



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	王雨桐
学号	1190200527
班号	1903002
电子邮件	<a href="mailto:1764027676@qq.com">1764027676@qq.com</a>
手机号码	18547105409

## 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() .....	2
3.1.2 generateMagicSquare() .....	6
3.2 Turtle Graphics .....	7
3.2.1 Problem 1: Clone and import .....	7
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 .....	11
3.2.5 Problem 7: Convex Hulls .....	12
3.2.6 Problem 8: Personal art .....	13
3.2.7 Submitting .....	14
3.3 Social Network .....	15
3.3.1 设计/实现 FriendshipGraph 类 .....	15
3.3.2 设计/实现 Person 类 .....	17
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 .....	18
4 实验进度记录 .....	20
5 实验过程中遇到的困难与解决途径 .....	21
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

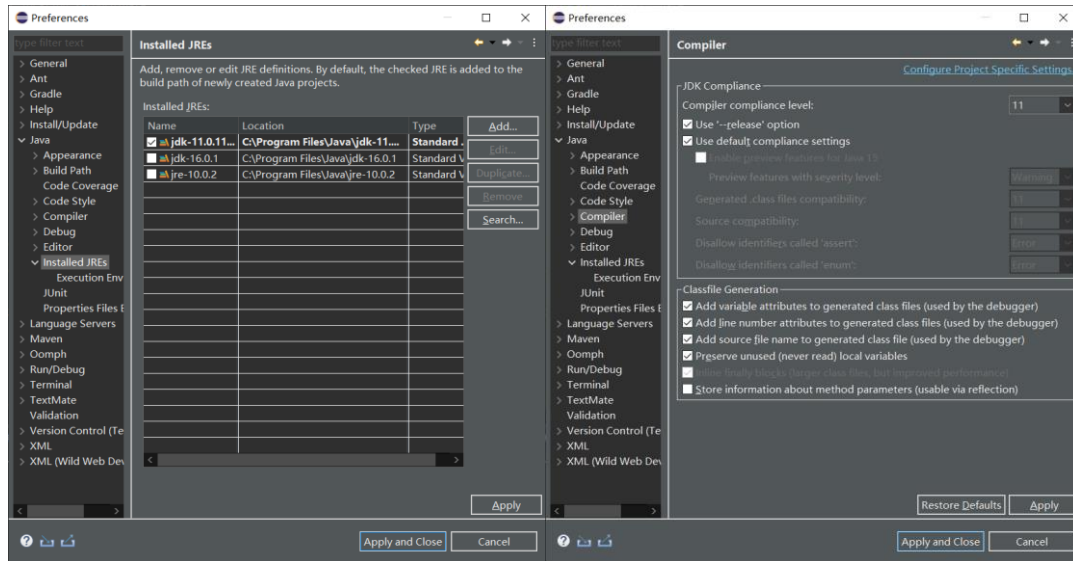
## 2 实验环境配置

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。

根据 MIT6.031 课程主页中的 Getting Started 章节对实验环境进行配置:

1. 在 Oracle 官网 <https://www.oracle.com/java/technologies/javase-downloads.html> 中下载 JDK16.0.1, 并运行安装程序。  
后担心过高的 JDK 版本会产生兼容问题, 于是又安装了 11.0.11 版本。
2. 在 Git 官网 <https://www.git-scm.com/> 中下载 Git, 运行安装程序。
3. 在系统环境变量中加入 JAVA\_HOME, 变量值为下载的 JDK 路径, 我的是 C:\Program Files\Java\jdk-16.0.1。然后在 Path 变量中加入两个条目: C:\Program Files\Java\jdk-16.0.1 (后改为 C:\Program Files\Java\jdk-11.0.11) 和 %JAVA\_HOME%\bin;%JAVA\_HOME%\jre\bin;  
在系统环境变量中加入 CLASSPATH, 变量值为 .;%JAVA\_HOME%\lib\dt.jar;%JAVA\_HOME%\lib\tools.jar;
4. 在 Eclipse 官网 <https://www.eclipse.org/downloads/> 中下载 Eclipse, 根据 Getting Started 章节中的步骤进行安装。  
后来对 JDK 降级后, Eclipse 的头文件引用出现了错误, 在查阅了网上的相关资料后, 了解到使用旧版的 JDK 只需在 Window->Preference->Java->Installed JREs 和 Window->Preference->Java->Compiler 中选择对应版本的 JDK 即可。



在这里给出你的 GitHub Lab1 仓库的 URL 地址。

<https://github.com/ComputerScienceHIT/HIT-Lab1-1190200527>

## 3 实验过程

### 3.1 Magic Squares

该任务共有两个要求:

要求 1 是编写一个 Java 程序, 能够读入从 1.txt 到 5.txt 文件中的数据并通过 `isLegalMagicSquare()` 函数判断其是否是一个 Magic Square。所谓 Magic Square, 是指每行、每列和两条对角线上的数字之和都相等的矩阵。若是, 则返回 `true`, 否则返回 `false` 并说明理由。其中对程序还有一定程度的健壮性要求, 包括对数据不符合 Magic Square 的定义、数据中出现负数等。

要求 2 是在类中加入一个现成的生成 Magic Square 的静态函数, 并在主函数中创建合适的函数接口来测试这一函数的功能。然后将这一函数做功能拓展, 增加其在某些非法输入下的健壮性, 并将生成的数据存入文本文件, 再用之前实现的方法验证其是否生成的是一个 Magic Square。

#### 3.1.1 `isLegalMagicSquare()`

##### 3.1.1.1 设计思路

首先需要考虑如何从文本文件中读入数据。我们可以按行读取, 然后根据特

定的字符将每个数字分隔开，再将其转换成数值，最后再判断是否为 Magic Square。对于判断过程，有三类特殊情况：1) 并非矩阵或矩阵行列数不相等、2) 数字并非正整数、3) 数字间并非用\t 分割。

### 3.1.1.2 设计过程

#### 1. 读入文件与数据处理：

- 1) 依次创建 File、FileReader、BufferedReader 对象，名称分别为 file、reader 以及 bfReader，然后添加所需要的头文件。
- 2) 使用 bfReader 从文件中读取一行数据到缓冲区，然后将其传递给 String 类型对象 myLine，并用 myLine 的 split 方法将一整行的文本数据以\t 为分隔符拆成若干的数字字符串，保存在一个字符串数组中。
- 3) 将数字字符串转换成 Integer 类型的数值，存储在整型二维数组中的一行内。如果遇到负数或者出现过的数字，则打印错误信息并退出。
- 4) 重复以上过程，直至文本文件中的每一行数据都被入读并且转换成整型数字存入到二维数组中。

