

Визуальное программи- рование

ЛЕКЦИЯ 2

Содержание лекции

01

ООП

02

Инкапсуляция

03

Наследование

04

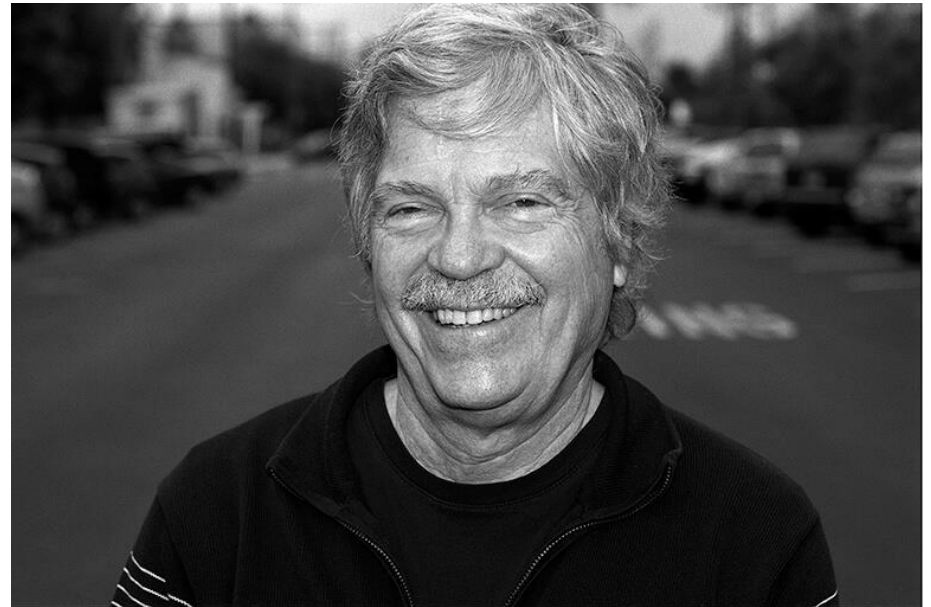
Полиморфизм

ОБЪЕКТНО- ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Кто придумал ООП

Алан Кёртис Кэй — американский учёный в области теории вычислительных систем.

Один из пионеров в областях объектно-ориентированного программирования и графического интерфейса.



Принципы ООП

- Все является объектом
- Каждый объект является экземпляром класса
- Класс определяет поведение объекта
- Классы организованы в иерархию наследования
- Каждый объект обладает независимой памятью
- Вычисления производятся путем взаимодействия между объектами

Понятие класса

Класс представляет собой шаблон, по которому определяется форма объекта.

В нем указываются данные и код, который будет оперировать этими данными.



Понятие класса

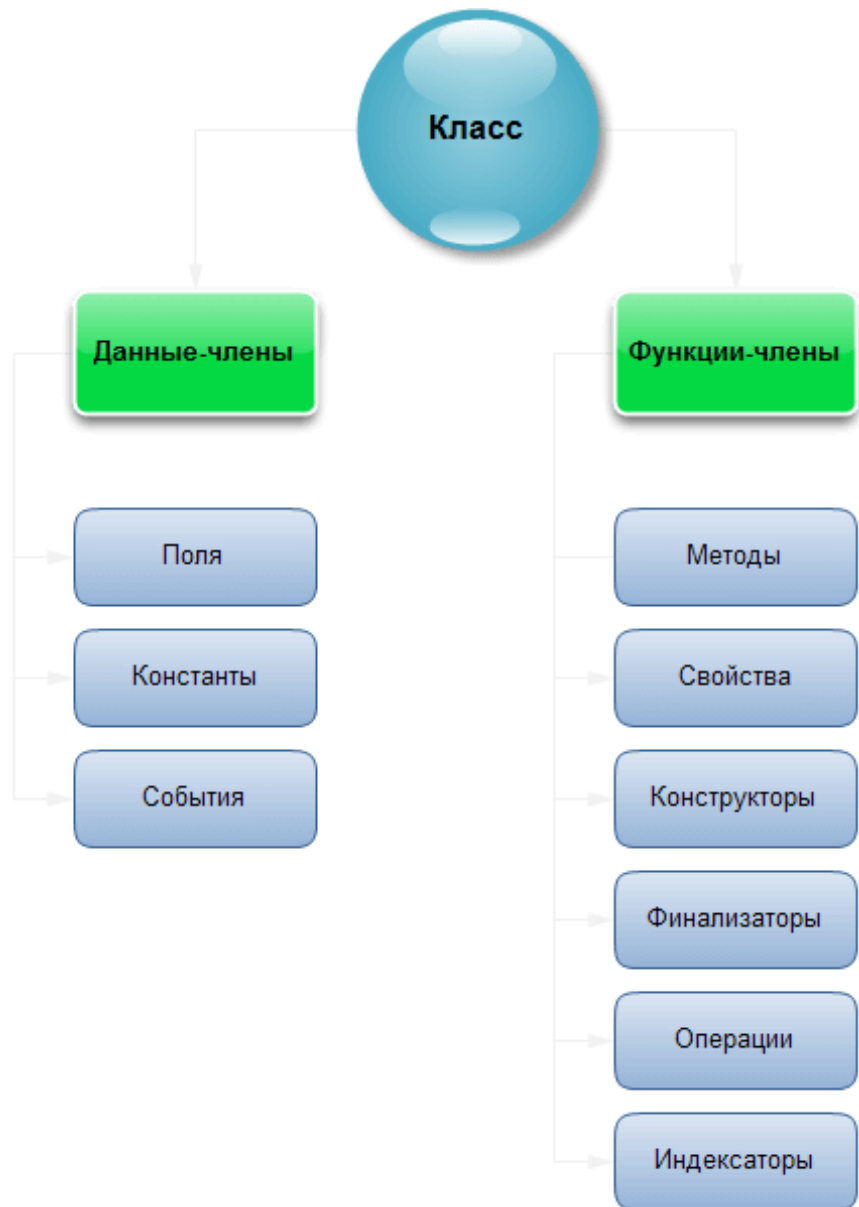
Класс является **логической абстракцией**.

