# Лекция 5. Классы (основы). Оконный интерфейс

**Что делать?**

1. Повторить материал об объявлении функций из лекции 3, пример 4.
2. Внимательно изучить теорию, понять и запомнить термины «класс», «объект», «атрибут», «поле», «метод», «ссылочный тип данных». Спросить преподавателя обо всем, что непонятно – это фундаментальная тема, т.к. язык C# – объектно-ориентированный.
3. Реализовать примеры классов из данного файла. Выполнять коммит тогда, когда это указано в задании, или чаще. Ответить на вопросы в тексте.
4. Выполнить задания на самостоятельную работу.
5. Сдать все через GitHub.

## Теоретическая часть. Основные понятия

C# – это объектно-ориентированный язык программирования. Понятия класса и объекта являются для него базовыми, без их понимания невозможно вести никакую разработку.

**Класс** – группа реальных или виртуальных объектов, обладающих общими свойствами (атрибутами) и методами поведения.

**Объект (экземпляр класса, инстанция)** – конкретный представитель класса с определенными значениями свойств.

Примеры:

|  |  |
| --- | --- |
| *Класс* | *Экземпляр* |
| Кошка | конкретная Мурка. |
| Студент | конкретный студент Н. Кузнецов. |
| Чек | чек №0001003234023 от 02.11.2012. |
| Командная строка | конкретное окно командной строки, которое появилось при запуске программы |

В зависимости от поставленной задачи, граница между экземпляром и классом может смещаться. Например, класс – автомобиль. Если задача – просто описать различные марки автомобилей, то экземпляром может быть «LADA Granta» или «Ford Focus» (не какой-то один реальный автомобиль, а в целом все автомобили этой марки). Но если задача – составление объявлений о продаже подержанных авто, то в этом случае экземпляром будет вполне конкретный автомобиль с конкретным владельцем, а его марка – всего лишь атрибут.

**Атрибут (свойство)** – это некоторая характеристика, присущая всем объектам данного класса. Ее значения могут отличаться, но у всех она присутствует.

Можно сказать, что объект отличается от класса тем, что у объекта атрибуты имеют конкретные значения, а у класса – просто описываются.

Примеры:

|  |  |  |
| --- | --- | --- |
| *Класс* | *Атрибуты* | *Экземпляр* |
| Кошка | Имя  Пол  Возраст  Окрас  Тип шерсти | Мурка  жен.  3  Черный с белыми «носочками»  Короткая |
| Студент | ФИО  Дата рождения  № студ. билета  Группа  Курс | Н.А. Кузнецов  15.06.1998  0100187  ПИ  1 |
| Чек | № чека  Дата и время  Кассир  Перечень товаров  Сумма | 0001003234023  02.11.2012 15:04  Добрынина  *(сами товары – тоже объекты)*  312,15 |
| Командная строка | Координата X  Координата Y  Ширина  Высота  Цвет текста  Выведенный текст | 100  315  640  480  White  “Введите число:” |

**Метод (операция)** – определенный способ поведения (действие), присущий всем экземплярам класса. Действие может быть как активным (сам объект что-то делает), так и пассивным (с объектом что-то делают).

Метод задается для всего класса и действует для всех его объектов.

|  |  |
| --- | --- |
| *Класс* | *Методы* |
| Кошка | Мяукать  Мурлыкать  Спать  Играть  Есть |
| Студент | Слушать лекцию  Выполнять задание  Идти в аудиторию  Прогуливать |
| Чек | Напечатать  Вычислить сумму |
| Командная строка | Вывести текст  Ввести текст  Очистить |

Перечень методов, опять же, зависит от задачи, ради которой мы описываем класс. В примере выше для класса «Студент» мы описали только методы поведения, связанные с учебой, хотя студент может выполнять еще массу других действий. И мы не можем добавить, например, метод «Играть на саксофоне», т.к. не все студенты это умеют.

Это были определения и примеры в общем смысле.

С точки зрения программирования, **класс – это структура данных**, т.е. описание того, какие данные (типы) и в каком порядке нужно хранить в памяти компьютера (атрибуты), и какие программы можно с этими данными выполнять (методы). А **объект – это уже записанные в памяти данные** с заполненными значениями атрибутов.

Класс объявляется ключевым словом class, затем идет уникальное **имя класса** (по правилам для идентификаторов, общепринято писать с заглавной буквы), затем в фигурных скобках {} – **описание класса**:

class Cat //класс Кошка

{

//здесь будет описание класса

}

Описание класса – перечень всех его атрибутов и методов.

**Атрибуты** в программировании могут быть двух основных типов: поля и свойства. Со свойствами мы разберемся немного позже, а **поля** являются аналогами переменных, объявленных внутри класса. Они также имеют тип и имя, и могут иметь начальное значение.

class Cat //класс Кошка

{

string name = ""; //имя

byte age = 0; //возраст

boolean female = false; // пол

double weight = 1; //вес

}

**Методы** являются подпрограммами (функциями) в теле класса. Мы уже создавали собственную функцию при разработке консольного калькулятора. Та функция размещалась в теле класса Program и, следовательно, была частью всей нашей программы.

