Темы урока

[**Цель занятия**](#_5gtd3bmzi15x) **2**

[**Классы, объекты**](#_la9357dn191h) **2**

[Как и когда мы используем классификацию в жизни?](#_lj4nyav9h6zr) 2

[Класс и объект на примере с кредитной картой](#_7iuz2zfxz76r) 3

[**Члены класса**](#_znkic6381v5) **4**

[**Инкапсуляция в ООП**](#_f53gjgxe0ayc) **4**

[**Уровни доступа к членам класса**](#_wgtf0r5gsx5b) **5**

[**Где определяется класс?**](#_tbo0845dani8) **6**

[**Пример класса Person: Поля и Свойства**](#_19633ikw52or) **6**

[Самостоятельная работа](#_gh0bf8jxs9h) 7

[Задание](#_1mjwm89aq71q) 7

[Решение](#_un811qddju91) 7

[**Пример создания экземпляра класса**](#_jgrh71ynk5xd) **7**

[Самостоятельная работа](#_pv57q3nbw4i4) 8

[Задание](#_2s0woy2nlb3s) 8

[Решение](#_cpvtgu8i79fb) 8

[**Инкапсуляция, read-only свойства**](#_mjxvtr6tvy92) **9**

[Самостоятельная работа](#_3t6tp5w92ugp) 9

[Задание](#_tsvxqx8bbo84) 9

[Решение](#_j8yhcb4vrkak) 10

[Самостоятельная работа (добавление логики в свойства)](#_vraxhipz6a71) 12

[Задание](#_ugr1dzso0xib) 12

[Решение](#_smrj25w3fu0u) 12

[Самостоятельная работа (добавление метода)](#_rzlt1kamwwh4) 13

[Задание](#_nyzm96derrie) 13

[Решение](#_wcsqquddmzjh) 13

[**Домашнее задание**](#_zgt5w27lwdgs) **14**

# 

# Цель занятия

Сегодня мы познакомимся с понятиями **класс** и **объект**.

Познакомимся какие у класса могут быть **члены**, а именно рассмотрим **поля**, **свойства** и **методы**. Ознакомимся с модификаторами доступа **public** и **private**.

# Классы, объекты

C# является полноценным **объектно-ориентированным языком**. Это значит, что программу на C# можно представить в виде взаимосвязанных взаимодействующих между собой объектов.

Значение термина **класс** в смысле ООП близко к значению этого термина в обычном — бытовом — смысле: **класс как группа предметов или явлений, обладающих общими признаками**. В ООП мы используем понятие **класс**, когда говорим о **шаблоне** объектов, с которыми работает программа. Такой шаблон позволяет нам объединить разрозненные переменные и действия над ними в структуру для более удобного использования.

Как и в обычной жизни, потребность в классификации возникает с какого-то порога сложности. Когда всё очень просто — классификация только всё усложняет.

## Как и когда мы используем классификацию в жизни?

Когда у вас есть одна единственная пара обуви, вам в принципе нечего классифицировать.

Начинаем последовательно усложнять эту ситуацию, у вас появляется пара летней и пара зимней обуви, и тут пока ещё всё нормально - обе пары стоят в прихожей, вы всегда можете легко обозреть и выбрать необходимую в зависимости от погоды. На этом этапе говорить о классификации можно, но она выглядит избыточно.

Однако, допустим, что вы шопоголик и у вас появилось много обуви. Пар 10–15. С этого момента вы начинаете оптимизировать пространство прихожей. На лето мы убираем зимнюю обувь, потому что она там не просто там бесполезна, но даже мешает — нам было бы трудно найти нужную пару, если в прихожей просто стоит “коробка с обувью” в которой ещё надо разобраться и найти парный комплект :) И вот в этот самый момент мы можем отследить, что мы, на самом деле, произвели пусть и простую, но, всё-таки, **классификацию** — у обуви появился **атрибут**: “подходящий сезон” **с возможными значениями** зима, лето, весна/осень. И часть обуви, которая не соответствует текущему сезону, мы убираем из прихожей.

На самом деле всё ещё сложнее: у нас есть резиновые сапоги, домашние или банные тапочки, деловые туфли, спортивная обувь, может быть даже ласты :)

Таким образом у нас в голове неявно появляются следующие умозаключения. Мы выделяем класс объектов “обувь”, к которым применяем следующие атрибуты:

* Подходящий сезон
* Цвет или тип расцветки (хотя бы категорийно “спокойный” или “весёленький”)
* Подходит ли для спорта?
* Подходит ли для дождливой погоды?
* Подходит ли вечернего наряда?

