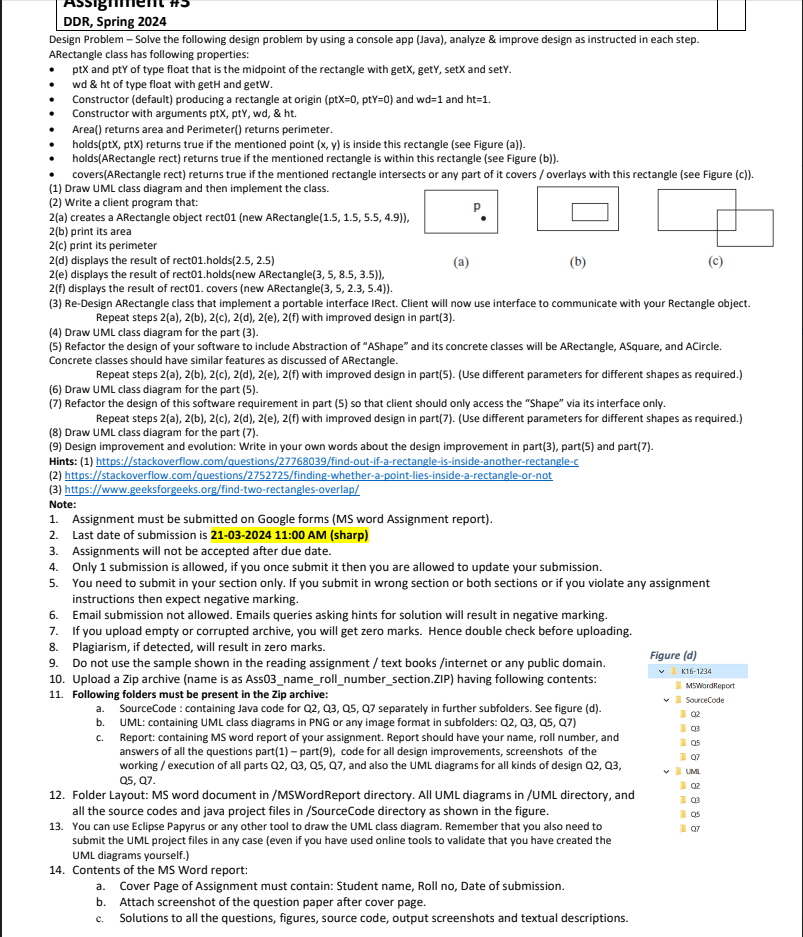
**Ali Nadir**

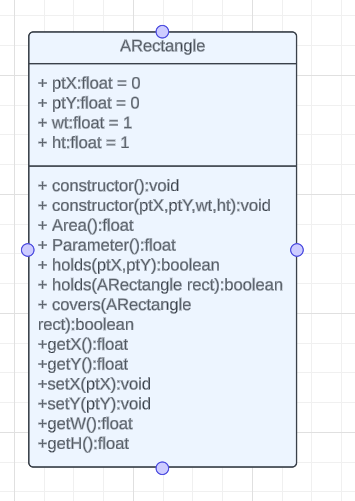
**20K-0325**

**Thursday, March 21st 2024**

****

**Q1:**

**Class UML**

****

**Class code:**

**package arectangle;**

**public class ARectangle {**

**private float ptX;**

**private float ptY;**

**private float wt;**

**private float ht;**

**public ARectangle(){**

**this.ptX = 0;**

**this.ptY = 0;**

**this.wt = 1;**

**this.ht = 1;**

**}**

**public ARectangle(float ptX, float ptY, float wt,float ht){**

**this.ptX = ptX;**

**this.ptY =ptY;**

**this.wt = wt;**

**this.ht = ht;**

**}**

**public float Area(){**

**return wt\*ht;**

**}**

**public float Parameter(){**

**return 2\*(ht+wt);**

**}**

**public float getX(){**

**return ptX;**

**}**

**public float getY(){**

**return ptX;**

**}**

**public void setX(float ptX){**

**this.ptX = ptX;**

**}**

**public void setY(float ptY){**

**this.ptY = ptY;**

**}**

**public float getW(){**

**return wt;**

**}**

**public float getH(){**

**return ht;**

**}**

**public boolean holds(float ptX,float ptY){**

**float halfWidth = this.wt / 2;**

**float halfHeight = this.ht / 2;**

**float leftX = this.ptX - halfWidth;**

**float rightX = this.ptX + halfWidth;**

**float topY = this.ptY - halfHeight;**

**float bottomY = this.ptY + halfHeight;**

**return (ptX >= leftX && ptX <= rightX && ptY >= topY && ptY <= bottomY);**

**}**

**public boolean holds (ARectangle rect){**

**float innerHalfHeight = rect.getH() / 2;**

**float innerHalfWidth = rect.getW() / 2;**

**float outerHalfWidth = this.wt / 2;**

**float outerHalfHeight = this.ht / 2;**

**float innerLeftX = rect.getX() - innerHalfWidth;**

**float innerRightX = rect.getX() + innerHalfWidth;**

**float innerTopY = rect.getY() - innerHalfHeight;**

**float innerBottomY = rect.getY() + innerHalfHeight;**

**float outerLeftX = this.ptX - outerHalfWidth;**

**float outerRightX = this.ptX + outerHalfWidth;**

**float outerTopY = this.ptY - outerHalfHeight;**

**float outerBottomY = this.ptY + outerHalfHeight;**

**return (**

**innerLeftX >= outerLeftX &&**

**innerRightX <= outerRightX &&**

**innerTopY >= outerTopY &&**

**innerBottomY <= outerBottomY**

**);**

**}**

**public boolean covers (ARectangle rect){**

**float r1LX = this.ptX-this.wt/2;**

**float r1LY = this.ptY+this.ht/2;**

**float r1RX = this.ptX+this.wt/2;**

**float r1RY = this.ptY-this.ht/2;**

**float r2LX = rect.getX()-rect.getW()/2;**

**float r2LY = rect.getY()+rect.getW()/2;**

**float r2RX = rect.getX()+rect.getW()/2;**

**float r2RY = rect.getY()-rect.getW()/2;**

**// if rectangle has area 0, no overlap**

**if (r1LX == r1RX || r1LY == r1RY || r2RX == r2LX || r2LY == r2RY)**

**return false;**

**// If one rectangle is on left side of other**

**if (r1LX > r1RX || r2LX > r1RX)**

**return false;**

**// If one rectangle is above other**

**if (r1RY > r2LY || r2RY > r1LY)**

**return false;**

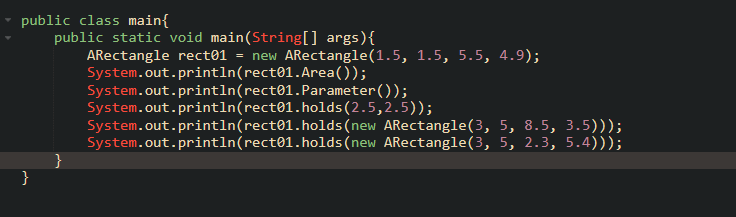
**return true;**

**}**

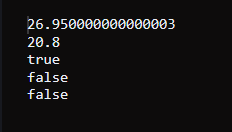
**}**

**Q2:**

**Code:**

****

**Output:**

****

**Q3:**

**Code:**

**public class ARectangle implements IRect {**

**private float ptX;**

**private float ptY;**

**private float wt;**

**private float ht;**

**public ARectangle(){**

**this.ptX = 0;**

**this.ptY = 0;**

**this.wt = 1;**

**this.ht = 1;**

**}**

**public ARectangle(float ptX, float ptY, float wt,float ht){**

**this.ptX = ptX;**

**this.ptY =ptY;**

**this.wt = wt;**

**this.ht = ht;**

**}**

**public float Area(){**

**return wt\*ht;**

**}**

**public float Parameter(){**

**return 2\*(ht+wt);**

**}**

**public float getX(){**

**return ptX;**

**}**

**public float getY(){**

**return ptX;**

**}**

**public void setX(float ptX){**

**this.ptX = ptX;**

**}**

**public void setY(float ptY){**

**this.ptY = ptY;**

**}**

**public float getW(){**

**return wt;**

**}**

**public float getH(){**

**return ht;**

**}**

**public boolean holds(float ptX,float ptY){**

**float halfWidth = this.wt / 2;**

**float halfHeight = this.ht / 2;**

**float leftX = this.ptX - halfWidth;**

**float rightX = this.ptX + halfWidth;**

**float topY = this.ptY - halfHeight;**

**float bottomY = this.ptY + halfHeight;**

**return (ptX >= leftX && ptX <= rightX && ptY >= topY && ptY <= bottomY);**

