Лекция № 7

I T E A

IT Education Academy

WWW.ITEA.UA

C# Base

Урок 7

Введение в ООП. Понятия классов и объектов

План урока

- Основы ООП
- Понятие класса и объекта
- Создание классов, их содержимое
- Создание объектов с помощью классов
- Модификаторы доступа
- Работа со свойствами
- Автоматически реализуемые свойства
- Конструкторы пользовательские и по умолчанию
- Частичные классы и методы
- Поля только для чтения

ООП - Объектно-ориентированное программирование

Объектно-ориентированное программирование (в дальнейшем ООП) — парадигма программирования, в которой основными концепциями являются понятия объектов и классов.

ООП при разработке дает возможность оперировать некоторыми сущностями, такими как классами и экземплярами класса – объектами.

У каждого объекта есть **свойства**. Например, свойства котенка: порода, имя, возраст, длина шерсти и т.д.

У каждого объекта есть **методы** (то есть действия, которые может делать объект). Например, методы котенка: спать(), кушать(), мурчать(), играть(), шкодить() и т.д.

Парадигмы ООП

Парадигма программирования — это совокупность идей и понятий, определяющих стиль написания компьютерных программ (подход к программированию).

К основным парадигмам ООП относятся:

Инкапсуляция - это свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе и скрыть детали реализации от пользователя.

Наследование - это свойство системы, позволяющее описать новый класс на основе уже существующего.

Полиморфизм - возможность объектов с одинаковой спецификацией иметь различную реализацию.

Абстракция – это придание объекту характеристик, которые четко определяют его концептуальные границы, отличая от всех других объектов. Позволяет работать с объектами, не вдаваясь в особенности их реализации.

Понятие класса и объекта

Класс — это логическая структура, позволяющая создавать свои собственные пользовательские типы.

Система типов NET

Хранение - **Стек**

Типы значений

Целочисленные типы
Типы с плавающей запятой
Тип decimal
Тип char
Перечисления enum
Структуры (struct)

Хранение - **Куча**

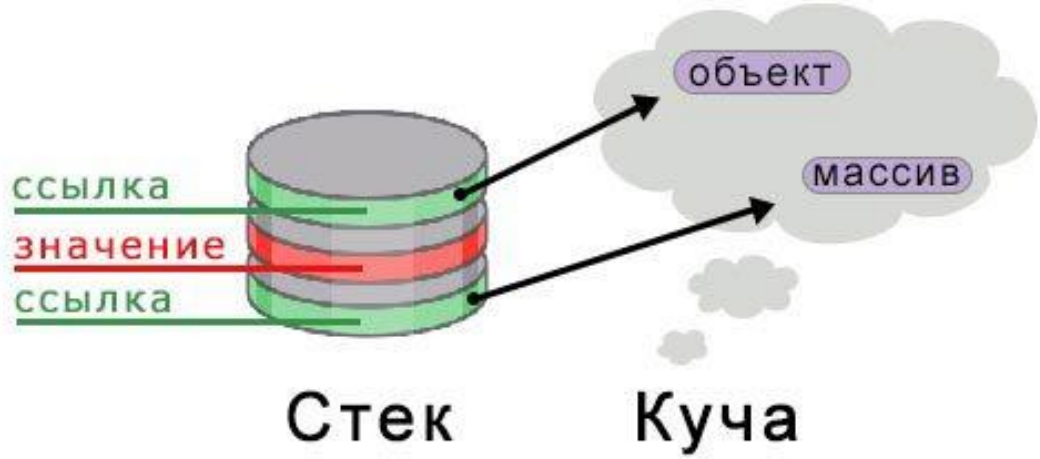
Ссылочные типы

Тип object
Тип string
Классы (class)
Интерфейсы (interface)
Делегаты (delegate)

Устройство памяти в .NET

Стек — это область оперативной памяти, которая создаётся для каждого потока. Он работает в порядке LIFO (Last In, First Out), то есть последний добавленный в стек кусок памяти будет первым в очереди на вывод из стека.

