

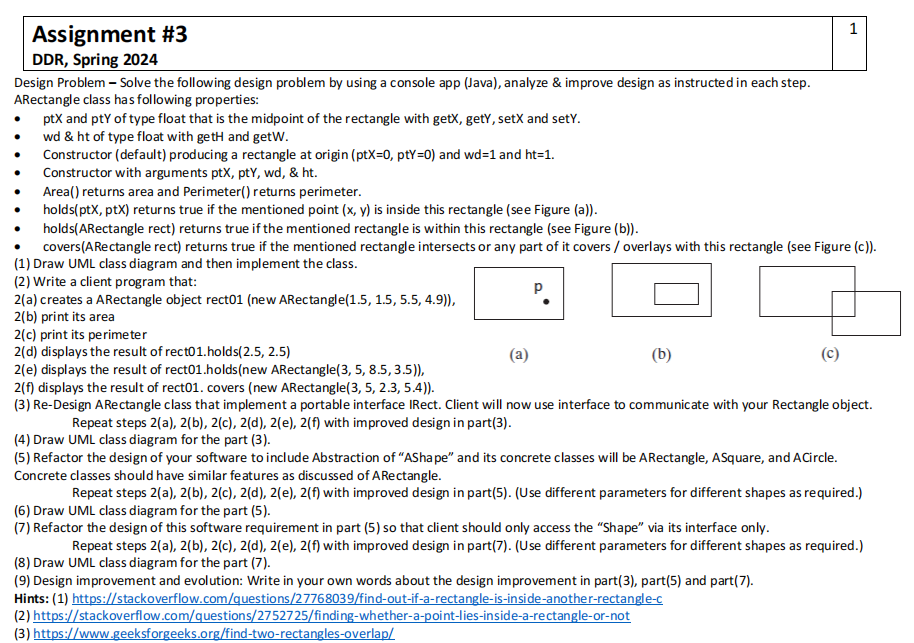
**ASSIGNMENT # 3**

**DESIGN DEFECTS & RESTRUCTURING (BCS-8A)**

**HASSAN ALI**

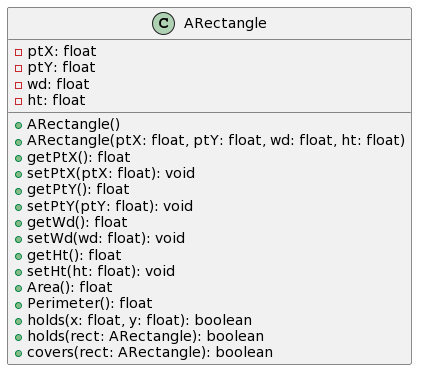
**K20-1052**

**March 20, 2024**



**Question**

**Answer 1:**



**Answer 2:**

**class ARectangle {**

**private float ptX;**

**private float ptY;**

**private float wd;**

**private float ht;**

**public ARectangle() {**

**this.ptX = 0;**

**this.ptY = 0;**

**this.wd = 1;**

**this.ht = 1;**

**}**

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

**this.ptX = ptX;**

**this.ptY = ptY;**

**this.wd = wd;**

**this.ht = ht;**

**}**

**public float getPtX() {**

**return ptX;**

**}**

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

**this.ptX = ptX;**

**}**

**public float getPtY() {**

**return ptY;**

**}**

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

**this.ptY = ptY;**

**}**

**public float getWd() {**

**return wd;**

**}**

**public void setWd(float wd) {**

**this.wd = wd;**

**}**

**public float getHt() {**

**return ht;**

**}**

**public void setHt(float ht) {**

**this.ht = ht;**

**}**

**public float Area() {**

**return wd \* ht;**

**}**

**public float Perimeter() {**

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

**}**

**public boolean holds(float x, float y) {**

**return (x >= ptX - wd / 2 && x <= ptX + wd / 2) && (y >= ptY - ht / 2 && y <= ptY + ht / 2);**

**}**

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

**float x1 = rect.ptX - rect.wd / 2;**

**float x2 = rect.ptX + rect.wd / 2;**

**float y1 = rect.ptY - rect.ht / 2;**

**float y2 = rect.ptY + rect.ht / 2;**

**return (x1 >= ptX - wd / 2 && x2 <= ptX + wd / 2) && (y1 >= ptY - ht / 2 && y2 <= ptY + ht / 2);**

**}**

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

**float x1 = ptX - wd / 2;**

**float x2 = ptX + wd / 2;**

**float y1 = ptY - ht / 2;**

**float y2 = ptY + ht / 2;**

**float rx1 = rect.ptX - rect.wd / 2;**

**float rx2 = rect.ptX + rect.wd / 2;**

**float ry1 = rect.ptY - rect.ht / 2;**

**float ry2 = rect.ptY + rect.ht / 2;**

**return (rx2 >= x1 && rx1 <= x2) && (ry2 >= y1 && ry1 <= y2);**

**}**

**}**

**public class main {**

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

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

**System.*out*.println("Area of rect01: " + rect01.Area());**

**System.*out*.println("Perimeter of rect01: " + rect01.Perimeter());**

**System.*out*.println("Does rect01 hold point (2.5, 2.5)? " + rect01.holds(2.5f, 2.5f));**

**ARectangle rect02 = new ARectangle(3, 5, 8.5f, 3.5f);**

**System.*out*.println("Does rect01 hold rect02? " + rect01.holds(rect02));**

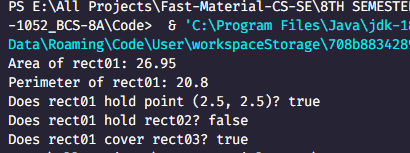
**ARectangle rect03 = new ARectangle(3, 5, 2.3f, 5.4f);**

**System.*out*.println("Does rect01 cover rect03? " + rect01.covers(rect03));**

**}**

}

Output:



**Answer 3:**

**interface IRect {**

**float Area();**

**float Perimeter();**

**boolean holds(float x, float y);**

**boolean holds(ARectangle rect);**

**boolean covers(ARectangle rect);**

**}**

**class ARectangle implements IRect {**

**private float ptX;**

**private float ptY;**

**private float wd;**

**private float ht;**

**public ARectangle() {**

**this.ptX = 0;**

**this.ptY = 0;**

**this.wd = 1;**

**this.ht = 1;**

**}**

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

**this.ptX = ptX;**

**this.ptY = ptY;**

**this.wd = wd;**

**this.ht = ht;**

**}**

**public float getPtX() {**

**return ptX;**

**}**

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

**this.ptX = ptX;**

**}**

**public float getPtY() {**

**return ptY;**

**}**

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

**this.ptY = ptY;**

**}**

**public float getWd() {**

**return wd;**

**}**

**public void setWd(float wd) {**

**this.wd = wd;**

**}**

**public float getHt() {**

**return ht;**

**}**

**public void setHt(float ht) {**

**this.ht = ht;**

**}**

***@Override***

**public float Area() {**

**return wd \* ht;**

**}**

***@Override***

**public float Perimeter() {**

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

**}**

***@Override***

**public boolean holds(float x, float y) {**

**return (x >= ptX - wd / 2 && x <= ptX + wd / 2) && (y >= ptY - ht / 2 && y <= ptY + ht / 2);**

**}**

***@Override***

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

**float x1 = rect.ptX - rect.wd / 2;**

**float x2 = rect.ptX + rect.wd / 2;**

**float y1 = rect.ptY - rect.ht / 2;**

**float y2 = rect.ptY + rect.ht / 2;**

**return (x1 >= ptX - wd / 2 && x2 <= ptX + wd / 2) && (y1 >= ptY - ht / 2 && y2 <= ptY + ht / 2);**

**}**

***@Override***

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

**float x1 = ptX - wd / 2;**

**float x2 = ptX + wd / 2;**

**float y1 = ptY - ht / 2;**

**float y2 = ptY + ht / 2;**

**float rx1 = rect.ptX - rect.wd / 2;**

**float rx2 = rect.ptX + rect.wd / 2;**

**float ry1 = rect.ptY - rect.ht / 2;**

**float ry2 = rect.ptY + rect.ht / 2;**

**return (rx2 >= x1 && rx1 <= x2) && (ry2 >= y1 && ry1 <= y2);**

**}**

**}**

**public class main {**

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

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

**System.*out*.println("Area of rect01: " + rect01.Area());**

**System.*out*.println("Perimeter of rect01: " + rect01.Perimeter());**

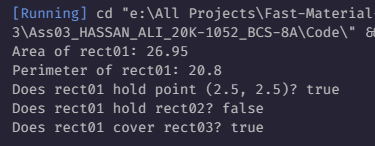
**System.*out*.println("Does rect01 hold point (2.5, 2.5)? " + rect01.holds(2.5f, 2.5f));**

**ARectangle rect02 = new ARectangle(3, 5, 8.5f, 3.5f);**

**System.*out*.println("Does rect01 hold rect02? " + rect01.holds(rect02));**

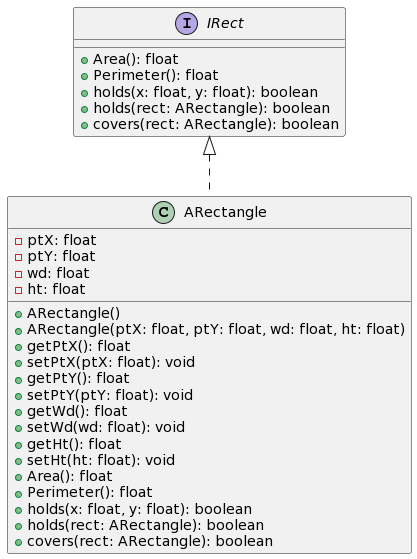
**ARectangle rect03 = new ARectangle(3, 5, 2.3f, 5.4f);**

**System.*out*.println("Does rect01 cover rect03? " + rect01.covers(rect03));**

Output:  