При этом обратите внимание, что **мы не пытаемся описать все характеристики наших объектов**! Только те, которые, которые важны нам для решения нашей конкретной задачи. Тем самым мы создаём упрощённую модель реального объекта.

## Класс и объект на примере с кредитной картой

Дальше можно всем вместе разобрать пример с кредитной картой — там уже есть слайды.

* Например, **класс** “кредитная карта” описывает основные свойства и действия, которые можно производить с кредиткой.
* **Объектом**, или **экземпляром класса**, принято называть конкретный предмет, в нашем примере с классом "кредитная карта" объектами могут выступать конкретные кредитки, которыми мы пользуемся.

Поскольку мы говорим о классе как о **модели** предметов реального мира, следует сказать, что для разных областей будут важны разные характеристики этих объектов.

* Пример **класса** "Кредитная карта":
  + Имя держателя
  + Номер
  + Срок действия
  + CVC-код
  + Физический размер
* **Объектом** же будет конкретная кредитка, имеющая конкретные значения указанных свойств:
  + Имя держателя : Andrei Goliakov
  + Номер : 2222 3333 4444 5555
  + Срок действиям : 10/23
  + CVC-код : 123
  + Физический размер : 85.6 (мм) × 53.98 (мм)

Можно задать вопросы:

— Из всех параметров выберите лишний? (скорее всего, все скажут “физический размер является лишним атрибутом”)

— Подумайте, почему вы выбрали именно этот параметр? (тут надо просто объяснить, что для проектирования банкомата физический размер будет важным атрибутом, а вот с точки зрения интернет-платежей такой атрибут, действительно, не видится полезным)

# Члены класса

Как мы выяснили, само понятие класса образуется вокруг некоторых группирующих свойств, которые могут отличаться значениями в конкретных случаях (у обуви это был целевой сезон, у кредитной карты - имя держателя, номер, и т.д.). Также к классу могут относиться какие-то действия (с его помощью или над ним - подготовить обувь к нужному сезону, изменить параметры карты).

Внутри класса определяются его члены:

* **поля** / **fields**: представляют собой данные, содержащиеся в классе, они служат для хранения информации об объекте,
* **свойства** / **properties**: представляют собой данные, совмещенные с реализацией доступа к этим данным, они также служат для хранения информации об объекте.
* **методы** / **methods**: функции, действия которые можно производить над или с помощью объекта.
* **конструкторы** / **constructors**: конструкторы вызываются при создании нового объекта данного класса. Конструкторы выполняют инициализацию объекта.
* существуют также другие, более специфичные члены, такие как **события**, **операторы**, **индексаторы**, и другие; мы рассмотрим некоторые из них на следующих уроках.

# Инкапсуляция в ООП

* **Инкапсуляция** (**encapsulation**) - это механизм, который **объединяет** данные и код, манипулирующий этими данными, а также **защищает** и то и другое от внешнего вмешательства или неправильного использования с помощью регулирования доступа к внутренним членам.
* Целью инкапсуляции является обеспечение **согласованности внутреннего состояния объекта**.
* Например, представим класс описывающий кошелек, средства с которого можно снять в различной валюте:

**класс** Кошелек

* **поле** Доступные средства в рублях : 6600
* **поле** Доступные средства в долларах : 100

Сейчас состояние объекта **согласовано**, так как количество денег в каждой из валют одинаково. Однако, в процессе работы с объектом такого класса **его легко ввести в неконсистентное состояние**, изменив доступную сумму только в одном месте, и не сделав этого во втором.

**Проблема** такого класса в том, что он **позволяет разработчику**, использующему его, **напрямую менять количество денег**, доступных в каждой из валют.

**класс** Кошелек

// поля лучше сделать закрытыми от разработчика

**закрытое (снаружи) поле** Доступные средства в рублях : 6600

**закрытое (снаружи) поле** Доступные средства в долларах : 100

// а вместо этого предоставить возможность менять количество

// денег согласованно

**доступный метод** Изменить количество средств в рублях(новая сумма)

Доступные средства в рублях = новая сумма

Доступные средства в долларах = новая сумма / 66

**доступный метод** Изменить количество средств в долларах(новая сумма)

Доступные средства в долларах = новая сумма

Доступные средства в рублях = новая сумма \* 66

# Уровни доступа к членам класса

При объявлении членов класса разработчик также определяет и уровень доступа к каждому из них, определяющие возможность их использования из другого кода в вашей или в других сборках.

**Существуют следующие модификаторы доступа:**

* public - Доступ к типу или члену возможен из любого другого кода в той же сборке или другой сборке, ссылающейся на него.
* private - Доступ к типу или члену возможен только из кода в том же классе (по умолчанию).