Физическое представление класса появится в оперативной памяти лишь после того, как будет создан **объект** этого класса.

Таким образом, объект – экземпляр некоторого класса.

Описание класса

```
class имя_класса {  
    // Объявление переменных экземпляра (полей)  
    доступ тип поле1;  
    доступ тип поле2;  
    //...  
    доступ тип полеN;  
  
    // Объявление методов  
    доступ возвращаемый_тип метод1 (параметры)  
    {  
        // тело метода  
    }  
    доступ возвращаемый_тип метод2 (параметры)  
    {  
        // тело метода  
    }  
    //...  
    доступ возвращаемый_тип методN(параметры)  
    {  
        // тело метода  
    }  
}
```

Данные-члены — это те члены, которые содержат данные класса. Они могут быть статическими. Член класса является членом экземпляра, если только он не объявлен явно как `static`.

Функции-члены — это члены, которые обеспечивают некоторую функциональность для манипулирования данными класса.

Данные-члены

Поля

Это любые переменные, ассоциированные с классом.

```
private double length;  
private double width;  
private string name = "name";
```

Данные-члены

Константы

Могут быть ассоциированы с классом тем же способом, что и переменные. Константа объявляется с помощью ключевого слова `const`. Если она объявлена как `public`, то в этом случае становится доступной извне класса.

```
private const string Name = "name";  
public const int Count = 0;
```

Данные-члены

События

Это члены класса, позволяющие объекту уведомлять вызывающий код о том, что случилось нечто достойное упоминания, например, изменение свойства класса либо некоторое взаимодействие с пользователем.

Клиент может иметь код, известный как обработчик событий, реагирующий на них.

```
public class EventClass
{
    public delegate void SampleEventHandler(string message);
    public event SampleEventHandler SampleEvent;
}

void DisplayMessage(string message) => Console.WriteLine(message);

EventClass EventExample = new EventClass();

EventExample.SampleEvent += DisplayMessage;
EventExample.SampleEvent -= DisplayMessage;
```

Функции-члены

Методы

Это функции, ассоциированные с определенным классом. Как и данные-члены, по умолчанию они являются членами экземпляра. Они могут быть объявлены статическими с помощью модификатора `static`.

```
public static int Add (int a, int b)
{
    return a + b;
}

ссылка: 1
public void Increment (ref int arg)
{
    arg += 10;
}

Ссылка: 0
public void Output ()
{
    int a = 10;
    Increment(ref a);
    Console.WriteLine($"Result: {a}");
}
```

Функции-члены

Свойства

Это наборы функций, которые могут быть доступны клиенту таким же способом, как общедоступные поля класса.

В C# предусмотрен специальный синтаксис для реализации чтения и записи свойств для классов, поэтому писать собственные методы с именами, начинающимися на Set и Get, не понадобится.

```
public double Length
{
    get
    {
        return length;
    }
    set
    {
        if (value > 0) this.length = value;
    }
}
```

Функции-члены

Конструкторы

Это специальные функции, вызываемые автоматически при инициализации объекта. Их имена совпадают с именами классов, которым они принадлежат, и они не имеют типа возврата. Конструкторы полезны для инициализации полей класса.



Функции-члены

```
public class Student
{
    ссылка: 1
    public string FirstName { get; set; }
    ссылка: 1
    public string LastName { get; set; }
    ссылка: 1
    public DateTime BirthDate { get; set; }

    ссылка: 1
    public Student()
    {
    }

    Ссылки: 0
    public Student(string firstName, string lastName, DateTime bday)
    {
        FirstName = firstName;
        LastName = lastName;
        BirthDate = bday;
    }
}
```

```
Student student1 = new Student();
Student student2 = new Student()
{ FirstName = "Ivan", LastName = "Ivanov", BirthDate = date };
Student student3 = new Student("Petr", "Petrov", date);
```


Функции-члены

Финализаторы

Вызываются, когда среда CLR определяет, что объект больше не нужен. Они имеют то же имя, что и класс, но с предшествующим символом тильды. Предсказать точно, когда будет вызван финализатор, невозможно.

```
class Car
{
    Ссылка: 0
    ~Car() // finalizer
    {
        // cleanup statements...
    }
}
```

Функции-члены

- В структурах определение методов завершения невозможно. Они применяются только в классах.
- Каждый класс может иметь только один метод завершения.
- Методы завершения не могут быть унаследованы или перегружены.
- Методы завершения невозможно вызвать. Они запускаются автоматически.
- Метод завершения не принимает модификаторов и не имеет параметров.

