

Немного истории

- Simula (1967)

Первый объектно-ориентированный язык.

- C++ (1983)

Первый объектно-ориентированный язык семейства C.

- Java (1995)

Объектно-ориентированный язык от Sun Microsystems.

- C# (2001)

Объектно-ориентированный язык от Microsoft.

Немного юмора

Which language will you use?

Perl is versatile...



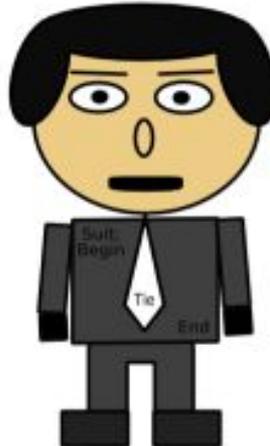
...but quickly gets out of control.

C++ is fast and powerful...



...but ugly and unsafe.

Delphi is neat and well structured...



...but well past its sell-by date.

Anyone can write VB...



...do you really want them to?

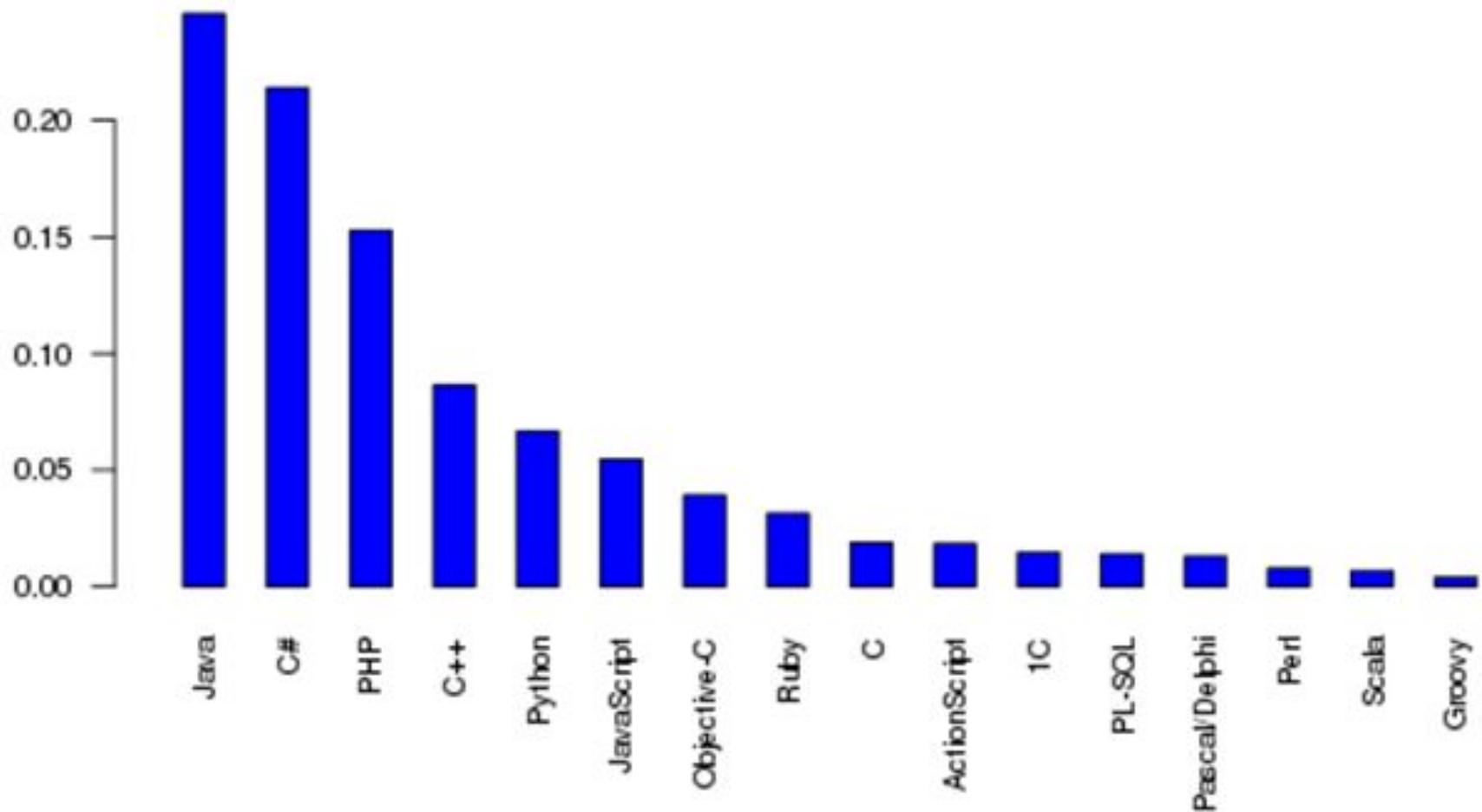
C# is managed and modern...



