

**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 | 沈子鸣 |
| 学号 | 1170301007 |
| 班号 | 1703010 |
| 电子邮件 | [2508754153@qq.com](mailto:2508754153@qq.com) |
| 手机号码 | 18800421860 |

**目录**

[1 实验目标概述 1](#_Toc3320606)

[2 实验环境配置 1](#_Toc3320607)

[3 实验过程 1](#_Toc3320608)

[3.1 Magic Squares 1](#_Toc3320609)

[3.1.1 isLegalMagicSquare() 3](#_Toc3320610)

[3.1.2 generateMagicSquare() 6](#_Toc3320611)

[3.2 Turtle Graphics 10](#_Toc3320612)

[3.2.1 Problem 1: Clone and import 11](#_Toc3320613)

[3.2.2 Problem 3: Turtle graphics and drawSquare 12](#_Toc3320614)

[3.2.3 Problem 5: Drawing polygons 13](#_Toc3320615)

[3.2.4 Problem 6: Calculating Bearings 15](#_Toc3320616)

[3.2.5 Problem 7: Convex Hulls 18](#_Toc3320617)

[3.2.6 Problem 8: Personal art 22](#_Toc3320618)

[3.2.7 Submitting 23](#_Toc3320619)

[3.3 Social Network 26](#_Toc3320620)

[3.3.1 设计/实现FriendshipGraph类 26](#_Toc3320621)

[3.3.2 设计/实现Person类 30](#_Toc3320622)

[3.3.3 设计/实现客户端代码main() 31](#_Toc3320623)

[3.3.4 设计/实现测试用例 31](#_Toc3320624)

[3.4 Tweet Tweet 32](#_Toc3320625)

[3.4.1 Problem 1: Extracting data from tweets 32](#_Toc3320626)

[3.4.2 Problem 2: Filtering lists of tweets 34](#_Toc3320627)

[3.4.3 Problem 3: Inferring a social network 37](#_Toc3320628)

[3.4.4 Problem 4: Get smarter 39](#_Toc3320629)

[4 实验进度记录 40](#_Toc3320630)

[5 实验过程中遇到的困难与解决途径 41](#_Toc3320631)

[6 实验过程中收获的经验、教训、感想 42](#_Toc3320632)

[6.1 实验过程中收获的经验和教训 42](#_Toc3320633)

[6.2 针对以下方面的感受 42](#_Toc3320634)

# 实验目标概述

本次实验通过求解四个问题，训练基本Java编程技能，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需求补全代码，能够为所开发的代码编写基本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。

* 基本的 Java OO编程
* 基于Eclipse IDE进行Java编程
* 基于JUnit的测试
* 基于Git的代码配置管理

# 实验环境配置

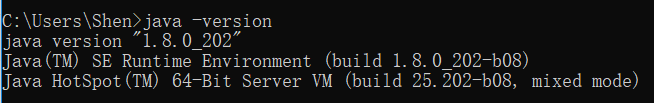
硬件环境：Intel Core i5-7200U，64位; 4G;

软件环境：Windows 10家庭中文版

IDE：Eclipse Java 2018-12

关于Java：JDK 8, JRE 8

关于Git：Git 2.20.1





我的实验环境在假期时就已经配置好了，参考了《Java轻松学》和《GitHub入门与实践》，配置过程中并未遇到印象深刻的问题。

GitHub Lab1仓库的URL地址：

https://github.com/ComputerScienceHIT/Lab1-1170301007

# 实验过程

## Magic Squares

首先题目给出了幻方的概念，这里的幻方被认为是一个由整数组成的n×n方阵，对每一行，每一列，两条对角线上的n个元素分别求和，结果均相等。

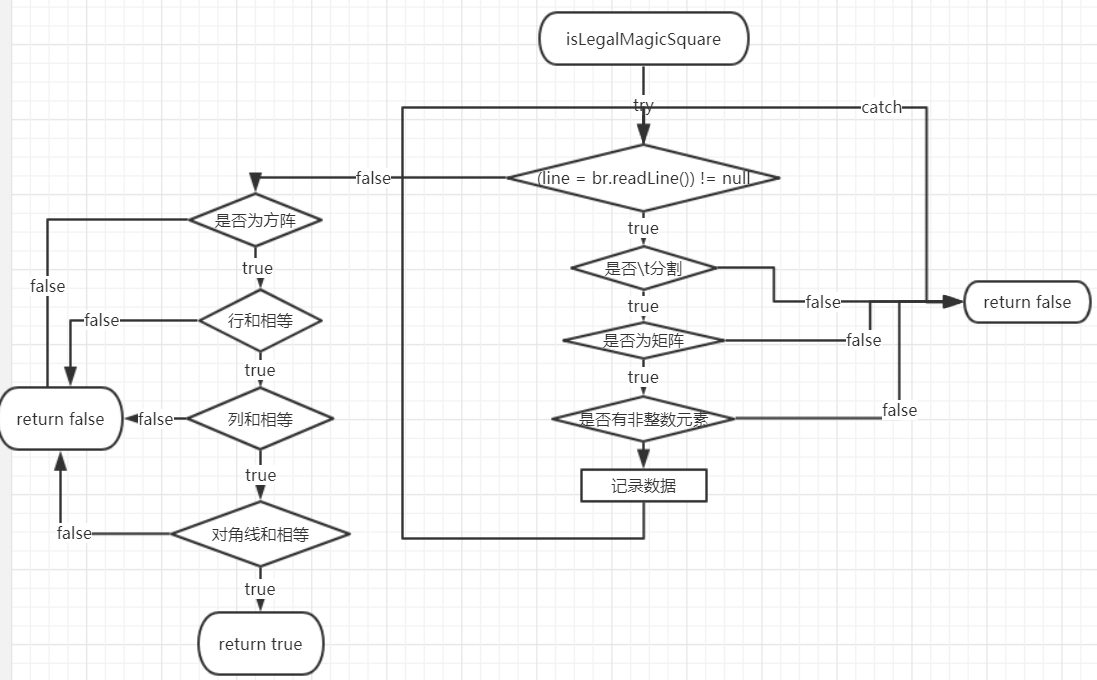
该实验要求写一个名为MagicSquare.java的Java程序，检查一个矩阵的行和、列和，以及对角线和，以此来判断给定矩阵是否为幻方。

* 实验给定了5个txt文档文本，每个txt文件中存储一个矩阵信息，这些文件要放到指定路径下：\src\P1\txt。
* 对每个文件，打开文件并检查各行的和的值相等，各列的和值相等，两条对角线的和相等。
* 返回一个boolean类型的结果来判断输入矩阵是否为幻方。
* 函数规约：boolean isLegalMagicSquare(String fileName)
* 在main()函数中调用五次isLegalMagicSquare()函数，将5个文本文件名分别作为参数输入进去，看其是否得到正确的输出（true, false）。
* 需要能够处理输入文件的各种特殊情况，例如：文件中的数据不符合Magic Square的定义（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。若遇到这些情况，终止程序执行（isLegalMagicSquare函数返回false），并在控制台输出错误提示信息。

要求二，是阅读理解给定函数代码，放到MagicSquare类中，在main()中测试它，还要加上注释，分析某些异常，修改代码以处理这些异常，还要把生成的幻方矩阵写入文件\src\P1\txt\6.txt中。

最后，按照项目目录要求，检查代码后，用git指令上传到远程仓库中。

### isLegalMagicSquare()



要判断是否为幻方，就要对各行、各列、两条对角线求和，可以用一个二维数组来存储矩阵元素，而在不知道数组的行列的情况下，决定使用嵌套List来存储矩阵元素。

1. ArrayList<ArrayList<Integer>> square = **new** ArrayList<ArrayList<Integer>>();

函数主体用一个大的try-catch结构，以防止意想不到的异常出现。

声明一些必要的变量后，开始读取文件，用while结构，不断读取矩阵的行，每次读取对这一行做分析处理。比如说，由于我们用了List嵌套结构存储矩阵矩阵，所以每读取一行时，要以List的形式存到变量square中。

对这一行还要进行分析。我们希望对所有不合法情况考虑完之后，再去判断它是不是幻方。用split方法用\t为标准分割行字串，分割结果是一个String类型数组，对这个数组中每个字串分析，这里每个字串按理说应该是整数字串，否则的话，我们要对其进行异常处理。比如说，如果字串中含有空格，我们就可以认为读取该行时，存在没有用\t分割的情况

1. **for**(String s : splitedLine) {
2. **if**(s.contains(" ")) {
3. System.out.println("ERROR! For " + fileName + ".There is a space!");
4. **return** **false**;
5. }
6. }

上述考虑了是否\t分割，接下来考虑每行元素个数是否相等，这决定了它是否一个矩阵。

1. /\* 判断是否为矩阵 \*/
2. **if**(rowN != 0 && splitedLine.length != columnN) {
3. System.out.println("ERROR! For " + fileName + ".The number of each row is not the same!");
4. **return** **false**;
5. }
6. columnN = splitedLine.length;   // 这里是用每行元素数去和上一行比较是否相等

接下来判断是否有非整型元素。根据Java API文档，Integer.valueOf()方法在处理一个不能被转换为整型的字串时会抛出一个异常，用try-catch接住这个异常，从而处理它。

1. /\* 接下来判断是否有非整数元素 \*/
2. **for**(**int** i=0; i<splitedLine.length; i++) {
3. **try** {
4. row.add(Integer.valueOf(splitedLine[i]));
5. } **catch** (Exception e) {
6. System.out.println("ERROR! For " + fileName + ".There is a non-integer number!");
7. **return** **false**;
8. }
9. }

最后，确认在读数据时没有其他意料之中的异常后，记录数据。

1. square.add(row);    // 在读文件的过程中，没有问题，即可保存为一行
2. rowN++; // 记录行数

然后退出while循环，至此，数据读取工作结束。

在判断幻方之前，还有一个异常要解决，就是该矩阵是不是方阵，用我们的行列计数器可以判断得到。

1. /\* 判断是否为方阵 \*/
2. **if**(rowN != columnN) {
3. System.out.println("ERROR! For " + fileName + ".It's not a square!");
4. **return** **false**;
5. }

接下来就可以开始判断是否为幻方了。也就是求和工作。首先声明两组变量。

1. /\* 求和过程中的正确和值 \*/
2. **int** rowRightSum = 0, colRightSum = 0, diaRightSum = 0;
3. /\* 求和过程中各行实际和值 \*/
4. **int** rowSum = 0, colSum = 0, diaSum = 0;

然后判断行和是否相等。

1. /\* 判断各行和是否相等 \*/
2. **for**(**int** i=0; i<rowN; i++) {
3. /\* 出现不相等的时刻，直接否定幻方 \*/
4. **if**(i>0 && rowRightSum!=rowSum) {
5. System.out.println("ERROR! For " + fileName + ".The sum of each row is not the same!");
6. **return** **false**;
7. }
8. rowSum = 0; // 和值归零
9. /\* 每行求和 \*/
10. **for**(**int** j=0; j<columnN; j++)
11. rowSum += square.get(i).get(j);
12. rowRightSum = rowSum;
13. }

判断列和是否相等。

1. /\* 判断各列和是否相等 \*/
2. **for**(**int** j=0; j<columnN; j++) {
3. /\* 出现不相等的时刻，直接否定幻方 \*/
4. **if**(j>0 && colRightSum!=colSum) {
5. System.out.println("ERROR! For " + fileName + ".The sum of each column is not the same!");
6. **return** **false**;
7. }
8. colSum = 0; // 和值归零
9. /\* 每列求和 \*/
10. **for**(**int** i=0; i<rowN; i++)
11. colSum += square.get(i).get(j);
12. colRightSum = colSum;
13. }

