机器人技术第二次作业

说明：程序中附带了 10 个实例，解除注释即可运行！

源程序：

```
//package robo;

import java.awt.AWTException;
import java.awt.Robot;
import java.awt.event.InputEvent;
import java.awt.event.KeyEvent;
import java.util.ArrayList;
import java.util.List;

//点类
class Point {
    private float x;
    private float y;

    public Point(float x, float y) {
        this.x = x;
        this.y = y;
    }

    public void prt() {
        System.out.println("点 Point , 点位置 ( " + this.x + " ," + this.y + " ) ");
    }

    // getter&setter
    public float getX() {
        return x;
    }

    public void setX(float x) {
        this.x = x;
    }

    public float getY() {
        return y;
    }

    public void setY(float y) {
        this.y = y;
    }
```

```
}

//直线类
class Line {
    private float A, B, C;

    public Line(float A, float B, float C) {
        this.A = A;
        this.B = B;
        this.C = C;
    }

    public void prt() {
        System.out.println("直线 Line , 方程为 { " + this.A + " x + " + this.B + " y
+ " + this.C + " = 0 }");
    }

    public float getA() {
        return A;
    }

    public void setA(float a) {
        A = a;
    }

    public float getB() {
        return B;
    }

    public void setB(float b) {
        B = b;
    }

    public float getC() {
        return C;
    }

    public void setC(float c) {
        C = c;
    }

}
```

```
//圆类
class Circle {
    private Point Center;// 圆心 point
    private float Rad;// 圆的半径

    public Circle(Point center, float rad) {
        this.setCenter(center);
        this.Rad = rad;
    }

    public void prt() {
        System.out.println(
                "圆 Circle, 圆心 (" + this.Center.getX() + ", " + this.Center.getY()
+ " ), 半径 " + this.Rad + "");
    }

    public Point getCenter() {
        return Center;
    }

    public void setCenter(Point center) {
        this.Center = center;
    }

    public float getRad() {
        return Rad;
    }

    public void setRad(float rad) {
        Rad = rad;
    }
}

//矩形类
class Rect {
    private static Point Q;// 四个顶点
    private static Point W;
    private static Point E;
    private static Point R;

    public Rect(Point A, Point B, Point C, Point D) {//指定所有顶点，适合任意矩形
        setQ(A);
        setW(B);
        setE(C);
```

```
        setR(D);
    }

    public Rect(Point P, Point Q) {//指定一条对角线，适合

        if(P.getX() < Q.getX() && P.getY() > Q.getY()) {//左上到右下
            setQ(P);
            setE(Q);
            Point D = new Point(P.getX(), Q.getY());
            Point B = new Point(Q.getX(), P.getY());
            setW(B);
            setR(D);
        }
        else {//右上到左下
            setW(P);
            setR(Q);
            Point A = new Point(Q.getX(), P.getY());
            Point C = new Point(P.getX(), Q.getY());
            setQ(A);
            setE(C);
        }
    }
    public void prt() {
        System.out.println("\n矩形 Rect ");
        System.out.println("{ 顶点 Q , 点位置 ( " + Q.getX() + " ," + Q.getY() +
" ) }");
        System.out.println("{ 顶点 W , 点位置 ( " + W.getX() + " ," + W.getY() +
" ) }");
        System.out.println("{ 顶点 E , 点位置 ( " + E.getX() + " ," + E.getY() +
" ) }");
        System.out.println("{ 顶点 R, 点位置 (" +R.getX() +"," +R.getY() +") }");
    }

    public static Point getQ() {
