**Практическая работа № 5**

**Работа с классами**

**Цель:** Формирование навыков программирования C# и работы с системой MS Visual Studio, *работа с классами*.

### Теоретические сведения:

Каждый объект реального мира обладает свойствами и поведением – набором статических и динамических характеристик. Поведение объекта зависит от его состояния и внешних воздействий. Понятие объекта в программировании похоже на обыденный смысл этого слова. **Объект** – совокупность данных, характеризующих его состояние, и методов, моделирующих его поведение.

В объектно-ориентированном программировании (ООП) предметная область представляется как совокупность взаимодействующих объектов. Реализуется **событийная модель** взаимодействия: объекты обмениваются сообщения- ми и, реагируя на них, выполняют определенные действия.

Основные **принципы ООП**: абстрагирование, инкапсуляция, полиморфизм, наследование.

**Абстрагирование** – выделение существенных для данной задачи характеристик объекта и отбрасывание второстепенных. Любой программный объект – это абстракция. Детали реализации объекта, как правило, скрыты, они используются через его интерфейс – совокупность правил доступа.

**Инкапсуляция** – сокрытие деталей реализации. Позволяет представить программу в укрупненном виде и защитить от нежелательных вмешательств.

**Полиморфизм** – использование одного имени (методов, операций, объектов) для решения нескольких схожих задач или для обращения к объектам разного типа. Идея полиморфизма – «один интерфейс, множество методов». Возможны различные способы реализации полиморфизма: перегрузка методов, перегрузка операций, виртуальные методы, переопределение методов, параметризованные классы. Чаще всего понятие полиморфизма связывают с механизмом виртуальных методов.

**Наследование** – это процесс, посредством которого один объект может приобретать свойства другого. Для объекта можно определить потомков, которые наследуют, корректируют или дополняют его поведение. Наследование дает возможность многократного использования программного кода.

**Создание и использование конструкторов.**

**Класс** – обобщенное понятие, описывающее характеристики и поведение множества сходных объектов (называемых **экземплярами** или просто объекта- ми этого класса). В программе класс является пользовательским **типом данных** и представляет собой блок кода, в котором описывавается одна сущность, на- пример, модель реального объекта или процесса. Основными элементами класса являются данные и методы их обработки.

Заголовок описания класса обязательно содержит служебное слово **class** и **Имя**, которое по правилам языка C# начинается с заглавной буквы. В теле класса в фигурных скобках **{ ... }** описываются его элементы. Тело может быть пустым.

[ модификаторы ] **class Имя** [ : предки ]

**{ тело класса }**

Перед заголовком класса могут указываться **модификаторы** (modifier), которые определяют некоторые общие свойства класса, а также доступность для других элементов программы: **internal** – внутренний (задается по умолчанию, можно не писать); **public** – общедоступный; **private** – закрытый; **protected** – защищенный (закрыт для посторонних, но доступен для наследников); **static** – статический; **abstract** – абстрактный; **sealed** – запечатанный (или бесплодный, не может иметь наследников).

Класс может содержать следующие функциональные элементы *(members):*

* **поля** – переменные класса;
* **свойства** – обеспечивают «умный» доступ к полям;
* **методы** – определяют поведение класса;
* **конструкторы** – служат для инициализации объектов (экземпляров класса);
* **исключения** – используются для обработки исключительных ситуаций;
* набор **операций**, позволяющих производить различные действия. Простейший пример – описание общедоступного класса с одним методом:

*public class Computer*

***{***

*public void ShowInfo()*

***{***

*Console.WriteLine( "RAM = 8 Гбайт" );*

***}***

***}***

Все классы .NET имеют общего предка – класс **object**, и организованы

в единую иерархическую структуру. Классы логически группируются в **пространства имен**, которые служат для упорядочивания имен классов и предотвращения конфликтов имен: в разных пространствах имена могут совпадать. Пространства имен могут быть вложенными. Любая программа использует пространство имен **System**, в котором собрано множество стандартных классов и методов .

Рассмотрим теперь описание экземпляра класса. Напомним, что класс является обобщенным понятием, определяющим характеристики и поведение множества конкретных объектов, называемых **экземплярами** (или просто объектами) этого класса. Классы создаются программистом до выполнения программы. Экземпляры класса (объекты) создаются системой во время выполнения. Программист задает создание экземпляра класса с помощью инструкции **new**, например:

*Computer comp1 = new Computer(“Asus”, 2048)*

*или*

*Computer comp2 == new Computer(“IBM”, 4096).*

При этом для каждого объекта выделяется отдельная область памяти для хранения его данных. Класс может содержать также и статические элементы, которые существуют в единственном экземпляре.

Содержащиеся в классе данные могут быть переменными или константами. Описанные в классе переменные называются **полями класса**. При описании полей обязательно указывают **тип** и **имя** (которое начинается с малой буквы).