判断对角线和是否相等，这部分简单一些。

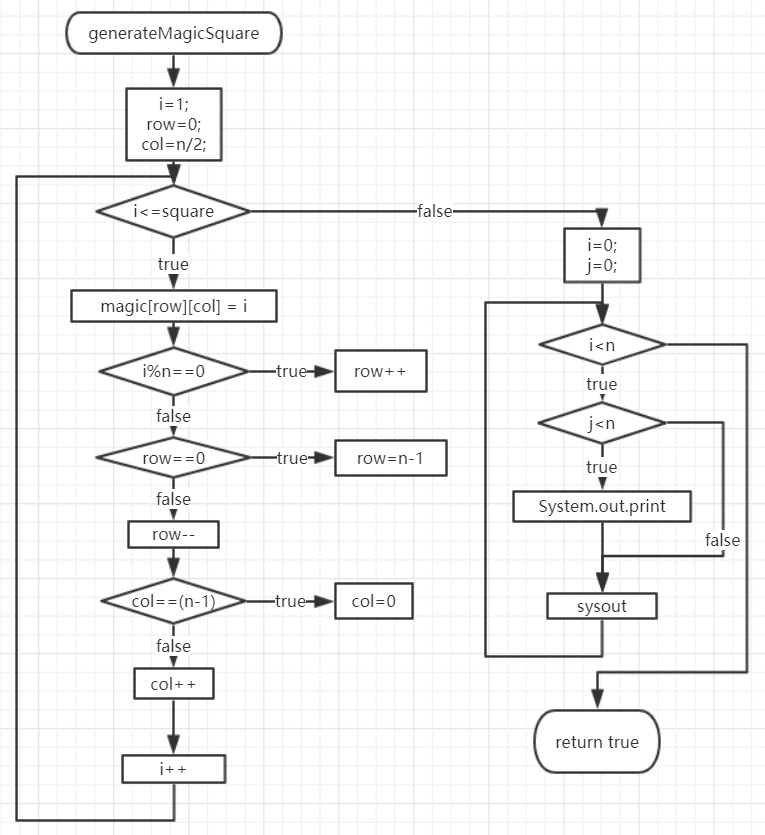
1. /\* 判断对角线和是否相等 \*/
2. **for**(**int** i=0; i<rowN; i++) {
3. diaRightSum += square.get(i).get(i);
4. diaSum += square.get(rowN-i-1).get(columnN-i-1);
5. }
6. /\* 如果不等，否定幻方 \*/
7. **if**(diaRightSum != diaSum) {
8. System.out.println("ERROR! For " + fileName + ".The sum of each diagonal is not the same!");
9. **return** **false**;
10. }

在isLegalMagicSquare函数的最后，返回true，说明在执行过程中，并未遇到我们意想中的异常情况发生，也没有否定幻方，那么就认为它是幻方了。

### generateMagicSquare()

按步骤给出你的设计和实现思路/过程/结果。

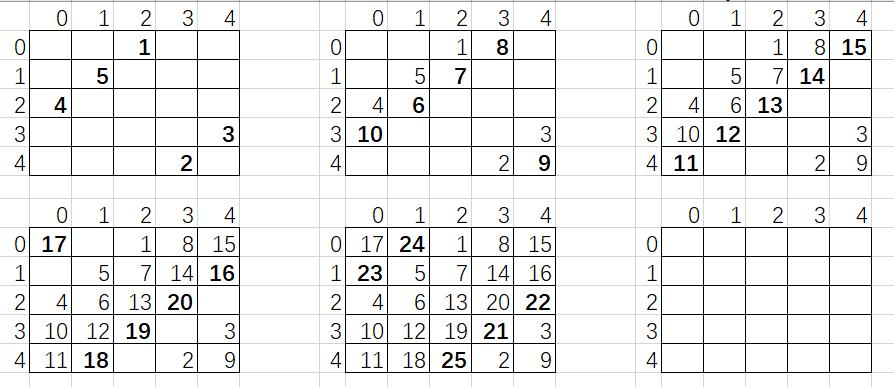
1. 程序流程图



1. 算法解释描述

这个算法不容易想出来，看了好久才明白为什么可以生成奇数阶的幻方。

这个算法其实是在用螺旋式填充的方法构造幻方，为了形象地描述，用图来说明5×5的幻方构造过程。



如上图所示，这里实际上循环了25次，但我把它们分为5组，每组5次循环，每组即填满一次螺旋。

该算法的核心是：**行触底反弹，列触顶反弹，向右上方向螺旋填充。**

该算法的每组螺旋，从底行中间列开始，向右上方向填充，遇到底行就反弹到顶行，遇到顶列，就反弹到底列，填充n次为一组。每组填充后，行数减一，开启下一次螺旋填充。

这种填充构造方式，首先可以保证n×n个数有规律地填满，其次，可以保证它是幻方。幻方可以这样证明：

* + - 对于行和相等。行数为n，螺旋填充n次，可以保证，第i行的5个数中总有4个数是比第i+1行大1的，而总有1个数是比第i+1行小4的，这样，第i行和第i+1行的和是相等的。
    - 对于列和相等。可以把该矩阵顺时针转90度，按原来的填充模式，可以看作是向左上方螺旋填充，同行和相等的证明。
    - 对于对角线和相等。这点没能给出严谨的证明，只知道对角线上的值是关于中间值对称的，比如5+21=13\*2，17+9=13\*2.所以对角线和相等是中间值的五倍。这点证明用n阶的范式可以看出来。

1. 原代码以及中文注释
2. **public** **static** **boolean** generateMagicSquare(**int** n) {
4. **int** magic[][] = **new** **int**[n][n];  // n阶方阵
5. /\* 初始化，第0行，取中间列，原代码中n取偶数或负数都会出现异常 \*/
6. **int** row = 0, col = n / 2, i, j, square = n \* n;
7. /\* 循环n\*n次，向矩阵中添加元素 \*/
8. **for** (i = 1; i <= square; i++) {
9. /\* 这里面用步长为1的循环来填充方阵，按下面的填充规律，可以实现一个magic square \*/
10. /\* 按下述if-else模式填充，填充模式是螺旋状的 \*/
11. magic[row][col] = i;
12. /\* 以n为步长，循环填充 \*/
13. **if** (i % n == 0)
14. row++;
15. **else** {  // 每轮填充过程中
16. **if** (row == 0)   // 行触底反弹，回到末行
17. row = n - 1;
18. **else**
19. row--;  // 行递减
20. **if** (col == (n - 1)) // 列触顶反弹，回到首行
21. col = 0;
22. **else**
23. col++;  // 列递增
24. }
25. }
27. **for** (i = 0; i < n; i++) {
28. **for** (j = 0; j < n; j++)
29. System.out.print(magic[i][j] + "\t" );
30. System.out.println();
31. }
32. **return** **true**;
33. }
35. 原代码中的异常分析

原代码中，在n为偶数时会抛出java.lang.ArrayIndexOutOfBoundsException异常，在n为负数时会抛出java.lang.NegativeArraySizeException异常，查询Java API文档，有如下解读。

public class **IndexOutOfBoundsException**

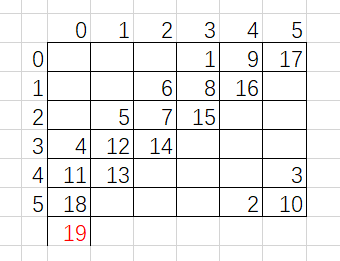
extends [RuntimeException](https://docs.oracle.com/javase/8/docs/api/java/lang/RuntimeException.html)

Thrown to indicate that an index of some sort (such as to an array, to a string, or to a vector) is out of range.

就是数组索引越界了。具体原因如下图，当n=6时。会有一个赋值导致该语句中的定长数组索引越界，抛出异常。

1. magic[row][col] = i;

如图



当n为负数时，则会在声明int magic[][]变量时就引发异常，查阅Java API文档。

public class **NegativeArraySizeException**

extends [RuntimeException](https://docs.oracle.com/javase/8/docs/api/java/lang/RuntimeException.html)

Thrown if an application tries to create an array with negative size.

定长数组声明时，数组长度决不能是负数。

1. 扩展后的generateMagicSquare()
2. **public** **static** **boolean** generateMagicSquare(**int** n) {
3. /\* 处理n为负数的异常 \*/
4. **if**(n<0) {
5. System.out.println("The arraysize must be non-negative!");
6. **return** **false**;
7. }
8. **int** magic[][] = **new** **int**[n][n];  // n阶方阵
9. /\* 初始化，第0行，取中间列，原代码中n取偶数或负数都会出现异常 \*/
10. **int** row = 0, col = n / 2, i, j, square = n \* n;
11. /\* 循环n\*n次，向矩阵中添加元素 \*/
12. **for** (i = 1; i <= square; i++) {
13. /\* 这里面用步长为1的循环来填充方阵，按下面的填充规律，可以实现一个magic square \*/
14. /\* 按下述if-else模式填充，填充模式是螺旋状的 \*/
15. /\* 处理n为偶数引发的索引越界异常 \*/
16. **try** {
17. magic[row][col] = i;
18. } **catch** (Exception e) {
19. System.out.println("There may be some index exception here!");
20. **return** **false**;
21. }
22. /\* 以n为步长，循环填充 \*/
23. **if** (i % n == 0)
24. row++;
25. **else** {  // 每轮填充过程中
26. **if** (row == 0)   // 行触底反弹，回到末行
27. row = n - 1;
28. **else**
29. row--;  // 行递减
30. **if** (col == (n - 1)) // 列触顶反弹，回到首行
31. col = 0;
32. **else**
33. col++;  // 列递增
34. }
35. }
37. **for** (i = 0; i < n; i++) {
38. **for** (j = 0; j < n; j++)
39. System.out.print(magic[i][j] + "\t" );
40. System.out.println();
41. }
43. /\* 扩展：写入文件/src/P1/txt/6.txt \*/
44. **try** {
45. File newSquare = **new** File("src/P1/txt/6.txt");
46. newSquare.createNewFile();
47. BufferedWriter writer =  **new** BufferedWriter(**new** FileWriter(newSquare));
48. **for**(i=0; i<n; i++) {
49. **for**(j=0; j<n; j++) {
50. writer.write(magic[i][j] + "\t");
51. }
52. writer.write("\n");
53. }
54. writer.flush();
55. writer.close();
56. } **catch** (Exception e) {
57. e.printStackTrace();
58. }
59. **return** **true**;
60. }

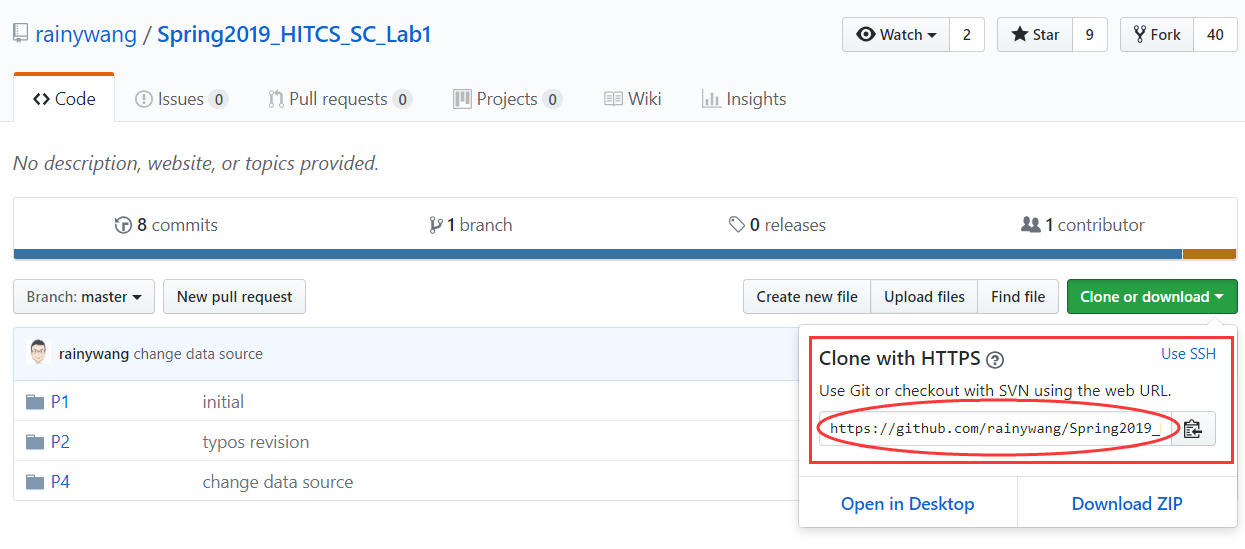
## Turtle Graphics

这个任务是要完成MIT 6.031的Problem Set 0: Turtle Graphics. 全部做下来之后，考察了我们阅读英文规约文档，按照要求完成函数，做出了一只模拟小乌龟，还有二维平面上的一些几何问题。这其中重点强化了我们的OOP编程，代码的复用，实施起来遇到的实际问题太多太多，还提高了我们解决问题的能力。幻方是面向过程变成，这次是面向对象编程，考察Java基本编程更加全面。

### Problem 1: Clone and import

关于Git开发参考了《GitHub入门与实践》。

先按照实验环境声明创建远程仓库，然后在本地某个目录下打开Git Bash开始clone任务代码。本次实验所需代码克隆路径如下：



记录下这段HTTPS路径（或SSH路径），在相应目录下Git Bash中clone。clone前要先使用命令

1. git init

以初始化git仓库，然后clone

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

然后任务相关代码就到了该目录下。

这里有可能遇到clone速度慢的情况，可以参考个人总结博客<https://blog.csdn.net/qq_41662115/article/details/86993426>.

