**Práctica 3:**

**SOCKETS**

***Curso 2023- 2024***

Introducción a Redes de Computadoras

Docente: Roberto Olea

DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

**OBJETIVOS DE LA PRÁCTICA**

Con esta práctica se pretende que el alumno sea capaz de comprender el funcionamiento de una aplicación de comunicaciones utilizando **sockets en java**. Para ello analizará, ejecutará y adaptará una aplicación ya implementada.

La primera parte de esta práctica es una breve introducción al entorno de desarrollo en eclipse que se utiliza para la ejecución de la práctica. En la segunda parte se definirán los fundamentos de la programación con sockets en Java. Para finalmente trabajar con la aplicación de mensajería entre dos ordenadores conectados en red.

**1.- CREAR UNA APLICACIÓN JAVA CON ECLIPSE**

Eclipse es una plataforma de software de Código abierto. Esta plataforma, es usada para la creación de entornos integrados de desarrollo (IDE), como el IDE de Java (Java Development Toolkit JDT). Eclipse fue desarrollado originalmente por IBM como el sucesor de VisualAge. Ahora está en manos de la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto.

Se puede descargar Eclipse, desde la web <http://www.eclipse.org/downloads>.

El ejecutable es el fichero **eclipse.exe**.

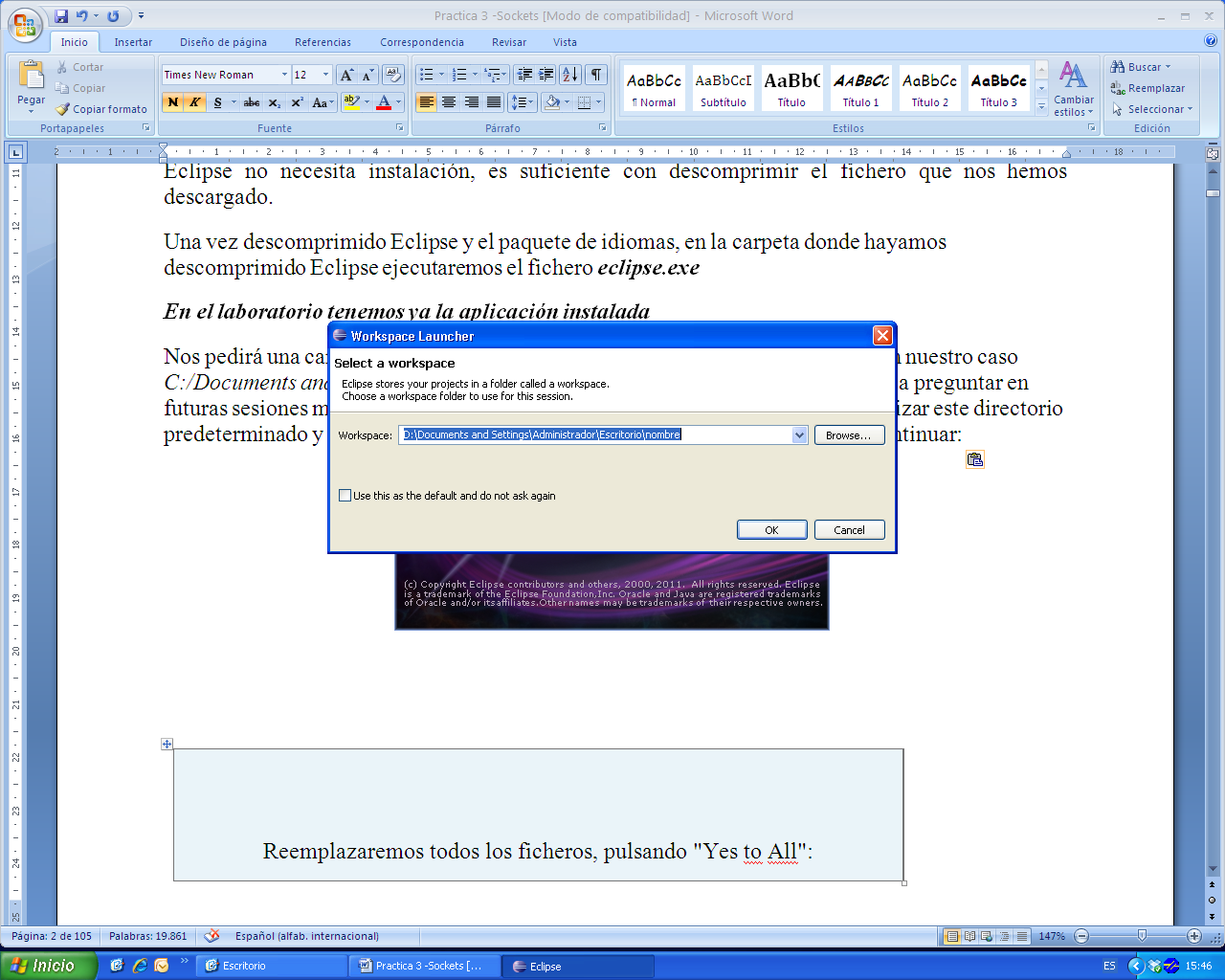


*En el laboratorio la aplicación está instalada.*

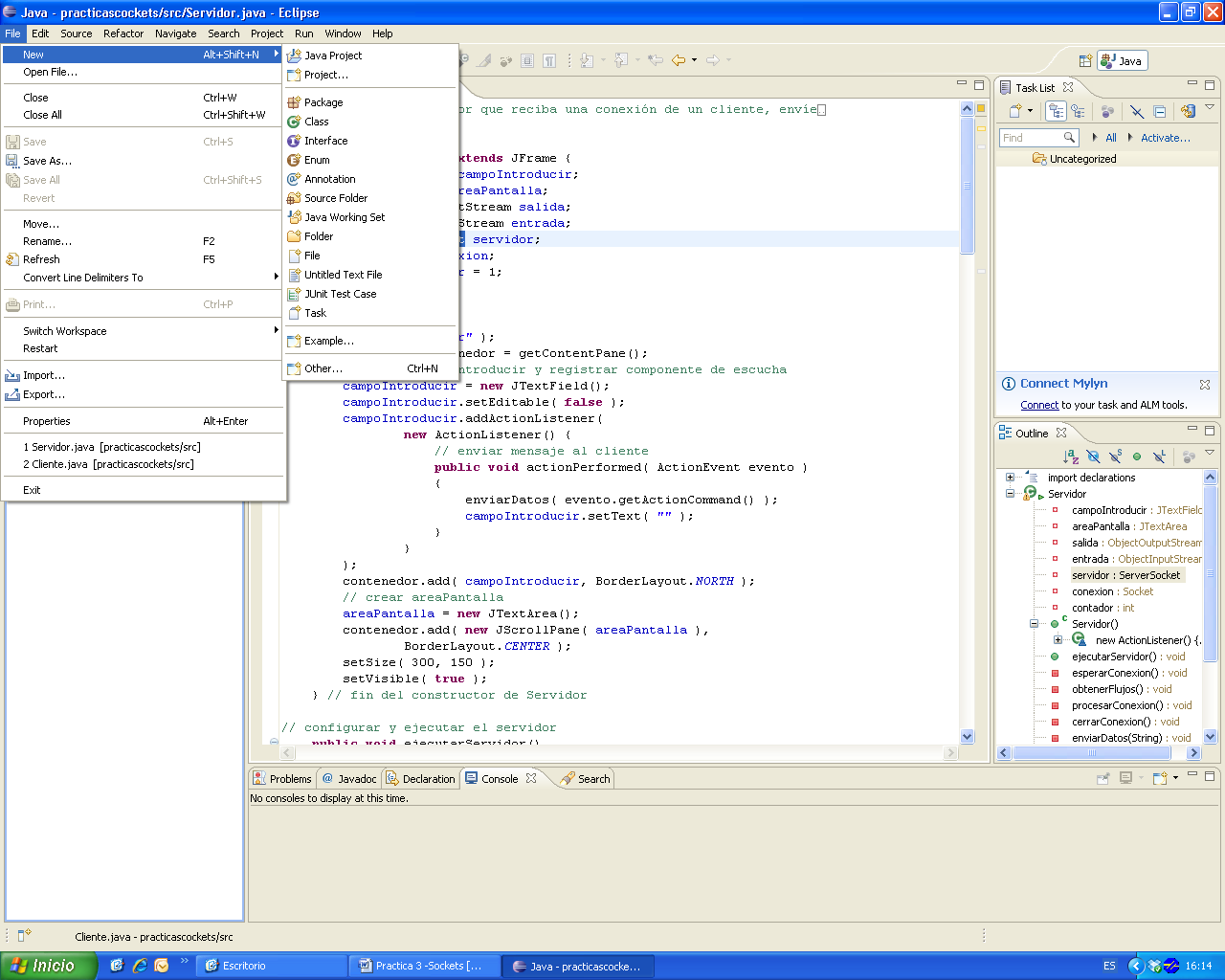
Al ejecutar la aplicación se pide un directorio para el espacio de trabajo (workspace) para esta sesión. *Podéis crear un directorio en el escritorio con vuestras iniciales* *(Recordar que si queréis guardar vuestro trabajo el escritorio no es el lugar).*

