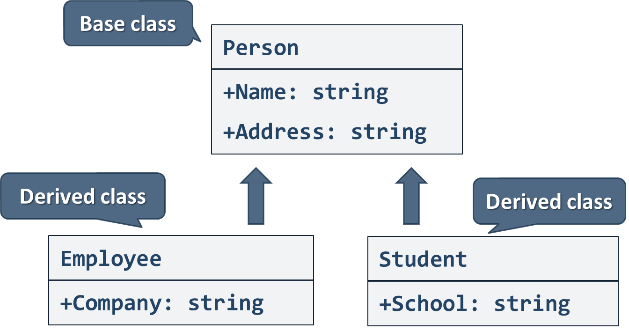
**Inheritance**

1. **Inheritance**

* Superclass - Parent class, Base Class
  + The class giving its members to its child class
* Subclass - Child class, Derived class
  + The class taking members from its base class

\* Абстрактни са, защото не отразяват действителността и не се ползват както примерно „котка“, а е „животно“

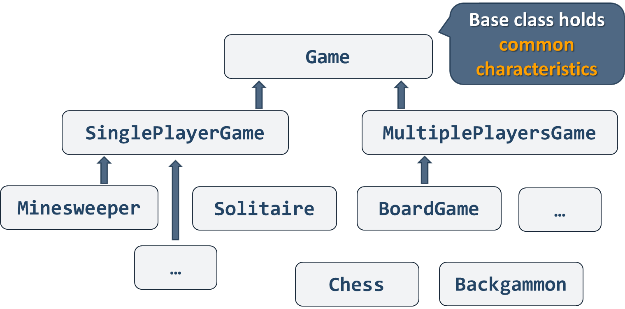
\* Идеално, защото не копираме код, а използваме вече написан – намалява бъговете и е по-чисто



\* Общите неща са само в един клас

1. **Class Hierarchies**

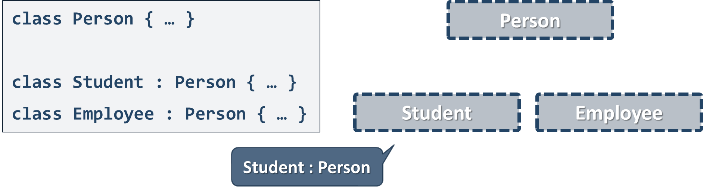
* **Inheritance** leads to **hierarchies** of classes and/or interfaces in an application:



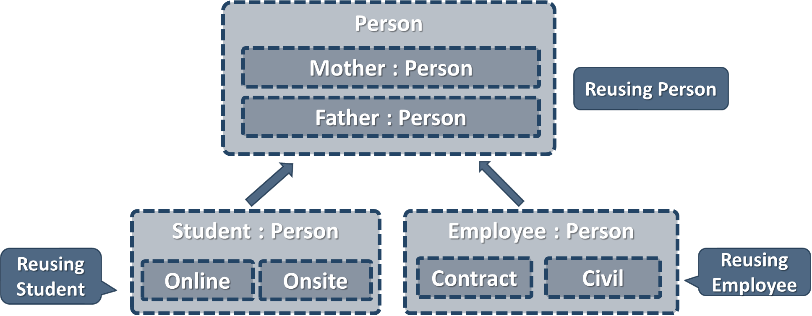
\* Получава се само, когато се замисляш има ли повтарящ се код, а не с предварително планиране