**Для начала важно запомнить эти два**!

Ниже приведены остальные пока, скорее, для информации, так как они относятся к изолированию членов класса для других сборок.

* protected - Доступ к типу или члену возможен только из кода в том же классе либо в классе, производном от этого класса.
* internal - Доступ к типу или члену возможен из любого кода в той же сборке, но не из другой сборки.
* protected internal - Доступ к типу или члену возможен из любого кода в той сборке, где он был объявлен, или из производного класса в другой сборке.
* private protected - Доступ к типу или члену возможен только из его объявляющей сборки из кода в том же классе либо в типе, производном от этого класса.

# 

# Где определяется класс?

Класс можно определять внутри пространства имен, вне пространства имен, внутри другого класса. Как правило, классы помещаются в отдельные файлы. Но в первом примере мы поместим новый класс в файле, где располагается класс **Program**. То есть файл **Program.cs** будет выглядеть следующим образом:

class Program

{

static void Main()

{

}

}

class Person

{

}

Открыть проект **L10\_C01\_class\_fields\_and\_properties\_demo**. В нём написать заготовку для класса Person.

# Пример класса Person: Поля и Свойства

Однако, говорить о классе имеет смысл только тогда, когда у него есть какие-то атрибуты, значения которых могут разниться в разных объектах этого класса. Их называют **членами** класса.

Давайте создадим простой класс описывающий какую-то персону. Допустим, для начала мы выделим следующие важные признаки: имя и возраст:

class Person

{

public string Name { get; set; }

public int Age { get; set; }

}

Разбираем пример на классе Person по презентации и демо-коду в проекте **L10\_C01\_class\_fields\_and\_properties\_demo** (стремимся к **L10\_C02\_class\_fields\_and\_properties\_final**).

**Заостряем внимание** на на том, что хорошим стилем будет

* писать явно модификатор доступа **private**
* придерживаться соглашения об именовании закрытых членов префиксом “нижнее подчёркивание” **\_**

## Самостоятельная работа

### Задание

Объявить класс домашнего питомца с именем Pet.

Определить в классе следующие закрытые поля:

* \_birthPlace: место рождения животного (страна, город)

Следующие общедоступные поля:

* Kind: вид животного (Mouse, Cat, Dog, и т.д.)
* Name: кличка питомца

Следующие общедоступные свойства:

* Sex: пол животного (одна латинская буква: M - мужской, F - женский)
* Age: возраст животного (в годах)

### Решение

См. проект **L10\_C02\_class\_fields\_and\_properties\_SW**.

public class Pet

{

public enum AnimalKind { Mouse, Cat, Dog }

private string \_birthPlace;

public AnimalKind Kind;

public string Name;

public char Sex { get; set; }

public byte Age { get; set; }

}

# Пример создания экземпляра класса

* Разбираем пример на классе Person по презентации и демо-коду в проекте **L10\_C01\_class\_fields\_and\_properties\_demo** (стремимся к **L10\_C02\_class\_fields\_and\_properties\_final**).
* **Заостряем внимание** на том, что
  + с объектом нельзя работать, если он не инициализирован с помощью ключевого слова **new**.
  + доступ к полям и свойствам осуществляется через **точку**.

## Самостоятельная работа

### Задание

В основном потоке программы создать экземпляр класса Pet с именем pet1 и заполнить его поля. Вывести на экран строку в формате:

$"{pet1.Name} is a {pet1.Kind} ({pet1.Sex}) of {pet1.Age} years old.")

Ниже создать еще один экземпляр класса Pet с именем pet2, задав ему значения полей при инициализации используя фигурные скобки. Вывести на экран такую же строку, только теперь уже для pet2:

$"{pet2.Name} is a {pet2.Kind} ({pet2.Sex}) of {pet2.Age} years old.")

### Решение

См. проект **L10\_C02\_class\_fields\_and\_properties\_SW**.

class Program

{

static void Main(string[] args)

{

Pet pet1 = new Pet();

pet1.Kind = Pet.AnimalKind.Cat;

pet1.Name = "Tom";

pet1.Sex = 'M';

pet1.Age = 8;

Console.WriteLine(

$"{pet1.Name} is a {pet1.Kind} " +

$"({pet1.Sex}) of {pet1.Age} years old.");

Pet pet2 = new Pet

{

Kind = Pet.AnimalKind.Mouse,

Name = "Minnie",

Sex = 'F',

Age = 1

};

Console.WriteLine(

$"{pet2.Name} is a {pet2.Kind} " +

$"({pet2.Sex}) of {pet2.Age} years old.");

}

}

# 

# Инкапсуляция, read-only свойства

Чтобы унифицировать формат вывода данных о нашей персоне, воспользуемся свойствами, доступными только для чтения – **read-only property** – у таких свойств есть описан только метод get (а метод set – отсутствует).