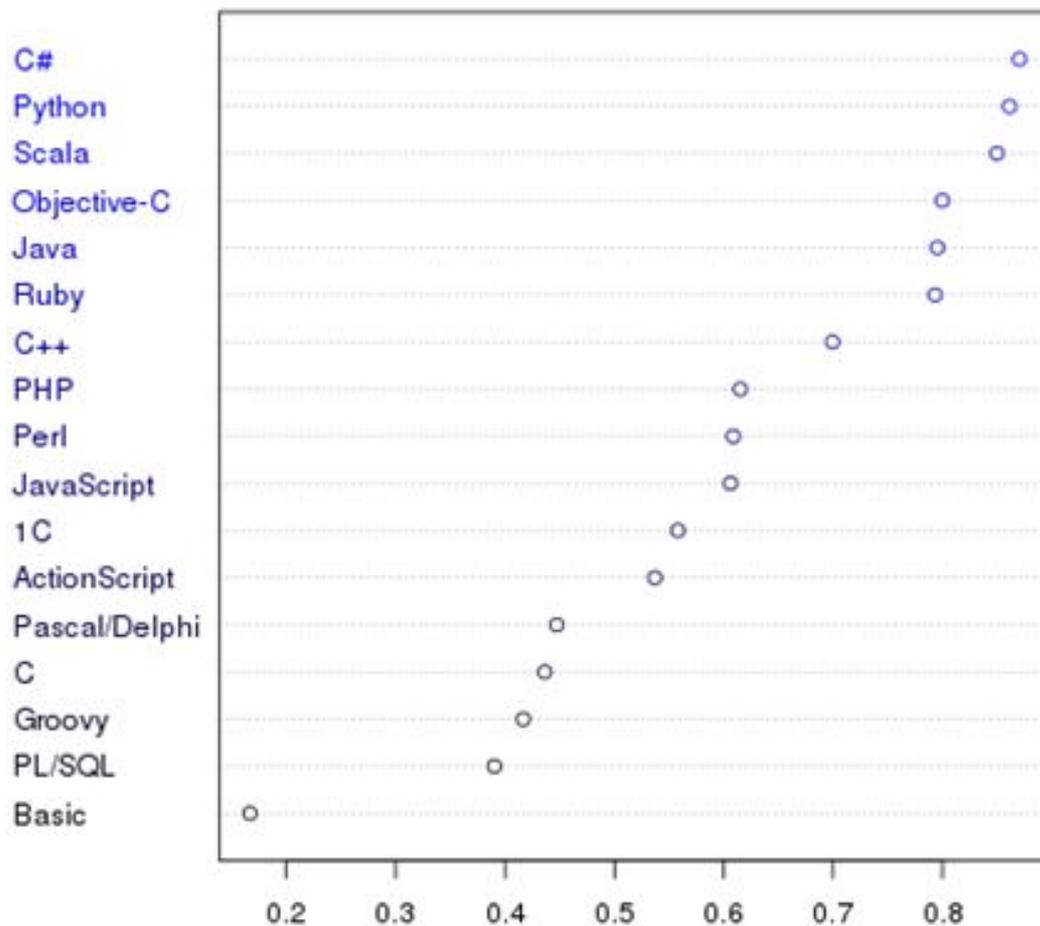
...but comes at quite a price.

Pick the right one for the job:java!

DOU.UA: На каком языке вы пишете для работы сейчас (2013)



