



哈爾濱工業大學  
HARBIN INSTITUTE OF TECHNOLOGY

# 2020 年春季学期 计算机学院《软件构造》课程

## Lab 1 实验报告

姓名	熊峰
学号	1190200708
班号	1903008
电子邮件	xiong257246@outlook.com
手机号码	13114991114

## 目录

1 实验目标概述 .....	1
2 实验环境配置 .....	1
3 实验过程 .....	2
3.1 Magic Squares .....	2
3.1.1 isLegalMagicSquare() .....	3
3.1.2 generateMagicSquare() .....	5
3.2 Turtle Graphics .....	7
3.2.1 Problem 1: Clone and import .....	8
3.2.2 Problem 3: Turtle graphics and drawSquare .....	8
3.2.3 Problem 5: Drawing polygons .....	9
3.2.4 Problem 6: Calculating Bearings .....	10
3.2.5 Problem 7: Convex Hulls .....	13
3.2.6 Problem 8: Personal art .....	15
3.2.7 Submitting .....	16
3.3 Social Network .....	17
3.3.1 设计/实现 FriendshipGraph 类 .....	17
3.3.2 设计/实现 Person 类 .....	18
3.3.3 设计/实现客户端代码 main() .....	18
3.3.4 设计/实现测试用例 .....	19
4 实验进度记录 .....	21
5 实验过程中遇到的困难与解决途径 .....	22
6 实验过程中收获的经验、教训、感想 .....	22
6.1 实验过程中收获的经验教训 .....	22
6.2 针对以下方面的感受 .....	22

## 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。

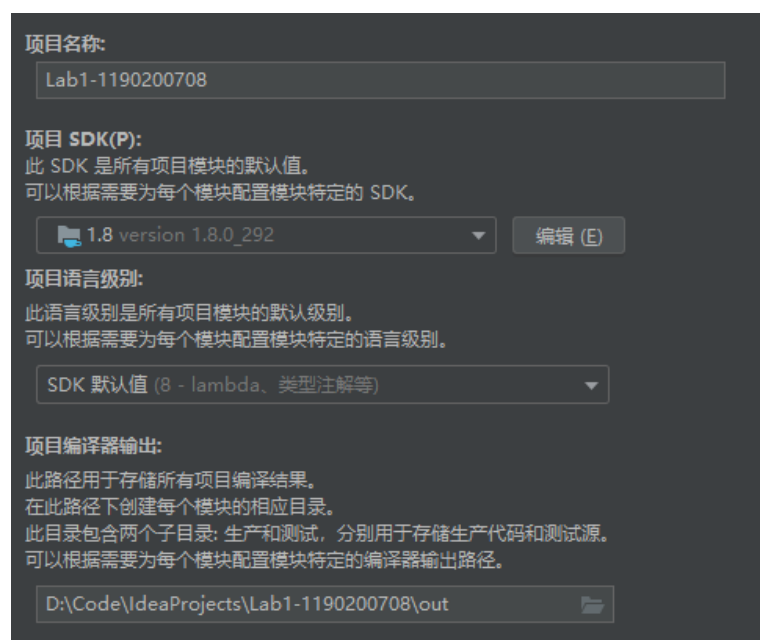
另一方面, 利用 Git 作为代码配置管理的工具, 学会 Git 的基本使用方法。

- 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

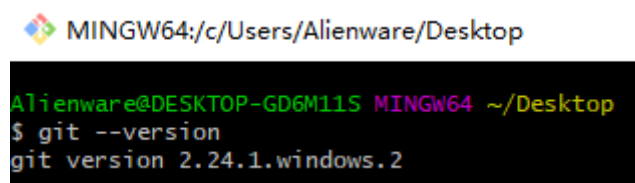
环境配置:

IntelliJ IDEA 2020.3.1 (JDK 1.8 Junit 4.12(下载到 lib 目录中))



Git:

Git version 2.24.1.windows.2



```
MINGW64:/c/Users/Alienware/Desktop
Alienware@DESKTOP-GD6M11S MINGW64 ~/Desktop
$ git --version
git version 2.24.1.windows.2
```

GtiHub Lab1 URL : <https://github.com/ComputerScienceHIT/HIT-Lab1-1190200708.git>

## 3 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！）。

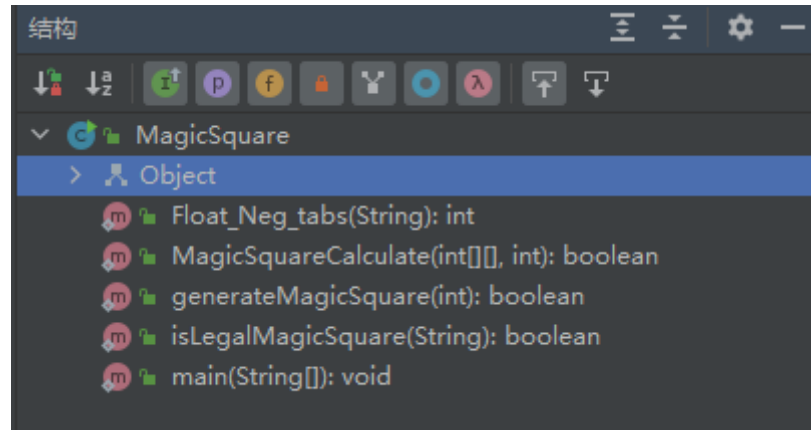
为了条理清晰，可根据需要在各节增加三级标题。

### 3.1 Magic Squares

MagicSquare 主要实现了以下功能：

1. 读取所给的五个 txt 文件，并用数组保存；
2. 检验所给的五个 txt 文件是否为幻方矩阵
  - 2.1 检验文件是否为空
  - 2.2 检验每行是否存在小数
  - 2.3 检验每行是否存在负数
  - 2.4 检验每行是否存在非法字符
  - 2.5 检验行的数量和列的数量是否相同
  - 2.6 检验每行元素个数是否相同
  - 2.7 检验行、列、对角线元素和是否相同
3. 若是幻方矩阵则返回 true，否则抛出并打印错误；
4. 检验要求创建的幻方矩阵的数是否合法，考虑奇数或偶数时的异常，若不存在异常，则生成  $n \times n$  的矩阵，并写入 6.txt 文件

函数结构如下：



### 3.1.1.1 isLegalMagicSquare()

#### 3.1.1.1.1 设计思路:

首先通过 `System.getProperty("user.dir") + "\\src\\P1\\txt\\" + fileName` 和 `Scanner` 读取文件，每次读取按行读取，读取后，将每行元素作为参数，调用 `Float_Neg_tabs()` 函数检验每行文本是否存在小数、负数、除 `\t` 以外的其他字符。若读取的数都为正数，且每行元素个数相同，且行数与列数相同，则调用 `MagicSquareCalculate()` 函数计算读取的矩阵是否为幻方矩阵。

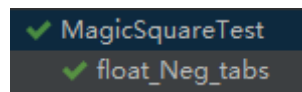
#### 3.1.1.1.2 所调用的函数

`public static int Float_Neg_tabs(String Number)`

功能：用于检验读取到的每行元素是否存在非法数据，若字符串合法则返回 0，若字符串中含有负数，则返回 1，若字符串中含有非 `\t` 的字符，则返回 2，若字符串结尾为非 `\t` 字符，则返回 3，若字符串中含有小数，则返回 4。

实现过程：对读取的字符串分析，若字符串最后一个字符为非 `'0'-'9'` 且非 `\t`，则返回 3，表明，当前字符串的结尾为非 `\t` 字符；对字符串每个字符检查，若存在字符大于 `'9'` 或者小于 `'0'`，则检查是否为 `'.'` 或 `\t` 或 `'-'`，若存在 `'-'` 符号，则表明存在负数，函数返回 1；若存在 `'.'` 符号，则表明存在小数，函数返回 4；若该符不为以上三种符号，则存在其他非法字符，返回 2；对以上检查完毕后，若无异常则，返回 0；

