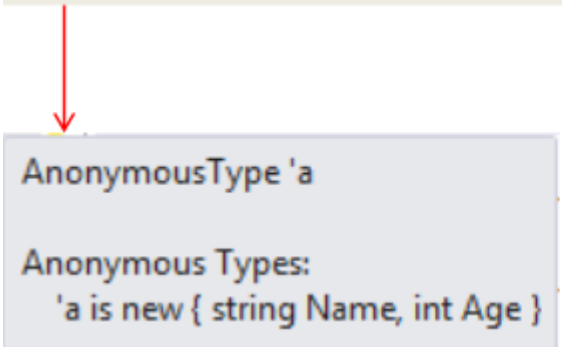
Anonymous type

Анонимные типы предлагают удобный способ инкапсуляции набора свойств в один объект без необходимости предварительного явного определения типа

```
var instance = new { Name = "Alex", Age = 27 };
```



Имя типа создается компилятором и недоступно на уровне исходного кода.

Использование анонимных типов

```
var instance = new { Name = "Alex", Age = 27, Id = new { Number = 123 } };
```

В анонимных типах могут быть вложенные анонимные типы

> Анонимные типы являются ссылочными типами, которые происходят непосредственно от класса `object`

Language Integrated Query

Language Integrated Query(LINQ) — проект Microsoft по добавлению синтаксиса языка запросов, напоминающего SQL, в языки программирования платформы .NETFramework

Все операции запроса LINQ состоят из трех различных действий

- Получение источника данных.
- Создание запроса.
- Выполнение запроса

LINQ – представляет стандартные, легко изучаемые шаблоны для создания запросов и обновления данных; технология может быть расширена для поддержки потенциально любого типа хранилища данных

Существует несколько разновидностей LINQ:

- **LINQ to Objects**: применяется для работы с массивами и коллекциями
- **LINQ to Entities**: используется при обращении к базам данных через технологию Entity Framework
- **LINQ to Sql**: технология доступа к данным в MS SQL Server
- **LINQ to XML**: применяется при работе с файлами XML
- **LINQ to DataSet**: применяется при работе с объектом DataSet
- **Parallel LINQ (PLINQ)**: используется для выполнения параллельной запросов

Простейшее определение запроса LINQ выглядит следующим образом:

```
from переменная in набор_объектов
select переменная;
```

Анонимные типы обычно используются в предложении `select` выражения запроса для возврата подмножества свойств из каждого объекта в исходной последовательности

```
var query = from x in numbers
            select new { Input = x, Output = x * 2 };
```

Список используемых методов расширения LINQ

- **Select**: определяет проекцию выбранных значений
- **Where**: определяет фильтр выборки
- **OrderBy**: упорядочивает элементы по возрастанию
- **OrderByDescending**: упорядочивает элементы по убыванию
- **ThenBy**: задает дополнительные критерии для упорядочивания элементов возрастанию
- **ThenByDescending**: задает дополнительные критерии для упорядочивания элементов по убыванию
- **Join**: соединяет две коллекции по определенному признаку
- **GroupBy**: группирует элементы по ключу
- **ToLookup**: группирует элементы по ключу, при этом все элементы добавляются в словарь
- **GroupJoin**: выполняет одновременно соединение коллекций и группировку элементов по ключу
- **Reverse**: располагает элементы в обратном порядке
- **All**: определяет, все ли элементы коллекции удовлятворяют определенному условию
- **Any**: определяет, удовлетворяет хотя бы один элемент коллекции определенному условию
- **Contains**: определяет, содержит ли коллекция определенный элемент
- **Distinct**: удаляет дублирующиеся элементы из коллекции
- **Except**: возвращает разность двух коллекцию, то есть те элементы, которые содерятся только в одной коллекции
- **Union**: объединяет две однородные коллекции
- **Intersect**: возвращает пересечение двух коллекций, то есть те элементы, которые встречаются в обоих коллекциях
- **Count**: подсчитывает количество элементов коллекции, которые удовлетворяют определенному условию
- **Sum**: подсчитывает сумму числовых значений в коллекции
- **Average**: подсчитывает среднее значение числовых значений в коллекции
- **Min**: находит минимальное значение
- **Max**: находит максимальное значение
- **Take**: выбирает определенное количество элементов
- **Skip**: пропускает определенное количество элементов
- **TakeWhile**: возвращает цепочку элементов последовательности, до тех пор, пока условие истинно
- **SkipWhile**: пропускает элементы в последовательности, пока они удовлетворяют заданному условию, и затем возвращает оставшиеся элементы
- **Concat**: объединяет две коллекции

- **Zip**: объединяет две коллекции в соответствии с определенным условием
- **First**: выбирает первый элемент коллекции
- **FirstOrDefault**: выбирает первый элемент коллекции или возвращает значение по умолчанию
- **Single**: выбирает единственный элемент коллекции, если коллекция содердит больше или меньше одного элемента, то генерируется исключение
- **SingleOrDefault**: выбирает первый элемент коллекции или возвращает значение по умолчанию
- **ElementAt**: выбирает элемент последовательности по определенному индексу
- **ElementAtOrDefault**: выбирает элемент коллекции по определенному индексу или возвращает значение по умолчанию, если индекс вне допустимого диапазона
- **Last**: выбирает последний элемент коллекции
- **LastOrDefault**: выбирает последний элемент коллекции или возвращает значение по умолчанию

Запросы LINQ

```
var query =
            employees
            .Where(emp => emp.Salary > 100000)
            .OrderBy(emp => emp.LastName)
            .OrderBy(emp => emp.FirstName)
            .Select(emp => new
            {
                LastName = emp.LastName,
                FirstName = emp.FirstName
            });
```

Любой LINQ-запрос, трансформируется в последовательность вызовов расширяющих методов

`let`-представляет новый локальный идентификатор, на который можно ссылаться в остальной части запроса.

Его можно представить, как локальную переменную видимую только внутри выражения запроса

```
var query = from emp in employees
            let fullName = emp.FirstName + " " + emp.LastName
            orderby fullName descending
            select fullName;

           foreach (var person in query)
               Console.WriteLine(person);
```

Конструкция `from` похожа на оператор `foreach`. LINQ-запрос выполнится при обращении к нему

Dynamic

```
static void Main()
{
    dynamic variable = 1;

    variable = "Hello world!";

    variable = DateTime.Now;

}
```

"События" – не могут быть типом `dynamic`

В C# 4.0 появился новый тип — `dynamic`. Тип является статическим типом, но объект типа `dynamic` обходит проверку статического типа.

В большинстве случаев он функционирует, как тип `object`.

Во время компиляции предполагается, что элементы с типом `dynamic` поддерживают любые операции.

Разработчику не нужно следить затем, откуда объект получает свое значение

В отличие от ключевого слова `var`, объект, объявленный как `dynamic`, может менять тип во время выполнения