Министерство образования Республики Беларусь
Учреждения образования
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»
Институт информационных технологий

Специальность «Программное обеспечение информационных технологий»

КОНТРОЛЬНАЯ РАБОТА

По дисциплине «Объектно-ориентированные технологии программирования и стандарты проектирования»

Вариант №11

Студент 2 курса, ЗФО
Группы №381574
Жгуновский Олег Борисович

Минск, 2025

Задание: сектор окружности

Описание свойств фигуры:

Площадь сектора:

$$S = \frac{R^2\alpha}{2}$$

Длина дуги:

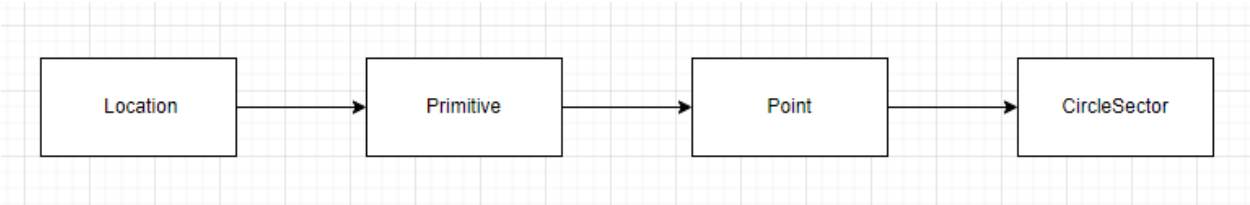
$$L = \alpha R$$

Периметр:

$$P = (\alpha + 2)R$$

Угол развёртки α, вычисляется с помощью параметров «начальный угол» и «конечный угол», с учётом направления сектора (против или по часовой стрелке)

Диаграмма иерархии классов:



Тестовый план:

Таблица 1 – Исходные данные и результаты

№	Исходные данные	Результаты
1	R = 10; Начальный угол: 0; Конечный угол: 90; Направление: CounterClockwise.	Периметр: 35,71; Площадь: 78,54; Угол развертки: 90,00°; Длина дуги: 15,71.
2	R = 25; Начальный угол: 0; Конечный угол: 90; Направление: CounterClockwise.	Периметр: 89,27; Площадь: 490,87; Угол развертки: 90,00°; Длина дуги: 39,27.

Таблица 1 – продолжение

№	Исходные данные	Результаты
3	R = 25; Начальный угол: 0; Конечный угол: 90; Направление: Clockwise.	Периметр: 167,81 Площадь: 1472,62 Угол развертки: 270,00° Длина дуги: 117,81

