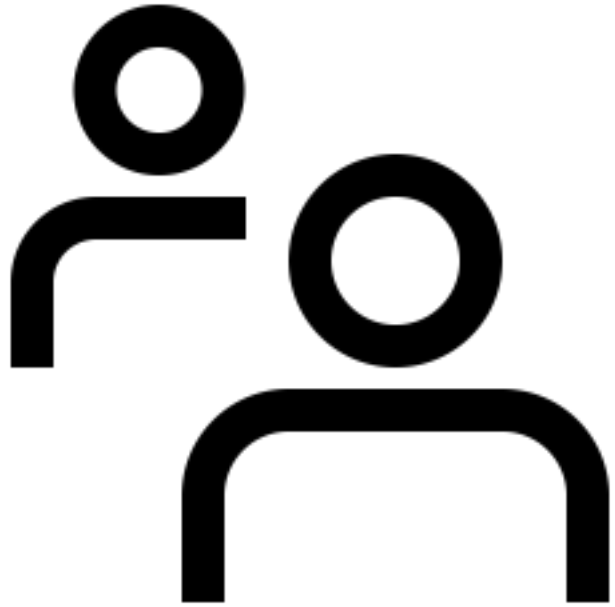
C#
V
OOP

Inheritance & Casts

Plan

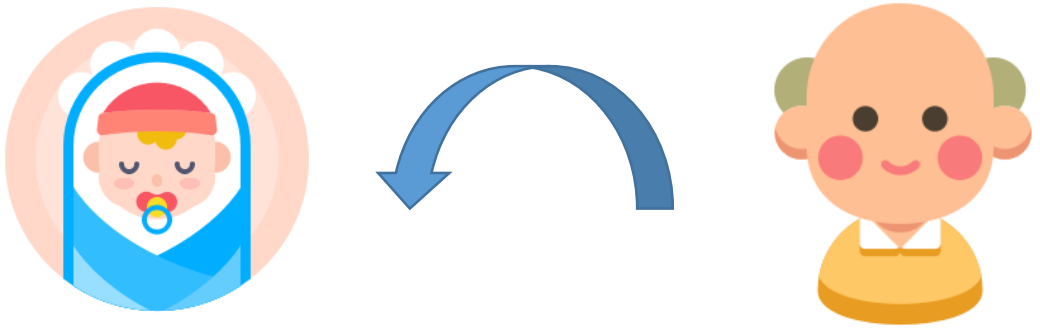
1. Inheritance
   parent
   child
2. Casts
   explicit (явное)
   implicit (неявное)
3. Operator overloading
4. Value Types



Inheritance…

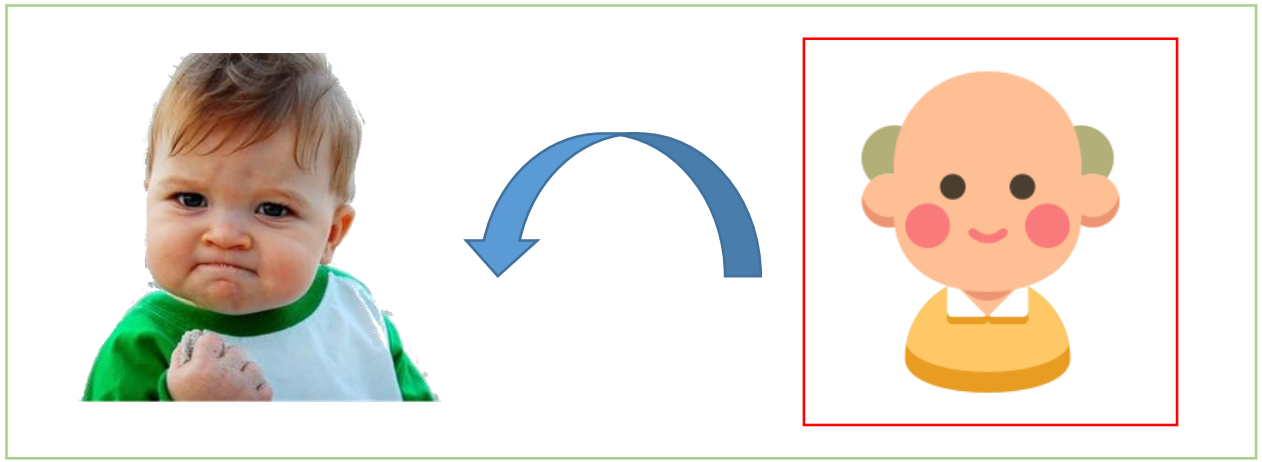
Хотим расширить класс новыми возможностями,
но при этом старые нас полностью устраивают**?**

Объект класса потомка можно положить в объект класса предка!!!



