

**2021年春季学期  
计算学部《软件构造》课程**

**Lab 1实验报告**

|  |  |
| --- | --- |
| 姓名 | 傅浩东 |
| 学号 | 1190202105 |
| 班号 | 1903002 |
| 电子邮件 | 1091288450@qq.com |
| 手机号码 | 13881165621 |

**目录**

[1 实验目标概述 1](#_Toc72681578)

[2 实验环境配置 1](#_Toc72681579)

[2.1 配置本次实验所需开发、测试、运行环境的过程 1](#_Toc72681580)

[2.2 GitHub Lab1仓库的URL地址 2](#_Toc72681581)

[3 实验过程 2](#_Toc72681582)

[3.1 Magic Squares 2](#_Toc72681583)

[3.1.1 isLegalMagicSquare() 3](#_Toc72681584)

[3.1.2 generateMagicSquare() 4](#_Toc72681585)

[3.2 Turtle Graphics 5](#_Toc72681586)

[3.2.1 Problem 1: Clone and import 5](#_Toc72681587)

[3.2.2 Problem 3: Turtle graphics and drawSquare 5](#_Toc72681588)

[3.2.3 Problem 5: Drawing polygons 6](#_Toc72681589)

[3.2.4 Problem 6: Calculating Bearings 7](#_Toc72681590)

[3.2.5 Problem 7: Convex Hulls 8](#_Toc72681591)

[3.2.6 Problem 8: Personal art 8](#_Toc72681592)

[3.2.7 Submitting 10](#_Toc72681593)

[3.3 Social Network 12](#_Toc72681594)

[3.3.1 设计/实现FriendshipGraph类 12](#_Toc72681595)

[3.3.2 设计/实现Person类 15](#_Toc72681596)

[3.3.3 设计/实现客户端代码main() 15](#_Toc72681597)

[3.3.4 设计/实现测试用例 15](#_Toc72681598)

[4 实验进度记录 17](#_Toc72681599)

[5 实验过程中遇到的困难与解决途径 18](#_Toc72681600)

[6 实验过程中收获的经验、教训、感想 18](#_Toc72681601)

[6.1 实验过程中收获的经验和教训 18](#_Toc72681602)

[6.2 针对以下方面的感受 19](#_Toc72681603)

# 实验目标概述

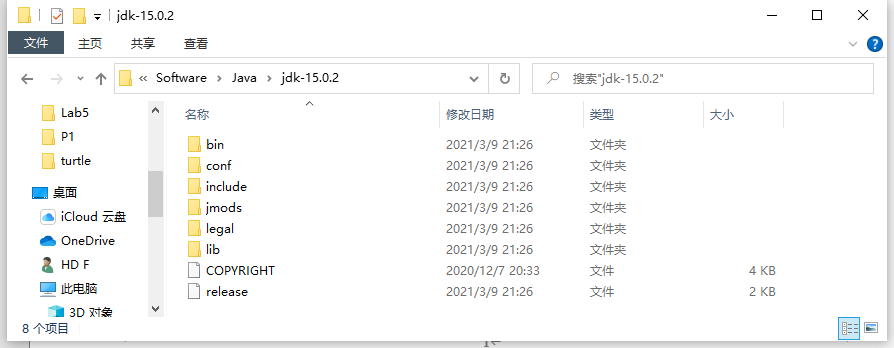
本次实验通过求解三个问题，训练基本 Java 编程技能，能够利用 Java OO 开 发基本的功能模块，能够阅读理解已有代码框架并根据功能需求补全代码，能够 为所开发的代码编写基本的测试程序并完成测试，初步保证所开发代码的正确性。 另一方面，利用 Git 作为代码配置管理的工具，学会 Git 的基本使用方法。

