

Лекция 1

Развитие программирования



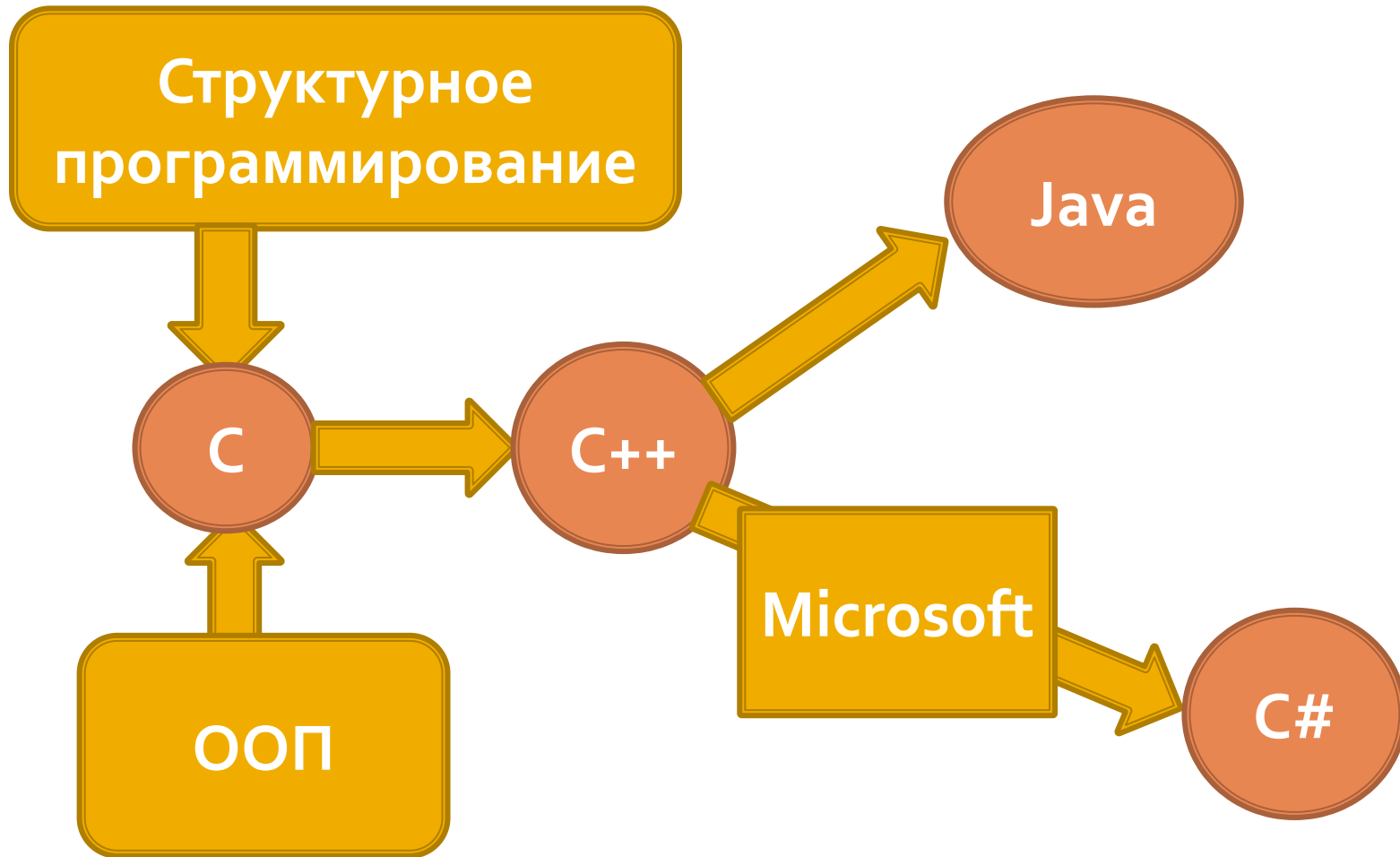
Откуда пришло

- Объектное и объектно-ориентированное программирование (ООП) возникло в результате развития идеологии процедурного (структурного) программирования, где данные и подпрограммы (процедуры, функции) их обработки формально **не связаны**.

Что есть сейчас

- В современном ООП большое значение имеют понятия
 - события (так называемое **событийно-ориентированное программирование**)
 - компонента (**компонентное программирование**).

