

**2019年春季学期  
计算机学院《软件构造》课程**

**Lab 1实验报告**

|  |  |
| --- | --- |
| 姓名 | 吕悦 |
| 学号 | 1173710115 |
| 班号 | 1737101 |
| 电子邮件 | aether379@gmail.com |
| 手机号码 | 13149522166 |

**目录**

[1 实验目标概述 1](#_Toc1392798)

[2 实验环境配置 1](#_Toc1392799)

[3 实验过程 1](#_Toc1392800)

[3.1 Magic Squares 1](#_Toc1392801)

[3.1.1 isLegalMagicSquare() 1](#_Toc1392802)

[3.1.2 generateMagicSquare() 3](#_Toc1392803)

[3.2 Turtle Graphics 4](#_Toc1392804)

[3.2.1 Problem 1: Clone and import 4](#_Toc1392805)

[3.2.2 Problem 3: Turtle graphics and drawSquare 5](#_Toc1392806)

[3.2.3 Problem 5: Drawing polygons 5](#_Toc1392807)

[3.2.4 Problem 6: Calculating Bearings 6](#_Toc1392808)

[3.2.5 Problem 7: Convex Hulls 7](#_Toc1392809)

[3.2.6 Problem 8: Personal art 8](#_Toc1392810)

[3.2.7 Submitting 8](#_Toc1392811)

[3.3 Social Network 9](#_Toc1392812)

[3.3.1 设计/实现FriendshipGraph类 9](#_Toc1392813)

[3.3.2 设计/实现Person类 10](#_Toc1392814)

[3.3.3 设计/实现客户端代码main() 10](#_Toc1392815)

[3.3.4 设计/实现测试用例 11](#_Toc1392816)

[3.4 Tweet Tweet 11](#_Toc1392817)

[3.4.1 Problem 1: Extracting data from tweets 11](#_Toc1392818)

[3.4.2 Problem 2: Filtering lists of tweets 13](#_Toc1392819)

[3.4.3 Problem 3: Inferring a social network 14](#_Toc1392820)

[3.4.4 Problem 4: Get smarter 15](#_Toc1392821)

[4 实验进度记录 17](#_Toc1392822)

[5 实验过程中遇到的困难与解决途径 17](#_Toc1392823)

[6 实验过程中收获的经验、教训、感想 18](#_Toc1392824)

[6.1 实验过程中收获的经验和教训 18](#_Toc1392825)

[6.2 针对以下方面的感受 18](#_Toc1392826)

# 实验目标概述

训练基本 Java 编程技能，能够利用 Java OO 开发基本的功能模块，能够阅读理解已有代码框架并根据功能需求补全代码，能够为所开发的代码编写基本的测试程序并完成测试，初步保证所开发代码的正确性。 另一方面利用 Git 作为代码配置管理的工具，学会 Git 的基本使用方法。

# 实验环境配置

由于在之前已经完成过Github的注册与Java开发环境的配置，所以只需完成github classroom的关联即可。

GitHub Lab1仓库的URL地址：<https://github.com/ComputerScienceHIT/Lab1-1173710115.git>

可使用git clone将空仓库克隆后在本地用命令行对仓库进行操作，命令行操作如下：

git init

git add .

git commit -m "first commit"

git remote add origin https://github.com/ComputerScienceHIT/Lab1-1173710115.git

git push -u origin master

# 实验过程

## Magic Squares

MagicSquare实现目标为测试五个文本中的矩阵是否为MagicSquare，并将结果输出。

### isLegalMagicSquare()

根据分析确定此函数需要实现以下功能：

* 读取文件
* 储存矩阵
* 判断行列之和
* 异常处理（行列数不相等、并非矩阵等）、矩阵中的某些数字 并非正整数、数字之间并非使用\t分割、等）

读取文件部分的实现调用的是以前积累的的代码：

BufferedReader in = new BufferedReader(

new InputStreamReader(

new FileInputStream("src/hello/Main.java")));

String line;

while ((line = in.readLine()) != null) {

System.out.println(line);

}

从文件中每次读入一行字符串，使用split函数将其分割，并转换为整型存储在矩阵中。

String[] data = line.split("\t");

square[row][col] = Integer.valueOf(data[col]);

判断行列之和即对二维数组进行一次遍历，将每行与每列的和进行累加后判断彼此是否相等即可，思路简单，代码省略。

输入文件中可能出现各种特殊情况，例如：文件中的数据不符合定义（行列数不相等、并非矩阵等）、矩阵中的某些数字并非正整数、数字之间并非使用\t 分割等。可使用Java中的异常处理程序try/catch对异常进行捕捉：

Exception e1 = new Exception("输入非矩阵");//自定义异常

if (data.length != length) throw e1; //如果矩阵每行长度不一样 抛出异常

矩阵中的某些数字并非正整数和数字之间并非使用\t分割的情况将触发NumberFormatException，在try后增加对应的catch从句即可捕捉此异常并处理，使函数返回。

try{

//...

} catch (NumberFormatException e) {

System.err.println("Error occured in "+ fileName);

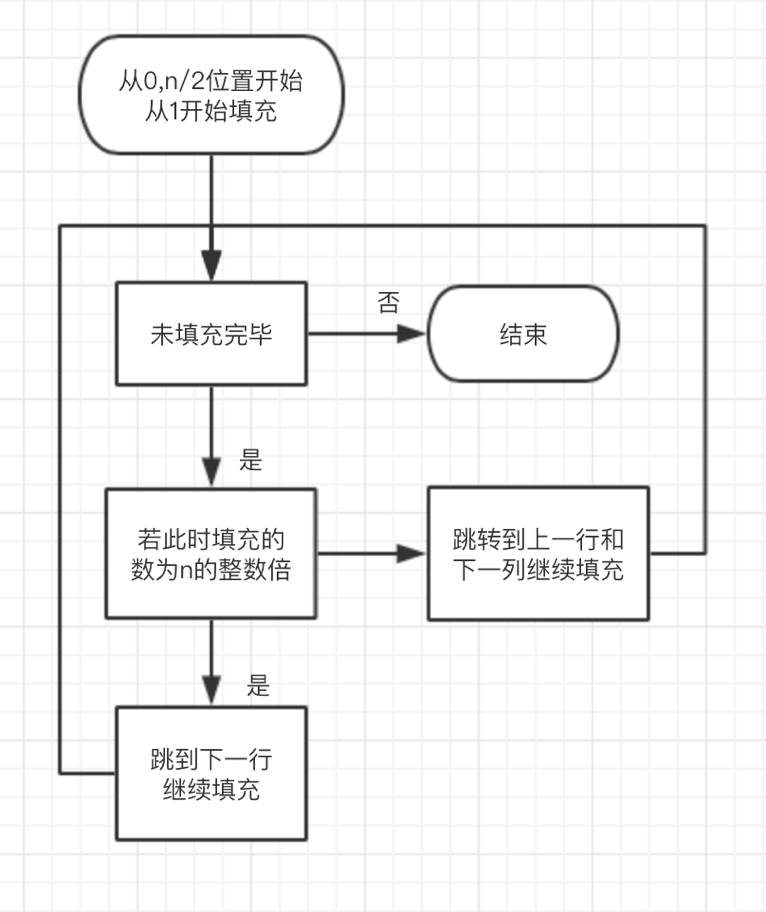
System.err.println("矩阵中的某些数字非正整数");

return false;

