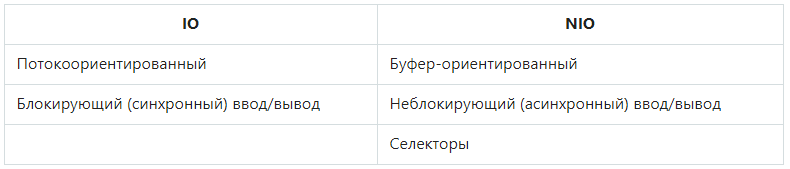
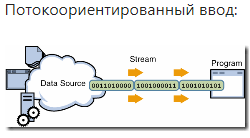
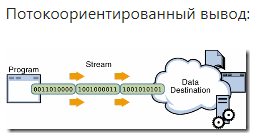
**Пакеты IO** и **NIO**



Основное отличие в том что IO – потокоориентированный, a NIO – буфер-ориентированный.

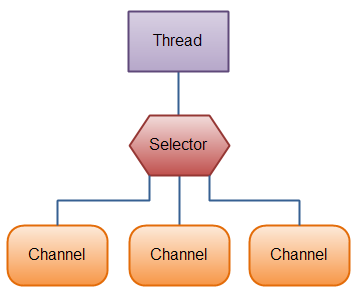
**Потокоориентированный (IO)** – подразумевает чтение/запись в поток / из потока одного или нескольких байт за единицу времени поочерёдно (информация нигде не кэшируется и невозможно двигаться по потоку вперёд или назад – что бы это осуществить нужно кэшировать в буфере).

**Буфер – ориентированный подход (NIO)** – данные сохраняются в буфер для дальнейшей обработки, появляется возможность двигаться вперёд и назад, но появляется необходимость следить за обьёмом этих данных в буфере, и что бы при чтении данных старые данные в буфере (ещё не обработанные) не уничтожились.

**Блокирующий ввод/вывод (IO)** – это значит что когда в потоке выполнения (tread) вызывается метод read() или write() метод любого класса из пакета java.io, проиходит блокировка до тех пор, пока данные не будут считаны или записаны. Поток в этот момент не может делать ничего другого.

**Неблокирующий ввод/вывод (NIO)** – может запрашивать считанные данные из канала (каналы – это логические (не физические) порталы, через которые осуществляется ввод/вывод данных, а буферы являются источниками или приёмниками этих переданных данных) и получить что-то, и работать дальше не ожидая пока данные будут доступны (не блокируется поток). Благодаря этому поток не простаивает и не тратит время в пустую. Есть понятие **Селектор** – который позволяет одному потоку мониторить (обслуживать) несколько каналов ввода/вывода, можно зарегестрировать несколько каналов с одним селектором.

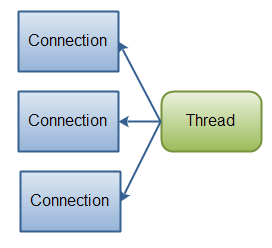


Использование IO и NIO в программе – это разные подходы, при использовании IO – вы уверены, что строка считалась полностью и можно работать. При использовании NIO – нужно проверять считалась ли строка полностью или там только часть строки.

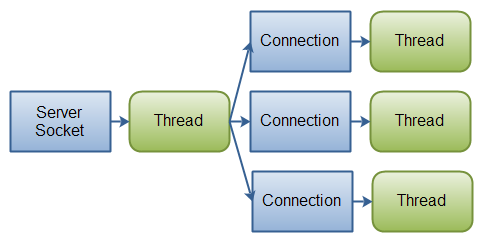
**ИТОГ:**

**Java NIO** позволяет управлять несколькими каналами (сетевыми соединениями или файлами) используя минимальное число потоков выполнения. Однако ценой такого подхода является более сложный, чем при использовании блокирующих потоков, парсинг данных. Если вам необходимо управлять тысячами открытых соединений одновременно, причем каждое из них передает лишь незначительный объем данных, выбор Java NIO для вашего приложения может дать преимущество.

Дизайн такого типа схематически изображен на следующем рисунке:



Если вы имеете меньшее количество соединений, по которым передаются большие объемы данных, то лучшим выбором станет классический дизайн системы ввода/вывода:



Важно понимать, что Java NIO отнюдь не является заменой Java IO. Его стоит рассматривать как усовершенствование – инструмент, позволяющий значительно расширить возможности по организации ввода/вывода. Грамотное использование мощи обоих подходов позволит вам строить хорошие высокопроизводительные системы. Стоит заметить, что с выходом версии Java 1.7 появился еще и Java NIO.2, но присущие ему новшества касаются, в первую очередь, работы с файловым вводом/выводом.