#### 2. 判断获得的二维数组是否为 Magic Square 及特殊情况的处理：

- 1) 首先判断该二维数组是否满足矩阵中的数字都为正整数且数字之间都用\t 分割：

这里对 split 后的每个短字符串判断其是否属于正则表达式[0-9]+（表示只由数字 0~9 组成）即可，因为无论是负数、小数还是没以他分割，split 后的短字符串里都会存在非数字字符，这时直接打印错误提示信息并返回 false 即可。

```
while ((myLine = bfReader.readLine()) != null) {
    String[] words = myLine.split("\\t");

    for (j = 0; j < words.length; j++) {
        if (!isLegalNumber(words[j])) {
            System.out.println("Wrong format!");
            bfReader.close();
            return false;
        }
    }
}
```

- 2) 判断当前待加入的数字是否已经出现过：

可以使用一个 HashSet 来存储所有出现过的数值。每当向二维矩阵中加入一个数值时，先在这个 HashSet 中查找该数字，若查找到该数字，则打印错误提示信息并返回 false。

```
public static Set<Integer> numDic = new HashSet<>();

if (numDic.contains(value)) {
    System.out.println("The number " + value + " appears more than one time!");
    bfReader.close();
    return false;
}
```

- 3) 判断数据是否为矩阵：

记录第 0 行的列数（此时已经排除了没有使用\t 分隔的情况，故 split 后得到的短字符串的个数即为列数），如果后面某行的列数与第 0 行列数

不相等, 证明不是矩阵, 打印错误提示信息并返回 `false`。

```
if (colNum == -1)
    colNum = words.length;
else if (words.length != colNum) {
    System.out.println("The data is not a matrix!");
    bfReader.close();
    return false;
}
```

- 4) 判断矩阵是否行列数相等:

在将数值写入二维数组时, 会使用两个变量 `int i` 和 `int j`, `i` 是行索引, `j` 是列索引。因此, 只需比较写入完成后, `i` 和 `j` 的值是否相等, 就可以判断二维数组的行列数是否相等。

```
if (i != j) {
    System.out.println("The number of rows and columns of the square are unequal!");
    return false;
}
```

- 5) 判断两个对角线上的和是否相等。用一个简单的单层 `for` 循环就可以求得两条对角线上数字的和。若不相等, 返回 `false`; 若相等, 记录下和的值。

```
for (i = 0; i < n; i++) {
    diag1_sum += square[i][i];
    diag2_sum += square[i][n - i - 1];
}
if (diag1_sum != diag2_sum)
    return false;
```

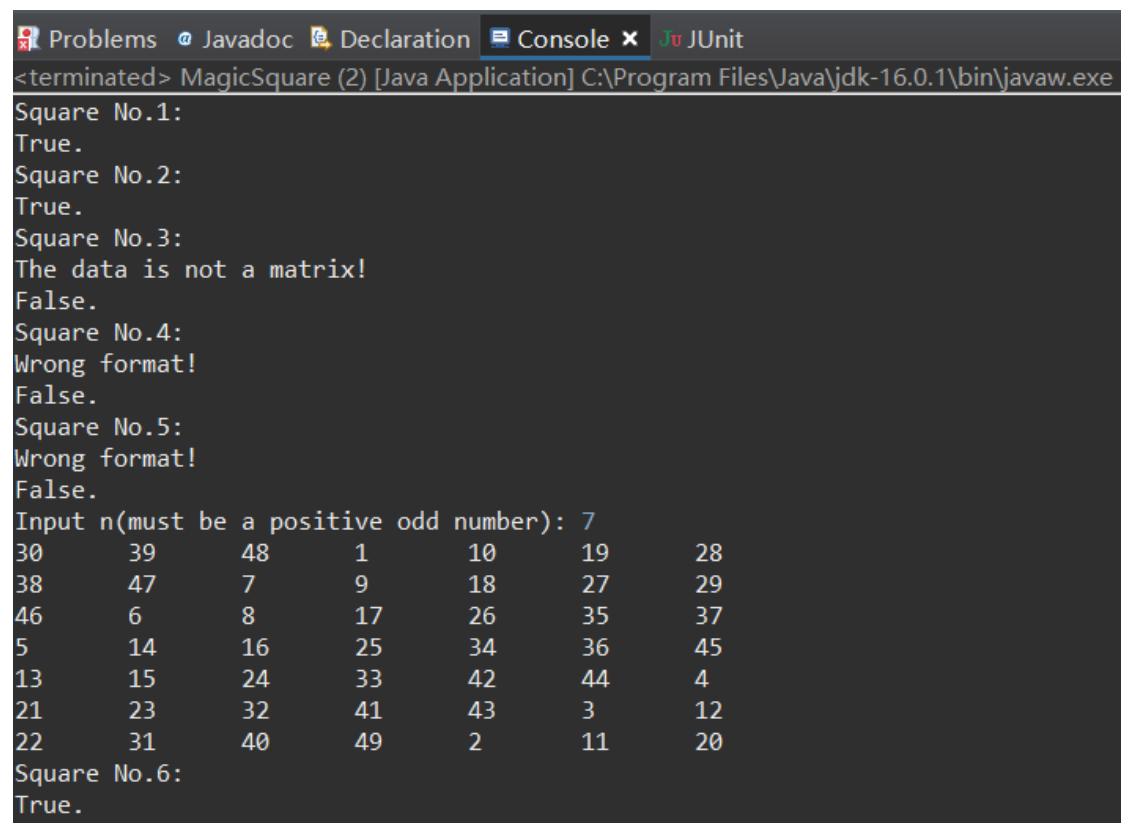
- 6) 用一个双层循环计算每一行和每一列上数字的和, 并将其与上一步所得的对角线和作比较。若不相等, 返回 `false`。待所有比较都结束后, 没有出现不相等的结果, 返回 `true`, 表示二维数组是一个 Magic Square。

```
for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
        row_sum += square[i][j];
        col_sum += square[j][i];
    }

    if (row_sum != sum || col_sum != sum)
        return false;

    row_sum = 0;
    col_sum = 0;
}
```