1. **Inheritance in C#**

* In C# inheritance is defined by the **:** operator

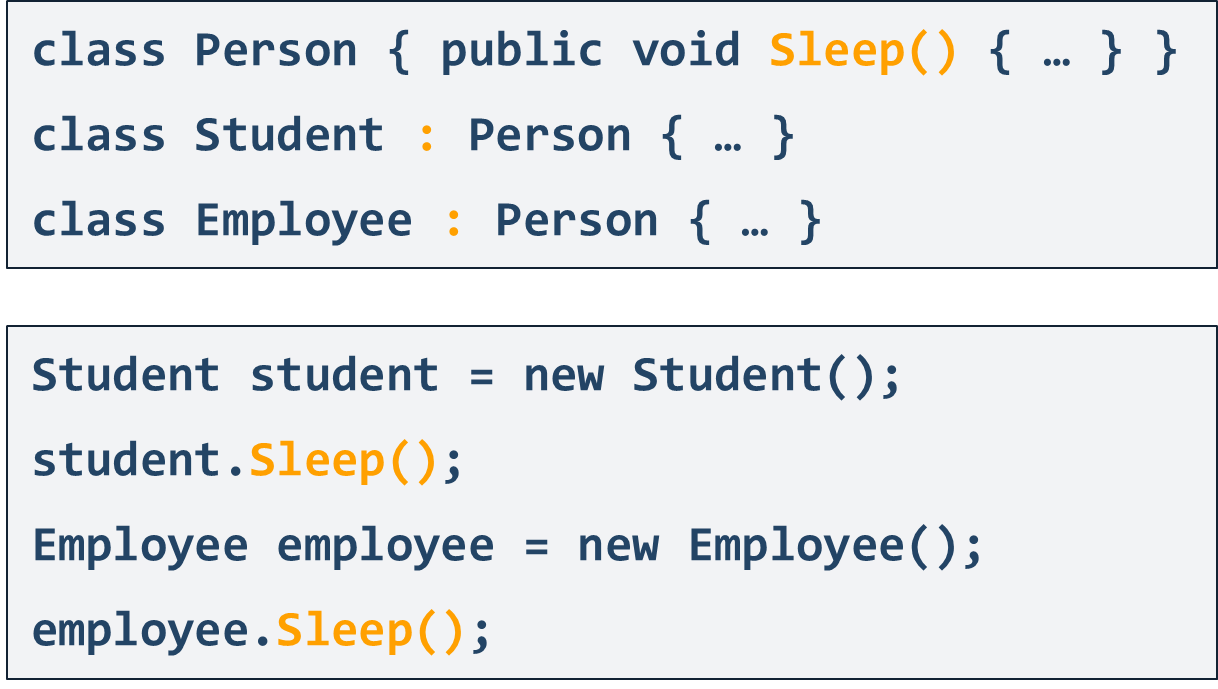


* Derived classes **take all members** from base classes



\* Без значение private или не – всичко наследява

* You can access inherited members as usual

****

* Constructors are **not inherited**

\* Длъжни са да използват основния конструктор, когато инцииализират своя си; Ако няма собствен, С# няма да знае как да използва базовия клас и как да го инстанцира.

public Cat(string name, int age, int miceKilled)

