13.2

Import java.util.ArrayList;

Public class Exercise\_02 {

Public static void main(String[] args) {

ArrayList<Number> list = new ArrayList<>();

For (int I = 0; I < 100; i++) list.add(i);

System.out.println(list);

System.out.println(“Shuffling list…”);

Shuffle(list);

System.out.println(list);

}

Public static void shuffle(ArrayList<Number> list) {

// simple solution

//Collections.shuffle(list);

// manual shuffle

For (int I = 0; I < list.size(); i++) {

Int randomIndex = (int) (Math.random() \* list.size());

Number temp = list.get(randomIndex);

List.set(randomIndex, list.get(i));

List.set(I, temp);

}

}

}

11.3

Import javafx.scene.control.TextField;

Public class Kit {

Public static final int NOT\_VALID = -1;

Public static String randomNumbers(int size) {

String s = “”;

For (int I = 0; I < size; i++) {

S += “ “ + (int) (Math.random() \* 100);

}

Return s;

}

Public static Object[] getArray(Object… objects) {

Object[] temp = new TextField[objects.length];

For (int I = 0; I < objects.length; i++) {

Temp[i] = objects[i];

}

Return temp;

}

Public static String binaryFormat(String binaryString) {

StringBuilder string = new StringBuilder(binaryString);

Int extraZeros = string.length() % 4;

If (extraZeros != 0) {

For (int I = 0; I < extraZeros; i++) {

String.insert(0, “0”);

}

}

For (int I = string.length() – 1; I >= 0; i--) {

If (I % 4 == 0 && I != 0) {

String.insert(I, “ “);

}

}

Return string.toString();

}

Public static String hexToBinary(String hex) {

Return decimalToBinary(hexToDecimal(hex));

}

Public static String decimalToBinary(long n) {

StringBuilder stringBuilder = new StringBuilder();

While (n > 0) {

stringBuilder.insert(0, n & 1);

n >>= 1;

}

Return stringBuilder.toString();

}

Public static String decimalToHex(long n) {

StringBuilder stringBuilder = new StringBuilder();

While (n > 0) {

stringBuilder.insert(0, decimalToHexChar(n % 16));

n /= 16;

}

Return stringBuilder.toString();

}

Public static long hexToDecimal(String hex) {

If (!isHexadecimal(hex)) return NOT\_VALID;

Long n = 0;

Int placeValue = hex.length() – 1;

For (int I = 0; I < hex.length(); i++) {

Char ch = hex.charAt(i);

If (isNumeric(ch)) {

N += (charToDecimal(ch) \* Math.pow(16, placeValue--));

} else {

N += hexLetterToInt(ch) \* Math.pow(16, placeValue--);

}

}

Return n;

}

Private static int hexLetterToInt(char ch) {

Ch = Character.toUpperCase(ch);

Return (ch – ‘A’ + 10);

}

/\*\* returns Z is n is > 15 \*/

Private static char decimalToHexChar(long n) {

If (n > 15) return ‘Z’;

If (n > 9) return (char)(n % 10 + ‘A’);

Return (char)(n + ‘0’);

}

Public static long stringToDecimal(String s) {

Long n = 0;

Int placeValue = s.length() – 1;

For (int I = 0; I < s.length(); i++) {

Int valid = charToDecimal(s.charAt(i));

If (valid == NOT\_VALID) return NOT\_VALID;

N += (Math.pow(10, placeValue--) \* valid);

}

Return n;

}

Public static String decimalToString(long n) {

StringBuilder s = new StringBuilder();

While (n != 0) {

s.insert(0, n % 10);

n /= 10;

}

Return s.toString();

}

Public static int charToDecimal(char ch) {

If (!isNumeric(ch)) return NOT\_VALID;

Return ch – ‘0’;

}

Public static boolean isNumeric(String s) {

For (int I = 0; I < s.length(); i++) {

If (!isNumeric(s.charAt(i))) return false;

}

Return true;

}

Public static boolean isNumeric(char ch) {

Return (ch >= ‘0’ && ch <= ‘9’);

}

Public static boolean isHexadecimal(String s) {

For (int I = 0; I < s.length(); i++) {

If (!isHexValid(s.charAt(i))) return false;

}

Return true;

}

Public static boolean isHexValid(char ch) {

Ch = Character.toUpperCase(ch);

Return isCharAtRange(ch, ‘0’, ‘9’) || isCharAtRange(ch, ‘A’, ‘F’);

}

Public static boolean isCharAtRange(char ch, char start, char end) {

Return !(ch < start || ch > end);

}

Public static int count(String str, char a) {

Int count = 0;

For (int I = 0; I < str.length(); i++) {

If (str.charAt(i) == a) count++;

}

Return count;

}

Public static void displayGrid(int[][] grid) {

For (int[] aGrid : grid) {

System.out.printf(“%20s”, “”);

For (int k = 0; k < aGrid.length; k++) {

System.out.printf(“|%-2d|”, aGrid[k]);

}

System.out.println(“”);

}

}

Public static void displayGrid(int[] grid) {

For (int I = 0; I < grid.length; i++) {

System.out.printf(“%-4d “, grid[i]);

If ((I + 1) % 10 == 0)

System.out.println(“”);

