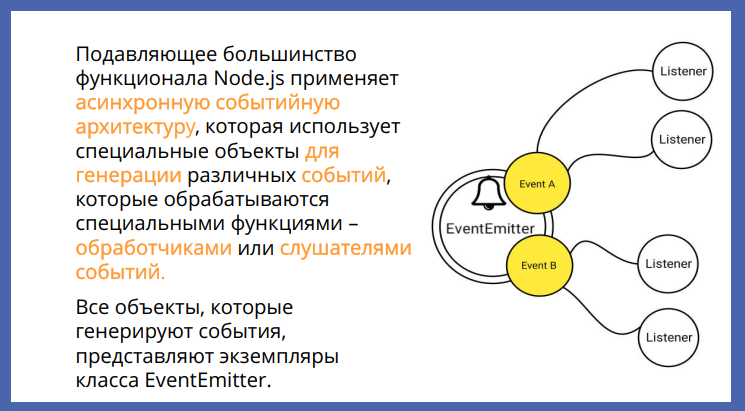
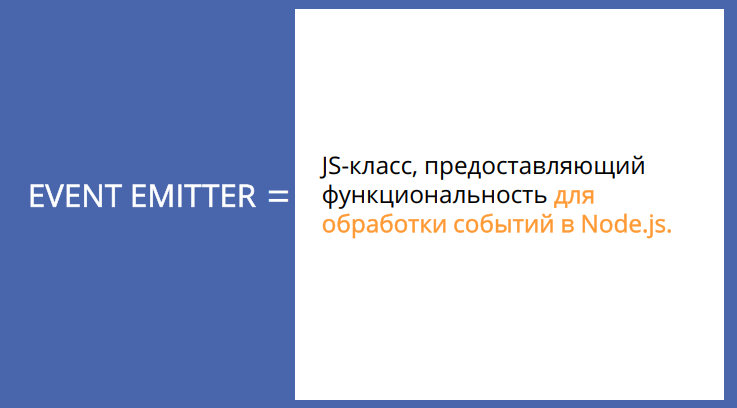
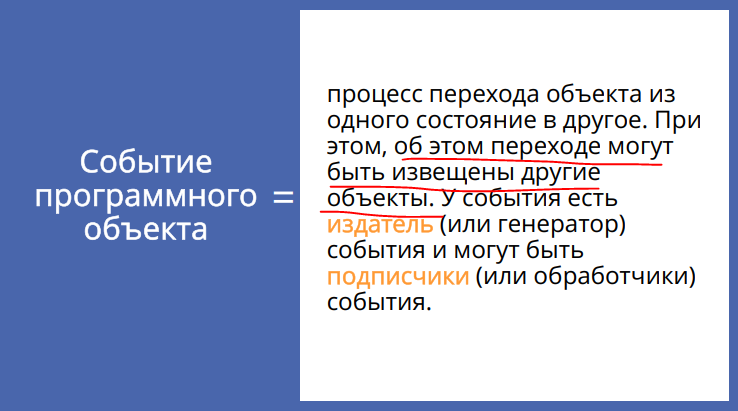
1. Дайте пояснение понятию «событие программного объекта».





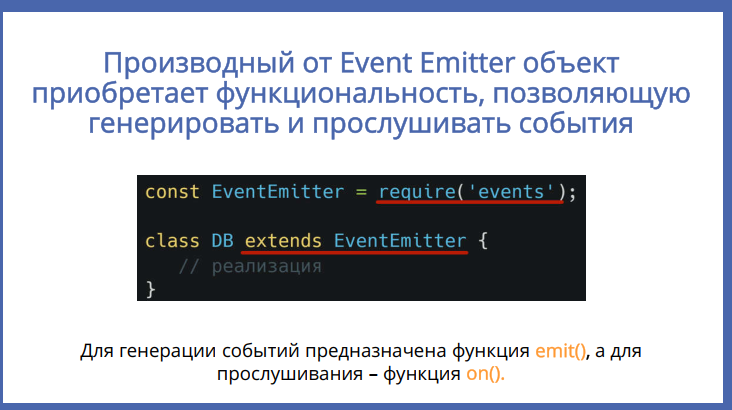


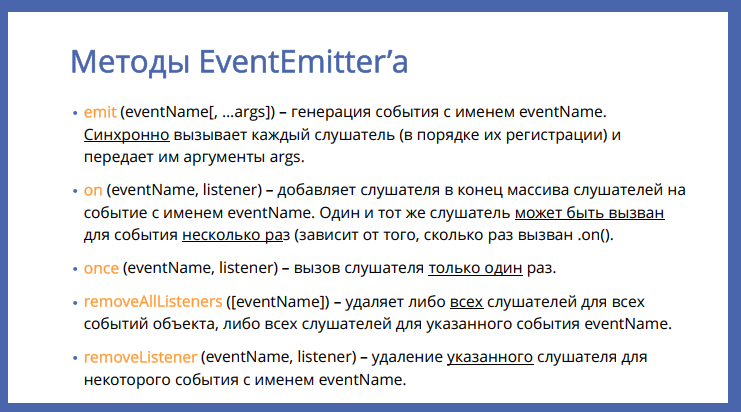
1. Какой встроенный механизм используется в **Node.js** для генерации и обработки событий. Поясните принцип его работы.

Подавляющее большинство функционала Node.js применяет асинхронную событийную архитектуру, которая использует специальные объекты - **эмиттеры** **для генерации различных событий, которые обрабатываются специальными функциями - обработчиками или слушателями событий**.

