# Java Network Programming

# О чем это?

- NIO – New Input/Output

Основные компоненты:

- Java NIO: Channels and Buffers
- Java NIO: Non-blocking IO
- Java NIO: Selectors

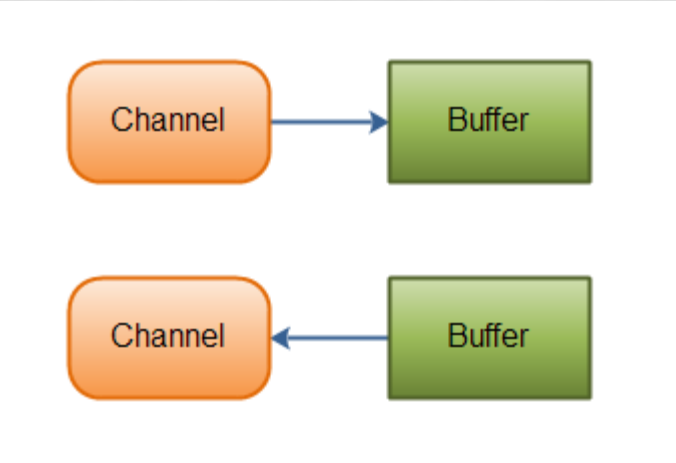
# NIO

## КАНАЛЫ, БУФЕРЫ И СЕЛЕКТОРЫ

# Каналы

# Каналы

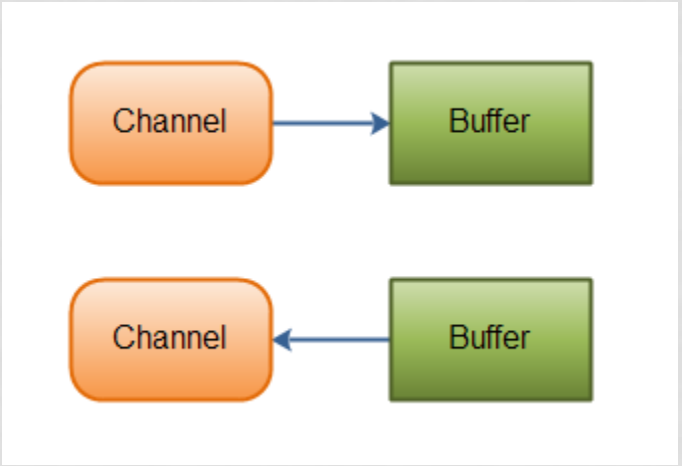
- Любой ввод-вывод в NIO завязан на *каналы* (Channel).
- Каналы напоминают Stream из обычного ввода-вывода
- Из канала данные могут прочитаны в *буфер* (Buffer)
- Данные из буфера могут быть записаны в канал



# Каналы

Каналы во многом похожи на Stream, но имеют ряд отличий:

- Можно и читать, и писать в канал. Stream обычно однонаправленны (либо чтение, либо запись)
- Каналы позволяют производить асинхронное чтение и запись
- Каналы всегда читают в буфер и пишут из буфера.



# Каналы

Основные реализации каналов в NIO:

- **FileChannel** – читает данные из файлов и в файлы
- **DatagramChannel** – передача данных по сети (UDP)
- **SocketChannel** – передача данных по сети (TCP)
- **ServerSocketChannel** – канал для приема входящих TCP соединений. На каждое новое соединение будет создан SocketChannel для общения с клиентом.

Они покрывают работу с сетью и файловый ввод-вывод

# Буферы

# Буферы

- Буфер – блок памяти в который можно писать данные и в дальнейшем считывать.