DOU.UA: Индекс удовлетворенности языком



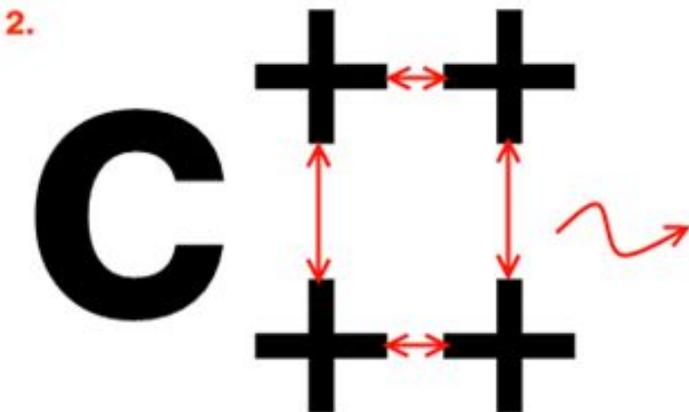
DOU.UA: финальная таблица результатов

№	Язык	Доля рынка сейчас	Изменение по сравнению с прошлым опросом	Рабочий сейчас	Доля рынка в перспективе	Начали бы проект сейчас	Как дополнительный рабочий	В хобби проекте	Индекс привязанности
1	Java	24.50	-1.6	719	24.25	828	508	731	0.795
2	C#	21.34		626	21.38	730	371	652	0.87
3	PHP	15.23	0.7	447	9.70	331	379	603	0.615
4	C++	8.62	-1.8	253	7.50	256	399	410	0.70
5	Python	6.65		351	10.28	351	381	448	0.86
6	JavaScript	5.45	1,75	160	5,71	195	1673	804	0.60
7	Objective-C	3.92		115	4.21	144	117	164	0.8
8	Ruby	3.13	-0.69	92	5.53	189	149	204	0.79
9	C	1.87		55	1,04	36	306	151	0.43
10	ActionScript	1.8		54	1.05	36	63	72	0.53
11	1C	1.4		43		30	23	26	0.55
12	PL-SQL	1.3		41		21	435	78	0.39
13	Pascal/Depthi	1.29		38		24	63	103	0.44

Cool - 'C like Object Oriented Language', C#, C Sharp, "Си шарп"

1. C + + + +

2. C # #



3. C #



Андерс Хейлсберг

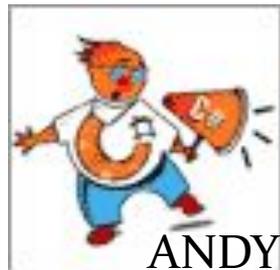
в **1980** – написал первый компилятор языка Паскаль

до **1996** – работа в Borland. Создал новое поколение компиляторов Паскаля — язык Delphi.

с **1996** – работа в Microsoft, проекты J++ и Microsoft Foundation Classes

с **2000**- возглавил группу по созданию и проектированию языка C#.

2012 – представил новый проект TypeScript



Особенности языка C#

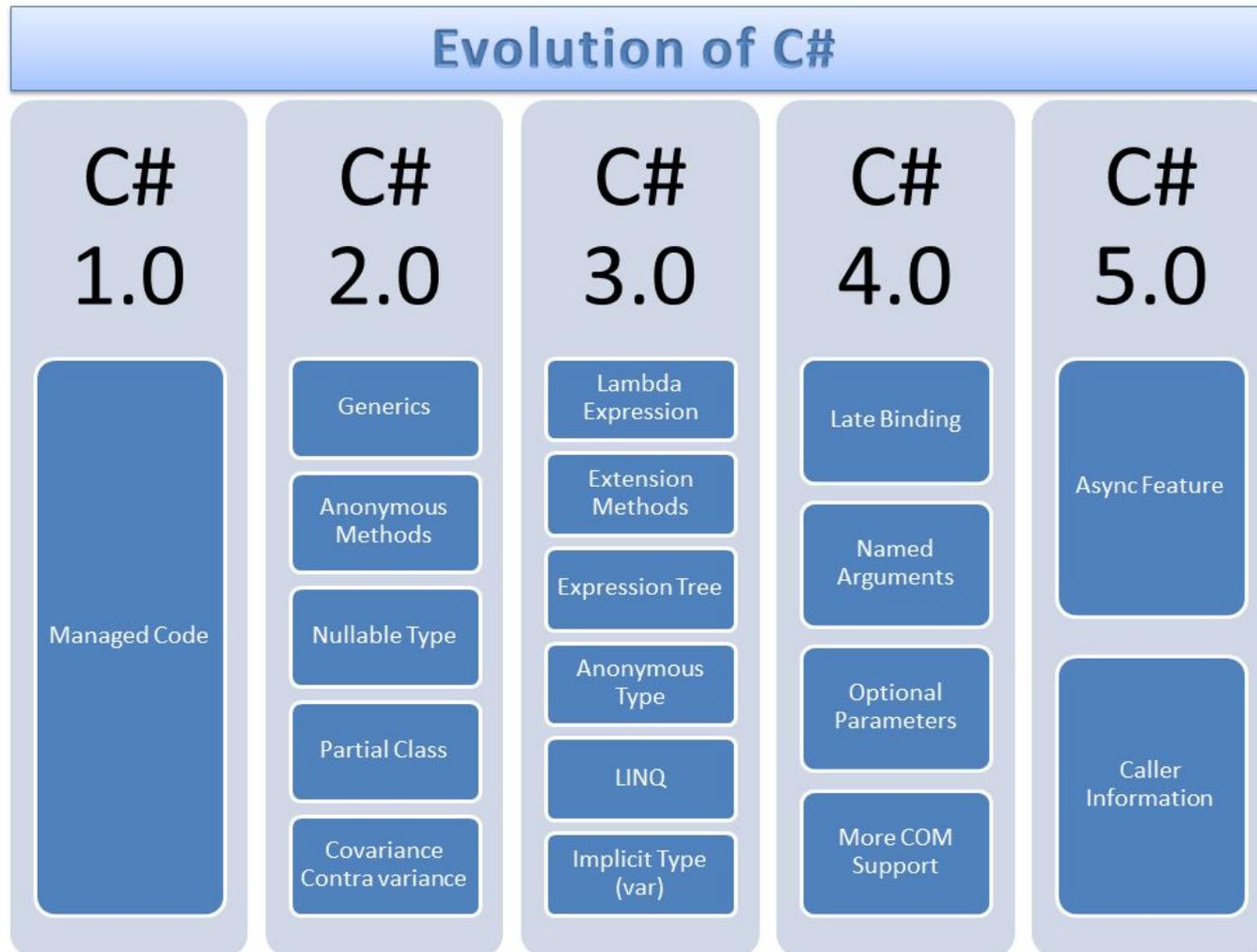
- Простота: простой C-синтаксис и конструкции, короткая кривая обучаемости
- По умолчанию, C# запрещает прямое манипулирование памятью, предоставляя взамен богатую систему типов и сборку мусора
- C# программа это набор типов, все данные и код хранятся внутри типов, нет глобальных переменных и функций
- В C#, как и в Java, нет множественного наследования

