哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

2024 年春季学期

计算学部《软件构造》课程

Lab 1 实验报告

姓名	罗煜婷
学号	2022111435
班号	2237102
电子邮件	2022111435@stu.hit.edu.cn
手机号码	18185715993

目录

1 实验目标概述

通过三个问题的求解，锻炼基本的 Java 编程能力，能够利用 JavaOO 开发基本的功能模块，并初步具备阅读已有代码和框架、理解并根据需求补全代码的能力。此外，练习使用常用的 Java 开发工具，学习并练习基于 JUnit 的测试方法，学习并练习使用 Git 工具进行代码管理和配置。

2 实验环境配置

简要陈述你配置本次实验所需开发、测试、运行环境的过程，必要时可以给出屏幕截图。

特别是要记录配置过程中遇到的问题和困难，以及如何解决的。

① 开发环境

开发环境：java 11.0.20

在官网下载 JDK11，由于原有环境为 JDK20，想要在 IDE 外的环境使用 JDK11，就需要调整 path 内两者的顺序，使 JDK11 在 JDK20 的上方，而且最好是放在路径的第一个，否则可能系统无法识别到 java 编译器。

IDE：IntelliJ IDEA 2023.1

一个可能的问题是，idea 可能不会自动表明 src、test、resource 等模块，可在 project structure 里设置

测试：JUnit

② Github Lab1：

https://github.com/ComputerScienceHIT/HIT-Lab1-2022111435

3 实验过程

请仔细对照实验手册，针对三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。

3.1 Magic Squares

任务分为两部分，采用不同思路进行解决。要求一须自行设计代码，意在锻炼基本的 Java 代码编写能力；要求二给出现有代码，只需要按照提示进行测试和运行，考验代码阅读分析能力。

3.1.1 `isLegalMagicSquare()`

3.1.1.1 思路

任务分为两部分实现，第一部分是文件输入和合法性判断，第二部分是题意实现，即幻方的判断。

3.1.1.2 具体实现

针对第一部分，采用 scanner 输入，spilt 进行划分（根据\t 划分，注意‘\’前要打转义符），再利用正则表达式进行字符合法性判断，此部分助教在课堂进行过演示，比较容易。而对是否为矩阵的判断，采取长度比较，设定首行长度为矩阵规模，同时记录已读入行数，每次读入的行存入一个一维数组，检验合法性后判断长度是否与首行相同，最后比较行数是否与列数相同，即可完成矩阵检测。

针对第二部分，为稍微提升效率，在读入过程中就对行列进行加和计算。初始设定两个小数组用于存储每行的和与每列的和，读入首行后对数组进行扩容。每行进行合法性检查的同时，计算行的加和，并将字符存入存储每列的和的数组中。读入完成后，对输入合法的矩阵，进行行之和数组和列之和数组的比较，采用 Arrays.equals 方法实现，完成比较。

3.1.1.3 结果

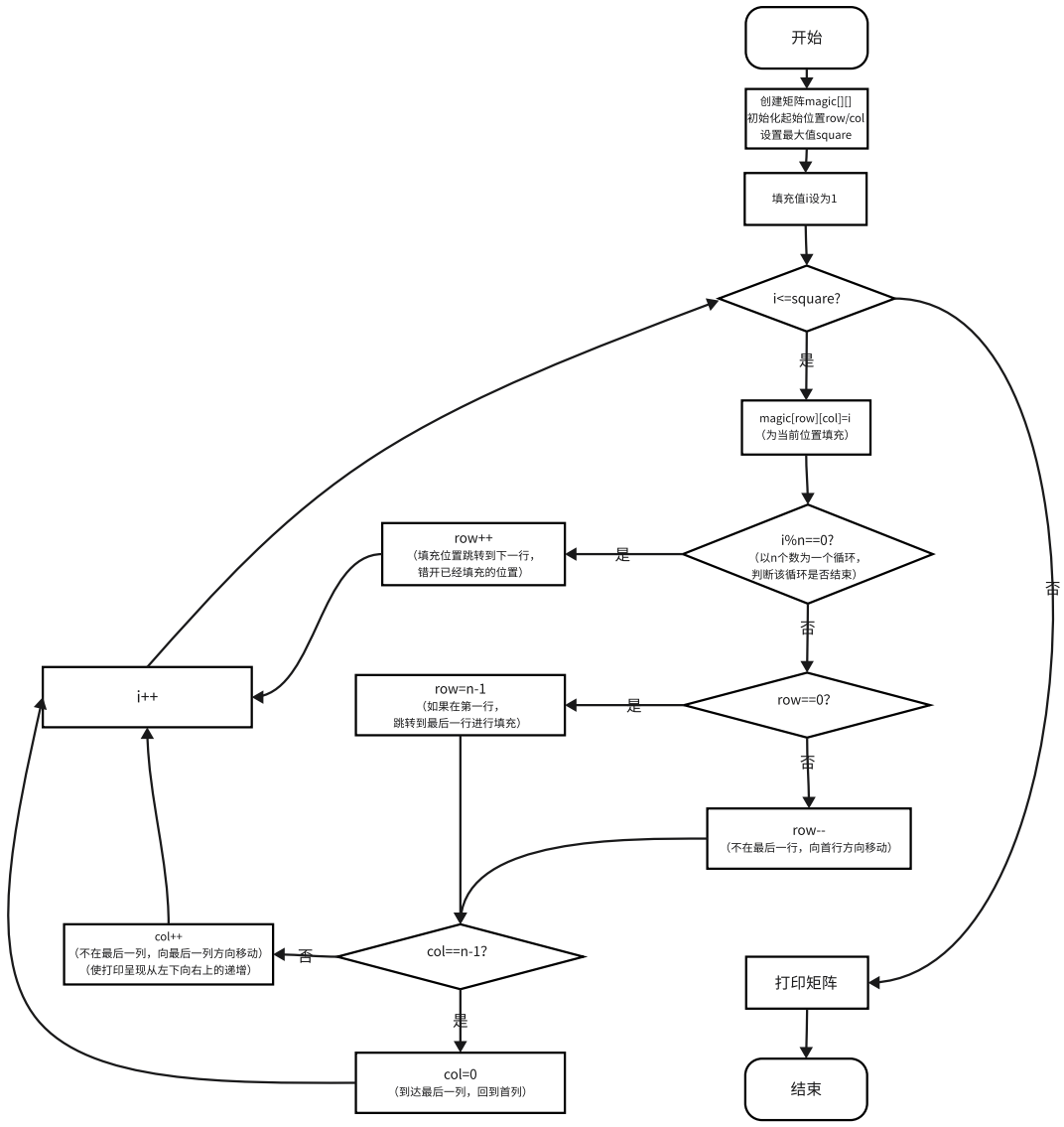
“1.txt”“2.txt”为 magic square，“3.txt”“4.txt”“5.txt”为非法输入。

```
The square of "src\P1\txt\1.txt" IS a Magic Square
The square of "src\P1\txt\2.txt" IS a Magic Square
java.io.IOException: The input is NOT a n*n matrix!
java.io.IOException: Illegal input!(character/point)
java.io.IOException: Illegal input!(character/point)
```

