

**Design Defects and Refactoring**

**CS-441**

**Assignment No. 2**

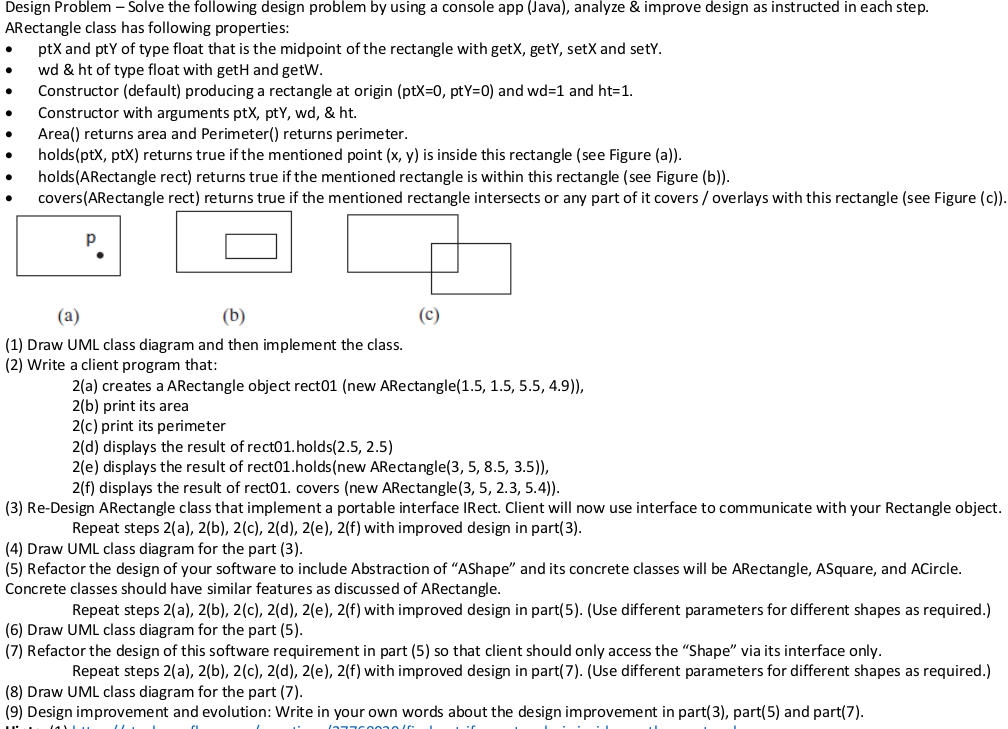
**Design Pattern**

**Armughan Ahmed**

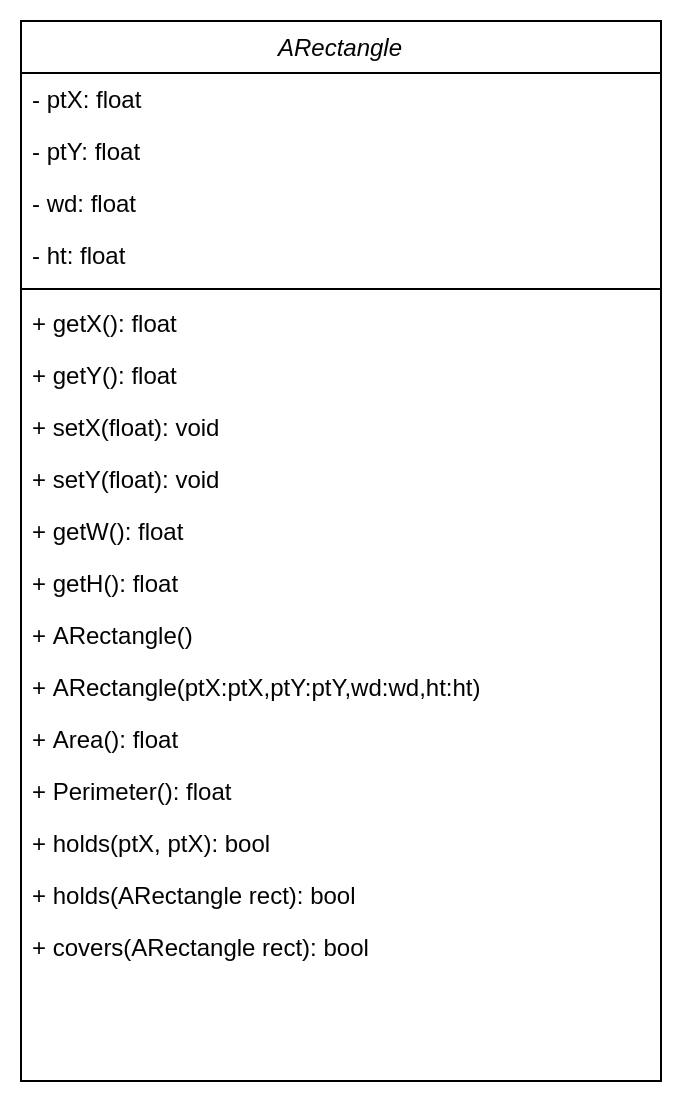
**17K-3696**

**SEC-BCS 9A**

**Submission Date: 22th March 2021**



**Question 1:**



**Code:**

public class ARectangle {

private float ptX;

private float ptY;

private float wd;

private float ht;

public ARectangle() {

ptX=0;

ptY=0;

wd=1;