Функции-члены

Операторы

Это простейшие действия вроде + или -. Когда вы складываете два целых числа, то, строго говоря, применяете операцию + к целым.

Однако C# позволяет указать, как существующие операции будут работать с пользовательскими классами (так называемая перегрузка операции).

Функции-члены

Индексаторы

Позволяют индексировать объекты таким же способом, как массив или коллекцию.

```
float[] temps = new float[10]
{
    56.2F, 56.7F, 56.5F, 56.9F, 58.8F,
    61.3F, 65.9F, 62.1F, 59.2F, 57.5F
};
Ссылка: 0
public int Length => temps.Length;
Ссылка: 0
public float this[int index]
{
    get => temps[index];
    set => temps[index] = value;
}
```

Partial

Можно разделить определение класса, структуры, интерфейса или метода между двумя или более исходными файлами. Каждый исходный файл содержит часть определения класса или метода, а во время компиляции приложения все части объединяются.

```
public partial class Employee
{
    Ссылка: 0
    public void DoWork()
    {
    }
}

ссылка: 1
public partial class Employee
{
    Ссылка: 0
    public void GoToLunch()
    {
    }
}
```

Partial

- Все части должны использовать ключевое слово `partial`.
- Для формирования окончательного типа все части должны быть доступны во время компиляции.
- Все части должны иметь одинаковые модификаторы доступа.
- Если какая-либо из частей объявлена абстрактной, то весь тип будет считаться абстрактным.
- Если какая-либо из частей объявлена запечатанной, то весь тип будет считаться запечатанным.
- Если какая-либо из частей объявляет базовый тип, то весь тип будет наследовать данный класс.

Статические классы

Статический класс может использоваться как обычный контейнер для наборов методов, работающих на входных параметрах, и не должен возвращать или устанавливать каких-либо внутренних полей экземпляра.

```
public static class TemperatureConverter
{
    Ссылка: 0
    public static double CelsiusToFahrenheit(string temperatureCelsius)
    {
        double celsius = Double.Parse(temperatureCelsius);
        double fahrenheit = (celsius * 9 / 5) + 32;
        return fahrenheit;
    }

    Ссылка: 0
    public static double FahrenheitToCelsius(string temperatureFahrenheit)
    {
        double fahrenheit = Double.Parse(temperatureFahrenheit);
        double celsius = (fahrenheit - 32) * 5 / 9;
        return celsius;
    }
}
```

Статические классы

- Содержит только статические члены
- Создавать его экземпляры нельзя
- Является запечатанным
- Не может содержать конструкторы экземпляров

По сути, создание статического класса аналогично созданию класса, содержащего только статические члены и закрытый конструктор.

Статические члены класса

- Нестатический класс может содержать статические методы, поля, свойства или события.
- Статический член вызывается для класса даже в том случае, если не создан экземпляр класса.
- Доступ к статическому члену всегда выполняется по имени класса, а не экземпляра.
- Существует только одна копия статического члена, независимо от того, сколько создано экземпляров класса.

Статические члены класса

- Статические методы и свойства не могут обращаться к нестатическим полям и событиям в их содержащем типе, и они не могут обращаться к переменной экземпляра объекта, если он не передается явно в параметре метода.
- Статические методы могут быть перегружены, но не переопределены, поскольку они относятся к классу, а не к экземпляру класса.
- Доступ к статическому члену всегда выполняется по имени класса, а не экземпляра.
- C# не поддерживает статические локальные переменные.

ИНКАПСУЛЯЦИЯ

Понятие инкапсуляции

Инкапсуляция — это механизм программирования, объединяющий вместе код и данные, которыми он манипулирует, исключая как вмешательство извне, так и неправильное использование данных.



Понятие инкапсуляции

Основной единицей инкапсуляции в C# является **класс**, который определяет форму объекта. Он описывает данные, а также код, который будет ими оперировать.

В C# описание класса служит для построения **объектов**, которые являются экземплярами класса. Следовательно, класс, по существу, представляет собой ряд схематических описаний способа построения объекта.

Модификаторы доступа

Private - доступ ограничен содержащим классом.

Public - неограниченный доступ.

Protected - доступ ограничен содержащим классом или классами, которые являются производными от содержащего класса.

Internal – доступ ограничен текущей сборкой.

НАСЛЕДОВАНИЕ

Понятие наследования

Наследование определяет способность языка позволять строить новые определения классов на основе определений существующих классов.