*Ej: D:\Documents and Settings\alumno\Escritorio\nombre.*

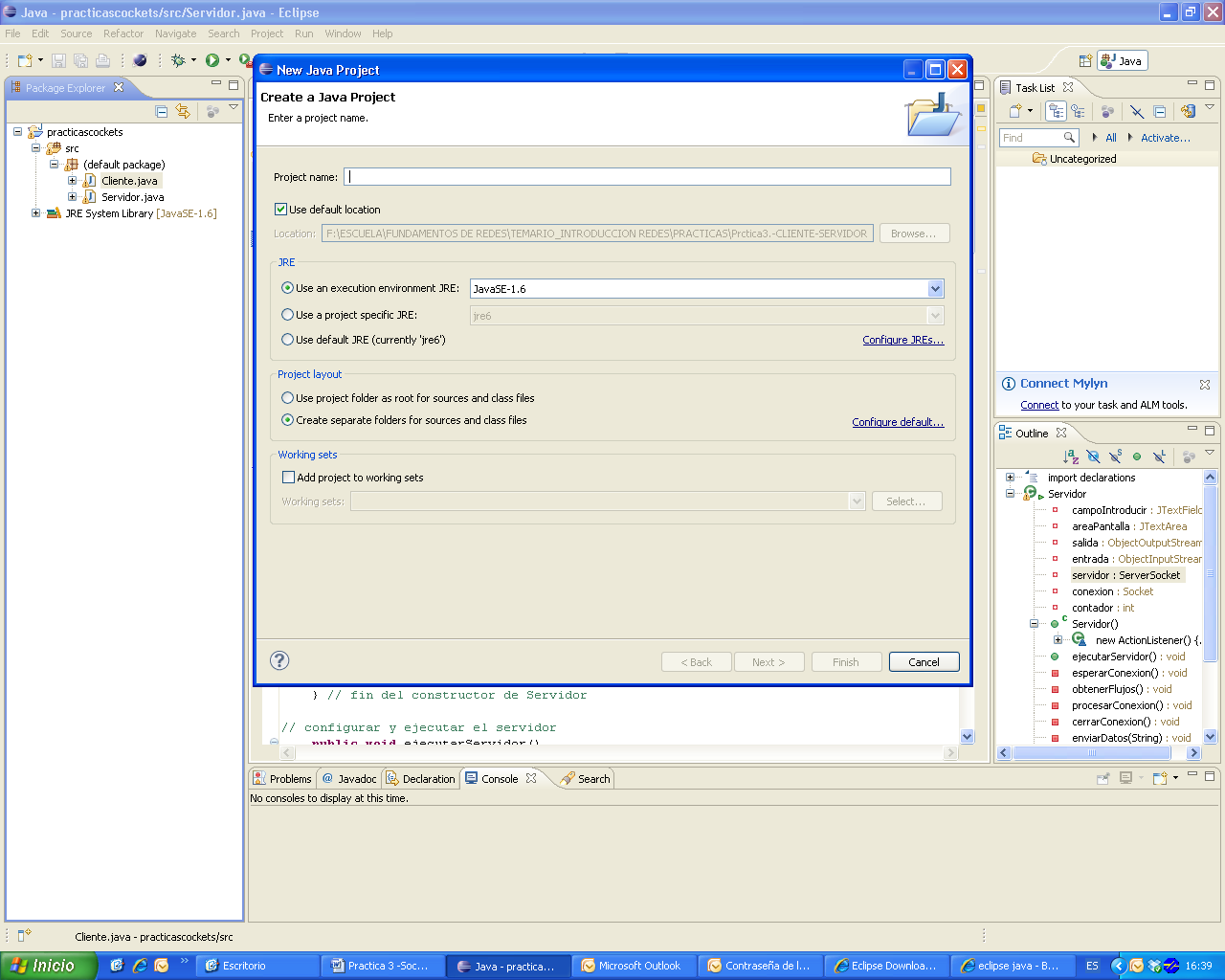
Se pulsa "OK" ("Aceptar") para continuar:

****

Todo programa en Eclipse requiere la creación de un "Proyecto", para esto se debe seleccionar desde el menú de opciones **"File" - "New" - "Java Proyect"**

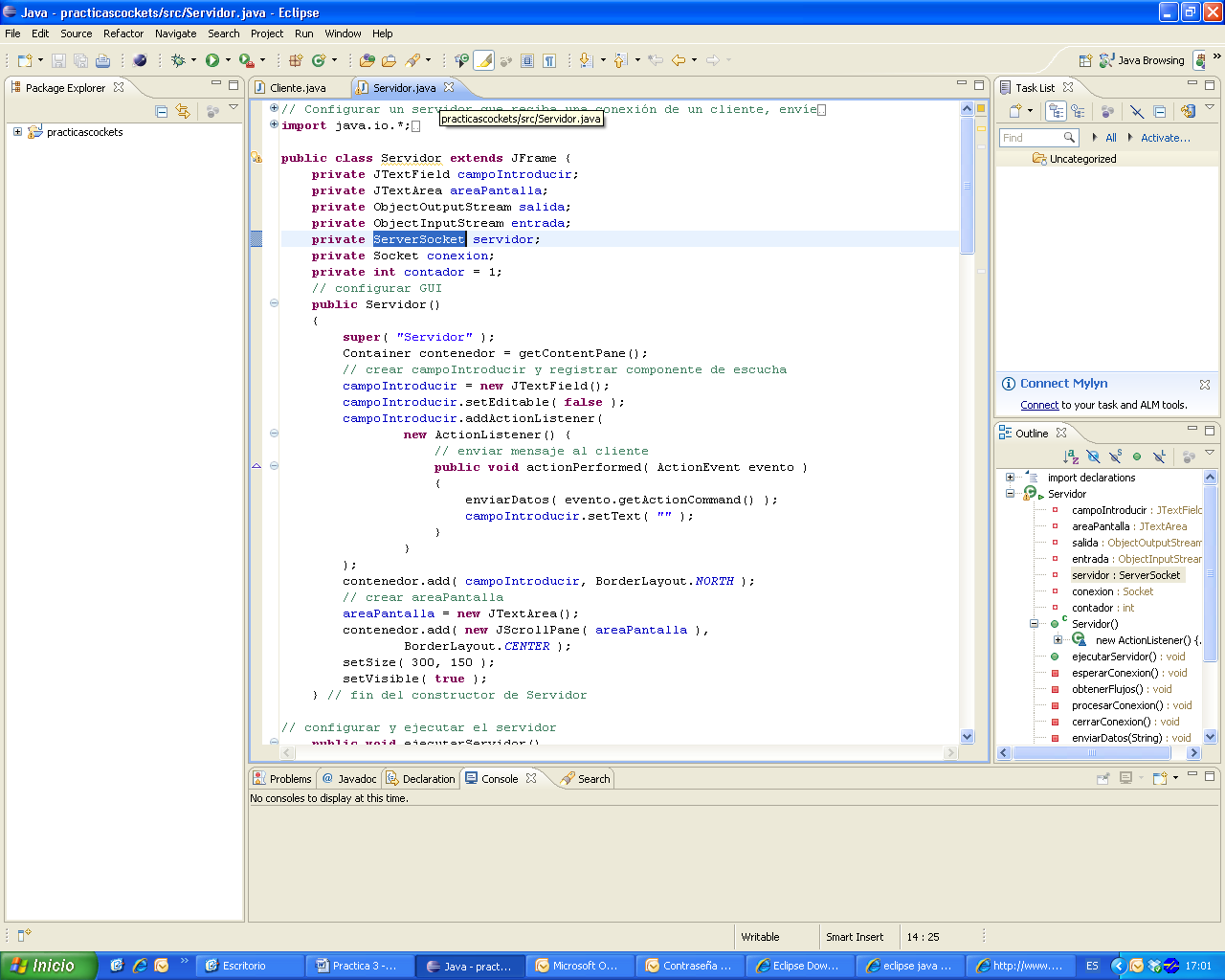


Aparece un menú de diálogo donde definir el nombre del proyecto:



En el campo de texto "**Project Name"** se le da un nombre y se dejan todas las otras opciones del diálogo con los valores por defecto. Presionamos el botón **"Finish"**.