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 Area() {

return wd\*ht;

}

public float Perimeter() {

return 2\*(wd+ht);

}

public boolean holds(float ptX, float ptY) {

return Math.sqrt(Math.pow(this.ptY - ptY, 2)) <= ht / 2 && Math.sqrt(Math.pow(this.ptX - ptX, 2)) <= wd / 2;

}

public boolean holds(ARectangle rect) {

return Math.sqrt(Math.pow(ptY - rect.getPtY(), 2))+ rect.getHt() / 2 <= ht / 2

&& Math.sqrt(Math.pow(ptX - rect.getPtX(), 2)) + rect.getWd() / 2 <= wd / 2 && ht / 2 + rect.getHt() / 2 <= ht

&& wd / 2 + rect.getWd() / 2 <= wd;

}

public boolean covers(ARectangle rect) {

return !holds(rect) && ((ptX + wd / 2 > rect.getPtX() - rect.getWd()) || (ptY + ht / 2 > rect.getPtY() - rect.getHt()))

&& (Math.sqrt(Math.pow(ptY - rect.getPtY(), 2)) < ht / 2 + rect.getHt() / 2)

&& (Math.sqrt(Math.pow(ptX - rect.getPtX(), 2))< wd / 2 + rect.getWd() / 2);

}

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 static void main(String[] args) {

}

}

**Question 2:**

public class ARectangle {

private double ptX;

private double ptY;

private double wd;

private double ht;

public ARectangle() {

ptX=0;

ptY=0;

wd=1;

ht=1;

}

public ARectangle(double ptX,double ptY,double wd,double ht) {

this.ptX=ptX;

this.ptY=ptY;

this.wd=wd;

this.ht=ht;

}

public double Area() {

return wd\*ht;

}

public double Perimeter() {

return 2\*(wd+ht);

}

public boolean holds(double ptX, double ptY) {

return Math.sqrt(Math.pow(this.ptY - ptY, 2)) <= ht / 2 && Math.sqrt(Math.pow(this.ptX - ptX, 2)) <= wd / 2;

}

public boolean holds(ARectangle rect) {

return Math.sqrt(Math.pow(ptY - rect.getPtY(), 2))+ rect.getHt() / 2 <= ht / 2

&& Math.sqrt(Math.pow(ptX - rect.getPtX(), 2)) + rect.getWd() / 2 <= wd / 2 && ht / 2 + rect.getHt() / 2 <= ht

&& wd / 2 + rect.getWd() / 2 <= wd;

}

public boolean covers(ARectangle rect) {

return !holds(rect) && ((ptX + wd / 2 > rect.getPtX() - rect.getWd()) || (ptY + ht / 2 > rect.getPtY() - rect.getHt()))

&& (Math.sqrt(Math.pow(ptY - rect.getPtY(), 2)) < ht / 2 + rect.getHt() / 2)

&& (Math.sqrt(Math.pow(ptX - rect.getPtX(), 2))< wd / 2 + rect.getWd() / 2);

}

public double getPtX() {

return ptX;

}

public void setPtX(double ptX) {

this.ptX = ptX;

}

public double getPtY() {

return ptY;

}

public void setPtY(double ptY) {

this.ptY = ptY;

}

public double getWd() {

return wd;

}

public void setWd(double wd) {

this.wd = wd;

}

public double getHt() {

return ht;

}

public void setHt(double ht) {

this.ht = ht;

}

public static void main(String[] args) {

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

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

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

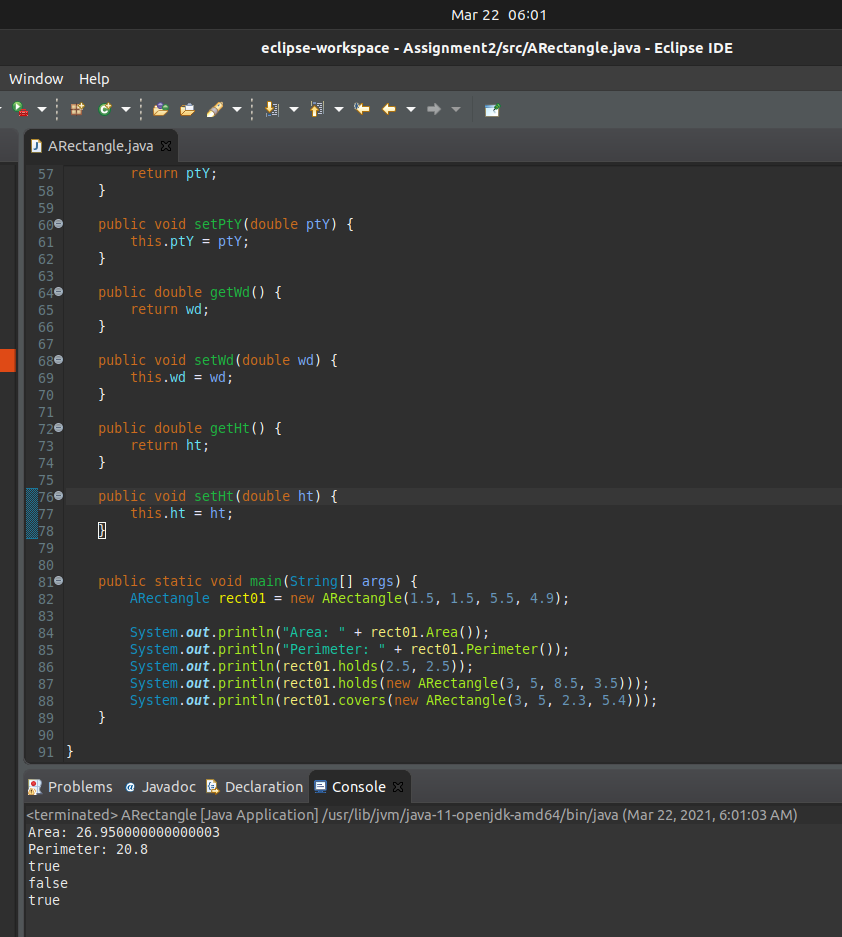
System.out.println(rect01.holds(2.5, 2.5));

