

**2018年春季学期  
计算机学院大二软件构造课程**

**Lab 1实验报告**

|  |  |
| --- | --- |
| 姓名 | 丁明泽 |
| 学号 | 1160300820 |
| 班号 | 1603008 |
| 电子邮件 | 570340617@qq.com |
| 手机号码 | 18846121293 |

**目录**

[1 实验目标概述 1](#_Toc506282558)

[2 实验环境配置 1](#_Toc506282559)

[3 实验过程 1](#_Toc506282560)

[3.1 Magic Squares 1](#_Toc506282561)

[3.1.1 isLegalMagicSquare() 1](#_Toc506282562)

[3.1.2 generateMagicSquare() 1](#_Toc506282563)

[3.2 Turtle Graphics 1](#_Toc506282564)

[3.2.1 Problem 1: Clone and import 2](#_Toc506282565)

[3.2.2 Problem 3: Turtle graphics and drawSquare 2](#_Toc506282566)

[3.2.3 Problem 5: Drawing polygons 2](#_Toc506282567)

[3.2.4 Problem 6: Calculating headings 2](#_Toc506282568)

[3.2.5 Problem 7: Personal art 2](#_Toc506282569)

[3.2.6 Submitting 2](#_Toc506282570)

[3.3 Social Network 2](#_Toc506282571)

[3.3.1 设计/实现FriendshipGraph类 2](#_Toc506282572)

[3.3.2 设计/实现Person类 2](#_Toc506282573)

[3.3.3 设计/实现客户端代码main() 2](#_Toc506282574)

[3.3.4 设计/实现测试用例 2](#_Toc506282575)

[3.4 Tweet Tweet（选作，额外记分） 3](#_Toc506282576)

[4 实验进度记录 3](#_Toc506282577)

[5 实验过程中遇到的困难与解决途径 3](#_Toc506282578)

[6 实验过程中收获的经验、教训、感想 3](#_Toc506282579)

# 实验目标概述

本次实验通过求解四个问题（其中一个可选），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 

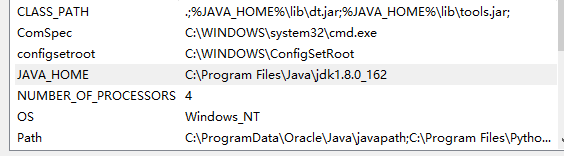
基于 Eclipse IDE 进行 Java 编程 

基于 JUnit 的测试 

基于 Git 的代码配置管理

# 实验环境配置

环境配置：现在eclipse的官网上面下载eclipse，然后安装，然后再从java上面下载JDK8，安装好，最后再写一下一下环境变量就OK。



在这里给出你的GitHub Lab1仓库的URL地址（Lab1-学号）。

[https](https://github.com/ComputerScienceHIT/Lab1-1160300820) : [//github.com/ComputerScienceHIT/Lab1-1160300820](https://github.com/ComputerScienceHIT/Lab1-1160300820)

# 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（但千万不要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

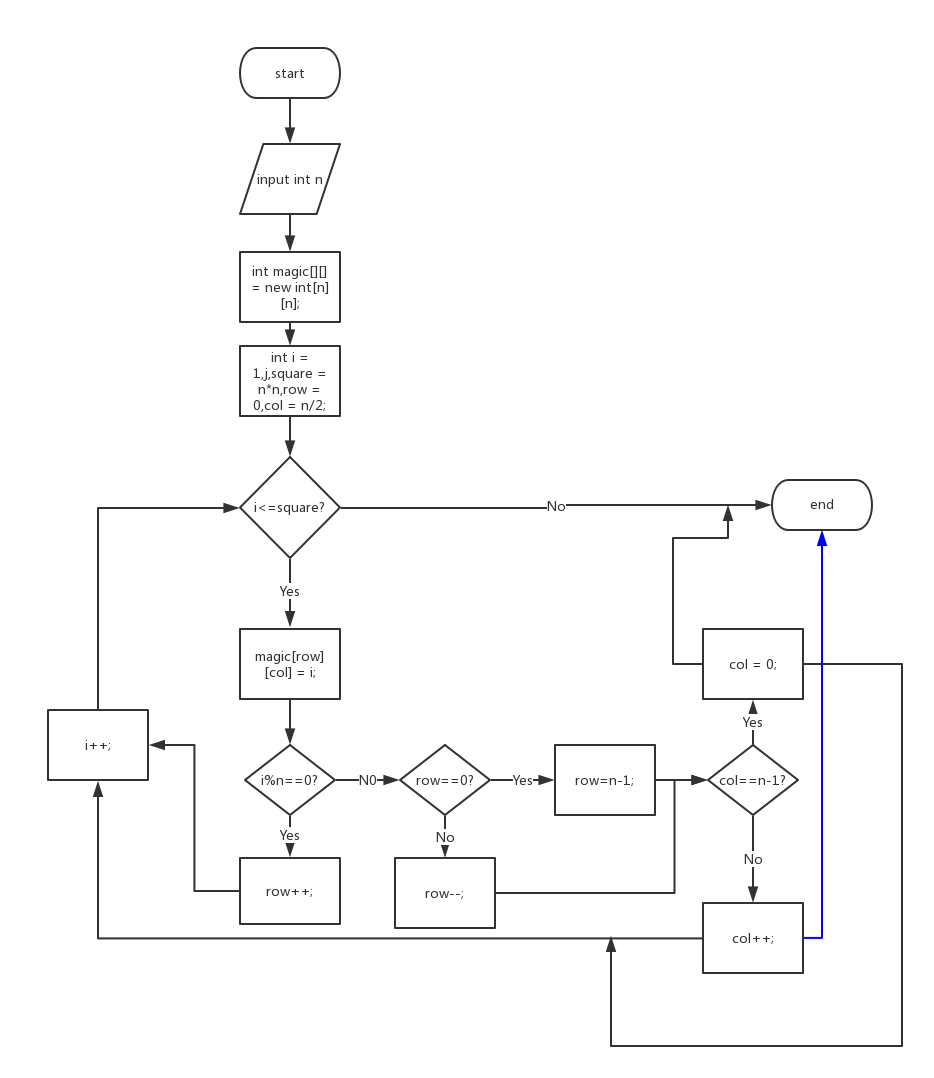
写出一个能判断输入的数阵是否是魔术矩阵，并看懂给出的生成魔术矩阵的函数。考察java中对list的运用和java的文件流。

