

**2021年春季学期**

**计算学部《软件构造》课程**

**Lab 1实验报告**

|  |  |
| --- | --- |
| 姓名 | 付希文 |
| 学号 | 1190201014 |
| 班号 | 1936601 |
| 电子邮件 | 964711432@qq.com |
| 手机号码 | 18553472308 |

**目录**

2 实验环境配置

3 实验过程

3.1 Magic Squares

3.1.1 isLegalMagicSquare()

3.1.2 generateMagicSquare()

3.2 Turtle Graphics

3.2.1 Problem 1: Clone and import

3.2.2 Problem 3: Turtle graphics and drawSquare

3.2.3 Problem 5: Drawing polygons

3.2.4 Problem 6: Calculating Bearings

3.2.5 Problem 7: Convex Hulls

3.2.6 Problem 8: Personal art

3.2.7 Submitting

3.3 Social Network

3.3.1 设计/实现FriendshipGraph类

3.3.2 设计/实现Person类

3.3.3 设计/实现客户端代码main()

3.3.4 设计/实现测试用例

4 实验进度记录

5 实验过程中遇到的困难与解决途径

6 实验过程中收获的经验、教训、感想

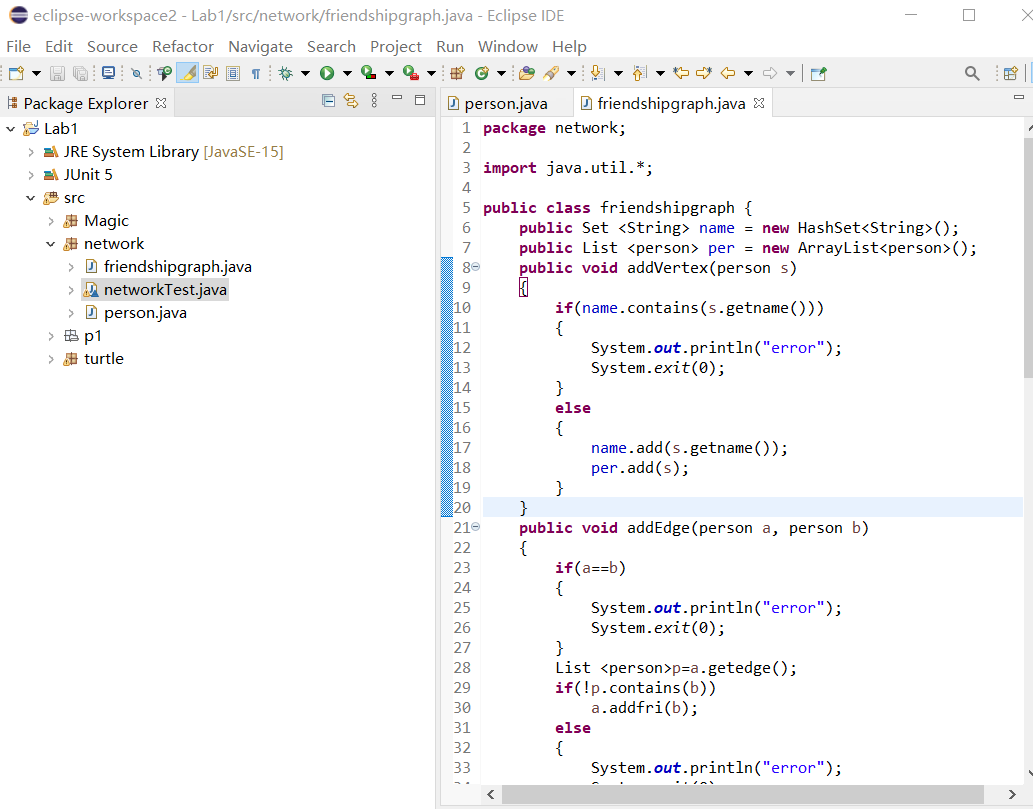
6.1 实验过程中收获的经验和教训

6.2 针对以下方面的感受

# **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**



地址 <https://github.com/ComputerScienceHIT/HIT-Lab1-1190201014>

# **实验过程**

请仔细对照实验手册，针对四个问题中的每一项任务，在下面各节中记录你的实验过程、阐述你的设计思路和问题求解思路，可辅之以示意图或关键源代码加以说明（但无需把你的源代码全部粘贴过来！）。

为了条理清晰，可根据需要在各节增加三级标题。

## **Magic Squares**

处理文件的输入，所有不合法输入退出并输出相关文字，合法输入判断每行每列以及对角线数值之和是否相等。

按目的生成一个MagicSquare。

### **isLegalMagicSquare()**

按行读取字符串，以\t分解成n个字符串，再用Integer.valueof转化为整数.通过字符串中是否含小数点判断输入是否为整数，通过字符串是否含有空格判断是否以\t隔开。

