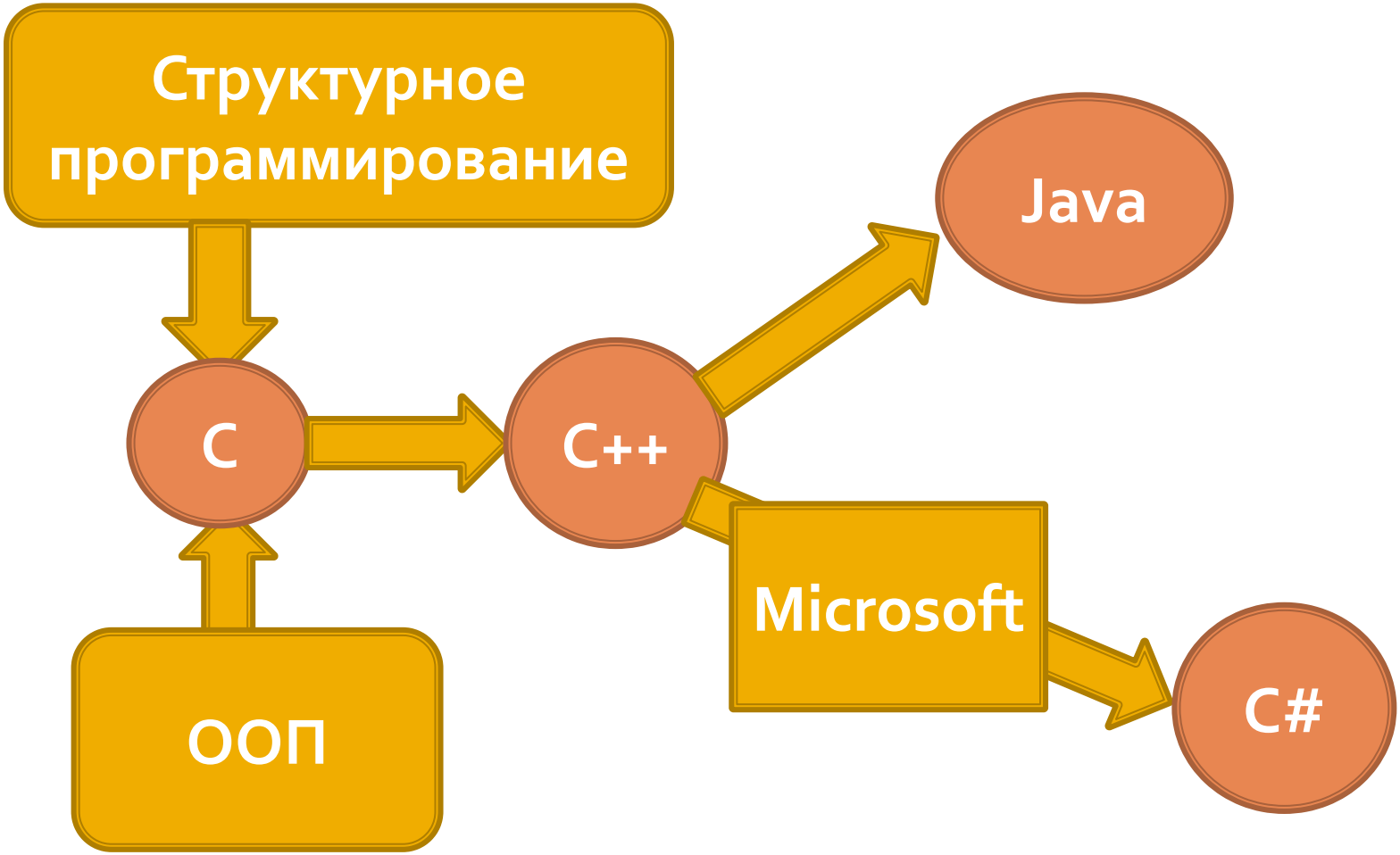
