## 实习二

### Part I

#### Problem 1: Import

运行环境：idea2021专业版；

首先在idea中创建一个新工程，然后将rules目录下的两个代码copy过去，然后，修改一小导入包的路径，在外部依赖包中导入Junit4依赖包。然后则无报错；

#### Problem 2: Warm up with mayUseCodeInAssignment

根据说明文档中的要求，是要我们完成mayUseCodeInAssignment（）函数的编写，然后运行测试案例，并且成功通过测试案例；

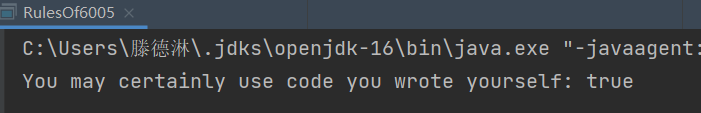
根据该函数的注释说明，可以大概的分析出该函数的功能，是一个判断代码是否可以使用的函数，类似于GitHub中的push功能（这是我的理解）；

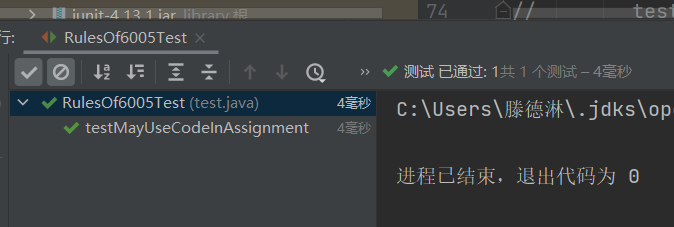
读懂题意之后可以很轻松的想到用分支来实现判断功能：

1）函数编写：

1. public static boolean mayUseCodeInAssignment(boolean writtenByYourself,
2. boolean availableToOthers, boolean writtenAsCourseWork,
3. boolean citingYourSource, boolean implementationRequired) {
4. */\*\**
5. \* 判断是否是自己写的代码,如果是自己写的则返回true或者
6. \* 是允许其他人写的和是表明了引用的
7. \* 或者是没有在课上写过的，没有布置任务的
8. \* 都返回true，其余返回false；
9. \*/
10. if (writtenByYourself || (!writtenAsCourseWork && !implementationRequired && availableToOthers && citingYourSource)) {
11. return true;
12. } else {
13. return false;
14. }
15. }

2）运行单元测试：



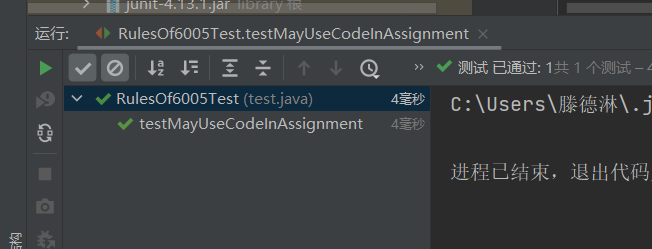


#### Problem 3: 为第一次试验添加单元测试

添加3个测试用例：

1. assertFalse("Expected false:  un-cited publicly-available code",
2. RulesOf6005.mayUseCodeInAssignment(false, false, true, true, true));
3. assertTrue("Expected true: self-written required code",
4. RulesOf6005.mayUseCodeInAssignment(true, false, false, true, true));
5. assertTrue("Expected true: self-written required code",
6. RulesOf6005.mayUseCodeInAssignment(true, false, true, false, true));

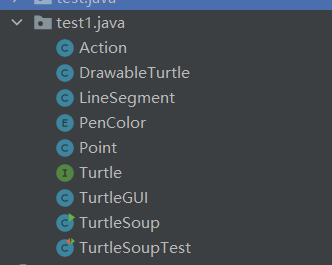
通过测试：



### Part II

#### Problem 4: drawSquare

首先我们需要完成drawSquare(Turtle turtle, int sideLength)并且需要使用介绍的forward和turn方法实现。最终效果是画一个正方形。



其中的**Turtle.java**接口即定义了我们等会需要使用的

· forward(units)

· turn(degrees)

