



哈爾濱工業大學  
HARBIN INSTITUTE OF TECHNOLOGY

# 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
2 实验环境配置 .....	1
2.1 配置本次实验所需开发、测试、运行环境的过程 .....	1
2.2 GitHub Lab1 仓库的 URL 地址 .....	2
3 实验过程 .....	2
3.1 Magic Squares .....	2
3.1.1 isLegalMagicSquare() .....	3
3.1.2 generateMagicSquare() .....	4
3.2 Turtle Graphics .....	5
3.2.1 Problem 1: Clone and import .....	5
3.2.2 Problem 3: Turtle graphics and drawSquare .....	5
3.2.3 Problem 5: Drawing polygons .....	6
3.2.4 Problem 6: Calculating Bearings .....	7
3.2.5 Problem 7: Convex Hulls .....	8
3.2.6 Problem 8: Personal art .....	8
3.2.7 Submitting .....	10
3.3 Social Network .....	12
3.3.1 设计/实现 FriendshipGraph 类 .....	12
3.3.2 设计/实现 Person 类 .....	15
3.3.3 设计/实现客户端代码 main() .....	15
3.3.4 设计/实现测试用例 .....	15
4 实验进度记录 .....	17
5 实验过程中遇到的困难与解决途径 .....	18
6 实验过程中收获的经验、教训、感想 .....	18
6.1 实验过程中收获的经验教训 .....	18
6.2 针对以下方面的感受 .....	19

# 1 实验目标概述

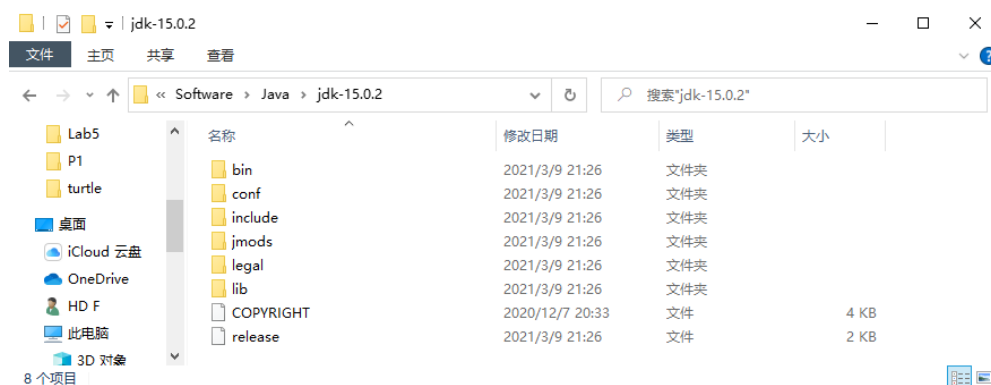
本次实验通过求解三个问题,训练基本 Java 编程技能,能够利用 Java OO 开发基本的功能模块,能够阅读理解已有代码框架并根据功能需求补全代码,能够为所开发的代码编写基本的测试程序并完成测试,初步保证所开发代码的正确性。另一方面,利用 Git 作为代码配置管理的工具,学会 Git 的基本使用方法。