Разбираем по презентации read-only свойства.

Добавляем классу Person в демо-проекте **L10\_C04\_class\_readonly\_properties\_demo** read-only свойтсво PropertiesString (стремимся к **L10\_C05\_class\_readonly\_properties\_final**) :

class Person

{

...

public string PropertiesString

{

get { return $"Name: {Name}, Age: {Age}."; }

}

}

Затем в методе основного потока программы выводим на экран новое свойство объекта:

Console.WriteLine(p1.PropertiesString);

**Заостряем внимание**, что изменять его значение **ни изнутри, ни снаружи** нельзя!

## Самостоятельная работа

### Задание

Модифицируйте класс домашнего питомца с именем Pet таким образом, чтобы у него появилось read-only свойство Description, формирующее строку вывода информации о питомце на экран.

Модифицируйте соответственно основную программу, чтобы она использовала свойство Description для вывода информации о питомце.

### 

### Решение

См. проект **L10\_C06\_class\_readonly\_properties\_SW**.

public class Pet

{

public enum AnimalKind { Mouse, Cat, Dog }

private string \_birthPlace;

public AnimalKind Kind;

public string Name;

public char Sex { get; set; }

public byte Age { get; set; }

public string Description {

get { return $"{Name} is a {Kind} ({Sex}) of {Age} years old."; }

}

}

class Program

{

static void Main(string[] args)

{

Pet pet1 = new Pet();

pet1.Kind = "Cat";

pet1.Name = "Tom";

pet1.Sex = 'M';

pet1.Age = 8;

Console.WriteLine(pet1.Description);

Pet pet2 = new Pet

{

Kind = "Mouse",

Name = "Minnie",

Sex = 'F',

Age = 1

};

Console.WriteLine(pet2.Description);

}

}

## Самостоятельная работа (добавление логики в свойства)

### Задание

Добавьте логику проверки значения при установке пола: задавать можно заглавные или строчные буквы M или F. Обеспечьте вывод значения пола в верхнем регистре.

### Решение

См. проект **L10\_C07\_class\_adding\_logic\_to\_properties\_SW**.

private char \_sex;

public char Sex

{

get

{

return \_sex;

}

set

{

if (value == 'f' || value == 'F' || value == 'm' || value == 'M')

{

\_sex = char.ToUpper(value);

}

else

{

throw new Exception("Invalid value");

}

}

}

## Самостоятельная работа (добавление метода)

### Задание

Добавьте логику проверки значения при установке пола: задавать можно заглавные или строчные буквы M или F. Обеспечьте вывод значения пола в верхнем регистре.

### Решение

См. проект **L10\_C08\_class\_adding\_method\_SW**.

Класс Pet:

public string Description

{

get

{

return $"{Name} is a {Kind} ({Sex}) of {Age} years old" +

$" (birth place: {\_birthPlace}.";

}

}

public void SetBirthPlace(string birthPlace)

{

\_birthPlace = birthPlace;

}

Основной поток программы:

pet1.SetBirthPlace("Moscow");

pet2.SetBirthPlace("St.Petersburg");

# Домашнее задание

Написать консольное приложение, запрашивающее имя и возраст для трех человек. Затем программа должна вывести на экран информацию о людях и их возрастах через 4 года в следующем формате (схожая задача задавалась на третьем уроке):

Name: <name of the person # 1>, age in 4 years: <age of the person #1 in 4 years>

Name: <name of the person # 2>, age in 4 years: <age of the person #2 in 4 years>

Name: <name of the person # 3>, age in 4 years: <age of the person #3 in 4 years>

* Программа не должна закрываться пока не нажата любая клавиша.
* Необходимо выполнить задание с использованием массива объектов нового класса и циклом for!
* В классе должны быть определены следующие свойства:
  + Имя (обычное свойство),
  + Возраст (обычное свойство),
  + Возраст через четыре года (read-only свойство),
  + Итоговая строка вывода должна быть также реализована в классе Person в виде read-only свойства.а информации (read-only свойство).
* Пример работы программы (при вводе данных без ошибок):

> Enter name 0: Andrei

> Enter age 0: 36

> Enter name 1: Vasili

> Enter age 1: 32

> Enter name 2: Marina

> Enter age 2: 18

> Name: Andrei, age in 4 years: 40.

> Name: Vasili, age in 4 years: 36.

> Name: Marina, age in 4 years: 22.

> Press any key to continue . . .

* Не забывать после создания массива инициализировать каждый объект внутри перед началом использования с помощью ключевого слова **new**,
* Не забывать обрабатывать все предсказуемые исключения.