}

**Answer 4:**



**Answer 5:**

**abstract class AShape {**

**protected float ptX;**

**protected float ptY;**

**public abstract float Area();**

**public abstract float Perimeter();**

**public abstract boolean holds(float x, float y);**

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

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

**public float getPtX() {**

**return ptX;**

**}**

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

**this.ptX = ptX;**

**}**

**public float getPtY() {**

**return ptY;**

**}**

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

**this.ptY = ptY;**

**}**

**}**

**class ARectangle extends AShape {**

**private float wd;**

**private float ht;**

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

**this.ptX = ptX;**

**this.ptY = ptY;**

**this.wd = wd;**

**this.ht = ht;**

**}**

**public float getWd() {**

**return wd;**

**}**

**public void setWd(float wd) {**

**this.wd = wd;**

**}**

**public float getHt() {**

**return ht;**

**}**

**public void setHt(float ht) {**

**this.ht = ht;**

**}**

***@Override***

**public float Area() {**

**return wd \* ht;**

**}**

***@Override***

**public float Perimeter() {**

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

**}**

***@Override***

**public boolean holds(float x, float y) {**

**return (x >= ptX - wd / 2 && x <= ptX + wd / 2) && (y >= ptY - ht / 2 && y <= ptY + ht / 2);**

**}**

***@Override***

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

**if (shape instanceof ARectangle) {**

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

**float x1 = rect.ptX - rect.wd / 2;**

**float x2 = rect.ptX + rect.wd / 2;**

**float y1 = rect.ptY - rect.ht / 2;**

**float y2 = rect.ptY + rect.ht / 2;**

**return (x1 >= ptX - wd / 2 && x2 <= ptX + wd / 2) && (y1 >= ptY - ht / 2 && y2 <= ptY + ht / 2);**

**}**

**return false;**

**}**

***@Override***

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

**if (shape instanceof ARectangle) {**

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

**float x1 = ptX - wd / 2;**

**float x2 = ptX + wd / 2;**

**float y1 = ptY - ht / 2;**

**float y2 = ptY + ht / 2;**

**float rx1 = rect.ptX - rect.wd / 2;**

**float rx2 = rect.ptX + rect.wd / 2;**

**float ry1 = rect.ptY - rect.ht / 2;**

**float ry2 = rect.ptY + rect.ht / 2;**

**return (rx2 >= x1 && rx1 <= x2) && (ry2 >= y1 && ry1 <= y2);**

**}**

**return false;**

**}**

**}**

**class ASquare extends AShape {**

**private float side;**

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

