“Programming Technology”
Documentation to Assignment 1 task 7

Made by: Illia Takhtamyshev

Neptun code: RP0KRP

Group: 05

E-mail: takhtamyshev17@gmail.com

Contents

Task 4

Class diagram 4

Short description of methods 8

Shape 8

1. Constructor “public Shape(double x, double y)” 8
2. Abstract method “public abstract double getArea()” 8
3. Abstract Method “public abstract ArrayList<Double> getBoundingBox()” 8
4. Method “public String toString()” 8

Circle 8

1. Constructor “public Circle(double x, double y, double radius)” 8
2. Method “public double getArea()” 8
3. Method “public ArrayList<Double> getBoundingBox()” 9
4. Method “public String toString()” 9

Triangle 9

1. Constructor “public Triangle(double x, double y, double sideLength)” ...9
2. Method “public double getArea()” 9
3. Method “public ArrayList<Double> getBoundingBox()” 9
4. Method “public String toString()” 9

Square 10

1. Constructor “public Square(double x, double y, double sideLength)” 10
2. Method “public double getArea()” 10
3. Method “public ArrayList<Double> getBoundingBox()” 10
4. Method “public String toString()” 10

Hexagon 10

1. Constructor “public Hexagon(double x, double y, double sideLength)” 10

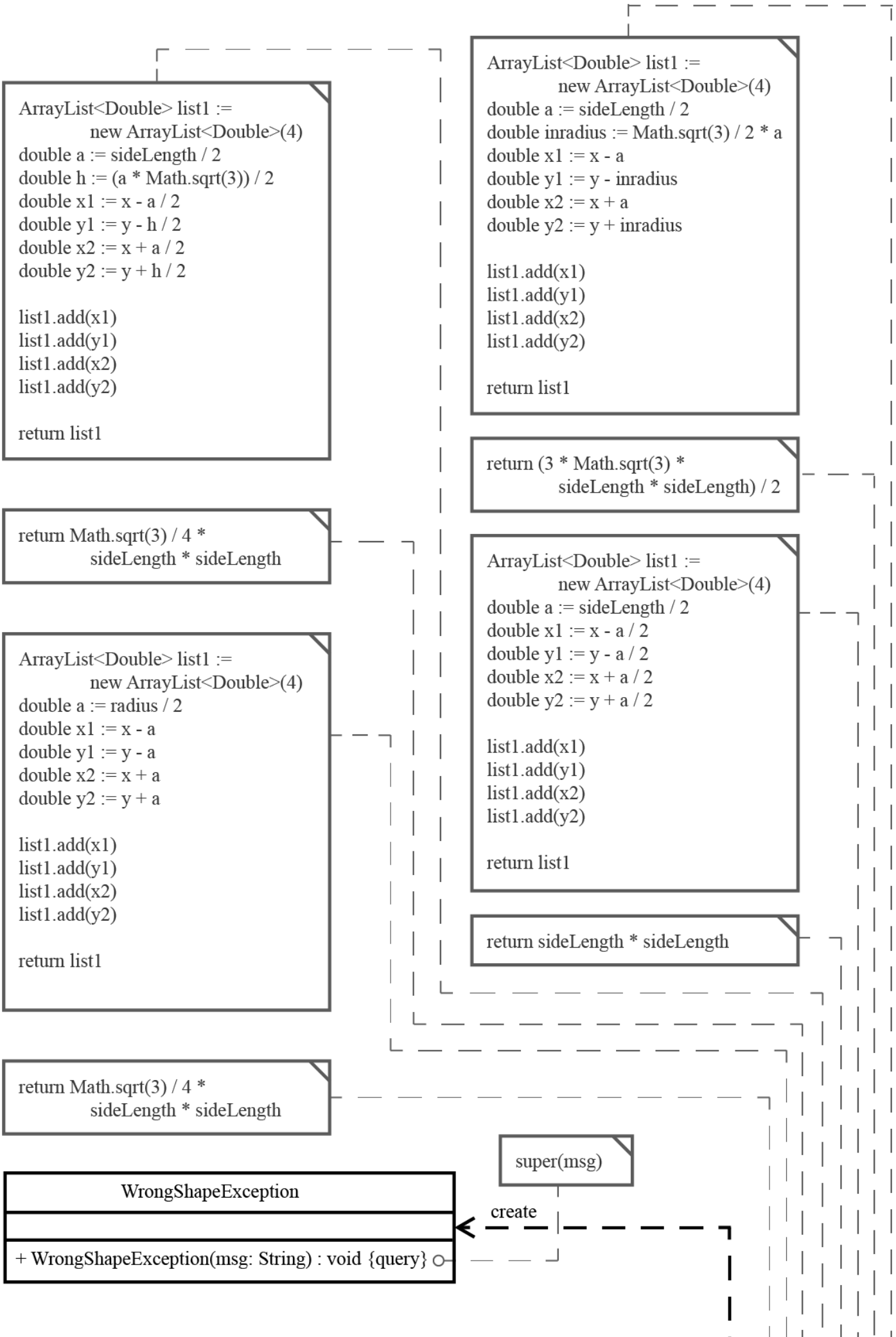
2. Method “public double getArea()” 10
3. Method “public ArrayList<Double> getBoundingBox()”.....10
4. Method “public String toString()” 11
BoundingBox 11
1. Constructor “public BoundingBox()” 11
2. Method “public void readInput(String fileName) throws WrongShapeException”.....11
3. Method “ArrayList<Double> getCommonBoundingBox()”.....11
4. Method “public void printShapes()” 11
WrongShapeException 11
1. Constructor “public WrongShapeException(String msg)”.....11
Testing.....12
1. Test with 4 different shapes (in1.txt) 12
Input: 12
Output: 12
Illustration: 12
2. Test with 3 hexagons (in2.txt).....13
Input: 13
Output: 13
Illustration: 13
3. Test with 5 shapes (2 of which are circles) (in3.txt).....14
Input: 14
Output: 14
Illustration: 14