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

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

}

}



**Question 3:**

public class IRect {

public static void main(String[] args) {

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

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

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

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.covers(new ARectangle(3, 5, 2.3, 5.4)));

}

}

public class ARectangle {

private double ptX;

private double ptY;

private double wd;

private double ht;

public ARectangle() {

ptX=0;

ptY=0;

wd=1;

ht=1;

}

public ARectangle(double ptX,double ptY,double wd,double ht) {

this.ptX=ptX;

this.ptY=ptY;

this.wd=wd;

this.ht=ht;

}

public double Area() {

return wd\*ht;

}

public double Perimeter() {

return 2\*(wd+ht);

}

public boolean holds(double ptX, double ptY) {

return Math.sqrt(Math.pow(this.ptY - ptY, 2)) <= ht / 2 && Math.sqrt(Math.pow(this.ptX - ptX, 2)) <= wd / 2;

}

public boolean holds(ARectangle rect) {

return Math.sqrt(Math.pow(ptY - rect.getPtY(), 2))+ rect.getHt() / 2 <= ht / 2

&& Math.sqrt(Math.pow(ptX - rect.getPtX(), 2)) + rect.getWd() / 2 <= wd / 2 && ht / 2 + rect.getHt() / 2 <= ht

&& wd / 2 + rect.getWd() / 2 <= wd;

}

public boolean covers(ARectangle rect) {

return !holds(rect) && ((ptX + wd / 2 > rect.getPtX() - rect.getWd()) || (ptY + ht / 2 > rect.getPtY() - rect.getHt()))

&& (Math.sqrt(Math.pow(ptY - rect.getPtY(), 2)) < ht / 2 + rect.getHt() / 2)

&& (Math.sqrt(Math.pow(ptX - rect.getPtX(), 2))< wd / 2 + rect.getWd() / 2);

}

public double getPtX() {

return ptX;

}

public void setPtX(double ptX) {

this.ptX = ptX;

}

public double getPtY() {

return ptY;

}

public void setPtY(double ptY) {

this.ptY = ptY;

}

public double getWd() {

return wd;

}

public void setWd(double wd) {

this.wd = wd;

}

public double getHt() {

return ht;