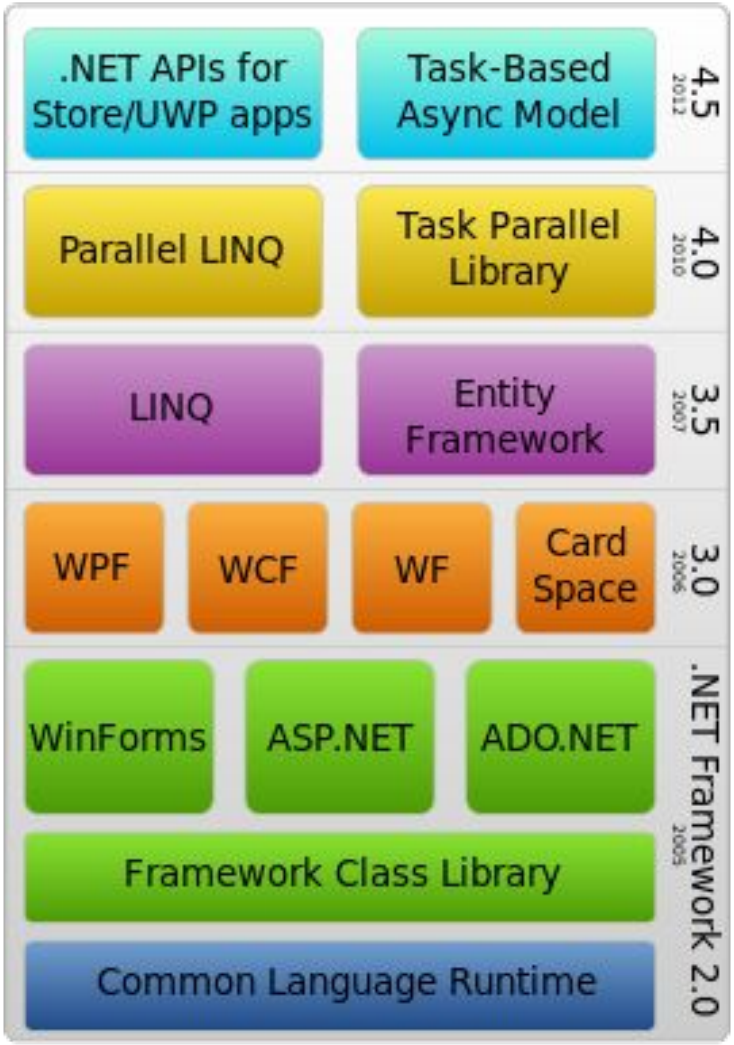
Лекция 1

