В.В. Подбельский

Использованы иллюстрации пособия Daniel Solis, Illustrated C#

Иллюстрации к курсу лекций по дисциплине «Программирование на C#»

Модуль 3. Лекция 3а

Отношения Между Типами

Благодарность Д. Д. Сагалову за помощь в переработке презентаций

Возможные Отношения Между Типами

1. Типы независимы;
2. Наследование – «является» (*is-a*).
3. Включение – «имеет» (*has-a*) / или «включается» (*is-part-of*);
4. Вложение;

Формы отношения включения:

– композиция;

– агрегация.

Тип «Точка на Плоскости»

1) Класс + Свойство

```
public class Point
{
    private double _x, _y;
    public double X { get { return _x; } init { _x = value; } }
    public double Y { get { return _y; } init { _y = value; } }
}
```

2) Класс + Авто. Свойство

```
public class Point
{
    public double X { get; init; }
    public double Y { get; init; }
}
```

3) Запись

```
public record class Point(double X, double Y);
```

Композиция Классов

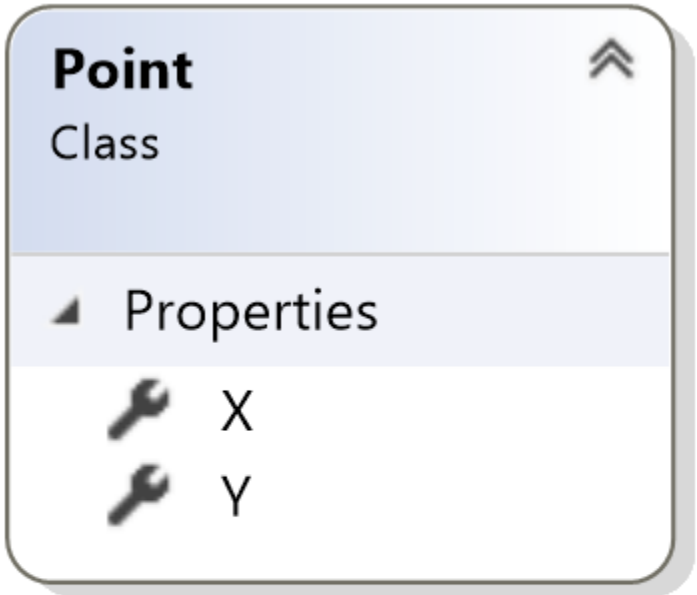
```
public class Point
{
    public double X { get; set; }
    public double Y { get; set; }
}

public class Circle
{
    public double Radius { get; set; }
    public double Length => 2 * Radius * Math.PI;
    public Point Center { get; set; } = new Point();

    public void Display()
       => Console.WriteLine($"Center: X = {Center.X}, Y = {Center.Y}; " +
                            $"Radius = {Radius}, Length = {Length, 6:f2}");
}
```

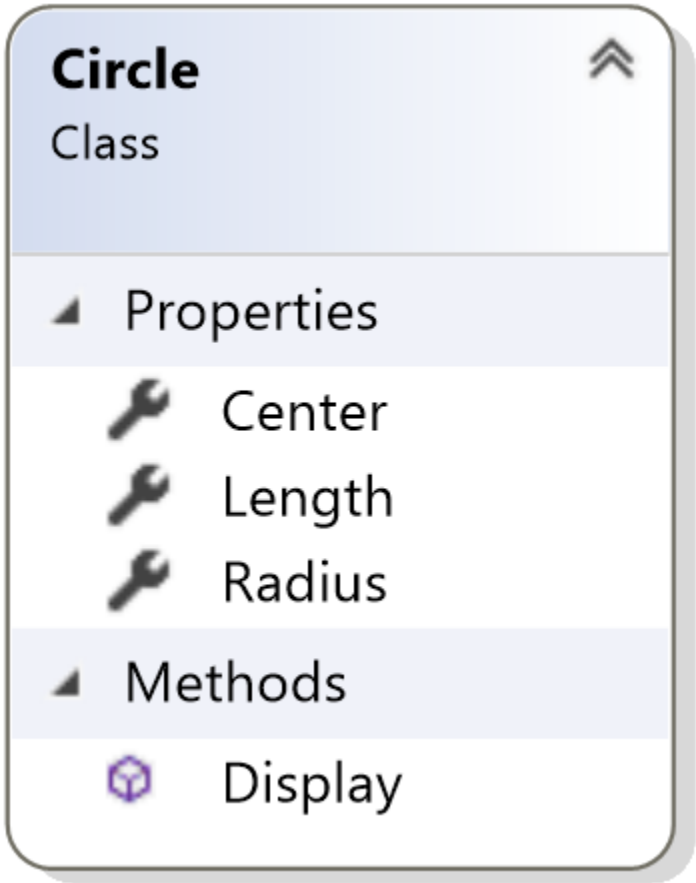
Экземпляр Класса Circle при Композиции

```
Circle circle = new();
circle.Center.X = 10;
circle.Center.Y = 20;
circle.Radius = 3.0;
circle.Display();
```