1. Класс предок (*parent*) – базовый класс от которого наследуются
2. Класс потомок (*child*) – класс, который унаследован от класса предка
3. *Child* наследуется от **одного** *Parent*

…inheritance



Объект класса потомка можно положить в объект класса предка!!!

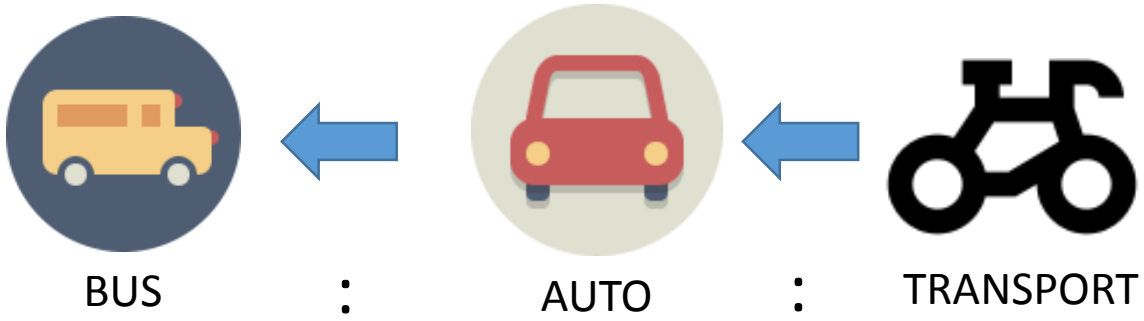
4. *Child* содержит все неприватные поля, свойства и методы *Parent*

5. *Child* **расширяет функциональность** *Parent*

У *Parent* может быть много *childs*, но у *child* всегда один *parent* (в C#)

Примеры цепочек наследования: **млекопитающее – человек – женщина**

Bus Example




```
class Auto: Transport{
    public EngineType EngType { get; set; }
    public void Refuel()
    {
        // дозаправка
    }
}
```

```
class Transport {
    public int NumWheels { get; set; }
    public void Move() { }
}
```

```
class Bus: Auto
{
    public int RouteNum { get; set; }
    public void OpenAllDoors()  { }
}
```

```
void Main(string[] args)
{
    Bus bus = new Bus();
    bus.NumWheels = 6;
    bus.Move();
    bus.Refuel();
    bus.RouteNum = 11;
}
```

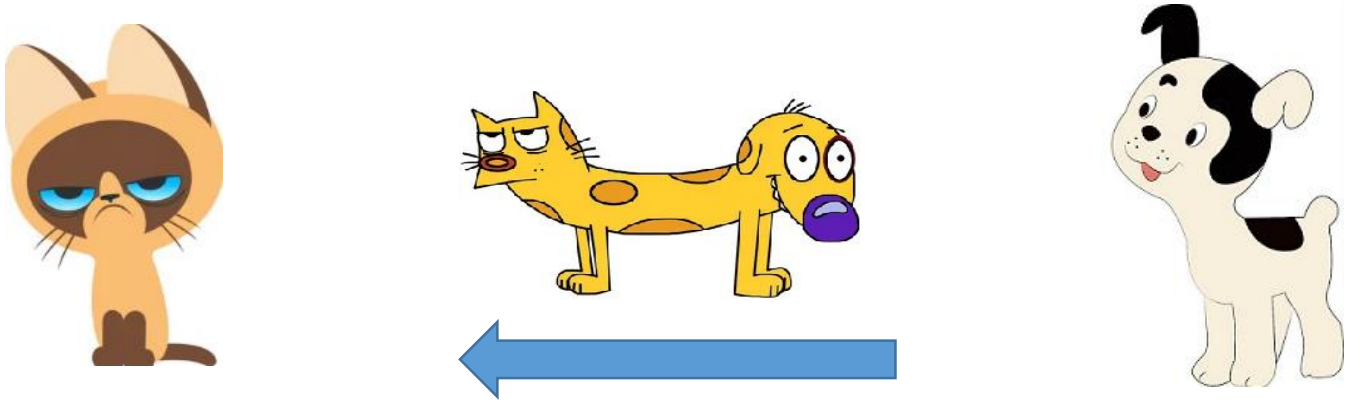
Модификаторы доступа (про protected)

Могут применяться к полям, методам и свойствам и определяют, возможен ли доступ к данным элементам извне.

public	Доступно вне класса
private	Доступно только из класса
protected	Не доступно вне класса, но доступно в наследующих данный класс типах

Casts (приведение типов)

Преобразование (конвертирование) одного типа в другой



Casts (primitive types)

Преобразование (конвертирование) одного типа в другой

Неявное – implicitly

потеря данных исключена

Автоматически

```
int century = 21;
double a = century;
```

```
int century = 21;
double a = (double)century;
```