Пример: "Hello world "

```
using System;

class Program
{
    static void Main()
    {
        Console.WriteLine("Hello world!");
    }
}
```

Эволюция C#



C#1

- Ссылочные типы
- Типы значений
- Примитивные типы
- Структуры
- Пространство имён
- Перечисления
- Делегаты
- Перечислители
- Атрибуты
- Обработка исключений
- Переопределение операторов

С#2 Главные особенности

- Обобщения (generics)
- Разделяемые классы
- Анонимные методы
- Итераторы
- Типы, допускающие значения NULL
- Поддержка 64-разрядных вычислений

C#2: Обобщения

Раньше:

```
public class SomeObjectContainer
{
    private object _obj;

    public SomeObjectContainer(object obj)
    {
        this._obj = obj;
    }

    public object GetObject()
    {
        return this._obj;
    }
}
```

```
SomeObjectContainer container = new SomeObjectContainer(25);
SomeObjectContainer container2 = new SomeObjectContainer(5);
```

```
Console.WriteLine((int)container.GetObject() + (int)container2.GetObject());
```

C#2: Обобщения

Сейчас:

```
public class GenericObjectContainer<T>
{
    private T _obj;

    public GenericObjectContainer(T obj)
    {
        this._obj = obj;
    }

    public T GetObject()
    {
        return this._obj;
    }
}

GenericObjectContainer<int> container = new GenericObjectContainer<int>(25);
GenericObjectContainer<int> container2 = new GenericObjectContainer<int>(5);
Console.WriteLine(container.GetObject() + container2.GetObject());
```

С#2: Обобщения

Преимущества использования обобщений

- Производительность
- Безопасность
- Повторное использование двоичного кода
- Избавление от “разбухания” кода

С#2: Разделяемый классы

- работа над большими проектами
- работа с использованием автоматически создаваемого источника кода

```
// file Employee_Work.cs
public partial class Employee
{
    public void DoWork()
    {
    }
}
```

```
// file Employee_Rest.cs
public partial class Employee
{
    public void GoToLunch()
    {
    }
}
```

C#2: Анонимные методы

Один из способов создания безымянного блока кода, связанного с конкретным экземпляром делегата.

```
string capturedString = "Done!";  
Action<string> printReverse = delegate(string text)  
{  
    char[] chars = text.ToCharArray();  
    Array.Reverse(chars);  
    Console.WriteLine(new string(chars));  
    Console.WriteLine(capturedString);  
};  
printReverse("Some text");
```

С#2: Итераторы

Раньше:

```
public IEnumerator GetEnumerator()
{
    return new IterationSampleIterator(this);
}

class IterationSampleIterator : IEnumerator
{
    private IterationSample parent;

    private int position;

    internal IterationSampleIterator(IterationSample parent)
    {
        this.parent = parent;
        position = -1;
    }

    public bool MoveNext()
    {
        if (position != parent.Values.Length)
        {
            position++;
        }
        return position < parent.Values.Length;
    }

    public object Current
    {
        get
        {
            if (position == -1 || position == parent.Values.Length)
            {
                throw new InvalidOperationException();
            }
            int index = position + parent.StartingPoint % parent.Values.Length;
            return parent.Values[index];
        }
    }

    public void Reset()
    {
        position = -1;
    }
}
```

