



Объектно-ориентированное программирование

Лекция 4. Классы и объекты. Поля класса, методы класса.

Определение класса в ООП

- ◆ ***Класс – это тип данных, задающий реализацию некоторой абстракции данных (сущности), характерной для моделируемой предметной области.***



Определение класса в ООП

- ◆ **Класс – это тип данных, задающий реализацию некоторой абстракции данных (сущности), характерной для моделируемой предметной области.**

- ◆ **моделируемая предметная область** – та часть реального мира, для которой создается программное обеспечение

- ◆ Банковская система
- ◆ Управление персоналом (отдел кадров)
- ◆ Управление реактором АЭС
- ◆ Разработка игровой стратегии



Определение класса в ООП

◆ ***Класс – это тип данных, задающий реализацию некоторой абстракции данных (сущности), характерной для моделируемой предметной области.***

◆ ***абстракция данных (сущность)*** – часть моделируемой предметной области, которую можно рассматривать как отдельный объект

- ◆ банковский счет (банковская система)
- ◆ температурный датчик (реактор АЭС)
- ◆ военный юнит (игровая стратегия)

И.М.Желакович БНТУ



Определение класса в ООП

- ◆ ***Класс – это тип данных, задающий реализацию некоторой абстракции данных (сущности), характерной для моделируемой предметной области.***

задающий реализацию – описывающий содержимое сущности, т. е. ее атрибуты (поля), действия (методы), реагирование сущности (события)



Определение класса в ООП

- ◆ ***Класс – это тип данных, задающий реализацию некоторой абстракции данных (сущности), характерной для моделируемой предметной области.***
- ◆ атрибуты – номер счета (банковский счет), текущая температура (датчик), уровень защиты (юнит)
- ◆ методы – закрытие счета, обновление значения тем-ры, перемещение юнита в др.точку
- ◆ события – сообщение о несанкционированном доступе (счет), сигнал о критическом значении тем-ры, ответная атака при нападении

Определение класса в ООП

- ◆ ***Класс – это тип данных, задающий реализацию некоторой абстракции данных (сущности), характерной для моделируемой предметной области.***
- ◆ ***тип данных*** – не содержит конкретных значений и не захватывает память под поля, это только описание, шаблон.

Определение класса в ООП

- ◆ ***Класс – это тип данных, задающий реализацию некоторой абстракции данных (сущности), характерной для моделируемой предметной области.***



Определение класса в ООП

- ◆ ***Класс – это модуль, архитектурная единица построения программной системы.***
- ◆ Модульность построения - основное свойство программных систем. В ООП программная система, строящаяся по модульному принципу, состоит из классов, являющихся основным видом модуля.

Проектирование в ООП

Объектно-ориентированная разработка программной системы основана на стиле, называемом проектированием от данных. Проектирование системы сводится к поиску абстракций данных,

Каждая из таких абстракций реализуется в виде класса, которые и становятся модулями - архитектурными единицами построения нашей системы. В основе класса лежит абстрактный тип данных.



Проектирование в ООП

- ◆ Спроектируем нашу аудиторию. Некоторые сущности выделить несложно – столы, стулья, доска, окна, дверь, студенты.

Некоторые сущности проявляются в процессе разработки – возможно, нам потребуются стены, пол и потолок, а для описания температуры, освещенности и концентрации CO_2 может выявиться сущность – КОМФОРТ. У каждой сущности выявляются атрибуты и



Проектирование в ООП

- ◆ Некоторые сущности могут включать в себя другие:
 - группа состоит из студентов
 - окна являются частью стены
 - светильники являются частью стены или потолка

Проектирование в ООП

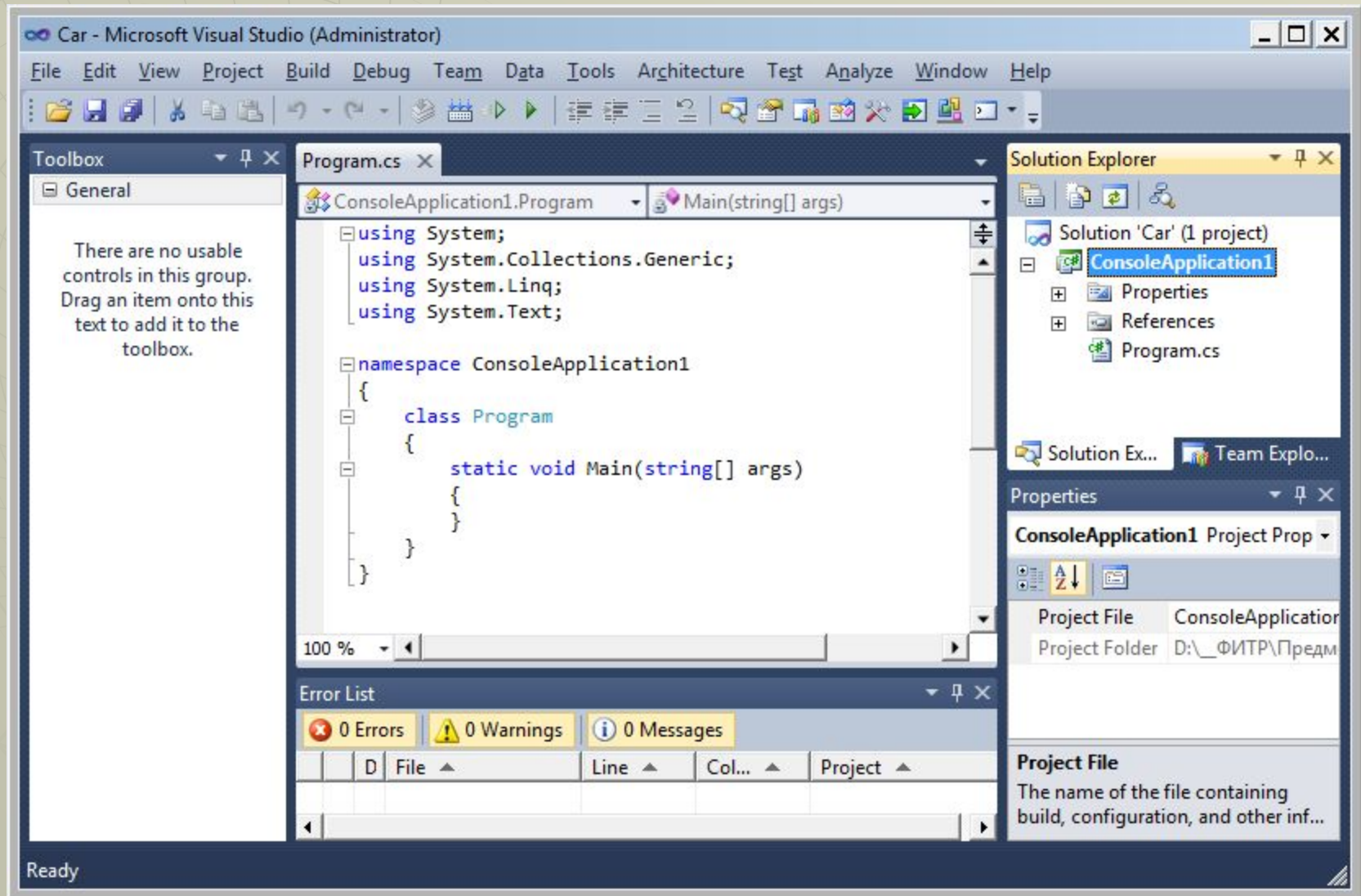
- ◆ В ООП используются понятия **клиент** и **сервер**. Сервер – тот, кто предоставляет услугу (в нашем случае – класс), клиент – тот, кто использует этот класс (программист, возможно тот же, кто писал класс, возможно – другой).