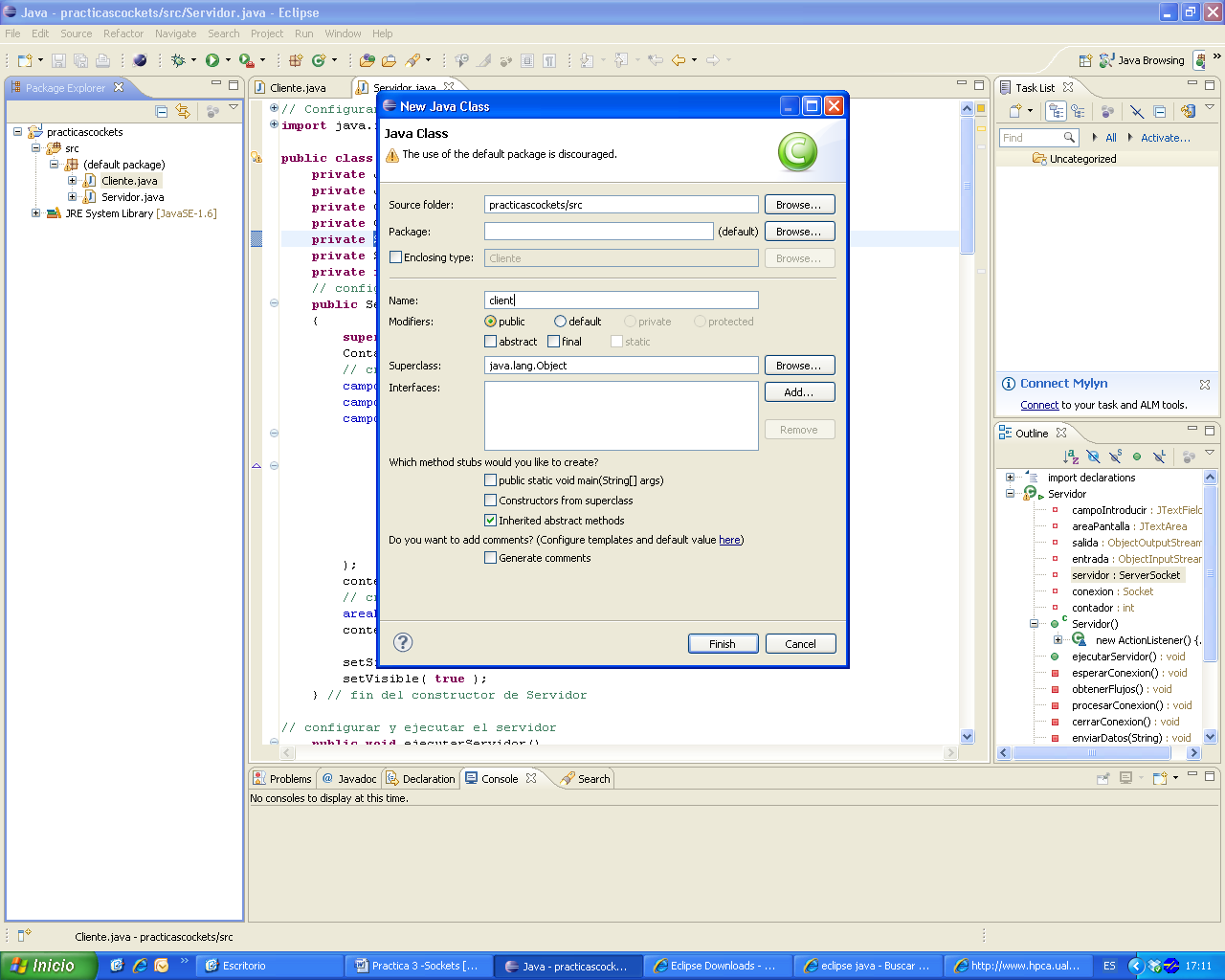
Ahora en la ventana de **"Package"** aparece el proyecto que se acaba de crear:



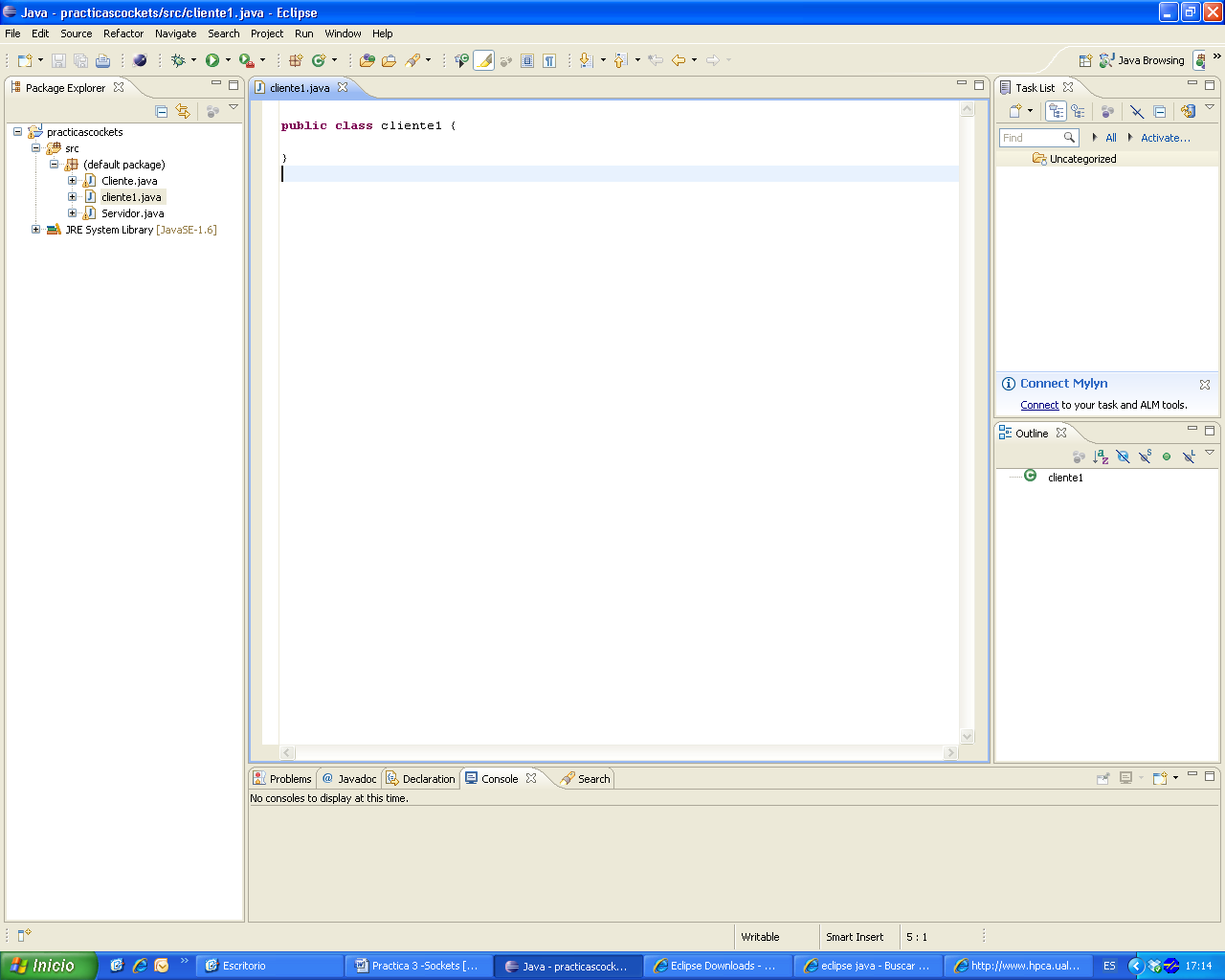
Todo programa en Java requiere como mínimo una clase. Para crear una clase se puede hacer seleccionando class desde el menú de opciones o desde la barra de iconos de eclipse.

|  |  |
| --- | --- |
|  |  |

En el diálogo que aparece se debe definir el nombre de la clase (define una clase Cliente y otra Servidor) los otros datos del diálogo se dejan con los valores por defecto.



