实习二

实习概括：

这次实验就是学着用turtle画一些图案，以及对图案进行一些角度计算。提供了两个函数，forward(units)和turn(degrees) ，forward(units)是指前进多少长度，turn(degrees)是指顺时针转多少度。

该任务需要我们Import已有的程序后，利用turtle按照要求画图，其中需要利用几何知识设计一些函数简化编程，最后可以发挥想象力进行Personal Art。首先分析turtle的package组成，了解类成员。

Problem 1: Import

（1）

在本地创建git仓库：

打开工作目录，右键，在当前目录打开Git Bash；

输入git init 指令，将当前目录创建为本地仓库；

当前目录多了一个隐藏文件夹.git，证明本地仓库已创建好了。

使用git管理本地开发：

和github仓库建立远程链接：访问github上的远程仓库并复制URL：

在本地仓库打开Git Bash，输入git remote add origin “URL”指令，建立远程链接；

输入git clone “URL”指令将远程仓库的内容下载到本地仓库；

将完成的项目复制到本地仓库中，输入git add .将所有文件全部加入本地仓库；

输入 git commit -m “备注” 指令，将文件暂存到本地目录并且添加备注说明；

输入 git push -u origin master指令，将本地仓库同步到远程仓库。

Problem 2: Warm up with may­Use­Code­In­Assignment

public static boolean mayUseCodeInAssignment(boolean writtenByYourself,

boolean availableToOthers, boolean writtenAsCourseWork,

boolean citingYourSource, boolean implementationRequired) {

// TODO: Fill in this method, then remove the exception

if(writtenByYourself && !(availableToOthers) && writtenAsCourseWork && citingYourSource && implementationRequired)

return ture;

else return false;

}

Problem 3: 为第一次试验添加单元测试

使用JUnit进行自动化的单元测试，运行单元测试。

1. 测试RulesOf6005Test，在RulesOf6005Test.java Package Explorer，Project View中右键单击该文件，然后转到Run As选项。单击JUnit Test，看到JUnit视图出现，并且看到绿色的条，表明mayUseCodeInAssignment正确实现，所有测试通过。
2. 尝试破坏我的实现并RulesOf6005Test再次运行。在JUnit视图中看到一个红色的条，单击test­May­Use­Code­In­Assignment，在底部的框中看到一个堆栈跟踪，简要说明了出了什么问题。双击故障堆栈跟踪中的一行在跟踪中显示该代码。该堆栈跟踪还包含Java库或JUnit本身的行。
3. 修复实现，使其再次通过测试。

Problem 4: drawSquare

该Problem要求使用现有的接口（forward和turn）实现drawSquare方法，要求该方法调用时，根据给定的变长画出一个正方形。使用一个简单的for循环即可解决。

/\*\*

\* Draw a square.

\*

\* @param turtle the turtle context

\* @param sideLength length of each side

\*/

public static void drawSquare(Turtle turtle, int sideLength) {

for (int i = 0; i < 4; i++) {

turtle.forward(sideLength);

turtle.turn(90);

}

}

Problem 5—10: Polygons and headings

Problem 5: Drawing polygons

该Problem需要实现两个方法：calculateRegularPolygonAngle()和drawRegularPolygon()方法，并使用Junit测试。calculateRegularPolygonAngle()方法要求根据给定的边数作为参数，计算出对应的正多边形的内角。只需要使用公式insideAngle=((sides-2)×180)/sides即可，由于insideAngle为double类型，而参与计算的数字都是int整型，如果直接计算可能会发生截断，需要写成2d或180d即可转为浮点运算。drawRegularPolygon()方法要求以边长与边数作为参数，控制turtle画出正多边形。使用循环即可，计算角度时可调用calculateRegularPolygonAngle()方法计算，唯一要注意的就是turtle旋转的角是外角，而不是计算出来的内角。

/\*\*

\* Determine inside angles of a regular polygon.