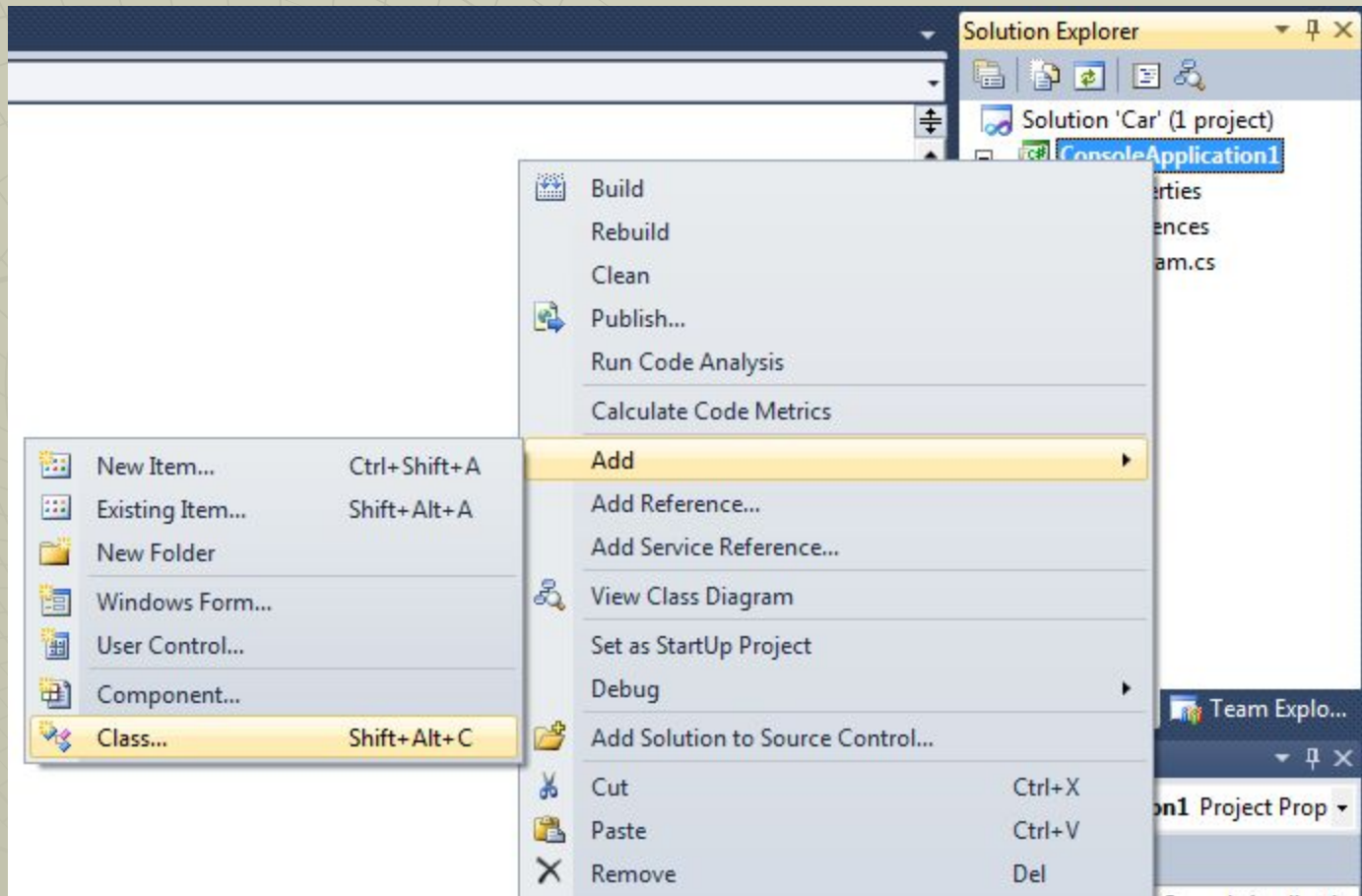
Класс CCar

- ◆ Рассмотрим создание и использование простейшего класса – автомобиль (Car).
- ◆ Для имен классов желательно использовать префикс – первую букву C (Class), чтобы не путать их в дальнейшем с объектами.
- ◆ Поэтому имя класса будет CCar

