4.ООП в .NET

Съдържание

- ООП – принципи и понятия
- ООП – предимства и особености
- ООП и .Net Framework
- Класове
  - Членове на класа
  - Модификатори за достъп
- Конструктори
- Полета

Съдържание

- Методи
  - Предаване на параметри
  - Method overloading
  - Статични методи
- Свойства(Properties)
- Структури
- Интерфейси

Съдържание

- Наследяване
  - Абстракти класове
  - Виртуални членове
  - Разлики между интерфейс и абстрактен клас

Що е то ООП ?

- ООП е:
  - Съвкупност от принципи и идеи, които позволяват да се моделират обектите от реалния свят със средствата на програмния език

Начин на мислене

Малко история

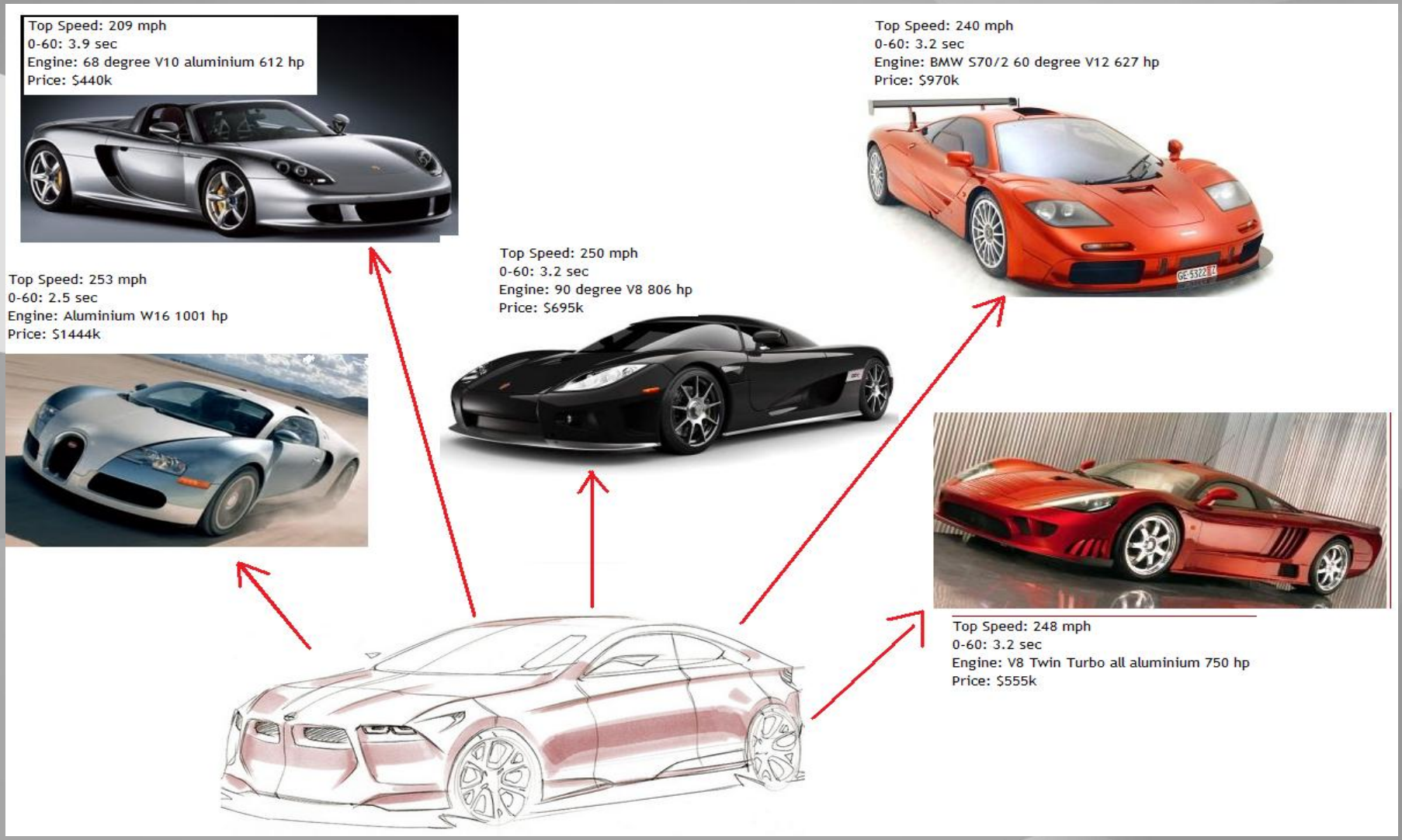
- 1960 –Simula , има концепции от ООП-то
- 1968 – Smalltalk – първият ООП език
- Java, C#
- Други обекто-ориентирани езици:
  - Ruby
  - Python