Объявление методов внутри другого класса принципиально не отличается.

class Cat //класс Кошка

{

string name = ""; //имя

byte age = 0; //возраст

boolean female = false; // пол

double weight = 1; //вес

// сказать текст speech

void Say(string speech)

{

//speech выводится в консоль

Console.WriteLine(speech);

}

// любит ли еду food?

boolean Likes(string food)

{

//любит мясо, рыбу и молоко

if (food == "мясо" || food == "рыба" || food == "молоко")

return true;

//все остальное – не любит

return false;

}

// съесть еду food весом weight

void Eat(string food, double foodWeight)

{

//если любит, то съест

if (Likes(food))

{

//и прибавит в весе

weight += foodWeight;

}

}

}

В то же время, класс можно рассматривать как собственный (созданный программистом) тип данных.

Cat murka = new Cat();

В данном случае murka – это экземпляр класса Cat, т.е. объект. Мы можем обращаться к полям и методам класса Cat с помощью знакомого нам оператора «точка»:

murka.name = "Мурка";

murka.age = 3;

murka.female = true;

murka.weight = 4.1;

murka.Eat("рыба",0.2);

Таким образом, мы уже использовали некоторые классы: Console, Math, Program.

Фактически, все, что есть в C#, является классами и объектами. Даже простейшие типы, такие как int, double, string имеют «класс-близнец», реализованный в .NET, соответственно Int, Double, String (с заглавной буквы). Именно этот факт позволяет нам использовать конструкции вида:

int.TryParse(str, out x)

str.Length

Т.е. на самом деле мы обращаемся к методам соответствующих классов.

Но между простейшим типом данных и классом есть существенная разница: класс является **ссылочным типом данных**, а int, double, boolean и т.д. – это **хранимые типы данных**. Это значит, что, если мы объявляем переменную

int x = 5;

то в переменной x сохраняется само значение 5 «как есть», в двоичной форме.

Но, когда мы объявляем

Cat murka = new Cat();

то в самой переменной murka нет объекта. Там размещается адрес в памяти (ссылка), где фактически находится этот объект.

Если мы объявим еще одну переменную и скопируем туда значение:

Cat murzik = murka;

то в переменной murzik будет копия **ссылки** на ту же самую кошку murka, а вовсе не новая кошка (правильнее было бы назвать переменную murka2, а не murzik).

Известные нам типы данных (а мы рассмотрели далеко не все типы):

Чтобы лучше понять, как это все работает, рассмотрим аналогию.

Представьте себе, что (оперативная) **память** компьютера – это такой склад, в котором все хранится на абсолютно одинаковых полках в одинаковых стандартных ящиках (**ячейки памяти**, размер 1 байт). Все полки пронумерованы подряд (**адрес** ячейки памяти). Причем все полки выглядят абсолютно одинаково, с одинаковыми ящиками, независимо от того, что в этих ящиках лежит, или они вообще пустые (все **в двоичном коде**).

Как найти какой-нибудь объект на этом огромном складе с миллионами или даже миллиардами полок? Надо просто знать номер полки, на которой он лежит. **Ссылка на объект (указатель, reference)** – это как раз запись с номером полки (адресом памяти). По сути, указатель является просто целым числом, как и тип int, и сам по себе ничего не говорит о том, что лежит на полке. Так что его одного может быть недостаточно для работы, нужно знать, что находится по указанному адресу. За это отвечает тип переменной, в которой мы храним указатель.

Объект может быть больше или меньше стандартной полки.

Если объект меньше полки, он все равно займет ее целиком, обычно нельзя хранить два объекта вместе в одном байте (тут есть исключения).

Если объект больше полки, то он будет занимать несколько полок подряд. Тогда надо еще знать, сколько места он занимает (размер, длину) или на какой полке заканчивается.

Даже одно целое число (тип int) занимает 4 байта. А инстанции классов – десятки и сотни байт.

Сам **класс** не хранится на складе. Это просто описание объектов, из чего они состоят и сколько места занимают (если знаем адрес начала объекта, то по размеру класса найдем, где он заканчивается).

Кто управляет этим складом? В первую очередь, операционная система (ОС). Она выделяет программам память для работы. Одна программа не должна свободно залезать в данные другой программы – за этим тоже следит ОС. Своей областью памяти программа распоряжается сама (т.е. делает то, что в нее заложил программист). Свою роль играет и .NET и среда CLR, в которой выполняются все программы на C#.

Непосредственно за создание объектов в памяти отвечает сам программист. **Создать объект** – значит:

1. объявить переменную, в которой будем хранить объект;

Cat Murka

1. выделить память – занять для нового объекта нужное количество ячеек памяти;

new Cat

1. записать в переменную указатель на созданный объект.

Cat Murka = new Cat;

Формально, можно создать объект, не помещая его в переменную, но это абсолютно бессмысленно – вы займете место в памяти, но доступа к соответствующим «полкам» у вас не будет.

Еще одно важное действие – удаление объектов, когда они больше не нужны (**освобождение памяти**, чтобы ее можно было использовать под другие объекты). Во многих языках программирования за освобождение памяти отвечает сам программист: он должен специальной командой удалить объект после работы с ним. Иначе этот объект продолжит «висеть» в памяти и отнимать у системы ресурсы – возникает **утечка памяти**.

В .NET за это отвечает сборщик мусора (garbage collector, GC). Среда .NET отслеживает состояние всех созданных объектов в памяти и удаляет их, если больше нет ни одной переменной, в которой хранится адрес объекта.

