

**УО «МОГИЛЕВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ
А.А.КУЛЕШОВА»**

СОЦИАЛЬНО-ГУМАНИТАРНЫЙ КОЛЛЕДЖ



Дисциплина

«Конструирование программ и языки программирования»

Классы

(4 часа)

Методические рекомендации к лабораторной работе №5

Могилев

Базовые понятия по объектно-ориентированному программированию. Понятия «Класс», «Конструктор класса», «Метод»,. Методические указания по лабораторной работе №5 по дисциплине «Конструирование программ и языки программирования». Для учащихся 3 курса очной формы обучения специальности 2–40 01 01 «Программное обеспечение информационных технологий».

Оглавление

1 Цель работы	4
2 Ход работы.....	5
3 Краткие теоретические сведения	6
3.1 Понятие «Класс»	6
3.1.1 Секция данных.....	7
3.1.2 Секция методов.	8
3.2 Процедуры и функции - методы класса	9
3.2.1 Описание методов (процедур и функций). Синтаксис	9
3.2.2 Список формальных аргументов.....	10
3.2.3 Тело метода. Вызов метода. Синтаксис.....	10
3.2.4 Вызовы статических и динамических методов	12
3.3 Добавление нового класса в MS Visual Studio 2015	13
4. Задания	16
5 Контрольные вопросы	22

1 Цель работы

- Познакомить с классами в C#.
- Дать описание процессу создания класса.
- Познакомить с атрибутами и методами классов.
- Дать описание методам доступа к элементам класса.

2 Ход работы

1. Изучение теоретического материала.
2. Выполнение практических индивидуальных заданий по вариантам (вариант уточняйте у преподавателя).
3. Оформление отчета.
 - 3.1. Отчет оформляется индивидуально каждым студентом. Отчет должен содержать задание, алгоритм и листинг программы.
 - 3.2. Отчет по лабораторной работе выполняется на листах формата А4. В состав отчета входят:
 - 1) титульный лист;
 - 2) цель работы;
 - 3) текст индивидуального задания;
 - 4) выполнение индивидуального задания.
4. Контрольные вопросы

3 Краткие теоретические сведения

3.1 Понятие «Класс»

