



## Exercise 4

### Part 1: Repetition of OOP basics - A simple Point class

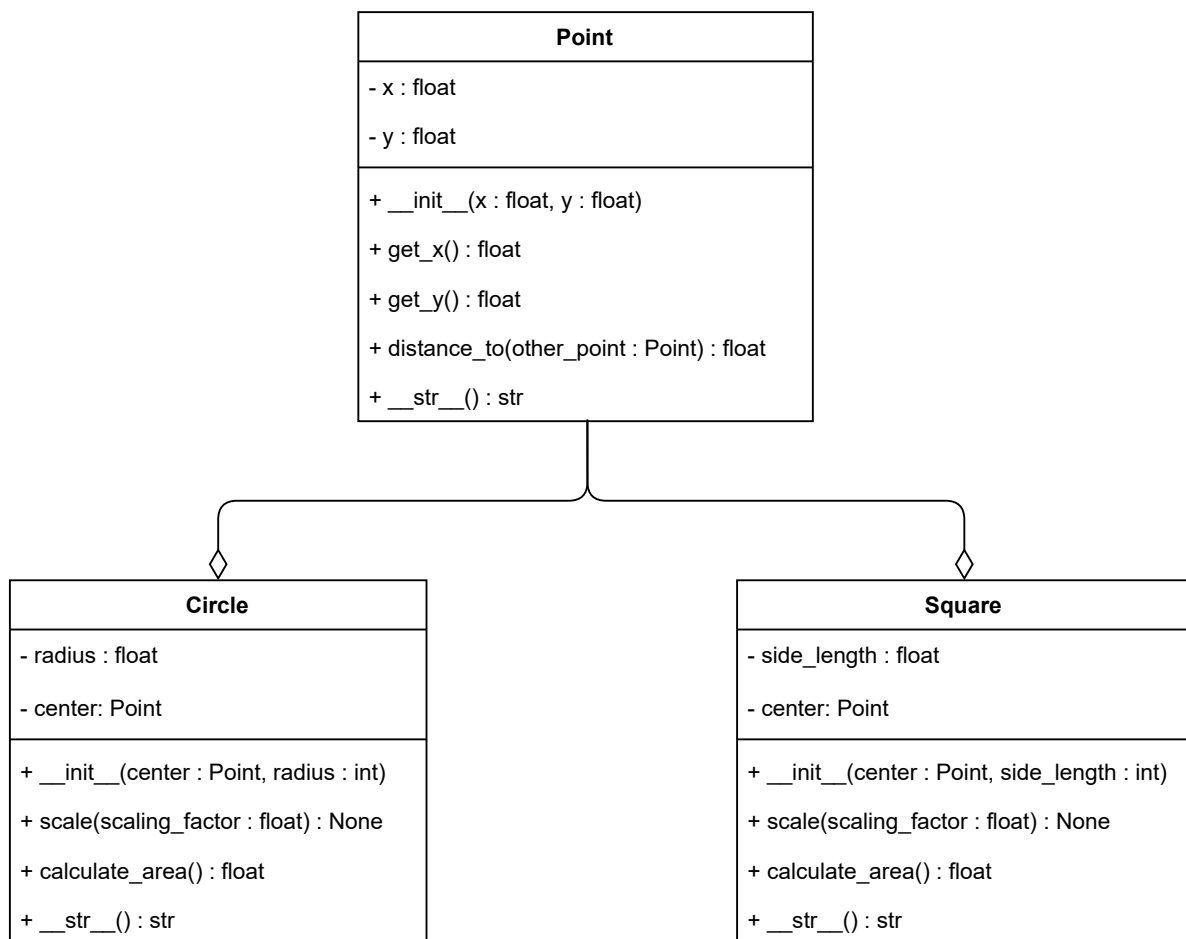
In this first part of the exercise we will do a small recap of what you learned about object-oriented-programming by creating a simple Point class whose attributes and methods are given in the UML listed below. We will implement this class together.

Point
- x : float - y : float
+ __init__(x : float, y : float) + get_x() : float + get_y() : float + distance_to(other_point : Point) : float + __str__() : str

