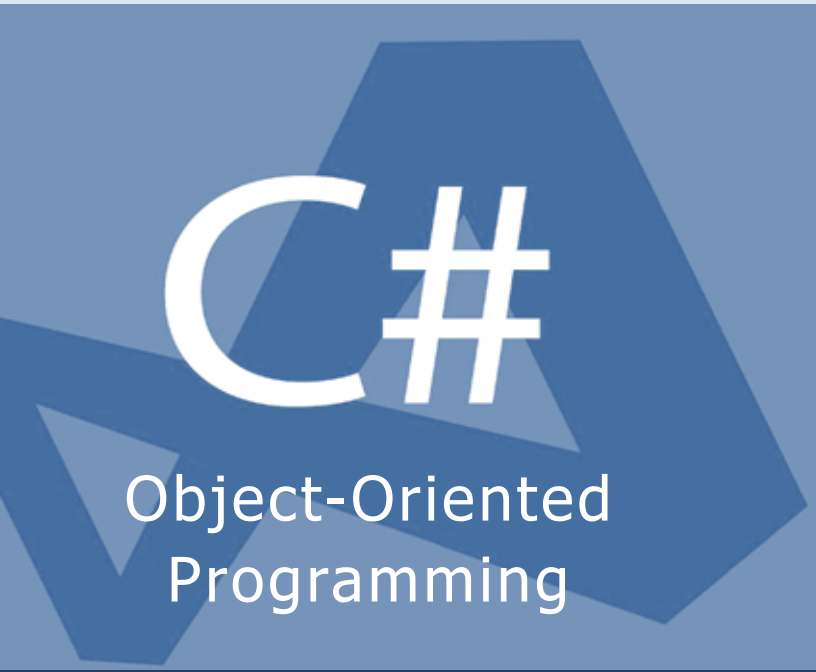


Object-Oriented Programming

1

Getting Started

д.т.н. Емельянов Виталий Александрович

: v.yemelyanov@gmail.com

Тенденции разработки ПО

- **переход от расчетов по формулам к сложным задачам моделирования систем;**
- **увеличение объемов обрабатываемых данных;**
- **повышение сложности программ**, увеличение их длины (до миллионов строк кода!).

Сложность программ превышает возможности одного человека (программиста).

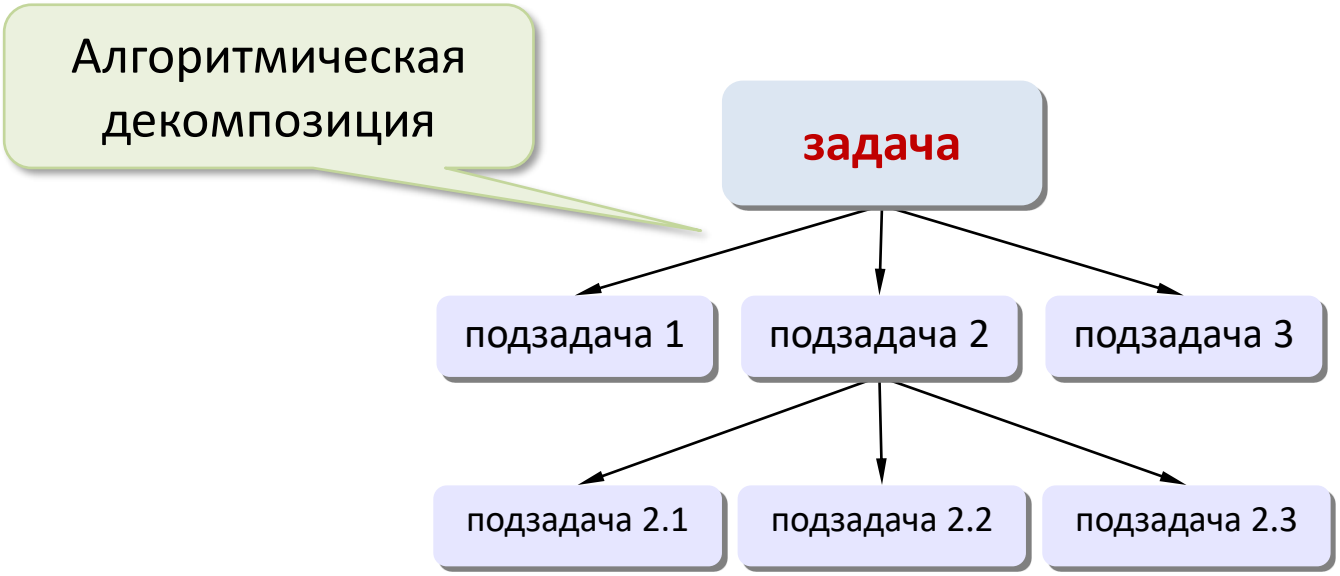
В итоге:

- коллективная разработка
- каждый делает свою часть независимо от других
- части программы легко «собрать» вместе

Как?