Развитие программирования



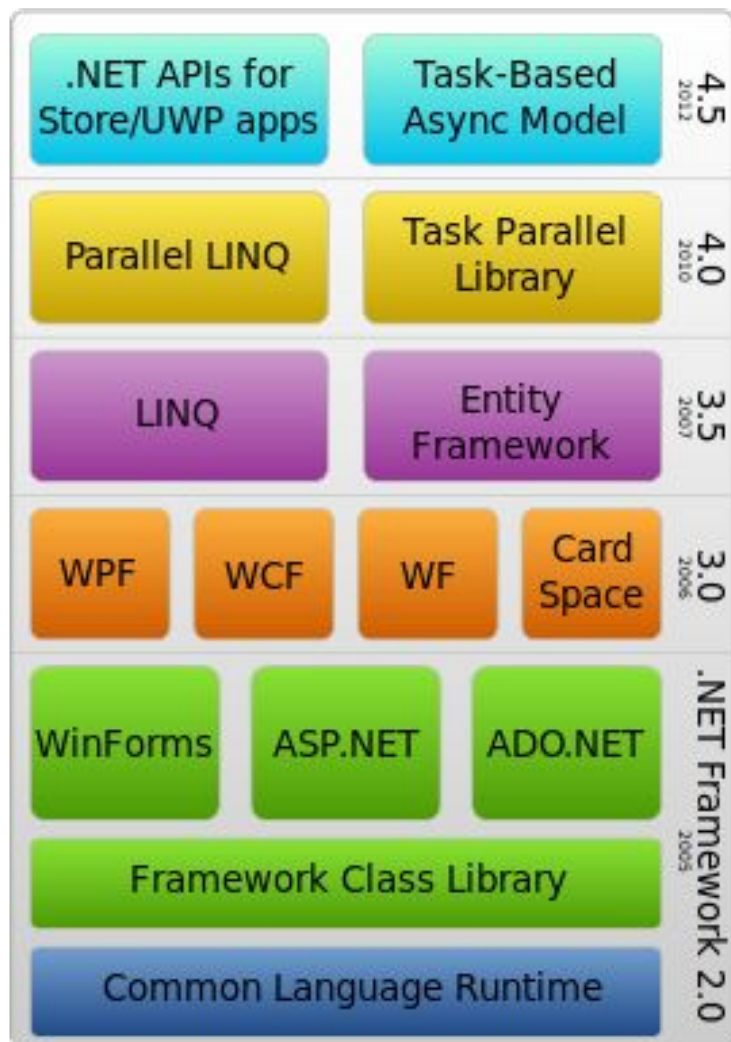
C#. История

- Язык C# появился на свет в июне 2000 г., в результате работы большой группы разработчиков компании Microsoft, возглавляемой **Андерсом Хейлсбергом** (Anders Hejlsberg).

Платформа .NET

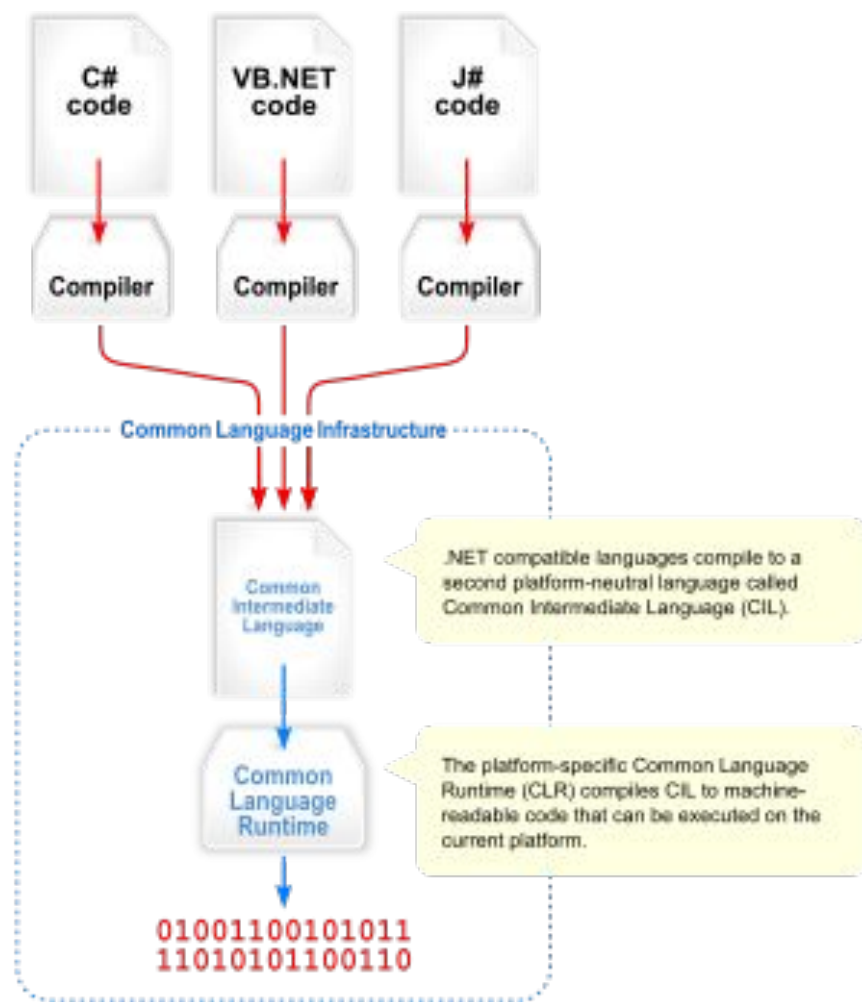
- Платформа .NET Framework состоит из **общезыковой среды выполнения** (среды CLR) и **библиотеки классов** .NET Framework.

