



ООП



otus.ru



Проверить, идет ли запись

Напишите «+» в чат, если меня слышно и видно



Тема урока

ООП



Нилов Павел

Преподаватель курса C# Professional, C# Basic

Контакты: [t.me/@NilovPavel](https://t.me/NilovPavel)

Правила вебинара



Активно
участвуем



Задаем вопрос
в чат



Вопросы вижу в чате,
могу ответить не сразу

Маршрут вебинара

1. Кратко об ООП

2. Инкапсуляция

3. Наследование

4. Решение задач

5. Ответы на вопросы

Цели вебинара



1. Узнать, что такое ООП.
2. Основные механизмы ООП.
3. Особенности синтаксиса ООП.

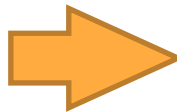
Кратко об ООП

Функциональное программирование

До ООП в разработке использовался другой подход — процедурный. Программа представляется в нем как набор процедур и функций — подпрограмм, которые выполняют определенный блок кода с нужными входящими данными.

Код ассемблера:

```
mov ax, 0x6h      ; заносим в AX число 6
mov cx, 0x8h      ; заносим в CX число 8
mov dx, cx        ; копируем CX в DX, DX = 8
add dx, ax        ; DX = DX + AX
```



Код C:

```
int main() {
    int a = 6;
    int b = 8;
    int sum = a + b;
    return 0;
}
```

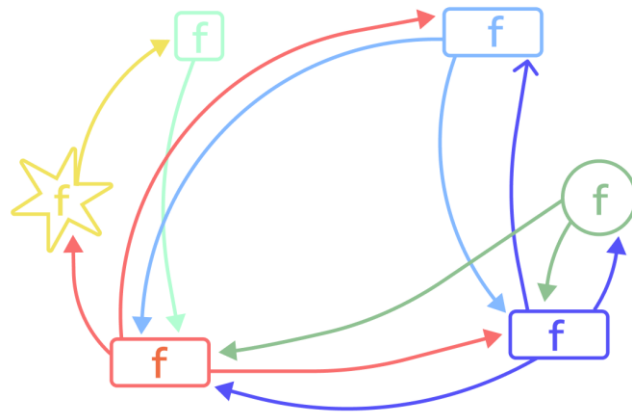


Функциональное программирование

Процедурное программирование хорошо подходит для легких программ без сложной структуры. Но если блоки кода большие, а функций сотни, придется редактировать каждую из них, продумывать новую логику. В результате может образоваться много плохо читаемого, перемешанного кода — «спагетти-кода» или «лапши».

Минусы использования функционального программирования **для крупных проектов**:

1. Сложно управлять кодом в виду того, что становится сложно декомпозировать код.
2. Отсутствовала инкапсуляция, в виду этого была нарушена связь между кодом и данными.
3. Необходимо было копировать код и, как следствие, его дублировать.



Что такое ООП?

Объектно-ориентированное программирование (ООП) — это подход, при котором программа рассматривается как набор объектов, взаимодействующих друг с другом. У каждого есть свойства и поведение.

ООП – это программирование с помощью классов и объектов.

Всё вокруг нас является
объектом.



У объекта есть
свойства
(еще называют параметрами)



У объекта есть
методы
(методы – это действия,
то есть что может
делать объект)

Например, машина – это объект.



У любой машины есть такие
свойства: модель, цвет, размер и т.д.

Методы машины:
затормозить ();
нажатьНаГаз ();
открытьДверь ();
закрытьДверь ();
и т.д.

Например, котёнок – это объект.



У любого котенка есть свойства:
порода, имя, цвет, длина шерсти,
возраст и т.д.

Методы котенка:
спать();
кушать();
играть ();
шкодить ();
и т.д.



Что из перечисленного ниже НЕ является принципом ООП?

