

**2018年春季学期  
计算机学院大二软件构造课程**

**Lab 3实验报告**

|  |  |
| --- | --- |
| 姓名 | 穆添愉 |
| 学号 | 1160301008 |
| 班号 | 1603010 |
| 电子邮件 | [1417553133@qq.com](mailto:1417553133@qq.com) |
| 手机号码 | 15636094072 |

**目录**

[1 实验目标概述 1](#_Toc513414438)

[2 实验环境配置 1](#_Toc513414439)

[3 实验过程 1](#_Toc513414440)

[3.1 待开发的四个应用场景 1](#_Toc513414441)

[3.2 基于语法的图数据输入 5](#_Toc513414442)

[3.3 面向复用的设计：Graph<L, E> 11](#_Toc513414443)

[3.4 面向复用的设计：Vertex 13](#_Toc513414444)

[3.5 面向复用的设计：Edge 16](#_Toc513414445)

[3.6 可复用API设计 21](#_Toc513414446)

[3.7 的可视化：第三方API的复用（选做） 25](#_Toc513414447)

[3.8 设计模式应用 30](#_Toc513414448)

[3.8.1 使用State/Memento模式进行Vertex的状态管理（选做） 30](#_Toc513414449)

[3.8.2 使用factory method模式构造Vertex对象 30](#_Toc513414450)

[3.8.3 使用factory method模式构造Edge对象 31](#_Toc513414451)

[3.8.4 使用abstract factory或builder模式构造Graph对象 32](#_Toc513414452)

[3.8.5 使用Strategy模式调用centrality度量算法 33](#_Toc513414453)

[3.8.6 使用Composite模式设计超边对象（选做） 34](#_Toc513414454)

[3.8.7 使用decorator模式构造不同特征的Edge对象（选做） 34](#_Toc513414455)

[3.8.8 使用其他设计模式（选做） 35](#_Toc513414456)

[3.9 图操作指令的输入和处理（选做） 35](#_Toc513414457)

[3.10 应用设计与开发 36](#_Toc513414458)

[3.10.1 单词网络GraphPoet 40](#_Toc513414459)

[3.10.2 微博社交网络SocialNetwork 41](#_Toc513414460)

[3.10.3 网络拓扑图NetworkTopology 42](#_Toc513414461)

[3.10.4 电影网络MovieGraph 42](#_Toc513414462)

[3.11 应对四个应用面临的新变化（任选两个） 43](#_Toc513414463)

[3.11.1 单词网络GraphPoet 43](#_Toc513414464)

[3.11.2 微博社交网络SocialNetwork 43](#_Toc513414465)

[3.11.3 网络拓扑图NetworkTopology 44](#_Toc513414466)

[3.11.4 电影网络MovieGraph 44](#_Toc513414467)

[4 实验进度记录 44](#_Toc513414468)

[5 实验过程中遇到的困难与解决途径 45](#_Toc513414469)

[6 实验过程中收获的经验、教训、感想 45](#_Toc513414470)

# 实验目标概述

编写具有可复用性和可维护性的软件, 主要使用以下软件构造技术:

1. 子类型、泛型、多态、重写、重载
2. 继承、代理、组合
3. 常见的 OO 设计模式
4. 语法驱动的编程、正则表达式
5. 基于状态的编程
6. API 设计

# 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。

在这里给出你的GitHub Lab3仓库的URL地址（Lab3-学号）。

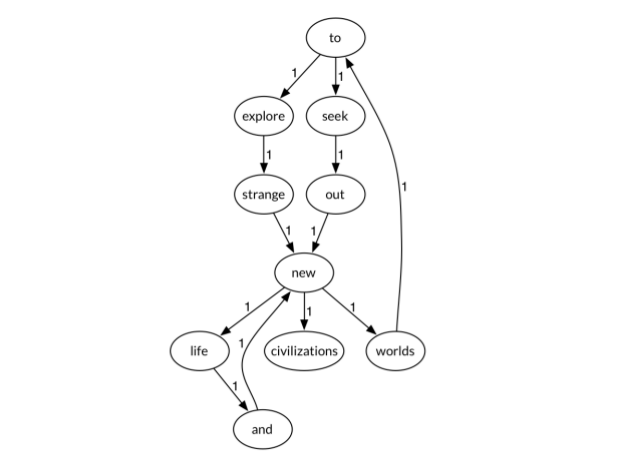
# 实验过程

## 待开发的四个应用场景

1. 单词网络(GraphPoet)

节点为”单词”, 边为两个单词在文本中的相邻关系, 边的权重是相邻出现的次数(值域为正整数)。属性为label。

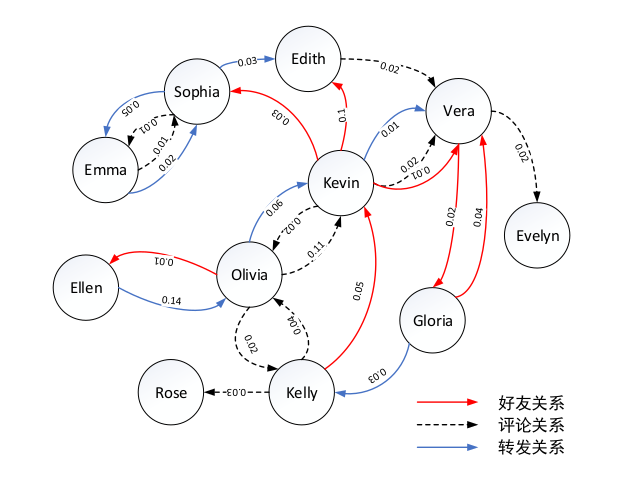
具体如下图所示：



1. 微博社交网络（SocialNetwork）

节点为“用户”，label为姓名，属性包含年龄、性别，边为两个用户的社交关系。两个用户之间最多可存在 3 种类型的社交关系边,分别表征“好友关系”(A 关注了 B,边的方向为 A->B)、评论关系(A 曾评论过 B 的微博,边的方向为 A->B)、转发关系(A 曾转发过 B 的微博,边的方向为 A->B)。图的总权重为1，所以每条边的权重范围是（0,1]，图中不能出现loop，即一个人与自己不能是好友/评论/转发关系。

具体如下图：



1. 网络拓扑图（Network Topology）

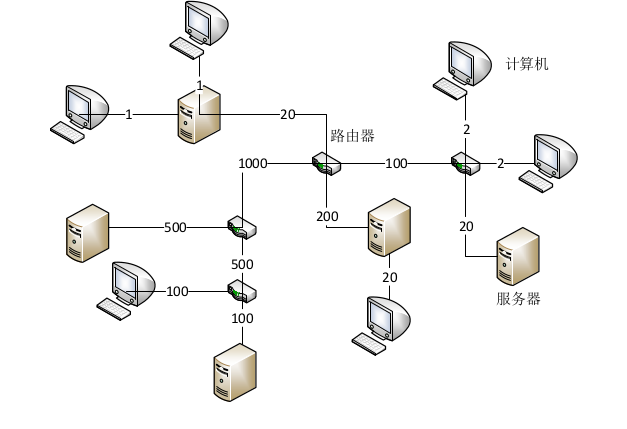
节点为“计算机”、“服务器”、“路由器”

label 为主机名,属性包括:IP 地址, 边为它们之间的网络连接关系

但计算机之间不能直接相连、服务器之间不能直接相连 ,权重为网络

连接的带宽(例如 1、20、100)。图中不能出现 loop,即一条边的起点和终点不能为同一个节点。

具体如下图：



1. 电影网络（MovieGraph）

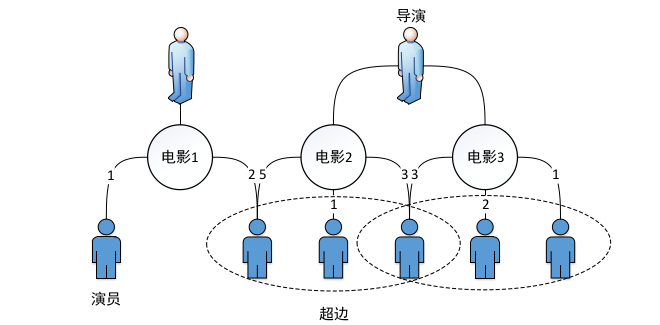
节点为“电影”(label 为电影名,还有三个属性:上映年份、拍摄国家、IMDb 评分)、“演员”(label 为姓名,属性:年龄、性别)。

“导演”(label 为姓名,属性:年龄、性别)。

有两种无向边:演员 A 参演了电影 M、导演 D 执导了电影 M。

演员和电影之间的边有权值(表示 A 在 M 中的角色次序,用正整数表示),导演和电影之间的边无权值。图中不能出现 loop,即一条边的起点和终点不能为同一个节点。参演过同一部电影的所有演员形成一条超边,超边无权值。

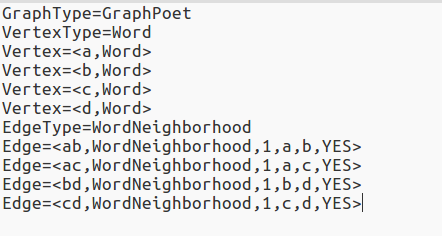
具体如下：



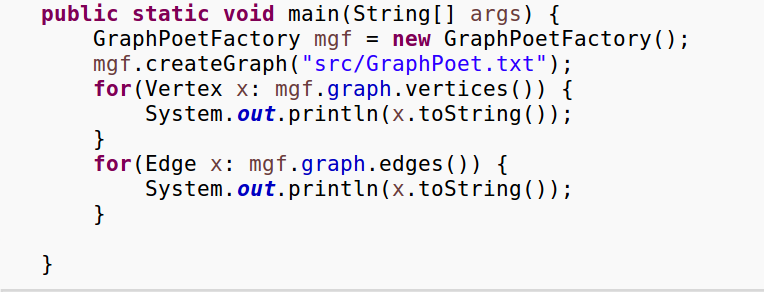
## 基于语法的图数据输入

正则表达式部分见后面的factory部分，来进行正则表达式的分析。

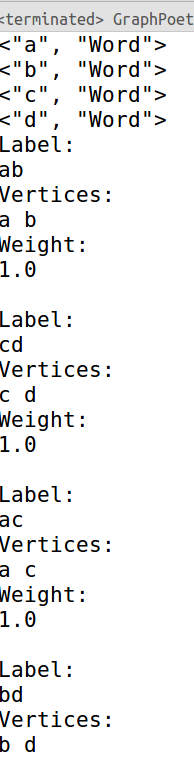
1. 对于GraphPoet  
   这个图的输入比较简单，根据样例，就可以写出一个类似的输入文件如下：



为了检测我的factory的正确性，我在factory中写了main方法来进行检测，main方法代码如下：

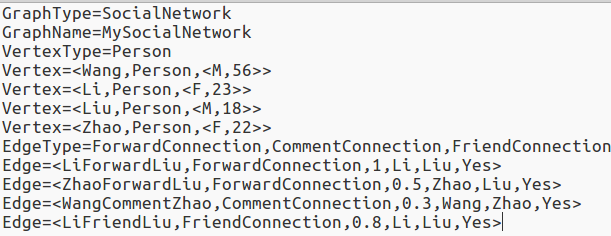


输出结果如下：

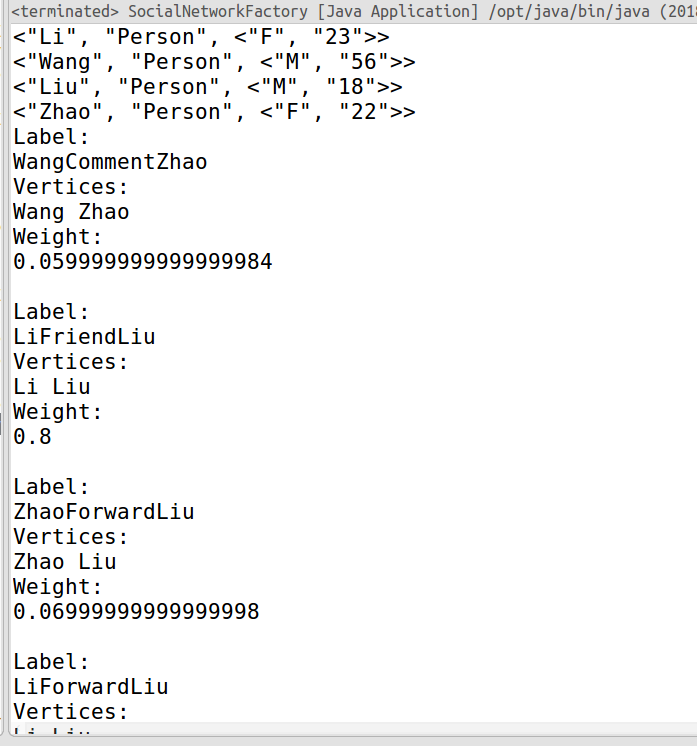


1. 微博社交网络

同样是根据样例写文件，文件如下：



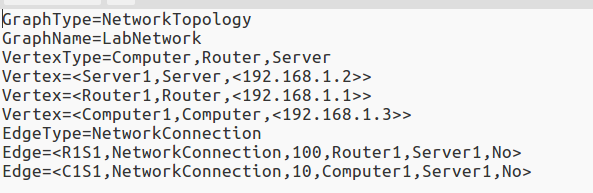
同样是调用main方法， 然后打印出图中的所有点和边 输出结果如下：



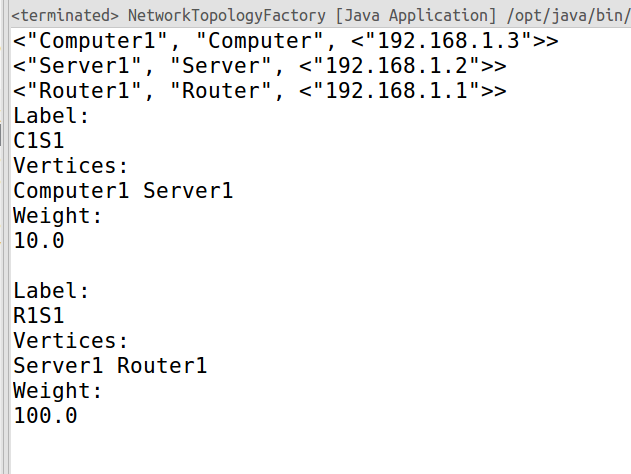
需要注意的是，对于SocialNetwork，加入一条边之后，就要进行权重的修改，由于是double类型的除法，会出现很多位小数，这样在测试的时候判断是否相等会为我们带来麻烦，所以需要有一个误差范围比如说0.001之类

