

**2021年春季学期  
计算学部《软件构造》课程**

**Lab 1实验报告**

|  |  |
| --- | --- |
| 姓名 | 谭方舟 |
| 学号 | 1190201610 |
| 班号 | 1903009 |
| 电子邮件 | 1975411670@qq.com |
| 手机号码 | 15056289830 |

**目录**

[1 实验目标概述 1](#_Toc71131534)

[2 实验环境配置 1](#_Toc71131535)

[3 实验过程 1](#_Toc71131536)

[3.1 Magic Squares 1](#_Toc71131537)

[3.1.1 isLegalMagicSquare() 1](#_Toc71131538)

[3.1.2 generateMagicSquare() 1](#_Toc71131539)

[3.2 Turtle Graphics 1](#_Toc71131540)

[3.2.1 Problem 1: Clone and import 2](#_Toc71131541)

[3.2.2 Problem 3: Turtle graphics and drawSquare 2](#_Toc71131542)

[3.2.3 Problem 5: Drawing polygons 2](#_Toc71131543)

[3.2.4 Problem 6: Calculating Bearings 2](#_Toc71131544)

[3.2.5 Problem 7: Convex Hulls 2](#_Toc71131545)

[3.2.6 Problem 8: Personal art 2](#_Toc71131546)

[3.2.7 Submitting 2](#_Toc71131547)

[3.3 Social Network 2](#_Toc71131548)

[3.3.1 设计/实现FriendshipGraph类 2](#_Toc71131549)

[3.3.2 设计/实现Person类 2](#_Toc71131550)

[3.3.3 设计/实现客户端代码main() 2](#_Toc71131551)

[3.3.4 设计/实现测试用例 3](#_Toc71131552)

[4 实验进度记录 3](#_Toc71131553)

[5 实验过程中遇到的困难与解决途径 3](#_Toc71131554)

[6 实验过程中收获的经验、教训、感想 3](#_Toc71131555)

[6.1 实验过程中收获的经验和教训 3](#_Toc71131556)

[6.2 针对以下方面的感受 3](#_Toc71131557)

# 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/IDEA IDE 进行 Java 编程

⚫ 基于 JUnit 的测试

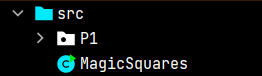
⚫ 基于 Git 的代码配置管理

# 实验环境配置

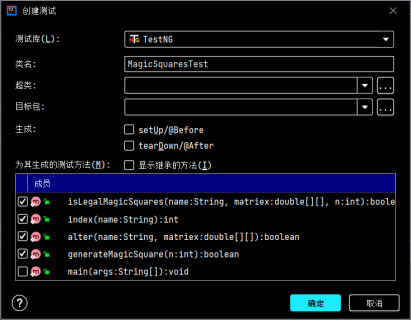
配置过程：

1、下载IntelliJ IDEA Community Edition 2020.3.2 x64。

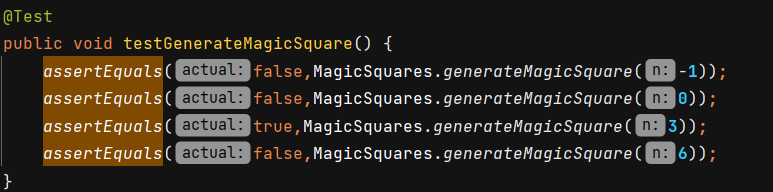
2、打开IDEA，在File中点开Setting，然后继续点击Plugging，下载Java开发所需要的插件以及中文插件

3、在IDEA中创建一个Java的工程，在src文件中编写工程的源码。

4、测试之前下载Junit插件用于测试。在工程所在的文件中添加一个名为test的文件夹，并且将文件夹标记为测试资源。这样测试用例就会自动在test中生成。在创建测试用例的时候首先在工程界面右键点击我们创建的类，然后点击生成->测试，在弹出的窗口中选择我们需要测试的成员方法。



1. 每写完一个成员函数，就在test文件夹中的测试文件中添加一个测试方法来测试成员方法的功能是否正常。



1. 在完成所有的成员方法的编写和测试之后，运行程序来完成任务。

困难与解决方法：

1. 在编写测试用例的时候，发现在使用assertEquals函数的时候没有代码自动补全的功能，而且在测试用例编写完成之后，测试程序无法运行。查阅相关资料之后发现是因为测试程序的配置有问题，好在IDEA有自动配置测试环境的功能，在测试用例中点击@Test,在窗口的上方会出现配置环境的选项，配置完成之后就每个测试方法之前就会出现测试的按钮：