}

### generateMagicSquare()

生成MagicSquare的流程图如下所示：



将产生的 magic square 写入文件\src\P1\txt\6.txt 中的实现依旧使用之前积累的代码：调用PrintWriter：

PrintWriter out = new PrintWriter(new BufferedWriter(

new OutputStreamWriter(

new FileOutputStream("src/P1/txt/6.txt"))));

for (i = 0; i < n; i++) {

for (j = 0; j < n; j++)

out.print(magic[i][j] + "\t");

out.println();

}

当输入的 n 不合法时(n 为偶数、n 为负数等)，不要该函数抛出异常并非法退出的话可以使用catch对异常进行捕捉和处理，在catch语句中提示错误并“优雅的”退出——函数输出 false 结束。

try{

//...

}catch (ArrayIndexOutOfBoundsException e) {

System.err.println("n不能为偶数，否则数组访问越界");

return false;

} catch (NegativeArraySizeException e) {

System.err.println("数组大小n不能为负数");

return false;

}

在main函数中将生成的MagicSquare在isLegalMagicSquare函数中测试通过。

## Turtle Graphics

Turtle Graphics任务目标为实现TurtleSoup.java中的多个函数并进行测试，通过实验熟悉Java语法和开发环境、Eclipse、Junit测试、Git版本控制。

### Problem 1: Clone and import

P2不是一个独立的仓库 git clone貌似不能直接clone下整个仓库 可以将整个整个都clone下来

$ git clone https://github.com/rainywang/Spring2019\_HITCS\_SC\_Lab1

如果有svn的话可以单独clone下P2文件夹

$ svn export <https://github.com/rainywang/Spring2019_HITCS_SC_Lab1/trunk/P2>

然后进入到clone下的文件夹，将其初始化为git仓库 commit初始版本

$ cd P1

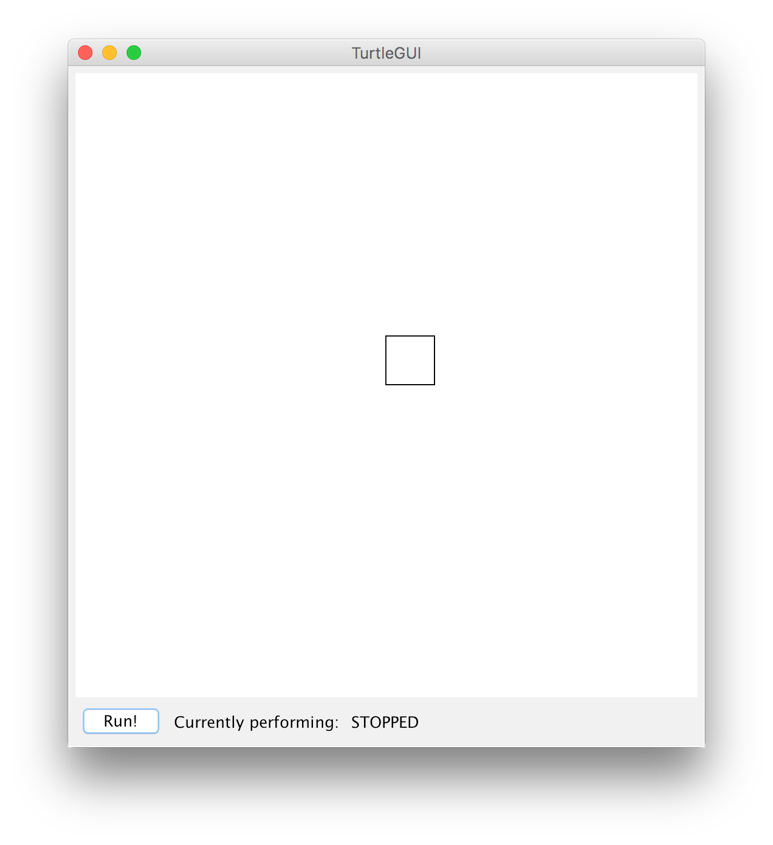
$ git init

$ git add .

$ git commit –m “origin”