1. 网络拓扑图

同样是根据样例写文件，文件如下：

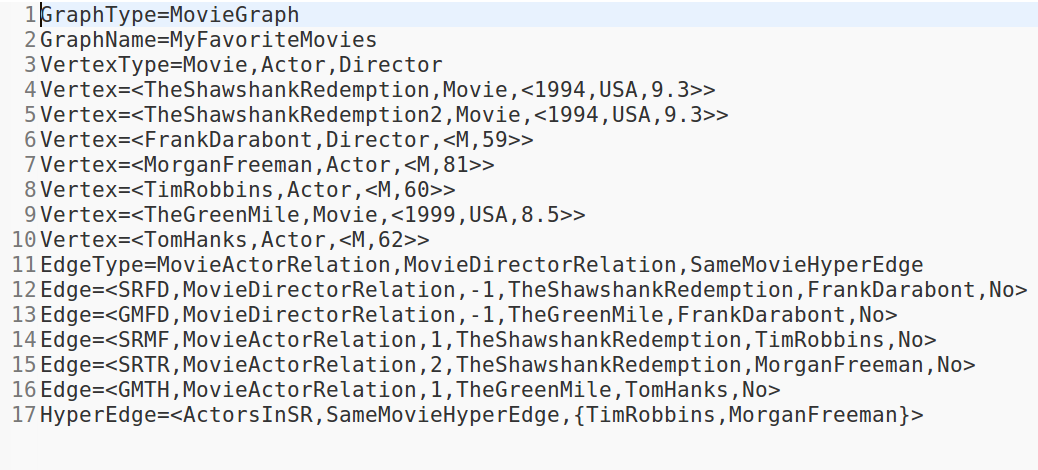


输出结果如下：



1. 电影网络图

输入文件如下：

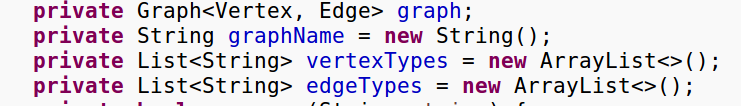


输出结果如下：

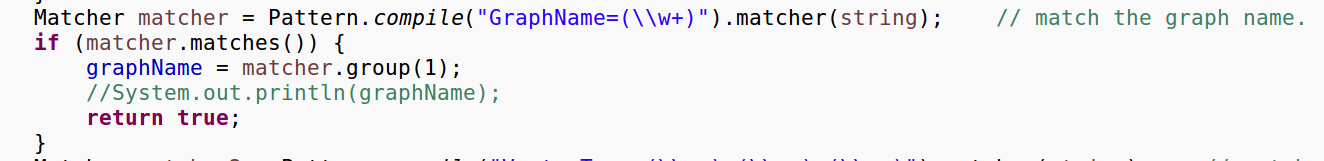


至此，四个图的正则解析结果基本是正确的。下面进行对其中的正则表达式的解析

首先，有以下成员变量

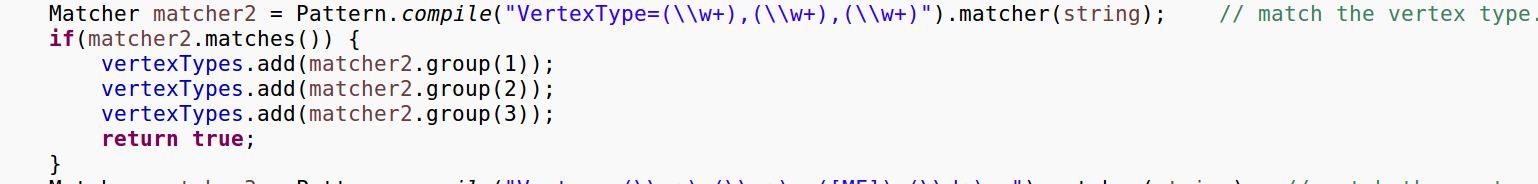


1. 对于GraphName的解析



（\\w+)用来匹配一串数字或字母， 在这里可以直接将匹配到的字符串赋给成员变量GraphName。

1. 对于VertexType的解析



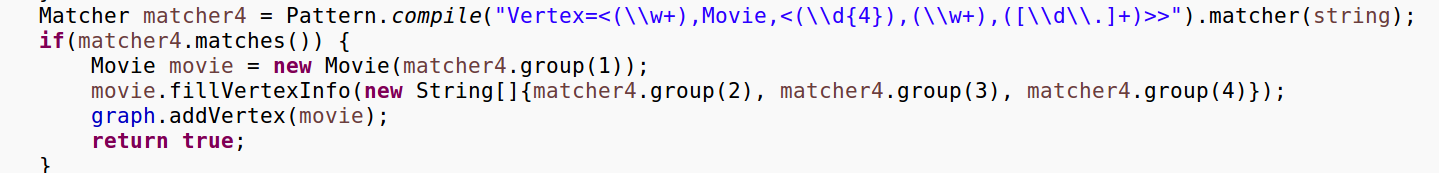
对于每个图，都会将匹配到的VertexType加入到存有vertexTypes的List中，这样便于今后可能会用到。

1. 对于Vertex的匹配



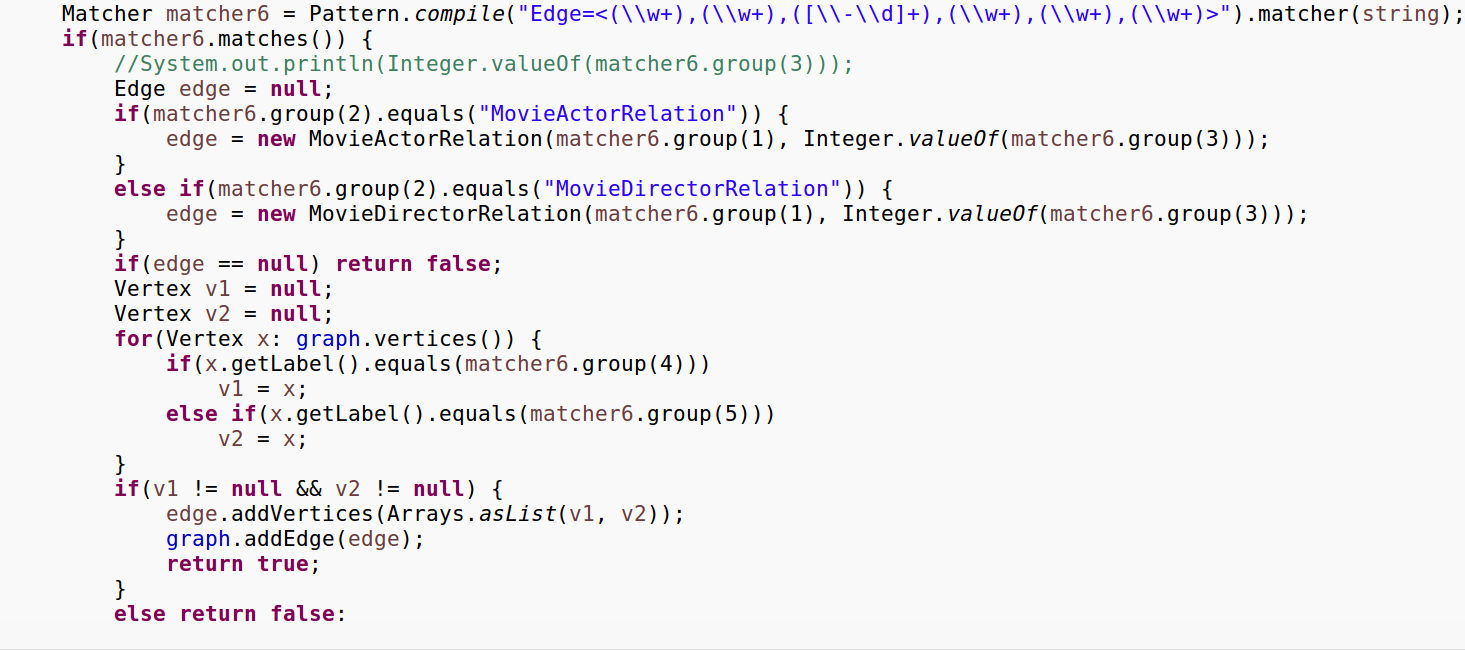
这里使用MovieGraph来进行举例，所以图中匹配的是Actor和Director，(\\d+)用来匹配一串数字，[MF]用来匹配性别，只能是M或者F，然后将匹配到的字符串进行处理（有的可能要转成数字）来进行Vertex节点信息的填写，然后将其加入到图中，这样，就完成了对于节点的匹配。

1. 对于特殊数字的限制



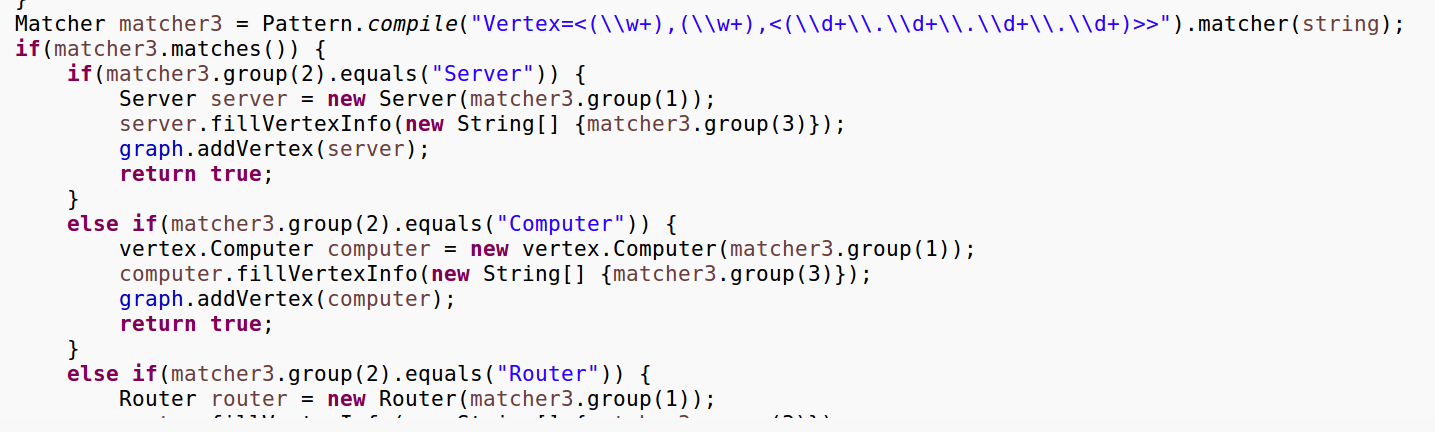
图中，展示的是对于年代的四位数字的限制和对于小数的匹配，如图，（\\d{4})表示只能匹配四位数字，（\\d\\.]表示对于小数的匹配。

1. 对于边的匹配



这里对于边的匹配有一点复杂，不只是匹配到符合要求的字符串就进行信息的填写，首先要进行判断的就是边里的节点是不是已经存在于图中了，如果不存在的话，按照前面的spec，应该直接报错，然后是进行判断edge是否为空，如果不为空，那么就可以成功加入到图中。