Явное – explicitly

потеря данных осознается программистом

Вручную

```
double g = 9.8;
int a = g;
```

```
double g = 21;
int a = (int)g;
```

Casts (reference types)

Преобразование (конвертирование) одного типа в другой

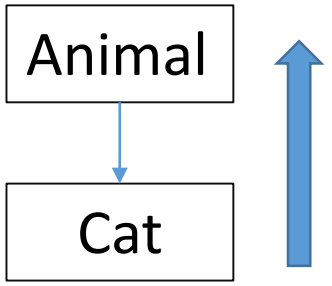
```
class Animal {
    public string Say { get; set; }
    public string Weight { get; set; }
    public void Eat() { }
}

class Cat : Animal{
    public Cat(){
        this.Say = "Mew";
    }
}

class Dog: Animal{
    public Dog(){
        this.Say = "Gaw";
    }
}
```

1. Upcasting (восходящее)

child можно положить в *parent*

```
Cat cat = new Cat();
Animal animal = cat;
```

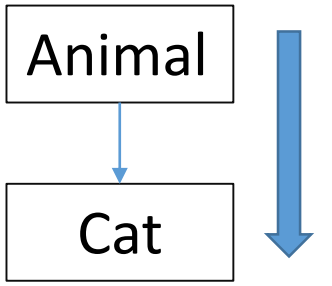
imlicitly



2. Downcasting (нисходящее)

```
Animal animal = new Cat();
Cat kitty = (Cat)animal;
```

explicitly



Casts (is & as)

Преобразование (конвертирование) одного типа в другой

is – проверяет тип переменной (true / false) – является ли

```
Animal animal = new Dog();
if (animal is Cat)
   Console.WriteLine("animal is cat");
else
   Console.WriteLine("animal is dog");
```

```
Animal animal = new Dog();
Cat cat;
if (animal is Cat)
   cat = (Cat)animal;
```

as – пытается выполнить приведение, если не получается, то возвращает null

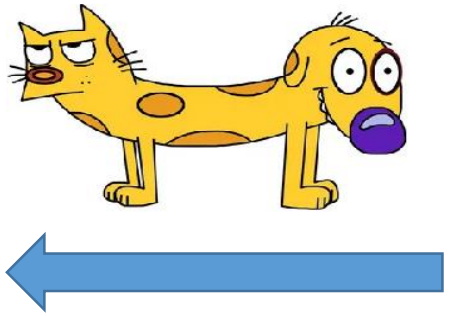
```
Animal animal = new Dog();
Cat cat = animal as Cat;  // cat = null
Dog dog = animal as Dog;  // dog = (Dog)animal
```

Casts operator methods

методы операторов приведения

```
class Dog:Animal
{
    public Dog(){
        this.Say = "Gaw";
    }

    public static implicit operator Cat(Dog dog)
    {
        Cat cat = new Cat();
        cat.Weight = dog.Weight;
        return cat;
    }
}
```

```
Dog dog = new Dog();
Cat cat = dog;
```