Куча — это хранилище памяти, также расположенное в ОЗУ, которое допускает динамическое выделение памяти и не работает по принципу стека: это просто склад для ваших переменных. Когда вы выделяете в куче участок памяти для хранения переменной, к ней можно обратиться не только в потоке, но и во всем приложении



Создание классов, их содержимое

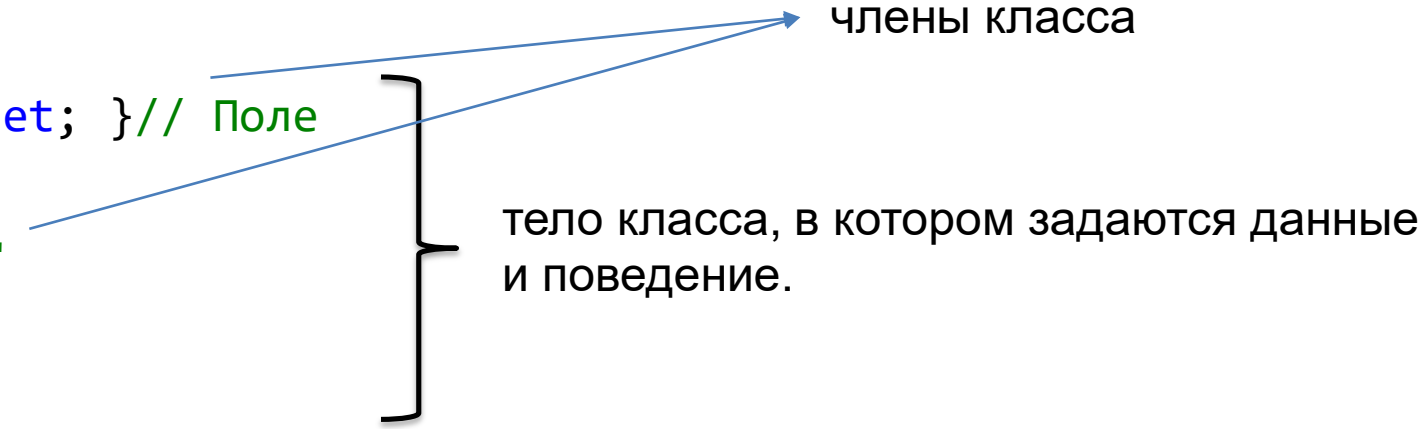
Класс — это логическая структура, позволяющая создавать свои собственные пользовательские типы.

По сути класс представляет новый тип, который определяется пользователем. Класс определяется с помощью ключевого слова **class**:

Ключевому слову **class** предшествует уровень доступа.

Класс может содержать в своем теле: поля, методы, свойства и события.

```
class MyClass
{
   public string field { get; set; }// Поле

   public void Method()// Метод
   {
      Console.WriteLine(field);
   }
}
```