Se presiona el botón **"Finish"** y se obtiene el archivo donde codificar la clase.



Para poder probar el entorno mete el siguiente código dentro de la clase

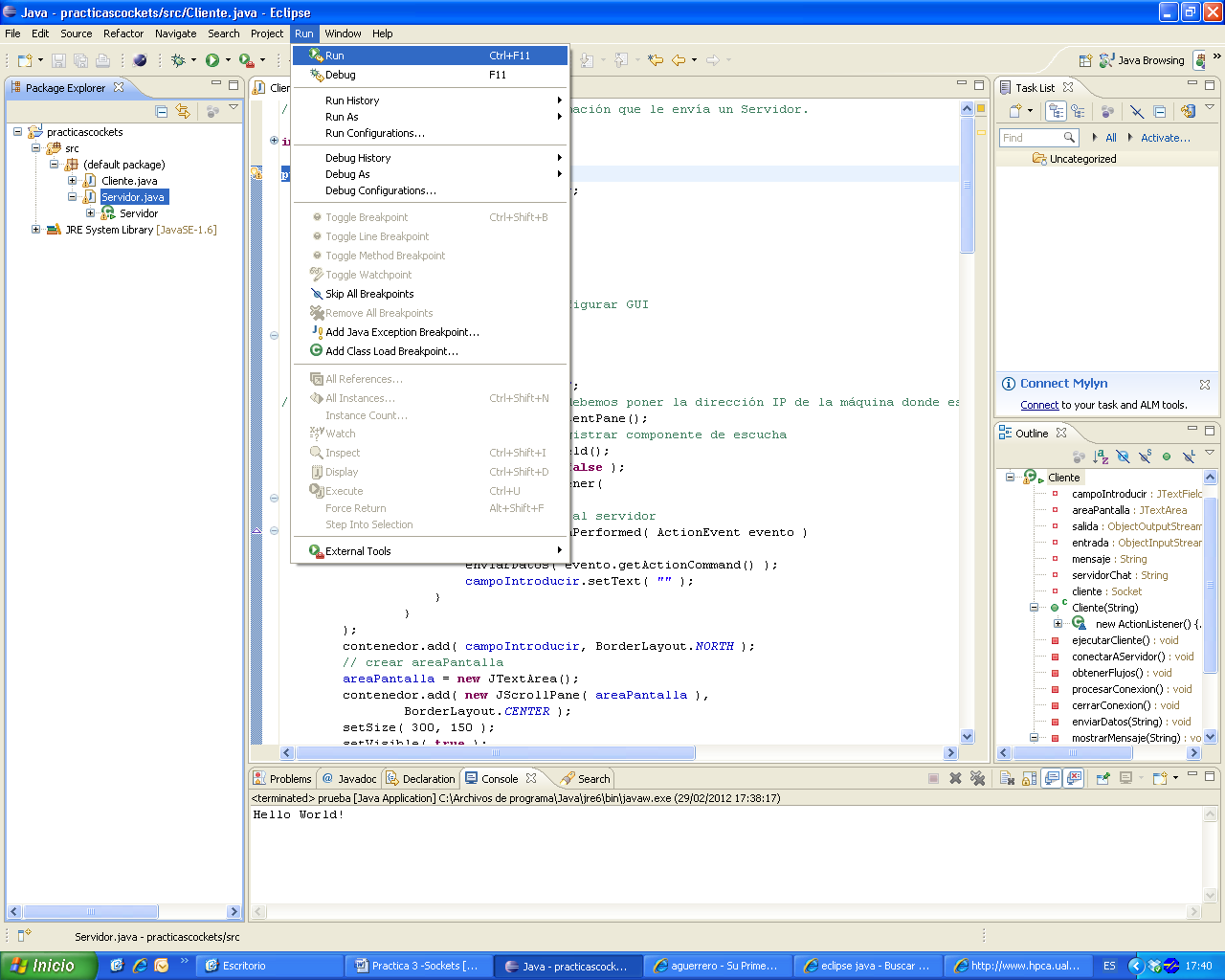
public static void main(String[] args) {

// Display "Hello World!"

