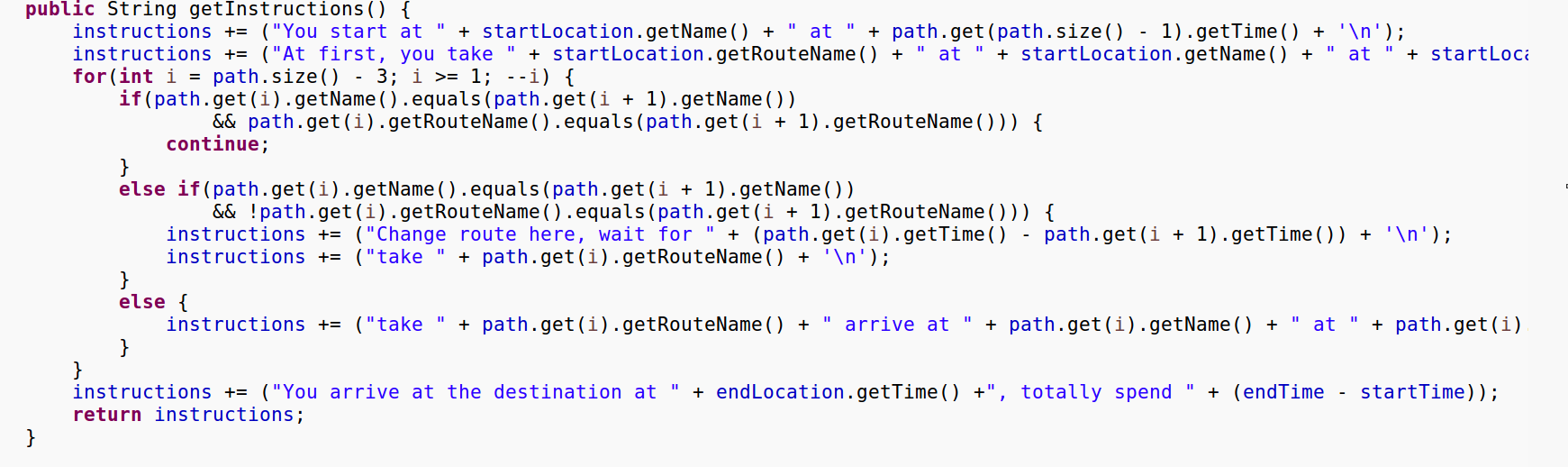


flag表示是否存在虚点，这个后续会介绍，便于Dijkstra搜索最短路。visit用于判断在Dijkstra中当前节点是否已经搜索出最短路。

# 3.3.2 Itinerary类设计

知名见意，这个类就是要记录我们最后规划出来的行程，所以设计如下：

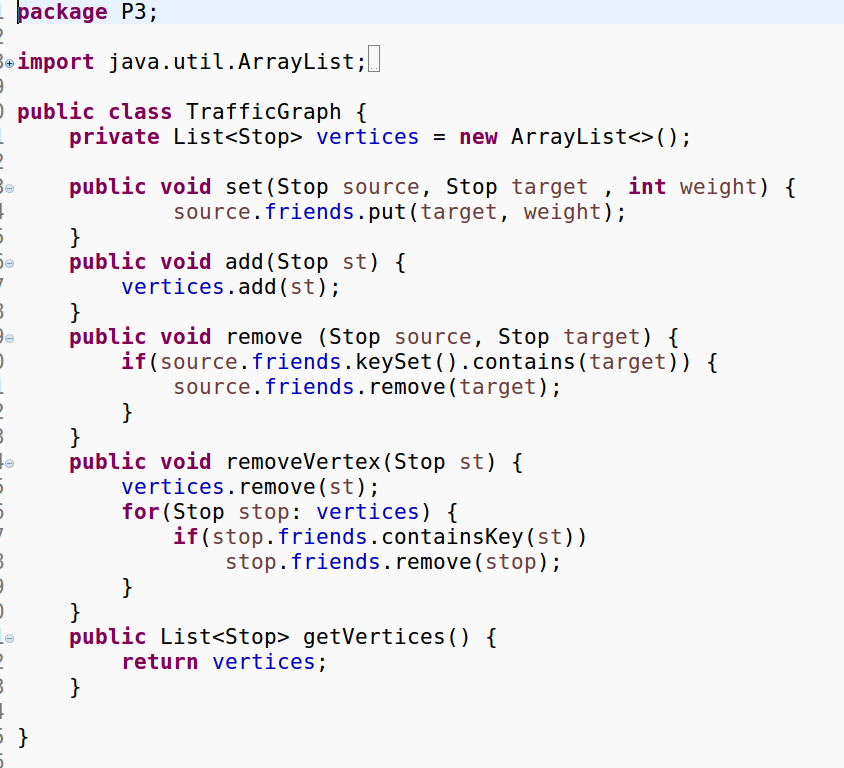




其中，我在构造函数中传进去了一个List<Stop> Path，也就是在RoutePlanner中计算出来的最短路。

# 3.3.3 TrafficGraph类设计

我重新实现了图，代码如下：



针对于我的Stop类中的数据结构来进行了优化和重写，并且优化了set函数，因为最后的图中涉及权重为0的边，为了防止换乘多次。

**3.3.4 RoutePlannerBuilder 设计**

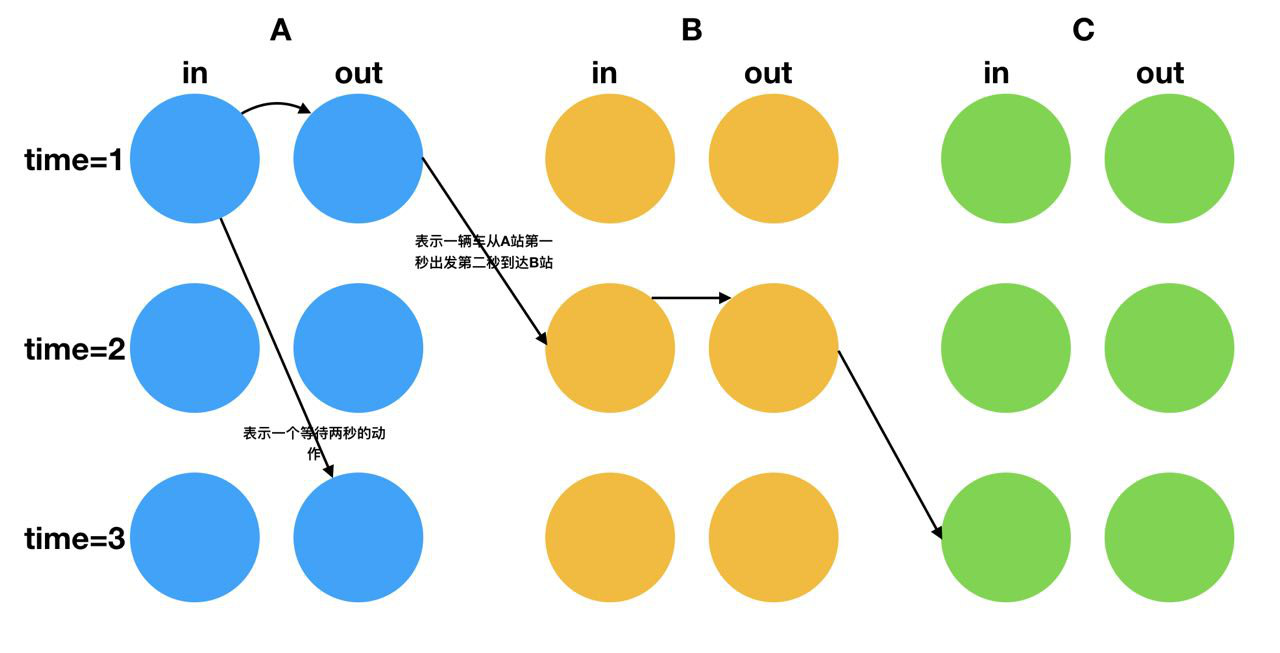
在这个类中，实现的功能就是从文件中读取数据进来，并保存在一个map中，我新建了一个Route类，作为map的key，然后value值是一个存着这条路线上所有stop的list，作为参数传进RoutePlanner对象中，便于建图。



**3.3.5 RoutePlanner 设计**

在这里首先实现了建图，也就是createGraph方法，这里建图的时候，如果想用Dijkstra来进行搜索最短路，则一定要涉及拆点，拆点的思路与如下：

如果要涉及换乘，那么将两条route上的该站都拆成两个点，其中一个是虚点，所有指向该点的路还指向该点，该点和他的虚点之间有一条权重为0的路，所有以该点为起点的路都从虚点来进行指出，具体思路如下图：



由于Dijkstra是单源最短路径，而起点和终点可能会涉及多个点在图中存在，所以这样比较麻烦，所以再新建两个虚点表示起点和终点，这样就实现了单源最短路径。

建图的时候最难的就是拆点，尤其是换乘站的拆点，代码设计如下：



# 实验进度记录

请尽可能详细的记录你的进度情况。