- 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

## 2 实验环境配置

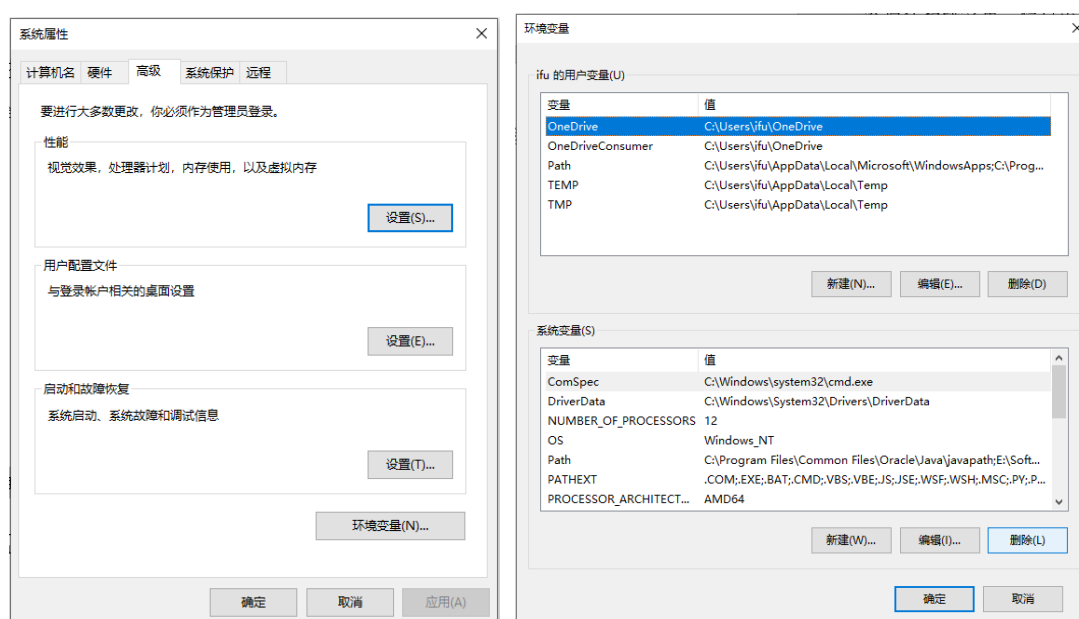
### 2.1 配置本次实验所需开发、测试、运行环境的过程

#### 1. 下载安装 JDK, 版本 15.0.2




#### 2. 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



### 3. 安装 Eclipse

 eclipse-inst-jre-win64.exe

2021/5/6 20:20

应用程序

### 4. 下载安装 git、配置 git 环境

## 2.2 GitHub Lab1 仓库的 URL 地址

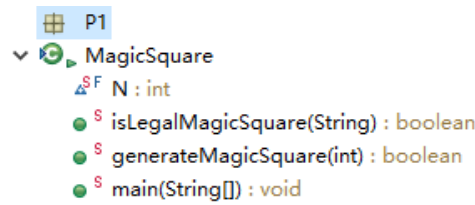
<https://github.com/ComputerScienceHIT/HIT-Lab1-1190202105>

## 3 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。

### 3.1 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  $n \times n$  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。



### 3.1.1 isLegalMagicSquare()

#### 设计思路

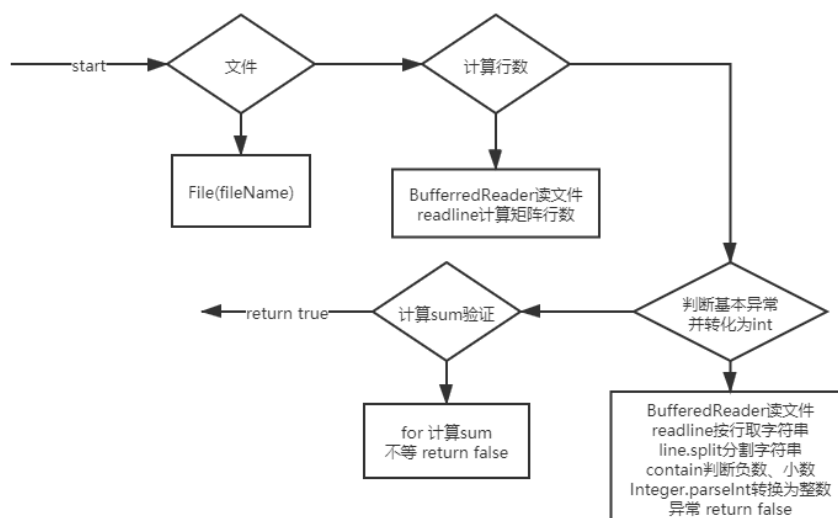
##### 1. 读入文件

首先是创建 `FileReader`、`BufferedReader` 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`。