1. 对于ip地址匹配



采用上述方式对于形如“192.168.2.1”的ip地址进行匹配

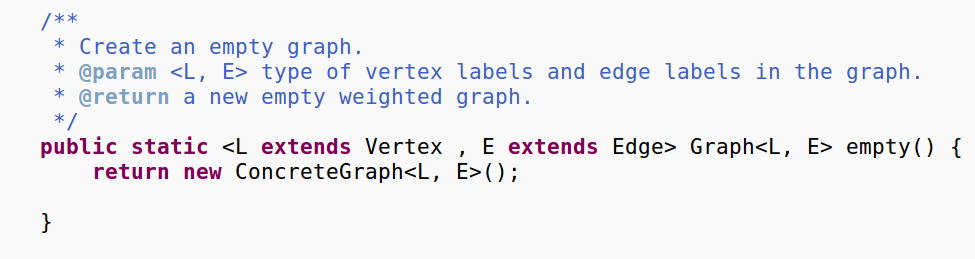
主要的正则表达式就是上述几种。

## 面向复用的设计：Graph<L, E>

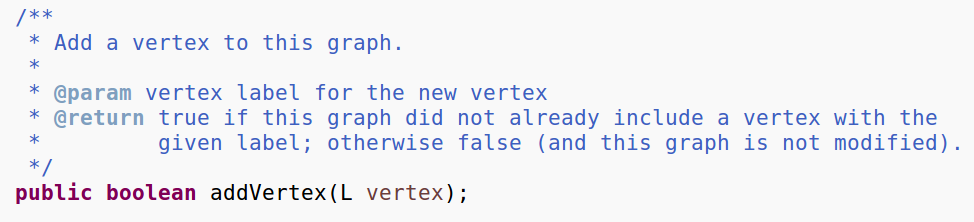
Graph<L, E>中主要是一些抽象出来的每个图都会有的公共方法，

有如下的抽象方法：

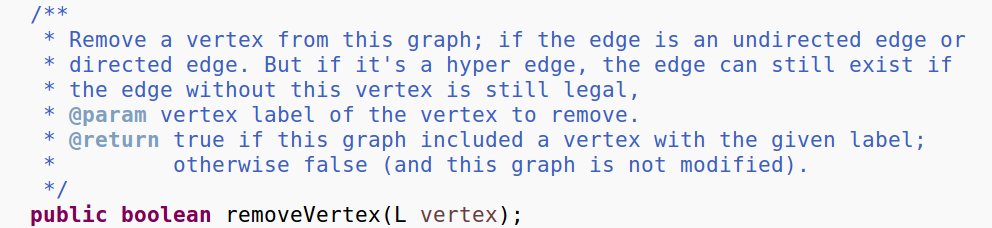
（1）empty（），用于产生一个空图



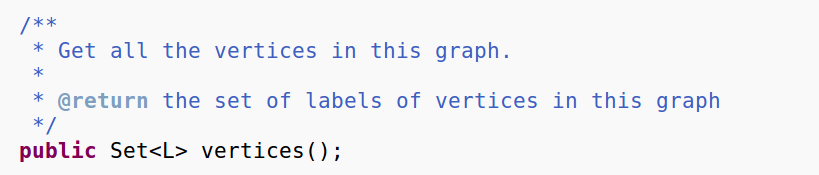
1. addVertex（）方法



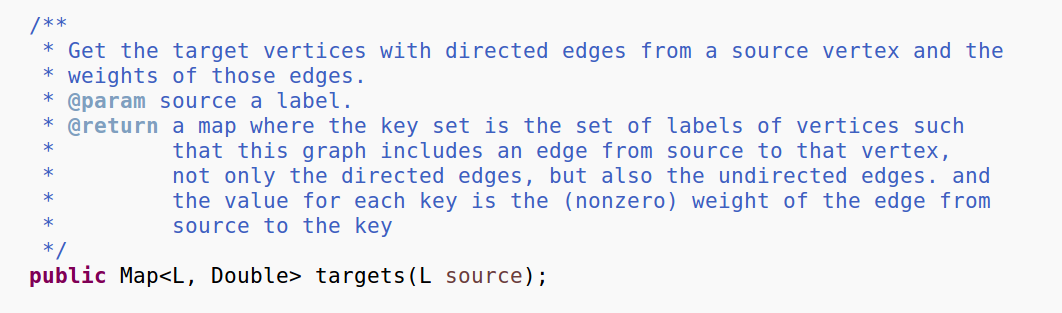
1. removeVertex()方法



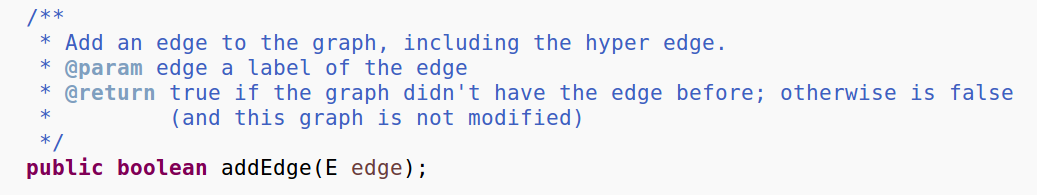
1. vertices（）方法



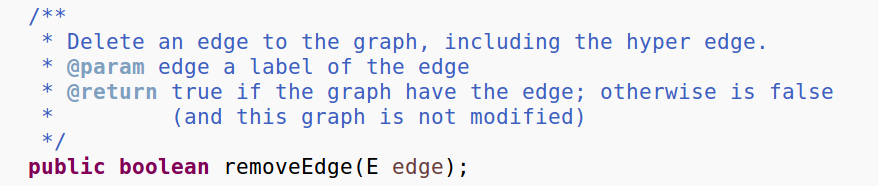
1. targets()方法



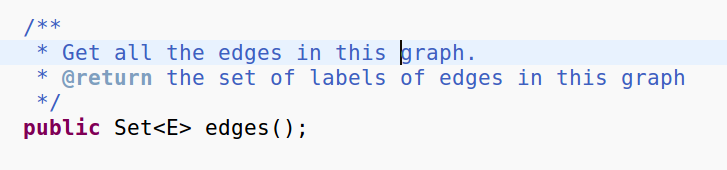
（6）addEdge（）方法



1. removeEdge（）方法



1. edges（）方法



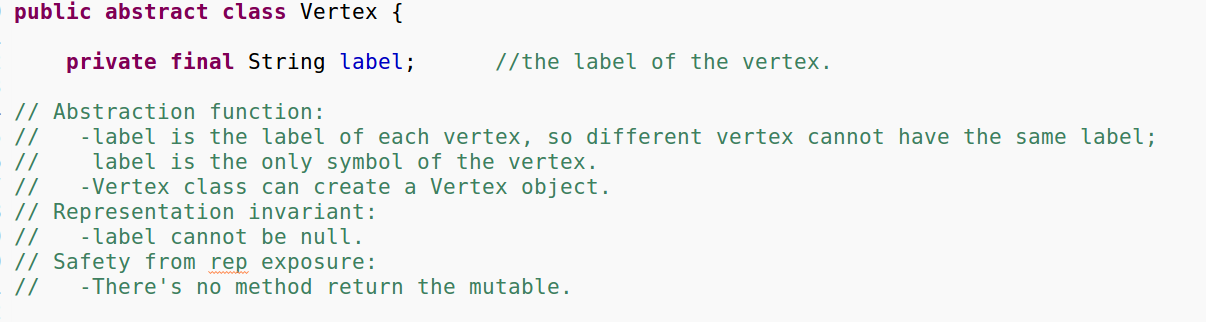
## 面向复用的设计：Vertex

将共有的属性和方法在抽象类定义出来，差异化的属性或方法写在子类中，使用继承和重写来实现。

将String label定义在Vertex抽象类中，每一个继承自该类或继承自该类的子类都具有该属性。将fillVertexInfo、getLabel（公有的）方法定义在Vertex中通过构造函数传入label。

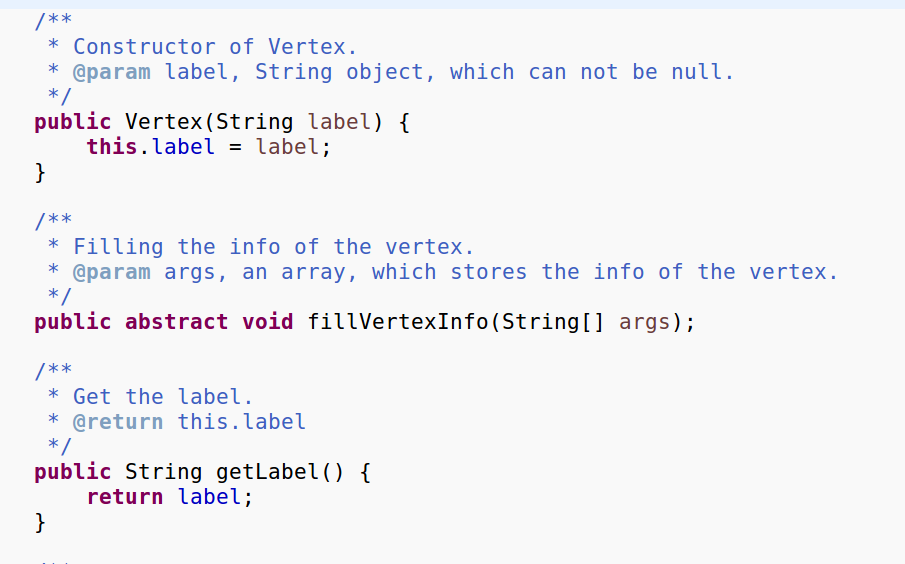
1. Word类直接继承自Vertex，重写toString、equals、hashCode方法，并实现父类中定义的抽象方法，由于Word类没有属性，fillVertexInfo方法的实现体为空。加入一个变量type = “Word”
2. Person类直接继承自Vertex，重写toString、equals、hashCode方法，添加性别、年龄属性，通过fillVertexInfo方法给特有的两个属性赋值。定义type=”Person”
3. NetworkVertex类继承自Vertex类，作为Computer、Server、Router的父类，添加三个类的共有属性— IP地址，并在Network Vertex中实现fillVertexInfo方法，定义type=”NetworkVertex”
4. Computer类继承自NetworkVertex，由于Vertex中的抽象方法在Network Vertex类中已经实现，在Computer类中只需定义type=”Computer”
5. Server类继承自NetworkVertex，只需定义type=”Server”
6. Router类继承自NetworkVertex，只需定义type=”Router”
7. MoviePerson继承自Vertex，作为Actor和Director的父类，定义年龄、性别两个属性，通过fillVertexInfo方法为两个属性赋值。
8. Actor和Directeor都继承自MoviePerson，只需分别定义type=”Actor”和type=”Director”

有如下的AF、RI等



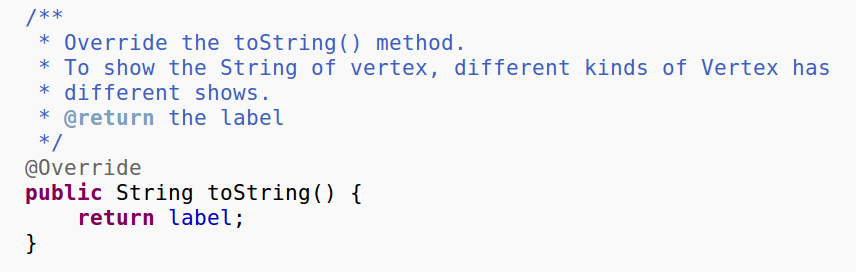
（1）

下图是构造器、填写信息部分、getLabel部分

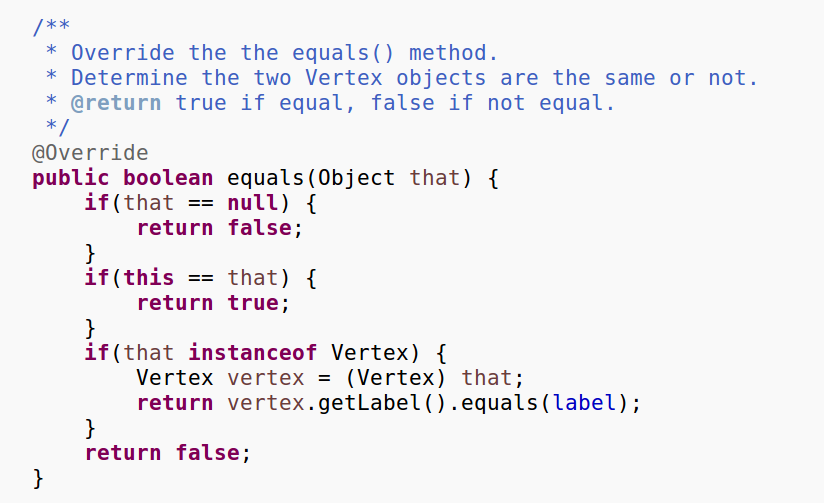


（2）

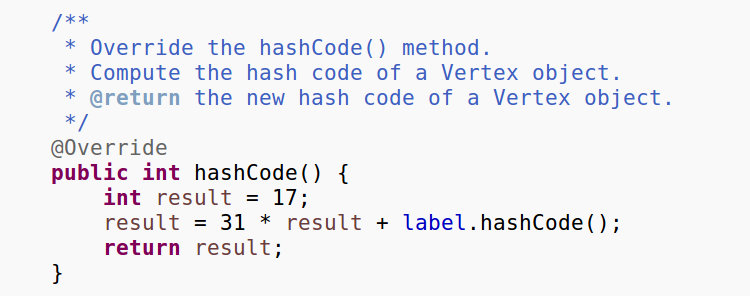
下图是重写的toString（）方法



1. 重写的equals（）方法



1. 重写的hashCode（）方法



利用label来进行hashCode的计算

## 面向复用的设计：Edge

将所有类型的边的共有属性和方法抽象出来，放在Edge抽象类中。

**toString()、equals()、hashCode()在每个子类中都需要重写，下面不再重复**

需要在Edge抽象类中定义的属性有label、weight，通过构造函数来传入label和weight；定义的抽象方法有getType()、addVertices、containVertex()、vertices()、sourceVertices()、targetVertices()。

**DirectedEdge**继承自Edge，作为所有具体有向边类型的父类，调用父类的构造函数。用两个Vertex类型的变量source和target表示边的起点和终点，重写addVertices()方法时，将vertices的第一个元素传给source，第二个元素传给target。SourceVertices（）方法，返回一个只包含起点source的集合，targetVertices()方法类似。

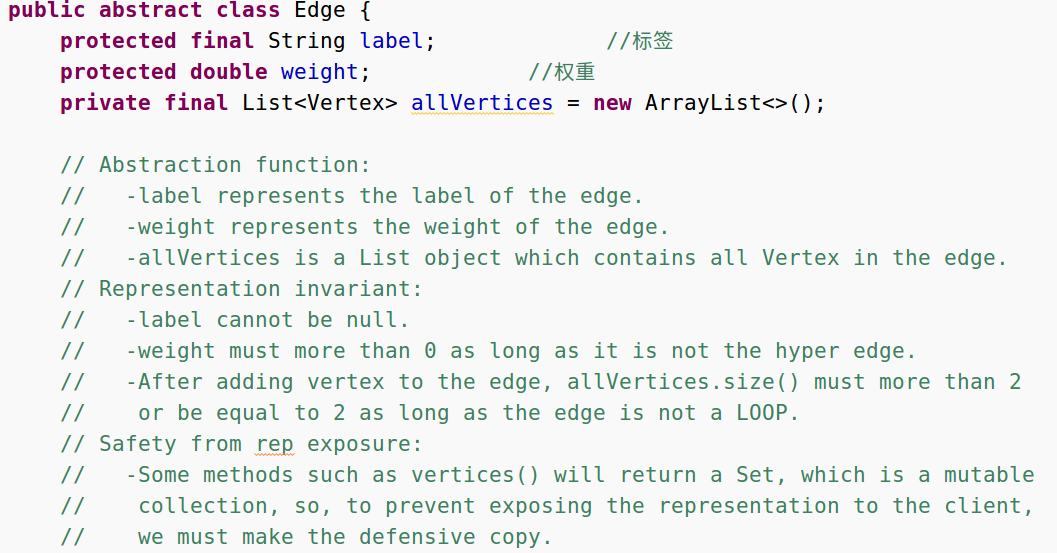
**UndirectedEdge**继承自Edge，作为所有具体无向边类型的父类，调用父类的构造函数。在这里我同样是用了Vertex类型的变量来存储边的端点，不考虑顺序。在实现sourceVertices()方法时，将包含两个端点的集合返回，targetVertices()方法类似。

**HyperEdge**继承自Vertex，作为具体的超边类型的父类。用一个Vertex类型的集合Set<Vertex>来存储超边上的点。重写addVertices()方法时，需要判断传入的列表中的元素是否已在边上，如果不在，将列表中的顶点加入集合，返回true；否则添加顶点并返回false。

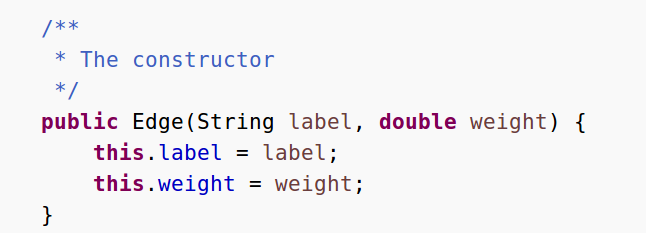
对于sourceVertices()和targetVertices()方法，对后续的实验并没有作用，在这里都是返回的该边上的Vertices的集合。

**WordEdge**为继承自DirectedEdge，为有向边。由于定义在Edge中的抽象方法已经在父类中得到实现并且不用改进。为WordEdge增加一个type=”WordEdge”，构造函数调用父类的。重写toString()、hashCode()、equals()方法时，为了简化可以先调用父类中已经写好的，然后将子类的特征添加进去。