Развитие программирования

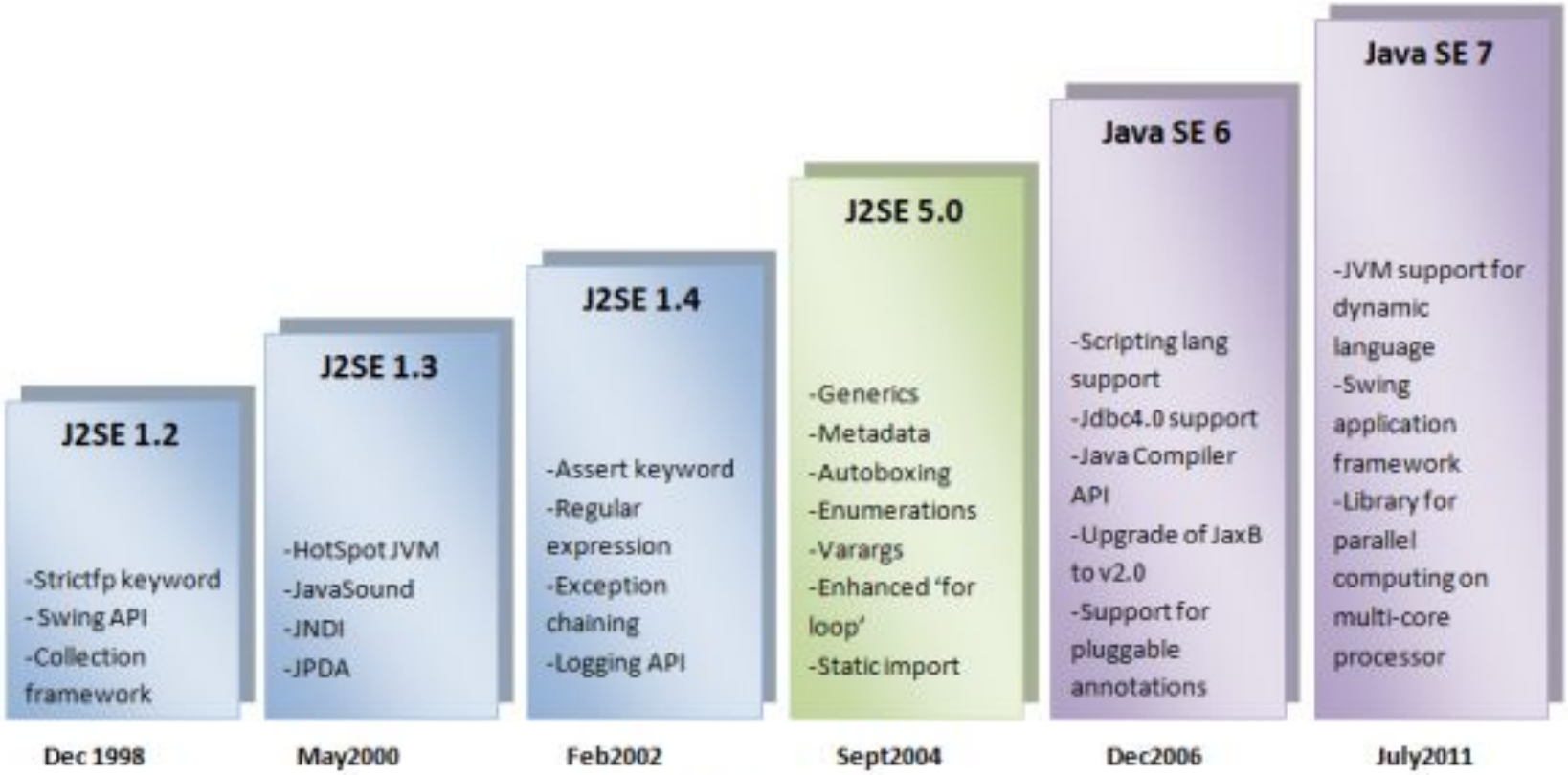


Платформа .NET



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

Java



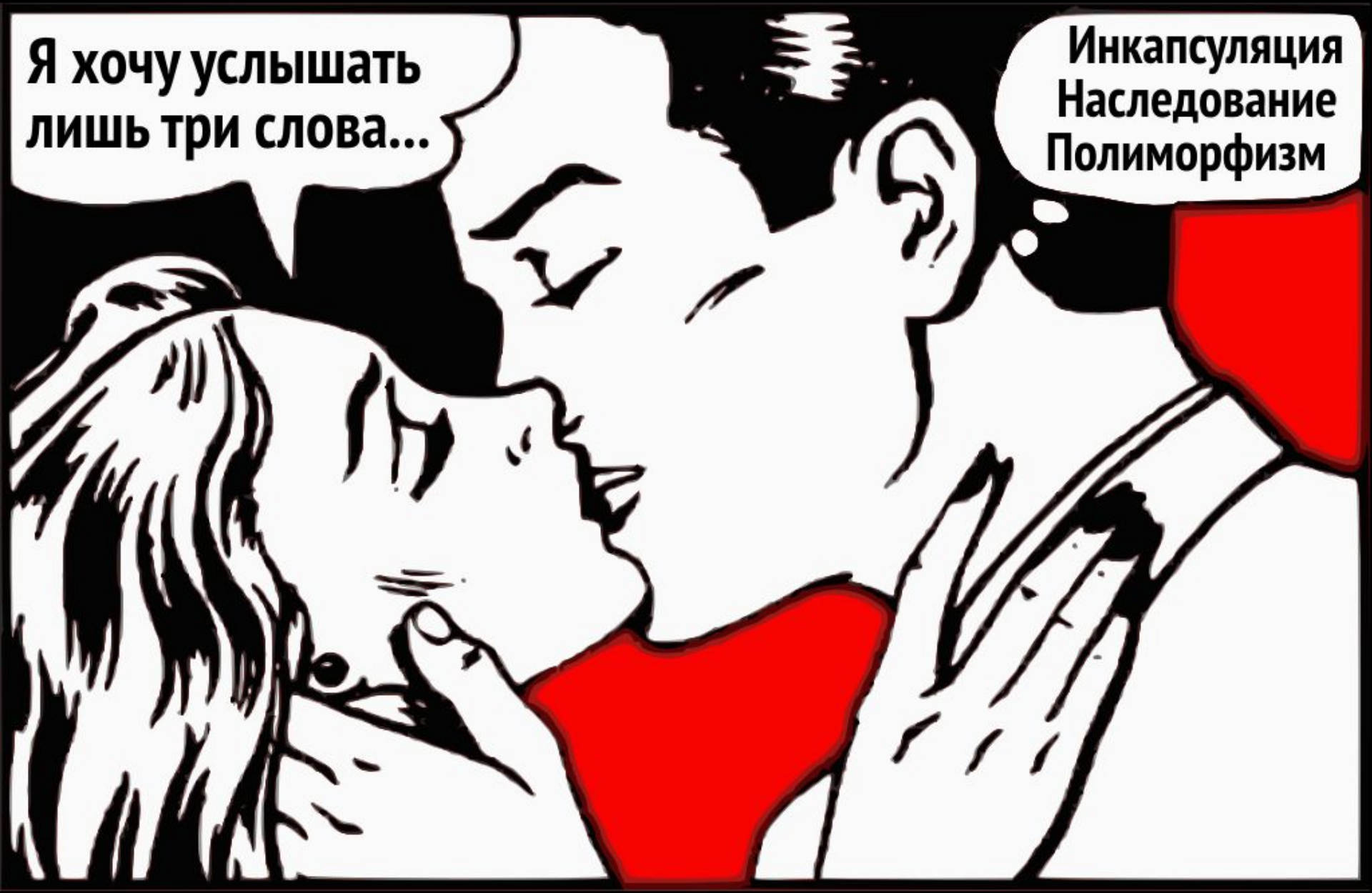
Java Standard Edition releases

ООП

Что это такое?

Определение

- **Объе́ктно-ориенти́рованное программи́рование** (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 OOP основаны на трёх основополагающих концепциях

