Programming Technology - Assignment 01 - Documentation

Description

Problem Statement:

We are tasked with creating a program that checks how many regular geometric shapes contain a given point on a 2D plane. The shapes are defined in a text file, and we will load them into a collection. Each shape is characterized by its type, center position, and size (either side length or radius). Our task is to determine how many of these shapes contain a specific point.

Shape Types:

We will work with four types of regular shapes:

1. **Circle (C)**: Defined by its center `(x, y)` and radius.
2. **Regular Triangle (T)**: An equilateral triangle with a specified center `(x, y)` and side length, with one side parallel to the x-axis.
3. **Square (S)**: Defined by its center `(x, y)` and side length, with one side parallel to the x-axis.
4. **Regular Hexagon (H)**: A regular hexagon with a specified center `(x, y)` and side length, where one side is parallel to the x-axis.

Input Data:

The shapes will be loaded from a text file. The first line of the file contains the number of shapes. Each subsequent line describes a shape with:

- **Shape type** (`C` for circle, `T` for triangle, `S` for square, `H` for hexagon),
- **Center coordinates** (`x, y`),
- **Size** (radius for circles, side length for polygons).

For example, a text file might look like this:

```
5
C,10.0,10.0,5.0
T,15.0,15.0,6.0
S,20.0,20.0,4.0
H,25.0,25.0,7.0
C,30.0,30.0,10.0
```

This represents:

- A **circle** with center `(10.0, 10.0)` and radius `5.0`.
- A **regular triangle** with center `(15.0, 15.0)` and side length `6.0`.
- A **square** with center `(20.0, 20.0)` and side length `4.0`.
- A **regular hexagon** with center `(25.0, 25.0)` and side length `7.0`.
- Another **circle** with center `(30.0, 30.0)` and radius `10.0`.

Steps:

1. **Input Parsing**:
    - We will load the shapes from the file and create instances of each shape.
    - We will store these shapes in a collection (ArrayList).

2. **Shape Representation**:
   - Each shape will be represented by a class, and all shapes must inherit from a common abstract parent `Shape`. The parent class will define common behavior for all shapes, such as the method to check if a point is inside the shape, and toString method.
   - The sub-classes (Circle, Triangle, Square, Hexagon) will implement the point-in-shape checking logic specific to each shape.
3. **Check if a Point is Inside**:
   - We will implement logic to check whether a given point `(px, py)` lies inside each shape. The method will be specific to the geometry of each shape:
     - **Circle - isInside()**
       - Use the distance formula to check if the distance between the point and the center is less than or equal to the length (radius).
     - **Triangle - isInside()**
       - First we find **R.** R is the distance between the center and the top vertex of the triangle.

$$R = \frac{L}{\sqrt{3}}$$

       - Where L is the length of the side of the triangle. Then it's vertices are

$$A, V_1 = (C_x - \frac{R\sqrt{3}}{2}, C_y - \frac{R}{2})$$

$$B, V_2 = (C_x + \frac{R\sqrt{3}}{2}, C_y - \frac{R}{2})$$

$$C, V_3 = (C_x, C_y + R)$$

       - Then we measure the area A of the triangle.
       - After, that we find PAB, PAC and PBC then we add them. There sum should be equal the area of the triangle ABC for the point to be inside or on the boundary of the triangle.
     - **Square - isInside()**
       - For a square, the x-coordinate of a vertex can be offset from the center by half the side length. Similar for the y-coordinate as well.
       - These are the formulated equations.

$$C_x + \frac{L}{2} >= P_x$$

$$C_x - \frac{L}{2} <= P_x$$

$$C_y + \frac{L}{2} >= P_y$$

$$C_y - \frac{L}{2} <= P_y$$

     - **Hexagon - isInside()**
       - Just as we did with the triangle, we are gonna find the vertices of the hexagon. As we know that it is a regular hexagon, all it's sides are equal, and the sum of it's angles is 720^o, making every angle 120^o. The vertices of a hexagon can be found using the following formulas:

$$V_n = (C_x + L(cos(\frac{n\pi}{3}), C_y + L(sin(\frac{n\pi}{3}))$$

       - Where n is a natural number from 1 to 6.

- After finding all the vertices we apply

> **RAY CASTING ALGORITHM**
> The number of intersections for a ray passing from the exterior of the polygon to any point: If odd, it shows that the point lies inside the polygon; if even, the point lies outside the polygon.

- **Edge of the Polygon**:
  Let's consider an edge of the polygon between two vertices:

$$V_1(x_1, y_1), V_2(x_2, y_2)$$

  This edge forms a line segment between these two points.

- **Condition for the Ray to Cross the Edge**:
  The ray cast from
  $P(x_p, y_p)$ can only cross this edge if the point P is vertically between V_1 and V_2. In other words, the Y-coordinate of P, denoted y_p, must be between the Y-coordinates of the vertices V_1 and V_2. If $(y_p > min(y1, y2))$ and $(y_p \leq max(y1, y2))$

- **Finding the Intersection Point**:
  If the Y-coordinate of
  P is within the vertical range of the edge, the next step is to check **where along the X-axis** the ray crosses the edge. This is done by calculating the X-coordinate of the intersection point.
  - To find this X-coordinate, we use the **equation of a line** connecting V_1 and V_2. The X-coordinate of the intersection point, denoted x_i, can be found by interpolating along the edge between the two vertices. The formula is:

$$xi = x_1 + \frac{(y_p - y_1)(x_2 - x_1)}{y_2 - y_1}$$

  This formula tells us how far along the edge the intersection occurs based on the relative position of y_p between y_1 and y_2.