Создание объектов с помощью классов

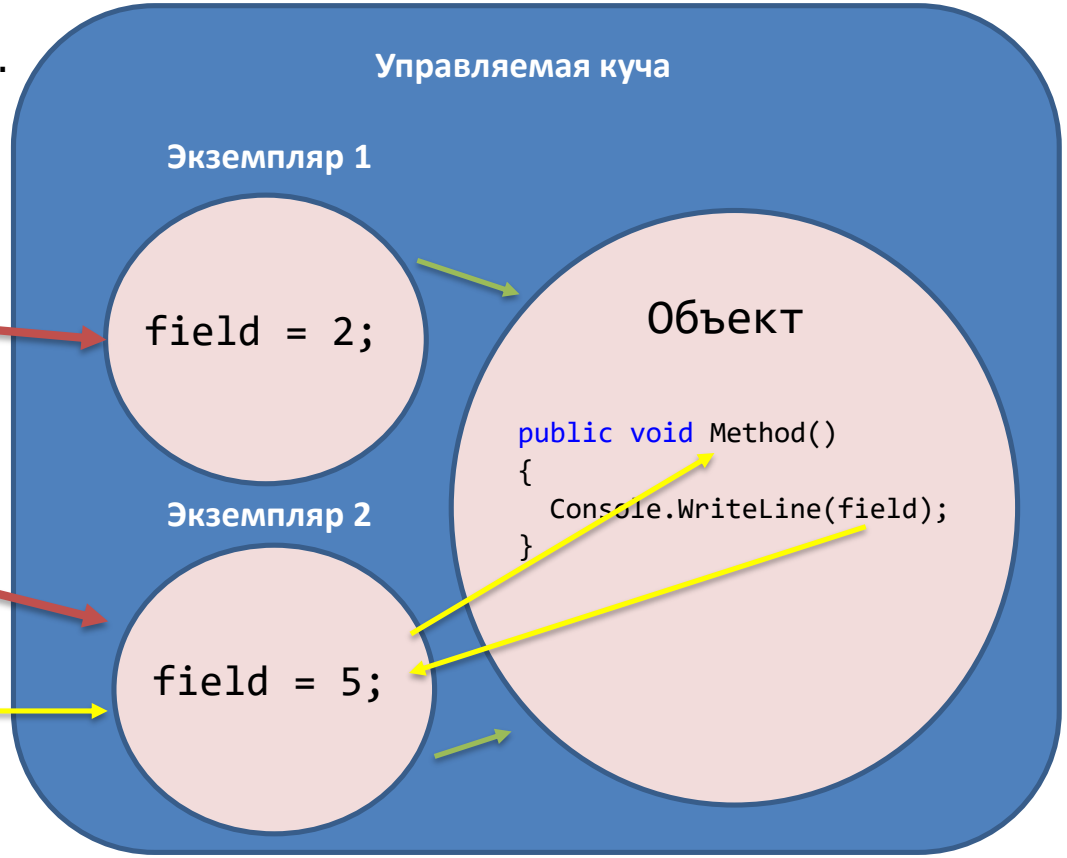
Класс и объект — это разные вещи. **Класс** определяет тип объекта, но не сам объект.

Объект — это конкретная сущность, основанная на классе и иногда называемая экземпляром класса.

Объекты содержат в себе статические поля и все методы.

Экземпляры содержат нестатические поля.

```
MyClass instance1 = new MyClass();
MyClass instance2 = new MyClass();

instance1.field = 2;
instance2.field = 5;

instance1.Method();
instance2.Method();
```



Модификаторы доступа

Все члены класса - поля, методы, свойства - все они имеют **модификаторы доступа.**

Модификаторы доступа позволяют задать допустимую область видимости для членов класса.

Модификаторы доступа определяют контекст, в котором можно употреблять данную переменную или метод.

В C# применяются следующие модификаторы доступа:

public: публичный, общедоступный класс или член класса. Такой член класса доступен из любого места в коде, а также из других программ и сборок.

private: закрытый класс или член класса. Представляет полную противоположность модификатору public. Такой закрытый класс или член класса доступен только из кода в том же классе или контексте.

Модификаторы доступа

Если для полей и методов не определен модификатор доступа, то по умолчанию для них применяется модификатор private.

Никогда не следует делать поля открытыми, это плохой стиль. Для обращения к полю, рекомендуется использовать методы доступа.

Члены, используемые только в классе, должны быть закрытыми.

Методы, получающие и устанавливающие значения закрытых данных, должны быть открытыми.

Если изменение члена приводит к последствиям, распространяющимся за пределы области действия самого члена, т.е. оказывает влияние на другие аспекты объекта, то этот член должен быть закрытым, а доступ к нему — контролируемым

Свойства

Кроме обычных методов в языке C# предусмотрены специальные методы доступа, которые называют свойства. Они обеспечивают простой доступ к полям классов и структур, узнать их значение или выполнить их установку.

```
int field;

public int Property
{
    get
    {
        return field;
    }
    set
    {
        field = value;
    }
}
```

Стандартное описание свойства имеет следующий синтаксис:

[модификатор_доступа] возвращаемый_тип произвольное_название {}

Свойство состоит из имени, типа и тела. В теле задаются методы доступа, через использование ключевых слов **set** и **get**.