**this.ptX = ptX;**

**this.ptY = ptY;**

**this.side = side;**

**}**

**public float getSide() {**

**return side;**

**}**

**public void setSide(float side) {**

**this.side = side;**

**}**

***@Override***

**public float Area() {**

**return side \* side;**

**}**

***@Override***

**public float Perimeter() {**

**return 4 \* side;**

**}**

***@Override***

**public boolean holds(float x, float y) {**

**return (x >= ptX - side / 2 && x <= ptX + side / 2) && (y >= ptY - side / 2 && y <= ptY + side / 2);**

**}**

***@Override***

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

**if (shape instanceof ASquare) {**

**ASquare square = (ASquare) shape;**

**float x1 = square.ptX - square.side / 2;**

**float x2 = square.ptX + square.side / 2;**

**float y1 = square.ptY - square.side / 2;**

**float y2 = square.ptY + square.side / 2;**

**return (x1 >= ptX - side / 2 && x2 <= ptX + side / 2) && (y1 >= ptY - side / 2 && y2 <= ptY + side / 2);**

**}**

**return false;**

**}**

***@Override***

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

**if (shape instanceof ASquare) {**

**ASquare square = (ASquare) shape;**

**float x1 = ptX - side / 2;**

**float x2 = ptX + side / 2;**

**float y1 = ptY - side / 2;**

**float y2 = ptY + side / 2;**

**float sx1 = square.ptX - square.side / 2;**

**float sx2 = square.ptX + square.side / 2;**

**float sy1 = square.ptY - square.side / 2;**

**float sy2 = square.ptY + square.side / 2;**

**return (sx2 >= x1 && sx1 <= x2) && (sy2 >= y1 && sy1 <= y2);**

**}**

**return false;**

**}**

**}**

**class ACircle extends AShape {**

**private float radius;**

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

**this.ptX = ptX;**

**this.ptY = ptY;**

**this.radius = radius;**

**}**

**public float getRadius() {**

**return radius;**

**}**

**public void setRadius(float radius) {**

**this.radius = radius;**

**}**

***@Override***

**public float Area() {**

**return (float) (Math.*PI* \* radius \* radius);**

**}**

***@Override***

**public float Perimeter() {**

**return (float) (2 \* Math.*PI* \* radius);**

**}**

***@Override***

**public boolean holds(float x, float y) {**

**return Math.*sqrt*((x - ptX) \* (x - ptX) + (y - ptY) \* (y - ptY)) <= radius;**

**}**

***@Override***

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

**if (shape instanceof ACircle) {**

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

**float distanceBetweenCenters = (float) Math.*sqrt*((circle.ptX - ptX) \* (circle.ptX - ptX)**

**+ (circle.ptY - ptY) \* (circle.ptY - ptY));**

**return distanceBetweenCenters + circle.radius <= radius;**

**}**

**return false;**

**}**

***@Override***

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

