**Событийно-ориентированное** программирование (event-driven programming) — парадигма программирования, в которой выполнение программы определяется событиями — действиями пользователя (клавиатура, мышь), сообщениями других программ и потоков, событиями операционной системы (например, поступлением сетевого пакета).

Событийно-ориентированное программирование, как правило, применяется в трех случаях:

- При построении пользовательских интерфейсов (в том числе графических);

- При создании серверных приложений в случае, если по тем или иным причинам нежелательно порождение обслуживающих процессов;

- При программировании игр, в которых осуществляется управление множеством объектов.

События позволяют классу или объекту уведомлять другие классы или объекты о возникновении каких-либо ситуаций.

Класс, отправляющий (или вызывающий) событие, называется **издателем(publisher)**.

Классы, принимающие (или обрабатывающие) событие, называются **подписчиками(subscriber)**.

Используйте ключевое слово **event** для объявления события в классе издателя.

В C# в стандартном приложении Windows Forms или веб-приложении пользователь подписывается на события, вызываемые элементами управления, такими как кнопки и поля со списками. Для просмотра событий, публикуемых элементом управления, и выбора некоторых из них для обработки можно воспользоваться средой IDE Visual C#. IDE автоматически добавит пустой метод обработчика событий и код, необходимый для подписки на событие.

**События имеют следующие свойства:**

* Издатель определяет момент вызова события, подписчики определяют предпринятое ответное действие.
* У события может быть несколько подписчиков. Подписчик может обрабатывать несколько событий от нескольких издателей.
* События, не имеющие подписчиков, никогда не возникают.
* Обычно события используются для оповещения о действиях пользователя, таких как нажатия кнопок или выбор меню и их пунктов в графическом пользовательском интерфейсе.
* Если событие имеет несколько подписчиков, то при его возникновении происходит синхронный вызов обработчиков событий.

В библиотеке классов .NET Framework в основе событий лежит делегат EventHandler и базовый класс EventArgs.

**Сигнатура обработчика событий должна соответствовать следующим соглашениям:**

* Метод обработчик события принимает ровно два параметра.
* Первый параметр называется sender и имеет тип Object. Это объект, вызвавший событие.
* Второй параметр называется - e и имеет тип EventArgs или тип производного класса от EventArgs. Это данные, специфичные для события.
* Тип возвращаемого значения метода обработчика — Void.

**Чтобы класс мог породить событие, необходимо подготовить три следующих элемента:**

* Класс, предоставляющий данные для события.
* Делегат события.
* Класс, порождающий событие.

**События** это особый тип многоадресных делегатов, которые можно вызвать только из класса или структуры, в которой они объявлены (класс издателя). Если на событие подписаны другие классы или структуры, их методы обработчиков событий будут вызваны, когда класс издателя инициирует событие.

**События** можно пометить как открытые (public), закрытые (private), защищенные (protected), внутренние (internal) или protected internal.

Событие можно объявить как статическое событие при помощи ключевого слова static. При этом событие становится доступным для вызова в любое время, даже если экземпляр класса отсутствует.

Событие может быть помечено как виртуальное событие при помощи ключевого слова virtual. Это позволяет производным классам переопределять поведение события при помощи ключевого слова override.

События могут быть абстрактными.

Контекстно-зависимое ключевое слово **add** используется для определения пользовательского метода доступа к событию, вызываемому при подписке клиентского кода к событию. Если указан пользовательский метод доступа **add**, то необходимо также указать метод доступа **remove**.

Контекстно-зависимое ключевое слово **remove** используется для определения пользовательского метода доступа к событию, вызываемому при отмене подписки клиентского кода от события. Если указан пользовательский метод доступа **remove**, то необходимо также указать метод доступа **add**.