* 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

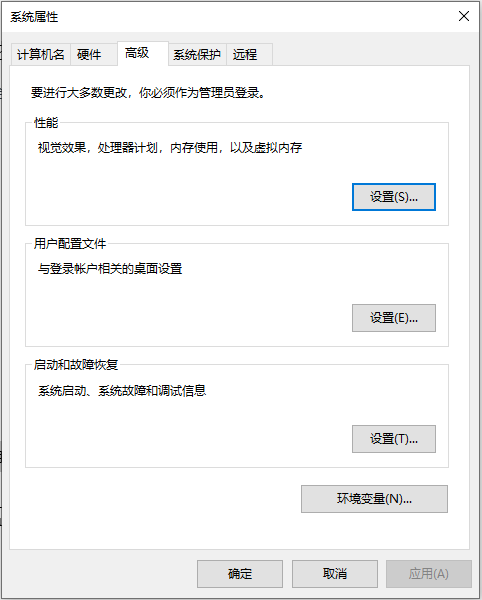
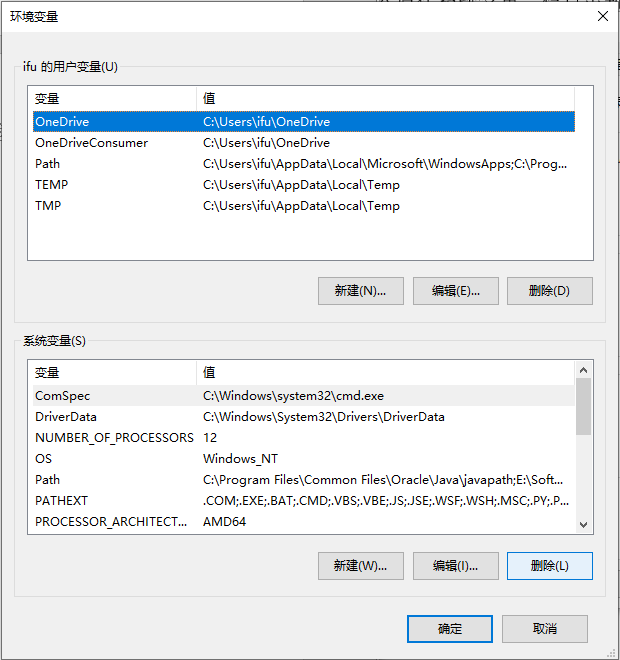
## 配置本次实验所需开发、测试、运行环境的过程

1. 下载安装JDK，版本15.0.2



1. 配置环境变量

计算机--属性--高级系统设置--环境变量新建JAVA\_HOME--用户变量增加CLASS\_PATH

1. 安装Eclipse



1. 下载安装git、配置git环境

## GitHub Lab1仓库的URL地址

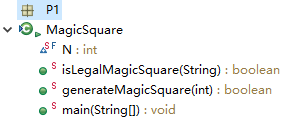
<https://github.com/ComputerScienceHIT/HIT-Lab1-1190202105>

# 实验过程

针对三个问题中的每一项任务，在下面各节中记录实验过程、阐述设计思路和问题求解思路，辅之以示意图或关键源代码加以说明。

## Magic Squares

此问题是要我们从读入的矩阵中判断是否为幻方，且根据算法能够得到任意阶幻方。首先什么是幻方，幻方是一个有对序列n的n×n数字排列，通常方阵里每个整数都不同，使所有行、所有列以及两个对角线总和为相同的常数，那么我们只需要验证上述条件是否满足。接下来是修改给出的代码自动生成幻方，并进行验证。要理解一段给出的代码添加注释并且画出流程图，判断输入值的情况并且处理异常。最后使用git将文件push到github仓库里面去。



### isLegalMagicSquare()

**设计思路**

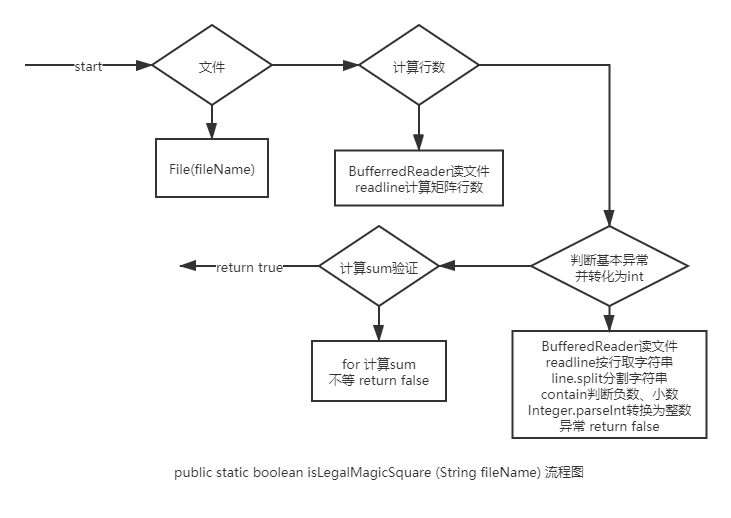
1. 读入文件

首先是创建FileReader、BufferdReader对象读入.txt文件，通过readline确定矩阵行数之后再次重新读取.txt文件。在这次读取中，我们首先通过String.split删除“\t”来分割一行字符串，排除行数与列数不相等、空格符号不是用“\t”等情况，判断是否存在小数和负数，即是否有“-”和“.”，若都满足则将字符串转化为int型数据。上述过程若满足“正整数方阵”这个要求将重复n次，直到文件再次被读完。若中途检查出不是正整数方阵这个要求，则直接报错，并返回false。

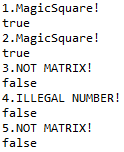
1. 计算比较每行、每列和对角和是否相等

首先计算第一行的和，将其设置为firstsum，用于与之后的每一个sum进行比较。然后通过嵌套的两个for循环，计算判断每一行、每一列以及两个对角和是否与firstsum相等。若存在不相等直接报错并返回false，若全部检测完并且没有任何错误则返回true。