**if (shape instanceof ACircle) {**

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

**float distanceBetweenCenters = (float) Math.*sqrt*((circle.ptX - ptX) \* (circle.ptX - ptX)**

**+ (circle.ptY - ptY) \* (circle.ptY - ptY));**

**return distanceBetweenCenters <= radius + circle.radius;**

**}**

**return false;**

**}**

**}**

**public class main {**

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

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

**AShape square = new ASquare(0, 0, 4.0f);**

**AShape circle = new ACircle(2.0f, 2.0f, 3.0f);**

**// Step 2(a) - Create ARectangle**

**System.*out*.println("Step 2(a): Created ARectangle");**

**// Step 2(b) - Print Area of rect01**

**System.*out*.println("Step 2(b): Area of rect01: " + rect01.Area());**

**// Step 2(c) - Print Perimeter of rect01**

**System.*out*.println("Step 2(c): Perimeter of rect01: " + rect01.Perimeter());**

**// Step 2(d) - Display holds(2.5, 2.5) for rect01**

**System.*out*.println("Step 2(d): Does rect01 hold point (2.5, 2.5)? " + rect01.holds(2.5f, 2.5f));**

**// Step 2(e) - Display holds(new ARectangle(3, 5, 8.5, 3.5)) for rect01**

**AShape rect02 = new ARectangle(3, 5, 8.5f, 3.5f);**

**System.*out*.println("Step 2(e): Does rect01 hold rect02? " + rect01.holds(rect02));**

**// Step 2(f) - Display covers(new ARectangle(3, 5, 2.3, 5.4)) for rect01**

**AShape rect03 = new ARectangle(3, 5, 2.3f, 5.4f);**

**System.*out*.println("Step 2(f): Does rect01 cover rect03? " + rect01.covers(rect03));**

**// Additional Testing**

**System.*out*.println("\nAdditional Testing:");**

**System.*out*.println("Area of Square: " + square.Area());**

**System.*out*.println("Perimeter of Square: " + square.Perimeter());**

**System.*out*.println("Does square hold point (1, 1)? " + square.holds(1, 1));**

**System.*out*.println("Does square hold rect02? " + square.holds(rect02));**

**System.*out*.println("Does square cover rect03? " + square.covers(rect03));**

**System.*out*.println("\nArea of Circle: " + circle.Area());**

**System.*out*.println("Perimeter of Circle: " + circle.Perimeter());**

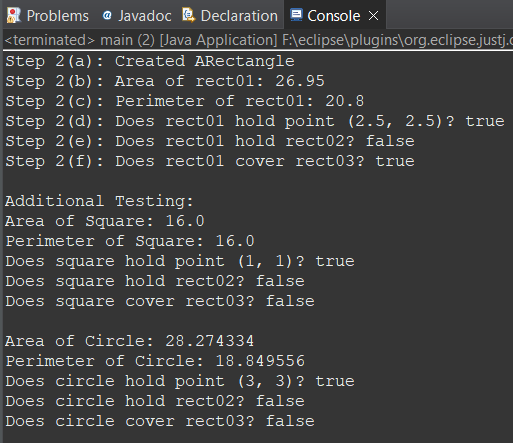
**System.*out*.println("Does circle hold point (3, 3)? " + circle.holds(3, 3));**

**System.*out*.println("Does circle hold rect02? " + circle.holds(rect02));**

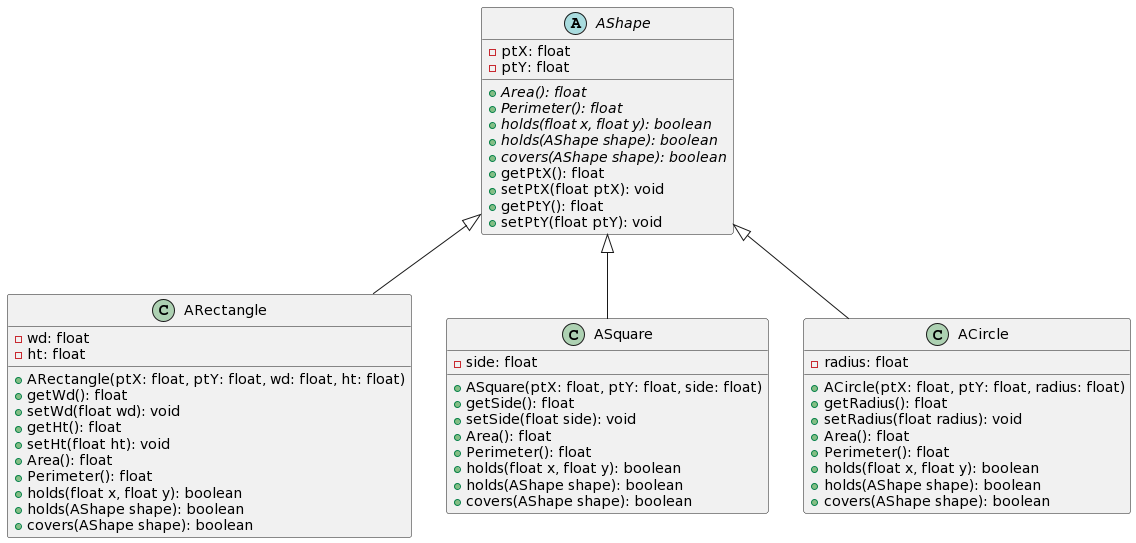
**System.*out*.println("Does circle cover rect03? " + circle.covers(rect03));**