        return Q;
    }

    public void setQ(Point q) {
        Q = q;
    }

    public static Point getW() {
        return W;
    }
```

```
    public void setW(Point w) {
        W = w;
    }

    public static Point getE() {
        return E;
    }

    public void setE(Point e) {
        E = e;
    }

    public static Point getR() {
        return R;
    }

    public void setR(Point r) {
        R = r;
    }

}

class Seg {// 线段类
    Point P1, P2;

    public Seg(Point A, Point B) {
        P1 = A;
        P2 = B;
    }
}

//几何计算类
class geo_cal {

    static double eps = 0.00001F;

    public static void clear() throws AWTException {// 清屏指令
        Robot r = new Robot();
        r.mousePress(InputEvent.BUTTON3_MASK); // 按下鼠标右键
        r.mouseRelease(InputEvent.BUTTON3_MASK); // 释放鼠标右键
        r.keyPress(KeyEvent.VK_CONTROL); // 按下 Ctrl 键
        r.keyPress(KeyEvent.VK_R); // 按下 R 键
        r.keyRelease(KeyEvent.VK_R); // 释放 R 键
```

```
            r.keyRelease(KeyEvent.VK_CONTROL); // 释放 Ctrl 键
            r.delay(100);
        }

        // 直线判断
        public static float xGetY(float A, float B, float C, float x) {// 已知直线上 x 求 y
            float y;
            if (A == 0) {
                // 平行于 x 轴
                y = -C / B;
                return y;
            } else if (B == 0) {
                return 0; // 若平行于 y 轴，默认返回与 x 轴交点
            } else {
                y = (-C - A * x) / B;
                return y;
            }
        }

        public static float yGetX(float A, float B, float C, float y) {// 已知直线上 y 求 x
            float x;
            if (A == 0) {
                return 0;// 直线与 x 轴平行
            } else if (B == 0) {
                x = -C / A;
                return x;
            } else {
                x = (-C - B * y) / A;
                return x;
            }
        }

        public static int L2L(Line L1, Line L2) {// 求直线交点
            // 重合返回 6，相交返回 1，平行返回 0，报错-1
            float px = 0, py = 0;// 作为返回值的参数
            float A1, A2, B1, B2, C1, C2;
            float m;

            L1.prt();
            L2.prt();

            // 参数赋值
```

```
        A1 = L1.getA();
        B1 = L1.getB();
        C1 = L1.getC();
        A2 = L2.getA();
        B2 = L2.getB();
        C2 = L2.getC();

        m = A1 * B2 - A2 * B1;
        // 计算开始
        if ((A1 == 0 && A2 == 0 && B1 * C2 == B2 * C1) || (B1 == 0 && B2 == 0 && A1
* C2 == A2 * C1)
                || (A1 * A2 * B1 * B2 != 0 && A1 / A2 == B1 / B2 && B1 / B2 == C1 /
C2)) {
            System.out.println("两直线重合！");
            return 6;
        }
        if (A1 != 0 && B1 != 0 && A2 != 0 && B2 != 0) {// 都不平行于轴
            if (Math.abs(m) < 0.0005f) {// 平行
                System.out.println("两直线平行！");
                return 0;
            } else {// 有交点
                px = (B1 * C2 - B2 * C1) / (A1 * B2 - A2 * B1);
                py = (A2 * C1 - A1 * C2) / (A1 * B2 - A2 * B1);
                System.out.println("两直线交点为：（" + px + "，" + py + "）!");
                return 1;
            }
        } else {// 有直线平行于轴
            if ((A1 == 0 && A2 == 0) || (B1 == 0 && B2 == 0)) {// 都平行于轴
                System.out.println("两直线平行！");
                return 0;
            } else if (A1 == 0 || B1 == 0) {// L1 平行于坐标轴