##### 2. 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



public static boolean isLegalMagicSquare (String fileName) 流程图

运行结果如下：

```

1.MagicSquare!
true
2.MagicSquare!
true
3.NOT MATRIX!
false
4.ILLEGAL NUMBER!
false
5.NOT MATRIX!
false

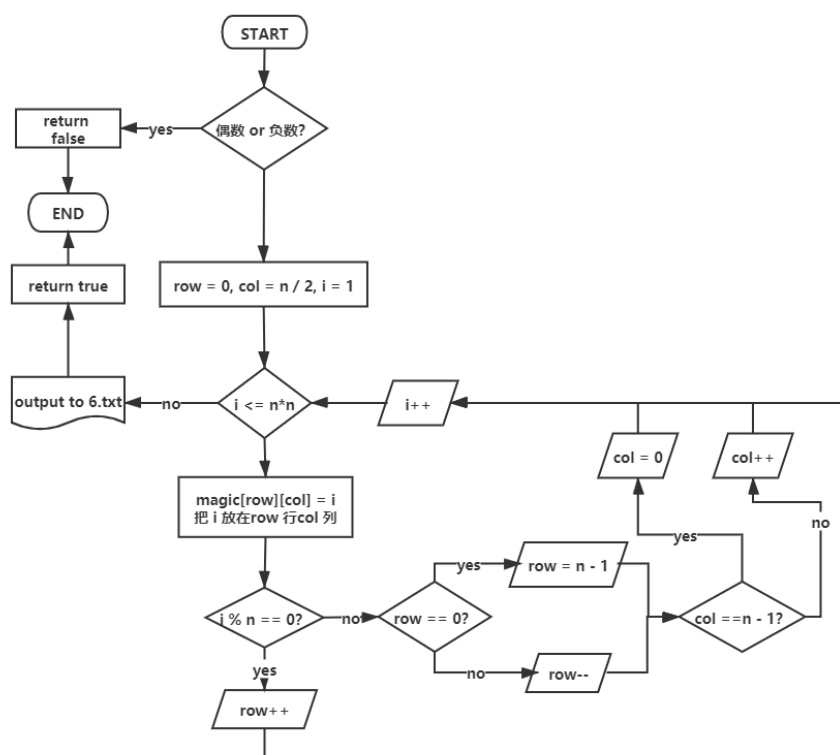
```

### 3.1.2 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  $n \times n$  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  $n^2$  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

```

Input a Number: 8 Input a Number: -4
6.Wrong Input      6.Wrong Input

```

```

Input a Number: 7
6.MagicSquare!
true

```

30	39	48	1	10	19	28
38	47	7	9	18	27	29
46	6	8	17	26	35	37
5	14	16	25	34	36	45
13	15	24	33	42	44	4
21	23	32	41	43	3	12
22	31	40	49	2	11	20

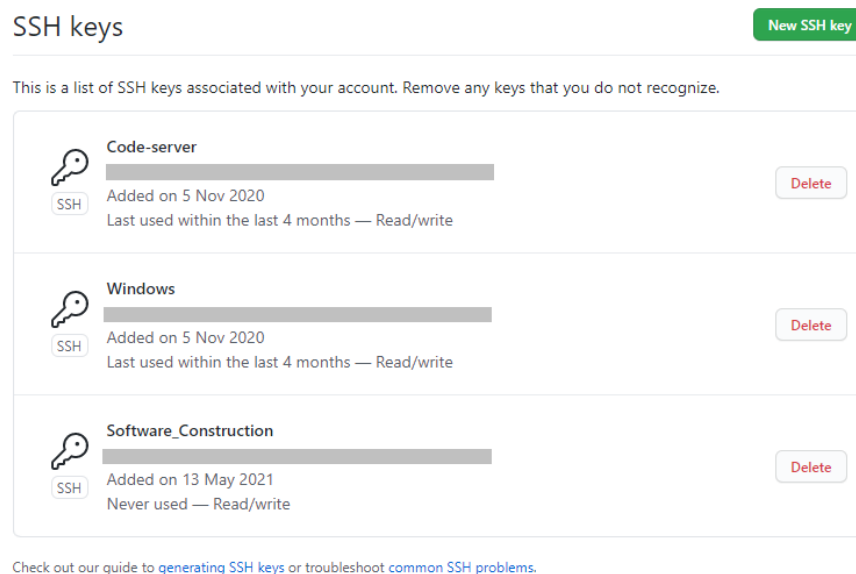
## 3.2 Turtle Graphics

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

### 3.2.1 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

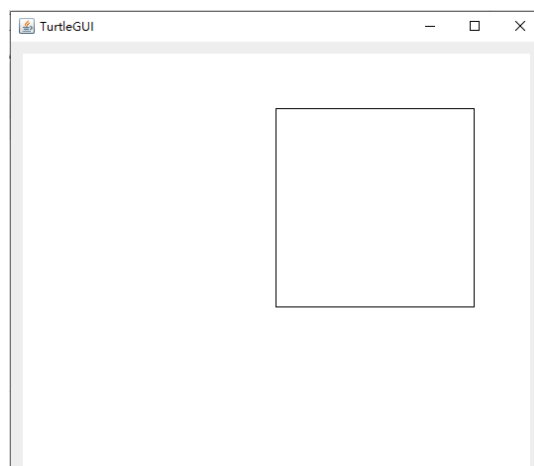


### 3.2.2 Problem 3: Turtle graphics and drawSquare

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。

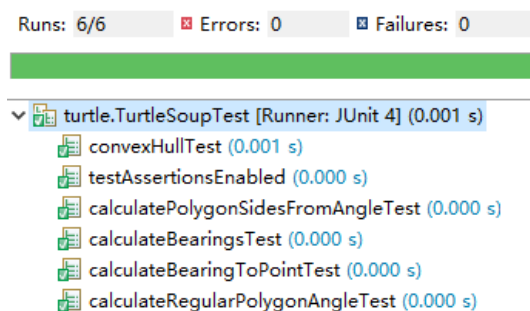
要求: 画出边长为指定值的正方形, 参数海龟对象 `turtle` 和编程 `sidelength`。

