ОБРАЗОВАТЕЛЬНЫЙ ЦЕНТР
ПРОГРАММИРОВАНИЯ И ВЫСОКИХ
ТЕХНОЛОГИЙ

2020г.

Введение в классы

AGENDA

- ✓ Синтаксис объявления класса
- ✓ Методы, объявление и вызов
- ✓ Использование параметров
- ✓ Модификаторы доступа
- ✓ Свойства и индексаторы
- ✓ Конструкторы
- ✓ Деструкторы
- ✓ Поля

Класс

Класс – новый тип данных, созданный и введенный пользователем, основной пользовательский тип в языке **C#**.

```
class <имя класса>
{
      [<элементы класса>]
}
```

Экземпляр класса этого типа данных называется объектом.

Класс представляет собой **ШАБЛОН**, по которому определяется форма объекта.
Чтобы использовать класс после объявления, необходима переменная - **объект**.

Объект объявляется как обычная переменная:
<имя класса> <имя объекта>;

Инициализацию объекта можно совместить с его объявлением:
<имя класса> <имя объекта> = new <имя класса>();

Доступ к элементам класса через объект осуществляется по синтаксису <имя объекта>.<имя члена>

Класс

Поля класса – обычные переменные уровня класса.

Константа – неизменяемая переменая уровня класса.

Методы описывают функциональность класса.

Свойства класса призваны предоставить защищенный доступ к полям.

Задача **конструктора** – начальная инициализация объекта или класса.

Финализатор автоматически вызывается сборщиком мусора и содержит завершающий код для объекта.

События представляют собой механизм рассылки уведомлений различным объектам.

Описание класса может содержать описание другого пользовательского типа – класса, структуры, перечисления, интерфейса, делегата. Обычно вложенные типы выполняют вспомогательные функции и явно вне основного типа не используются.

Методы, объявление и вызов

```
<тип> <имя метода>([<список параметров>])
{
      <тело метода>
}
```

Доступ к методам экземпляра класса выполняется через оператор «.»

Возможность выполнения определенной функциональности при различных исходных данных. Решают задачу повторного использования исходного кода.

Возвращают значения при помощи **return.**

Можно создавать много методов с одинаковым именем, но различным списком передаваемых параметров – **перезгрузка**.

```
string GetValue(string str)
{
      return str;
}
```

```
string GetValue(string str, int num)
{
```

Использование параметров

Параметры позволяют передать в метод некоторые входные данные.

Существует два способа передачи параметров в метод в языке **C#**: по **значению** и по **ссылке**. При передаче по значению метод получает не саму переменную, а ее копию. А при передаче параметра по ссылке метод получает адрес переменной в памяти.

Существуют слежующие вида параметров:

- Параметры, передаваемые по ссылке – используют модификатор **ref.**
- Выходные параметры – объявляются с модификатором **out.**
- Входные параметры – объявляются с модификатором **in.**

Язык C# не поддерживает использования ключевого слова in для входных параметров, все параметры, кроме ref и out параметров, являются in-параметрами по умолчанию.

- Параметры-списки – применяется модификатор **params**.

Необязательные параметры – при вызове метода необходимо предоставить значения для всех его параметров. Но **C#** также позволяет использовать необязательные параметры. Для таких параметров указывается значение по умолчанию. Следует учитывать, что после необязательных параметров все последующие параметры также должны быть необязательными.

Модификаторы доступа

Для поддержания принципа инкапсуляции элементы класса могут снабжаться специальными модификаторами доступа:

private – элемент с данным модификатором доступен только в том типе, в котором определен. Например, поле доступно только в содержащем его классе.

protected – элемент виден в типе, в котором определен, и в наследниках этого типа (даже если наследники расположены в других сборках). Данный модификатор может применяться только в типах, поддерживающих наследование, то есть в классах.

internal – элемент доступен без ограничений, но только в той сборке, где описан.

protected internal – комбинация модификаторов **protected** и **internal**. Элемент виден в содержащей его сборке без ограничений, а вне сборки – только в наследниках типа (шире, чем **protected**).

public – элемент доступен без ограничений как в той сборке, где описан, так и в других сборка, к которым подключается сборка с элементом.

Модификаторы доступа

Specifier	Same assembly			Other assembly	
	Declared class	other classes	Derived classes	other classes	Derived classes
Private	Yes	No	No	No	No
Public	Yes	Yes	Yes	Yes	Yes
Protected	Yes	No	Yes	No	Yes
Internal	Yes	Yes	Yes	No	No
Protected Internal	Yes	Yes	Yes	No	Yes
Private Protected (C# 7.2)	Yes	No	Yes	No	No

Свойства и индексаторы

Кроме обычных методов в языке **C#** предусмотрены специальные методы доступа, которые называют свойства.

get – получение значения.
set – установка значения.
Использование ключевого слова **value**.

```
// Автосвойства
class Person
{
    public string Name  { get;  set; }
}

// Сокращенная форма записи
private string name;
public string Name => name;
```

Свойства и индексаторы

Кроме обычных методов в языке **C#** предусмотрены специальные методы доступа, которые называют свойства.

get – получение значения.
set – установка значения.
Использование ключевого слова **value**.

```
// Развернутая запись свойства
class Person
{
    private string name;

    public string Name
    {
        get { return name; }
        set { name = value; }
    }
}
```

Свойства и индексаторы

Блоки **set** и **get** не обязательно одновременно должны присутствовать в свойстве. Если свойство определяют только блок **get**, то свойство доступно только для чтения, но не изменить. И, наоборот, если свойство имеет только блок **set**, тогда это свойство доступно только для записи - можно только установить значение, но нельзя получить значение.