- **Checking if the Intersection is to the Right of the Point**:
  Now that we have the X-coordinate of the intersection point, we check if it lies to the right of the point $P(x_p, y_p)$. For the ray to intersect the edge, this condition must hold: $x_p < x_i$
  This ensures that the intersection happens to the right of
  P, where the ray is cast.

Output:

- After determining how many shapes contain the point, the program will print the number of shapes that contain the given point.

Example Output:

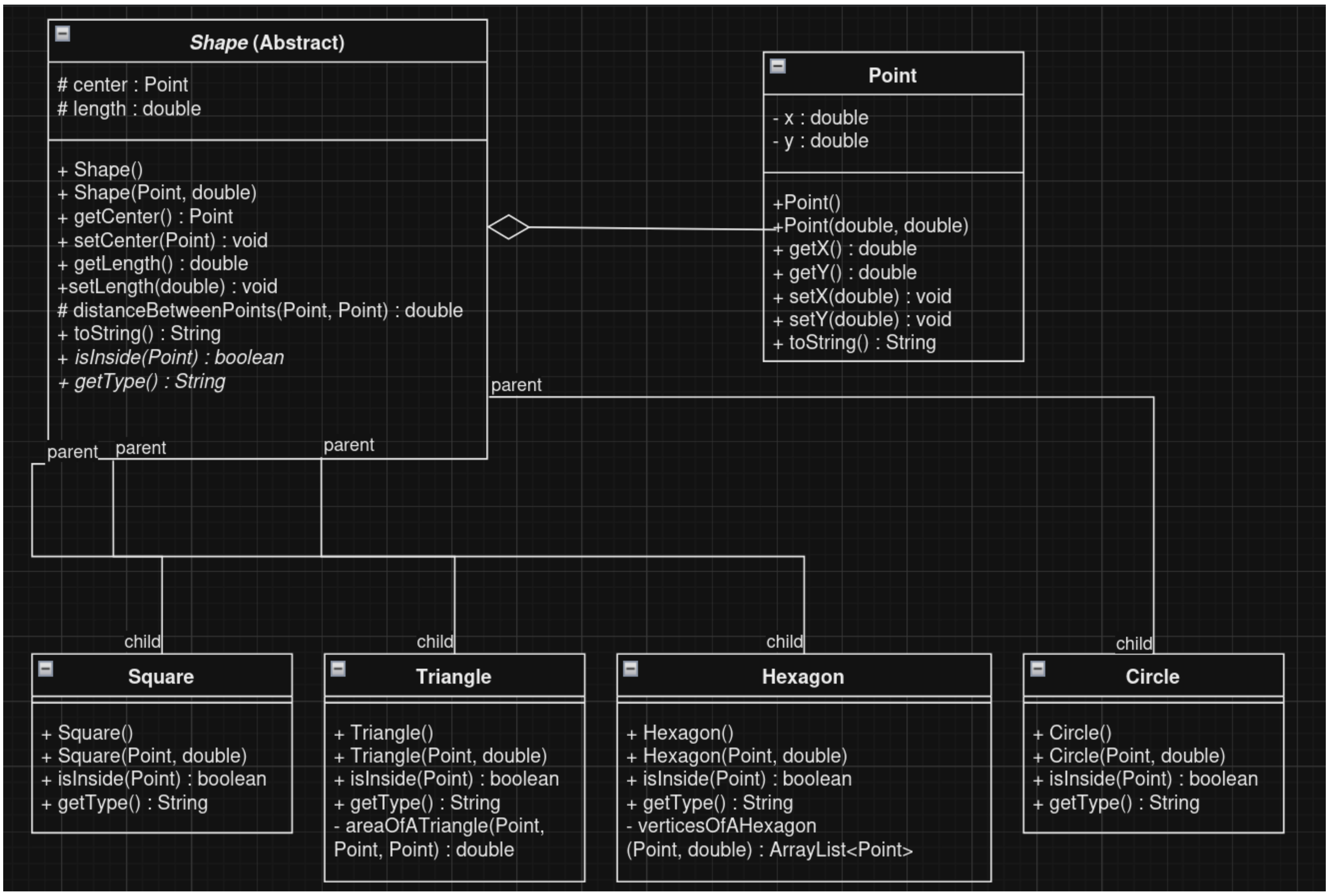
For the input file provided above, and if the given point is `(18.0, 18.0)`, the program would:

1. Check each shape in the collection.
2. Print how many of the shapes contain the point.

For example:

```
The point (18.0, 18.0) is contained within 1 shape.
```

UML


Shape (Abstract)
center : Point
length : double
+ Shape()
+ Shape(Point, double)
+ getCenter() : Point
+ setCenter(Point) : void
+ getLength() : double
+setLength(double) : void
distanceBetweenPoints(Point, Point) : double
+ toString() : String
+ isInside(Point) : boolean
+ getType() : String
Point
- x : double
- y : double
+Point()
+Point(double, double)
+ getX() : double
+ getY() : double
+ setX(double) : void
+ setY(double) : void
+ toString() : String
parent
parent
parent
parent
child
child
child
child
Square
+ Square()
+ Square(Point, double)
+ isInside(Point) : boolean
+ getType() : String
Triangle
+ Triangle()
+ Triangle(Point, double)
+ isInside(Point) : boolean
+ getType() : String
- areaOfATriangle(Point, Point, Point) : double
Hexagon
+ Hexagon()
+ Hexagon(Point, double)
+ isInside(Point) : boolean
+ getType() : String
- verticesOfAHexagon (Point, double) : ArrayList<Point>
Circle
+ Circle()
+ Circle(Point, double)
+ isInside(Point) : boolean
+ getType() : String