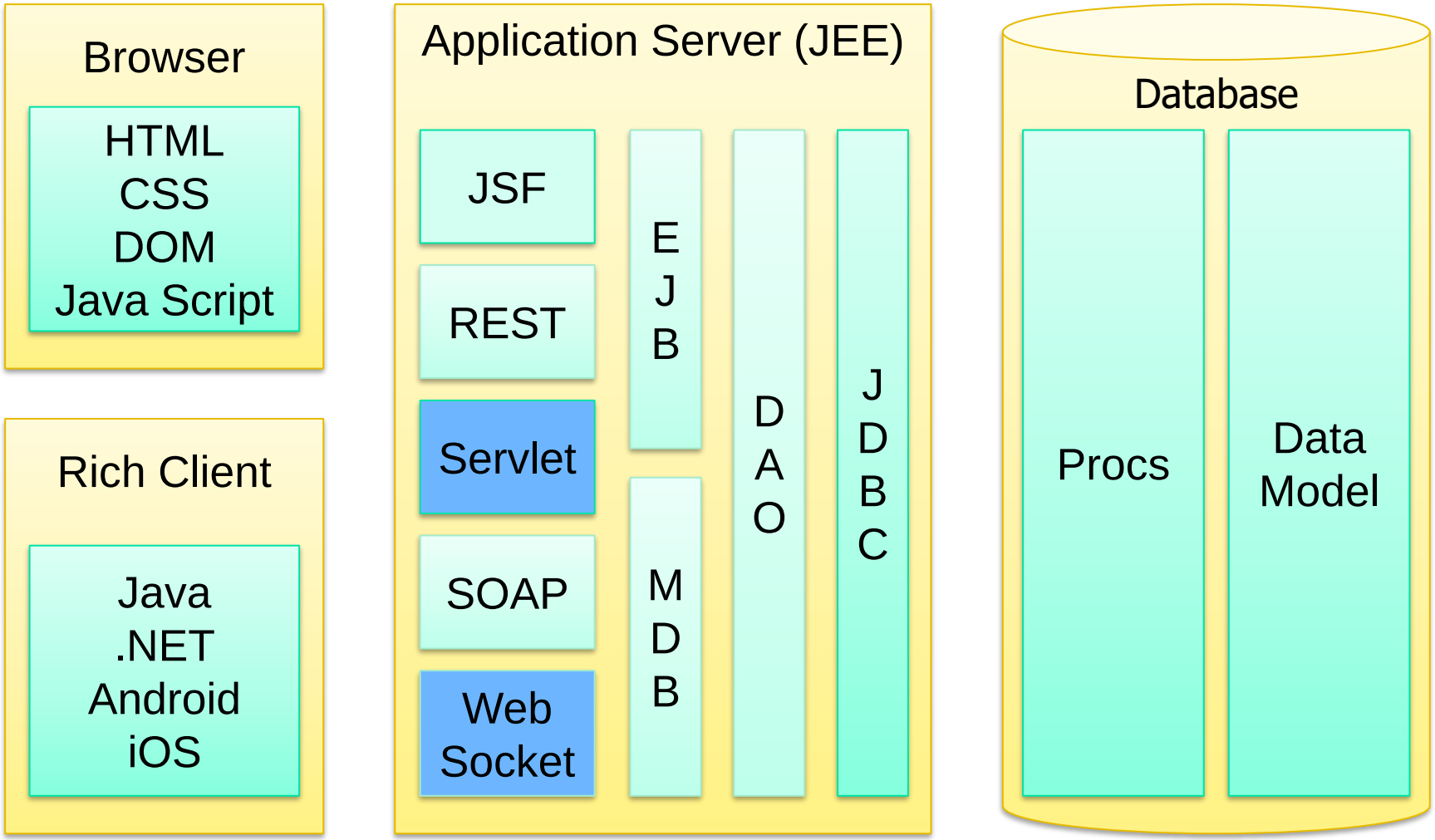
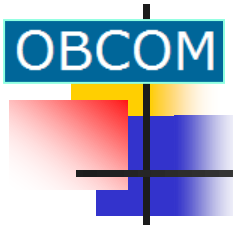
Aplicaciones Web (parte 7)

Eduardo Ostertag Jenkins, Ph.D.
OBCOM INGENIERIA S.A. (Chile)
Eduardo.Ostertag@obcom.cl

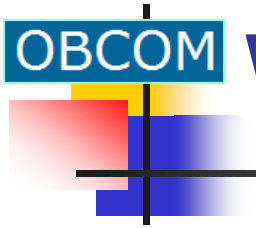
WebSocket y Servlet





Protocolo WebSocket

- Protocolo de comunicaciones que provee un canal de comunicación full-duplex a través una única conexión TCP/IP
- Full-duplex: capaz de transmitir en ambas direcciones de un canal al mismo tiempo
- Ambos, el cliente y el servidor, pueden enviar mensajes cuando quieran ☺
- El servidor ya no tiene que esperar a recibir un requerimiento del cliente ☺

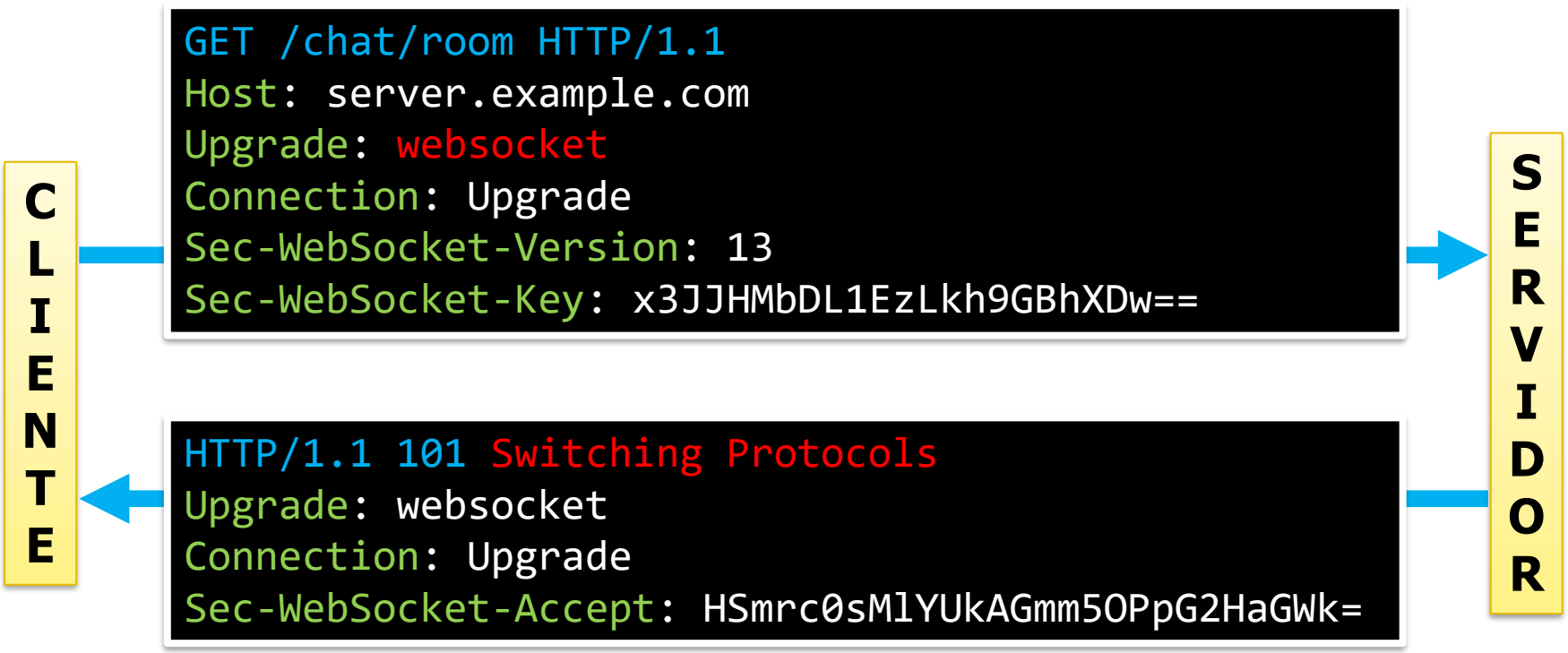


WebSocket no es HTTP

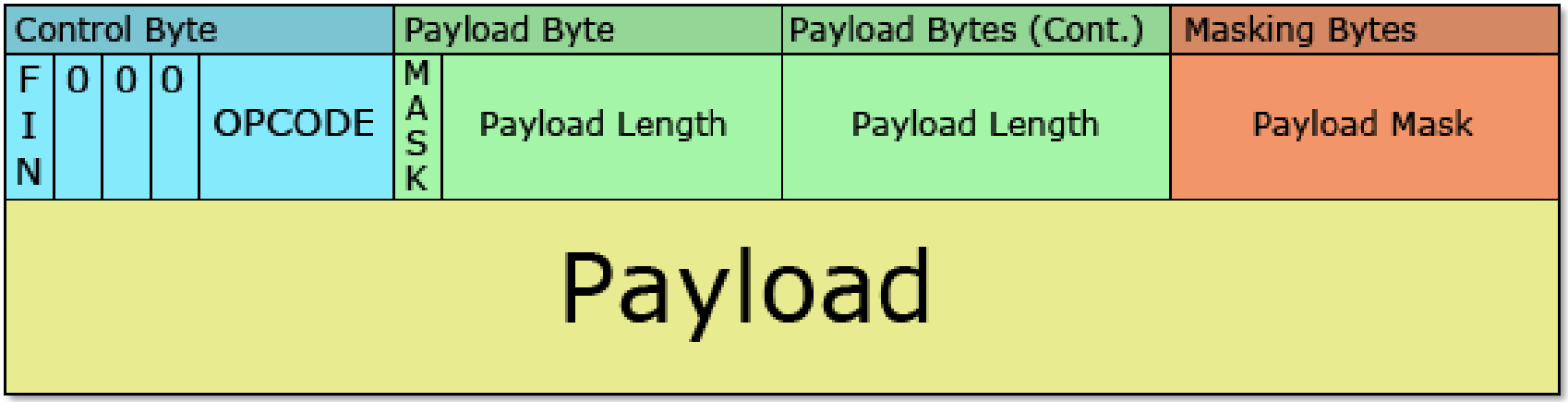
- WebSocket está diseñado para funcionar por los puertos 80 y 443 de HTTP
- WebSocket opera a través de servidores proxy y otros intermediarios HTTP
- WebSocket es compatible con HTTP, pero es distinto de HTTP
- Un canal WebSocket parte vía HTTP, pero luego se cambia con un “upgrade header”

Iniciar canal WebSocket

```
ws://host:port/path?query  (inseguro)
wss://host:port/path?query (seguro)
```

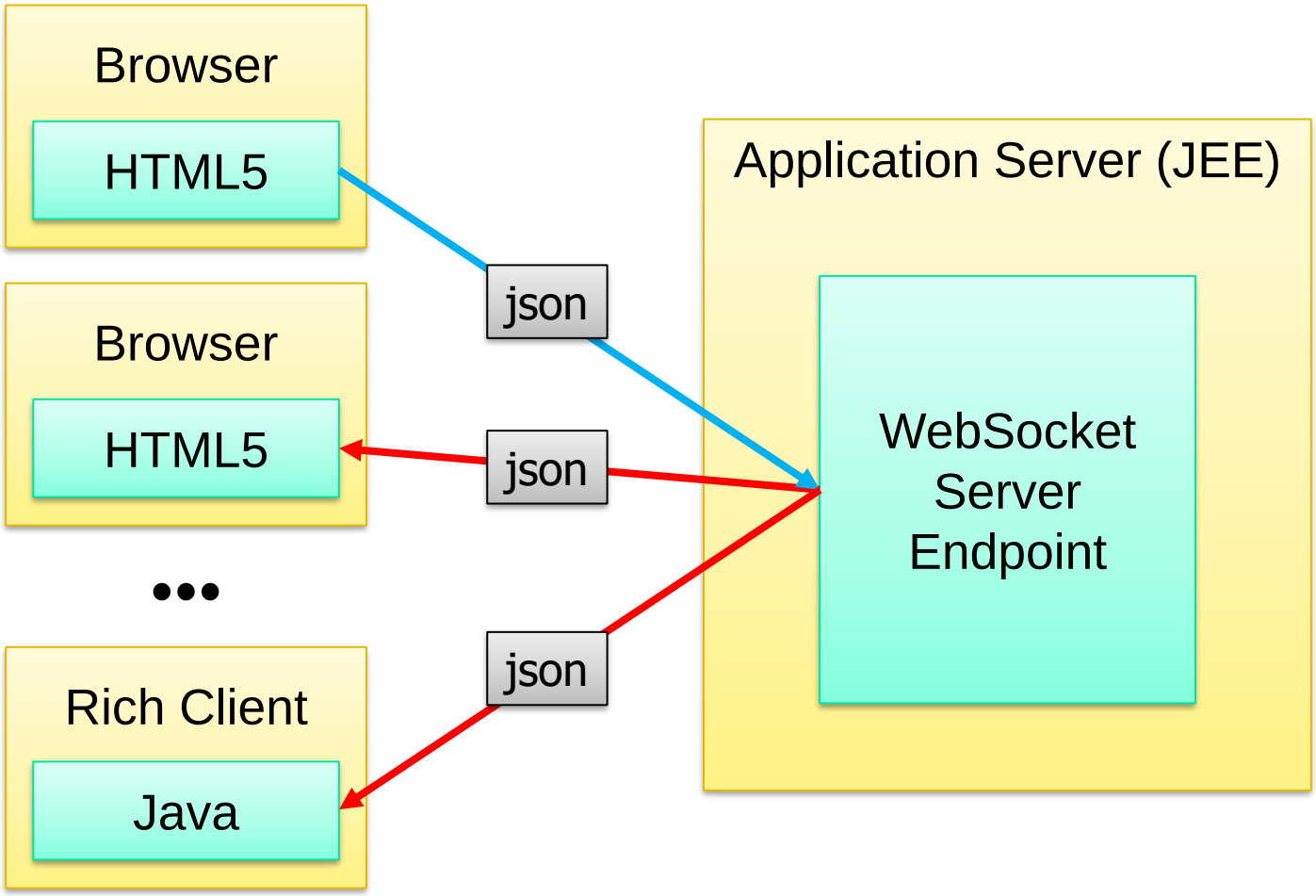


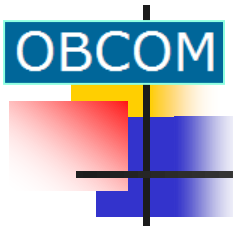
Mensaje binario WebSocket



- Non-Control Frames
  - **Text** (UTF-8 bytes)
  - **Binary** (raw bytes)
  - **Continue** (previous)
- Control Frames
  - **Close** (channel)
  - **Ping** (question)
  - **Pong** (answer)

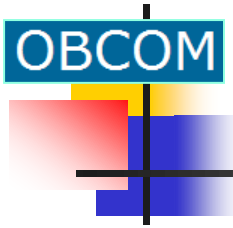
Ejemplo: chat room





WebSocket Server Endpoint

- Requiere una clase “Server Endpoint”
- Decorada con @**ServerEndpoint**(…)
- Con método @**OnOpen**
  - Invocado cuando se abre una sesión
- Con método @**OnClose**
  - Invocado cuando se cierra una sesión
- Con método @**OnMessage**
  - Invocado cuando se recibe un mensaje
- Con método @**OnError**
  - Invocado cuando ocurre un error asíncrono



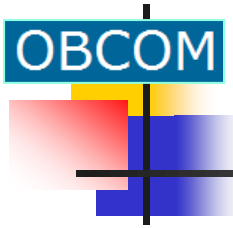
Clase Server Endpoint

```
import javax.websocket.*;
import javax.websocket.server.*;
@ServerEndpoint(
    value = "/chat",
    encoders = ChatMessageEncoder.class,
    decoders = ChatMessageDecoder.class)
public class ChatServerEndpoint
{
    @OnOpen {...}
    @OnClose {...}
    @OnMessage {...}
    @OnError {...}
}
```

Método @OnOpen

```
@ServerEndpoint(…)
public class ChatServerEndpoint
{
    ...
    @OnOpen
    public void onOpen(Session sesión, EndpointConfig cofig)
    {
        ...
    }
    ...
}
```

Método invocado cuando se abre una nueva sesión



Método @OnClose

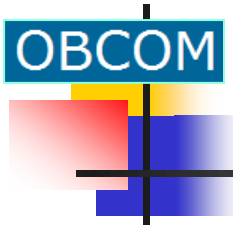
```
@ServerEndpoint(…)
public class ChatServerEndpoint
{
    ...
    @OnClose
    public void onClose(Session session, CloseReason reason)
    {
        ...
    }
    ...
}
```

Método invocado cuando se cierra una sesión

Método @OnError

```
@ServerEndpoint(…)
public class ChatServerEndpoint
{
    ...
    @OnError
    public void onError(Session session, Throwable thrown)
    {
        ...
    }
    ...
}
```

Método invocado cuando ocurre un error asíncrono



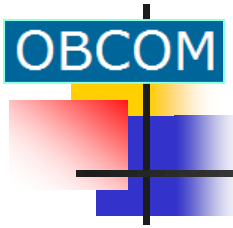
Método @OnMessage

```
@ServerEndpoint(…)
public class ChatServerEndpoint
{
    ...
    @OnMessage
    public void onMessage(ChatMessage cmessage, Session session)
    {
        ...
    }
    ...
}
```

Método invocado cuando se recibe un mensaje

Clase ChatMessage

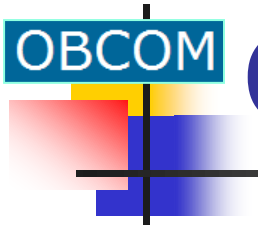
```
public class ChatMessage
{
    private String sender;
    private String message;

    public ChatMessage(String sender, String message)
    {
        this.sender = sender;
        this.message = message;
    }

    public String getSender()
    {
        return sender;
    }


    public String getMessage()
    {
        return message;
    }
}
```



Clase ChatMessageDecoder

```
import javax.websocket.*;
public class ChatMessageDecoder implements Decoder.Text<ChatMessage>
{
    ...
    @Override
    public ChatMessage decode(String json)
        throws DecodeException
    {
        ... parse json and obtain sender and message
        return new ChatMessage(sender, message);
    }
    ...
}
```

Convierte un String en un objeto ChatMessage



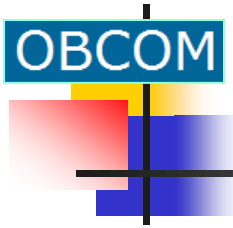
ChatMessageDecoder.decode

```
import javax.json.*;
import javax.websocket.*;
...
    @Override
    public ChatMessage decode(String json)
        throws DecodeException
    {
        StringReader textReader = new StringReader(json);
        JsonReader jsonReader = Json.createReader(textReader);
        JsonObject jsonObject = jsonReader.readObject();
        String sender = jsonObject.getString("sender", null);
        String message = jsonObject.getString("message", null);
        return new ChatMessage(sender, message);
    }
```

Clase ChatMessageEncoder

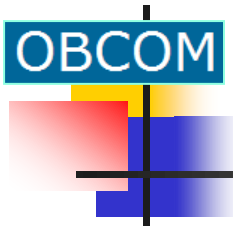
```
import javax.websocket.*;
public class ChatMessageEncoder implements Encoder.Text<ChatMessage>
{
    ...
    @Override
    public String encode(ChatMessage cmessage)
        throws EncodeException
    {
        ... build json string using cmessage fields
        return json;
    }
    ...
}
```

Convierte un objeto ChatMessage en un String



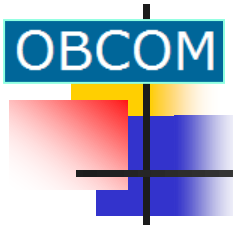
ChatMessageEncoder.encode

```
import javax.json.*;
import javax.websocket.*;
...
    @Override
    public String encode(ChatMessage cmessage)
        throws EncodeException
    {
        String sender = cmessage.getSender();
        String message = cmessage.getMessage();
        JsonObjectBuilder builder = Json.createObjectBuilder();
        if (sender != null) builder.add("sender", sender);
        if (message != null) builder.add("message", message);
        return builder.build().toString();
    }
```



WebSocket Client Endpoint

- Requiere una clase "Client Endpoint"
- Decorada con @**ClientEndpoint**(…)
- Con método @**OnOpen**
  - Invocado cuando se abre una sesión
- Con método @**OnClose**
  - Invocado cuando se cierra una sesión
- Con método @**OnMessage**
  - Invocado cuando se recibe un mensaje
- Con método @**OnError**
  - Invocado cuando ocurre un error asíncrono

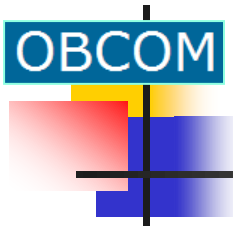


Clase Client Endpoint

```
import javax.websocket.*;
@ClientEndpoint(
    encoders = ChatMessageEncoder.class,
    decoders = ChatMessageDecoder.class)
public class ChatClientEndpoint
{
    @OnOpen {...}
    @OnClose {...}
    @OnMessage {...}
    @OnError {...}
    public static void main(String[] args) {...}
}
```

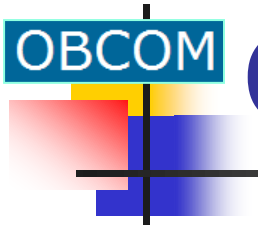
Método main del cliente

```
public static void main(String[] args)
{
    String host = System.getProperty("host", "localhost:8080");
    URI path = new URI("ws://" + host + "/SampleWebSocketWeb/chat");

    WebSocketContainer container = ContainerProvider.getWebSocketContainer();
    ChatClientEndpoint client = new ChatClientEndpoint();

    try (Session session = container.connectToServer(client, path)) {
        for (;;) {
            System.out.printf("> ");
            String line = System.in.readLine(); ☺
            if (line == null || line.equals("quit")) break;
            session.getBasicRemote().sendObject(new ChatMessage(user, line));
        }
    }
}
```



Bibliotecas para clientes Java

- API de **WebSocket** (javax.websocket.*)
  - **tyrus-standalone-client-jdk-1.16.jar**
- API de **JSON** (javax.json.*)
  - **jakarta.json-1.1.6.jar**
- Descargar desde **Maven Repository**
  - **https://mvnrepository.com/**
- Estas bibliotecas ya forman parte de los servidores **JEE**, y **no** se necesitan para construir ni ejecutar **Server Endpoints**



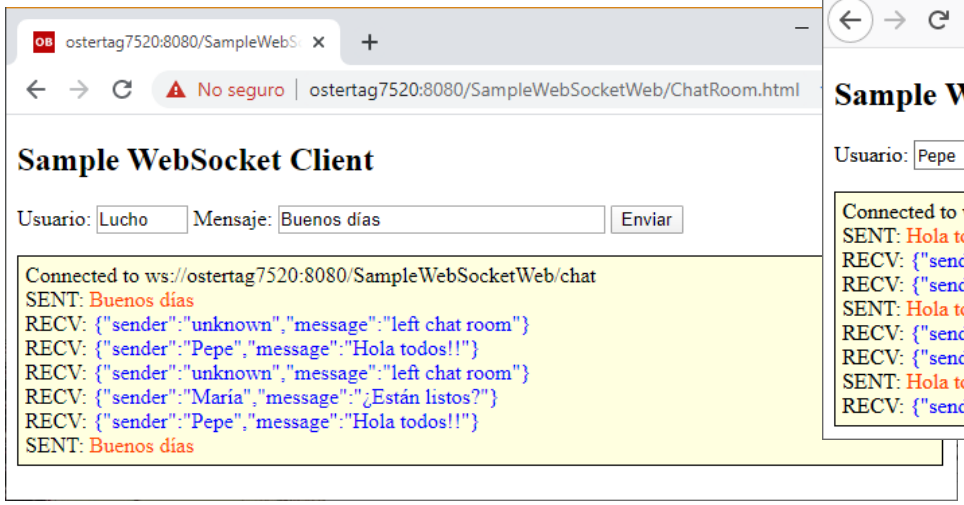
Cliente HTML WebSocket

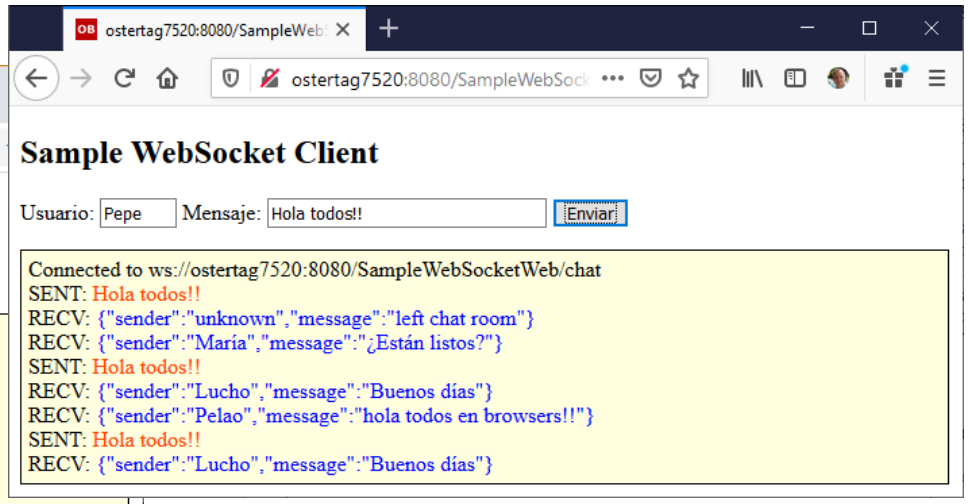
- Las API de WebSocket están incluidas en todos los navegadores (browsers) modernos
- Documento con detalles de compatibilidad:
  - https://developer.mozilla.org/en-US/docs/Web/API/Websockets_API
- En **JavaScript** se programa los mismos eventos que usamos para construir Server y Client **Endpoinds**:
  - **OnOpen**, **OnClose**, **OnError** y **OnMessage**
- Los mensajes son normalmente strings **JSON**

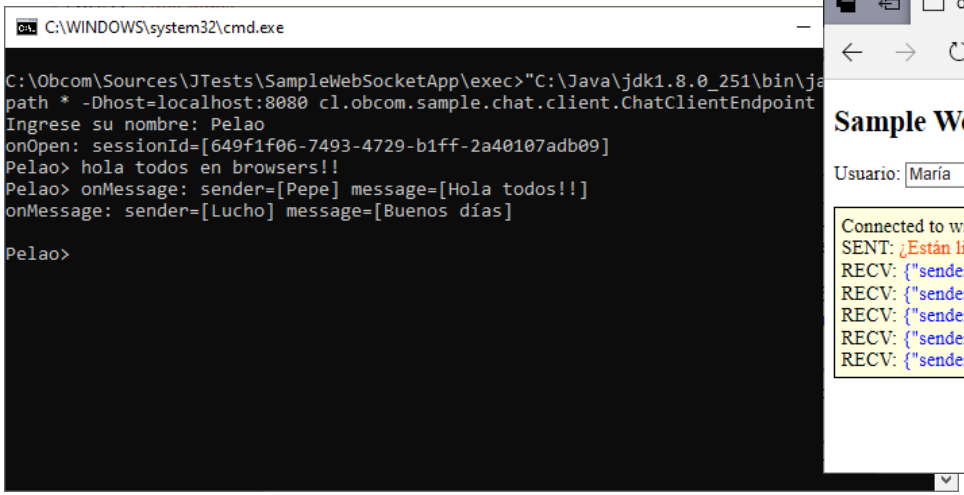
JavaScript y WebSocket

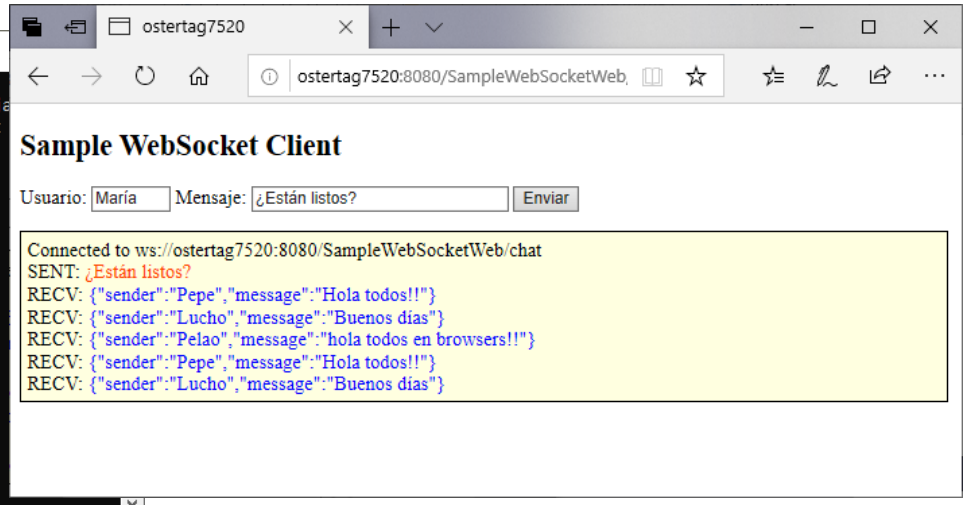
```
...
var websocket;
function initWebSocket() {
    var wsURL = "ws://" + window.location.host + "/chat/room";
    websocket = new WebSocket(wsURL);

    websocket.onopen = function(event) {
        displayOutput("Connected to " + wsURL);
    };
    websocket.onclose = function(event) {
        displayOutput("Disconnected from " + wsURL);
    };
    websocket.onmessage = function(event) {
        displayOutput("Received: " + event.data);
    };
    websocket.onerror = function(event) {
        displayOutput("Error: " + event.data);
    };
}
window.addEventListener("load", initWebSocket);
...
websocket.send(JSON.stringify({"sender":sender, "message":message}));
```

Todos hablando con todos ☺









Muchas gracias

Muchas gracias