}

}

Public static void displayTimesTable(int x, int y) {

// display 8x8 numbered grid to help solve exercise

For (int I = 1; I <= y; i++) {

System.out.printf(“%20s”, “”);

For (int k = 1; k <= x; k++) {

System.out.printf(“|%-2d|”, I \* k);

}

System.out.println(“”);

}

}

Public static int[] makeUnsortedArray(int size, int range) {

Int[] array = new int[size];

For (int I = 0; I < array.length; i++) {

Array[i] = (int) (Math.random() \* range + 1);

}

Return array;

}

Public static double[][] makeUnsortedArray(int rows, int columns, int range) {

Double[][] array = new double[rows][columns];

For (int I = 0; I < array.length; i++) {

For (int j = 0; j < array[i].length; j++) {

Array[i][j] = (int) (Math.random() \* range + 1);

}

}

Return array;

}

Public static int[] merge(int[] list1, int[] list2) {

Int[] merge = new int[list1.length + list2.length];

Int m = 0, l1 = 0, l2 = 0;

Boolean l1isFinished = false;

Boolean l2isFinished = false;

While (!l1isFinished || !l2isFinished) {

If (l1 == list1.length) l1isFinished = true;

If (l2 == list2.length) l2isFinished = true;

If (l1isFinished && l2isFinished) break;

}

Return merge;

}

Public static boolean isSorted(int[] numbers) {

For (int I = 0; I < numbers.length – 1; i++) {

If (numbers[i] > numbers[I + 1]) return false;

}

Return true;

}

Public static void displayMatrix(double[][] matrix) {

For (double[] aMatrix : matrix) {

For (int column = 0; column < aMatrix.length; column++) {

System.out.printf(“%5.0f “, aMatrix[column]);

}

System.out.printf(“\n”);

}

}

Public static void display(Object[] objects) {

Int count = 1;

For (Object o : objects) {

System.out.printf(“%4s\n”, o.toString());

If (count % 10 == 0) {

System.out.println(“”);

}

Count++;

}

}

}

Import java.util.ArrayList;

Import java.util.Collections;

Public class Exercise\_03 {

Public static void main(String[] args) {

ArrayList<Number> numbers = new ArrayList<>();

For (int I = 0; I < 100; i++) {

Numbers.add(i);

}

Collections.shuffle(numbers);

System.out.println(“Shuffled: “);

Kit.display(numbers.toArray());

System.out.println(“Sorted: “);

Sort(numbers);

Kit.display(numbers.toArray());

}

Public static void sort(ArrayList<Number> list) {

For (int I = 0; I < list.size() – 1; i++) {

Number currentMin = list.get(i);

Int currentIndex = I;

For (int j = I + 1; j < list.size(); j++) {

If (list.get(j).doubleValue() < currentMin.doubleValue()) {

currentMin = list.get(j);

currentIndex = j;

}

}

If (currentIndex != i) {

List.set(currentIndex, list.get(i));

List.set(I, currentMin);

}

}

13.5

Import javafx.scene.shape.Rectangle;

Public class MyRectangle2D extends GeometricObject {

Private static final int X = 0;

Private static final int Y = 1;

Public double centerX;

Public double centerY;

Public double width;

Public double height;

Public MyRectangle2D(double centerX, double centerY, double width, double height) {

This.centerX = centerX;

This.centerY = centerY;

This.width = width;

This.height = height;

}

Public MyRectangle2D(MyPoint centerPoint, double width, double height) {

This(centerPoint.x, centerPoint.y, width, height);

}

Public MyRectangle2D(Rectangle rectangle) {

This(rectangle.getX(), rectangle.getY(), rectangle.getWidth(), rectangle.getHeight());

}

Public MyRectangle2D() {

This(0, 0, 1, 1);

}

Public MyPoint getCenterP() {

Return new MyPoint(centerX, centerY);

}

Public MyPoint getP1() {

Return new MyPoint(centerX – width / 2, centerY + height / 2);

}

Public MyPoint getP2() {

Return new MyPoint(centerX + width / 2, centerY + height / 2);

}

Public MyPoint getP3() {

Return new MyPoint(centerX – width / 2, centerY – height / 2);

}

Public MyPoint getP4() {

Return new MyPoint(centerX + width / 2, centerY – height / 2);

}

Public double getCenterX() {

Return centerX;

}

Public void setCenterX(double centerX) {

This.centerX = centerX;

}

Public double getCenterY() {

Return centerY;

}

Public void setCenterY(double centerY) {

This.centerY = centerY;

}

Public double getWidth() {

Return width;

}

Public void setWidth(double width) {

This.width = width;

}

Public double getHeight() {

Return height;

}

Public void setHeight(double height) {

This.height = height;

}

/\*\* returns the perimeter of the rectangle. \*\*/

@Override

Public double getPerimeter() {

Return width \* 2 + height \* 2;

}

/\*\* returns the area of the rectangle \*\*/

@Override

Public double getArea() {

Return width \* height;

}

Public boolean contains(double x, double y) {

// Get max X & Y

Double maxX = getMax(getP1().x, getP2().x, getP3().x, getP4().x);

Double maxY = getMax(getP1().y, getP2().y, getP3().y, getP4().y);