Вывод:
Center: X = 10, Y = 20; Radius = 3, Length = 18.85

Диаграмма Композиции Классов





Агрегация Классов

```
public class Circle
{
    public double Radius { get; set; }
    public double Length => 2 * Radius * Math.PI;

    private Point _center;
    public Circle(Point point, double radius)
        => (_center, Radius) = (point, radius);

    public void Display()
        => Console.WriteLine($"Center: X = {_center.X}, Y = {_center.Y}; " +
                             $"Radius = {Radius}, Length = {Length,6:f2}");

}
```

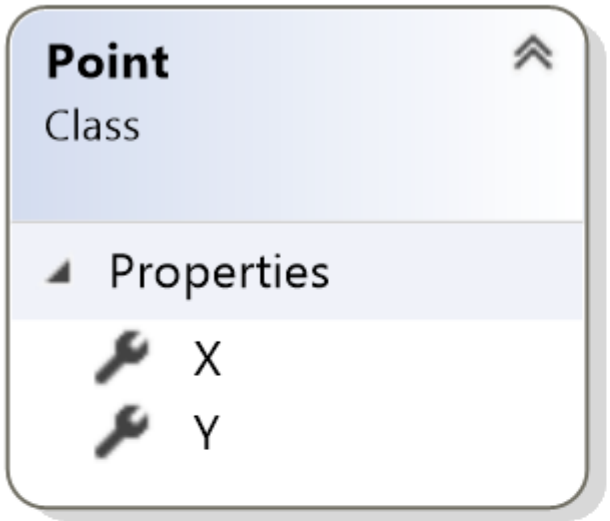
Теперь точка передаётся в конструктор, её жизненный цикл не контролируется объектом Circle.

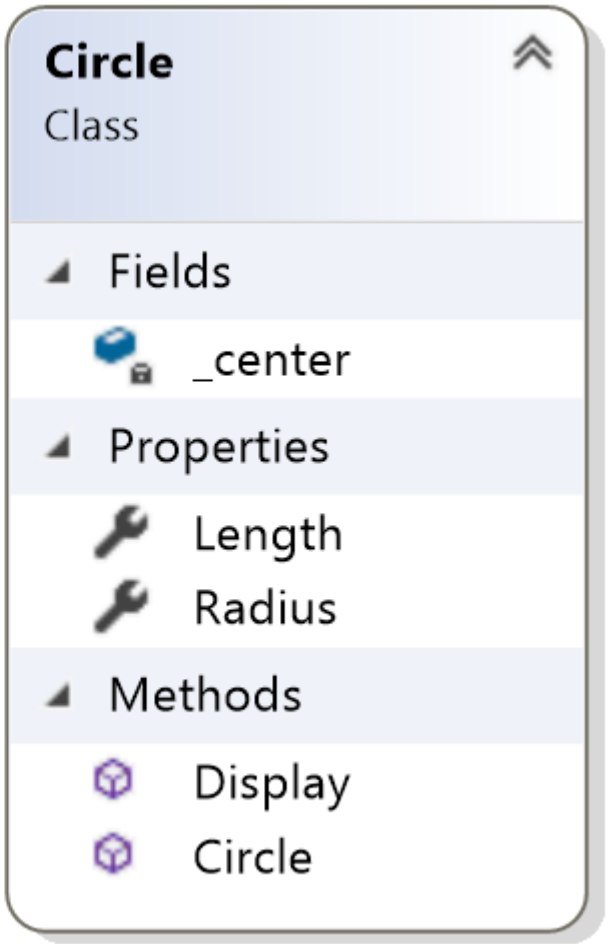
Экземпляр Класса Circle при Агрегации

```
Point pt = new Point();
pt.X = 10;
pt.Y = 20;
Circle circle = new Circle(pt, 10);
circle.Display();
```

