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	徐伟嘉
学号	1180300401
班号	1803004
电子邮件	1180300401@stu.hit.edu.cn
手机号码	13136760067

目录

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

JDK、Eclipse、Git

```
C:\Users\32046>java -version
java version "1.8.0_221"
Java(TM) SE Runtime Environment (build 1.8.0_221-b11)
Java HotSpot(TM) 64-Bit Server VM (build 25.221-b11, mixed mode)
```

在这里给出你的 GitHub Lab1 仓库的 URL 地址（Lab1-学号）。

Lab1-1180300401

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

写 java 程序，判断所给矩阵是否为魔幻方针（各行各列以及对角线之和都相等。），以文本形式输入。

在 main()函数中调用五次 isLegalMagicSquare()函数，将 5 个文本 文件名分别作为参数输入进去，看其是否得到正确的输出（true, false）。

需要能够处理输入文件的各种特殊情况，

例如：文件中的数据不符合 Magic Square 的定义（行列数不相等、并非矩阵等）、矩阵中的某些数字 并非正整数、数字之间并非使用\t 分割等。若遇到这些情况，终止程序 执行（isLegalMagicSquare 函数返回 false），并在控制台输出错误 提示信息。

3.1.1 isLegalMagicSquare()

1. 读取 txt 中的数据
2. 用 split 将数据存入二维数组 numb[ ][ ]
   并对其中的数据进行检验，判断是否存在非法输入。

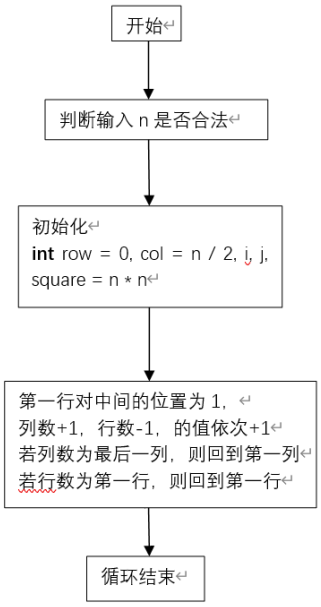
```
String[] line = s.split("\n");

int numb[][] = new int[row][clo];

for (int i = 0; i < row; i++) {
    String[] data = line[i].split("\t");
        for (int j = 0; j < clo; j++) {
            try {
                int num = Integer.valueOf(data[j]).intValue();
                numb[i][j] = num;
            } catch (NumberFormatException e) {
                System.out.print("存在非法输入\n");
                return false;
            }
        }
  }
```

3. 判断 txt 中矩阵行、列、对角线的和是否相等
   最后得到结果：

```
1.txt:It's a magic square.
2.txt:It's a magic square.
3.txt:行列数不等
It's not a magic square.
4.txt:存在非法输入
It's not a magic square.
5.txt:行列数不等
It's not a magic square.
```

3.1.2 generateMagicSquare()

按步骤给出你的设计和实现思路/过程/结果。
函数流程图：



1. 当输入的 n 不合法时，函数输出 false 退出：

```
if(n<=0|n%2==0)//n为奇数或者负数
{
    System.out.println("输入不合法");
    return false;
}
```

2. 将得到的矩阵输入到文件中：

```
try {
  PrintWriter out = new PrintWriter(new BufferedWriter(
            new OutputStreamWriter(
                    new FileOutputStream("src/P1/txt/6.txt")
                    )
            )
    );
  for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++)
            out.write(magic[i][j] + "\t");
            out.write("\n");   }
  out.flush();
  out.close();

} catch (FileNotFoundException e) {
    e.printStackTrace();
}
```