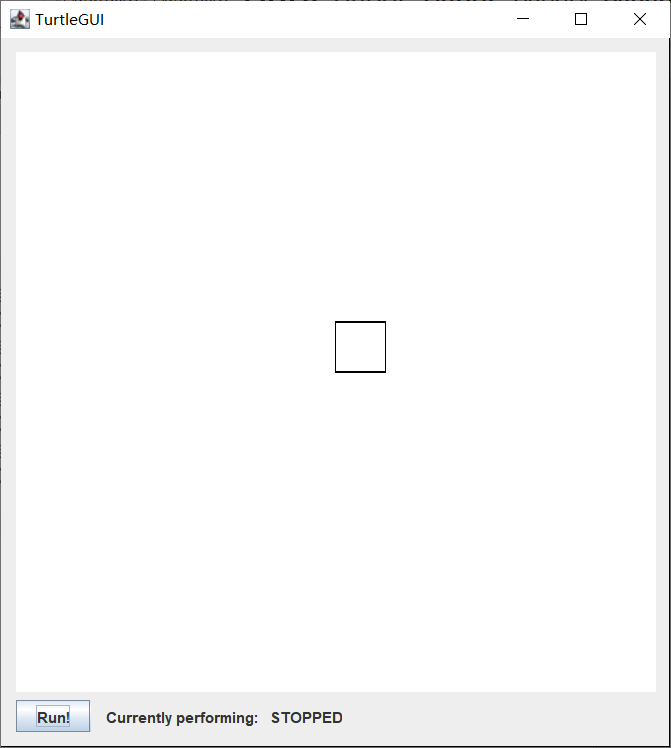
这两个功能，然后**DrawableTurtle.java**实现了这个接口；

所以，我们只需要调用这两个函数进行长度，角度的变化就可以画出一个正方形，边长作为一个参数传入，角度每一次转变90度；

代码如下：

1. public static void drawSquare(Turtle turtle, int sideLength) {
2. turtle.forward(sideLength);
3. turtle.turn(90);
4. turtle.forward(sideLength);
5. turtle.turn(90);
6. turtle.forward(sideLength);
7. turtle.turn(90);
8. turtle.forward(sideLength);
9. turtle.turn(90);
10. *//throw new RuntimeException("implement me!");*
11. }