Платформа .NET



- Платформа постоянно развивается, в ней появляются новые возможности, новые библиотеки

Среда выполнения



- исполняющая среда для байт-кода CIL (MSIL), в который компилируются программы, написанные на .NET-совместимых языках программирования

Самое главное

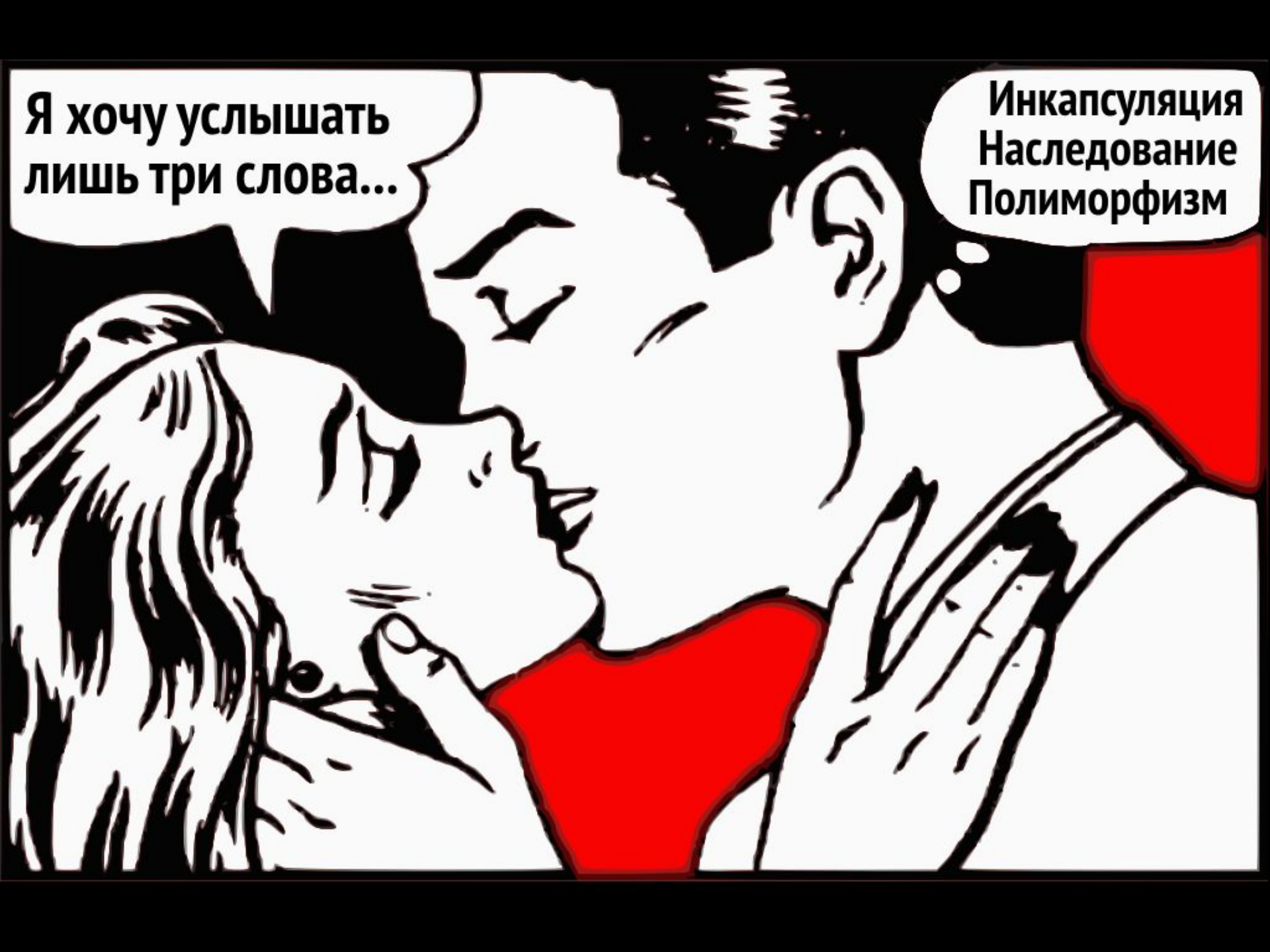
- Основным постулатом C# является высказывание: **"всякая сущность есть объект"**.
- Язык основан на **строгой компонентной архитектуре** и реализует **передовые механизмы обеспечения безопасности кода**.

ООП

Что это такое?

Определение

- **Объёктно-ориентированное программирование (ООП)** — парадигма программирования, в которой основными концепциями являются понятия **объектов** и **классов** (либо, в менее известном варианте языков с прототипированием, — прототипов)



Я хочу услышать
лишь три слова...