*Замечание*. Это чрезвычайно упрощенное представление о работе GC, достаточное для наших учебных примеров, но совершенно недостаточное для практики. После более подробного ознакомления с классами желательно вернуться к этому вопросу и глубже изучить принципы управления памятью в .NET.

Еще один важный момент, связанный с созданием объектов – это применение конструкторов. **Конструктор** – это особый метод, который вызывается сразу после создания объекта. Обычно он используется для инициализации (ввода начальных значений) полей.

Например, для нашего класса Cat, мы создаем кошку с пустым именем, возрастом 0, мужского пола и весом 1 кг. Конечно, потом мы можем изменить значения этих полей, но лучше было бы сразу создать объект с правильными значениями полей.

Добавим в наш класс Cat конструктор. В языке C# он объявляется как метод с таким же именем, как и класс, и не возвращающий никакого значения (даже типа void).

class Cat //класс Кошка

{

string name = ""; //имя

byte age = 0; //возраст

boolean female = false; // пол

double weight = 1; //вес

//конструктор

public Cat (string name, int age, boolean female, double weight)

{

this.name = name;

this.age = age;

this.female = female;

this.weight = weight;

}

... //дальше все остальные методы, которые были предложены выше

}

Мы использовали простейший вариант конструктора, переписывающий значения из параметров в соответствующие поля.

Здесь мы использовали два новых для нас зарезервированных слова:

public (публичный, общедоступный) – означает, что конструктор можно использовать не только внутри класса, но и в любой другой точке программы;

this (это) – встроенная переменная, доступная только внутри методов класса, которая означает текущий экземпляр объекта (в какой бы переменной он не был записан). Мы использовали ее, чтобы отличить аргументы конструктора и поля класса, которые называются одинаково (и это не ошибка). Т.е. в строке

this.name = name;

this.name – это поле текущего объекта, а просто name – это аргумент, указанный в скобках.

Теперь при создании нового объекта класса Cat мы будем перечислять в скобках все значения аргументов.

Cat murka = new Cat("Мурка", 3, true, 3.1);

### Задание на самостоятельную работу

#### Задание

Придумайте собственный пример класса и его экземпляра (на любую тему). Опишите свой пример в документе Word (название класса, перечень его атрибутов и методов, экземпляр с заполненными атрибутами). Реализовывать программно не нужно.

Класс должен содержать не менее 5 атрибутов и не менее 3 методов. Как минимум один метод должен иметь параметры.

## Прямоугольник

Спроектировать класс «Прямоугольник», заданный двумя парами координат: верхнего левого угла и правого нижнего угла. Реализовать методы для расчета ширины и высоты, периметра, площади, а также проверки, является ли данный прямоугольник квадратом. Отдельные методы должны перемещать прямоугольник по плоскости без изменения его размера, изменять ширину и высоту прямоугольника без изменения положения верхнего левого угла.

### Указания к выполнению

Сразу оговоримся, что рисовать прямоугольник на экране мы не будем, а будем только хранить и обрабатывать информацию о нем.

Реализуем консольное меню для работы с нашим прямоугольником.

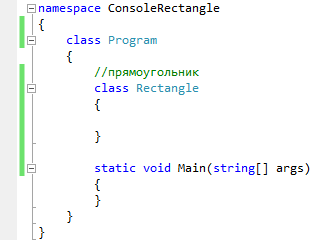
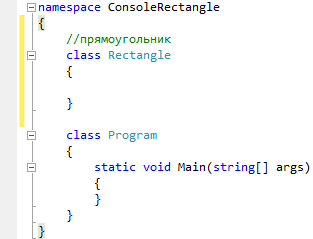
1. При запуске программы у пользователя запрашиваются начальные координаты прямоугольника.
2. На экран выводятся все характеристики прямоугольника (координаты, ширина, высота, периметр, площадь) и меню:
   1. является ли прямоугольник квадратом
   2. переместить прямоугольник
   3. изменить ширину
   4. изменить высоту

Esc - выход

Создайте новое консольное приложение под именем ConsoleRectangle (консольный прямоугольник).

В первую очередь, необходимо объявить класс прямоугольника. Назовем его Rectangle.

Объявлять свой класс можно как внутри класса Program, так и снаружи. Пока вы работаете с ним только в пределах одной программы, разницы не будет.



Мы не будем реализовывать сразу все пункты задания, а будем создавать наш класс постепенно, одну за другой добавляя ему новые возможности.

Начнем с полей класса и конструктора.

В задании сказано, что прямоугольник задается координатами двух углов (именно так вы рисуете прямоугольники в Word или Paint). Обратите внимание, что на экране ось Y направлена сверху вниз, а не снизу вверх, как мы обычно ее изображаем. Т.е. чем ниже точка, тем больше у нее координата *y*.

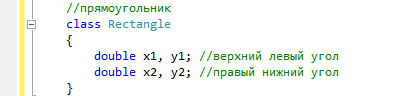
(x1, y1)

(x2, y2)

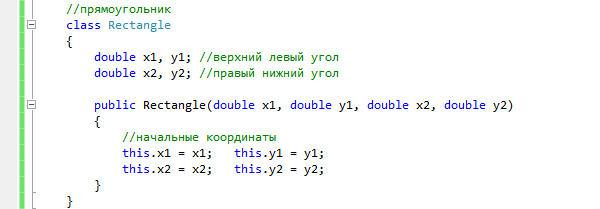
x

y

Добавим в наш класс поля x1, y1, x2, y2. Начальные значения задавать не будем, они будут вводиться в конструкторе.

