Brandon Bardwell

3/22/20

Computer Science II

Module 5: Assignment 1:

For this assignment I used Solution Manual to help me code this, I typed it all by hand to help see what the code itself would be doing.

<http://soultionmanual.blogspot.com/2016/12/chapter-13-exercise-6-introduction-to.html>

My Code:

**public** **class** Main {

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

ComparableCircle c1 = **new** ComparableCircle(0,0,5);

ComparableCircle c2 = **new** ComparableCircle(0,0,10);

ComparableCircle c3 = (ComparableCircle) GeometricObject.*max*(c1, c2);

System.***out***.println(c1);

System.***out***.println(c2);

System.***out***.println("Max circle = " + c3.getRadius());

System.***out***.println(c3);

}

}

**class** ComparableCircle **extends** Circle2D { // GeometricObject Implements compareTo

ComparableCircle() {

}

ComparableCircle(**double** x, **double** y, **double** radius) {

**super**(x, y, radius);

}

}

**public** **abstract** **class** GeometricObject **implements** Comparable<GeometricObject> {

**private** String color = "white";

**private** **boolean** filled;

**private** java.util.Date dateCreated;

/\*\* Construct a default geometric object \*/

**protected** GeometricObject() {

dateCreated = **new** java.util.Date();

}

/\*\* Construct a geometric object with color and filled value \*/

**protected** GeometricObject(String color, **boolean** filled) {

dateCreated = **new** java.util.Date();

**this**.color = color;

**this**.filled = filled;

}

/\*\* Return color \*/

**public** String getColor() {

**return** color;

}

/\*\* Set a new color \*/

**public** **void** setColor(String color) {

**this**.color = color;

}

/\*\* Return filled. Since filled is boolean,

\* the get method is named isFilled \*/

**public** **boolean** isFilled() {

**return** filled;

}

/\*\* Set a new filled \*/

**public** **void** setFilled(**boolean** filled) {

**this**.filled = filled;

}

/\*\* Get dateCreated \*/

**public** java.util.Date getDateCreated() {

**return** dateCreated;

}

/\*\* Return a string representation of this object \*/

@Override

**public** String toString() {

**return** "created on " + dateCreated + "\ncolor: " + color +

" and filled: " + filled;

}

@Override

**public** **int** compareTo(GeometricObject o) {

**if** (getArea() > o.getArea())

**return** 1;

**else** **if** (getArea() < o.getArea())

**return** -1;

**else**

**return** 0;

}

**public** **static** GeometricObject max(GeometricObject o1, GeometricObject o2) {

**return** (o1.compareTo(o2) >= 0) ? o1 : o2;

}

**public** **static** **double** sumArea(GeometricObject[] a) {

**double** sum = 0;

**for** (GeometricObject o : a) {

sum += o.getArea();

}

**return** sum;

}

/\*\* Abstract method getArea \*/

**public** **abstract** **double** getArea();

/\*\* Abstract method getPerimeter \*/

**public** **abstract** **double** getPerimeter();

}

**import** javafx.scene.shape.Circle;

**public** **class** Circle2D **extends** GeometricObject {

**private** **double** x;

**private** **double** y;

**private** **double** radius;

**public** Circle2D(Circle c) {

**this**(c.getCenterX(), c.getCenterY(), c.getRadius());

}

**public** Circle2D(**double** x, **double** y, **double** radius) {

**this**.x = x;

**this**.y = y;

**this**.radius = radius;

}

**public** Circle2D() {

**this**(0, 0, 1);

}

**public** **double** getX() {

**return** x;

}

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

**this**.x = x;

}

**public** **double** getY() {

**return** y;

}

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

**this**.y = y;

}

**public** **double** getRadius() {

**return** radius;

}

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

**this**.radius = radius;

}

@Override

**public** **double** getArea() {

**return** radius \* radius \* Math.***PI***;

}

@Override

**public** **double** getPerimeter() {

**return** 2 \* radius \* Math.***PI***;

}

**public** **boolean** contains(Circle2D circle2D) {

**double** distance = getPoint().distance(circle2D.x, circle2D.y);

**if** (distance <= Math.*abs*(**this**.radius - circle2D.radius)) {

**return** **true**;

} **else** {

**return** **false**;

}

}

**public** **static** **boolean** c1ContainsC2(Circle c1, Circle c2) {

Circle2D cir1 = **new** Circle2D(c1);

Circle2D cir2 =**new** Circle2D(c2);

**return** cir1.contains(cir2);

}

**public** **static** **boolean** c1OverlapsC2(Circle c1, Circle c2) {

Circle2D cir1 = **new** Circle2D(c1);

Circle2D cir2 =**new** Circle2D(c2);

**if** (cir1.contains(cir2) || cir2.contains(cir1)) **return** **false**;

**return** cir1.overlaps(cir2);

}

**public** **boolean** overlaps(Circle2D circle2D) {

**double** distance = getPoint().distance(circle2D.x, circle2D.y);

**if** (distance <= **this**.radius + circle2D.radius)

**return** **true**;

**else**

**return** **false**;

}

**private** MyPoint getPoint() {

**return** **new** MyPoint(**this**.x, **this**.y);

}

@Override

**public** String toString() {

**return** "Circle2D{" +

"x=" + x +

", y=" + y +

", radius=" + radius +

'}';

}

**public** **boolean** contains(**double** x, **double** y) {

**double** distance = getPoint().distance(x, y);

**if** (distance <= radius) **return** **true**;

**else** **return** **false**;

}

}