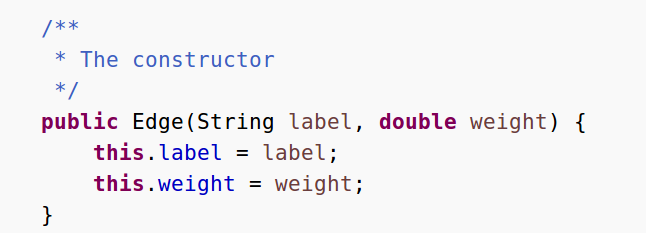
有如下的 AF、RI等



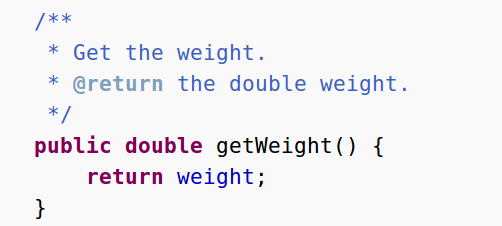
1. 构造器



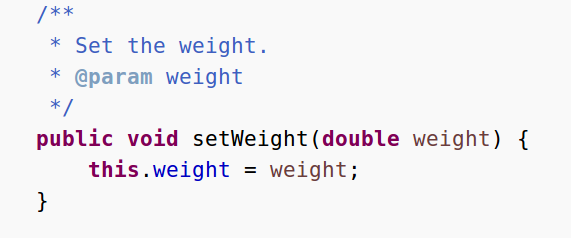
1. getLabel（）方法



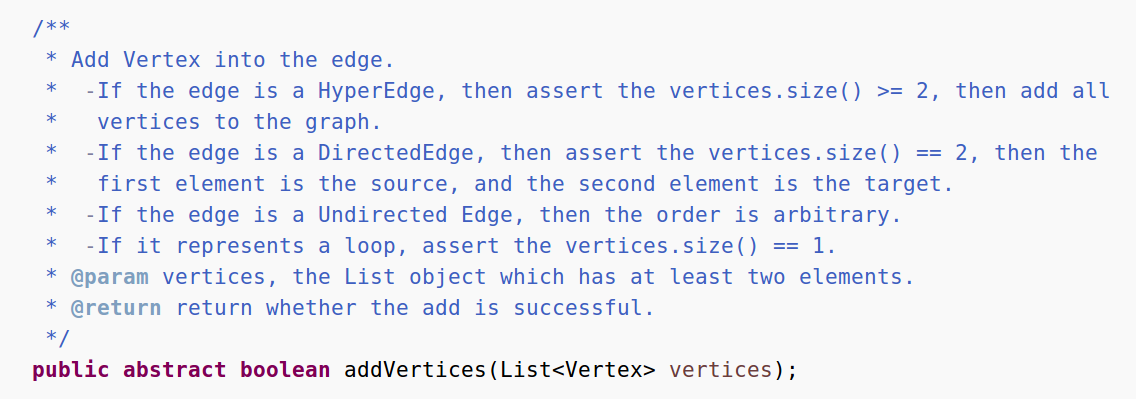
1. getWeight（）方法



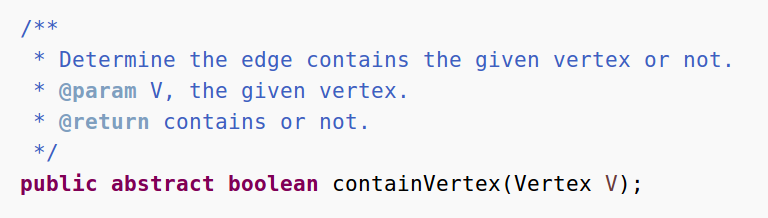
1. setWeight（）方法



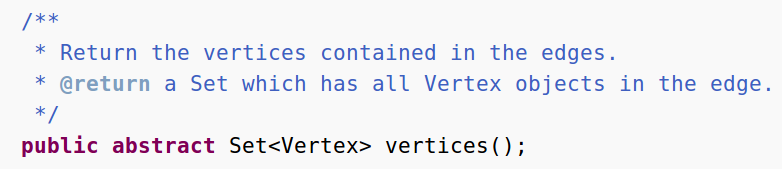
1. addVertices（）方法



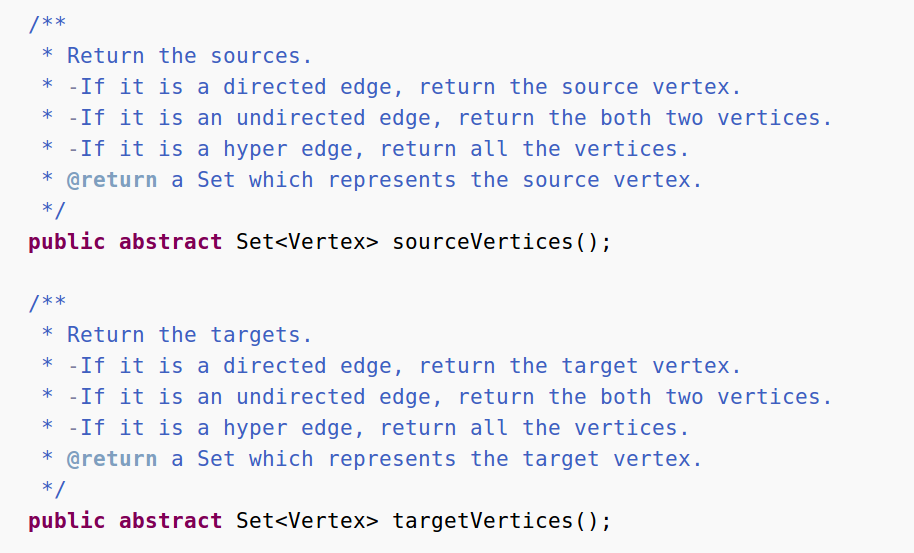
1. containsVertex（）方法



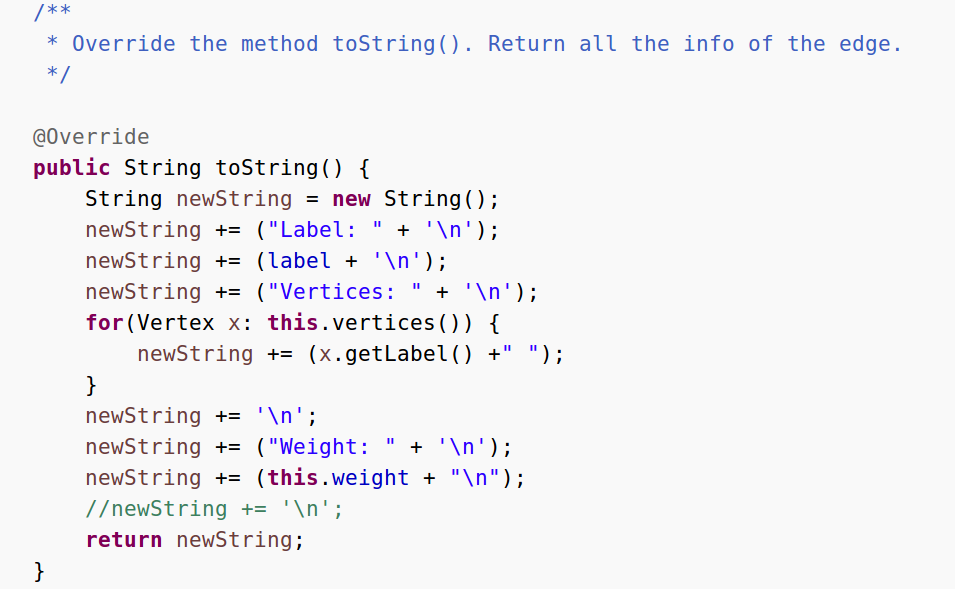
1. vertices（）方法



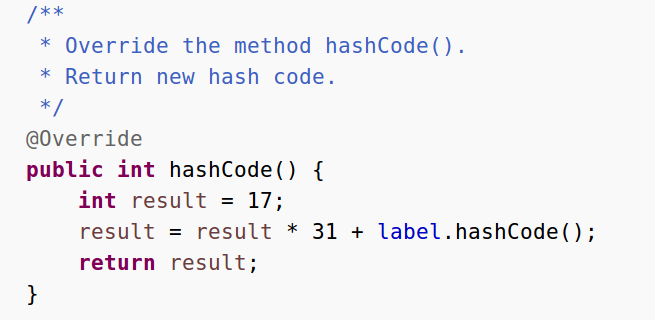
1. sourceVertices（）/ targetVertices（）



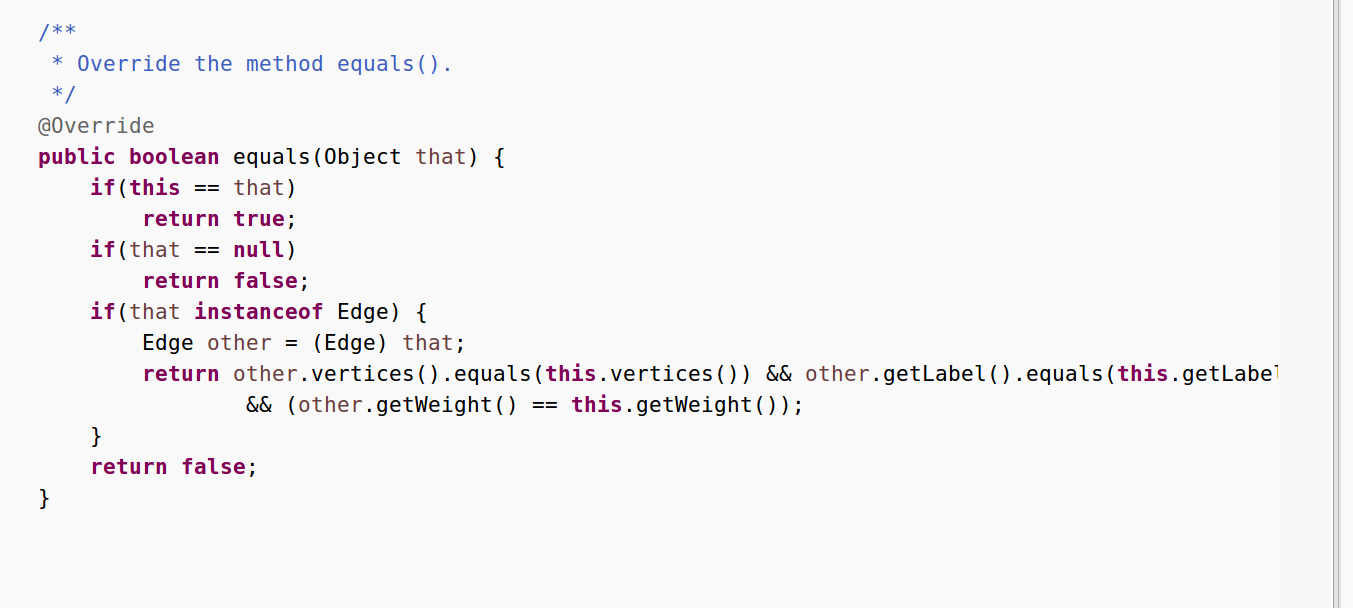
1. 重写toString（）方法



1. 重写hashCode（）方法



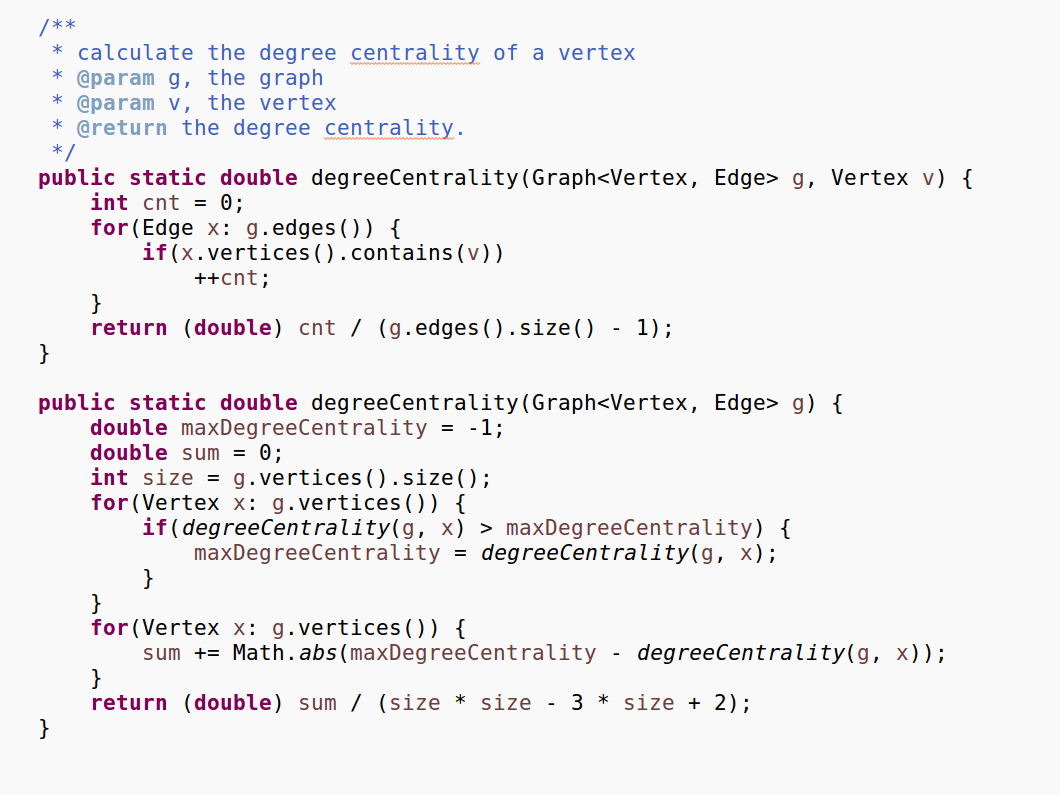
1. 重写equals（）方法



## 可复用API设计

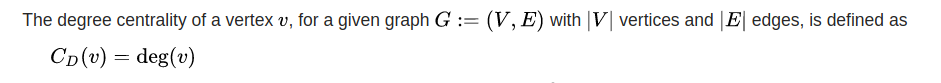
1. degreeCentrality

代码如下：

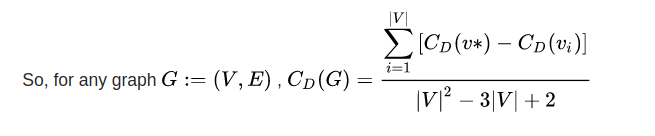


这里采用维基百科上给出的公式：

计算一个点的degree centrality：



计算一个图的degree centrality：



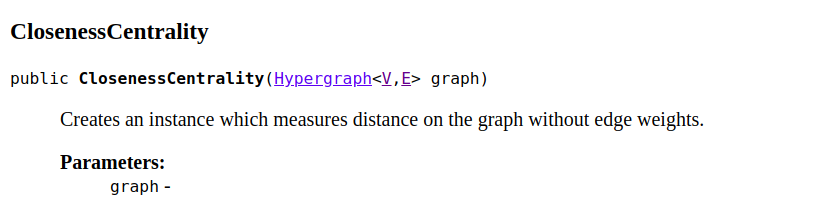
1. closeness centrality的计算

这里，本来打算用维基百科上的方法，想通过Dijkstra算法来求解最短路，并代入公式计算，但是老师说了可以调用第三方库，于是采用了调用JUNG包的方法，使用其中已经封装好的API，代码如下图