本地Git管理开发主要有文件的上传和提交、远程push和远程pull等操作。

文件的上传命令为git add *xx*，参数为文件名，git add .即为将当前目录下的所有文件上传，这里仅将文件上传到暂存区，距离真正的提交还差一条git commit命令，这条命令可以加-m *string*参数，*string*是一条字串，用来表示本次提交的信息。

远程push和pull通常要先使用如下命令，来将远程仓库用origin标识符标记：

1. git remote add origin https://github.com/rainywang/Spring2019\_HITCS\_SC\_Lab1.git

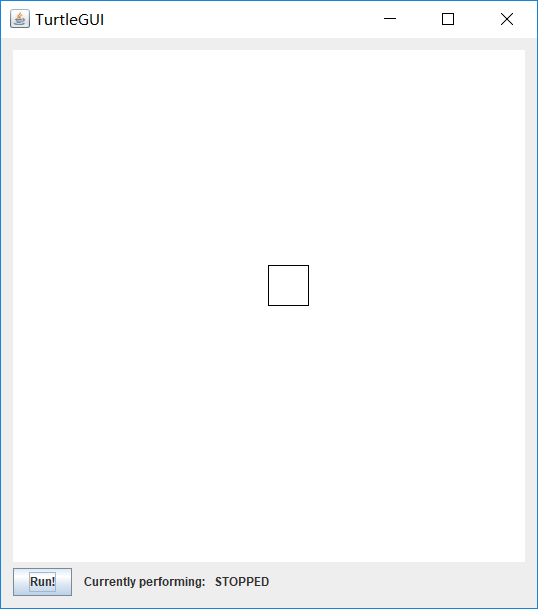
这样用origin可以来表示远程仓库。将本地文件上传可以用git push origin master命令，即将本地文件上传到了远程仓库的origin分支。后续的push用git push命令也可以了。Pull操作的命令是git pull origin master.

### Problem 3: Turtle graphics and drawSquare

这个问题内容是让我们完善TurtleSoup.java中的drawSquare(Turtle turtle, int sideLength)，要求用Turtle.java中的forward(units)和turn(degrees)来做。结果应该是要在屏幕上绘制一个正方形。那问题很简单了，代码如下：

1. **public** **static** **void** drawSquare(Turtle turtle, **int** sideLength) {
2. **for** (**int** i = 0; i < 4; i++) {
3. turtle.forward(sideLength);
4. turtle.turn(90);
5. }
6. }

结果截图：



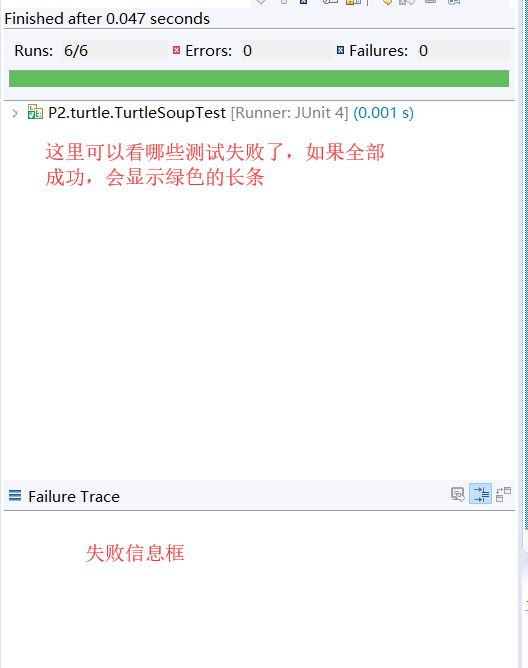
### Problem 5: Drawing polygons

这部分是完善TurtleSoup.java中的方法，不能修改已有方法的方法签名，但可以在TurtleSoup.java中添加自己定义的辅助方法，还可以在TurtleSoupTest.java中添加一些额外测试。

先完善方法calculateRegularPolygonAngle，这是要求任意一个正多边形的内角，用相应的数学公式非常简单，内角和=180\*（边数-2），代码如下：

1. **public** **static** **double** calculateRegularPolygonAngle(**int** sides) {
2. **double** angle = 180 - (**double**) 360 / sides;
3. **return** angle;
4. }

6.031中还介绍了JUnit的基本使用。以JUnit test运行TurtleSoupTest.java就可以来测试TurtleSoup.java了，运行时Eclipse左侧会出现下图所示的工作区。



JUnit test的正确使用方式是在测试方法中测试一些案例，尽可能地让功能函数报错，以期待找到一些bug，这样帮助我们寻找bug，修改bug，完善代码。

这部分问题的另外一个要求是完善drawRegularPolygon方法，这个方法的要求和calculateRegularPolygonAngle的恰恰相反，这是要给定内角，求的对应正多边形的边数。

上一个问题的解决方法，我们小学的时候就学过公式了，设内角度数为α，边数为n，那么有α = 180 × (n - 2) ÷n。这里面有一个问题是，n既然作为分母，结果难免会出现除不尽的情况，比如说正七边形，那么我们在Java中做浮点数除法时，得到的浮点数类型结果，必然不是精确值，但这种精确程度我们也可以接受了。

但是，在做这个问题的时候，我们要求边数n，那么结果是一个整数，我们会出现浮点数舍入到整数的情况。在没有指定舍入方法的情况下，默认是向零舍入，正数向下，负数向上，这样如果你在计算过程中得到的n是5.99...的话，那么结果就是5了，而不是应该舍入的6.用Math.round()方法解决这一问题，代码如下：

1. **public** **static** **int** calculatePolygonSidesFromAngle(**double** angle) {
2. **int** sides = (**int**) Math.round(360 / (180 - angle));
3. **return** sides;
4. }

完成这部分问题之后用Git上传提交一下，保存当前代码版本。

### Problem 6: Calculating Bearings

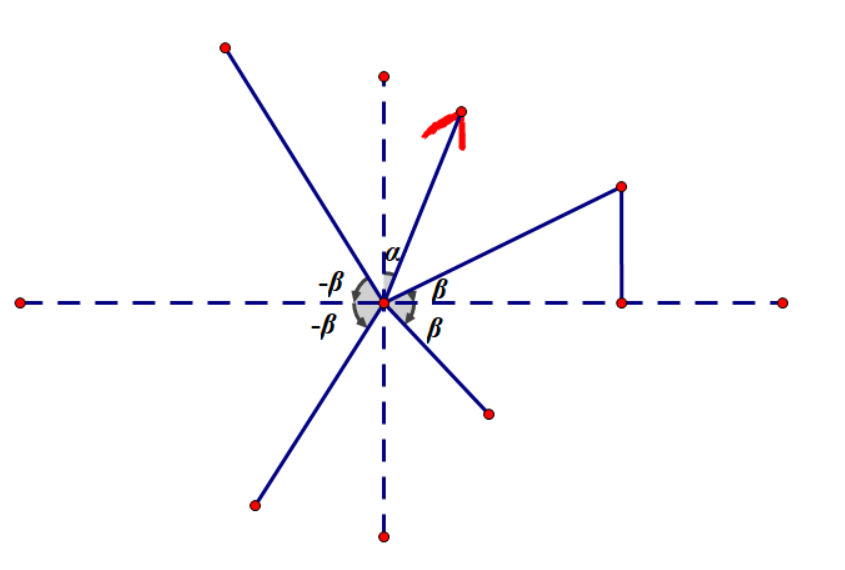
到这里开始进入到本次实验比较扎心的部分了，接下来的凸包问题更让人头疼。这部分要完成两个方法，一个是calculateBearingToPoint，是在给定一个向量和y正半轴夹角，该向量是小乌龟的面朝向，再给定小乌龟的二维坐标，和目的坐标，求小乌龟想要到达目的地前，最少需要顺时针旋转的角度。第二个方法是calculateBearings，这里简单考察了方法的复用，给定了两个列表，分别是n个点的x坐标和y坐标，从第一个点开始，假设面朝y正半轴，要求一个列表，该列表中有n-1个元素，第i个元素是从第i个点到达第i+1个点所要顺时针旋转的角度。

对于第一个方法，我在纸上对两给点的方位给出了8种情况的分析，最终合并8种情况为4种情况，两点互为上下或左右分别是两种情况，这种情况比较特殊，考虑起来很简单。再有就是两点并不互为上下左右，这样的话除了考虑当前点的朝向，计算旋转角度时还要考虑两点夹角，这里就把两点连线和水平线所夹的锐角作为参考，并认为这个锐角是-90°到90°之间的，记为β。这样的话，如果目标点在当前点的一三象限方位，那么β是正的，否则β是负的。合并四个方位发现，虽然一四象限和二三象限时的β正负各自不同，但是得到的结果是一样的，所以最后是四种大的情况。

这四种大的情况中还包含小的情况：

* 两点互为上下。
  + 目标点在上
    - 返回（360-α）%360
  + 目标点在下
    - α在0~180°
      * 返回180-α
    - α在180~360°
      * 返回540-α
* 两点互为左右。
  + 目标点在左
    - α在0~270°
      * 返回270-α
    - α在270~360°
      * 返回630-α
  + 目标点在右
    - α在0~90°
      * 返回90-α
    - α在90~270°
      * 返回450-α
* 目标点在左上或左下
  + α在0~270°
    - 270-α-β
  + α在270~360°
    - 630-α-β
* 目标点在右上或右下
  + α在0~90°
    - 90-α-β
  + α在90~270°
    - 450-α-β

附一张简单的角度图：



代码如下：

1. **public** **static** **double** calculateBearingToPoint(**double** currentBearing, **int** currentX, **int** currentY, **int** targetX,
2. **int** targetY) {
3. /\* 对于当前点和目标点的方位作分析，可以大致分为四种情况 \*/
4. /\* 该问题分析时本分为8种情况，根据结果对各个情况合并，得到四种情况 \*/
5. /\* 两点互为上下方位 \*/
6. **if** (currentX == targetX) {
7. **if** (currentY > targetY) // 目标点在下面
8. **return** (currentBearing >= 0 && currentBearing <= 180) ? (180 - currentBearing) : (540 - currentBearing);
9. **else** // 目标点在上面
10. **return** (360 - currentBearing) % 360; // 对360取余是因为如果currentBearing=0那么结果应该是0而不是360
11. }
12. /\* 两点互为左右方位 \*/
13. **if** (currentY == targetY) {
14. **if** (currentX > targetX) // 目标点在左侧
15. **return** (currentBearing >= 0 && currentBearing <= 270) ? (270 - currentBearing) : (630 - currentBearing);
16. **else** // 目标点在右侧
17. **return** (currentBearing >= 0 && currentBearing <= 90) ? (90 - currentBearing) : (450 - currentBearing);
18. }
19. /\* beta是当前点和目标点连线与水平线所夹锐角，取值范围由-90°到90° \*/
20. **double** beta = Math.atan((**double**) (currentY - targetY) / (currentX - targetX)) / Math.PI \* 180;
21. **if** (currentX > targetX) // 目标点在左上或左下
22. **return** (currentBearing >= 0 && currentBearing <= 270) ? (270 - currentBearing - beta)
23. : (630 - currentBearing - beta);
24. **else** { // 目标点在右上或右下
25. **return** (currentBearing >= 0 && currentBearing <= 90) ? (90 - currentBearing - beta)
26. : (450 - currentBearing - beta);
27. }
28. }