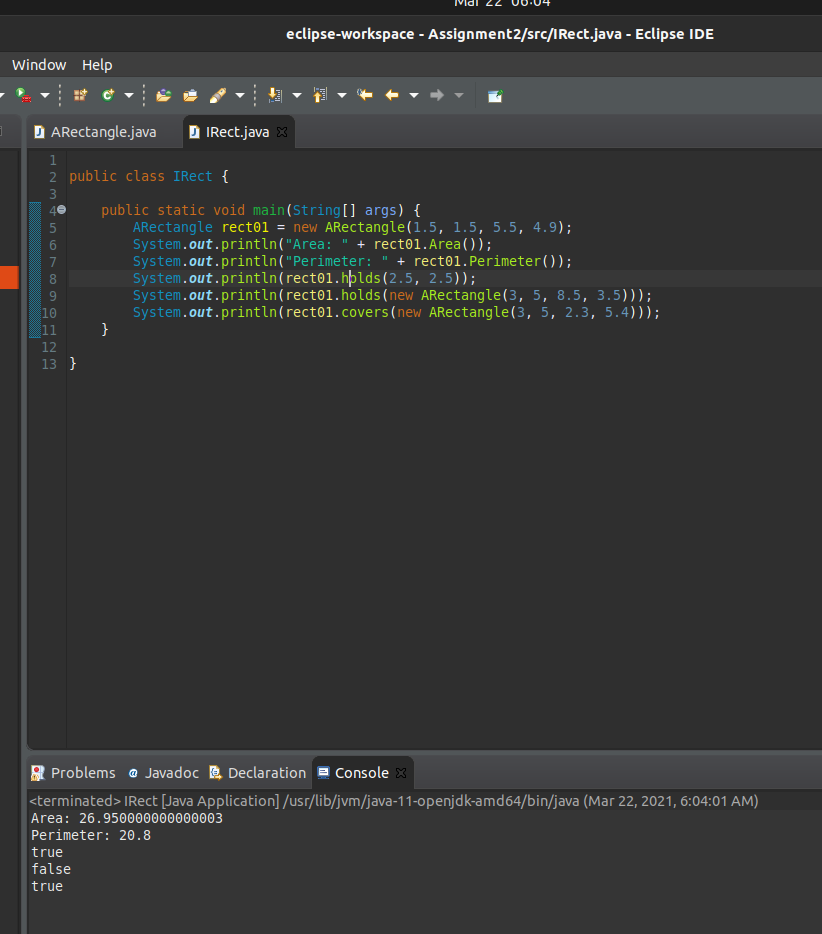
}

public void setHt(double ht) {

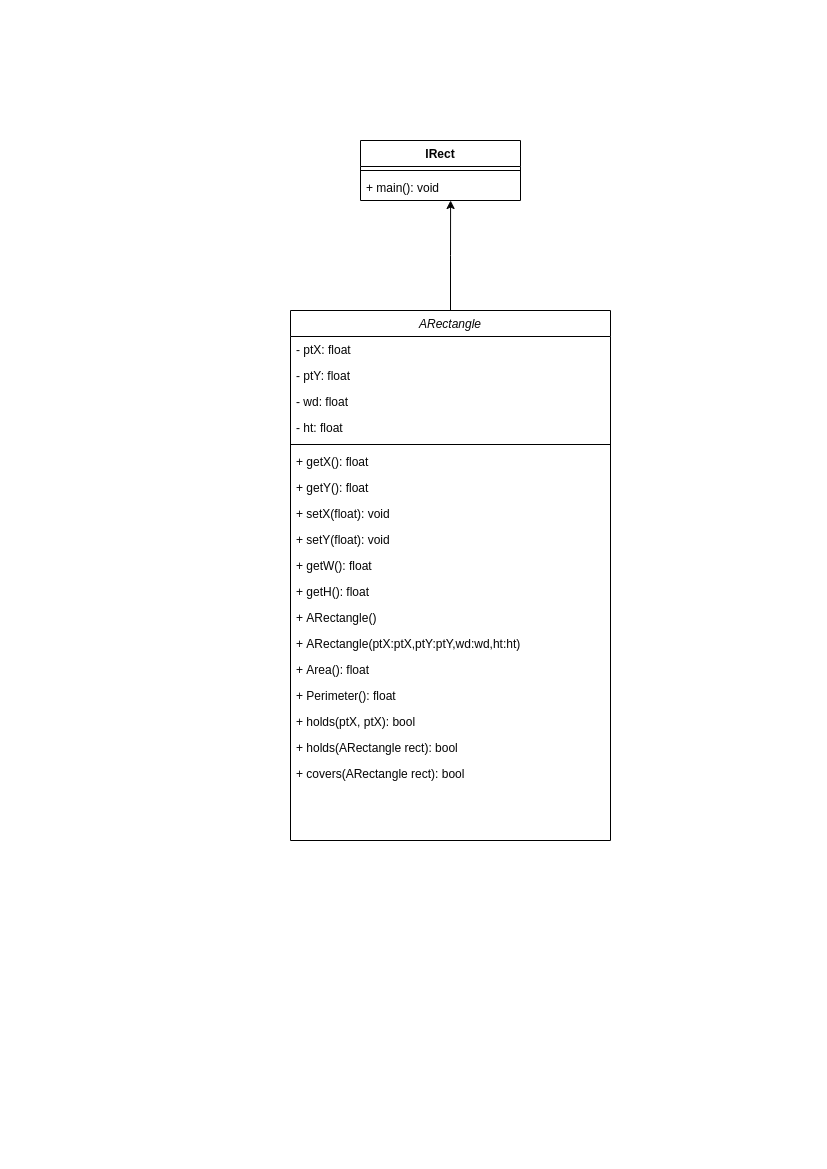
this.ht = ht;

}

}



**Question 4:**



**Question 5:**

public class ASquare extends AShape{

private double area;

public ASquare() {

ptX=0;

ptY=0;

this.area=1;

}

public ASquare(double ptX, double ptY, double area) {

this.ptX=ptX;

this.ptY=ptY;

this.area=area;

}

@Override

public double Area() {

return Math.pow(area, 2);

}

@Override

public double Perimeter() {

return 4 \* area;

}

@Override

public boolean holds(double ptX, double ptY) {

return Math.sqrt(Math.pow(this.ptY - ptY, 2))<= area / 2 && Math.sqrt(Math.pow(this.ptX - ptX, 2))<= area / 2;

}

public boolean holds(ASquare s) {

return Math.sqrt(Math.pow(ptY - s.getPtY(), 2)) + s.Area() / 2 <= area / 2

&& Math.sqrt(Math.pow(ptX - s.Area(), 2)) + s.Area() / 2 <= area / 2 && area / 2 + s.Area() / 2 <= area

&& area / 2 + s.Area() / 2 <= area;