**}**

**public boolean holds (ARectangle rect){**

**float innerHalfHeight = rect.getH() / 2;**

**float innerHalfWidth = rect.getW() / 2;**

**float outerHalfWidth = this.wt / 2;**

**float outerHalfHeight = this.ht / 2;**

**float innerLeftX = rect.getX() - innerHalfWidth;**

**float innerRightX = rect.getX() + innerHalfWidth;**

**float innerTopY = rect.getY() - innerHalfHeight;**

**float innerBottomY = rect.getY() + innerHalfHeight;**

**float outerLeftX = this.ptX - outerHalfWidth;**

**float outerRightX = this.ptX + outerHalfWidth;**

**float outerTopY = this.ptY - outerHalfHeight;**

**float outerBottomY = this.ptY + outerHalfHeight;**

**return (**

**innerLeftX >= outerLeftX &&**

**innerRightX <= outerRightX &&**

**innerTopY >= outerTopY &&**

**innerBottomY <= outerBottomY**

**);**

**}**

**public boolean covers (ARectangle rect){**

**float r1LX = this.ptX-this.wt/2;**

**float r1LY = this.ptY+this.ht/2;**

**float r1RX = this.ptX+this.wt/2;**

**float r1RY = this.ptY-this.ht/2;**

**float r2LX = rect.getX()-rect.getW()/2;**

**float r2LY = rect.getY()+rect.getW()/2;**

**float r2RX = rect.getX()+rect.getW()/2;**

**float r2RY = rect.getY()-rect.getW()/2;**

**// if rectangle has area 0, no overlap**

**if (r1LX == r1RX || r1LY == r1RY || r2RX == r2LX || r2LY == r2RY)**

**return false;**

**// If one rectangle is on left side of other**

**if (r1LX > r1RX || r2LX > r1RX)**

**return false;**

**// If one rectangle is above other**

**if (r1RY > r2LY || r2RY > r1LY)**

**return false;**

**return true;**

**}**

**}**

**public interface IRect{**

**public float Area();**

**public float Parameter();**

**public boolean holds (ARectangle rect);**

**public boolean holds (float ptX,float ptY);**

**public boolean covers (ARectangle rect);**

**}**

**public class main{**

**public static void main(String[] args){**

**IRect rect01 = new ARectangle(1.5, 1.5, 5.5, 4.9);**

**System.out.println(rect01.Area());**

**System.out.println(rect01.Parameter());**

**System.out.println(rect01.holds(2.5,2.5));**

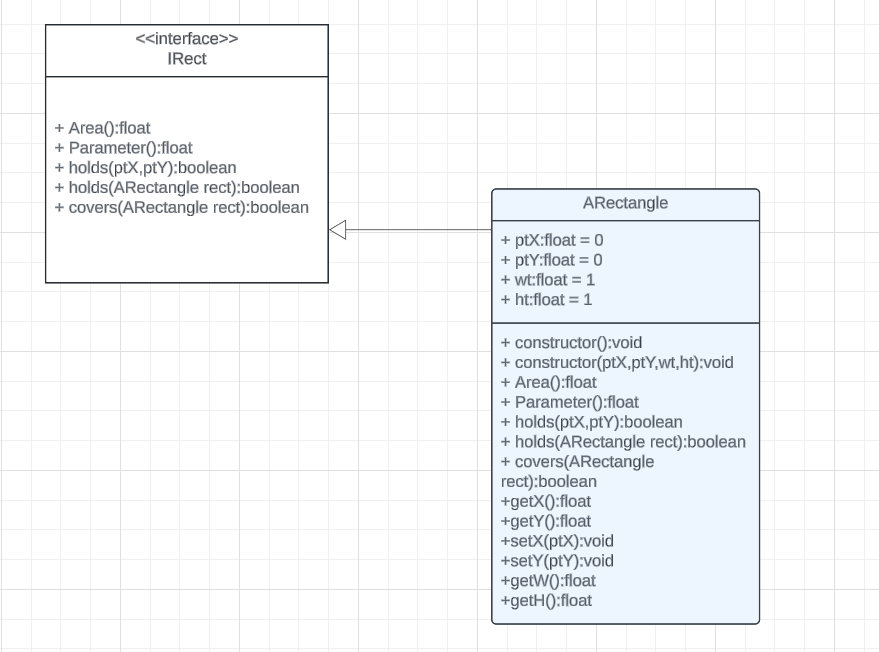
**System.out.println(rect01.holds(new ARectangle(3, 5, 8.5, 3.5)));**