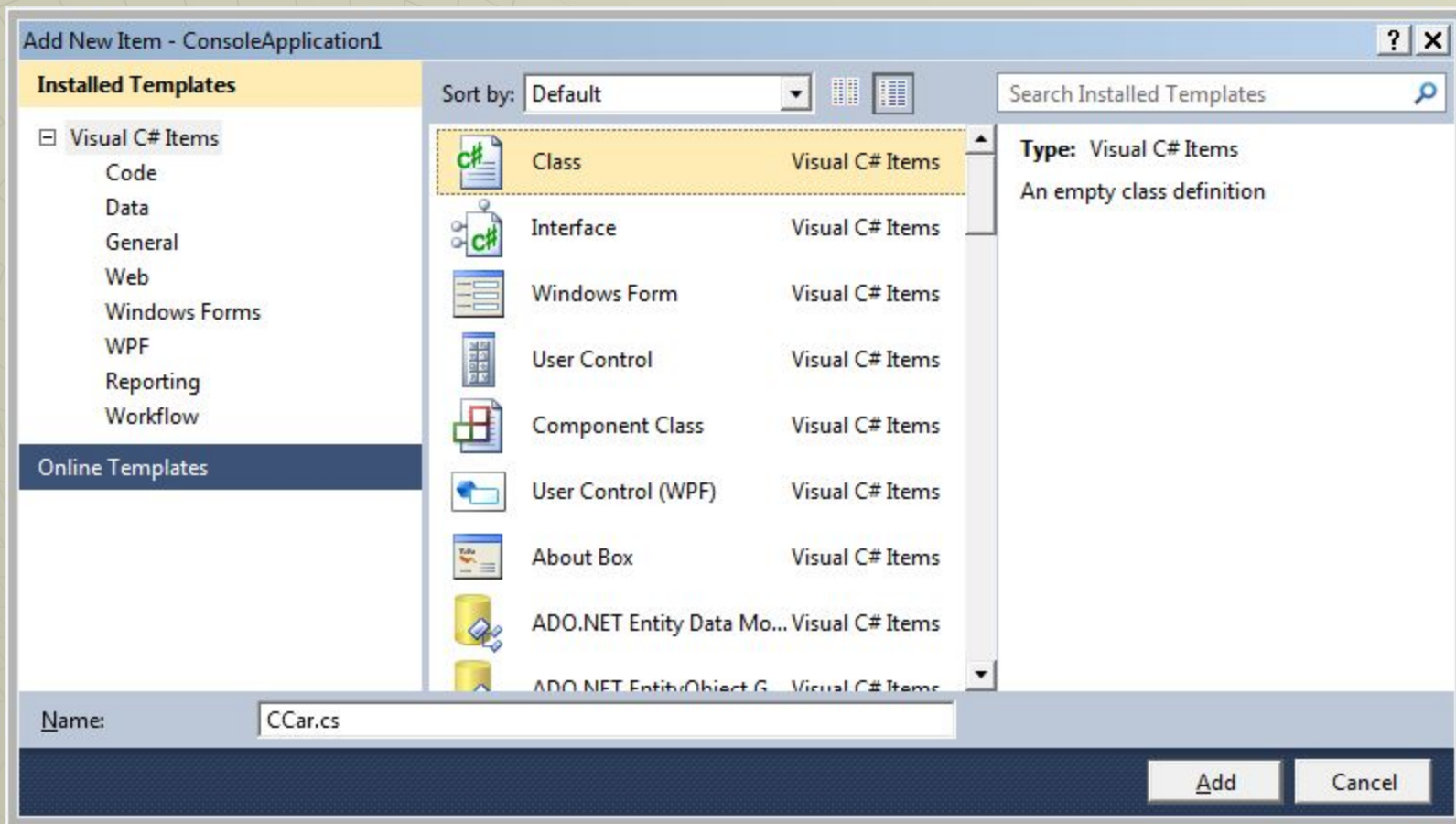
Создание нового проекта



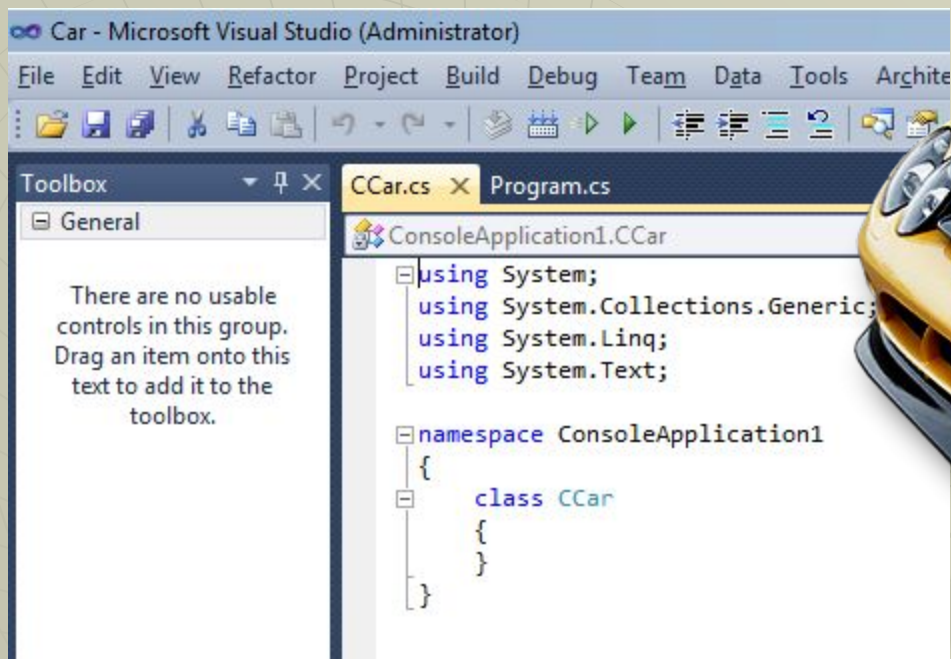
Добавление нового класса (правый клик на имени проекта)



Задание имени класса



Класс CCar



- ◆ Желательно каждый класс описывать в отдельном модуле
- ◆ Заготовка для класса создается автоматически

Класс CCar

```
namespace ConsoleApplication1
{
    class CCar
    {
        public string carName; // у авто есть имя
        public int currSpeed; // текущая скорость
    }
}
```

- ◆ у нашего класса пока только два атрибута (поля)
- ◆ public – поля будут видны во всем проекте (и в классе Program)

Класс CCar

```
class Program
{
    static void Main(string[] args)
    {
        CCar carPavel;
        carPavel = new CCar();

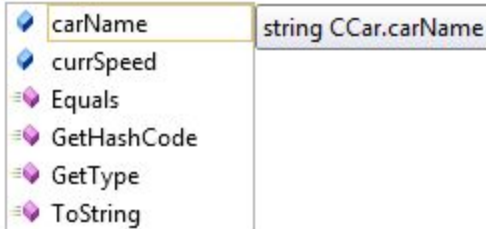
        |
    }
}
```

- ◆ Объекты создаются в том классе, где они используются – в Program



Класс CCar

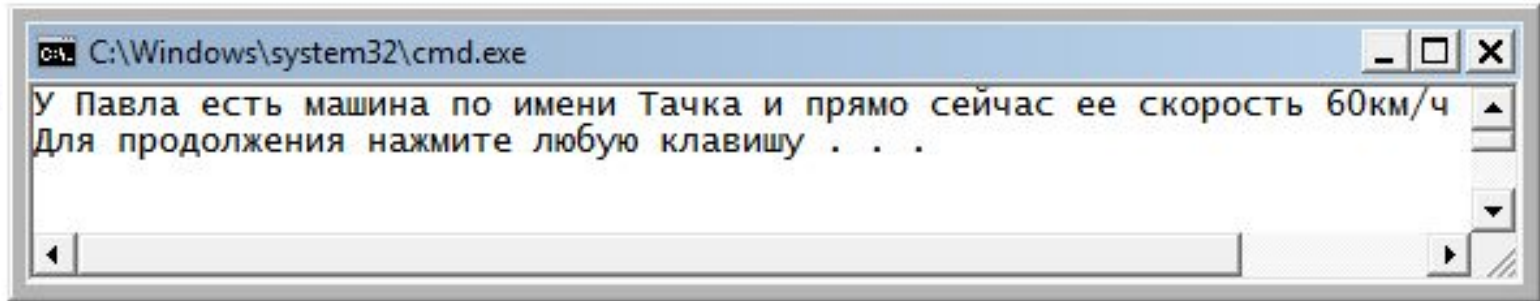
```
class Program
{
    static void Main(string[] args)
    {
        CCar carPavel;
        carPavel = new CCar();
        carPavel.|
    }
}
```