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。

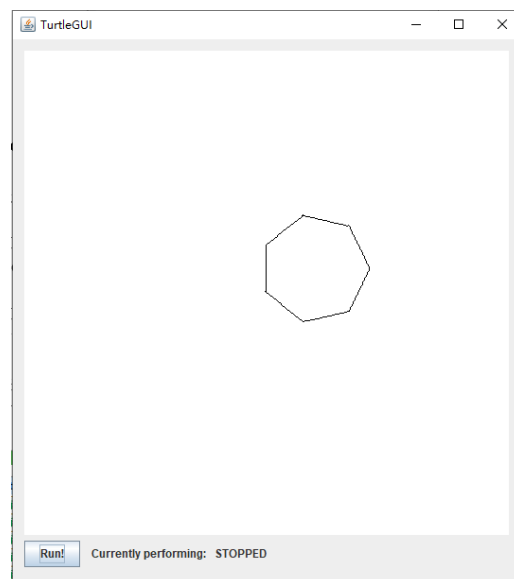


### 3.2.3 Problem 5: Drawing polygons

1. 首先 Implement 函数 `calculateRegularPolygonAngle`  
$$\text{return } (sides - 2) * 180.0 / sides;$$
  
通过边数来计算正多边形的每个内角大小;
2. Implement 函数 `calculatePolygonSidesFromAngle`  
$$\text{return } (int) \text{Math.round}(360.0 / (180.0 - angle));$$
  
通过角度大小来计算正多边形变数;
3. 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  $180.0 - \text{内角大小}$ , 每次前进 `sideLength` 长度, 循环 `sides` 次即可。
4. 运行 `TurtleSoupTest.java` 中的 Junit 测试结果, 以及 `sideLength` 为 50 的正七边形的绘制。







### 3.2.4 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。

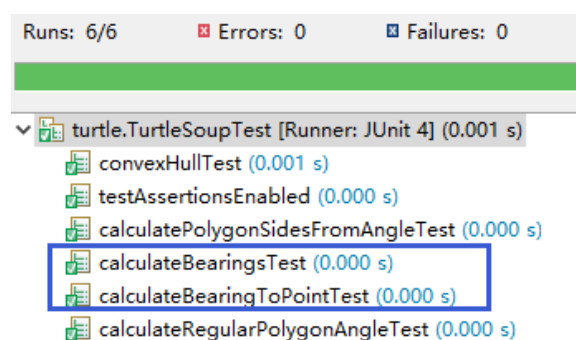
```
double radian = Math.atan2(.....);  
if (radian < 0) radian += 2 * Math.PI;  
double angle = radian / Math.PI * 180.0;
```

Turtle 的  $0^\circ$  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。

2. 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>`。
3. 运行 `TurtleSoupTest.java` 中的 Junit 测试结果。



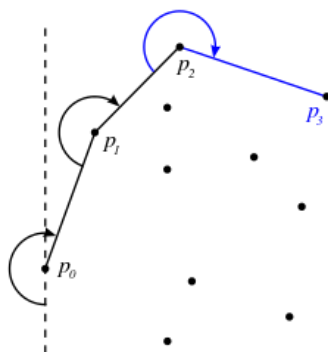
### 3.2.5 Problem 7: Convex Hulls

#### 1. 首先对 Gift Wrapping Algorithm 进行分析。

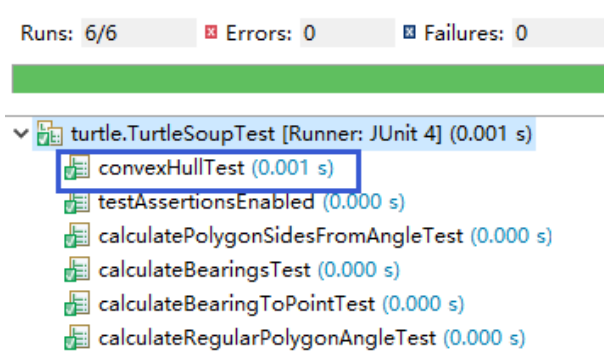
首先确定所有点中最右下角的点为原点，去点集合里面寻找另一个点，使得上一个点到该点旋转角度最小，使其包含进 `Set<Point> convexHull`，若算得多个点角度一样，则取最最远的那个点。反复进行上述操作，直到最后一个点旋转最小的点是原点，所得到的集合 `Set` 就是所求的 Convex Hull。