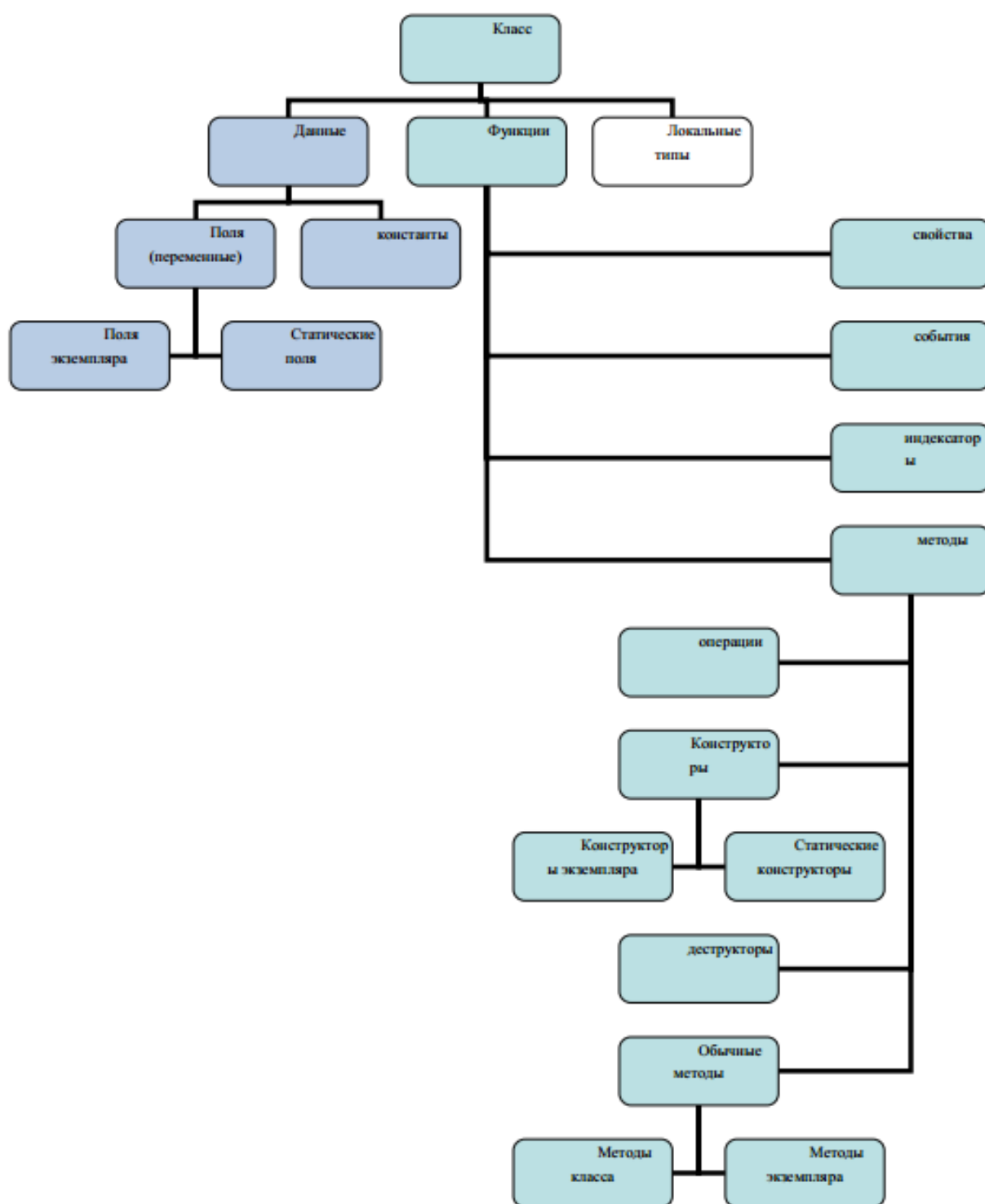


Рисунок 3.1 Класс

Класс – это абстрактное понятие, сравнимое с понятием *категория* в его обычном смысле.

Спецификаторы класса

1. `new` – используется для вложенных классов. Задаёт новое описание класса взамен

унаследованного от предка. Применяется в иерархиях объектов.

2. `public` – доступ не ограничен
3. `protected` – используется для вложенных классов. Доступ только из элементов данного и производных классов
4. `internal` – доступ только из данной программы (сборки)
5. `protected internal` – доступ только из данного и производных классов или из данной программы (сборки)
6. `private` – используется для вложенных классов. Доступ только из элементов класса, внутри которого описан данный класс
7. `abstract` – абстрактный класс. Применяется в иерархиях объектов.
8. `sealed` – бесплодный класс. Применяется в иерархиях объектов.
9. `static` – статический класс. Введен в версию языка 2.0.

Спецификаторы 2-6 называются **спецификаторами доступа**.

Они определяют, откуда можно непосредственно обращаться к данному классу. Спецификаторы доступа могут присутствовать в описании только в вариантах, приведенных в таблице, а также могут комбинироваться с остальными спецификаторами

Класс в объектно-ориентированном программировании – это абстрактный тип данных, который включает в себя не только **данные**, но и **функции и процедуры**. Функции и процедуры класса называются **методами** и содержат исходный **код**, предназначенный для обработки внутренних **данных** объекта данного класса.

Класс можно также охарактеризовать так: **Класс** – описание множества **объектов** и выполняемых над ними **действий**.

Класс описывается (декларируется) следующим образом:

```
Class <Имя_класса>
{
    Секция данных (поля класса)
    Секция методов
}
```

3.1.1 Секция данных.

В секции данных можно описывать переменные, константы, структуры, перечисления и прочее – все, что можно отнести к понятию «данные». Каждый член этой секции может быть описан следующим образом:

[модификатор доступа][статический/нестатический]<тип данных><Имя поля>;

Модификаторы доступа служат для ограничения доступа как к полям класса, так и к методам класса. Ограничения необходимы для того, что бы облегчить создание сложных программных систем. Рассмотрим два основных модификатора доступа – **private** и **public**.

Модификатор **private**. Если при объявлении класса вы не укажете модификаторы доступа для полей и методов класса, то по умолчанию компилятор будет полагать, что для них нужно использовать модификатор доступа **private**.

Модификатор **private** запрещает доступ к полям и методам класса извне самого класса.

Поля и методы, объявленные с модификатором **private**, будут доступны только в методах самого класса, т.е. это «личные» поля класса (если не указать вообще модификатора доступа, то по умолчанию считается, что тип доступа к данным и методам **private**).

Пример:

```
class Point
{
    private int xPos;
    private int yPos;
}
```

Модификатор **public**. Если поле или метод класса определены с модификатором доступа **public**, они доступны вне объявления базового класса или производных классов. Это, в частности, означает, что методы, объявленные вне класса, могут свободно обращаться к таким полям и методам.

Предыдущий пример можно модифицировать следующим способом:

```
class Point
{
    public int xPos;
    public int yPos;
}
```

И тогда к координатам точки можно будет обратиться непосредственно из другого метода. Так обычно не поступают, т.к. это является нарушением одного из основных принципов объектно-ориентированного программирования. Чаще всего поля класса имеют уровень доступа **private**, а для того что бы их прочитать или модифицировать использует специально написанные методы. Это необходимо для того, что бы случайно не нарушить хранимую в полях информацию.

Кроме объявления модификатора доступа, при описании поля или метода необходимо указать статическое это поле или метод – **static** или нестатическое (по умолчанию слово **static** опускается и поле или метод считаются нестатическими).

Ключевым словом **static** помечаются те данные и методы, которые относятся ко всему классу – они считаются статическими, т.е. расположенными в самом классе, а те данные, которые располагаются в объектах (экземплярах класса) – являются нестатическими.