1. 在完成测试环境的配置之后，在使用assertEquals函数也出现了问题，查阅资料之后发现该函数不是Java自带的函数，需要导入相关的类。在参考其他使用该函数的测试文件之后，在测试文件中导入了org.testng.annotations.Test和static org.testng.Assert.\*。

URL地址：

# 实验过程

## Magic Squares

该实验通过对幻方的检验与生成，不仅考察了对于Java的文件读取、字符串处理、数组处理方面的能力，而且通过对幻方生成条件的判断考察对于Java中数组的边界条件以及将程序运行过程可视化的能力。

### isLegalMagicSquare()

实现思路：如果一个储存在文件中的二维数组能符合实验的部分要求（数字之间用制表符隔开、没有缺少的元素）转换成为一个矩阵，那么我们首先判断该矩阵中是否存在非正数，如果存在非正数则返回false，否则对矩阵中的每一行每一列以及两个对角线求和，在所求的和之中如果存在不一样的元素则返回false否在返回true。

实现过程：

public static boolean isLegalMagicSquares**(**String name**,**double**[][]** matriex**,**int n**){**

double**[]** num**=new** double**[**2**\***n**+**2**];**

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

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

**if(**matriex**[**i**][**j**]<=**0**){**

System**.**out**.**println**(**name**+**"存在非正整数数。"**);**

**return** **false;**

**}**

num**[**i**]+=**matriex**[**i**][**j**];**

num**[**i**+**n**]+=**matriex**[**j**][**i**];**

**}**

**}**

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

num**[**2**\***n**]** **+=** matriex**[**i**][**i**];**

num**[**2**\***n**+**1**]** **+=** matriex**[**i**][**n**-**1**-**i**];**

**}**

**for(**int i**=**0**;**i**<**2**\***n**+**1**;**i**++)** **{**

**for(**int j**=**i**+**1**;**j**<**2**\***n**+**2**;**j**++){**

**if(**num**[**i**]!=**num**[**j**]){**

**return** **false;**

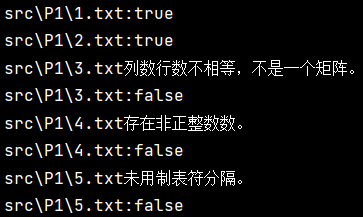
**}**

**}**

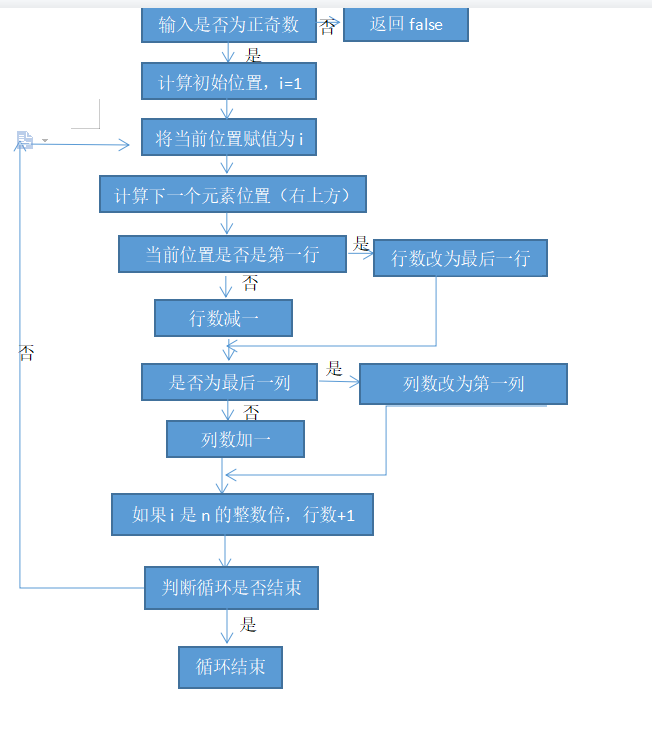
**}**

**return** **true;**

**}**

**结果：**

### generateMagicSquare()

实现思路：

实现过程： public static boolean generateMagicSquare**(**int n**)** **{**

**if(**n**<=**0**){**

System**.**out**.**println**(**"输入的n是非整数"**);**

**return** **false;** **}**

int magic**[][]** **=** **new** int**[**n**][**n**];**

double Magic**[][]** **=** **new** double**[**n**][**n**];**

int row **=** 0**,** col **=** n **/** 2**,** i**,** j**,** square **=** n **\*** n**;**

