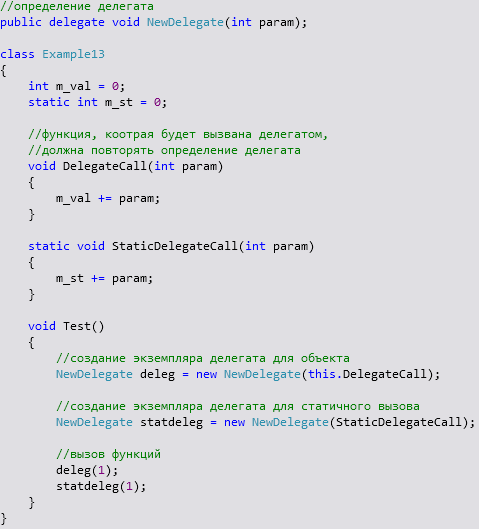
Делегаты и события

# Делегаты.

# Синтаксис.

Делегаты – специальные классы, являющиеся оберткой для вызова других функций.



Рассмотрим этот пример подробно:



Здесь создается новый делегат *NewDelegate*, который может вызывать функции, у которых один параметр типа Int32 и которые ничего не возвращают.



Эти 2 функции могут быть использованы объявленным делегатом. Они отличаются между собой тем, что одна статичная, а другая – нет, но набор параметров одинаковый.



Создается экземпляр делегата, используется нестатичная функция, поэтому требуется указывать, к какому объекту принадлежит метод.



Аналогично для статичной функции. В этом случае не нужно указывать объект.



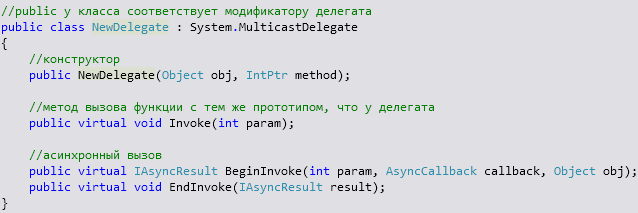
Вызов функций из делегатов одинаковый.

# Внутренняя реализация.

Объявляя делегат в виде



в действительности создается новый класс приблизительно следующего типа:



Таким образом все делегаты – классы, наследующиеся от специального класса *MulticastDelegate*, от которого напрямую наследоваться нельзя. Сам класс *MulticastDelegate* наследуется от класса *Delegate*, который уже наследуется от *Object*. Такое наследование исторически сложилось. Первоначально планировалось, что будут 2 типа делегатов. Один должен был содержать только одну ссылку на метод, и такие делегаты должны были наследоваться от *Delegate*. Второй тип должен был содержать ссылку на несколько функций. В конечном итоге все делегаты могут содержать несколько ссылок на функций.

Среди других полей класса *MulticastDelegate* отметим следующие:

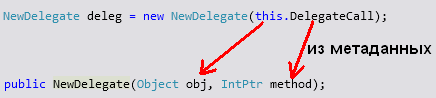
* *\_target* (*System.Object*) – ссылка на объект, чей метод будет вызван. Если метод статичный, то равна null.
* *\_methodPtr* (*System.IntPtr*) – внутренняя переменная, идентифицирующая метод для CLR в метаданных.
* *\_invocationList* (*System.Object*) – связанный список делегатов при создании цепочки делегатов.

Для *\_target* и *\_methodPtr* есть публичные readonly свойства. Свойство *Target* фактически возвращает значение поля *\_target*. Свойство *Method* преобразует *\_methodPtr* в объект типа *System.Reflection.MethodInfo*.

При создании экземпляра делегата:

**

вызывается конструктор сгенерированного класса. Параметром ob*j* здесь будет экземпляр класса, чей метод будет вызываться. Параметр *method* получает CLR получая идентификатор метода из метаданных (в таблице *MethodDef*, если метод объявлен в том же модуле, *MethodRef*, если в другом).



Внутри конструктора параметры *obj* и *method* присваиваются полям ­\_*target* и *\_methodPtr* соответственно. Полю *invocationList* присваивается null.

