**Событие, это ситуация, при возникновении которой, произойдут некоторые действия. Само событие имеет определенную структуру.**

**События**

Класс, решивший иметь события, должен уметь, по крайней мере, три вещи:

* объявить событие в классе;
* вызвать событие, передав обработчику необходимые для обработки аргументы. (Под вызовом события понимается некоторый механизм, позволяющий объекту уведомить клиентов класса, что у него произошло событие);
* проанализировать, при необходимости, результаты события, используя значения выходных аргументов события, возвращенные обработчиком.

Генерируя событие, класс посылает сообщение *получателям события*  - объектам некоторых других классов. Будем называть класс, генерирующий событие, классом ***отправителем сообщения (sender)***. Класс, чьи объекты получают сообщения, будем называть классом - ***получателем сообщения (receiver)***.



**Класс sender. Делегаты и события**

***Обработчик события - это обычный метод с аргументами. Cообщение, посылаемое при генерировании события, является аналогом вызова метода. Поскольку сигнатура посылаемого сообщения должна соответствовать сигнатуре принимаемого сообщения, то объявление события синтаксически должно задавать сигнатуру метода.***

В C# каждое событие определяется делегатом, описывающим сигнатуру сообщения. Объявление события - это двухэтапный процесс:

* Вначале объявляется делегат - функциональный класс, задающий сигнатуру. Как отмечалось при рассмотрении делегатов, объявление делегата может быть помещено в некоторый класс, например, *класс Sender*. Но, чаще всего, это объявление находится вне класса в пространстве имен. Поскольку одна и та же сигнатура может быть у разных событий, то для них достаточно иметь одного делегата.
* Если делегат определен, то в *классе Sender*, создающем события, достаточно объявить событие как экземпляр соответствующего делегата. Это делается точно так же, как и при объявлении функциональных экземпляров делегата. Исключением является добавление служебного слова **event**. Формальный синтаксис объявления таков:

**[модификаторы]event [тип, заданный делегатом] [имя события]**

Модификатором является модификатор доступа - public.

Причины возникновения события могут быть разными. Например изменение списка - при добавлении удалении или очистке будет генерироваться одно и тоже событие Changed.

delegate void MyEventHandler();

// Объявить класс, содержащий событие.

class MyEvent

{

public event MyEventHandler SomeEvent;

// Этот метод вызывается для запуска события.

public void OnSomeEvent()

{

if (SomeEvent != null)

SomeEvent();

}

}

Методы экземпляра и статические методы могут быть использованы в качестве обработчиков событий, но между ними имеется одно существенное отличие. Когда статический метод используется в качестве обработчика, уведомление о событии распространяется на весь класс. А когда в качестве обработчика используется метод экземпляра, то события адресуются конкретным экземплярам объектов.

Следовательно, каждый объект определенного класса, которому требуется получить уведомление о со­ бытии, должен быть зарегистрирован отдельно.

***Пример*** : пределено три класса. Первый класс будет считать до 100, используя цикл. Два других класса будут ждать, когда в первом классе счетчик досчитает, например, до 77, и после этого каждый выведет в консоль фразу «Пора действовать, ведь уже 77!». Проще говоря, при обнаружении значения 77, вызовутся по методу, соответственно для каждого класса. Разложим все по полкам.

**Рекомендации по обработке событий в среде .NET Framework**

В C# разрешается формировать какие угодно разновидности событий. Но ради совместимости программных компонентов со средой .NET Framework следует придерживаться рекомендаций, установленных для этой цели корпорацией Microsoft.

Эти рекомендации, по существу, сводятся к следующему требованию: у обработчиков событий должны быть два параметра. Первый из них — ссылка на объект, формирующий событие, второй — параметр типа EventArgs, содержащий любую дополнительную информацию о событии, которая требуется обработчику. Таким образом, .NET - совместимые обработчики событий должны иметь следующую общую форму.