ООП – основни понятия

- Клас
- Обект
- Инстанциране на обекти
- Интерфейс
- Свойство
- Метод
- Абстракция на данни
- Абстракция на действия


СВ. КЛИМЕНТ ОХРИДСКИ

Клас != Обект


Top Speed: 209 mph
0-60: 3.9 sec
Engine: 68 degree V10 aluminium 612 hp
Price: $440k
Top Speed: 240 mph
0-60: 3.2 sec
Engine: BMW S70/2 60 degree V12 627 hp
Price: $970k
Top Speed: 253 mph
0-60: 2.5 sec
Engine: Aluminium W16 1001 hp
Price: $1444k
Top Speed: 250 mph
0-60: 3.2 sec
Engine: 90 degree V8 806 hp
Price: $695k
Top Speed: 248 mph
0-60: 3.2 sec
Engine: V8 Twin Turbo all aluminium 750 hp
Price: $555k

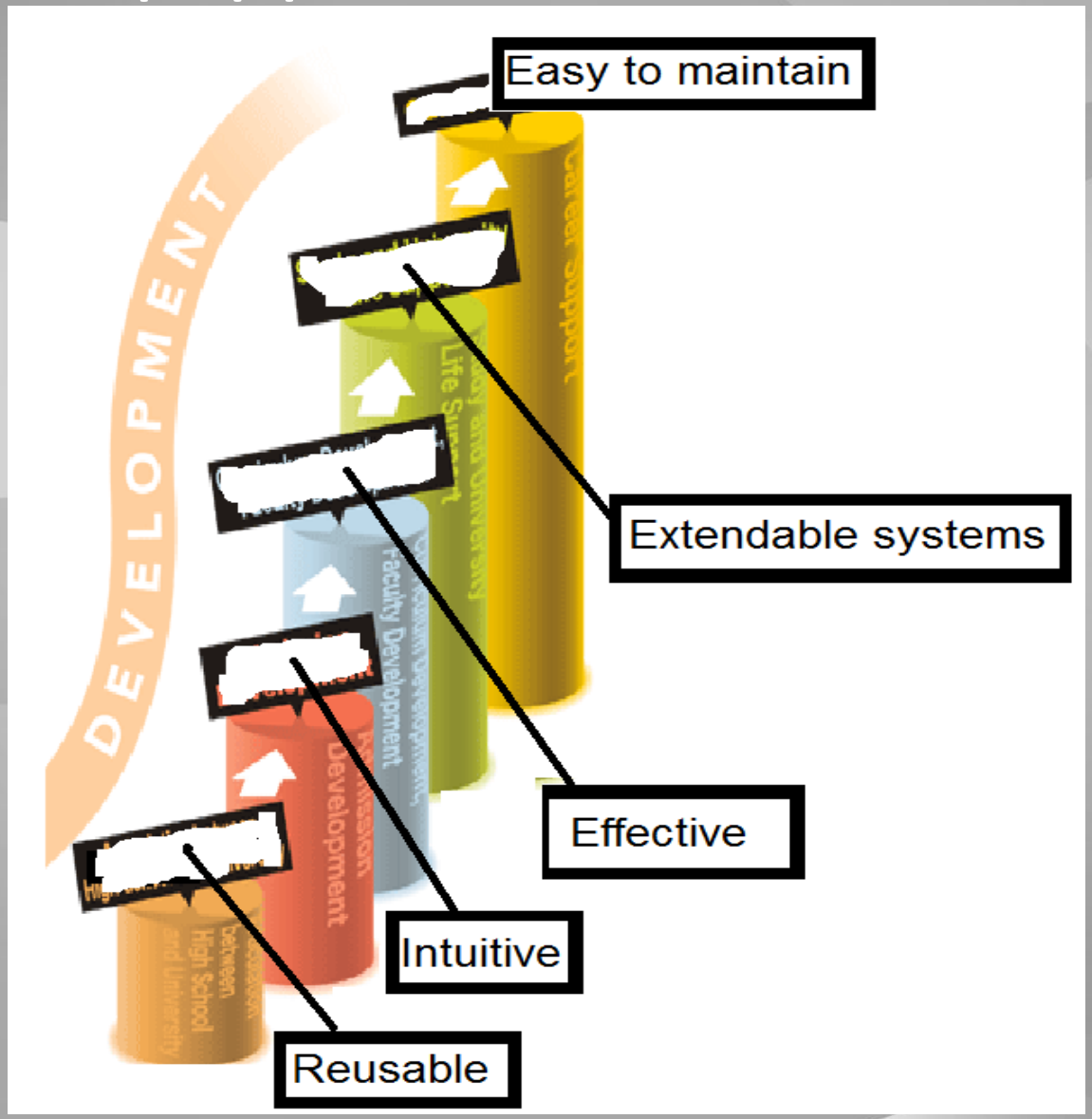
Светата Троица на ООП

Капсулация

Наследяване

Полиморфизъм