**if**(str.contains(".")|| str.contains(" "))

{**return** *f*=**false**;}

记录读入的行数和列数，判断是否相等

**if**(t!=p)

{**return** *f*=**false**;}

判断每个数是否为正数

**if**(a[t][i+1]<=0)

{**return** *f*=**false**;}

输入有误，输出输入有误，且函数返回值为false

合法输入最后判断每行每列及对角线是否相等，返回对应的值

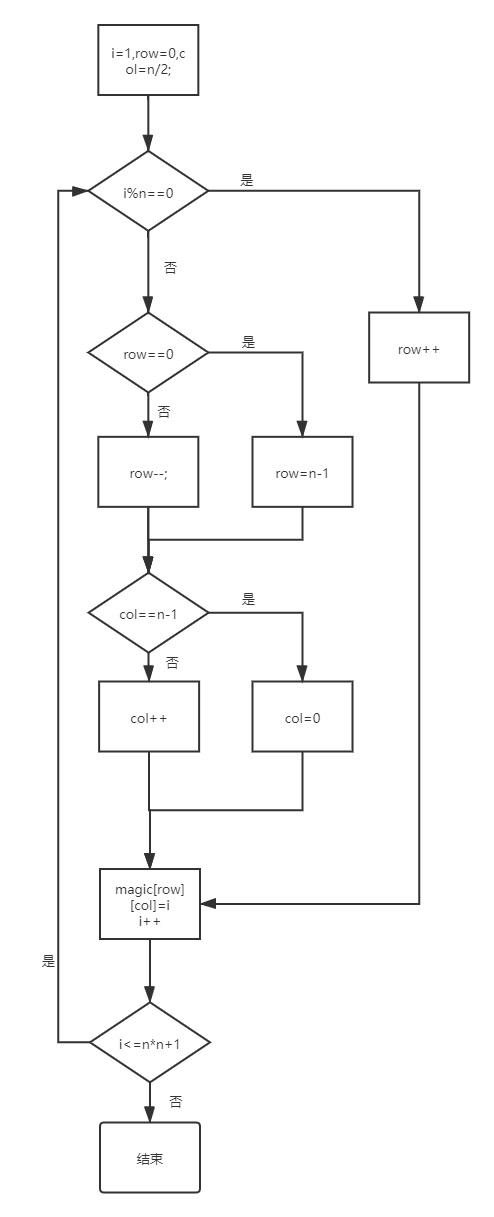
### **generateMagicSquare()**

具体实现：

row 从0开始减，对n取模，每n次加一

col 从n/2开始加一并对n取模

Magic[row][col]所得值从1到n\*n



问题：n为偶数的时候，row会等于n，数组越界

n为负数时，int[n][n]时越界

## **Turtle Graphics**

### 从Github上获取任务的源代码，在本地创建git仓库，使用git管理本地开发 2、画出正方形。 3、正多边形内角的计算 4、计算向量之间的夹角 5、求凸包 6、绘制自己的图形

### **Problem 1: Clone and import**

代码直接在网页上点击code下载zip文件

从本地创建git仓库：1、git init初始化本地仓库2、git add remote添加远程库源3、在远程仓库创建master分支4、git pull origin master将远程仓库同步到本地5、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**

Forward，向前走指定步长，turn，转指定角度

所以循环四次，每次先foward 再 turn（90）

### **Problem 5: Drawing polygons**

根据数学公式直接求多边形内角

**public** **static** **double** calculateRegularPolygonAngle(**int** sides) {

**double** ret=360.0/sides;

ret=180-ret;

**return** ret;

}

通过角度求边数和上边函数内容相反，只需要sides用Math.round()处理一下即可

**public** **static** **int** calculatePolygonSidesFromAngle(**double** angle) {

**double** tmp=180-angle;

**double** sides=360/tmp;

**int** ret=(**int**)Math.*round*(sides);

**return** ret;

}

计算出多边形内角后只需要先超当前方向走指定长度，再turn内角的度数，循环sides次，便可构造出正多边形

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

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

**for**(**int** i=1;i<=sides;i++)

{

turtle.forward(sideLength);

turtle.turn(180-angle);

}

}

### **Problem 6: Calculating Bearings**

求出从起点到重点的向量，用atan2函数可以得到其与x正半轴的夹角，再用90-此夹角-初始方向与y正半轴夹角即可，负数加上360

**public** **static** **double** calculateBearingToPoint(**double** currentBearing, **int** currentX, **int** currentY,**int** targetX, **int** targetY) {