### isLegalMagicSquare()

1. 思路：判断是否是一个魔术矩阵，根据定义即可，即只要检测出这个矩阵的每行每列和两个对角线的元素之和都相等即可。
2. 过程：先依次读入矩阵，将矩阵中的数都存入一个放int型数字（如果是小数和负数就返回错误）的list中然后先检测是不是一个方阵，然后再依次求出各行各列以及两条对角线元素之和是不是相等，如果都检测完且都正确之后则返回正确。
3. 结果：成功实现了所要求的功能。

### generateMagicSquare()

1. 流程图：



首先定义一个n行n列的二维数组来存储生成的矩阵，初始化行索引row=0，列索引col = n/2，令循环从i=1开始，每次加一，直到i大于n的平方，循环体中先判断i是不是能被n整除，如果是，则令row+1，如果不是则令继续判断row是不是为0，如果是则令row = n-1，如果不是则令row-1，接着判断col是不是等于n-1，如果是，则令col=0，如果不是则令col+1，循环结束后就生成了一个magic square。

1. 异常分析：

（1）

Exception in thread "main" java.lang.NegativeArraySizeException at MagicSquare.generateMagicSquare(MagicSquare.java:11)atMagicSquare.main(MagicSquare.java:121)

因为n为负数，这时候程序要创建一个大小为负数的数组，这时候就会抛出上述异常。

1. Exceptioninthread"main"java.lang.ArrayIndexOutOfBoundsException12atMagicSquare.generateMagicSquare(MagicSquare.java:17)atMagicSquare.main(MagicSquare.java:121)

假设一个数组的大小为n，则该数组的索引范围为0-（n-1），如果索引值不在这个范围内，则访问数组越界，就会抛出上面的异常。

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

**else**

{

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

**else** row--;

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

**else** col++;

}

}

分析上面的一段代码很容易知道，当i循环到等于n/2+1时，row = n，这时候再执行magic[row][col] = i,就会访问数组越界，就会抛出上面的异常信息。

## Turtle Graphics

考察对java中turtle的基本运用，考察边角与坐标之间简单的数学运算

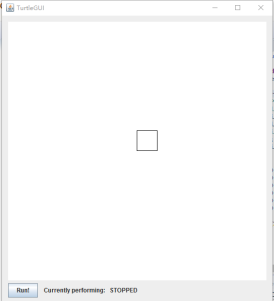
### Problem 1: Clone and import

$ git clone [git@github.com。。。从github上面克隆代码](mailto:git@github.com。。。从github上面克隆代码)

$ git init 创建本地仓库

### Problem 3: Turtle graphics and drawSquare

利用turtle中的forward和 turn函数画一个正方形，依次调用forward（）四次，每次给他的参数值都相同，调用三次turn（），每次转过90度，这样就可以画出一个正方形了。



### Problem 5: Drawing polygons

根据给出的边数sides，利用公式：180.0\*(sides-2)/sides算出内角度数，然后sides-1次turn，180-内角度数就是每次要turn的角度，然后sides次forward，每次forward的长度都是一样的即可；或者利用给出的角度angle，利用公式：