[ модификаторы ] **тип имя** [ = начальное\_значение ]

Можно также указывать модификаторы, определяющие доступность, а также задавать начальное значение (т. е. инициализировать поле).

Все поля в C# сначала автоматически инициализируются нулем соответствующего типа. Например, полям типа int присваивается 0, полям типа double – 0.0, а ссылкам на объекты – значение null). После этого полю присваивается значение, заданное при его явной инициализации.

**Метод** – именованный функциональный элемент класса, выполняющий вычисления и другие действия. Метод определяет поведение класса. В программе метод представляет собой блок кода, содержащий ряд инструкций, к которому можно обратиться по имени. Он описывается один раз, а вызываться может многократно по необходимости.

При описании метода в заголовке обязательно указывают его **тип** (который соответствует типу возвращаемого методом значения) и **Имя** (с заглавной буквы). В круглых скобках после имени указывают параметры метода, которых может и не быть, но скобки обязательны.

[ модификаторы ] **тип Имя(** [параметры] **)**

**{ тело метода }**

В теле метода в фигурных скобках **{ ... }** описываются выполняемые этим методом действия. Тело метода может быть пустым.

Доступность и другие характеристики метода задаются модификаторами (смысл которых будет раскрываться по мере выполнения работ): **public, private, protected, internal, abstract, virtual, override, new, static, sealed.** Методы класса имеют доступ к его полям непосредственно или через свойства (**set – get**). Поскольку поля хранят данные, а методы выполняют действия, для облегчения чтения и понимания кода программы рекомендуется поля называть существительными, а методы глаголами.

**Параметры метода** определяют множество значений аргументов, которые можно передавать в метод. Для каждого параметра обязательно задавать его **тип** и **имя**. Передаваемые в метод аргументы должны соответствовать объявленным параметрам по количеству, типам и порядку. Имя метода вместе с количеством и типами его параметров составляет **сигнатуру** метода. В сигнатуру не входит тип возвращаемого методом значения. Методы различают благодаря сигнатурам. В классе не должно быть методов с одинаковыми сигнатурами.

В языке С# возможны четыре типа параметров: параметры-значения, параметры-ссылки (ref), выходные параметры (out), параметры-массивы (params).

При вызове метода сначала выделяется память под его параметры. Каждому из параметров сопоставляется соответствующий аргумент. Проверяется соответствие типов аргументов и параметров, и производятся необходимые преобразования типов (или выдается сообщение о невозможности). После этого выполняются вычисления и/или другие действия (тело метода). В результате значение заявленного типа передается в точку вызова метода. Если методу задан тип **void**, он ничего не возвращает (т. е. является процедурой в терминологии Pascal). После выполнения метода управление передается на выражение, следующее после его вызова.

Методы реализуют функционал класса. Хорошо спроектированный метод должен решать только одну задачу, а не все сразу. Необходимо четко представлять, какие параметры должен получать метод, и какие результаты выдавать. Необходимо стремиться к максимальному сокращению области действия каждой переменной. Это упрощает отладку программы, поскольку ограничивает область поиска ошибки.

Заметим, что элементы (поля, методы), характеризующие класс в целом, следует описывать как **статические**. Статический метод (с модификатором *static*) может обращаться только к статическим полям класса. Статический метод вызывается через имя класса, а обычный – через имя экземпляра.

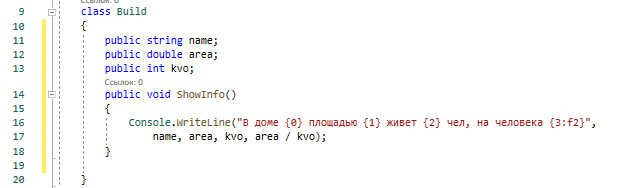
**Конструктор** – особый вид метода, предназначенный для инициализации объекта (конструктор экземпляра) или класса (статический конструктор). Конструктор объекта вызывается при создании экземпляра класса с помощью ключевого слова **new**. Имя конструктора **совпадает** с именем класса. Конструктор не имеет никакого типа, даже **void**.

Класс может иметь несколько конструкторов с разными параметрами для разных вариантов инициализации. Если не указано ни одного конструктора или некоторые поля не были инициализированы, полям значимых типов присваивается ноль (**0** или **0.0**), полям ссылочных типов – значение null. Конструктор, вызываемый без параметров, называется *конструктором по умолчанию*.

**Порядок выполнения работы:**

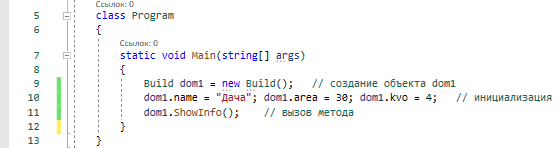
**Пример 1**

1. Создадим проект **сon5\_ФИО**.
2. В едином пространстве имен **namespace con5** с шаблоном класса **Program** создадим класс **Build** (проект строения). По умолчанию он имеет модификатор доступа **internal.**
3. В этом классе объявим три поля **name** (имя), **area** (площадь), **kvo** (количество жильцов) с модификаторами доступа **public**. Создадим метод **ShowInfo()**, который вычисляет площадь на одного жильца и выводит информацию о строении.