**System.out.println(rect01.holds(new ARectangle(3, 5, 2.3, 5.4)));**

**}**

**}**

**Q4:**

****

**Q5:**

**Code:**

**public abstract class AShape{**

**public abstract double Area();**

**public abstract double Parameter();**

**public abstract boolean holds (double ptX, double ptY);**

**public abstract boolean holds (AShape shape);**

**public abstract boolean covers (AShape shape);**

**}**

**public class ACircle extends AShape {**

**private double ptX;**

**private double ptY;**

**private double radius;**

**public ACircle(){**

**this.ptX = 0;**

**this.ptY = 0;**

**this.radius = 1;**

**}**

**public ACircle(double ptX, double ptY, double radius){**

**this.ptX = ptX;**

**this.ptY =ptY;**

**this.radius = radius;**

**}**

**@Override**

**public double Area(){**

**return 3.142\*radius\*radius;**

**}**

**@Override**

**public double Parameter(){**

**return 2\*3.142\*radius;**

**}**

**public double getX(){**

**return ptX;**

**}**

**public double getY(){**

**return ptX;**

**}**

**public void setX(double ptX){**

**this.ptX = ptX;**

**}**

**public void setY(double ptY){**

**this.ptY = ptY;**

**}**

**public double getRadius(){**

**return this.radius;**

**}**

**@Override**

**public boolean holds(double ptX,double ptY){**

**return Math.pow(ptX-this.ptX,2) + Math.pow(ptY - this.ptY,2) <= Math.pow(radius,2);**

**}**

**@Override**

**public boolean holds(AShape shape) {**

**if (shape instanceof ACircle) {**

**ACircle circle = (ACircle) shape;**

**double distanceSquared = (Math.pow(ptX - circle.getX(), 2) +**

**Math.pow(ptY - circle.getY(), 2));**

**double combinedRadius = radius + circle.getRadius();**

**return distanceSquared <= combinedRadius \* combinedRadius;**

**}**

**return false; // For other shapes, return false**

**}**

**@Override**

**public boolean covers(AShape shape) {**

**if (shape instanceof ACircle) {**

**ACircle circle = (ACircle) shape;**

**double distanceSquared = (Math.pow(ptX - circle.getX(), 2) +**

**Math.pow(ptY - circle.getY(), 2));**

**double combinedRadius = radius + circle.getRadius();**

**return distanceSquared <= combinedRadius \* combinedRadius;**

**}**

**return false; // For other shapes, return false**

**}**

**}**

**public class ARectangle extends AShape {**

**private double ptX;**

**private double ptY;**

**private double wt;**

**private double ht;**

**public ARectangle(){**

**this.ptX = 0;**

**this.ptY = 0;**

**this.wt = 1;**

**this.ht = 1;**

**}**

**public ARectangle(double ptX, double ptY, double wt,double ht){**

**this.ptX = ptX;**

**this.ptY =ptY;**

**this.wt = wt;**

**this.ht = ht;**

**}**

**@Override**

**public double Area(){**

**return wt\*ht;**

**}**

**@Override**

**public double Parameter(){**

**return 2\*(ht+wt);**

**}**

**public double getX(){**

**return ptX;**

**}**

**public double getY(){**

**return ptX;**

**}**

**public void setX(double ptX){**

**this.ptX = ptX;**

**}**

**public void setY(double ptY){**

**this.ptY = ptY;**

**}**

**public double getW(){**

**return wt;**

**}**

**public double getH(){**

**return ht;**

**}**

**@Override**

**public boolean holds(double ptX,double ptY){**

**double halfWidth = this.wt / 2;**

**double halfHeight = this.ht / 2;**

**double leftX = this.ptX - halfWidth;**

**double rightX = this.ptX + halfWidth;**

**double topY = this.ptY - halfHeight;**

**double bottomY = this.ptY + halfHeight;**

**return (ptX >= leftX && ptX <= rightX && ptY >= topY && ptY <= bottomY);**

**}**

**@Override**

**public boolean holds (AShape shape){**

**if(shape instanceof ARectangle){**

**ARectangle rect = (ARectangle) shape;**

**double innerHalfHeight = rect.getH() / 2;**

**double innerHalfWidth = rect.getW() / 2;**

**double outerHalfWidth = this.wt / 2;**

**double outerHalfHeight = this.ht / 2;**

**double innerLeftX = rect.getX() - innerHalfWidth;**

**double innerRightX = rect.getX() + innerHalfWidth;**

**double innerTopY = rect.getY() - innerHalfHeight;**

**double innerBottomY = rect.getY() + innerHalfHeight;**

**double outerLeftX = this.ptX - outerHalfWidth;**

**double outerRightX = this.ptX + outerHalfWidth;**

**double outerTopY = this.ptY - outerHalfHeight;**

**double outerBottomY = this.ptY + outerHalfHeight;**

**return (**

**innerLeftX >= outerLeftX &&**

**innerRightX <= outerRightX &&**

**innerTopY >= outerTopY &&**

**innerBottomY <= outerBottomY**

**);**

**}else**

**return false;**

**}**

**@Override**

**public boolean covers (AShape shape){**

**if(shape instanceof ARectangle){**

**ARectangle rect = (ARectangle) shape;**

**double r1LX = this.ptX-this.wt/2;**

