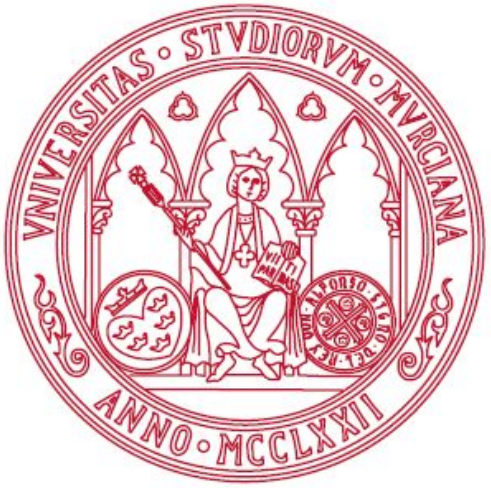


Universidad de Murcia

Grado en Ingeniería Informática

Curso 2020/2021

Programación para las comunicaciones

Profesor: Humberto Martínez Barberá

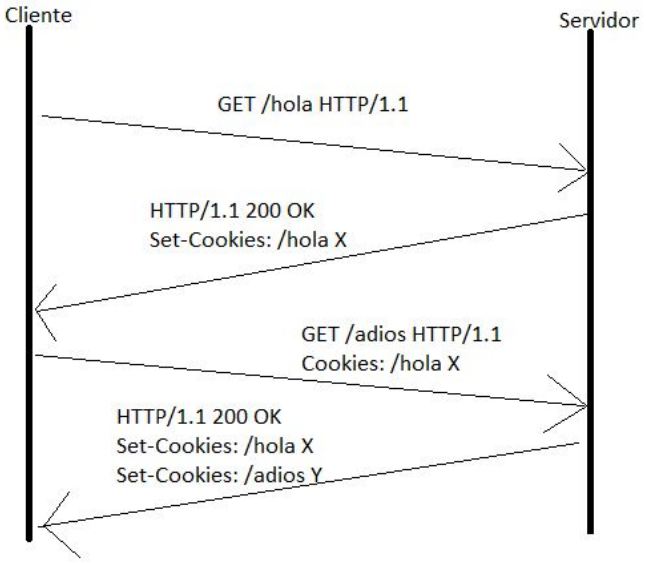
Práctica I

Jaime Ortiz Aragón - 49196689B - jaime.ortiza@um.es

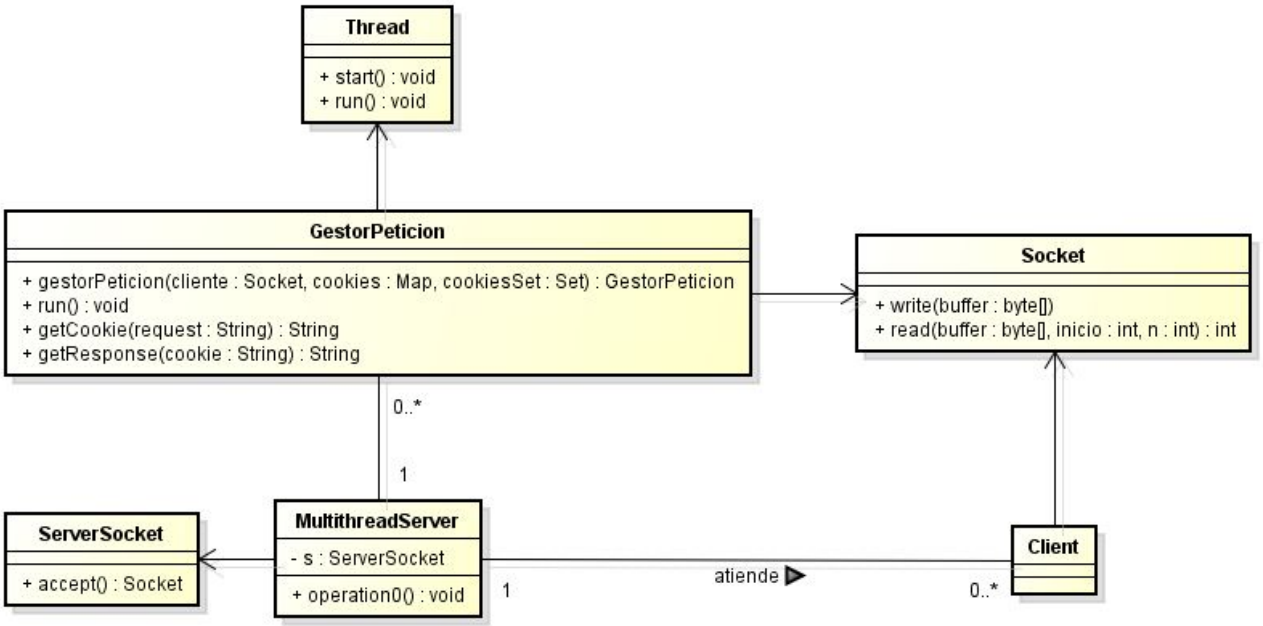
- **ESCENARIO**

Para realizar esta primera práctica, he partido del código que se nos proporcionaba en el documento *PPC-Practica1-java.pdf* en el apartado de las "Comunicaciones en Java", concretamente el referente a la estructura del servicio ECO multihilo.

El objetivo era simular un carrito de la compra con los recursos pedidos al cliente. Este es un ejemplo de cómo funciona la aplicación, que detallaremos más adelante, con las funciones específicas de cada agente.



En nuestro caso, hay dos clientes. Por un lado podemos hacer una prueba local de Eclipse lanzando el servidor y el cliente, que se van a comunicar en este caso por el puerto 9999 mediante sockets. Para comunicarse de manera ordenada, siguen el esquema de productor/consumidor, en el que el cliente manda una petición al servidor y se queda bloqueado en la lectura hasta que este, que estaba bloqueado en su lectura, lee lo que ha escrito el cliente y manda su respuesta. Esta comunicación se realiza siguiendo el esquema HTTP.



- **SERVIDOR**

El servidor cuenta con un ServerSocket, que actúa como servidor y un socket, que hace referencia al cliente, el cual cobra protagonismo cuando el servidor acepta la comunicación. En el constructor se indica en el puerto que va a escuchar el servidor y durante un tiempo *timeout* o hasta que el usuario decida externamente para la aplicación, se generarán peticiones del cliente. Esta comunicación se inicia con el método *accept*. Para