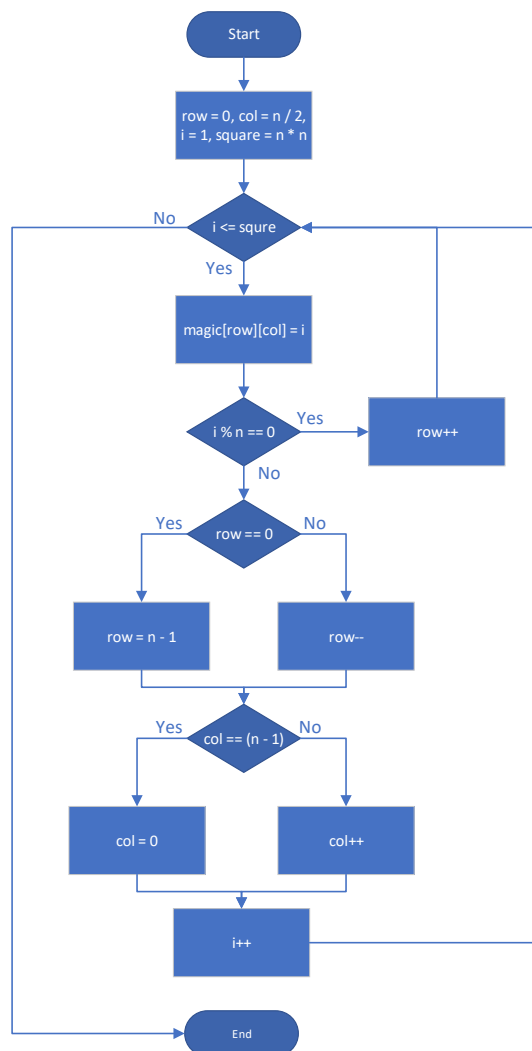
### 3.1.1.3 设计结果



```
<terminated> MagicSquare (2) [Java Application] C:\Program Files\Java\jdk-16.0.1\bin\javaw.exe
Square No.1:
True.
Square No.2:
True.
Square No.3:
The data is not a matrix!
False.
Square No.4:
Wrong format!
False.
Square No.5:
Wrong format!
False.
Input n(must be a positive odd number): 7
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
Square No.6:
True.
```

### 3.1.2 generateMagicSquare()

#### 3.1.2.1 流程图



#### 3.1.2.2 设计思路

调整好函数的接口以后，我们只需在原来的基础上对函数做功能的扩展。扩展的内容有：1) 判断参数  $n$  是否为负数或偶数，若是，则“优雅地”退出、2) 将产生的数据写入 6.txt 文件中。

#### 3.1.2.3 设计过程

首先对参数  $n$  做合法性检查。如果发现  $n$  为负数或者偶数，则打印错误提示信息，并返回 `false`，从而做到“优雅地”退出。

为实现写入文件操作，可在生成操作结束后，按行优先的顺序访问生成的二



维数组，在每两个数字之间输出一个\t，并在数组每行结束后输出换行，便可以得到符合任务要求的 Magic Square 的文本文件。

### 3.1.2.4 设计结果

```
Input n(must be a positive odd number): 7
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
```

## 3.2 Turtle Graphics

该任务首先需要用 git 工具克隆一个远程仓库到本地以便随时 commit 和 push 自己编写的代码。然后我们需要完成绘图工具 Turtle Graphics 的实现包括计算正多边形的内角、计算从一点走向另一点需要转过的角度、给定一个点的集合求其凸包等，并且利用其方法画出正方形以及最终完成一副艺术作品。在实现其方法时，我们还需要使用 JUnit 对代码进行单元测试以验证其正确性。

### 3.2.1 Problem 1: Clone and import

如何从 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。

1. 在文件管理器中将要放置本地仓库的位置处右键，点击 Git Bash Here，打开 git 工具的 shell 界面：



2. 输入以下指令设置自己的用户名和邮箱:  
`git config --global user.name "Yutong Wang"`  
`git config --global user.email 1764027676@qq.com`
3. 访问实验手册中的网址  
[https://classroom.github.com/a/K\\_B5NlIB](https://classroom.github.com/a/K_B5NlIB)  
获取自己本次实验的远程仓库，我的远程仓库地址是：  
<https://github.com/ComputerScienceHIT/HIT-Lab1-1190200527>。
4. 在 Bash 中运行以下指令：  
`Git clone https://github.com/ComputerScienceHIT/HIT-Lab1-1190200527`  
便可以提交代码的远程仓库克隆到本地。
5. 从实验手册提供的网址 <http://web.mit.edu/6.031/www/fa18/psets/ps0/> 中处下载实验所需的 Project 包，解压后将其 import 到 eclipse 中。具体方法为：
  - 1) File → Import... → General → Existing Projects into Workspace
  - 2) 选中刚刚获取的 P2 文件夹，即可将其 import 到 eclipse 中。

### 3.2.2 Problem 3: Turtle graphics and drawSquare

#### 3.2.2.1 设计思路

此任务是让我们了解了 Turtle graphics 绘图工具的基本原理后，利用 forward（前进）和 turn（转体）函数画一个正方形。经过尝试可以发现，turtle 的初始朝向为正上，转体动作的方向为顺时针，角度单位为度。这些方法和信息足够我们