**实现流程图**



**运行结果如下：**

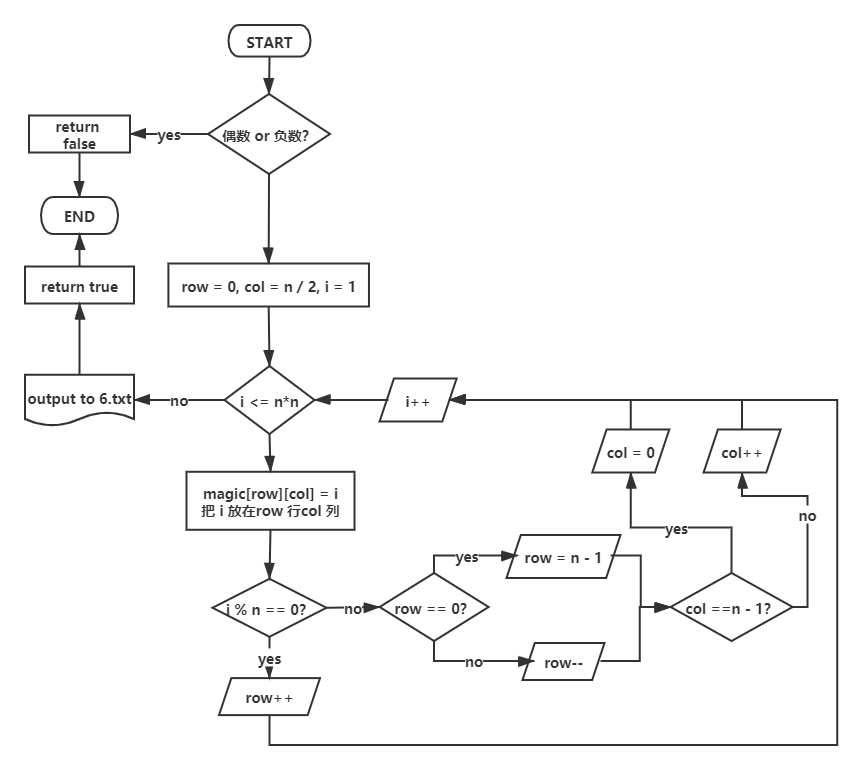


### generateMagicSquare()

**设计思路**

参数为一个奇数n，首先判断参数，如果是负数或偶数，输出提示并作异常处理。创建一个n×n的二维方阵，从第一行的中间开始，依次向右上角移动一格，并依次往里填充1到n2的数字。若遇到边界就返回到另一边，如遇到第一行，则跳到最后一行，遇到最后一列就跳到第一列，每输入n个数向下移动一格，从而填充完整个矩。

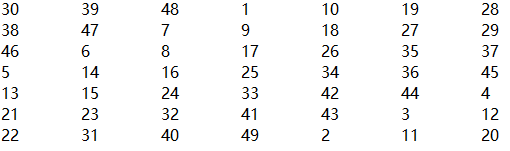
**流程图**



**运行结果**







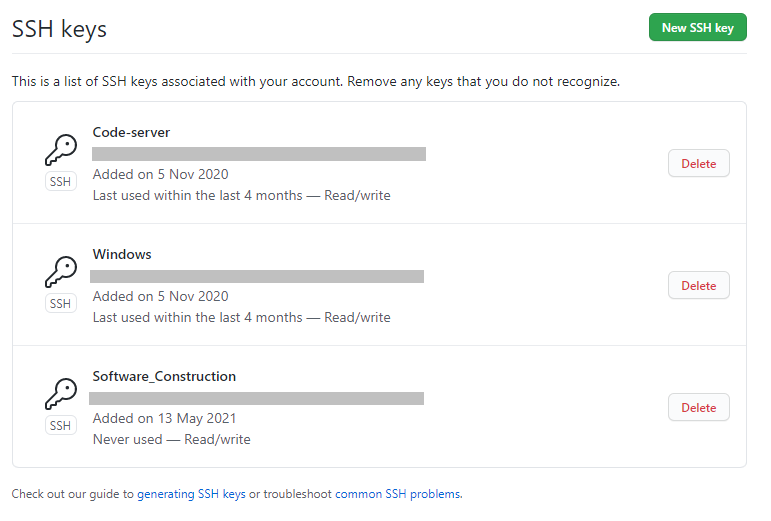
## Turtle Graphics

Clone已有代码，按照要求在文件implement各项任务，最后能用turtle作图。

### Problem 1: Clone and import

从GitHub获取该任务的代码、在本地创建git仓库、使用git管理本地开发。

* 在准备克隆代码的文件夹下，右键，选择Git Bash Here，打开控制台
* 输入ssh-keygen -t rsa -C email@email.com 生成 ssh key
* 回到github上，进入 Account Settings，添加 SSH Key
* git clone项目代码路径