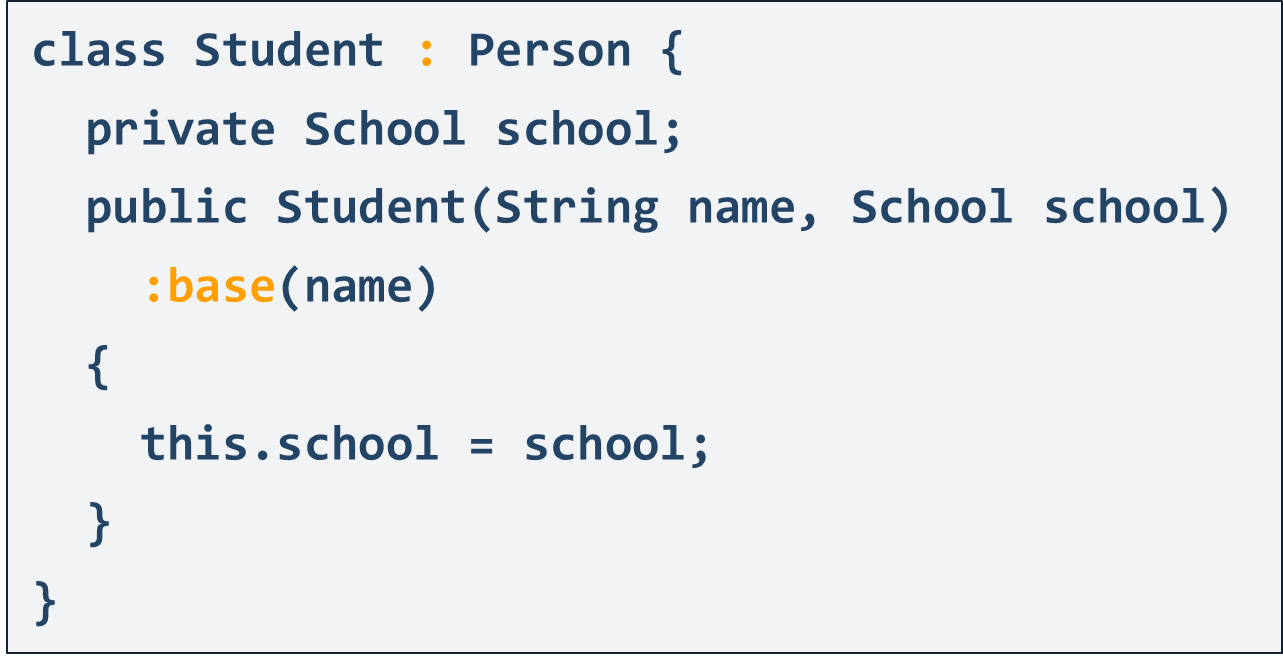
:base(name, age)

{

this.MiceKilled = miceKilled;

}

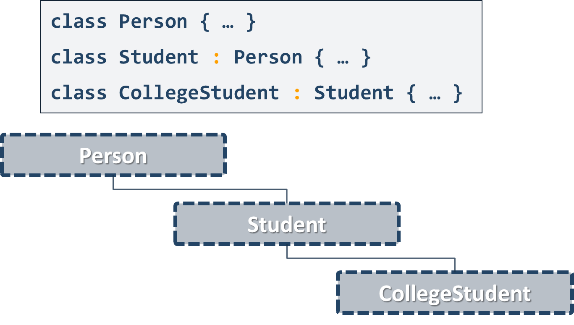
* Constructors **can be reused** by the child classes