#### 2. 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”。



#### 3. Junit 测试结果



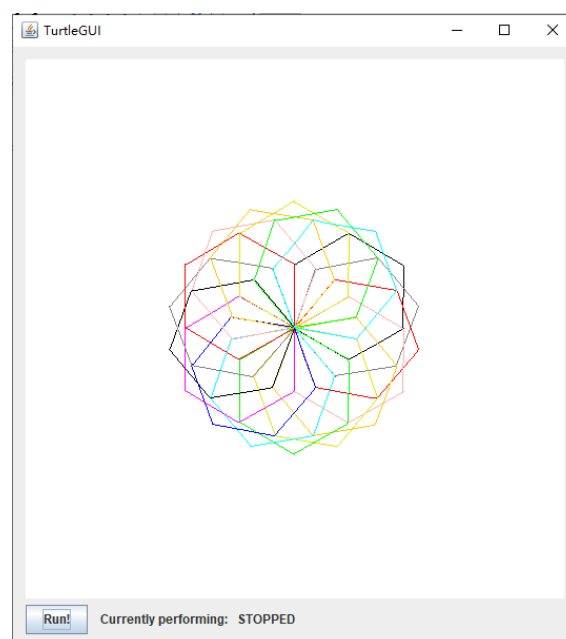
### 3.2.6 Problem 8: Personal art

#### 3.2.6.1 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。

```
public static void drawPersonalArt(Turtle turtle) {  
    int step = 60, count = 18;  
    for (int i = 0; i < count; i++) {  
        switch (i % 10) {  
            case 0:  
                turtle.color(PenColor.BLACK);  
                break;  
            case 1:  
                turtle.color(PenColor.GRAY);  
                break;  
            case 2:  
                turtle.color(PenColor.RED);  
                break;  
            case 3:  
                turtle.color(PenColor.PINK);  
                break;  
            case 4:  
                turtle.color(PenColor.ORANGE);  
                break;  
            case 5:  
                turtle.color(PenColor.YELLOW);  
                break;  
            case 6:  
                turtle.color(PenColor.GREEN);  
                break;  
            case 7:  
                turtle.color(PenColor.CYAN);  
                break;  
            case 8:  
                turtle.color(PenColor.BLUE);  
                break;  
            case 9:  
                turtle.color(PenColor.MAGENTA);  
                break;  
        }  
        drawRegularPolygon(turtle, 360 / step, step);  
        turtle.turn(360 / count);  
    }  
}
```