运行结果：通过 Junit4.12 测试，结果如下。



```
@Test
public void float_Neg_tabs() {

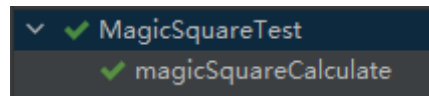
    String str0 = "12\t32\t14";
    String str1 = "-13\t23\t13";
    String str2 = "*12\t42\t13";
    String str3 = "51\t15\t32 ";
    String str4 = "0.45\t0.32\t14";
    assertEquals( expected: 0, MagicSquare.Float_Neg_tabs(str0));
    assertEquals( expected: 1, MagicSquare.Float_Neg_tabs(str1));
    assertEquals( expected: 2, MagicSquare.Float_Neg_tabs(str2));
    assertEquals( expected: 3, MagicSquare.Float_Neg_tabs(str3));
    assertEquals( expected: 4, MagicSquare.Float_Neg_tabs(str4));
}
```

```
public static boolean MagicSquareCalculate(int[][] Num, int count)
```

功能：计算每行每列对角线元素和是否相同，若相同则返回 true，否则返回 false。

实现过程：首先计算第一行元素和，与每行、每列、写对角线元素和比较，若相等，则表明该矩阵为幻方矩阵，否则不为幻方矩阵。

运行结果：通过 Junit4.12 测试，与预期结果一致，表明函数实现所需功能。



```
@Test
public void magicSquareCalculate() {
    int count = 7;
    int [][]numbers = new int[7][7];
    assertEquals( expected: true, MagicSquare.MagicSquareCalculate(numbers,count));
    numbers[0][0]=1;
    assertEquals( expected: false, MagicSquare.MagicSquareCalculate(numbers,count));
    int row = 0, col = count / 2, i, j, square = count * count;
    for (i = 1; i <= square; i++) {
        numbers[row][col] = i;
        if (i % count == 0)
            row++;
        else {
            if (row == 0)
                row = count - 1;
            else
                row--;
            if (col == (count - 1))
                col = 0;
            else
                col++;
        }
    }
    assertEquals( expected: true, MagicSquare.MagicSquareCalculate(numbers,count));
}
```

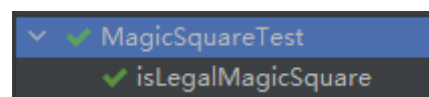
### 3.1.1.3 过程

读取文件后，调用函数判断每行内容是否合法，若不合法，则抛出相应异常，再检查每行元素个数是否相同，若不相同，则抛出异常；若读取的行数超过列数时，抛出异常；若读取文件完毕后，行数小于列数，抛出异常。

调用函数，计算读取的矩阵是否为幻方矩阵。

### 3.1.1.4 运行结果

通过 Junit4.12 测试，发现运行结果与预期一致。



```
@Test
public void isLegalMagicSquare() {
    try {
        boolean a = MagicSquare.isLegalMagicSquare( fileName: "test1.txt");
    } catch (Exception e) {
        assertEquals( expected: "文件为空!", e.getMessage());
    }
    try {
        boolean a = MagicSquare.isLegalMagicSquare( fileName: "test2.txt");
    } catch (Exception e) {
        assertEquals( expected: "文本中含有负数!", e.getMessage());
    }
    try {
        boolean a = MagicSquare.isLegalMagicSquare( fileName: "test3.txt");
    } catch (Exception e) {
        assertEquals( expected: "文本中含有非'\t'的字符!", e.getMessage());
    }
    try {
        boolean a = MagicSquare.isLegalMagicSquare( fileName: "test4.txt");
    } catch (Exception e) {
        assertEquals( expected: "文本行结尾为非'\t'的字符!", e.getMessage());
    }
    try {
        boolean a = MagicSquare.isLegalMagicSquare( fileName: "test5.txt");
    } catch (Exception e) {
        assertEquals( expected: "文本存在小数!", e.getMessage());
    }
    try {
        assertEquals( expected: true, MagicSquare.isLegalMagicSquare( fileName: "1.txt"));
    } catch (Exception e) {
    }
}
```

### 3.1.2 generateMagicSquare()

#### 3.1.2.1 设计思路

修改后的程序:

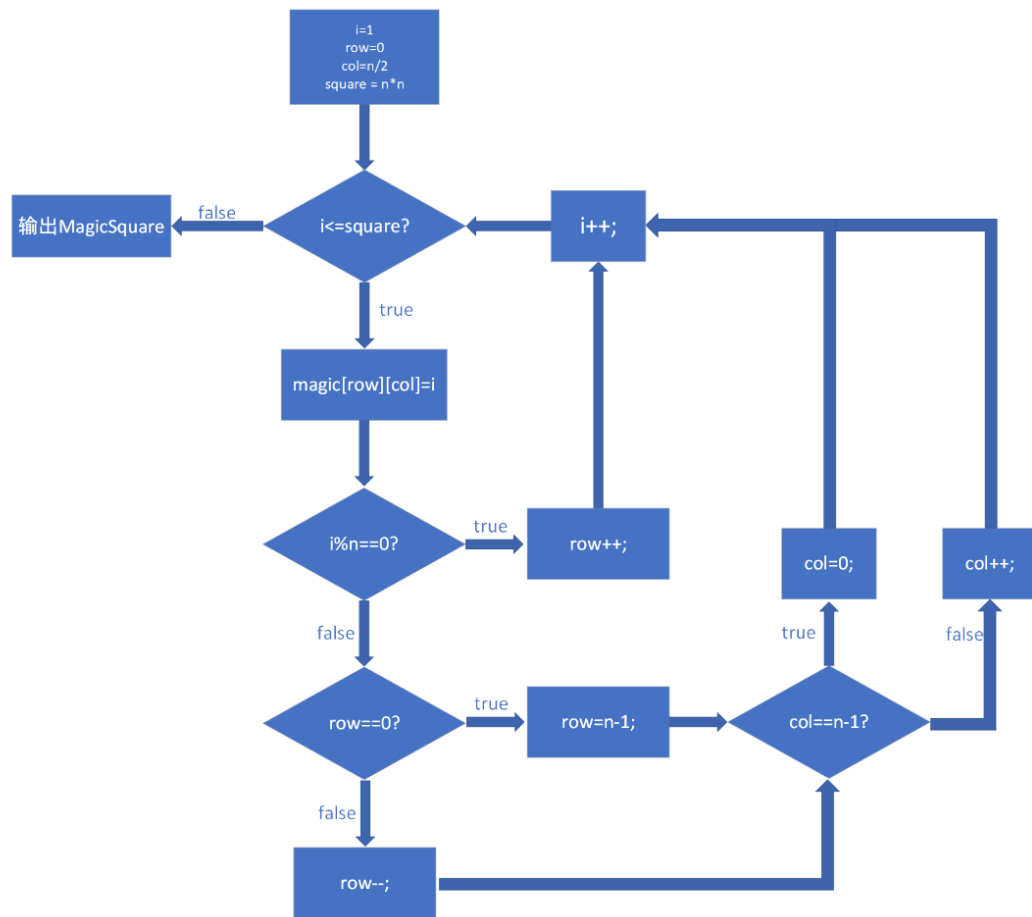
首先判断参数是否为负数或偶数,并抛出相应异常。若检验出参数为正奇数,则此时可以通过所给代码生成幻方矩阵。并将幻方矩阵通过 `PrintWriter` 写入到 6.txt 中,并通过函数判断是否为幻方矩阵。

源程序生成幻方矩阵思路 (Siamese 方法):

1. 首先将 1 放置在第一行中间;
2. 顺序将 2, 3, ... 等数依次放在右上方格中;
3. 当右上方格出界的时候,由另一边进入;
4. 当右上方格中已经填有数,则把数填入正下放的表格中;
5. 按照以上步骤直到填写完所有  $N^2$  个方格。