其中，调用了ClosenessCentrality类来进行计算即可。

官网给出的document如下：



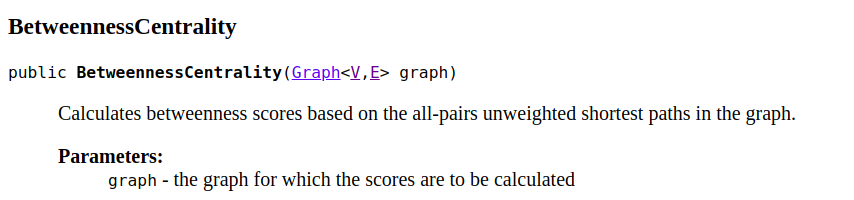
由此，计算出某一个Vertex的closeness centrality。

1. betweenness centrality的计算

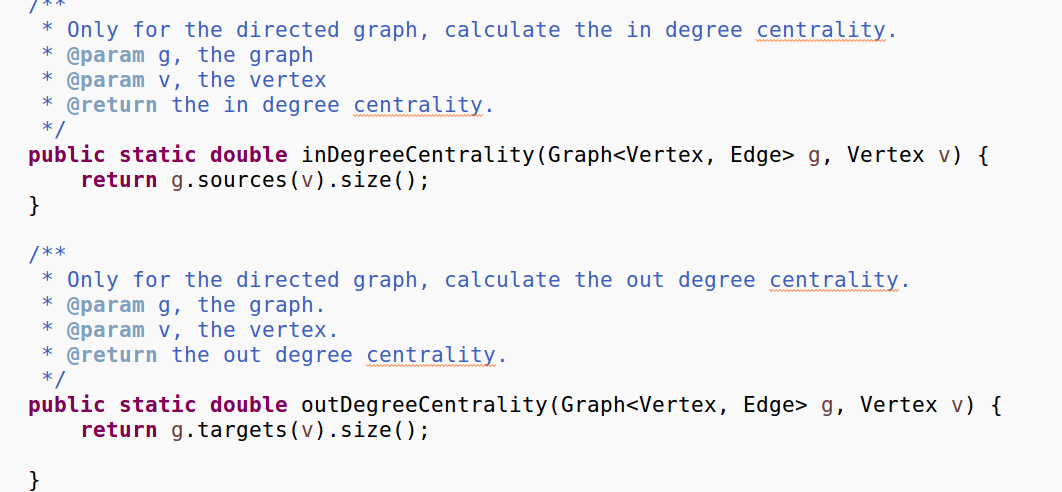


这里同样是采用调用第三方库的方法，来进行计算

官方文档如下：



1. indegreeCentrality/ outdegreeCentrality的计算



直接调用targets（）方法就可以计算。

（5）GraphDistance的计算

* **Distance(Graph<L, E> g, L start, L end)**

计算节点start和end之间的最短距离，使用dijkstra算法求得节点start的单源最短路径，取终点为end的最短距离返回。

* **Eccentricity(Graph<L, E> g, L v)**

一个顶点的离心率为该顶点到其他顶点最短距离的最大值。

实现方法：dijkstra算法求得v的单源最短路径，取单点到其他顶点最短距离的最大值。**此处需要注意，不能考虑无法到达的顶点。**

* **Radius(Graph<L, E> g)**

图的半径：图中顶点离心率的最小值。

实现方法：调用上面实现的eccentricity()方法，计算图中每一个顶点的离心率，取最小值。

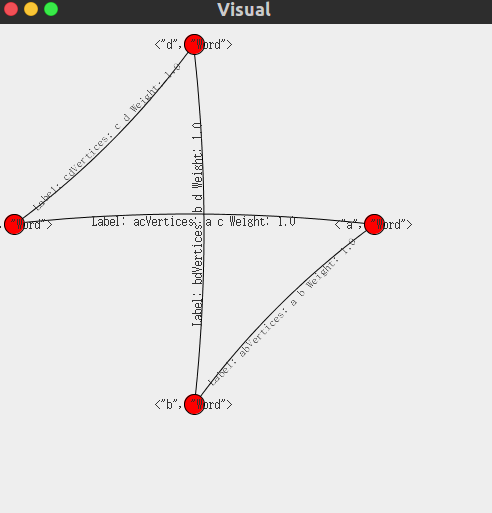
* **Diameter(Graph<L, E> g)**

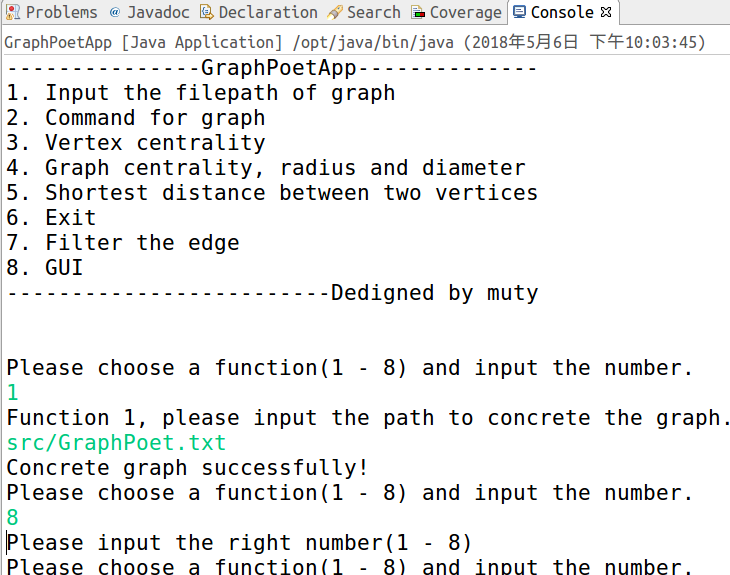
图的直径：与图的半径相反，为图中顶点离心率的最大值。

实现方法：调用上面实现的eccentricity()方法，计算每一个顶点的离心率取最大值。

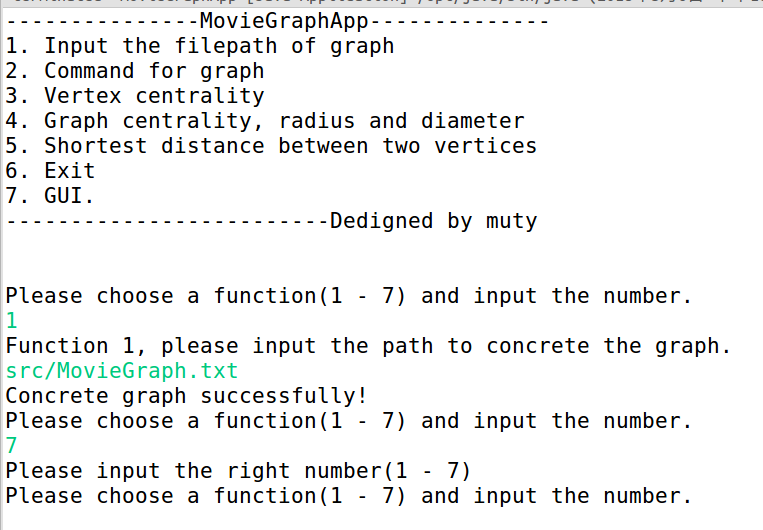
## 的可视化：第三方API的复用（选做）

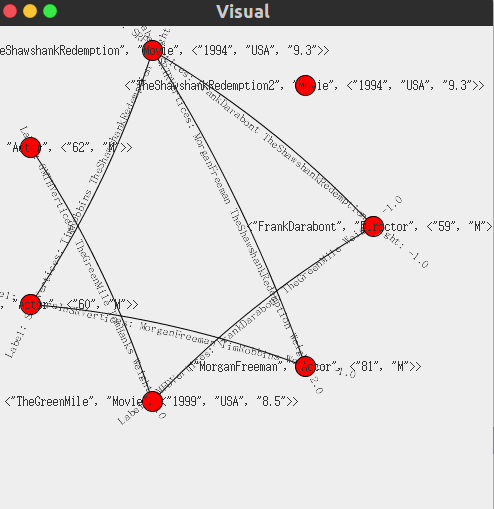
1. GraphPoet



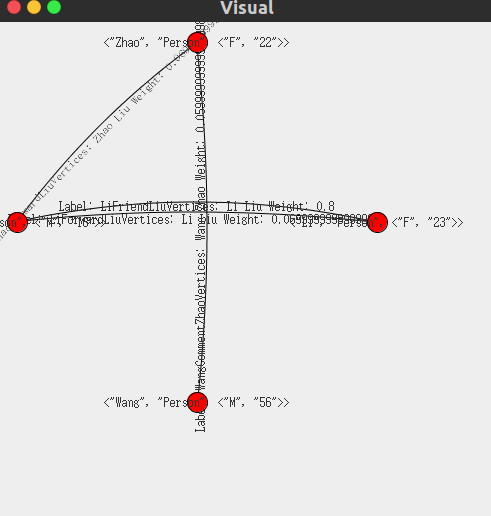