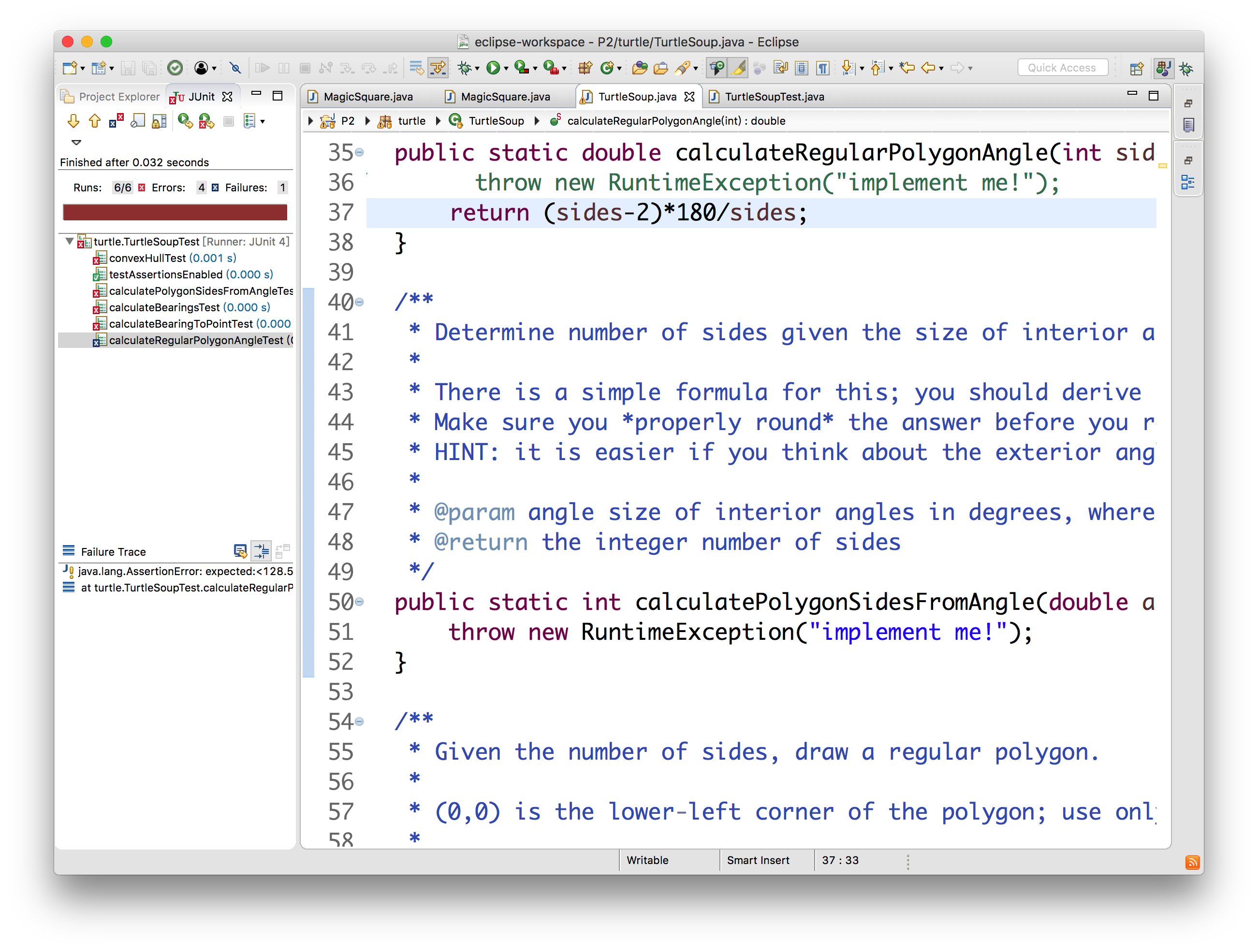
### Problem 3: Turtle graphics and drawSquare

drawSquare的需求非常简单，只需使用forward画边turn旋转90度即可。经过4次for循环可画出方形的四个边，效果如图

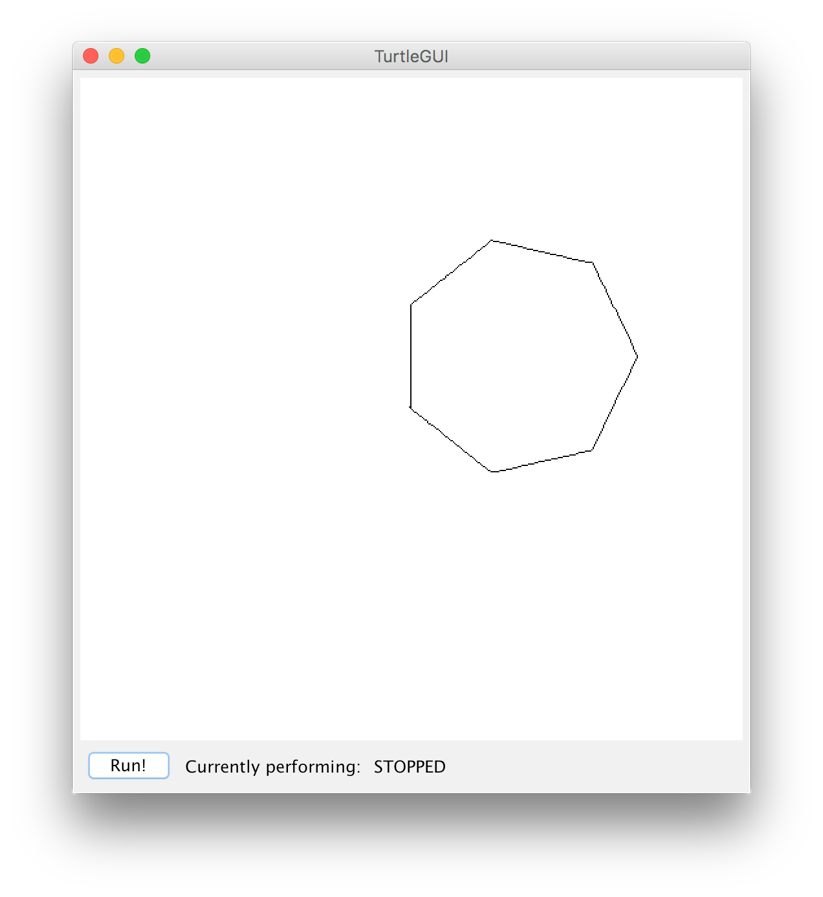


### Problem 5: Drawing polygons

calculateRegularPolygonAngle函数要求返回正多边形的内角，计算公式非常简单。使用JUnit可测试返回的内角度数是否正确。测试通过与未通过如图所示：

../Desktop/屏幕快照%202019-02-26%20下午10.03.59.png

drawRegularPolygon中调用calculateRegularPolygonAngle可获取多边形内角之度数，但是应注意turn旋转角度应是内角的补角的度数。结果如图所示



### lem 6: Calculating Bearings

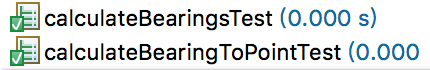
该阶段需实现两个函数：calculatingBearingToPoint 和calculatingBearings。使用向量点乘的数学原理及公式可算出结果。可以把从起点到终点抽象为向量，旋转的角度即向量与y轴之间的夹角。点乘与向量长度的商即角度的余弦值。对角度的取值范围应进行处理。可调用Java中的Math类中的常量与函数进行数学运算。

double length = Math.sqrt(x\*x + y\*y);

double angle = Math.acos((double)y / length)/Math.PI\*180;

return angle >= currentBearing ? angle - currentBearing : angle - currentBearing + 360;

运行calculatingBearingTest可对自己的函数进行测试与修改。通过JUnit测试后的图片如下所示：



### Problem 7: Convex Hulls

在这里使用暴力的算法——枚举来选择Convex hulls。Convex hull即最小的包括全部点的定点集，在这里使用了简单的数学方法去测试点是否为Convex hull的点：（debug很久）

具体思想：对于一个n个点集合中的两个点p1和p2，这两个点可以确定一条直线。当且仅当该集合中的其它点都位于这条直线的同一边时，这条直线就是Convex hulls的边界，也就是说p1和p2就是Convex hulls中的顶点。