Принципы ООП

Инкапсуляция

Наследование

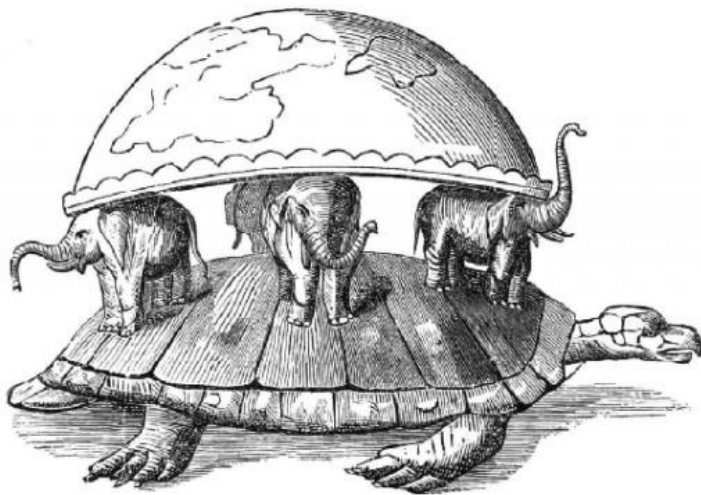
Абстракция

Полиморфизм

Принципы ООП

Объектно-ориентированное программирование основано на следующих принципах:

1. **Инкапсуляция**
2. **Наследование**
3. **Полиморфизм (типов)**
4. **Абстракция**

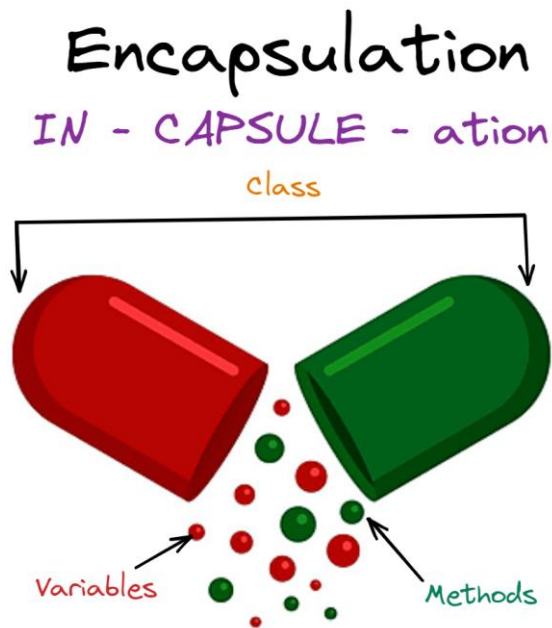


Инкапсуляция

Что такое инкапсуляция?

Инкапсуляция заключается в объединении данных и методов, которые с ними работают, внутри одного объекта.

При этом доступ к данным объекта ограничивается (через модификаторы доступа), чтобы скрыть внутреннюю реализацию и предоставить только необходимый интерфейс.



Содержимое класса

В C# класс может содержать:

1. Поля (Fields)
2. Свойства (Properties)
3. Методы (Methods)
4. Конструкторы (Constructors)



Класс:
программист

Объект:
разработчик Иван

Атрибуты:
зарплата, обязанности

Методы:
написание кода

```
class Coder
{
    private string name;
    private string salary;

    public string Name
    {
        get { return this.name; }
    }

    public void WriteCode()
    {
        ...
    }

    public Coder(string name)
    {
        this.name = name;
    }
}
```

Права доступа

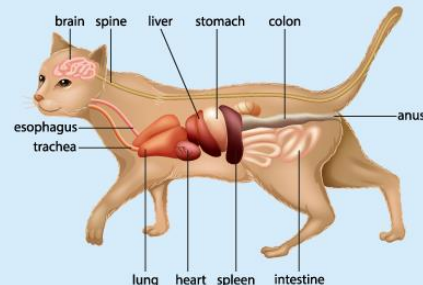
Права доступа к членам класса в программировании нужны для управления видимостью и доступом к данным и методам объекта. Они обеспечивают **инкапсуляцию**, один из ключевых принципов объектно-ориентированного программирования (ООП), помогая скрыть внутреннюю реализацию объекта и предоставляя только необходимый для работы интерфейс.

Модификатор	Описание
public	Член доступен из любого места, где виден объект. Член доступен только внутри класса, в котором он определён. Это значение по умолчанию.
private	Член доступен в классе и его наследниках.
protected	Член доступен внутри текущей сборки (assembly).
internal	

Понятно, что делать с объектом
(кормить, что же еще)



Не понятно, как именно
взаимодействовать с объектом
(слишком много деталей)



Модификаторы доступа на бытовых примерах

`public` предоставляет возможность миру взаимодействовать с созданным объектом.