// Get min X & Y

Double minX = getMin(getP1().x, getP2().x, getP3().x, getP4().x);

Double minY = getMin(getP1().y, getP2().y, getP3().y, getP4().y);

If (x < minX || x > maxX || y < minY || y > maxY) return false;

Return true;

}

Public boolean contains(MyRectangle2D r2) {

Double xDistance = (r2.centerX > centerX) ? r2.centerX – centerX : centerX – r2.centerX;

Double yDistance = (r2.centerY > centerY) ? r2.centerY – centerY : centerY – r2.centerY;

// if the r2 is inside this rectangle

// this is only valid if the rectangles are parallel

Return (xDistance <= (width – r2.width) / 2 && yDistance <= (height – r2.height) / 2);

}

Public boolean overlaps(MyRectangle2D r2) {

// subtract from the highest number

Double xDistance = (r2.centerX > centerX) ? r2.centerX – centerX : centerX – r2.centerX;

Double yDistance = (r2.centerY > centerY) ? r2.centerY – centerY : centerY – r2.centerY;

// if the r2 overlaps this rectangle

// this is only valid if the rectangles are parallel

Return (xDistance <= (width + r2.width) / 2 && yDistance <= (height + r2.height) / 2);

}

Public static boolean r1OverLapsR2(Rectangle r1, Rectangle r2) {

Return r1.intersects(r2.getBoundsInLocal()) || r2.intersects(r1.getBoundsInLocal());

}

Public javafx.scene.shape.Rectangle getRectangle() {

Return new Rectangle(centerX, centerY, width, height);

}

Public boolean contains(MyPoint[] points) {

For (MyPoint p : points) {

If (!contains(p)) return false;

}

Return true;

}

Public boolean contains(MyPoint point) {

Return contains(point.x, point.y);

}

Public static MyRectangle2D getRectangle(double[][] points) {

// find leftLowestPoint

// find rightMostPoint

// find the center from highest and lowest point

// x distance = width

// y distance = height

MyPoint[] myPoints = new MyPoint[points.length];

For (int I = 0; I < points.length; i++) {

myPoints[i] = new MyPoint(points[i][0], points[i][1]);

}

Return getRectangle(myPoints);

}

Public static MyRectangle2D getRectangle(MyPoint[] points) {

MyPoint leftMost = getLeftMostPoint(points);

MyPoint rightMost = getRighMostPoint(points);

Double width = Math.abs(rightMost.x – leftMost.x);

MyPoint highest = getHighestPoint(points);

MyPoint lowest = getLowestPoint(points);

Double height = Math.abs(highest.y – lowest.y);

Double centerX = highest.getCenterPoint(lowest).x;

Double centerY = leftMost.getCenterPoint(rightMost).y;

Return new MyRectangle2D(centerX, centerY, width, height);

}

Private static MyPoint getLeftMostPoint(MyPoint[] points) {

MyPoint leftMost = points[0];

For (int I = 0; I < points.length; i++) {

If (leftMost.x > points[i].x) {

leftMost = points[i];

}

}

Return leftMost;

}

Private static MyPoint getRighMostPoint(MyPoint[] points) {

MyPoint rightMost = points[0];

For (int I = 0; I < points.length; i++) {

If (rightMost.x < points[i].x) {

rightMost = points[i];

}

}

Return rightMost;

}

Private static MyPoint getHighestPoint(MyPoint[] points) {

MyPoint highest = points[0];

For (int I = 0; I < points.length; i++) {

If (highest.y < points[i].y) {

Highest = points[i];

}

}

Return highest;

}

Private static MyPoint getLowestPoint(MyPoint[] points) {

MyPoint lowest = points[0];

For (int I = 0; I < points.length; i++) {

If (lowest.y > points[i].y) {

Lowest = points[i];

}

}

Return lowest;

}

Private double getMax(double… n) {

Double max = n[0];

For (int I = 1; I < n.length; i++) {

If (max < n[i]) {

Max = n[i];

}

}

Return max;

}

Private double getMin(double… n) {

Double min = n[0];

For (int I = 1; I < n.length; i++) {

If (min > n[i]) {

Min = n[i];

}

}

Return min;

}

@Override

Public String toString() {

Return “MyRectangle2D{“ +

“centerX=” + centerX +

“, centerY=” + centerY +

“, width=” + width +

“, height=” + height +

‘}’;

}

} 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;

}

}

Public class Exercise\_05 {

Public static void main(String[] args) {

Circle2D c1 = new Circle2D(0, 0, 5);

Circle2D c2 = new Circle2D(0, 0, 2);

Circle2D maxCircle =(Circle2D)GeometricObject.max(c1, c2);

System.out.println(“Circle 1: “ + c1);

System.out.println(“Circle 2: “ + c2);

System.out.println(“Max circle is: “ + maxCircle);

MyRectangle2D r1 = new MyRectangle2D(0, 0, 10, 5);

MyRectangle2D r2 = new MyRectangle2D(0, 0, 10, 10);

MyRectangle2D maxRectangle = (MyRectangle2D) GeometricObject.max(r1, r2);

System.out.println(“MyRec2 1: “ + r1);

System.out.println(“MyRec2 2: “ + r2);

System.out.println(“Max MyRec2 is: “ + maxRectangle);

}

13.6

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;