（图中对非法输入只判断一种异常即会退出）

3.1.2 generateMagicSquare()

3.1.2.1 函数流程图



程序核心代码如下：

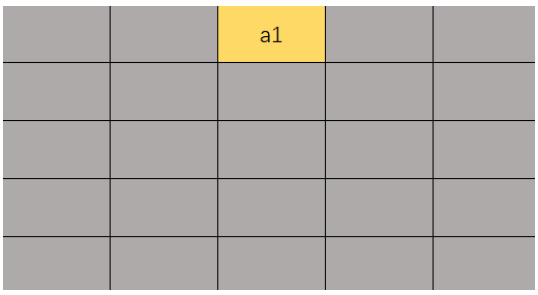
```
while (scanner.hasNext()) {
    String tempInput = scanner.nextLine();
    String[] input = tempInput.split( regex: "\\t");
    int sum=0;
    for(int j=0;j<input.length;j++){
        if(!input[j].matches( regex: "[0-9]+")) throw new IOException("Illegal input!(character/point)");
        sum+=Integer.parseInt(input[j]);
        if(j==colSum.length) colSum=Arrays.copyOf(colSum, newLength: j+1);
        colSum[j]+=Integer.parseInt(input[j]);//列的和加上对应的数
    }
    if(index==0  && input.length!=maxLine) {
        maxLine=input.length;//修改数组大小
        rowSum=Arrays.copyOf(rowSum,maxLine);
    }
    if(input.length!=maxLine) throw new IOException("The input is NOT a n*n matrix!");
    rowSum[index++]=sum;//一行计算完毕
}
if (index == 0) throw new IOException("Empty File");
if(maxLine!=index) throw new IOException("The input is NOT a n*n matrix!");
return Arrays.equals(colSum, rowSum);
```

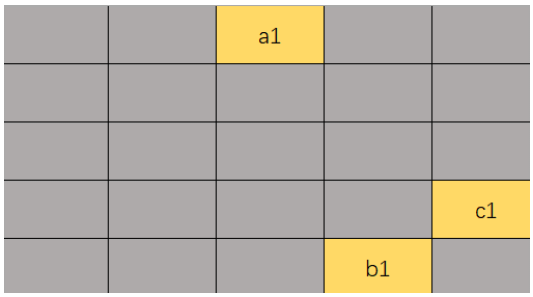
3.1.2.2 原理解释

该函数能根据奇数 n 生成 Magic Square，可以理解成一个 1~n 的循环放置，与 1~n 有所不同的只有每次到 n 以后都会增加一个大小 n 的基底。

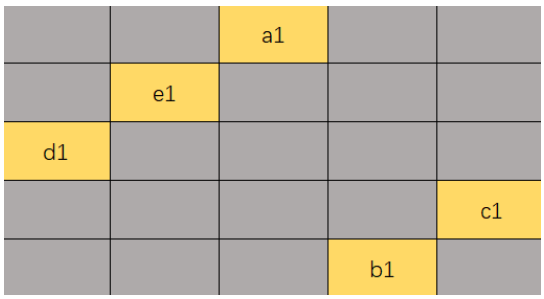
根据对函数的分析，以（cow，rol）为坐标的填充位置会向右上移动，每当超出边界时就回到对应下界，而每完成一个大小 n 的填充，就会向下方移动一行，以保证不重复填充。

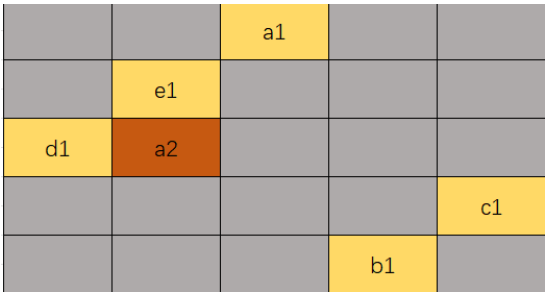
如下方左图，起始设置在 row=0，col=n/2 处，对于奇数来说，就是居中的位置，经过两次循环，达到下右图的效果。





经过 n 次填充，达到下左图的效果，此时，如果继续向右上前进，会覆盖已有的 a1 数字，因此向下移动，开始第二轮循环，即 a2（在实际填充中 e1 为 5，a2 为 6），如下右图。