По сути, наследование позволяет расширять поведение базового (или родительского) класса, наследуя основную функциональность в производном подклассе (также именуемом дочерним классом).



Понятие наследования

Всякий раз, когда один класс наследует от другого, после имени производного класса указывается имя базового класса, отделяемое двоеточием.

```
public class Circumference : Figure
```

Понятие наследования

Производный класс может иметь **только один** прямой базовый **класс**.

Однако наследование является **транзитивным**. Если ClassC является производным от ClassB, а ClassB — от ClassA, ClassC наследует члены, объявленные в ClassB и ClassA.

При этом один класс можно унаследовать от **нескольких интерфейсов**.

Абстрактные классы

- Абстрактный (**abstract**) класс можно использовать, только если новый класс является производным от него.
- Абстрактный класс может содержать один или несколько сигнатур методов, которые сами объявлены в качестве абстрактных. Эти сигнатуры задают параметры и возвращают значение, но не имеют реализации (тела метода).
- Абстрактному классу необязательно содержать абстрактные члены; однако если класс все же содержит абстрактный член, то сам класс должен быть объявлен в качестве абстрактного.
- Производные классы, которые сами не являются абстрактными, должны предоставить реализацию для любых абстрактных методов из абстрактного базового класса.

Абстрактные классы

В реальности не существует геометрической фигуры как таковой. Есть круг, квадрат, квадрат, но просто фигуры нет. Однако же и круг, и прямоугольник имеют что-то общее и являются фигурами.

Для описания подобных сущностей, которые **не имеют конкретного воплощения**, предназначены абстрактные классы.

```
public abstract class Figure
{
    private double length;
    Ссылка: 0
    public string Name { get; set; }

    ссылка: 1
    public virtual double Length...
}
```

Абстрактные и виртуальные методы

- Когда базовый класс объявляет метод как **virtual**, производный класс может переопределить (**override**) метод с помощью своей собственной реализации.
- Если базовый класс объявляет метод как **abstract**, этот метод должен быть переопределен в любом неабстрактном классе, который прямо наследует от этого класса.
- Если производный класс сам является абстрактным, то он наследует абстрактные члены, не реализуя их.
- Помимо **методов**, виртуальными могут быть **свойства**, **события** и **индексаторы**.

Абстрактные и виртуальные методы

Переопределение виртуального свойства Length.

```
public override double Length
{
    get
    {
        return 2*Math.PI*Radius;
    }
}
```

Ключевое слово `base`

Ключевое слово `base` используется для доступа к членам базового из производного класса в следующих случаях:

- Вызов метода базового класса, который был переопределен другим методом.
- Определение конструктора базового класса, который должен вызываться при создании экземпляров производного класса.

Ключевое слово base

Дан базовый класс Item, для него описан конструктор, который принимает два аргумента.

```
public class Item
{
    ссылка: 1
    public string Name { get; private set; }
    ссылка: 1
    public int Price { get; private set; }

    ссылка: 1
    public Item (string name, int price)
    {
        this.Name = name;
        this.Price = price;
    }
}
```


Ключевое слово base

В производном классе Table конструктор использует конструктор базового класса Item.

```
public class Table : Item
{
    ссылка: 1
    public string Color { get; private set; }
    Ссылка: 0
    public Table (string tableName,
                  int tablePrice,
                  string color)
        : base (tableName, tablePrice)
    {
        this.Color = color;
    }
}
```

Интерфейсы

Интерфейс — это ссылочный тип, определяющий набор членов.

Интерфейс может определять реализацию по умолчанию для любого из этих членов или для них всех.


Интерфейсы

В классе может быть реализовано несколько интерфейсов, хотя производным он может быть только от одного прямого базового класса.

```
public class MyCollection: IEnumerable<int>, IList<int>
{
    ...
}
```

Интерфейсы

Набор членов интерфейса должны реализовать все классы и структуры, реализующие интерфейс.

```
ICollection<int>

• interface System.Collections.Generic.ICollection<T>
Represents a collection of objects that can be individually accessed by index.

T является int

CS0535: "MyCollection" не реализует член интерфейса "ICollection<int>.IndexOf(int)".
CS0535: "MyCollection" не реализует член интерфейса "ICollection<int>.Insert(int, int)".
CS0535: "MyCollection" не реализует член интерфейса "ICollection<int>.RemoveAt(int)".
CS0535: "MyCollection" не реализует член интерфейса "ICollection<int>.this[int]".
CS0535: "MyCollection" не реализует член интерфейса "ICollection<int>.Add(int)".
CS0535: "MyCollection" не реализует член интерфейса "ICollection<int>.Clear()".
CS0535: "MyCollection" не реализует член интерфейса "ICollection<int>.Contains(int)".
CS0535: "MyCollection" не реализует член интерфейса "ICollection<int>.CopyTo(int[], int)".
CS0535: "MyCollection" не реализует член интерфейса "ICollection<int>.Remove(int)".
CS0535: "MyCollection" не реализует член интерфейса "ICollection<int>.Count".
CS0535: "MyCollection" не реализует член интерфейса "ICollection<int>.IsReadOnly".

Показать возможные решения (Alt+ВВОДилиCtrl+ю)
```

Интерфейсы

```
public class MyCollection: IEnumerable<int>, IList<int>
{
    ...
}
```



• interface System.Collections.Generic.IEnumerable<out T>

Exposes the enumerator, which supports a simple iteration over a collection of a specified type.