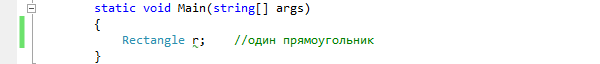


Добавим конструктор:



На этом пока оставим сам класс (мы сделали для него минимум – возможность создать новый объект).

Перейдем к самой программе. В первую очередь, объявим переменную для нашего прямоугольника, но создавать его пока не будем – мы ведь еще не ввели координаты.



Далее, после запуска программы пользователь должен ввести все четыре координаты, и будет создан новый прямоугольник. В принципе, можно обойтись одной командой:

r = new Rectangle(double.Parse(Console.ReadLine()),

double.Parse(Console.ReadLine()),

double.Parse(Console.ReadLine()),

double.Parse(Console.ReadLine()));

Но вы и сами видите, насколько это неудобно и непонятно. Кроме того, мы не отлавливаем ошибки и не выводим пользователю никаких пояснений.

Второй вариант – использовать промежуточные переменные для координат. Проверим правильность ввода и сообщим пользователю, что нужно вводить.

// координата x для верхнего левого угла

double x1;

Console.Write("Введите координату x левого верхнего угла: ");

while (!double.TryParse(Console.ReadLine(), out x1))

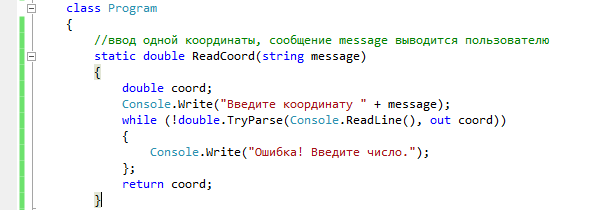
{

Console.Write("Ошибка! Введите число.");

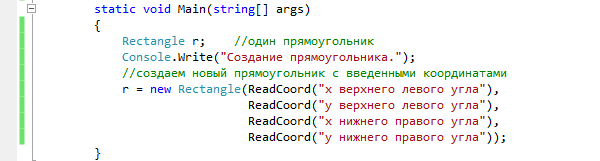
};

Эту конструкцию придется повторить еще 3 раза для остальных координат. При копировании и вставке очень велика вероятность опечаток (например, вместо y2 написать y1). Да и места займет много.

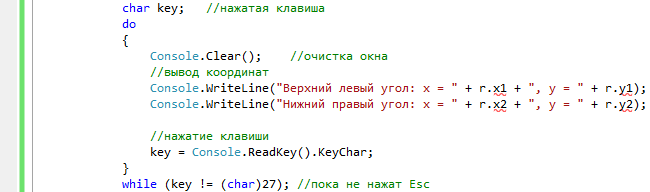
Правильнее всего – создать отдельный метод для ввода координат, как мы делали для консольного калькулятора. При этом нам нужно каждый раз выводить разное пояснение пользователю о том, что он должен ввести, а значит, текст сообщения следует передавать в качестве параметра:



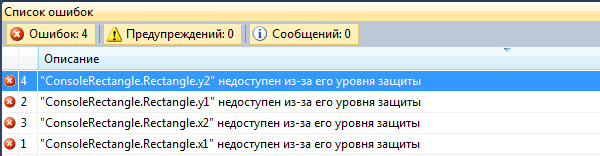
Теперь можно создать прямоугольник почти как в первоначальном варианте, но со всеми проверками и пояснениями. Кроме того, текстовый аргумент позволяет лучше видеть, где какая координата:



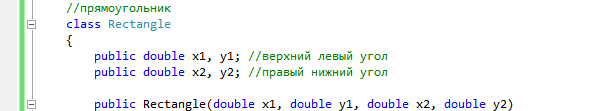
Проверим, что получилось – выведем на экран наши координаты. Меню пока выводить не будем, а вот выход по нажатию ESC – сделаем.



Как видите, среда VisualStudio подчеркнула координаты красным цветом, что свидетельствует об ошибке. Мы намерено допустили ее, чтобы акцентировать на этом внимание.