#### 3.1.2.2 流程图

以下流程图为源代码生成幻方矩阵的流程。



### 3.1.2.3 运行结果

通过 Junit4.12 对代码测试，在负数、偶数时候抛出异常，在正奇数时候，生成对应的幻方矩阵，符合预期要求。

```

✓ 测试 已通过: 1 共 1 个测试 - 3 ms
C:\Users\Alienware\.jdk\corret
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
  
```



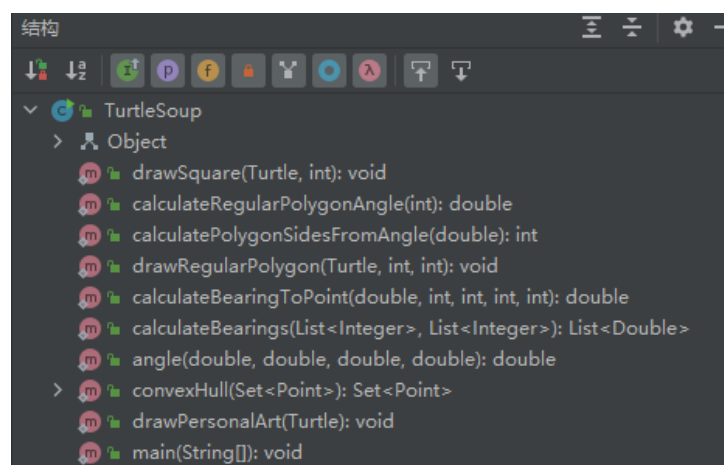
```
@Test
public void generateMagicSquare() {

    try {
        assertEquals( expected: true, MagicSquare.generateMagicSquare( n: 7));
        assertEquals( expected: true, MagicSquare.generateMagicSquare( n: 8));
    } catch (Exception e) {
        assertEquals( expected: "n为偶数，造成数组越界!", e.getMessage());
    }
    try {
        boolean a = MagicSquare.generateMagicSquare( n: -1);
    } catch (Exception e) {
        assertEquals( expected: "n为负数，不支持申请大小为负数的数组!", e.getMessage());
    }
}
```

### 3.2 Turtle Graphics

Turtle Graphics 主要实现以下几个功能:

1. 画正方形
2. 计算正多边形内角角度
3. 通过正多边形内角，计算正多边形的边数
4. 画正多边形
5. 已知当前对 y 轴正半轴的偏角，求从点 current 到点 target 需要旋转的角度
6. 当前方向为 y 轴正半轴时，求从 xCoords 的点到 yCoords 的点需要旋转的角度的 List
7. 当前方向为 x 轴正半轴时，求从点 A 到点 B 需要旋转的角度
8. 在给定一些点的坐标的时候，求这些点的凸包的集合
9. 绘画出个人作品
10. 测试以上功能



### 3.2.1 Problem 1: Clone and import

使用 git 命令 `git clone [url]` 下载一个项目和他的整个代码历史

`git clone https://github.com/rainywang/Spring2021\_HITCS\_SC\_Lab1`

在本地使用 `git init` 命令, 即可新建一个 Git 代码库。

在本地使用 `git` 命令 `git log` 即可查看 `git` 日志详细信息。

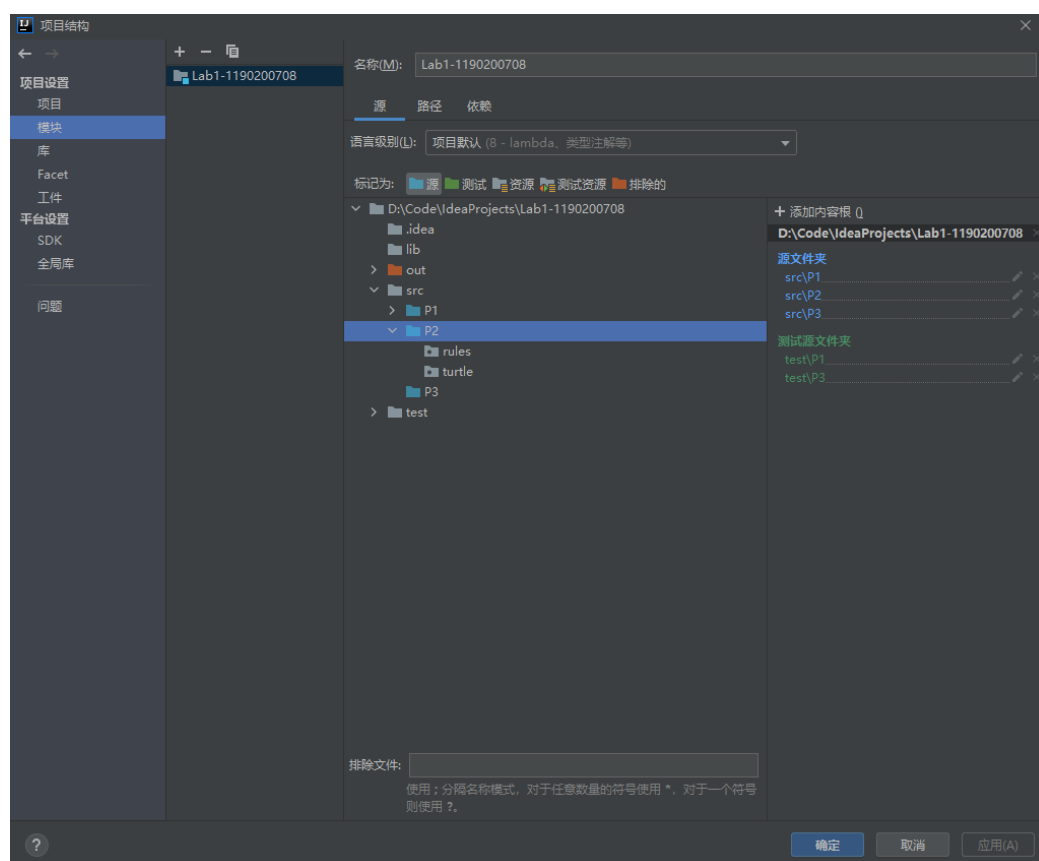
在 IntelliJ IDEA 中打开项目结构, 选择模块, 在 `src` 中建立 `P2` 目录, 并标记为源根, 将即可正常运行。

```
$ git log
commit e1338fe707786da23d3d86bab09c267b5d9ff5a5 (HEAD -> master, origin/master)
Author: Baileys <xiong257246@outlook.com>
Date: Sun May 23 16:11:26 2021 +0800

    modification

commit de05ee3080f091fb217df998c173345d50859091
Author: Baileys <xiong257246@outlook.com>
Date: Sun May 23 13:53:16 2021 +0800

    finish
```



### 3.2.2 Problem 3: Turtle graphics and drawSquare

#### 3.2.2.1 设计思路

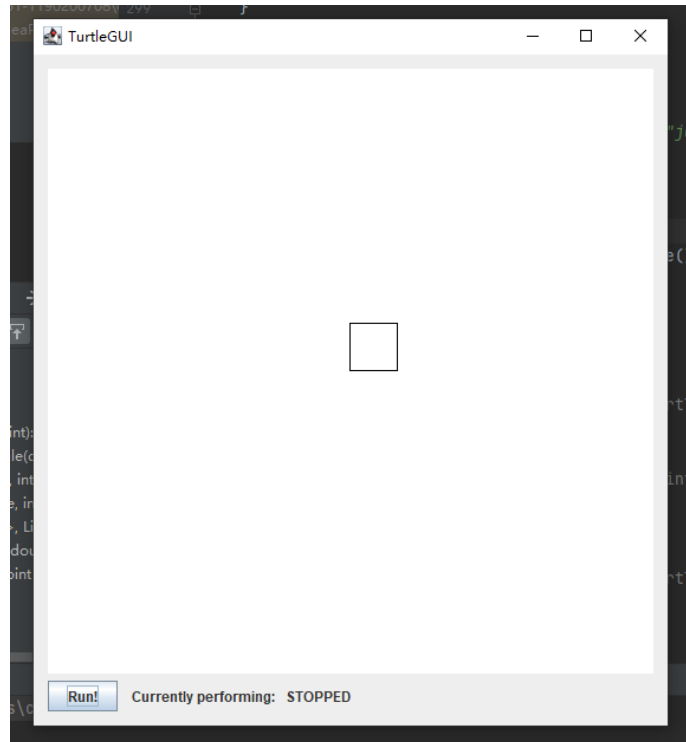
利用 turtle 的 forward 和 turn 方法, 完成转动角度, 向前移动等操作, 画出边长, 并确定多边形的内角。