3.1.2 Секция методов.

Первыми формами модульности, появившимися в языках программирования, были *процедуры* и *функции*. Они позволяли задавать определенную функциональность и многократно выполнять один и тот же параметризованный программный код при различных значениях параметров. Поскольку *функции* в математике использовались издавна, то появление их в языках программирования было совершенно естественным. Уже с первых шагов *процедуры* и *функции* позволяли решать одну из важнейших задач, стоящих перед программистами, – задачу по-

вторного использования программного кода. Встроенные в язык **функции** давали возможность существенно расширить возможности языка программирования.

3.2 Процедуры и функции - методы класса

Долгое время **процедуры** и **функции** играли не только функциональную, но и архитектурную роль. Весьма популярным при построении программных систем был метод функциональной декомпозиции "сверху вниз", и сегодня еще играющий важную роль. Но с появлением ООП архитектурная роль функциональных модулей отошла на второй план. Для объектно-ориентированных языков, к которым относится и язык C#, в роли архитектурного модуля выступает **класс**. Программная система строится из модулей, роль которых играют **классы**, но каждый из этих модулей имеет содержательную начинку, задавая некоторую абстракцию данных.

Процедуры и **функции** связываются теперь с **классом**, они обеспечивают функциональность данных **класса** и называются методами **класса**. Главную роль в программной системе играют данные, а **функции** лишь служат данным. В C# **процедуры** и **функции** существуют только как **методы** некоторого **класса**, они не существуют вне **класса**.

В языке C# нет специальных ключевых слов – **procedure** и **function**, но присутствуют сами эти понятия. Синтаксис объявления метода позволяет однозначно определить, чем является **метод** – **процедурой** или **функцией**. Прежнюю роль библиотек **процедур** и **функций** теперь играют **библиотеки классов**.

Процедуры и функции. Отличия.

Функция отличается от **процедуры** двумя особенностями:

- всегда вычисляет некоторое значение, возвращаемое в качестве результата **функции**;
- вызывается в выражениях.

Процедура C# имеет свои особенности:

- возвращает формальный результат `void`, указывающий на отсутствие результата;
- вызов **процедуры** является оператором языка;
- имеет входные и выходные аргументы, причем выходных аргументов - ее результатов -

может быть достаточно много.

3.2.1 Описание методов (процедур и функций). Синтаксис

Синтаксически в описании метода различают две части – **описание заголовка** и **описание тела** метода.

Рассмотрим синтаксис заголовка метода:

```
[модификаторы]<void| тип_результата>  
<имя_метода>([список_формальных_аргументов])
```

Имя метода и **список формальных аргументов** составляют **сигнатуру метода**. В **сигнатуру** не входят имена формальных. В **сигнатуру** не входит и тип возвращаемого результата.

Квадратные скобки (метасимволы синтаксической формулы) показывают, что модификаторы могут быть опущены при описании метода.

Обязательным при **описании заголовка** является указание типа результата, имени метода и круглых скобок, наличие которых необходимо и в том случае, если сам список формальных

аргументов отсутствует. Формально тип результата метода указывается всегда, но значение `void` однозначно определяет, что метод реализуется *процедурой*. Тип результата, отличный от `void`, указывает на *функцию*. Вот несколько простейших примеров описания методов:

```
void A() {...};  
  
int B(){...};  
  
public void C(){...};
```

Методы А и В являются *private*, а метод С – *public*. Методы А и С реализованы *процедурами*, а метод В – *функцией*, возвращающей целое значение.

3.2.2 Список формальных аргументов

Список формальных аргументов метода может быть пустым, и это довольно типичная ситуация для методов *класса*. Список может содержать фиксированное число аргументов, разделяемых символом запятой.

Рассмотрим теперь синтаксис объявления формального аргумента:

```
[ref|out|params]<тип_аргумента> <имя_аргумента>
```

Обязательным является указание типа и имени аргумента. Заметьте, никаких ограничений на тип аргумента не накладывается. Он может быть любым скалярным типом, массивом, *классом*, структурой, интерфейсом, перечислением, функциональным типом.

Несмотря на фиксированное число формальных аргументов, есть возможность при *вызове метода* передавать ему *произвольное число фактических аргументов*. Для реализации этой возможности в списке формальных аргументов необходимо задать ключевое слово *params*. Оно задается один раз и указывается только для последнего аргумента списка, объявляемого как массив произвольного типа. При *вызове метода* этому формальному аргументу соответствует *произвольное число фактических аргументов*.

Содержательно, все аргументы метода разделяются на три группы: *входные*, *выходные* и *обновляемые*. Аргументы первой группы передают информацию методу, их значения в теле метода только читаются. Аргументы второй группы представляют собой результаты метода, они получают значения в ходе работы метода. Аргументы третьей группы выполняют обе функции. Их значения используются в ходе вычислений и обновляются в результате работы метода. Выходные аргументы всегда должны сопровождаться ключевым словом *out*, обновляемые – *ref*. Что же касается входных аргументов, то, как правило, они задаются без ключевого слова, хотя иногда их полезно объявлять с параметром *ref*, о чем подробнее скажу чуть позже. Заметьте, если аргумент объявлен как выходной с ключевым словом *out*, то в теле метода обязательно должен присутствовать оператор присваивания, задающий значение этому аргументу. В противном случае возникает ошибка еще на этапе компиляции. Чаще всего можно обойтись без ключевых слов *params*, *out*, *ref*.