ИНКАПСУЛЯЦИЯ

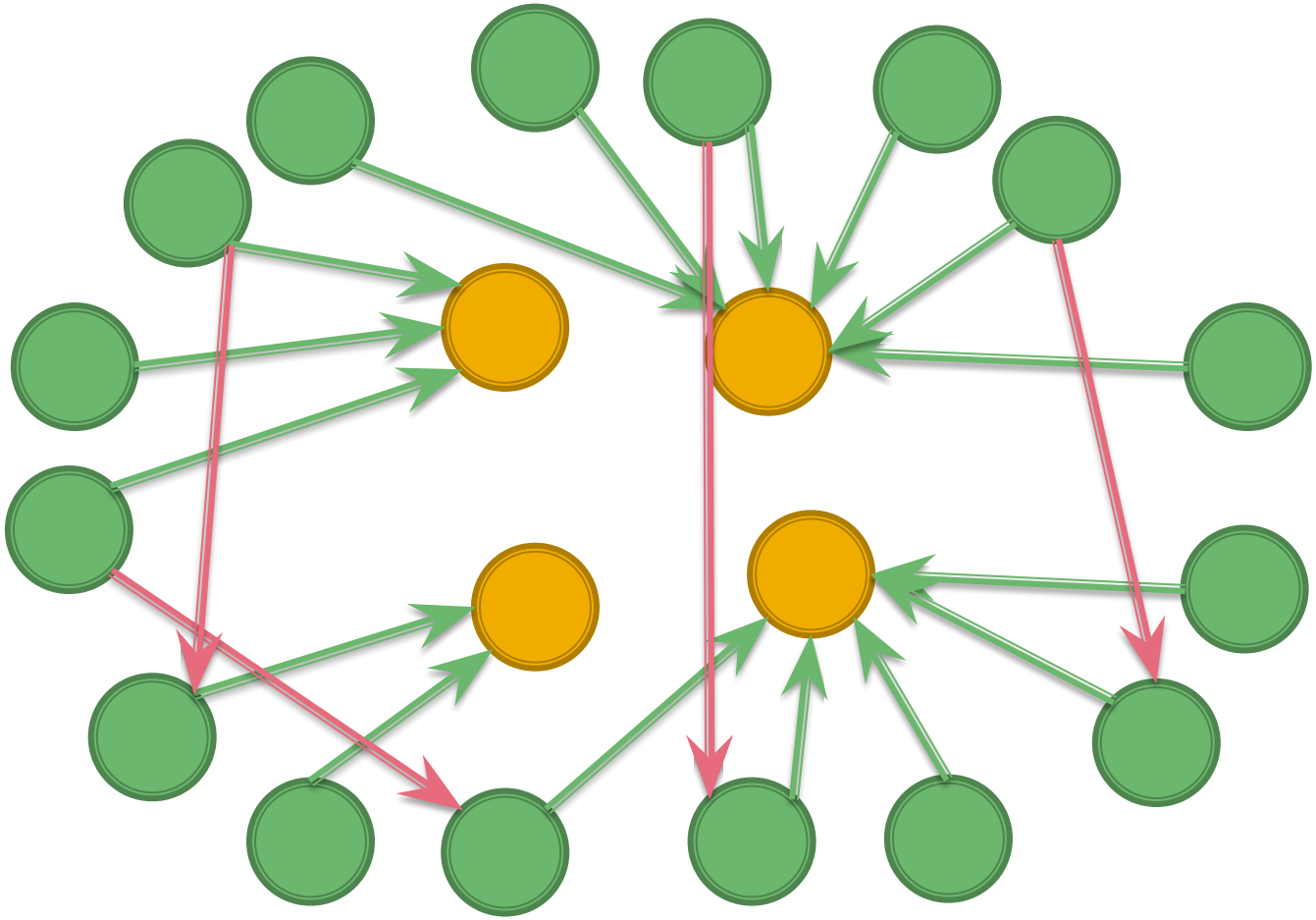
НАСЛЕДОВАНИЕ

ПОЛИМОРФИЗМ

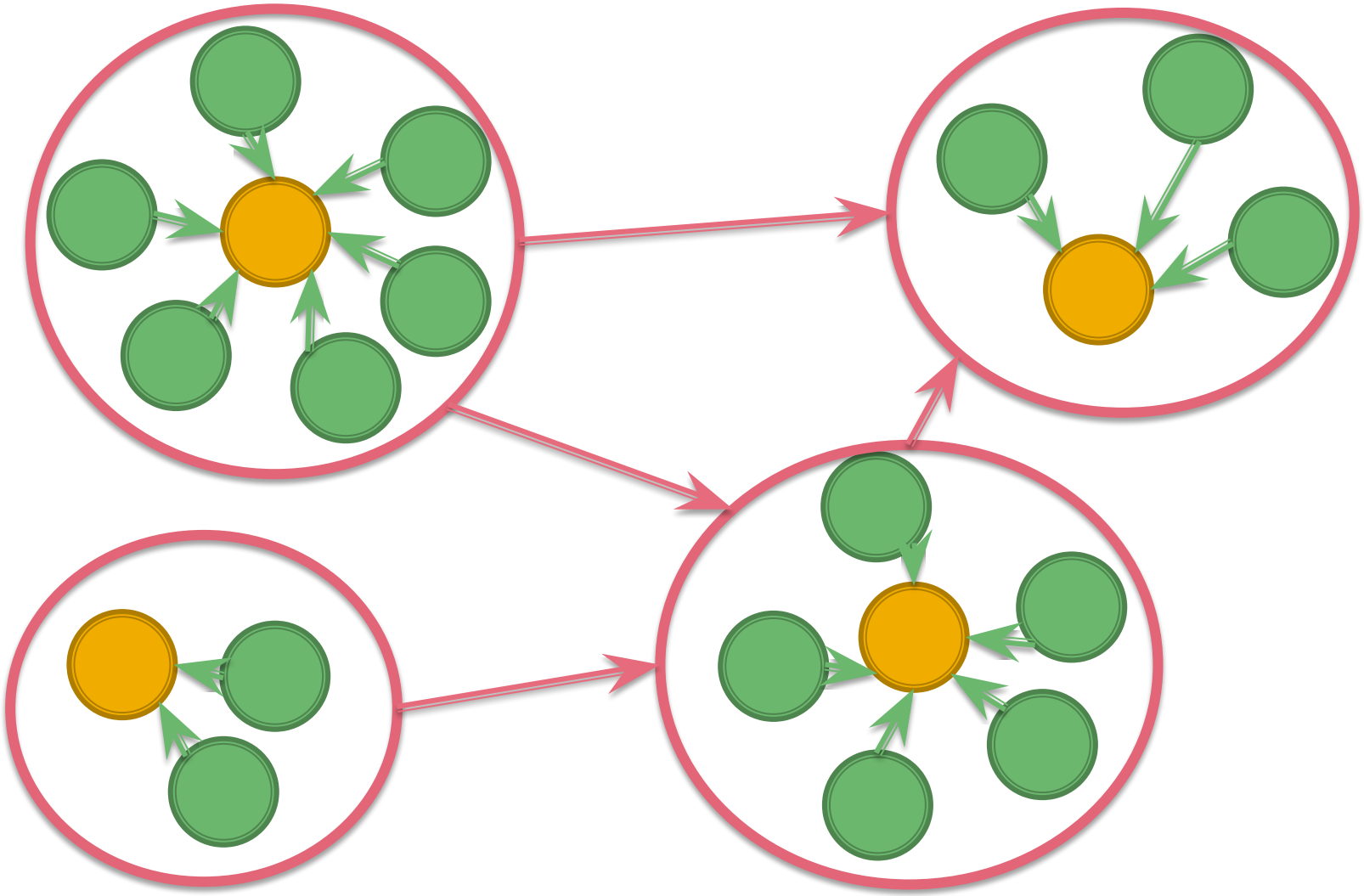
ИнКАПСУЛяция

- 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



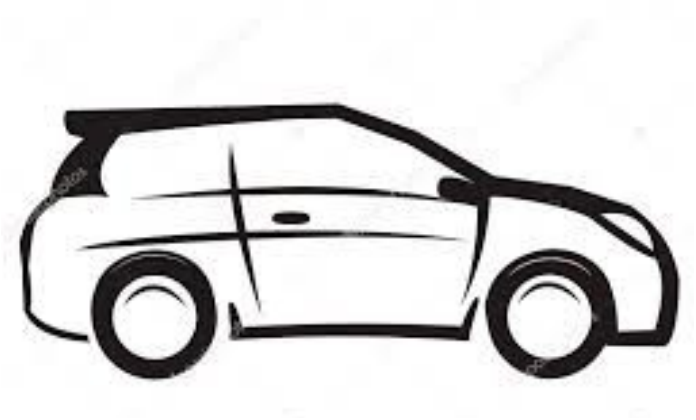
Абстракция

- 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

Абстракция



Автосалон

Страховая компания

Владелец

ГИБДД

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

```
namespace ConsoleAppLection01
{
    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**.

Поля