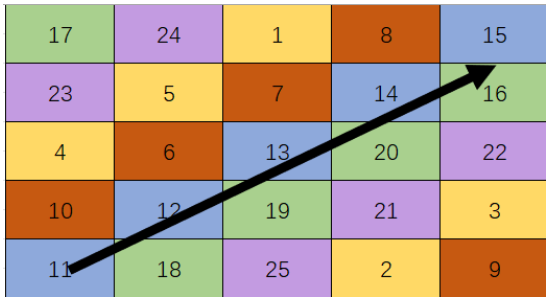




经过 n*n 次填充后，图形呈下左图，每个颜色是一个 n 次循环的结果，abcde 代表 1~5 的轮次，1~5 代表填充的先后顺序，同时，相同颜色代表同一个 n 重循环（如 d3 表示第 3 次循环中的第 d 个位置，即第四个，d3 处应该填充的数字是 2*5+4=14）。下右图的箭头表示基本的流向，从走向上来看，就像一个斜着打开的卷纸。

观察上方左侧图可以发现，每行每列都有 1~5 这五个数字的某个间隔为 1 的循环组合，以及 a~e 的间隔为 1 或间隔为 2 的循环组合，故 1~5 的循环排列必然导致每行每列的和相同。然后实际填充时，每一轮循环都会加上上一轮循环的最大数（最后一个数），我们称之为基底，这个基底每一轮循环就会增加 5，同时由于“卷纸”式的递增，每行每列都会在一轮循环中增加一个基底，故行和列的总和仍相同，相对静止。

b4	d5	a1	c2	e3
c5	e1	b2	d3	a4
d1	a2	c3	e4	b5
e2	b3	d4	a5	c1
a3	c4	e5	b1	d2



3.1.2.3 异常出现

根据实验手册进行测试，发现输入负数和偶数都出现了异常。

输入负数时，追溯到“main”方法中存在 NegativeArraySizeException 异常，意味数组负长度异常，通俗理解就是数组长度设置为负数，通过回溯追踪到出现在生成幻方的方法中。这很好理解，由于生成方法中需要先初始化一个 n*n 的矩阵 magic，而负数 n 传入后会导致初始化出错，产生对应异常。

而输入偶数时也会产生异常，根据提示是 ArrayIndexOutOfBoundsException，数组下标越界错误，同样在生成幻方的方法中出现。

出现这个错误的原因与奇数和偶数的特性有关，i%n==0 时，余数为零，表示一个“斜线”已经填满，如果再进行“行-1，列+1”的操作，就会回到重复的位置

而 row++表明每次斜线的起点向数字的前面移动的一位（如一开始以 3 开始，回到 2 以后，就会以 2 开始），但此方法忽略了 row++==n+1 的情况，后续没有进行相应处理。

在二维数组中，col 的位置是 col+1（如 n=6，则 col=3，从 magic[0][3]开始，第一行第四列），每个斜线循环结束的数字在起始数字的左下方，而经过 row++，下一个斜线的开始就是起始数字的下方第三行，左侧一列（如 n=6，起始为(1,4)，则下次开始为(3,3)(x 行 y 列)）。

若 row++前就在第 n 行（row=n-1），则下次起始的行刚好停在 n+1 行，所以可以得出结论，如果某一次起点会跳到 n+1 行，就会越界。

由于矩阵的特性，一共会经历 n-1 次 row++。对奇数来说，第一次起点到达边界是（n-1）/2 次跳跃后，会到达第 n 行，不越界；后续继续进行（n-1）/2 次

跳跃，在 n-1 行结束计算，不会越界；而对偶数来说，在（m-1）/2 次跳跃后，会到达 m-1 行（想象成奇数矩阵多一行就便于理解了），一圈赋值后到达 m 行，row++，越界。

下图为修改矩阵后的测试。输入奇数，正常运行并测试是否为幻方，输入偶数或负数，提示并跳出。

```
Enter a number
-1
Illegal input of n!
The square of "src\P1\txt\1.txt" IS a Magic Square
The square of "src\P1\txt\2.txt" IS a Magic Square
java.io.IOException: The input is NOT a n*n matrix!
java.io.IOException: Illegal input!(character/point)
java.io.IOException: Illegal input!(character/point)
```

```
Enter a number
6
Illegal input of n!
The square of "src\P1\txt\1.txt" IS a Magic Square
The square of "src\P1\txt\2.txt" IS a Magic Square
java.io.IOException: The input is NOT a n*n matrix!
java.io.IOException: Illegal input!(character/point)
java.io.IOException: Illegal input!(character/point)
```

3.1.2.4 其他思考

如果想要不越界，只要在 row++后判断是否越界即可。但是这样做，虽然可以输出一个矩阵了，却不是我们想要的魔法矩阵。

如果每次都想象成一个在对角线方向的 1~n 的循环，实际上在行和列上都应该是一个 1~n 间的某个排列的循环置换。对奇数和偶数行列数的矩阵，都是在列方向上形成 12...n 的循环置换。而在行上，由于新循环会从上一个循环结尾数 n 的下方开始，故 n 的右侧是（n+2）%n。

对奇数 n 来说，2 是 1~n 的集合在加法上的群的生成元，故一行可以覆盖 1~n 的所有数，如 n=5，则一行为 1 3 5 2 4（的置换）

而对偶数 n 来说，2 不是这样的一个生成元，会导致 n/2 个元素交替出现，如 n=6，则一行为 1 3 5 1 3 5 或 2 4 6 2 4 6 的（的置换）

故而采用这种方式来构造阶数为偶数的魔法矩阵是不可行的。

3.2 Turtle Graphics

任务基于一个 robot 运动模拟算法 turtle 展开，围绕绘图中需要的几个参数计算展开，综合考察 java 编程能力、问题分析和解决能力和数学基础。最后设置一道开放综合题，考察对代码的熟练程度，并培养审美素养。