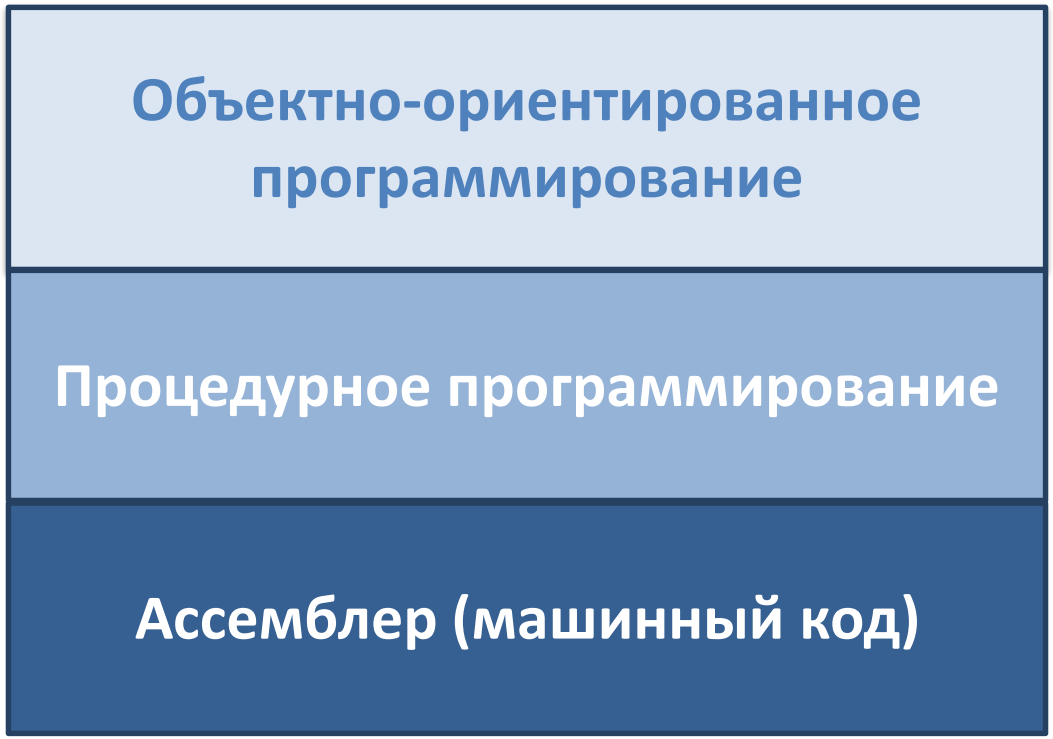
Тенденции разработки ПО

Структурное и процедурное программирование:



ПРОБЛЕМА: **«Люди воспринимают мир, состоящий из объектов» (Р. Декарт).**

Парадигмы программирования



Как человеку проще мыслить

Как человеку проще записывать действия

Что понимает компьютер

Объектно-ориентированное программирование (ООП)

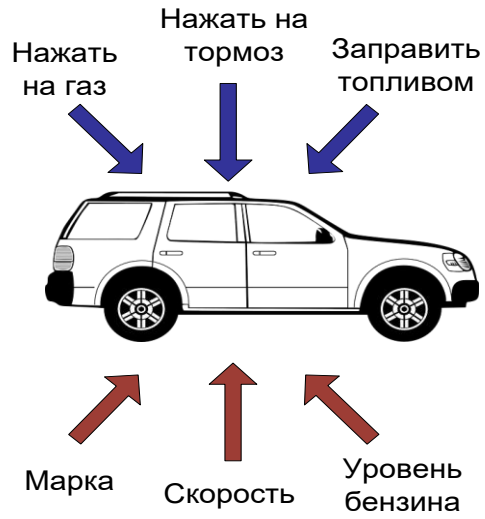
ООП – это парадигма программирования, в которой базовым является понятие объекта.

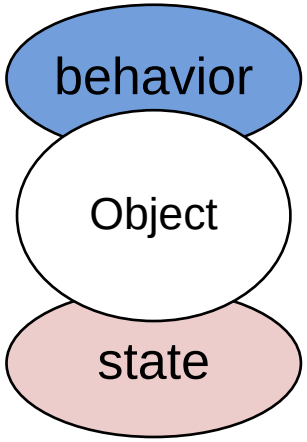
Программа в ходе работы представляет собой набор взаимодействующих объектов

Объект в программе – это абстракция реального объекта

Абстракция – это выделение существенных свойств объекта, отличающих его от других объектов

Понятие объекта





Объект реального мира:

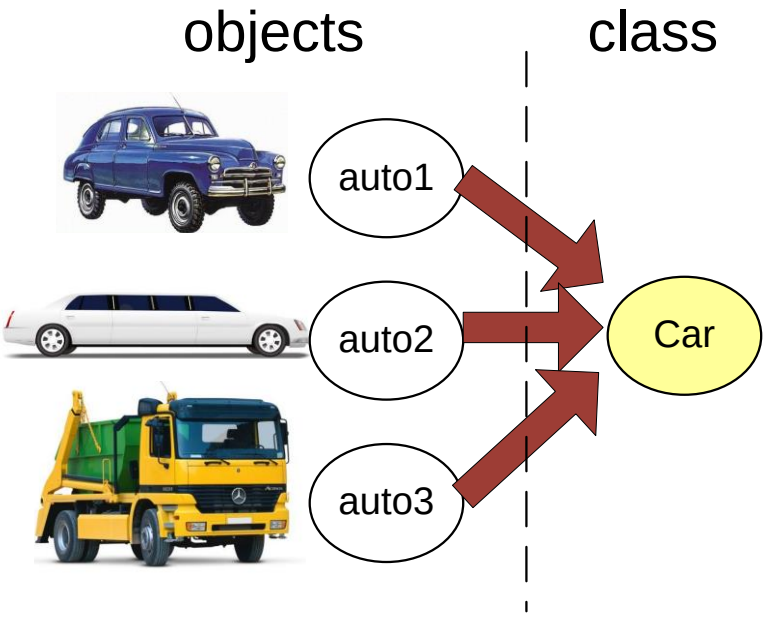
- Имя (Название)
- Состояние (state)
- Линия поведения (behavior)

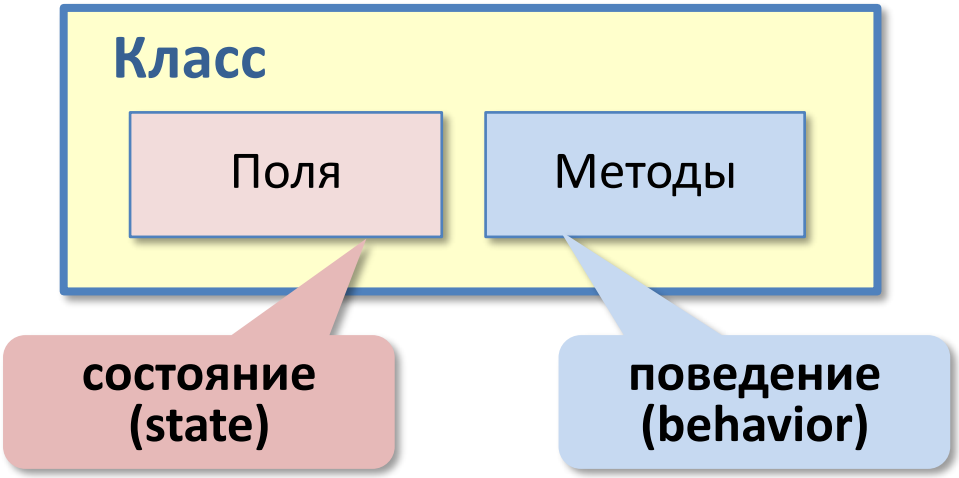
Программный объект:

- Имя (Идентификатор)
- Поля (fields)
- Методы (methods)

Понятие класса

- **Класс (class)** описывает признаки состояния и поведение множества схожих объектов.
- **Класс** – это пользовательский ***тип данных***





Создание класса

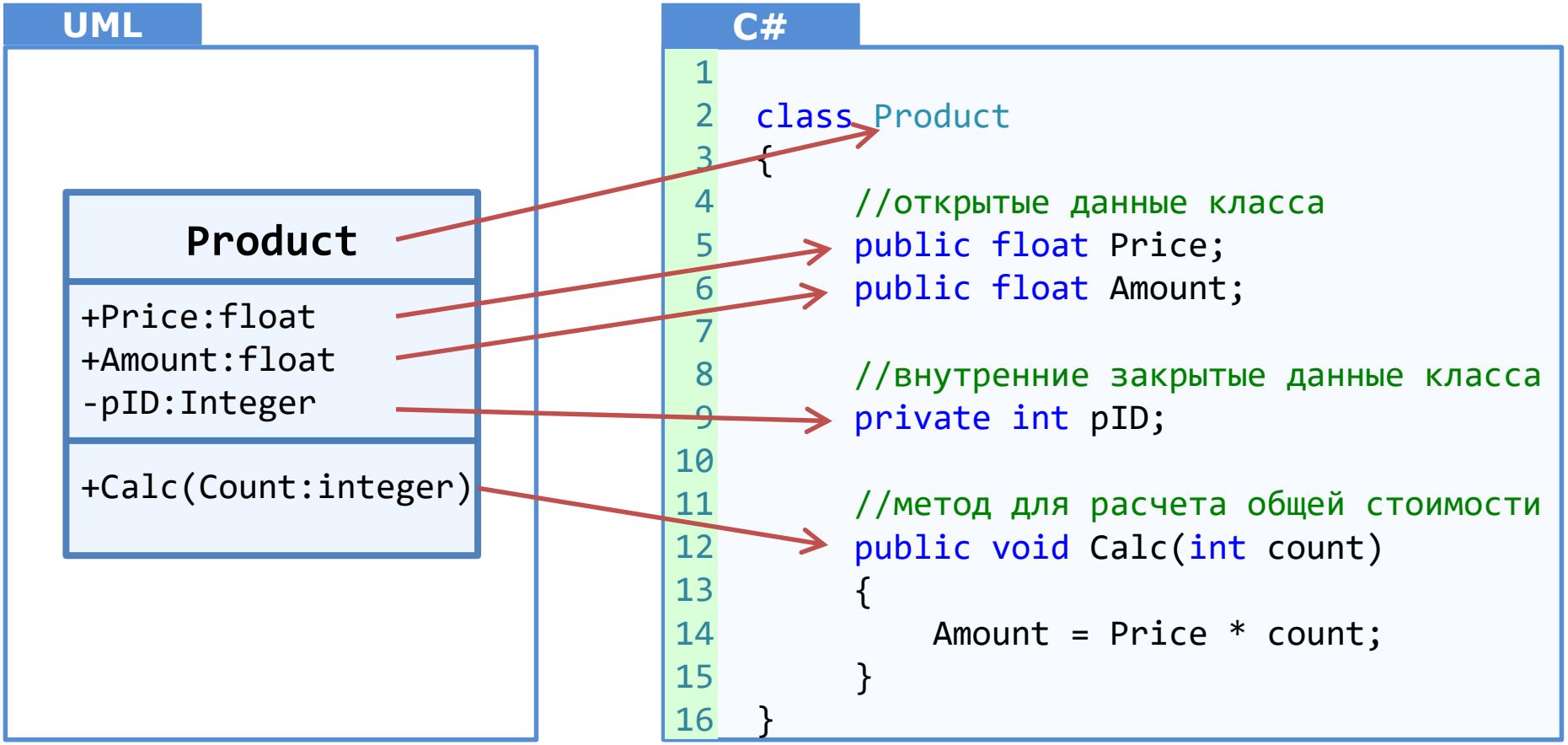
Класс на C# описывается следующим образом:

C#

```
[модификаторы]  class  имя_класса  [ : предки]
        {
            //тело класса (поля и методы)
        }
```

Доступность элементов / Модификатор:	Class (базовый класс)	Subclass (класс-наследник)	World
public	Да	Да	Да
protected	Да	Да	Нет
private	Да	Нет	Нет

Создание класса



Создание объекта класса (экземпляра класса)

- **Объекты (экземпляры классов)** создаются с помощью оператора new.
- Для получения доступа к данным объекта или для вызова методов, используется оператор точка «.»
- При создании объекта, копия данных, описываемых классом записывается в память и присваивается переменной ссылочного типа.