Защо да използваме ООП



ООП и .NET

Всичко е обект

ООП и .NET - Обобщение

- В NET Framework обектно-ориентираният подход е залегнал на най-дълбоко

архитектурно ниво

- Всички .NET езици са обектно-ориентирани

Класове

Клас (ООП) =

- Клас (.NET)
- Структура(.NET)

Членове на клас

Class Car

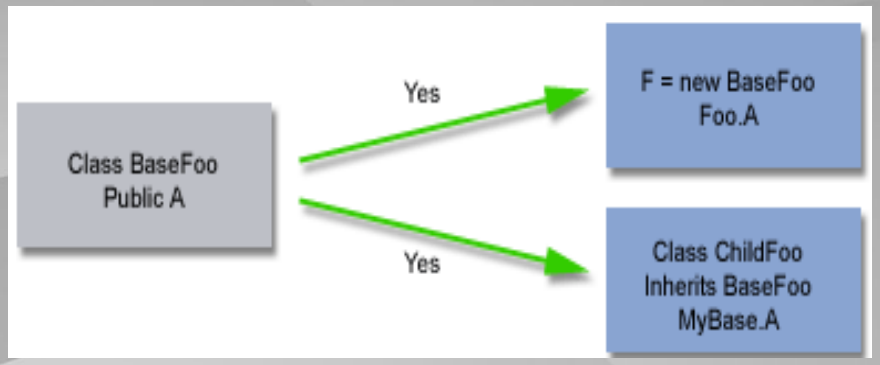
{

- Полета — private string model;
- Конструктори — public Car() ;
- Методи — public void Move();
- Свойства — public int Power { get; set; }
- Константи — public const NUMBER_OF_WHEELS = 4;
- Индексатори — public this[] {get … set…}
- Събития — public Event OnSpeedUp;
- Оператори — public static bool operator >(Car c1, Car c2)
- Вложени типове — public class Engine {}

