**Assignment 2**

package geometricfigure;

// Imported to help sort the array based on the compareTo() method of ComparableSquare.

import java.util.Arrays;

// This class will act as a demo application to create and display different geometric figures.

public class GeometricFigureDemo

{

// Main method.

public static void main(String[] args)

{

// Create a rectangle named r1.

Rectangle r1 = new Rectangle(2,3);

// Computing the area for r1.

r1.computeArea();

// Display r1.

System.out.println("Rectangle R1: ");

r1.displayOne();

System.out.println("------------------------------------------");

// Creating a square named s1.

// We will test the squares 2 argument constructor by passing a different height and width.

// Since squares must have the same height and width, the constructor will simply pass the height twice.

Square s1 = new Square(2,3);

// Computing the area for s1.

s1.computeArea();

// Display s1.

System.out.println("Square S1: ");

s1.displayOne();

System.out.println("------------------------------------------");

// Creating a triangle named t1.

Triangle t1 = new Triangle(2,3);

// Computing the area for t1.

t1.computeArea();

// Dispaly t1.

System.out.println("Triangle T1: ");

t1.displayOne();

System.out.println("------------------------------------------");

// Creating an array arr of 2 squares, one rectangle and one triangle.

GeometricFigure[] arr = new GeometricFigure[4];

arr[0] = new Square(2);

arr[1] = new Square(3);

arr[2] = new Rectangle(3,4);

arr[3] = new Triangle(5.2,6.1);

// Display the array using an enhanced for loop.

System.out.println("Mixed array demo(2 squares, 1 rectangle, 1 triangle): ");

for(GeometricFigure x:arr)

{

x.displayOne();

}

System.out.println("------------------------------------------");

// Create 2 ComparableSquare objects named cs1 and cs2 respectively.

ComparableSquare cs1 = new ComparableSquare(2);

ComparableSquare cs2 = new ComparableSquare(5);

// Calculate the areas for cs1 and cs2.

cs1.computeArea();

cs2.computeArea();

System.out.println("ComparbleSquare Larger Area Test: ");

switch (cs1.compareTo(cs2))

{

// If cs1's area is larger than cs2's area. We will indicate that cs1's area is larger.

case 1:

System.out.println("cs1's area is larger.");

break;

// Otherwise, if cs2's area is larger than cs1's area. We will indicate that cs2's area is larger.

// In this instance, this is the case we expect to be executed.

case -1:

System.out.println("cs2's area is larger.");

break;

// Otherwise, both the areas are equal and we should indicate as such.

default:

System.out.println("The areas are equal.");

break;

}

System.out.println(); // Empty Line.

System.out.println("------------------------------------------");

// Create an array named csArr of 4 ComparableSquare objects.

ComparableSquare[] csArr = new ComparableSquare[4];

csArr[0] = new ComparableSquare(3);

csArr[1] = new ComparableSquare(1);

csArr[2] = new ComparableSquare(5);

csArr[3] = new ComparableSquare(4);

// Compute the area for every element using an enhanced for loop.

for(ComparableSquare x:csArr)

{

x.computeArea();

}

// Sort the array based on the compareTo method in the ComparableSquare class.

Arrays.sort(csArr);

// Print the array in its compared state using an enhanced for loop;

System.out.println("Printing ComparableSquare array in its sorted state(sorted by ascending area)");

for(ComparableSquare x:csArr)

{

x.displayOne();

}

System.out.println("------------------------------------------");

}

}

package geometricfigure;

// This class will model a Rectangle.

// A rectangle is a geometric figure so we should inherit that class.

public class Rectangle extends GeometricFigure

{

// The area for a rectangle is calculated by multiplying the height and width.

@Override

public void computeArea()

{

this.area = this.height \* this.width;

}

// Constructor that will call the constructor of geometric figure(2 arguments: height and width).

public Rectangle(double height,double width)

{

super(height,width);

}

}

package geometricfigure;

// This class will model a Square.

// All squares are rectangles so we should inherit that class.

public class Square extends Rectangle

{

// 2 argument constructor.

public Square(double height,double width)

{

// The height and width must be the same for a square.

super(height,height);

}

// 1 Argument constuctor.

// This can be done because for a square, the height and width must be the same.

public Square(double height)

{

super(height,height);

}

// The square uses the rectangle's computeArea() method, therefore it is not needed again here.

}

package geometricfigure;

// This class will model a triangle.

// Triangles are geometric figures so we should inherit that class.

public class Triangle extends GeometricFigure

{

// The area for the triangle is determined by multiplying the width by half the height.