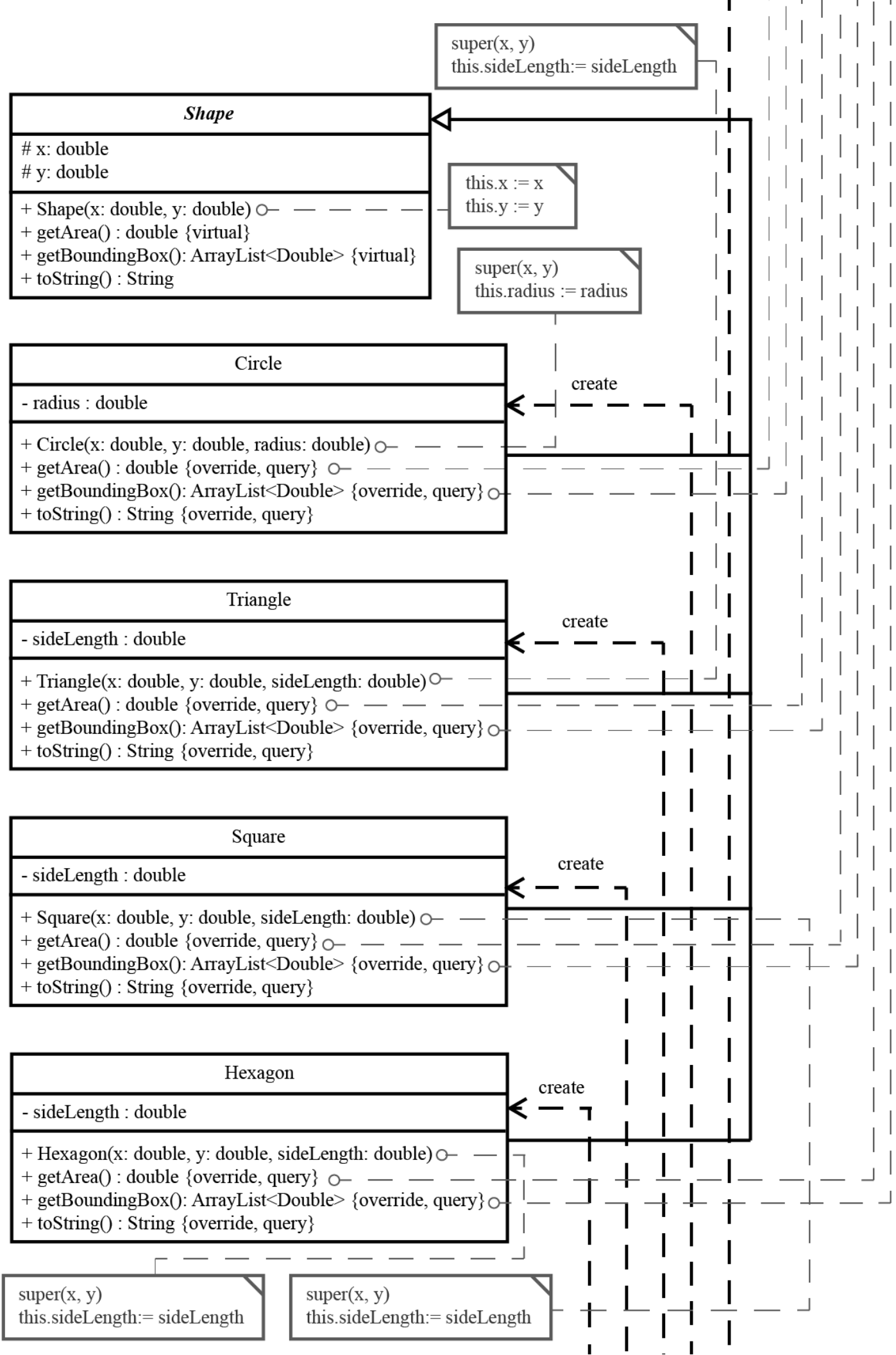
Task

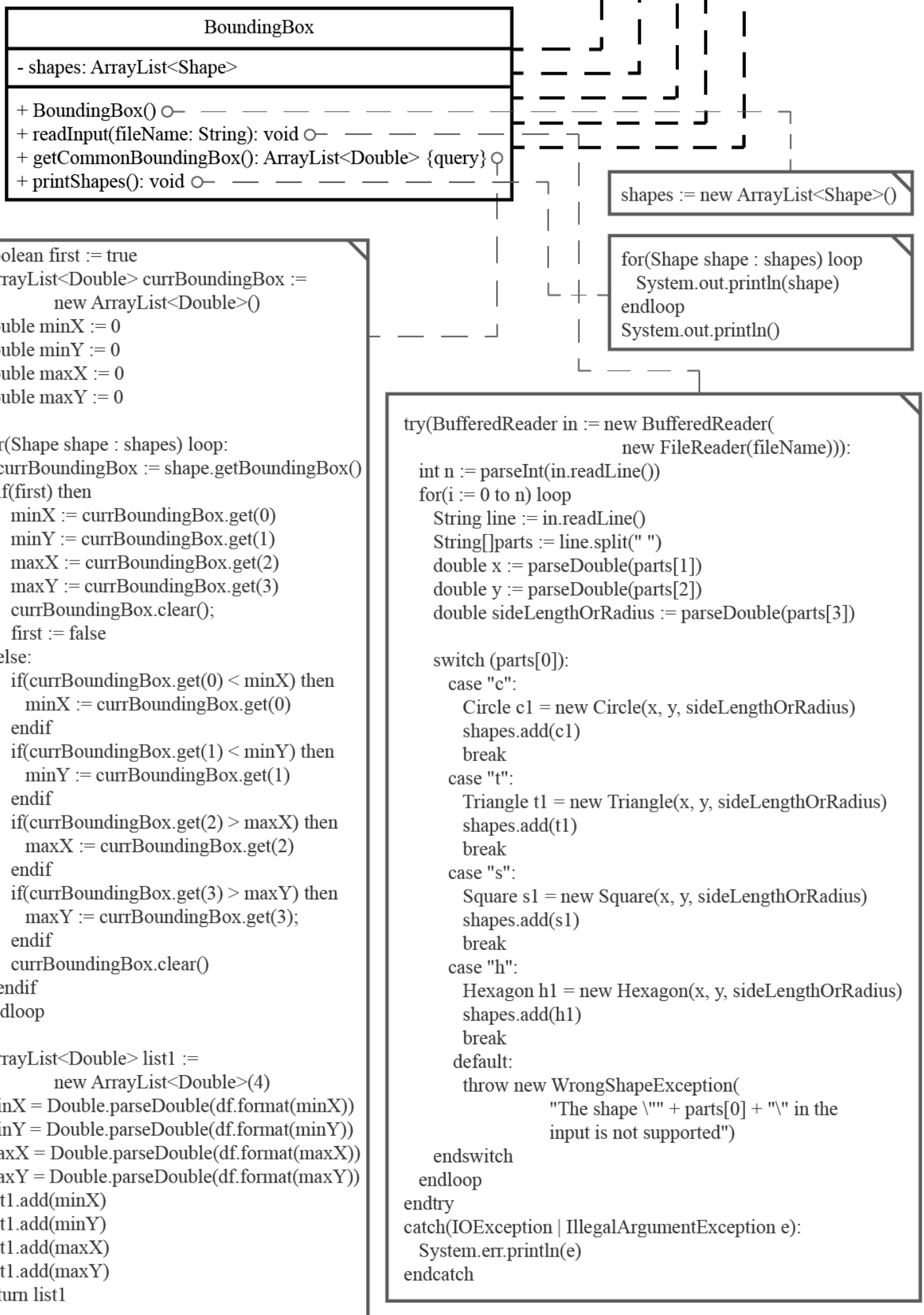
Fill a collection with several regular shapes (circle, regular triangle, square, regular hexagon).
Determine the smallest bounding box, which contains all the shapes, and its sides parallel with an x or y axis.