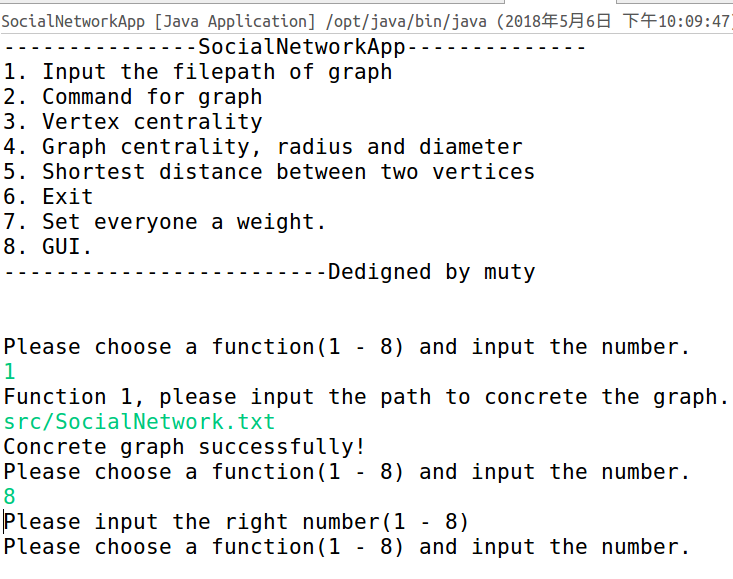
1. MovieGraph



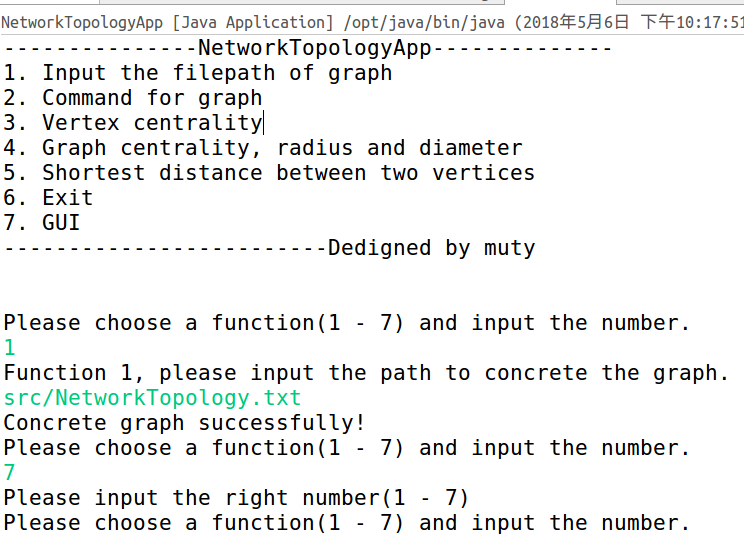


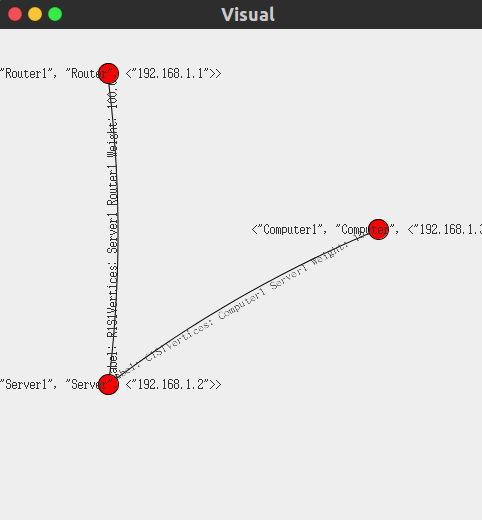
1. SocialNetwork



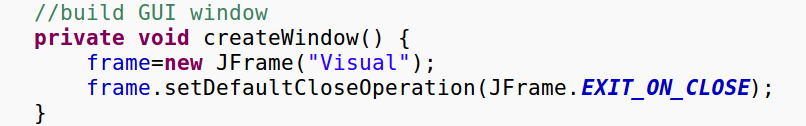


1. NetworkTopology

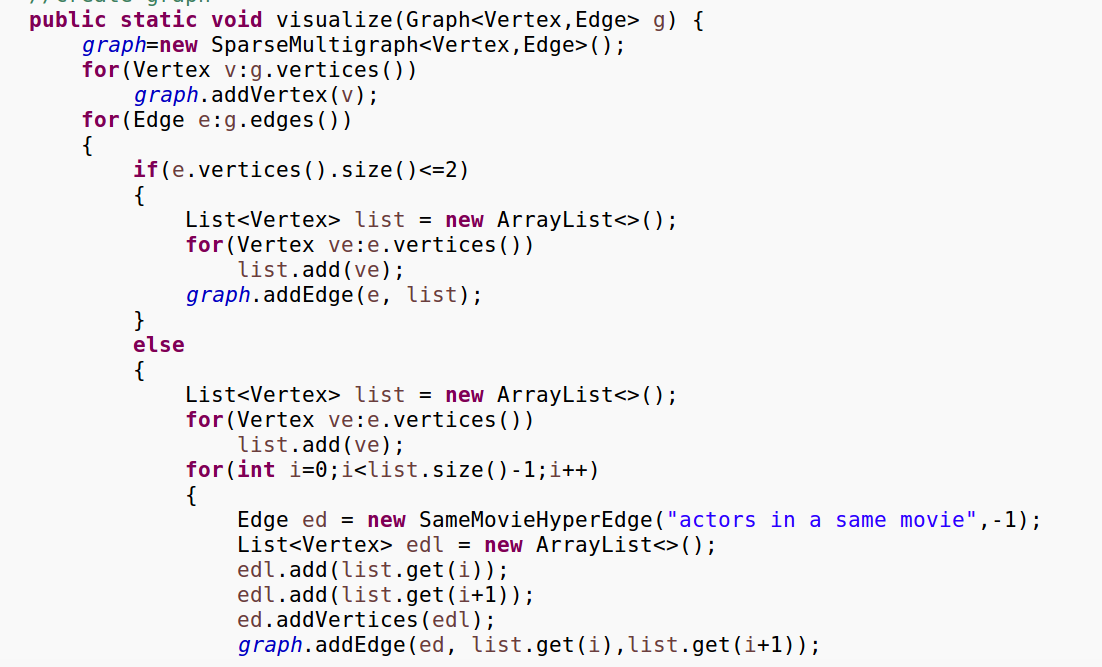




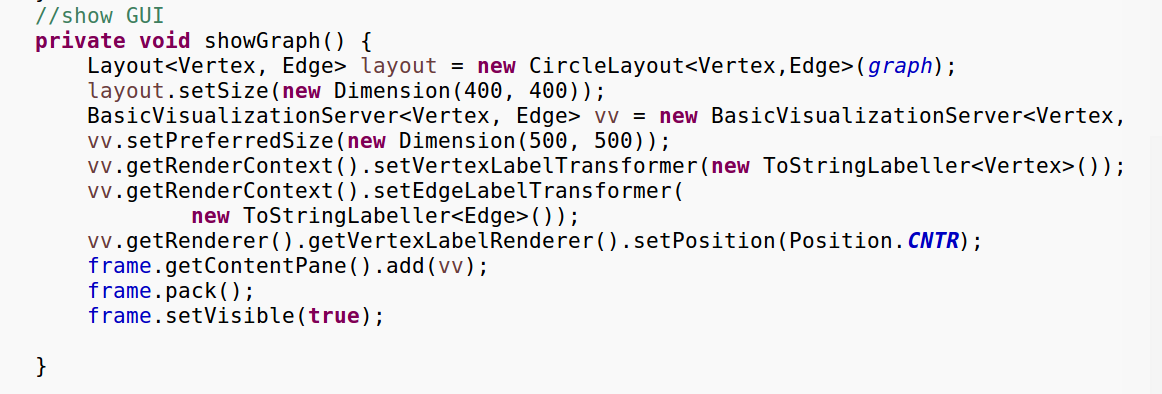
GraphVisualizationHelper.java代码如下：



这里是初始化一个Window，用来存放我们的图。



这里是创建图的过程，将我们的图中的点和边复制到我们作为参数的图中。



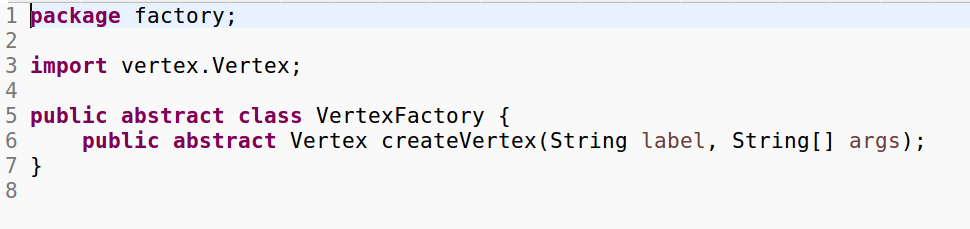
这里是将GUI图像展现出来部分的代码

## 设计模式应用

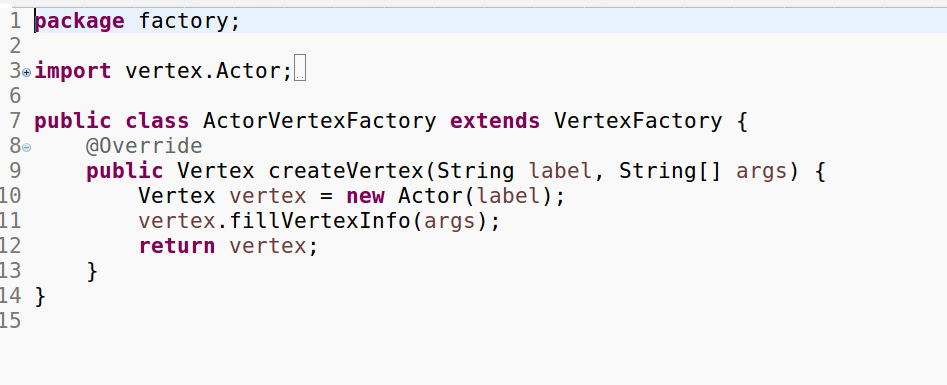
### 使用State/Memento模式进行Vertex的状态管理（选做）

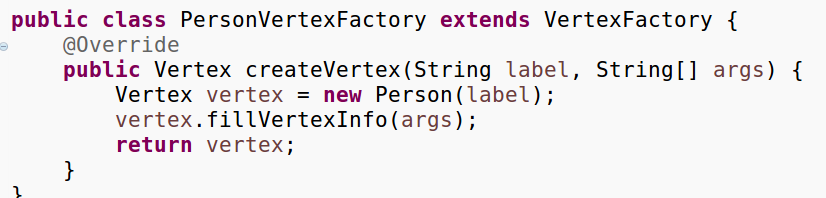
### 使用factory method模式构造Vertex对象

Vertex工厂代码如下：



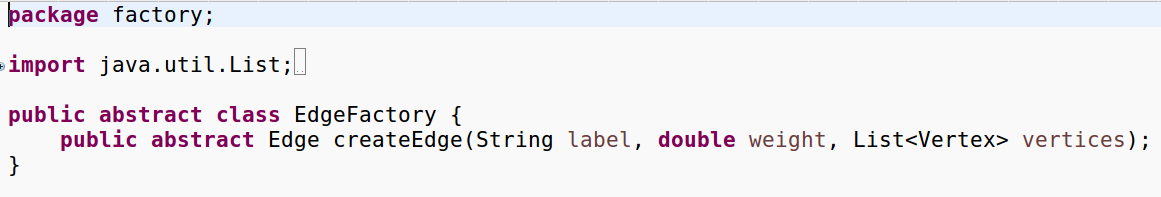
具体的vertex可以继承自vertexfactory，代码如下：



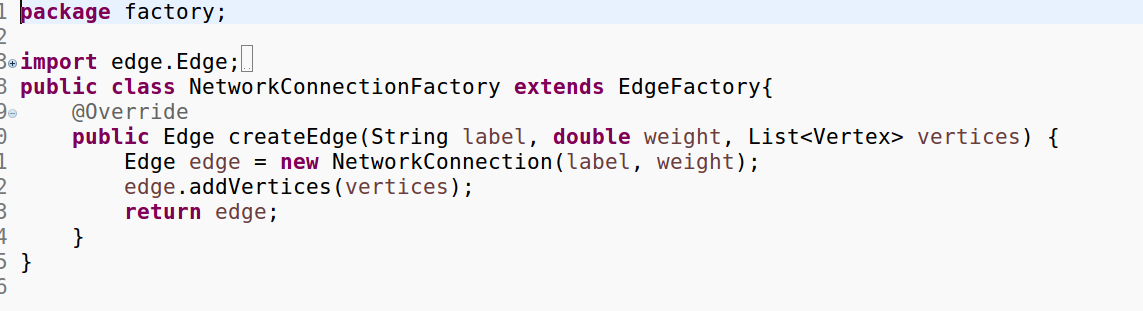


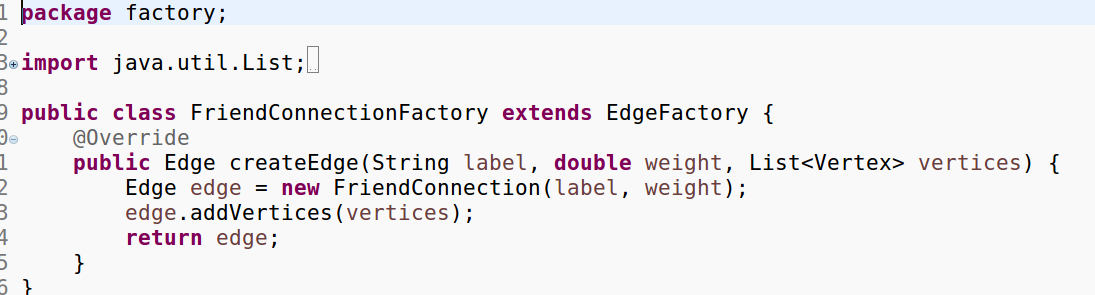
### 使用factory method模式构造Edge对象

有如下的抽象类：



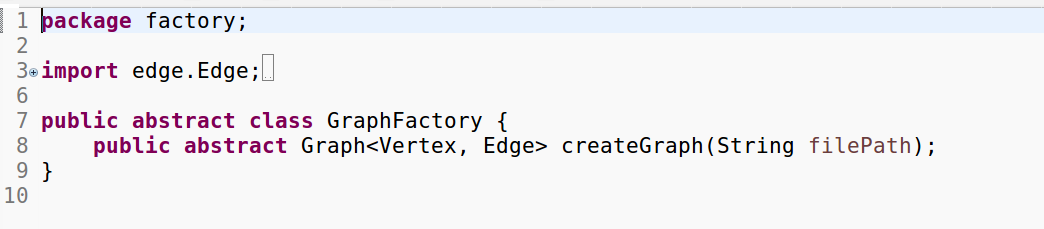
当想生成具体的边类的时候，就可以采用继承的方法，如下图所示：



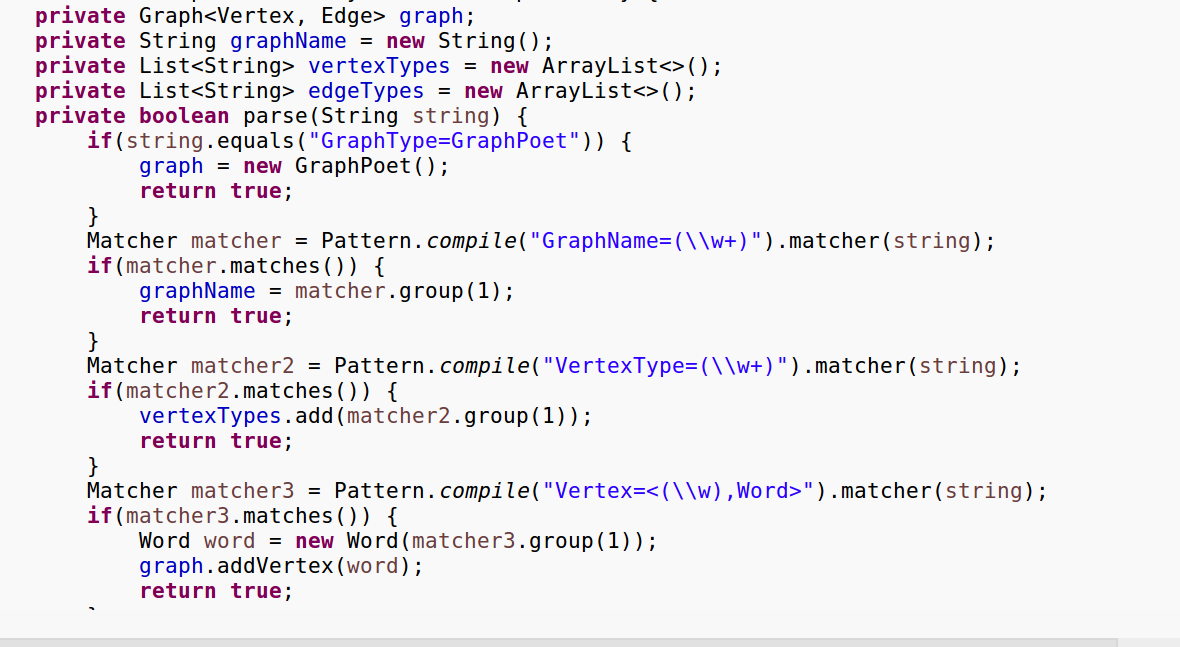


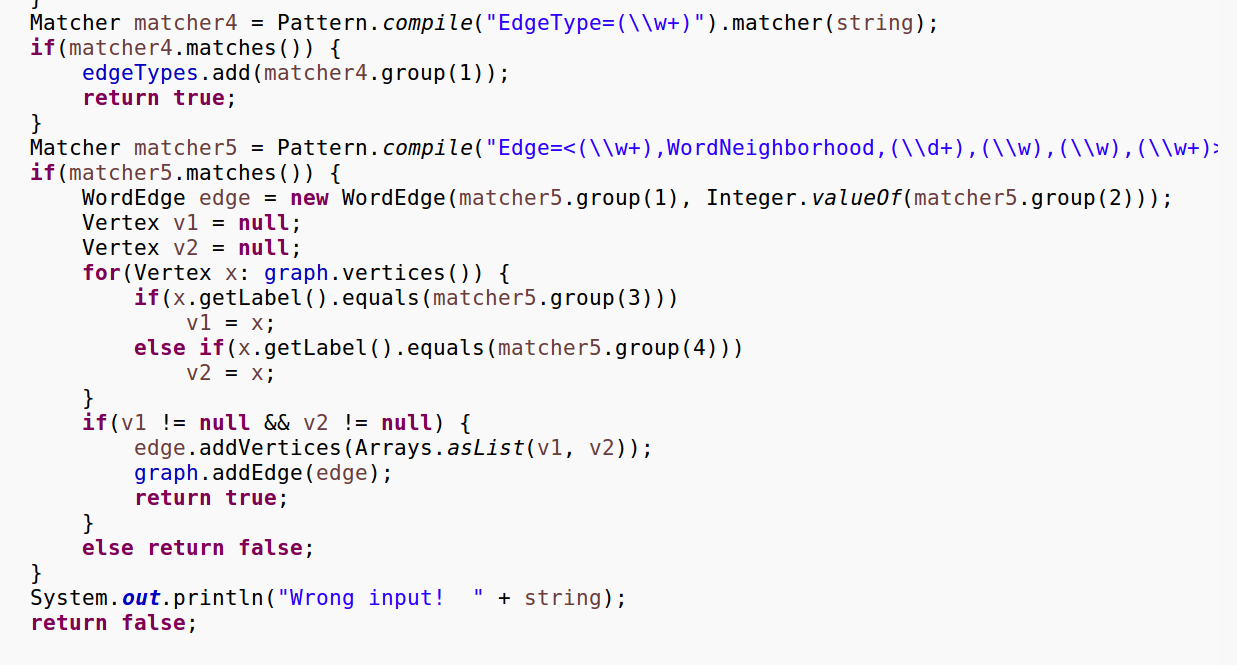
### 使用abstract factory或builder模式构造Graph对象