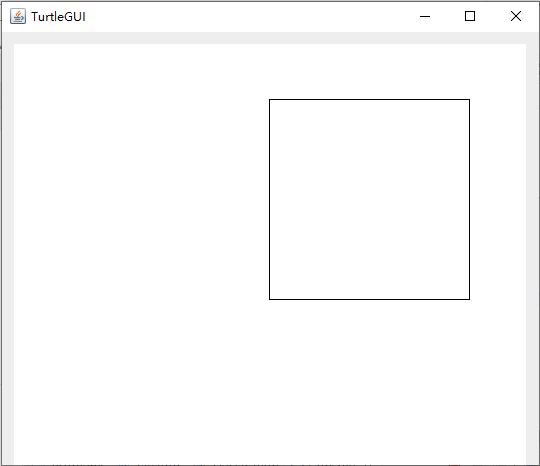


### Problem 3: Turtle graphics and drawSquare

**已知：**forward(units) 将海龟按units像素向当前方向移动， units是整数。turn(degrees) 将海龟按角度degrees顺时针旋转，degrees是双精度浮点数。

**要求：**画出边长为指定值的正方形，参数海龟对象 turtle 和编程 sidelength。

**实现方法：**执行四次前进sideLength长度和旋转90度即可。如下图，是sidelength为200的正方形。



### Problem 5: Drawing polygons

1. 首先Implement函数.calculateRegularPolygonAngle

*return (sides - 2) \* 180.0 / sides;*

通过边数来计算正多边形的每个内角大小；

1. Implement函数.calculatePolygonSidesFromrAngle

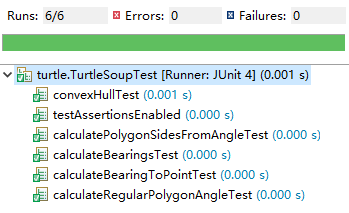
*return (int) Math.round(360.0 / (180.0 - angle));*

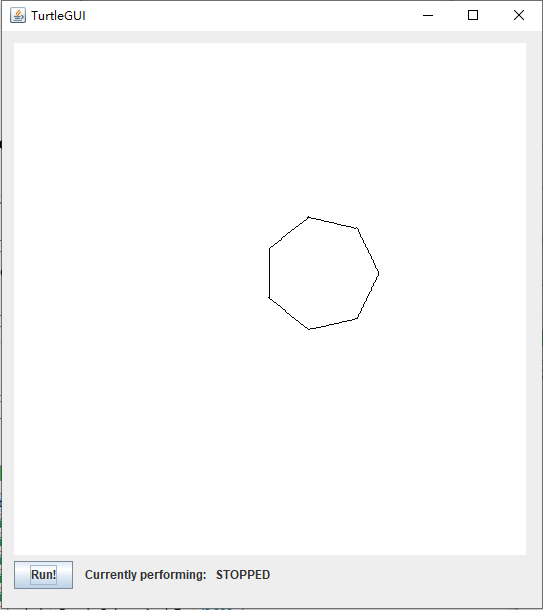
通过角度大小来计算正多边形变数；

1. 最后Implement函数.drawRegularPolygonAngle

利用给的边数sides计算出每个内角大小，每次旋转180.0 – 内角大小，每次前进sideLength长度，循环sides次即可。

1. 运行TurtleSoupTest.java中的Junit测试结果，以及sideLength为50的正七边形的绘制。





### Problem 6: Calculating Bearings

1. 首先Implement函数.calculateBearingToPoint，基于此首先明确Bearing大于等于0且小于360。我们先用Math.atan2计算两点形成的边与右向形成的弧度，再转化为正的角度。

*double radian = Math.atan2(……)；*

*if (radian < 0) radian += 2 \* Math.PI;*

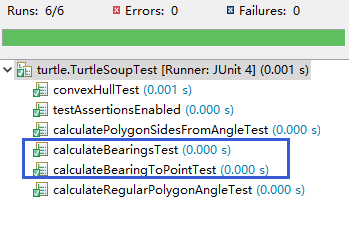
*double angle = radian / Math.PI \* 180.0;*

Turtle的0°是指向上的，但我们所计算的结果是基于右指向的坐标系，因此再将angle转化为正确的表示。

*angle = angle > 90 ? 90 + 360 – angle : 90 – angle;*

最后减去当前角度，将旋转角度转化为0到360范围内的正数。

1. 然后Implement函数.calculateBearings时，将起点作为第一个点，反复调用. calculateBearingToPoint循环xCoords.size次，生成数据储存在List，最后返回一个List<double>。
2. 运行TurtleSoupTest.java中的Junit测试结果。