**for** **(**i **=** 1**;** i **<=** square**;** i**++)** **{**

magic**[**row**][**col**]** **=** i**;**

**if** **(**i **%** n **==** 0**){**

row**++;**

**if(**row**>=**n**&&**i**<**square**)** **{**

System**.**out**.**println**(**"输入的n是一个偶数。"**);**

**return** **false;** **}}**

**else** **{**

**if** **(**row **==** 0**)**row **=** n **-** 1**;**

**else** row**--;**

**if** **(**col **==** **(**n **-** 1**))**col **=** 0**;**

e**lse** col**++;** **}** **}**

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

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

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

System**.**out**.**println**();**

**}**

File file **=** **new** File**(**"src\\P1\\6.txt"**);**

**try** **{**

FileWriter out **=** **new** FileWriter**(**file**);**

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

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

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

Magic**[**i**][**j**]** **=** **(**double**)** magic**[**i**][**j**];** **}**

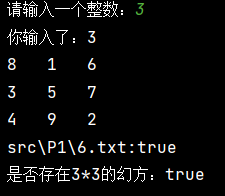
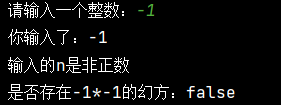
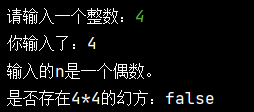
out**.**write**(**"\r\n"**);** **}**

out**.**close**();**

**}** **catch** **(**IOException e**)** **{**

e**.**printStackTrace**();** **}**

System**.**out**.**println**(**"src\\P1\\6.txt:"**+**isLegalMagicSquares**(**"src\\P1\\6.txt"**,**Magic**,**n**));** **return** **true;** **}**

**结果：**

## Turtle Graphics

首先补充海龟图的相关代码不仅要求要有良好的几何基础，计算图形的闭包的时候更要求要有相关的算法知识，以及对Java成员方法的理解和Java标准方法的运用。在补全成员方法的时候要求对Java的抽象类有足够的认识。

### Problem 1: Clone and import

从Github上获取代码：网页download。

从本地创建git仓库：

1. git config 初始化用户名和邮箱
2. 输入git clone https://github.com/ComputerScienceHIT/Lab1-1190201610.git进行关联。

3、git init初始化本地仓库

4、git add remote添加远程库源

5、在远程仓库创建master分支

6、git pull origin master将远程仓库同步到本地

7、git add \* ->git commit -m “”->git push将本地文件加入到本地仓库，将本地仓库同步到远程仓库。

### Problem 3: Turtle graphics and drawSquare

实现思路：只要海龟走一条边再转一次90°，循环四次就可以。

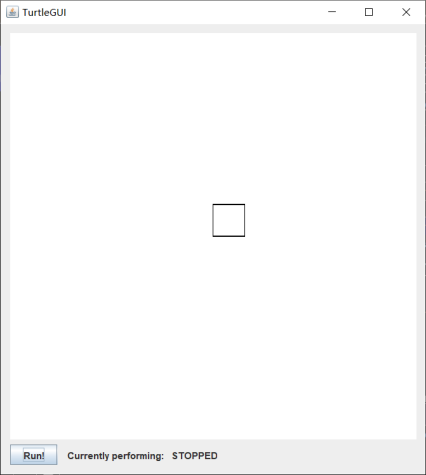
实现过程：

public static void drawSquare**(**Turtle turtle**,** int sideLength**){**

**for(**int i **=** 0**;**i**<**4**;**i**++)** **{**

turtle**.**forward**(**sideLength**);**

turtle**.**turn**(**90**);}** **}**

**结果：**

### Problem 5: Drawingpolygons

实现思路：先计算出正多边形的内角，然后再让海龟走一条边，转一次外角然后循环边数次。

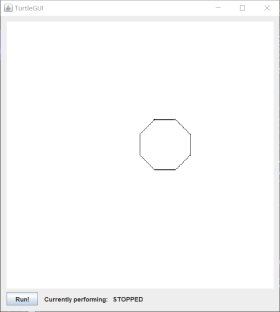
实现过程：public static void drawRegularPolygon**(**Turtle turtle**,** int sides**,** int sideLength**){**

double angle **=** 180.0**-**calculateRegularPolygonAngle**(**sides**);**

**for(**int i**=**0**;**i**<**sides**;**i**++){**

turtle**.**forward**(**sideLength**);**

turtle**.**turn**(**angle**);}}**

**结果：**

### Problem 6: CalculatingBearings

实现思路：因为已经写好了成员方法calculatebearingtopoint（），所以在实现该方法的时候我们只需要将前后两个函数调用calculatebearingtopoint（）就可以。