#### 3.2.2.2 过程

使 turtle 首先向当前方向移动 sideLength 个单位, 然后使用 turn 方法, 向右转 90 度, 再向当前方向移动 sideLength 个单位, 再使用 turn 方法, 向右转 90 度, 再向当前方向移动 sideLength 个单位, 再向右转 90 度, 再向当前方向移动 sideLength 个单位, 即可得到一个边长为 sideLength 的正方形。

#### 3.2.2.3 运行结果

该图形的运行结果如下, 成功绘制出边长为 40 个单位的正方形。



### 3.2.3 Problem 5: Drawing polygons

#### 3.2.3.1 设计思路

##### 1. calculateRegularPolygonAngle

利用公式即可求正多边形内角。

$$\text{Angle} = 180 * (\text{sides} - 2) / \text{sides}$$

##### 2. drawRegularPolygon

使用已经实现的正多边形内角公式, 即可计算出要绘制的正多边形的内角, 进行绘制。

#### 3.2.3.2 过程

##### 1. calculateRegularPolygonAngle

利用公式很容易计算出所求的内角。

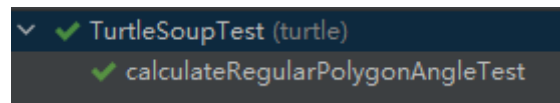
##### 2. drawRegularPolygon

利用已经实现的计算正多边形内角的公式, 即很容易计算出内角, 通过 turtle 的 turn 方法, 即可实现角度的转换, 在每绘制一条边的时候, 转动角度 180-angle。

再进行绘制。如此循环，即可画出正多边形。

### 3.2.3.3 运行结果

通过 Junit4.12 对 `calculateRegularPolygonAngleTest` 检验，与预期结果一致。

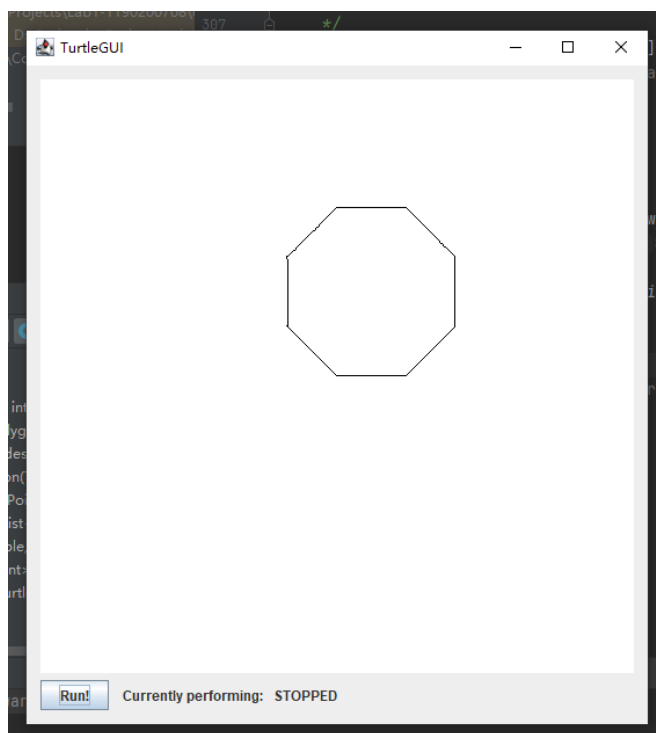


```
/**
 * Tests calculateRegularPolygonAngle.
 */
@Test
public void calculateRegularPolygonAngleTest() {
    assertEquals( expected: 60.0, TurtleSoup.calculateRegularPolygonAngle( sides: 3), delta: 0.001);
    assertEquals( expected: 128.57, TurtleSoup.calculateRegularPolygonAngle( sides: 7), delta: 0.01);
    assertEquals( expected: 108.0, TurtleSoup.calculateRegularPolygonAngle( sides: 5), delta: 0.001);
}
```

对 `drawRegularPolygon` 的功能检验：

例如：绘制一个边长为 60 的八边形，即调用 `drawRegularPolygon` 函数，  
`drawRegularPolygon(turtle1,8,60);`

运行结果如下，符合预期结果。



## 3.2.4 Problem 6: Calculating Bearings

### 3.2.4.1 设计思路

#### 1. `calculateBearingToPoint`

通过斜率计算角度，即可计算出要旋转的角度

#### 2. `calculateBearings`

通过斜率计算角度，将所得到的角度保存到 `List` 中

## 3.2.4.1 过程

## 1. calculateBearingToPoint

比较 currentX 与 targetX 的值

(1)若 currentX 与 targetX 的值相等, 则说明两点在直线  $x=currentX$  上。

若  $currentY > targetY$ , 则说明此时需要将角度调整为 y 的负半轴, 因此此时转动的角度为  $((540 - currentBearing) \% 360 + 360) \% 360$

若  $currentY < targetY$ , 则说明此时需要将角度调整为 y 的正半轴, 因此此时转动的角度为  $((360 - currentBearing) \% 360 + 360) \% 360$ ;

由于 turtle 方法只能向右转动, 通过加 360 并取余得到为正的角。

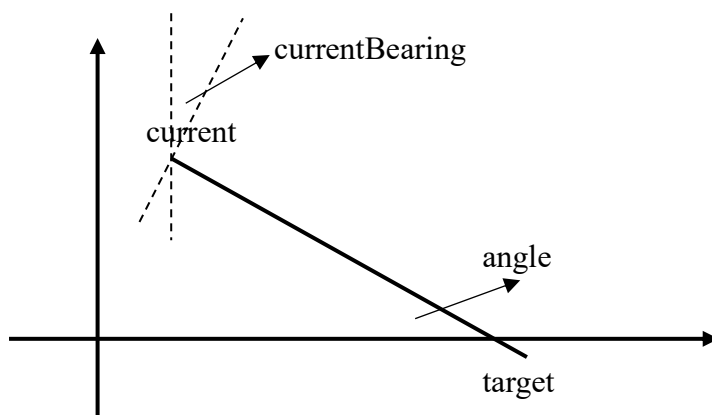
(2)若此时  $currentX < targetX$ , 则说明此时 current 在 target 左侧, 即可通过斜率计算公式, 和 Math 库函数中的 atan 函数, 计算角度。

与 x 轴正半轴角度为  $angle = (Math.atan((double)(targetY - currentY) / (double)(targetX - currentX))) * 180 / Math.PI$ ;