**}**

**}**

Output:  


**Answer 6:**



**Answer 7:**

**interface Shape {**

**float Area();**

**float Perimeter();**

**boolean holds(float x, float y);**

**boolean holds(Shape shape);**

**boolean covers(Shape shape);**

**}**

**class Rectangle implements Shape {**

**private float ptX;**

**private float ptY;**

**private float wd;**

**private float ht;**

**public Rectangle(float ptX, float ptY, float wd, float ht) {**

**this.ptX = ptX;**

**this.ptY = ptY;**

**this.wd = wd;**

**this.ht = ht;**

**}**

**public float getWd() {**

**return wd;**

**}**

**public void setWd(float wd) {**

**this.wd = wd;**

**}**

**public float getHt() {**

**return ht;**

**}**

**public void setHt(float ht) {**

**this.ht = ht;**

**}**

***@Override***

**public float Area() {**

**return wd \* ht;**

**}**

***@Override***

**public float Perimeter() {**

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

**}**

***@Override***

**public boolean holds(float x, float y) {**

**return (x >= ptX - wd / 2 && x <= ptX + wd / 2) && (y >= ptY - ht / 2 && y <= ptY + ht / 2);**

**}**

***@Override***

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

**if (shape instanceof Rectangle) {**

**Rectangle rect = (Rectangle) shape;**

**float x1 = rect.ptX - rect.wd / 2;**

**float x2 = rect.ptX + rect.wd / 2;**

**float y1 = rect.ptY - rect.ht / 2;**

**float y2 = rect.ptY + rect.ht / 2;**

**return (x1 >= ptX - wd / 2 && x2 <= ptX + wd / 2) && (y1 >= ptY - ht / 2 && y2 <= ptY + ht / 2);**

**}**

**return false;**

**}**

***@Override***

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

**if (shape instanceof Rectangle) {**

**Rectangle rect = (Rectangle) shape;**

**float x1 = ptX - wd / 2;**

**float x2 = ptX + wd / 2;**

**float y1 = ptY - ht / 2;**

**float y2 = ptY + ht / 2;**

**float rx1 = rect.ptX - rect.wd / 2;**

**float rx2 = rect.ptX + rect.wd / 2;**

**float ry1 = rect.ptY - rect.ht / 2;**

**float ry2 = rect.ptY + rect.ht / 2;**

**return (rx2 >= x1 && rx1 <= x2) && (ry2 >= y1 && ry1 <= y2);**

**}**

**return false;**

**}**

**}**

**class Square implements Shape {**

**private float ptX;**

**private float ptY;**

**private float side;**

**public Square(float ptX, float ptY, float side) {**

**this.ptX = ptX;**

**this.ptY = ptY;**

**this.side = side;**

**}**

**public float getSide() {**

**return side;**

**}**

**public void setSide(float side) {**

**this.side = side;**

**}**

***@Override***

**public float Area() {**

**return side \* side;**

**}**

***@Override***

**public float Perimeter() {**

**return 4 \* side;**

**}**

***@Override***

**public boolean holds(float x, float y) {**

**return (x >= ptX - side / 2 && x <= ptX + side / 2) && (y >= ptY - side / 2 && y <= ptY + side / 2);**

**}**

***@Override***

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