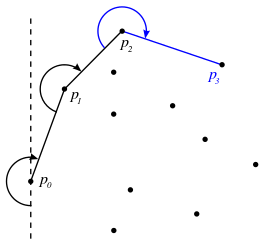
### Problem 7: Convex Hulls

1. 首先对Gift Wrapping Algorithm进行分析。

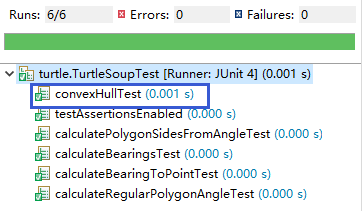
首先确定所有点中最右下角的点为原点，去点集合里面寻找另一个点，使得上一个点到该点旋转角度最小，使其包含进Set<Point> convexHull，若算得多个点角度一样，则取最最远的那个点。反复进行上述操作，直到最后一个点旋转最小的点是原点，所得到的集合Set就是所求的Convex Hull。

1. 什么是凸包问题

假设平面上有多个点，过某些点作一个多边形，使这个多边形能把所有点都“包”起来。当这个多边形是凸多边形的时候，我们就叫它“凸包”。



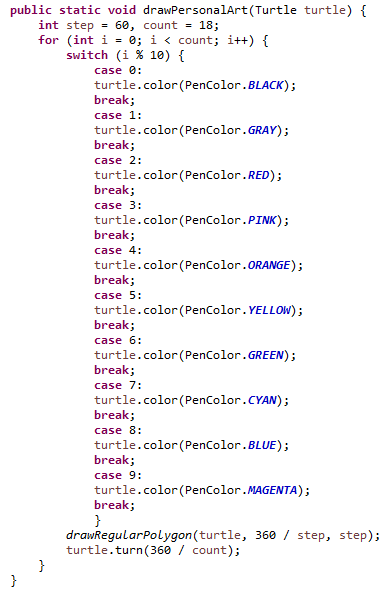
1. Junit测试结果



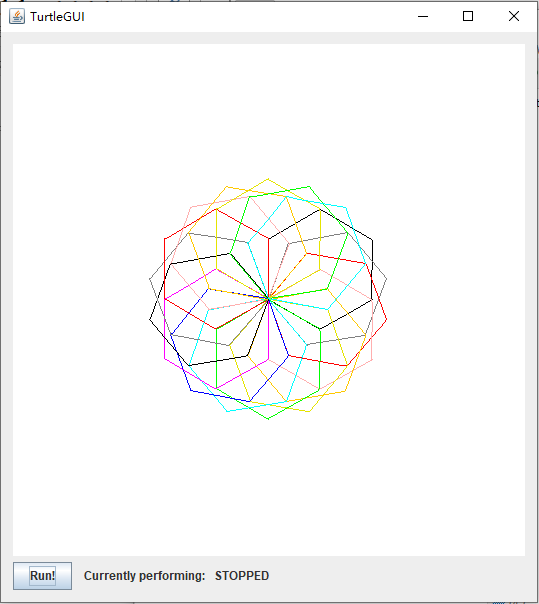
### Problem 8: Personal art

#### 代码与思路

从原点开始，每画一个多边形则转一定的角度、变一个颜色，直到回到最初的方向。其中step表示边长，count表示通过该次旋转回到开始的方向。



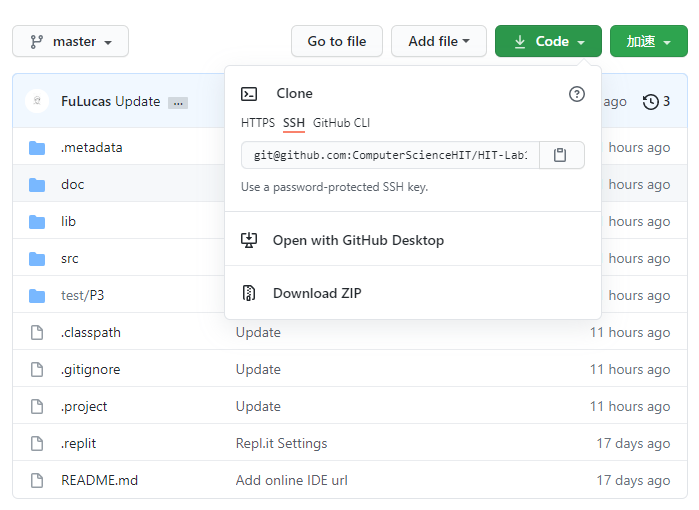
#### 作画结果



### Submitting

通过Git提交当前版本到GitHub上你的Lab1仓库。

#### 添加远程仓库 URL

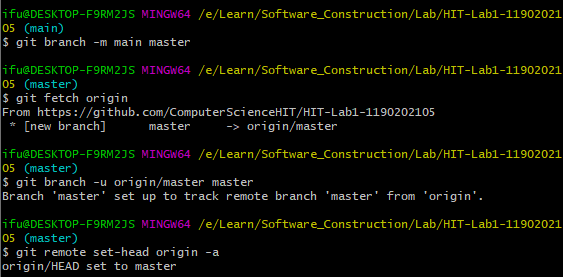


输入ssh-keygen -t rsa -C email@email.com 生成 ssh key

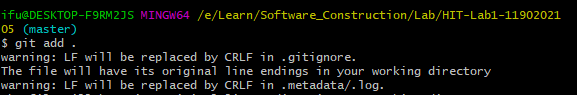