**int** dy=targetY-currentY;

**int** dx=targetX-currentX;

**double** pi=Math.*acos*(-1);

**double** angle=Math.*atan2*(dy,dx)\*180/pi;

**double** ret=90.0-angle-currentBearing;

**if**(ret<0) ret+=360.0;

**return** ret;

}

调用上面补全的函数，直接求每两点须偏转的角度

**public** **static** List<Double> calculateBearings(List<Integer> xCoords, List<Integer> yCoords) {

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

**double** ang=0;

**int** x=xCoords.get(0);

**int** y=yCoords.get(0);

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

{

**int** xx=xCoords.get(i);

**int** yy=yCoords.get(i);

ang=*calculateBearingToPoint*(ang,x,y,xx,yy);

ret.add(ang);

x=xx; y=yy;

}

**return** ret;

}

### **Problem 7: Convex Hulls**

使用gift-wrapping算法，先找左下角的点，再循环多次，每次在所有点中找到需偏转角度最小的点，加入凸包直到再次找到最左下角的点为止，中途将所有选中的点存储，最后统一加入返回值的set中

**public** **static** Set<Point> convexHull(Set<Point> points) {

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

**double** xx=100000000.0; **double** yy=100000000.0;

LinkedHashSet<Point> ret= **new** LinkedHashSet<Point>();

**boolean**[] lis=**new** **boolean**[10000];

**int** temp=0; **int** as=0;

**for**(Point p : points)

{

temp++;

**double** x=p.x();

**double** y=p.y();

**if**(xx>x || (xx==x && yy>y))

{

xx=x; yy=y;

as=temp;

}

}

lis[as]=**true**;

Point a=**new** Point(xx, yy);

Point b=a;

**double** nowang=0.0;

**for**(**int** i=0;i<points.size();i++)

{

**double** angle=360.0;

**int** nowx=(**int**)a.x();

**int** nowy=(**int**)a.y();

**int** ansx=0; **int** ansy=0;

**int** tmp=0; temp=0;

**for**(Point p : points)

{

tmp++;

**int** x=(**int**)p.x();

**int** y=(**int**) p.y();

**if**(x==nowx && y==nowy) **continue**;

**double** ang=*calculateBearingToPoint*(nowang,nowx,nowy,x,y);

**if**(angle>ang)

{

angle=ang;

ansx=x;

ansy=y;

temp=tmp;

}

}

lis[temp]=**true**;

nowang+=angle;

a=**new** Point(ansx,ansy);

**int** x=(**int**)b.x();

**int** y=(**int**)b.y();

**if**(x==ansx && y==ansy) **break**;

}

temp=0;

**for**(Point p : points)

{

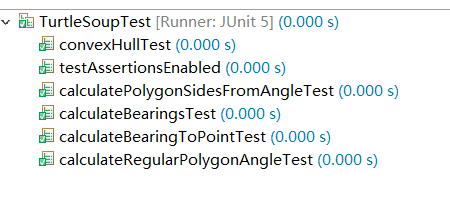
temp++;

**if**(lis[temp]==**true**) ret.add(p);

}

**return** ret;

}



### **Problem 8: Personal art**

**先朝一个方向画正四边形到100个边的正多边形**

**再转向另一个方向画一边**

**public** **static** **void** drawPersonalArt(Turtle turtle) {

*drawSquare*(turtle, 40);

**for**(**int** i=1;i<=100;i++)

*drawRegularPolygon*(turtle,i,i\*5);

turtle.turn(180);

**for**(**int** i=1;i<=100;i++)

*drawRegularPolygon*(turtle,i,i\*5);

turtle.draw();

}

### **Submitting**

## git add \*->git commit -m “”->git push提交到Lab1仓库

## **Social Network**

写一个类实现图的结构，并bfs求两点之间最短距离

### **设计/实现FriendshipGraph类**

用一个set记录是否名字重复出现

用List将每出现的人存储进去

重复的人出现报错，其余的直接加进去

**public** **void** addVertex(person s)

{

**if**(name.contains(s.getname()))

{

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

System.*exit*(0);

}

**else**

{

name.add(s.getname());

per.add(s);

}

}。

自环直接报错，重边不再加，剩下的用person里的addfri加边

**public** **void** addEdge(person a, person b)

{

**if**(a==b)

{

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

System.*exit*(0);

}

List <person>p=a.getedge();

**if**(!p.contains(b))

a.addfri(b);

**else**

{

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

System.*exit*(0);