**if (shape instanceof Square) {**

**Square square = (Square) shape;**

**float x1 = square.ptX - square.side / 2;**

**float x2 = square.ptX + square.side / 2;**

**float y1 = square.ptY - square.side / 2;**

**float y2 = square.ptY + square.side / 2;**

**return (x1 >= ptX - side / 2 && x2 <= ptX + side / 2) && (y1 >= ptY - side / 2 && y2 <= ptY + side / 2);**

**}**

**return false;**

**}**

***@Override***

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

**if (shape instanceof Square) {**

**Square square = (Square) shape;**

**float x1 = ptX - side / 2;**

**float x2 = ptX + side / 2;**

**float y1 = ptY - side / 2;**

**float y2 = ptY + side / 2;**

**float sx1 = square.ptX - square.side / 2;**

**float sx2 = square.ptX + square.side / 2;**

**float sy1 = square.ptY - square.side / 2;**

**float sy2 = square.ptY + square.side / 2;**

**return (sx2 >= x1 && sx1 <= x2) && (sy2 >= y1 && sy1 <= y2);**

**}**

**return false;**

**}**

**}**

**class Circle implements Shape {**

**private float ptX;**

**private float ptY;**

**private float radius;**

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

**this.ptX = ptX;**

**this.ptY = ptY;**

**this.radius = radius;**

**}**

**public float getRadius() {**

**return radius;**

**}**

**public void setRadius(float radius) {**

**this.radius = radius;**

**}**

***@Override***

**public float Area() {**

**return (float) (Math.*PI* \* radius \* radius);**

**}**

***@Override***

**public float Perimeter() {**

**return (float) (2 \* Math.*PI* \* radius);**

**}**

***@Override***

**public boolean holds(float x, float y) {**

**return Math.*sqrt*((x - ptX) \* (x - ptX) + (y - ptY) \* (y - ptY)) <= radius;**

**}**

***@Override***

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

**if (shape instanceof Circle) {**

**Circle circle = (Circle) shape;**

**float distanceBetweenCenters = (float) Math.*sqrt*((circle.ptX - ptX) \* (circle.ptX - ptX)**

**+ (circle.ptY - ptY) \* (circle.ptY - ptY));**

**return distanceBetweenCenters + circle.radius <= radius;**

**}**

**return false;**

**}**

***@Override***

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

**if (shape instanceof Circle) {**

**Circle circle = (Circle) shape;**

**float distanceBetweenCenters = (float) Math.*sqrt*((circle.ptX - ptX) \* (circle.ptX - ptX)**

**+ (circle.ptY - ptY) \* (circle.ptY - ptY));**

**return distanceBetweenCenters <= radius + circle.radius;**

**}**

**return false;**

**}**

**}**

**public class main {**

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

**Shape rect01 = new Rectangle(1.5f, 1.5f, 5.5f, 4.9f);**

**Shape square = new Square(0, 0, 4.0f);**

**Shape circle = new Circle(2.0f, 2.0f, 3.0f);**

**// Step 2(a) - Create Rectangle**

**System.*out*.println("Step 2(a): Created Rectangle");**

**// Step 2(b) - Print Area of rect01**

**System.*out*.println("Step 2(b): Area of rect01: " + rect01.Area());**

**// Step 2(c) - Print Perimeter of rect01**

**System.*out*.println("Step 2(c): Perimeter of rect01: " + rect01.Perimeter());**

**// Step 2(d) - Display holds(2.5, 2.5) for rect01**

**System.*out*.println("Step 2(d): Does rect01 hold point (2.5, 2.5)? " + rect01.holds(2.5f, 2.5f));**

**// Step 2(e) - Display holds(new Rectangle(3, 5, 8.5, 3.5)) for rect01**

**Shape rect02 = new Rectangle(3, 5, 8.5f, 3.5f);**

**System.