Вывод:
Center: X = 10, Y = 20; Radius = 10, Length = 62.83

Диаграмма Агрегации Классов





Вложенный Класс

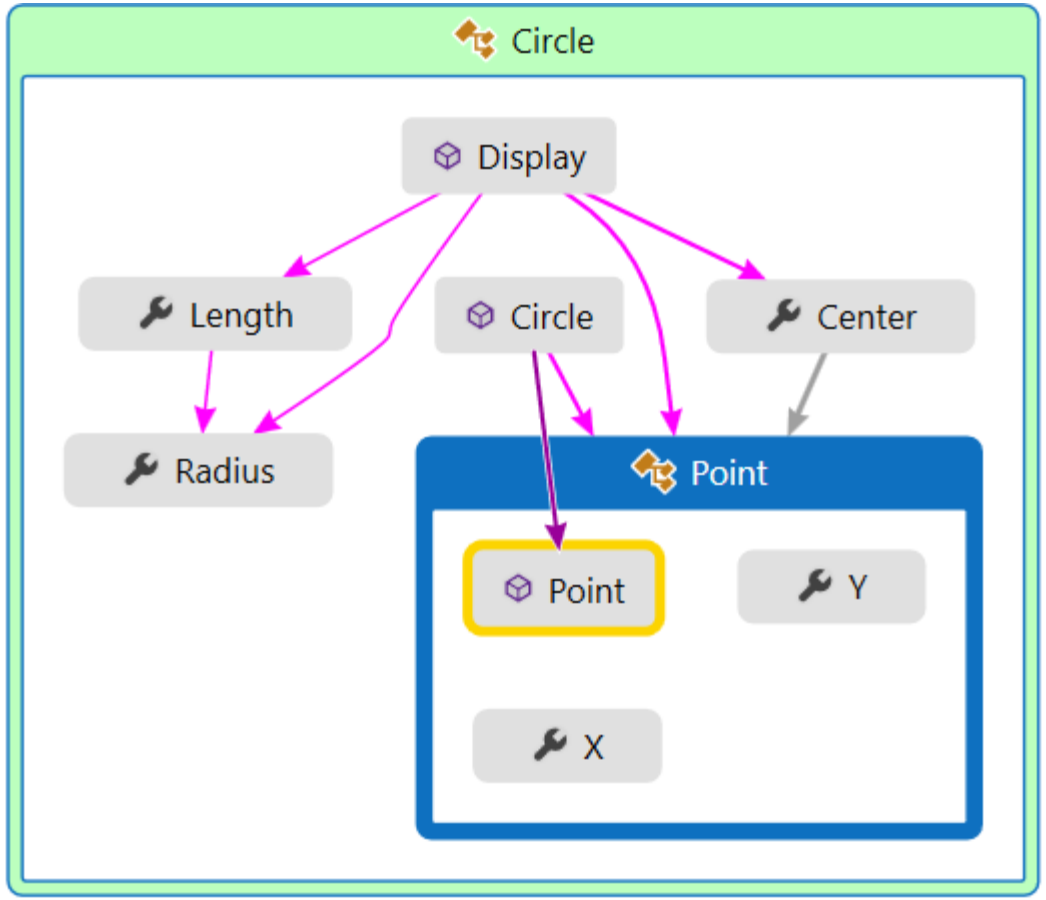
```
public class Circle
{
    public Point Center { get; set; } = new Point(); // центр окружности

    public double Radius { get; set; }
    public double Length => 2 * Radius * Math.PI;

    public void Display()
        => Console.WriteLine($"Center: X = {Center.X}, Y = {Center.Y}; " +
                             $"Radius = {Radius}, Length = {Length,6:f2}");
    public class Point
    {
        public double X { get; set; }
        public double Y { get; set; }
    }
}
```