Инкапсуляция
Наследование
Полиморфизм

3 концепции

- Все языки ООР основаны на трёх основополагающих концепциях

ИНКАПСУЛЯЦИЯ

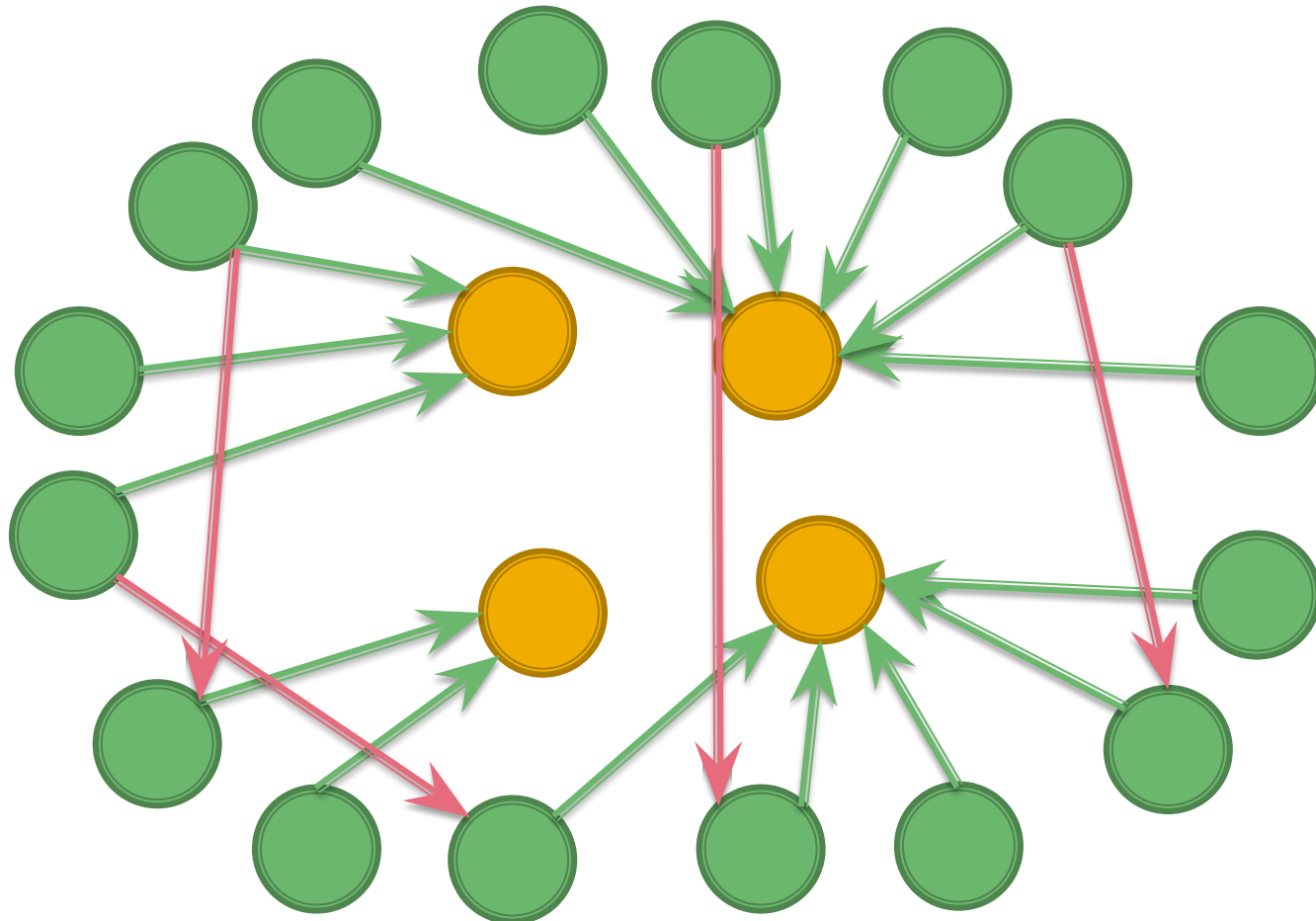
НАСЛЕДОВАНИЕ

ПОЛИМОРФИЗМ

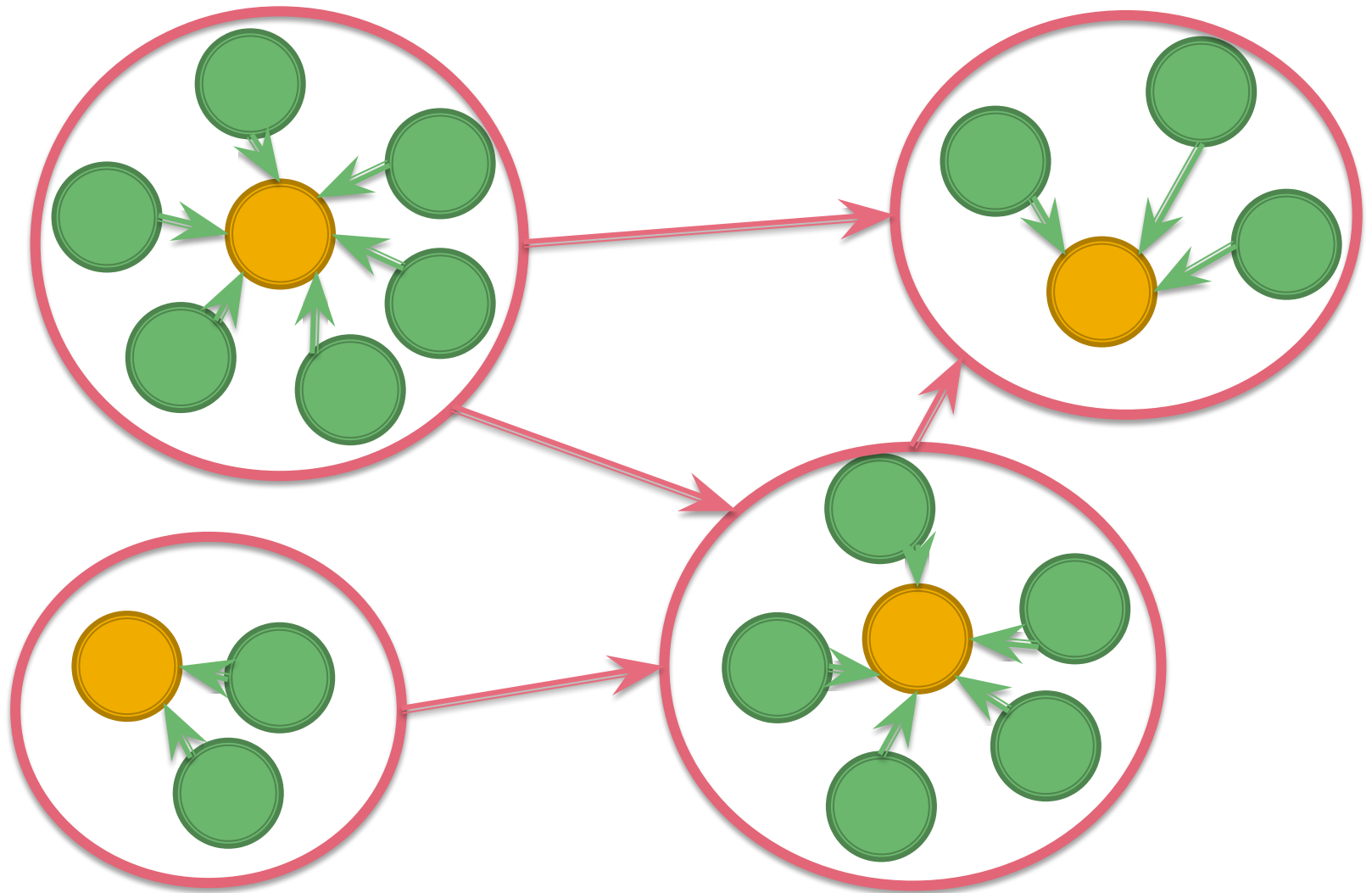
ИнКАПСУЛяция

- это механизм, который **объединяет** данные и код, манипулирующий этими данными, а также **защищает** и то, и другое **от внешнего вмешательства** или **неправильного использования**.