3.2.1 Problem 1: Clone and import

想要获取任务代码，第一步是安装 git 工具，在官网上下载并安装（由于之前已经安装过，不再展示安装图像），初学时最好练习在命令行使用 git 指令，一方面是加强对 git 的认知，另一方面可以为 Linux 系统的使用打下基础，熟练后，也可以使用一些图形化管理工具，如 TortoiseGit；第二步，从实验手册提

```
[redacted]\git>git clone https://github.com/rainywang/Spring2022_HITCS_SC_Lab1.git test
Cloning into 'test'...
remote: Enumerating objects: 85, done.
Receiving objects: 100% (85/85), 570.53 KiB | 1.15 MiB/s, done.
Resolving deltas:   0% (0/24)
Resolving deltas: 100% (24/24), done.
```

供的仓库进行代码下载，本实验提供的代码在同一个仓库下，打开 GitHub 仓库，点击右侧“code”获取仓库 URL，在想要放置代码的文件夹打开命令行，使用 git clone +URL（+本地仓库名）的方式克隆即可。

```
[redacted].git\myLab1>git add src/P2/turtle/TurtleSoup.java

[redacted].git\myLab1>git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   src/P2/turtle/TurtleSoup.java

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        .idea/
        Lab-1 Report Template.docx
        "generateMagicSquare\346\265\201\347\250\213\345\233\276.svg"
        myLab1.iml
        src/P1/
        src/P2/.gitignore
        src/P2/rules/
        src/P2/turtle/Action.java
        src/P2/turtle/DrawableTurtle.java
        src/P2/turtle/LineSegment.java
        src/P2/turtle/PenColor.java
        src/P2/turtle/Point.java
        src/P2/turtle/Turtle.java
        src/P2/turtle/TurtleGUI.java
        src/P2/turtle/TurtleSoupTest.java
        src/P3/
        test/
        ~$b-1 Report Template.docx


[redacted]it\myLab1>git commit -m "完成了方法drawSquare的编写"
[master (root-commit) 97a0772] 完成了方法drawSquare的编写
 1 file changed, 151 insertions(+)
 create mode 100644 src/P2/turtle/TurtleSoup.java
```

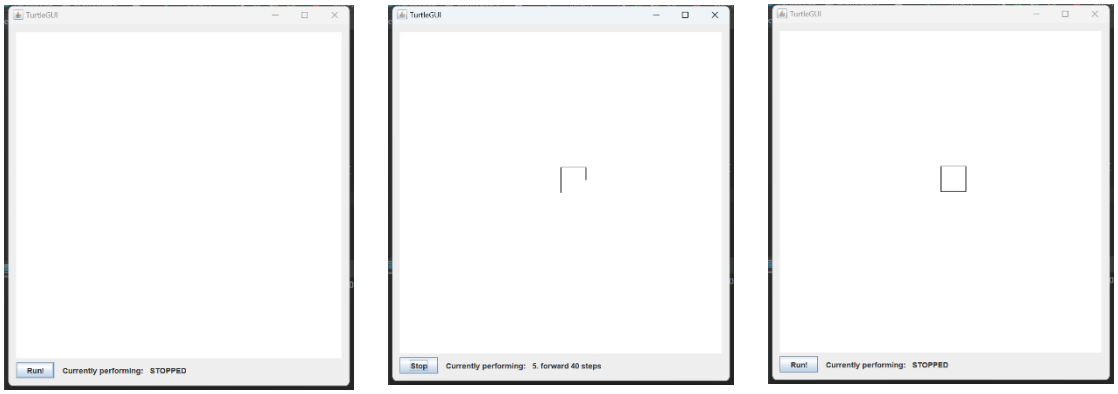
在本地利用 git 进行版本管理，如更新了一个文件，想要提交到本地仓库，需要先利用 git add +文件名。

再使用 git commit -m “”指令将文件加入本地仓库，同时在双引号处加入注释，这是一个好习惯。

3.2.2 Problem 3: Turtle graphics and `drawSquare`

利用已有的 forward（向前走若干步）方法和 turn（向右侧转若干度）来完成小乌龟画矩形的任务。

下图左侧，点击 Run 运行，过程某时刻如中间，画完后如下方右侧图




```
turtle.forward(sideLength);//先走第一条边
for(int i=0;i<3;i++)//转弯90度三次，画出一个正方形
{
    turtle.turn( degrees: 90);
    turtle.forward(sideLength);
}
```

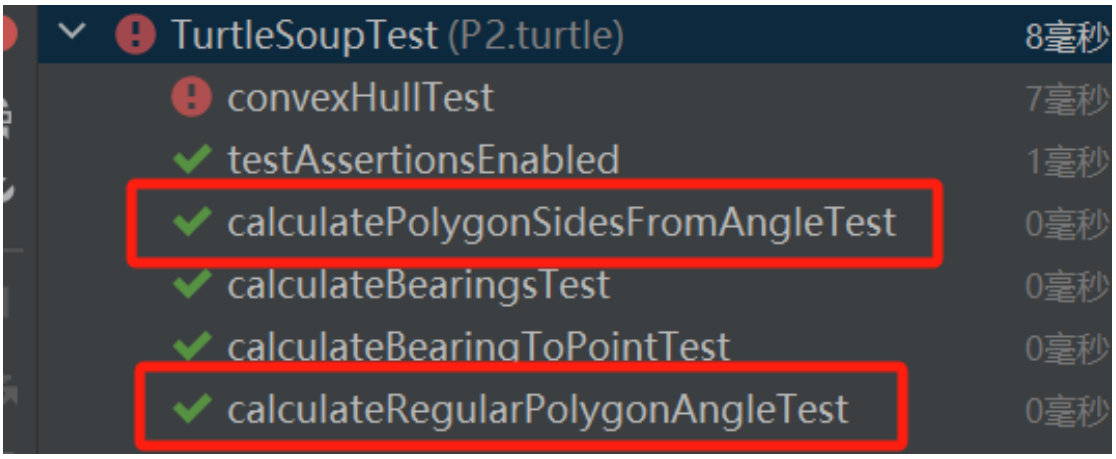
```
>git commit -m "完成了方法drawSquare的编写"
[master (root-commit) 97a0772] 完成了方法drawSquare的编写
 1 file changed, 151 insertions(+)
 create mode 100644 src/P2/turtle/TurtleSoup.java
```