**if**(angle<60)

{

System.***out***.println("wrong!angle < 60");

**return** -1;

}

**else** **if**(360/(180-angle)%1>0.5)

**return** (**int**)(360/(180-angle))+1;

**else**

**return** (**int**)(360/(180-angle))

来算出边数，然后再turn和forward，如何turn和forward同上，这样就画出了正多边形。

### Problem 6: Calculating headings

利用两向量的点积除以这两个向量的模长之积等于两个向量夹角的余弦值来算出给出的两个点所连成的向量与y轴之间的夹角：

**int** x = targetX-currentX;

**int** y = targetY-currentY;

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

**double** angle = Math.*toDegrees*(Math.*acos*(cos));

然后再接上一些细节处理就得到了结果：

**return** (angle-currentHeading+360)%360;

### Problem 7: Personal art

PenColor k = **null**;

**int** i=0;

**while**(i<90)

{

turtle.color(k.***RED***);

turtle.forward(10);

turtle.turn(4);

i++;

}

turtle.color(k.***GREEN***);

turtle.turn(90);

turtle.forward(140);

**for**(i=0;i<9;i++)

{

turtle.color(k.***BLACK***);

turtle.turn(120);

turtle.forward(140);

turtle.color(k.***GREEN***);

turtle.turn(180);

turtle.forward(140);

i++;

}

turtle.color(k.***CYAN***);

turtle.turn(180);

turtle.forward(70);

turtle.turn(90);

i=0;

**while**(i<90)

{

turtle.color(k.***RED***);

turtle.forward(10);

turtle.turn(8);

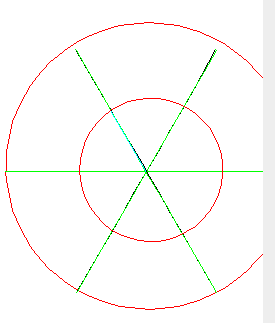
i++;

}

turtle.turn(90);

}

首先每次turn过很小的角度并forward一段非常小的距离，然后这样每次转过90度后就往回forward一大段距离并在中间用上面的方法再画一个小圆，再加上一些颜色的变化，最后就画成了一个黄绿色的车轮状的东西。



### Submitting

先把当前版本存到本地的git仓库里面，然后再import到自己学号关联的远程仓库上面即可

## Social Network

就是可以扩展到有向图的无向图的建立，插入顶点，加入边和求解最短路径等一些数据结构问题的考察。

### 设计/实现FriendshipGraph类

1. 首先定义图的节点类，利用邻接表来表示图：

**static** **class** node{

String p;

node next;

**public** node(String p,node next) {

**this**.p = p;

**this**.next = next;

}

}

2.定义方法addvertex来处理像图中加入新的顶点，该方法就是实例化一个node，但在这之前要检验要加入的顶点原来是否在图中：

**for**(node m:*fsp*)

{

**if**(m.p.equals(name.point))

{

System.***out***.println("the name is not unique");

**return** **false**;

}

}

1. 定义方法addedge来加入边，注意因为要求能像有向图扩展，所以每次，加边都要做两次，即加入a到b和b到a的边
2. 定义方法getdistance，因为图中每条边的权重都是1，所以可以用广度优先搜索的方法来得到两个定点之间的最短距离

### 设计/实现Person类

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

String point;

**public** Person(String name) {

point = name;

}

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

}

}

这部分只是简单地定义一个类来存储一个字符串来代表图中人物的名字。

### 设计/实现客户端代码main()

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));

先利用friendshipgraph类实例化一个图graph，然后接着一系列的加入顶点，加边，找两个顶点的最短距离。

### 设计/实现测试用例

FriendShipGraph graph = **new** FriendShipGraph();

Person paul = **new** Person("Paul");

Person james = **new** Person("James");

Person harden = **new** Person("Harden");

Person jordan = **new** Person("Jordan");

@Test

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

**assert**(FriendShipGraph.*addVertex*(paul));

**assert**(FriendShipGraph.*addVertex*(james));

**assert**(FriendShipGraph.*addVertex*(harden));

**assert**(FriendShipGraph.*addVertex*(jordan));

}

@Test

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

**assert**(FriendShipGraph.*addEdge*(james, harden));

**assert**(FriendShipGraph.*addEdge*(harden, james));

**assert**(FriendShipGraph.*addEdge*(paul, harden));

**assert**(FriendShipGraph.*addEdge*(harden, paul));

**assert**(FriendShipGraph.*addEdge*(james, paul));

**assert**(FriendShipGraph.*addEdge*(paul, james));

}

@Test

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

FriendShipGraph.*addVertex*(paul);

FriendShipGraph.*addVertex*(james);

FriendShipGraph.*addVertex*(harden);

FriendShipGraph.*addVertex*(jordan);