Метод **set** автоматически срабатывает тогда, когда свойству пытаются присвоить значение. Это значение представлено ключевым словом **value**.

Метод **get** автоматически срабатывает тогда, когда мы пытаемся получить значение.

Свойства только для чтения и только для записи

Свойство только для чтения ReadOnly	Свойство только для записи WriteOnly
Метод доступа get – используется для получения значения из переменной.	Метод доступа set - используется для записи значения в переменную.

`int field;`
 	
`public int Property`
`{`
`  get`
`  {`
`  return field;`
`  }`
`}`
	`int field;`
	
	`public int Property`
	`{`
	`  set`
	`  {`
	`  field = value;`
	`  }`
	`}`

Автоматически реализуемые свойства

Начиная с версии C# 3.0, появилась возможность для реализации очень простых свойств, не прибегая к явному определению переменной, которой управляет свойство. Вместо этого базовую переменную для свойства автоматически предоставляет компилятор. Такое свойство называется автоматически реализуемым и принимает следующую общую форму:

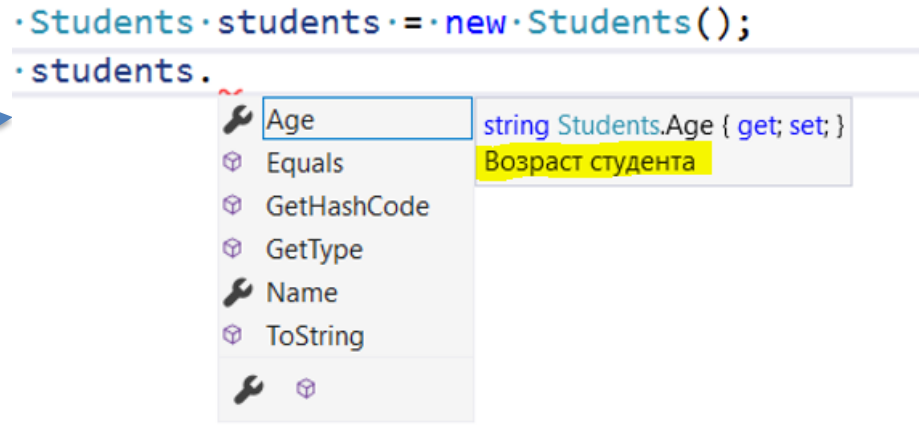
```
тип имя { get; set; }
```

После обозначений аксессора - get и мутатора - set сразу же следует точка с запятой, а тело отсутствует. Такой синтаксис предписывает компилятору создать автоматически переменную, иногда еще называемую поддерживающим полем, для хранения значения. Такая переменная недоступна непосредственно и не имеет имени. Но в то же время она может быть доступна через свойство.

Автоматически реализуемые свойства

Автоматически реализуемые свойства это более лаконичная форма свойств, их есть смысл использовать, когда в методах доступа get и set не требуется дополнительная логика.

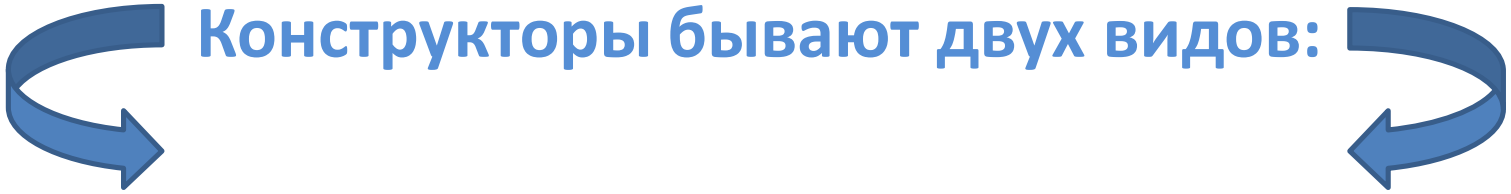
```
public class Students
{
    /// <summary>
    /// Имя студента
    /// </summary>
    public string Name { get; set; }

    /// <summary>
    /// Возраст студента
    /// </summary>
    public string Age { get; set; }
}
```



Отображения описания авто свойства.

