JAVA

FUNDAMENTALS

ZEROTURNAROUND

NETWORKING

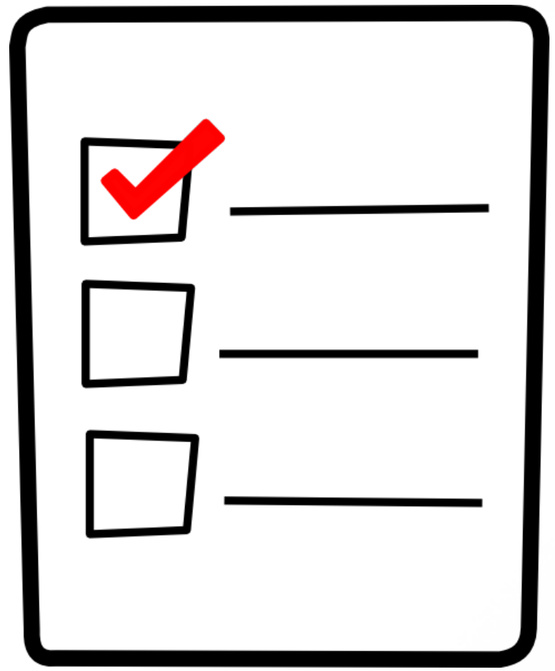
Mihhail Lapushkin

mihhail.lapushkin@zeroturnaround.com

April 24, 2017

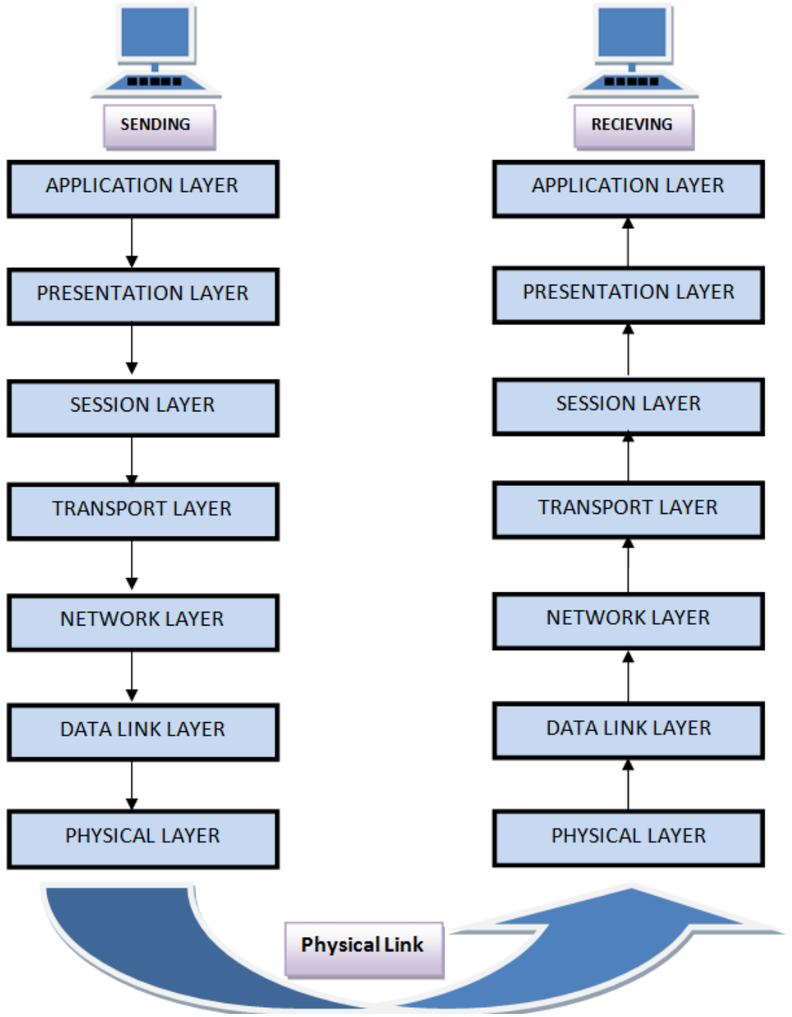
AGENDA

- Networking models
- Transport protocols
- Application protocols
- Homework



NETWORKING MODELS

OSI MODEL


SENDING
APPLICATION LAYER
PRESENTATION LAYER
SESSION LAYER
TRANSPORT LAYER
NETWORK LAYER
DATA LINK LAYER
PHYSICAL LAYER
RECIEVING
APPLICATION LAYER
PRESENTATION LAYER
SESSION LAYER
TRANSPORT LAYER
NETWORK LAYER
DATA LINK LAYER
PHYSICAL LAYER
Physical Link