}

}

Public class Exercise\_06 {

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);

}

}

13.9

Import javafx.scene.shape.Rectangle;

Public class MyRectangle2D extends GeometricObject {

Private static final int X = 0;

Private static final int Y = 1;

Public double centerX;

Public double centerY;

Public double width;

Public double height;

Public MyRectangle2D(double centerX, double centerY, double width, double height) {

This.centerX = centerX;

This.centerY = centerY;

This.width = width;

This.height = height;

}

Public MyRectangle2D(MyPoint centerPoint, double width, double height) {

This(centerPoint.x, centerPoint.y, width, height);

}

Public MyRectangle2D(Rectangle rectangle) {

This(rectangle.getX(), rectangle.getY(), rectangle.getWidth(), rectangle.getHeight());

}

Public MyRectangle2D() {

This(0, 0, 1, 1);

}

Public MyPoint getCenterP() {

Return new MyPoint(centerX, centerY);

}

Public MyPoint getP1() {

Return new MyPoint(centerX – width / 2, centerY + height / 2);

}

Public MyPoint getP2() {

Return new MyPoint(centerX + width / 2, centerY + height / 2);

}

Public MyPoint getP3() {

Return new MyPoint(centerX – width / 2, centerY – height / 2);

}

Public MyPoint getP4() {

Return new MyPoint(centerX + width / 2, centerY – height / 2);

}

Public double getCenterX() {

Return centerX;

}

Public void setCenterX(double centerX) {

This.centerX = centerX;

}

Public double getCenterY() {

Return centerY;

}

Public void setCenterY(double centerY) {

This.centerY = centerY;

}

Public double getWidth() {

Return width;

}

Public void setWidth(double width) {

This.width = width;

}

Public double getHeight() {

Return height;

}

Public void setHeight(double height) {

This.height = height;

}

/\*\* returns the perimeter of the rectangle. \*\*/

@Override

Public double getPerimeter() {

Return width \* 2 + height \* 2;

}

/\*\* returns the area of the rectangle \*\*/

@Override

Public double getArea() {

Return width \* height;

}

Public boolean contains(double x, double y) {

// Get max X & Y

Double maxX = getMax(getP1().x, getP2().x, getP3().x, getP4().x);

Double maxY = getMax(getP1().y, getP2().y, getP3().y, getP4().y);

// Get min X & Y

Double minX = getMin(getP1().x, getP2().x, getP3().x, getP4().x);

Double minY = getMin(getP1().y, getP2().y, getP3().y, getP4().y);

If (x < minX || x > maxX || y < minY || y > maxY) return false;

Return true;

}

Public boolean contains(MyRectangle2D r2) {

Double xDistance = (r2.centerX > centerX) ? r2.centerX – centerX : centerX – r2.centerX;

Double yDistance = (r2.centerY > centerY) ? r2.centerY – centerY : centerY – r2.centerY;

// if the r2 is inside this rectangle

// this is only valid if the rectangles are parallel

Return (xDistance <= (width – r2.width) / 2 && yDistance <= (height – r2.height) / 2);

}

Public boolean overlaps(MyRectangle2D r2) {

// subtract from the highest number

Double xDistance = (r2.centerX > centerX) ? r2.centerX – centerX : centerX – r2.centerX;

Double yDistance = (r2.centerY > centerY) ? r2.centerY – centerY : centerY – r2.centerY;

// if the r2 overlaps this rectangle

// this is only valid if the rectangles are parallel

Return (xDistance <= (width + r2.width) / 2 && yDistance <= (height + r2.height) / 2);

}

Public static boolean r1OverLapsR2(Rectangle r1, Rectangle r2) {

Return r1.intersects(r2.getBoundsInLocal()) || r2.intersects(r1.getBoundsInLocal());

}

Public javafx.scene.shape.Rectangle getRectangle() {

Return new Rectangle(centerX, centerY, width, height);

}

Public boolean contains(MyPoint[] points) {

For (MyPoint p : points) {

If (!contains(p)) return false;

}

Return true;

}

Public boolean contains(MyPoint point) {

Return contains(point.x, point.y);

}

Public static MyRectangle2D getRectangle(double[][] points) {

// find leftLowestPoint

// find rightMostPoint

// find the center from highest and lowest point

// x distance = width

// y distance = height

MyPoint[] myPoints = new MyPoint[points.length];

For (int I = 0; I < points.length; i++) {

myPoints[i] = new MyPoint(points[i][0], points[i][1]);

}

Return getRectangle(myPoints);

}

Public static MyRectangle2D getRectangle(MyPoint[] points) {

MyPoint leftMost = getLeftMostPoint(points);

MyPoint rightMost = getRighMostPoint(points);

Double width = Math.abs(rightMost.x – leftMost.x);

MyPoint highest = getHighestPoint(points);

MyPoint lowest = getLowestPoint(points);

Double height = Math.abs(highest.y – lowest.y);

Double centerX = highest.getCenterPoint(lowest).x;

Double centerY = leftMost.getCenterPoint(rightMost).y;

Return new MyRectangle2D(centerX, centerY, width, height);

}

Private static MyPoint getLeftMostPoint(MyPoint[] points) {

MyPoint leftMost = points[0];