3.2.3 Тело метода. Вызов метода. Синтаксис

Синтаксически тело метода является *блоком*, который представляет собой последовательность операторов и описаний переменных, заключенную в фигурные скобки. Если речь идет о теле *функции*, то в *блоке* должен быть хотя бы один оператор перехода, возвращающий значение *функции* в форме *return <выражение>*.

Как уже отмечалось, метод может вызываться в выражениях или быть вызван как оператор. В качестве оператора может использоваться любой метод - как *процедура*, так и *функция*.

Если же попытаться вызвать *процедуру* в выражении, то это приведет к ошибке еще на этапе компиляции. Возвращаемое *процедурой* значение `void` несовместимо с выражениями. Так что в выражениях могут быть вызваны только *функции*.

Сам *вызов метода*, независимо от того, *процедура* это или *функция*, имеет один и тот же синтаксис:

```
имя_метода ([список_фактических_аргументов])
```

Если это оператор, то вызов завершается точкой с запятой.

Рассмотрим следующий пример (класс, описывающий базовое поведение прямоугольника):

```
class Rectangle
{
    private int _xPos;
    private int _yPos;
    private double _height;
    private double _weight;
    private static int _angle = 90;
    private static string _className = "Rectangle";

    public Rectangle(int x1, int y1, int x2, int y2)
    {
        _xPos = x1;
        _yPos = y1;
        _height = y2 - y1;
        _weight = x2 - x1;
    }

    public double GetHeight()
    {
        return _height;
    }

    public double GetWeight()
    {
        return _weight;
    }

    public double GetPerimeter()
    {
        double perimeter = (_height + _weight) * 2;
        return perimeter;
    }
}
```

```

    public void SetRectanglePosition(int xPos, int yPos)
    {
        _xPos = xPos;
        _yPos = yPos;
    }
    public static string GetClassName()
    {
        return _className;
    }
}

```

В качестве полей класса используются следующие переменные: `xPos`, `yPos`, `_height`, `_weight` – координаты вершины треугольника, высота и ширина, эти переменные должны быть у каждого объекта (прямоугольника) разные. Кроме этого, есть два поля одинаковые для всех прямоугольников – `_angle` (угол прямоугольника) и имя класса – `_className`, эти переменные объявлены как статические. Все поля закрыты для доступа извне (модификатор доступа **private**).

Далее описывается специальный метод – `public Rectangle(...)` – его имя совпадает с именем класса – это конструктор объекта, специальный метод, который вызывается при создании экземпляра класса (объекта), в него передаются данные для инициализации всех полей объекта, в этом методе рассчитываются все данные необходимые объекту (`_height` и `_weight` – высота и ширина).

Далее описаны методы, возвращающие или рассчитывающие данные – `GetHeight`, `GetWeight` и `GetPerimeter`.

Далее описывается метод, который меняет позицию уже созданного прямоугольника (перемещает на координатной плоскости) – `SetRectanglePosition`, в этот метод передатся два параметра – новые координаты вершины прямоугольника.

Далее описывается статический метод `GetClassName` – он возвращает общую для всех прямо- угольников информацию – имя класса.

3.2.4 Вызовы статических и динамических методов

Для того что бы вызвать динамический метод, необходимо создать экземпляр класса (объект), для этого используется ключевое слово **new**.

Общий вид создания объекта выглядит следующим образом:

```

<ИмяКласса> <ИмяОбъекта> = new <ИмяКонструктораКласса> ([список па-
                                     раметров])

```

При создании объекта список параметров может быть пустым.

Пример:

```

Rectangle rect = new Rectangle(1, 1, 10, 10);

```

При создании объекта `rect` – вызывается конструктор класса, в него передаются координаты двух вершин прямоугольника – 1,1 и 10,10. В конструкторе инициализируются поля `_xPos`

и `_xPos`, а также рассчитывается длина и ширина прямоугольника.

После того, как объект `rect` создан, можно вызывать его методы:

```
double perimeter;  
perimeter = rect.GetPerimeter();  
rect.SetRectanglePosition(15, 15);
```

В примере создается переменная `perimeter` и вычисляется (вызывается динамический метод `GetPerimeter`), а также задаются другие координаты прямоугольника (вызывается метод `SetRectanglePosition`).