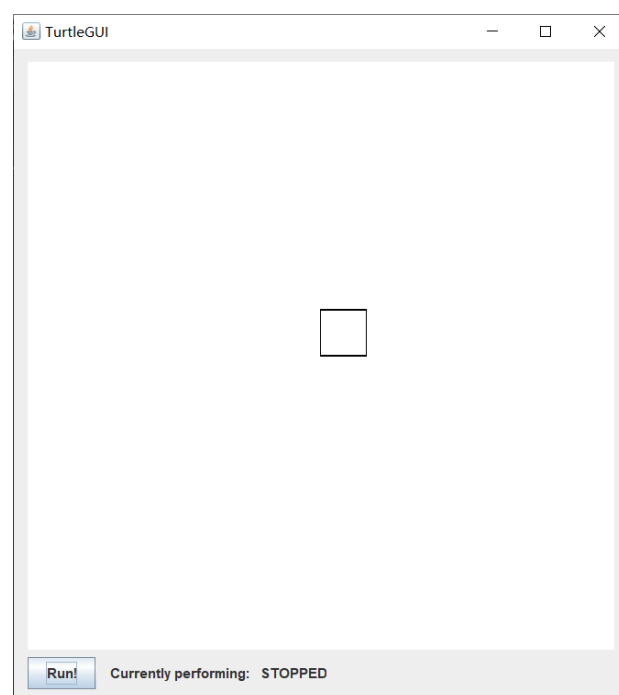
画出一个简单的正方形。

### 3.2.2.2 设计过程

将“前进、转体 90 度”的动作重复四次，即可在界面中画出一个正方形。  
以下为代码实现：

```
public static void drawSquare(Turtle turtle, int sideLength) {  
    //throw new RuntimeException("implement me!");  
  
    for (int i = 0; i < 4; i++) {  
        turtle.forward(sideLength);  
        turtle.turn(90);  
    }  
}
```

### 3.2.2.3 设计结果



## 3.2.3 Problem 5: Drawing polygons

### 3.2.3.1 设计思路

此任务要求实现的方法 `drawRegularPolygon` 是给定边数和边长，在界面中画出一个正多边形。要想仅使用 `forward` 和 `turn` 方法实现这一功能，就必须计算出这个多边形的内角角度，即先实现 `calculateRegularPolygonAngle` 方法。

### 3.2.3.2 设计过程

#### 1. calculateRegularPolygonAngle 的实现

对于正  $n$  边形, 其内角  $\theta$  有以下公式:

$$\theta = (n - 2) \times 180^\circ / n$$

于是实现代码如下:

```
public static double calculateRegularPolygonAngle(int sides) {  
    //throw new RuntimeException("implement me!");  
    return (double)(sides - 2) * 180 / sides;  
}
```


#### 2. drawRegularPolygon 的实现

计算出多边形的内角之后, 就可以仿照画正方形的方法利用循环以及 forward 和 turn 方法画出给定边数和边长的正多边形:

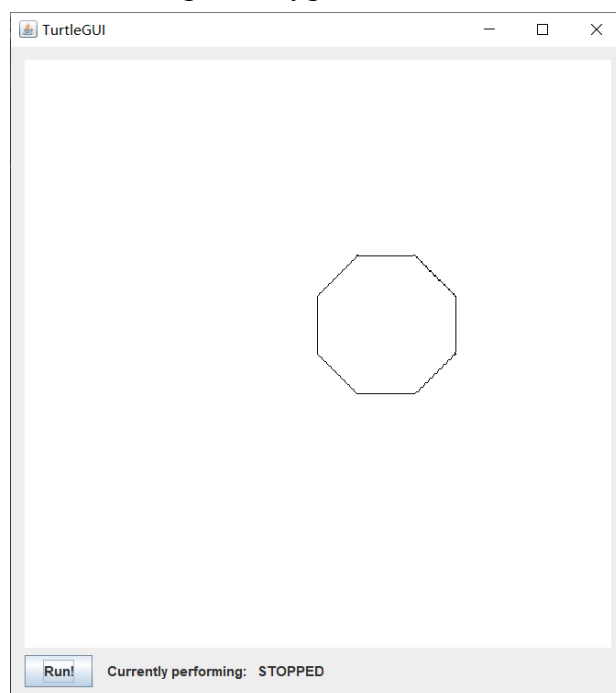
```
public static void drawRegularPolygon(Turtle turtle, int sides, int sideLength) {  
    //throw new RuntimeException("implement me!");  
    for (int i = 0; i < sides; i++) {  
        turtle.forward(sideLength);  
        turtle.turn(180 - calculateRegularPolygonAngle(sides));  
    }  
}
```

### 3.2.3.3 设计结果

#### 1. 使用 Junit 对 calculateRegularPolygonAngle 进行测试的结果:

 calculateRegularPolygonAngleTest (0.000 s)

#### 2. 使用 drawRegularPolygon 画出的图像:



### 3.2.4 Problem 6: Calculating Bearings

#### 3.2.4.1 设计思路

此任务的目标是实现方法 `calculateBearingToPoint`，已知当前朝向、坐标以及目标点的坐标，求如果要到达目标点需要转过角度的度数。然后利用这一方法实现 `calculateBearings`，即给定一系列点的坐标，计算出依次经过这些点时转过角度的序列。经过前面的分析可知，`turtle` 的初始朝向为正上方，即正上方角度为  $0^\circ$ ，且转体方向为顺时针。通过这些规定，利用反三角函数以及一些角度上的变换，就可以实现转体角度的计算。

#### 3.2.4.2 设计过程

1. 首先使用 `Math.atan2` 函数（输入两点纵坐标之差与横坐标之差）计算出两点之间连线倾角的弧度值，然后将其转换成角度值。此时角度值是以 x 轴方向为  $0^\circ$ ，x 轴上方逆时针旋转角度值从  $0^\circ$  变化到  $180^\circ$ ，下方顺时针旋转角度值从  $-0^\circ$  变化到  $-180^\circ$ 。

```
targetBearing = Math.atan2(targetY - currentY, targetX - currentX) / Math.PI * 180.0;
```

2. 将角度值调整至  $0^\circ \sim 360^\circ$ ，只需将负的角度值加  $360^\circ$ ：

```
if (targetBearing < 0)
    targetBearing += 360.0;
```

3. `turtle` 是以 y 轴正方向为  $0^\circ$ ，顺时针旋转角度增大，因此再次对角度做变换，用  $360^\circ$  减去 `targetBearing` 将其改为顺时针旋转，再加上  $90^\circ$  让 x 轴方向的角度值为  $90^\circ$ 。
4. 此时第一象限中的角度值为  $360^\circ \sim 450^\circ$ ，将其减去  $360^\circ$  即可调整为正确的  $0^\circ \sim 90^\circ$ 。

```
targetBearing = 360.0 - targetBearing + 90.0;
if (targetBearing >= 360)
    targetBearing -= 360.0;
```