3.2.3 Problem 5: Drawing polygons

计算正 n 边形每个内角度数的公式为：（n 边形的边数-2）*180°/n 边形的边数，计算度数需要注意的是，并非每个边数的角度都是整数，因此 angle 应该是一个 double 类型，且计算时要注意强制转换；同时，考虑到用户输入的可能性，对边数进行基本的合法性检查。

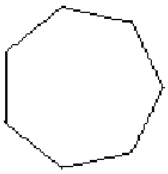
同样，将内角视为未知量，解计算边的方程，可以根据每个角的大小计算出正多边形的边数，则 n=360/（180-内角度数）。

测试结果如下所示



绘制多边形的代码和七边形效果如下：

```
public static void drawRegularPolygon(Turtle turtle, int sides, int sideLength) {
    double angle=calculateRegularPolygonAngle(sides);
    for(int i=0;i<sides;i++){
        turtle.forward(sideLength);
        turtle.turn( degrees: 180-angle);
    }
    //throw new RuntimeException("implement me!");
}
```



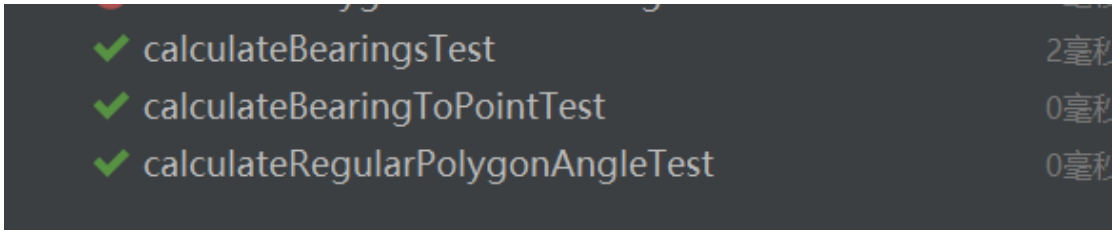
3.2.4 Problem 6: Calculating Bearings

第一题计算轴承，结合 turn 的定义，即从目标点到下一个点需要顺时针旋转的角度，需要注意，要结合本次出发时的方向进行下次旋转方向的计算。已知初始点和终止点，可以利用两者在 x 轴上和 y 轴上的距离差 instanceX 和 instanceY 进行计算，此处使用的公式是 90-（y/x 利用反正弦公式计算的度数）得出第一步，然后根据 x 的正负判断是否需要增加 180 度，再减去初始角度，如果减去以后变成负数，则加上 360 度，将逆时针旋转调整为顺时针旋转。

第二题为编写一组坐标间切换时需要旋转的角度，返回一个 double 类型的数组。编写过程中因为初始化数组时忘了加上“=new xxx（）”，还折腾了一会儿。

第一题代码如下方第一张图，测试结果如第二张图。

```
public static double calculateBearingToPoint(double currentBearing, int currentX, int currentY,
                                             int targetX, int targetY) {
    if(currentX==targetX&&currentY==targetY) return 0;
    int instanceX,instanceY;//距离
    instanceX=targetX-currentX;
    instanceY=targetY-currentY;
    double bearing=90-Math.toDegrees(Math.atan((double)instanceY/instanceX));//利用正切计算偏离y轴的角度
    if(instanceX<0) bearing+=180;//x与y异号，结合图形可知，需要再加180度
    bearing-=currentBearing;
    if(bearing<0) bearing+=360;//角度为负变成逆时针旋转，加360°调整为顺时针旋转
    return bearing;
    //throw new RuntimeException("implement me!");
}
```



3.2.5 Problem 7: Convex Hulls

实现这个问题的一种简单方法是网站推荐的礼物包装算法，这个算法我也看到若干版本，一种是暴力计算当前点到其他点的极角，时间复杂度为 O（n^2），另一种是使用栈，保留第一点到第二点构成的向量和第二点到栈顶点的向量之间的点积小于 0（即夹角小于 90°）的点，时间复杂度为 O（nlogn）。对于第二种方法我有些不理解，一是对于求得结果的完备性，二是对于“叉乘小于 0”，第二点显然只会留下使得形状外凸的点，而一些无法构成这样形状的点集则不会被识别为“凸”包。

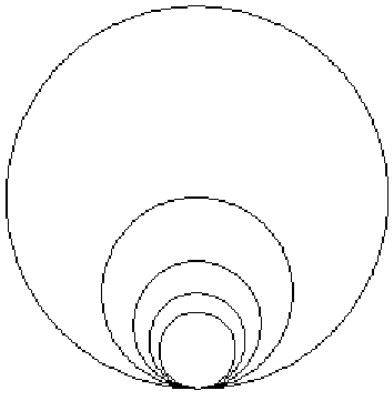
再次查阅资料可知，第二种做法名为 Graham 扫描法，这个做法每加入一个点则将该点与前两个点连线，直到最新点和当前点的向量相对次新点和最新点的向量为逆时针旋转的关系。实际上每个点都会被比较到，并且会经历入栈和出栈的过程。