Инкапсуляция наглядно



Инкапсуляция наглядно



Наследование. Аналогия

КЛАССИФИКАЦИЯ ЖИВОТНОГО МИРА



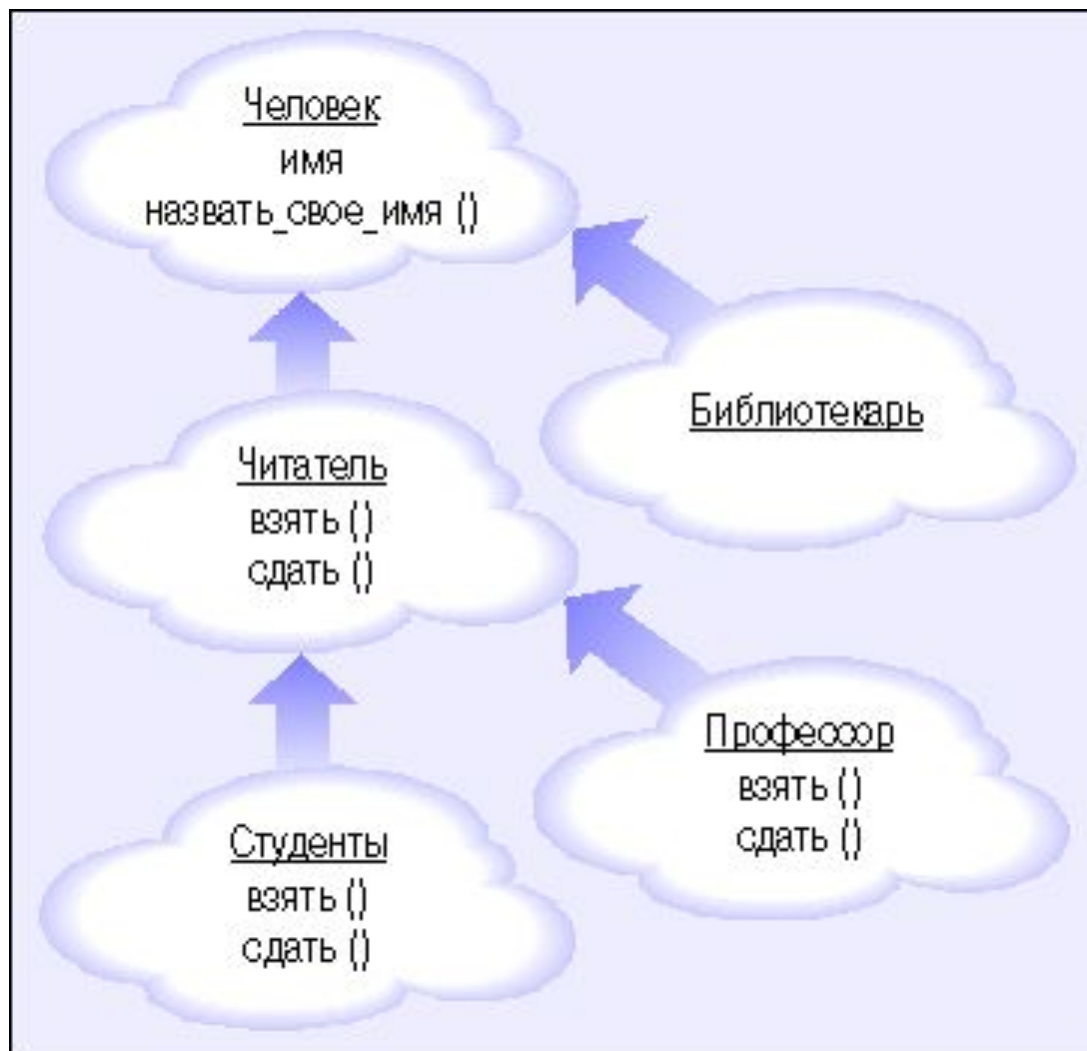
Наследование

- это процесс, посредством которого один объект **может приобретать свойства другого**. Точнее, объект может **наследовать основные свойства** другого объекта и **добавлять к ним** черты, характерные только для него.

Полиморфизм

- это свойство, которое позволяет **одно и то же имя** использовать для решения **двух или более** схожих, но технически разных задач. Целью полиморфизма, применительно к объектно-ориентированному программированию, является **использование одного имени для задания общих для класса действий**.

Полиморфизм наглядно



Абстракция

- придание объекту характеристик, которые отличают его от всех других объектов, четко определяя его концептуальные границы. Основная идея состоит в том, чтобы отделить способ использования составных объектов данных от деталей их реализации в виде более простых объектов

Абстракция

- Фундаментальная идея состоит в разделении несущественных деталей реализации подпрограммы и характеристик существенных для корректного ее использования. Такое разделение может быть выражено через специальный «**интерфейс**», сосредотачивающий описание всех возможных применений программы

Классы и объекты

Классы

- это элемент, описывающий абстрактный тип данных и его частичную или полную реализацию.
- Наряду с понятием «**объекта**» класс является ключевым понятием в ООП.

Классы

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApplication1
{
    0 references
    class Class1
    {
    }
}
```

Объект класса

- это переменная от класса
- `Class1 cl = new Class1();`
- `<Название класса> <имя переменной> = new <Название класса>(<параметры>);`

Что есть у классов

- Поля
- Константы
- Свойства
- Методы
- Конструкторы и деструктор
- События
- Индексаторы
- Операторы
- Вложенные типы

Поля



Поля

- Поля инициализируются непосредственно перед вызовом конструктора для экземпляра объекта.
- Поля могут быть отмечены модификаторами доступа
- Также при необходимости поле может быть объявлено с модификатором **static**.

Константы

- Константы представляют собой **неизменные значения**, известные во время компиляции и неизменяемые на протяжении времени существования программы. Константы объявляются с модификатором **const**. Только встроенные типы C# (за исключением System.Object) могут быть объявлены как const.

Константы

- Используемое для инициализации константы выражение может **ссылаться на другую константу**, если при этом не создается циклическая ссылка

2 references

```
class Class1
{
    int x;

    string s;

    DateTime date;

    const int c = 40;

    const char cd = 'd';
}
```

Свойства

- это член, предоставляющий гибкий механизм для чтения, записи или вычисления значения **закрытых** полей. Свойства фактически представляют собой специальные методы, называемые **методами доступа**. Это позволяет легко получать доступ к данным и помогает **повысить безопасность и гибкость методов**.

Свойства

- У свойства могут быть два ключевых слова **set** и **get**

```
class Class1
{
    private int x = -1;

    0 references
    public int X
    {
        set { if (value > 0 && value < 10) this.x = value; }
        get { if (this.x > -1) return this.x; else return 100; }
    }
}
```

```
public class Class2
```

```
{
```

```
    0 references
```

```
    public void method()
```

```
    {
```

```
        Class1 c1 = new Class1();
```

```
        c1.X = 4;
```

```
    }
```

```
}
```

Свойства. Особенности

0 references

```
public int X
{
    get { if (this.x > -1) return this.x; else return 100; }
}
```

0 references

```
public int X
{
    set { if (value > 0 && value < 10) this.x = value; }
}
```

0 references