С#2: Итераторы

Сейчас:

```
public IEnumerator GetEnumerator()  
{  
    for (int index = 0; index < Values.Length; index++)  
    {  
        yield return Values[(index + StartingPoint) % Values.Length];  
    }  
}
```

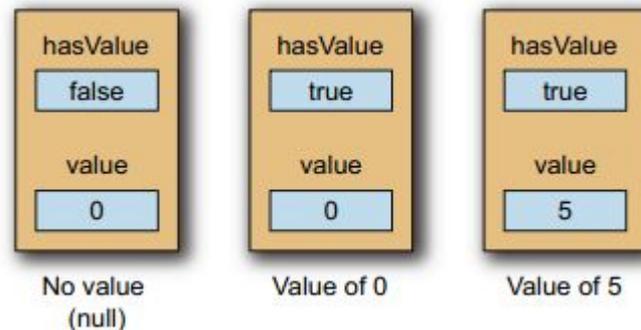
С#2: Типы, допускающие значения NULL

Раньше:

- Использование магических значений (DateTime.MinValue, double.NaN)
- Использование ссылочного класса-обертки
- Дополнительный флаг

Сейчас:

```
Nullable<int> nullable = 5;  
nullable = null;  
DateTime? anotherNullable = DateTime.Now;  
DateTime date = anotherNullable.GetValueOrDefault();  
...
```



С#3 – Главные особенности

- Неявно типизируемые локальные переменные
- Анонимные типы
- Инициализаторы объектов и коллекций
- Автоматическая реализация свойств
- Методы расширения
- Лямбда-выражения
- Деревья выражений
- Запросы LINQ

С#3: Неявно типизируемые локальные переменные

Раньше:

```
Dictionary<string, List<int>> myDictionary = new Dictionary<string, List<int>>();
```

Сейчас:

```
var myDictionary = new Dictionary<string, Dictionary<string, string>>();
```

Абсолютная необходимость для анонимных типов

```
var tempData = new { Field1 = "data1", Field2 = "data" };
```

C#3: Анонимные типы

```
var tom = new { Name = "Tom", Age = 6 };  
var holly = new { Name = "Holly", Age = 34 };  
var jon = new { Name = "Jon", Age = 33 };
```

С#3: Инициализаторы объектов

Раньше:

```
Customer c = new Customer();  
c.Name = "James";  
c.Age = 30;
```

Сейчас:

```
Customer c = new Customer { Name = "James", Age = 30 };
```

С#3: Инициализаторы коллекций

Раньше:

```
List<string> names = new List<string>();  
names.Add("Holly");  
names.Add("Jon");  
names.Add("Tom");  
names.Add("Robin");  
names.Add("William");
```

Сейчас:

```
var names = new List<string>  
{  
    "Holly", "Jon", "Tom", "Robin", "William"  
};
```

С#3: Автоматическая реализация свойств

Раньше:

```
private string name;
```

```
public string Name
```

```
{
```

```
    get
```

```
    {
```

```
        return name;
```

```
    }
```

```
private set
```

```
{
```

```
    name = value;
```

```
}
```

```
}
```

Сейчас:

```
public string Name { get; private set; }
```

C#3: Методы расширения

Определение

```
public static int WordCount(this string str)
{
    return str.Split(new char[] { ' ', '.', '?' },
                    StringSplitOptions.RemoveEmptyEntries).Length;
}
```

Использование

```
string s = "Hello Extension Methods";
int i = s.WordCount();
```

С#3: Лямбда-выражения

Раньше:

Использование анонимного метода для создания экземпляра делегата

```
Func<string, int> returnLength = delegate(string text) { return text.Length; };
```

Сейчас:

```
Func<string, int> returnLength = text => text.Length;
```

```
Func<string, int>
```

Этот делегат можно использовать для представления метода, который можно передавать в качестве параметра без явного объявления пользовательского делегата.

Идентичен:

```
public delegate int SomeDelegate(string arg1);
```

С#3: Деревья выражений

Деревья выражений представляют код в виде древовидной структуры данных, каждый узел в которой является выражением, например вызовом метода или двоичной операцией, такой как $x < y$.

Описание выражения: $2 + 3$ в виде дерева:

```
Expression firstArg = Expression.Constant(2);  
Expression secondArg = Expression.Constant(3);  
Expression add = Expression.Add(firstArg, secondArg);
```

Мы можем скомпилировать выражение в делегат и выполнить:

```
Func<int> compiled = Expression.Lambda<Func<int>>(add).Compile();  
Console.WriteLine(compiled());
```

А также представлять лямбда-выражения в дерево:

```
MethodInfo method = typeof(string).GetMethod("StartsWith", new[] { typeof(string) });  
var target = Expression.Parameter(typeof(string), "x");  
var methodArg = Expression.Parameter(typeof(string), "y");  
Expression[] methodArgs = new[] { methodArg };  
Expression call = Expression.Call(target, method, methodArgs);  
var lambdaParameters = new[] { target, methodArg };  
var lambda = Expression.Lambda<Func<string, string, bool>>(call, lambdaParameters);
```

