

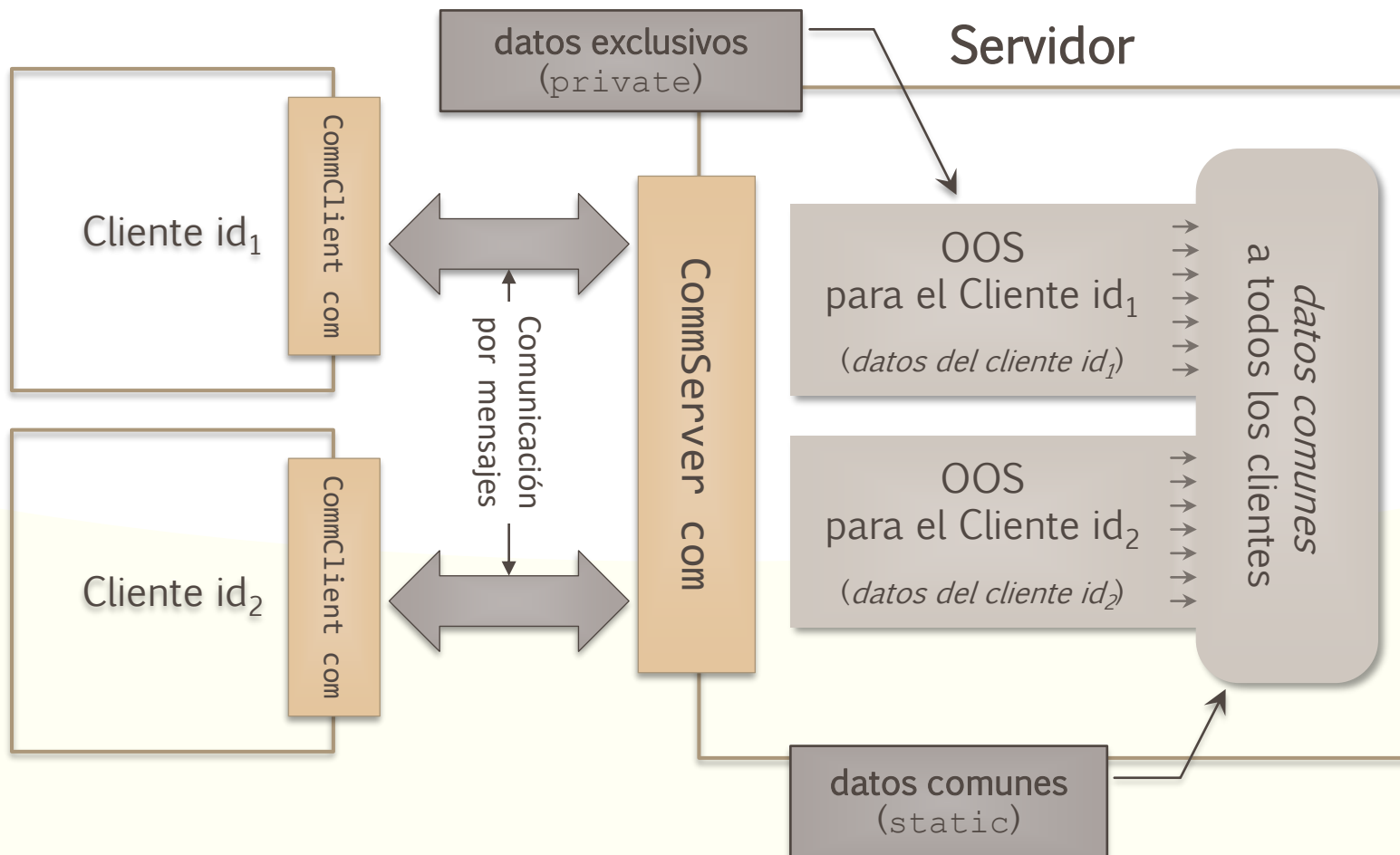
PRÁCTICAS DE POE

Programa Servidor (conexión de múltiples clientes)

Programa Cliente



Esquema del servidor a obtener



Saludador. Servidor de cliente único

```
// crear el canal de comunicación del servidor
com = new CommServer();
// registrar las operaciones del servidor
registrarOperaciones(com);
// poner el servicio a la escucha indefinidamente
while (true) {
    // 1. Esperar por un cliente
    idClient = com.waitForClient();
    // 2. Crear el OOS para el cliente idClient
    objServicio = new SaludadorOOS();
    // 3. Conversar con el cliente idClient hasta que éste
    // finalice. Ciclo petición-respuesta
    while (!com.closed(idClient)) { ... }
    // 4. Cerrar el OOS del cliente idClient
    if (objServicio != null) {
        objServicio.close();
    }
}
```

Operación bloqueante

Ejecución concurrente en un hilo para que varios clientes se puedan conectar simultáneamente.