在坐标平面上穿过两个点(x1, y1)，(x2, y2)的直线公式如下：

ax + by = c

其中 a = y2 – y1, b = x1 – x2, c = x1\*y2 – y1\*x2。

根据上面的公式，我们只需要把每个点代入公式ax+by-c，判断公式计算结果的符号是否全部大于0或者小于0，如果是则是Convex hulls上的点，否则不是。该算法的时间效率为O（n^3）。

//统计点在直线上侧和下侧的情况

for (Point point : points) {

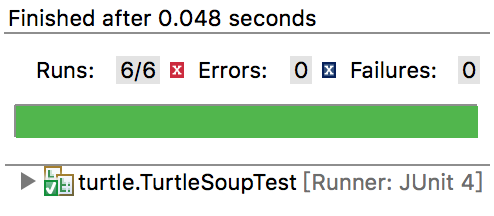
double result = a\*point.x() + b\*point.y() - c;

if (result > 0) flag++;

if (result < 0) flag--;

}

测试结果如图所示：（get a green bar :D）



### Problem 8: Personal art

这一部分自由发挥即可 在这里我先画一朵花再画五角星，循环七次得到一图案。

### 屏幕快照%202019-02-27%20上午11.46.46.png

在画花的时候将enum中的PenColor颜色遍历，得到多种颜色。

for (PenColor color : PenColor.values()) {

turtle.color(color);

turtle.forward(100);

turtle.turn(110);

}

### Submitting

首先使用add将全部内容加载到工作区，commit提交后push到github远程仓库。

$ git add .

$ git commit –m “finish P2”

$ git push

## Social Network

这个任务中的需求为建立一个无向图，并能够计算任意两个点之间的最短距离，并使用JUnit对代码中的函数进行测试。

### 设计/实现FriendshipGraph类

思考FriendshipGraph特征，提取出以下需求：

* 能够在图中新增顶点 addVertex
* 能够在图中新增边 addEdge
* 能够计算图中任意两个点的最短路径 getDistance
* 能够处理非常规输入

考虑到顶点集可能有变化，所以使用ArrayList<Person>储存图的顶点集，二维矩阵储存无向图，借助java.util.LinkedList和java.util.Queue进行广度优先搜索来计算最短路径。在每次的getDistance过程中生成新的图，性能降低但是避免了数组越界，增强了可维护性。

* getDistance

在此函数中先根据关系新建图储存在二维数组中，然后对此图进行深度优先搜索。bfs的核心代码如下所示：

while (!queue.isEmpty()) {

cnt++;

k = queue.poll();

for (int j = 0; j < n; j++) {

if (graph[k][j] != 0 && !visited[j]) {

queue.add(j);// 若没访问过且认识 将其加入队列

visited[j] = true;

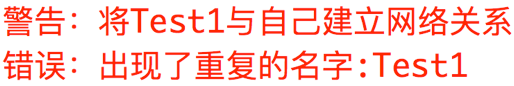
if (vertex.get(j).equals(p2))

return cnt;// 找到 返回距离

}}}

* addVertex/addEdge

在addVertex/addEdge中应对函数参数进行检查，处理非法输入。考虑到其“Each person has a unique name”的约束条件，添加时进行判断，若出现问题则输出错误提示。