**void *обработчик*(object *отправитель,* EventArgs е) {**

**// ...**

**}**

class MEventArgs : EventArgs

{

public int EventNum;

}

delegate void MEventHandler(object source, MEventArgs arg);

class NetEvents

{

static int count = 0;

public event MEventHandler SomeEvent;

public void OnSomeEvent()

{

MEventArgs arg = new MEventArgs();

if (SomeEvent != null)

{

arg.EventNum = count++;

SomeEvent(this, arg);

}

}

}

class XX

{

public void Handler(object source, MEventArgs arg)

{

Console.WriteLine("Событие " + arg.EventNum +

" получено объектом класса X."); Console.WriteLine("Источник: " + source); Console.WriteLine();

}

}

class YY

{

public void Handler(object source, MEventArgs arg)

{

Console.WriteLine("Событие " + arg.EventNum +

" получено объектом класса Y."); Console.WriteLine("Источник: " + source); Console.WriteLine();

}

}

***Игнорирование коллег***

Задумывались ли вы, какую роль играет ключевое слово event, появляющееся при объявлении события? Событие, объявленное в классе, представляет экземпляр делегата. В предыдущей лекции, когда речь шла о делегатах, их экземпляры объявлялись без всяких дополнительных ключевых слов.

Слово "event" играет важную роль, позволяя решить проблему, названную нами "***игнорированием коллег***". В чем ее суть? В том, что некоторые из *классов Receiver* могут вести себя некорректно по отношению к своим коллегам, занимающимся обработкой того же события. При присоединении обработчика события в *классе Receiver* можно попытаться вместо присоединения обработчика выполнить операцию присваивания, игнорируя, тем самым, уже присоединенный список обработчиков. Взгляните еще раз на процедуру OnConnect класса Receiver2; там демонстрируется такая попытка в закомментированном операторе. Аналогично, в процедуре OffConnect вместо отсоединения (операции -) можно попытаться присвоить событию значение null, отсоединяя тем самым всех других обработчиков.

***С этим как-то нужно бороться. Ключевое слово "event" дает указание компилятору создать для события закрытое поле, доступ к которому можно получить только через два автоматически создаваемых для события метода: Add, выполняющий операцию присоединения "+=", и Remove, выполняющий обратную операцию отсоединения "-=".***

Никаких других операций над событиями выполнять нельзя. Тем самым, к счастью, решается проблема *игнорирования коллег*. Ошибки некорректного поведения *класса Receiver* ловятся еще на этапе трансляции.

***Переопределение значений аргументов события***

Обработчику события, как правило, передаются входные и выходные аргументы, характеризующие событие. Они необходимы, чтобы обработчик мог нужным образом обработать событие. Но работа с аргументами требует аккуратного с ними обращения. Могут возникать проблемы, связанные с тем, что обработчик может *переопределить значения аргументов* в процессе своей работы.

Пример списка с событиями!!!

Таким образом, обработчикам можно запретить изменение входных аргументов события. Но есть еще выходные аргументы события, значения которых определяются в обработчике; в нашем примере это аргумент Permit.

И здесь возникает коллизия интересов - каждый обработчик по своему может формировать значения выходных аргументов, не обращая внимания на результаты работы предыдущих обработчиков. Преимуществом в таких ситуациях обладает последний работающий обработчик события.

Эта проблема остается открытой, в языке C# здесь "дыра" - нет специальных средств, позволяющих избежать или, по крайней мере, предупредить о возникновении подобной ситуации. Вся ответственность лежит на программисте, который может выбрать некоторую стратегию решения проблемы, отдавая, например, предпочтение решению одного из обработчиков или вырабатывая итоговое решение, учитывающее все частные решения.

Итак, если событие имеет аргументы, то все входные аргументы должны быть закрыты для обработчиков события. Если обработчиков несколько, то лучше или не использовать выходных аргументов, или аккуратно запрограммировать логику обработчиков, которая учитывает решения, полученные коллегами - ранее отработавшими обработчиками события.