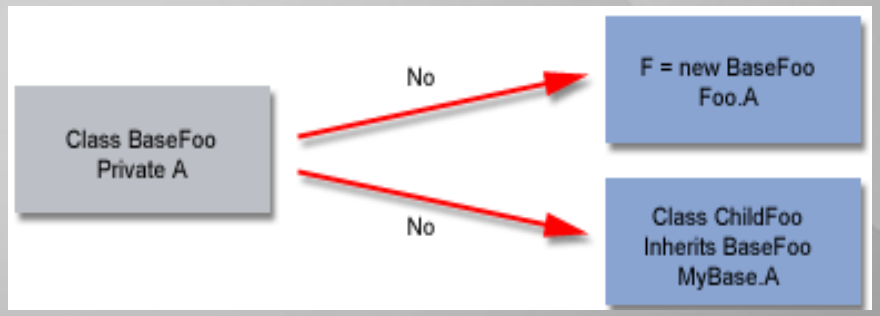
}

Модификатори за достъп

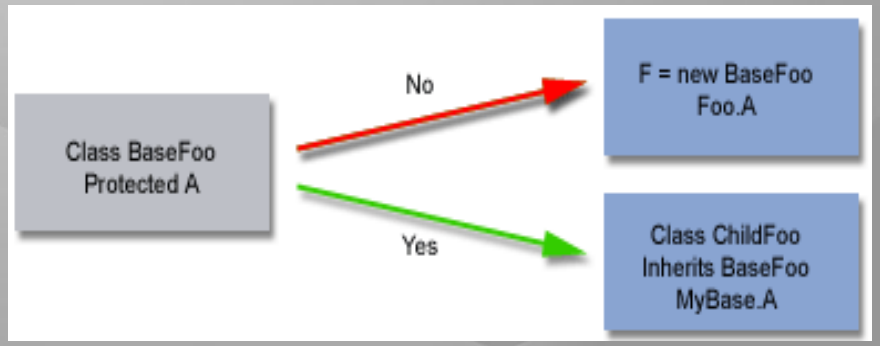
Public



Private

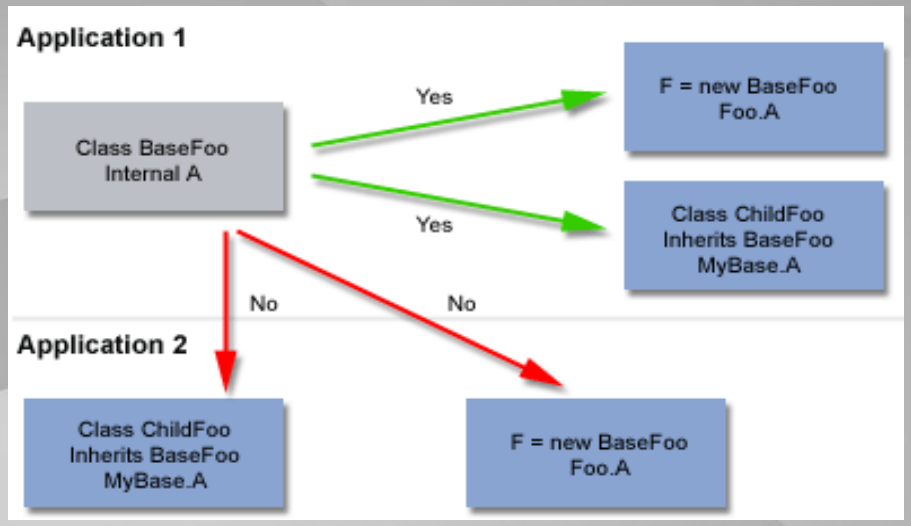


Protected

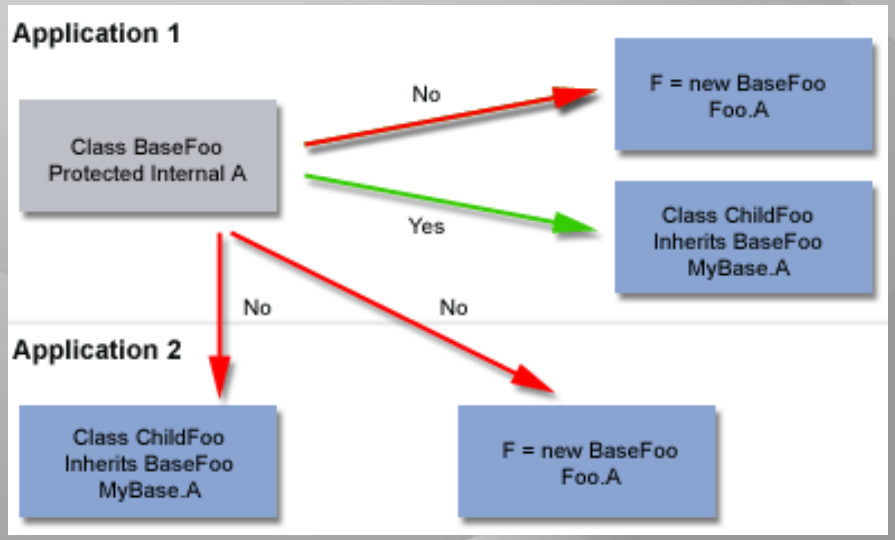


Модификатори за достъп

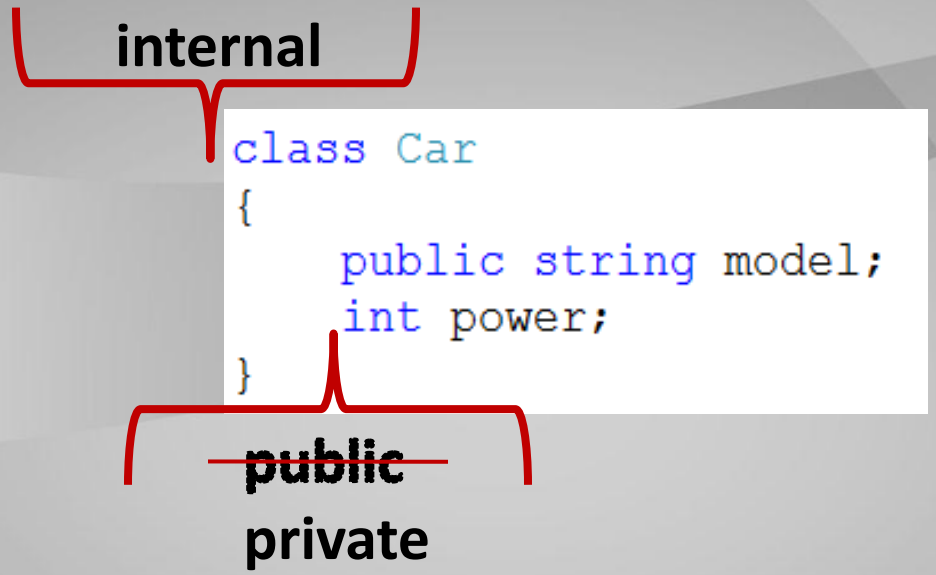
Internal



Protected Internal



Модификатори за достъп



- Private е модификатора по подразбиране
- За всеки член на класа изрично трябва да се укаже модификатор(за разлика от C++)

Създаване на обект

! Attention C++ developers

```
//C++ code
CMyClass myClass;
```
→ Създава инстанция в стека

```
//C++ code
CMyClass myclass = new CMyClass();
```