C#

```
имя_Класса имя_ссылки;           //объявление ссылки
имя_ссылки = new имя_Класса(); //создание объекта класса

имя_ссылки.имяПоля = 0;          //обращение к полю
имя_ссылки.имяМетода();          //вызов метода
```

Создание объекта класса (экземпляра класса)

C#

```
17  static void Main(string[] args)
18  {
19      int number=0;
20
21      Product orderedProduct = new Product(); /* создание объекта с именем
22                                                 orderedProduct класса Product*/
23      orderedProduct.Price = 10;     /* обращение к полю Price
24                                        и задание цены за единицу товара=10 */
25
26      Console.WriteLine("Введите количество товара в шт.: ");
27      number = Convert.ToInt32(Console.ReadLine());
28
29      orderedProduct.Calc(number);  /* вызов метода Calc() класса Product и
30                                       передача параметра number */
31
32      Console.WriteLine("Стоимость товара="+orderedProduct.Amount);
33      Console.ReadKey();
34  }
35
```

Конструктор

Конструктор – особый вид метода в классе

По смыслу:

Вызывается автоматически при каждом создании объекта. Типовое применение: установка полей в заданное значение

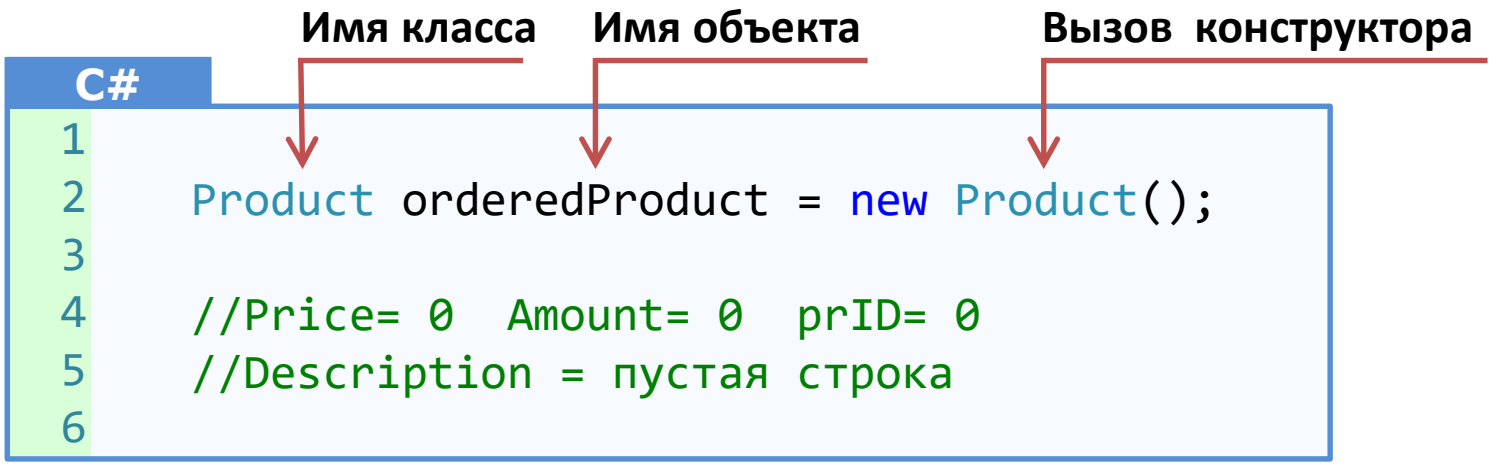
По синтаксису:

- **Имя должно совпадать с именем класса**
- **Не имеет возвращаемого объекта**
- **Должен иметь модификатор public**

Конструктор

Особенности конструктора:

- Каждый класс имеет конструктор, принятый по умолчанию
- Конструктор по умолчанию не имеет параметров и устанавливает все поля в значение по умолчанию в зависимости от их типа
- Программист может определить собственный конструктор. В этом случае конструктор по умолчанию удаляется.
- Конструктор может иметь параметры.



Конструктор

C#

```
1  class Product
2  {
3        public float Price;
4        public float Amount;
5        private int pID;
6
7        // Конструктор
8        public Product(int ID)
9        {
10        pID = ID;
11        if (pID == 1111)
12                  Price = 10;
13              else
14                  Price = 20;
15       }
16
17       public void Calc(int count)
18       {
19           Amount = Price * count;
20       }
21 }
```

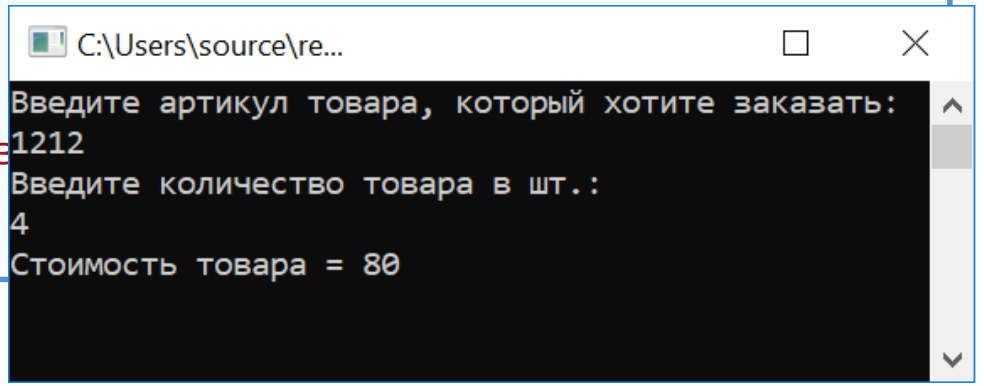
Конструктор с параметром, созданный программистом