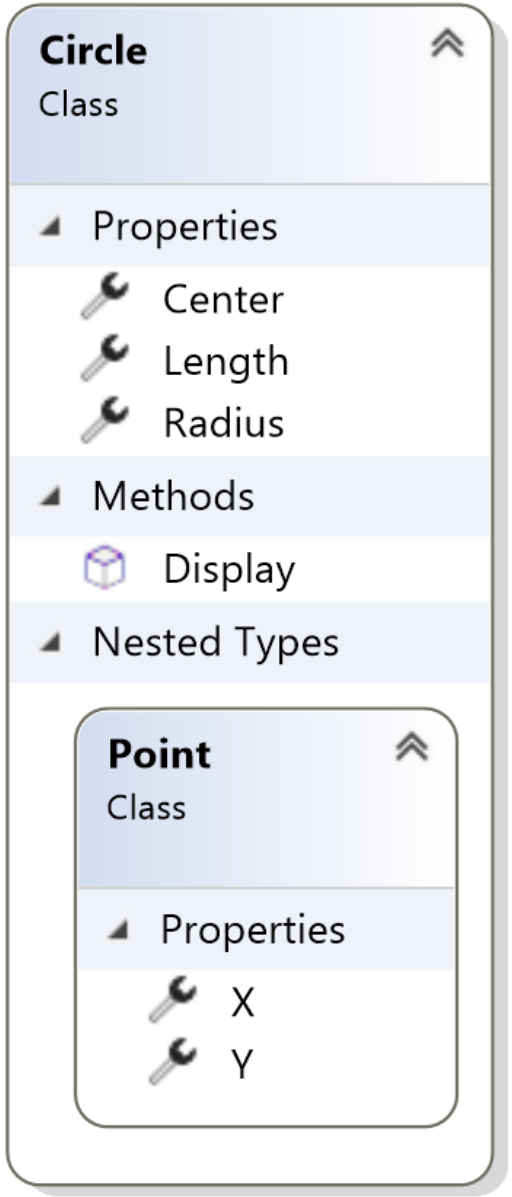
Экземпляр Класса Circle при Вложении Класса Point

```
Circle circle = new Circle();
circle.Center.X = 100;
circle.Center.Y = 200;
circle.Radius = 30.0;
circle.Display();
```

Вывод:
Center: X = 100, Y = 200; Radius = 30, Length = 188.50

Диаграмма Вложения Классов





ПКМ (class Circle):
Code Map -> Show On Code Map

Наследование

Синтаксис:

```
атрибуты opt
Модификаторы opt
class идентификатор_производного_класса
: идентификатор_базового_класса
{
    тело_производного_класса…
}
```

Наследование от Класса Point

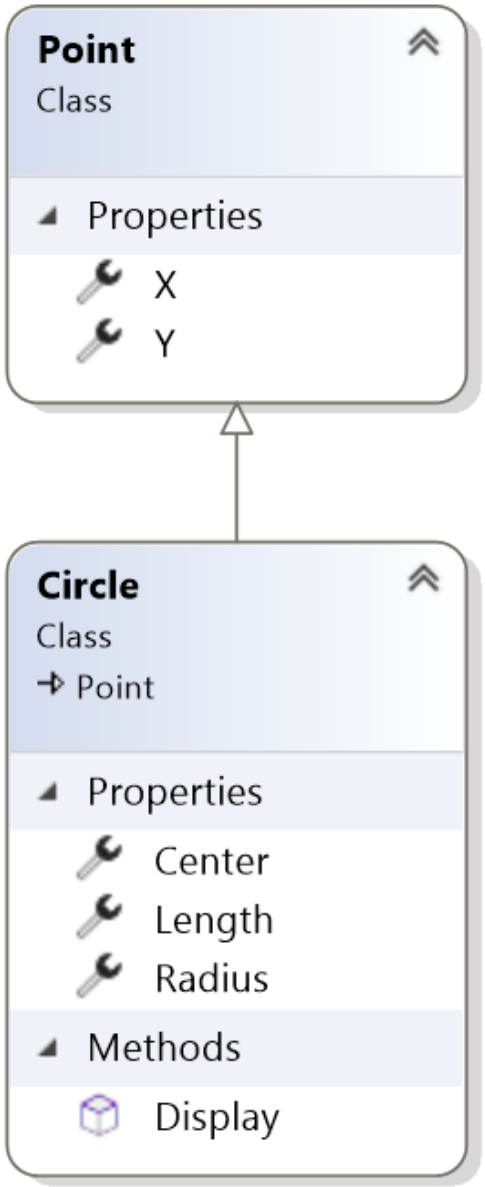
```
public class Circle : Point
{
    public double Radius { get; set; }
    public double Length => 2 * Radius * Math.PI;
    public Point Center
    {
        get => new Point() { X = this.X, Y = this.Y };
        set { X = value.X; Y = value.Y; }
    }
    public void Display()
        => Console.WriteLine($"Center: X = {X}, Y = {Y}; " +
                             $"Radius = {Radius}, Length = {Length,6:f2}");
}
```

Экземпляр Класса Circle – Наследника Point

```
Circle circle = new Circle();
circle.X = 24;              // Свойство из Point.
circle.Y = 10;              // Свойство из Point.
circle.Radius = 2;          // Свойство из Circle.
circle.Display();
circle = new Circle();
circle.Display();
```