实现过程：public static List**<**Double**>** calculateBearings**(**List**<**Integer**>** xCoords**,** List**<**Integer**>** yCoords**)** **{**

List**<**Double**>**angles **=** **new** ArrayList**<**Double**>();**

double currentBearing **=** 0.0**,**angle**;**

**if(**xCoords**.**size**()** **<** 2**)** **{**

angle**=**calculateBearingToPoint**(**currentBearing**,**xCoords**.**get**(**0**),**yCoords**.**get**(**0**),**

xCoords**.**get**(**1**),**yCoords**.**get**(**1**));**

angles**.**add**(**angle**);**

**return** angles**;**

**}**

**for(**int i **=** 0**;**i **<** xCoords**.**size**()-**1**;**i**++){**

Angle=calculateBearingToPoint**(**currentBearing**,**xCoords**.**get**(**i**),**yCoords**.**get**(**i**),**

xCoords**.**get**(**i**+**1**),**yCoords**.**get**(**i**+**1**));**

angles**.**add**(**angle**);**

currentBearing **=** **(**currentBearing**+**angle**)%**360**;**

**}**

**return** angles**;**

**}**

**结果：**

### Problem 7: ConvexHulls

实现思路：任意凸包上的点，以该点建立一个极角坐标系，该点连结其它所有点的极角中，该点逆时针方向的第一凸包点到该点极角最小，例如P0，到所有点的极角中P0P1极角最小。

算法中首先找到最左边或者最下面的的点，这个点必然在凸包上，然后以该点和上一个点连线作为极轴计算该点连接点极角最小的，循环下去，直到找到最初的起点。在计算极角的时候需要用到相似三角形、余弦定理、反三角函数的知识。

实现过程：public static Set**<**Point**>** convexHull**(**Set**<**Point**>** points**)** **{**

**if** **(**points**.**size**()** **<=** 3**)** **{** **return** points**;** **}**

Point**[]** P **=** points**.**toArray**(new** Point**[**points**.**size**()]);**

Point a **=** P**[**0**];**

Set**<**Point**>** convexHullPoints **=** **new** HashSet**<**Point**>();**

**for** **(**Point i **:** points**)** **{**

**if** **(**i**.**x**()** **<** a**.**x**()** **||** **(**i**.**x**()** **==** a**.**x**()** **&&** i**.**y**()** **<** a**.**y**()))**  a **=** i**;**

**}**

Point current **=** a**,** min **=** **null,** last **=** a**;**

double x1 **=** 0.0**,** y1 **=** **-**1.0**;**

**do** **{**

convexHullPoints**.**add**(**current**);**

double minangle **=** Double**.**MAX\_VALUE**,** x2 **=** 0.0**,** y2 **=** 0.0**;**

**for** **(**Point i **:** points**)** **{**

**if** **((!**convexHullPoints**.**contains**(**i**)** **||** i **==** a**)** **&&** **(**i **!=** last**))** **{**

double dx **=** i**.**x**()** **-** current**.**x**(),** dy **=** i**.**y**()** **-** current**.**y**();**

double anlge **=** Math**.**acos**((**x1**\*** dx **+** dy **\*** y1**)** **/** Math**.**sqrt**(**dx **\*** dx **+** dy **\*** dy**)/**Math**.**sqrt**(**x1**\***x1**+**y1**\***y1**));**