Основные наследники класса Buffer:

- ByteBuffer
- CharBuffer
- DoubleBuffer
- FloatBuffer
- IntBuffer
- LongBuffer
- ShortBuffer

# Буфер. Типичное использование

Типичный сценарий использования выглядит так:

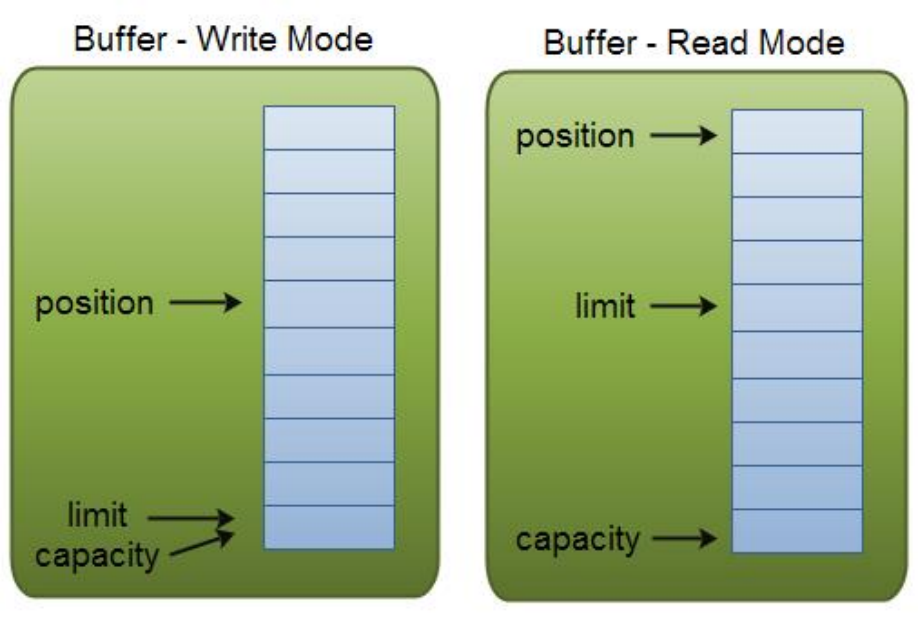
1. Запись данных в Buffer
2. Вызов buffer.flip()
3. Чтение данных из буфера
4. Вызов buffer.clear() или buffer.compact()

# Буфер. Внутреннее устройство

По смыслу, буфер – массив с
- заданной вместимостью (capacity);
- отмеченной границей до куда можно двигаться (limit)
- отмеченной текущей позицией чтения/записи (position)

Операция flip()
- ставит limit в текущее значение position
- приравнивает position к нулю

Используется для переключения между режимами чтения/записи



# Получение буфера

Для работы с буфером прежде всего необходимо выделить под него память:

- ByteBuffer buf = ByteBuffer.allocate(48);

Буфер на 48 байт

- CharBuffer buf = CharBuffer.allocate(1024);

Буфер на 1024 символа

# Запись данных в буфер

Для записи данных в буфер существуют два способа:

- Запись данных из канала в буфер

```
int bytesRead = inChannel.read(buf); //read into buffer.
```

- Запись данных вручную, используя метод put

```
buf.put(127);
```

# Чтение данных из буфера

Аналогично записи существует два способа:

- //read from buffer into channel.

```
int bytesWritten = inChannel.write(buf);
```

- byte aByte = buf.get();

# Пример

```
RandomAccessFile aFile = new RandomAccessFile("data/nio-data.txt", "rw");
FileChannel inChannel = aFile.getChannel();

ByteBuffer buf = ByteBuffer.allocate(48);

int bytesRead = inChannel.read(buf);
while (bytesRead != -1) {

        System.out.println("Read " + bytesRead);
        buf.flip();

        while(buf.hasRemaining()){
                System.out.print((char) buf.get());
        }

        buf.clear();
        bytesRead = inChannel.read(buf);
}
aFile.close();
```

# Mark и Reset

С помощью данных операций можно пометить место в буфере и потом к нему вернуться.