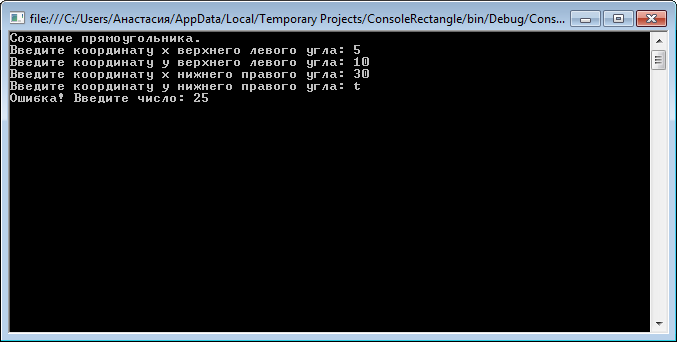


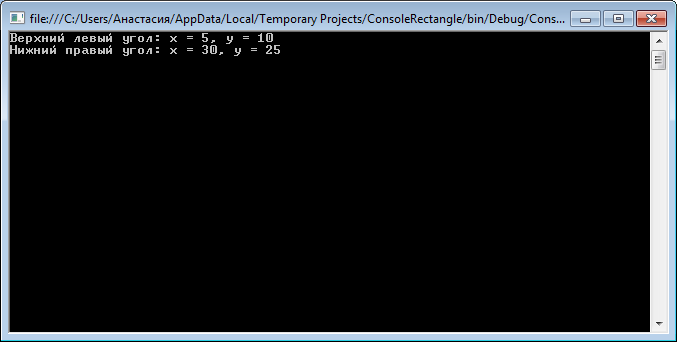
Дело в том, что поля, объявленные в классе, по умолчанию являются **приватными** (private), т.е. доступными только **внутри** класса. Нам же нужно их использовать снаружи, в нашей программе. Тогда, как и конструктор, поля нужно объявить с ключевым словом public:



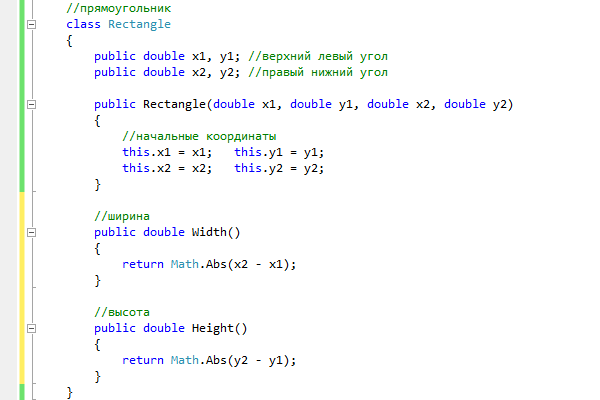
Теперь программа должна откомпилироваться и запуститься без ошибок.

* Зато есть несколько недочетов в оформлении сообщений пользователю. Исправьте их самостоятельно.





Добавим в наш класс методы для вычисления ширины и высоты прямоугольника. Как вы уже могли догадаться, их тоже следует объявлять с ключевым словом public, иначе они будут недоступны в остальной программе.

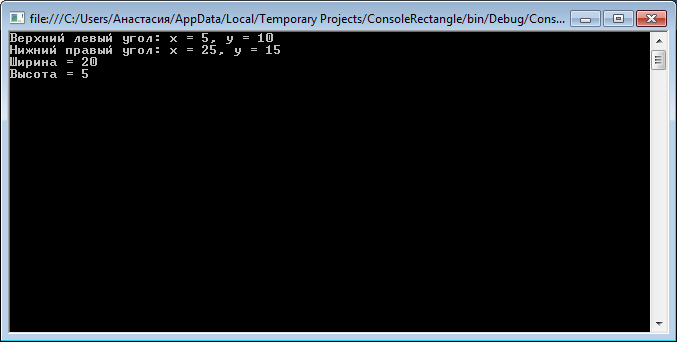


Как видите, методы реализуются в одну строчку. Им не нужны никакие аргументы для вычислений, т.к. координаты хранятся в самом объекте. Мы также берем разность координат по модулю (без знака), т.к. при вводе координат не контролируем, чтобы x2 было больше x1, а y2 больше y1.

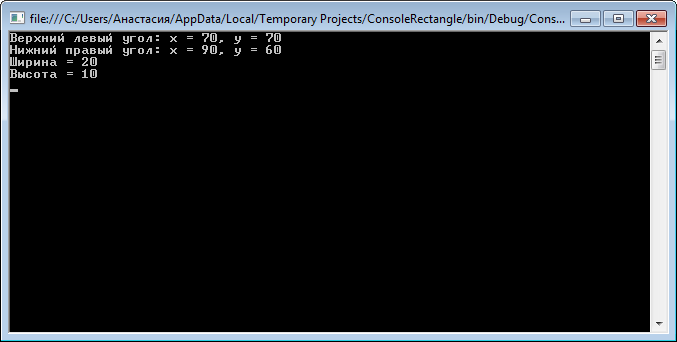
Добавим вывод ширины и высоты в консоли:



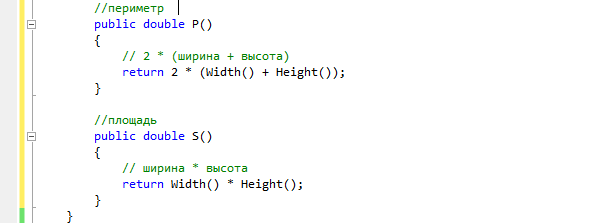
Проверим работу программы:



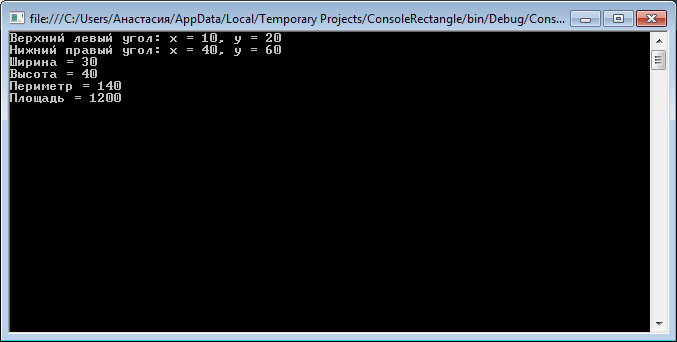
Программа работает, даже если y2 < y1.