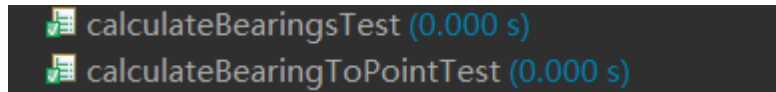
5. 至此已经得知要想从当前点出发走直线到达目标点，在 `turtle` 的坐标系中朝向的角度应为多少。要计算相较于之前需要转体多少度，可以直接将 `targetBearing` 减去 `currentBearing`。需要注意的是，这一角度值的该变量有可能是负值，但 `turtle` 逆时针旋转是不允许的，因此需要加上  $360^\circ$  变成顺时针旋转的角度。

```
double bearing = targetBearing - currentBearing;
if (bearing < 0)
    bearing += 360.0;
return bearing;
```

6. `calculateBearingToPoint` 实现后，在 `calculateBearings` 中只需使用一个循环遍历点列表中每一对相邻的坐标点，使用 `calculateBearingToPoint` 方法依次计算出转体的角度值，并将其存入一个 `List` 即可。

### 3.2.4.3 设计结果

### 使用 JUnit 测试的结果:

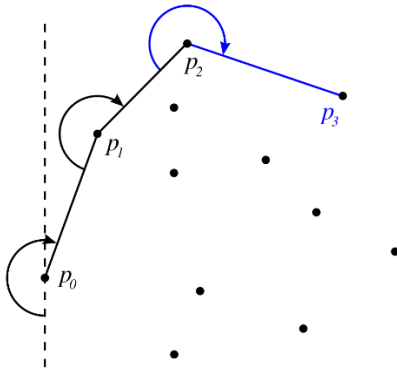


### 3.2.5 Problem 7: Convex Hulls

### 3.2.5.1 设计思路

此任务是求一个给定顶点集的凸包，通过测试用例中给定的例子可以发现一个细节：如果凸包连线上有多个顶点，则凸包顶点集中只包含连线两端的两个顶点，而其他位于连线上的顶点均不位于凸包顶点集内。忽视此细节可能会导致测试不能通过。

为实现 `convexHull` 方法，可以选择较为简单直观的 `gift-wrapping` 算法。该算法的原理是：最左端的顶点一定位于凸包中，将该点设为当前出发点，从该点出发向其余所有顶点作连线。存在某个顶点，除该点和当前凸包点以外的所有顶点均位于连线的同一侧，那么这个点在凸包中。更新当前出发点为该点，重复以上过程，直到回到最初的起始点，算法结束。



### 3.2.5.2 设计过程

为了能够复用之前实现的 `calculateBearingToPoint` 方法，我们将该算法做一些实现细节上的改动。设位于起始点处时，初始的朝向为  $0^\circ$ ，即 y 轴正方向。现遍历除当前点外的所有顶点，利用 `calculateBearingToPoint` 求从当前出发点、以当前朝向移至目标顶点（即遍历的每个点）所需要转体的角度，选取其中转体角度最小的一个目标点加入到凸包集合中，并将当前出发点更新为该顶点。重复以上过程，直至回到初始的凸包点，算法结束，得到顶点集的凸包。

在找起始点时,为了使设计的算法能够正确运行,起始点应为横坐标最小的点。若该横坐标值下有多个顶点,则选取纵坐标最小的那个点。

```
Point startPoint = new Point(Double.MAX_VALUE, Double.MAX_VALUE);
for (Point p : points) {
    if (p.x() < startPoint.x() || (p.x() == startPoint.x() && p.y() < startPoint.y()))
        startPoint = p;
}
```

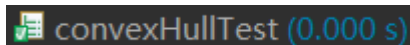
回到思考中提到的细节问题,当最小转体角度方向上有多个顶点时,只取距当前出发点 `pointInHull` 最远的那个顶点。现拟在更新最小转体角度顶点 `tempPoint` 时解决这一问题。原本的更新为当计算出的转体角度 `tempAngle` 小于当前最小转体角度 `minAngle` 时就更新,这样不能保证选到距离最远的那个点。我们可以将更新条件稍作修改。增加一个条件判断:若计算出的转体角度等于当前最小的转体角度(证明此时在一条线上发现多个顶点),则比较此顶点和 `tempPoint` 到 `pointInHull` 的距离,若大于,则更新 `tempPoint` 为此顶点,否则跳过该顶点。这样便可以保证当多个顶点的转体角度相同时,总是选取最远的那个点。

```
tempAngle = calculateBearingToPoint(pAngle, (int)pointInHull.x(), (int)pointInHull.y(), (int)p.x(), (int)p.y());
//System.out.println(pointInHull.x() + ", " + pointInHull.y() + " to " + p.x() + ", " + p.y() + ": " + tempAngle);
if (tempAngle < minAngle || (tempAngle == minAngle && (Math.pow((p.x() - pointInHull.x()), 2) + Math.pow((p.y() - pointInHull.y()), 2))
    > (Math.pow((tempPoint.x() - pointInHull.x()), 2) + Math.pow((tempPoint.y() - pointInHull.y()), 2)))) {
    minAngle = tempAngle;
    pAngle = calculateBearingToPoint(0, (int)pointInHull.x(), (int)pointInHull.y(), (int)p.x(), (int)p.y());
    tempPoint = p;
}
```

当计算完所有顶点的转体角度时, `tempPoint` 即为凸包中的一个点,将这个点加入凸包后更新 `pointInHull` 为这个点,重复上面的过程,直到 `tempPoint` 最终为起始点,证明当前集合中的点已经将其余所有点包围,这个集合就是凸包,算法结束。

### 3.2.5.3 设计结果

使用 JUnit 测试的结果:



convexHullTest (0.000 s)

## 3.2.6 Problem 8: Personal art

### 3.2.6.1 设计思路

最后要完成的任务,是利用以上实现的各种方法创作一幅个人作品。可以利用循环画出重复性的图案,利用 `Drawing polygons` 画出各种正多边形,利用 `turtle.turn` 进行旋转,利用 `turtle.color` 对颜色进行更改。