Пример: завести автомобиль ключом зажигания. Для того, чтобы запустить двигатель, человек не должен залезать в двигатель, у него есть интерфейс (публичный метод запуска двигателя).



```
class Car
{
    public void RunEngine()
    { ... }
}
```

Модификаторы доступа на бытовых примерах

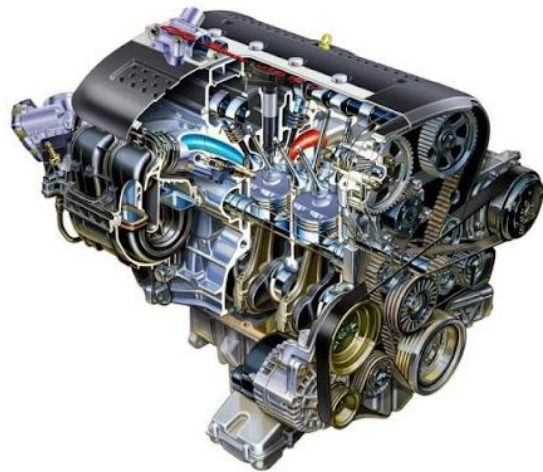
`private` не предоставляет миру возможность взаимодействовать с созданным объектом, а служит только для внутреннего использования.

Пример: двигатель автомобиля передает крутящий момент(`GetTwist()`). Все, что происходит внутри двигателя: подача топлива, подача воздуха и пр. – скрыто от мира. Все узлы, агрегаты, характеристики – тоже.

```
class Engine
{
    public void GetTwist()
    { ... }

    private void GetAir()
    { ... }

    private int angularVelocity;
}
```



Модификаторы доступа на бытовых примерах

`protected` не предоставляет миру возможность взаимодействовать с созданным объектом и служит только для использования в наследовании.

Пример: все (в любом случае большинство) автомобили состоят из одних и тех же элементов. Скажем, что у любого автомобиля есть такой показатель, как скорость, но у разных автомобилей она может быть разной.



```
class Car
{
    protected virtual int GetSpeed()
    { return 0; }
}

class SuperCar : Car
{
    protected override int GetSpeed()
    { return 200; }
}
```

Инкапсуляция дает следующую информацию

Итого, инкапсуляция определяет:

1. Как выглядит создаваемый объект во внешнем мире.
2. Какими характеристиками он обладает.
3. Какое поведение он реализует.

Решение задач

Задачи:

1. Опишите класс Student, который:

1.1 Содержит имя.

1.2 Предоставляет имя, через свойство.

1.3 Содержит возраст.

1.4 Предоставляет возраст через метод.

1.5 Содержит конструктор, принимающий имя студента и возраст.

1.6 Содержит поле, показывающее, является ли студент совершеннолетним.

1.7 При инициализации поля имени в конструкторе вычисляет, является ли студент совершеннолетним в приватном методе.

2. Опишите класс Dog

2.1 Содержит имя.

2.2 Содержит состояние: `enum { sit, lie, stand }`

2.3 Содержит метод, выполняющий команду "{Имя}, сидеть". Меняет состояние собаки на соответствующее.

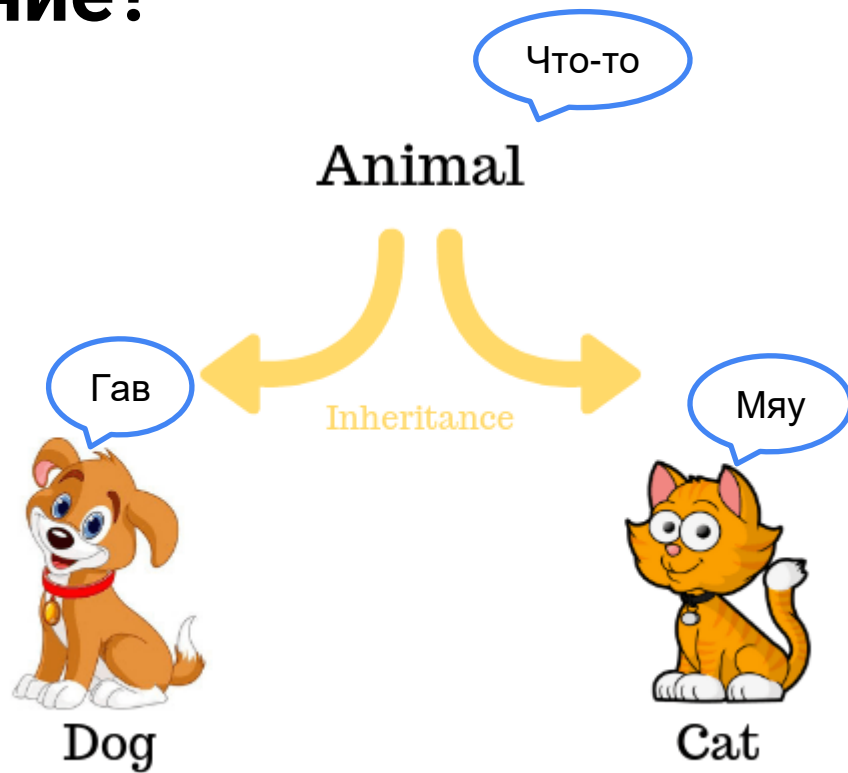
2.4 Содержит метод, выполняющий команду "{Имя}, лежать". Меняет состояние собаки на соответствующее.