如图所示, 即可得到当前的与 x 轴的夹角, 以靠近 y 的正半轴为正, 靠近 y 的负半轴为负, 不难得出, 实际需要转动的角度为

$((450 - angle - currentBearing) \% 360 + 360) \% 360$

同理, 通过加 360 并取余得到为正的角。



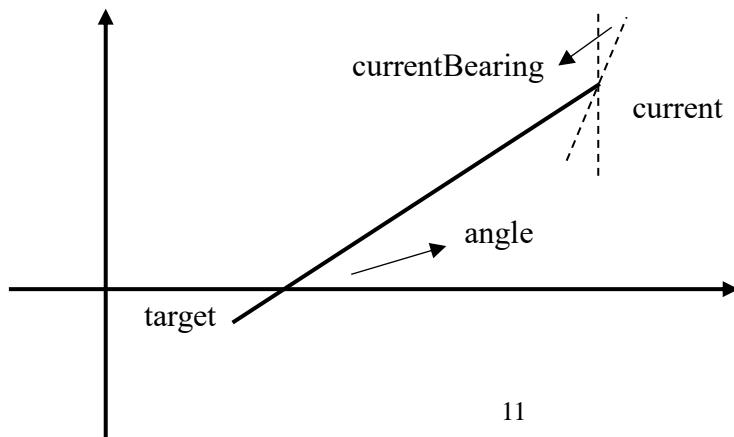
(3)若此时  $currentX > targetX$ , 则说明此时 current 在 target 右侧, 即可通过斜率计算公式, 和 Math 库函数中的 atan 函数, 计算角度。

与 x 轴正半轴角度为  $angle = (Math.atan((double)(targetY - currentY) / (double)(targetX - currentX))) * 180 / Math.PI$ ;

如图所示, 即可得到当前的与 x 轴的夹角, 以靠近 y 的正半轴为正, 靠近 y 的负半轴为负, 不难得出, 实际需要转动的角度为

$((630 - angle - currentBearing) \% 360 + 360) \% 360$

同理, 通过加 360 并取余得到为正的角。



## 2. calculateBearings

新建变量 `currentBearing`, 保存当前与 y 轴正半轴的角度

比较 `xCoords.get(i)` 与 `xCoords.get(i + 1)` 的值( $i$  从 0 开始到  $n-2$ )

(1) 若 `xCoords.get(i) = xCoords.get(i + 1)`

判断 `yCoords.get(i)` 与 `yCoords.get(i + 1)` 的关系

此时与 1 中(1)情况类似(详细分析在 1 中),

若 `yCoords.get(i) > yCoords.get(i + 1)`

`element = ((540 - currentBearing) % 360 + 360) % 360;`

若 `yCoords.get(i) < yCoords.get(i + 1)`

`element = ((360 - currentBearing) % 360 + 360) % 360;`

(2) 若 `xCoords.get(i) < xCoords.get(i + 1)`

此时与 1 中(2)情况类似(详细分析在 1 中),

`element = ((450 - angle - currentBearing) % 360 + 360) % 360;`

(3) 若 `xCoords.get(i) > xCoords.get(i + 1)`

此时与 1 中(3)情况类似(详细分析在 1 中),

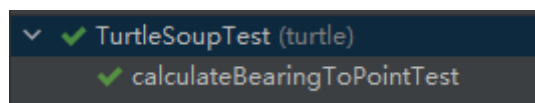
`element = ((630 - angle - currentBearing) % 360 + 360) % 360;`

此时 `currentBearing += element` 并对 360 求余, 保存旋转后的角度。并将 `element` 元素添加到 List 中。

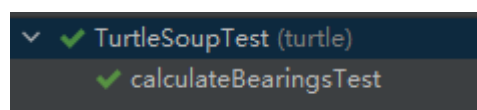
当 `xCoords` 与 `yCoords` 中的元素遍历后, 返回 List。

### 3.2.4.3 运行结果

使用 Junit4.12 测试, 与预期结果一致。



```
@Test
public void calculateBearingToPointTest() {
    assertEquals( expected: 0.0, TurtleSoup.calculateBearingToPoint( currentBearing: 0.0, currentX: 0, currentY: 0, targetX: 0, targetY: 1), delta: 0.001);
    assertEquals( expected: 90.0, TurtleSoup.calculateBearingToPoint( currentBearing: 0.0, currentX: 0, currentY: 0, targetX: 1, targetY: 0), delta: 0.001);
    assertEquals( expected: 359.0, TurtleSoup.calculateBearingToPoint( currentBearing: 1.0, currentX: 4, currentY: 5, targetX: 4, targetY: 6), delta: 0.001);
}
```



```
@Test
public void calculateBearingsTest() {
    List<Integer> xpoints = new ArrayList<>();
    List<Integer> ypoints = new ArrayList<>();
    xpoints.add(0);
    xpoints.add(1);
    xpoints.add(1);
    ypoints.add(0);
    ypoints.add(1);
    ypoints.add(2);

    List<Double> result = TurtleSoup.calculateBearings(xpoints, ypoints);
    assertEquals( expected: 2, result.size());
    assertEquals( expected: 45.0, result.get(0), delta: 0.001);
    assertEquals( expected: 315.0, result.get(1), delta: 0.001);
}
```

### 3.2.5 Problem 7: Convex Hulls

#### 3.2.5.1 设计思路

利用 graham 凸包扫描算法, 实现求凸包的功能。Graham 扫描的思想是先找到凸包上的一个点, 然后从那个点开始按逆时针方向逐个找凸包上的点, 实际上就是进行极角排序, 然后对其查询使用。

#### 3.2.5.2 过程

1. 将 points 中的所有点, 放在坐标系中, 以最下面的点为坐标原点, 对所有点进行平移;

2. 求每个点与 O 点的连线与 x 轴正半轴所形成的夹角, 并对其进行排序, 若角度相同, 则取与 O 点较近的元素, 认为其较小, 此时认为 O 点最小;

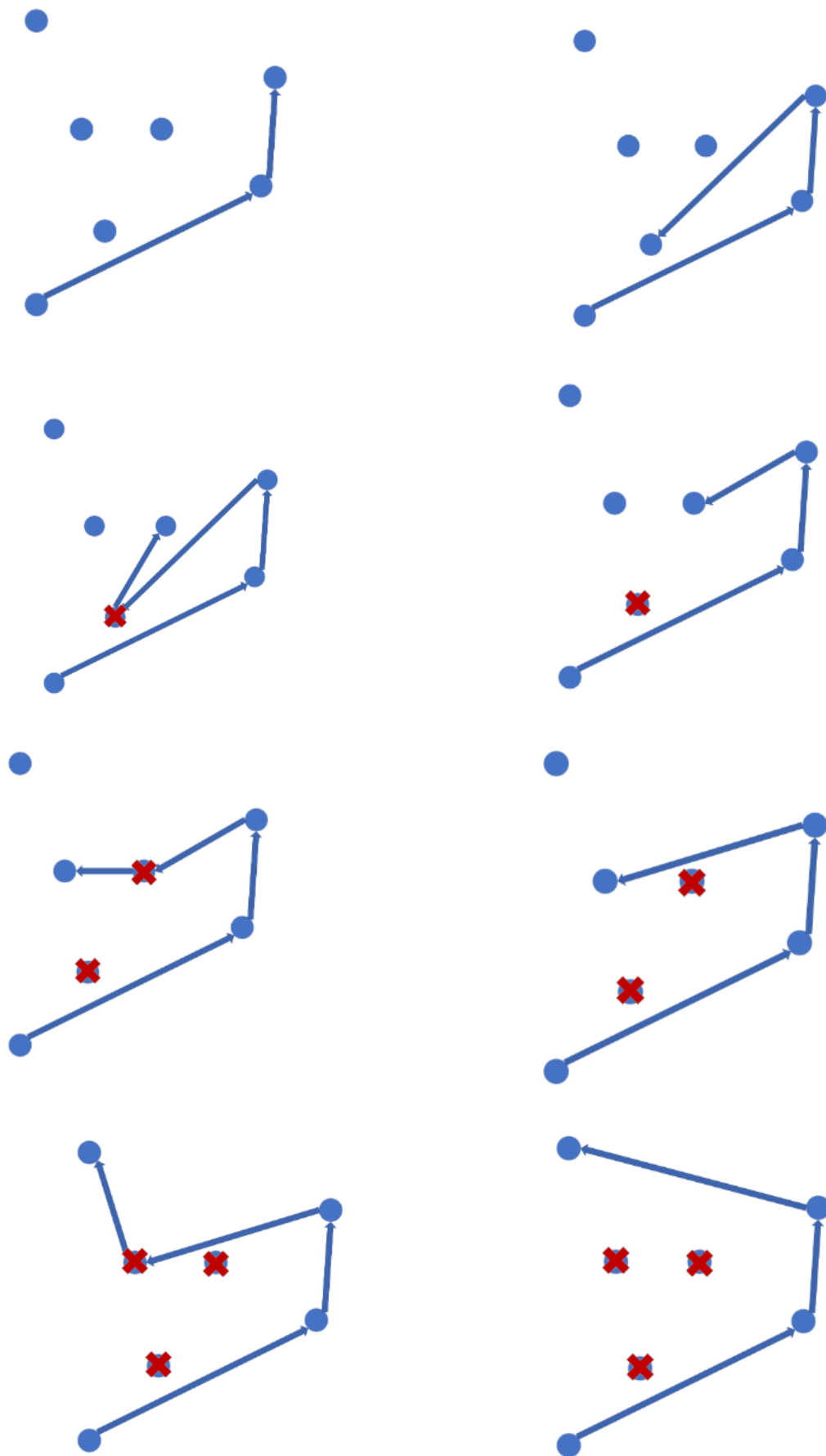
3. 取步骤 2 中, 所排列的集合中, 取出最小的 3 个点, 假设当前的方向一致为 X 正半轴, 分别计算 point1 到 point2、point2 到 point3, 需要转动的角度 angle1、angle2。由于第一次取出的是 O 点与最下角的点, 因此第一次一定有  $\text{angle1} < \text{angle2}$ 。此时将 point1、point2、point3 也存入栈内。