```
class Class1
{
    int _field;

    double field2;

    char ch_first;
}
```

Константы

- Константы предназначены для описания таких значений, которые не должны изменяться в программе. Для определения констант используется ключевое слово const:

```
class Class1
{
    hide

    const int arraySize = 10;
}
```

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
{
    hide

    private int x = -1;

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

Свойства. Пример

```
class Program
{
    static void Main(string[] args)
    {
        Class1 cl = new Class1();

        cl.x = 5;

        cl.X = 5;
    }
}
```

Свойства. Особенности

```
class Class1
{
    hide

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

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

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

Методы

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

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.

Методы

```
class Class1
{
    hide

    void Method() { }

    int Method1(int f) { return f + x; }

    string Method3(int sd) { return "№" + sd + ": " + x; }
}
```

Модификатор out и ref

```
class Class1
{
    hide

    void Sum(int x, int y, out int a)
    {
        a = x + y + this.x;
    }

    void Addition(ref int x, int y)
    {
        this.x = y;

        if (this.x > 50) { x = this.x; }
    }
}
```

Необязательные параметры

- C# позволяет использовать необязательные параметры. Для таких параметров нам необходимо объявить значение по умолчанию.

```
public int Func(int x, int y, int z = 5, int s = 4)
{
    return x + y + z + s;
}
```

Именованные параметры

- Именованные параметры позволяют нарушать порядок передачи параметров в функцию.

```
static void Main(string[] args)
{
    Class1 cl = new Class1();

    cl.Func(y: 2, x: 3, s: 7);
}
```

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**.

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.

Пример

```
public class Class1
{
    hide

    private int x = -1;

    public int Func(int x, int y, int z = 5, int s = 4)...
}
```

```
static void Main(string[] args)
{
    Class1 cl = new Class1();

    cl.Func(y: 2, x: 3, s: 7);

    cl.x = 45;
}
```

Типы данных

Типы данных

Ссылочные (классы, массивы, интерфейсы, делегаты)

типы-значения (элементарные типы, перечисления, структуры)

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 };

class Program
{

    static void Main(string[] args)
    {
        int xVal = (int)Colors.Red;