到github添加 SSH Key

如上图获取项目代码路径，克隆一个github 项目到本地

#### 修改main分支为master分支

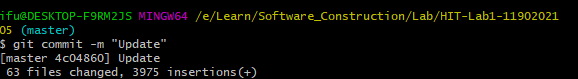


#### 添加上传文件



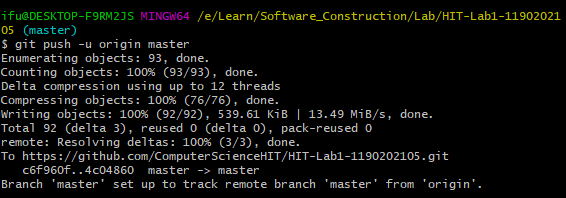
Git add . 前后都有空格的点，添加所有文件

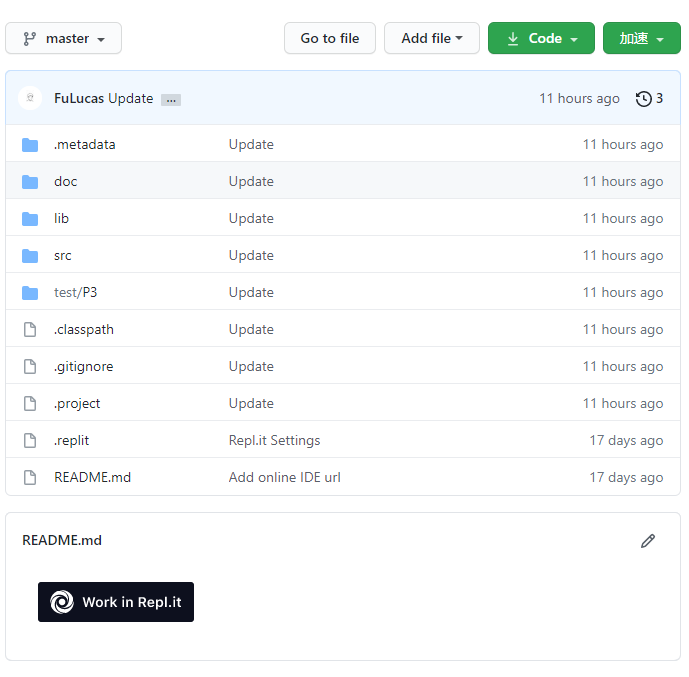
#### 修改日志



Git commit -m “update” 修改文件新日志

#### 上传

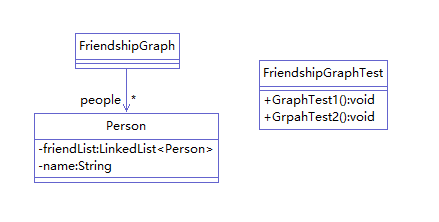




Git push -u origin master

## Social Network

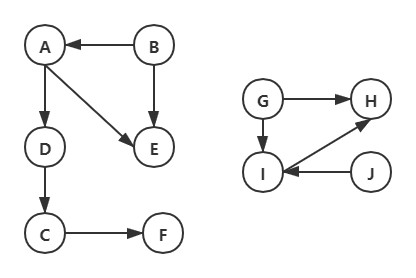
要求设计一张社交网络图，基于连接人与人，并且能计算任意两人 之间的联系情况。网络图基于两个类，分别是 FriendshipGraph 类和 Person 类。实际上就是构建有个有向图，使得可以计算两个点之间的最短距离，在构建图的过程中也能应对一些异常情况。

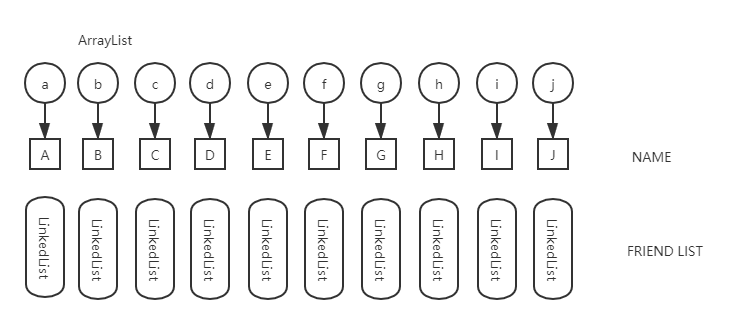


### 设计/实现FriendshipGraph类

给出你的设计和实现思路/过程/结果。

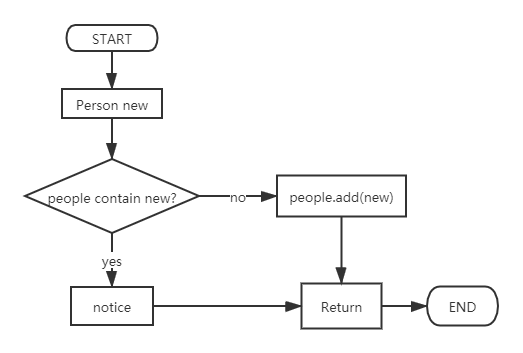
如图所示，创建网络时，将Person添加到Friend List当中，形成了一个有向图，姑且不考虑朋友之间的相互性，即无向性。接下来按照给出的例子，要对Person、addVertex、addEdge、getDistance进行分析。