Конструктор

Кроме обычных методов в классах используются также и специальные методы, которые называются **конструкторами**. Конструкторы вызываются при создании нового объекта данного класса. Конструкторы выполняют инициализацию объекта.



Конструкторы бывают двух видов:

Конструкторы по умолчанию

```
public MyClass()
{
}
```

Пользовательские конструкторы

```
public MyClass(int argument)
{
}
```

Конструкторы не имеют возвращаемых значений. Имя конструктора всегда совпадает с именем класса.

Если в теле класса не определен ни один пользовательский конструктор, то всегда используется «невидимый» конструктор по умолчанию.

Конструктор

Каждый раз, когда создается класс или структура, вызывается конструктор. Класс или структура может иметь несколько конструкторов, принимающих различные аргументы.

Задача конструктора по умолчанию – инициализация полей значениями по умолчанию.

Задача пользовательского конструктора – инициализация полей предопределенными пользователем значениями.

Если в классе имеется пользовательский конструктор, и при этом требуется создавать экземпляры класса с использованием конструктора по умолчанию, то конструктор по умолчанию должен быть определен в теле класса явно, иначе возникнет ошибка на уровне компиляции.

Конструкторы, которые вызывают другие конструкторы

Один конструктор может вызывать другой конструктор того же класса, если после сигнатуры вызывающего конструктора поставить ключевое слово this и указать набор параметров, который должен совпадать по количеству и типу с набором параметров вызываемого конструктора.

Вызывающий конструктор

```
public Person(string position, string name):
                this(name, 0)
{
   Position = position;
}
```

Вызываемый конструктор

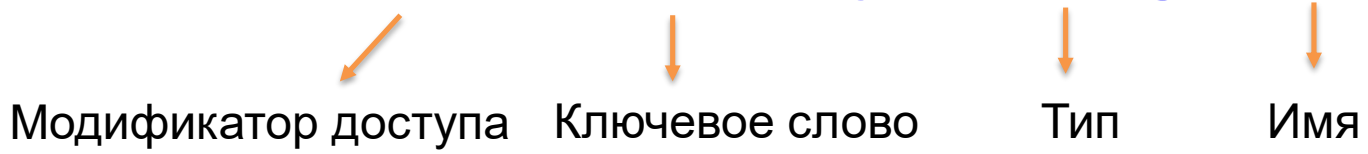
```
public Person(string name, int age)
{
   Name = name;
   Age = age;
}
```

При попытке вызова конструктора с несуществующим набором будет ошибка уровня компиляции.

Поля только для чтения

Поля для чтения можно инициализировать при их объявлении либо инициализировать и изменять в конструкторе. Инициализировать или изменять их значение в других местах нельзя, можно только считывать их значение.

```
public readonly string field = "Hello!";
```



readonly - это модификатор, который можно использовать только для полей.

Константы

Константы предназначены для описания таких значений, которые не должны изменяться в программе. Для определения констант используется ключевое слово const:

Константы характеризуются следующими признаками:

1. Константа должна быть проинициализирована при определении
2. После определения значение константы не может быть изменено

```
public const double PI = 3.141;
```

Модификатор доступа Ключевое слово Тип Имя

Сильные и слабые ссылки

Создание экземпляра класса по сильной ссылке	Создание экземпляра класса по слабой ссылке
`MyClass instance = new MyClass();` `instance.Method();`	`new MyClass().Method();`

Слабые ссылки (weak references) - это вспомогательный механизм обслуживания ссылок на управляемые объекты.