Все **объекты, которые генерируют события, представляют экземпляры класса EventEmitter.**

**С помощью функции eventEmitter.on() к определенному событию по имени цепляется функция обработчика**. Причем для одного события можно указать множество обработчиков. Когда объект EventEmitter генерирует событие, происходит выполнение всех этих обработчиков.





**Событийно-ориентированное программирование** (*event-driven programming*) - **парадигма программирования, в которой выполнение программы определяется событиями**. *Event-driven programming (EDP)* ещё переводят как *программирование, управляемое событиями.*

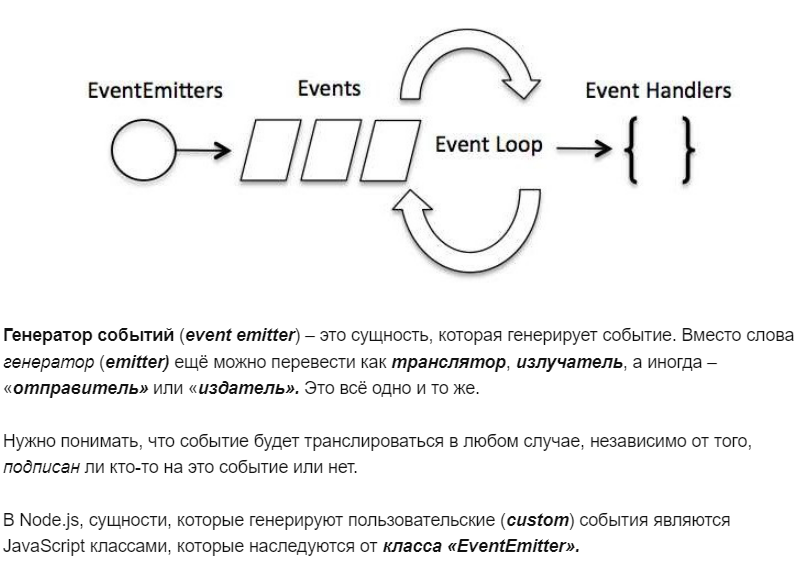
**Событие***(event)* – **какое-то значимое событие, возникающее в программируемой среде (среде выполнения кода) и на которые можно отреагировать (обработать событие).** Таким образом, программист может явно задать конкретное поведении программы при наступлении конкретного события.

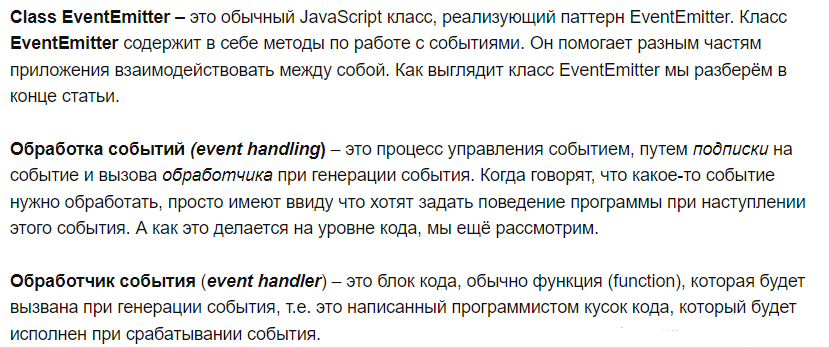
Основные среды выполнения JavaScript – **это Node.js и браузеры**. События можно **разделить на системные (system) и пользовательскими (custom).**

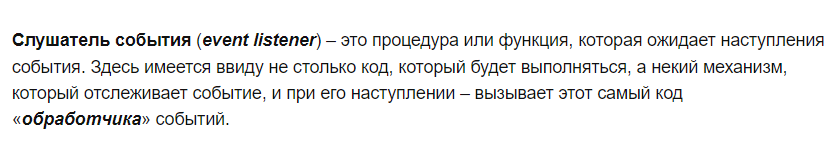
**Системные события***(system events)***– это низкоуровневые события, информацию о которых Node.js получает из операционной системы**. За обработку системных событий отвечает лежащая в основе ядра Node.js **библиотека libuv**, написанная на низкоуровневом языке программирования «C». Сам язык JavaScript слишком высокоуровневый и не имеет возможностей для обработки системных событий.

**Пользовательские события***(custom events)* – это события, которые человек создает сам. **Пользовательские события в Node.js создаются и обрабатываются с помощью специальной сущности – JavaScript класса *EventEmitter*. Имеется виду, что пользовательские события – это не те события, которые вызываются действиями человека, а те события, которые созданы программистами с использованием языка программирования JavaScript.**

**EventEmitter – это в первую очередь паттерн проектирования, суть которого заключается в том, чтобы дать возможность с любого места в нашем приложении сообщить о каком-либо событии.** Все, кто были «подписаны» на это событие сразу же об этом узнают и как-то отреагируют.







1. Наследуем класс extend EventEmitter – он перенимает базовую функциональность по генерации и обработке событий (а именно реализацию всех стандартных методов) **от EventEmitter и сможет сам генерировать события методом emit и добавлять слушателей методом on**
2. Создаем экземпляр класса, который наследуется от EventEmitter
3. Добавляем объекту класса слушателей методом on(имя\_события, listener – функция, которая сработает)
4. Генерируем у созданного объекта событие методом emit (имя\_события, параметры). Параметры будут переданы обработчикам (в функции listener) в том порядке