Наследование

Что такое наследование?

Наследование – это механизм, который позволяет использовать возможности других классов.

Наследование позволяет определить дочерний класс, который использует (наследует), расширяет или изменяет возможности родительского класса. Класс, члены которого наследуются, называется *базовым классом*. Класс, который наследует члены базового класса, называется *производным классом*.



Синтаксис наследования

Наследоваться в рамках языка C# допустимо от:

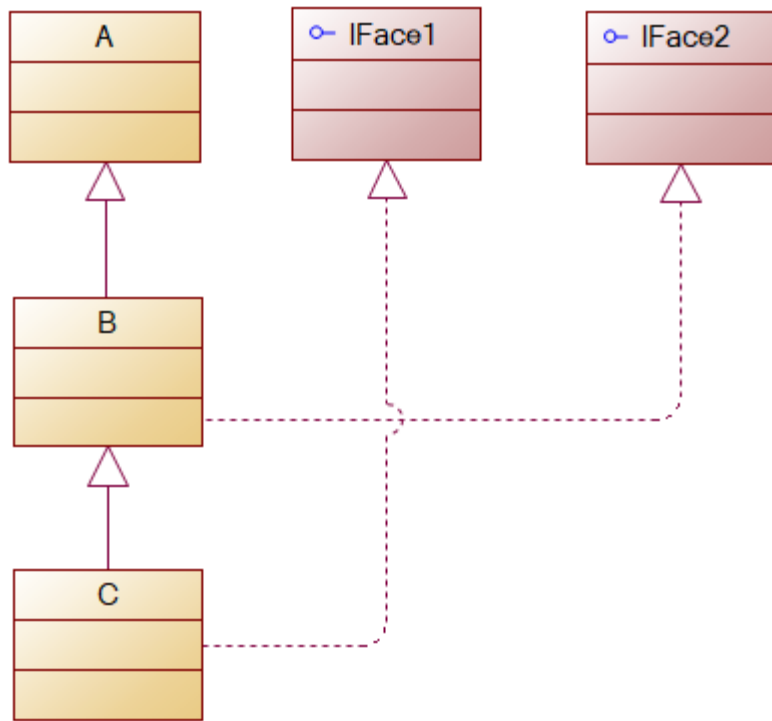
1. Классов.
2. Интерфейсов.

Наследование обозначается символом «:», следующего после имени класса с указанием всех типов, перечисленных через «,», от которых он наследуется.

Класс может наследоваться только от одного класса, и от любого количества интерфейсов.

```
class {className}
    : {abstract classNameParent}
    , {interfaceName1}
    ...
    , {interfaceNameN}
{
    //Реализация интерфейсных методов
}
```


