Copyright 2003 Bill Campbell and Ethan Bolker

/**
 * A LinearEquation models equations of the form y = mx + b.
 *
 * @version 2
 */

public class LinearEquation
{
    private double m;      // The equation's slope
    private double b;      // The equation's y-intercept

    /**
     * Construct a LinearEquation from a slope and y-intercept.
     *
     * @param m the slope.
     * @param b the y-intercept.
     */
    public LinearEquation( double m, double b )
    {
        this.m = m;
        this.b = b;
    }

    /**
     * Construct a LinearEquation from two points.
     *
     * @param x1 the x coordinate of the first point
     * @param y1 the y coordinate of the first point
     * @param x2 the x coordinate of the second point
     * @param y2 the y coordinate of the second point
     */
    public LinearEquation( double x1, double y1,
                           double x2, double y2 )
    {
        m = (y2 - y1) / (x2 - x1);
        b = y1 - x1 * m;
    }

    /**
     * Compute y, given x.
     *
     * @param x the input value.
     * @return the corresponding value of y: mx+b.
     */
    public double compute( double x )
    {
        return m*x + b;
    }

    /**
     * Compute the inverse of this linear equation.
     *
     * @return the LinearEquation object you get by "solving for x".
     */
    public LinearEquation getInverse()
    {
        return new LinearEquation( 1.0/m, -b/m );
    }
}

/**
 * @version 2
 */

public class LinearEquation
{
    private double m;      // The equation's slope
    private double b;      // The equation's y-intercept

    /**
     * Construct a LinearEquation from a slope and y-intercept.
     *
     * @param m the slope.
     * @param b the y-intercept.
     */
    public LinearEquation( double m, double b )
    {
        this.m = m;
        this.b = b;
    }

    /**
     * Construct a LinearEquation from two points.
     *
     * @param x1 the x coordinate of the first point
     * @param y1 the y coordinate of the first point
     * @param x2 the x coordinate of the second point
     * @param y2 the y coordinate of the second point
     */
    public LinearEquation( double x1, double y1,
                           double x2, double y2 )
    {
        m = (y2 - y1) / (x2 - x1);
        b = y1 - x1 * m;
    }

    /**
     * Compute y, given x.
     *
     * @param x the input value.
     * @return the corresponding value of y: mx+b.
     */
    public double compute( double x )
    {
        return m*x + b;
    }

    /**
     * Compute the inverse of this linear equation.
     *
     * @return the LinearEquation object you get by "solving for x".
     */
    public LinearEquation getInverse()
    {
        return new LinearEquation( 1.0/m, -b/m );
    }
}

/**
 * @version 2
 */

public class LinearEquation
{
    private double m;      // The equation's slope
    private double b;      // The equation's y-intercept

    /**
     * Construct a LinearEquation from a slope and y-intercept.
     *
     * @param m the slope.
     * @param b the y-intercept.
     */
    public LinearEquation( double m, double b )
    {
        this.m = m;
        this.b = b;
    }

    /**
     * Construct a LinearEquation from two points.
     *
     * @param x1 the x coordinate of the first point
     * @param y1 the y coordinate of the first point
     * @param x2 the x coordinate of the second point
     * @param y2 the y coordinate of the second point
     */
    public LinearEquation( double x1, double y1,
                           double x2, double y2 )
    {
        m = (y2 - y1) / (x2 - x1);
        b = y1 - x1 * m;
    }

    /**
     * Compute y, given x.
     *
     * @param x the input value.
     * @return the corresponding value of y: mx+b.
     */
    public double compute( double x )
    {
        return m*x + b;
    }

    /**
     * Compute the inverse of this linear equation.
     *
     * @return the LinearEquation object you get by "solving for x".
     */
    public LinearEquation getInverse()
    {
        return new LinearEquation( 1.0/m, -b/m );
    }
}