                if (A1 == 0) {// x 轴
                    py = -C1 / B1;
                    px = yGetX(A2, B2, C2, py);
                    System.out.println("两直线交点为：（" + px + "，" + py + "）!");
                    return 1;
                } else {// y 轴
                    px = -C1 / A1;
                    py = xGetY(A2, B2, C2, px);
                    System.out.println("两直线交点为：（" + px + "，" + py + "）!");
                    return 1;
                }
            } else if (A2 == 0 || B2 == 0) {// L2 平行于坐标轴
                if (A2 == 0) {// x 轴
```

```
                    py = -C2 / B2;
                    px = yGetX(A1, B1, C1, py);
                    System.out.println("两直线交点为：（" + px + "，" + py + "）!");
                    return 1;
                } else {// y 轴
                    px = -C2 / A2;
                    py = xGetY(A1, B1, C1, px);
                    System.out.println("两直线交点为：（" + px + "，" + py + "）!");
                    return 1;
                }
            }

        }
        return -1;
    }

    public static float P2LD(Point P, Line L) {// 求点到直线的距离
        float A = L.getA();
        float B = L.getB();
        float C = L.getC();
        return (float) (Math.abs(A * P.getX() + B * P.getY() + C) / (Math.sqrt(A * A
+ B * B)));
    }

    // 圆判断
    public static void L2C(Circle Cir, Line L) {// 求直线和圆的交点
        // Point[] points = null;//可能会有多个交点
        float A = L.getA();
        float B = L.getB();
        float C = L.getC();
        float Rad = Cir.getRad();

        Cir.prt();
        L.prt();

        float dis = P2LD(Cir.getCenter(), L);
        if (Math.abs(dis) > Rad) {// 没有交点
            System.out.println("直线和圆相离！");
            return;
        } else if (Math.abs(dis - Cir.getRad()) < 0.0000001f) {// 相切

            float angle = (float) Math.atan(-A / B);
            float ix = (float) (Cir.getCenter().getX() + Rad * Math.cos(Math.PI +
angle));
```

```
            float iy = (float) (Cir.getCenter().getX() + Rad * Math.sin(Math.PI +
angle));

            System.out.println("直线和圆相切，交点为：（" + ix + "，" + iy + "）!");

            return;
        } else {// 相交
            if (Math.abs(B) < 0.00001f) {// L 与 y 轴平行
                float h = (float) Math.sqrt(Rad * Rad - dis * dis);// 求未知边长
                float x = -C / A;// L 上所有点的横坐标
                float y1 = Cir.getCenter().getY() + h;
                float y2 = Cir.getCenter().getY() - h;

                System.out.println("直线和圆相交，一个交点为：（" + x + "，" + y1 +
"）!");
                System.out.println("另一个交点为：（" + x + "，" + y2 + "）!");
            } else {// L 是一般直线
                float k = -A / B;// 化成 y = kx + b 的形式
                float b = -C / B;
                float c = Cir.getCenter().getX();
                float d = Cir.getCenter().getY();
                float r = Cir.getRad();

                // 以下直接套公式
                float x1 = (float) (-(Math.sqrt(
                        (k * k + 1) * r * r - c * c * k * k + (2 * c * d + 2 * b *
c) * k - d * d - 2 * b * d - b * b)
                        + (d + b) * k + c) / (k * k + 1));

                float x2 = (float) ((Math.sqrt(
                        (k * k + 1) * r * r - c * c * k * k + (2 * c * d + 2 * b *
c) * k - d * d - 2 * b * d - b * b)
                        - (d + b) * k - c) / (k * k + 1));