4. 继续从排列好的集合中取出新的元素 point3, 取出栈顶两个元素作为 point2 和 point1, 再将这两个元素压栈, 若满足  $\text{angle1} < \text{angle2}$ , 则将 point3 存在栈中。若  $\text{angle1} \geq \text{angle2}$ , 则进入步骤 5 中。

5. 若  $\text{angle1} \geq \text{angle2}$ , 则先弹一次栈, 该点不满足凸包的定义, 再弹两次栈分别作为 point2 和 point1, 此时计算 angle1 和 angle2, 若满足  $\text{angle1} < \text{angle2}$ , 则将 point3 压栈, 并进入步骤 4, 否则继续进行步骤 5。

6. 重复以上操作, 直到 points 中所有元素都被遍历后, 栈中的元素即为所求的凸包。

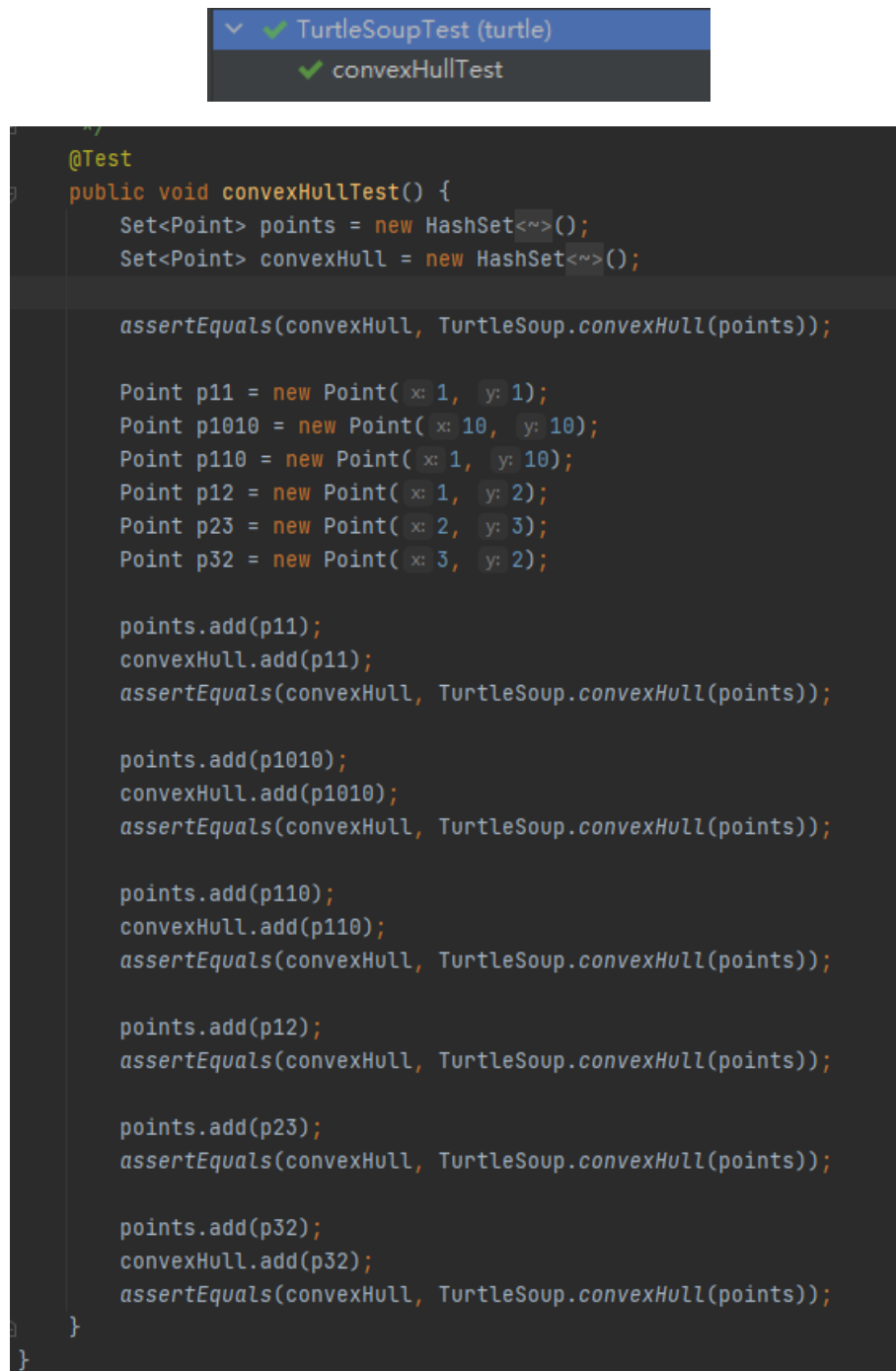
以下为实例:





### 3.2.5.3 运行结果

运行结果如下，与预期结果一致。



```
@Test
public void convexHullTest() {
    Set<Point> points = new HashSet<>();
    Set<Point> convexHull = new HashSet<>();

    assertEquals(convexHull, TurtleSoup.convexHull(points));

    Point p11 = new Point(1, 1);
    Point p1010 = new Point(10, 10);
    Point p110 = new Point(1, 10);
    Point p12 = new Point(1, 2);
    Point p23 = new Point(2, 3);
    Point p32 = new Point(3, 2);

    points.add(p11);
    convexHull.add(p11);
    assertEquals(convexHull, TurtleSoup.convexHull(points));

    points.add(p1010);
    convexHull.add(p1010);
    assertEquals(convexHull, TurtleSoup.convexHull(points));

    points.add(p110);
    convexHull.add(p110);
    assertEquals(convexHull, TurtleSoup.convexHull(points));

    points.add(p12);
    assertEquals(convexHull, TurtleSoup.convexHull(points));

    points.add(p23);
    assertEquals(convexHull, TurtleSoup.convexHull(points));

    points.add(p32);
    convexHull.add(p32);
    assertEquals(convexHull, TurtleSoup.convexHull(points));
}
```