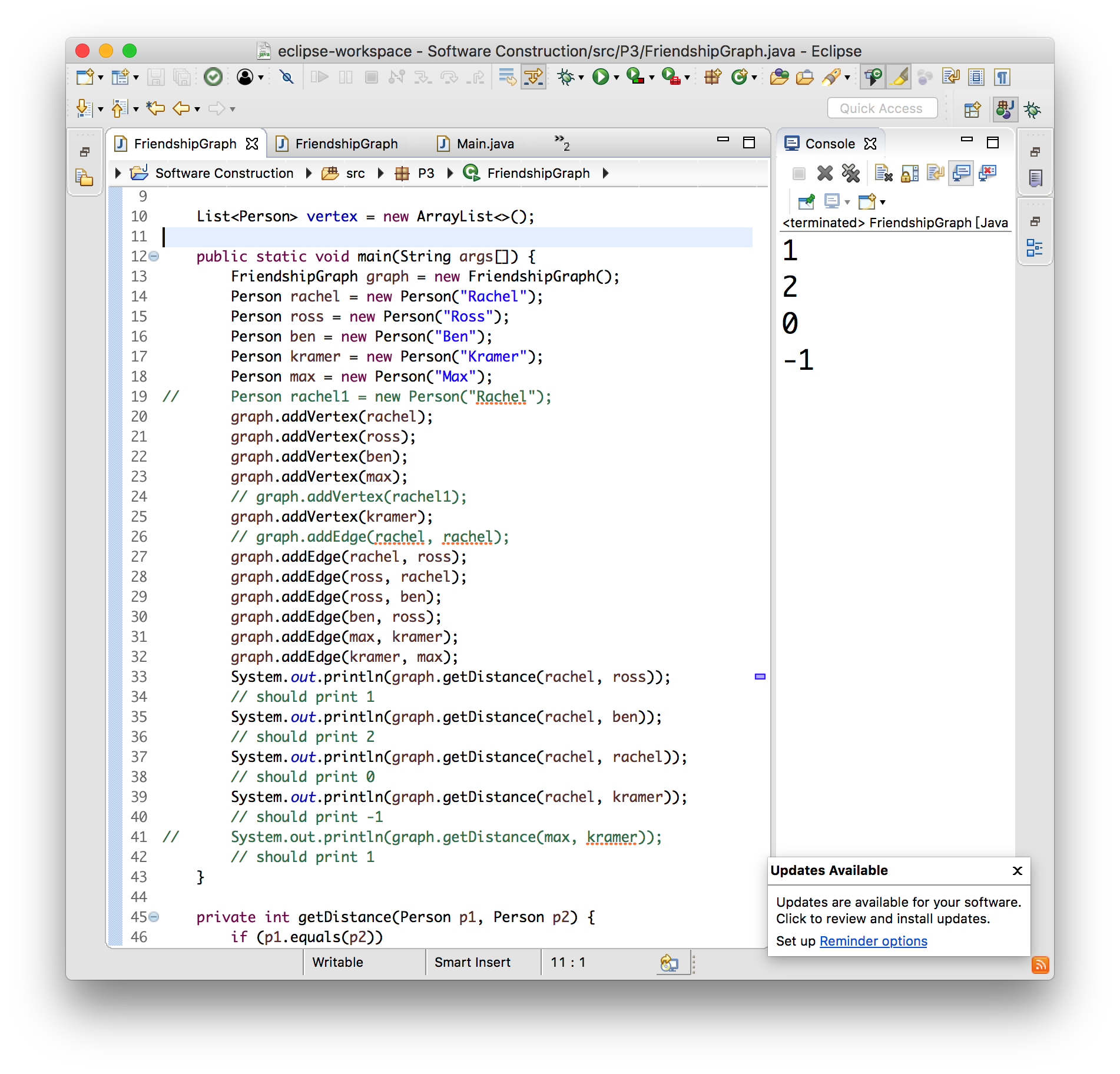
### 设计/实现Person类

思考Person的特征，抽象出以下两点特征：

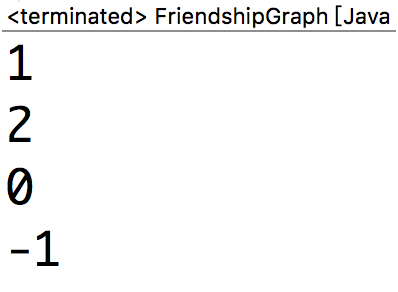
- 姓名 （String）

* - 朋友／认识的人 （使用List<Person>储存）
* 为了保证变量的私有性，可设置两个get接口供外界访问成员变量。
* 若新增此Person对象的认识的人，可直接在list中添加Person对象。

### 设计/实现客户端代码main()



输出与期望相符合：



将第十行代码注释掉，结果如下

### 设计/实现测试用例

在Test中应对FriendshipGraph的三个函数都进行测试。实现难点为实验要求中对权限进行了要求：“Use proper access modifiers (public, private, etc.) for your fields and methods. If a field/method can be private, it should be private.”而三个函数均可为private，JUnit只能直接测试public函数。在询问了TA之后选择在查找解决方案。最终选择使用Java反射的语言特性进行测试，核心代码如下所示：

Method addEdge = graph.getClass().getDeclaredMethod("addEdge", Person.class, Person.class);

addEdge.setAccessible(true);

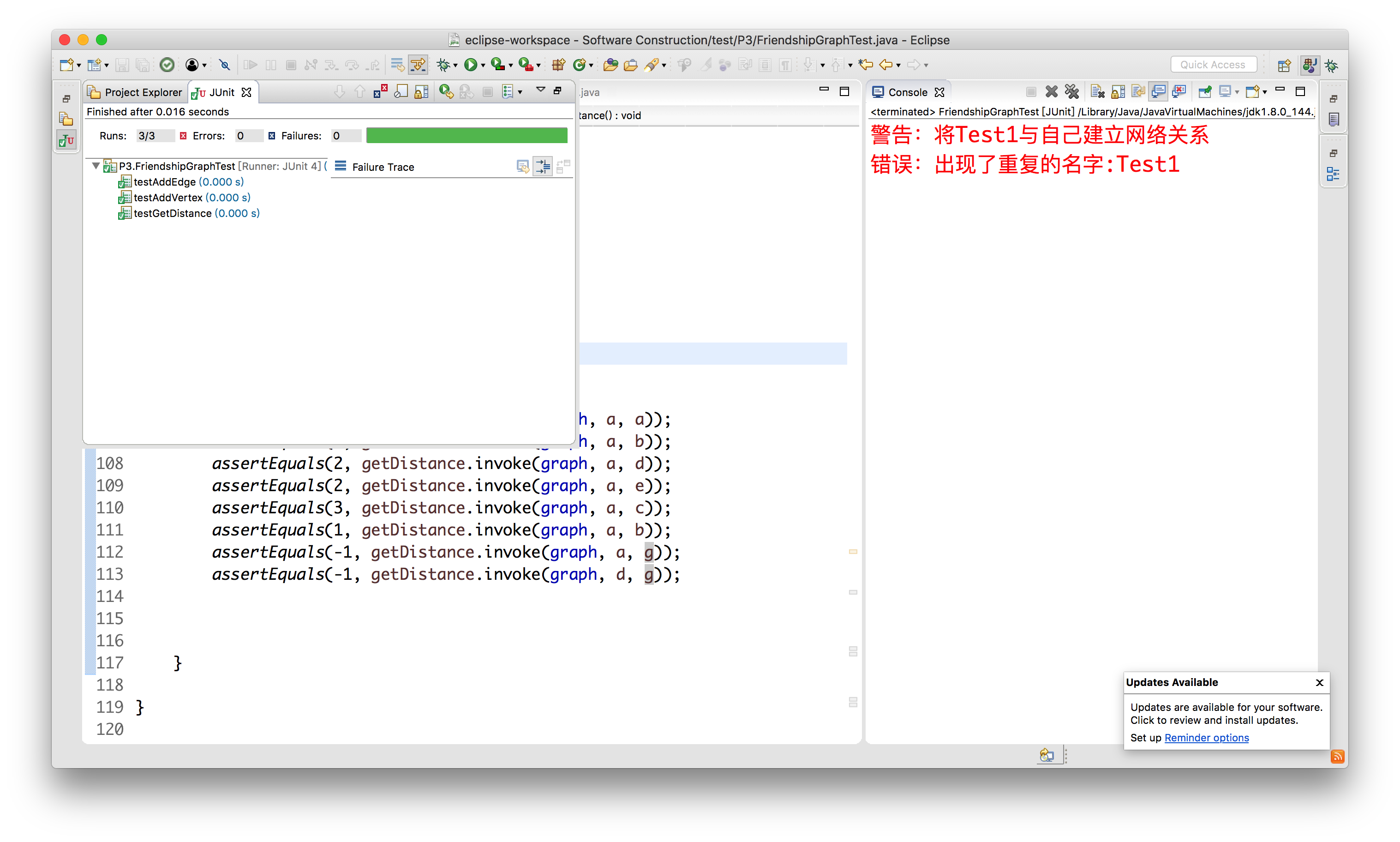
Person p1 = new Person("Test1");

addEdge.invoke(graph, p1, p1);// 调用

测试要点：

* addVertex 应测试特殊情况：增加同名的person
* addEdge 应测试特殊情况：与自己建立关系
* getDisance 自创一个非连通图进行测试