Полностью аналогично добавляется расчет периметра и площади. При этом нам не надо опять высчитывать длину и ширину, мы воспользуемся тем, что уже создали:



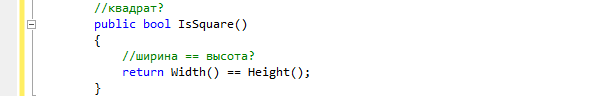
Самостоятельно добавьте в основную программу строки для вывода этой информации в консоль:



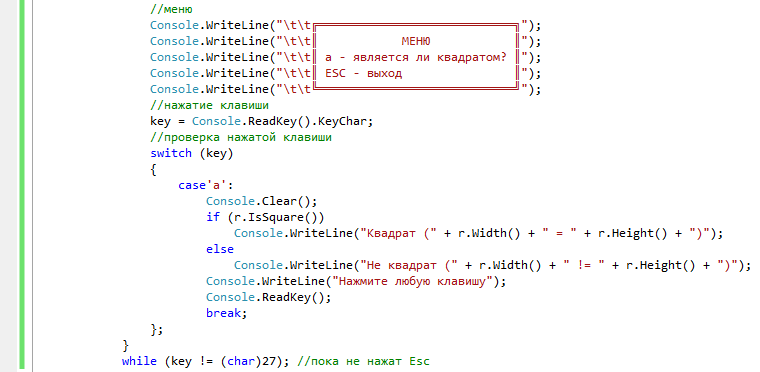
Таким образом, мы реализовали стартовый функционал программы без меню.

Выполните коммит с комментарием «Л5. П1. Создание прямоугольника».

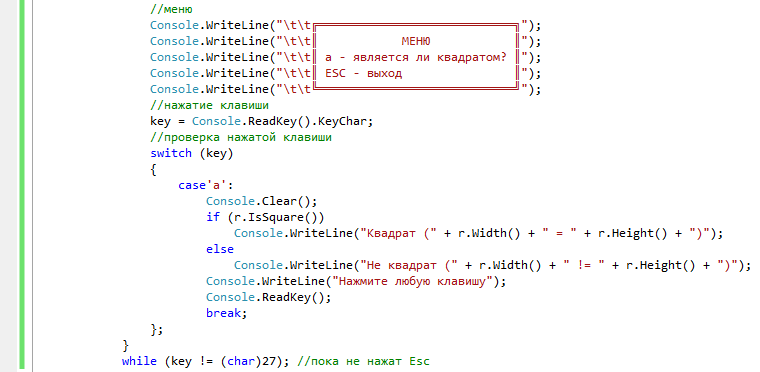
Начнем реализовывать меню. Добавим проверку, является ли прямоугольник квадратом (ширина = высоте). Проверка «является - не является» – это всегда метод, возвращающий true/false, т.е. тип bool. Этот метод тоже реализуется в 1 строку.

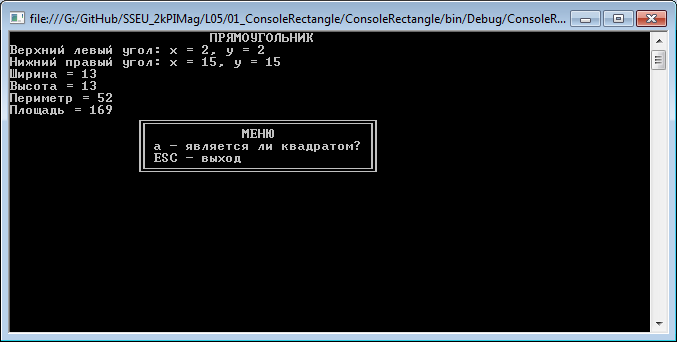


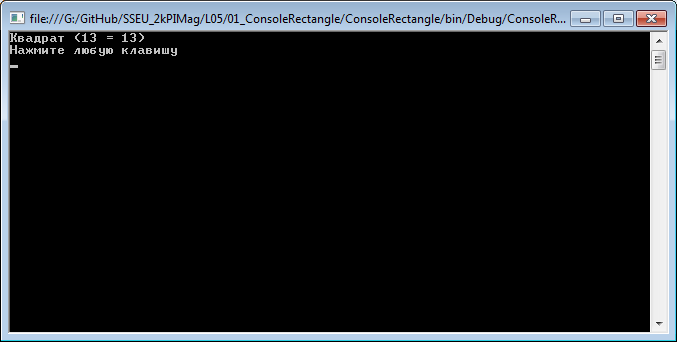
Добавим меню для вызова этого метода. Для разнообразия, в этот раз будем использовать в качестве команд латинские буквы.