* Derived class instance **contains** instance of its base class

****

* Inheritance has a **transitive relation**

****

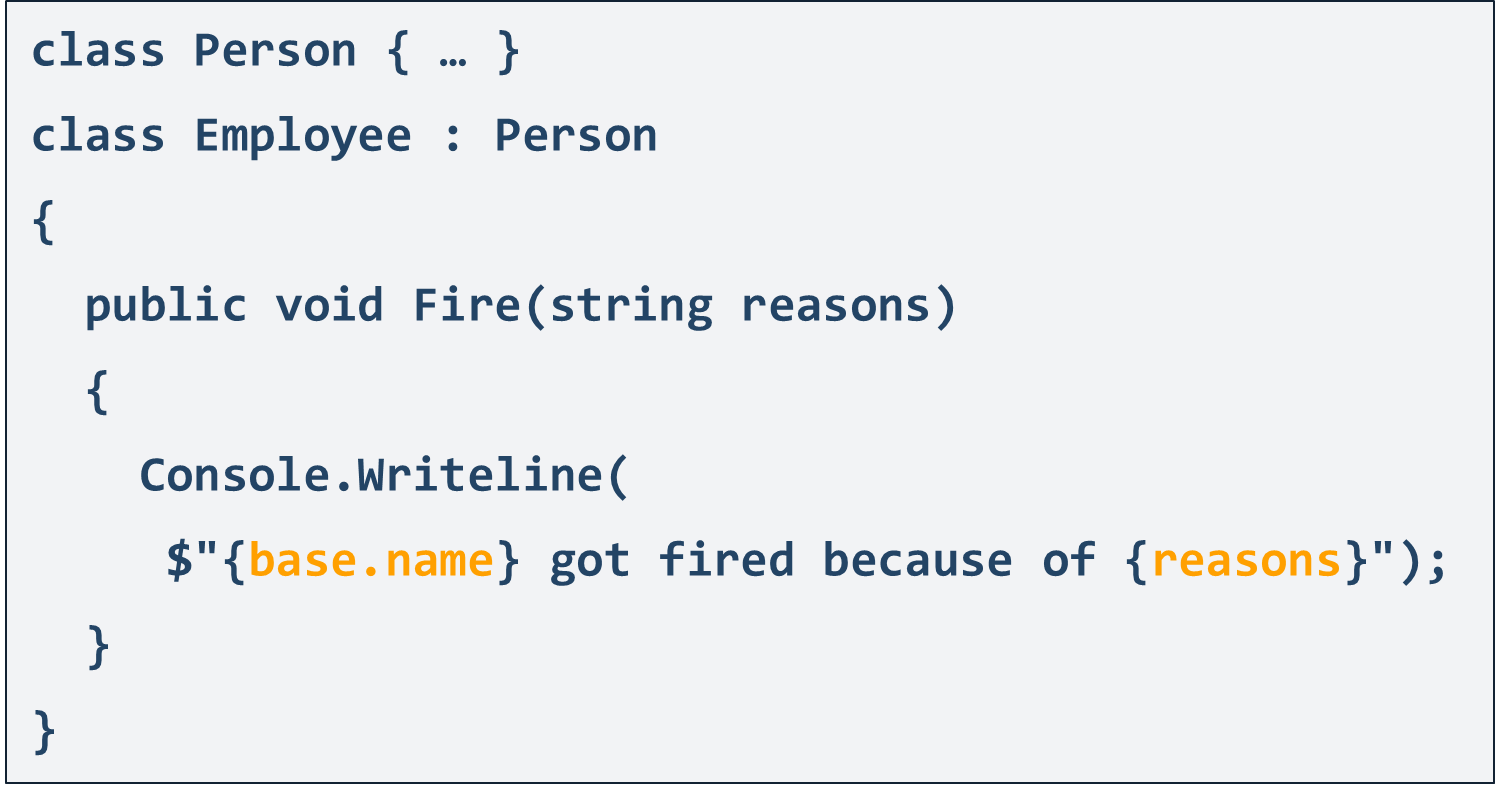
\* Връзката „е“, т.е едното е другото, а не „има“ (студентът Е човек, двигателят НЕ Е кола)

* In C# there is no **multiple** inheritance
* Only **multiple interfaces can be implemented**

****

1. **Accessing Members of the Base Class**

* Use the **base** keyword



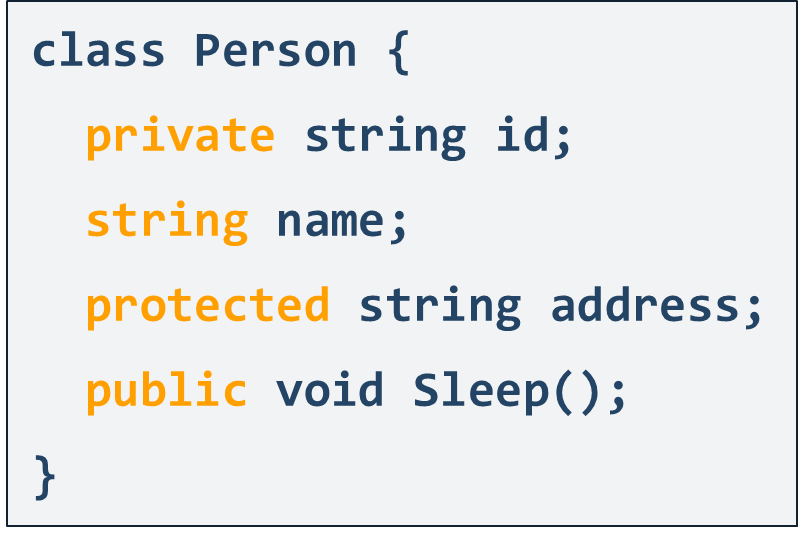
\* Всичко, което е в базовия, се извиква с base. ИЗРИЧНО се търси там! Ако викаш с this., то търсиш в текущия клас, а не в базовия!