}

public boolean covers(ASquare s) {

return !holds(s) && ((ptX + area / 2 > s.getPtX() - s.Area()) || (ptY + area / 2 > s.getPtY() - s.Area()))

&& (Math.sqrt(Math.pow(ptY - s.getPtY(), 2)) < area / 2 + s.Area() / 2)

&& (Math.sqrt(Math.pow(ptX - s.getPtX(), 2))< area / 2 + s.Area() / 2);

}

public double getArea() {

return area;

}

public void setArea(double area) {

this.area = area;

}

}

public abstract class AShape {

public double ptX;

public double ptY;

public abstract double Area();

public abstract double Perimeter();

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

public double getPtX() {

return ptX;

}

public void setPtX(double ptX) {

this.ptX = ptX;

}

public double getPtY() {

return ptY;

}

public void setPtY(double ptY) {

this.ptY = ptY;

}

}

public class ARectangle extends AShape {

private double wd;

private double ht;

public ARectangle() {

ptX=0;

ptY=0;

wd=1;

ht=1;

}

public ARectangle(double ptX,double ptY,double wd,double ht) {

this.ptX=ptX;

this.ptY=ptY;

this.wd=wd;

this.ht=ht;

}

@Override

public double Area() {

return wd\*ht;

}

@Override

public double Perimeter() {

return 2\*(wd+ht);

}

@Override

public boolean holds(double ptX, double ptY) {

return Math.sqrt(Math.pow(this.ptY - ptY, 2)) <= ht / 2 && Math.sqrt(Math.pow(this.ptX - ptX, 2)) <= wd / 2;

}

public boolean holds(ARectangle rect) {

return Math.sqrt(Math.pow(ptY - rect.getPtY(), 2))+ rect.getHt() / 2 <= ht / 2

&& Math.sqrt(Math.pow(ptX - rect.getPtX(), 2)) + rect.getWd() / 2 <= wd / 2 && ht / 2 + rect.getHt() / 2 <= ht

&& wd / 2 + rect.getWd() / 2 <= wd;

}

public boolean covers(ARectangle rect) {

return !holds(rect) && ((ptX + wd / 2 > rect.getPtX() - rect.getWd()) || (ptY + ht / 2 > rect.getPtY() - rect.getHt()))

&& (Math.sqrt(Math.pow(ptY - rect.getPtY(), 2)) < ht / 2 + rect.getHt() / 2)

&& (Math.sqrt(Math.pow(ptX - rect.getPtX(), 2))< wd / 2 + rect.getWd() / 2);

}

public double getWd() {

return wd;

}

public void setWd(double wd) {

this.wd = wd;

}

public double getHt() {

return ht;

}

public void setHt(double ht) {

this.ht = ht;

}

}

public class ACircle extends AShape{

private double radius;

public ACircle() {

ptX=0;

ptY=0;

this.radius=1;

}

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

this.ptX = ptX;

this.ptY = ptY;

this.radius=radius;

}

@Override

public double Area() {

return Math.PI \* Math.pow(radius, 2);

}

@Override

public double Perimeter() {

return 2 \* Math.PI \* radius;

}

@Override

public boolean holds(double ptX, double ptY) {

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

}

public boolean holds(ACircle c) {

return Math.sqrt(Math.pow(c.getPtX() - ptX, 2) + Math.pow(c.getPtY() - ptY, 2)) <= Math.abs(radius - c.getRadius());

}

public boolean covers(ACircle c) {

return Math.sqrt(Math.pow(c.getPtX() - ptX, 2) + Math.pow(c.getPtY() - ptY, 2)) <= radius + c.getRadius();

}

public double getRadius() {

return radius;