```
public int X { set; get; }
```

Методы

- это блок кода, содержащий ряд инструкций. Программа инициирует выполнение инструкций, вызывая метод и указывая все аргументы, необходимые для этого метода. В С# **все инструкции выполняются в контексте метода.**

Методы

```
class Class1
{
    0 references
    void method() { }

    0 references
    int method1(int f) { return f + 4; }

    0 references
    string method3(int sd) { return "№" + sd; }
}
```

Сигнатура метода

- Методы объявляются в классе или в структуре путем указания **модификаторов доступа, необязательных модификаторов**, (abstract или sealed), **возвращаемого значения, имени метода** и **всех параметров** этого метода. Все эти части вместе представляют собой **сигнатуру метода**.

Вложенные типы

- Тип, определенный внутри класса, называется вложенным типом

```
class Class1
{
    0 references
    class Class2
    {
        int et = 6;
    }
}
```

Модификаторы доступа

- public
- protected
- internal
- protected internal
- **private (по умолчанию)**

```
public class Class1 {...}
```

Зачем они? public

- Общий (**public**) доступ является уровнем доступа с максимальными правами. Ограничений доступа к общим членам **не** существует.

public. Пример

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;
```

```
namespace ConsoleApplication1
```

```
{  
    public class Class1  
    {  
        public int x;  
  
        public void method()  
        {  
            this.x = 6;  
        }  
    }  
}
```

```
namespace ConsoleApplication2
```

```
{
```

Зачем они? `private`

- Закрытый (**`private`**) доступ является уровнем доступа с минимальными правами. **Доступ** к закрытым членам можно получить только **внутри** тела **класса**, в которой они объявлены.

private. Пример

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;
```

```
namespace ConsoleApplication1
```

```
{  
    public class Class1  
    {  
        private int x;  
  
        public void method()  
        {  
            this.x = 6;  
        }  
    }  
}
```

```
namespace ConsoleApplication2
```

```
{
```

Типы данных

Ссылочные
(классы, массивы,
интерфейсы,
делегаты)

типы-значения
(элементарные типы,
перечисления,
структуры)

Типы-значения

- Типы значений состоят из двух основных категорий:
 - Структуры
 - Перечисления

Структурные типы

- Структуры делятся на следующие категории:
 - Числовые типы
 - Целочисленные типы
 - Типы с плавающей запятой
 - decimal
 - bool
 - Структуры, определяемые пользователем.

Перечисления

- Перечисление объявляется с помощью ключевого слова **enum**, идентифицируется по имени и представляет собой непустой список неизменяемых именованных значений интегрального типа.

Перечисления

```
enum Colors { Red = 1, Green = 2, Blue = 4,  
Yellow = 8 };
```

```
int xVal = (int)Colors.Red;
```

```
Colors t = Green;
```

Упаковка и распаковка

- Упаковка представляет собой процесс преобразования типа значения в тип **object** или в любой другой тип интерфейса, реализуемый этим типом значения.
- Когда тип значения упаковывается средой CLR, она создает оболочку значения внутри `System.Object` и сохраняет ее в управляемой куче.

Упаковка и распаковка

- По сравнению с простыми операциями присваивания операции упаковки и распаковки являются весьма **затратными** процессами с точки зрения вычислений. При выполнении упаковки типа значения необходимо создать и разместить новый объект.

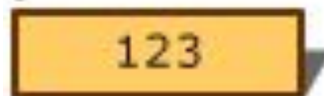
Boxing и unboxing

- Упаковка используется для хранения типов значений в куче со сбором мусора.

```
int i = 123;  
object o = i;
```

В стеке

i



int i=123;

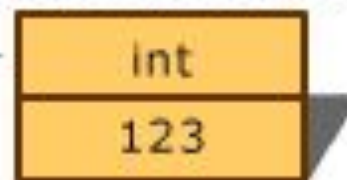
В куче

o



object o=i;

(упакованная i)



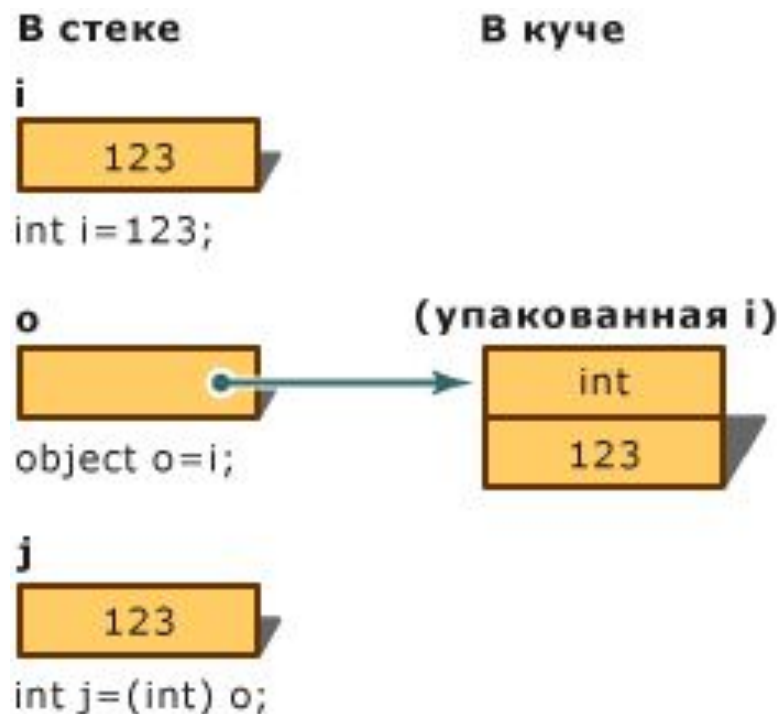
Boxing и unboxing

- Распаковка является явным преобразованием из типа object в тип значения.

