PSP - U4 Actividades auto evaluación

Descargar estos apuntes

Índice

> **Warning**
>
> Para todas las aplicaciones, **las excepciones deben controlarse en el código del programa**. Si no se indica lo contrario, en el bloque catch siempre se mostrará la pila de llamadas usando el método `printStackTrace()` de la excepción capturada junto con la versión traducida del mensaje de error asociado, método `getLocalizedMessage()`.

> **Nomenclatura de las clases**
>
> Para cada actividad, crea un proyecto nuevo con el nombre de la clase principal
>
> - Crea una/varias nuevas clases dentro del paquete psp.u4. para cada ejercicio.
> - Las clases deben tener el prefijo psp.actividades.U4AXX_xxxxxxxx y darles un nombre identificativo de la funcionalidad que tienen en el diseño (por ejemplo psp.actividades.U4A20_Subasta).

1. Proyecto U4A01_Simplex

En esta actividad vamos a implementar una aplicación cliente / servidor en la que se realiza un comunicación de tipo `simplex`, es decir la comunicación se realiza sólo en una dirección, en nuestro caso del cliente al servidor.

Clase psp.actividades.U4A01_SimplexClient
Nuestro `cliente` va a enviar líneas de texto al servidor. **La dirección del server y el puerto se reciben como argumentos del programa**.
El texto a enviar lo vamos a leer de la entrada estándar (System.in) a través del teclado.

Para indicarle al servidor que queremos cerrar la comunicación, es decir, que ya no le vamos a enviar más mensajes, enviaremos el texto “END OF TRANSMISSION”.

Clase psp.actividades.U4A01_SimplexServer

Nuestro `servidor` va a esperar la información que le llega desde el cliente. El puerto donde escucha el servidor se recibe como argumento del programa.
Cada vez que reciba un texto, lo mostrará por consola precedido por **CLIENT>**.

Cuando el servidor reciba el texto “END OF TRANSMISSION” ya no esperará más mensajes del cliente y cerrará la comunicación y todos los recursos utilizados.

2. Proyecto U4A02_SimplexMultithread

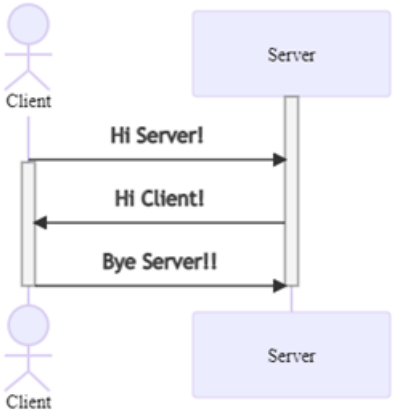
Partiendo de la actividad anterior, realiza las modificaciones necesarias para que el servidor pueda atender a varios clientes a la vez.

Para diferenciar a un cliente de otro, en vez de preceder la salida con “CLIENT>”, se usará el texto del primer mensaje que reciba de cada cliente para preceder sus mensajes. Por lo tanto ese primer mensaje determinará el identificador del usuario en el servidor.

A diferencia del caso anterior, el servidor estará ejecutándose indefinidamente a la espera de que se conecten clientes. Ahora bien, cuando cada worker del servidor (hilo) reciba el texto “END OF TRANSMISSION” sí cerrará la comunicación con el cliente que esté atendiendo y finalizará el hilo, pero no el proceso servidor.

3. Actividad U4A03_ProtocoloSaludo

En esta actividad vamos a implementar una aplicación cliente / servidor en la que se implemente el protocolo que se muestra a continuación.



La actividad U4A03_ProtocoloSaludo se puede afrontar de diversas maneras:

- Si pensamos que todo en la comunicación va a ir correctamente, es un ejercicio muy sencillo. Podéis empezar por ahí. **Como actividad de autoevaluación, realizad el ejercicio de esta forma**.
- Ahora bien, después debéis plantear la actividad justo al contrario, es decir, que cualquier mensaje puede llegar en cualquier momento. **Este planteamiento lo veremos cuando hablemos de protocolos con estado**
  El proceso que debe garantizar la "seguridad" y "corrección" del protocolo SIEMPRE es el proceso servidor, ya que podemos usar clientes genéricos como Telnet, netcat o un cliente que no controle el protocolo, aunque los que nosotros desarrollemos estén preparados para cumplir también el protocolo.

Para esto, debéis programar el servidor usando estados y el cliente... lo dejo a vuestra elección.

Lanzamiento de las clases

Fijaos que en el diagrama el servidor debe estar activo antes de que los clientes empiecen a hacer solicitudes.

4. Actividad U4_EtiquetandoComunicación

Modifica el servidor y el cliente básicos de forma que se intercambien un único mensaje en modo dúplex y se produzca la siguiente salida

```
SERVIDOR
(Servidor) Esperando conexiones...
(Servidor) Conexión establecida.
(Servidor) Abriendo canales de texto...
(Servidor) Canales de texto abiertos.
(Servidor) Leyendo mensaje...
(Servidor) Mensaje leído.
(Servidor) Mensaje recibido: Mensaje enviado desde el cliente
(Servidor) Enviando mensaje...
(Servidor) Mensaje enviado.
(Servidor) Cerrando canales de texto.
(Servidor) Canales de texto cerrados.
(Servidor) Cerrando conexiones...
(Servidor) Conexiones cerradas.
```

```
CLIENTE
(Cliente) Estableciendo conexión...
(Cliente) Conexión establecida.
(Cliente) Abriendo canales de texto...
(Cliente) Canales de texto abiertos.
(Cliente) Enviando mensaje...
(Cliente) Mensaje enviado.
(Cliente) Mensaje leído.
(Cliente) Mensaje recibido: Mensaje enviado desde el servidor
(Cliente) Cerrando canales de texto.
(Cliente) Canales de texto cerrados.
(Cliente) Cerrando conexiones...
(Cliente) Conexiones cerradas.
```

Objetivo de la actividad

Lo IMPORTANTE en esta actividad es que los mensajes de salida se ubiquen en el sitio correcto para comprender el flujo de la comunicación entre cliente y servidor.

5. Actividad U4_SumadorOnline

Crea un servidor que recoja los números que se le envían desde un cliente y devuelva la suma de los números recibidos.

El servidor recibirá los números hasta que reciba un 0. En ese momento devolverá la suma de los números recibidos y cerrará la conexión.

Prueba el servidor con la aplicación Telnet.

> **Instalar Telnet en Windows**
>
> Para instalar Telnet en Windows, sigue las instrucciones de este enlace.
>
> Puedes ver la referencia del comando telnet en la documentación de Microsoft.

6. Actividad U4_SignoZodiaco

Crea un servidor que devuelva el `signo zodiaco europeo y chino` de una persona en función de la fecha de nacimiento que se le envía desde un cliente.

El servidor recibirá la fecha de nacimiento en formato dd/mm/yyyy y devolverá el signo del zodiaco al que pertenece esa persona.

> **Gestión de las fechas**
>
> En la actividad, debéis gestionar las fechas adecuadamente. Pensad que toda la información se envía / recibe en formato texto, por lo que habrá que hacer la conversión de texto a fecha y viceversa.
>
> Java tiene múltiples clases para trabajar con fechas, pero en este caso, podéis usar la clase `SimpleDateFormat` para convertir texto a fecha y viceversa.
>
> En el site de Baeldung tenéis una serie de artículos para trabajar con las fechas en Java.

7. Actividad U4_ListaCompra

Vamos a realizar en Java la creación de una lista de la compra mediante comunicación con sockets TCP. El esquema a seguir será el siguiente:

1. Arranca el servidor
2. Arranca el cliente que solicitará al usuario un producto para añadir a la lista de la compra.
3. El cliente escribirá el producto leído en el flujo de salida.
4. El servidor lee el producto del flujo de entrada:
   4.1. Si el producto es “salir”, el cliente finaliza y el servidor finaliza.
   4.2. Si el producto no es “salir”, dicho producto se añadirá a un fichero de texto.
5. El servidor informa “Producto xxx añadido a la lista de la compra” y sigue esperando nuevos productos.
6. El cliente finaliza su ejecución.

El nombre del archivo de texto será el del día de ejecución del programa en formato `dd-mm-yyyy.txt`. El cliente informa “Se ha volcado el producto xxx a la lista de la compra” y finaliza.

> **Mejora posible**

En el caso anterior, para ir añadiendo nuevos productos, el cliente deberá ejecutarse de nuevo.

Una posible mejora sería que el cliente pudiera enviar varios productos antes de finalizar. En este caso, el servidor debería recibir los productos e ir volcándolos al fichero de texto.

Si el cliente envía el comando "salir", forzaría la finalización del servidor.

Si queremos que el servidor siga, deberíamos usar otro comando para finalizar la comunicación con ese cliente y permitir que el servidor pueda seguir atendiendo a otros clientes.