Мы можем применять модификаторы доступа не только ко всему свойству, но и к отдельным блокам – либо **get**, либо **set**.

При использовании модификаторов в свойствах следует учитывать ряд ограничений:

- Модификатор для блока **set** или **get** можно установить, если свойство имеет оба блока (и **set**, и **get**).
- Только один блок **set** или **get** может иметь модификатор доступа, но не оба сразу.
- Модификатор доступа блока **set** или **get** должен быть более ограничивающим, чем модификатор доступа свойства.

Свойства и индексаторы

Позволяют индексировать объекты внутри класса и обращаться к данным по индексу.

<тип> this [<тип параметр>] { get { ... } set { ... } }

```
class Person
{
    public string Name { get; set; }
}

class People
{
    Person[] data;
    public People() { data = new Person[5]; }

    public Person this[int index] // индексатор
    {
        get { return data[index]; }
        set { data[index] = value; }
    }
}
```

Конструкторы

Особый метод, который вызывается при вызове оператора **new**. Имя данного метода совпадает с именем класса.

В случае явного отсутствия конструктора вызывается конструктор по умолчанию.

В классе может много конструкторов, но с разным списком передаваемых параметров.

Обязательно необходимо указывать модификатор доступа.

При вызове конструктора происходит инициализация переменных класса значениями по умолчанию:

- **bool – false;**
- ссылочный тип – **null;**
- численные типы – **0.**

```
<модификатор доступа> <имя класса>([<параметры>])
{
      <тело конструктора>
}
```

Деструкторы

Особый метод, который вызывается при уничтожении объекта.

При определении используется **~**.

Имя деструктора соответствует имени класса.

Деструктор не может иметь модификаторов доступа.

В деструктор вкладывается логика освобождения неуправляемых ресурсов.

```
~<имя класса>()
{
      <тело конструктора>
}
```

Константы, поля и readonly

Поля класса – это обычные переменные уровня класса.

Используется для хранения данных внутри объекта.

Может быть любого типа данных.

Константы – специальные поля предназначенные для описания таких значений, которые не должны изменяться в программе.

Для определения констант используется ключевое слово **const**.

Константа должна быть проинициализирована при определении.

После определения значение константы не может быть изменено.

Константа по умолчанию является статическим полем.

Константы, поля и readonly

Поля для чтения – это переменные уровня класса, значение которых нельзя изсменить после создания экзепляра класса.

Отличие от констант:

- Константы должны быть определены во время компиляции, а поля для чтения могут быть определены во время выполнения программы.
- Соответственно инициализировать константу можно устанновить только при ее определении.
- Поле для чтения можно инициализировать либо при его определении, либо в конструкторе класса.
- Константы не могут использовать модификатор **static**, так как уже неявно являются статическими. Поля для чтения могут быть как статическими, так и не статическими.

```
class SomeClass
{
        const double pi = 3.14; // константа
        string name = "None"; // поле
        readonly int age = 18; // поле для чтения
}
```

Область видимости переменных

Переменная имеет определенный контекст или область видимости.

Контекст класса – переменные, определенные на уровне класса, доступны в любом методе этого класса.

Контекст метода – переменные, определенные на уровне метода, являются локальными и доступны только в рамках данного метода. В других методах они недоступны.

Контекст блока кода – переменные, определенные на уровне блока кода, также являются локальными и доступны только в рамках данного блока. Вне своего блока кода они не доступны.

Использование this

Доступ ко внутренним переменным и методам внутри объекта.

```
public string Name;

   public MyClass(string Name)
   {
      this.Name = Name;
   }
```

Упорядочить вызов конструкторов при создании объекта.

```
   public MyClass()
   { }

   public MyClass(string name) : this()
   { }
```

Задание

1. Создайте ConsoleApplication, которое будет реализовывать работу с калькулятором.
Сам калькулятор реализуется типом Calculator.В нем присутствуют 4 метода для выполнения арифметических операций, с именами: Add, Sub, Multiply, Divide.
Каждый метод должен принимать два целочисленных аргумента и возвращать результат.
Пользователь вводит операнды и оператор через Console.

2. Создайте ConsoleApplication. Спроектируйте тип MyMagicClass, который будет реализовывать функционал:
а) определять каким является вводимое число -> + или -.
б) определять, на какие значения из перечисленных (2, 5, 3, 6, 9) делится без остатка на вводимое число.
в) передавать произвольный набор int и возвращать те, которые делаться на 3.