}

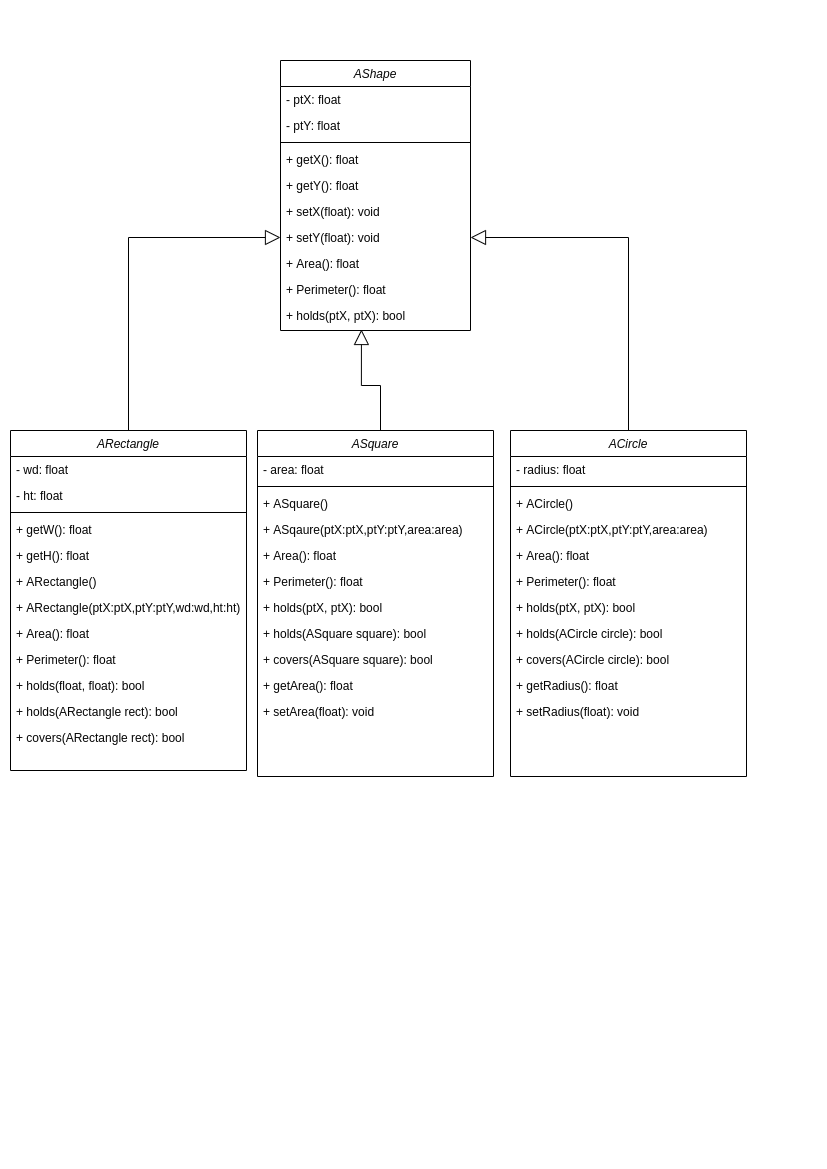
public void setRadius(double radius) {

this.radius = radius;

}

}

**Question 6:**



**Question 7:**

import java.util.\*;

public class Shape {

public static void main(String[] args) {

while(true) {

System.out.println();

Scanner sc= new Scanner(System.in);

System.out.println("Enter 1 for Rectangle");

System.out.println(" 2 for Square");

System.out.println(" 3 for Circle");

System.out.println(" any character to quit\n");

int choice= sc.nextInt();

if(choice==1) {

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

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

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

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.covers(new ARectangle(3, 5, 2.3, 5.4)));

}

else if(choice==2) {

ASquare rect01 = new ASquare(1.5, 1.5, 5.5);

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

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

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

System.out.println(rect01.holds(new ASquare(3, 5, 8.5)));

System.out.println(rect01.covers(new ASquare(3, 5, 2.3)));

}

else if(choice==3) {

ACircle rect01 = new ACircle(1.5, 1.5, 5.5);

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

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

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

System.out.println(rect01.holds(new ACircle(3, 5, 8.5)));

System.out.println(rect01.covers(new ACircle(3, 5, 2.3)));

}

else {

System.out.println("Application ended! Assignment by k17-3696");

break;

}

}

}

}

public class ASquare extends AShape{

private double area;

public ASquare() {

ptX=0;

ptY=0;

this.area=1;

}

public ASquare(double ptX, double ptY, double area) {

this.ptX=ptX;

this.ptY=ptY;

this.area=area;

}

@Override

public double Area() {

return Math.pow(area, 2);

}

@Override

public double Perimeter() {

return 4 \* area;

}

@Override

public boolean holds(double ptX, double ptY) {

return Math.sqrt(Math.pow(this.ptY - ptY, 2))<= area / 2 && Math.sqrt(Math.pow(this.ptX - ptX, 2))<= area / 2;

}

public boolean holds(ASquare s) {

return Math.sqrt(Math.pow(ptY - s.getPtY(), 2)) + s.Area() / 2 <= area / 2

&& Math.sqrt(Math.pow(ptX - s.Area(), 2)) + s.Area() / 2 <= area / 2 && area / 2 + s.Area() / 2 <= area

&& area / 2 + s.Area() / 2 <= area;

}

public boolean covers(ASquare s) {

return !holds(s) && ((ptX + area / 2 > s.getPtX() - s.Area()) || (ptY + area / 2 > s.getPtY() - s.Area()))

&& (Math.sqrt(Math.pow(ptY - s.getPtY(), 2)) < area / 2 + s.Area() / 2)

&& (Math.sqrt(Math.pow(ptX - s.getPtX(), 2))< area / 2 + s.Area() / 2);

}

public double getArea() {

return area;

}

public void setArea(double area) {

this.area = area;

}

}

public abstract class AShape {

public double ptX;

public double ptY;

public abstract double Area();

public abstract double Perimeter();

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

public double getPtX() {

return ptX;

}

public void setPtX(double ptX) {

this.ptX = ptX;

}

public double getPtY() {

return ptY;

}

public void setPtY(double ptY) {

this.ptY = ptY;

}

}

public class ARectangle extends AShape {

private double wd;

private double ht;

public ARectangle() {

ptX=0;

ptY=0;

wd=1;

ht=1;

}

public ARectangle(double ptX,double ptY,double wd,double ht) {

this.ptX=ptX;

this.ptY=ptY;

this.wd=wd;

this.ht=ht;