1. **Inheritance and Access Modifiers**

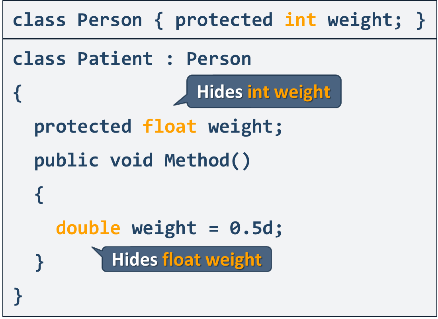
* Derived classes **can access all public** and **protected** members
* They can access **internal** members **if in same assembly**

**\* Protected internal –** В същото асембли могат САМО да го виждат наследниците

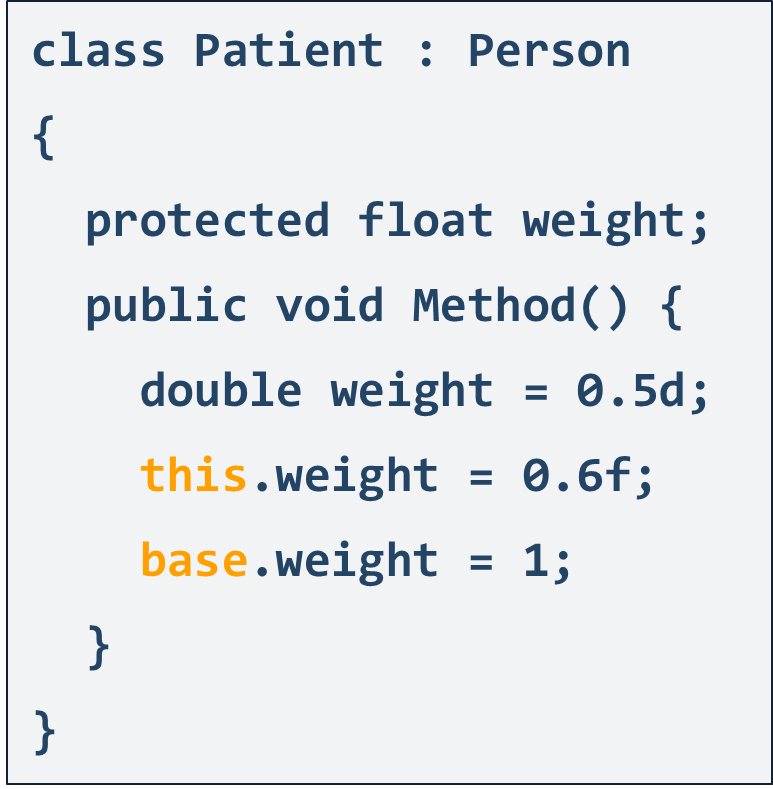
* **Private** fields are **not inherited** in subclasses