3. 用 isLegalMagicSquare()函数进行判断：

```
6.txt:It's a magic square.
```

3.2 Turtle Graphics

利用添加到徽标语言中的 turtle 图形，向屏幕上的“乌龟”发送一系列指令，这只“乌龟”会移动。

3.2.1 Problem 1: Clone and import

登录网址下载任务文件

3.2.2 Problem 3: Turtle graphics and drawSquare

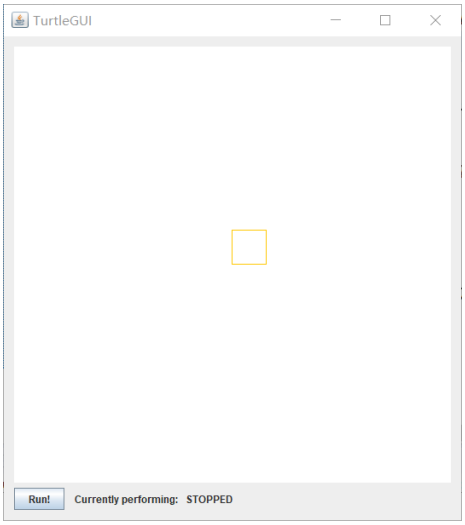
重复以下步骤四次：

1. 向前走（forword）变长
2. 转 90 度

```
for(int i=0;i<4;i++)
{
    turtle.forward(sideLength);
    turtle.turn(90);
}
```

（可以用 color 函数变换颜色）

得到：



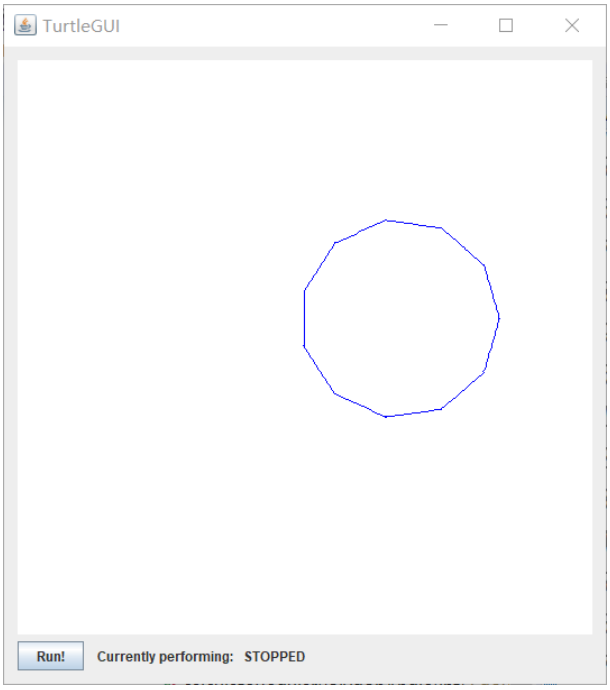
3.2.3 Problem 5: Drawing polygons

1. 通过边数（sides）算角度：calculateRegularPolygonAngle(int sides)
   角度等于=（sides-2）*180/sides
   （注意转换类型）
2. 通过角度（angle）算变数：calculatePolygonSidesFromAngle(double angle)
   判断异常；
   边数=（（2*angle）/（180-angle））+ 2
   （要注意四舍五入）
3. 画多边形：
   步骤和画正方形相似

```
    if(sides<=2)
        System.out.println("输入边数不合法");
    double turnangle=180-calculateRegularPolygonAngle(sides);
    try{
        for(int i=0;i<sides;i++)
    {
        turtle.forward(sideLength);
        turtle.turn(turnangle);
    }
}catch(Exception e) {
    throw new RuntimeException("implement me!");
}
```

（可以用 color 函数变换颜色）

用 *drawRegularPolygon*(turtle,11,50)进行测试
得到结果：



3.2.4 Problem 6: Calculating Bearings

1. calculateBearingToPoint（当前点到目标点角度偏移量）：
   通过 arctan 函数算出两点之间的角度（需注意分母为 0 的情况）
   再减去当前方向和 y 轴正方向的夹角 ，即为需要偏移的夹角（需注意最后角度的正负性）

```
angle=90-(180/Math.PI)*Math.atan2((targetY-currentY),(targetX-c

if(angle<0)
    angle=angle+360;
```