}

@Override

public double Area() {

return wd\*ht;

}

@Override

public double Perimeter() {

return 2\*(wd+ht);

}

@Override

public boolean holds(double ptX, double ptY) {

return Math.sqrt(Math.pow(this.ptY - ptY, 2)) <= ht / 2 && Math.sqrt(Math.pow(this.ptX - ptX, 2)) <= wd / 2;

}

public boolean holds(ARectangle rect) {

return Math.sqrt(Math.pow(ptY - rect.getPtY(), 2))+ rect.getHt() / 2 <= ht / 2

&& Math.sqrt(Math.pow(ptX - rect.getPtX(), 2)) + rect.getWd() / 2 <= wd / 2 && ht / 2 + rect.getHt() / 2 <= ht

&& wd / 2 + rect.getWd() / 2 <= wd;

}

public boolean covers(ARectangle rect) {

return !holds(rect) && ((ptX + wd / 2 > rect.getPtX() - rect.getWd()) || (ptY + ht / 2 > rect.getPtY() - rect.getHt()))

&& (Math.sqrt(Math.pow(ptY - rect.getPtY(), 2)) < ht / 2 + rect.getHt() / 2)

&& (Math.sqrt(Math.pow(ptX - rect.getPtX(), 2))< wd / 2 + rect.getWd() / 2);

}

public double getWd() {

return wd;

}

public void setWd(double wd) {

this.wd = wd;

}

public double getHt() {

return ht;

}

public void setHt(double ht) {

this.ht = ht;

}

}

public class ACircle extends AShape{

private double radius;

public ACircle() {

ptX=0;

ptY=0;

this.radius=1;

}

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

this.ptX = ptX;

this.ptY = ptY;

this.radius=radius;

}

@Override

public double Area() {

return Math.PI \* Math.pow(radius, 2);

}

@Override

public double Perimeter() {

return 2 \* Math.PI \* radius;

}

@Override

public boolean holds(double ptX, double ptY) {

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

}

public boolean holds(ACircle c) {

return Math.sqrt(Math.pow(c.getPtX() - ptX, 2) + Math.pow(c.getPtY() - ptY, 2)) <= Math.abs(radius - c.getRadius());

}

public boolean covers(ACircle c) {

return Math.sqrt(Math.pow(c.getPtX() - ptX, 2) + Math.pow(c.getPtY() - ptY, 2)) <= radius + c.getRadius();

}

public double getRadius() {

return radius;