Пример синтаксиса

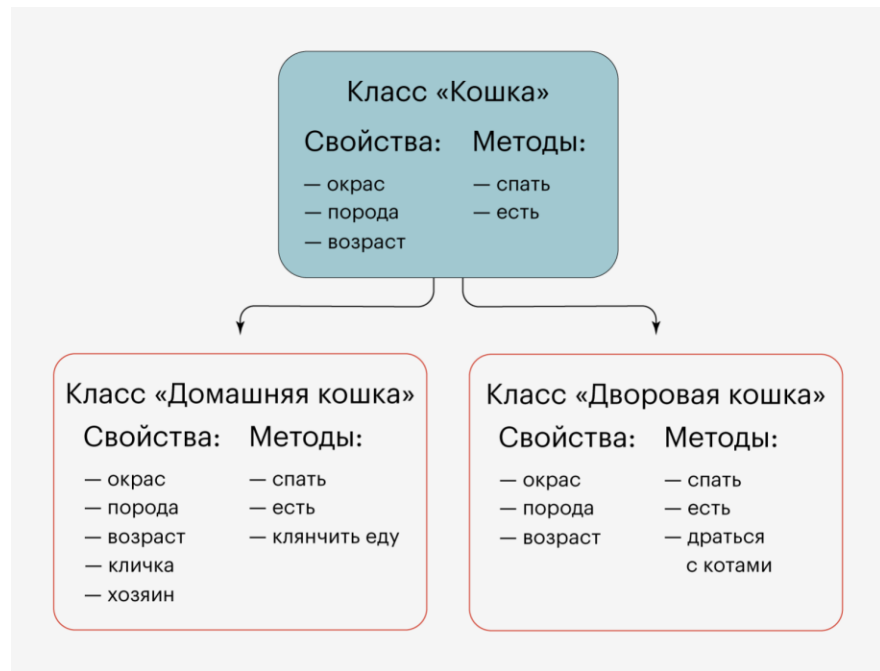


```
abstract class A
{
    //код класса
}
interface IFace1
{
    //код интерфейса
}
interface IFace2
{
    //код интерфейса
}
class B : A, IFace2
{
    //код класса
}
class C : B, IFace1
{
    //код класса
}
```

Что наследуется

В C#, при наследовании у родительского (базового) класса наследуются все **публичные** и **защищенные** (protected) члены класса, включая:

1. **Поля** (если они имеют protected или public модификаторы).
2. **Методы** (с protected или public модификаторами).
3. **Свойства** (с protected или public модификаторами).



Некоторые нюансы наследования

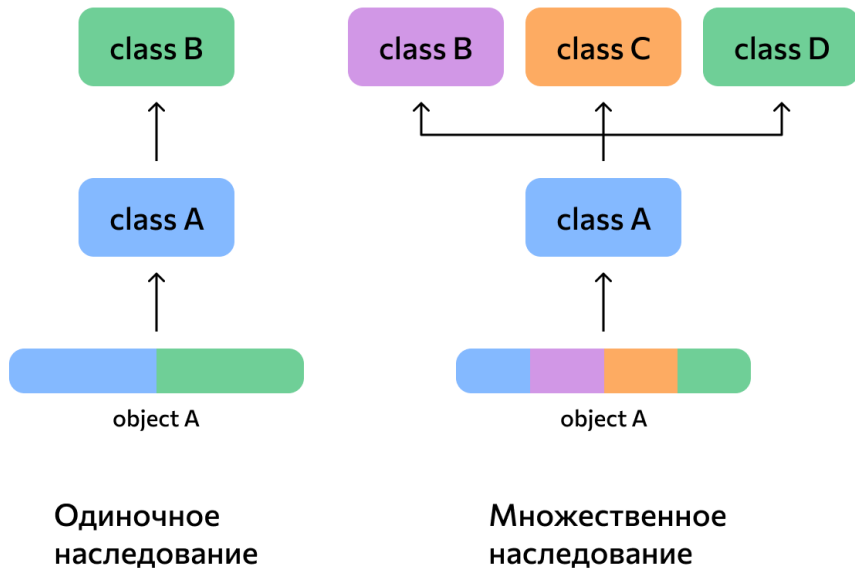
В рамках .Net классами не наследуются:

Классы и методы, помеченные ключевым словом `sealed`(запечатанный).