                float y1 = (float) (-(k * (Math.sqrt(k * k * r * r + r * r - c * c *
k * k + 2 * c * d * k
                        + 2 * b * c * k - d * d - 2 * b * d - b * b) + c) + d * k *
k - b) / (k * k + 1));

                float y2 = (float) (-(k * (-Math.sqrt(k * k * r * r + r * r - c * c
* k * k + 2 * c * d * k
                        + 2 * b * c * k - d * d - 2 * b * d - b * b) + c) + d * k *
k - b) / (k * k + 1));
```

```
                System.out.println("直线和圆相交，一个交点为：（" + x1 + "，" + y1
+ "）!");
                System.out.println("另一个交点为：（" + x2 + "，" + y2 + "）!");
            }
        }
    }

    // 矩形判断
    public static Point minus(Point A, Point B) {// 自定义运算
        Point v = new Point(A.getX() - B.getX(), A.getY() - B.getY());
        return v;
    }

    public static float multi(Point A, Point B) {
        return A.getX() * B.getY() - A.getY() * B.getX();
    }

    public static int inter(Point A, Point B, Seg S1)/// 判断直线 AB 是否与线段 S1(CD)
相交
    {
        Point C = S1.P1;
        Point D = S1.P2;
        if ((multi(minus(C, A), minus(B, A)) * multi(minus(D, A), minus(B, A))) < eps)
            return 1;
        return 0;
    }

    public static Point inter_point(Point A, Point B, Seg S1)/// 返回直线 AB 和线段
S1(CD)的交点
    {
        Point C = S1.P1;
        Point D = S1.P2;

        if (inter(A, B, S1) == 0) {
            System.out.println("不相交！");
            return null;
        }

        float area1 = Math.abs(multi(minus(B, A), minus(C, A)));
        float area2 = Math.abs(multi(minus(B, A), minus(D, A)));
        float x = (area1 * D.getX() + area2 * C.getX()) / (area1 + area2);
        float y = (area1 * D.getY() + area2 * C.getY()) / (area1 + area2);
        Point p = new Point(x, y);
```

```
//      System.out.println("直线与矩形边 ( " + C.getX() + " , " + C.getY() + " ) " +
" -- " + " ( " + D.getX() + " , "
//              + D.getY() + " ) " + "交点是:" + p.getX() + " , " + p.getY());
        return p;
    }

    public static Point[] getP(Line L) {// 输入直线求直线上的点
        Point[] points = new Point[2];
        float A = L.getA();
        float B = L.getB();
        float C = L.getC();

        if (B == 0) {// 该线平行于 y 轴
            points[0] = new Point(-C / A, -1);
            points[1] = new Point(-C / A, 1);
        } else if (A == 0) {// x 轴
            points[0] = new Point(-1, -C / B);
            points[1] = new Point(1, -C / B);
        } else {
            points[0] = new Point(-1, xGetY(A, B, C, -1));
            points[1] = new Point(1, xGetY(A, B, C, 1));
        }
        return points;
    }

    public static Line getL(Seg S1) {// 求线段所在的直线
        Point P = S1.P1;
        Point Q = S1.P2;

        float X1 = P.getX();
        float X2 = Q.getX();
        float Y1 = P.getY();
        float Y2 = Q.getY();

        float A = Y2 - Y1;
        float B = X1 - X2;
        float C = X2 * Y1 - X1 * Y2;

        return new Line(A, B, C);
    }

    public static boolean cp(Point A, Point B) {//比较两点坐标是否相同
        float X1 = A.getX();
        float X2 = B.getX();
```

```
        float Y1 = A.getY();
        float Y2 = B.getY();

        if(Math.abs(X1 - X2) < eps && Math.abs(Y1 - Y2) < eps) {
            return true;
        }
        else {
            return false;