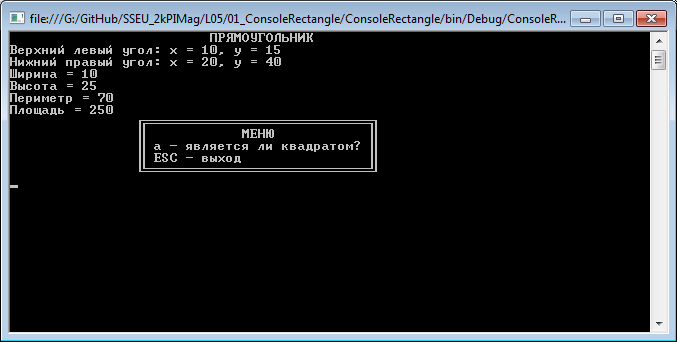


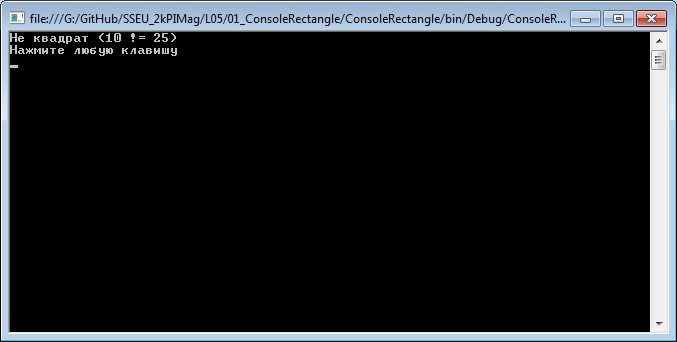
При нажатии буквы ‘a’ нужно вывести, является ли прямоугольник квадратом. Для контроля выведем еще длину и ширину:



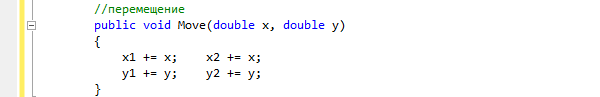






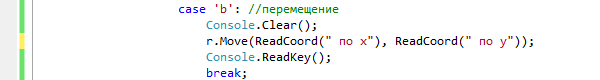


Следующий шаг – перемещение прямоугольника на плоскости. Это значит, что нужно одновременно изменить его координаты x и y для обоих углов.

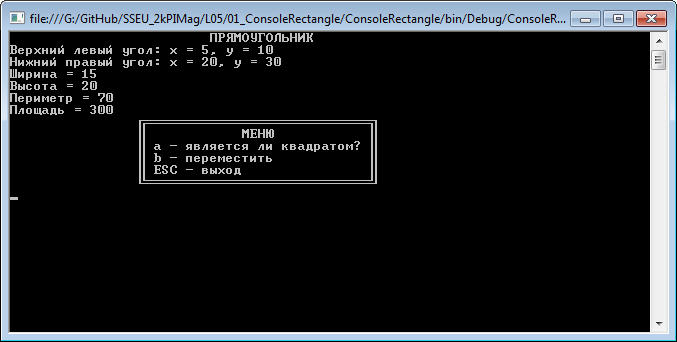


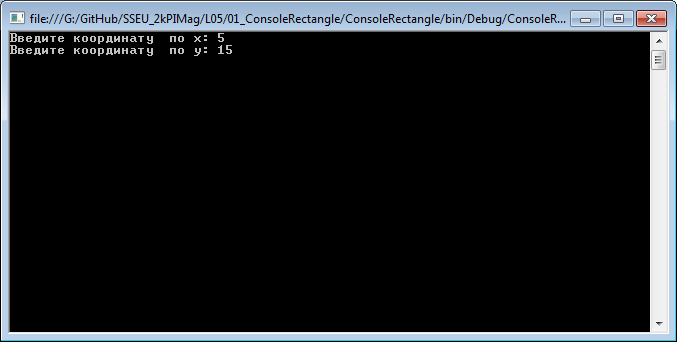
Метод не возвращает никакого значения (void).

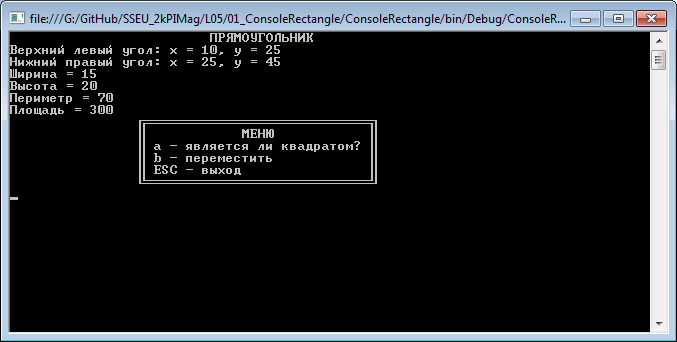
Добавим в меню новую команду. Для считывания координат (насколько подвинуть прямоугольник по каждой оси), воспользуемся ранее созданным методом ReadCoord.



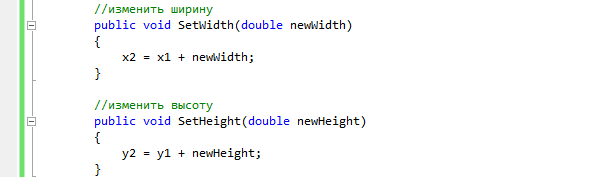
Надпись в меню добавьте самостоятельно.





