**Делегаты**

[Делегат](http://msdn.microsoft.com/ru-ru/library/900fyy8e.aspx) - это тип, который представляет собой ссылки на методы с определенным списком параметров и возвращаемым типом. При создании экземпляра делегата этот экземпляр можно связать с любым методом с совместимой сигнатурой и возвращаемым типом. Метод можно вызвать (активировать) с помощью экземпляра делегата.

Делегаты используются для передачи методов в качестве аргументов к другим методам. Обработчики событий — это нe что иное, как методы, вызываемые с помощью делегатов. Вы создаёте свой метод, а класс, такой как элемент управления Windows, может вызывать ваш метод при возникновении определенного события. В следующем примере показано объявление делегата:

public delegate int PerformCalculation(int x, int y);

Делегату может быть присвоен любой метод, соответствующий типу делегата, из любого доступного класса или структуры. Этот метод должен быть статическим методом или методом экземпляра. Это позволяет программно изменять вызовы метода, а также включать новый код в существующие классы. В контексте перегрузки метода его сигнатура не содержит возвращаемое значение. Однако в контексте делегатов, сигнатура содержит возвращаемое значение. Другими словами, метод должен иметь тот же возвращаемый тип, что и делегат.

Благодаря возможности ссылаться на метод как на параметр, делегаты оптимально подходят для задания функций обратного вызова. Например, ссылка на метод, сравнивающий два объекта, может быть передана в качестве аргумента алгоритму сортировки. Поскольку код сравнения находится в отдельной процедуре, алгоритм сортировки может быть написан в более обобщенном виде.

[Общие сведения о делегатах](javascript:void(0))

Делегаты имеют следующие свойства.

* Делегаты похожи на указатели функций в C++, но являются типобезопасными.
* Делегаты позволяют производить передачу методов подобно обычным параметрам.
* Делегаты можно использовать для определения методов обратного вызова.
* Делегаты можно связывать друг с другом; например, при появлении одного события можно вызывать несколько методов.
* Точное соответствие методов типу делегата не требуется.
* В C# версии 2.0 введена концепция [анонимных методов](http://msdn.microsoft.com/ru-ru/library/0yw3tz5k.aspx), которые позволяют передавать блоки кода в виде параметров вместо использования отдельно определенного метода. В C# 3.0 для краткой записи встроенных блоков кода введены лямбда-выражения. В результате компиляции как анонимных методов, так и лямбда-выражений (в определенном контексте) получаются типы делегатов. В настоящее время эти возможности называются анонимными функциями.

**Лямбда-выражения**

Лямбда-выражение — это [анонимная функция](http://msdn.microsoft.com/ru-ru/library/0yw3tz5k.aspx), с помощью которой можно создавать типы [делегатов](http://msdn.microsoft.com/ru-ru/library/ms173172.aspx) или [деревьев выражений](http://msdn.microsoft.com/ru-ru/library/bb397951.aspx). С помощью лямбда-выражений можно написать локальные функции, которые затем можно передавать в другие функции в качестве аргументов или возвращать из них в качестве значения. Лямбда-выражения особенно полезны при написании выражений запросов LINQ.

Для создания лямбда-выражения можно определить входные параметры (если таковые имеются) слева лямбда-оператора [=>](http://msdn.microsoft.com/ru-ru/library/bb311046.aspx) и поместить блок выражений или выписки на другую сторону. Например, лямбда-выражение x => x \* x принимает параметр с именем x и возвращает значение x, возведённое в квадрат. Можно назначить это выражение типу делегата.

Оператор **=>** имеет такой же приоритет, как и присваивание (**=**), и является [правоассоциативным](http://msdn.microsoft.com/ru-ru/library/ms173145.aspx).

Лямбда-операторы используются в запросах LINQ с синтаксисом на основе методов в качестве аргументов методов стандартных операторов запроса, таких как [Where](http://msdn.microsoft.com/ru-ru/library/system.linq.enumerable.where.aspx).