2. calculateBearings（）
   创建一个 List 用于储存角度
   循环调用 calculateBearingToPoint 函数（注意初始角度的叠加）

```
for(int i=1;i<xCoords.size();i++){
System.out.println(bearing);
angles.add(calculateBearingToPoint(bearing,xCoords.get(i-1),yCoords.ge
bearing+=angles.get(i-1);
bearing=bearing%360.0;
```

3.2.5 Problem 7: Convex Hulls

凸包问题：

1. 小于三点时直接返回

```
if(points.size() <=3)
    return points;
```

2. 找到一个边界点（左下或右上）

```
for(Point point : points)
{
    if(minPoint == null){
        minPoint = point;
        continue;
    }
    if(minPoint.x() < point.x())
        minPoint = point;
    else if(minPoint.x() == point.x())
    {
        if(point.y() > minPoint.y())
        minPoint = point;
    }
}

shellPoint.add(minPoint); //加入集合
```

3. 从第一个点出发依次计算和其他点的旋转角，找旋转角最小的那个
4. 如果有两个点旋转角相等，则选距离更远的那个

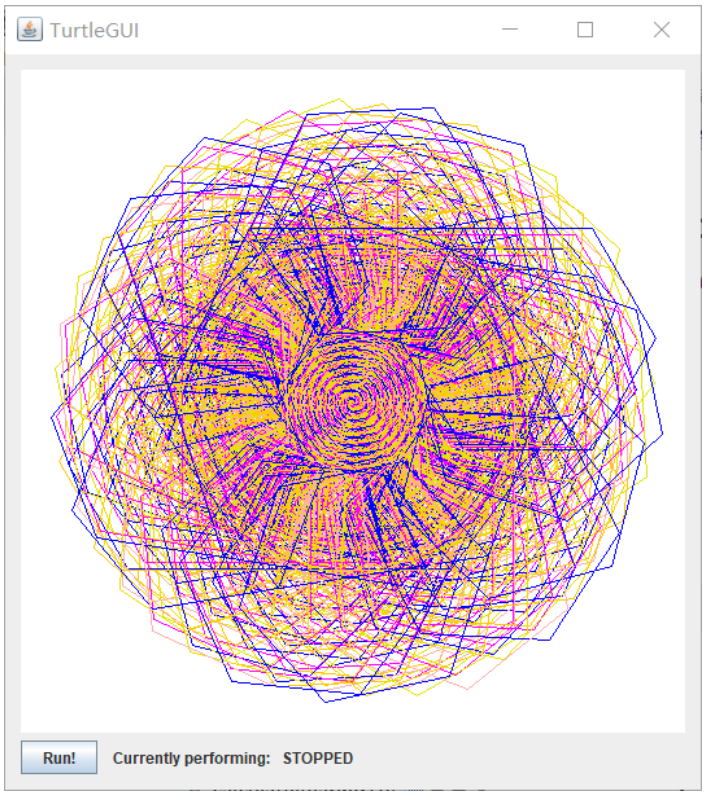
```
if(nextBearing == nowBearing){
    if(nextLength < (Math.pow(point.x()-nowPoint.x(), 2)+Math.pow(point.y(
    {
        nextLength = Math.pow(point.x()-nowPoint.x(), 2)+Math.pow(point.y(
        nextPoint = point;
    }
}//倾角一样取长度最大
else if(nextBearing > nowBearing) {
    nextLength = Math.pow(point.x()-nowPoint.x(), 2)+Math.pow(point.y()-no
    nextBearing = nowBearing;
    nextPoint = point;
}
```