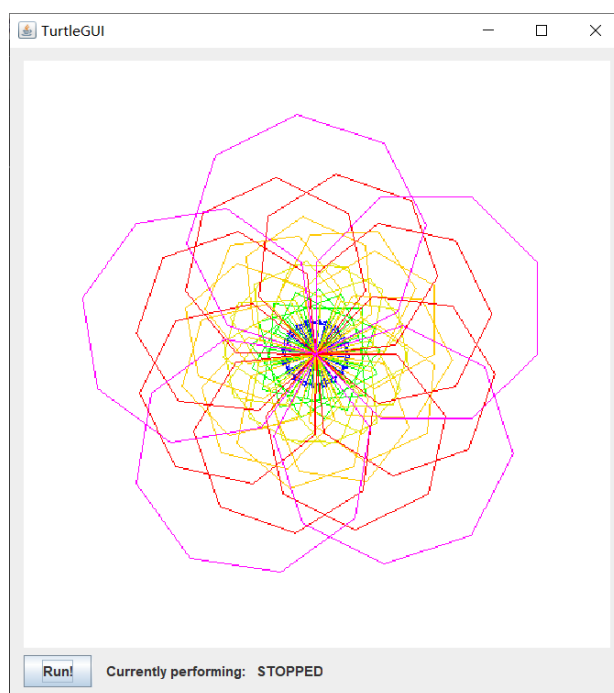
这里拟画出一个围绕一个中心点、以各种多边形为不同的花瓣的彩色花。

### 3.2.6.2 设计过程

使用一个双层循环,外层循环决定多边形的边数、边长和颜色,内层循环决定所画多边形的密度(即每旋转多少角度画出一个多边形):

```
for (int i = 0; i < 6; i++) {  
    switch(i) {  
        case 0:  
            turtle.color(PenColor.BLUE);  
            break;  
        case 1:  
            turtle.color(PenColor.GREEN);  
            break;  
        case 2:  
            turtle.color(PenColor.YELLOW);  
            break;  
        case 3:  
            turtle.color(PenColor.ORANGE);  
            break;  
        case 4:  
            turtle.color(PenColor.RED);  
            break;  
        case 5:  
            turtle.color(PenColor.MAGENTA);  
    }  
    for (int j = 0; j < 20 - 3 * i; j++) {  
        turtle.turn(360.0 / (20 - 3 * i));  
        drawRegularPolygon(turtle, i + 3, (i + 3) * 10);  
    }  
}
```

### 3.2.6.3 设计结果



### 3.2.7 Submitting

如何通过 Git 提交当前版本到 GitHub 上你的 Lab1 仓库。

1. 在本地仓库的文件目录下启动 Git Bash。



2. 使用 `git status` 命令查看文件改动的状态。
3. 确定需要更新的文件后, 对该文件使用 `git add` 命令。
4. 使用 `git commit` 命令进行 `commit` 操作, 在 `vim` 中编辑版本更新的信息, 完成后保存并退出, `commit` 完成。
5. 使用 `git push origin master` 命令即可将改动上传到远程仓库 Lab1。

### 3.3 Social Network

本任务是实现 `Person` 和 `FriendshipGraph` 两个类, 用它们构建一个人际关系的网, 其实质就是一个以 `Person` 对象为顶点构成的有向图。在这个有向图中, 可以添加顶点、添加有向边以及计算两个顶点之间的最短路径长度。

#### 3.3.1 设计/实现 `FriendshipGraph` 类

##### 3.3.1.1 实现思路

要实现有向图的各种功能, 需要定义顶点对象 `Person`。对于 `addVertex` 方法, 在 `FriendshipGraph` 里使用一个 `ArrayList` 存储所有的顶点, 并且使用 `HashSet` 来保存已经出现过的名字, 防止名字重复。增加顶点时只需将 `Person` 加入到 `List` 中即可; 对于 `addEdge` 方法, 存储顶点之间有向边的工作可以交给 `Person` 类来处理, `FriendshipGraph` 只负责调用 `Person` 中的特定方法; 对于 `getDistance` 方法, 由于顶点间的边长均视为 1, 故可以使用 `BFS` 方法计算两个顶点间的最短路径。

##### 3.3.1.2 实现过程

1. 构造函数  
声明一个 `ArrayList<Person>` 类型的对象 `people` 用来保存图里的所有节点 `Person`; 声明一个 `HashSet<String>` 类型的对象 `nameDic` 作为名字的字典, 用来查找名字是否已经在图中存在。
2. `AddVertex`  
首先通过 `nameDic` 查找待加入的 `Person` 的名字是否已经出现过, 若是, 则输出提示信息并返回 -1; 否则将其加入到 `people` 中, 并将其名字添加到 `nameDic` 中, 返回 0。

```
public int addVertex(Person p) {
    if (nameDic.contains(p.getName())) {
        System.out.println("The name already exists!");
        return -1;
    }
    people.add(p);
    nameDic.add(p.getName());
    return 0;
}
```

### 3. addEdge

首先判断待添加关系的两个 Person 对象的名字是否已经出现在 nameDic 中, 若没有, 说明图中没有这个 Person, 输出提示信息并返回-1; 否则调用 Person 的 addFriend 方法 (待实现), 将其中一个 Person 加入到另一个 Person 的朋友列表中, 返回 0。

```
public int addEdge(Person src, Person dst) {
    if (!(nameDic.contains(src.getName()) && nameDic.contains(dst.getName()))) {
        System.out.println("The name does not exist!");
        return -1;
    }
    dst.addFriend(src);
    return 0;
}
```

### 4. getDistance

本方法使用 BFS 方法遍历所有顶点是比较简单高效的方法。首先声明一个队列, 用来做广度优先搜索, 再声明一个 HashMap 对象, 用来保存每个 Person 顶点到源顶点的最短路径长度。先将源顶点入队, 将其最短路径长度设为 0, 然后每次从队中取出一个顶点, 遍历其所有朋友 (可通过调用 Person 类中的方法获得源顶点的朋友数组), 若该 Person 没有访问过 (即 HashMap 中没有该对象的距离信息), 则将该点入队, 然后将其最短路径长度设为刚刚出队的顶点的距离值+1 (因为这个顶点比刚出队顶点到源顶点的距离长一条边) 存入 HashMap 中。重复以上过程, 直到访问到目标顶点, 按上述方法计算出其到源顶点的最短距离并返回。若队列变空仍未访问到目标顶点, 则证明图中不存在这个 Person (这一步也可以用 nameDic 来判断), 返回-1。

```
public int getDistance(Person src, Person dst) {
    if (src == dst)
        return 0;

    Queue<Person> myqueue = new LinkedList<>();
    Map<Person, Integer> distance = new HashMap<>();
    myqueue.offer(src);
    distance.put(src, 0);
    while (!myqueue.isEmpty()) {
        Person cur = myqueue.poll();
        for (Person p : cur.getFriends()) {
            if (!distance.containsKey(p)) {
                distance.put(p, distance.get(cur) + 1);
                myqueue.offer(p);
                if (p == dst)
                    return distance.get(p);
            }
        }
    }
    return -1;
}
```

#### 3.3.1.3 实现结果

见 3.3.3 节及 3.3.4 节。

### 3.3.2 设计/实现 Person 类

#### 3.3.2.1 实现思路

根据 FriendshipGraph 的实现过程, Person 类中应包含以下内容:

1. 属性:
  - 1) 这个人的名字
  - 2) 这个人的朋友列表
2. 方法
  - 1) Person 类的构造函数
  - 2) 向朋友列表中添加朋友的方法
  - 3) 返回自己名字的方法
  - 4) 返回朋友列表的方法

#### 3.3.2.2 实现过程

名字用 String 类型的变量 name 存储即可,朋友列表用 List<Person>类型的变量 friends 存储即可。

```
private String name;  
private List<Person> friends;
```

构造函数的参数应为这个人的名字字符串,在函数内部将该对象的 name 赋值为这个字符串,然后将 friends 实例化为 ArrayList。

```
public Person(String name) {  
    this.name = name;  
    friends = new ArrayList<>();  
}
```

对于 addFriend 方法,如果此人的朋友列表中已经有待添加的人,则无需再次添加,可以输出提示信息后不执行添加操作:

```
public void addFriend(Person p) {  
    if (friends.contains(p))  
        System.out.println("This Edge already exists!");  
    else  
        friends.add(p);  
}
```

其余的方法较为简单,直接展示代码即可:

```
public List<Person> getFriends() {  
    return friends;  
}  
  
public String getName() {  
    return name;  
}
```

### 3.3.2.3 实现结果

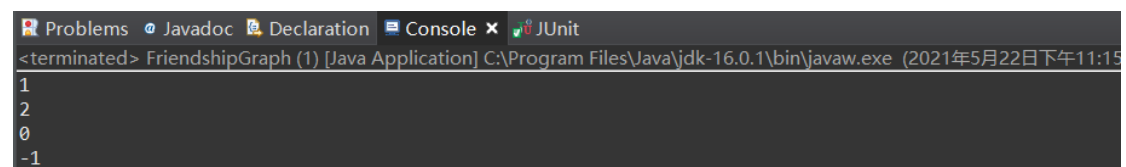
见 3.3.3 节及 3.3.4 节。

### 3.3.3 设计/实现客户端代码 main()

给出你的设计和实现思路/过程/结果。

将实验手册中的用户代码粘贴至 main 函数中，稍加调整后运行：

```
public static void main(String args[]) {
    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));
    // 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
}
```



```
<terminated> FriendshipGraph (1) [Java Application] C:\Program Files\Java\jdk-16.0.1\bin\javaw.exe (2021年5月22日下午11:15)
1
2
0
-1
```

### 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。

首先实例化一个 FriendshipGraph 类的对象 graph。

#### 1. 测试 addVertex 的正确性

分别构造 4 个对象 zhao、qian、sun、li 并使用 addVertex 将其加入到 graph 对象当中，断言这些 Person 对象在 graph 的顶点列表中（该列表可以通过 graph 的方法获得），若断言正确，则可以证明 addVertex 的正确性。

```
Person zhao = new Person("Zhao");
Person qian = new Person("Qian");
Person sun = new Person("Sun");
Person li = new Person("Li");
graph.addVertex(zhao);
graph.addVertex(qian);
graph.addVertex(sun);
graph.addVertex(li);
assertTrue(graph.getPeople().contains(zhao));
assertTrue(graph.getPeople().contains(qian));
assertTrue(graph.getPeople().contains(sun));
assertTrue(graph.getPeople().contains(li));
```

同时, 若一个 Person 对象的名字已经在图中存在了, 根据不重名原则, 该对象不应该被加入到图中, 图中顶点的数量应该变 (仍为 4)。

```
graph.addVertex(zhao);
assertEquals(4, graph.getPeople().size());
```

## 2. 测试 addEdge 的正确性

构造几个 Person 对象, 将其加入到 graph 中后, 再使用 addEdge 方法添加朋友关系。断言添加过的朋友会存在于 Person 对象的朋友列表中 (朋友列表可以通过 Person 当中的方法获得), 若断言正确, 则可证明 addEdge 的正确性。

```
Person zhou = new Person("Zhou");
Person wu = new Person("Wu");
Person zheng = new Person("Zheng");
Person wang = new Person("Wang");
Person nobody = new Person("Nobody"); //Not in the graph.

graph.addVertex(zhou);
graph.addVertex(wu);
graph.addVertex(zheng);
graph.addVertex(wang);
graph.addEdge(zhou, wu);
graph.addEdge(wu, zhou);
graph.addEdge(zhou, zheng);
graph.addEdge(zheng, zhou);
graph.addEdge(zheng, wang);
graph.addEdge(wang, zheng);
assertTrue(zhou.getFriends().contains(wu));
assertTrue(wu.getFriends().contains(zhou));
assertTrue(zhou.getFriends().contains(zheng));
assertTrue(zheng.getFriends().contains(zhou));
assertTrue(zheng.getFriends().contains(wang));
assertTrue(wang.getFriends().contains(zheng));
```

同时, 对于图中不存在的 Person 对象 (名字不在 nameDic 中), 想要将其加入到某个人的朋友列表中, 应该是不被允许的。对于已经添加过的边, 再次添加也应该没有效果。

```
graph.addEdge(zhou, nobody);
assertTrue(!zhou.getFriends().contains(nobody)); //Cannot find Nobody's name in nameDic.

graph.addEdge(zhou, wu);
assertEquals(2, zhou.getFriends().size()); //No use adding an edge twice.
```

