**ShapeTwoD class**

**Text

Description automatically generated**

**Text

Description automatically generated**

The default constructor and the user defined constructor is initialized. All the appropriate mandatory functions are also included in the base ShapeTwoD class. The functions computeArea, isPointInShape and isPointOnShape are to be overridden by the subclasses. We have an additional static function called getTFValue that converts the user input string (WS or NS) into a Boolean type ( 1 or 0). This function can be called without having any ShapeTwoD objects created.

**Rectangle, Square, Cross SubClass**

Text

Description automatically generated

Instance variables xValues and yValues are vectors that contain the vertices of the user inputs. The vertex inputs are manually inputted through the readIn function that is called in the main.

The variable area allows us to return the area in the getArea function. Since the number of vertex will not change, we set it at a const 4. The computeArea function takes the 4 user input coordinates and sorts the x and y coordinates in ascending order. I then take the highest x and highest y values and subtract the lowest x and lowest y value respectively. This gives me the length and breadth of the rectangle. It does not matter which is the length/breadth. Once I get the 2 lengths, I multiply them to get the area of the rectangle.

Text

Description automatically generated

For the perimeter calculation, I use an approach that involved checking segments of the lines. Since a rectangle is a closed polygon. I loop through the 4 coordinates and check the distance between them. If the coordinates along the segment is unbroken then I do not have a break in my segment, however if the coordinates are broken, eg(1,1)----(3,1). I have a break in my line segment, I then have if-else statements to check if it is the x or y value that needs to increase/decrease to fill up the gap. All the gaps are added into a vector that is returned to my class instance variable.

Text

Description automatically generated

For my area calculation, I get together the biggest x and y value and the smallest x and y value. This gives me the top left and the bottom right coordinate of the rectangle, I then loop through every coordinate from this range and add them into a vector. From the vector, I use a for loop with an iterator to erase all the user inputted values from the vector that contains all the coordinates. This leaves me with only the coordinates within the rectangle which are the coordinates in the shape.

All the computing are done when the object is created. The constructor calls all the functions to do the computing.

The isInShape and isOnShape uses the vectors that contain the area coordinates and the perimeter coordinates and checks if the int x and int y values are inside. It then returns the Boolean type.

My getAllCoords, getPCoords and getACoords are functions to help with my shapes report. They take the vectors that contain the appropriate coordinates and format them for output.

My Square and Cross subclasses follow very similar computing. The way I compute the area and perimeter is through the same segmentation technique and it can apply to any of these polygons.

**Circle SubClass**

Text

Description automatically generated

For the calculation of the circle perimeter, I just have to take the NSEW coordinates from the centre. This is simple with the given radius, I just need to add and subtract the radius from the correct x and y values.

For the circle area calculation, I take the top left and bottom right coordinates to get the box the circle is within. I then loop through the box and compare the coordinates with the radius of the circle. Since the north south east and west are the extreme ends, if they are not within this lines then the coordinate will be out of the circle area.

As according to the previous functions, getPCoords, getACoords and centreRadius are to help with the formatting and output of the shape report.

**optionOne**

**Text

Description automatically generated**

The first option takes in the shape and type that the user wants to input. It creates the shape object, then the constructor calls the readIn function so that the user can input the vertex coordinates. The shape objects are stored in 4 different vectors that are class specific.

**optionTwo**

Text

Description automatically generated

The option 2 simply goes through all the objects created and calls the ComputeArea for each object.

**optionThree**

**Text

Description automatically generated**

Printing the shape reports for each individual object. The toString method is called first to display the name and type of the Shape. GetAllCoords then displays all the vertex that the user had inputted. getPCoords displays the perimeter only and getACoords displays the area only. This is consistent for all the shapes.

**optionFour**

Text

Description automatically generated

For option 4a. I take in all the areas of the shapes and put it into a vector. I then sort the area in ascending order. I then make a for loop that goes through all the areas and checks through every shape vector whether the area matches. If it matches, it outputs the shape information.

Text

Description automatically generated

For option 4b, it is the exact same as 4a just with the sort function in descending order.

Text

Description automatically generated

For 4c I assume that warpspace is sorted NS followed by WS and the area is sorted in ascending order. I first order all the different shapes according to their area. Then I run a for loop twice, one to display the normal space and then the next one to display the warp space.