- ◆ при обращении к объекту видны все поля и методы, а также их типы

Класс CCar

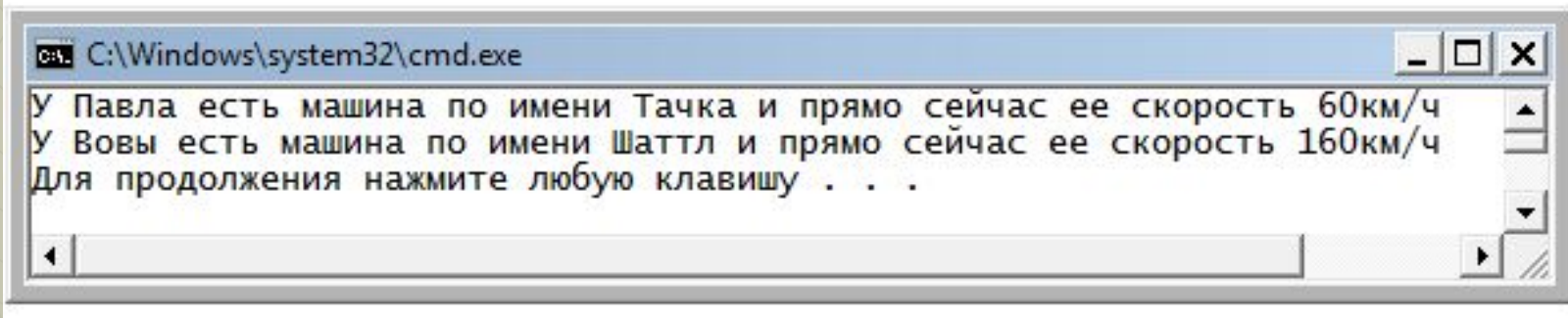
```
class Program
{
    static void Main(string[] args)
    {
        CCar carPavel;
        carPavel = new CCar();
        carPavel.carName = "Тачка";
        carPavel.currSpeed = 60;
        Console.WriteLine("У Павла есть машина по имени " + carPavel.carName +
            " и прямо сейчас ее скорость " + carPavel.currSpeed + "км/ч");
    }
}
```



- ◆ Мы можем использовать поля, чтобы присвоить им значения и чтобы получить их значения.

Класс CCar

```
CCar carVova = new CCar();  
carVova.carName = "Шаттл";  
carVova.currSpeed = 160;  
Console.WriteLine("У Вовы есть машина по имени " + carVova.carName +  
    " и прямо сейчас ее скорость " + carVova.currSpeed + "км/ч");
```



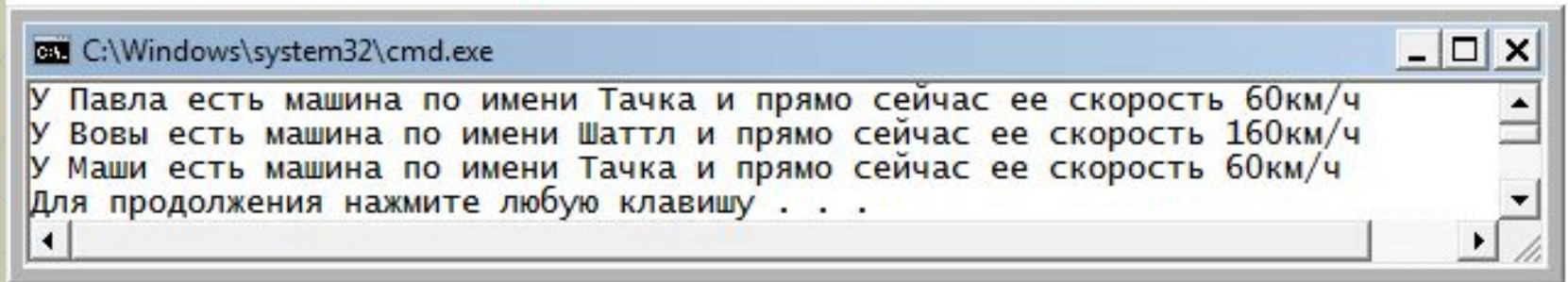
```
cmd.exe C:\Windows\system32\cmd.exe  
У Павла есть машина по имени Тачка и прямо сейчас ее скорость 60км/ч  
У Вовы есть машина по имени Шаттл и прямо сейчас ее скорость 160км/ч  
Для продолжения нажмите любую клавишу . . .
```

- ◆ Добавим еще один объект.
- ◆ Итого у нас есть 2 объ



Класс CCar

```
CCar carMasha = carPavel;  
Console.WriteLine("У Маши есть машина по имени " + carMasha.carName +  
    " и прямо сейчас ее скорость " + carMasha.currSpeed + "км/ч");
```



```
C:\Windows\system32\cmd.exe  
У Павла есть машина по имени Тачка и прямо сейчас ее скорость 60км/ч  
У Вовы есть машина по имени Шаттл и прямо сейчас ее скорость 160км/ч  
У Маши есть машина по имени Тачка и прямо сейчас ее скорость 60км/ч  
Для продолжения нажмите любую клавишу . . .
```

- ◆ И еще один.
- ◆ Сколько всего объектов у нас есть?

Класс СCar

```
public void Print()
{
    Console.WriteLine("Машина по имени " + carName +
        " движется со скоростью " + currSpeed + "км/ч");
}
```

- ◆ Для удобства добавим к нашему классу метод – вывод всей информации о полях класса

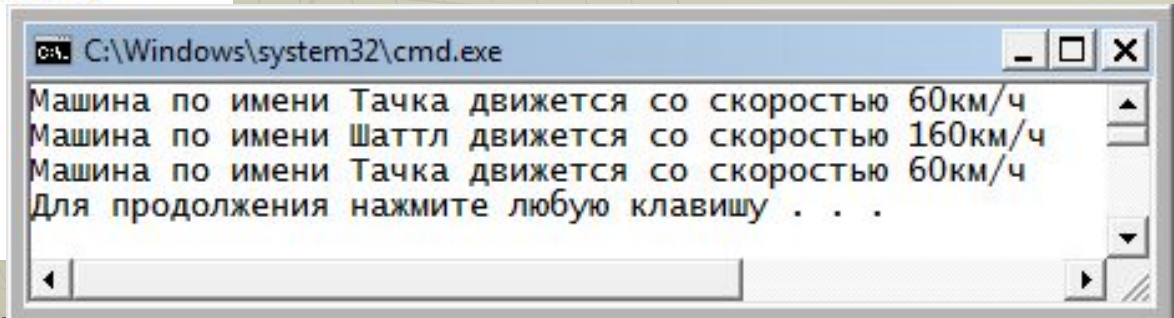
Класс CCar

- ◆ Теперь пользоваться нашим классом стало намного удобнее

```
class Program
{
    static void Main(string[] args)
    {
        CCar carPavel;
        carPavel = new CCar();
        carPavel.carName = "Тачка";
        carPavel.currSpeed = 60;
        carPavel.Print();

        CCar carVova = new CCar();
        carVova.carName = "Шаттл";
        carVova.currSpeed = 160;
        carVova.Print();

        CCar carMasha = carPavel;
        carMasha.Print();
    }
}
```



```
C:\Windows\system32\cmd.exe
Машина по имени Тачка движется со скоростью 60км/ч
Машина по имени Шаттл движется со скоростью 160км/ч
Машина по имени Тачка движется со скоростью 60км/ч
Для продолжения нажмите любую клавишу . . .
```