Равноценно

```
Expression<Func<string, string, bool>> expression = (x, y) => x.StartsWith(y);
```

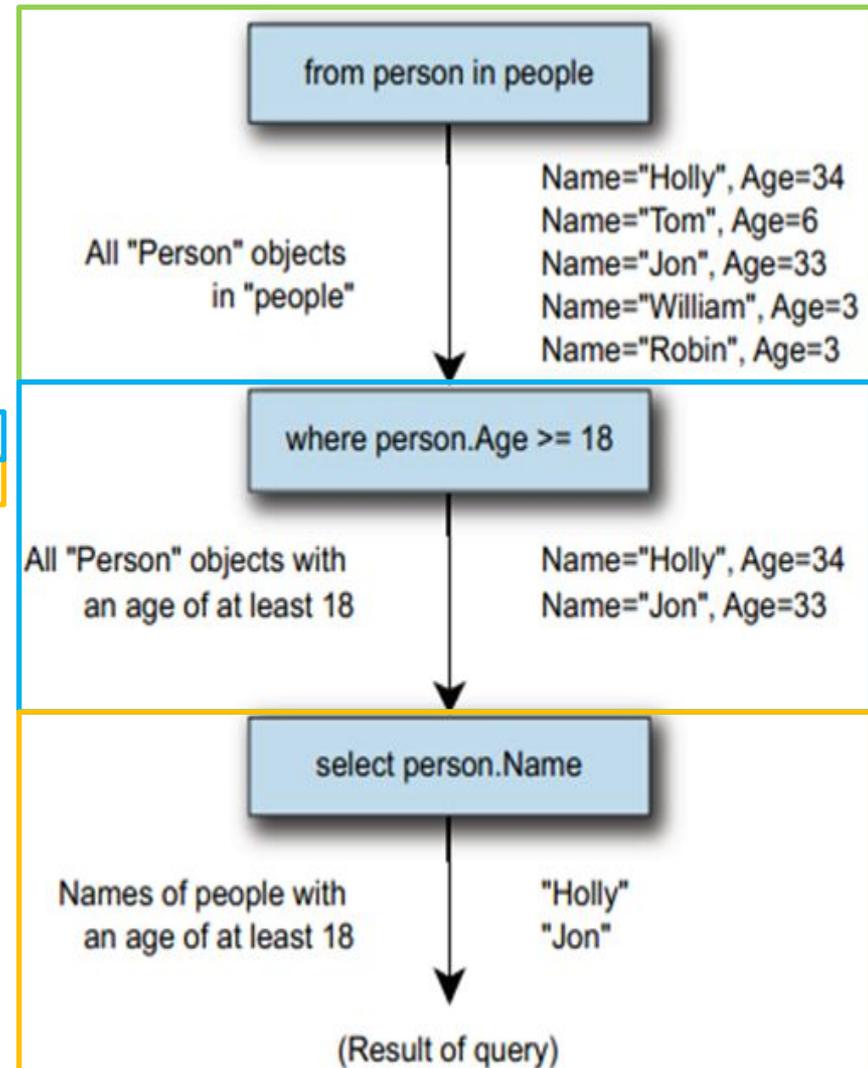
С#3: Запросы LINQ

Пример (используя выражения):