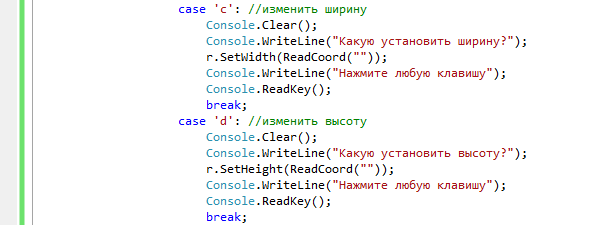


Осталось добавить возможность менять ширину и высоту прямоугольника. Соответствующие методы будут не сложнее предыдущих:

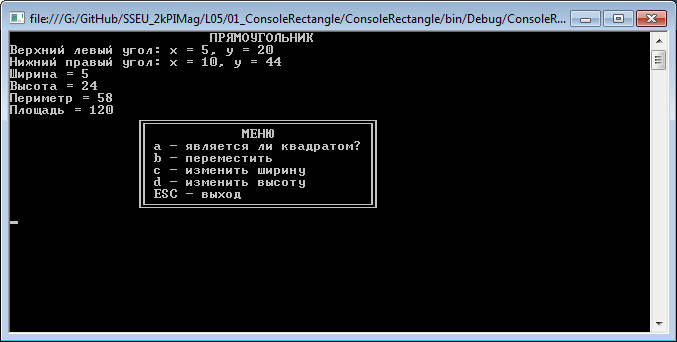


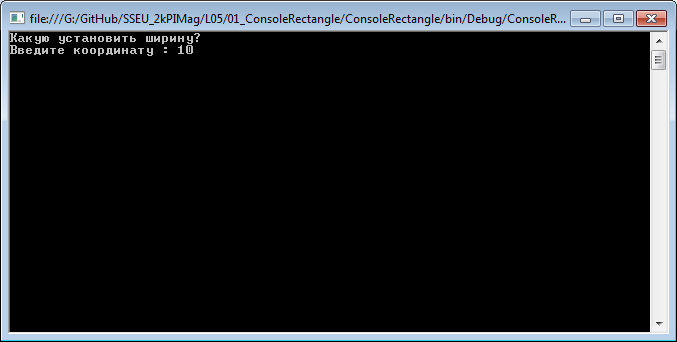
* Какой тип возвращают эти методы? Почему?

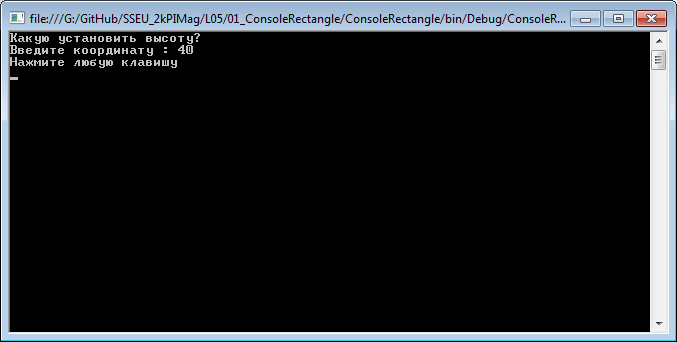
Команды меню:

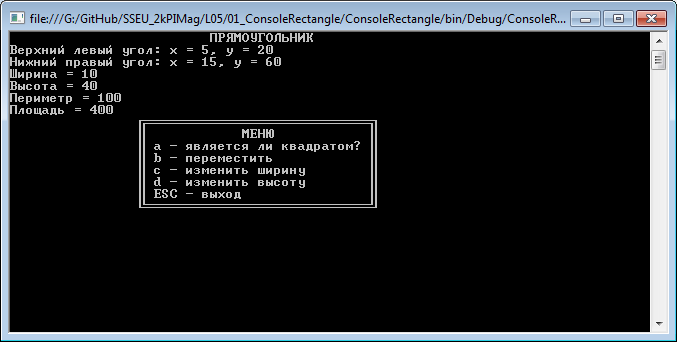


Проверим работу программы:









Выполните коммит с комментарием «Л5. П1. Прямоугольник. Реализовано меню».

### Задание на самостоятельную работу

#### Задание

Реализуйте класс «Окружность», задающийся координатами центра окружности и ее радиусом. Реализовать методы вычисления площади и периметра окружности, перемещения окружности на плоскости, уменьшения площади в указанное число раз, а также проверки, попадает ли заданная точка внутрь окружности.

## Сотрудник

Спроектируйте и реализуйте класс «Сотрудник». О сотруднике необходимо хранить ФИО, дату рождения, должность, оклад, работает ли он по совместительству. Сотрудника можно принять на работу (назначить должность и оклад), выдать зарплату (сообщить сумму к выдаче с учетом премии и НДФЛ, премия вводится отдельно).

Создать оконный интерфейс для заполнения полей одного экземпляра класса и вызова его методов.

### Указания к выполнению

### Задание на самостоятельную работу

#### Задание

Спроектируйте и реализуйте класс «Смартфон». Самостоятельно подберите 5-7 полей, являющихся основными характеристиками смартфона.

Реализовать метод «Сравнение», который выдает в текстовом виде результаты сравнения всех полей переданного в качестве параметра экземпляра с текущим.

Например, если есть атрибут «Диагональ экрана», и у текущего смартфона диагональ экрана больше, то можно выдать сообщение вида: «Моя диагональ экрана больше, чем у <название второго смартфона> на <xx> дюймов».

Создать оконный интерфейс для заполнения полей двух экземпляров класса и вызова метода их сравнения.