tratar las cookies de cada cliente, se almacena un mapa que relaciona el nombre con el número de veces que este recurso se ha solicitado. De este modo, cuando nos llega la petición de un recurso, solo tenemos que comprobar si este está en nuestro mapa para formar la respuesta. La lectura de esta petición será la que deshaga el bloqueo en el que se encuentra el proceso, al haber llegado a una operación *read* sobre el socket. Como respuesta, en cuanto a las cabeceras de cookies, devolvemos una cabecera *Set-Cookie* por cada cookie que hemos almacenado en el mapa para dicho cliente, que se escribirá por el socket haciendo uso de *write.*

- **CLIENTE**

La aplicación soporta tanto peticiones/respuestas de un cliente que se lanza en Eclipse, como las de un cliente como puede ser el navegador. En mi caso he utilizado *Google Chrome* para todas las pruebas. La petición se forma con el método *GET*, nombre del recurso y la versión (*HTTP/1.1* en nuestro caso). En cuanto a las cabeceras *Cookie*, se generará una por cada uno de los recursos solicitados al cliente. En la prueba local se trata de simular el proceso que lleva a cabo el navegador.

- **MEJORA**

Como mejora, he incluido que cada petición incluya el número de veces que se ha pedido cada cookie. Esto lo he implementado con un mapa en el que la clave es el nombre de la cookie y el valor es el número de veces que se ha pedido. Esta estructura también la he utilizado para hacer sensación de persistencia y que un cliente pueda realizar varias peticiones seguidas añadiendo los recursos a sus cabeceras *Cookie.*

Este es el funcionamiento de la aplicación para cualquier petición de un nuevo cliente. Una vez establecida la conexión, las peticiones se realizan en bucle entre las operaciones 3-9, hasta que se cierra el socket.