理解了算法并进行实现，测试结果和代码如下。

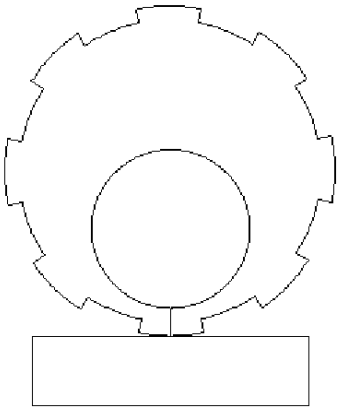
```java
public static Set<Point> convexHull(Set<Point> points) {
    if(points.size()<=3) return points;
    Point LTPoint=null;//找到最左上角的点的位置|这个点必然在集合中
    Set<Point> Convex=new HashSet<>();
    for(Point point:points){//找到最左上的点
        if(LTPoint==null) LTPoint=point;
        else{
            if(point.x()<LTPoint.x()) LTPoint=point;
            else if(point.x()== LTPoint.x()){
                if(point.y()>LTPoint.y()) LTPoint=point;
            }
        }
    }
    Point nowPoint=LTPoint;
    Convex.add(LTPoint);
    while(true){//回到起点或所有点都被加入就停止
        double minAngle=360;
        Point tempPoint=null;
        for(Point point:points){
            if(nowPoint==point||(Convex.contains(point)&&point!=LTPoint)) continue;//为了不破坏原有set，
            double angle=calculateBearingToPoint( currentBearing: 0,(int)nowPoint.x(),(int)nowPoint.y(),(
            //System.out.println(angle);
            if(angle<minAngle) {
                tempPoint = point;
                minAngle=angle;
            }
            else if(angle==minAngle){
                int nowIns=(int)((point.x()-nowPoint.x())*(point.x()-nowPoint.x())+(point.y()-nowPoin
                int minIns=(int)((tempPoint.x()-nowPoint.x())*(tempPoint.x()-nowPoint.x())+(tempPoint
                if(nowIns>minIns) tempPoint=point;
            }
        }
        nowPoint=tempPoint;//当前点成为最小角度点
```

3.2.6 Problem 8: Personal art

每次旋转 1°、前进 1 可画一个圆。想要圆更大可以增大前进距离或减小转向角度，反之，想要圆更小可以做相反的操作。



用画圆的方式试着画了校徽，但是还是因为没法写小数，在有限画布内只能画成这样。



3.2.7 Submitting

如何通过 Git 提交当前版本到 GitHub 上你的 Lab1 仓库。

第一步，git add P2，将文件夹加入暂存区。

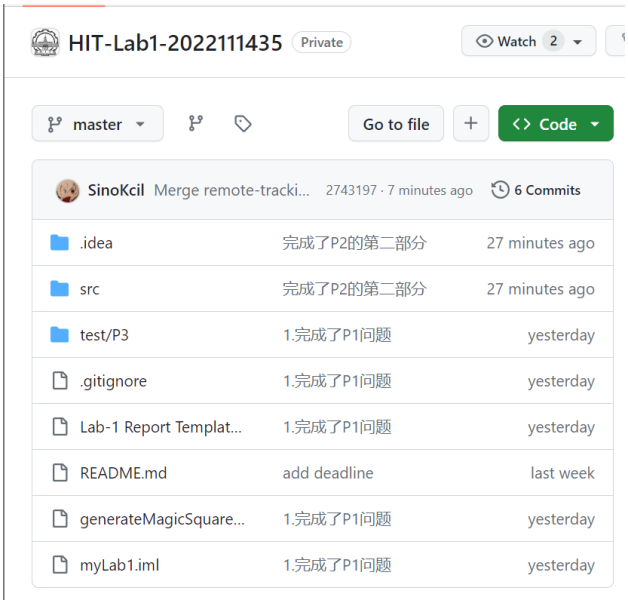
第二步，git commit -m “”，将暂存区提交至本地仓库。

第三步，git remote add origin https://github.com/ComputerScienceHIT/HIT-Lab1-2022111435.git，在本地添加远程仓库的地址，在本地创建远程仓库，名为 origin。

第四步，git pull 拉取更新本地仓库和远程仓库。

由于 push 报错，增加第五步，git fetch+git merge 远程仓库名/分支名，更新本地仓库。

第六步，git push -u 远程仓库名 分支名



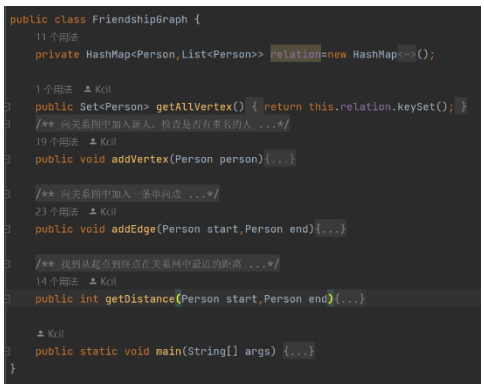
3.3 Social Network

任务背景为构建人际关系图并完成关系中的一些操作，考察了对 Java 类构建、方法编写、Map 使用等，同时考察了对图和搜索算法的掌握。

3.3.1 设计/实现 FriendshipGraph 类

给出你的设计和实现思路/过程/结果。

实现该类需要先完成 Person 类的编写或 Person 的接口编写，根据 spec 要求，需要完成 addEdge、addVertex 和 getDistance 的编写，并将测试代码粘贴到 main（）函数中。使用 HashMap 存储关系图，类型是 Person 的类型，关系使用 List 记录，在增加顶点和边的方法中都应该尽量保持代码的健壮性，对不合规的数据进行检测。在获取距离的方法中使用 BFS 算法，利用队列和存放是否访问的 Map 实现。为检测加入点的操作否顺利，在类中加入了一个 getAllVertex 类，利用 map 的 keySet（）方法返回 set。加入 main 函数后运行如下。