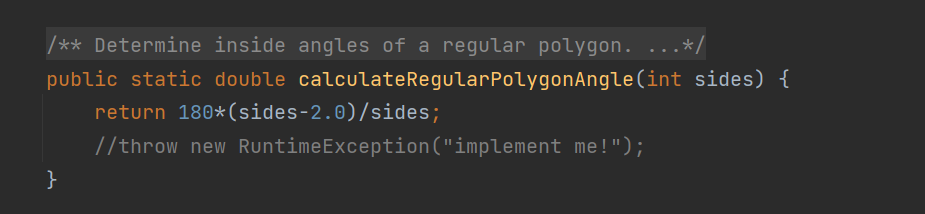
运行结果：



#### Problems 5—10: Polygons and headings

1. 完善calculateRegularPolygonAngle(int sides)函数：

求正多边形内角，我们可以很容易查出正多边形内角和公式为180\*（n-2），故每个内角为180 \*（n-2）/n：

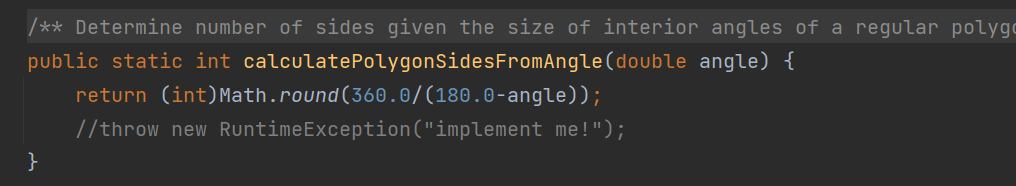


运行测试函数calculateRegularPolygonAngleTest()：

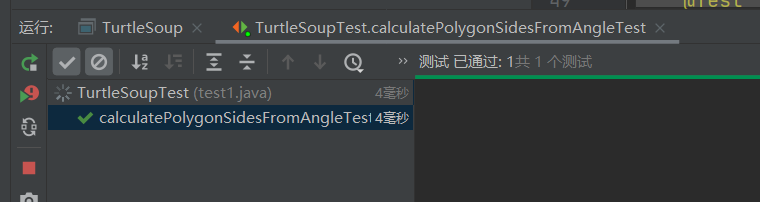


1. 完善calculatePolygonSidesFromAngle(double angle)函数：

这个函数就是前面的倒推函数，用角度和求边数

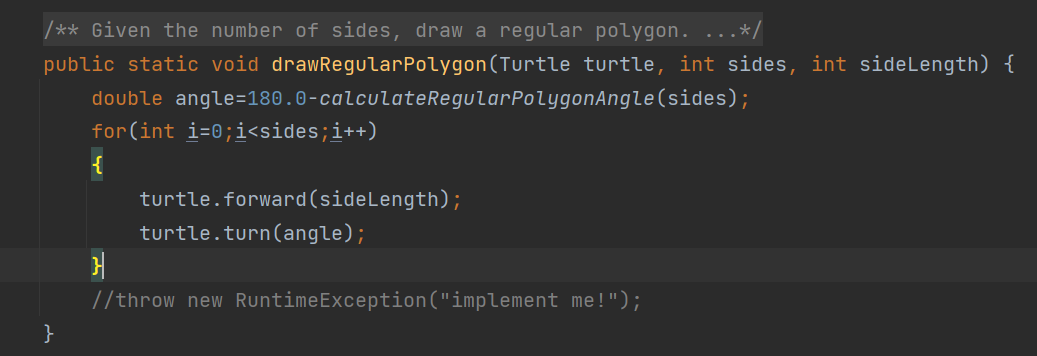


运行测试函数calculatePolygonSidesFromAngleTest()

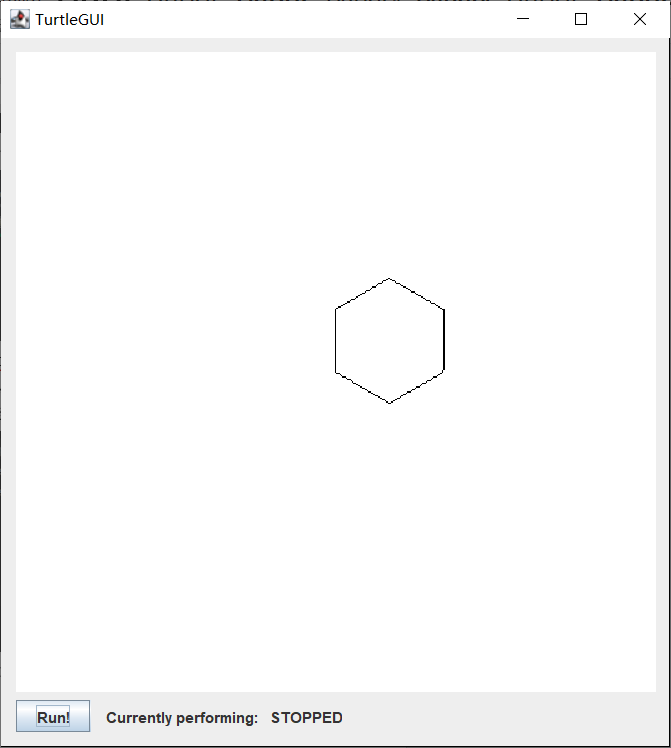


1. 完善drawRegularPolygon(Turtle turtle, int sides, int sideLength)函数：

该函数的功能是画一个长度为x的正n边形，我们可以使用前面完成的函数进行辅助完成本函数功能；前面可以求取其内角度，从而达到正多边形的要求；

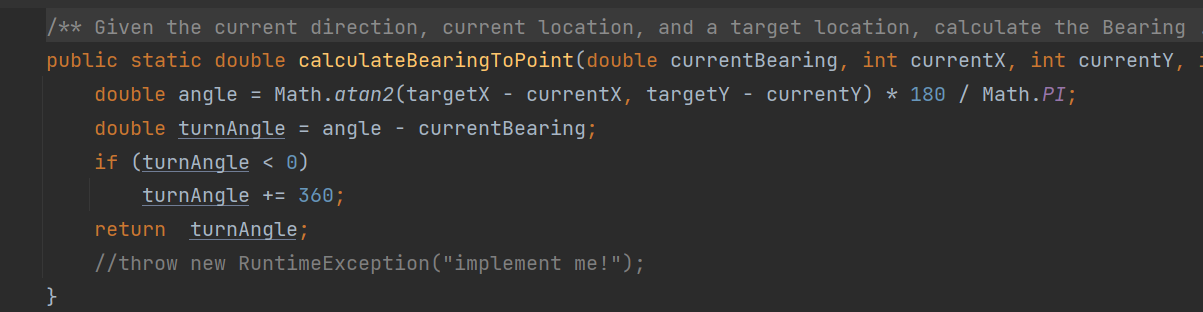


完成之后，在主函数main之中调用该函数，运行结果如下：（边：6，长度：50）

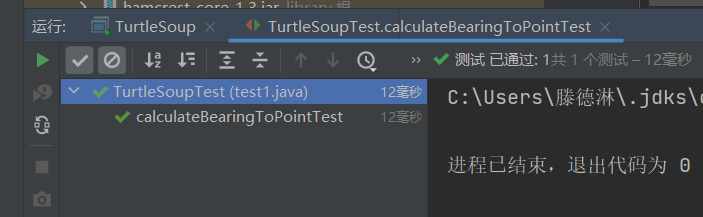


1. 完善calculateBearingToPoint(double currentBearing, int currentX, int currentY, int targetX, int targetY)函数：