T является int

CS0535: "MyCollection" не реализует член интерфейса "IEnumerable<int>.GetEnumerator()".

CS0535: "MyCollection" не реализует член интерфейса "IEnumerable.GetEnumerator()".

[Показать возможные решения](#) (Alt+ВВОДилиCtrl+ю)

Запечатанный класс

Класс может предотвратить наследование от других классов или наследование от любых его членов, объявив себя или члены как **sealed**.


```
public sealed class Example  
{  
    ...  
}
```

Запечатанный класс

```
public sealed class Example  
{  
    ...  
}
```

Ссылок: 0

```
public class OtherExample: Example  
{  
    ...  
}
```

 class VPClasses.Classes.Example

CS0509: "OtherExample": не может быть производным от запечатанного типа "Example".

[Показать возможные решения](#) (Alt+ВВОДилиCtrl+ю)

ПОЛИМОРФИЗМ

Понятие полиморфизма

Полиморфизм обозначает способность языка трактовать связанные объекты в сходной манере.

Этот принцип позволяет базовому классу определять набор членов (формально называемый **полиморфным интерфейсом**), которые доступны всем наследникам.

Полиморфный интерфейс класса конструируется с использованием любого количества виртуальных или абстрактных членов.

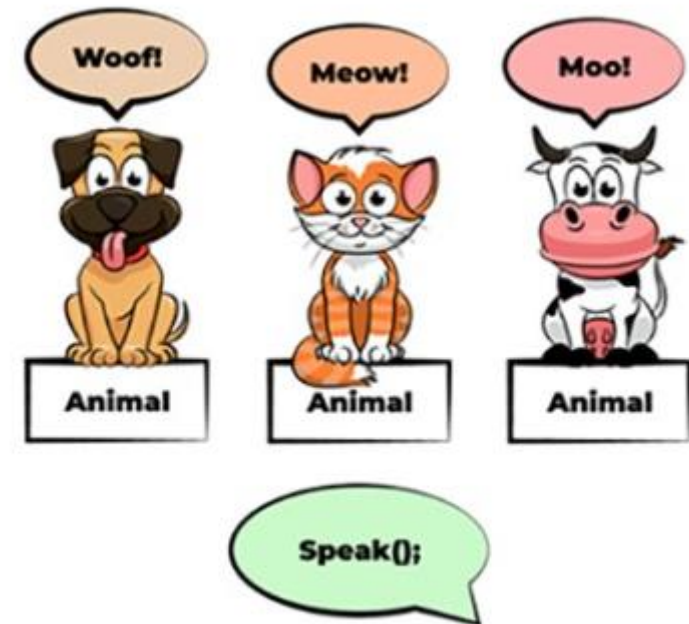


Понятие полиморфизма

Один интерфейс — множество реализаций.

Полиморфизм помогает упростить программу, позволяя использовать один и тот же интерфейс для описания общего класса действий.

Выбрать конкретное действие (т.е. метод) в каждом отдельном случае — это задача компилятора.



Раннее и позднее связывание

- **Раннее связывание** – связанное с формированием кода на этапе компиляции. При раннем связывании, программный код формируется на основе известной информации о типе ссылки. Как правило, это ссылка на базовый класс в иерархии классов.
- **Позднее связывание** – связанное с формированием кода на этапе выполнения. Если в иерархии классов встречается цепочка виртуальных методов (с помощью слов **virtual**, **override**), то компилятор строит так называемое позднее связывание. При позднем связывании вызов метода происходит на основании типа объекта, а не типа ссылки на базовый класс.

Раннее и позднее связывание

Виртуальный член — это член базового класса, определяющий реализацию по умолчанию, которая может быть изменена (переопределена) в производном классе.