这个类设计中，设计了太多次关于引用和赋值的抉择，关于 hash map 设置为 Person 还是 String 就来来回回改了很多次，脑子里总是想到上课时老师说的引用可能会被更改（虽然我自己肯定不会做这种更改），最终决定尝试使用 Person 作为 map 类型。想要窥探 graph 的状态但是有不想让外界更改，于是去搜集了关于 clone（）方法的资料，发现这个方法是 protected 的，无法直接使用，必须进行修改。时间的限制让我放弃了克隆。

不过在这个过程中我发现了比较有趣的事情，关于 String 的机制。在 java 内部有一个字符串池存放所有的字符串常量，初次被创建的字符串会被加入这个池中，这个 String 引用会指向这个字符串。而在这之后，如果创建了一个 String 指向的是同一个字符串内容，则该新 String 会指向第一个 String 的地址，即这个“同样内容的字符串”的地址，然后根据这个地址去寻找字符串。但是这个新 String 引用与第一个 String 引用的地址是不一样的。

```
list.add(a);
if(a==b) System.out.println("yes!");
else System.out.println("no!");
if(list.contains(b)) System.out.println("Contain!");
else System.out.println("No Contain!");
System.out.println(a+" "+a.hashCode());
System.out.println(b+" "+b.hashCode());
System.out.println("abc".hashCode());
```

```
abc
abc
no!
Contain!
abc 96354
abc 96354
96354
```

上左图为代码设计，右图为输出，可见 a 与 b 的地址本身不同，经查阅得知，输入后返回的是一个对象（如果 a 与 b 都是常量可能可行，但此处不可行，也不建议使用“==”比较两个 string）；使用 list 提供的方法 contains 可行，此处我暂时不知道是因为指向的字符串地址相同还是因为内容相同；而后三个打印 hashcode 均相同，且连续几次测试同一个输入，前后的 hashcode 都相同。

但还有一处希望测试的地方，即如果同一程序中将一个字符串弃用较长时间又复用是否还指向同一位置，由于时间和性能问题无法完成，等待查阅资料后这个疑问不言自明。

主要代码如下。

```
23 个用法  Kcil
public void addEdge(Person start,Person end){
    if(!this.relation.containsKey(start)) throw new IllegalArgumentException("该起
    if(!this.relation.containsKey(end)) throw new IllegalArgumentException("该终点
    if(start==end) throw new IllegalArgumentException("起点和终点相同！");
    List<Person> list = relation.get(start);
    if(list.contains(end)) throw new IllegalArgumentException("已有该关系！");
    list.add(end);
}
```

```
public int getDistance(Person start,Person end){
    if(!this.relation.containsKey(start)||!this.relation.containsKey(end))
        throw new IllegalArgumentException("有一个人不在关系网中！");
    if(start==end) return 0;
    if(this.relation.get(start).isEmpty()||this.relation.get(end).isEmpty())
    Queue<Person> queue=new LinkedList<>();
    Map<Person,Integer> flag=new HashMap<>();
    flag.put(start,0);
    queue.add(start);
    while(!queue.isEmpty()){
        Person nowPerson= queue.remove();
        int dis=flag.get(nowPerson)+1;
        for(Person person : this.relation.get(nowPerson)){
            if(person==end) return dis;
            if(!flag.containsKey(person)){
                queue.add(person);
                flag.put(person, dis);
            }
        }
    }
    return -1;
```

3.3.2 设计/实现 Person 类

Person 类只有一个私有属性 name，表示名字，为了后续安全考虑设置为

final，同时构建方法中增加了对空白输入的检测。由于要求中提出要检测重复姓名，为了不修改 main 中的代码，在 Person 中增加了一个 static 的 list，以检测同一域里的重复姓名。类内只有初始化的构造方法和获取名字的 get 方法。

```
    2 个用法
    static List<String> allPerson=new ArrayList<>();//题目中提到不要使用stat
    2 个用法
    private final String name;

    17 个用法  Kcil
    public Person(String name) throws IllegalArgumentException {
        if(name.trim().length()==0) throw new IllegalArgumentException("
        if(allPerson.contains(name)) throw new IllegalArgumentException(
        this.name = name;
        allPerson.add(name);
    }
    Kcil
    public String getName() { return this.name; }
}
```

3.3.3 设计/实现客户端代码 main()

此处直接将实验手册内需要执行的代码复制，测试结果如 3.3.1 图 2。

注释掉 main 中第十行（rachel->rose）后，结果应为-1、-1、0、-1,测试结果如下第一张图；将 Ross 换成 Rachel，程序将会抛出异常，提示已有重名的人，测试结果如下第二张图。

```
        //graph.addEdge(rachel, ross);
        graph.addEdge(ross, rachel);
        graph.addEdge(ross, ben);
        graph.addEdge(ben, ross);
        System.out.println(graph.getDistance(rachel, ross));//should print 1
        System.out.println(graph.getDistance(rachel, ben));//should print 2
        System.out.println(graph.getDistance(rachel, rachel));//should print 0
        System.out.println(graph.getDistance(rachel, kramer));//should print -1
    }
```

```
FriendshipGraph
F:\develop\Java\jdk-11.0.20\bin\java.exe "-javaagent:D:\IntelliJ IDEA 2023.1\lib\idea_rt.jar
-1
-1
0
-1

进程已结束,退出代码0
```