Each shape can be represented by its center and side length (or radius), if we assume that one side of the polygons are parallel with x axis, and its nodes lies on or above this side. Load and create the shapes from a text file. The first line of the file contains the number of the shapes, and each following line contain a shape. The first character will identify the type of the shape, which is followed by the center coordinate and the side length or radius. Manage the shapes uniformly, so derive them from the same super class.

Class diagram


ArrayList<Double> list1 :=
new ArrayList<Double>(4)
double a := sideLength / 2
double h := (a * Math.sqrt(3)) / 2
double x1 := x - a / 2
double y1 := y - h / 2
double x2 := x + a / 2
double y2 := y + h / 2
list1.add(x1)
list1.add(y1)
list1.add(x2)
list1.add(y2)
return list1
ArrayList<Double> list1 :=
new ArrayList<Double>(4)
double a := sideLength / 2
double inradius := Math.sqrt(3) / 2 * a
double x1 := x - a
double y1 := y - inradius
double x2 := x + a
double y2 := y + inradius
list1.add(x1)
list1.add(y1)
list1.add(x2)
list1.add(y2)
return list1
return Math.sqrt(3) / 4 *
sideLength * sideLength
return (3 * Math.sqrt(3) *
sideLength * sideLength) / 2
ArrayList<Double> list1 :=
new ArrayList<Double>(4)
double a := radius / 2
double x1 := x - a
double y1 := y - a
double x2 := x + a
double y2 := y + a
list1.add(x1)
list1.add(y1)
list1.add(x2)
list1.add(y2)
return list1
ArrayList<Double> list1 :=
new ArrayList<Double>(4)
double a := sideLength / 2
double x1 := x - a / 2
double y1 := y - a / 2
double x2 := x + a / 2
double y2 := y + a / 2
list1.add(x1)
list1.add(y1)
list1.add(x2)
list1.add(y2)
return list1
return sideLength * sideLength
return Math.sqrt(3) / 4 *
sideLength * sideLength
super(msg)
WrongShapeException
create
+ WrongShapeException(msg: String) : void {query}