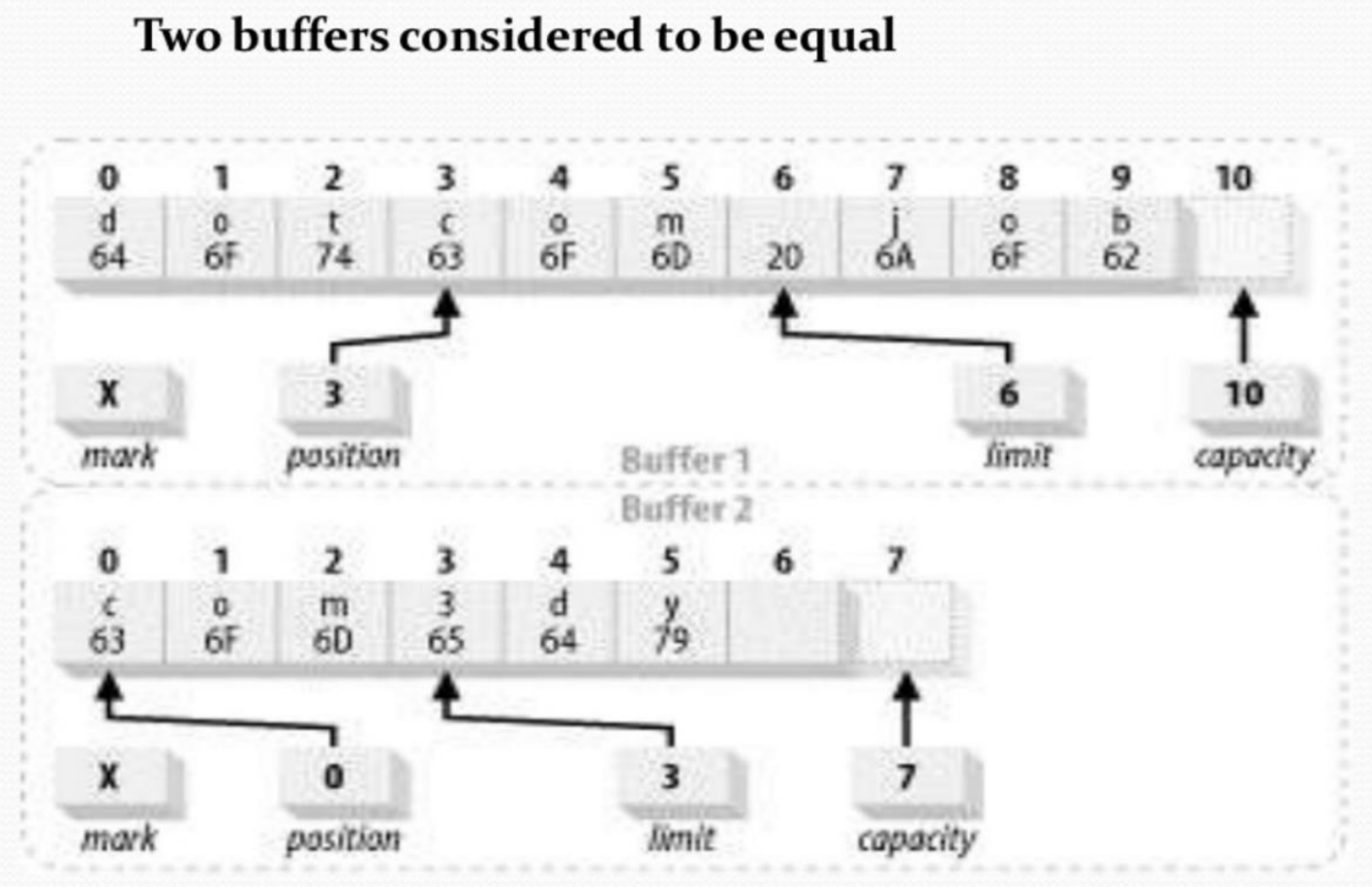
```
buffer.mark();
…
//call buffer.get() a couple of times, e.g. during parsing.
…
buffer.reset();  //set position back to mark.
```