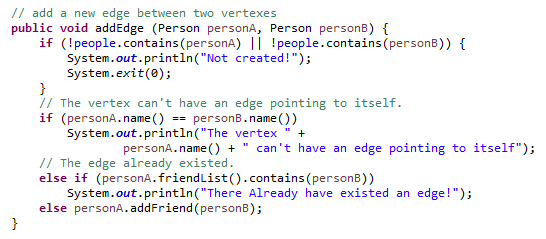
#### Method addVertex()

创建节点的时候，若是个新人，则直接加入List<Person>，否则提示重复，但不会整个程序全部退出。



#### Method addEdge()

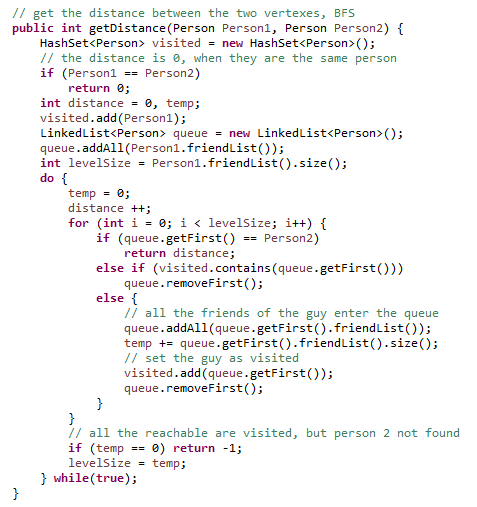
addEdge的功能是为不同的顶点之间添加边，由于类要保留拓展到有向图的功能，所以每一次运行addEdge都只会添加一条单向边。判断两个人是否都在集合people中，若否，则打印错误提示信息，并退出程序；若是，则朋友列表中添加相应的Person。



#### Method getDistance()

1. BFS算法：从原始顶点开始一层一层地进行搜索，每经过一次顶点标记为visited（这里设置一个Set<Person> visited），下经过就直接跳过，并且把它的非终点全部入队；每搜索完一层则distance++；若达到终点则返回层数。若最终队列里面不存在任何顶点则表示没有从原点到终点的路径。程序最开始还对两点的相同性进行检测，若是相同直接返回0.
2. 选择List储存全部顶点，每个顶点又是一个Person，存有自己的名字和朋友列表（Person），那么就形成了一个有向图。

***代码如下：***



### 设计/实现Person类

由上述可以看出，每一个人对应到一个Person对象，且至少包含人的名字和朋友列表，定义几个方法分别返回名字name()、返回朋友列表friendList()、添加新朋友addFriend()，如下所示。

**public** **class** Person {

**private** String name; // Save the name

**private** LinkedList<Person> friendList; // Save one's friends

// Method

**public** Person(String str) {

**this**.name = str;

friendList = **new** LinkedList<Person>();

}

// add friend

**public** **void** addFriend(Person newFriend) {

friendList.add(newFriend);

}

// get one's name

**public** String name() {

**return** name;

}

// get one's friend list

**public** List<Person> friendList() {

**return** **this**.friendList;

}

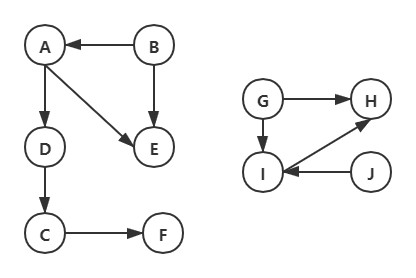
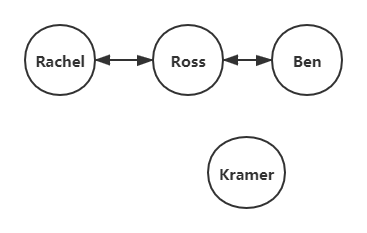
}

### 设计/实现客户端代码main()

直接使用的是实验要求报告中给出的实例。

### 设计/实现测试用例

给出两个测试，首先按照下图进行创建关系，一是简单的实验要求上的，另一个是自己设计的。



创建代码分别有：

*图1：*

Person rachel = **new** Person("Rachel");

Person ross = **new** Person("Ross");

Person ben = **new** Person("Ben");

Person kramer = **new** Person("Kramer");

graph.addVertex(rachel);

graph.addVertex(ross);

graph.addVertex(ben);

graph.addVertex(kramer);

graph.addEdge(rachel, ross);

graph.addEdge(ross, rachel);

graph.addEdge(ross, ben);

graph.addEdge(ben, ross);