TCP/IP MODEL

Application	HTTP FTP SMTP DNS SSH
Transport	TCP UDP
Network	IPv4 IPv6
Physical	Ethernet Wi-Fi

TRANSPORT PROTOCOLS

TCP

- **Transmission Control Protocol**
- Connection oriented (client - server)
- Guarantees **reliable**, **ordered** delivery
- Controls flow and congestion

UDP

- **User Datagram Protocol**
- Connectionless
- No guarantee of ordering or delivery
- Low latency

SOCKETS

- A **socket** is an endpoint for communications between two machines
- A **socket address** is a combination of an IP address and a port number
- A socket has typically two associated socket addresses — local and remote

java.net.Socket

```
new Socket()

new Socket(InetAddress address, int port)

new Socket(String host, int port)

void connect(SocketAddress endpoint)

InputStream getInputStream()

OutputStream getOutputStream()
```

java.net.ServerSocket

```
new ServerSocket()

new ServerSocket(int port)

void bind(SocketAddress endpoint)

Socket accept()
```

```
try (
    Socket s = new Socket(InetAddress.getLoopbackAddress(), 8080);
    OutputStream out = s.getOutputStream();
    InputStream in = s.getInputStream();
) {
  byte[] request = {1, 1, 2, 3, 5, 8};
  out.write(request);

  byte[] response = new byte[10];
  in.read(response);

  System.out.println("Client got: " + Arrays.toString(response));
}
```

```
try (
    ServerSocket server = new ServerSocket(8080);
    Socket client = server.accept();
    InputStream in = client.getInputStream();
    OutputStream out = client.getOutputStream();
) {
  byte[] request = new byte[10];
  int n = in.read(request);

  System.out.println("Server got: " + Arrays.toString(request));

  for (int i = 0; i < n; i++) request[i]++;
  out.write(request);
}
```

```
Server got: [1, 1, 2, 3, 5, 8, 0, 0, 0, 0]

Client got: [2, 2, 3, 4, 6, 9, 0, 0, 0, 0]
```

```
try (
    Socket s = new Socket(InetAddress.getLoopbackAddress(), 8080);
    BufferedWriter writer = new BufferedWriter(
        new OutputStreamWriter(s.getOutputStream(), "UTF-8"));
    BufferedReader reader = new BufferedReader(
        new InputStreamReader(s.getInputStream(), "UTF-8"))

) {

  writer.write("Ping\n");
  writer.flush();

  String response = reader.readLine();
  System.out.println("Client got: " + response);
}
```

```
try (
    ServerSocket server = new ServerSocket(8080);
    Socket client = server.accept();
    BufferedWriter writer = new BufferedWriter(
        new OutputStreamWriter(client.getOutputStream(), "UTF-8"));
    BufferedReader reader = new BufferedReader(
        new InputStreamReader(client.getInputStream(), "UTF-8"))
) {

  String request = reader.readLine();
  System.out.println("Server got: " + request);

  writer.write("Pong\n");
  writer.flush();
}
```

```
Server got: Ping

Client got: Pong
```