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 计划任务 | 实际完成情况 |
| 3.20-3.23 | 空闲时间大概五小时 | 完成P1 | 没有完成 |
| 3.24 | 空闲时间大概五小时 | 完成P1 | 完成 |
| 3.25 | 两小时 | 完成P2 | 完成 |
| 3.28-4.2 | 大概八小时 | 完成P3 | 完成 |

# 实验过程中遇到的困难与解决途径

其实P1和P2相对好写, 只是P3比较难, 当然, 这可能也是他作为附加实验的原因, 但其实想通了也没觉得有很难, 其实难就难在拆点建图的思想, 如果没有奥赛经验的话, 很难想到这一点, 当然, 我也是通过室友讲解, 才学会了这个拆点建图的方法, 因为一个Stop可能有多条路线经过, 而且还有可能有多个班次, 这样如果不拆点, 就造成了多重边, 而Dijkstra不能处理多重边情况, 所以一定要拆点, 其次就是代码实现的时候, 拆点的时候真的很容易就写出bug, 我用了大概三天去写代码加上debug, 还是有难度的.

# 实验过程中收获的经验、教训、感想

节除了总结你在实验过程中收获的经验和教训，也可就以下方面谈谈你的感受（非必须）：

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？

ADT可以复用, 直接面向场景一次性, 太特殊, 只针对于当前情况, ADT可以针对不同相似场景来进行一些小的改动.

1. 使用泛型和不使用泛型的编程，对你来说有何差异？

更熟悉不使用泛型的编程, 因为之前都没有使用泛型编程, 所以这次有一点不适应

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？

不适应

1. 本实验设计的ADT在三个应用场景下使用，这种复用带来什么好处？

复用时难度降低, 不用从零实现, 可以减小很多工作量

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？

在工程上可以有效的保护我们的代码, 避免黑客等恶意破坏, 这在将来的应用中十分重要. 愿意

1. 关于本实验的工作量、难度、deadline。

适中.