该函数的功能是要完成一点到另一点的角度偏移，根据斜率可以很容易知道怎么计算角度偏移，调用Math类中的方法，进行解决；

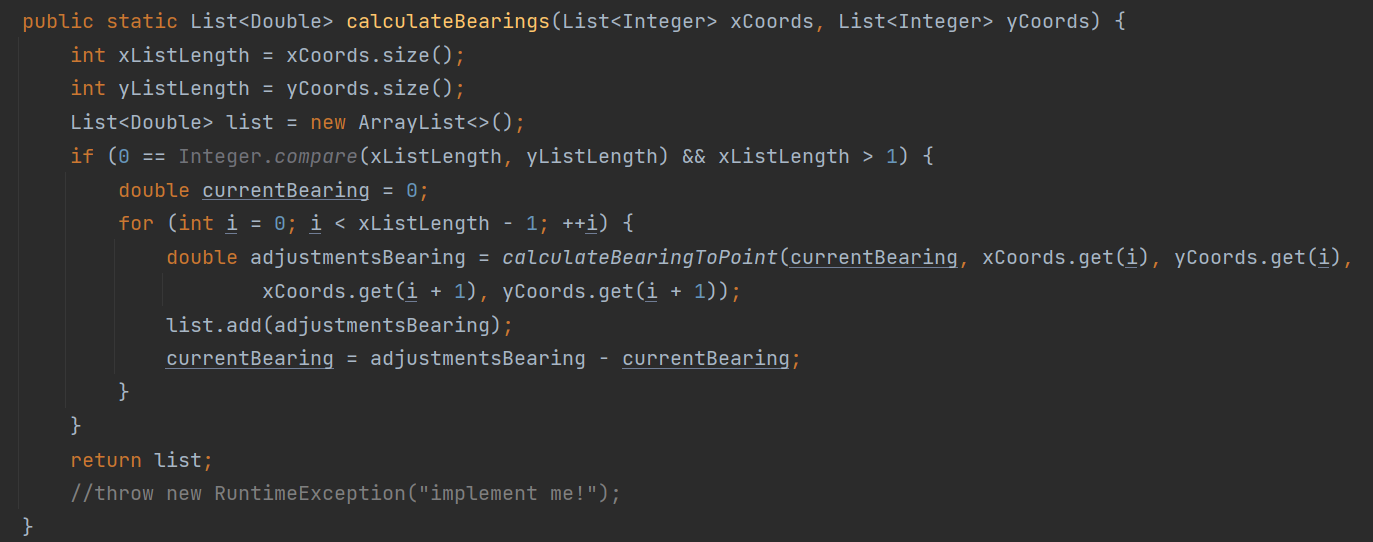


运行测试函数calculateBearingToPointTest()：

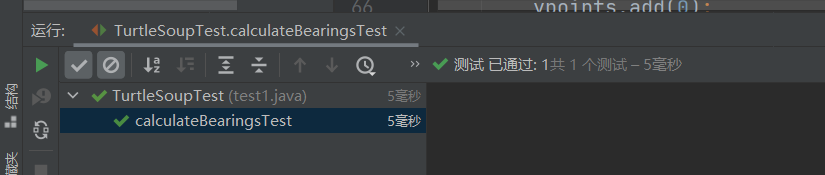


1. 完善calculateBearings(List<Integer> xCoords, List<Integer> yCoords)函数：

该函数的作用是给定一系列点，然后依次算出每两个点之间的斜率，并然后一个链表



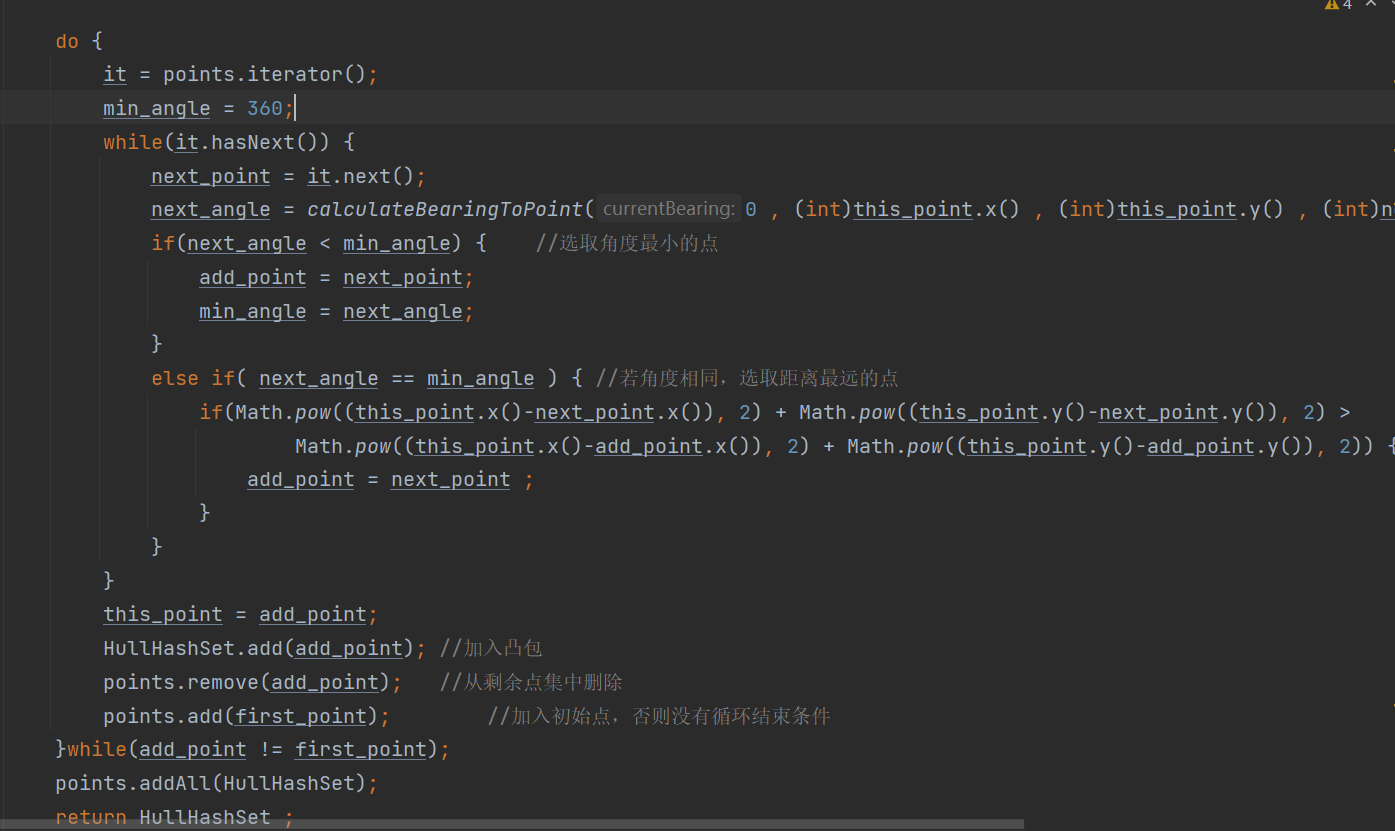