## Part 2: Try it on your own - Implementation of Circle and Square classes

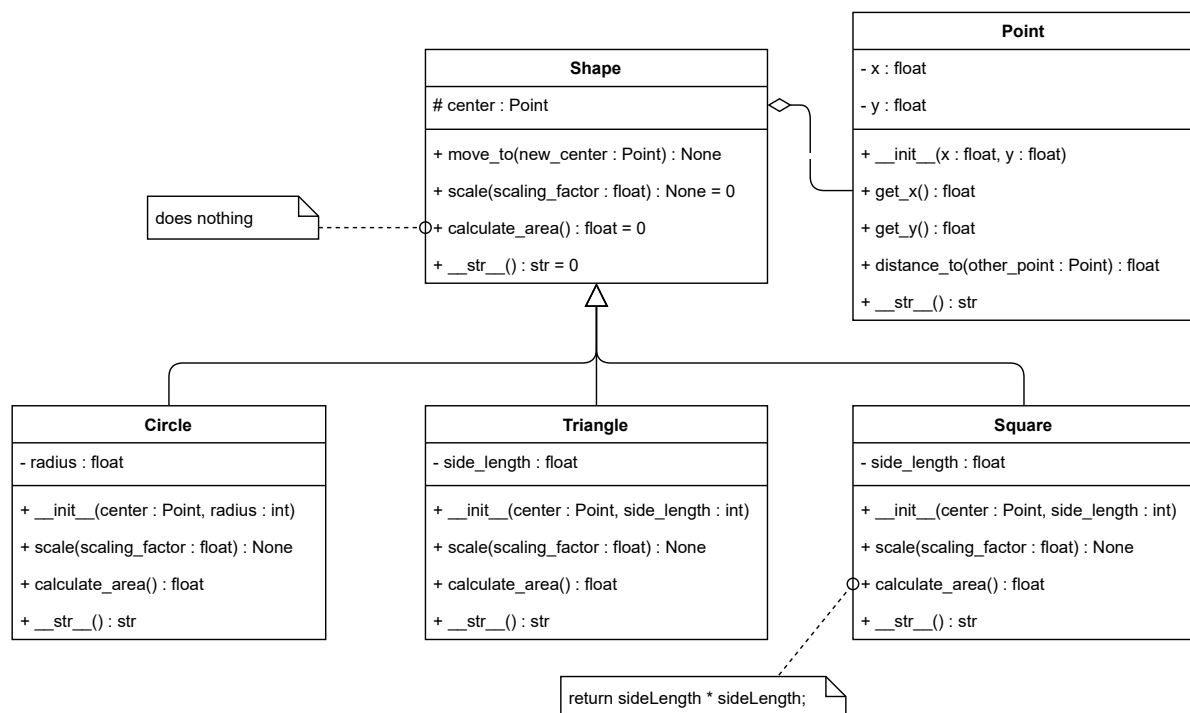
In this part of the exercise we want to implement a Circle and a Square class. In order to draw geometrical objects (e.g. within a CAD program) every of them needs a center point to be clearly defined inside the related coordinate system. This is why each object from type Circle or Square needs to be instantiated with a Point inside the constructor's argument.

Your task is to create the classes Circle and Square in combination with the Point class. Use the UML-Diagram to get an overview of the relations between these classes and for the required attributes and methods to be implemented.

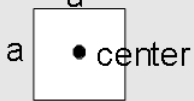
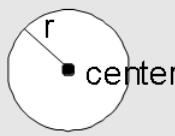
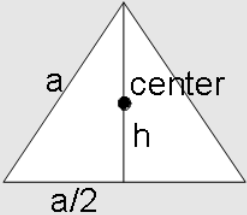


### Part 3: But how about Extensions? - Using abstract base classes

Imagine you need to draw more than Circle or Square objects. For instance Triangles or any other Shape you can think of. Every class than has to define a center point, scale method, calculation of their area and so on. This will generate much redundant code. In order to make our code more open for extensions (e.g. new shapes) without increasing the vulnerability in our code through changes, we will use a abstract base class. In our example this is the abstract Shape class. The relation between all our classes is shown in the UML-Diagram below.



In case you don't have the area calculation formulas ready, you can use those below:

Square	$A = a \cdot a$	
Circle	$A = \pi \cdot r^2$	
Triangle	$h = \frac{1}{2} \cdot \sqrt{3} \cdot a$ $A = \frac{1}{2} \cdot a \cdot h$	

In the following code snippet, objects of the types Circle, Triangle and Square with different dimensions and centers are created. The objects are stored in a list. Afterwards a for-loop is used to iterate through the list, print information about the particular objects and to invoke `calculate_area()` on all the objects. Although on all objects the same method are invoked, due to polymorphism the particular objects will behave differently depending on their implementations.

```
from shape import Shape
from circle import Circle
from triangle import Triangle
from square import Square
from point import Point

def create_shapes() -> list[Shape]:
    # create points
    p1 = Point(5.1, -8.6)
    p2 = Point(17.3, 9.5)
    p3 = Point(-4.2, 10.0)

    # using points to create one of each shapes
    circle = Circle(p1, 3)
    triangle = Triangle(p2, 5)
    square = Square(p3, 2)

    # create list of type shape
    shapesList: list[Shape] = [circle, triangle, square]
    return shapesList

def manage_shapes(shapes: list[Shape]) -> None:
    # prints information of all objects in list
    print("Print information about shapes:")
    for shape in shapes:
        print(f'{shape} has an area of {shape.calculate_area()}')

    # moves circle to (0|0) and sets radius to 2
    p4 = Point(0.0, 0.0)
    shapes[0].move_to(p4)
    shapes[0].scale(2.0)

    # prints all information again to check if previous worked
    print("Print information about shapes:")
    for shape in shapes:
        print(f'{shape} has an area of {shape.calculate_area()}')

if __name__ == "__main__":
    shapes = create_shapes()
    manage_shapes(shapes)
```

To be able to execute the above code you have to provide an implementation according to the UML diagram.