Для того, что бы вызвать статический метод, необходимо обратиться не к объекту, а к самому классу:

```
string className;  
className = Rectangle.GetClassName();
```

Полный пример использования класса `Rectangle`:

```
static void Main(string[] args)  
{  
    Rectangle rect = new Rectangle(1, 1, 10, 10);  
    double perimeter;  
    perimeter = rect.GetPerimeter();  
    string className;  
    className = Rectangle.GetClassName();  
    Console.WriteLine(perimeter);  
    Console.WriteLine(className);  
    rect.SetRectanglePosition(15, 15);  
    Console.ReadKey();  
}
```

3.3 Добавление нового класса в MS Visual Studio 2017

Чтобы добавить новый класс, необходимо открыть `Solution Explorer` и щелкнуть правой клавишей мыши на имени проекта (рисунок 3.2), выбрать пункт «Add-> New Item...» или «Add-> Class»

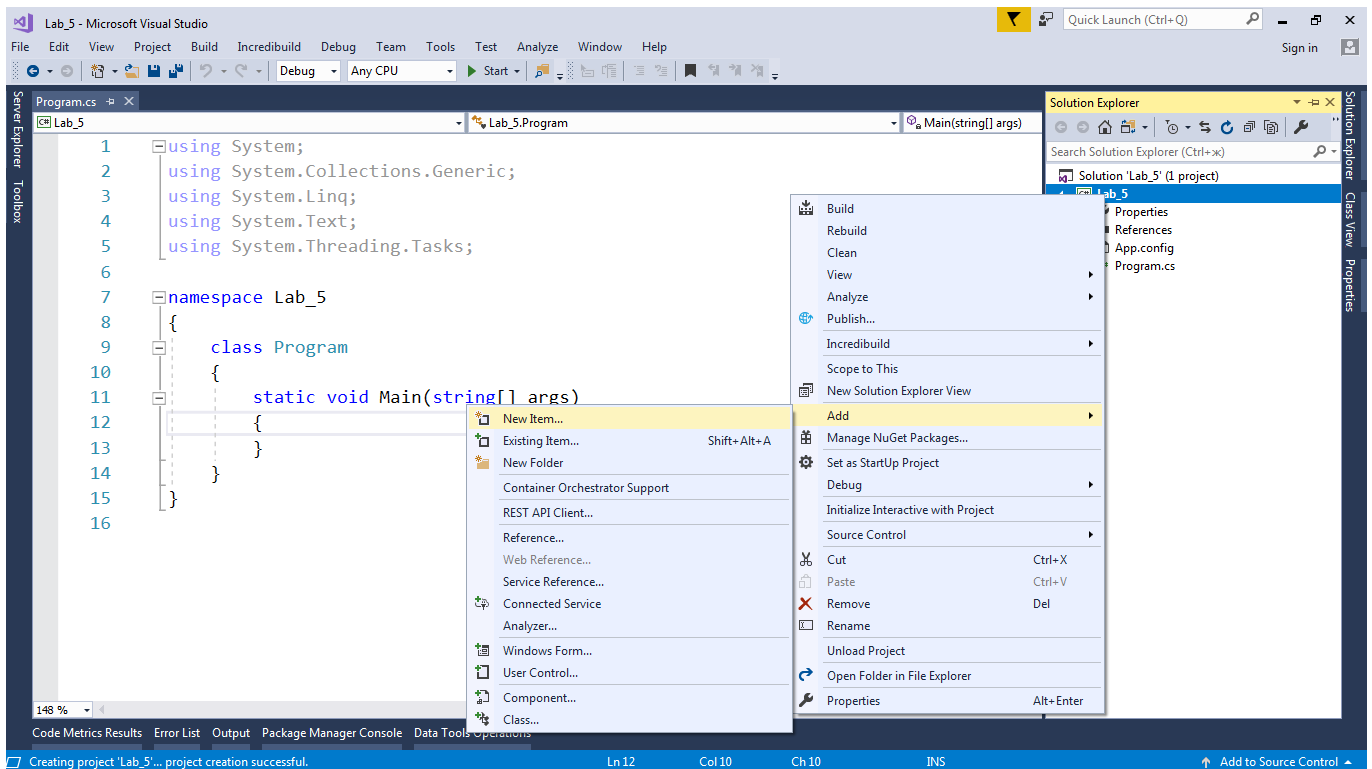


Рисунок 3.2 – Добавление нового класса.

В открывшемся окне (Рисунок 3.3) – выбрать шаблон «Class», набрать его имя (Name) – Rectangle, и нажать кнопку «Add».