**if** **(** **(**anlge **==** minangle **&&** dx **\*** dx **+** dy **\*** dy **>** Math**.**pow**(**i**.**x**()** **-** min**.**x**(),** 2**)** **+** Math**.**pow**(**i**.**y**()** **-** min**.**y**(),** 2**))||**anlge **<** minangle **)** **{**

min **=** i**;** minangle **=** anlge**;** x2 **=** dx**;** y2 **=** dy**;** **}** **}}**

x1 **=** x2**;** y1 **=** y2**;**last **=** current**;** current **=** min**;**

**}** **while** **(**current **!=** a**);return** convexHullPoints**;** **}**

**结果：**

### Problem 8: Personal art

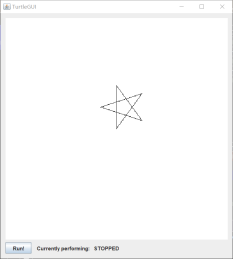
实现思路：用turtle.forward和turtle.turn画一个五角星。

实现过程： public static void drawPersonalArt**(**Turtle turtle**){**

**for(**int i**=**0**;**i**<**5**;**i**++){**

turtle**.**forward**(**100**);**

turtle**.**turn**(**144**);** **}}**

结果：

### Submitting

1. git add .将所有的提交文件添加到本地库
2. git commit -m “注释”
3. git push将本地库提交到远程Lab1仓库

## Social Network

这项实验需要建立两个类：一个类储存个人信息比如个人的姓名、这个人所交的朋友有哪些；另外一个类用来储存所有人的集合，以及哪些人直接存在单向的朋友关系。这个不仅考察有向图的知识而且在getdistance（）方法中考察了广搜的知识。

### 设计/实现FriendshipGraph类

设计：在FriendshipGraph（）类中首先需要为该类添加两个属性，一个是人名的String集合names，一个是个人的Person类的集合crow。而且根据需要，添加addVertex（）方法，在crow中添加个人以及在names中添加人名；添加方法addEgde（），在一方的friends中添加另外一个人作为朋友；还有一个方法getdistance（）用广度优先搜索的算法来获取两个人之间的距离，如果作为参数的两个人是同一个人则返回0，如果单方面没有朋友关系就返回-1。

实现过程：public class FriendshipGraph **{**

public List**<**Person**>**crow **=** **new** ArrayList**<**Person**>();**

public Set**<**String**>**names **=** **new** HashSet**<**String**>();**

public void addVertex**(**Person a**){**

**if(**names**.**contains**(**a**.**myname**())){**

System**.**out**.**println**(**"已存在名为"**+**a**.**myname**()+**"的人。"**);**

System**.**exit**(**0**);** **}**

crow**.**add**(**a**);**

names**.**add**(**a**.**myname**());** **}**

public void addEdge**(**Person p**,**Person f**){**

**if(**p**.**Freiends**().**contains**(**f**)){**

System**.**out**.**println**(**f**.**myname**()+**"已经是"**+**p**.**myname**()+**"的朋友。"**);**

System**.**exit**(**0**);** **}**p**.**newfriend**(**f**);}**

public int getDistance**(**Person p1**,**Person p2**){**

**if(**p1**==**p2**){** **return** 0**;}**

Queue**<**Person**>**queue **=** **new** LinkedList**<>();**

queue**.**offer**(**p1**);**

Map**<**Person**,**Integer**>**graph **=** **new** HashMap**<>();**

graph**.**put**(**p1**,**0**);**

**while** **(!**queue**.**isEmpty**()){**

Person per **=** queue**.**poll**();**

List**<**Person**>**Friends **=** per**.**Freiends**();**

**for(**Person e**:** Friends**){**

**if(!**graph**.**containsKey**(**e**)){**

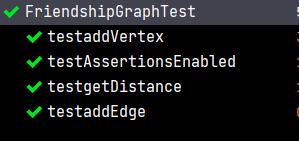
queue**.**offer**(**e**);** graph**.**put**(**e**,**graph**.**get**(**per**)+**1**);**

**if(**e**==**p2**){**

**return** graph**.**get**(**p2**);**

**}** **}** **}** **}**

**return** **-**1**;** **}}**

**结果：**

### 设计/实现Person类

设计：在Person类中应该有两个属性，一个是个人的名字name还有一个是一个人的所交的朋友FriendList。在Person类中还应该添加方法newfriend（）来在FriendList中添加元素来表示添加朋友，添加myname（）方法来返回这个人的姓名，添加Friends（）方法来返回这个人有哪些朋友。