测试结果：



## Tweet Tweet

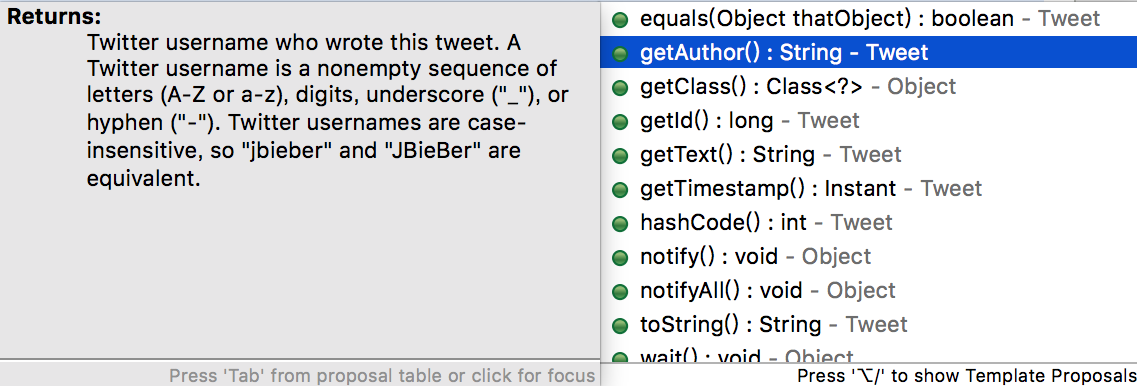
### Problem 1: Extracting data from tweets

这个问题的需求分为两部分：

* 获取这段数据的跨越的时间
* 获取所有在推文中出现的被提及的名字

获取timespan非常简单，调用Instant对时间进行解析后遍历tweets即可。通过isAfter和isBefore可得出最早和最迟时间。

twitter用户名有以下要求：



用户名应该是非空的字母、数字、“-”和“\_”的集合。且用户名大小写不敏感。

在推文中出现的被提及的名字要求如下：“The username-mention cannot be immediately preceded or followed by any character valid in a Twitter username。”即用户名提及不能在前或在后紧跟着在用户名中有效的字符。故在这里使用Java中的正则表达式进行提取。

考虑特殊情况，分别使用两个RE进行提取。

**"[^A-Za-z0-9\_-]@[\\w|-]+"**：其中[^A-Za-z0-9\_-]表示在用户名中有效的字符不应出现在“@”前。“\w”表示匹配任何字类字符，包括下划线。与“[A-Za-z0-9\_]”等效。[\\w|-]+表示出现多个字母或数字或“-”或“\_”。

**"^@[\\w|\_|-]+"**表明匹配@后的多个字母或数字或“-”或“\_”。这个正则表达式用来匹配位于推文最开始的username-mention。因为推文最开始的username-mention前面没有空格和其他字符，所以会被上面的RE遗漏。

使用RE进行匹配的核心代码如下：

Pattern pattern = Pattern.compile("[^A-Za-z0-9\_-]@[\\w|-]+");

for (Tweet tweet : tweets) {

Matcher matcher = pattern.matcher(tweet.getText());

while (matcher.find()) {

mentionedUsers.add(matcher.group().toLowerCase());

}

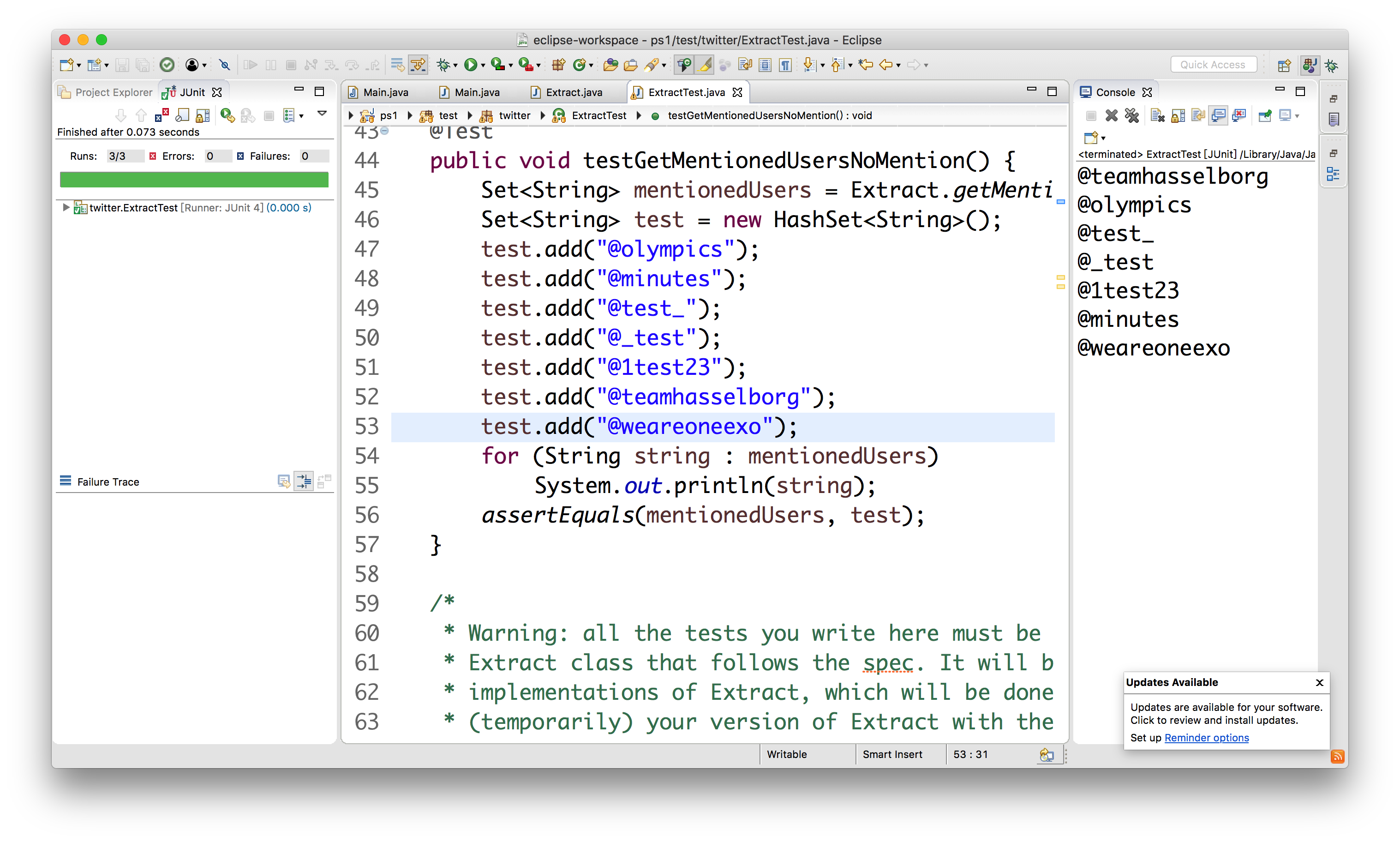
}

其中find方法返回第一个匹配到的RE，之后每次调用find都返回下一个。g由于用户名大小写不敏感，所以将用户名中的字母全部转化为小写字母后并进行trim处理后加入Set。