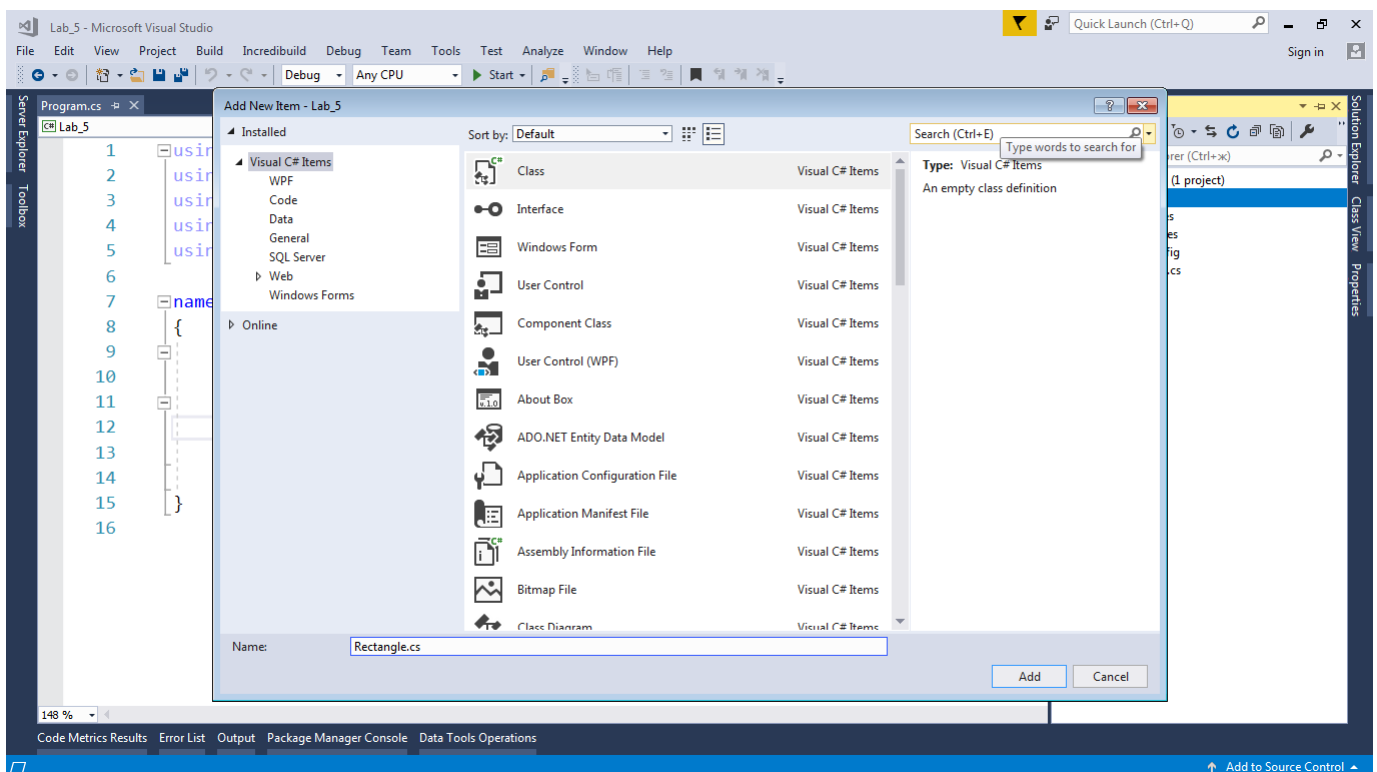


Рисунок 3.3 – Диалог добавления нового класса.

В открывшемся окне появится заготовка нового класса:
using System;

```

using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab_5 // namespace Lab_5.Rectangle
{
    class Rectangle
    {
    }
}

```

Далее необходимо добавить содержимое класса. В случае, если у классов разное пространство имен, то для вызова в методе Main необходимо подключить соответствующее имя пространства имен (например, `using Lab_5.Rect;`):

```

using System;
//using Lab_5.Rect;

namespace Task10
{
    class Starter
    {
        static void Main(string[] args)
        {
            Rectangle rect = new Rectangle(1, 1, 10, 10);
            double perimeter;
            perimeter = rect.GetPerimeter();
            string className;
            className = Rectangle.GetClassName();
            Console.WriteLine(perimeter);
            Console.WriteLine(className);
            rect.SetRectanglePosition(15, 15);
            Console.ReadKey();
        }
    }
}

```

4. Задания

Задание 1. Выполнить задания по вариантам. (базовый уровень)

В классе Rectangle дописать следующие методы:

1. GetSquare – вычислить площадь;
2. GetXPos – вернуть координату X вершины прямоугольника;
3. GetYPos – вернуть координату Y вершины прямоугольника;
4. GetRectangleAngle – вернуть значение угла прямоугольника.
5. SetClassName – задать имя класса.
6. Дописать метод Main таким образом, что бы использовались все методы класса Rectangle.

Реализовать класс (за основу принять класс Rectangle). Количество и содержание методов проработать самостоятельно.