Класс СCar

```
public void SpeedUp(int up)
{ // метод наращивает скорость
  currSpeed += up;
}
|
}
```

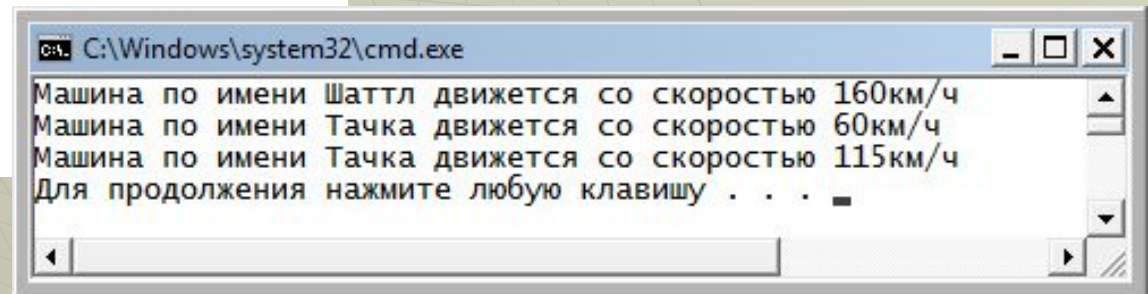
- ◆ Добавим метод посложнее

Класс СCar

```
class Program
{
    static void Main(string[] args)
    {
        .....

        for (int i = 1; i <= 10; i++ )
            carPavel.SpeedUp(i);
        carPavel.Print();
    }
}
```

- ◆ Разгоним один автомобиль

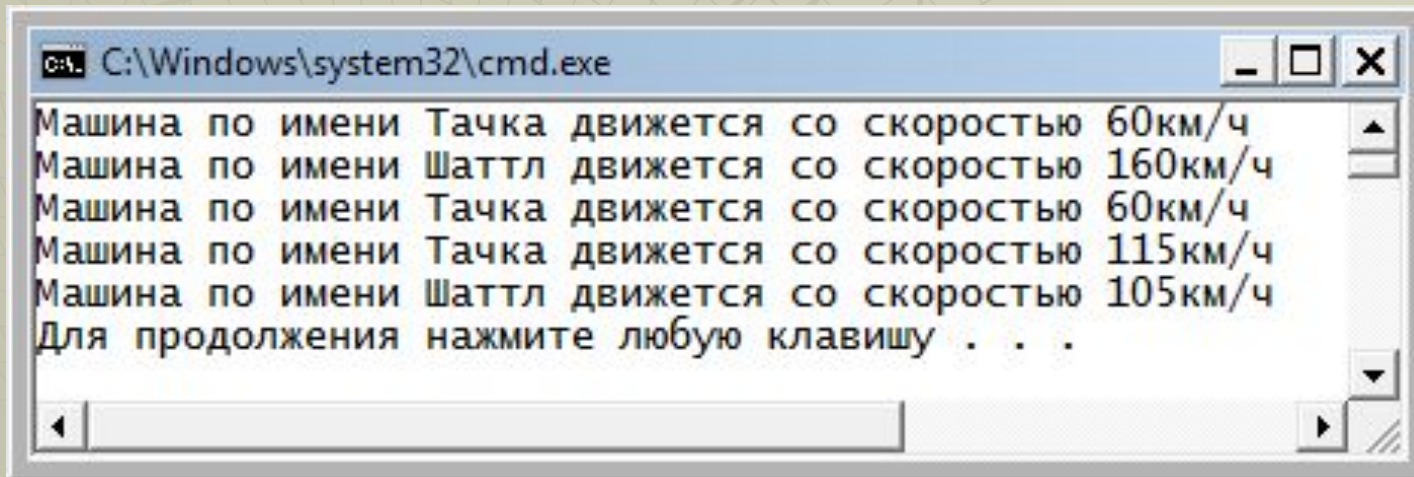


```
C:\Windows\system32\cmd.exe
Машина по имени Шаттл движется со скоростью 160км/ч
Машина по имени Тачка движется со скоростью 60км/ч
Машина по имени Тачка движется со скоростью 115км/ч
Для продолжения нажмите любую клавишу . . .
```


Класс СCar

```
for (int i = 1; i <= 10; i++)  
    carVova.SpeedUp(-i);  
carVova.Print();
```

- И притормозим другой



```
C:\Windows\system32\cmd.exe  
Машина по имени Тачка движется со скоростью 60км/ч  
Машина по имени Шаттл движется со скоростью 160км/ч  
Машина по имени Тачка движется со скоростью 60км/ч  
Машина по имени Тачка движется со скоростью 115км/ч  
Машина по имени Шаттл движется со скоростью 105км/ч  
Для продолжения нажмите любую клавишу . . .
```

Класс CCar

```
class CCar
{

    public int m_s()
    { // метод возвращает скорость в м/с
      return currSpeed * 1000 / 3600;
    }
}
```

- ◆ Еще один вариант метода – с возвращаемым значением

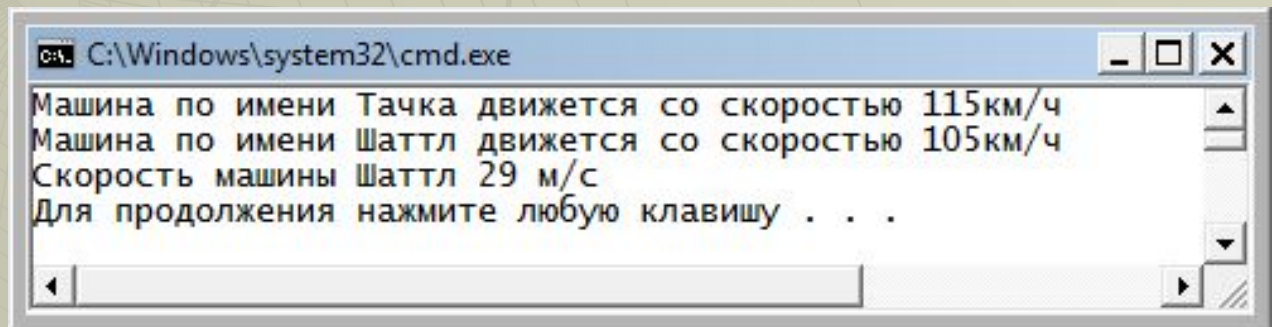
Класс CCar

```
class Program
{
    static void Main(string[] args)
    {
        .....

        for (int i = 1; i <= 10; i++)
            carVova.SpeedUp(-i);|
        carVova.Print();

        Console.WriteLine("Скорость машины " + carVova.carName + " " + carVova.m_s() + " м/с");
    }
}
```

◆ И его использование



```
C:\Windows\system32\cmd.exe
Машина по имени Тачка движется со скоростью 115км/ч
Машина по имени Шаттл движется со скоростью 105км/ч
Скорость машины Шаттл 29 м/с
Для продолжения нажмите любую клавишу . . .
```

Класс CCar



- ◆ Насколько один автомобиль едет быстрее, чем другой?
- ◆ Метод работает с двумя объектами.