首先有一个抽象工厂如下：



然后有四个具体的图工厂类如下：



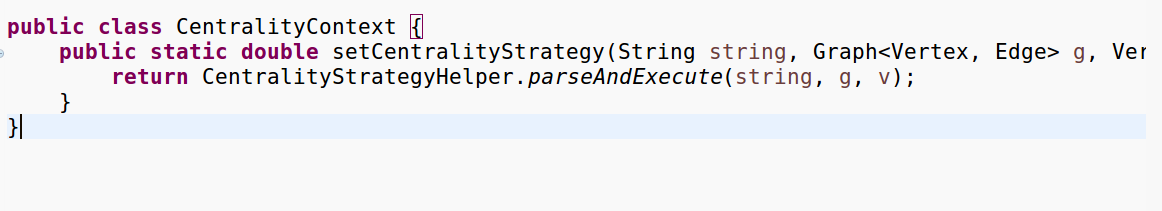


每个工厂类都会有一个parse方法来解析输入的文件，然后根据解析结果来进行建边或者建点加入到图中。

### 使用Strategy模式调用centrality度量算法

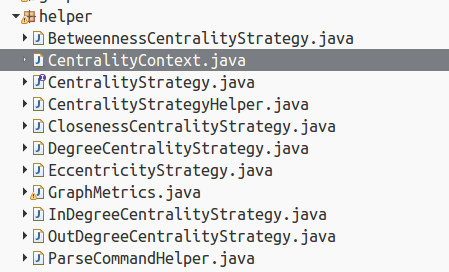
首先有一个CentralityContext类，通过传入不同的参数（String）

来进行setCentralityStrategy。



其中的String就代表着要求的不同的度。

然后有一个CentralityStrategyHelper类，去根据不同的string来调用不同的strategy方法，这些方法都实现一个接口，CentralityStrategy，由此实现了将输入与实现分开， 所以，最终目录如下



### 使用Composite模式设计超边对象（选做）

即在相同的类型的HyperEdge对象之间建立起一个树型层次结构，修改HyperEdge中的属性为一个Vertex类型，并将所有的超边加入到一个List<HyperEdge>对象中，如果在超边中需要添加一个顶点，需要新建一个超边对象，Vertex类型赋值为新建的顶点对象，向新创建的HyperEdge对象中的List只添加Vertex，并将Vertex添加至上层HyperEdge的List中。

### 使用decorator模式构造不同特征的Edge对象（选做）

最后写完代码，尝试使用Decorator模式去进行修改，发现修改量很大，首先对于本次试验，在不考虑给边增加特性的时候，Decorator的作用不是特别大，其次，如果想要Decorator模式，需要将Edge定义为接口，而不是抽象类，所以，需要从根本去进行修改，时间仓促，遂放弃，在这里特此说明原因。

### 使用其他设计模式（选做）

为了方便四个应用的客户端获得最短距离。由于GraphDistance中的最短距离方法需要传入的参数为Vertex v或者说是L v类型，但是从客户端只能传入String类型。所以在不改变原有代码的前提下，采用**适配器模式。**客户端只需要传入顶点的label，由适配器来确定具体的Vertex类型，然后再调用GraphDistance中的方法，并返回或输出最短距离。

## 图操作指令的输入和处理（选做）

（1）vertex --add“newVertex”“Word”

向图中加入新的节点

1. vertex --delete regex

regex是一个可用于正则匹配的字符串，比如说 vertex --delete “[a-zA-Z]\d{3}[a-zA-Z]”是删除label中有三个连续数字且数字前后为字母的节点。

1. edge --add label type [weighted=Y|N] weight [directed=Y|N] v1, v2

用于增加一条边，但是我觉得这条指令有一点小瑕疵，就是一旦type确定，这条边是不是有向边就已经确定了，没有必要加上[weighted]这一部分。

1. edge --delete regex

删除一条边，和删除节点是类似的。按照边的类型来进行加边

1. 超边的添加

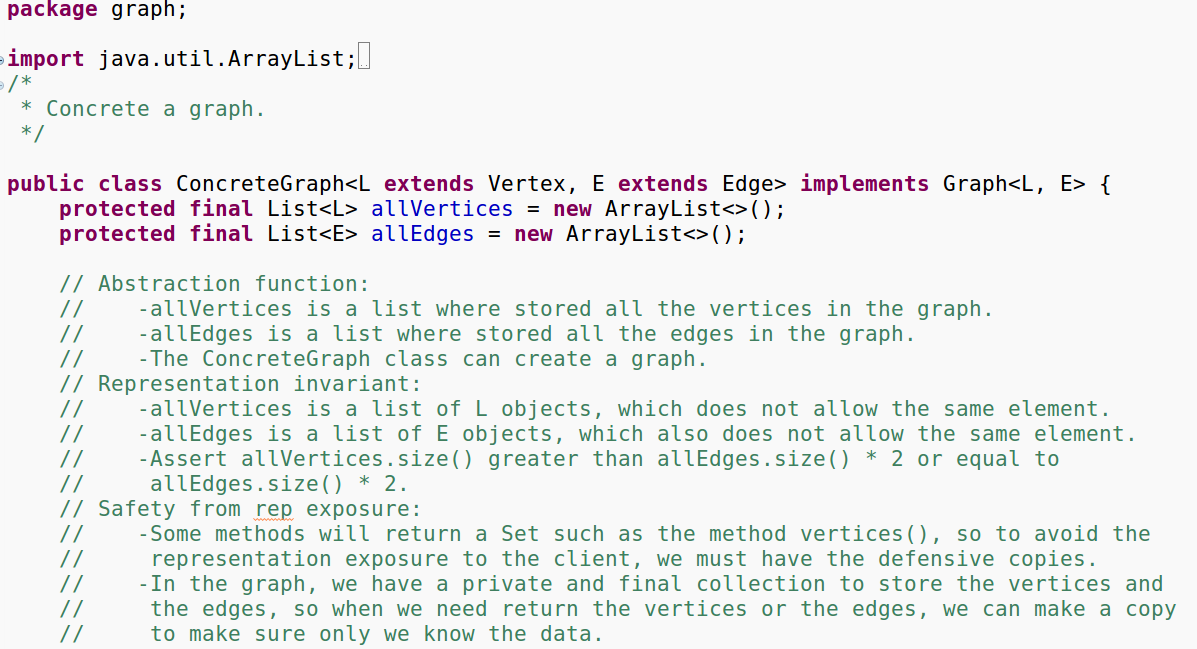
HyperEdge --add label type vertex1，vertex2 ... vertexn

关于超边是单独进行加边处理的，就是先要判断所有的点是不是都在graph中存在，如果不存在的话，则不能加边成功。

## 应用设计与开发

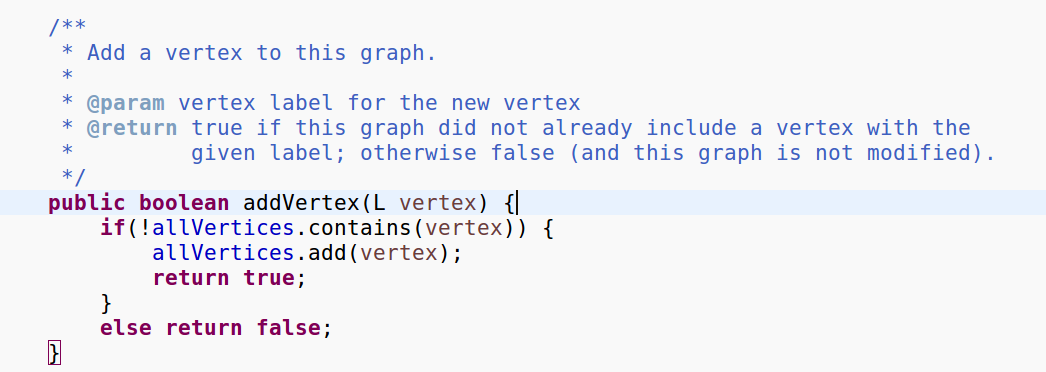
大部分的功能实现都是在concreteGraph这个Graph的实现类中实现的，只是有部分操作针对具体情况要进行override，所以，这里先来介绍一下concreteGraph类

首先有如下的AF、RI、rep exposure



1. addVertex（）方法：

首先在图中我有一个list去存储所有的点和所有的边，于是，就要先遍历一遍allVertices列表，去判断点是否已经在图中，如果已经存在的话，就返回flase，否则，加点，返回true。



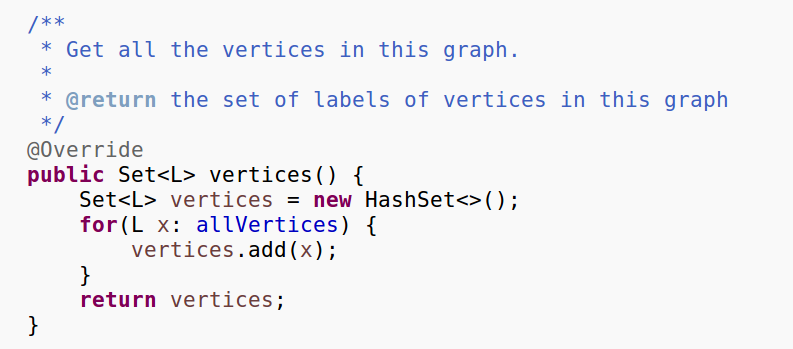
1. removeVertex（）方法

删点，不只是在allVertices中将点删除掉，还要遍历存储边的列表，将那些含有这个点的边删除掉。对于超边来说，如果删除掉其中一个点，该超边如果仍然满足超边的要求的话，依旧保留超边。