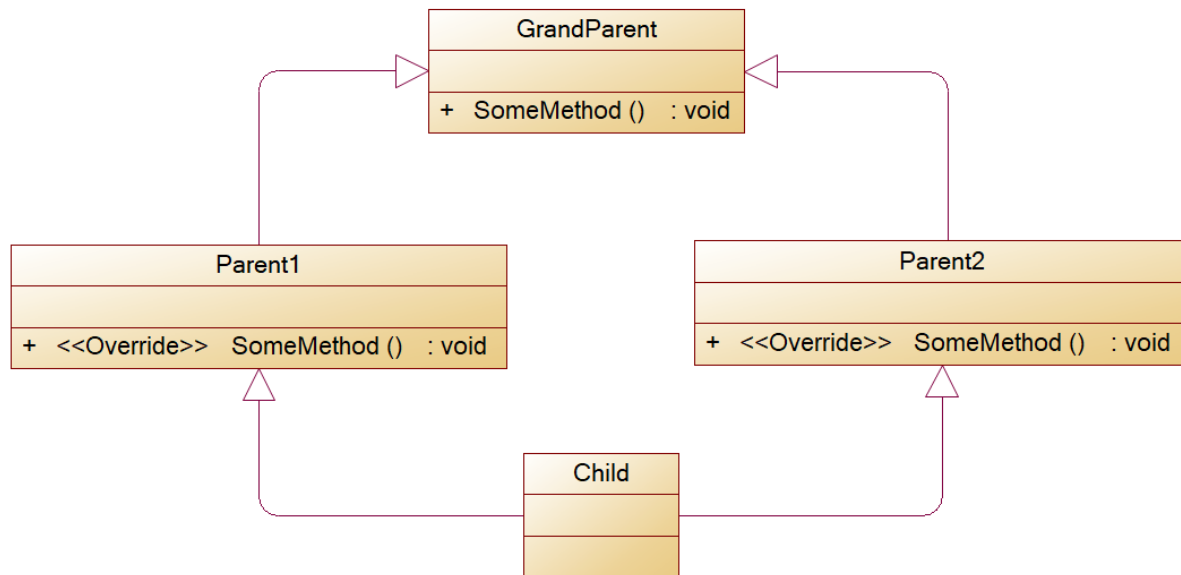
Также есть нюансы, о которых не стоит забывать

1. **Конструкторы.** Код конструктора класса-потомка будет воспроизводить код класса-предка. При этом сначала будет выполняться код класса-предка, потом – класса-потомка.
2. **Статические члены (`static`).** Статические члены класса будут унаследовать поведение класса-предка.
3. **Закрытые члены (`private`).** К приватным полям и методам класса напрямую обращаться нельзя. При этом допустимо использовать методы для работы с данными членами.

Множественное наследование запрещено



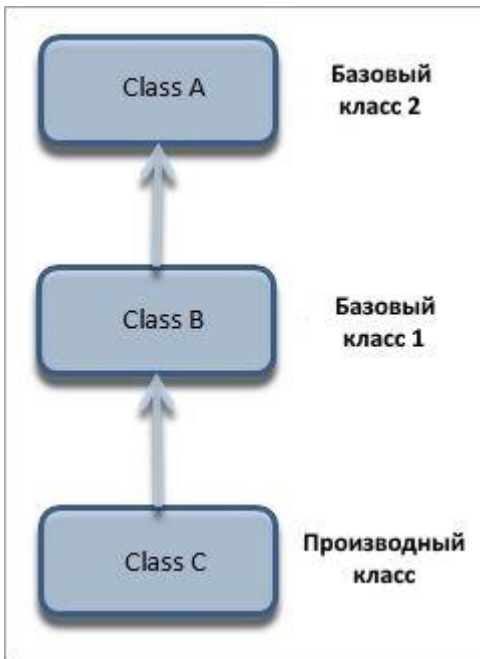
Почему в С# множественное наследование запрещено



Реализация какого класса метода **SomeMethod** достанется классу **Child**?

Транзитивное наследование

Транзитивное наследование позволяет определить иерархию наследования для набора типов. Другими словами, тип С может наследовать возможности типа В, который в свою очередь наследует от типа А. Благодаря транзитивности наследования члены типа А будут доступны для типа С.



Ключевое слово **base**

Ключевое слово base используется для доступа к членам базового класса из производного класса.

Используйте его, если вы хотите:

1. Вызвать метод базового класса.
2. Определить конструктор базового класса, который должен вызываться при создании экземпляров производного класса.

```
public class BaseClass
{
    protected void GetInfo() { }

    public BaseClass()
    { Console.WriteLine("in BaseClass()"); }
}

public class DerivedClass : BaseClass
{
    private void SomeMethod()
    { base.GetInfo(); }

    public DerivedClass() : base()
    { }
}
```

Ключевое слово **virtual**

Ключевое слово virtual может применяться для:

1. Методов
2. Свойств
3. Событий

virtual позволяет иметь реализацию в методе по умолчанию и переопределять ее в дочерних классах.

```
class Animal
{
    private string name = "Animal";

    public virtual string Name
    { get { return name; } set { this.name = value; } }

    public virtual void Speak()
    { Console.WriteLine("Animal makes a sound"); }
}

class Dog : Animal
{
    public override string Name { get; set; }

    public override void Speak()
    { Console.WriteLine("Dog barks"); }
}
```



Ключевое слово **override**

Ключевое слово [override](#) в C# используется для переопределения виртуальных методов и свойств объявленных в базовом классе.

