# Лабораторная работа №7

## Иерархии классов

### Вариант 1

Цель работы: приобретение практических навыков построения иерархий классов на языке C#.

### Задание к работе

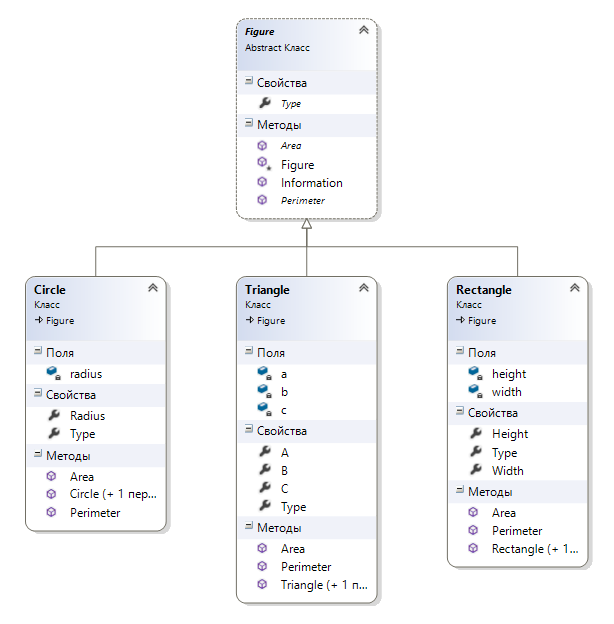
1. Изучить главы 8, 9, 10 учебника «Visual C# 2010: полный курс».
2. По заданному описанию класса в соответствии с вариантом создать иерархию классов на языке C#.
3. Создать программу для решения поставленной задачи, использующую созданную структуру классов.

### Задание к первому варианту

1. Создать абстрактный класс Figure с методами вычисления площади и периметра, а также методом, выводящим информацию о фигуре на экран.
2. Создать производные классы: Rectangle (прямоугольник), Circle (круг), Triangle (треугольник) со своими методами вычисления площади и периметра.
3. Создать массив фигур и вывести полную информацию о фигурах на экран.

## Решение

### Блок-схема иерархии классов



### Результат работы программы

|  |
| --- |
| Сообщение на экране |
| Rectangle (1x2):  Area: 2.00  Perimeter: 6.00  Circle (R = 3):  Area: 28.27  Perimeter: 18.85  Triangle:  Area: 9.92  Perimeter: 15.00 |