        Colors t = Colors.Green;
    }
}
```

Структурные типы

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

Типы данных

Тип	Значения	Разрядность
bool	true или false	
byte	целое 0…255	1
sbite	целое -128…127	1
short	целое -32768…32767	2
usort	целое 0…65535	2
int	целое -2147483648…2147483647	4
uint	целое 0…4294967295	4
long	целое -9223372036854775808…9223372036854775807	8
ulong	целое 0…18446744073709551615	8
float	-3.4*10^38…3.4*10^38 с плавающей точкой	4
double	+-5*10^324…+-1.7*10^308 с плавающей точкой	8
decimal	0…+-79228162514264337593543950335 – целый 0…+-7,9228162514264337593543950335 – (28 разр.)	16
char	Одиночный символ Unicode	2
string	Набор символов Unicode	
object	Значение любого типа данных	8(x86)/16(x64)

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

Упаковка и распаковка

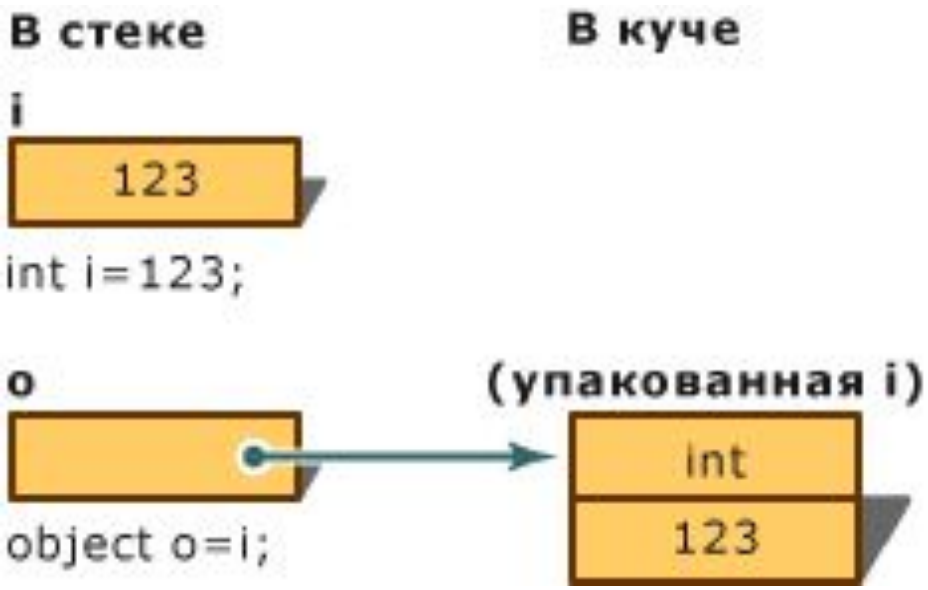
- Упаковка представляет собой процесс преобразования типа значения в тип **object** или в любой другой тип интерфейса, реализуемый этим типом значения.
- Когда тип значения упаковывается средой CLR, она создает оболочку значения внутри System.Object и сохраняет ее в управляемой куче.

Boxing и unboxing

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

  int i = 123;

  object o = i;



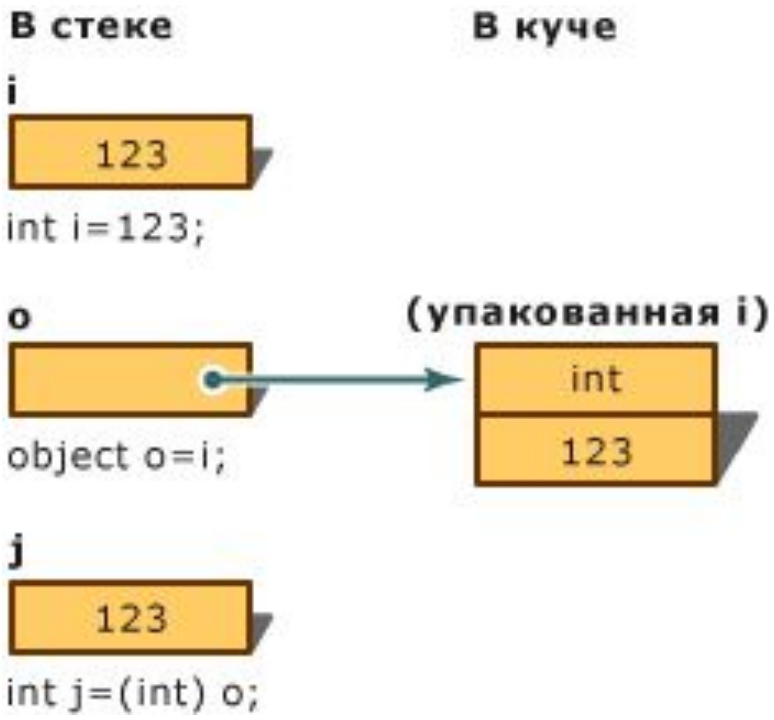
Boxing и unboxing

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

  int i = 123;

  object o = i;

  int j = (int)o;



Это важно

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());

Переполнение

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

```
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

```
int x = 2000000000;
int y = 2000000000;
int z = 0;
try
{
    z = checked(x + unchecked(x + y));
}
catch (OverflowException e)
{
    Console.WriteLine("Переполнение при выполнении сложения");
}
return z;
```

checked и unchecked

```
unchecked
{
   w = x+y;
}
checked
{
   z = x+w;
}
```

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	= += –= *= /= %= &= |= ^= <<= >>=