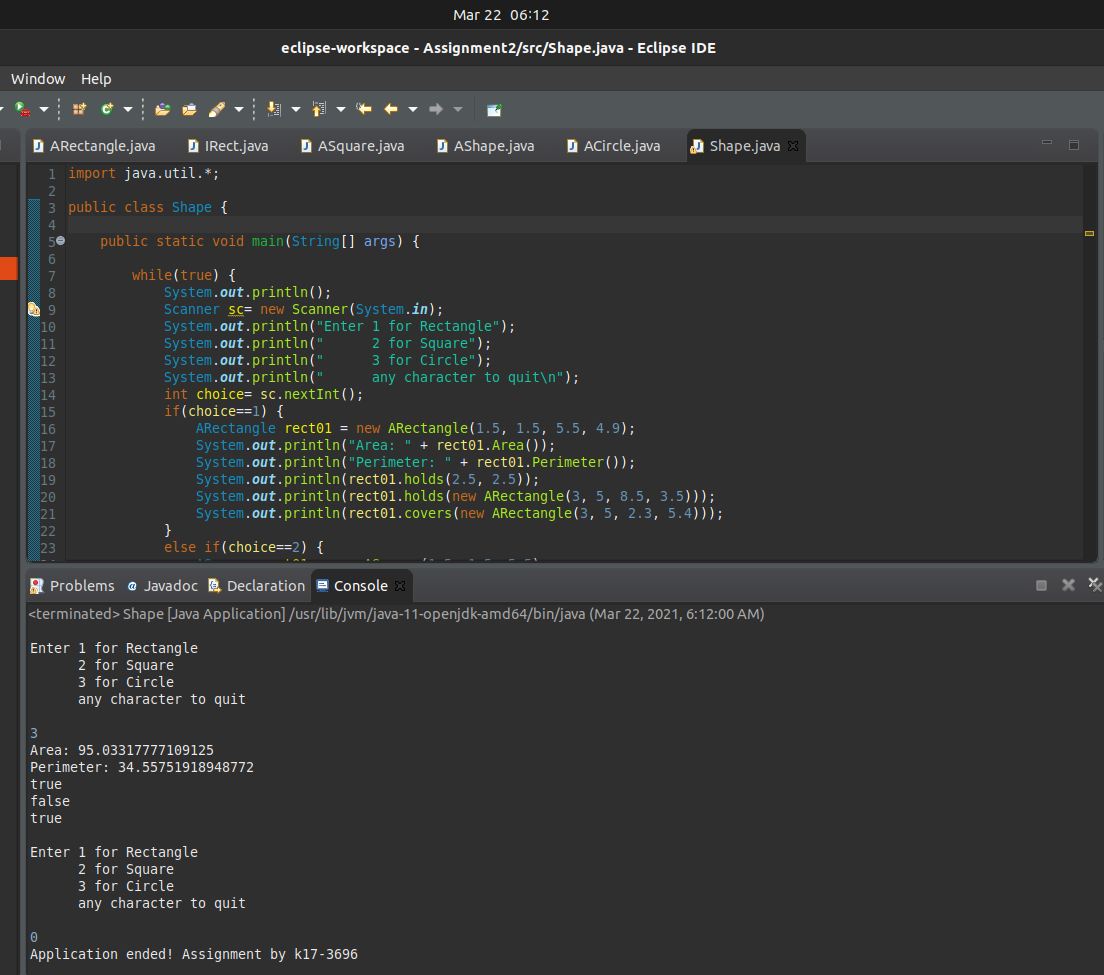
}

public void setRadius(double radius) {

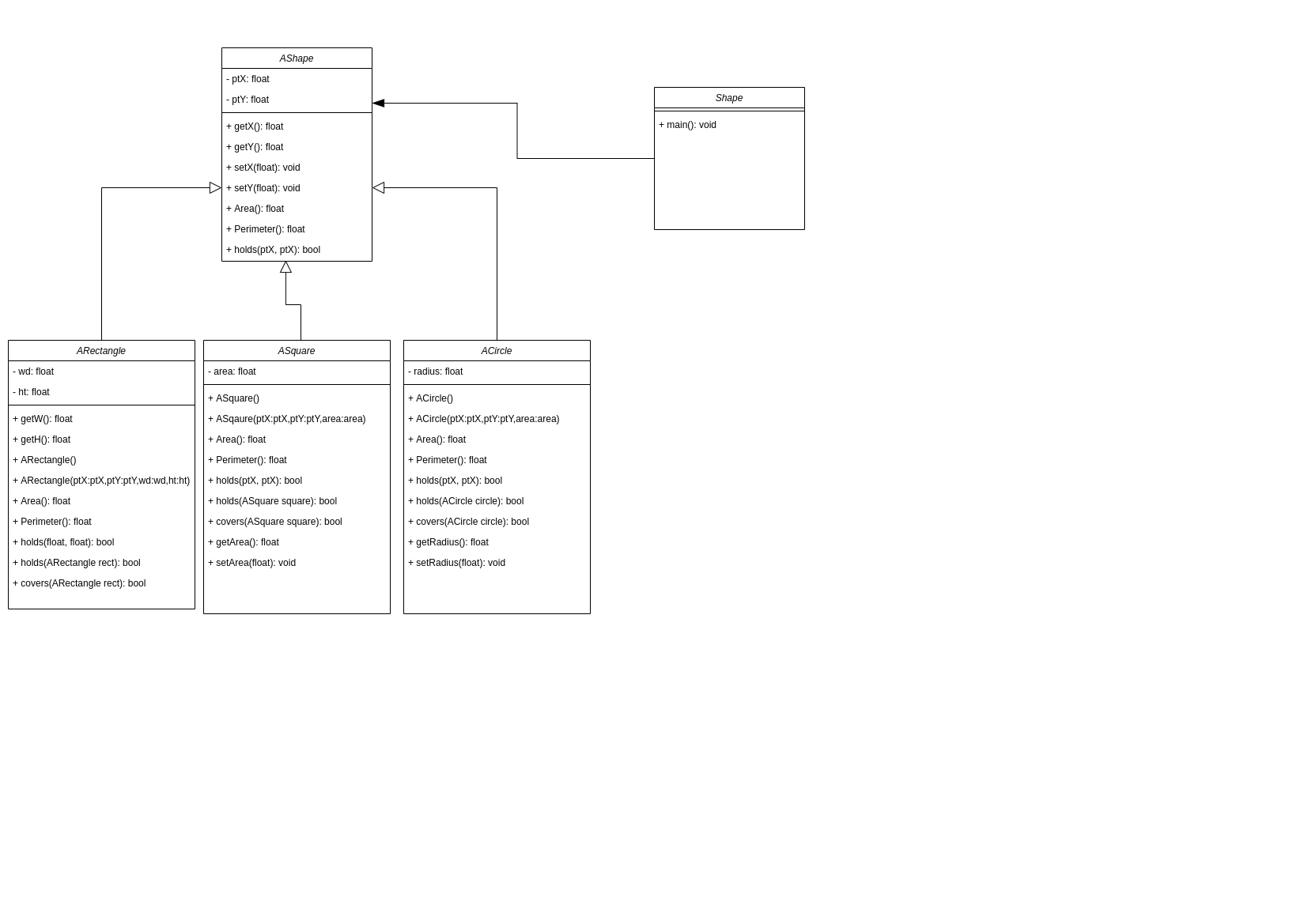
this.radius = radius;

}

}



**Question 8:**



**Question 9:**

First we build an ARectangle rectangle class. Then we made a class for the client, so the client doesn't need to access the rectangle class directly he can just use it through the IRect client class. After that we added an abstract AShape class to add more shapes e.g. ACircle and ASquare. In the end, we added the client interface class Shape so the client can access the shapes by selecting choices from the client menu.