Листинг программы:

```
using System;

namespace GeometryFigures
{
    // Класс для описания положения
    public class Location
    {
        public double X { get; set; }
        public double Y { get; set; }

        public Location() : this(0, 0) { }

        public Location(double x, double y)
        {
            X = x;
            Y = y;
        }

        public Location(Location other) : this(other.X, other.Y) { }
    }

    // Класс для ограничивающей области
    public class Clip
    {
        public double MinX { get; set; }
        public double MaxX { get; set; }
        public double MinY { get; set; }
        public double MaxY { get; set; }

        public Clip() : this(0, 0, 0, 0) { }

        public Clip(double minX, double maxX, double minY, double maxY)
        {
            MinX = minX;
            MaxX = maxX;
            MinY = minY;
            MaxY = maxY;
```

```
    }

    public Clip(Clip other) : this(other.MinX, other.MaxX, other.MinY, other.MaxY) { }
}

// Статический класс с методами проверки
public static class Geometry
{
    public static Clip GlobalClip { get; set; } = new Clip(-100, 100, -100, 100);
    public const double Epsilon = 1e-5;

    public static bool IsWithinBounds(Location location)
    {
        return location.X >= GlobalClip.MinX - Epsilon &&
               location.X <= GlobalClip.MaxX + Epsilon &&
               location.Y >= GlobalClip.MinY - Epsilon &&
               location.Y <= GlobalClip.MaxY + Epsilon;
    }
}

// Класс с оформительскими свойствами
public class Primitive : Location
{
    public ConsoleColor Color { get; set; }
    public bool IsVisible { get; set; }

    public Primitive() : base()
    {
        Color = ConsoleColor.White;
        IsVisible = true;
    }

    public Primitive(double x, double y, ConsoleColor color, bool isVisible) : base(x, y)
    {
        Color = color;
        IsVisible = isVisible;
    }

    public Primitive(Primitive other) : base(other.X, other.Y)
    {
        Color = other.Color;
        IsVisible = other.IsVisible;
    }
}

// Класс точки
public class Point : Primitive
{
    public Point() : base() { }

    public Point(double x, double y, ConsoleColor color, bool isVisible)
```

```
        : base(x, y, color, isVisible) { }

    public Point(Point other) : base(other) { }
}

public enum RotationDirection { Clockwise, CounterClockwise }

public class CircleSector : Point
{
    public double Radius { get; private set; }
    public double StartAngle { get; private set; }
    public double EndAngle { get; private set; }
    public RotationDirection Direction { get; private set; }

    public CircleSector() : base()
    {
        Radius = 1;
        StartAngle = 0;
        EndAngle = 90;
        Direction = RotationDirection.CounterClockwise;
    }

    public CircleSector(double x, double y, ConsoleColor color, bool isVisible,
                        double radius, double startAngle, double endAngle,
                        RotationDirection direction) : base(x, y, color, isVisible)
    {
        Radius = radius;
        StartAngle = startAngle;
        EndAngle = endAngle;
        Direction = direction;
        ValidateBounds();
    }

    public CircleSector(CircleSector other) : base(other)
    {
        Radius = other.Radius;
        StartAngle = other.StartAngle;
        EndAngle = other.EndAngle;
        Direction = other.Direction;
    }

    // Периметр
    public double Perimeter => ArcLength + 2 * Radius;

    // Площадь сектора
    public double Area => (Math.PI * Radius * Radius) * (SweepAngle / 360);

    // Угол развертки в градусах
    public double SweepAngle => CalculateSweepAngle();

    // Длина дуги
```

```
public double ArcLength => (Math.PI * Radius * 2) * SweepAngle / 360;

public Clip GetBoundingClip()
{
    double minX = X;
    double maxX = X;
    double minY = Y;
    double maxY = Y;

    foreach (var point in GetCriticalPoints())
    {
        minX = Math.Min(minX, point.X);
        maxX = Math.Max(maxX, point.X);
        minY = Math.Min(minY, point.Y);
        maxY = Math.Max(maxY, point.Y);
    }

    return new Clip(minX, maxX, minY, maxY);
}

public void Resize(double radius, double startAngle, double endAngle)
{
    Radius = radius;
    StartAngle = startAngle;
    EndAngle = endAngle;
    ValidateBounds();
}

public void SetDirection(RotationDirection direction)
{
    Direction = direction;
    ValidateBounds();
}

private double CalculateSweepAngle()
{
    double angle = Direction == RotationDirection.CounterClockwise
        ? EndAngle - StartAngle
        : StartAngle - EndAngle;

    if (angle < 0) angle += 360;
    return angle;
}

private System.Collections.Generic.IEnumerable<Location> GetCriticalPoints()
{
    // Возвращает критические точки для вычисления ограничивающей области
    yield return new Location(X, Y);

    int steps = 10;
    double step = SweepAngle / steps;
```

```
        for (int i = 0; i <= steps; i++)
        {
            double angle = StartAngle + (Direction == RotationDirection.CounterClockwise
                ? step * i
                : -step * i);

            double rad = angle * Math.PI / 180;
            yield return new Location(
                X + Radius * Math.Cos(rad),
                Y + Radius * Math.Sin(rad)
            );
        }
    }

    private void ValidateBounds()
    {
        foreach (var point in GetCriticalPoints())
        {
            if (!Geometry.IsWithinBounds(point))
                throw new ArgumentException("Фигура выходит за границы!");
        }
    }
}

public static class FigureModifier
{
    public static void ModifyFigure(ref CircleSector sector)
    {
        while (true)
        {
            Console.WriteLine("\nТекущие параметры сектора:");
            Console.WriteLine($"1. X: {sector.X}");
            Console.WriteLine($"2. Y: {sector.Y}");
            Console.WriteLine($"3. Цвет: {sector.Color}");
            Console.WriteLine($"4. Видимость: {sector.IsVisible}");
            Console.WriteLine($"5. Радиус: {sector.Radius}");
            Console.WriteLine($"6. Начальный угол: {sector.StartAngle}°");
            Console.WriteLine($"7. Конечный угол: {sector.EndAngle}°");
            Console.WriteLine($"8. Направление: {sector.Direction}");
            Console.WriteLine($"9. Показать характеристики");
            Console.WriteLine("10. Выход");

            Console.Write("Выберите параметр для изменения: ");
            string choice = Console.ReadLine();

            try
            {
                switch (choice)
                {
                    case "1":
```

```
            Console.Write("Новое значение X: ");
            sector.X = double.Parse(Console.ReadLine());
            break;
        case "2":
            Console.Write("Новое значение Y: ");
            sector.Y = double.Parse(Console.ReadLine());
            break;
        case "3":
            Console.WriteLine("Доступные цвета:");
            foreach (var color in Enum.GetValues(typeof(ConsoleColor)))
                Console.WriteLine(color);
            Console.Write("Выберите цвет: ");
            sector.Color = (ConsoleColor)Enum.Parse(
                typeof(ConsoleColor), Console.ReadLine(), true);
            break;
        case "4":
            sector.IsVisible = !sector.IsVisible;
            break;
        case "5":
            Console.Write("Новый радиус: ");
            double r = double.Parse(Console.ReadLine());
            sector.Resize(r, sector.StartAngle, sector.EndAngle);
            break;
        case "6":
            Console.Write("Новый начальный угол: ");
            double sa = double.Parse(Console.ReadLine());
            sector.Resize(sector.Radius, sa, sector.EndAngle);
            break;
        case "7":
            Console.Write("Новый конечный угол: ");
            double ea = double.Parse(Console.ReadLine());
            sector.Resize(sector.Radius, sector.StartAngle, ea);
            break;
        case "8":
            Console.WriteLine("Доступные направления:");
            foreach (var dir in Enum.GetValues(typeof(RotationDirection)))
                Console.WriteLine(dir);
            Console.Write("Выберите направление: ");
            sector.SetDirection((RotationDirection)Enum.Parse(
                typeof(RotationDirection), Console.ReadLine(), true));
            break;
        case "9":
            Console.WriteLine($"Периметр: {sector.Perimeter:F2}");
            Console.WriteLine($"Площадь: {sector.Area:F2}");
            Console.WriteLine($"Угол развертки: {sector.SweepAngle:F2}°");
            Console.WriteLine($"Длина дуги: {sector.ArcLength:F2}");
            break;
        case "10":
            return;
        default:
            Console.WriteLine("Неверный выбор!");
```

```
                        break;
                    }
                }
                catch (Exception ex)
                {
                    Console.WriteLine($"Ошибка: {ex.Message}");
                }
            }
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            CircleSector sector = new CircleSector(0, 0, ConsoleColor.Green, true, 10, 0, 90,
RotationDirection.CounterClockwise);
            FigureModifier.ModifyFigure(ref sector);
        }
    }
}
```