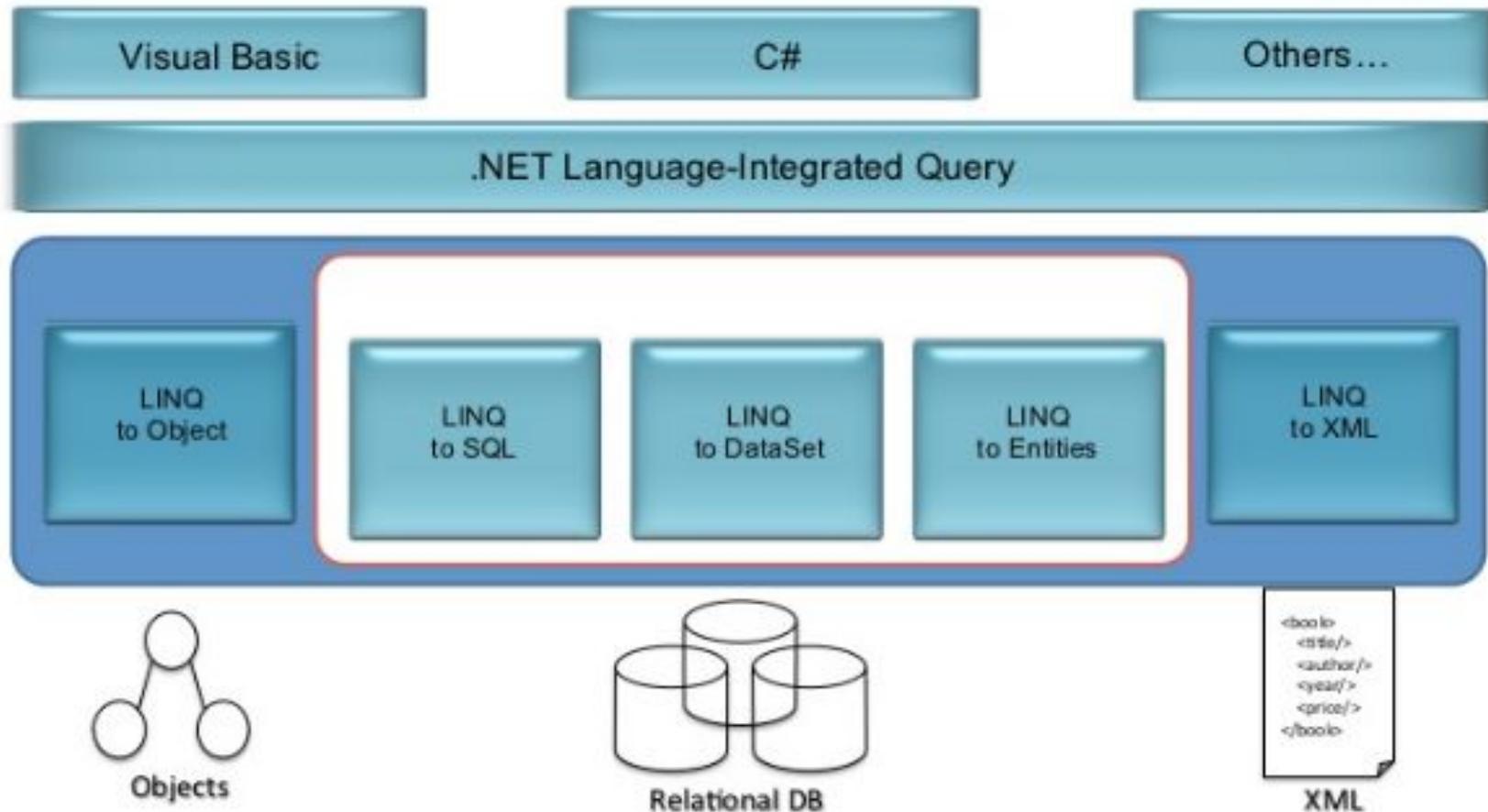
```
var people = dataContext.GetSomePeople();  
var adultNames = from person in people  
                  where person.Age >= 18  
                  select person.Name;
```

Тот же пример (используя цепочку методов):

```
var adultNames = people  
                  .Where(person => person.Age >= 18)  
                  .Select(person => person.Name);
```



С#3: Архитектура LINQ



С#4 - Главные особенности

- Именованные и необязательные аргументы
- Улучшенное взаимодействие с COM
- Динамическое позднее связывание
- Контракты в коде
- Запросы Parallel LINQ
- Обобщенная ковариантность и контрвариантность

С#4: Необязательные параметры

Раньше:

```
// call with query (assume command type is text, no parameters)
IDataReader ExecuteQuery(string query)
```

```
// call with query and command type (assume no parameters)
IDataReader ExecuteQuery(string query, CommandType commandType)
```

```
// call with query, command type, and parameters
IDataReader ExecuteQuery(string query, CommandType commandType,
    IEnumerable<DbParameter> parameters)
```

Сейчас:

```
IDataReader ExecuteQuery(string query,
    CommandType commandType = CommandType.Text,
    IEnumerable<DbParameter> parameters = null)
```

С#4: Именованные параметры

Раньше:

```
public static void Main(string[] args)
{
    var msg1 = new MailMessage("Mail 1",           // subject
                               "Body 1",           // body
                               "to@mail.com");     // to

    var msg2 = new MailMessage("Mail 1",           // subject
                               "Body 1",           // body
                               "to@mail.com",      // to
                               "from@mail.com");   // from

    var msg3 = new MailMessage("Mail 1",           // subject
                               "Body 1",           // body
                               "to@mail.com",      // to
                               "from@site.com",    // from
                               "cc@mail.com");     // cc
}
```

Сейчас:

```
var msg1 = new MailMessage(
    from: "from@site.com",
    to: "to@mail.com",
    cc: "cc@mail.com",
    subject: "Mail 1",
    body: "Body 1"
);

var msg2 = new MailMessage(
    "Mail 1",
    body: "Body 1",
    to: "to@mail.com"
);
```