实现过程：public class Person **{**

private String name**;**

private List**<**Person**>** FriendList**;**

public Person**(**String name**){**

**this.**name **=** name**;**

FriendList **=** **new** ArrayList**<>();**

**}**

public void newfriend**(**Person p**){**

FriendList**.**add**(**p**);**

**}**

public String myname**(){**

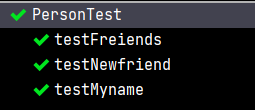
**return** **this.**name**;**

**}**

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

**return** FriendList**;**

**}}**

**结果：**

### 设计/实现客户端代码main()

设计：在main（）中，新建不同的Person，通过相互建立单方面的朋友关系形成有向图，然后通过getdistance（）方法来获取每个人之间的距离。然后按实验要求去掉 graph.addEdge(rachel, ross);这段代码看看实验结果会是怎么样。

实验过程:public static void main**(**String argv**[]){**

FriendshipGraph graph **=** **new** FriendshipGraph**();**

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**);**

System**.**out**.**println**(**graph**.**getDistance**(**rachel**,** ross**));** System**.**out**.**println**(**graph**.**getDistance**(**rachel**,** ben**));**

System**.**out**.**println**(**graph**.**getDistance**(**rachel**,** rachel**));**

System**.**out**.**println**(**graph**.**getDistance**(**rachel**,** kramer**));}**

**结果：**去掉graph.addEdge(rachel, ross);后

### 设计/实现测试用例

设计：在每个测试用例中通过创建不用的Person个体然后再Person中添加朋友，通过查看朋友集合中的最后一个元素是否与预期的相符合来判断方法的功能是否正常。

实现过程

@Testpublic void testaddVertex**(){**

FriendshipGraph graph **=** **new** FriendshipGraph**();**

Person Rose **=** **new** Person**(**"Rose"**);**

Person Jay **=** **new** Person**(**"Jay"**);**

Person Joe **=** **new** Person**(**"Joe"**);**

graph**.**addVertex**(**Rose**);**

assertEquals**(**"Rose"**,**graph**.**crow**.**get**(**graph**.**crow**.**size**()-**1**).**myname**());**

graph**.**addVertex**(**Jay**);**

assertEquals**(**Jay**,**graph**.**crow**.**get**(**graph**.**crow**.**size**()-**1**));**

graph**.**addVertex**(**Joe**);**

assertEquals**(**Joe**,**graph**.**crow**.**get**(**graph**.**crow**.**size**()-**1**));**

**}**

@Test public void testaddEdge**(){**

FriendshipGraph graph **=** **new** FriendshipGraph**();**

Person Rose **=** **new** Person**(**"Rose"**);**

Person Jay **=** **new** Person**(**"Jay"**);**

graph**.**addVertex**(**Rose**);**

graph**.**addVertex**(**Jay**);**

assertEquals**(**0**,**graph**.**crow**.**get**(**0**).**Freiends**().**size**());**

assertEquals**(**0**,**graph**.**crow**.**get**(**1**).**Freiends**().**size**());**

graph**.**addEdge**(**Rose**,**Jay**);**

assertEquals**(**Jay**,**graph**.**crow**.**get**(**0**).**Freiends**().**get**(**0**));**

graph**.**addEdge**(**Jay**,**Rose**);**

assertEquals**(**Rose**,**graph**.**crow**.**get**(**1**).**Freiends**().**get**(**0**));**