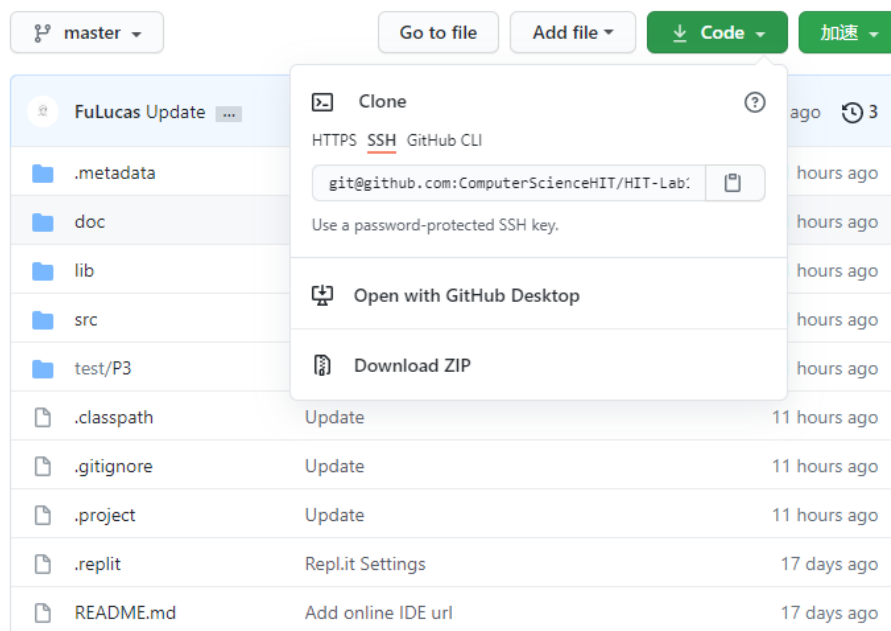
### 3.2.6.2 作画结果



### 3.2.7 Submitting

通过 Git 提交当前版本到 GitHub 上你的 Lab1 仓库。

#### 3.2.7.1 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

#### 3.2.7.2 修改 main 分支为 master 分支

```
ifu@DESKTOP-F9RM2JS MINGW64 /e/Learn/Software_Construction/Lab/HIT-Lab1-11902021
05 (main)
$ git branch -m main master

ifu@DESKTOP-F9RM2JS MINGW64 /e/Learn/Software_Construction/Lab/HIT-Lab1-11902021
05 (master)
$ git fetch origin
From https://github.com/ComputerScienceHIT/HIT-Lab1-1190202105
* [new branch]      master    -> origin/master

ifu@DESKTOP-F9RM2JS MINGW64 /e/Learn/Software_Construction/Lab/HIT-Lab1-11902021
05 (master)
$ git branch -u origin/master master
Branch 'master' set up to track remote branch 'master' from 'origin'.

ifu@DESKTOP-F9RM2JS MINGW64 /e/Learn/Software_Construction/Lab/HIT-Lab1-11902021
05 (master)
$ git remote set-head origin -a
origin/HEAD set to master
```

### 3.2.7.3 添加上传文件

```
ifu@DESKTOP-F9RM2JS MINGW64 /e/Learn/Software_Construction/Lab/HIT-Lab1-1190202105 (master)
$ git add .
warning: LF will be replaced by CRLF in .gitignore.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in .metadata/.log.
```

Git add . 前后都有空格的点，添加所有文件

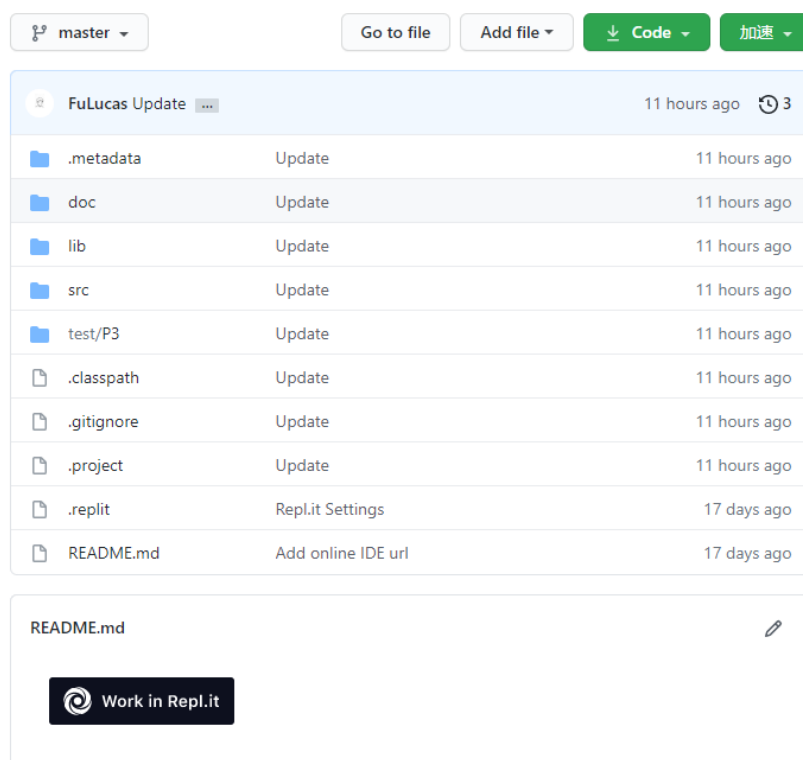
### 3.2.7.4 修改日志

```
ifu@DESKTOP-F9RM2JS MINGW64 /e/Learn/Software_Construction/Lab/HIT-Lab1-1190202105 (master)
$ git commit -m "Update"
[master 4c04860] Update
63 files changed, 3975 insertions(+)
```