第二个方法只需要复用第一个方法的代码就好了，需要注意第一个方法的方法签名，除了需要给定当前点和目标点的坐标，还要给定当前朝向，而当前朝向是现有方法不能给出的，要自己更新，不过这个更新比求旋转角度更简单一些。代码如下：

1. **public** **static** List<Double> calculateBearings(List<Integer> xCoords, List<Integer> yCoords) {
2. List<Double> bearings = **new** ArrayList<Double>();
3. **double** currentBearing = 0; // 初始朝向为y正半轴
4. **for** (**int** i = 0; i < xCoords.size() - 1; i++) { // 结果列表中要有n-1个元素
5. /\* 复用calculateBearingToPoint方法来计算到下一个目标点的旋转角度 \*/
6. bearings.add(calculateBearingToPoint(currentBearing, xCoords.get(i), yCoords.get(i), xCoords.get(i + 1),
7. yCoords.get(i + 1)));
8. /\* 更新到达下一个目标点后的朝向 \*/
9. **if** (xCoords.get(i) == xCoords.get(i + 1)) // 两点是上下方位
10. currentBearing = yCoords.get(i) > yCoords.get(i + 1) ? 180 : 0;
11. **else** **if** (yCoords.get(i) == yCoords.get(i + 1)) { // 两点是左右方位
12. currentBearing = xCoords.get(i) > xCoords.get(i + 1) ? 270 : 90;
13. } **else** { // 两点非正方位
14. **double** beta = Math
15. .atan((**double**) (yCoords.get(i) - yCoords.get(i + 1)) / (xCoords.get(i) - xCoords.get(i + 1)))
16. / Math.PI \* 180;
17. **if** (yCoords.get(i) > yCoords.get(i + 1)) // 目标点在下面
18. currentBearing = 180 + beta;
19. **else** // 目标点在上面
20. currentBearing = xCoords.get(i) > xCoords.get(i + 1) ? 360 + beta : beta;
21. }
22. }
23. **return** bearings;
24. }

### Problem 7: Convex Hulls

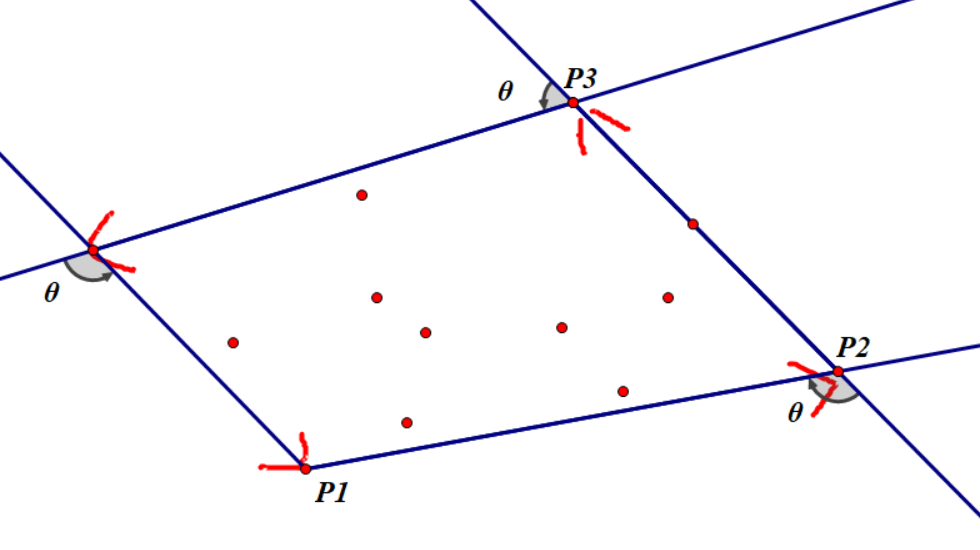
经典的凸包问题。给定平面上一些点的二维坐标，要求一个尽可能小的点集，该集合中的点在平面上围成一个凸多边形，且该凸多边形包含给定的所有点。

6.031实验描述中建议可以用gift-wrapping算法来做，也叫做Jarvis算法，在网上找了很久关于该算法的介绍，反复考虑一些特殊情况后，终于通过了一些特殊案例的测试。

该算法的大致思想是，答案要求凸多边形有两点，一点是尽可能小，还有就是要包含所有的给定点。要想尽可能小，那么凸多边形就一定是给定点中的一部分点组成的，这样是最小的，要想包含所有的给定点，可以这样做，先找到给定点集的边缘点，比如说最左下的那个点，以它作为凸包的起点，然后寻找第二个点，第二个点应该满足和起点连线与x正半轴夹角最小，从第三个点开始遍历，从第一次遍历开始，将第一个找到的点记为P1，第二个找到的点记为P2，要寻找的下一个点记为P3，每次寻找P3时，应满足P3P2和P2P1的夹角最小，这样步进寻找，得到的边连接起来可以囊括所有的点，直到找到的P3回到起点为止。

在构建凸包过程中，对P3的寻找，可能会遇到P2P3直线上有多个点，那么就要找距离P2最远的那一个，放进凸包中。还有就是只有一个点和两个点的情况，这时规定给定点全是凸包中的点，这时凸包是一个点，或是一条线。

附上一张角度示意图。总是希望两向量夹角θ是最小的，并且取最远的那个点。



代码如下（包含了自己添加的三个辅助函数）：

1. **public** **static** Set<Point> convexHull(Set<Point> points) {
2. Set<Point> hullPoints = **new** HashSet<Point>(); // 凸包点集
3. Iterator<Point> iterator = points.iterator(); // 给定点集迭代器
4. /\* p1,p2,p3分别是遍历过程中，按序找到的凸包点 \*/
5. Point p1 = iterator.hasNext() ? iterator.next() : null;
6. Point p2 = null, p3 = null;
7. Point next; // 迭代变量
9. /\* 给定点集不为空，否则退出 \*/
10. **if** (points.size() > 0) {
11. /\* 开始迭代，寻找起点 \*/
12. iterator = points.iterator();
13. **while** (iterator.hasNext()) {
14. next = iterator.next();
15. /\* 找到给定点集最左下的点作为起点 \*/
16. **if** (next.y() < p1.y())
17. p1 = next;
18. **else** **if** (next.y() == p1.y() && next.x() < p1.x())
19. p1 = next;
20. }
21. hullPoints.add(p1); // 起点加入凸包
22. /\* 给定点集有2个及更多的点 \*/
23. **if** (points.size() > 1) {
24. /\* 开始迭代，寻找一个与起点不同的点作为第二个点 \*/
25. iterator = points.iterator();
26. p2 = iterator.next();
27. **while** (p2.equals(p1))
28. p2 = iterator.next();
29. /\* 迭代，寻找满足条件的第二个点 \*/
30. iterator = points.iterator();
31. **while** (iterator.hasNext()) {
32. next = iterator.next();
33. /\* 条件：与起点不同，且与起点连线与x正半轴夹角最小，即夹角cos值最大 \*/
34. **if** (!next.equals(p1) && !next.equals(p2) && cosBetweenTwoPoints(p1.x(), p1.y(), next.x(),
35. next.y()) > cosBetweenTwoPoints(p1.x(), p1.y(), p2.x(), p2.y())) {
36. p2 = next;
37. }
38. /\* 还要满足在共线的点中，距离起点最远 \*/
39. **else** **if** (cosBetweenTwoPoints(next.x(), next.y(), p1.x(), p1.y()) == cosBetweenTwoPoints(p2.x(),
40. p2.y(), p1.x(), p1.y())
41. && distanceBetweenTwoPoints(next.x(), next.y(), p1.x(),
42. p1.y()) > distanceBetweenTwoPoints(p2.x(), p2.y(), p1.x(), p1.y())) {
43. p2 = next;
44. }
45. }
46. hullPoints.add(p2); // 第二个点加入凸包
47. /\* 给定点集有多于2个点，这时凸包是凸多边形 \*/
48. **if** (points.size() > 2) {
49. /\* 永真循环，直到遍历回到起点跳出循环 \*/
50. **while** (**true**) {
51. /\* 迭代，寻找下一个点 \*/
52. iterator = points.iterator();
53. p3 = iterator.next();
54. /\* 寻找满足条件的下一个点 \*/
55. **while** (iterator.hasNext()) {
56. next = iterator.next();
57. /\* 与前两个点不同 \*/
58. **if** (next.equals(p1) || next.equals(p2))
59. **continue**;
60. /\* P3P2与P2P1夹角最小 \*/
61. **if** (cosFromP1P2ToP2P3(p1.x(), p1.y(), p2.x(), p2.y(), next.x(),
62. next.y()) > cosFromP1P2ToP2P3(p1.x(), p1.y(), p2.x(), p2.y(), p3.x(), p3.y())) {
63. p3 = next;
64. }
65. /\* 在共线的点里找距离P2最大的那个 \*/
66. **else** **if** (cosFromP1P2ToP2P3(p1.x(), p1.y(), p2.x(), p2.y(), next.x(),
67. next.y()) == cosFromP1P2ToP2P3(p1.x(), p1.y(), p2.x(), p2.y(), p3.x(), p3.y())
68. && distanceBetweenTwoPoints(p2.x(), p2.y(), next.x(),
69. next.y()) > distanceBetweenTwoPoints(p2.x(), p2.y(), p3.x(), p3.y())) {
70. p3 = next;
71. }
72. }
73. /\* 跳出循环条件，遍历回到起点 \*/
74. **if** (hullPoints.contains(p3))
75. **break**;
76. **else** {
77. /\* 找到了下一个点，加入凸包，并更新P1,P2，开启下一轮迭代 \*/
78. hullPoints.add(p3);
79. Point temp = p2;
80. p2 = p3;
81. p1 = temp;
82. }
83. }
84. }
85. }
86. }
87. **return** hullPoints;
88. }

辅助函数cosBetweenTwoPoints返回了(x1,y1)和(x2,y2)连线与x正半轴所夹角的cosine值。

1. **public** **static** **double** cosBetweenTwoPoints(**double** x1, **double** y1, **double** x2, **double** y2) {
2. **double** cosalpha = Math.acos((x2 - x1) / distanceBetweenTwoPoints(x1, y1, x2, y2));
3. **return** cosalpha;
4. }

辅助函数cosFromP1P2ToP2P3返回了向量P1P2(x2-x1, y2-y1)和向量P2P3(x3-x2, y3-y2)所夹角的cosine值。

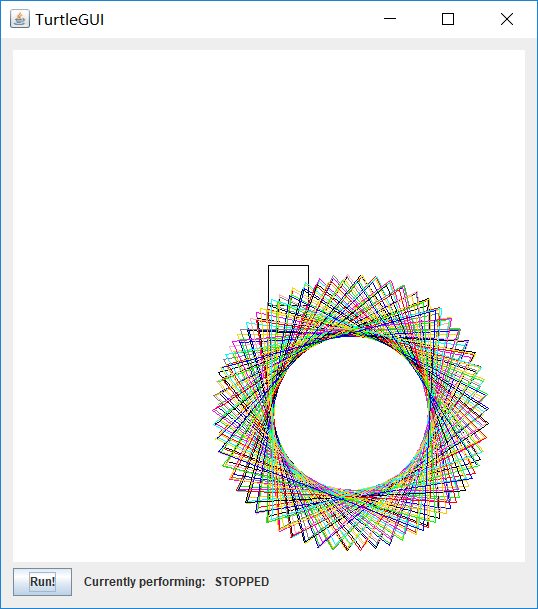
1. **public** **static** **double** cosFromP1P2ToP2P3(**double** x1, **double** y1, **double** x2, **double** y2, **double** x3, **double** y3) {
2. **double** P1P2X = x2 - x1, P1P2Y = y2 - y1;
3. **double** P2P3X = x3 - x2, P2P3Y = y3 - y2;
4. **double** costheta = (P1P2X \* P2P3X + P1P2Y \* P2P3Y) / (Math.sqrt(Math.pow(P1P2X, 2) + Math.pow(P1P2Y, 2))
5. \* Math.sqrt(Math.pow(P2P3X, 2) + Math.pow(P2P3Y, 2)));
6. **return** costheta;
7. }