1. В методе **Main()** класса **Program** создадим объект **dom1** класса **Build,**

(т. е. построим дом по проекту **Build**), используя конструктор по умолчанию (без параметров). Зададим значения полей (параметры нашего дома). Вызовем метод **ShowInfo()**.



1. Протестируем программу, изменяя параметры объекта **dom1**. Заметим,

что при использовании конструктора по умолчанию значения полей задавать не совсем удобно.



1. Модифицируем программу. Создадим в классе **Build** собственный конструктор с тремя параметрами (**nm, ar, k**). Конструктор по умолчанию (без параметров) теперь тоже необходимо записать в явном виде.



Служебное слово **this** используют для указания на конкретный экземпляр объекта, а также для устранения неоднозначности между полями и локальными переменными или параметрами методов. Например, чтобы не плодить большое количество имен, полям (т. е. переменным класса) и используемым внутри метода локальным переменным (значения которых передаются через соответствующие параметры), обычно дают одинаковые имена. При этом конструктор с параметрами будет выглядеть так:

public Build(string *name*, double *area*, int *kvo*)

{ this.**name** = *name*; this.**area** = *area*; this.**kvo** = *kvo*; }

Служебное слово **this** указывает на поля класса **Build** (выделены полужирным), которым при создании объекта будут присвоены значения локальных переменных, переданные через одноименные параметры (выделены курсивом).

В дальнейшем мы тоже будем придерживаться такой практики задания имен.

1. В методе **Main()** класса **Program** создадим объект **dom2** класса **Build,** (т. е. построим новый дом по тому же проекту **Build**), используя теперь конструктор с параметрами. Снова вызовем метод **ShowInfo()**.



1. Протестируем программу, изменяя параметры объекта **dom2**. Заметим, что использование собственного конструктора гораздо удобнее.
2. Модифицируйте программу примера **1** (**сon5**), добавив поле **floor** (количество этажей). Создайте два объекта типа **Build** с разными параметрами и способами инициализации.

**Пример 2.**

Разработать класс для представления объекта «Прямоугольный параллелепипед». Реализуйте все необходимые поля данных (закрытые) и методы позволяющие:

– устанавливать и считывать значения полей данных;

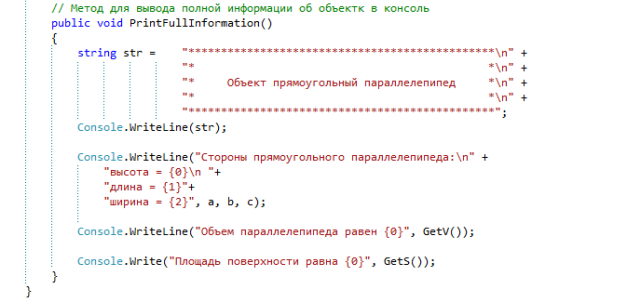
– вычислять объем прямоугольного параллелепипеда;

– вычислять площадь поверхности прямоугольного параллелепипеда;

– выводить полную информацию об объекте в консоль.

1. Реализуйте класс Параллелепипед со всеми необходимыми методами:Изображение выглядит как текст

   Автоматически созданное описание



1. Демонстрация использование объекта класса:



**Задание 3.** Создайте приложения, в которых определяются классы, поля, конструкторы, методы, создаются и инициализируются 2–3 объекта.

1. Класс **Student**. Метод **ShowInfo** выводит фамилию, имя, курс, возраст.
2. Класс **Computer.** Метод **Info** выводит модель (IBM, Asus, Sony) и параметры компьютера (объем ОЗУ и жесткого диска).
3. Класс **Tovar**. Метод **Kupi** выводит название (тетрадь, книга, ручка), цену, наличие на складе (есть, нет), количество.
4. Класс **Pogoda**. Метод **Show** выводит город (Минск, Брест, Гомель), температуру, осадки (ясно, пасмурно, гроза), направление и скорость ветра.
5. Kласс **Transport**. Метод **ShowInfo** выводит параметры транспортного средства: тип (автомобиль, мотоцикл, велосипед), цвет, скорость, масса.
6. Класс **Animal**. Метод **Golos** выводит вид (кошка, собака, попугай), имя

(Мурка, Шарик, Кеша), голос (мяу, гав, ррр).

1. Класс **Figura.** Метод **ShowArea** выводит название (квадрат, прямоугольник) и параметры фигуры (основание, высоту), вычисляет и выводит площадь.