1	Эллипс	9	Конус
2	Окружность	10	Цилиндр
3	Трапеция	11	Пирамида
4	Треугольник	12	Призма
5	Параллелограмм	13	Прямоугольник
6	Ромб	14	Прямоугольная пирамида
7	Квадрат	15	Октаэдр
8	Куб	16	Сфера

Задание 2. Выполнить задания по вариантам. (Нормальный уровень)

1. Создать класс Point, разработав следующие элементы класса:
 - а. Поля:
 - int x, y;
 - б. Конструкторы, позволяющие создать экземпляр класса:
 - с нулевыми координатами;
 - с заданными координатами.
 - в. Методы, позволяющие:
 - вывести координаты точки на экран;
 - рассчитать расстояние от начала координат до точки;
 - переместить точку на плоскости на вектор (a, b).
 - г. Свойства:
 - получить-установить координаты точки (доступное для чтений и записи);
 - позволяющие умножить координаты точки на скаляр (доступное только для записи).
2. Создать класс Student, разработав следующие элементы класса:
 - а. Поля
 - string name
 - int group, course;

- б. Конструкторы:
- позволяющий создать экземпляр класса с заданными данными о студенте;
 - конструктор со значениями по умолчанию.
- в. Методы, позволяющие:
- вывести информацию о студентах;
 - позволяющий отсортировать студентов по курсам;
 - приводит расчёт количества студентов в каждой из групп.
- г. Свойства:
- позволяющее узнать о количестве студентов на курсе;
 - позволяющее получить-установить значение полей (доступное для чтения и записи).
3. Создать класс Triangle, разработав следующие элементы класса:
- а. Поля:
- `int a, b, c;`
- б. Конструктор, позволяющий создать экземпляр класса с заданными длинами сторон.
- в. Методы, позволяющие:
- вывести длины сторон треугольника на экран;
 - рассчитать периметр треугольника;
 - рассчитать площадь треугольника.
- г. Свойства:
- позволяющее получить-установить длины сторон треугольника (доступное для чтения и записи);
 - позволяющее установить, существует ли треугольник с данными длинами сторон (доступное только для чтения).
4. Создать класс Rectangle, разработав следующие элементы класса:
- а. Поля:
- `int a, b;`
- б. Конструктор, позволяющий создать экземпляр класса с заданными длинами сторон.
- в. Методы, позволяющие:
- вывести длины сторон прямоугольника на экран;
 - рассчитать периметр прямоугольника;
 - рассчитать площадь прямоугольника.
- г. Свойства:
- получить-установить длины сторон прямоугольника (доступное для чтения и записи);
 - позволяющее установить, является ли данный прямоугольник квадратом (доступное только для чтения).
5. Создать класс Money, разработав следующие элементы класса:
- а. Поля:
- `int first; //номинал купюры`
 - `int second; //количество купюр`
- б. Конструктор, позволяющий создать экземпляр класса с заданными значениям полей.
- в. Методы, позволяющие:
- вывести номинал и количество купюр;
 - определить, хватит ли денежных средств на покупку товара на сумму N рублей.

- определить, сколько шт товара стоимости n рублей можно купить на имеющиеся денежные средства.
- г. Свойства:
- позволяющее получить-установить значение полей (доступное для чтения и записи);
 - позволяющее рассчитать сумму денег (доступное только для чтения).
6. Создать класс для работы с датой. Разработать следующие элементы класса:
- а. Поле
- DateTime data.
- б. Конструкторы, позволяющие установить:
- заданную дату
 - дату 1.01.2009
- в. Методы, позволяющие:
- вычислить дату предыдущего дня;
 - вычислить дату следующего дня;
 - определить сколько дней осталось до конца месяца.
- г. Свойства:
- позволяющее установить или получить значение поля класса (доступно для чтения и записи)
 - позволяющее определить год високосным (доступно только для чтения)
7. Создать класс Point, разработав следующие элементы класса:
- а. Поля:
- int x, y;
- б. Конструкторы, позволяющие создать экземпляр класса:
- с нулевыми координатами;
 - с заданными координатами.
- в. Методы, позволяющие:
- вывести координаты точки на экран;
 - рассчитать расстояние от начала координат до точки;
 - переместить точку на плоскости на вектор (a, b).
- г. Свойства:
- получить-установить координаты точки (доступное для чтений и записи);
 - позволяющие умножить координаты точки на скаляр (доступное только для записи).
8. Создать класс Triangle, разработав следующие элементы класса:
- а. Поля:
- int a, b, c;
- б. Конструктор, позволяющий создать экземпляр класса с заданными длинами сторон.
- в. Методы, позволяющие:
- вывести длины сторон треугольника на экран;
 - рассчитать периметр треугольника;
 - рассчитать площадь треугольника.
- г. Свойства:
- позволяющее получить-установить длины сторон треугольника (доступное для чтения и записи);