→ Създава инстанция в heap-а

В .NET инстанция се създава **само с new**

Конструктори

- Служат за инициализация на обект
- Извиква се чрез запазената дума new
- Не връщат стойност
- Имат специална сигнатура
- Може да имаме повече от един конструктор
- Ако не е дефиниран конструктор , то компилаторът създава такъв по подразбиране

Constructor Initializers

- Приемат 2 форми
  - this(…) – извиква конструктор от същия клас
  - base(…) –извиква конструктор на базов клас

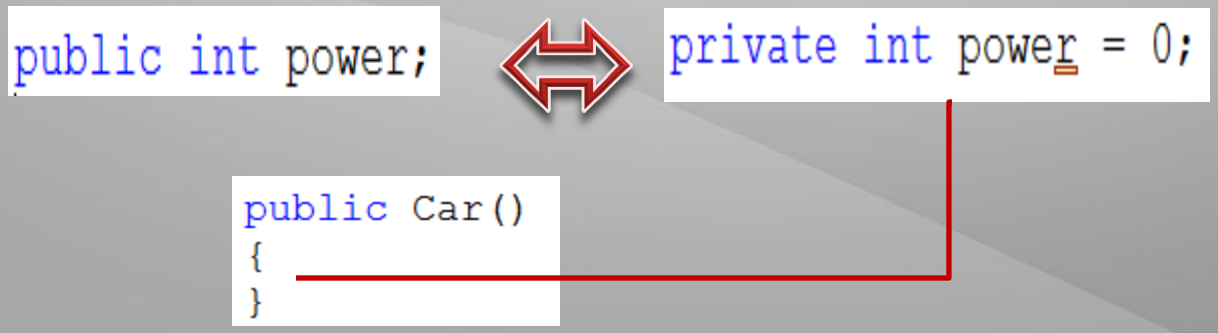
```
public SportCar(int power)
    : base("Toyota Celica")
{
    this.power = power;
}
```

```
public SportCar()
    :this(150)
{
}
```