For (int I = 0; I < points.length; i++) {

If (leftMost.x > points[i].x) {

leftMost = points[i];

}

}

Return leftMost;

}

Private static MyPoint getRighMostPoint(MyPoint[] points) {

MyPoint rightMost = points[0];

For (int I = 0; I < points.length; i++) {

If (rightMost.x < points[i].x) {

rightMost = points[i];

}

}

Return rightMost;

}

Private static MyPoint getHighestPoint(MyPoint[] points) {

MyPoint highest = points[0];

For (int I = 0; I < points.length; i++) {

If (highest.y < points[i].y) {

Highest = points[i];

}

}

Return highest;

}

Private static MyPoint getLowestPoint(MyPoint[] points) {

MyPoint lowest = points[0];

For (int I = 0; I < points.length; i++) {

If (lowest.y > points[i].y) {

Lowest = points[i];

}

}

Return lowest;

}

Private double getMax(double… n) {

Double max = n[0];

For (int I = 1; I < n.length; i++) {

If (max < n[i]) {

Max = n[i];

}

}

Return max;

}

Private double getMin(double… n) {

Double min = n[0];

For (int I = 1; I < n.length; i++) {

If (min > n[i]) {

Min = n[i];

}

}

Return min;

}

@Override

Public String toString() {

Return “MyRectangle2D{“ +

“centerX=” + centerX +

“, centerY=” + centerY +

“, width=” + width +

“, height=” + height +

‘}’;

}

}

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();

}

Public class Exercise\_09 {

Public static void main(String[] args) {

Circle c1 = new Circle(10);

Circle c2 = new Circle(10);

MyRectangle2D r1 = new MyRectangle2D(0, 0, 5, 5);

System.out.println(“c1 equals c2? “ + c1.equals(c2));

System.out.println(“c1 equals r1? “ + c1.equals(r1));

}

}

Class Circle extends GeometricObject {

Private double radius;

Public Circle() {

}

Public Circle(double radius) {

This.radius = radius;

}

Public Circle(double radius,

String color, boolean filled) {

This.radius = radius;

setColor(color);

setFilled(filled);

}

/\*\* Return radius \*/

Public double getRadius() {

Return radius;

}

/\*\* Set a new radius \*/

Public void setRadius(double radius) {

This.radius = radius;

}

/\*\* Return area \*/

Public double getArea() {

Return radius \* radius \* Math.PI;

}

/\*\* Return diameter \*/

Public double getDiameter() {

Return 2 \* radius;

}

/\*\* Return perimeter \*/

Public double getPerimeter() {

Return 2 \* radius \* Math.PI;

}

@Override

Public boolean equals(Object obj) {

Return obj instanceof Circle && radius == ((Circle) obj).radius;

}

/\* Print the circle info \*/

Public void printCircle() {

System.out.println(“The circle is created “ + getDateCreated() +

“ and the radius is “ + radius);

}

13.11

Public class Octagon extends GeometricObject implements Cloneable {

// all 8 sides of the octagon have equal length

Double side;

Public Octagon() {

This(5);

}

Public Octagon(double side) {

This.side = side;

}

@Override

Public double getArea() {

Return (2.0 + 4.0 / Math.sqrt(2) \* side \* side);

}

@Override

Public double getPerimeter() {

Return side \* 8;

}

Public Object clone() {

Object o = null;

Try {

O = super.clone();

} catch (CloneNotSupportedException ex) {

Ex.printStackTrace();

}

Return o;

}

@Override

Public boolean equals(Object o) {

Return o instanceof Octagon && getArea() == ((Octagon)o).getArea();

}

}

Public class Exercise\_11 {

Public static void main(String[] args) {

Octagon oc1 = new Octagon(10);

Octagon oc2 = (Octagon)oc1.clone();

System.out.println(“compareTo: “ + (oc1.compareTo(oc2) == 0));

System.out.println(“equals: “ + oc1.equals(oc2));

System.out.println(“Have the same reference? “ + (oc1 == oc2));

}

}

13.14

Public class Rational extends Number implements Comparable<Rational> {

// Data fields for numerator and denominator

Private long[] r = new long[2];

Private static int NUMERATOR = 0;

Private static int DENOMINATOR = 1;

/\*\* Default constructor \*/

Public Rational() {

This(0, 1);

}

/\*\* Construct a rational with specified numerator and denominator \*/

Public Rational(long numerator, long denominator) {

Long gcd = gcd(numerator, denominator);

R[NUMERATOR] = ((denominator > 0) ? 1 : -1) \* numerator / gcd;

R[DENOMINATOR] = Math.abs(denominator) / gcd;

}

/\*\* Find GCD of two numbers \*/

Private long gcd(long n, long d) {

Long t1 = Math.abs(n);

Long t2 = Math.abs(d);

Long remainder = t1 % t2;

While (remainder != 0) {

T1 = t2;

T2 = remainder;

Remainder = t1%t2;

}

Return t2;

}

/\*\* Return numerator \*/

Public long getNumerator() {

Return r[NUMERATOR];

}

/\*\* Return denominator \*/

Public long getDenominator() {

Return r[DENOMINATOR];

}

/\*\* Add a rational number to this rational \*/

Public Rational add(Rational secondRational) {

Long n = r[NUMERATOR] \* secondRational.getDenominator() +

R[DENOMINATOR] \* secondRational.getNumerator();