\*

\* There is a simple formula for calculating the inside angles of a polygon; you

\* should derive it and use it here.

\*

\* @param sides number of sides, where sides must be > 2

\* @return angle in degrees, where 0 <= angle < 360

\*/

public static double calculateRegularPolygonAngle(int sides) {

return (sides - 2) \* 180.0 / sides;

}

/\*\*

\* Given the number of sides, draw a regular polygon.

\*

\* (0,0) is the lower-left corner of the polygon; use only right-hand turns to

\* draw.

\*

\* @param turtle the turtle context

\* @param sides number of sides of the polygon to draw

\* @param sideLength length of each side

\*/

public static void drawRegularPolygon(Turtle turtle, int sides, int sideLength) {

double angle = 180 - calculateRegularPolygonAngle(sides);

for (int i = 0; i < sides; i++) {

turtle.forward(sideLength);

turtle.turn(angle);

}

}

Calculating headings

该Problem需要实现两个方法：calculateBearingToPoint()和calculateBearings()方法，并进行Junit测试。calculateBearingToPoint()方法要求计算在当前点、当前朝向的情况下，顺时针转向目标点所需要转动的角度。给定的参数有当前朝向的度数、当前点的X与Y值和目标点的X与Y值，当前朝向的度数以向上方为0度。基本是个数学问题，比较繁琐，分多种情况讨论即可，需要用到反正切函数Math.atan2方法。calculateBearings()方法的参数为两个List，第一个List中为X的值，第二个为Y的值，需要返回一个List，包含所有需要转动的角度。只需要遍历两个集合，调用calculateBearingToPoint()方法即可，可设置一个变量里路当前朝向的角度。

/\*\*

\* Given the current direction, current location, and a target location,

\* calculate the Bearing towards the target point.

\*

\* The return value is the angle input to turn() that would point the turtle in

\* the direction of the target point (targetX,targetY), given that the turtle is

\* already at the point (currentX,currentY) and is facing at angle

\* currentBearing. The angle must be expressed in degrees, where 0 <= angle <

\* 360.

\*

\* HINT: look at http://en.wikipedia.org/wiki/Atan2 and Java's math libraries

\*

\* @param currentBearing current direction as clockwise from north

\* @param currentX current location x-coordinate

\* @param currentY current location y-coordinate

\* @param targetX target point x-coordinate

\* @param targetY target point y-coordinate

\* @return adjustment to Bearing (right turn amount) to get to target point,

\* must be 0 <= angle < 360

\*/

public static double calculateBearingToPoint(double currentBearing, int currentX, int currentY, int targetX,

int targetY) {

double angle = Math.atan2(targetX - currentX, targetY - currentY) \* 180 / Math.PI - currentBearing;

if (angle < 0)

angle += 360;

return angle;

}

/\*\*

\* Given a sequence of points, calculate the Bearing adjustments needed to get

\* from each point to the next.

\*

\* Assumes that the turtle starts at the first point given, facing up (i.e. 0

\* degrees). For each subsequent point, assumes that the turtle is still facing

\* in the direction it was facing when it moved to the previous point. You

\* should use calculateBearingToPoint() to implement this function.

\*

\* @param xCoords list of x-coordinates (must be same length as yCoords)

\* @param yCoords list of y-coordinates (must be same length as xCoords)

\* @return list of Bearing adjustments between points, of size 0 if (# of

\* points) == 0, otherwise of size (# of points) - 1

\*/

public static List<Double> calculateBearings(List<Integer> xCoords, List<Integer> yCoords) {

List<Double> list = new ArrayList<>();

int N = xCoords.size();

double face = 0;

int i = 0;

while (i < N - 1) {

double k = calculateBearingToPoint(face, xCoords.get(i), yCoords.get(i), xCoords.get(i + 1),

yCoords.get(i + 1));

list.add(k);

face = (face + k) % 360;

i++;

}

System.out.println(list);

return list;

}

Problem 11: Personal art

在画正多边形的基础上，步长不是一直相同，而是越来越长，并且角度比画正多边形需要的角度多一些，每次拐弯变换颜色。

Size是螺旋的大小

Step的每一步的长度，每走一步拐弯一次

Densi是密度，角度越小，螺旋越密

ColorNum是色彩的数量，更改时要在switch里更改

/\*\*

\* Draw your personal, custom art.