Demo Time - Конструктори

Член променливи

- Същност – съдържат данни
- Освен модификатори за достъп могат да имат и модификатори : const, readonly, static
- Инициализиране – при декларацията или автоматично
- Полетата се инициализират при извикване на конструктора



Static vs Instance Members

- По подразбиране всички членове са по инстанция(т.е не са статични)
- Статичните членове:
  - се създават, когато приложението,
  - съдържащо класа се зареди
  - съществуват през целия живот на приложението
  - имат само едно копие, независимо колко обекта от този клас са създадени
  - Достъпват се през класа (НЕ МОГАТ ДА СЕ ДОСТЪПВАТ ПРЕЗ ИНСТАНЦИЯ)

Static vs Instance Members

Сигнатурата на Main е:

public **static** void Main().

Защо?

Constants vs readonly fields

```
public const int NUMBER_OF_WHEELS = 4;
```

Const = compile time константи

```
public readonly string model;

public Car(string model)
{
    this.model = model;
}
```

Readonly = run time константи

Constants vs readonly fields

- Константи
  - Инициализират се единствено при дефинирането им
  - Не могат да се променят
  - Статични са
- Полетата за четене
  - Инициализират се в конструктора
  - Достъпват се през инстанция на класа

Статичен конструктор

- Използва се за инициализиране на статични полета
- Клас може да има само един статичен конструктор
- Няма параметри

Constants vs readonly fields

- Съществува ли поле, което
  - Не се достъпва през инстанция
  - Може да задаваме стойността му runtime

Методи

- Сигнатура на метод
- Предаване на параметри
  - Ref
  - In – по подразбиране
  - Out
- Предаване на променлив брой параметри – ключовата дума params
- Методи с еднакви имена
- Статични методи

Свойства

- Properties as Smart Fields
- Свойствата са методи, които изглеждат като полета
- Имат getter и setter
- Свойства само за четене
- Lazy initialization

Demo Time – Членове на клас

Интерфейс

- Договор между класа ,който го наследява и потребителя на класа
- Определят поведение

Аз съм Class.
Ще имплементирам всички членове на интерфейса

Интерфейси - същност

Interface IVehicle

{

- Методи `void Move();`
- Свойства `int MaxSpeed{get;}`
- Събития `event OnSpeedUp;`
- Индексатори `int this[]{get;}`

- ~~Оператори~~
- ~~Конструктори~~
- ~~Полета~~
- ~~Вложени типове~~

!

}

Интерфейси - особености

- Всички членове са public

```
public interface IVehicle
{
    public void Move();
    int MaxSpeed { get; set; }
}
```

Error

1 The modifier 'public' is not valid

- Има само дефиниции, без имплементации
- Не могат да се инстанцират

```
IVehicle vehicle = new IVehicle();
```

Error

1 Cannot create an instance

Имплементиране/Наследяване на интерфейси

- В C# няма множествено наследяване

Но

```
public class Car : IVehicle, IRaceble, IValidate
```

- Клас може да наследява много интерфейси
- Ако клас наследява интерфейс, то той имплементира всичките му членове
- Всички членове на интерфейса се имплементират като public

Cast object to Interface

- Къде е проблема в следния код?

```
Car myCar = new Car();
IVehicle vehicle = (IVehicle)myCar;
vehicle.Move();
```

- Какво ще стане , ако Car НЕ имплементира IVehicle?

IS vs AS

- IS връща bool → *expression is type*

```
Car myCar = new Car();
if (myCar is IVehicle)
{
    IVehicle vehicle = (IVehicle)myCar;
    vehicle.Move();
}
```

- AS връща null или кастнатия обект

→ *object = expression as type*

```
Car myCar = new Car();
IVehicle vehicle = myCar as IVehicle;
if (vehicle == null)
{
    vehicle.Move();
}
```