**double r1LY = this.ptY+this.ht/2;**

**double r1RX = this.ptX+this.wt/2;**

**double r1RY = this.ptY-this.ht/2;**

**double r2LX = rect.getX()-rect.getW()/2;**

**double r2LY = rect.getY()+rect.getW()/2;**

**double r2RX = rect.getX()+rect.getW()/2;**

**double r2RY = rect.getY()-rect.getW()/2;**

**// if rectangle has area 0, no overlap**

**if (r1LX == r1RX || r1LY == r1RY || r2RX == r2LX || r2LY == r2RY)**

**return false;**

**// If one rectangle is on left side of other**

**if (r1LX > r1RX || r2LX > r1RX)**

**return false;**

**// If one rectangle is above other**

**if (r1RY > r2LY || r2RY > r1LY)**

**return false;**

**return true;**

**}**

**return false;**

**}**

**}**

**public class ASquare extends AShape {**

**private double ptX;**

**private double ptY;**

**private double len;**

**public ASquare(){**

**this.ptX = 0;**

**this.ptY = 0;**

**this.len = 1;**

**}**

**public ASquare(double ptX, double ptY, double wt,double ht){**

**this.ptX = ptX;**

**this.ptY =ptY;**

**this.len = 1;**

**}**

**@Override**

**public double Area(){**

**return len\*len;**

**}**

**@Override**

**public double Parameter(){**

**return 4\*len;**

**}**

**public double getX(){**

**return ptX;**

**}**

**public double getY(){**

**return ptX;**

**}**

**public void setX(double ptX){**

**this.ptX = ptX;**

**}**

**public void setY(double ptY){**

**this.ptY = ptY;**

**}**

**public double getL(){**

**return len;**

**}**

**@Override**

**public boolean holds(double ptX,double ptY){**

**double halflength = this.len / 2;**

**double leftX = this.ptX - halflength;**

**double rightX = this.ptX + halflength;**

**double topY = this.ptY - halflength;**

**double bottomY = this.ptY + halflength;**

**return (ptX >= leftX && ptX <= rightX && ptY >= topY && ptY <= bottomY);**

**}**

**@Override**

**public boolean holds (AShape shape){**

**if(shape instanceof ASquare){**

**ASquare rect = (ASquare) shape;**

**double innerHalflength = rect.getL() / 2;**

**double outerHalflength = this.len / 2;**

**double innerLeftX = rect.getX() - innerHalflength;**

**double innerRightX = rect.getX() + innerHalflength;**

**double innerTopY = rect.getY() - innerHalflength;**

**double innerBottomY = rect.getY() + innerHalflength;**

**double outerLeftX = this.ptX - outerHalflength;**

**double outerRightX = this.ptX + outerHalflength;**

**double outerTopY = this.ptY - outerHalflength;**

**double outerBottomY = this.ptY + outerHalflength;**

**return (**

**innerLeftX >= outerLeftX &&**

**innerRightX <= outerRightX &&**

**innerTopY >= outerTopY &&**

**innerBottomY <= outerBottomY**

**);**

**}else**

**return false;**

**}**

**@Override**

**public boolean covers (AShape shape){**

**if(shape instanceof ASquare){**

**ASquare rect = (ASquare) shape;**

**double r1LX = this.ptX-this.len/2;**

**double r1LY = this.ptY+this.len/2;**

**double r1RX = this.ptX+this.len/2;**

**double r1RY = this.ptY-this.len/2;**

**double r2LX = rect.getX()-rect.getL()/2;**

**double r2LY = rect.getY()+rect.getL()/2;**

**double r2RX = rect.getX()+rect.getL()/2;**

**double r2RY = rect.getY()-rect.getL()/2;**

**// if rectangle has area 0, no overlap**

**if (r1LX == r1RX || r1LY == r1RY || r2RX == r2LX || r2LY == r2RY)**

**return false;**

**// If one rectangle is on left side of other**

**if (r1LX > r1RX || r2LX > r1RX)**

**return false;**

**// If one rectangle is above other**

**if (r1RY > r2LY || r2RY > r1LY)**

**return false;**

**return true;**

**}**

**return false;**

**}**

**}**

**public class main{**

**public static void main(String[] args){**

**System.out.println("For rectangle:\n");**

**AShape shape = new ARectangle(1.5, 1.5, 5.5, 4.9);**

**System.out.println(shape.Area());**

**System.out.println(shape.Parameter());**

**System.out.println(shape.holds(2.5,2.5));**

**System.out.println(shape.holds(new ARectangle(3, 5, 8.5, 3.5)));**

**System.out.println(shape.holds(new ARectangle(3, 5, 2.3, 5.4)));**

**System.out.println("\nFor circle:\n");**

**shape = new ACircle(1.5, 1.5, 4.5);**

**System.out.println(shape.Area());**

**System.out.println(shape.Parameter());**

**System.out.println(shape.holds(2.5,2.5));**

**System.out.println(shape.holds(new ACircle(3, 5, 5.0)));**

**System.out.println(shape.holds(new ACircle(3, 5, 5.0)));**

**System.out.println("\nFor square:\n");**

**shape = new ASquare(1.5, 1.5, 4.5);**

**System.out.println(shape.Area());**

**System.out.println(shape.Parameter());**

**System.out.println(shape.holds(2.5,2.5));**

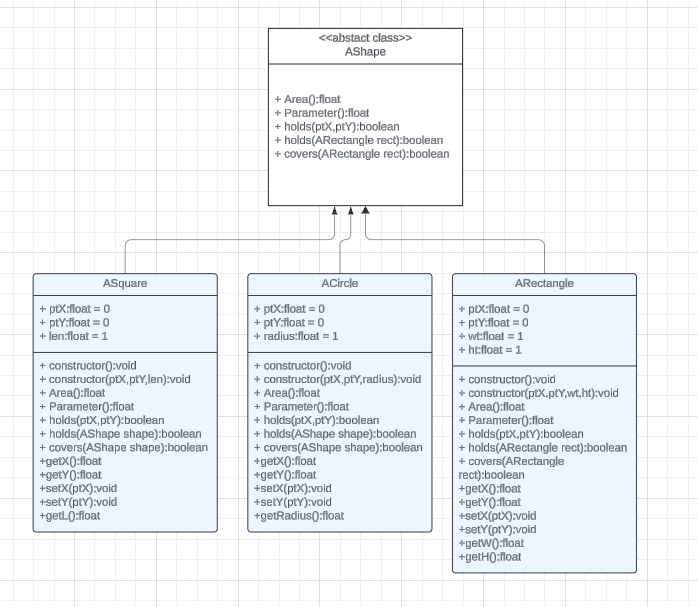
**System.out.println(shape.holds(new ASquare(3, 5, 5.0)));**