Конструктор

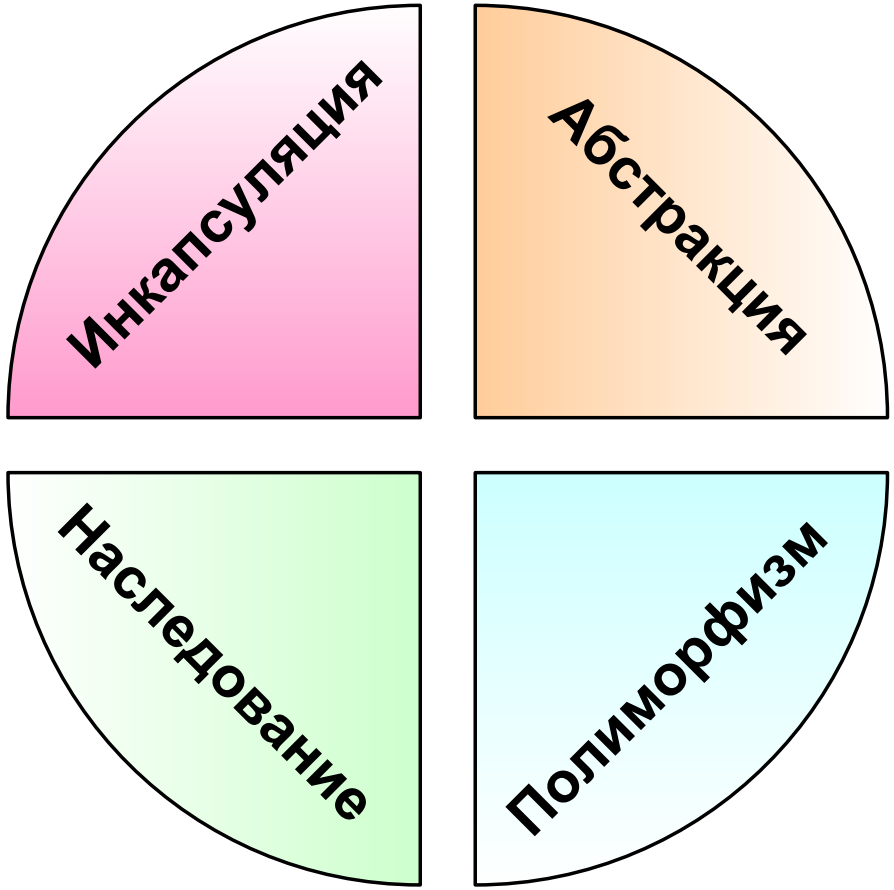
C#

```
22 static void Main(string[] args)
23 {
24   int number = 0;
25   Console.WriteLine("Введите артикул товара, который хотите заказать: ");
26
27   int vendorCode = Convert.ToInt32(Console.ReadLine());
28
29
30   Product orderedProduct = new Product(vendorCode);
31
32   Console.WriteLine("Введите количество товара в шт.: ");
33   number = Convert.ToInt32(Console.ReadLine());
34
35
36   orderedProduct.Calc(number);
37   Console.WriteLine("Стоимость тов
38   Console.ReadKey();
39 }
```

Вызов конструктора



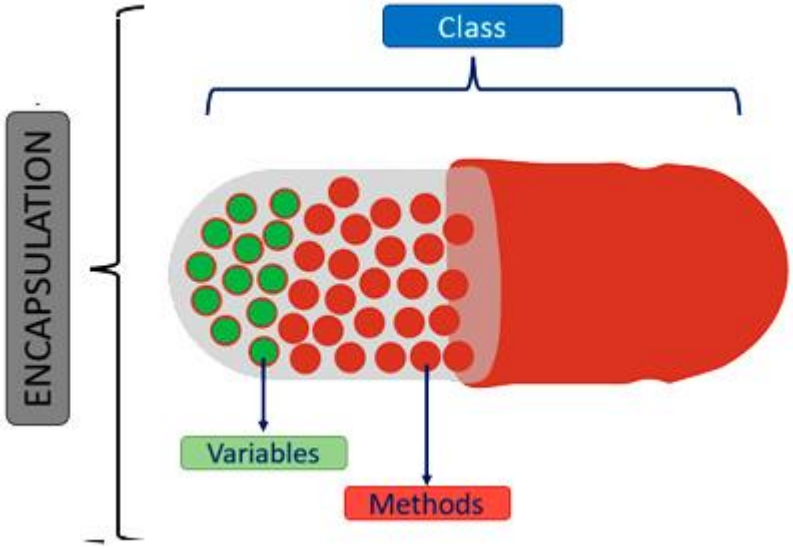
Базовые принципы ООП

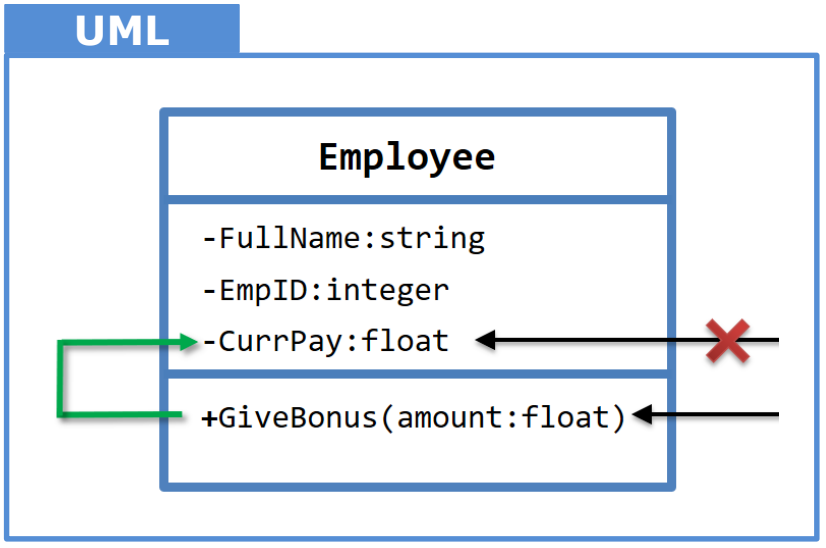


Инкапсуляция

Инкапсуляция – скрытие деталей внутреннего устройства класса от внешних по отношению к нему объектов или классов.