## 3. 测试 getDistance 的正确性

构造几个 Person 对象, 将其加入到 graph 中后, 再加入一些朋友关系。断言其中一些顶点之间的最短路径距离为正确的数值, 若断言正确, 即可证明 getDistance 的正确性。

```

Person feng = new Person("Feng");
Person chen = new Person("Chen");
Person chu = new Person("chu");
Person wei = new Person("Wei");
Person jiang = new Person("Jiang");
Person shen = new Person("Shen");
Person han = new Person("Han");
Person yang = new Person("Yang");

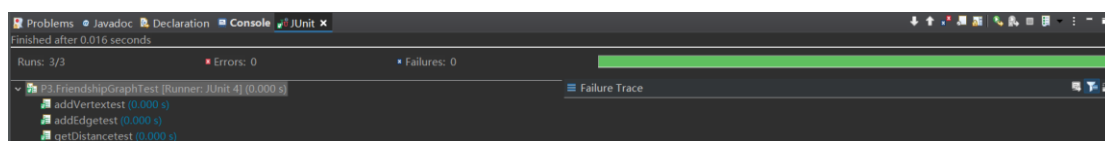
graph.addVertex(feng);
graph.addVertex(chen);
graph.addVertex(chu);
graph.addVertex(wei);
graph.addVertex(jiang);
graph.addVertex(shen);
graph.addVertex(han);
graph.addVertex(yang);
graph.addEdge(feng, chen);
graph.addEdge(chen, feng);
graph.addEdge(chen, jiang);
graph.addEdge(jiang, chen);
graph.addEdge(feng, wei);
graph.addEdge(wei, feng);
graph.addEdge(feng, chu);
graph.addEdge(chu, feng);
graph.addEdge(chu, wei);
graph.addEdge(wei, chu);
graph.addEdge(wei, jiang);
graph.addEdge(jiang, wei);

graph.addEdge(wei, han);
graph.addEdge(han, wei);
graph.addEdge(han, chu);
graph.addEdge(chu, han);

graph.addEdge(han, shen);
graph.addEdge(shen, han);
graph.addEdge(jiang, shen);
graph.addEdge(shen, jiang);
assertEquals(1, graph.getDistance(feng, chen));
assertEquals(2, graph.getDistance(feng, jiang));
assertEquals(0, graph.getDistance(feng, feng));
assertEquals(3, graph.getDistance(chen, han));
assertEquals(2, graph.getDistance(chu, shen));
assertEquals(-1, graph.getDistance(feng, yang));
assertEquals(-1, graph.getDistance(yang, wei));

```

最后运行测试程序，查看测试的结果：



## 4 实验进度记录

日期	时间段	任务	实际完成情况
2021-05-13	14:30-15:30	学习 java 的标准输入输出操作以及文件的读写操作	按计划完成
2021-05-16	19:00-21:00	编写问题 1 的 isLegalMagicSquare 函数、generateMagicSquare 函数并进行测试	延期半小时完成
2021-05-18	18:30-22:30	编写问题 2 的 drawSquare 函数、calculateRegularPolygonAngle 函数、drawRegularPolygon 函数并进行测试	提前半小时完成
2021-05-19	19:00-22:30	编写问题 2 的 calculateBearingToPoint 函数、calculateBearings 函数并完成测试	按计划完成
2021-05-	13: 45-15:	编写问题 2 的 convexHull 函数	遇到困难，未完

20	30		成
2021-05-20	16:00-22:00	编写问题 2 的 <code>convexHull</code> 函数并进行测试 完成问题 2 中的 <code>Personal Art</code> 部分	按计划完成
2021-05-21	18:30-23:00	编写问题 3 的 <code>FriendshipGraph</code> 类和 <code>Person</code> 类	按计划完成
2021-05-22	15:00-17:00	编写问题 3 的测试用例并进行测试	提前半小时完成

## 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
问题 1 中编写 <code>isLegalMagicSquare</code> 函数时,判断 5.txt 中的数据是否为 Magic Square 时报错	每向二维数组中填充一个数时,打印当前的行、列值,找到发生错误的数据位置。再根据二维数组中的位置找到 5.txt 文件中对应的位置,发现此位置有两个数字中间没有用 <code>\t</code> 分隔而是用了三个空格。于是向程序中加入了没有以 <code>\t</code> 分隔时的特殊情况处理。
问题 1 中对于数据不是正整数或没有用 <code>\t</code> 分隔的特殊情况单纯使用 <code>split</code> 无法进行处理	使用正则表达式判断 <code>split</code> 后的字符串是否只由数字组成。上面提到的几种特殊情况都可以靠这一方法解决
问题 2 中实现 <code>convexHull</code> 函数时, Gift Wrapping 算法中判断所有点是否在某一连线的一侧的过程较为复杂	复用之前实现的 <code>calculateBearingToPoint</code> 函数,转为计算到每个点需要转过的角度,将其中转过角度最小的那个点加入到凸包的顶点集中,便可以实现求顶点集的凸包
问题 2 中测试 <code>convexHull</code> 函数时,测试结果总是比正确结果多一个点	画出测试用例中顶点集的坐标图,发现其中点(1,2)和点(1,10)均位于凸包的边界上。而正确结果中只加入了边界线的端点(1,10)而没有加入线上的顶点(1,2)。因此在程序中需要增加一个条件判断,若两个点位于同一个方向上,能够确保永远选择较远的那个点
将 JDK 换为更低版本后 Eclipse 报错	在网上查询资料后得以解决(详见第 2 节实验环境配置)

## 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. 对 java 语言中的基本语法以及面向对象的思想有了初步的了解和掌握。
2. 学会了编写测试用例并使用 JUnit 工具对代码进行测试。
3. 学会了使用 git 工具进行版本管理以及向 GitHub 远程仓库上传文件。

### 6.2 针对以下方面的感受

- (1) Java 编程语言是否对你的口味？

是。Java 语法以及语言风格与 C++相似，比较容易上手。

- (2) 关于 Eclipse IDE；

Eclipse 的配置过程较为麻烦，尤其是在对 JDK 做降级操作时。在仿照手册中的目录要求安排项目文件的层次时也遇到了许多困难。

- (3) 关于 Git 和 GitHub；

使用起来方便快捷，只是偶尔 GitHub 的网络连接质量不佳。

- (4) 关于 CMU 和 MIT 的作业；

英文阅读稍有不便。

- (5) 关于本实验的工作量、难度、deadline；

工作量较大，但难度适中，deadline 比较紧张。

- (6) 关于初接触“软件构造”课程；

感觉突然接触了许多陌生的知识和内容，希望老师的引导能够稍加详细。