Git commit -m “update” 修改文件新日志

### 3.2.7.5 上传

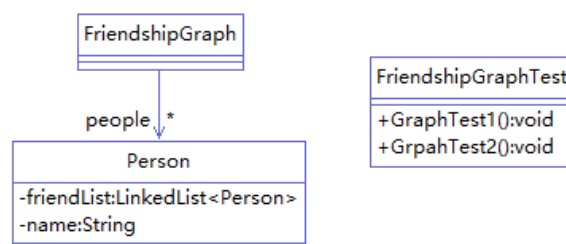
```
ifu@DESKTOP-F9RM2JS MINGW64 /e/Learn/Software_Construction/Lab/HIT-Lab1-1190202105 (master)
$ git push -u origin master
Enumerating objects: 93, done.
Counting objects: 100% (93/93), done.
Delta compression using up to 12 threads
Compressing objects: 100% (76/76), done.
Writing objects: 100% (92/92), 539.61 KiB | 13.49 MiB/s, done.
Total 92 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), done.
To https://github.com/ComputerScienceHIT/HIT-Lab1-1190202105.git
 c6f960f..4c04860 master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```



Git push -u origin master

### 3.3 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。

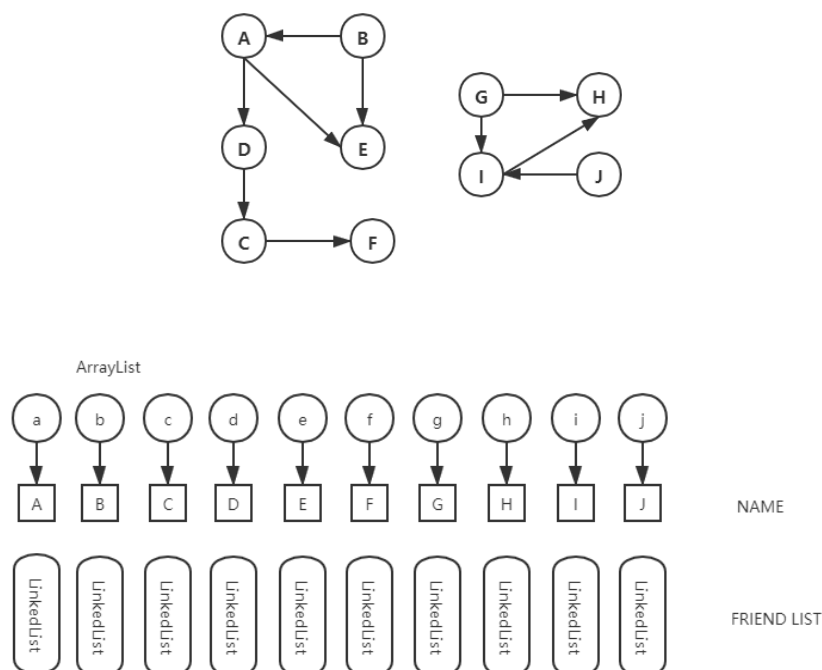


#### 3.3.1 设计/实现 FriendshipGraph 类

给出你的设计和实现思路/过程/结果。

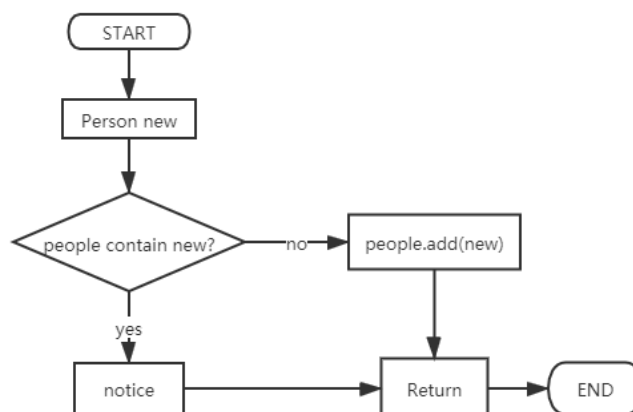
如图所示，创建网络时，将 `Person` 添加到 `Friend List` 当中，形成了一个有向图，姑且不考虑朋友之间的相互性，即无向性。接下来按照给出的例子，要对

Person、addVertex、addEdge、getDistance 进行分析。



### 3.3.1.1 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。



### 3.3.1.2 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`。

```

// add a new edge between two vertexes
public void addEdge (Person personA, Person personB) {
    if (!people.contains(personA) || !people.contains(personB)) {
        System.out.println("Not created!");
        System.exit(0);
    }
    // The vertex can't have an edge pointing to itself.
    if (personA.name() == personB.name())
        System.out.println("The vertex " +
            personA.name() + " can't have an edge pointing to itself");
    // The edge already existed.
    else if (personA.friendList().contains(personB))
        System.out.println("There Already have existed an edge!");
    else personA.addFriend(personB);
}

```

### 3.3.1.3 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：