# Сравнение буферов

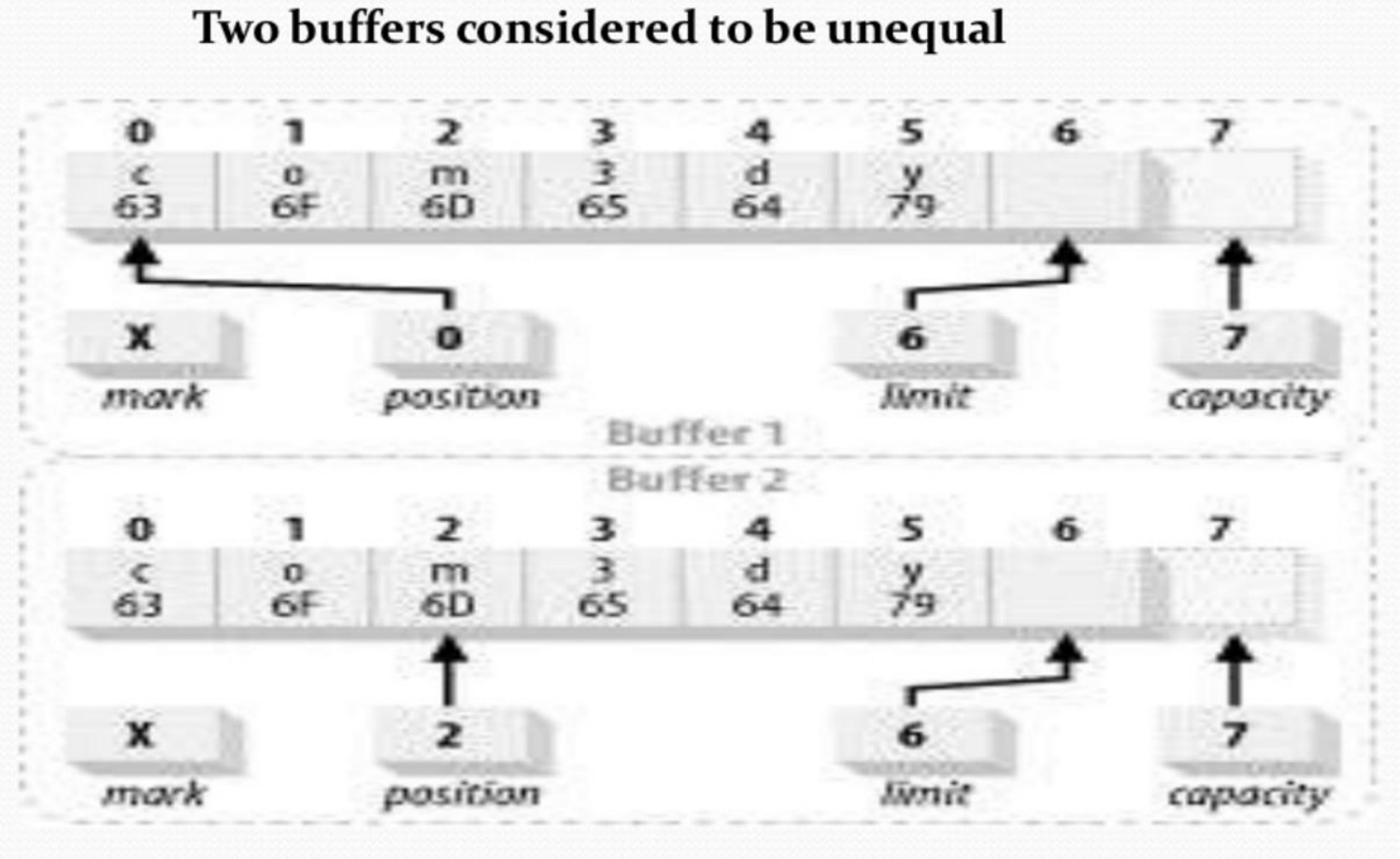
Буферы считаются одинаковыми только, если:

1. Они одного типа
2. В них осталось одинаковое количество элементов (от position до limit)
3. Последовательности оставшихся элементов, которые будут получены с помощью метода get – идентичны.

# Сравнение буферов



# Сравнение буферов



# Сравнение буферов

В случае сравнения с помощью compareTo сравниваются оставшиеся элементы в буферах. Буфер считается «меньше», если верно что-то из следующего:

- Первый отличающийся элемент в нем меньше
- Элементы одинаковые, но в нем осталось меньше элементов

# Scatter / Gather

- Scattering чтение из канала позволяет производить чтение в несколько буферов (сначала заполняем первый, потом второй и т.д.)
- Gathering запись производить запись последовательно из нескольких буферов в канал.

# Scatter / Gather

- Scattering чтение из канала позволяет производить чтение в несколько буферов (сначала заполняем первый, потом второй и т.д.)
- Gathering запись производить запись последовательно из нескольких буферов в канал.