- Поля закрываются от доступа извне
- Доступ к закрытым полям выполняется через открытые методы, играющие роль интерфейса (интерфейсные методы)





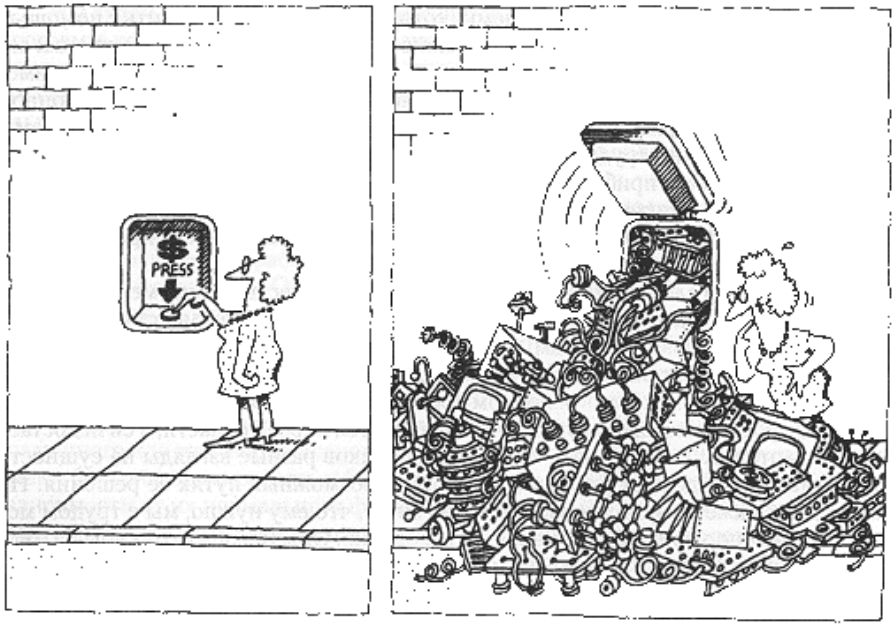
Инкапсуляция

Инкапсуляция = **скрытые поля** + **открытые интерфейсные методы**

C#

```
 1  class Employee
 2  {
 3      //закрытые данные класса
 4      private string FullName;
 5      private int EmpID;
 6      private float CurrPay;
 7
 8      //конструктор
 9      public Employee(string FName, int EmployeeID, float CurrentPay)
10      {
11          FullName = FName;
12          EmpID = EmployeeID;
13          CurrPay = CurrentPay;
14      }
15
16      //метод для увеличения зарплаты работника
17      public void GiveBonus(float amount)
18      {
19          CurrPay = CurrPay + amount;
20      }
21  }
```

Зачем нужна инкапсуляция?

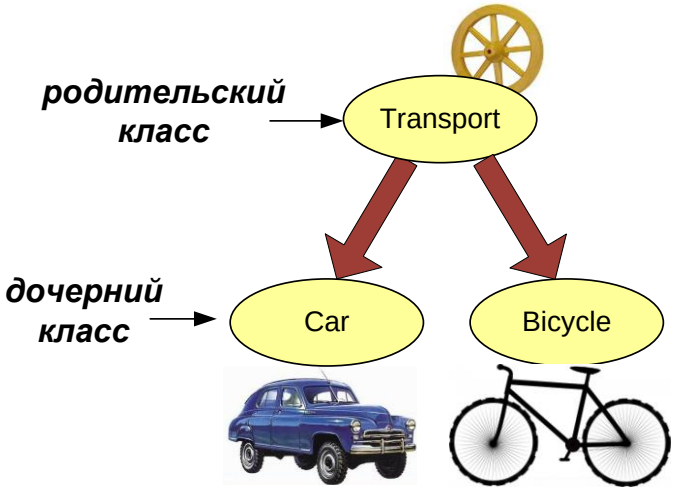


- борьба со сложностью
- безопасность внутренних данных
- возможность изменять «внутренности», не меняя интерфейс

Наследование

Наследование (**inheritance**) – механизм создания новых классов на основе существующих.

- При наследовании ***дочернему классу*** (***subclass***) передаются поля и методы ***родительского класса*** (***superclass***)
- У класса может быть один родитель и любое количество дочерних классов



Наследование

Наследование в разработке кода:

- Определение нового типа данных путем указания отличий от ранее определенного типа данных.
- Определяемый тип данных будет содержать поля и методы предка (унаследованные поля и методы) и поля и методы, определенные в нем самом.

Наследование в проектировании:

UML

C#

```
//Класс предок

class ClassA
{
    //Поля
    //Методы()
}
```

C#

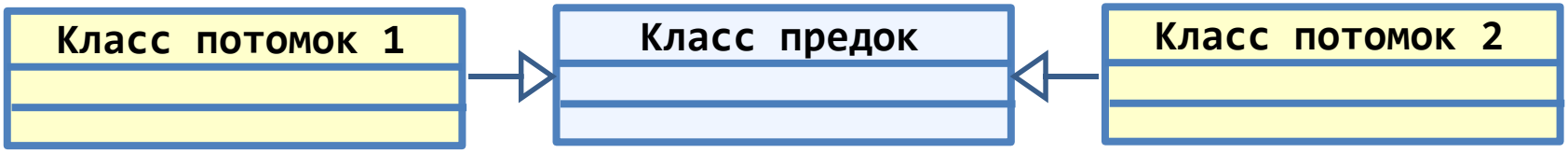
```
//Класс потомок(наследник)

class ClassB : ClassA
{
   //Дополнительные поля
   //Дополнительные методы()
}
```

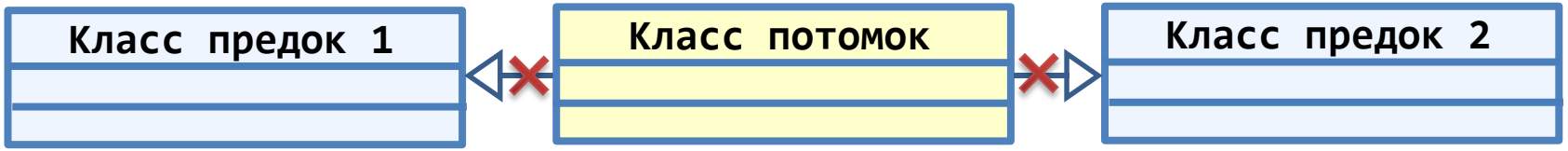
Наследование

Особенности наследования в C# (Java, PHP и др.):

- Класс предок может иметь несколько классов потомков



- Класс может иметь не более одного предка **(множественное наследование запрещено, в отличие от C++ и UML)**



Наследование

Объявление классов-наследников Manager и SalesPerson:

C#

```
1  class Manager : Employee
2  {
3    //менеджерам необходимо знать кол-во
4    //имеющихся у них опционов на акции
5    private int numberOfOptions;
6
7    public void NumOptions(int count)
8    {
9       numberOfOptions = count;
10   }
11 }
12
13
14 class SalesPerson : Employee
15 {
16   //продавцам надо знать объем продаж
17   private int numberOfSales;
18
19   public void NumSales(int count)
20   {
21       numberOfSales = count;
22   }
23 }
```

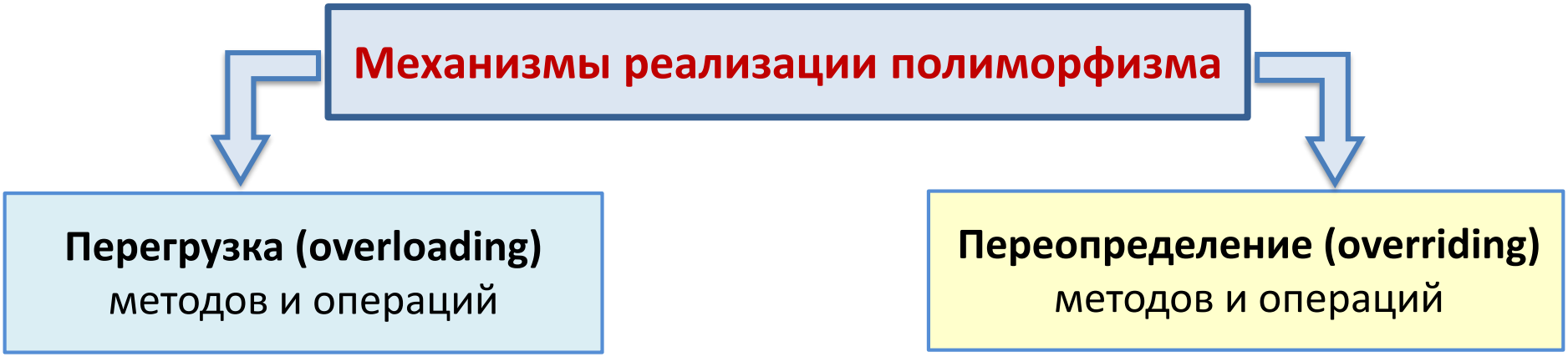
Создание объекта класса-наследника SalesPerson:

```
24
25
26 static void Main(string[] args)
27 {
28  //создаем объект продавец с именем Ivan:
29
30  SalesPerson Ivan = new SalesPerson();
31
32  //Этот член унаследован от
33  //базового класса Employee:
34
35  Ivan.GiveBonus(5000);
36
37  //А это уникальный член
38  //определенный в классе-наследнике:
39
40  Ivan.NumSales(80);
41 }
42
43
44
45
46
```

Полиморфизм

Полиморфизм (polymorphism) – реализация одной и той же операции по разному

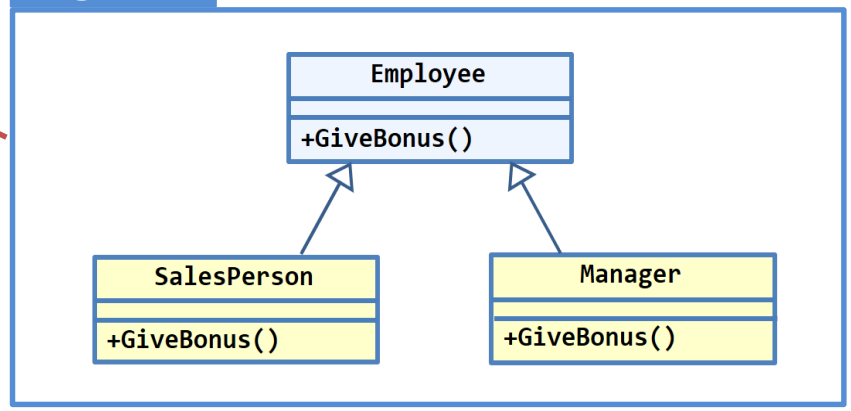
- имеется несколько реализаций алгоритма
- выбор реализации осуществляется в зависимости от типа объекта и типа параметров



Полиморфизм: переопределение методов (overriding)

Переопределим реакцию объектов производных классов на метод базового класса (метод увеличения зарплаты сотрудников):

UML



1. Метод базового класса, который будет переопределен должен быть объявлен, как виртуальный (ключевое слово `virtual`):

C#

```
1  class Employee
2  {
3
4      //метод для увеличения зарплаты работника
5      public virtual void GiveBonus(float amount)
6      {
7          CurrPay = CurrPay + amount;
8      }
9  }
```

Полиморфизм: переопределение методов (overriding)

2. Переопределяя виртуальный метод в классе-наследнике, необходимо заново определить его, используя ключевое слово `override`:

C#

```
10  class SalesPerson : Employee
11  {
12
13    public override void GiveBonus(float amount)
14    {
15      int SalesBonus = 0;  //кратность надбавки
16                           //надбавка зависит от объема продаж
17      if (numberOfSales >=0  &&  numberOfSales <=100)
18          SalesBonus = 2;
19      else
20          SalesBonus = 5;
21
22      base.GiveBonus(amount * SalesBonus);
23    }
24  }
```

Полиморфизм: переопределение методов (overriding)

Переопределение метода GiveBonus() класса Manager:

C#

```
25  class Manager : Employee
26  {
27    public int numOfSalesPerson = 10; //количество продавцов в подчинении
28
29    public override void GiveBonus(float amount)
30    {
31        base.GiveBonus(amount+amount*numOfSalesPerson*(float)0.1);
32    }
33  }
```

Вызов переопределенных методов:

C#

```
34  static void Main(string[] args)
35  {
36     //Улучшенная система премирования
37     Manager Nick = new Manager();          //Создаем менеджера Nick
38     Nick.GiveBonus(5000);
39
40     SalesPerson Ivan = new SalesPerson(); //Создаем продавца Ivan
41     Ivan.GiveBonus(1000);
42  }
```