### 3.2.6 Problem 8: Personal art

#### 3.2.6.1 设计思路

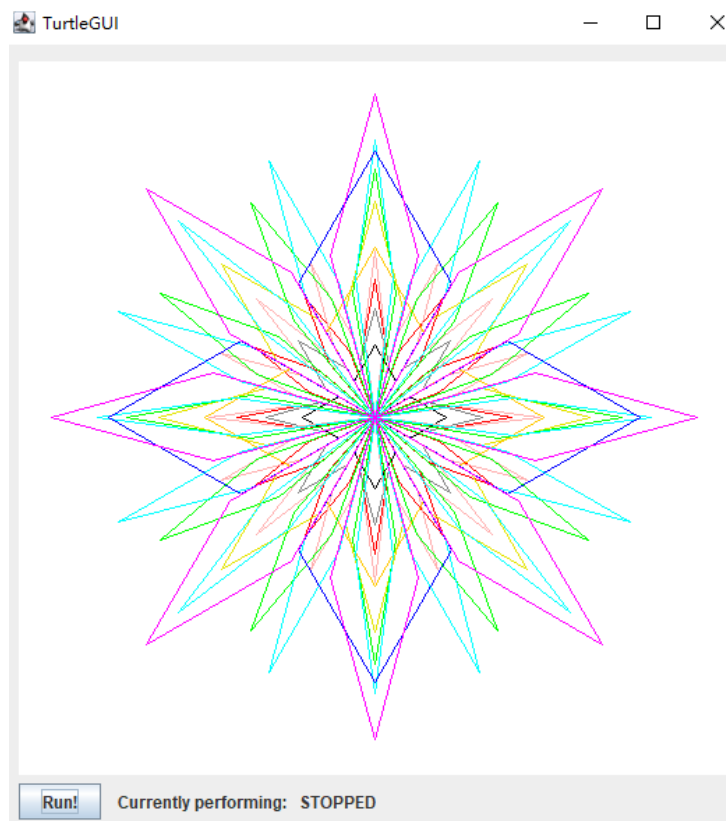
利用循环, `foreach` 等操作, 变换颜色, 旋转角的度数, 画出的图形的个数, 完成一副具有个人色彩的绘画。

### 3.2.6.2 过程

首先利用 `foreach` 语句, 取出 `pencolor` 中的所有颜色, 用每个颜色的笔画一些团, 利用循环取余等操作, 完成在循环数不同时, 绘画出不同图形的操作。

### 3.2.6.3 运行结果

最终绘制的图形如图所示:



## 3.2.7 Submitting

通过 `git` 命令

`git add .` 添加文件到本地库

`git commit -m "TurtleSoup"` 提交文件到本地库

`git remote add origin git@github.com:ComputerScienceHIT/HIT-Lab1-1190200708.git` 关联到远程库

`git push origin master` 推送到 github

### 3.3 Social Network

Social Network 主要实现了以下功能:

1. 建立 java 版的数据结构, 有向图。
2. 并完成增加边、节点的操作。
3. 广度搜索计算两个人之间最短的距离。
4. 对输入进行检查, 若不符合条件抛出相应的异常。

#### 3.3.1 设计/实现 FriendshipGraph 类

##### 3.3.1.1 设计思路

用一个 HashSet 保存有向图的节点, 在每一个 Person 的对象中, 设私有变量, 保存每个人认识的人。通过 HashSet 实现类似邻接表的存储结构, 并且能够避免添加重复等问题, 查找效率较高。人与人之间距离应该用广度搜索, 方便记录距离。

##### 3.3.1.2 过程

设置两个私有变量, private HashSet<Person> personArrayList 保存有向图的每一个节点, 即 Person; count 保存当前已经存入的人数。

设置一个构造函数, 对 FriendshipGraph 初始化。

addVertex 函数用于添加顶点, 将图中的每个人都保存到 personArrayList 中。

addEdge 函数用来添加边, 若需要添加一条边, 则调用 person1 的方法, 将 person2 的信息保存到 person1 的私有变量中。

getDistance 函数通过广度搜索获得 person1 与 person2 的之间的距离。

广度搜索步骤:

1. 将 person1 保存到 HashSet<Person> Search 中;
2. 将 person1 认识的人保存到 HashSet<Person> Search\_Next 中;
3. 将 Search 和 Search\_Next 都保存到新的 HashSet<Person> Search\_assist 中;
4. 初始化 distance = 1;
5. 若 Search\_Next 中含有 person2, 返回 distance, 即为所求距离。否则将 Search 清空, 将 Search\_Next 保存到 Search 中, 在将 Search 中所有的人认识的人的 HashSet 中所有的元素添加到 Search\_Next, 再将 Search\_Next 中所有在 Search\_assist 中保存的元素消除掉, 再将所有 Search\_Next 元素添加到 Search\_assist 中。此时 distance++;
6. 重复步骤 5, 若 Search\_Next 最终为空, 则不存在距离, 返回值为-1; 否则, person1 与 person2 之间存在路径, 且 person1 与 person2 距离为 distance。

### 3.3.2 设计/实现 Person 类

#### 3.3.2.1 设计思路

Person 中存在私有变量 `HashSet<Person> friend`, 保存当前 Person 的朋友, 并提供一个方法, 用于添加当前 Person 的 friend。

#### 3.3.2.2 过程

用一个私有变量 `HashSet<Person> friend`, 保存当前 Person 的朋友; 再用一个私有变量保存当前 Person 的姓名。

设置一个构造函数, 用所给字符串命名当前 Person。

设置一个方法, 用于给当前 Person 添加 friend。

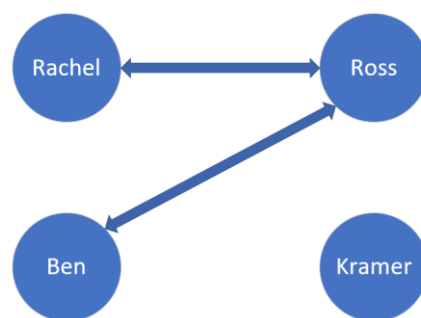
即可完成对每个人, 及其好友的设定。

### 3.3.3 设计/实现客户端代码 main()

main 函数中的代码采用实验指导提供的代码, 并对其检验。

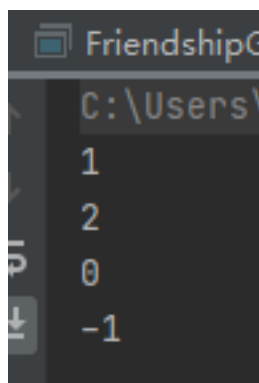
设计思路: 首先构造四个 Person 对象, 并添加四条边, 通过检查距离是否相同验证函数编写是否正确。

过程: 按照 P3 所给测试代码, 四人的关系如下。



运行结果:

运行结果与预期一致。



```
14. System.out.println(graph.getf
    //should print 1
15. System.out.println(graph.getf
    //should print 2
16. System.out.println(graph.getf
    //should print 0
17. System.out.println(graph.getf
    //should print -1
```

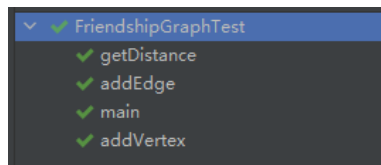
### 3.3.4 设计/实现测试用例

设计思路:

分别检验各个函数是否检测出非法数据, 并抛出异常; 并对 P3 所给代码修改后的结果作出预期, 并用 Test 检验。

过程与运行结果:

所有测试成功运行。



1. 检验 addVertex 函数时候, 检验添加重复的人产生的结果。当添加两个 kramer 时候, 抛出异常。

```
public void addVertex() {
    FriendshipGraph graph = new FriendshipGraph();
    Person rachel = new Person( name: "Rachel");
    Person ross = new Person( name: "Ross");
    Person ben = new Person( name: "Ben");
    Person kramer = new Person( name: "Kramer");
    try {
        graph.addVertex(rachel);
        graph.addVertex(ross);
        graph.addVertex(ben);
        graph.addVertex(kramer);
        graph.addVertex(kramer);
    }
    catch (Exception exception)
    {
        assertEquals( expected: "当前已经存储姓名为" + kramer.Name() + "的人!", exception.getMessage());
    }
}
```