### Текст программы

|  |  |
| --- | --- |
| 001 | **using** System; |
| 002 |  |
| 003 | **namespace** OOP\_07 |
| 004 | { |
| 005 | **class** Program |
| 006 | { |
| 007 | **static** **void** Main(**string**[] args) |
| 008 | { |
| 009 | Figure[] n = { **new** Rectangle(1, 2), **new** Circle(3),  **new** Triangle(4, 5, 6) }; |
| 010 |  |
| 011 | **foreach** (Figure figure **in** n) |
| 012 | { |
| 013 | figure.Information(); |
| 014 | Console.WriteLine(); |
| 015 | } |
| 016 | } |
| 017 | } |
| 018 |  |
| 019 | **abstract** **class** Figure |
| 020 | { |
| 021 | **protected** Figure() { } |
| 022 |  |
| 023 | **public** **abstract** **double** Area(); |
| 024 | **public** **abstract** **double** Perimeter(); |
| 025 | **public** **abstract** **string** Type { **get**; } |
| 026 |  |
| 027 | **public** **void** Information() |
| 028 | { |
| 029 | Console.WriteLine(*"{0}:"*, Type); |
| 030 | Console.WriteLine(*"\tArea: {0:N2}"*, Area()); |
| 031 | Console.WriteLine(*"\tPerimeter: {0:N2}"*, Perimeter()); |
| 032 | } |
| 033 | } |
| 034 |  |
| 035 | **class** Rectangle : Figure |
| 036 | { |
| 037 | **private** **double** width; |
| 038 | **private** **double** height; |
| 039 |  |
| 040 | **public** Rectangle() |
| 041 | { |
| 042 | width = height = 0; |
| 043 | } |
| 044 |  |
| 045 | **public** Rectangle(**double** width, **double** height) |
| 046 | { |
| 047 | **this**.width = width; |
| 048 | **this**.height = height; |
| 049 | } |
| 050 |  |
| 051 | **public** **double** Width |
| 052 | { |
| 053 | **get** { **return** width; } |
| 054 | **set** { width = value; } |
| 055 | } |
| 056 |  |
| 057 | **public** **double** Height |
| 058 | { |
| 059 | **get** { **return** height; } |
| 060 | **set** { height = value; } |
| 061 | } |
| 062 |  |
| 063 | **public** **override** **string** Type |
| 064 | { |
| 065 | **get** { **return** *"Rectangle ("* + width.ToString() + *"x"* +  height.ToString() + *")"*; } |
| 066 | } |
| 067 |  |
| 068 | **public** **override** **double** Area() |
| 069 | { |
| 070 | **return** width \* height; |
| 071 | } |
| 072 |  |
| 073 | **public** **override** **double** Perimeter() |
| 074 | { |
| 075 | **return** 2 \* (width + height); |
| 076 | } |
| 077 | } |
| 078 |  |
| 079 | **class** Circle : Figure |
| 080 | { |
| 081 | **private** **double** radius; |
| 082 |  |
| 083 | **public** Circle() |
| 084 | { |
| 085 | radius = 0; |
| 086 | } |
| 087 |  |
| 088 | **public** Circle(**double** radius) |
| 089 | { |
| 090 | **this**.radius = radius; |
| 091 | } |
| 092 |  |
| 093 | **public** **double** Radius |
| 094 | { |
| 095 | **get** { **return** radius; } |
| 096 | **set** { radius = value; } |
| 097 | } |
| 098 |  |
| 099 | **public** **override** **string** Type |
| 100 | { |
| 101 | **get** { **return** *"Circle (R = "* + radius.ToString() + *")"*; } |
| 102 | } |
| 103 |  |
| 104 | **public** **override** **double** Area() |
| 105 | { |
| 106 | **return** Math.PI \* radius \* radius; |
| 107 | } |
| 108 |  |
| 109 | **public** **override** **double** Perimeter() |
| 110 | { |
| 111 | **return** 2 \* Math.PI \* radius; |
| 112 | } |
| 113 | } |
| 114 |  |
| 115 | **class** Triangle : Figure |
| 116 | { |
| 117 | **private** **double** a; |
| 118 | **private** **double** b; |
| 119 | **private** **double** c; |
| 120 |  |
| 121 | **public** Triangle() |
| 122 | { |
| 123 | a = b = c = 0; |
| 124 | } |
| 125 |  |
| 126 | **public** Triangle(**double** a, **double** b, **double** c) |
| 127 | { |
| 128 | **this**.a = a; |
| 129 | **this**.b = b; |
| 130 | **this**.c = c; |
| 131 | } |
| 132 |  |
| 133 | **public** **double** A |
| 134 | { |
| 135 | **get** { **return** a; } |
| 136 | **set** { a = value; } |
| 137 | } |
| 138 |  |
| 139 | **public** **double** B |
| 140 | { |
| 141 | **get** { **return** b; } |
| 142 | **set** { b = value; } |
| 143 | } |
| 144 |  |
| 145 | **public** **double** C |
| 146 | { |
| 147 | **get** { **return** c; } |
| 148 | **set** { c = value; } |
| 149 | } |
| 150 |  |
| 151 | **public** **override** **string** Type |
| 152 | { |
| 153 | **get** { **return** *"Triangle"*; } |
| 154 | } |
| 155 |  |
| 156 | **public** **override** **double** Area() |
| 157 | { |
| 158 | **double** p = Perimeter() / 2; |
| 159 |  |
| 160 | **return** Math.Sqrt(p \* (p - a) \* (p - b) \* (p - c)); |
| 161 | } |
| 162 |  |
| 163 | **public** **override** **double** Perimeter() |
| 164 | { |
| 165 | **return** a + b + c; |
| 166 | } |
| 167 | } |
| 168 | } |