辅助函数distanceBetweenTwoPoints返回了两点(x1,y1)和(x2,y2)之间的距离。

1. **public** **static** **double** distanceBetweenTwoPoints(**double** x1, **double** y1, **double** x2, **double** y2) {
2. **double** distance = Math.sqrt(Math.pow((x1 - x2), 2) + Math.pow((y1 - y2), 2));
3. **return** distance;
4. }

### Problem 8: Personal art

该问题想考验一下我们对Java GUI和对象方法的理解能力，以及艺术想象力。用forward和turn方法，还可以用color方法，来绘制一些有趣的线图。我绘制了一个这个：



代码如下：

1. **public** **static** **void** drawPersonalArt(Turtle turtle) {
2. turtle.turn(108.0);
3. **for** (**int** i = 0; i < 50; i++) {
4. **for** (**int** j = 0; j < 5; j++) {
5. turtle.forward(220);
6. turtle.turn(108.0);
7. }
8. turtle.turn(3);
9. turtle.color(PenColor.**class**.getEnumConstants()[i % 10]);
10. }
11. }

### Submitting

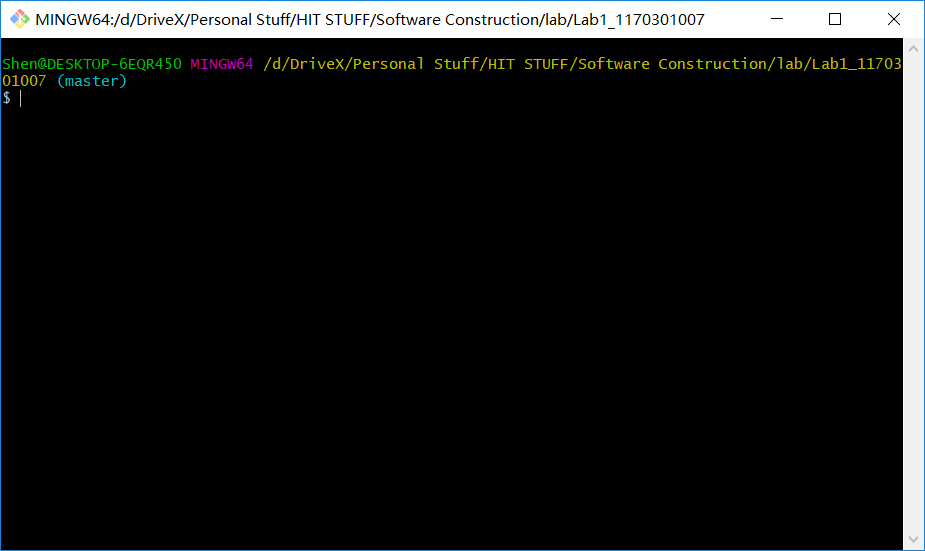
如何通过Git提交当前版本到GitHub上你的Lab1仓库。

该过程中设计到的命令有：

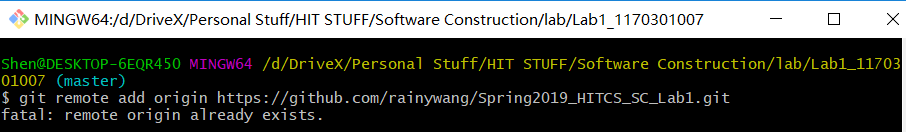
* git init
* git add
* git commit
* git remote add origin
* git push origin master

下面以实例说明：

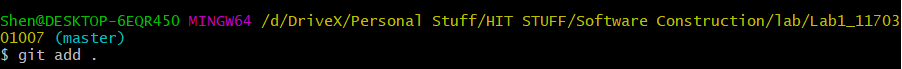
先在项目路径下打开Git Bash，关于本地仓库和远程仓库的创建工作在3.2.1中应该就已经做好了。这一步也可在任何一个目录下打开Git Bash，然后用文件管理操作进入到项目路径下。



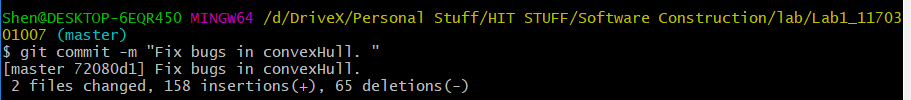
然后，将远程仓库的HTTPS添加到标识符origin中。如果之前已经添加过，那么就会出现如下fatal提示：



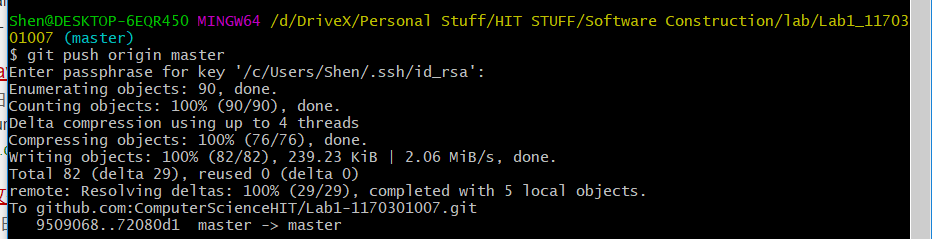
到此，可以用origin来表示远程仓库。为了上传提交或更新提交我们在本地的项目开发，先将代码上传到暂存区，git add .即为将当前目录下所有文件上传到暂存区。



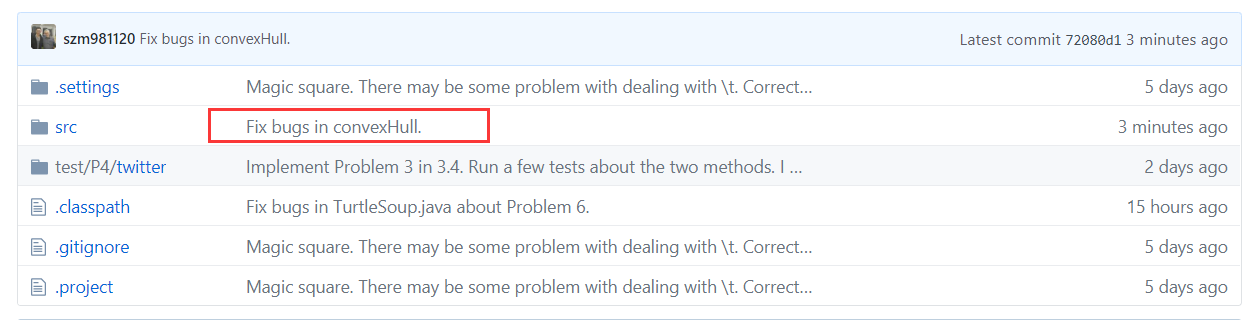
然后提交更新，-m参数后面可以添加描述本次提交信息的字串。

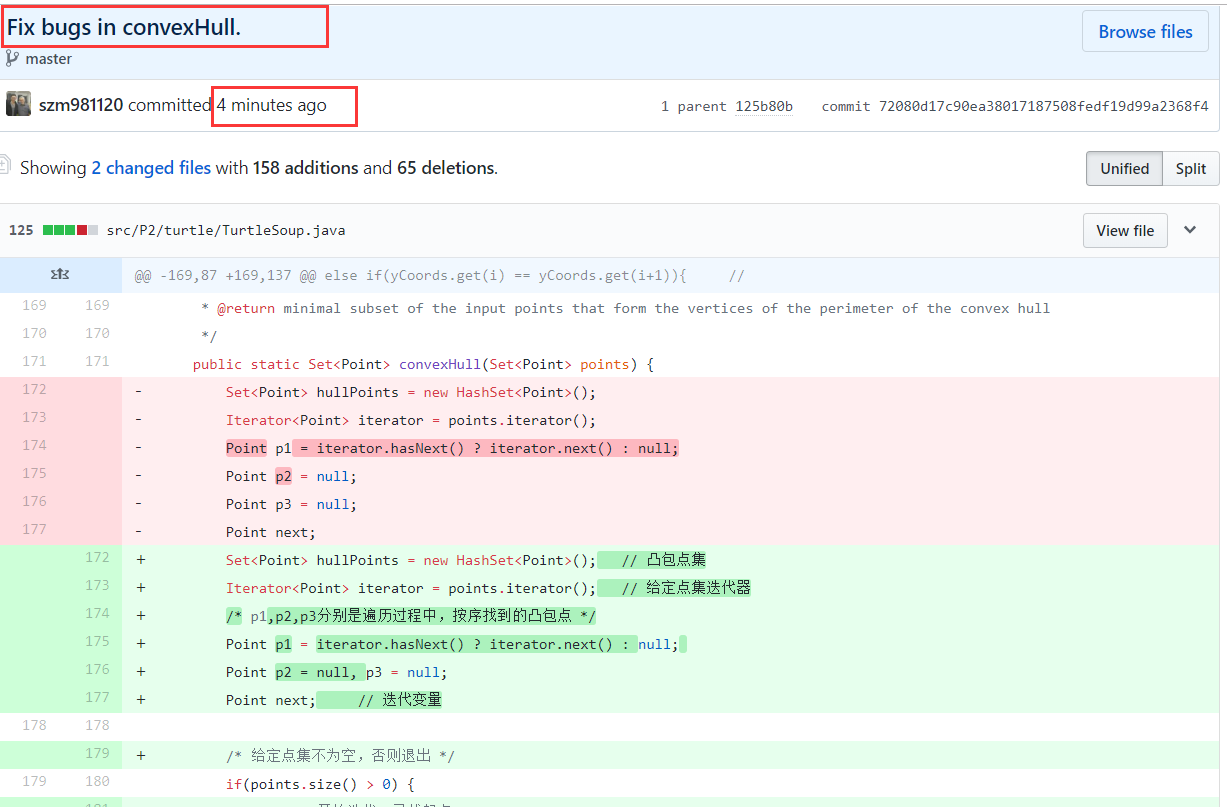


这时，本次提交就已经写入到本地Git管理历史中了。要想上传到远程仓库，用这个。可能要输入密码，这个密码是SSH密钥，是一种安全的信息传输协议。

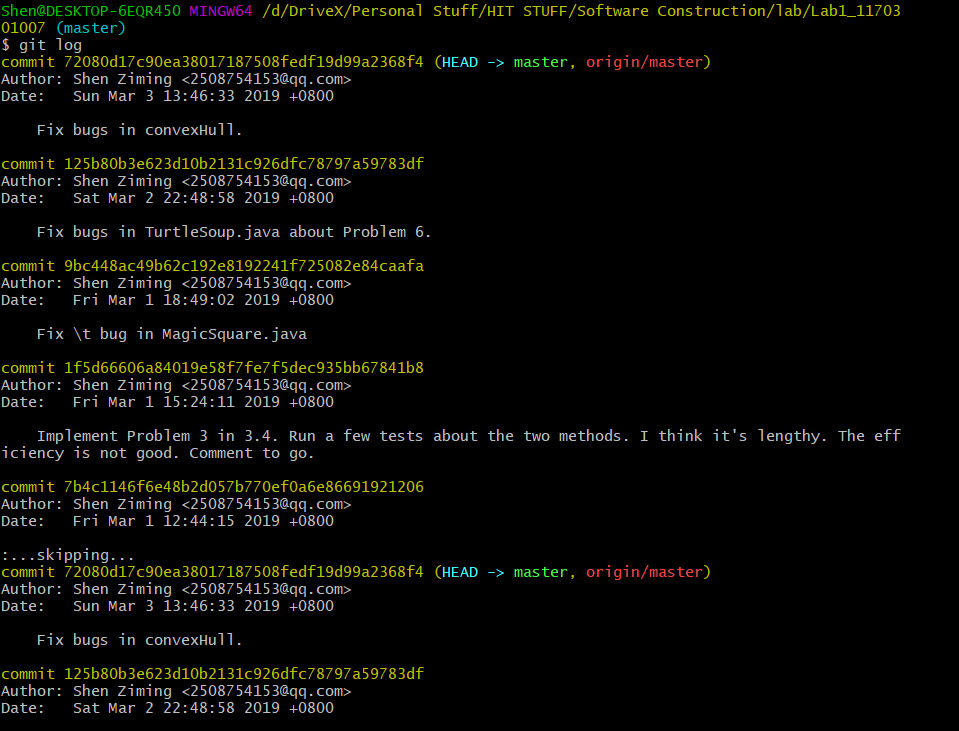


然后在远程仓库中我们可以看到本次提交的信息，和更改状态。





还可以查看本地的提交历史日志。截图给出的是我的最新的几次提交。



## Social Network

该问题的实质是图问题中的最短路径问题，要求的是源点到给定点的最短路径，图的各条边权值均为1，图为无向图，但是后期可能要扩展到有向图。点和边手动输入。要求两个人的距离，也就是从一个人到另外一个人的最短路长。