super(x, y)
this.sideLength:= sideLength
Shape
x: double
y: double
+ Shape(x: double, y: double)
+ getArea() : double {virtual}
+ getBoundingBox(): ArrayList<Double> {virtual}
+ toString() : String
this.x := x
this.y := y
super(x, y)
this.radius := radius
Circle
- radius : double
+ Circle(x: double, y: double, radius: double)
+ getArea() : double {override, query}
+ getBoundingBox(): ArrayList<Double> {override, query}
+ toString() : String {override, query}
create
Triangle
- sideLength : double
+ Triangle(x: double, y: double, sideLength: double)
+ getArea() : double {override, query}
+ getBoundingBox(): ArrayList<Double> {override, query}
+ toString() : String {override, query}
create
Square
- sideLength : double
+ Square(x: double, y: double, sideLength: double)
+ getArea() : double {override, query}
+ getBoundingBox(): ArrayList<Double> {override, query}
+ toString() : String {override, query}
create
Hexagon
- sideLength : double
+ Hexagon(x: double, y: double, sideLength: double)
+ getArea() : double {override, query}
+ getBoundingBox(): ArrayList<Double> {override, query}
+ toString() : String {override, query}
create
super(x, y)
this.sideLength:= sideLength
super(x, y)
this.sideLength:= sideLength


BoundingBox
- shapes: ArrayList<Shape>
+ BoundingBox()
+ readInput(fileName: String): void
+ getCommonBoundingBox(): ArrayList<Double> {query}
+ printShapes(): void
shapes := new ArrayList<Shape>()
for(Shape shape : shapes) loop
System.out.println(shape)
endloop
System.out.println()
boolean first := true
ArrayList<Double> currBoundingBox :=
new ArrayList<Double>()
double minX := 0
double minY := 0
double maxX := 0
double maxY := 0
for(Shape shape : shapes) loop:
currBoundingBox := shape.getBoundingBox()
if(first) then
minX := currBoundingBox.get(0)
minY := currBoundingBox.get(1)
maxX := currBoundingBox.get(2)
maxY := currBoundingBox.get(3)
currBoundingBox.clear();
first := false
else:
if(currBoundingBox.get(0) < minX) then
minX := currBoundingBox.get(0)
endif
if(currBoundingBox.get(1) < minY) then
minY := currBoundingBox.get(1)
endif
if(currBoundingBox.get(2) > maxX) then
maxX := currBoundingBox.get(2)
endif
if(currBoundingBox.get(3) > maxY) then
maxY := currBoundingBox.get(3);
endif
currBoundingBox.clear()
endif
endloop
ArrayList<Double> list1 :=
new ArrayList<Double>(4)
minX = Double.parseDouble(df.format(minX))
minY = Double.parseDouble(df.format(minY))
maxX = Double.parseDouble(df.format(maxX))
maxY = Double.parseDouble(df.format(maxY))
list1.add(minX)
list1.add(minY)
list1.add(maxX)
list1.add(maxY)
return list1
try(BufferedReader in := new BufferedReader(
new FileReader(fileName))):
int n := parseInt(in.readLine())
for(i := 0 to n) loop
String line := in.readLine()
String[]parts := line.split(" ")
double x := parseDouble(parts[1])
double y := parseDouble(parts[2])
double sideLengthOrRadius := parseDouble(parts[3])
switch (parts[0]):
case "c":
Circle c1 = new Circle(x, y, sideLengthOrRadius)
shapes.add(c1)
break
case "t":
Triangle t1 = new Triangle(x, y, sideLengthOrRadius)
shapes.add(t1)
break
case "s":
Square s1 = new Square(x, y, sideLengthOrRadius)
shapes.add(s1)
break
case "h":
Hexagon h1 = new Hexagon(x, y, sideLengthOrRadius)
shapes.add(h1)
break
default:
throw new WrongShapeException(
"The shape \"" + parts[0] + "\" in the
input is not supported")
endswitch
endloop
endtry
catch(IOException | IllegalArgumentException e):
System.err.println(e)
endcatch

Short description of methods

Shape

1. **Constructor “public Shape(double x, double y)”**

- Constructor for class Shape
- Takes two parameters which represent coordinates of the center
- Initializes “x” and “y” variables with values which are passed as arguments

2. **Abstract method “public abstract double getArea()”**

- Abstract method declared without implementation
- Subclasses which extend Shape class must provide own implementations of the method
- It will calculate area of the shapes represented by the subclasses