При использовании синтаксиса на основе методов для вызова метода [Where](http://msdn.microsoft.com/ru-ru/library/system.linq.enumerable.where.aspx) класса [Enumerable](http://msdn.microsoft.com/ru-ru/library/system.linq.enumerable.aspx) (как это происходит в LINQ to Objects и LINQ to XML) параметром является тип делегата [System.Func<T, TResult>](http://msdn.microsoft.com/ru-ru/library/bb549151.aspx). Лямбда-выражение — это наиболее удобный способ создания такого делегата. При вызове того же метода, к примеру, в классе [System.Linq.Queryable](http://msdn.microsoft.com/ru-ru/library/system.linq.queryable.aspx) (как это делается в LINQ to SQL) типом параметра будет [System.Linq.Expressions.Expression](http://msdn.microsoft.com/ru-ru/library/system.linq.expressions.expression.aspx)<Func>, где Func — это любые делегаты Func с числом входных параметров не более шестнадцати. Опять же, лямбда-выражения представляют собой самый быстрый способ построения такого дерева выражений. Лямбда-выражения позволяют вызовам **Where** выглядеть одинаково, хотя, на самом деле, объекты, созданные из лямбда-выражений, будут иметь разные типы.

[**Выражения-лямбды**](javascript:void(0))

Лямбда-выражение с выражением с правой стороны оператора => называется *выражением-лямбдой*. Выражения-лямбды широко используются при создании таких конструкций как [Деревья выражений](http://msdn.microsoft.com/ru-ru/library/bb397951.aspx). Выражения-лямбды возвращают результат выражения и имеют следующую основную форму: (input parameters) => expression

Если лямбда имеет только один входной параметр, скобки можно не ставить, во всех остальных случаях они обязательны. Если входных параметров два и более, то они разделяются запятыми и заключаются в скобки: (x, y) => x == y

Иногда компилятору бывает трудно или даже невозможно вывести типы входных параметров. В этом случае типы можно указать в явном виде, как показано в следующем примере:

(int x, string s) => s.Length > x

Отсутствие входных параметров задаётся пустыми скобками.

() => SomeMethod()

[**Лямбды операторов**](javascript:void(0))

*Лямбда операторов* (или операторная лямбда) напоминает выражение-лямбду, за исключением того, что оператор (или операторы) заключается в фигурные скобки: (input parameters) => {statement;}

Тело лямбды операторов может состоять из любого количества операторов; однако на практике обычно используется не больше двух-трёх.

delegate void TestDelegate(string s);

…

TestDelegate myDel = n => { string s = n + " " + "World"; Console.WriteLine(s); };

myDel("Hello");

Лямбды операторов, как и анонимные методы, не могут использоваться для создания деревьев выражений.

[Асинхронные лямбда-выражения](javascript:void(0))

Можно легко создавать лямбда-выражения и операторы, которые включают асинхронную обработку с помощью ключевых слов [async](http://msdn.microsoft.com/ru-ru/library/hh156513.aspx) и (use "await" instead of "направит"; translation corrector doesn't allow to change links) [направит](http://msdn.microsoft.com/ru-ru/library/hh156528.aspx).

[Лямбды со стандартными операторами запросов](javascript:void(0))

Многие стандартные операторы запросов имеют входной параметр, тип которого принадлежит семейству универсальных методов-делегатов [Func<T, TResult>](http://msdn.microsoft.com/ru-ru/library/bb549151.aspx). Эти делегаты используют параметры типа для определения количества и типов входных параметров, а также тип возвращаемого значения делегата. Делегаты **Func** очень полезны для инкапсуляции пользовательских выражений, которые применяются к каждому элементу в наборе исходных данных.

[Область действия переменной в лямбда-выражениях](javascript:void(0))

Лямбда-выражения могут ссылаться на *внешние переменные* находящиеся в области метода, в котором определена лямбда-функция, или в области типа, который содержит лямбда-выражение. Переменные, полученные таким способом, сохраняются для использования в лямбда-выражениях, даже если бы в ином случае они оказались за границами области действия и уничтожились сборщиком мусора. Внешней переменной должно быть присвоено определённое значение, прежде чем она сможет использоваться в лямбда-выражениях.

Следующие правила применимы к области действия переменной в лямбда-выражениях.

* Захваченная переменная не будет уничтожена сборщиком мусора до тех пор, пока делегат, который на нее ссылается, не перейдет в статус подлежащего уничтожению при сборке мусора.
* Переменная, объявленная в лямбда-выражении, невидима во внешнем методе.
* Лямбда-выражение не может непосредственно захватывать параметры **ref** или **out** из метода, в котором они находятся.
* Оператор Return в лямбда-выражении не приводит к возврату (завершению) метода, в котором объявлено/вызвано лямбда-выражение.
* Лямбда-выражение не может содержать оператора **goto**, оператора **break** или оператора **continue** внутри лямбда-функции, если целевой объект перехода находится вне блока. Если целевой объект находится внутри блока, то наличие оператора перехода за пределами лямбда-функции также будет ошибкой.