- позволяющее установить, существует ли треугольник с данными длинами сторон (доступное только для чтения).
9. Создать класс Rectangle, разработав следующие элементы класса:
- а. Поля:
 - `int a, b;`
 - б. Конструктор, позволяющий создать экземпляр класса с заданными длинами сторон.
 - в. Методы, позволяющие:
 - вывести длины сторон прямоугольника на экран;
 - рассчитать периметр прямоугольника;
 - рассчитать площадь прямоугольника.
 - г. Свойства:
 - получить-установить длины сторон прямоугольника (доступное для чтения и записи);
 - позволяющее установить, является ли данный прямоугольник квадратом (доступное только для чтения).
10. Создать класс Money, разработав следующие элементы класса:
- а. Поля:
 - `int first;` //номинал купюры
 - `int second;` //количество купюр
 - б. Конструктор, позволяющий создать экземпляр класса с заданными значениям полей.
 - в. Методы, позволяющие:
 - вывести номинал и количество купюр;
 - определить, хватит ли денежных средств на покупку товара на сумму N рублей.
 - определить, сколько шт товара стоимости n рублей можно купить на имеющиеся денежные средства.
 - г. Свойства:
 - позволяющее получить-установить значение полей (доступное для чтения и записи);
 - позволяющее рассчитать сумму денег (доступное только для чтения).
11. Создать класс для работы с датой. Разработать следующие элементы класса:
- а. Поле
 - `DateTime data.`
 - б. Конструкторы, позволяющие установить:
 - заданную дату
 - дату 1.01.2009
 - в. Методы, позволяющие:
 - вычислить дату предыдущего дня;
 - вычислить дату следующего дня;
 - определить сколько дней осталось до конца месяца.
 - г. Свойства:
 - позволяющее установить или получить значение поле класса (доступно для чтения и записи)
 - позволяющее определить год высокосным (доступно только для чтения)
12. Создать класс Trapeze, разработав следующие элементы класса:
- а. Поля:

- `int a, b;`
 - б. Конструктор, позволяющий создать экземпляр класса с заданными длинами сторон.
 - в. Методы, позволяющие:
 - вывести длины сторон трапеции на экран;
 - рассчитать периметр трапеции;
 - рассчитать площадь трапеции.
 - г. Свойства:
 - получить-установить длины сторон прямоугольника (доступное для чтения и записи);
 - позволяющее установить, является ли данная трапеция равнобедренной (доступное только для чтения).
13. Создать класс `Student`, разработав следующие элементы класса:
- а. Поля
 - `string name, lastname`
 - `int age, course;`
 - б. Конструкторы:
 - позволяющий создать экземпляр класса с заданными данными о студенте;
 - конструктор без параметров.
 - в. Методы, позволяющие:
 - вывести информацию о студентах;
 - позволяющий отсортировать студентов по возрасту;
 - приводит расчёт количества студентов на каждом из курсов.
 - г. Свойства:
 - позволяющее узнать о количестве несовершеннолетних студентах;
 - позволяющее получить-установить значение полей (доступное для чтения и записи).
14. Создать класс `Triangle`, разработав следующие элементы класса:
- а. Поля:
 - `int a, b, c;`
 - б. Конструкторы:
 - позволяющий создать экземпляр класса с заданными длинами сторон;
 - без параметров.
 - в. Методы, позволяющие:
 - вывести длины сторон треугольника на экран;
 - рассчитать полупериметр треугольника;
 - рассчитать периметр треугольника по формуле Геррона.
 - г. Свойства:
 - позволяющее получить-установить длины сторон треугольника (доступное для чтения и записи);
 - позволяющее установить, существует ли треугольник с данными длинами сторон (доступное только для чтения).
15. Создать класс `Money`, разработав следующие элементы класса:
- а. Поля:
 - `int first; //номинал купюры`
 - `int second; //количество купюр`

- б. Конструктор, позволяющий создать экземпляр класса с заданными значениям полей.
- в. Методы, позволяющие:
 - вывести номинал и количество купюр;
 - определить, хватит ли денежных средств на покупку товара на сумму N \$.
 - Конвертировать валюту (BYN->\$ и наоборот).
- г. Свойства:
 - позволяющее получить-установить значение полей (доступное для чтения и записи);
 - позволяющее рассчитать сумму денег в BYN и \$ (доступное только для чтения).

5 Контрольные вопросы

1. Что понимается под термином «класс»?
2. Какие элементы определяются в составе класса?
3. Каково соотношение понятий «класс» и «объект»?
4. Что понимается под термином «члены класса»?
5. Перечислите пять разновидностей членов класса специфичных для языка C#.
6. Что понимается под термином «конструктор»?
7. Сколько конструкторов может содержать класс языка C#?
8. Приведите синтаксис описания класса в общем виде. Проиллюстрируйте его фрагментом программы на языке C#.
9. Какие модификаторы типа доступа Вам известны?
10. В чем заключаются особенности доступа членов класса с модификатором public?
11. В чем заключаются особенности доступа членов класса с модификатором private?
12. В чем заключаются особенности доступа членов класса с модификатором protected?
13. В чем заключаются особенности доступа членов класса с модификатором internal?
14. Какое ключевое слово языка C# используется при создании объекта?
15. Приведите синтаксис создания объекта в общем виде. Проиллюстрируйте его фрагментом программы на языке C#.
16. В чем состоит назначение конструктора?
17. Каждый ли класс языка C# имеет конструктор?
18. Какие умолчания для конструкторов приняты в языке C#?
19. Каким значением инициализируются по умолчанию значения ссылочного типа?
20. В каком случае конструктор по умолчанию не используется?
21. Приведите синтаксис конструктора класса в общем виде. Проиллюстрируйте его фрагментом программы на языке C#.