5. 直到回到第一个点

```
if(minPoint.equals(nowPoint))//回到第一个点
{
    break;
}
```

3.2.6 Problem 8: Personal art

循环 1000 次，每次往前走 i/6，画一个六边形，转 63 度；
再结合颜色变换得到如下图形：



3.2.7 Submitting

3.3 Social Network

拟画社交网络图

3.3.1 设计/实现 FriendshipGraph 类

3.3.1.1 设置成员变量

```
public ArrayList<Person> person ;  //保存人名
public int edge[][];//邻接矩阵
private HashMap<Person,Integer> relation;
boolean X[] ;//最短路径标记
```

3.3.1.2 函数 Addvertex

扫描 person 里是否有需加入点的同名，有输出错误，没有在矩阵中加入顶点。

```
public void  addVertex(Person name) {//加入顶点
    for(Person P:person)
    {
        if(P.Getname() == name.Getname())//如果名字重复，则输出异常
        {
            System.err.println("There are same names!");
            System.exit(0);
        }
    }
    person.add(name);//没有相同的名字则加入person
    relation.put(name, person.indexOf(name));
}
```

3.3.1.3 函数 addEdge

获得所添加名字在矩阵中的编号，修改 Edge 为 1;

```
public void addEdge(Person name1, Person name2) {
    int a,b;
    a = relation.get(name1);
    b = relation.get(name2);
    edge[a][b] = 1;
}
```

3.3.1.4 函数 getDistance

使用 Dijkstra 算法实现求两点间的最短路径

```
for(int i=0; i<n-1; i++)//循环n-1次，每次都能加入一个点，最后每个点都已经加入
{
    w = 0;
    temp = 100;
    for(int j=0; j<n; j++)
    {
        if(!X[j]&&cost[j]<temp)
        {
            temp = cost[j];
            w = j;//
        }
    }
    X[w] = true;//寻找一个当前最小的cost[j]，体现了贪心算法的思想
    for(int v=0; v<n ;v++)
    {
        if(X[v]!=true)
        {
            sum = cost[w] + edge[w][v];
            if(sum<cost[v])
            {
                cost[v] = sum;
            }
        }
```

3.3.2 设计/实现 Person 类

包含名字的字符串，构造 getname 函数：

```
public class Person {
    private String nameString;

    public Person(String namString)
    {
        this.nameString = namString;
    }
    public String Getname()
    {
        return this.nameString;
    }
}
```

3.3.3 设计/实现客户端代码 main()

3.3.3.1 所给 main 函数，若不修改得到结果为：

```
<terminated> FriendshipGraph [Java Application] I
1
2
0
-1
```

3.3.3.2

如果将上述代码的第 10 行注释掉（意即 rachel 和 ross 之间只存在单向的 社交关系 ross->rachel），结果为：

```
<terminated> FriendshipGraph [Java Application] D:\java\bin\javaw.exe (2020年3月14日 下午11:26:21)
-1
-1
0
-1
```

3.3.3.3

如果将第 3 行引号中的“Ross”替换为“Rachel”，结果为：

```
Problems  Javadoc  Declaration  Console
<terminated> FriendshipGraph [Java Application] D:\java\bin\javaw.exe (2020年3月14日 下午11:29:35)
There are same names!
```

3.3.4 设计/实现测试用例

3.3.4.1 addVerTexTest

运用函数，检测 person 里是否包含：

```
@Test
public void addVertexTest() {//加顶点测试
        FriendshipGraph  graph= new FriendshipGraph();
        Person A = new Person("A");
        Person B = new Person("B");
        Person C = new Person("C");
        Person D = new Person("D");
        graph.addVertex(A);
        graph.addVertex(B);
        graph.addVertex(C);
        graph.addVertex(D);
        assertTrue(graph.person.contains(A));//判断是否加入
        assertTrue(graph.person.contains(B));
        assertTrue(graph.person.contains(C));
        assertTrue(graph.person.contains(D));
    }
```