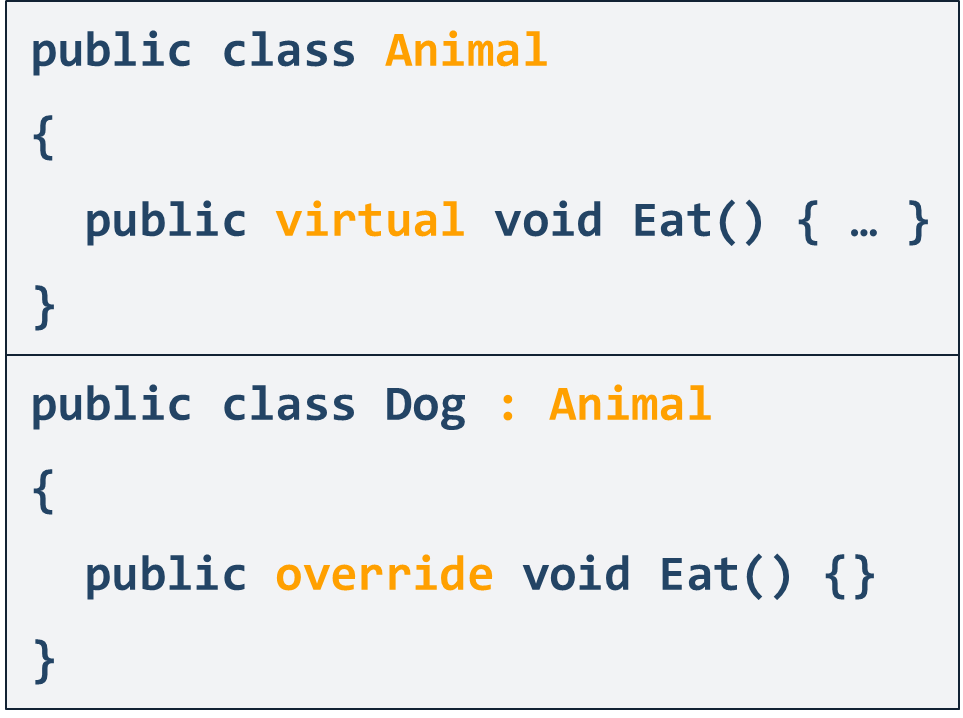
* Derived classes **can hide** superclass variables



* Use **base** and **this** to specify member access

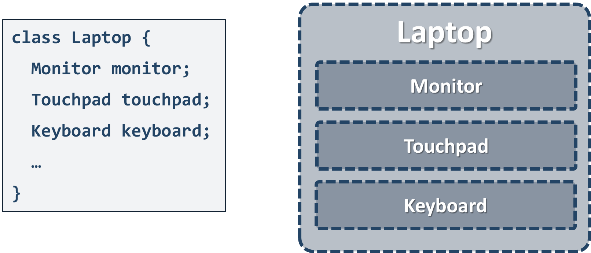


* **virtual** – defines a method that **can be overridden** by the classes that inherit it. Базовото го маркираме с virtual, а презаписващите – с override. Леко с this, за да не влезе в рекурсия. Затова return $“{base.SayHello()} + Bark!“;

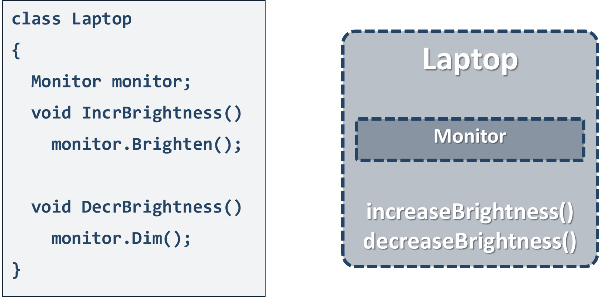


1. **Composition and Delegation**

* Using classes to define classes



**Delegation**



Клас за валидиране на условия в сетърите:

public static class Validator

{

Public static void ValidateTextLength(string text, int length, string message = null)

{

if(text.Length < length)

{

Throw new ArgumentException($“{message } must be more than {length}”);

}

}

}

и после в конструктура на класа, който влаидираме, добавяме:

Validator.ValidateTextLength(name, 6, $”{nameof(Animal)} {nameof(Animal.Name)}”);

\* НИКОГА, НИКОГА, НИКОГА не презаписваш променливи, защото после, ако ги викаш през единия клас дават една стойност, а през другия – друга. Така Name може да е string през Animal и int през Dog