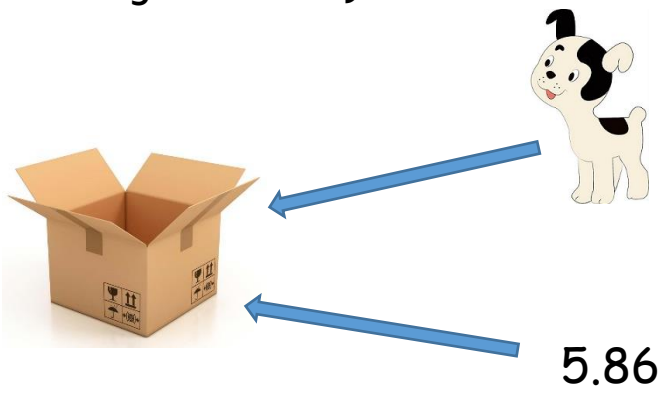


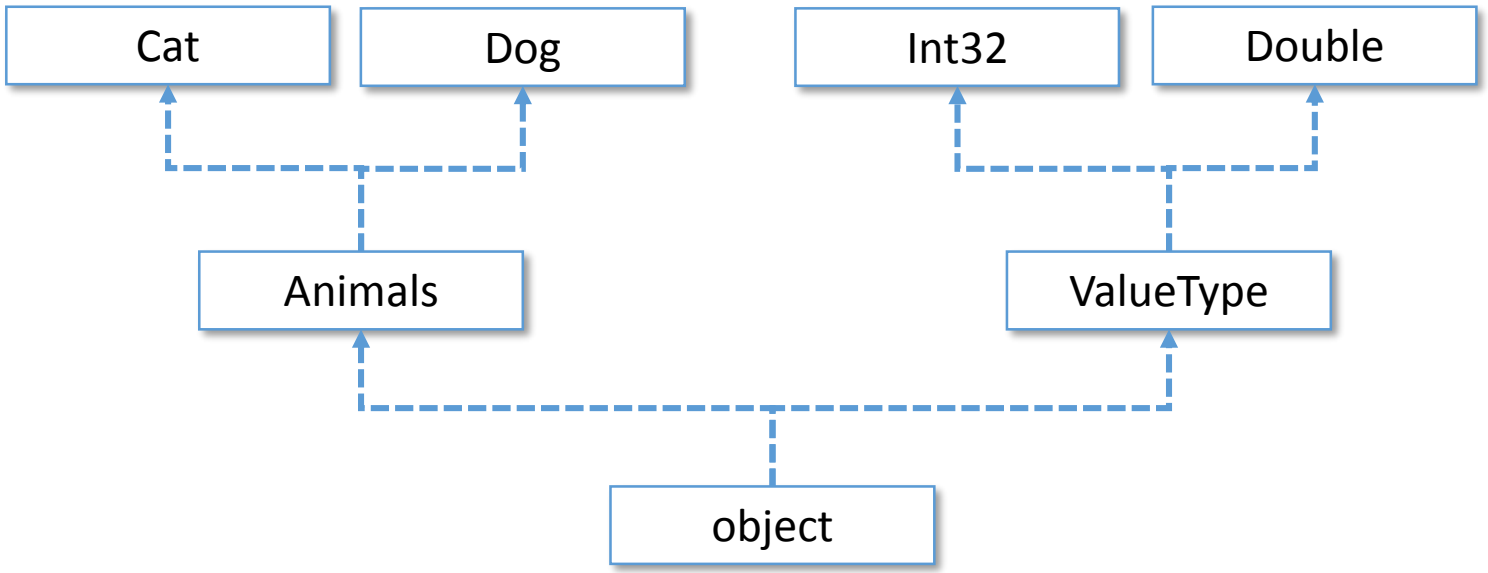
System.Object

Все типы производные от System.Object

В переменную типа `object` можно положить объект любого типа

```
object obj = null;
Dog dog = new Dog();
obj = dog;
obj = 5.86;
```





Sealed запечатывание

Запечатать класс – **запретить наследование**

```
class Animal
{
    public string Say { get; set; }
    public string Weight { get; set; }
    public void Eat() { }
}
```

```
sealed class Cat : Animal
{
    public Cat()
    {
        this.Say = "Mew";
    }
}
```

```
class Tiger : Cat
{

}
```

class ForPresentation.Tiger

'Tiger": не может быть производным от запечатанного типа "Cat".

Показать возможные решения (Alt+ВВОДилиCtrl+ю)

Operator overloading перегрузка операторов

```
sealed class Cat : Animal
{
    public static Cat operator +(Cat a, Cat b)
    {
        return new Cat();
    }
}
```

```
static void Main(string[] args)
{
    Cat cat1 = new Cat();
    Cat cat2 = new Cat();
    Cat kitten = cat1 + cat2;
}
```

Структуры (Value Types)

- Является значимым типом
- Объект, который имеет фиксированное состояние
- Имеется небольшое кол-во данных принадлежащих одной сущности.
- Не наследуется

```
struct Point
{
   readonly public int x, y;
   public Point(int x, int y)
   {
      this.x = x;
      this.y = y;
   }
}

static void Main(string[] args)
{
   Point pt = new Point(2, 5);
   Console.WriteLine(pt.x);
}
```

Все поля желательно делать неизменяемыми `readonly`

Value Types значимые типы

Примеры:

```
int
double
char
…
```

И ещё все: `enum`

Ещё примеры: System.Drawing.Rectangle
System.Drawing.Point

Value Types значимые типы

- Хранятся в стеке
- При создании объекта должны быть проинициализированы все поля
- Нельзя использовать не проинициализированный объект значимого типа
- Immutable (неизменяемый) в том смысле, что отсутствуют члены типа (методы), способные изменить состояние объекта (значения полей)

```
struct Point
{
    int x, y;
    public Point()
    {

    }
}
static void Mai
```

Point.Point()

Структуры не могут содержать явных конструкторов без параметров.

Поле "Program.Point.x" должно быть полностью определено до возврата управления в вызывающий метод.

Поле "Program.Point.y" должно быть полностью определено до возврата управления в вызывающий метод.

```
Point pt = new Point(2, 5);
pt.x = 10;
Co
```

(локальная переменная) Point pt

Присваивание значений доступному только для чтения полю допускается только в конструкторе и в инициализаторе переменных.


Спасибо за внимание!