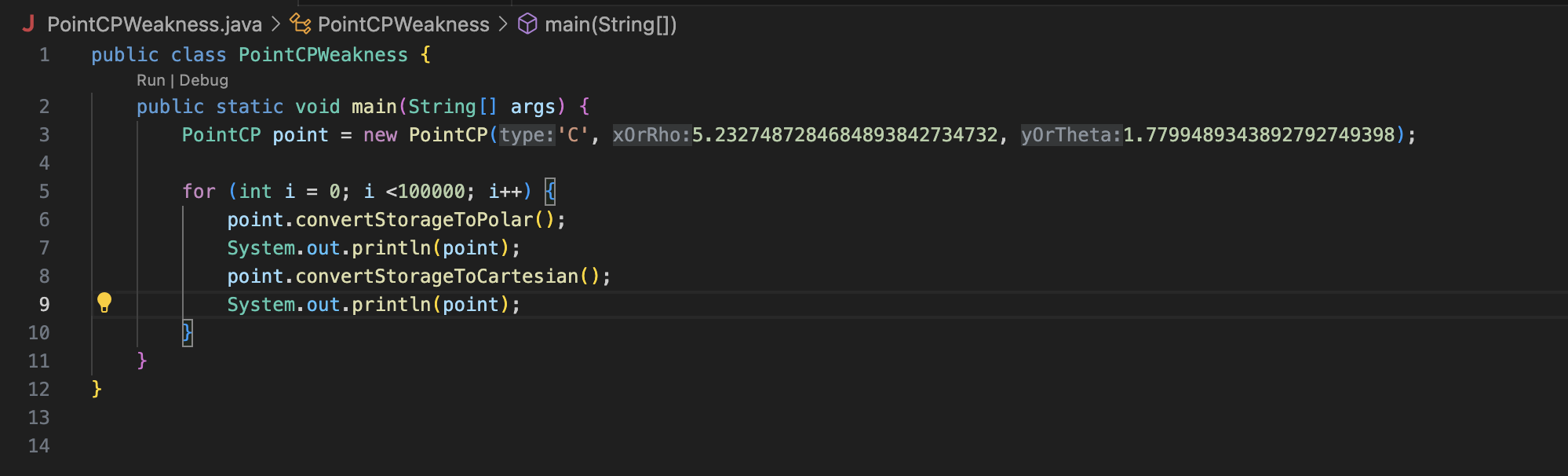
**Question 25:**

**a- Users of the PointCP class can benefit from explicitly modifying the internal storage format for their convenience. This flexibility allows users to select their preferred format and effortlessly switch between the two. Furthermore, it simplifies the process of utilizing data in both Cartesian and polar formats for various operations.**

**b- The decimal point of given coordinates can veer off slightly with more and more decimal points being added as it’s constantly changed between cartesian and polar storage types.**

**c-**

****

**Question 27:**

**public abstract class PointCP**

**{**

**protected char typeCoord;**

**public PointCP(char typeCoord)**

**{**

**if(typeCoord != 'C' && typeCoord != 'P')**

**throw new IllegalArgumentException("Invalid coordinate type.");**

**this.typeCoord = typeCoord;**

**}**

**abstract public class Point {**

**abstract double getX();**

**abstract double getY();**

**abstract double getRho();**

**abstract double getTheta();**

**abstract double getDistance(PointCP other);**

**abstract PointCP rotatePoint(double rotation);**

**public abstract PointCP convertStorageToPolar();**

**public abstract PointCP convertStorageToCartesian();**

**public static class Polar extends PointCP {**

**private double rho;**

**private double theta;**

**public Polar(double rho, double theta)**

**{**

**super('P');**

**this.rho = rho;**

**this.theta = theta;**

**}**

**public double getX()**

**{**

**return (Math.cos(Math.toRadians(theta)) \* rho);**

**}**

**public double getY()**

**{**

**return (Math.sin(Math.toRadians(theta)) \* rho);**

**}**

**public double getRho()**

**{**

**return rho;**

**}**

**public double getTheta()**

**{**

**return theta;**

**}**

**public double getDistance(Point other)**

**{**

**double deltaX = getX() - other.getX();**

**double deltaY = getY() - other.getY();**

**return Math.sqrt((Math.pow(deltaX, 2) + Math.pow(deltaY, 2)));**

**}**

**public PointCP rotatePoint(double rotation)**

**{**

**double radRotation = Math.toRadians(rotation);**

**double X = getX();**

**double Y = getY();**

**return new Cartesian(**

**(Math.cos(radRotation) \* X) - (Math.sin(radRotation) \* Y),**

**(Math.sin(radRotation) \* X) + (Math.cos(radRotation) \* Y));**

**}**

**public String toString()**

**{**

**return "Stored as polar [" + getRho() + "," + getTheta() + "]";**

**}**

**}**

**public static class Cartesian extends PointCP {**

**private double x;**

**private double y;**

**public Cartesian(double x, double y)**

**{**

**super('C');**

**this.x = x;**

**this.y = y;**

**}**

**public double getX()**

**{**

**return x;**

**}**

**public double getY()**

**{**

**return y;**

**}**

**public double getRho()**

**{**

**return (Math.sqrt(Math.pow(x, 2) + Math.pow(y, 2)));**

**}**

**public double getTheta()**

**{**

**return Math.toDegrees(Math.atan2(y, x));**

**}**

**public double getDistance(Point other)**

**{**

**double deltaX = getX() - other.getX();**

**double deltaY = getY() - other.getY();**

**return Math.sqrt((Math.pow(deltaX, 2) + Math.pow(deltaY, 2)));**

**}**

**public PointCP rotatePoint(double rotation)**

**{**

**double radRotation = Math.toRadians(rotation);**

**double X = getX();**

**double Y = getY();**

**return new Cartesian(**

**(Math.cos(radRotation) \* X) - (Math.sin(radRotation) \* Y),**

**(Math.sin(radRotation) \* X) + (Math.cos(radRotation) \* Y));**

**}**

**public String toString()**

**{**

**return "Stored as Cartesian (" + getX() + "," + getY() + ")";**

**}**

**}**

**}**

**}**