Вывод:
Center: X = 24, Y = 10; Radius = 2, Length = 12.57
Center: X = 0, Y = 0; Radius = 0, Length = 0.00

Диаграмма Наследования Классов



Наследование от Класса с protected-Членами. Часть 1

```
// Базовый класс круга.
public class Disk
{
    protected double radius;
    protected Disk(double radius) => this.radius = radius;
    protected double Area => radius * radius * Math.PI;
}
```

Наследование от Класса с protected-Членами. Часть 2

```
// Класс кольцо.
public class Ring : Disk
{
    // Радиус внутренней окружности:
    new double radius;
    public Ring(double outerRadius, double innerRadius)
                : base(outerRadius) { radius = innerRadius; }
    public new double Area => base.Area - Math.PI * radius * radius;
    public void Print()
        => Console.WriteLine($"Ring: Outer radius = {base.radius:f2}, " +
                             $"Inner radius = {radius:f2}, Area = {Area:f3}");
}
```

Экземпляр Класса Ring – Наследника Disk

```
Ring ring = new Ring(10.0, 4.0);
ring.Print();
```

Вывод:
Ring: Outer radius = 10.00, Inner radius = 4.00, Area = 263.894

Экранирование Методов Базового Класса. Часть 1

```
using System;

public class Figure
{
    // Размеры вдоль осей:
    protected double dx, dy;
    public void Print()
        => Console.WriteLine($"dx = {dx:f2}, dy = {dy:f2}");
}
```

Экранирование Методов Базового Класса. Часть 2

```
public class Rectangle : Figure {
    public Rectangle(double xi, double yi) => (dx, dy) = (xi, yi);
    public new void Print() {
        Console.Write("Rectangle: ");
        base.Print();
    }
}

public class Square : Rectangle {
    public Square(double side) : base(side, side) { }
    public new void Print() {
        Console.Write("Square: ");
        base.Print();
    }
}
```

Вывод Результатов Экранирования в Наследниках-1

```
Figure figure = new Figure();
figure.Print();
Rectangle rectangle = new Rectangle(3.0, 4.0);
rectangle.Print();
Square square = new Square(5.0);
square.Print();
```

Вывод:
dx = 0.00, dy = 0.00
Rectangle: dx = 3.00, dy = 4.00
Square: Rectangle: dx = 5.00, dy = 5.00

Вывод Результатов Экранирования в Наследниках-2

```
Figure fig1 = new Figure();
Figure fig2 = new Rectangle(3.0, 4.0);
Figure fig3 = new Square(5.0);
fig1.Print();
fig2.Print();
fig3.Print();
```

Вывод:
dx = 0.00, dy = 0.00
dx = 3.00, dy = 4.00
dx = 5.00, dy = 5.00

Вывод Результатов при Виртуальном Print

```
public virtual void Print() // В Figure.

public override void Print() // В наследниках Figure.

Figure fig1 = new Figure();
Figure fig2 = new Rectangle(3.0, 4.0);
Figure fig3 = new Square(5.0);
fig1.Print();
fig2.Print();
fig3.Print();
```

Вывод:
dx = 0.00, dy = 0.00
Rectangle: dx = 3.00, dy = 4.00
Square: Rectangle: dx = 5.00, dy = 5.00

Абстрактный Класс и Абстрактные Методы

```
public abstract class Figure
{
    // Размеры вдоль осей:
    protected double dx, dy;
    public abstract double Area { get; }
    public abstract void Print();
}
```

Наследники Абстрактного Класса Figure

```
class Rectangle : Figure
{
    public Rectangle(double xi, double yi) => (dx, dy) = (xi, yi);

    public override double Area => dx * dy;
    public override void Print()
        => Console.WriteLine($"Rectangle with area = {Area:f2}\n" +
                             $"Dimensions: dx = {dx:f2}, dy = {dy:f2}");
}

class Square : Rectangle
{
    public Square(double side) : base(side, side) { }
    public override void Print()
        => Console.WriteLine($"Square with area = {Area:f2}\n" +
                             $"Side = {dx:f2}");
}
```

Вывод для Наследников Абстрактного Класса Figure

```
Figure figure = new Rectangle(3.0, 4.0);
figure.Print();
figure = new Square(5.0);
figure.Print();
Square square = new Square(8.0);
square.Print();
```

Вывод:

Rectangle with area = 12.00
Dimensions: dx = 3.00, dy = 4.00
Square with area = 25.00
Side = 5.00
Square with area = 64.00
Side = 8.00