该问题给出了一个范例测试代码，要求能运行该代码，并得出正确的输出结果。

该问题主要围绕getDistance方法和Person类。getDistance方法中计算两人距离，Person类则规定了每个人的各种属性，还可以规定对每个人的操作方法。

最短路径的实现可以用图的广度优先搜索算法，在刚开始实施时先假设每个人都有不同的名字，后期会处理名字不同的异常。该问题还要求处理非法输入，并写一些额外的测试案例来测试代码，确保代码的正确性。测试文件应该在路径\test\P3.Test中。

完成本阶段问题后，用git指令上传到远程仓库中。

### 设计/实现FriendshipGraph类

给出你的设计和实现思路/过程/结果。

FriendshipGraph类在这里其实是实现了一个无向图，其中定义的方法都是和图的操作有关。这其中包括了FriendshipGraph()构造方法、addVertex(Person person)添加顶点、addEdge(Person p1, Person p2)添加无向边（或双向有向边），getDistance(Person p1, Person p2)计算两人关系距离和main()主函数。

由于我们要做一个图的操作，所以要有邻接表或邻接矩阵，这里预先不知道有多少人在图中，所以用变长数组来做邻接表，表头的类型是Person类对象，所以要有映射的话，因该用Map类。

1. **private** HashMap<Person, ArrayList<Person>> adj; // 邻接表

main()主函数中写的是实验手册中给定的代码，用来验证正确性。

FriendshipGraph()构造方法要构造一个实例化对象，所以代码如下：

1. **public** FriendshipGraph() {
2. **this**.adj = **new** HashMap<Person, ArrayList<Person>>();
3. }

addVertex(Person person)是要往图中添加一个顶点，这个顶点是一个Person类对象，添加一个顶点之后，邻接表要有相应的变化，把这个添加进去的人映射到一个空的列表上就好了。但是在此之前，为了避免添加了重名的人，做了一个重名检查，如果遇到重名，那么报出异常，重名不区分大小写。

1. **public** **void** addVertex(Person person) throws Exception {
2. **for** (Person p : adj.keySet()) {
3. **if** (p.getName().equalsIgnoreCase(person.getName())) {
4. **throw** **new** Exception(person.getName() + " has already existed!");
5. }
6. }
7. ArrayList<Person> newArray = **new** ArrayList<Person>();
8. **this**.adj.put(person, newArray);
9. }

addEdge(Person p1, Person p2)是要往图中添加一条有向边，表示p1到p2有关系存在，也是要对邻接表相应的变化一下，把这p2添加到p1的邻接表中。这里面我后来又加了异常处理，就是说，我们在构造图的时候，有可能加入了一条边，但这条边的两个顶点我们还未加入，这时候需要处理一下异常，这种输入是不合法的。这种添加边的方式就可以扩展到有向图中了。如果要给两个人添加相互关系的话，一定要调用两次addEdge才行。

1. **public** **void** addEdge(Person p1, Person p2) throws Exception {
2. **if** (adj.containsKey(p1)) {
3. adj.get(p1).add(p2);
4. } **else** {
5. **throw** **new** Exception("Person doesn't exist!");
6. }
7. }

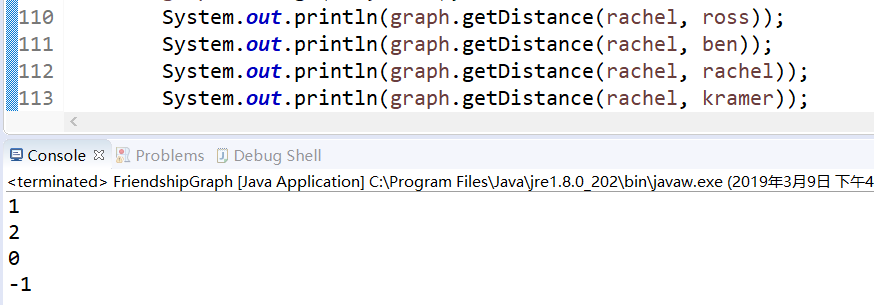
getDistance(Person p1, Person p2)开始真正计算两个人的关系距离了，用广度优先搜索算法，先得到一个边的映射关系edgeTo，比如说从源点A到目标点P的有向路径中有两相邻点B和C，那么在edgeTo中，C映射到B。在遍历过程中还需要将每个点的遍历标记由假置为真。

1. **int** distance = 0; // 距离初始化
2. Queue<Person> queue = **new** LinkedList<Person>(); // BFS遍历过程的队列
3. /\*
4. \* edgeTo映射：假设A映射到B，那么表示A与B有直接关系 且在由源点到目标点的有向路径中，A在B的后面，即B是A的前置
5. \*/
6. HashMap<Person, Person> edgeTo = **new** HashMap<Person, Person>();
7. /\* 所有人遍历标记置假 \*/
8. **for** (Person p : adj.keySet())
9. p.marked = **false**;
10. p1.marked = **true**; // 源点遍历标记置真
11. queue.add(p1); // 源点加入队列
12. /\* BFS遍历，得到edgeTo \*/
13. **while** (!queue.isEmpty()) {
14. Person tempPerson = queue.remove();
15. /\* 对出队的元素的所有邻接点遍历 \*/
16. **for** (Person p : adj.get(tempPerson)) {
17. **if** (!p.marked) { // 找到了一个没有遍历过的点
18. edgeTo.put(p, tempPerson); // 记录前置后置关系
19. p.marked = **true**; // 遍历标记置真
20. queue.add(p); // 一个新的遍历过的点入队
21. }
22. }
23. }

最后要想计算目标点到源点的关系距离是多少，只需要先判断目标点是否是遍历过的，如果是，就从目标点在edgeTo中映射回溯到源点，回溯步数就是结果；否则，目标点到源点的关系距离为0.

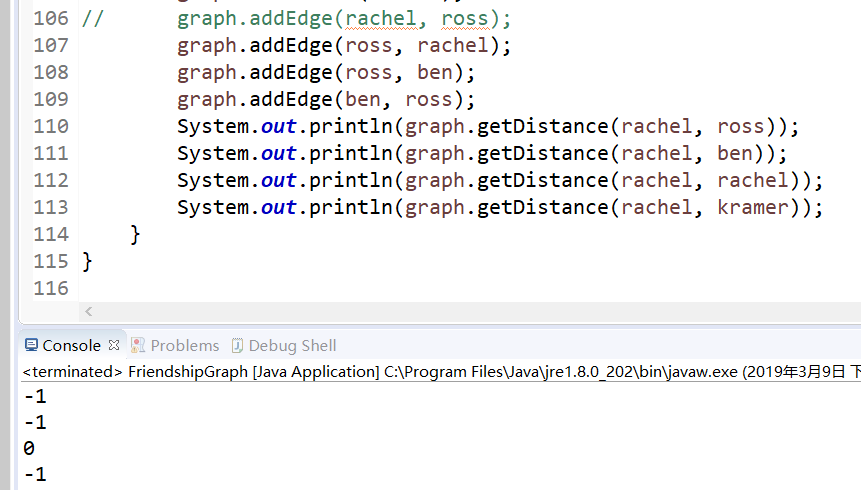
1. /\* 此时BFS遍历完成，用edgeTo来反向回溯源点到目标点的路径，同时计算路径长度 \*/
2. **if** (!p2.marked)
3. distance = -1;
4. **else**
5. **for** (Person p = p2; !p.equals(p1); p = edgeTo.get(p))
6. distance++;
7. **return** distance;

运行结果演示。

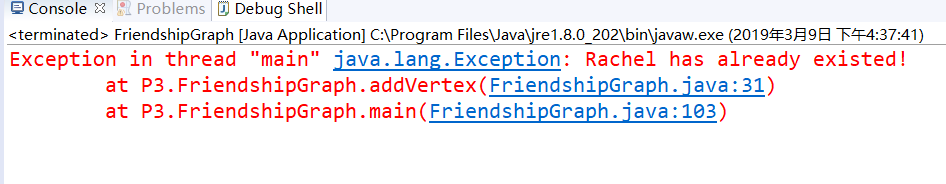


**异常分析**

如果将实验手册中给定的main代码中的第10行注释掉，也就是Rachel和Ross之间只存在Ross到Rachel的单向关系，那么同样的结果会出现，-1，-1，0，-1，因为Kramer是独立的，没有变化，而Rachel到Ben是要经过Ross的，如果Rachel和Ross失去联系，那和Ben也失去联系了。



另外，我们还有可能在构造顶点时，出现了两个人重名的情况。我们默认任意两人没有重名，但是如果不幸地将ross变量的名字也初始化成了Rachel，那么应该给出报错。这点我是在addVertex中实现的，报出一个异常来处理重名事故的发生。



### 设计/实现Person类

Person类中定义了每个人的属性，构造方法中初始化了这些属性，还有一些其它的方法。每个人的属性有名字和遍历标记，遍历标记可以是公开的，但名字最好是私有的，因为一个人的名字一旦确定了，就不希望再更改了，最好把它封装在Person类中。所有我们还是写一个getName的公开方法，来获取返回人的名字。代码如下：

1. /\*\*
2. \*
3. \* @author Shen
4. \*
5. \*/
6. **public** **class** Person {
8. **private** String name;
9. **public** boolean marked;
11. /\*\*
12. \*
13. \* @param name construct an object with name
14. \*/
15. **public** Person(String name) {
16. **this**.name = name; // 初始化名字
17. **this**.marked = **false**; // 遍历标记置假
18. }
20. /\*\*
21. \*
22. \* @return the person's name
23. \*/
24. **public** String getName() {
25. **return** **this**.name;
26. }
27. }

### 设计/实现客户端代码main()

### 设计/实现测试用例

给出你的设计和实现思路/过程/结果。

这里要从头写一个FriendshipGraphTest类，里面用JUnit来测试FriendshipGraph里面的三个方法，仿照TurtleSoupTest.java中的JUnit的测试模式，在FriendshipGraphTest里面添加四个方法，testAssertionsEnabled()、addVertexTest()、addEdgeTest()和getDistanceTest()。

testAssertionsEnabled()方法是用来测试assertions是有效的，确保我们的测试正确性。

剩下的方法是对关系图的测试，我们创建了6个全局Person对象，作为关系图中的备选顶点。

1. Person rachel = **new** Person("Rachel");
2. Person ross = **new** Person("Ross");
3. Person ben = **new** Person("Ben");
4. Person jack = **new** Person("Jack");
5. Person kramer = **new** Person("Kramer");
6. Person me = **new** Person("me");

addVertexTest()方法中，可以添加一些Person对象，然后检查它们：

1. **public** **void** addVertexTest() throws Exception {
2. FriendshipGraph graph = **new** FriendshipGraph();
3. assertTrue("expected empty", graph.adj.isEmpty());
4. graph.addVertex(rachel);
5. assertTrue("expected containing rachel", graph.adj.containsKey(rachel));
6. }

addEdgeTest()方法中，可以添加一些Person对象以及对应的边，然后检查这些边：

1. **public** **void** addEdgeTest() throws Exception {
2. FriendshipGraph graph = **new** FriendshipGraph();
3. assertTrue("expected empty", graph.adj.isEmpty());
4. graph.addVertex(ben);
5. graph.addVertex(jack);
6. graph.addEdge(ben, jack);
7. assertTrue("expected connection between ben and jack", graph.adj.get(ben).contains(jack));
8. assertFalse("expected NO connection between ross and jack", graph.adj.get(ben).contains(ross));
9. }

getDistanceTest()方法中先构建一个关系图，然后测试关系距离结果即可。千万别忘了构造无向图时对同一对顶点调用两次addEdge

1. **public** **void** getDistanceTest() throws Exception {
2. FriendshipGraph graph = **new** FriendshipGraph();
3. graph.addVertex(ben);
4. graph.addVertex(jack);
5. graph.addVertex(kramer);
6. graph.addVertex(rachel);
7. graph.addVertex(ross);
8. graph.addVertex(me);
9. graph.addEdge(jack, ben);
10. graph.addEdge(ben, jack);
11. graph.addEdge(jack, rachel);
12. graph.addEdge(rachel, jack);
13. graph.addEdge(jack, ross);
14. graph.addEdge(ross, jack);
15. graph.addEdge(ben, kramer);
16. graph.addEdge(kramer, ben);
18. assertEquals("expected distance is 2", 2, graph.getDistance(jack, kramer));
19. assertEquals("expected distance is 3", 3, graph.getDistance(rachel, kramer));
20. assertEquals("expected distance is -1", -1, graph.getDistance(jack, me));
21. }