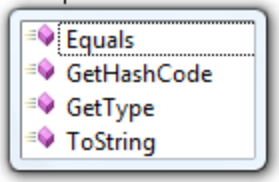
- **AS** е по-бързо от **IS**

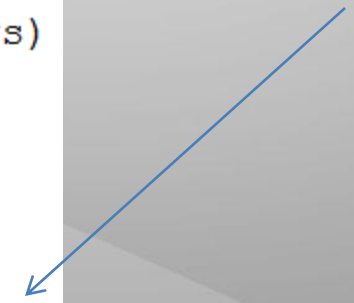
Implement interface **Explicitly** name hiding

```
public class Car :IVehicle
{
    void IVehicle.Move()
    {
        Console.WriteLine("I am moving");
    }
}
```

```
static void Main(string[] args)
{

    Car myCar = new Car();
    myCar.
}
```





Къде ми е метода?

Явна имплементация на интерфейси Name Hiding

- Членовете на интерфейса не са част от интерфейса на класа
- За да се достъпи член на интерфейса от инстанция на класа е нужно преобразуване на типове

+ на name hiding-а:

- Помага при колизиите на имена
- По – малък интерфейс на класа

Demo Time: Name Collision & C#

Извод

Винаги имплементирайте интерфейсите явно

Наследяване в C#

- В C# няма множествено наследяване
- Производния клас наследява всички членове на базовия клас
- Само конструкторите не се наследяват
- Модификаторите за достъп на наследените компоненти в производния клас ⇔ модификаторите в базовия клас
- Всеки производен клас може да се третира като базов(upcasting)

Наследяване

минутка за размисъл

A derived class should require no more and should promise no less than its base class

Upcasting & Downcasting

- Downcasting – ((Derived)base)
  - C# позволява downcasting дори когато типа на base не е Derived=> кодът се компилира, но ще хвърли runtime грешка
- Upcasting = ((Base)derived)
  - винаги се преобразува, но...

! Derived derived = new Base()

Е

грешка

Sealed classes

- Класове, които не могат да се наследяват
- Дефинират се със запазената дума sealed
- Абстрактните класове не могат да са sealed
- Членовете им не трябва да имат модификатор за достъп protected

```
sealed class MyClass
{
}
```

Абстрактни класове

- Класове, за които не всички членове са имплементирани
- Хибрид между клас и интерфейс
- Дефинира се със запазената дума abstract
- Класът-Наследник на абстрактен клас трябва да имплементира абстрактните методи

Правила

- Абстрактните класове не може да са sealed

```
//Грешка
abstract sealed class absClass
{
}
```

- Абстрактните членове не може да са static

```
//Грешка
public abstract static int MaxSpeed;
```

- Абстрактните членове не може да са virtual

```
//Грешка
public abstract virtual int MaxSpeed();
```

- Абстрактните членове не може да са private

```
//Грешка
private abstract int MaxSpeed();
```

Абстрактен клас и Интерфейс РАЗЛИКИ

Вие сте…

- …
- …
- …
- …

Едно добро парче код ;)...hm

```
public void DrivingComport(Vehicle baseVehicle)
{
    if (baseVehicle is Jeep)
    {
        Console.WriteLine("Driving is safe");
    }
    else if (baseVehicle is SportCar)
    {
        Console.WriteLine("I can cause backache");
    }
    else if (baseVehicle is Plane)
    {
        Console.WriteLine("Don't complain, you choose to use low costs");
    }
}
```

Strive For Polymorphism

```
public void DrivingComport(Vehicle baseVehicle)
{
    Console.WriteLine(baseVehicle.ComfortDescription());
}
```

- Приемане на различни форми от един обект
- Извикване на метод на Клас-Наследник през обект от базовия клас
- Late & Early binding

Полиморфизъм - понятия

```
[abstract ] class BaseClass
{
    abstract methods;
     virtual methods
}

class DerivedClass : AbsClass
{
     override methods;
     new methods
}
```

Полиморфизъм - понятия

- public **abstract** void Move() – **трябва** да се **имплементира** в Класа- Наследник
- public **virtual** void Move() – **може** да се **предефинира** в Класа- Наследник
- public **override** void VMethod() – **предефинира** базов метод
- public [**new**] void VMethod() – **скрива** базов метод, когато достъпваме обект от наследен клас през референция на базов клас се извиква метода от базовия клас

DemoTime
Abstract classes & Polimorphism

Q & A