        }
    }

    public static void R2L(Rect rec, Line L) throws AWTException {// 矩形与直线交点
        // 变量声明，取得 L 上两点 P1,P2
        Point[] points = getP(L);
        Point P1 = points[0];
        Point P2 = points[1];

        int[] flg = new int[5];// 标志哪些边相交
        Point[] cop = new Point[5];// 交点数组
        Seg[] bian = new Seg[5];

        for (int i = 0; i < 5; i++) {
            cop[i] = new Point(9999, 9999);
        }

        // 矩形顶点，各边为 QW, WE, ER, RQ
        Point Q = Rect.getQ();
        Point W = Rect.getW();
        Point E = Rect.getE();
        Point R = Rect.getR();

        // 各边定义为线段
        Seg B1 = new Seg(Q, W);
        Seg B2 = new Seg(W, E);
        Seg B3 = new Seg(E, R);
        Seg B4 = new Seg(R, Q);

        Line L1 = getL(B1);
        Line L2 = getL(B2);
        Line L3 = getL(B3);
        Line L4 = getL(B4);

        int flag;// 重合标志位
        flag = (L2L(L, L1) + L2L(L, L2) + L2L(L, L3) + L2L(L, L4));
```

```
        // System.out.println(flag);

        int flag_in;
        flg[1] = inter(P1, P2, B1);// 1 表示与该边相交
        flg[2] = inter(P1, P2, B2);
        flg[3] = inter(P1, P2, B3);
        flg[4] = inter(P1, P2, B4);
        flag_in = flg[1] + flg[2] + flg[3] + flg[4];

        bian[1] = B1;
        bian[2] = B2;
        bian[3] = B3;
        bian[4] = B4;

        rec.prt();
        System.out.println("");
        L.prt();
        System.out.println("");

        System.out.println("结果为----------------------");
        if (flag > 5) {// 直线与某边重合
            System.out.println("直线与矩形边重合！");
            return;
        } else if (inter(P1, P2, B1) == 0 && inter(P1, P2, B2) == 0 && inter(P1, P2,
B3) == 0
                && inter(P1, P2, B4) == 0) {// 与四条边都不相交
            System.out.println("直线与矩形不相交！");
        } else {// 一般相交情况
            if (flag_in == 2) {// 相切或者与两个边相交
                for (int i = 1; i < 5; i++) {
                    if (flg[i] == 1) {//
                        cop[i] = inter_point(P1, P2, bian[i]);
                    }
                }
            } else if (flag_in == 3) {// 过一个顶点且与另一边相交
                for (int i = 1; i < 5; i++) {
                    if (flg[i] == 1) {//
                        cop[i] = inter_point(P1, P2, bian[i]);
                    }
                }
            } else {// 过对角线
                for (int i = 1; i < 5; i++) {
                    if (flg[i] == 1) {//
                        cop[i] = inter_point(P1, P2, bian[i]);
```

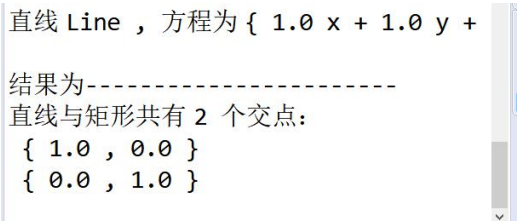
```
                    }
                }
            }

        }

        // 判断结束，交点已存储到 cop 数组中
        // 打印输出

//      for (int i = 1; i < 5; i++) {
//          if (cop[i].getX() != 9999 && cop[i].getY() != 9999) {//
//              System.out.println("交点是: {" + cop[i].getX() + ", " + cop[i].getY()
+ " }");
//          }
//      }

        //考虑到过顶点的情况，需要删除重复的

        Point[] out = new Point[5];
        for (int i = 0; i < 5; i++) {
            out[i] = new Point(9999, 9999);
        }

        for (int i = 1; i < 5; i++) {//去重循环
            int sflag = 0;
            for (int j = 1; j < 5; j++) {
                if(cp(out[j], cop[i])) {
                    sflag = 1;
                }
            }
            if(sflag == 1) {
                continue;