```

// get the distance between the two vertexes, BFS
public int getDistance(Person Person1, Person Person2) {
    HashSet<Person> visited = new HashSet<Person>();
    // the distance is 0, when they are the same person
    if (Person1 == Person2)
        return 0;
    int distance = 0, temp;
    visited.add(Person1);
    LinkedList<Person> queue = new LinkedList<Person>();
    queue.addAll(Person1.friendList());
    int levelSize = Person1.friendList().size();
    do {
        temp = 0;
        distance ++;
        for (int i = 0; i < levelSize; i++) {
            if (queue.getFirst() == Person2)
                return distance;
            else if (visited.contains(queue.getFirst()))
                queue.removeFirst();
            else {
                // all the friends of the guy enter the queue
                queue.addAll(queue.getFirst().friendList());
                temp += queue.getFirst().friendList().size();
                // set the guy as visited
                visited.add(queue.getFirst());
                queue.removeFirst();
            }
        }
        // all the reachable are visited, but person 2 not found
        if (temp == 0) return -1;
        levelSize = temp;
    } while(true);
}

```



### 3.3.2 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。

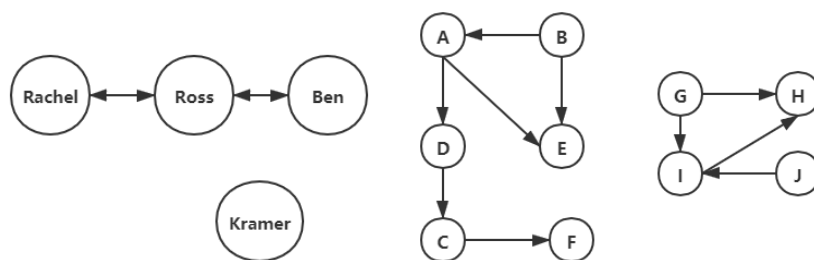
```
public class Person {  
    private String name; // Save the name  
    private LinkedList<Person> friendList; // Save one's friends  
  
    // Method  
    public Person(String str) {  
        this.name = str;  
        friendList = new LinkedList<Person>();  
    }  
    // add friend  
    public 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;  
    }  
}
```

### 3.3.3 设计/实现客户端代码 main()

直接使用实验要求报告中给出的实例。

### 3.3.4 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);

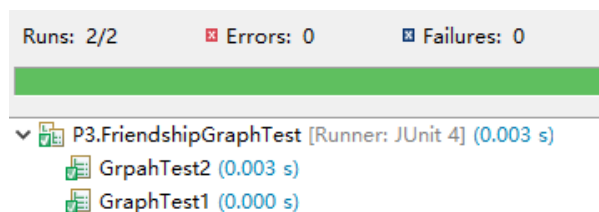
graph.addEdge(a, d);
graph.addEdge(a, e);
graph.addEdge(b, a);
graph.addEdge(b, e);
graph.addEdge(c, f);
graph.addEdge(d, c);
graph.addEdge(g, h);
graph.addEdge(g, i);
graph.addEdge(i, h);
graph.addEdge(j, i);
```

它们的测试用例如下:

```
assertEquals(1, graph.getDistance(rachel, ross));
assertEquals(2, graph.getDistance(rachel, ben));
assertEquals(0, graph.getDistance(rachel, rachel));
assertEquals(-1, graph.getDistance(rachel, kramer));

assertEquals(3, graph.getDistance(a, f));
assertEquals(-1, graph.getDistance(a, b));
assertEquals(-1, graph.getDistance(e, f));
assertEquals(2, graph.getDistance(d, f));
assertEquals(-1, graph.getDistance(a, j));
assertEquals(0, graph.getDistance(i, i));
assertEquals(-1, graph.getDistance(g, j));
assertEquals(1, graph.getDistance(i, h));
```

JUnit 检测结果:



## 4 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 <code>isLegalMagicSquare</code> 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 <code>generateMagicSquare</code> 函数	按计划完成

		数, 及其他剩余内容	
2021-05-16	21:00-23:20	编写问题 2 的 drawSquare 函数、calculateRegularPolygonAngle 函数、calculatePolygonSidesFromAngle 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

## 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
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 来储存访问过的人 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

## 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. 还是不太习惯 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。

## 6.2 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。

(2) 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。

(3) 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。

(4) 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

(5) 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

(6) 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