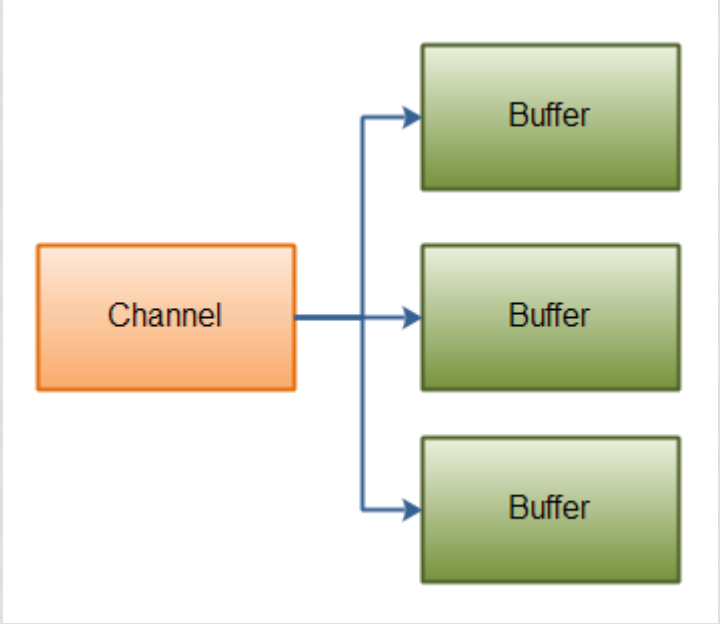
- Данные технологии очень полезны, если вам приходится работать с несколькими частями передаваемого сообщения (например, заголовок и тело сообщения).
- Однако, для такого чтения требуется, чтобы части имели наперед заданный размер

# Scattering Reads

```
ByteBuffer header = ByteBuffer.allocate(128);
ByteBuffer body   = ByteBuffer.allocate(1024);

ByteBuffer[] bufferArray = { header, body };

channel.read(bufferArray);
```

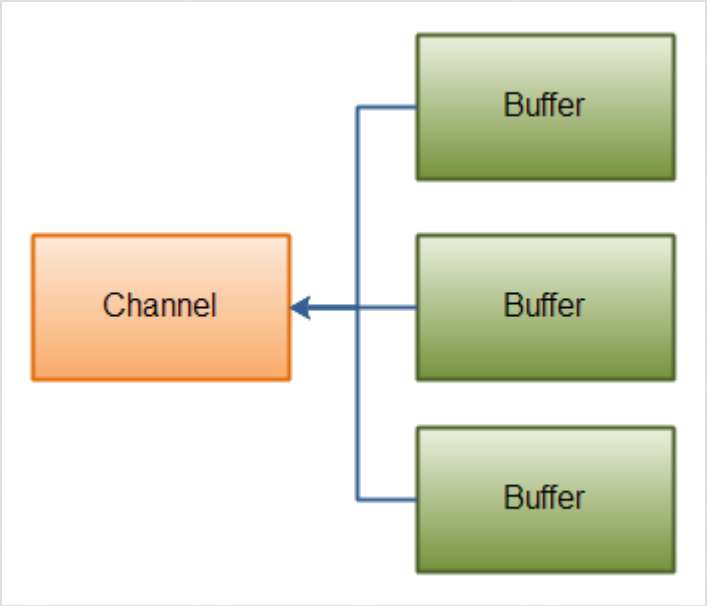


# Gathering Writes

```
ByteBuffer header = ByteBuffer.allocate(128);
ByteBuffer body   = ByteBuffer.allocate(1024);
//write data into buffers

ByteBuffer[] bufferArray = { header, body };

channel.write(bufferArray);
```