ACCEPTING MULTIPLE CONNECTIONS

- The server needs to be able to work with several clients simultaneously
- This can be achieved by using multiple **threads**

```
try (ServerSocket server = new ServerSocket(8080)) {
  ExecutorService executor = Executors.newFixedThreadPool(30);

  while (true) {
    Socket client = server.accept();
    executor.execute(() -> {
      OutputStream out = client.getOutputStream();
      InputStream in = client.getInputStream();
      // perform I/O on the client
    });
  }
}
```

java.net.DatagramSocket

```
new DatagramSocket()

new DatagramSocket(int port)

new DatagramSocket(SocketAddress addr)
```

java.net.DatagramSocket

```
void bind(SocketAddress addr)

void connect(InetAddress address, int port)

void send(DatagramPacket packet)

void receive(DatagramPacket packet)
```

```
try (DatagramSocket socket = new DatagramSocket()) {
  byte[] data = {1, 1, 2, 3, 5, 8};
  DatagramPacket request = new DatagramPacket(data, data.length);
  request.setAddress(InetAddress.getLoopbackAddress());
  request.setPort(8888);
  socket.send(request);

  DatagramPacket response = new DatagramPacket(new byte[10], 10);
  socket.receive(response);
  System.out.println("Endpoint #1 got: " +
          Arrays.toString(response.getData()) + " from " +
          response.getSocketAddress());
}
```

```
try (DatagramSocket socket = new DatagramSocket(8888)) {
  DatagramPacket request = new DatagramPacket(new byte[10], 10);
  socket.receive(request);

  System.out.println(
      "Endpoint #2 got: " + Arrays.toString(request.getData()) +
      " from " + request.getSocketAddress());

  byte[] data = Arrays.copyOf(request.getData(), request.getLength());

  for (int i = 0; i < data.length; i++) data[i]++;

  DatagramPacket response = new DatagramPacket(data, data.length);
  response.setSocketAddress(request.getSocketAddress());
  socket.send(response);
}
```

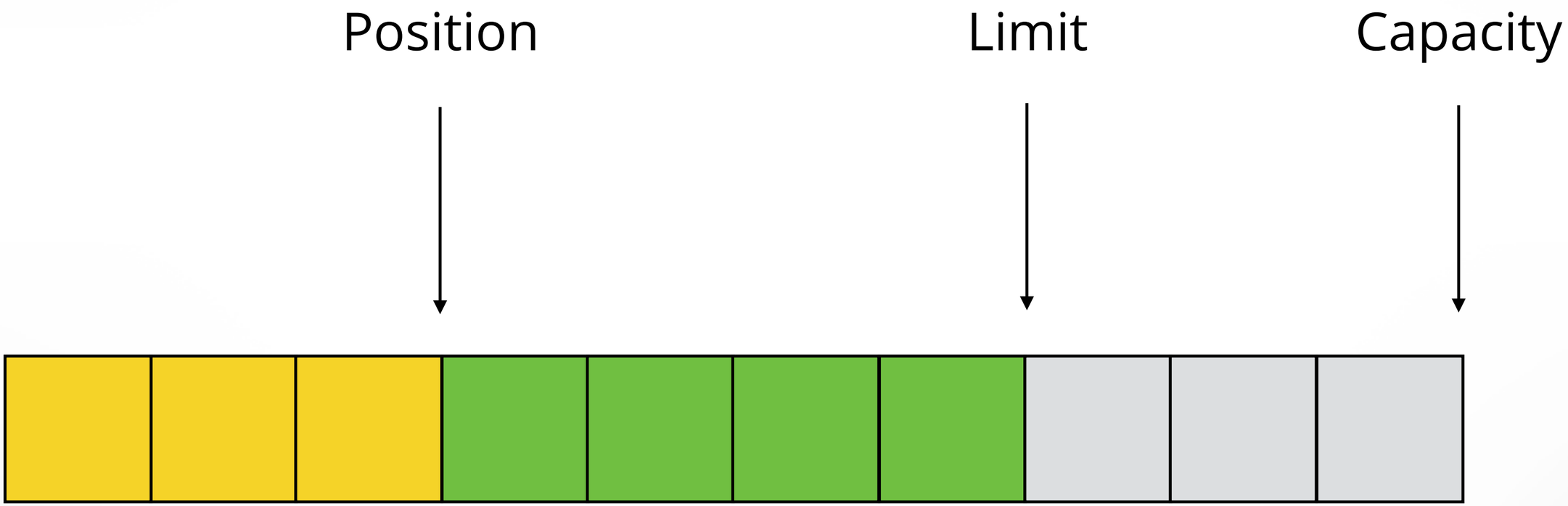
```
Endpoint #2 got: [1, 1, 2, 3, 5, 8, 0, 0, 0, 0] from /127.0.0.1:53908

Endpoint #1 got: [2, 2, 3, 4, 6, 9, 0, 0, 0, 0] from /127.0.0.1:8888
```

java.nio.channels

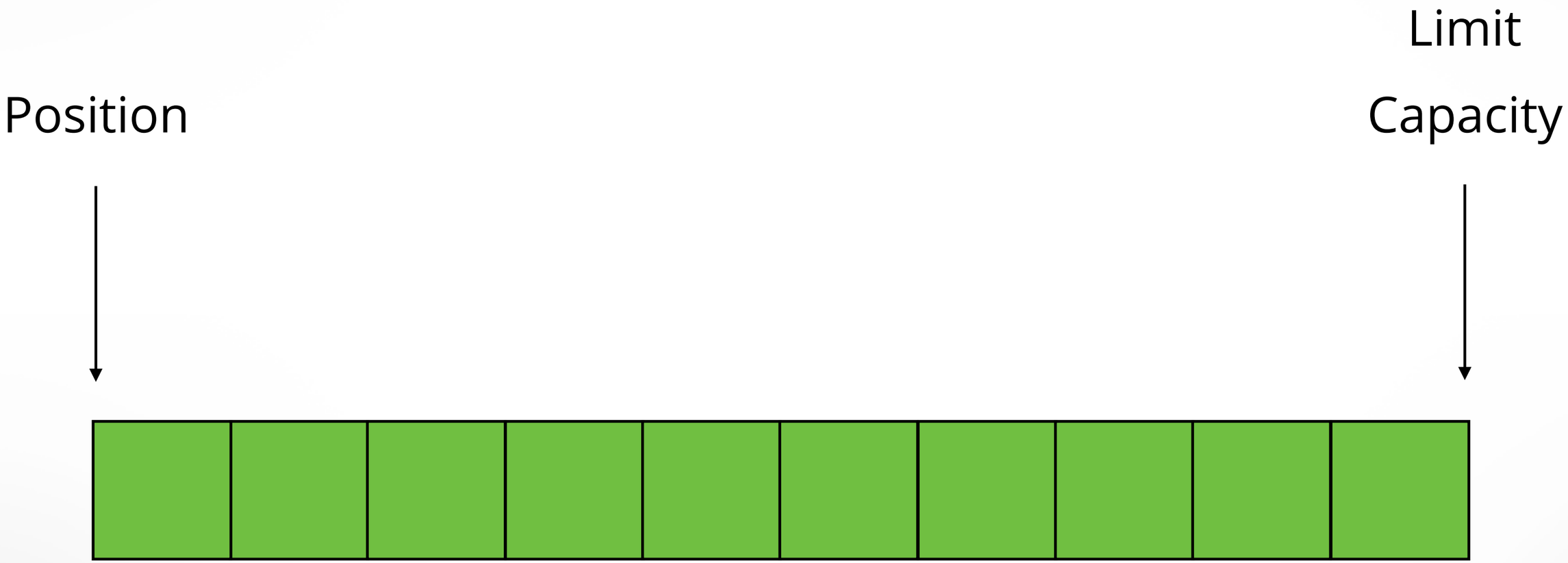
- **Socket channels**
- Access the same socket from different threads
- Manage multiple sockets in one thread
- Use special buffers for read/write operations

```
Selector selector = Selector.open();
for (int i = 0; i < 100; i++) {
  SocketChannel client = SocketChannel.open();
  client.configureBlocking(false);
  client.register(selector, OP_CONNECT | OP_READ | OP_WRITE);
  client.connect(…);
}
while (true) {
  selector.select();
  Set<SelectionKey> selectedKeys = selector.selectedKeys();
  Iterator<SelectionKey> keyIterator = selectedKeys.iterator();
  while (keyIterator.hasNext()) {
    SelectionKey key = keyIterator.next();
    if (key.isConnectable()) {
      // client connected
    }
    if (key.isWritable()) {
      // write to the socket
    }
    if (key.isReadable()) {
      // read from the socket
    }
    keyIterator.remove();
  }
}
```

java.nio.ByteBuffer

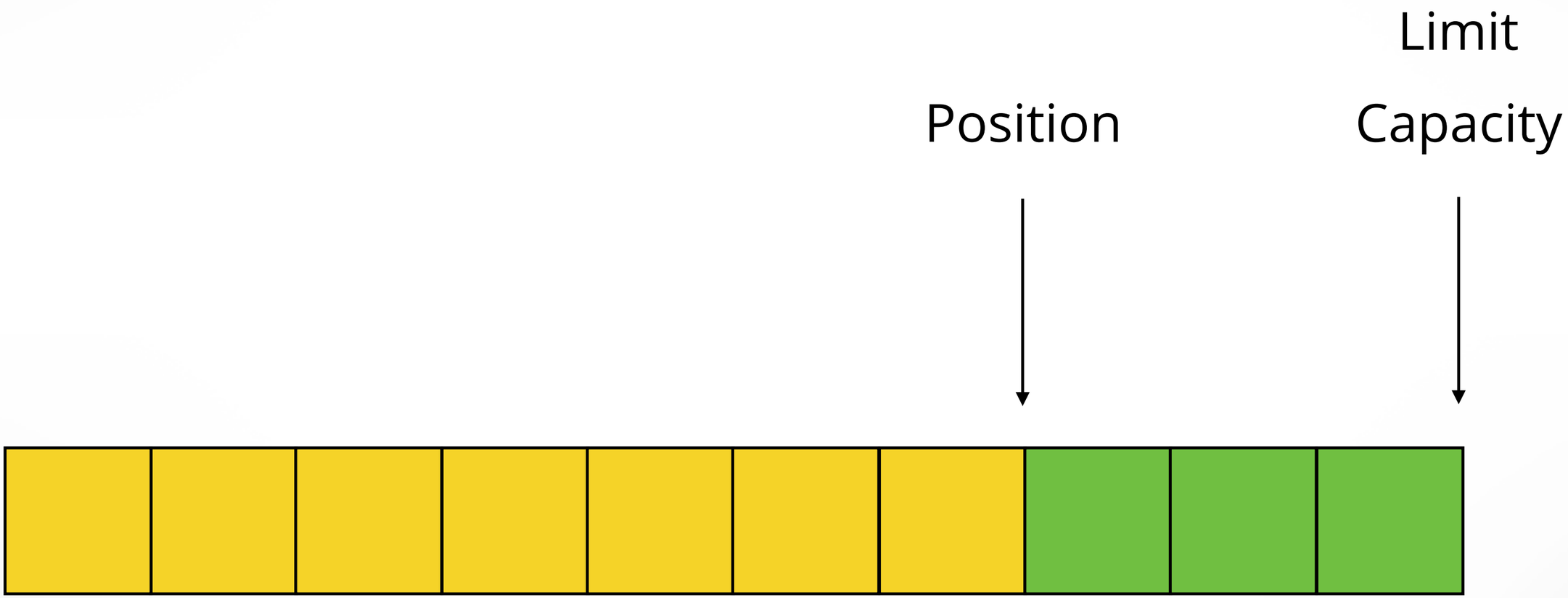


ByteBuffer.allocate()



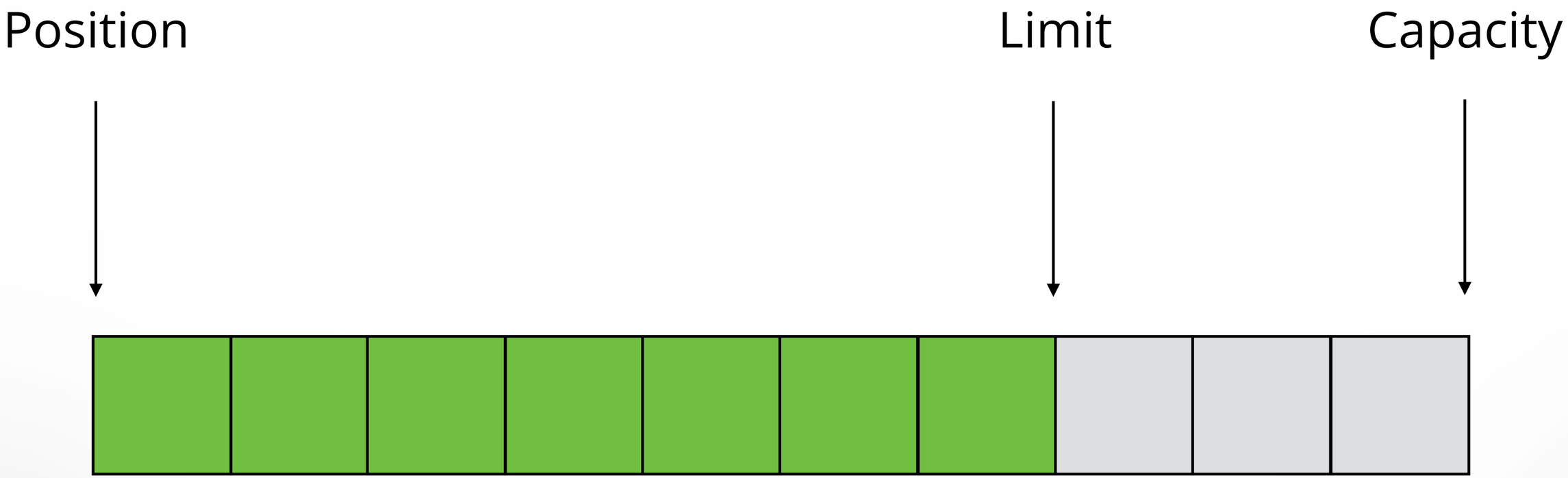
WRITE MODE

channel.read(buffer)



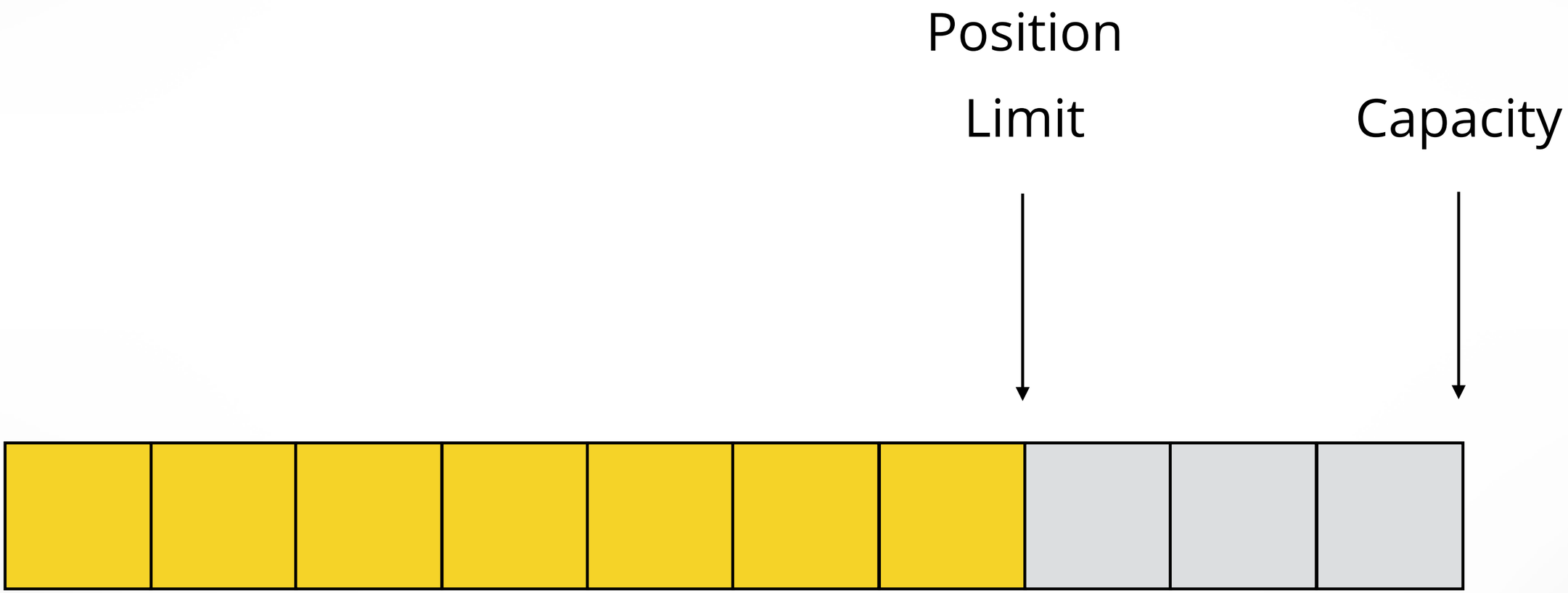
WRITE MODE

buffer.flip()



READ MODE

buffer.get(array)



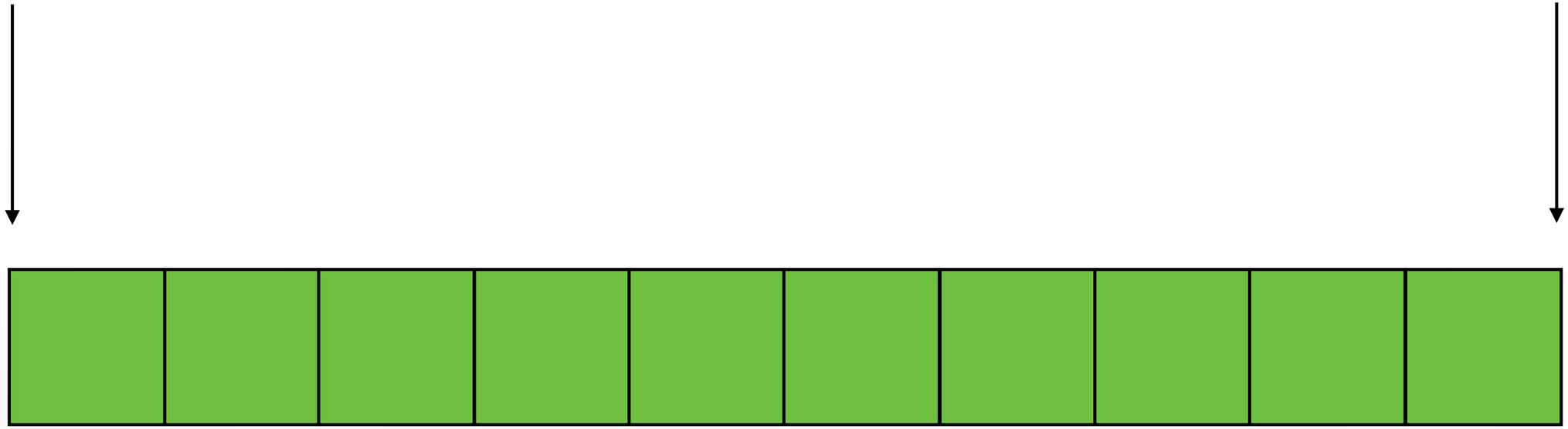
READ MODE

buffer.clear()

Limit

Position

Capacity



WRITE MODE

```
// Each client should have a unique buffer
ByteBuffer buffer = ByteBuffer.allocate(1024);
// ...
SocketChannel channel = (SocketChannel) key.channel();
if (key.isReadable()) {
   int bytesRead = channel.read(buffer);
   // check bytesRead and act accordingly
   buffer.flip();
   byte[] data = new byte[buffer.limit()];
   buffer.get(data);
   buffer.clear();
}
```

APPLICATION PROTOCOLS

HIGHER LEVEL PROTOCOLS

- TCP/UDP is fairly low level
- It can get complicated very fast
- You will likely be reinventing the wheel
- You are better off using a higher level protocol

APPLICATION LAYER PROTOCOLS

- Application layer defines the rules for formatting certain messages
- **HTTP** — web pages, REST APIs
- **FTP** — file transfer
- **SMTP** — sending email
- **RTP** — real-time audio/video streaming
- **SSH** — secure operation of remote servers