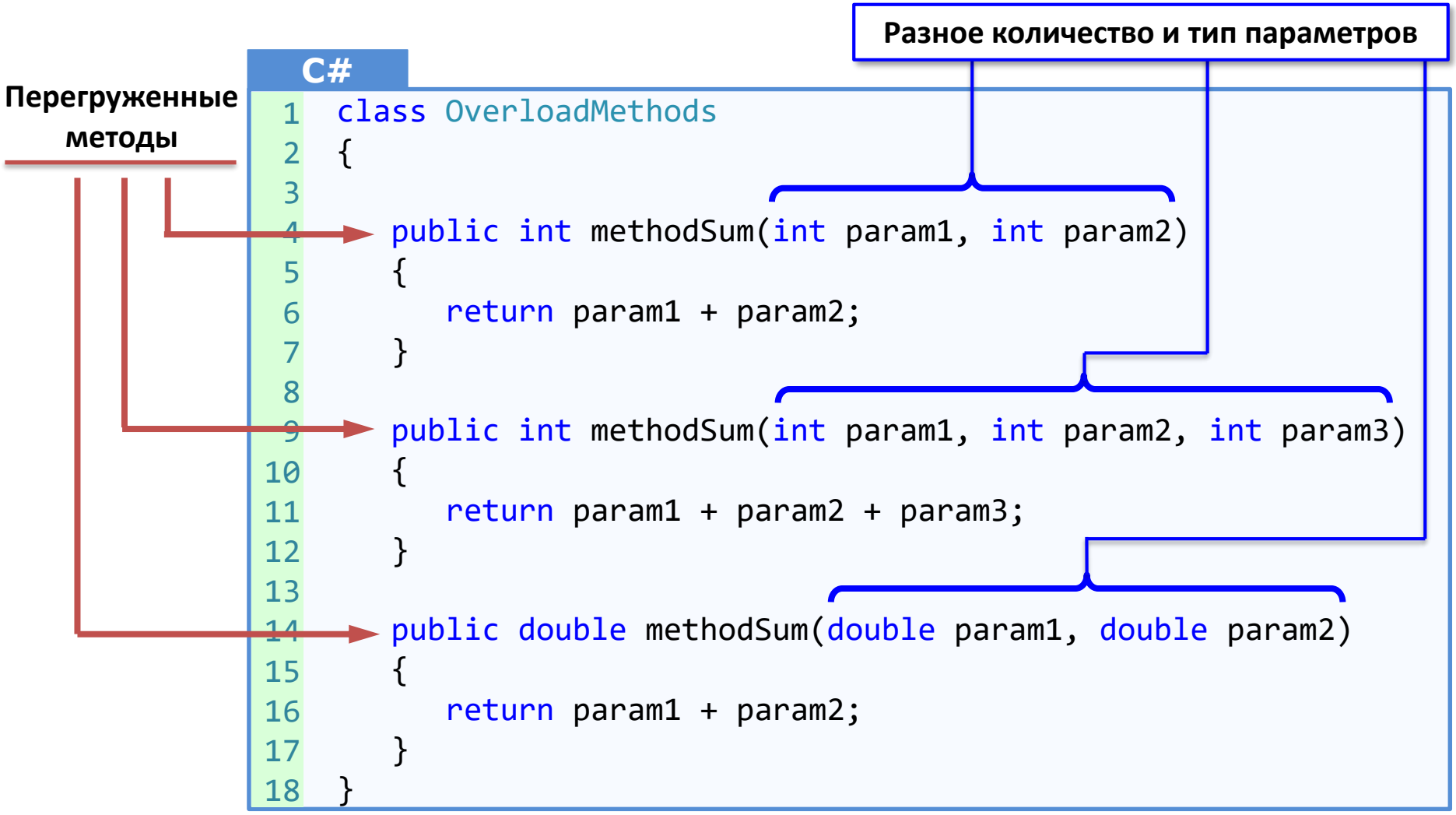
Полиморфизм: перегрузка методов (overloading)

В C# допускается совместное использование одного и того же имени двумя или более методами одного и того же класса, при условии, что их параметры объявляются по-разному. В этом случае говорят, что методы перегружаются, а сам процесс называется ***перегрузкой методов***.

Условия перегрузки методов:

- разный тип передаваемых параметров
- разное количество передаваемых параметров
- комбинация первых двух случаев

Полиморфизм: перегрузка методов (overloading)



Полиморфизм: перегрузка методов (overloading)

C#

```
1  class Employee
2  {
3      //закрытые данные класса
4      private string FullName;
5      private int EmpID;
6      private float CurrPay;
7
8      //метод для увеличения зарплаты работника
9      public void GiveBonus(float amount)
10     {
11         CurrPay = CurrPay + amount;
12     }
13
14     //метод увеличения зарплаты работника от объема продаж
15     public void GiveBonus(float amount, int sales)
16     {
17         CurrPay = CurrPay + amount*sales;
18     }
19 }
```

Перегруженные методы

Steve Jobs, 1994, интервью журналу Rolling Stone:

Не могли бы вы в нескольких словах объяснить, что же такое объектно-ориентированное программное обеспечение?

- «Объекты – они как люди. Они живые вещи, у которых есть разум, позволяющий им знать, как сделать ту или иную вещь, у них есть память. Вы взаимодействуете с ними на очень высоком уровне абстракции, словно с людьми.
- Например, я – ваш объект, занимающийся чисткой ваших вещей. Вы можете дать мне грязную одежду и послание “доставь мои вещи в прачечную”. Я знаю, где в Сан-Франциско лучшая прачечная, я говорю по-английски и у меня есть деньги в кармане. Я выхожу, ловлю такси, посещаю прачечную и возвращаюсь с вашими вещами и словами: “Вот, ваша чистая одежда”.
- Вы не знаете, как я это сделал. Не знаете, где эта прачечная или вы говорите по-французски, а может у вас нет денег, чтобы поймать такси. Однако, я знал, как все это сделать, а вам – это знать необязательно. Вся сложность процесса скрыта внутри меня, но наше с вами общение было простым – в этом вся суть объектов. **СЛОЖНОСТИ – ВНУТРИ, НО ИНТЕРФЕЙС – ДОСТУПЕН КАЖДОМУ.»**