3.3.4.2 addEdgeTest

运用函数，判断邻接矩阵是否改变：

```
@Test
public void addEdgeTest() //加边测试
{
    FriendshipGraph  graph = new FriendshipGraph();
    Person A = new Person("A");
    Person B = new Person("B");
    Person C = new Person("C");
    Person D = new Person("D");
    graph.addVertex(A);
    graph.addVertex(B);
    graph.addVertex(C);
    graph.addVertex(D);
    graph.addEdge(A, B);
    graph.addEdge(B, A);
    graph.addEdge(B, C);
    graph.addEdge(C, B);
    graph.addEdge(D, B);
    graph.addEdge(B, D);
    graph.addEdge(A, C);
    graph.addEdge(C, A);
    assertEquals(graph.edge[0][1],1);
    assertEquals(graph.edge[1][0],1);
    assertEquals(graph.edge[1][2],1);
    assertEquals(graph.edge[2][1],1);
```

3.3.4.3 getDistanceTest

运用前两个函数构造关系表，检测给出最短路径长度是否正确：

```
public void getDistanceTest()
{
    FriendshipGraph  graph = new FriendshipGraph();
    Person A = new Person("A");
    Person B = new Person("B");
    Person C = new Person("C");
    Person D = new Person("D");
    graph.addVertex(A);
    graph.addVertex(B);
    graph.addVertex(C);
    graph.addVertex(D);
    graph.addEdge(A, B);
    graph.addEdge(B, A);
    graph.addEdge(B, C);
    graph.addEdge(C, B);
    graph.addEdge(D, B);
    graph.addEdge(B, D);
    graph.addEdge(A, C);
    graph.addEdge(C, A);
    assertEquals("except distance",1,graph.getDistance(A, C));
    assertEquals("except distance",1,graph.getDistance(B, D));
    assertEquals("except distance",1,graph.getDistance(B, C));
    assertEquals("except distance",1,graph.getDistance(A, B));
```

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。

每次结束编程时，请向该表格中增加一行。不要事后胡乱填写。

不要嫌烦，该表格可帮助你汇总你在每个任务上付出的时间和精力，发现自己不擅长的任务，后续有意识的弥补。

日期	时间段	任务	实际完成情况
2020-03-02	13:00-19:30	编写问题 1 的 isLegalMagicSquare 函数并进行测试	遇到困难，未完成
2020-03-03	15:30-17:30	编写问题 1 的 isLegalMagicSquare 函数并进行测试	遇到困难，未完成
2020-03-04	15:30-19:30	编写问题 1 的 isLegalMagicSquare 函数并进行测试	按计划完成
2020-03-05	18:00-19:00	编写问题 1 的 generateMagucSquare() 函数并进行测试	按计划完成
2020-03-06	18:00-19:00	编写问题 2 的 problem1,3	按计划完成
2020-03-09	18:00-19:00	编写问题 2 的 problem5,6	按计划完成
2020-03-10	18:30-23:30	编写问题 2 的 problem7,8	遇到困难未完成
2020-03-11	8:00-9:00	编写问题 2 的 problem7,8	按计划完成
2020-03-12	18：30-21:00	编写问题 3	遇到困难未完成
2020-03-14	20:00-23:30	编写问题 3	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
编写问题一时，由于初次接触 java 语言，在读取文件操作中卡住	上网查解决方法
编写问题二时，对于 Set 和 List 的转换不太懂，Set 的具体语法也不太掌握	上网学习相关方法

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

6.2 针对以下方面的感受

(1) Java 编程语言是否对你的口味？

感觉很多函数熟悉了之后用起来还是很方便的（list，set，map）。

(2) 关于 Eclipse IDE

界面简单，好操作。

(3) 关于 Git 和 GitHub

解决问题，提交作业都很方便。

(4) 关于 CMU 和 MIT 的作业

简洁明了，对于知识点的运用都很紧凑，对于学 java 的新手来说收获很大。

(5) 关于本实验的工作量、难度、deadline

很合理，时间刚好。

(6) 关于初接触“软件构造”课程

对于深入理解软件很有帮助。