Long d = r[DENOMINATOR] \* secondRational.getDenominator();

Return new Rational(n, d);

}

/\*\* Subtract a rational number from this rational \*/

Public Rational subtract(Rational secondRational) {

Long n = r[NUMERATOR] \* secondRational.getDenominator()

* R[DENOMINATOR] \* secondRational.getNumerator();

Long d = r[DENOMINATOR] \* secondRational.getDenominator();

Return new Rational(n, d);

}

/\*\* Multiply a rational number to this rational \*/

Public Rational multiply(Rational secondRational) {

Long n = r[NUMERATOR] \* secondRational.getNumerator();

Long d = r[DENOMINATOR] \* secondRational.getDenominator();

Return new Rational(n, d);

}

/\*\* Divide a rational number from this rational \*/

Public Rational divide(Rational secondRational) {

Long n = r[NUMERATOR] \* secondRational.getDenominator();

Long d = r[DENOMINATOR] \* secondRational.r[NUMERATOR];

Return new Rational(n, d);

}

@Override

Public String toString() {

If (r[DENOMINATOR] == 1)

Return r[NUMERATOR] + “”;

Else

Return r[NUMERATOR] + “/” + r[DENOMINATOR];

}

/\*\* Override the equals method in the Object class \*/

Public boolean equals(Object parm1) {

Return (this.subtract((Rational) (parm1))).getNumerator() == 0;

}

/\*\* Override the hashCode method in the Object class \*/

Public int hashCode() {

Return new Double(this.doubleValue()).hashCode();

}

/\*\* Override the abstract intValue method in java.lang.Number \*/

Public int intValue() {

Return (int)doubleValue();

}

/\*\* Override the abstract floatValue method in java.lang.Number \*/

Public float floatValue() {

Return (float)doubleValue();

}

/\*\* Override the doubleValue method in java.lang.Number \*/

Public double doubleValue() {

Return r[NUMERATOR] \* 1.0 / r[DENOMINATOR];

}

/\*\* Override the abstract longValue method in java.lang.Number \*/

Public long longValue() {

Return (long)doubleValue();

}

@Override

Public int compareTo(Rational o) {

If ((this.subtract(o)).getNumerator() > 0)

Return 1;

Else if ((this.subtract(o)).getNumerator() < 0)

Return -1;

Else

Return 0;

}

}

Public class Exercise\_14 {

Public static void main(String[] args) {

Rational rational = new Rational(4, 10);

Rational = rational.multiply(new Rational(2, 4));

System.out.println(rational);

}

}

13.15

Import java.math.BigDecimal;

Import java.math.RoundingMode;

Public class BigRational extends Number implements Comparable<BigRational> {

// Data fields for numerator and denominator

Private BigDecimal numerator = BigDecimal.ZERO;

Private BigDecimal denominator = BigDecimal.ONE;

/\*\* Construct a rational with default properties \*/

Public BigRational() {

This(BigDecimal.ZERO, BigDecimal.ONE);

}

/\*\* Construct a rational with specified numerator and denominator \*/

Public BigRational(BigDecimal numerator, BigDecimal denominator) {

BigDecimal gcd = gcd(numerator, denominator);

This.numerator = ((denominator.compareTo(BigDecimal.ZERO) > 0) ? BigDecimal.ONE : new BigDecimal(-1)).multiply(numerator).divide(gcd);

This.denominator = denominator.abs().divide(gcd);

}

Public BigRational(String decimal) {

Int index = (decimal.contains(“.”)) ? decimal.indexOf(‘.’) : decimal.indexOf(‘/’);

BigDecimal d;

BigDecimal n;

// if string is in decimal form

If (decimal.contains(“.”)) {

Int power = decimal.substring(index + 1, decimal.length()).length();

D = new BigDecimal(Math.pow(10,power));

N = new BigDecimal(new StringBuilder(decimal).deleteCharAt(index).toString());

} else {

// if string contains ‘/’

N = new BigDecimal(decimal.substring(0, index));

D = new BigDecimal(decimal.substring(index + 1, decimal.length()));

}

BigDecimal gcd = gcd(n, d);

This.numerator = ((d.compareTo(BigDecimal.ZERO) > 0) ? BigDecimal.ONE : new BigDecimal(-1)).multiply(n).divide(gcd);

This.denominator = d.abs().divide(gcd);

}

/\*\* Find GCD of two numbers \*/

Private static BigDecimal gcd(BigDecimal n, BigDecimal d) {

BigDecimal n1 = n.abs();

BigDecimal n2 = d.abs();

BigDecimal remainder = n1.remainder(n2);

While (remainder.compareTo(BigDecimal.ZERO) > 0) {

N1 = n2;

N2 = remainder;

Remainder = n1.remainder(n2);

}

Return n2;

}

/\*\* Return numerator \*/

Public BigDecimal getNumerator() {

Return numerator;

}

/\*\* Return denominator \*/

Public BigDecimal getDenominator() {

Return denominator;

}

/\*\* Add a rational number to this rational \*/

Public BigRational add(BigRational secondBigRational) {

BigDecimal n1 = numerator.multiply(secondBigRational.getDenominator());