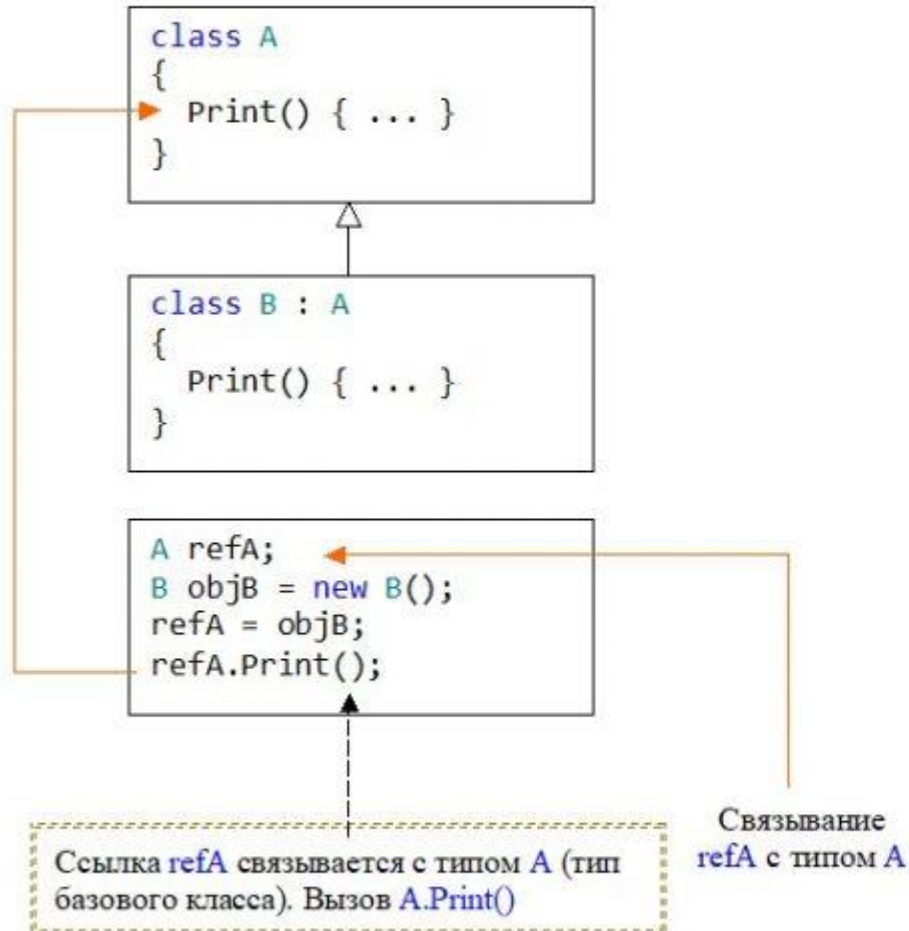
Абстрактный член — это член базового класса, который не предусматривает реализации по умолчанию, а предлагает только сигнатуру.

Раннее и позднее связывание

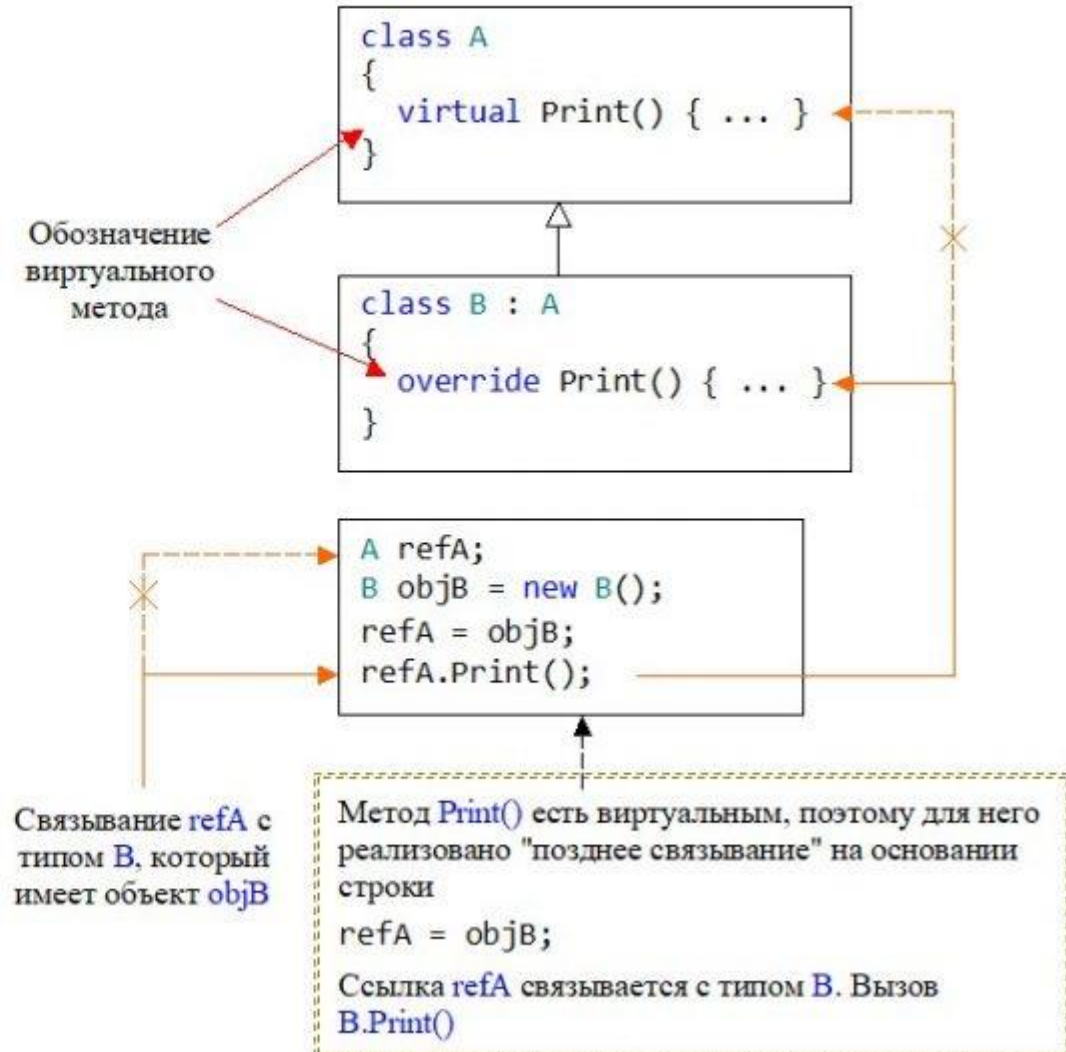
- Если в иерархии унаследованных классов объявляется **невиртуальный** элемент, то реализуется **раннее** связывание;
- Если в иерархии унаследованных классов объявляется **виртуальный** элемент, то выполняется **позднее** связывание.