\*

\* Many interesting images can be drawn using the simple implementation of a

\* turtle. For this function, draw something interesting; the complexity can be

\* as little or as much as you want.

\*

\* @param turtle the turtle context

\*/

public static void drawPersonalArt(Turtle turtle) {

int Size = 400, Step = 1, Densi = 1, ColorNum = 5;

for (int i = 1; i <= Size; i++) {

switch (i % ColorNum) {

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.RED);

break;

case 4:

turtle.color(PenColor.MAGENTA);

break;

case 5:

turtle.color(PenColor.ORANGE);

break;

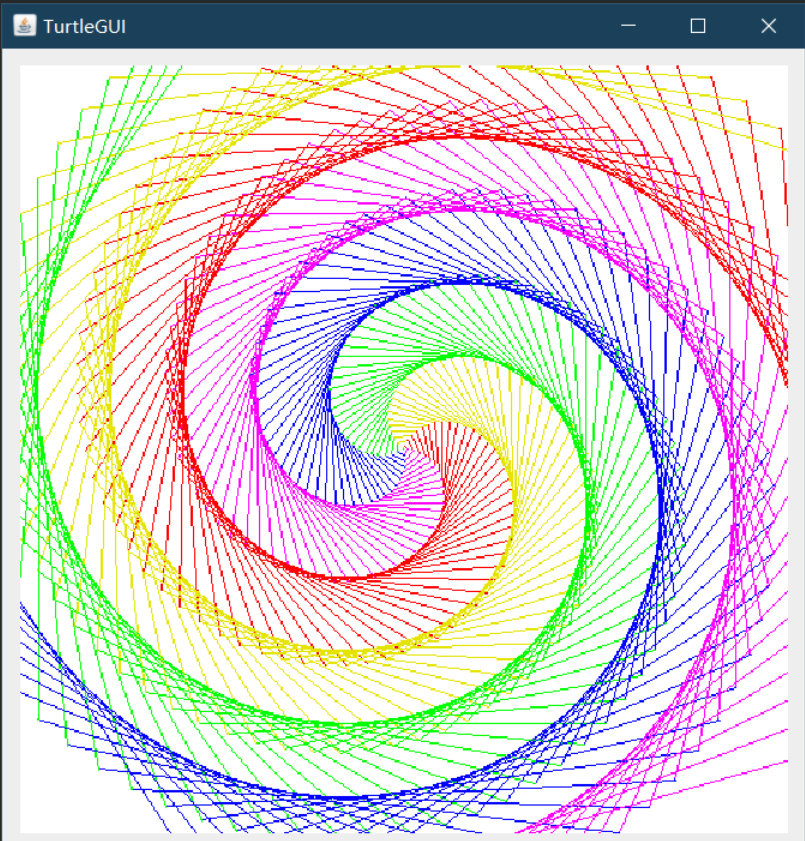
}

turtle.forward(Step \* i);

turtle.turn(360/ColorNum + Densi);

}

}



实习二总结

这个任务中需要注意的内容就是如何用库函数求解偏转角度以及计算结果的精度舍入问题。在Java的Math库中，提供了这样的一个方法：atan2方法。该方法通过传入一个向量，即可计算出其对应的角的大小。这个方法的返回值，实际上是直角坐标系y轴正方向和向量所成角的大小，而且返回值为对应角的弧度值。因而要对这个角有一个正确的认识，并将其单位进行一个简单的转换。