BigDecimal n2 = denominator.multiply(secondBigRational.getNumerator());

BigDecimal n = n1.add(n2);

BigDecimal d = denominator.multiply(secondBigRational.getDenominator());

Return new BigRational(n, d);

}

/\*\* Subtract a rational number from this rational \*/

Public BigRational subtract(BigRational secondBigRational) {

BigDecimal n1 = numerator.multiply(secondBigRational.getDenominator());

BigDecimal n2 = denominator.multiply(secondBigRational.getNumerator());

BigDecimal n = n1.subtract(n2);

BigDecimal d = denominator.multiply(secondBigRational.getDenominator());

Return new BigRational(n, d);

}

/\*\* Multiply a rational number to this rational \*/

Public BigRational multiply(BigRational secondBigRational) {

BigDecimal n = numerator.multiply(secondBigRational.getNumerator());

BigDecimal d = denominator.multiply(secondBigRational.getDenominator());

Return new BigRational(n, d);

}

/\*\* Divide a rational number from this rational \*/

Public BigRational divide(BigRational secondBigRational) {

BigDecimal n = numerator.multiply(secondBigRational.getDenominator());

BigDecimal d = denominator.multiply(secondBigRational.numerator);

Return new BigRational(n, d);

}

@Override

Public String toString() {

If (denominator.equals(BigDecimal.ONE))

Return numerator + “”;

Else

Return numerator + “/” + denominator;

}

@Override // Override the equals method in the Object class

Public boolean equals(Object other) {

If ((this.subtract((BigRational)(other))).getNumerator().equals(BigDecimal.ZERO))

Return true;

Else

Return false;

}

@Override // Implement the abstract intValue method in Number

Public int intValue() {

Return (int)doubleValue();

}

@Override // Implement the abstract floatValue method in Number

Public float floatValue() {

Return (float)doubleValue();

}

@Override // Implement the doubleValue method in Number

Public double doubleValue() {

Return numerator.divide(denominator).doubleValue();

}

Public BigDecimal bigDecimalDouble() {

Return numerator.divide(denominator, 100, RoundingMode.HALF\_DOWN);

}

@Override // Implement the abstract longValue method in Number

Public long longValue() {

Return (long)doubleValue();

}

@Override // Implement the compareTo method in Comparable

Public int compareTo(BigRational o) {

If (this.subtract(o).getNumerator().compareTo(BigDecimal.ZERO) > 0)

Return 1;

Else if (this.subtract(o).getNumerator().compareTo(BigDecimal.ZERO) < 0)

Return -1;

Else

Return 0;

}

}

Import java.math.BigDecimal;

Public class Exercise\_15 {

Public static void main(String[] args) {

BigRational bigRational = new BigRational(new BigDecimal(4), new BigDecimal(8));

System.out.println(bigRational);

bigRational = bigRational.multiply(bigRational);

System.out.println(bigRational);

}

}

13.19

Import java.math.BigDecimal;

Import java.math.RoundingMode;

Public class BigRational extends Number implements Comparable<BigRational> {

// Data fields for numerator and denominator

Private BigDecimal numerator = BigDecimal.ZERO;

Private BigDecimal denominator = BigDecimal.ONE;

/\*\* Construct a rational with default properties \*/

Public BigRational() {

This(BigDecimal.ZERO, BigDecimal.ONE);

}

/\*\* Construct a rational with specified numerator and denominator \*/

Public BigRational(BigDecimal numerator, BigDecimal denominator) {

BigDecimal gcd = gcd(numerator, denominator);

This.numerator = ((denominator.compareTo(BigDecimal.ZERO) > 0) ? BigDecimal.ONE : new BigDecimal(-1)).multiply(numerator).divide(gcd);

This.denominator = denominator.abs().divide(gcd);

}

Public BigRational(String decimal) {

Int index = (decimal.contains(“.”)) ? decimal.indexOf(‘.’) : decimal.indexOf(‘/’);

BigDecimal d;

BigDecimal n;

// if string is in decimal form

If (decimal.contains(“.”)) {

Int power = decimal.substring(index + 1, decimal.length()).length();

D = new BigDecimal(Math.pow(10,power));

N = new BigDecimal(new StringBuilder(decimal).deleteCharAt(index).toString());

} else {

// if string contains ‘/’

N = new BigDecimal(decimal.substring(0, index));

D = new BigDecimal(decimal.substring(index + 1, decimal.length()));

}

BigDecimal gcd = gcd(n, d);

This.numerator = ((d.compareTo(BigDecimal.ZERO) > 0) ? BigDecimal.ONE : new BigDecimal(-1)).multiply(n).divide(gcd);

This.denominator = d.abs().divide(gcd);

}

/\*\* Find GCD of two numbers \*/

Private static BigDecimal gcd(BigDecimal n, BigDecimal d) {

BigDecimal n1 = n.abs();

BigDecimal n2 = d.abs();

BigDecimal remainder = n1.remainder(n2);

While (remainder.compareTo(BigDecimal.ZERO) > 0) {

N1 = n2;

N2 = remainder;

Remainder = n1.remainder(n2);

}

Return n2;

}

/\*\* Return numerator \*/

Public BigDecimal getNumerator() {

Return numerator;