Раннее и позднее связывание

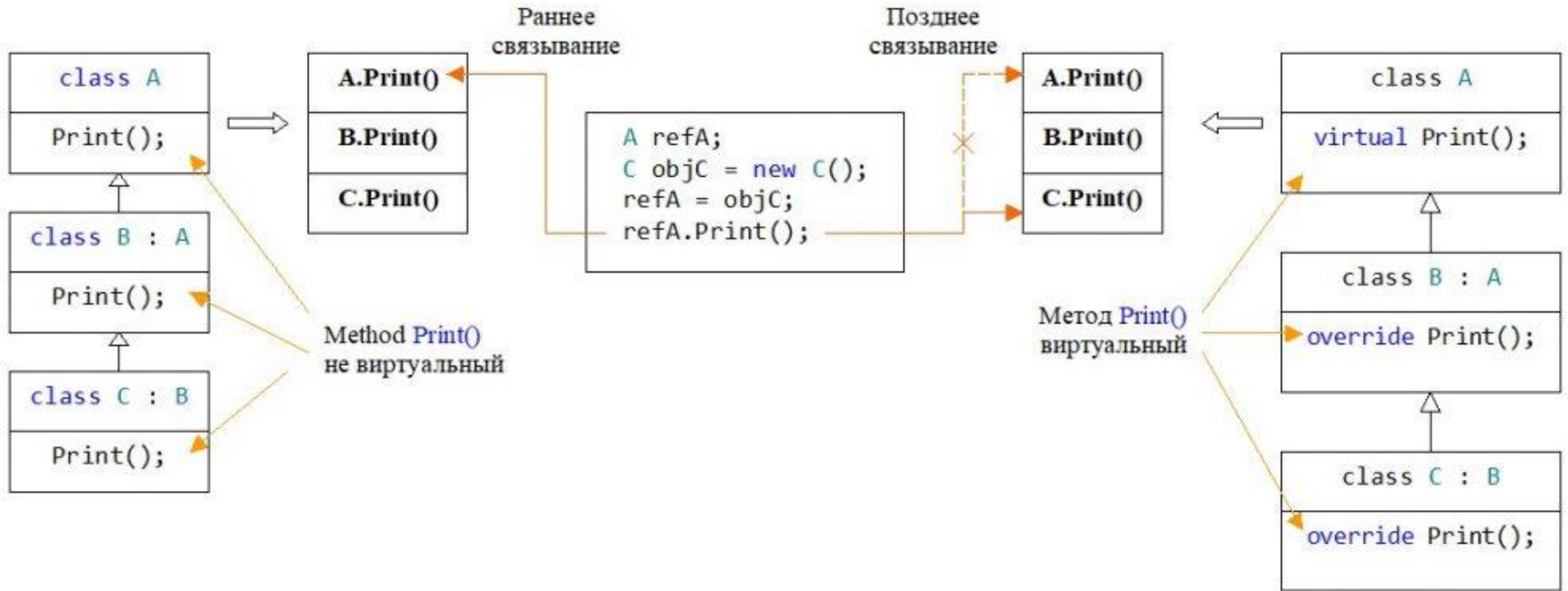
Раннее связывание



Позднее связывание для метода `Print()`
(определяется по словам `virtual`, `override`)



Раннее и позднее связывание



Перекрытие метода

С помощью ключевого слова **new** можно перекрыть метод базового класса в подчиненном.

Нужный метод выбирается на этапе компиляции.