```
int i = 123;
```

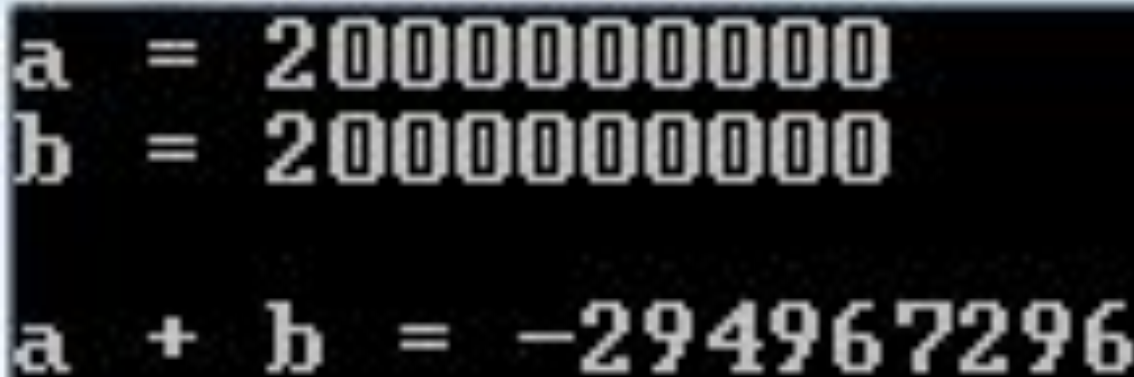
```
object o = i;
```

```
int j = (int)o;
```



Переполнение

```
int a = 2000000000;  
int b = 2000000000;  
Console.WriteLine("a = {0}\nb = {1}\n", a, b);  
Console.WriteLine("a + b = {0}", a + b);
```

A screenshot of a console window with a black background and white text. It displays the output of the C# code above. The first two lines show 'a = 2000000000' and 'b = 2000000000'. The third line shows 'a + b = -294967296', which is the result of integer overflow where the sum wraps around from the maximum positive value to a negative value.

```
a = 2000000000  
b = 2000000000  
  
a + b = -294967296
```

Переполнение

- Причиной некорректных результатов выполнения арифметических операций является особенность представления значений арифметических типов.
- Арифметические типы имеют ограниченные размеры. Поэтому любая арифметическая операция может привести к **переполнению**.

checked и unchecked

```
short x = 32767;
short y = 32767;
short z = 0;
try
{
    z = checked(x + unchecked(x+y));
}
catch (System.OverflowException e)
{
    Console.WriteLine("Переполнение при выполнении
сложения");
}
return z;
```

checked и unchecked

unchecked

{

W = X+y;

}

checked

{

Z = X+W;

}

Область видимости

- Переменные можно объявлять в любом месте блока. Точка объявления переменной в буквальном смысле соответствует месту ее создания.
- **Новый блок – новая область видимости.** Объекты, объявляемые во **внутренних блоках, не видны во внешних блоках.**
- Блок ограничивается { }

Область видимости

- Объекты, объявленные в методе и во внешних блоках, видны и во внутренних блоках.
- Одноименные объекты во вложенных областях конфликтуют.
- Объекты, объявляемые в блоках одного уровня вложенности в методе, не видны друг для друга. Конфликта имен не происходит.

Пример

```
{  
    int a = 2;  
    {  
        int a = 6;  
    }  
    Console.ReadKey();  
}
```

```
{  
    {  
        int a = 2;  
    }  
    {  
        int a = 6;  
    }  
    Console.ReadKey();  
}
```

Объявление и инициализация

- В чем разница?
- `int a;`
- `int a = 8;`
- `Console.WriteLine(a.ToString());`

Приоритет операций

1	() [] . (постфикс)++ (постфикс)— new sizeof typeof unchecked
2	! ~ (имя типа) +(унарный) -(унарный) ++(префикс) —(префикс)
3	* / %
4	+ -
5	<< >>
6	< > <= >= is
7	== !=
8	&
9	^
10	
11	&&
12	
13	?:
14	= += -= *= /= %= &= = ^= <<= >>=

Привидение типов

- Используемые в программе типы характеризуются собственными диапазонами значений, которые определяются свойствами типов – в том числе и размером области памяти, предназначенной для кодирования значений соответствующего типа.

Привидение типов

- `int a = 10;`
- `short d = 30;`
- `long l = 40005;`
- `int df = a + d + l;`
- `System.Convert`