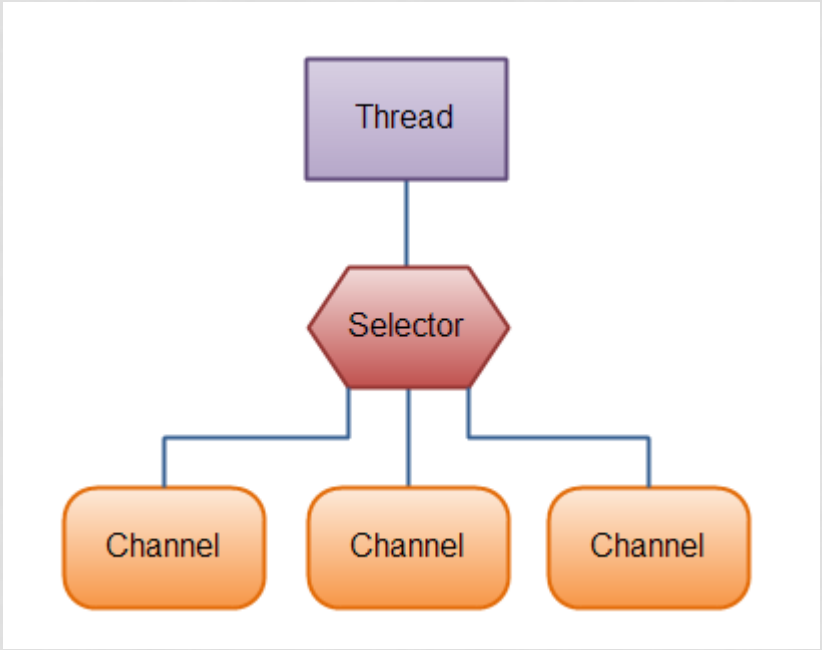


# Селекторы

# Селекторы

Селектор позволяет одному потоку оперировать одновременно с несколькими каналами. Это полезно, если, например, поддерживается большое количество одновременных соединений, но передающийся объем данных не столь высок.

Достигается это за счет того, что селектор умеет возвращать список каналов готовых для того, чтобы из них можно было читать/писать.



# Создание и регистрация

- Создание селектора:

```
Selector selector = Selector.open();
```

- Регистрация каналов в селекторе

```
channel.configureBlocking(false);
SelectionKey key = channel.register(selector, SelectionKey.OP_READ);
```

Последний параметр обозначает какие изменения в канале должен «слушать» селектор.

- SelectionKey.OP_CONNECT
- SelectionKey.OP_ACCEPT
- SelectionKey.OP_READ
- SelectionKey.OP_WRITE

Их можно объединять с помощью операции «или»

```
int interestSet = SelectionKey.OP_READ | SelectionKey.OP_WRITE;
```

# SelectionKey

SelectionKey – объект, который был получен с помощью метода register. У него есть ряд интересный свойств:

- The interest set
- The ready set
- The Channel
- The Selector
- An attached object (optional)

# SelectionKey. Interest Set

- Interest Set – набор свойств, которыми мы интересуемся.

```
int interestSet = selectionKey.interestOps();

boolean isInterestedInAccept  = interestSet & SelectionKey.OP_ACCEPT;
boolean isInterestedInConnect = interestSet & SelectionKey.OP_CONNECT;
boolean isInterestedInRead    = interestSet & SelectionKey.OP_READ;
boolean isInterestedInWrite   = interestSet & SelectionKey.OP_WRITE;
```

# SelectionKey. Ready Set

Ready Set – набор свойств, к выполнению которых готов канал.

Можно так:

```
int readySet = selectionKey.readyOps();
```

И дальше аналогично предыдущему.

Или так:

```
selectionKey.isAcceptable();
selectionKey.isConnectable();
selectionKey.isReadable();
selectionKey.isWritable();
```

# SelectionKey. Channel + Selector

Используя SelectionKey можно получить доступ к каналу и самому селектору.

```
Channel  channel  = selectionKey.channel();

Selector selector = selectionKey.selector();
```

# SelectionKey. Attaching Objects

К любому SelectionKey можно «прицепить» дополнительный объект. Это удобный способ распознать канал, на который мы реагируем. Например можно прицеплять сокет, связанный с каналом, или id канала…

Два варианта:

- selectionKey.attach(theObject);
  Object attachedObj = selectionKey.attachment();

- SelectionKey key = channel.register(selector, SelectionKey.OP_READ, theObject);

# Выбор каналов с помощью селектора

После того, как мы зарегистрировали каналы в селекторе можно использовать один из методов select. Эти методы получают каналы, которые готовы к действиям, которые нам интересны.