**System.out.println(shape.holds(new ASquare(3, 5, 5.0)));**

**}**

**}**

**Q6:**

****

**Q7:**

**Code:**

**public class ACircle implements IShape {**

**private float ptX;**

**private float ptY;**

**private float radius;**

**public ACircle(){**

**this.ptX = 0;**

**this.ptY = 0;**

**this.radius = 1;**

**}**

**public ACircle(float ptX, float ptY, float radius){**

**this.ptX = ptX;**

**this.ptY =ptY;**

**this.radius = radius;**

**}**

**@Override**

**public float Area(){**

**return 3.142\*radius\*radius;**

**}**

**@Override**

**public float Parameter(){**

**return 2\*3.142\*radius;**

**}**

**public float getX(){**

**return ptX;**

**}**

**public float getY(){**

**return ptX;**

**}**

**public void setX(float ptX){**

**this.ptX = ptX;**

**}**

**public void setY(float ptY){**

**this.ptY = ptY;**

**}**

**public float getRadius(){**

**return this.radius;**

**}**

**@Override**

**public boolean holds(float ptX,float ptY){**

**return Math.pow(ptX-this.ptX,2) + Math.pow(ptY - this.ptY,2) <= Math.pow(radius,2);**

**}**

**@Override**

**public boolean holds(IShape shape) {**

**if (shape instanceof ACircle) {**

**ACircle circle = (ACircle) shape;**

**float distanceSquared = (Math.pow(ptX - circle.getX(), 2) +**

**Math.pow(ptY - circle.getY(), 2));**

**float combinedRadius = radius + circle.getRadius();**

**return distanceSquared <= combinedRadius \* combinedRadius;**

**}**

**return false; // For other shapes, return false**

**}**

**@Override**

**public boolean covers(IShape shape) {**

**if (shape instanceof ACircle) {**

**ACircle circle = (ACircle) shape;**

**float distanceSquared = (Math.pow(ptX - circle.getX(), 2) +**

**Math.pow(ptY - circle.getY(), 2));**

**float combinedRadius = radius + circle.getRadius();**

**return distanceSquared <= combinedRadius \* combinedRadius;**

**}**

**return false; // For other shapes, return false**

**}**

**}**

**public class ARectangle implements IShape {**

**private float ptX;**

**private float ptY;**

**private float wt;**

**private float ht;**

**public ARectangle(){**

**this.ptX = 0;**

**this.ptY = 0;**

**this.wt = 1;**

**this.ht = 1;**

**}**

**public ARectangle(float ptX, float ptY, float wt,float ht){**

**this.ptX = ptX;**

**this.ptY =ptY;**

**this.wt = wt;**

**this.ht = ht;**

**}**

**@Override**

**public float Area(){**

**return wt\*ht;**

**}**

**@Override**

**public float Parameter(){**

**return 2\*(ht+wt);**

**}**

**public float getX(){**

**return ptX;**

**}**

**public float getY(){**

**return ptX;**

**}**

**public void setX(float ptX){**

**this.ptX = ptX;**

**}**

**public void setY(float ptY){**

**this.ptY = ptY;**

**}**

**public float getW(){**

**return wt;**

**}**

**public float getH(){**

**return ht;**

**}**

**@Override**

**public boolean holds(float ptX,float ptY){**

**float halfWidth = this.wt / 2;**

**float halfHeight = this.ht / 2;**

**float leftX = this.ptX - halfWidth;**

**float rightX = this.ptX + halfWidth;**

**float topY = this.ptY - halfHeight;**

**float bottomY = this.ptY + halfHeight;**