HTTP

- One of the most well-known and versatile application layer protocols
- Initially designed to transmit hypertext (web pages)
- HTTP has a simple request-response model
  - Client sends a request
  - Server sends a response
- The protocol is stateless, however state is often implemented on top

```
try (
    Socket s = new Socket("freegeoip.net", 80);
    Writer writer = new OutputStreamWriter(s.getOutputStream(), "UTF-8");
    BufferedReader reader = new BufferedReader(
      new InputStreamReader(s.getInputStream(), "UTF-8"));
) {
  writer.write(
      "GET /json/ HTTP/1.1\r\n" +
      "Host: freegeoip.net\r\n" +
      "Connection: close\r\n" + "\r\n");
  writer.flush();

  String line;
  while ((line = reader.readLine()) != null)
    System.out.println(line);
}
```

```
HTTP/1.1 200 OK
Content-Type: application/json
Vary: Origin
X-Database-Date: Wed, 07 Sep 2016 19:05:53 GMT
X-Ratelimit-Limit: 10000
X-Ratelimit-Remaining: 9993
X-Ratelimit-Reset: 2383
Date: Mon, 03 Oct 2016 06:58:53 GMT
Content-Length: 221
Connection: close

{"ip":"156.189.58.56","country_code":"EE","country_name":"Estonia","r
egion_code":"","region_name":"","city":"","zip_code":"","time_zone":"
Europe/Tallinn","latitude":59,"longitude":26,"metro_code":0}
```

java.net.URL

```
new URL(String spec)

URLConnection openConnection()

InputStream openStream()
```

```
URL url = new URL("http://freegeoip.net/json/");

try (BufferedReader reader = new BufferedReader(
     new InputStreamReader(url.openStream(), "UTF-8"))) {
  String line;

  while ((line = reader.readLine()) != null) {
    System.out.println(line);
  }
}
```

{"ip":"156.189.58.56","country_code":"EE","country_name":"Estonia","region_code":"","region_name":"","city":"","zip_code":"","time_zone":"Europe/Tallinn","latitude":59,"longitude":26,"metro_code":0}

```
URL url = new URL("http://www.asciitohex.com/");
HttpURLConnection connection = (HttpURLConnection) url.openConnection();
connection.setRequestMethod("POST");
connection.setDoOutput(true);
connection.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");

try (OutputStreamWriter writer =
  new OutputStreamWriter(connection.getOutputStream(), "UTF-8")) {
  writer.write("b64=SmF2YSBGdW5kYW1lbnRhbHM%3D");
  writer.flush();
}

try (BufferedReader reader = new BufferedReader(
    new InputStreamReader(connection.getInputStream(), "UTF-8"))) {
  String line;
  while ((line = reader.readLine()) != null)
    System.out.println(line);
}
```

```
<!doctype html>
<html lang="en">
...
<textarea name="ascii" id="ascii">Java Fundamentals</textarea>
...
```

APACHE HTTP COMPONENTS

- Nice fluent API for making HTTP requests

```
<dependency>
  <groupId>org.apache.httpcomponents</groupId>
  <artifactId>fluent-hc</artifactId>
  <version>4.5.3</version>
</dependency>
```

```
String response = Request.Get("http://freegeoip.net/json/")
    .execute()
    .returnContent()
    .asString();
System.out.println(response);

String response = Request.Post("http://www.asciitohex.com/")
    .bodyForm(Form.form()
        .add("b64", "SmF2YSBGdW5kYW1lbnRhbHM=").build())
    .execute()
    .returnContent()
    .asString();
System.out.println(response);
```

JETTY

- Allows to easily embed an HTTP server into any application

```
<dependency>
  <groupId>org.eclipse.jetty</groupId>
  <artifactId>jetty-server</artifactId>
  <version>9.4.3.v20170317</version>
</dependency>
```

```
class MyHandler extends AbstractHandler {
  @Override
  public void handle(String path,
                     Request baseRequest,
                     HttpServletRequest request,
                     HttpServletResponse response)
      throws IOException, ServletException {
    // Use request.getPathInfo() to access the path
    // and request.getQueryString() to access the query string
    baseRequest.setHandled(true);
    response.setStatus(HttpServletResponse.SC_OK);
    response.getWriter().println("Hello");
  }
}
Server server = new Server(8080);
server.setHandler(new MyHandler());
server.start();
server.join();
```

HOMEWORK 13

https://github.com/JavaFundamentalsZT/jf-hw-net