3. **Abstract Method “public abstract ArrayList<Double> getBoundingBox()”**

- Abstract method declared without implementation
- Subclasses which extend Shape class must provide own implementations of the method
- It will calculate bounding box of the shapes represented by the subclasses

4. **Method “public String toString()”**

- Overridden method of Object class
- Provides string representation of an object
- It returns a string containing the class name of the specific shape with the center coordinates

Circle

1. **Constructor “public Circle(double x, double y, double radius)”**

- Constructor for Circle class which inherits from Shape
- Takes three parameters. Two for center coordinate and one for radius
- Calls the parent constructor using “super” and initializes radius

2. **Method “public double getArea()”**

- Overrides abstract method getArea() inherited from Shape

- Calculates and returns area of the circle

3. **Method "public ArrayList<Double> getBoundingBox()"**

- Overrides abstract method getBoundingBox() inherited from Shape
- Calculates and returns bounding box of the circle
- Bounding box is represented by ArrayList of Double values. First two elements are "x" and "y" for the bottom-left point and second two values are "x" and "y" for the top-right point.

4. **Method "public String toString()"**

- Overrides method toString() inherited from Shape
- Returns string representation of the Circle class

Triangle

1. **Constructor "public Triangle(double x, double y, double sideLength)"**

- Constructor for Triangle class which inherits from Shape
- Takes three parameters. Two for center coordinate and one for side length
- Calls the parent constructor using "super" and initializes side length

2. **Method "public double getArea()"**

- Overrides abstract method getArea() inherited from Shape
- Calculates and returns area of the triangle

3. **Method "public ArrayList<Double> getBoundingBox()"**

- Overrides abstract method getBoundingBox() inherited from Shape
- Calculates and returns bounding box of the triangle
- Bounding box is represented by ArrayList of Double values. First two elements are "x" and "y" for the bottom-left point and second two values are "x" and "y" for the top-right point.

4. **Method "public String toString()"**

- Overrides method toString() inherited from Shape
- Returns string representation of the Triangle class

Square

1. **Constructor "public Square(double x, double y, double sideLength)"**

- Constructor for Square class which inherits from Shape
- Takes three parameters. Two for center coordinate and one for side length
- Calls the parent constructor using "super" and initializes side length

2. **Method "public double getArea()"**

- Overrides abstract method getArea() inherited from Shape
- Calculates and returns area of the square

3. **Method "public ArrayList<Double> getBoundingBox()"**

- Overrides abstract method getBoundingBox() inherited from Shape
- Calculates and returns bounding box of the square
- Bounding box is represented by ArrayList of Double values. First two elements are "x" and "y" for the bottom-left point and second two values are "x" and "y" for the top-right point.

4. **Method "public String toString()"**

- Overrides method toString() inherited from Shape
- Returns string representation of the Square class

Hexagon

1. **Constructor "public Hexagon(double x, double y, double sideLength)"**

- Constructor for Hexagon class which inherits from Shape
- Takes three parameters. Two for center coordinate and one for side length
- Calls the parent constructor using "super" and initializes side length

2. **Method "public double getArea()"**

- Overrides abstract method getArea() inherited from Shape
- Calculates and returns area of the hexagon

3. **Method "public ArrayList<Double> getBoundingBox()"**

- Overrides abstract method getBoundingBox() inherited from Shape
- Calculates and returns bounding box of the hexagon
- Bounding box is represented by ArrayList of Double values. First two elements are "x" and "y" for the bottom-left point and second two values are "x" and "y" for the top-right point.