**}**

@Test public void testgetDistance**(){**

FriendshipGraph graph **=** **new** FriendshipGraph**();**

Person Rose **=** **new** Person**(**"Rose"**);**

Person Jay **=** **new** Person**(**"Jay"**);**

Person Joe **=** **new** Person**(**"Joe"**);**

Person Alice **=** **new** Person**(**"Alice"**);**

graph**.**addVertex**(**Rose**);**

graph**.**addVertex**(**Jay**);**

graph**.**addVertex**(**Joe**);**

graph**.**addVertex**(**Alice**);**

graph**.**addEdge**(**Rose**,**Jay**);**

assertEquals**(**1**,**graph**.**getDistance**(**Rose**,**Jay**));**

graph**.**addEdge**(**Jay**,**Joe**);**

assertEquals**(**2**,**graph**.**getDistance**(**Rose**,**Joe**));**

graph**.**addEdge**(**Rose**,**Joe**);**

assertEquals**(**1**,**graph**.**getDistance**(**Rose**,**Joe**));**

assertEquals**(-**1**,**graph**.**getDistance**(**Rose**,**Alice**));**

**}**

public void testNewfriend**()** **{**

Person Rose **=** **new** Person**(**"Rose"**);**

Person Jay **=** **new** Person**(**"Jay"**);**

Rose**.**newfriend**(**Jay**);**

assertEquals**(**Jay**,**Rose**.**Freiends**().**get**(**0**));**

**}**

public void testMyname**()** **{**

Person Rose **=** **new** Person**(**"Rose"**);**

assertEquals**(**"Rose"**,**Rose**.**myname**());**

**}**

public void testFreiends**()** **{**

Person Rose **=** **new** Person**(**"Rose"**);**

Person Jay **=** **new** Person**(**"Jay"**);**

Person Joe **=** **new** Person**(**"Joe"**);**

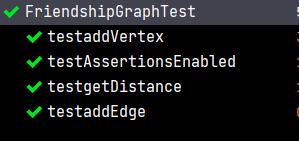
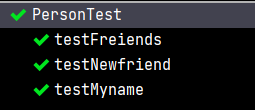
Rose**.**newfriend**(**Jay**);**

assertEquals**(**Jay**,**Rose**.**Freiends**().**get**(**Rose**.**Freiends**().**size**()-**1**));**

Rose**.**newfriend**(**Joe**);**

assertEquals**(**Joe**,**Rose**.**Freiends**().**get**(**Rose**.**Freiends**().**size**()-**1**));**

**}**

**结果：**

# 实验进度记录

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 任务 | 实际完成情况 |
| 2021-05-15 | 18:30-19:30 | 编写问题1的isLegalMagicSquare函数并进行测试 | 按计划完成 |
| 2021-5-16 | 18：30-19：30 | 编写turtle相关的方法 | 延期1小时完成 |
| 2021-5-18 | 16：00-17：00 | 编写问题2的convexHull | 遇到困难，未完成 |
| 2021-5-19 | 16：00-17：00 | 编写实验剩下的主体部分 | 按计划完成 |
| 2021-5-20 | 21：00-22：00 | 对各个方法编写测试用例 | 按计划完成 |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的困难 | 解决途径 |
| 在编写convexhell()时候找不到计算极角的计算公式 | 在演算纸上进行画图，利用相似三角形、余弦定理、反三角函数的知识得出相对简单的计算公式。 |
| 在编写getdistance()方法的时候没有想到将距离和个人关联的办法 | 在查阅相关的资料得知，在Java中有一种Map类可以关联两个种类，可以实现距离和个人的关联。 |
| 在convexhell（）的测试用例中始终无法得到正确的结果。 | 在分析测试用例的时候发现自己的代码扩大了凸包的集合，所以将凸包中的条件增多，以达到最小的凸包。 |

# 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

在这次的实验中，虽然遇到许多的问题，比如对于Java的面向对象编程的理解还是不够深刻、对于Java中存在的数据结构的运用还不够熟练。但是在进行这次实验的时候，通过查阅资料、参考相关的博客解决问题，这不仅在一定程度上填补在Java相关方面的知识的空白，而且提升了在Java中对象、类、方法、属性的概念和它们之间的关系。但是在编程过程中没有及时编写测试用例的习惯，所以在MagiSqares实验中，最后完成代码的编写之后发现出现了错误，所以浪费了许多时间在查找错误和重新编写代码。所以在以后的项目的编写时没每完一个方法就要及时编写测试用例。

## 针对以下方面的感受

1. Java编程语言是否对你的口味？

因为从接触编程开始一直是使用C语言进行编程，尤其是数据结构和CSAPP都是以C语言作为依托，Java是自学的语言，对于其面向对象编程的特点还是不太适应。

1. 关于Eclipse IDE；

Eclipse IDE是在界面上相对枯燥、没有中文插件、而且在编写代码的时候没有自动导入类的功能，所以使用的时IDEA。

1. 关于Git和GitHub；

Git和Github是外网的平台，全是英文界面，而且校园网本来就慢，在Github上面操作着实不太方便。

1. 关于CMU和MIT的作业；

学校和CMU和MIT对标，模仿和学习CMU和MIT的作业是可以理解的，希望学校可以对其资料有一些靠谱的翻译。

1. 关于本实验的工作量、难度、deadline；

实验的工作量还可以，但是报告要写的东西太多，完全没有必要，实验难度把控还可以，deadline也是大多数人可以接受的。

1. 关于初接触“软件构造”课程；

感觉还可以，对着门课程还是产生了极大的兴趣。