- int select()

Блокирующий метод. Ждет пока хотя бы один из каналов будет готов

- int select(long timeout)

Ждет не больше чем timeout

- int selectNow()

Выдает результат сразу. Возможно пустой

# Выбор каналов с помощью селектора

- Все три метода возвращают число каналов готовых к «интересным» действиям.
- Точнее сколько каналов стали готовыми, начиная с прошлого вызова методов select!
- Т.е. если однажды мы вызвали select и получили 1 (т.е. один канал готов), после чего еще один канал стал готов, то вызвав еще раз select мы снова получим 1, не зависимо от того обработали ли мы первый канал или нет.

# selectedKeys()

- После вызова select формируется set из готовых каналов.

```
Set<SelectionKey> selectedKeys = selector.selectedKeys();
```

Получаемые здесь SelectionKey ровно те, которые были при вызове метода register (в том числе содержат те же прикрепленные объекты и т.д.)

# Пример

```
Set<SelectionKey> selectedKeys = selector.selectedKeys();
Iterator<SelectionKey> keyIterator = selectedKeys.iterator();

while(keyIterator.hasNext()) {
        SelectionKey key = keyIterator.next();
        if(key.isAcceptable()) {
            // a connection was accepted by a ServerSocketChannel.
        } else if (key.isConnectable()) {
            // a connection was established with a remote server.
        } else if (key.isReadable()) {
            // a channel is ready for reading
        } else if (key.isWritable()) {
                    // a channel is ready for writing
        }
        keyIterator.remove();
}
```

# SocketChannel

# SocketChannel

SocketChannel – канал для обеспечения соединения по TCP

SocketChannel может быть получен двумя способами:

- Создан самостоятельно и привязан к какому-то серверу
- Получен на сервере при подключении нового клиента

# SocketChannel

Открытие канала:

```
SocketChannel socketChannel = SocketChannel.open();
socketChannel.connect(new InetSocketAddress("http://aptu.ru", 80));
```

Закрытие

```
socketChannel.close();
```

# Чтение

```
ByteBuffer buf = ByteBuffer.allocate(48);

int bytesRead = socketChannel.read(buf);
```

# Запись

```
String newData = "New String to write to file..." +
        System.currentTimeMillis();

ByteBuffer buf = ByteBuffer.allocate(48);
buf.clear();
buf.put(newData.getBytes());

buf.flip();

while(buf.hasRemaining()) {
    channel.write(buf);
}
```

# Неблокирующий режим

- connect()

```
socketChannel.configureBlocking(false);
socketChannel.connect(new InetSocketAddress("http://aptu.ru", 80));

while(! socketChannel.finishConnect() ){
    //wait, or do something else...
}
```

# Неблокирующий режим

- write()

В неблокирующем режиме метод может закончиться и ничего не записать. Поэтому метод надо вызывать в цикле, пока не кончится буфер.

- read()

В этом режиме метод read может не прочитать ничего. Поэтому надо внимательно следить за количеством прочитанных байт.

# ServerSocketChannel

# ServerSocketChannel

```
ServerSocketChannel serverSocketChannel = ServerSocketChannel.open();

while(true){
    SocketChannel socketChannel =
            serverSocketChannel.accept();

    //do something with socketChannel...
}

serverSocketChannel.close();
```

# Неблокирующий режим

```
ServerSocketChannel serverSocketChannel =
        ServerSocketChannel.open();

serverSocketChannel.socket().bind(new InetSocketAddress(9999));
serverSocketChannel.configureBlocking(false);

while(true){
   SocketChannel socketChannel =
        serverSocketChannel.accept();

   if(socketChannel != null){
      //do something with socketChannel...
      }
}
```

# DatagramChannel

# DatagramChannel

```
DatagramChannel channel = DatagramChannel.open();
channel.socket().bind(new InetSocketAddress(9999));

ByteBuffer buf = ByteBuffer.allocate(48);
buf.clear();
channel.receive(buf);


….

String newData = "New String to write to file..." + System.currentTimeMillis();

buf.clear();
buf.put(newData.getBytes());
buf.flip();

int bytesSent = channel.send(buf, new InetSocketAddress(“aptu.ru", 80));
```