@Override

public void computeArea()

{

this.area = this.width \* (this.height / 2);

}

// 2 argument constructor that will call the parents constructor.

public Triangle(double height,double width)

{

super(height,width);

}

}

package geometricfigure;

// This class will model a square that can be compared to another comparable square by its area.

// Since a square is a geometric figure we should inherit that class.

// Since we need a way to compare 2 comparable squares, we will impliment the Comparable interface.

public class ComparableSquare extends Square implements Comparable<ComparableSquare>

{

// Override the interface's compareTo() method to compare two comparable squares based on their area.

@Override

public int compareTo(ComparableSquare other)

{

// Return 1 if the calling object's area is bigger than other's area.

if(this.area > other.area)

{

return 1;

}

// Return -1 is if the calling object's area is smaller than the other's area.

if(this.area < other.area)

{

return -1;

}

// Otherwise, the two areas are equal and we return 0.

return 0;

}

// Constructors that will call squares constructors.

// The same logic will be used as in the Square class to make sure the height and width are the same. (See comments on Square method).

public ComparableSquare(double height, double width)

{

super(height,height);

}

public ComparableSquare(double height)

{

super(height,height);

}

}

package geometricfigure;

// This class will act as a demo application to create and display different geometric figures.

// Imported to help sort the array based on the compareTo() method of ComparableSquare.

import java.util.Arrays;

public class GeometricFigureDemo

{

// Main method.

public static void main(String[] args)

{

// Create a rectangle named r1.

Rectangle r1 = new Rectangle(2,3);

// Computing the area for r1.

r1.computeArea();

// Display r1.

System.out.println("Rectangle R1: ");

r1.displayOne();

System.out.println("------------------------------------------");

// Creating a square named s1.

// We will test the squares 2 argument constructor by passing a different height and width.

// Since squares must have the same height and width, the constructor will simply pass the height twice.

Square s1 = new Square(2,3);

// Computing the area for s1.

s1.computeArea();

// Display s1.

System.out.println("Square S1: ");

s1.displayOne();

System.out.println("------------------------------------------");

// Creating a triangle named t1.

Triangle t1 = new Triangle(2,3);

// Computing the area for t1.

t1.computeArea();

// Dispaly t1.

System.out.println("Triangle T1: ");

t1.displayOne();

System.out.println("------------------------------------------");

// Creating an array arr of 2 squares, one rectangle and one triangle.

GeometricFigure[] arr = new GeometricFigure[4];

arr[0] = new Square(2);

arr[1] = new Square(3);

arr[2] = new Rectangle(3,4);

arr[3] = new Triangle(5.2,6.1);

// Display the array using an enhanced for loop.

System.out.println("Mixed array demo(2 squares, 1 rectangle, 1 triangle): ");

for(GeometricFigure x:arr)

{

x.displayOne();

}

System.out.println("------------------------------------------");

// Create 2 ComparableSquare objects named cs1 and cs2 respectively.

ComparableSquare cs1 = new ComparableSquare(2);

ComparableSquare cs2 = new ComparableSquare(5);

// Calculate the areas for cs1 and cs2.

cs1.computeArea();

cs2.computeArea();

System.out.println("ComparbleSquare Larger Area Test: ");

switch (cs1.compareTo(cs2))

{

// If cs1's area is larger than cs2's area. We will indicate that cs1's area is larger.

case 1:

System.out.println("cs1's area is larger.");

break;

// Otherwise, if cs2's area is larger than cs1's area. We will indicate that cs2's area is larger.

// In this instance, this is the case we expect to be executed.

case -1:

System.out.println("cs2's area is larger.");

break;

// Otherwise, both the areas are equal and we should indicate as such.

default:

System.out.println("The areas are equal.");

break;

}

System.out.println(); // Empty Line.

System.out.println("------------------------------------------");

// Create an array named csArr of 4 ComparableSquare objects.

ComparableSquare[] csArr = new ComparableSquare[4];

csArr[0] = new ComparableSquare(3);

csArr[1] = new ComparableSquare(1);

csArr[2] = new ComparableSquare(5);

csArr[3] = new ComparableSquare(4);

// Compute the area for every element using an enhanced for loop.

for(ComparableSquare x:csArr)

{

x.computeArea();

}

// Sort the array based on the compareTo method in the ComparableSquare class.

Arrays.sort(csArr);

// Print the array in its compared state using an enhanced for loop;

System.out.println("Printing ComparableSquare array in its sorted state(sorted by ascending area)");

for(ComparableSquare x:csArr)

{

x.displayOne();

}

System.out.println("------------------------------------------");

}

}