Частичные классы

В C# существует возможность создать **частичный класс** (интерфейс или структуру). То есть мы можем иметь несколько файлов с определением одного и того же класса, и при компиляции все эти определения будут скомпилированы в одно.

Для разделения класса на несколько частей, используется ключевое слово partial.

```
partial class PartialClass
{
   public void Method1(){}
}
```

```
partial class PartialClass
{
   public void Method2() {}
}
```

```
static void Main()
{
   PartialClass partialClass = new PartialClass();
   partialClass.Method1();
   partialClass.Method2();
}
```

Частичные методы

Частичные классы могут содержать **частичные методы**. Таким методы также определяются с ключевым словом partial. Причем определение частичного метода без тела метода находится в одном частичном классе, а реализация этого же метода - в другом частичном классе.

```
partial class PartialClass
{
   partial void Method();
}
```

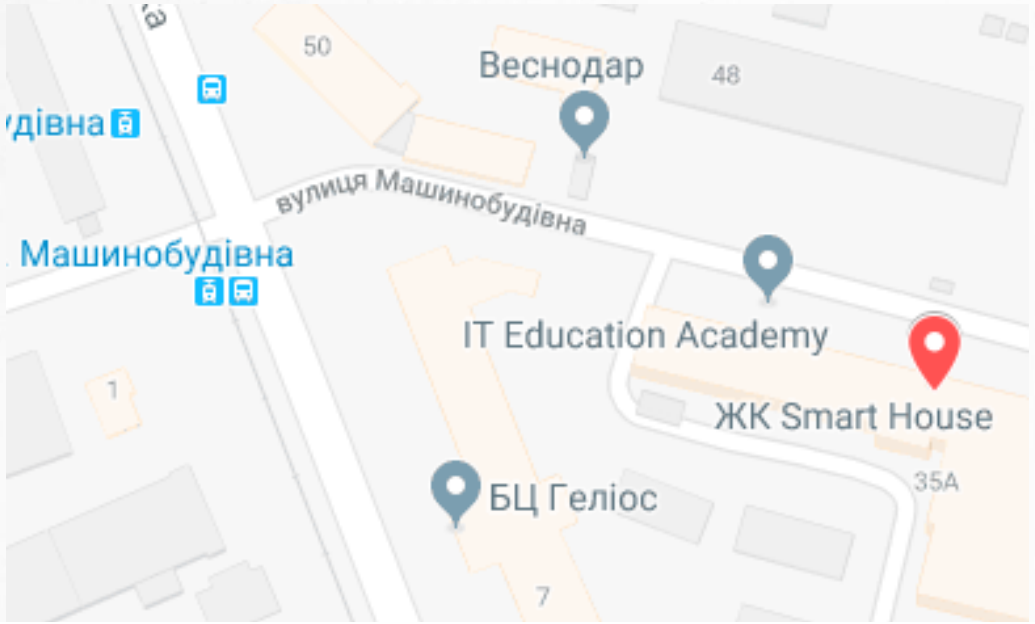
```
partial class PartialClass
{
   partial void Method(){/*Реализация*/}

   public void CallPartialMethod()
   {Method();}
}
```

```
static void Main()
{
   PartialClass partialClass = new PartialClass();
   partialClass.CallPartialMethod();
}
```

Правила использования частичных методов

1. Частичные методы должны быть определенны только в частичных классах.
2. Частичные методы должны быть помечены ключевым словом partial.
3. Частичные методы всегда являются private, попытка явного использования с ними модификатора доступа приведет к ошибке.
4. Частичные методы должны возвращать void.
5. Частичные методы могут быть нереализованными.
6. Частичные методы могут не иметь аргументов.


Презентация окончена.
Спасибо за внимание!



I T E A

КОНТАКТНЫЕ ДАННЫЕ

ITEA

ЖК “Smart House”, ул. Машиностроительная, 41 (м.Берестейская)

ЖК «Корона» улица Срибнокильская,1 м. Позняки

+38 (044) 599-01-79
facebook.com/Itea
info@itea.ua
itea.ua