*out*.println("Step 2(e): Does rect01 hold rect02? " + rect01.holds(rect02));**

**// Step 2(f) - Display covers(new Rectangle(3, 5, 2.3, 5.4)) for rect01**

**Shape rect03 = new Rectangle(3, 5, 2.3f, 5.4f);**

**System.*out*.println("Step 2(f): Does rect01 cover rect03? " + rect01.covers(rect03));**

**// Additional Testing**

**System.*out*.println("\nAdditional Testing:");**

**System.*out*.println("Area of Square: " + square.Area());**

**System.*out*.println("Perimeter of Square: " + square.Perimeter());**

**System.*out*.println("Does square hold point (1, 1)? " + square.holds(1, 1));**

**System.*out*.println("Does square hold rect02? " + square.holds(rect02));**

**System.*out*.println("Does square cover rect03? " + square.covers(rect03));**

**System.*out*.println("\nArea of Circle: " + circle.Area());**

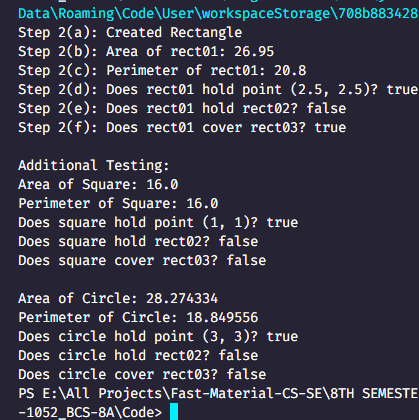
**System.*out*.println("Perimeter of Circle: " + circle.Perimeter());**

**System.*out*.println("Does circle hold point (3, 3)? " + circle.holds(3, 3));**

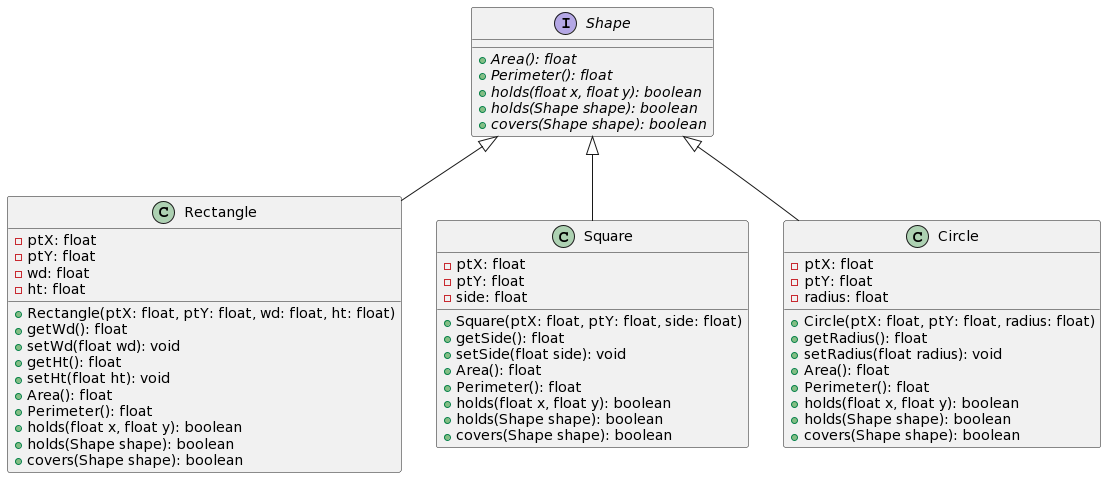
**System.*out*.println("Does circle hold rect02? " + circle.holds(rect02));**

**System.*out*.println("Does circle cover rect03? " + circle.covers(rect03));**

**}**

Output:  


**Answer 8:**

****

**Answer 9:**

**In part 3, The introduction of the portable interface IRect for the ARectangle class establishes a clear contract for methods accessible to the client, facilitating loose coupling and effortless interchangeability of implementations. This approach enhances code readability and maintainability by effectively separating concerns within the software design.**

**In part 5, By introducing the abstraction layer AShape, we've created a handy common interface that all shapes can follow. This means we're sticking to the inheritance principle, which helps us reuse code and keep things modular. It also sets up a nice structure for adding new shapes down the road without much fuss. Plus, it makes our code easier to maintain and read, all because we're focusing on what shapes do rather than how each one is specifically implemented. It's like giving our shapes their own dance moves but letting them all follow the same beat.**

**In part 7, The adherence to the "programming to an interface" principle in this design offers a significant advantage in terms of flexibility, providing a framework that allows for a wide range of implementations. This approach facilitates the seamless swapping of implementations without necessitating modifications to the client code, thus promoting a high degree of modularity. Moreover, the establishment of a clear and standardized contract for all shapes through the interface greatly enhances the readability and maintainability of the codebase. By enabling the use of polymorphism, the design allows for a more extensible and adaptable code structure, capable of accommodating future changes and expansions with ease while maintaining a structured and organized architecture.**