С#4: Улучшенное взаимодействие с COM

Раньше:

```
object missing = Type.Missing;
var app = new Application {Visible = true};
app.Documents.Add(ref missing, ref missing,
                  ref missing, ref missing);
Document doc = app.ActiveDocument;
Paragraph para = doc.Paragraphs.Add(ref missing);
para.Range.Text = "Hello C# 4.0";

object filename = @"d:\hello.doc";
object format = WdSaveFormat.wdFormatDocument97;
doc.SaveAs(ref filename, ref format,
           ref missing, ref missing, ref missing,
           ref missing, ref missing, ref missing,
           ref missing, ref missing, ref missing,
           ref missing, ref missing);
doc.Close(ref missing, ref missing, ref missing);
app.Application.Quit(ref missing, ref missing, ref missing);
```

С#4: Улучшенное взаимодействие с COM

Сейчас:

```
var app = new Application { Visible = true };
app.Documents.Add();
Document doc = app.ActiveDocument;
Paragraph para = doc.Paragraphs.Add();
para.Range.Text = "Hello C# 4.0";

doc.SaveAs(@"d:\hello.doc", WdSaveFormat.wdFormatDocument97);
doc.Close();
app.Application.Quit();
```

С#4: Динамическое позднее СВЯЗЫВАНИЕ

Ключевое слово **dynamic** позволяет разработчику объявить объект, привязка к методам которого, будет осуществляться на этапе выполнения, а не компиляции.

```
class Person
{
    public void Go() {}
}

public static void Main(string[] args)
{
    dynamic p = new Person();
    p.Go();
    p.Go(100); // ok at compile-time
    p.Eat(); // still ok
    var name = p.name; // still ok
}
```

Добавляет возможность написания чистого кода и **взаимодействия с динамическими языками** вроде IronPython и IronRuby.

В отличие от других встроенных типов языка С# (например, string, int, object и т.п.), **dynamic** не имеет прямого сопоставления ни с одним из базовых типов ВСЛ. Вместо этого, **dynamic** – **специальный псевдоним для System.Object** с дополнительными метаданными, необходимыми для правильного позднего связывания.

C#5 – Главные особенности

- Информационные атрибуты вызываемого объекта
- Асинхронное программирование с помощью `async/await`

С#5: Информационные атрибуты вызывающего объекта

С помощью информационных атрибутов вызывающего объекта можно получить сведения об вызывающем объекте для метода. Можно получить путь к файлу исходного кода, номер строки в исходном коде и имя вызывающего объекта. Эти сведения полезны для трассировки, отладки и создания средств диагностики.

Раньше:

```
static void MethodA()
{
    InsertLog("MethodA");
    DoSomething();
}

static void MethodB()
{
    InsertLog("MethodB");
    DoSomething();
}

static void DoSomething()
{
    // Some important thing will be here.
}

static void InsertLog(string methodName)
{
    Console.WriteLine("{0} called DoSomething at {1}", methodName, DateTime.Now);
    Console.WriteLine("-----");
}
```

С#5: Информационные атрибуты вызывающего объекта

Сейчас:

```
static void MethodA()
{
    DoSomething();
}

static void MethodB()
{
    DoSomething();
}

static void DoSomething([CallerMemberName] string memberName = "",
                        [CallerFilePath] string sourceFilePath = "",
                        [CallerLineNumber] int sourceLineNumber = 0)
{
    InsertLog(memberName, sourceFilePath, sourceLineNumber);
    // Some important thing will be here.
}

static void InsertLog(string methodName, string sourceFilePath, int sourceLineNumber)
{
    Console.WriteLine("{0} called methodB at {1}", methodName, DateTime.Now);
    Console.WriteLine("Source file path: {0}", sourceFilePath);
    Console.WriteLine("Source line number: {0}", sourceLineNumber);
    Console.WriteLine("-----");
}
```

С#5: Асинхронное программирование с помощью async/await

**TO BE
CONTINUED** 

Подведем итог по C#

- Разработан в 1998—2001 годах
- Перенял многое от своих предшественников —C++, Java, Delphi, Модула и Smalltalk
- Стандартизирован в ECMA (ECMA-334) и ISO (ISO/IEC 23270)
- Простой C-подобный синтаксис
- Объектно-ориентированный
- Имеет статическую типизацию (а с версии 4 - динамическую).
- C# 1 – Управляемый код
- C# 2 – Обобщения (generics)
- C# 3 – LINQ
- C# 4 – dynamic
- C# 5 – асинхронные методы
- Высокий индекс привязанности! (dou.ua)
- Отличный инструмент для решения большого круга задач

СПАСИБО!