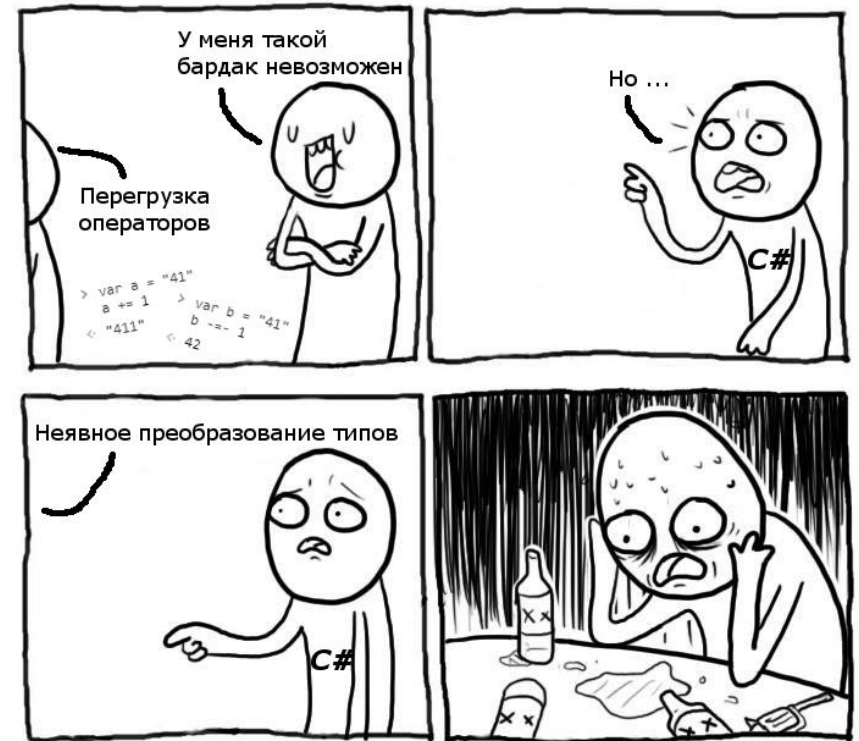
```
1 reference
public class User
{
    0 references
    public string GetInfo()
    {
        return "This is user";
    }
}

0 references
public class Student : User
{
    0 references
    public new string GetInfo()
    {
        return "This is student";
    }
}
```


Перегрузка методов

Внутри одного класса можно определить несколько методов с одним именем, которые должны отличаться количеством и/или типом входных параметров.

Нужный метод выбирается на этапе компиляции.



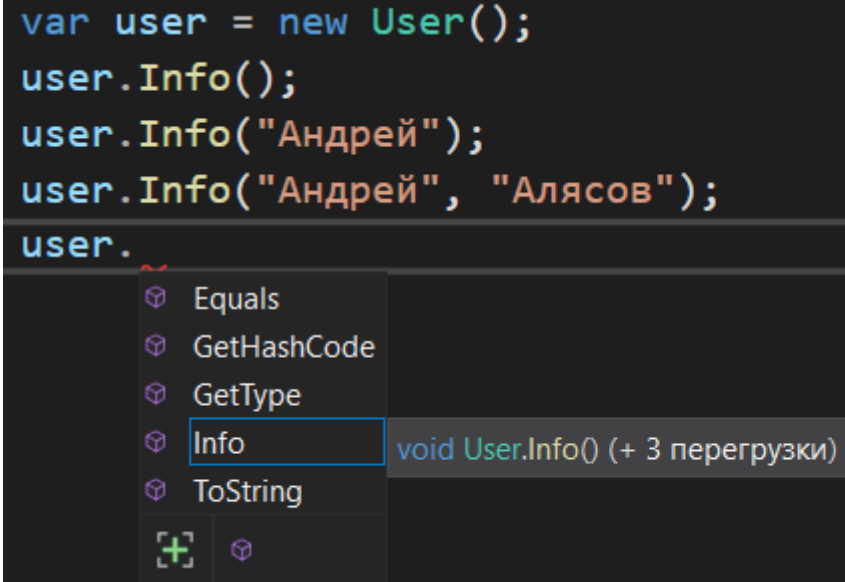
Перегрузка методов

```
public void Info()
{
    Console.WriteLine("Пустой метод\n");
}
Ссылок: 0
public void Info(string name)
{
    Console.WriteLine($"Имя: {name}");
}
Ссылок: 0
public void Info(string name, string lastName)
{
    Console.WriteLine($"Имя: {name}\nФамилия: {lastName}");
}
Ссылок: 0
public void Info(string name, string lastName, int age)
{
    Console.WriteLine($"Имя: {name}\nФамилия: {lastName}\nВозраст: {age}");
}
```

Перегрузка методов

При вызове метода можно посмотреть количество перегрузок и их сигнатуры.

```
var user = new User();  
user.Info();  
user.Info("Андрей");  
user.Info("Андрей", "Алясов");  
user.
```



- Equals
- GetHashCode
- GetType
- Info
- ToString

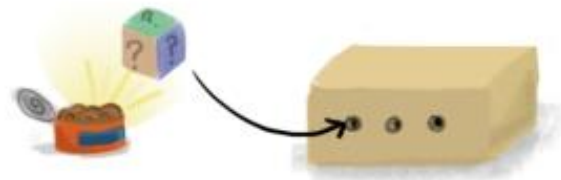
void User.Info() (+ 3 перегрузки)

Incapsulation



every animal eats
and then poop

Polymorphism



each animal can eat
its own type of food

Inheritance



you can create new type of animal
changing or adding properties

ВОПРОСЫ ПО ЛЕКЦИИ