Пример использования:

```
Текущие параметры сектора:
1. X: 0
2. Y: 0
3. Цвет: Green
4. Видимость: True
5. Радиус: 10
6. Начальный угол: 0°
7. Конечный угол: 90°
8. Направление: CounterClockwise
9. Показать характеристики
10. Выход
Выберите параметр для изменения: 9
Периметр: 35,71
Площадь: 78,54
Угол развертки: 90,00°
Длина дуги: 15,71

Текущие параметры сектора:
1. X: 0
2. Y: 0
3. Цвет: Green
4. Видимость: True
5. Радиус: 10
6. Начальный угол: 0°
7. Конечный угол: 90°
8. Направление: CounterClockwise
9. Показать характеристики
```

```
10. Выход
Выберите параметр для изменения: 8
Доступные направления:
Clockwise
CounterClockwise
Выберите направление: ClockWise

Текущие параметры сектора:
1. X: 0
2. Y: 0
3. Цвет: Green
4. Видимость: True
5. Радиус: 10
6. Начальный угол: 0°
7. Конечный угол: 90°
8. Направление: Clockwise
9. Показать характеристики
10. Выход
Выберите параметр для изменения: 5
Новый радиус: 12

Текущие параметры сектора:
1. X: 0
2. Y: 0
3. Цвет: Green
4. Видимость: True
5. Радиус: 12
6. Начальный угол: 0°
7. Конечный угол: 90°
8. Направление: Clockwise
9. Показать характеристики
10. Выход
Выберите параметр для изменения: 6
Новый начальный угол: 1

Текущие параметры сектора:
1. X: 0
2. Y: 0
3. Цвет: Green
4. Видимость: True
5. Радиус: 12
6. Начальный угол: 1°
7. Конечный угол: 90°
8. Направление: Clockwise
9. Показать характеристики
10. Выход
Выберите параметр для изменения: 3
Доступные цвета:
Black
DarkBlue
DarkGreen
```

```
DarkCyan
DarkRed
DarkMagenta
DarkYellow
Gray
DarkGray
Blue
Green
Cyan
Red
Magenta
Yellow
White
Выберите цвет: Blue

Текущие параметры сектора:
1. X: 0
2. Y: 0
3. Цвет: Blue
4. Видимость: True
5. Радиус: 12
6. Начальный угол: 1°
7. Конечный угол: 90°
8. Направление: Clockwise
9. Показать характеристики
10. Выход
Выберите параметр для изменения: 9
Периметр: 80,76
Площадь: 340,55
Угол развертки: 271,00°
Длина дуги: 56,76

Текущие параметры сектора:
1. X: 0
2. Y: 0
3. Цвет: Blue
4. Видимость: True
5. Радиус: 12
6. Начальный угол: 1°
7. Конечный угол: 90°
8. Направление: Clockwise
9. Показать характеристики
10. Выход
Выберите параметр для изменения: 10
```