}

/\*\* Return denominator \*/

Public BigDecimal getDenominator() {

Return denominator;

}

/\*\* Add a rational number to this rational \*/

Public BigRational add(BigRational secondBigRational) {

BigDecimal n1 = numerator.multiply(secondBigRational.getDenominator());

BigDecimal n2 = denominator.multiply(secondBigRational.getNumerator());

BigDecimal n = n1.add(n2);

BigDecimal d = denominator.multiply(secondBigRational.getDenominator());

Return new BigRational(n, d);

}

/\*\* Subtract a rational number from this rational \*/

Public BigRational subtract(BigRational secondBigRational) {

BigDecimal n1 = numerator.multiply(secondBigRational.getDenominator());

BigDecimal n2 = denominator.multiply(secondBigRational.getNumerator());

BigDecimal n = n1.subtract(n2);

BigDecimal d = denominator.multiply(secondBigRational.getDenominator());

Return new BigRational(n, d);

}

/\*\* Multiply a rational number to this rational \*/

Public BigRational multiply(BigRational secondBigRational) {

BigDecimal n = numerator.multiply(secondBigRational.getNumerator());

BigDecimal d = denominator.multiply(secondBigRational.getDenominator());

Return new BigRational(n, d);

}

/\*\* Divide a rational number from this rational \*/

Public BigRational divide(BigRational secondBigRational) {

BigDecimal n = numerator.multiply(secondBigRational.getDenominator());

BigDecimal d = denominator.multiply(secondBigRational.numerator);

Return new BigRational(n, d);

}

@Override

Public String toString() {

If (denominator.equals(BigDecimal.ONE))

Return numerator + “”;

Else

Return numerator + “/” + denominator;

}

@Override // Override the equals method in the Object class

Public boolean equals(Object other) {

If ((this.subtract((BigRational)(other))).getNumerator().equals(BigDecimal.ZERO))

Return true;

Else

Return false;

}

@Override // Implement the abstract intValue method in Number

Public int intValue() {

Return (int)doubleValue();

}

@Override // Implement the abstract floatValue method in Number

Public float floatValue() {

Return (float)doubleValue();

}

@Override // Implement the doubleValue method in Number

Public double doubleValue() {

Return numerator.divide(denominator).doubleValue();

}

Public BigDecimal bigDecimalDouble() {

Return numerator.divide(denominator, 100, RoundingMode.HALF\_DOWN);

}

@Override // Implement the abstract longValue method in Number

Public long longValue() {

Return (long)doubleValue();

}

@Override // Implement the compareTo method in Comparable

Public int compareTo(BigRational o) {

If (this.subtract(o).getNumerator().compareTo(BigDecimal.ZERO) > 0)

Return 1;

Else if (this.subtract(o).getNumerator().compareTo(BigDecimal.ZERO) < 0)

Return -1;

Else

Return 0;

}

}

Import java.util.Scanner;

Public class Exercise\_19 {

Public static void main(String[] args) {

Scanner input = new Scanner(System.in);

System.out.print(“Enter a decimal number: “);

String s = input.next();

System.out.println(“The fractional number is “ + new BigRational(s));

}

}

Mini Project

​Public​ ​class​ ​Exercise\_13\_05​ {

​        ​/\*​\* Main method ​\*/

​        ​Public​ ​static​ ​void​ ​main​(​String​[] ​args​) {

​                ​//​ Create two Circle objects

​                ​Circle​ circle1 ​=​ ​new​ ​Circle​(​15​, ​“​red​”​, ​true​);

​                ​Circle​ circle2 ​=​ ​new​ ​Circle​(​10​, ​“​blue​”​, ​false​);

​                ​//​ Display circle1

​                ​System​.​out​.​println(​“​\n​Circle 1: ​“​);

​                Print(circle1);

​                ​//​ Display circle2

​                ​System​.​out​.​println(​“​\n​Circle 2: ​“​);

​                Print(circle2);

​                ​//​ Display larger circle

​                Print(​“​\n​The larger of the two circles was ​“​);

​                Print(​Circle​.​max(circle1, circle2));

​                ​//​ Create two Rectangle objects

​                ​Rectangle​ rectangle1 ​=​ ​new​ ​Rectangle​(​4​, ​5​, ​“​green​”​, ​true​);

​                ​Rectangle​ rectangle2 ​=​ ​new​ ​Rectangle​(​4.2​, ​5​, ​“​orange​”​, ​true​);

​                ​//​ Display circle1

​                ​System​.​out​.​println(​“​\n​Rectangle 1: ​“​);

​                Print(circle1);

​                ​//​ Display circle2

​                ​System​.​out​.​println(​“​\n​Rectangle 2: ​“​);

​                Print(circle2);

​                ​//​ Display larger circle

​                Print(​“​\n​The larger of the two rectangles was ​“​);

​                Print(​Rectangle​.​max(rectangle1, rectangle2));

​        }

​        ​//​ Displays a string

​        ​Public​ ​static​ ​void​ ​print​(​String​ ​s​) {

​                ​System​.​out​.​println(s);

​        }

​        ​//​ Displays a GeometricObject

​        ​Public​ ​static​ ​void​ ​print​(​GeometricObject​ ​o​) {

​                ​System​.​out​.​println(o);

​        }