*图2：*

Person a = **new** Person("A");

Person b = **new** Person("B");

Person c = **new** Person("C");

Person d = **new** Person("D");

Person e = **new** Person("E");

Person f = **new** Person("F");

Person g = **new** Person("G");

Person h = **new** Person("H");

Person i = **new** Person("I");

Person j = **new** Person("J");

graph.addVertex(a);

graph.addVertex(b);

graph.addVertex(c);

graph.addVertex(d);

graph.addVertex(e);

graph.addVertex(f);

graph.addVertex(g);

graph.addVertex(h);

graph.addVertex(i);

graph.addVertex(j);

graph.addEdge(a, d);

graph.addEdge(a, e);

graph.addEdge(b, a);

graph.addEdge(b, e);

graph.addEdge(c, f);

graph.addEdge(d, c);

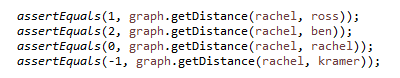
graph.addEdge(g, h);

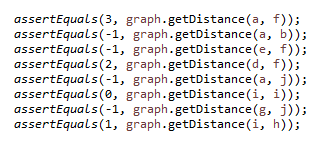
graph.addEdge(g, i);

graph.addEdge(i, h);

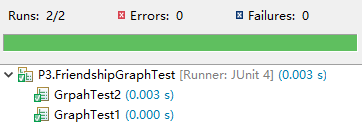
graph.addEdge(j, i);

它们的测试用例如下：





Junit检测结果：



# 实验进度记录

请使用表格方式记录你的进度情况，以超过半小时的连续编程时间为一行。

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 任务 | 实际完成情况 |
| 2021-05-13 | 14:00-15:30 | 配置git环境、clone所需文件，编写问题1的isLegalMagicSquare函数并进行测试 | 延期两小时完成 |
| 2021-05-16 | 18:30-20:30 | 编写问题1的generateMagicSquare函数，及其他剩余内容 | 按计划完成 |
| 2021-05-16 | 21:00-23:20 | 编写问题2的drawSquare函数、calculateRegularPolyonAngle函数、calculatePolyonSidesFromAngle函数、calculateBearingToPoint函数、calculateBearings函数，并进行Junit测试 | 按计划完成  最后两个函数耗时很长 |
| 2021-05-17 | 19:00-23:00 | 编写问题2的convexHull函数 | 遇到困难，未完成 |
| 2021-05-19 | 19:00-21:00 | 编写问题2的convexHull函数 | 按计划完成 |
| 2021-05-19 | 21:30-22:00 | 编写问题2的drawPersonalArt函数 | 延期半小时完成 |
| 2021-05-22 | 17:00-24:00 | 编写问题3 | 延期一小时完成 |
| 2021-05-23 | 9:00-15:00 | 撰写报告 | 遇到困难，未完成 |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的困难 | 解决途径 |
| 对于Eclipse环境不熟悉 | 查阅资料、询问同学 |
| 问题2的calculateBearingToPoint函数编写过程，最开始没有发现坐标系的转换。 | 翻看其他代码，发现最开始的方向是向上的，而用atan2计算出来的弧度是相对于指向右的坐标，因此对函数返回值进行变换。 |
| 问题2的convexHull的编写存在问题，对于选点的操作进行过好几个版本的替代 | 参考网上博客、维基百科的概念和解决方案等 |
| 问题3的BFS算法不是很熟，同时设计的Person在逻辑上不是很易于思考 | 带入C编写的逻辑，设计了一个visited来储存访问过的person |
| 3个问题在一个文件里面不知怎么创建过程 | 未解决，分别创建三个工程来编写三个代码，再手动合并在一起 |

# 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

1. 更加熟悉了Java中Set、List等的操作和内在联系。
2. 还是不习惯Java中有些语句的简单操作，有的时候习惯性地带入C的编写习惯，造成时间浪费。、
3. 对类、接口、方法等概念不熟悉。
4. 在写之前应该先写伪代码，防止错误。
5. 要学会查阅资料、文档。

## 针对以下方面的感受

1. Java编程语言是否对你的口味？

比较适合我的胃口，比较易懂，并且带有许多简单操作，节约了不少时间。

1. 关于Eclipse IDE；

支持插件安装支持，但是很多冲突都不知道是怎么回事，代码补全来不及配置，一般是用vscode编写代码。建议教vscode的配置。

1. 关于Git和GitHub；

首先，github的使用很艰难，需要翻墙才能稳定操作；git的使用还不熟练，对于分支的操作更是不熟，但是初步接触下来，觉得是一个很强大的工具。

1. 关于CMU和MIT的作业；

实验内容多，带有趣味性，很有意思但有的时候耗时有些长

1. 关于本实验的工作量、难度、deadline；

工作量大、难度也大、deadline有点早、实验和考试时间有冲突

1. 关于初接触“软件构造”课程；

很多东西都不会，简单的Java基础并不能支持完成这次实验