## Tweet Tweet

### Problem 1: Extracting data from tweets

在该问题中，需要测试并完善Extract.java中的一些方法，包括getTimespan()和getMentionedUsers(). 完善这些方法，并写一些针对它们的测试案例，在ExtractTest.java中通过测试，确保正确性。

这部分内容还要求我们仔细阅读Timespan类的规约文档，学习Instant类，因为这部分内容会用到这些。

getTimespan(List<Tweet> tweets)方法要求我们返回一个Timespan类型变量，该变量的start和end属性构成了一个时间区间，该区间包含所有的tweet发表时间，那么这个问题其实是一个求最值的问题。代码如下：

1. **public** **static** Timespan getTimespan(List<Tweet> tweets) {
2. Instant start, end; // timespan的起始和结束
3. /\* 初始化 \*/
4. start = tweets.get(0).getTimestamp();
5. end = tweets.get(0).getTimestamp();
6. **for** (Tweet t : tweets) { // 遍历所有tweet
7. **if** (t.getTimestamp().isBefore(start)) // 求出发表时间的最早值
8. start = t.getTimestamp();
9. **if** (t.getTimestamp().isAfter(end)) // 求出发表时间的最晚值
10. end = t.getTimestamp();
11. }
12. Timespan timespan = **new** Timespan(start, end);
13. **return** timespan;
14. }

getMentionedUsers(List<Tweet> tweets)方法要求我们返回一个Set集合，其中包含了所有在tweet中被@的用户的名字，这个问题是一个字符串解析的问题。首先要遍历所有的tweet，然后在每一个tweet的text中寻找关键字@，检查@附近的字符的合法性，构建出满足要求的被提及的用户名，还要注意用户名不区分大小写。

其实这个问题要想考虑一个好的健壮性，还是挺繁琐的，tweet中@有各种各样的情况。邮件中的@，连续@两个人比如说@shen@jack，那么@jack是失效的，等等情况，都要考虑进去。

代码如下：

1. **public** **static** Set<String> getMentionedUsers(List<Tweet> tweets) {
2. Set<String> mentionedUsers = **new** HashSet<String>();
3. **for** (Tweet t : tweets) {    // 遍历所有tweet
4. String text = t.getText();
5. **int** index = 0;  // index是每次检索字符@时，检索的起始索引，每次检索后要更新
6. **while** ((index = text.indexOf("@", index)) >= 0) {
7. String username = "";   // 初始化被提及的用户名
8. **int** i = 0;  // i是对当前检索到的@附近的字串的一个检查过程中的循环变量
9. /\* 如果@在字串的开始
10. \* 或者@的前面仍有非法字符
11. \* 那么可以暂时认定是一次成功的检索 \*/
12. **if** (index == 0 || !validCharacterInUsername(text.charAt(index - 1))) {
13. /\* 从@后面的一个索引位开始检索字串，直到末尾或遇到一个非法字符终止 \*/
14. **for** (i = index + 1; i < text.length() && validCharacterInUsername(text.charAt(i)); i++)
15. username += String.valueOf(text.charAt(i)); // 这期间都是合法的用户名
16. username = username.toLowerCase();  // 用户名不区分大小写
17. /\* 如果该用户名未被搜集
18. \* 或用户名长度大于0，就是存在用户名
19. \* 那么就把它加入到mentionedUsers \*/
20. **if** (!mentionedUsers.contains(username) && username.length() > 0)
21. mentionedUsers.add(username);
22. index = i;  // 更新检索起始索引
23. } **else** {
24. index += 1; // 当前检索的@不是真正的提及标志，继续检索
25. }
26. }
27. }
28. **return** mentionedUsers;
29. }

实现这个方法还有一个辅助方法validCharacterInUsername(char c)，它返回一个boolean型变量，意为字符c是否为合法的用户名字符。

1. **public** **static** boolean validCharacterInUsername(**char** c) {
2. **if** (c >= 'A' && c <= 'Z') {
3. **return** **true**;
4. } **else** **if** (c >= 'a' && c <= 'z') {
5. **return** **true**;
6. } **else** **if** (c == '\_' || c == '-') {
7. **return** **true**;
8. } **else** **if** (c >= '0' && c <= '9') {
9. **return** **true**;
10. } **else** {
11. **return** **false**;
12. }
13. }

### Problem 2: Filtering lists of tweets

在该问题中，需要测试并完善Filter.java中的一些方法，包括writtenBy(), inTimespan()和containing(). 完善这些方法，并写一些针对它们的测试案例，在FilterTest.java中通过测试，确保正确性。

writtenBy(List<Tweet> tweets, String username)方法是要返回一个Tweet对象的列表，其中包含了所有作者为username的tweet，作者名不区分大小写，这个实现只需要一个遍历和一个String.equals IgnoreCase()方法就好了。代码如下：

1. **public** **static** List<Tweet> writtenBy(List<Tweet> tweets, String username) {
2. List<Tweet> writtenByList = **new** ArrayList<Tweet>();
3. **for**(Tweet t : tweets) {
4. **if**(t.getAuthor().equalsIgnoreCase(username))    // 用tweet的作者和username作比较，忽略大小写
5. writtenByList.add(t);
6. }
7. **return** writtenByList;
8. }

inTimespan(List<Tweet> tweets, Timespan timespan)要求返回一个Tweet对象的列表，其中包含所有发表时间在timespan内的tweet. 在看了Timespan.java中规定的属性和方法后，这题也就非常简单了。用一个遍历，再加上Timespan的isAfter和isBefore方法确定tweet发表时间的范围就好了，还要注意，在时间的区间端点处也要考虑进去。代码如下：

1. **public** **static** List<Tweet> inTimespan(List<Tweet> tweets, Timespan timespan) {
2. List<Tweet> inTimespanList = **new** ArrayList<Tweet>();
3. **for** (Tweet t : tweets) {
4. **if** (t.getTimestamp().equals(timespan.getStart()) || t.getTimestamp().equals(timespan.getEnd())
5. || (t.getTimestamp().isAfter(timespan.getStart()) && t.getTimestamp().isBefore(timespan.getEnd())))
6. inTimespanList.add(t);
7. }
8. **return** inTimespanList;
9. }

containing(List<Tweet> tweets, List<String> words)这个方法要返回一个Tweet对象列表，其中包含所有内容中有words列表中单词的tweet，这个问题细节比较多，但大致上分为三层循环，第一层遍历tweet，第二层遍历目标单词，第三层遍历当前tweet中的内容单词，在最内层循环中如果触发了搜索成功的条件，则可以直接跳出二层循环。细节多是因为在内容中，要找一个目标单词，不区分大小写，而且该单词前后缀可能有合法的标点符号，所以并不是单词完全匹配才能找到，比如说”morning”和”morning.”应该看作是一样的。

代码如下：

1. **public** **static** List<Tweet> containing(List<Tweet> tweets, List<String> words) {
2. List<Tweet> containingList = **new** ArrayList<Tweet>();
3. /\* 遍历所有tweet \*/
4. **for** (Tweet t : tweets) {
5. /\* 因为要搜索的词不区分大小写，把tweet内容全部变为小写后在用空格分割 \*/
6. String[] text = t.getText().toLowerCase().split(" ");
7. secondloop: **for** (String w : words) { // 逐个检查目标单词
8. **for** (String s : text) { // 对每个目标单词，逐个检查tweet内容中的单词
9. /\*
10. \* 如果某单词包含目标单词， 且无任何其它前缀， 且和目标单词长度相同或者后缀有合法标点，
11. \* 那么这就是一个搜索成功的单词，该tweet加入结果列表，并直接跳出外层循环
12. \*/
13. **if** (s.contains(w.toLowerCase())
14. && (s.indexOf(w.toLowerCase()) == 0
15. || validPunctuation(s.charAt(s.indexOf(w.toLowerCase()) - 1)))
16. && (s.indexOf(w.toLowerCase()) + w.length() == s.length()
17. || validPunctuation(s.charAt(w.length())))) {
18. containingList.add(t);
19. **break** secondloop;
20. }
21. }
22. }
23. }
24. **return** containingList;
25. }

有一个辅助函数是用来检测字符是否是合法的标点符号的，对复杂情况的单词检测有帮助。

1. **public** **static** boolean validPunctuation(**char** c) {
2. String validPunctuationString = ".,!?':;\"";
3. **if** (validPunctuationString.indexOf(String.valueOf(c)) == -1)
4. **return** **false**;
5. **else**
6. **return** **true**;
7. }

### Problem 3: Inferring a social network

该问题要求完善SocialNetwork.java中的一些方法。guessFollowsGraph()方法构建了一个社交网络，网络中有所有tweet中被提及的所有用户。社交网络从tweet内容中，大概体现出谁在twitter上关注了谁。influencers()方法返回一个用户列表，该列表按粉丝数量降序排序。该问题还要求我们给这两个方法写一些测试，验证代码的正确性。暂时，判定A是不是关注了B的条件，就是看A的tweet中有没有@B。

这里还可以运行Main.java，来实战拉取大量twitter上的tweet，并输出前十名网红，再去twitter上查找一下，这也可以验证代码是不是可靠。

guessFollowsGraph()方法要求返回一个映射，key是粉丝，value是对应key关注的所有用户集合。要解决这个问题，可以复用一下Extract.java中的一些代码，用来检索tweet内容中被@的用户名。不同的是，作者@自己不能算关注自己，而对于一个被关注的人，如果他没有发表tweet，也要把他加入关系图，毕竟他是受欢迎的，在后面完善influencers方法时方便一些。代码如下：

1. **public** **static** Map<String, Set<String>> guessFollowsGraph(List<Tweet> tweets) {
2. Map<String, Set<String>> guessFollowsGraphMap = **new** HashMap<String, Set<String>>();
3. **for** (Tweet t : tweets) { // 遍历所有tweet
4. /\* 关系映射图中的原像集中的用户一定只能是发表tweet的作者 \*/
5. guessFollowsGraphMap.put(t.getAuthor().toLowerCase(), **new** HashSet<String>());
6. String text = t.getText(); // 查看内容
7. /\* 注：下面的操作复用了Extract.java中的一些代码 \*/
8. **int** index = 0; // index是每次检索字符@时，检索的起始索引，每次检索后要更新
9. **while** ((index = text.indexOf("@", index)) >= 0) {
10. String mentionedName = ""; // 初始化被提及的用户名
11. **int** i = 0; // i是对当前检索到的@附近的字串的一个检查过程中的循环变量
12. /\* 如果@在字串的开始
13. \* 或者@的前面仍有非法字符
14. \* 那么可以暂时认定是一次成功的检索 \*/
15. **if** (index == 0 || !Extract.validCharacterInUsername(text.charAt(index - 1))) {
16. /\* 从@后面的一个索引位开始检索字串，直到末尾或遇到一个非法字符终止 \*/
17. **for** (i = index + 1; i < text.length() && Extract.validCharacterInUsername(text.charAt(i)); i++)
18. mentionedName += text.charAt(i);    // 这期间都是合法的用户名
19. /\* 作者@自己，不能算做关注 \*/
20. **if** (!t.getAuthor().equalsIgnoreCase(mentionedName))
21. guessFollowsGraphMap.get(t.getAuthor().toLowerCase()).add(mentionedName.toLowerCase());
22. /\* 对于一个被关注的人，如果他没有发表tweet，也要加入到关系图中 \*/
23. **if** (!guessFollowsGraphMap.keySet().contains(mentionedName.toLowerCase()))
24. guessFollowsGraphMap.put(mentionedName.toLowerCase(), **new** HashSet<String>());
25. index = i;  // 更新检索起始索引
26. } **else** {
27. index += 1; // 当前检索的@不是真正的提及标志，继续检索
28. }
29. }
30. }
31. **return** guessFollowsGraphMap;
32. }

influencers()方法要求返回一个List<String>，其中包含所有的用户名，并且按粉丝数量降序排列。思想是要先构造一个用户名到粉丝数的Map<String, Integer>, 统计好所有用户的粉丝数，保存到这个Map中，然后再构造一个结果List<String>, 遍历Map, 每次遍历取一个粉丝数最多的key，加入到结果List中，并在Map中把这个key-value删除，这样的话，就是按粉丝数量降序的顺序加入List的。代码如下：