Вызов функции делегата



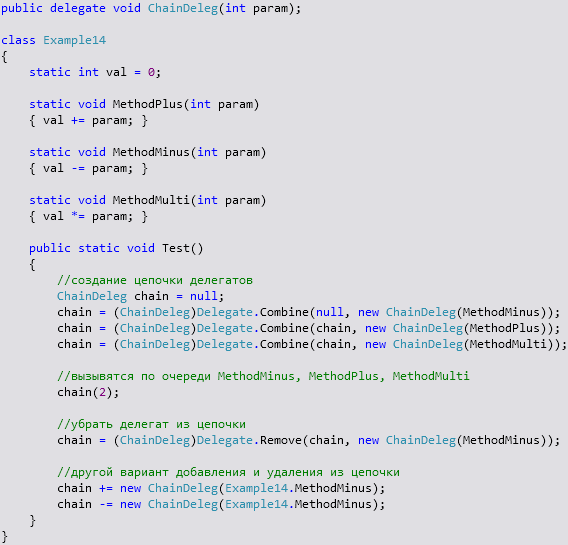
равносилен вызову метода *Invoke* делегата:



Внутри *Invoke* делегат использует внутренние поля \_*target* и *\_methodPtr*, чтобы вызвать необходимую функцию.

# Цепочки делегатов.

В предыдущих примерах делегаты были ссылками на один метод. Одна есть возможность сделать ссылкой на несколько методов. Для этого используется цепочки делегатов. Ссылкой будет тот же экземпляр делегата, однако поля *\_target* и *\_methodPtr* будут неиспользуемые, а *invocationList* будет содержать список ссылок на делегаты. При вызове *Invoke* у делегата все методы, на которые ссылается делегат, будут вызваны по очереди, как они заносились в спиисок.



Рассмотрим подробнее.



Здесь статический метод *Delegate.Combine* видит, что мы пытаемся объединить в список нулл и новый экземпляр делегата, поэтому он просто вернет ссылку на новый делегат.



Здесь метод *Delegate.Combine* видит, что оба параметры – делегаты, поэтому он создает новый экземпляр делегата, в нем инициализирует *invocationList* – создает новый список и вносит в него старый делегат, на который указывал *chain*, а также новый созданный.



Здесь метод *Delegate.Combine* снова видит, что оба параметры – делегаты, поэтому он создает еще один новый экземпляр делегата. В нем создается новый список делегатов, в который включает все делегаты, которые были в списке первого делегата, а также добавляет еще один новый созданный. Делегат, на который указывала переменная *chain*, а также старый список делегатов становятся мусором.



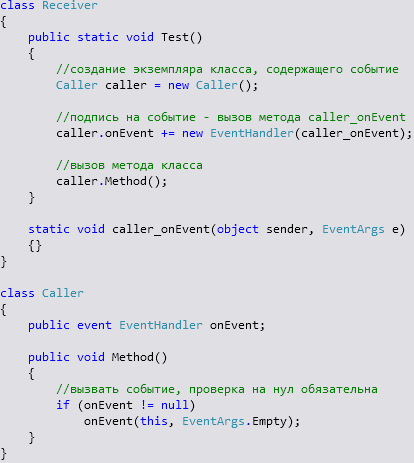
Этот оператор удаляет делегат из списка.



Эти синтаксические конструкции идентичны вызовам соответственно *Combine* и *Remove*.

# События.

Как правило, делегаты напрямую не используются, вместо них часто используются события.



Рассмотрим пример подробнее.



Здесь объявляется событие, видимое вне класса. *EventHandler* – стандартный делегат в .NET, используемый для простых событий. Этот делегат определяет прототип методов, которые могут подписаться на это событие. Он выглядит следующим образом:



Параметр *sender*, как правило (рекомендуется передавать), содержит указатель на объект, вызвавший событие. Класс *EventArgs* является пустым, не содержащим никаких полей. Он рекомендуется в качестве базового класса для данных, если их нужно передавать подписавшемуся методу. Если передавать подписавшему классу нечего, рекомендуется передавать константное значение *EventArgs.Empty*.



Таким образом мы подписываемся на событие *onEvent* экземпляра класса *Caller*. При этом создается цепочка делегатов. Когда произойдет это событие, будет вызван метод *caller\_onEvent*, прототип которого совпадает с прототипом делегата *EventHandler*.



Таким образом, класс уведомляет подписавшихся методов, что произошло событие. Обратите внимание, что *onEvent* может быть нулом, если никто не подписался на событие, поэтому проверка на нул обязательна.