编写测试用例时应考虑到多种情况：

* 用户名以字母、数字、“-”“\_”开头／结尾
* 用户名前／后跟随有效字符
* 用户名前／后跟随无效字符
* 用户名前无空格
* 重复出现的大小写不一样而字母相同的用户名
* @后为空

测试结果：



### Problem 2: Filtering lists of tweets

这个问题的需求分为三部分：

* 给定timespan获取在timespan范围内的推文
* 给定用户名，找出此用户发布的所有twitter
* 给定关键词吗，寻找所有至少包含一个关键词的推文

inTimespan很好实现，仅需使用Instant类解析时间然后比较EpochSecond，将所有在timespan中的tweet加入list返回即可。

编写测试用例时应注意测试timespan之外的tweet是否在集合内。

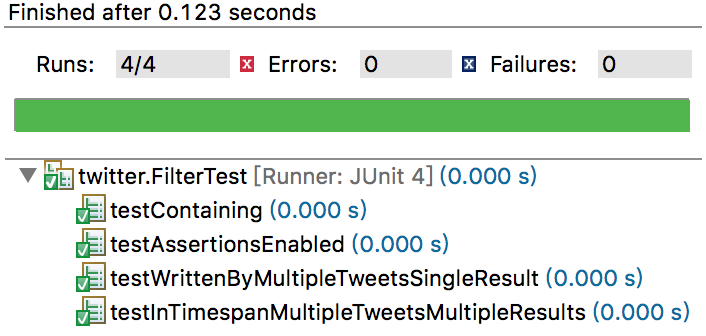
writtenBy同样很好实现，仅需获得tweet的作者后与给定字符串比较即可。最后返回包括所有tweet的list。

编写测试用例时应注意测试同一作者的多个推文是否全都加入了集合、若不存在作者集合是否空。

containing需要利用RE，在所有tweet中进行匹配，返回list集合。

其中正则匹配模式为"\\b"+word.toLowerCase()+"\\b"。\b表示匹配一个字边界，即字与空格间的位置示。由于匹配对大小写不敏感，所以一律把它们转化为小写字母进行比较。

编写测试用例时应注意测试如果目标单词在推文中是子字符串，那么它是否会被加入到集合。如目标单词“tract”，推文中如果有“contract”时应匹配失败。



### Problem 3: Inferring a social network

这个问题的需求分为两部分：

* 默认认为作者关注了他提及的人，根据这个基础猜测社交网络：根据推文提取作者和作者提及的人。社交网络由map<string、set<string>>表示，其中map[A]是个人A在twitter上关注的一组人，所有人都由其twitter用户名表示。
* 根据猜测的社交网络统计关注数，并根据关注数排序。

实现猜测社交网络：调用Problem 1中的getMentionedNames获得用户提及的人，若提及的人的Set包括自己，除去自己后加入map。同时应注意大小写处理。核心代码如下：

Map<String, Set<String>> followsGraph = new HashMap<String, Set<String>>();

for (Tweet tweet : tweets) {

Set<String> mentionedNames = Extract.getMentionedUsers(Arrays.asList(tweet));

String author = tweet.getAuthor().toLowerCase();

followsGraph.put(author, mentionedNames);

}

实现统计关注数：使用Map<String, Integer>储存用户的关注数，String代表用户名，Integer为int包裹类型，代表用户的followers数量。使用Map.Entry对followsGraph进行遍历，对于所有被提及且在映射中作为键存在的名字都对其统计关注者数量。

Map<String, Integer> followers = new HashMap<String, Integer>();

for(Map.Entry<String, Set<String>> entry:followsGraph.entrySet()){

for (String name : entry.getValue()) {

followers.put(name, n+1);

}

}

实现排序：将Map.Entry组织为一个list后，对list自定义排序。list的排序中使用到了Collections.sort函数，参数为排序的list对象和和排序方法Comparator对象。在这里自定义Comparator的方式比较特殊：直接在函数的参数部分新建匿名对象后对其compare方式进行重写，简化了代码部分。核心代码如下：

List<Map.Entry<String,Integer>> list = new ArrayList<Map.Entry<String, Integer>>(followers.entrySet());

Collections.sort(list,new Comparator<Map.Entry<String, Integer>>(){

@Override

public int compare(Entry<String, Integer> o1, Entry<String, Integer> o2) {

if (o1.getValue() > o2.getValue()) return -1;

else return 1;

}

});

编写测试用例时应考虑到以下特殊情况：

- 用户无法follow自己

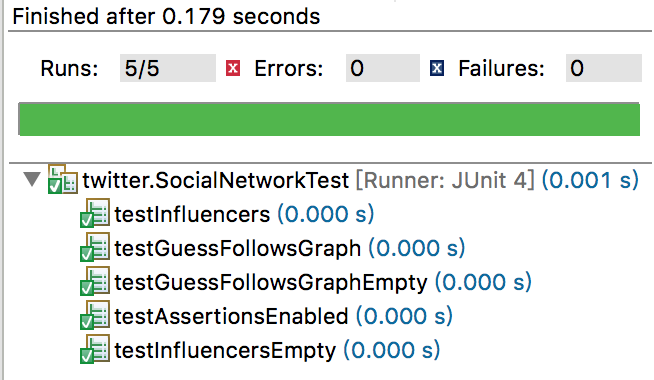
- guessFollowsGraph中测试同样字符但大小写表示不同的用户名是否被统计或重复