3.3.4 设计/实现测试用例

给出你的设计和实现思路/过程/结果。

对增加点、边和找最短距离设计测试用例。

由于静态属性的设计，每个方法里创造的 Person 都必须不重名（变量名可重复），尝试将创建 Person 放置在方法外作为公共属性发现会提示创建了重复的人，在不同方法创建同名的人也会报错。这提示三件事，第一是测试顺序可能与方法顺序无关，第二是这些 static 属性对任意方法都相同，第三是公共区域的属性可能不止会执行一次。

测试增加点和边主要针对在该抛出异常的情况下是否会抛出异常，采用等价类划分的方式，增加边分为起点和终点相同、边已存在两类。寻找最短距离采用判断结果是否相等的方式，采用等价类划分的方式，分为可直达、需要中转才能到达、有不止一条路径、没有路径可到达、有向图的初步测试。

4 实验进度记录

日期	时间段	任务	实际完成情况
2024-3-13	10:00-11:30	完成问题 1 的要求 1	按计划完成
2024-3-13	19:30-22:00	完成问题 1 的要求 2，额外分析了偶数会导致生成方法出现错误的原因	按计划完成
2024-3-18	20:00-22:20	完成 P2 problem1~6	按计划完成
2024-3-19	10:20-13:20	完成 P2	按计划完成
2024-3-20	19:30-22:30	完成 P3 编写和部分文档	延时一小时完成
2024-3-21	16:00-17:30	根据手册对 P3 进行其他测试并完成所有文档	按计划完成
2024-3-21	19:00-21:00	尝试使用 javac 在命令行构建 项目	未完全完成

5 实验过程中遇到的困难与解决途径

遇到的困难	解决途径
试图修改幻方生成函数使其可以产生偶数阶幻方失败	分析了该方法无法生成的原因并撰写分析
求解凸包时坐标正确但测试错误	题目检测的是 Point 的地址，故需要把原点放入 hash set。为了避免改变原 set，需要增加一个对重复点的检查。
git push 提示落后于远程仓库	远程仓库原有一个 README.md，且本地仓库未同步过远程仓库。故需要先 fetch 分支进行合并
git merge 提示仓库无关无法合并	在 merge 命令后加入--allow-unrelated-histories，强制拉取
测试类在测试文件夹里，但无法运行测试	测试类是自己在琢磨 idea 的时候从头写的没有配好，查询后发现一定要在同名类上加上 public 才会被识别为单元测试类
想要测试 Person 的构造类以在出现重复名字的时候报错，但是失败	
试图将文件脱离 idea 环境，但是寻找资源文件时出现了问题	寻找助教，修改了资源地址为‘./’形式

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（必答）

课程中强调了可变类型和不可变类型，看似很简单，在实验中才感觉到易错性和设计的难度。

易错体现在对不可变物的判断，是区域的值不可变而引用指向的区域可变，还是引用指向的区域不可变而区域的值可变，对我来说比较容易弄混，还有 String 等比较特殊的类型，对地址等的比较就比较特别，还有 List 等类，内部引用是怎么设计的，这些都是我还不熟悉的方法。

而设计的难度就在于纠结使用什么类型构造类的内部，这可能会导致代码的冗余程度变化，类内部的可变性和不可变性也可能会影响外部使用和代码的冗余程度，而纠结的地方主要是传递的类型是否安全（假设有人不正确使用代码）和内容是否会改变，决定了是否需要克隆等。

最后一个反思点在于报告，写了太多，看起来很冗长，不太好。

6.2 针对以下方面的感受（必答）

(1) Java 编程语言是否对你的口味？与你熟悉的其他编程语言相比，Java 有何优势和不足？

Java 是一门优秀的面向对象语言，对我来说，它强大的静态检查和内存

管理、垃圾回收都很友好，在调试和检查阶段具有无与伦比的优势，跟 C 语言相比，简直就是一个温柔的保姆。但跟 C 语言相比，它的运行速度更慢，占用内存更大，且难以接触机器底层，这是相对的不足，也决定了它的应用场景与 C 语言的不同。

(2) 关于 Eclipse 或 IntelliJ IDEA，它们作为 IDE 的优势和不足；

我使用的是 IntelliJ IDEA，它的界面设计很优雅，交互性好，一键即跑，简单快捷，配置项目也很容易；但是想更深入地学习 IDEA 需要花费更多的精力，且占用内存更多，对于小项目有些“杀鸡用牛刀”的味道，大项目也可能卡顿，且脱离编译环境后难以自行构建项目，迁移比较困难。

(3) 关于 Git 和 GitHub，是否感受到了它在版本控制方面的价值；

从版本控制上来看，Git 仿佛给了程序员一次“悔过重来”的机会，也给了程序员更多试错的机会，是修改的保障。

(4) 关于 CMU 和 MIT 的作业，你有何感受；

作为名校的实验和作业，实验内容并不算难，但在代码编写和算法设计上都存在相应的难度阶梯，且指导详细，明确地将实验内容导向课程学习的目的，避免学生在无意义的地方耗费大量时间。

(5) 关于本实验的工作量、难度、deadline；

代码编写之外，撰写实验报告和打包花费了较长时间；编写代码难度较小，但是内容比较多；deadline 非常宽松。

(6) 关于初接触“软件构造”课程；

最初我以为本课程是讲述软件应用的基本构造方法、构造工具等，现在看来并非如此，正如老师上课常提到的“想想你的客户”，我们设计的东西的“客户”也是程序员，而不是普通用户。虽然本课程不是一门专门的编程课，但是整体内容就是在引入面向对象的编程思想和编程方法，是教会我们如何设计更好的更安全的代码、如何对代码的使用进行规范（如 specification）。课程中以 Java 为例，我想实际上其他的面向对象语言也是相似的，不过是具体实现方法和运用细节有所差异。学习这种思想，可以说相当于掌握了很多种方法。