**return (ptX >= leftX && ptX <= rightX && ptY >= topY && ptY <= bottomY);**

**}**

**@Override**

**public boolean holds (IShape shape){**

**if(shape instanceof ARectangle){**

**ARectangle rect = (ARectangle) shape;**

**float innerHalfHeight = rect.getH() / 2;**

**float innerHalfWidth = rect.getW() / 2;**

**float outerHalfWidth = this.wt / 2;**

**float outerHalfHeight = this.ht / 2;**

**float innerLeftX = rect.getX() - innerHalfWidth;**

**float innerRightX = rect.getX() + innerHalfWidth;**

**float innerTopY = rect.getY() - innerHalfHeight;**

**float innerBottomY = rect.getY() + innerHalfHeight;**

**float outerLeftX = this.ptX - outerHalfWidth;**

**float outerRightX = this.ptX + outerHalfWidth;**

**float outerTopY = this.ptY - outerHalfHeight;**

**float outerBottomY = this.ptY + outerHalfHeight;**

**return (**

**innerLeftX >= outerLeftX &&**

**innerRightX <= outerRightX &&**

**innerTopY >= outerTopY &&**

**innerBottomY <= outerBottomY**

**);**

**}else**

**return false;**

**}**

**@Override**

**public boolean covers (IShape shape){**

**if(shape instanceof ARectangle){**

**ARectangle rect = (ARectangle) shape;**

**float r1LX = this.ptX-this.wt/2;**

**float r1LY = this.ptY+this.ht/2;**

**float r1RX = this.ptX+this.wt/2;**

**float r1RY = this.ptY-this.ht/2;**

**float r2LX = rect.getX()-rect.getW()/2;**

**float r2LY = rect.getY()+rect.getW()/2;**

**float r2RX = rect.getX()+rect.getW()/2;**

**float r2RY = rect.getY()-rect.getW()/2;**

**// if rectangle has area 0, no overlap**

**if (r1LX == r1RX || r1LY == r1RY || r2RX == r2LX || r2LY == r2RY)**

**return false;**

**// If one rectangle is on left side of other**

**if (r1LX > r1RX || r2LX > r1RX)**

**return false;**

**// If one rectangle is above other**

**if (r1RY > r2LY || r2RY > r1LY)**

**return false;**

**return true;**

**}**

**return false;**

**}**

**}**

**public class ASquare implements IShape {**

**private float ptX;**

**private float ptY;**

**private float len;**

**public ASquare(){**

**this.ptX = 0;**

**this.ptY = 0;**

**this.len = 1;**

**}**

**public ASquare(float ptX, float ptY, float len){**

**this.ptX = ptX;**

**this.ptY =ptY;**

**this.len = 1;**

**}**

**@Override**

**public float Area(){**

**return len\*len;**

**}**

**@Override**

**public float Parameter(){**

**return 4\*len;**

**}**

**public float getX(){**

**return ptX;**

**}**

**public float getY(){**

**return ptX;**

**}**

**public void setX(float ptX){**

**this.ptX = ptX;**

**}**

**public void setY(float ptY){**

**this.ptY = ptY;**

**}**

**public float getL(){**

**return len;**

**}**

**@Override**

**public boolean holds(float ptX,float ptY){**

**float halflength = this.len / 2;**

**float leftX = this.ptX - halflength;**

**float rightX = this.ptX + halflength;**

**float topY = this.ptY - halflength;**

**float bottomY = this.ptY + halflength;**

**return (ptX >= leftX && ptX <= rightX && ptY >= topY && ptY <= bottomY);**

**}**

**@Override**

**public boolean holds (IShape shape){**

**if(shape instanceof ASquare){**

**ASquare rect = (ASquare) shape;**

**float innerHalflength = rect.getL() / 2;**

**float outerHalflength = this.len / 2;**

**float innerLeftX = rect.getX() - innerHalflength;**

**float innerRightX = rect.getX() + innerHalflength;**

**float innerTopY = rect.getY() - innerHalflength;**

**float innerBottomY = rect.getY() + innerHalflength;**

**float outerLeftX = this.ptX - outerHalflength;**

**float outerRightX = this.ptX + outerHalflength;**

**float outerTopY = this.ptY - outerHalflength;**

**float outerBottomY = this.ptY + outerHalflength;**

**return (**

**innerLeftX >= outerLeftX &&**

**innerRightX <= outerRightX &&**

**innerTopY >= outerTopY &&**

**innerBottomY <= outerBottomY**

**);**

**}else**

**return false;**

**}**

**@Override**

**public boolean covers (IShape shape){**

**if(shape instanceof ASquare){**

**ASquare rect = (ASquare) shape;**

**float r1LX = this.ptX-this.len/2;**

**float r1LY = this.ptY+this.len/2;**

**float r1RX = this.ptX+this.len/2;**

**float r1RY = this.ptY-this.len/2;**

**float r2LX = rect.getX()-rect.getL()/2;**

**float r2LY = rect.getY()+rect.getL()/2;**

**float r2RX = rect.getX()+rect.getL()/2;**

**float r2RY = rect.getY()-rect.getL()/2;**

**// if rectangle has area 0, no overlap**

**if (r1LX == r1RX || r1LY == r1RY || r2RX == r2LX || r2LY == r2RY)**

**return false;**

**// If one rectangle is on left side of other**

**if (r1LX > r1RX || r2LX > r1RX)**

**return false;**

**// If one rectangle is above other**

**if (r1RY > r2LY || r2RY > r1LY)**

**return false;**

**return true;**

**}**

**return false;**

**}**

**}**

**public interface IShape{**

**public float Area();**

**public float Parameter();**

**public boolean holds (float ptX, float ptY);**

**public boolean holds (IShape shape);**

**public boolean covers (IShape shape);**

**}**

**public class main{**

**public static void main(String[] args){**

**System.out.println("For rectangle:\n");**

**IShape shape = new ARectangle(1.5, 1.5, 5.5, 4.9);**

**System.out.println(shape.Area());**

**System.out.println(shape.Parameter());**

**System.out.println(shape.holds(2.5,2.5));**

**System.out.println(shape.holds(new ARectangle(3, 5, 8.5, 3.5)));**

**System.out.println(shape.holds(new ARectangle(3, 5, 2.3, 5.4)));**

**System.out.println("\nFor circle:\n");**

**shape = new ACircle(1.5, 1.5, 4.5);**

**System.out.println(shape.Area());**

**System.out.println(shape.Parameter());**

**System.out.println(shape.holds(2.5,2.5));**

**System.out.println(shape.holds(new ACircle(3, 5, 5.0)));**

**System.out.println(shape.holds(new ACircle(3, 5, 5.0)));**

**System.out.println("\nFor square:\n");**

**shape = new ASquare(1.5, 1.5, 4.5);**

**System.out.println(shape.Area());**

**System.out.println(shape.Parameter());**

**System.out.println(shape.holds(2.5,2.5));**

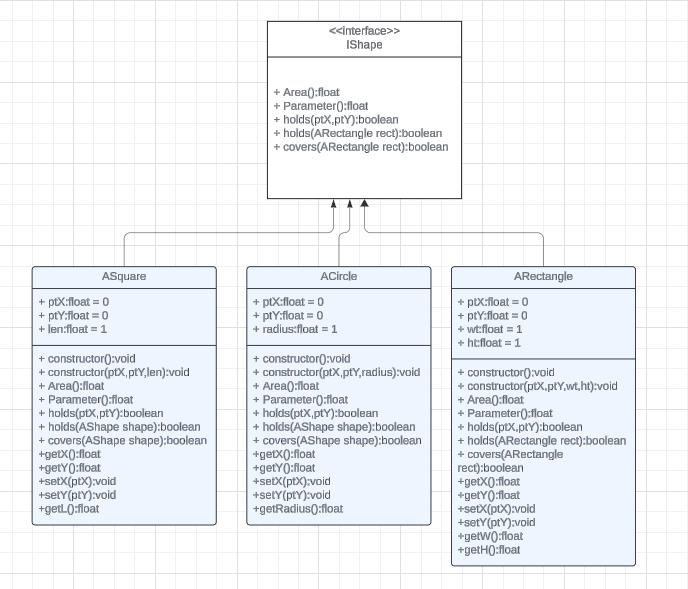
**System.out.println(shape.holds(new ASquare(3, 5, 5.0)));**

**System.out.println(shape.holds(new ASquare(3, 5, 5.0)));**

**}**

**}**

**Q8:**

****

**Q9:**

We could move the setter and getters of the center coordinates to the abstract class