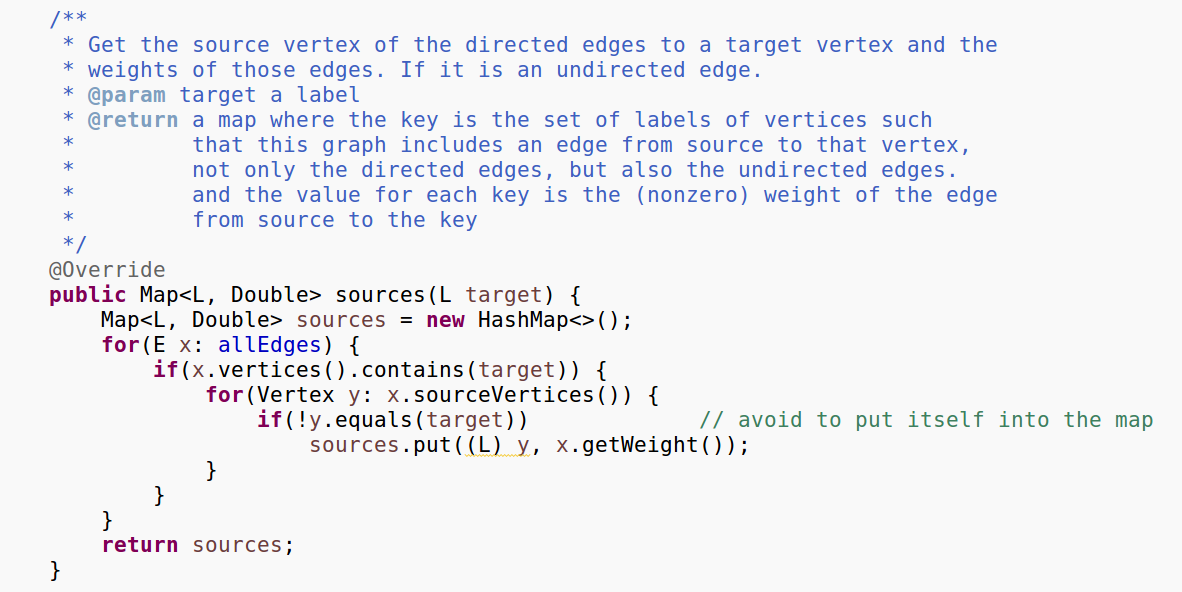


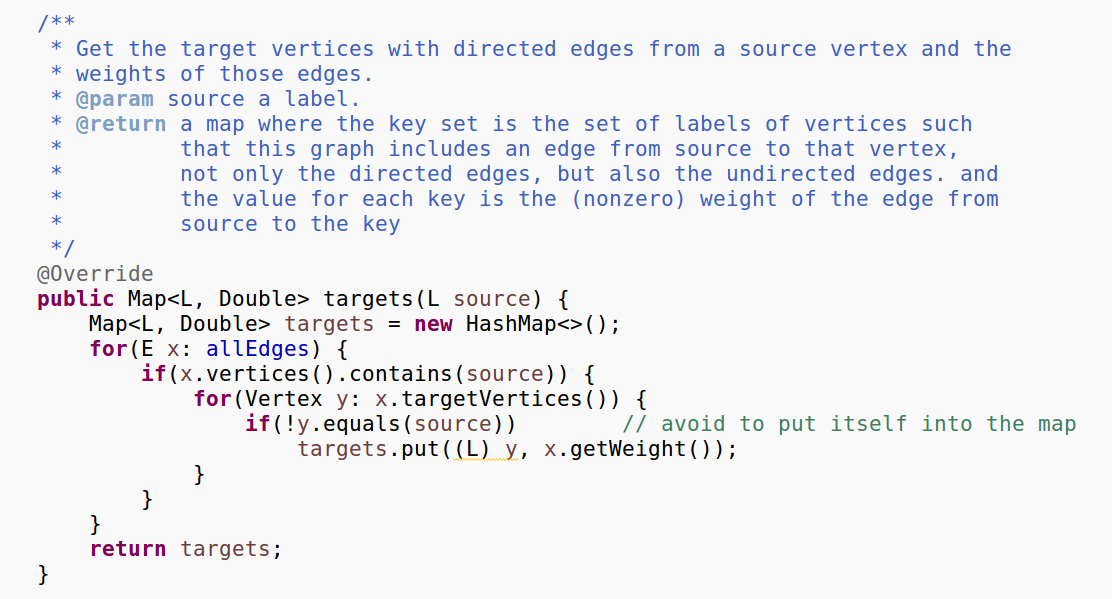
1. vertices（）方法

将allVertices中的所有元素全部都加入到set中，然后返回这个set即可



1. sources() / targets() 方法

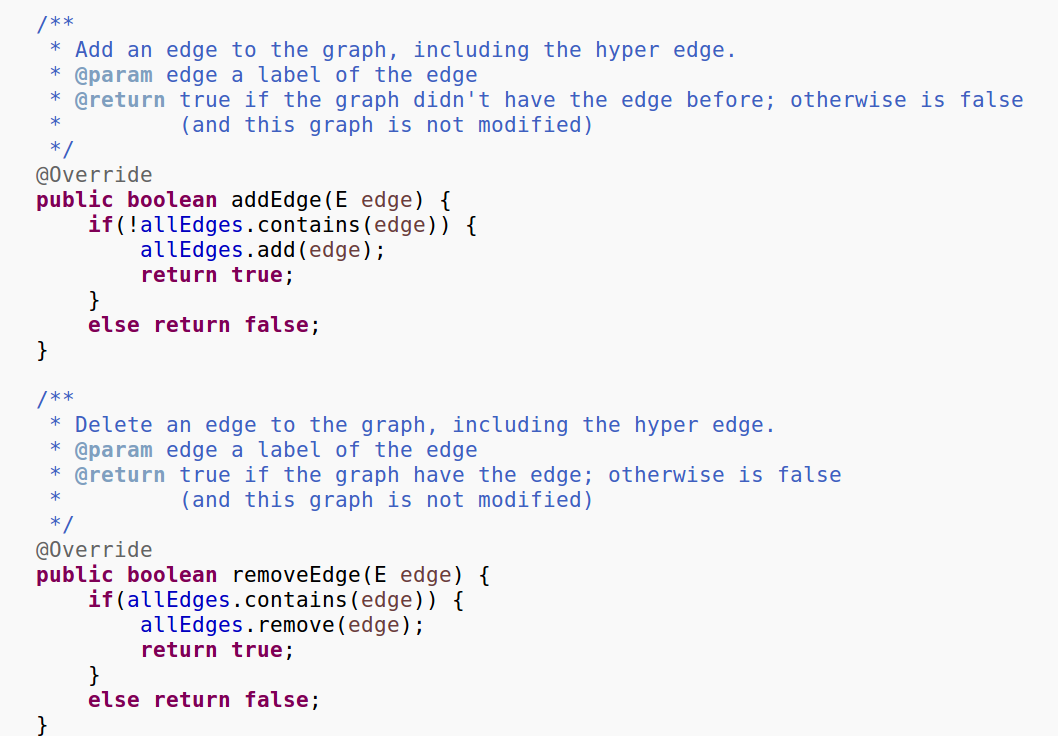




这个针对于有向图而言，返回起点或是终点，但是对于无向图，将两个点都返回即可，遍历list去寻找所有的点就可以。

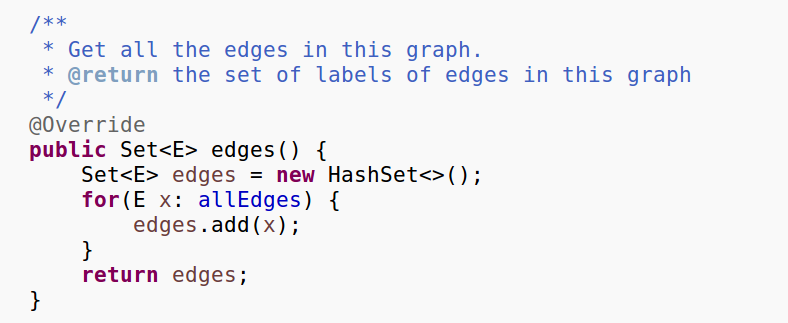
1. addEdge（）/ removeEdge（）

加边和删边操作，在allEdges中进行删除操作即可。



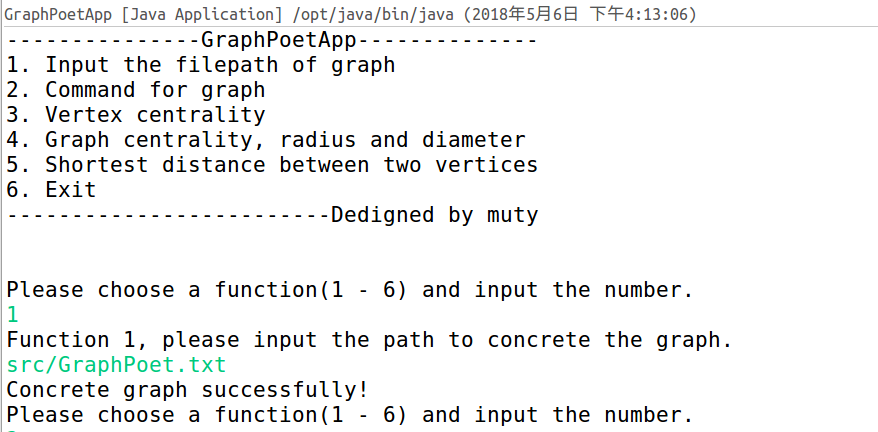
（6）edges（）

只需要将allEdges中的元素复制到一个list中，再返回该list即可。

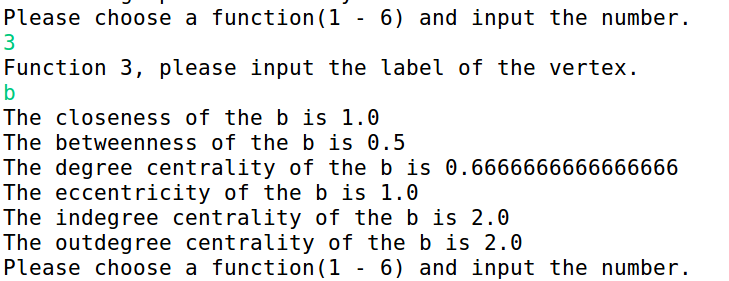


### 单词网络GraphPoet

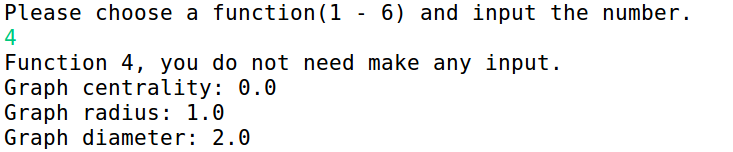
GraphPoetApp如下：



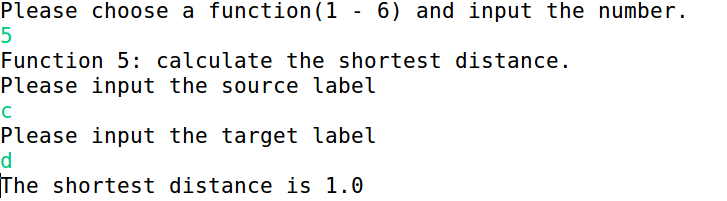
首先一定要调用第一个功能，因为首先一定要生成图, 然后才能进行后续操作。其次是一些图的指令，也就是之前提到过的加点加边等操作，利用正则表达式来进行解析，从而通过parseAndExecute类来进行相应的操作。3可以用来进行对于点的centrality计算，具体操作如下图：



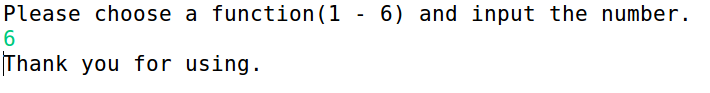
第4部分也就是对于图的一些相关属性来进行计算，操作如下图：



第5部分可以进行最短距离的计算，操作如下图：

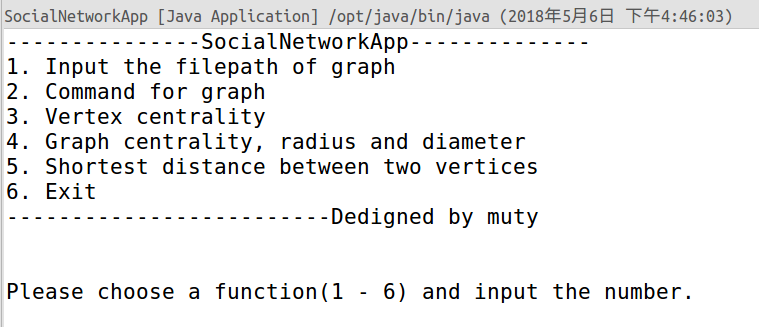


最后是退出程序的部分，如下图：



剩下的三个App文件也是类似的运行方式，不再提供具体的截图。

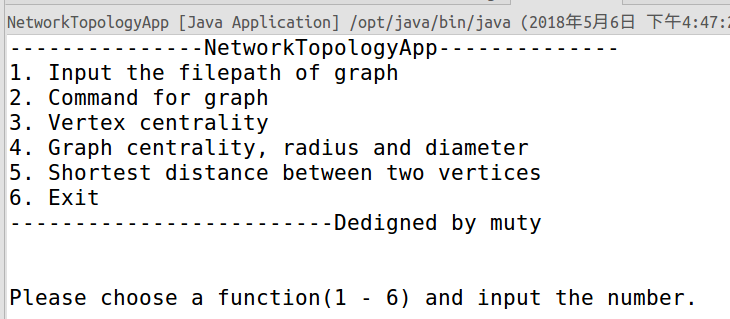
### 微博社交网络SocialNetwork



功能：与之前的相类似，在GraphPoet中已经详细阐述过。

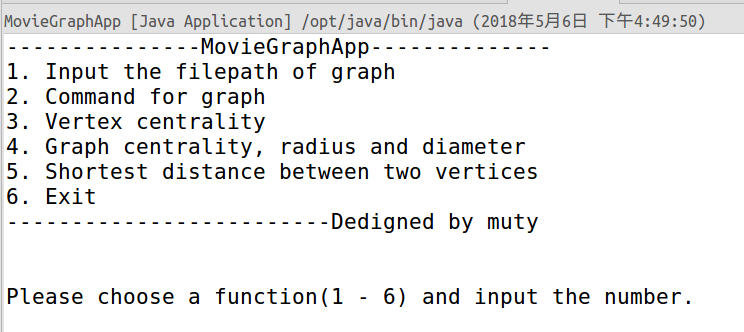
**关于这个图，我有一个疑问，就是实验手册中说加边删边都要修改其他边的权重，那么在读文件依次加边的时候，是不是也要修改其他边的权重，如果修改的话，那么也就是说最后得到的图和文件中的图完全不一样，还有，这样的话一定要保证第一条边权重为1，可是在解析文件的时候很难保证会先添加那条权重为1的边，一旦不能先添加权重为1的边，整个图的权重和就不再是1。**

### 网络拓扑图NetworkTopology



功能：与之前的相类似，在GraphPoet中已经详细阐述过。

### 电影网络MovieGraph

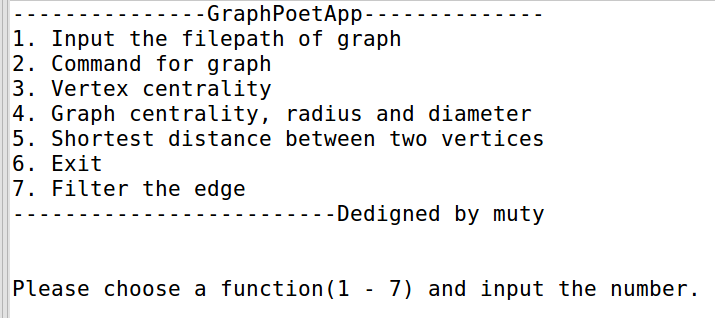


功能：与之前的相类似，在GraphPoet中已经详细阐述过。

## 应对四个应用面临的新变化（任选两个）

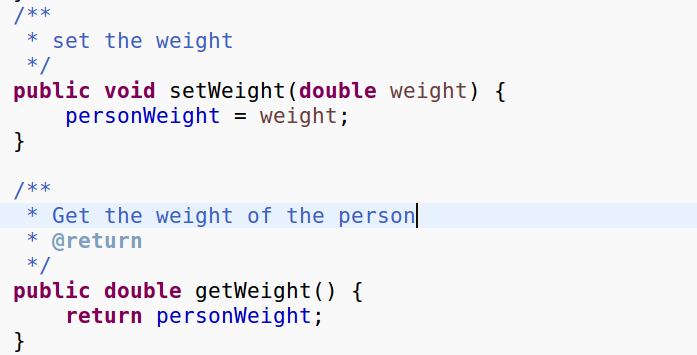
### 单词网络GraphPoet

GraphPoetApp中新加入一个功能, 目的就是设置最小边权重, 从而将所有边权重小于minWeight的边删掉

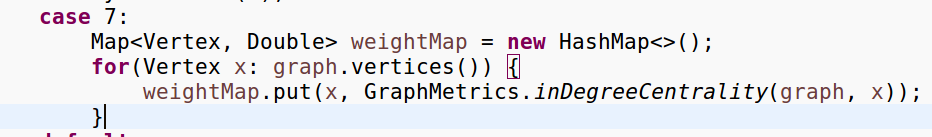


### 微博社交网络SocialNetwork

向Person.java中新增方法

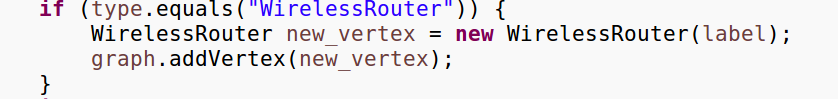


新建Map，存储节点的权重



### 网络拓扑图NetworkTopology

新建一个WirelessRouter类即可，只是需要在解析命令的时候多加一段代码用来新建WirelessRouter类的节点就好，改动不是特别大



新建WirelessRouter类，修改VertexFactory，修改Helper目录下的parseCommandHelper.java，修改factory目录下的NetworkTopology.java

### 电影网络MovieGraph

1. 创建MovieActorRelation类，继承自UndirectedEdge类。
2. EdgeFactory的静态工厂方法中增加该类的一个分支
3. ParseAndExecute中添加边的方法中，如果图为MovieGraph类型，在可以加入的边的类型中加入MoviePublicRelation

# 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 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# 实验过程中遇到的困难与解决途径

有关于接口和泛型的一些处理还是处理的不太熟， 所以就花了不少时间，还有就是关于centrality那里的计算，并且这个实验的工作量比较大，用了很多时间。

# 实验过程中收获的经验、教训、感想

本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. 重新思考Lab2中的问题：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？本实验设计的ADT在四个图应用场景下使用，你是否体会到复用的好处？
2. 重新思考Lab2中的问题：为ADT撰写复杂的specification, invariants, RI, AF，时刻注意ADT是否有rep exposure，这些工作的意义是什么？你是否愿意在以后的编程中坚持这么做？
3. 之前你将别人提供的API用于自己的程序开发中，本次实验你尝试着开发给别人使用的API，是否能够体会到其中的难处和乐趣？
4. 在编程中使用设计模式，增加了很多类，但在复用和可维护性方面带来了收益。你如何看待设计模式？
5. 你之前在使用其他软件时，应该体会过输入各种命令向系统发出指令。本次实验你开发了一系列命令行指令，使用语法和正则表达式去解析它们并映射到对后台程序的调用。你对语法驱动编程有何感受？
6. 关于本实验的工作量、难度、deadline。

**关于这门实验，首先是老师对很多细节考虑不周，但是我们理解，毕竟只是几个老师来想试验，也是第一次开这门课，这我们完全理解，但是在QQ群中对要求一改再改，导致昨晚和同学交流才发现有很多要求都被修改了，代码也要重新写，这让我表示很气愤，实验手册没有说清楚，按照自己的理解辛辛苦苦写好的代码又要改，改来改去，并且，所有要求都是在聊天记录中呈现的，还得翻记录，经常有老师不能自圆其说的现象发生，另外，有同学要求说将修改的一些要求写个文件放到群里，老师当时答应了，可是知道现在（截止日期前一天）还没有看到这样的文件，希望老师可以今后可以改正。**

1. 到目前为止你对《软件构造》课程的评价。