检验 P3 题目中的, 将 Ross 改为 Rachel。当人名存在重复时, 抛出异常。这样使每个人都有独一无二的名字

```
try {
    FriendshipGraph graph = new FriendshipGraph();
    Person rachel = new Person( name: "Rachel");
    Person ross = new Person( name: "Rachel");
    Person ben = new Person( name: "Ben");
    Person kramer = new Person( name: "Kramer");
    graph.addVertex(rachel);
    graph.addVertex(ross);
}
```

```
FriendshipGraph x TurtleSoupTest.convexHullTest x
C:\Users\Alienware\.jdk\corretto-1.8.0_292\bin
Got a Exception: 当前已经存储姓名为Rachel的人!
```

2. 检验 addEdge 函数时候, 检验添加尚未添加的人与其他人的边产生的结果, 成功抛出当前未存入 Rachel 和当前 Ben 的好友已经有 Ross 好友的异常。

```

    try {
        graph.addEdge(rachel, ross);
    }
    catch (Exception exception)
    {
        assertEquals( expected: "当前关系图中尚未存入姓名为" + rachel.Name() + "的人!", exception.getMessage());
    }

    try {
        graph.addEdge(rachel, ross);
        graph.addEdge(ross, rachel);
        graph.addEdge(ross, ben);
        graph.addEdge(ben, ross);
        graph.addEdge(ben, ross);
    }
    catch (Exception exception)
    {
        assertEquals( expected: ben.Name()+"的好友中已经有"+ross.Name()+"的人!", exception.getMessage());
    }
}

```

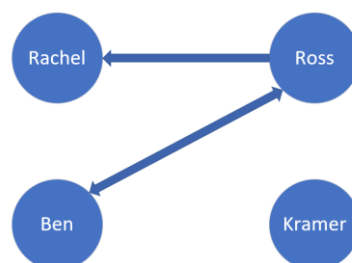
3. 检验 `getDistance` 函数时候, 检验参数若为当前尚未添加的人产生的结果。

```

}
catch (Exception exception)
{
    assertEquals( expected: "当前不存在姓名为Rachel或姓名为a的人!", exception.getMessage());
}

```

4. `main` 函数中的检验为, 将 P3 中给出的代码第十行注释掉之后的运行结果。在注释掉第十行后, 四人的关系如下, 预测结果为-1、-1、0、1。



```

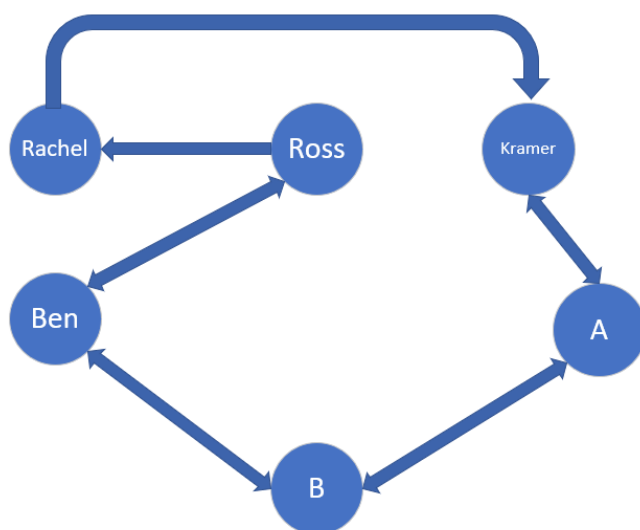
try{
    graph.addVertex(rachel);
    graph.addVertex(ross);
    graph.addVertex(ben);
    graph.addVertex(kramer);}
//graph.addEdge(rachel, ross);
graph.addEdge(ross, rachel);
graph.addEdge(ross, ben);
graph.addEdge(ben, ross);}
catch(Exception exception)
{
}

try {
    assertEquals( expected: -1, graph.getDistance(rachel, ross));
    assertEquals( expected: -1, graph.getDistance(rachel, ben));
    assertEquals( expected: 0, graph.getDistance(rachel, rachel));
    assertEquals( expected: -1, graph.getDistance(rachel, kramer));
}

```

5. `main` 函数的其他检验, 由于题设给的例子中人数较少, 因此此处自己加了一点检验。六人的关系如下:、  
由图可得, `ben` 与 `a` 的距离为 2, `ben` 与 `kramer` 的距离为 3, `b` 与 `kramer`

的距离为 2，kramer 与 rachel 的距离为 5。



```

assertEquals( expected: 2, graph.getDistance(ben,a));
assertEquals( expected: 3, graph.getDistance(ben,kramer));
assertEquals( expected: 2, graph.getDistance(b,kramer));
assertEquals( expected: 5, graph.getDistance(kramer,rachel));
}
catch (Exception exception)
{
}

```

✓ 测试 已通过: 1 共 1 个测试 - 2 ms

2 ms C:\Users\Alienware\.jdk\corretto-1.8.0\_292\bin\java.exe ...

## 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
2021-05-18	20:00-22:30	编写问题 1 的 <code>isLegalMagicSquare</code> 函数并进行测试	按计划完成
2021-05-19	15:00-20:00	编写问题 1 剩余部分	按计划完成
2021-05-20	15:45-18:00	测试问题 1 中是否存在问题，处理问题 1 中需要抛异常的部分。	按计划完成
2021-05-20	19:00-22:00	编写问题 2	按计划完成
2021-05-21	17:00-22:00	编写问题 2 剩余部分及问题 3，并进行测试	按计划完成
2021-05-22	16:00-22:00	测试问题 3 剩余部分，并完善注释	按计划完成

2021-05-23	10:00-12:00	测试 P1、P2、P3 所有功能，并完善注释，完成报告	按计划完成
------------	-------------	-----------------------------	-------

## 5 实验过程中遇到的困难与解决途径

遇到的难点	解决途径
在问题 1 中遇到，文件读取遇到问题。	通过查阅相关资料解决问题。
在问题 2 中的凸包算法遇到问题。	通过查阅相关资料并上互联网搜索资料结局问题。
对于路径的设定有问题。	通过与同学讨论并查阅资料解决。

## 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. 在写代码时，应该注意添加版本控制，一开始没有注意这点，在使用 eclipse 打开 idea 的文件后，尝试编译后，发现 idea 的源码无法运行了。
2. 对内存的了解不够深刻，在 P2 求凸包处，发现仅仅构造出在数值上相等的点无法通过测试。
3. 应该注意在写代码时候就完成报告，在写完代码后在写报告没有写代码时即刻书写的想法丰富。

### 6.2 针对以下方面的感受

- (1) Java 编程语言是否对你的口味？
- (2) 关于 Eclipse IDE
- (3) 关于 Git 和 GitHub
- (4) 关于 CMU 和 MIT 的作业
- (5) 关于本实验的工作量、难度、deadline
- (6) 关于初接触“软件构造”课程

- (1) Java 语言的语法与 C 语言和 C++语言很接近，使得学习起来比较容易。并且对 C++来说进行了简化和一定的提高，提供了丰富的类库和 API 文档，以及第三方开发包工具包，因此实际体验非常好。
- (2) Eclipse 在外观设计方面较 IDEA 略有差距，但 eclipse 总体体验还不错。实际体验上可能 IDEA 的自动修复与智能提示等功能更加强大。



- (3) GitHub 让我可以将自己的代码保存到网上, 使得在任意终端都可以查看自己的代码, Git 使本地回退版本更加容易, 总体体验很棒。
- (4) CMU 与 MIT 的作业设计非常精妙, 略有难度, 但总体不错。
- (5) 工作量略大, 但是感觉报告内容过多, 希望可以略微削减报告内容。  
Deadline 提前规划后还好。
- (6) 了解了软件构造一些基本概念, 对我写程序有很大益处。