Оно позволяет изменить поведение базового члена в производном классе, сохраняя полиморфизм.

```
class Animal
{
    public virtual void Speak()
    {
        Console.WriteLine("Animal makes a sound");
    }
}

class Dog : Animal
{
    public override void Speak()
    {
        Console.WriteLine("Dog barks");
    }
}
```

Вопросы, на которые отвечает наследование

Итого:

1. Какими свойствами должен обладать объект, созданный из класса-наследника.
2. Какое поведение будет реализовывать объект, созданный из класса наследника.

Решение задач

1 Описать класса-родителя и класс наследника

1.1 Описать класс Person

1.1.1 Персона содержит поле "имя".

1.1.2 Персона содержит не переопределяемый метод, который возвращает значение поля.

1.1.3 Персона содержит переопределяемый метод SayAboutYourSelf(), в котором сообщает свое имя. (virtual)

1.1.4 Персона содержит конструктор, который принимает 1 параметр - имя.

1.2 Описать класс Employee

1.2.1 Работник содержит поле "Компания".

1.2.2 Работник содержит переопределяемый метод DoWork().

1.2.3 Работник переопределяет метод SayAboutYourSelf(), в котором сообщает свое имя и говорит на каком предприятии работает. (override)

1.2.4 Работник содержит конструктор, который принимает имя и название компании как параметры. (использовать base)

Решение задач

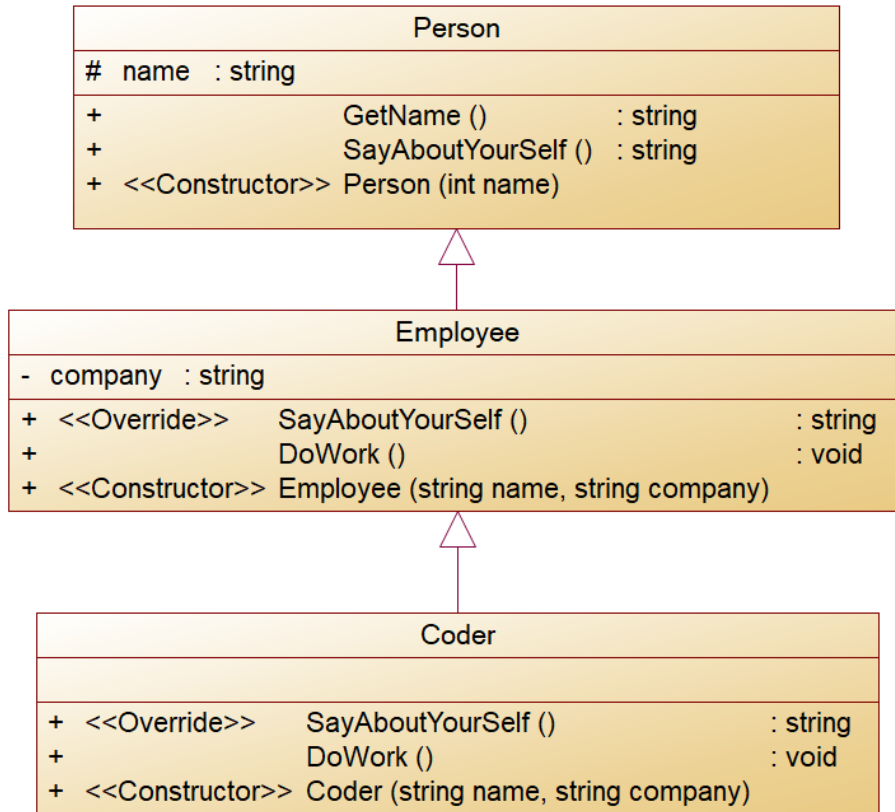
1.3 Описать класс Coder

1.3.1 Программист содержит

конструктор, переопределяемый
Employee с теми же параметрами
(base)

1.3.2 Программист пишет код в методе DoWork() (override)

1.3.3 Программист содержит переопределяемый метод SayAboutYourSelf(), в котором сообщает то же, что и работник + говорит на каком языке пишет.(override + base)



Вопросы?



Задаем
вопросы в чат



Ставим “-”,
если вопросов нет