            }
            else {
                out[i] = cop[i];
            }

        }

        int k = 0;
        for (int i = 1; i < 5; i++) {//输出
            if (out[i].getX() != 9999 && out[i].getY() != 9999) {//
                k++;
            }
```

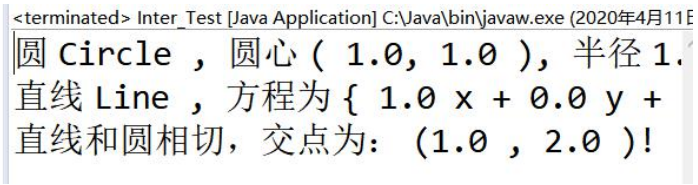
```
        }

        System.out.println("直线与矩形共有 " + k + " 个交点：" );
        for (int i = 1; i < 5; i++) {//输出
            if (out[i].getX() != 9999 && out[i].getY() != 9999) {//
                System.out.println(" { " + out[i].getX() + " , " + out[i].getY() +
" }");
            }
        }

    }
}

public class Inter_Test {// 主测试程序

    /*
     * 说明 测试用例已经提供，解除注释即可使用 需要输入的参数已经列出
     */

    public static void main(String[] argc) throws AWTException {// 主函数

        /*
         *  测试直线交点；
         *  需要输入两条直线的参数；
         */

        // 实例 1：一般相交
//      Line L1 = new Line(9,5,3);
//      Line L2 = new Line(9,0,0);
//      geo_cal.L2L(L1,L2);

        // 实例 2：重合
//      Line L1 = new Line(4,6,8);
//      Line L2 = new Line(2,3,4);
//      geo_cal.L2L(L1,L2);

        // 实例 3：平行
//      Line L1 = new Line(9,5,3);
//      Line L2 = new Line(9,5,0);
//      geo_cal.L2L(L1,L2);

        /*
         *  测试直线与圆交点；
         *  需要输入圆心、半径、直线参数；
```

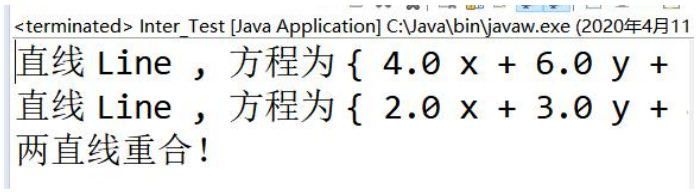
```
         */

        // 实例 1：直线与圆相交
//      Point p1 = new Point(0,0);
//      Line L3 = new Line(2,0.5f,-1);
//      Circle C1 = new Circle(p1,1);
//
//      geo_cal.L2C(C1,L3);

        // 实例 2：直线与圆相切
//      Point p1 = new Point(1,1);
//      Line L3 = new Line(1,0,-2);
//      Circle C1 = new Circle(p1,1);
//
//      geo_cal.L2C(C1,L3);

        // 实例 3：直线与圆相离
//      Point p1 = new Point(1,1);
//      Line L3 = new Line(1,0,-7);
//      Circle C1 = new Circle(p1,1);
//
//      geo_cal.L2C(C1,L3);

        /*
         * 测试直线与矩形交点
         * 以下都定义了 4 个顶点；
         * 在计算某边与坐标轴平行的矩形时，可选择 AC 或 BD（对角线）作为参数；
         * 如果要计算一般的矩形，需要输入全部顶点；
         */

        // 实例 1：两个交点，其中一个是顶点
        Point A = new Point(-1, 1);
        Point B = new Point(1, 1);
        Point C = new Point(1, 0);
        Point D = new Point(-1, 0);

        Rect Rec1 = new Rect(B,D);
        Line LT = new Line(-1, -1, 1);
        geo_cal.R2L(Rec1, LT);

        // 实例 2：直线和一条边重合
//      Point A = new Point(-1,1);
//      Point B = new Point(1,1);
//      Point C = new Point(1,0);
```

```
//        Point D = new Point(-1,0);
//
//        Rect Rec1 = new Rect(A,C);
//        Line LT = new Line(1,0,-1);
//        geo_cal.R2L(Rec1,LT);

          // 实例 3：直线过对角线
//        Point A = new Point(-1,1);
//        Point B = new Point(1,1);
//        Point C = new Point(1,-1);
//        Point D = new Point(-1,-1);
//
//        Rect Rec1 = new Rect(B,D);//只用两个顶点定义
//        Line LT = new Line(1,-1,0);
//        geo_cal.R2L(Rec1,LT);

          // 实例 4：一般相交，有两个交点
//        Point A = new Point(0,2);
//        Point B = new Point(2,2);
//        Point C = new Point(2,0);
//        Point D = new Point(0,0);
//
//        Rect Rec1 = new Rect(A,C);
//        Line LT = new Line(1,1,-1);
//        geo_cal.R2L(Rec1,LT);


    }
}
```

运行截图：






<terminated> Inter_Test [Java Application] C:\Java\bin\javaw.exe (2020年4月11
直线 Line , 方程为{ 4.0 x + 6.0 y +
直线 Line , 方程为{ 2.0 x + 3.0 y +
两直线重合!