1. **public** **static** List<String> influencers(Map<String, Set<String>> followsGraph) {
2. List<String> influencersList = **new** ArrayList<String>();
3. Map<String, Integer> followerNumberMap = **new** HashMap<String, Integer>(); // 用户名到粉丝数的映射
4. /\*
5. \* 先完善映射 遍历关注关系图
6. \*/
7. **for** (String s : followsGraph.keySet()) {
8. **int** number = 0; // 粉丝数变量
9. **for** (Set<String> set : followsGraph.values())
10. **if** (set.contains(s)) // 如果某用户关注人群中有s，那么s粉丝数加一
11. number++;
12. followerNumberMap.put(s, number); // 完善映射关系
13. }
14. /\*
15. \* 下面按映射表中的value值降序加入结果List 由于我们每次遍历得到一个最大粉丝数的用户， 都会在映射表中删除这个用户，
16. \* 所以，一个循环条件就是Map不为空
17. \*/
18. **int** max = 0;
19. **while** (!followerNumberMap.isEmpty()) {
20. String mostPopularPerson = null;
21. /\* 每次都要遍历用户集 \*/
22. **for** (String s : followerNumberMap.keySet()) {
23. **if** (followerNumberMap.get(s) >= max) { // 找到拥有最大粉丝数量的用户
24. max = followerNumberMap.get(s);
25. mostPopularPerson = s;
26. }
27. }
28. **if** (mostPopularPerson != null) {
29. influencersList.add(mostPopularPerson); // 把当前最大粉丝数量的用户加入结果List
30. followerNumberMap.remove(mostPopularPerson); // 最大粉丝数量用户加入List之后移出Map
31. }
32. max = 0; // 粉丝数计数器重置
33. }
34. **return** influencersList;
35. }

### Problem 4: Get smarter

这部分属于个人发挥了，要求改进guessFollowsGraph推断twitter社交网络关系，使算法更贴近生活。6.031给出了三种推断关系，要求我们实现至少一种，并在MySocialNetworkTest.java中给出相应测试。我大致想了想手册中给出的三个参考改进方案，然后选择了第二种来改进我的social network。

第一种是common hashtag. 是说如果几个人hashtag了相同的标签，那么可以认为这几个人有联系，但未必彼此关注，如果想根据这点来判断几个人彼此关注，并成为彼此的粉丝的话，有点不符合实际，而要给hashtag标签项加上影响权重，虽然是一种比较贴近实际的实现方式，但对于本次实验有点太麻烦了。

第二种是triadic closure. 这是社会网络理论中乔治·西梅尔提出的概念，认为如果A和B，B和C之间都存在强关系，即相互关系，那么可以认为A和C之间也存在关系，或者是强关系，或者是弱关系。该概念无法在庞大的复杂网络中实用，但可以作为预测的标准。

我采用了这种方式来改善社交网络预测。并规定AB, BC之间如果存在强关系，那么AC之间也是存在强关系。在实施过程中，我还遇到了一个新的异常，指的是在使用迭代器的迭代过程中，不能更改迭代对象，比如remove()或add()，否则会触发java.util.ConcurrentModificationException异常，解决方案有点复杂，我用了一种变相的解决方式，具体在我的博客中有写到，这里给出我解决之后的代码：

1. /\* Triadic closure \*/
2. HashMap<String, String> waitingAddedFollow = **new** HashMap<String, String>();
3. **for** (String A : guessFollowsGraphMap.keySet()) {
4. **for** (String B : guessFollowsGraphMap.get(A)) {
5. **for** (String C : guessFollowsGraphMap.get(B)) {
6. **if** (guessFollowsGraphMap.get(B).contains(A) && guessFollowsGraphMap.get(C).contains(B)) {
7. waitingAddedFollow.put(A, C);
8. }
9. }
10. }
11. }
12. **for** (String A : waitingAddedFollow.keySet()) {
13. String C = waitingAddedFollow.get(A);
14. **if** (!guessFollowsGraphMap.get(A).contains(C))
15. guessFollowsGraphMap.get(A).add(C);
16. **if** (!guessFollowsGraphMap.get(C).contains(A))
17. guessFollowsGraphMap.get(C).add(A);
18. }
19. /\* END triadic closure \*/

第三种方案是说，如果A关注了B，而B关注了C，那么在B转发C的tweet时，会暗示A和C存在某种联系。

改善的社交网络测试案例在/test/P4/twitter/MySocialNetworkTest.java中，已经通过了给定的测试样例。

# 实验进度记录

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 任务 | 实际完成情况 |
| 2019-02-25 | 14:00-15:30 | 编写问题1的isLegalMagicSquare函数并进行测试 | 按计划完成 |
| 2019-03-2 | 12:00-12:35 | 完善问题1的generateMagicSquare异常处理，并初步完成问题1的实验报告 | 按计划完成 |
| 2019-03-2 | 17:30-18:20 | 完成实验报告3.2.1和3.2.2 | 按计划完成 |
| 2019-03-2 | 21:20-22:45 | 完成实验报告3.2.3和3.2.4 | 按计划完成 |
| 2019-03-3 | 9:30-11:00 | 完善凸包问题的共线情况 | 按计划完成 |
| 2019-03-3 | 12:40-13:50 | 完成实验报告3.2.5，3.2.6和3.2.7 | 按计划完成 |
| 2019-03-3 | 14:30-15:00 | 完成实验报告3.3，和3.3.1的一半 | 按计划完成 |
| 2019-03-4 | 13:45-15:30 | 完成实验报告3.3.1, 3.3.2, 3.3.3和3.3.4 | 按计划完成 |
| 2019-03-4 | 17:00-18:30 | 完成实验报告3.4.1, 3.4.2, 3.4.3 | 按计划完成 |
| 2019-03-5 | 19:00-21:30 | 完成剩下的所有实验报告 | 按计划完成 |
| 2019-03-6 | 晚22:00 | 完成个人博客 | 按计划完成 |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
| 字符串中的路径表示法 | 如果使用的Eclipse IDE的话，那么项目路径默认是在项目文件夹内，在当前状态下，诸如src，bin文件夹都是和当前路径平级的，这些文件夹下再寻址就要用到斜杠或者反斜杠了。  可以注意到，Windows系统中，资源管理器中的文件路径显示都是反斜杠 \。  而反斜杠在Java字符串中属于转义字符（在markdown标记语言和其它许多编程语言中都是这样），那么在用反斜杠时，切记连续键入两个反斜杠，以忽略转义！  不过我们也可以用正斜杠 / 来写目录路径，虽然在Windows中使用反斜杠，而正斜杠通常用于Linux系统中。  不建议使用绝对路径来描述文件位置，可移植性差，极差。但它的斜杠要求和上面说的相同，只是你需要从根目录开始逐级寻址。 |
| 使用迭代器进行迭代时，改变迭代对象，遇到了java.util.ConcurrentModificationException异常 | 原因是在用迭代器进行迭代时，改变了迭代对象，这个和迭代器的迭代过程原理有关，涉及问题属于迭代过程的实现层面。  解决方案分为单线程和多线程解决方案，但这种解决方式还是太复杂，我就使用变相的解决方式，先不改变迭代对象，而把要改变的操作记录下来，等出了迭代过程后，再改变 |
| break跳出多层循环 | 有时使用了三层循环，也许会使用更多嵌套，虽然这种规模的循环嵌套极大地降低了程序的性能，但是不得已的话也是会遇到的。  在这种情况下，如果想在内层循环，触发某种条件从而跳出多层循环，只用break是不够的，需要在循环for关键字的前面加上循环标记，像这样：  Secondloop: for(……)  循环标记的名字可以任意给定，然后在break处显式表明需要跳出哪层循环，如：  break Secondloop;  这样就跳出了指定的循环。某种程度上，也算是缓解了多层循环嵌套带来了性能低下。 |

# 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

**先附上本人的Lab1实验博客链接。**

[**https://blog.csdn.net/qq\_41662115/article/details/87950830**](https://blog.csdn.net/qq_41662115/article/details/87950830)

再附上本人的参考文献

<https://blog.csdn.net/exterminator/article/details/71548931>

<https://blog.csdn.net/qq_37723158/article/details/79024308>

<https://blog.csdn.net/nickwong_/article/details/51502969>

<https://blog.csdn.net/xmc281141947/article/details/53887126>

<https://blog.csdn.net/wudinaniya/article/details/77508229>

<https://blog.csdn.net/kiddd_fu/article/details/78247290>

<https://blog.csdn.net/jinqianwang/article/details/80030060>

<https://blog.csdn.net/bone_ace/article/details/46239187>

<https://www.cnblogs.com/dw3306/p/9340031.html>

<http://www.runoob.com/java/java-stack-class.html>

<https://www.jianshu.com/p/7a86c56c632b>

<http://www.runoob.com/java/data-queue.html>

<https://stackoverflow.com/questions/20057771/no-junit-tests-found-in-eclipse>

<https://blog.csdn.net/harryfin/article/details/84620582>

<https://www.cnblogs.com/snowater/p/8024776.html>

从这么多的参考文献也可以看出来了，本次实验的最大收获就是疯狂查博客，其实这也不是第一次这样搞了，以前的实验，环境配置，就要查遍全网才能找到合适自己的解法。什么StackOverflow都是要科学上网的，百度搜不到的东西就去bing上找，CSDN的个人博客总结不能只查一篇就去做，要多查几篇，再决定解决方案。  
 这次实验的总结博客在此时此刻也已经写完了，并没有涉及太多和算法有关的东西，主要是一些小tips，记录自己做实验时遇到的坑，和自己觉得有用的总结，不过也不是所有的坑和总结都记录下来了，事情总是发生了很久才去记录下来，很多遇到的问题过去就过去了，没记下来。

这次实验结束后，对Java的基础语法其实还是没有一个全面深刻的认识，对OOP的印象也不是很深刻，但基本的套路已经掌握了，还没有到进阶的水平吧。

然后就是报告的排版感觉还不是很舒服。

## 针对以下方面的感受

1. Java编程语言是否对你的口味？

还行吧，比C强，IDE靠谱，逻辑比较清晰

1. 关于Eclipse IDE

吹就完事了，IBM贼强。这次用了import，refractor的move，还有智能补全，auto-format。以前我还在有netbeans平台，现在，赶紧投入Eclipse的怀抱！

1. 关于Git和GitHub

Git Bash感觉还是怪怪的。GitHub的仓库删除功能有点隐蔽，还有文件夹不能为空的设定，还有创建文件夹的方式也是让人一头雾水。使用起来有点抽象，不过用命令还是挺方便的，就是抽象，毕竟还没有使用过GUI，现在都还是代码指令，抽象得很。

1. 关于CMU和MIT的作业

我知道老师想让我们好，不过一次给出CMU+MIT的作业是不是有点多，光是实验手册就要读好久。

1. 关于本实验的工作量、难度、deadline

作业量真的多了，不过多也是相对的，难度这次一般吧，细节太多了，可能乙方就是要累死累活处理一下规约以外的东西，相对于deadline来说这次实验挺松的。我们早听说软件构造难搞，所以周一实验下来就开始肝实验了，第一周基本完成代码，然后完善两天再写报告，前后十天左右就基本结束了，后期再完善一下交工，马上投入到实验二。

1. 关于初接触“软件构造”课程

行了，知道这门课艰难了。艰，在于实验多且杂；难，在于课堂小测。我们会好好学习的，毕竟这门课很有可能是今后混饭吃的主要工具了。