- 如果A不跟随任何人，那么map[A]可能是空的集合或A不作为映射中的键存在；即使A在网络中有关注者也同样成立。测试时应注意符合条件的A是否在map中。

- Twitter用户名不区分大小写，实现功能时一律将其转化为小写。

- 用户名最多应在映射或任何给定映射[A]集中作为键出现一次。

测试结果如下：



### Problem 4: Get smarter

这里选择Triadic closure思路实现：如果AB间存在关系，BC间也存在关系，那么AC间可能也存在某种关系。具体实现思路与方法如下：

若A关注B，B关注C，那么A也可能关注C。所以在guessFollowsGraph中遍历map，对map中出现的每一个key值A，遍历A所关注的人集合B，再遍历B所关注人集合C。将集合C中的所有人都加入A的关注中即可。但是在这里由于迭代器线程锁定，所以需要用一个list暂存变化，待迭代结束后更改内容。

for (Map.Entry<String,Set<String>> entry:followsGraph.entrySet()) {

List<String> temp = new ArrayList<>();

for (String string : entry.getValue()) { //遍历A的关注B

if (followsGraph.containsKey(string)) {//遍历B的关注C

for (String string1 : followsGraph.get(string)) {

if (!entry.getKey().equals(string1))

temp.add(string1);//将B关注的人加入到A的关注中

}

}

}

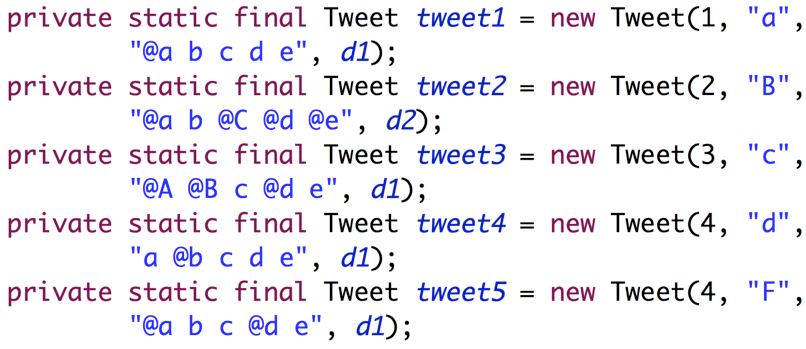
for (String string : temp) {

entry.getValue().add(string);

}

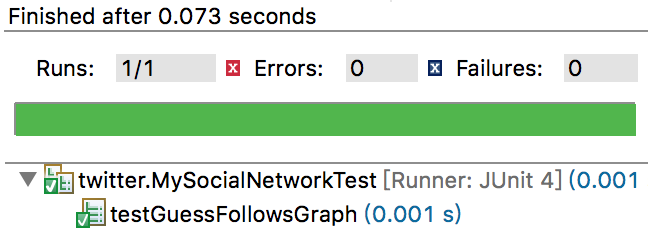
}

在MySocialNetworkTest中进行测试，测试用例为



其中d仅关注b，在添加Triadic closure之前关注者人数为1，添加后测试d的关注者人数为4，符合预期。

总测试结果如下：



# 实验进度记录

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 任务 | 实际完成情况 |
| 2019-02-25 | 14:15-15:30 | 编写问题1的isLegalMagicSquare的大部分内容 | 大致按计划完成 |
| 2019-02-25 | 16:50-15:25 | 完成编写isLegalMagicSquare函数并进行测试 完成相关部分的实验报告 | 完成 |
| 2019-02-25 | 20:55-22:55 | 完成问题1的所有问题和实验报告 绘制流程图 上传代码 | 完成 |
| 2019-02-26 | 19:00-21:30 | 完成问题2的任务1-5 、配置JUnit | 未按计划完成 |
| 2019-02-26 | 22:00-23:30 | 完成问题2的任务6并debug | 未按计划完成 |
| 2019-02-27 | 10:00-11:05 | 完成测试问题2中的convexHells函数 | 完成 |
| 2019-02-27 | 15:45-17:45 | 完成问题3的代码部分 | 完成 |
| 2019-02-28 | 19:00-21:30 | 完成问题3的测试部分 | 完成 |
| 2019-03-01 | 14:00-15:00 | 编写问题4-1的大部分内容 | 大致完成 |
| 2019-03-01 | 18:00-22:30 | 完成问题4-2及测试 | 完成 |
| 2019-03-02 | 17:00-20:00 | 完成问题4-3及测试 | 完成 |
| 2019-03-02 | 20:20-21:30 | 完成问题4-4及测试 | 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 |
| JUnit测试private方法 | piazza上询问TA是否应该将private更改为public  查询CSDN：  <https://blog.csdn.net/iameyama/article/details/50411212>  <https://blog.csdn.net/qq_34566601/article/details/74990357> |
| 测试RE 获得RE代码 | 利用在线RE测试工具：<http://tool.oschina.net/regex/>  查阅教程：  <http://www.runoob.com/java/java-regular-expressions.html> |
| map的遍历方法 | <https://blog.csdn.net/yueaini10000/article/details/78933289> 使用迭代器进行迭代 |
| arraylist的排序 | <https://blog.csdn.net/u011299745/article/details/52654023> 使用Collection.sort，重写Comparable方法 |
| java.util.ConcurrentModificationException | <https://www.cnblogs.com/frankliiu-java/articles/1759460.html>在个人博客上了解错误出现的原因，清楚原理后选择用list暂存变化，等到迭代结束时开始改变map。 |

# 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

- 在完成问题4时感觉对很多英语表述的理解都很模糊，应该继续加强英语学习，适应阅读英文文档

- 接触了一些之前没有接触到的Java语法细节与应用

- 第一次自定义稍微复杂一点的RE进行匹配，感受到了正则表达式的强大

- 第一次使用JUnit，接触代码测试

- 发现解决Turtle问题时涉及到的数学知识遗忘的很厉害，耽误了很多时间

## 针对以下方面的感受

1. Java编程语言是否对你的口味？

超级喜欢这门OO语言，强类型+静态语言+垃圾收集器+超多超方便的封装（ArrayList、Set、Map）好感++。

1. 关于Eclipse IDE

很喜欢这个IDE。界面美观大方功能强大，比CB稳定，比Xcode轻便。用起来非常顺手。

1. 关于Git和GitHub

因为以前在学长的建议下之前已经配齐Git和github，所以这次试验用起来就非常方便而且对它们也比较熟悉了。平时在环境配置经常用到git clone，github上有非常多有用或有意思的项目，在这门课接触这两个“程序员必备”感觉非常实用。

1. 关于CMU和MIT的作业

从CSAPP这门课开始就非常羡慕CMU的作业，每个都设计的非常有趣而且难度适中，作业量也还可以，文档描述也很清晰（只是我英语水平太差）。感觉他们的作业都是精心设计的，在已有的代码框架上进行补充和实现，而印象里HIT的作业都是要自己从0开始去写的。

1. 关于本实验的工作量、难度、deadline

因为之前接触过java+git+github所以这个实验写的还蛮快，一周内可以轻松搞定，难度也适中，ddl不紧。

1. 关于初接触“软件构造”课程

从接触Java的OOP开始就觉得“写代码”容易，但“写好代码”不是易事，一直都很好奇如何设计类的耦合程度之类的设计问题，所以对这门课充满了好奇与期待。