Класс CCar

- ◆ Здесь важно продумать, от какого объекта будет вызван метод (первый автомобиль) и какой объект будет подан в качестве аргумента (автомобиль, с которым сравниваем).
- ◆ Вызов будет выглядеть так:
`car1.faster(car2)`

Класс CCar

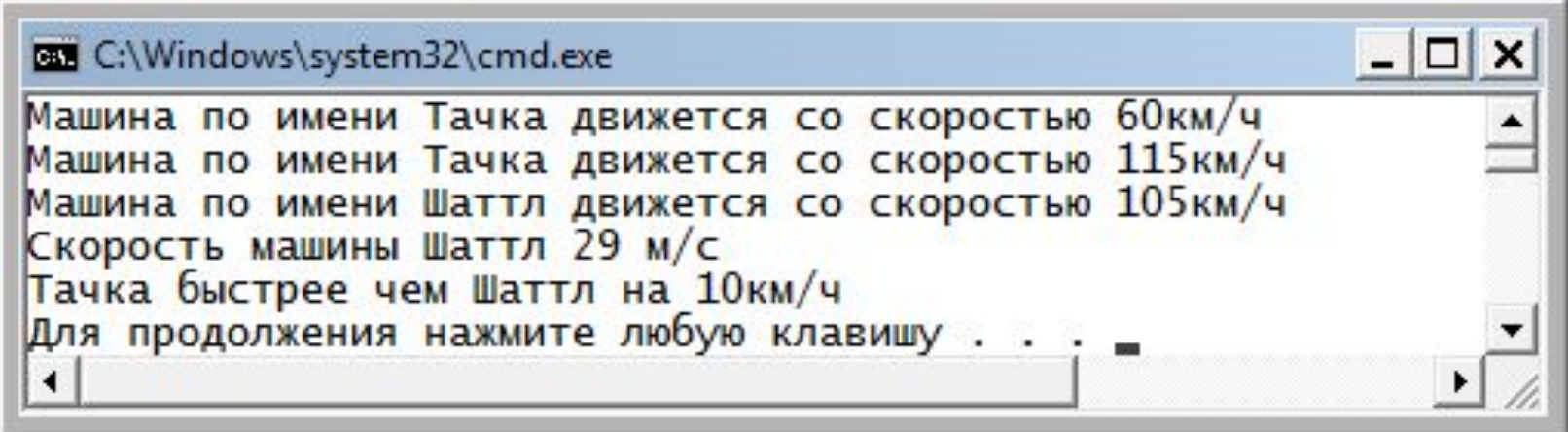
- ◆ А что он будет возвращать? На сколько км/ч быстрее

```
    }  
    public int faster(CCar car2)  
    {  
        return currSpeed - car2.currSpeed;  
    }  
}
```

Класс СCar

- ◆ Так это метод будет вызываться:

```
Console.WriteLine(carPavel.carName + " быстрее чем " + carVova.carName +  
    " на " + carPavel.faster(carVova) + "км/ч");
```



```
C:\Windows\system32\cmd.exe  
Машина по имени Тачка движется со скоростью 60км/ч  
Машина по имени Тачка движется со скоростью 115км/ч  
Машина по имени Шаттл движется со скоростью 105км/ч  
Скорость машины Шаттл 29 м/с  
Тачка быстрее чем Шаттл на 10км/ч  
Для продолжения нажмите любую клавишу . . .
```

Класс CCar

- ◆ Но если мы хотим упростить жизнь пользователю класса (т.е. классу Program), можно весь вывод перенести внутрь метода

```
public void faster(CCar car2)
{
    Console.WriteLine(carName + " быстрее чем " + car2.carName +
        " на " + (currSpeed - car2.currSpeed) + " км/ч");
}
```

- ◆ И вызов в Program будет намного короче:

```
carPavel.faster(carVova);
```


Подведение итогов. Метод

- ◆ Метод – это функция, описанная внутри класса
- ◆ Метод вызывается от объекта класса и ему доступны поля объекта, который его вызвал
- ◆ Метод может возвращать результат или иметь тип `void`
- ◆ Метод может принимать аргументы (или не принимать)

Конструкторы

```
CCar carPavel;  
carPavel = new CCar();  
carPavel.carName = "Тачка";  
carPavel.currSpeed = 60;  
carPavel.Print();
```

```
CCar carVova = new CCar();  
carVova.carName = "Шаттл";  
carVova.currSpeed = 160;  
carVova.Print();
```

- ◆ такое создание объектов не слишком удобно

Конструкторы

```
CCar carPavel = new CCar("Тачка", 60);  
CCar carVova = new CCar("Шаттл", 160);
```

- ◆ Так было бы удобнее

Конструкторы

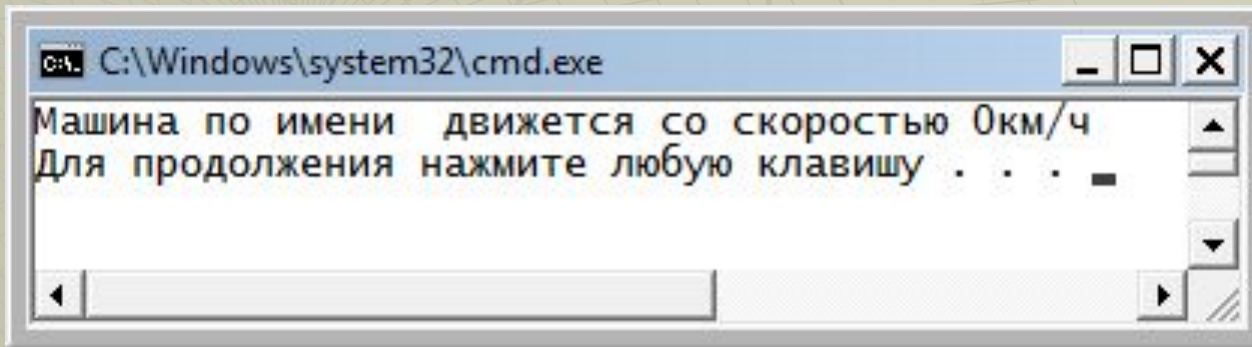
- ◆ Конструктор – особый метод класса, используемый при создании новых объектов данного класса.
- ◆ Конструктор всегда имеет то же имя, что и класс.
- ◆ Конструктор никогда не имеет возвращаемого значения.
 - ◆ Класс может иметь несколько конструкторов, различающихся к-вом и типами аргументов.