测试函数：



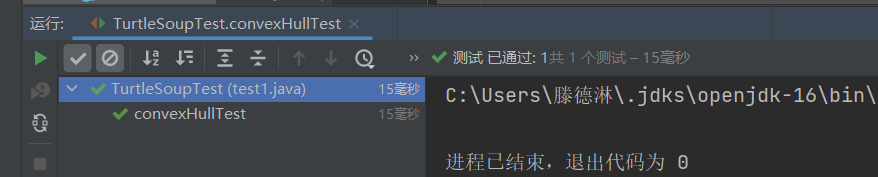
1. 完善Set<Point> convexHull(Set<Point> points)函数：

该函数是计算凸包问题，我的做法是先找出最左侧点，然后往后扫描，找到角度离当前点最小的点，如果角度相同就找距离最小的点，直到所有点都扫描完，然后退出循环，就找到了凸包组成的点了；



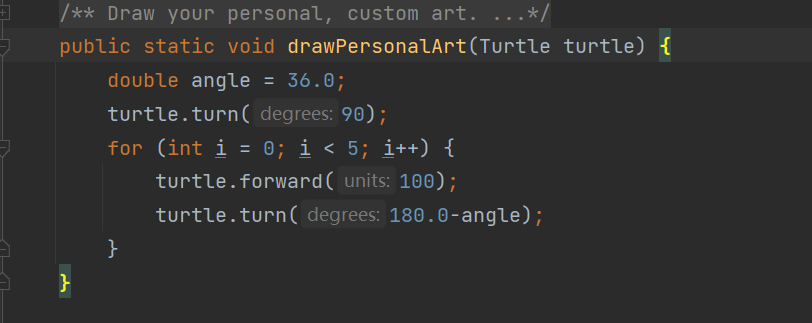


测试函数：



#### Problem 11: Personal art

完善drawPersonalArt(Turtle turtle)函数，并且在主函数中调用函数；



程序运行效果：