}

}

写一个队列，用map存储每个person对应的距离，bfs每次找队首的邻接点，不在

Map里的入队，知道front==tail，由于边长度都为一，bfs先遍历到的就是最近距离

**public** **int** getDistance(person a,person b)

{

**if**(a==b) **return** 0;

Map<person,Integer> dis= **new** HashMap<>();

person[] q=**new** person[10000];

**int** front=0,tail=1; q[tail]=a;

dis.put(a, 0);

**while**(front<tail)

{

front++; person p=q[front];

List<person> edge=p.getedge();

**for**(person e : edge)

{

**if**(!dis.containsKey(e))

{

tail++; q[tail]=e;

**int** dist=dis.get(p);

dis.put(e, dist+1);

**if**(e==b) **return** dist+1;

}

}

}

**return** -1;

}

### **设计/实现Person类**

存一个名字，一个邻接点的表

**private** String name;

**private** List <person> edge;

初始化

**public** person(String name)

{

**this**.name=name;

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

}

返回名字

**public** String getname()

{

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

}

添加朋友关系

**public** **void** addfri(person b)

{

**this**.edge.add(b);

}

返回一个点的邻接表

**public** List <person> getedge()

{

**return** **this**.edge;

}

### **设计/实现客户端代码main()**

**按照实验给出的模板写即可。**

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);

### **设计/实现测试用例**

加入两个点看是否正确

**public** **void** testaddVertex()

{

friendshipgraph graph= **new** friendshipgraph();

person a= **new** person("stu");

person b= **new** person("stu2");

graph.addVertex(a);

*assertEquals*(a,graph.per.get(0));

graph.addVertex(b);

*assertEquals*(b,graph.per.get(1));

}

加入两条边测试其邻接表中的点

**public** **void** testaddEdge()

{

friendshipgraph graph= **new** friendshipgraph();

person a= **new** person("stu");

person b= **new** person("stu2");

graph.addVertex(a);

graph.addVertex(b);

graph.addEdge(a, b);

graph.addEdge(b, a);

*assertEquals*(b,a.getedge().get(0));

*assertEquals*(a,b.getedge().get(0));

}

具体构造一幅图，看手推的答案和程序跑的是否一样

**public** **void** testgetDistance()

{

friendshipgraph graph= **new** friendshipgraph();

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");

graph.addVertex(a);

graph.addVertex(b);

graph.addVertex(c);

graph.addVertex(d);

graph.addVertex(e);

graph.addVertex(f);

graph.addEdge(a, b);

graph.addEdge(b, a);

graph.addEdge(a, c);

graph.addEdge(c, d);

graph.addEdge(e, f);

graph.addEdge(f, e);

*assertEquals*(1,graph.getDistance(a, b));

*assertEquals*(1,graph.getDistance(b, a));

*assertEquals*(1,graph.getDistance(a, c));

*assertEquals*(-1,graph.getDistance(c, a));

*assertEquals*(2,graph.getDistance(a, d));

*assertEquals*(3,graph.getDistance(b, d));

*assertEquals*(1,graph.getDistance(e, f));

*assertEquals*(-1,graph.getDistance(a, e));

# }

# **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。

每次结束编程时，请向该表格中增加一行。不要事后胡乱填写。

不要嫌烦，该表格可帮助你汇总你在每个任务上付出的时间和精力，发现自己不擅长的任务，后续有意识的弥补。

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 任务 | 实际完成情况 |
| 2021-05-11 | 13.45-15.30 | 编写问题1的isLegalMagicSquare函数并进行测试 | 按计划完成 |
| 2021-05-18 | 13.45-15.30 | 完成问题1，2 | 按计划完成 |
| 2021-05-22 | 9.00-16.00 | 完成问题3及报告攥写 | 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 |
| Junit测试报错给的行和实际出错的示例不符，一开始没找到错误 | 点开具体信息，查看set里面需求的内容和程序输出内容发现小bug，及时更改。 |
| Java函数库不会使用 | 去csdn学习具体使用方法 |
|  |  |

# **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**

**加深了许多java库函数的使用知识，明白了github和git的使用方法。**

## **针对以下方面的感受**

1. Java编程语言是否对你的口味？

还好，上手有点困难，封装的函数太多，有很多不知道用法，但封装后的函数使用起来很方便。

1. 关于Eclipse IDE；

很好用的IDE，可以自动补全库，提示函数的使用方法，补全函数。

1. 关于Git和GitHub；

暂时没找到具体好用的地方

1. 关于CMU和MIT的作业；

实验质量很高，很有针对性，涉及算法方面的知识也比较通俗易懂，通过函数一步步实现，理解起来很方便。

1. 关于本实验的工作量、难度、deadline；

工作量和难度都适中，希望可以提供汉语版本的实验指导。

1. 关于初接触“软件构造”课程；