Конструкторы

- ◆ Если класс не имеет ни одного конструктора, компилятор создает конструктор по умолчанию.

```
CCar carVova = new CCar();  
carVova.Print();
```



The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window contains the following text:

```
Машина по имени движется со скоростью 0км/ч  
Для продолжения нажмите любую клавишу . . .
```

Конструкторы

- ◆ Добавление хотя бы одного конструктора отменяет создание конструктора по умолчанию

```
class CCar
{
    public string carName; // у авто есть имя
    public int currSpeed; // текущая скорость
    public CCar(string carName, int currSpeed)
    {
        carName = "NoName";
        currSpeed = 0;
    }
    public void Print()
    {}
}
```

```
CCar carPavel = new CCar();
```

```
CCar carVova = new CCar();
```

Перегруженные конструкторы

```
public CCar()  
{  
    carName = "NoName";  
    currSpeed = 0;  
}  
public CCar(string carName)  
{  
    this.carName = carName;  
}  
public CCar(string carName, int currSpeed)  
{  
    this.carName = carName;  
    this.currSpeed = currSpeed;  
}
```

Перегруженные конструкторы

- ◆ При вызове конструктора появляется подсказка. Желательно видеть в подсказке осмысленные имена полей.

```
CCar carPavel = new CCar(  
CCar carVova = n
```

▲ 3 of 3 ▼ CCar.CCar(string carName, int currSpeed)

Ключевое слово `this`

- ◆ Чтобы отличать имена полей от имен аргументов, используется слово `this`.

```
public CCar(string carName, int currSpeed)
{
    this.carName = carName;
    this.currSpeed = currSpeed;
}
```

Цепочки конструкторов

- ◆ При создании автомобиля пользователь может указать нереальную скорость (-10 или 100000).
- ◆ Желательно добавить проверку.
- ◆ Придется добавлять проверку в каждый конструктор?

Цепочки конструкторов

- ◆ Изменим только один конструктор (самый подробный)

```
public CCar(string carName, int currSpeed)
{
    this.carName = carName;
    if (currSpeed < 0 | currSpeed > 300) this.currSpeed = 0;
    else this.currSpeed = currSpeed;
}
```

Цепочки конструкторов

```
public CCar()  
{  
    carName = "NoName";  
    currSpeed = 0;  
}  
public CCar(string carName)  
{  
    this.carName = carName;  
    currSpeed = 0;  
}  
public CCar(int currSpeed)  
{  
    carName = "NoName";  
    this.currSpeed = currSpeed;  
}
```

```
public CCar(): this("NoName",0)  
{  
}  
public CCar(string carName): this(carName,0)  
{  
}  
public CCar(int currSpeed): this("NoName",currSpeed)  
{  
}
```


Свойства

- ◆ Есть еще более грамотный способ обеспечить корректность полей класса – **СВОЙСТВО**.
- ◆ **Свойство – это метод (или пара метода), которые с точки зрения клиентского кода ведут себя как поле.**
- ◆ Такой подход позволят при работе с полями выполнять дополнительную обработку – проверку бизнес-правил (соответствие определенным критериям)

Свойства

```
class CCar
{
    public string carName; // у авто есть имя
    private int _currSpeed; // текущая скорость
    public int currSpeed
    {
        get
        {
            return _currSpeed;
        }
        set
        {
            _currSpeed = value;
        }
    }
}
public CCar(): this("NoName", 0)
```

- ◆ **private** для поля
- ◆ ИМЯ поля с **_**
- ◆ **public** для свойства
- ◆ Внешнее имя
- ◆ **get** и **set**
- ◆ **value**

Свойства

```
public int currSpeed
{
    get
    {
        return _currSpeed;
    }
    set
    {
        if (value < 0 | value > 300) _currSpeed = 0;
        else _currSpeed = value;
    }
}
```

- ◆ Проверку на корректность лучше делать внутри свойства

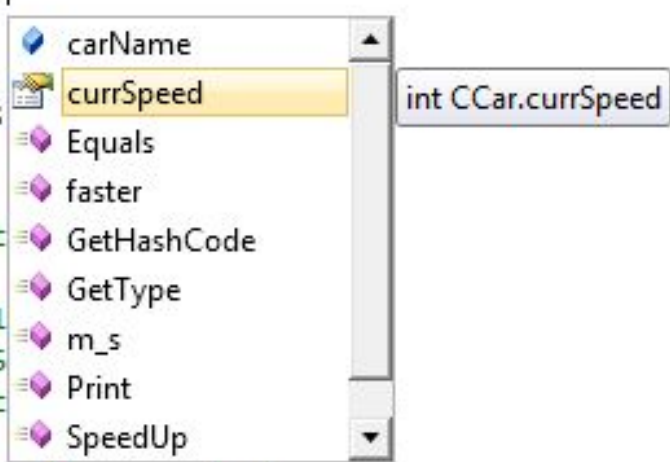
Свойства

```
CCar carVova = new CCar();
carVova.currSpeed = 250;
int s = carVova.|

carVova.Print();

// CCar carMasha
// carMasha.Print

// for (int i = 1
//     carPavel.S
// carPavel.Print
```



The screenshot shows a code completion menu for the 'currSpeed' property of a CCar object. The menu lists various methods and properties, with 'currSpeed' highlighted. A tooltip shows the type 'int CCar.currSpeed'.

- ◆ Свойство выглядит иначе, чем поле

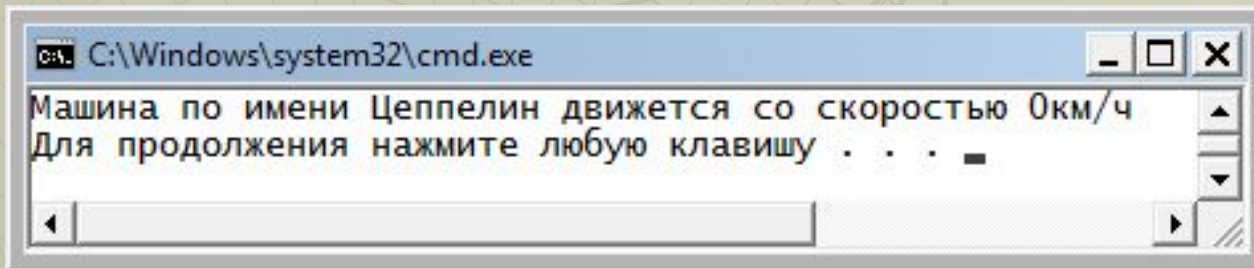
Свойства

```
CCar carVova = new CCar();  
carVova.currSpeed = 250;           // здесь вызывается конструкция set  
int s = carVova.currSpeed;        // здесь вызывается конструкция get
```