4. **Method “public String toString()”**

- Overrides method toString() inherited from Shape
- Returns string representation of the Hexagon class

BoundingBox

1. **Constructor “public BoundingBox()”**

- Constructor for BoundingBox class
- Initializes ArrayList with name “shapes” for storing shapes

2. **Method “public void readInput(String fileName) throws WrongShapeException”**

- Reads input data from a file. File name is passed as a parameter
- It expects that on the first line of the file there will be number of shapes and after that shapes line by line. The first character will be type of the shape, then x and y coordinates of its center and radius or side length.
- It creates instances of Circle, Triangle, Square and Hexagon depending on the type of the shape. After that it adds it to the “shapes” list.
- Throws “WrongShapeException” if there is unsupported shape

3. **Method “ArrayList<Double> getCommonBoundingBox()”**

- Calculates common bounding box for all the shapes in the “shapes” list
- It iterates from shape to shape getting their bounding box coordinates and updates min and max values of “x” and “y” coordinates
- The bounding box is represented as ArrayList of Double values where first two are minimum “x” and “y” for bottom-left point and second two are maximum “x” and “y” for top-right point

4. **Method “public void printShapes()”**

- Iterates through shapes list and prints string representation of every shape

WrongShapeException

1. **Constructor “public WrongShapeException(String msg)”**

- Constructor for WrongShapeException class
- It takes parameter “msg” which represents an error
- Calls “super” constructor of its parent “Exception”

Testing

1. Test with 4 different shapes (in1.txt)

Input:

4

t 3 4 6

c 7 1 5

s 7 7 4

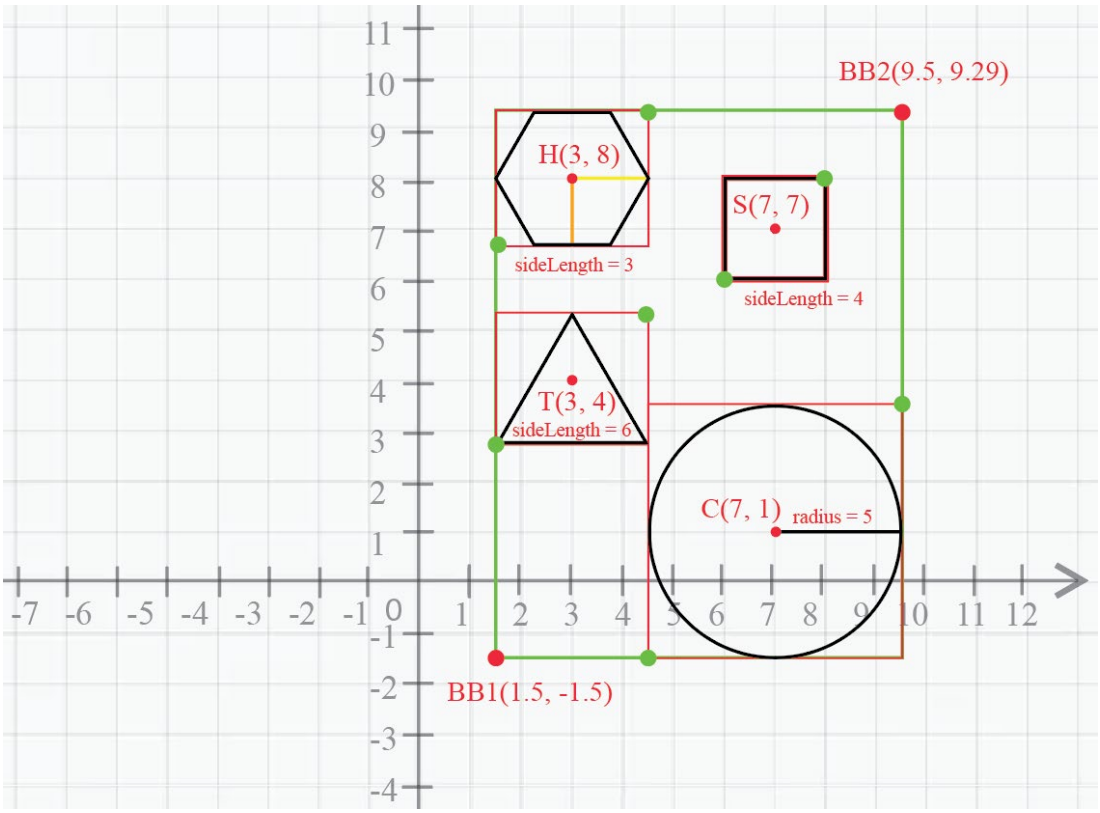
h 3 8 3

Output:

BB1(1.5, -1.5)

BB2(9.5, 9.29)

Illustration:



2. Test with 3 hexagons (in2.txt)

Input:

3

h -4 4 6

h 5 7 4

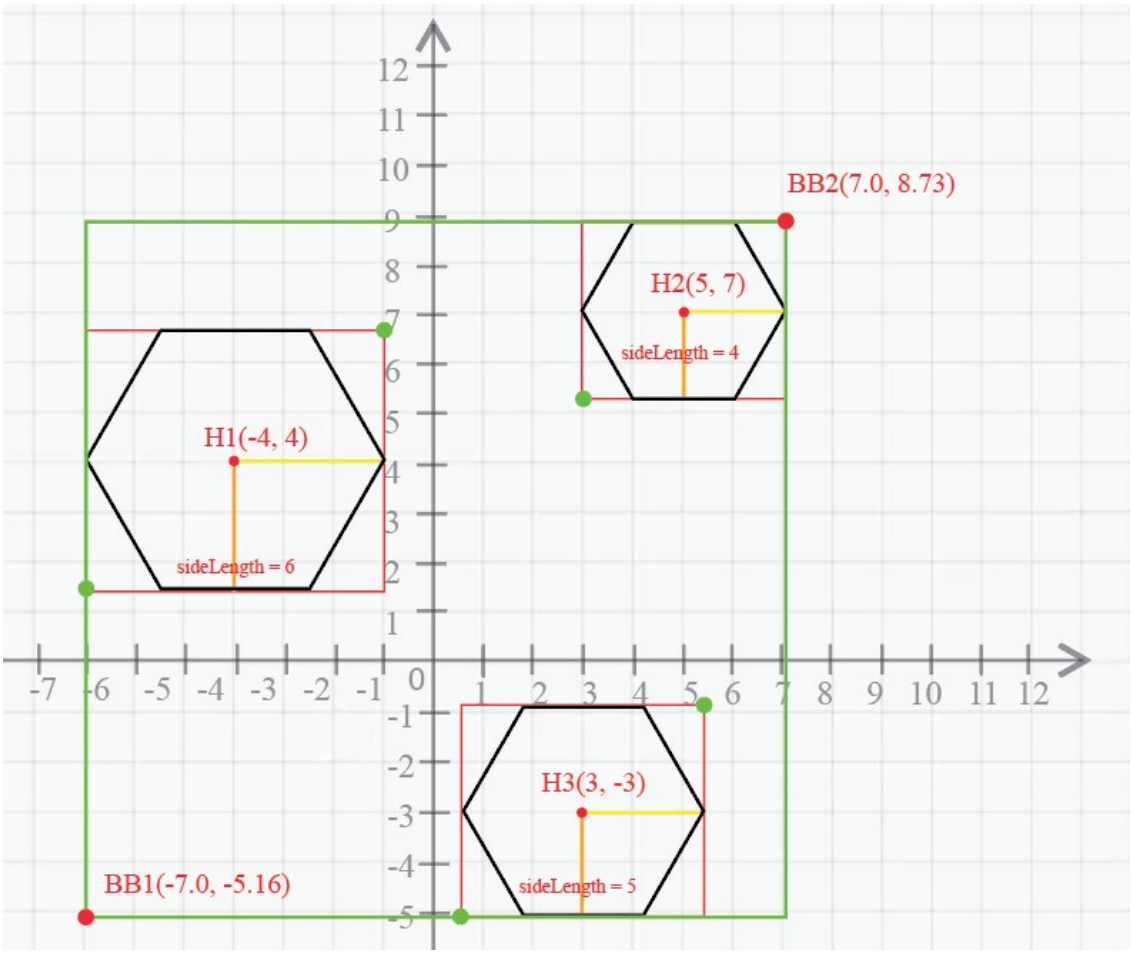
h 3 -3 5

Output:

BB1(-7.0, -5.16)

BB2(7.0, 8.73)

Illustration:



3. Test with 5 shapes (2 of which are circles) (in3.txt)

Input:

5

c -1 6 6

c 4 7 7

t 6 3 9

h -1 -3 4

s 5 -3 8

Output:

BB1(-4.0, -5.0)

BB2(8.25, 10.5)

Illustration: