Ejercicios 1. Sockets

Ejercicio 1.1) [Proyecto] Crea un servidor HTTP sencillo que siempre devuelva la misma página HTML. Este servidor debe ser compatible con los navegadores Web habituales por lo que la respuesta debe seguir el formato HTTP. Para llevar a cabo esta tarea es recomendable crear las siguientes clases

- `HTTPRequest`: Clase que contiene la información de una petición HTTP, cuyo constructor recibirá un Reader y parseará para recuperar la información de una petición HTTP. La información que contendrá será la siguiente:
  - **Método**. Puede ser GET, POST, PUT, DELETE, OPTIONS, TRACE, CONNECT, HEAD, aunque en este proyecto solo utilizaremos los cuatro primeros. Más información aquí.
  - **Nombre del recurso solicitado (sin los parámetros)**. Por ejemplo, en una petición para `/index.php?param1=value1`, el nombre del recurso será `index.php`.
  - **Parámetros de la consulta**. En el caso de GET formarán parte del rescurso solicitado y en el caso de POST formarán parte del recurso solicitado o del cuerpo de la petición. La mejor forma de almacenar estos valores es utilizar un `Map<String, String>`.
  - **Parámetros de la cabecera**. Se encuentran después de la primera línea y siguen el formato `Cabecera: Valor`. La cabecera finaliza cuando hay una línea en blanco. La mejor forma de almacenar estos valores es utilizar un `Map<String, String>`.
  - **Longitud del contenido**: En el caso de que haya contenido, la cabecera incluirá el parámetro `Content-Length`, que indicará la longitud del cuerpo.
- `HTTPResponse`: Clase que contiene la información de una respuesta HTTP. Deberá disponer de un método `print(Writer)` que escriba la respuesta en formato HTTP correcto. La información que contendrá será la siguiente:
  - **Estado**. Código de estado del servidor. Más información (aquí) [http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html].
  - **Versión de HTTP**. Normalmente, HTTP/1.1.
  - **Contenido**. Será una cadena de texto con el cuerpo de la respuesta.
  - **Parámetros de cabecera**. La mejor forma de almacenar estos valores es utilizar un Map<String, String>.

Nota: Para poder iniciar y parar el servidor necesitarás utilizar un hilo y desbloquear al servidor que está a la espera de conexiones. Para ello, puedes utilizar la base de código de la clase `HybridServer`. No te preocupes si hay alguna parte de este código que no entiendes bien, pues la próxima semana veremos el tema de "Multihilo" en el que hablaremos sobre este tipo de estructuras.

Ejercicio 1.2) Crea un juego de tres en raya entre dos aplicaciones de modo que la información entre las aplicaciones se transmita en un objeto `TicTacToe` (tienes una implementación de ejemplo en Faitic, junto a este enunciado). Este objeto representa un tablero sencillo de tres en raya sin comprobaciones de reglas (es posible hacer trampa). Para simplificar el problema puedes dar por supuesto que los usuarios siempre van a realizar un movimiento válido y lo van a introducir de forma correcta.

El proceso del juego será algo de este estilo:

1. Jugador 1 introduce jugada.
2. La consola del Jugador 1 muestra el nuevo estado del tablero.
3. Jugador 1 envía el objeto tablero al Jugador 2.
4. Si Jugador 1 ha ganado se avisa por consola y finaliza el juego en Jugador 1.
5. La consola del Jugador 2 muestra el tablero recibido.
6. Si Jugador 1 ha ganado se avisa por consola y finaliza el juego en Jugador 2.
7. Jugador 2 introduce jugada.
8. La consola del Jugador 2 muestra el nuevo estado del tablero.
9. Jugador 2 envía el objeto tablero al Jugador 1.
10. Si Jugador 2 ha ganado se avisa por consola y finaliza el juego en Jugador 2.
11. La consola del Jugador 1 muestra el tablero recibido.
12. Si Jugador 2 ha ganado se avisa por consola y finaliza el juego en Jugador 1.
13. Vuelta a 1.