- ◆ **get и set вызываются в зависимости от контекста**

Свойства

```
CCar carVova = new CCar("Цеппелин");  
carVova.currSpeed = 350;           // здесь вызывается конструкция set  
carVova.Print();
```



- ◆ Сработало ограничение на скорость свыше 300 км/ч

Что не так в нашем классе?

```
class CCar
{
    public string carName; // у авто есть имя
    private int _currSpeed; // текущая скорость
    public int currSpeed
    {
        get
        {
            return _currSpeed;
        }
        set
        {
            if (value < 0 | value > 300) _currSpeed = 0;
            else _currSpeed = value;
        }
    }
    public CCar(): this("NoName",0){}
    public CCar(string carName): this(carName,0){}
    public CCar(int currSpeed): this("NoName",currSpeed){}
    public CCar(string carName, int currSpeed)
    {
        this.carName = carName;
        if (currSpeed < 0 | currSpeed > 300) this.currSpeed = 0;
        else this.currSpeed = currSpeed;
    }
}
```

Свойства

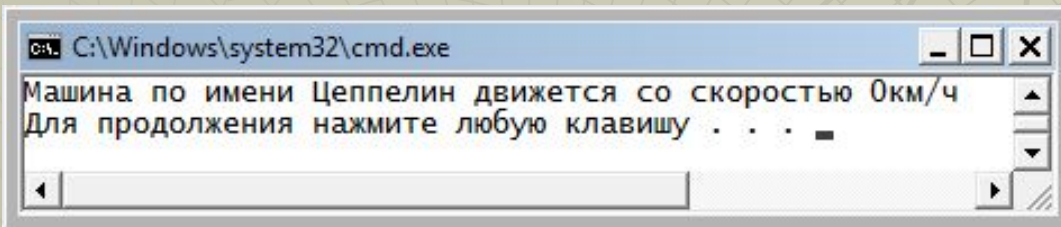
```
public CCar(): this("NoName",0){}
public CCar(string carName): this(carName,0){}
public CCar(int currSpeed): this("NoName",currSpeed){}
public CCar(string carName, int currSpeed)
{
    this.carName = carName;
    this.currSpeed = currSpeed; |
}
```

- ◆ Мы вполне можем вернуть прежний вариант конструктора, ведь теперь здесь будет использоваться свойство.

Свойства

```
CCar carVova = new CCar("Цеппелин", 440);  
carVova.Print();
```

И проверка
на
корректность
выполняется
при вызове
конструктор
а



Свойства

- ◆ Лучшее место в классе для проверки бизнес-правил – это свойство!

Модификаторы доступа

- ◆ Любой член класса может иметь один из модификаторов доступа:
 - **private** (по умолчанию) – доступ только внутри класса
 - **public** – для всех подключенных сборок
 - **protected** – только для своего класса и наследников
 - **internal** (по умолчанию) – только для своей сборки

Свойства

- ◆ Используя модификаторы доступа можно управлять свойствами:
 - создать свойство только для чтения – клиент сможет только получать значение, но не изменять его
 - свойство только для записи – клиент сможет записывать значение, но не сможет прочитать

Сбор мусора



- ▶ Если не существует ни одной ссылки на объект, то предполагается, что этот объект больше не нужен, и занимаемая им память освобождается.

Сбор мусора



Поскольку на сбор мусора требуется определенное время, динамическая система C# активизирует этот процесс только по необходимости или в специальных случаях.

Деструкторы



- ◆ **Деструктор** – метод, который должен вызываться непосредственно перед тем, как объект будет окончательно разрушен системой сбора мусора.

Деструкторы

```
~имя_класса()  
{  
    // код деструктора  
}
```

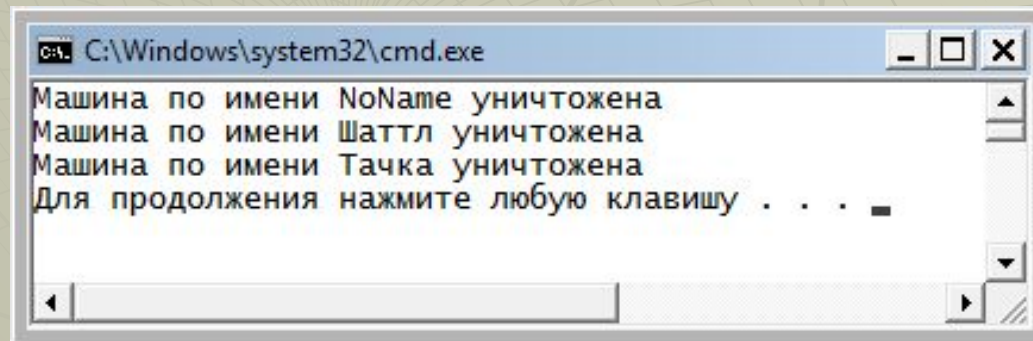
```
~CCar()  
{  
    Console.WriteLine("Машина по имени " + carName + " уничтожена");  
}
```


Деструкторы

```
class Program
{
    static void Main(string[] args)
    {
        CCar carPavel = new CCar("Тачка", 60);

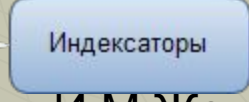
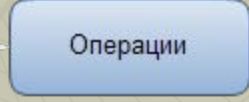
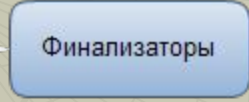
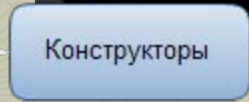
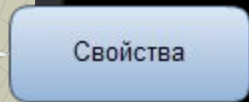
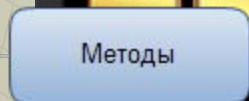
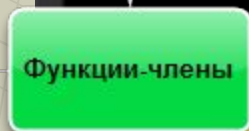
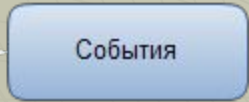
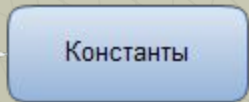
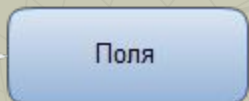
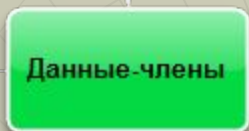
        CCar carVova = new CCar("Шаттл", 160);

        CCar carMasha = new CCar();
    }
}
```



```
C:\Windows\system32\cmd.exe
Машина по имени NoName уничтожена
Машина по имени Шаттл уничтожена
Машина по имени Тачка уничтожена
Для продолжения нажмите любую клавишу . . .
```

Состав класса



Класс