Creación y ejecución de hilos (1)

- La clase Thread
 - Permite crear un hilo de ejecución en un programa (por ejemplo, en el `main` del servidor)
 - Los hilos son procesos ligeros que pueden compartir memoria
 - Creación de un hilo
 - Constructor `Thread(Runnable object)`
 - Comienzo de su ejecución
 - Método `start()` de la clase
 - Final de su ejecución
 - Porque termine de ejecutar su código (método `run`)
 - Si se lanza una excepción no capturada

Saludador. Servidor para varios clientes

```
try {  
    // crear el canal de comunicación del servidor  
    com = new CommServer();  
    // registrar las operaciones del servidor  
    registrarOperaciones(com);  
    // poner el servicio a la escucha indefinidamente  
    while (true) {  
        // 1. Esperar por un cliente  
        idClient = com.waitForClient();  
        // 2. Lanzar el hilo donde se creará el OOS y tendrá  
        //     lugar el intercambio de mensajes con el cliente  
        new Thread(new Hilos(idClient, com)).start();  
    }  
} catch (IOException | ChannelException e) {  
    // excepciones críticas, se para el servidor  
    System.err.printf("Error: %s\n", e.getMessage());  
    e.printStackTrace();  
}
```

Creación y ejecución de hilos (2)

- La interfaz Runnable
 - Interfaz funcional a implementar por cualquier clase cuyas instancias se pretendan ejecutar mediante un hilo.
 - Especifica el método `run()`. La acción (no tiene retorno) que definirá la secuencia de instrucciones que ha de ejecutar el hilo.

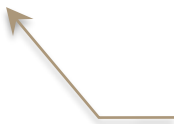
Saludador. Hilos

```
public void run() {  
    SaludadorOOS objServicio;  
    try {  
        // Crear el OOS para el hilo del cliente idClient  
        objServicio = new Saludador();  
        // Ciclo petición-respuesta para el cliente idClient  
        while (!com.closed(idClient)) { ... }  
    } catch (IOException | ChannelException e) {  
        System.err.printf("Error: %s\n", e.getMessage());  
    } finally {  
        // Cerrar el OOS  
        if (objServicio != null) {  
            objServicio.close();  
        }  
    }  
}
```

Cliente (1)

Operaciones sin respuesta (void y sin excepciones)

```
private static void sin_respuesta()  
    throws IOException, ChannelException {  
    // crear el mensaje de la petición  
    ProtocolMessages petition = new ProtocolMessages(str_id, args);  
    // enviar la petición  
    com.send();  
}
```



IOException
ChannelException

Cliente (2)

Operaciones con respuesta (dato o excepción)

```
private static Object con_respuesta() // también puede ser void
    throws IOException, ChannelException {
    // crear el mensaje de la petición
    ProtocolMessages petition = new ProtocolMessages(str_id, args);
    // enviar la petición
    com.send();
    try {
        // esperar por la respuesta
        ProtocolMessages respuesta = com.waitReply();
        // procesar la respuesta o la excepción
        Object obj = com.processReply(respuesta);
    } catch (ClassNotFoundException e) { ...
    } catch (UnknownOperation e) { ...
    } catch (IOException | ChannelException e) {
        throw e;
    } catch (Exception e) { ...
    }
    return obj;
}
```

IOException
ChannelException

ClassNotFoundException
IOException
UnknownOperation
ChannelException

IOException
Exception

Cliente (3)

- Captura de una excepción
 - Normalmente un mensaje indicando el error:
`System.err.println(e.getMessage());`
 - Para *Exception*:
`System.err.printf("%s: %s\n",
e.getClass().getSimpleName(), e.getMessage());`
y así obtener el tipo de excepción que se ha producido
- Además de las excepciones de la biblioteca, también deben capturarse las excepciones que se puedan producir en el servicio

Cliente (4)

- Clase *Menu*

- Permite crear y gestionar un menú de opciones seleccionables
- *Menu(str_título, str_prompt, iterativo?)*
- *add(texto_opción, función_anónima)*
- *add(texto_opción, objeto)*
- *run()*
- *runSelection()*
- *getObject()*
- *input()*