FriendShipGraph.*addEdge*(james, harden);

FriendShipGraph.*addEdge*(harden, james);

FriendShipGraph.*addEdge*(paul, harden);

FriendShipGraph.*addEdge*(harden, paul);

FriendShipGraph.*addEdge*(james, paul);

FriendShipGraph.*addEdge*(paul, james);

*assertEquals*(0, FriendShipGraph.*getDistance*(james, james));

*assertEquals*(1, FriendShipGraph.*getDistance*(james, harden));

*assertEquals*(1, FriendShipGraph.*getDistance*(james, paul));

*assertEquals*(-1, FriendShipGraph.*getDistance*(james, jordan));

}

}

这部分的设计思路类似于main函数中。

## Tweet Tweet（选作，额外记分）

3.3.1 Extract

1. Gettimespan：遍历所有的tweet，找到其中最早和最晚的发布时间，这样就得到了符合要求的timespan
2. Getmentionedusers:根据要求可得正则表达式：[^a-zA-Z0-9\_-]@[a-zA-Z0-9\_-]+

用该正则表达式去搜索匹配所有tweet的text，最后就能得到要求的名单

3.3.2 Filter

1.writtenby:遍历所有tweet，找到所有author是username的tweet即可

2.intimespan:遍历所有tweet，找到发布时间在timespan.start和timespan.end之间的所有tweet即可

3.containing：遍历所有tweet，找到text中含有words中有的单词的tweet即可，因为不区分大小写，可以先把words中的字符串和text中的字符串都转成小写再做匹配。

4.socialnetwork：

1.getsfellowmap：遍历所有tweet，用正则表达式[^a-zA-Z0-9\_-]@[a-zA-Z0-9\_-]+搜索匹配tweet里面的text，如果搜索匹配到了相应的字符串，如果name不是map的一个key，则把切割出来的名字name和该tweet的author加入到map中，并记为author fellows name；

如果name已经是map中的关键字了，则把author直接加入map.get(name)中。

1. influencers：按题意，用一个冒泡排序即可，为了方便处理，先把map的关键字集合变成数组：

String[] strings = **new** String[followsGraph.keySet().toArray().length];

strings = followsGraph.keySet().toArray(strings);

冒泡排序过程：

**for**(i=0;i<strings.length;i++)

{

biggest = followsGraph.get(strings[i]).size();

name = strings[i];

**for**(j=i+1;j<strings.length;j++)

{

**if**(biggest<followsGraph.get(strings[j]).size())

{

biggest = followsGraph.get(strings[j]).size();

name = strings[j];

strings[j] = strings[i];

strings[i] = name;

}

}

inf.add(name);

}

1. P4（additional rule）：在原有的基础上增加一条判断a是不是b的追随者的规则：如果a追随b，而且b追随c，那么a也算追随c；

这样只需在原来建好的graph的基础上，遍历graph，只要a的追随者有追随者b，而且b不在a的追随者中，那么则把b加入到a的追随者中：

**for**(String str1:gfg.keySet())

{

**if**(gfg.get(str1).size()!=0)

{

**for**(String str2:gfg.get(str1))

{

**if**(gfg.get(str2).size()!=0)

{

**for**(String str3:gfg.get(str2))

{

**if**(!gfg.get(str1).contains(str3))

gfg.get(str1).add(str3);

}

}

}

}

}

# 实验进度记录

请尽可能详细的记录你的进度情况。

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 计划任务 | 实际完成情况 |
| 2018-03-09 | 20:00-22:00 | 编写问题1的isLegalMagicSquare函数并进行测试 | 按计划完成 |
| 2018-03-10 | 8:00-11:00 | 编写P2所要求的各个函数并进行测试 | 按计划完成 |
| 2018-03-11 | 8:00-11:00 | 编写P3所要求的各个函数并进行测试 | 按计划完成 |
| 2018-03-12 | 15:30-17:30 | 学习正则表达式的相关内容并编写Extact的第一个方法和Filter的前两个方法 | 按计划完成 |
| 2018-03-12 | 20:00-23:00 | 编写完成Extract.java，Filter.java和Socialnetwork.java的部分内容并进行测试 | 按计划完成 |
| 2018-03-13 | 8:00-9:00 | 写完P4并进行测试 | 按计划完成 |
| 2018-03-14 | 20:00-22:30 | 完成实验报告 | 按计划完成 |

# 实验过程中遇到的困难与解决途径

在做到P4的时候，因为以前没有学过正则表达式，所以一下陷入了困境，最后通过上网查找博客教程和看书等学习了正则表达式的用法，这才完成了P4.

# 实验过程中收获的经验、教训、感想

通过本次实验，更好的掌握了java的一些基本操作，尤其是学会了以前没有使用过的正则表达式；遇到困难之后，一定要多翻书，多上网搜索资料。