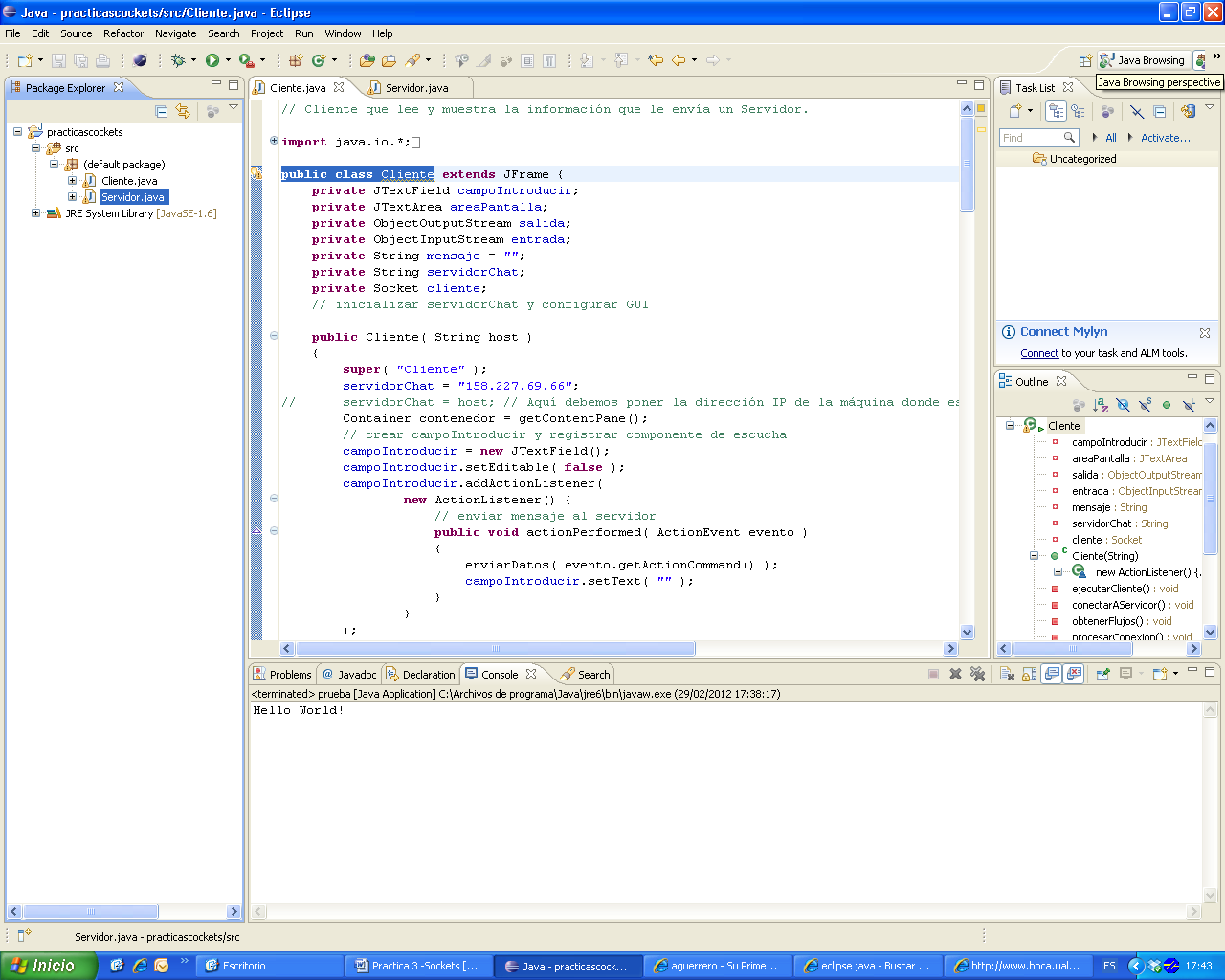
System.out.println("Hello World!");

}

Seleccionar la clase y ejecutar desde el menú de opciones o con el icono correspondiente.



Si no hay errores de codificación se verá el resultado de la ejecución en una ventana del Eclipse llamada "Console" que aparece en la parte inferior

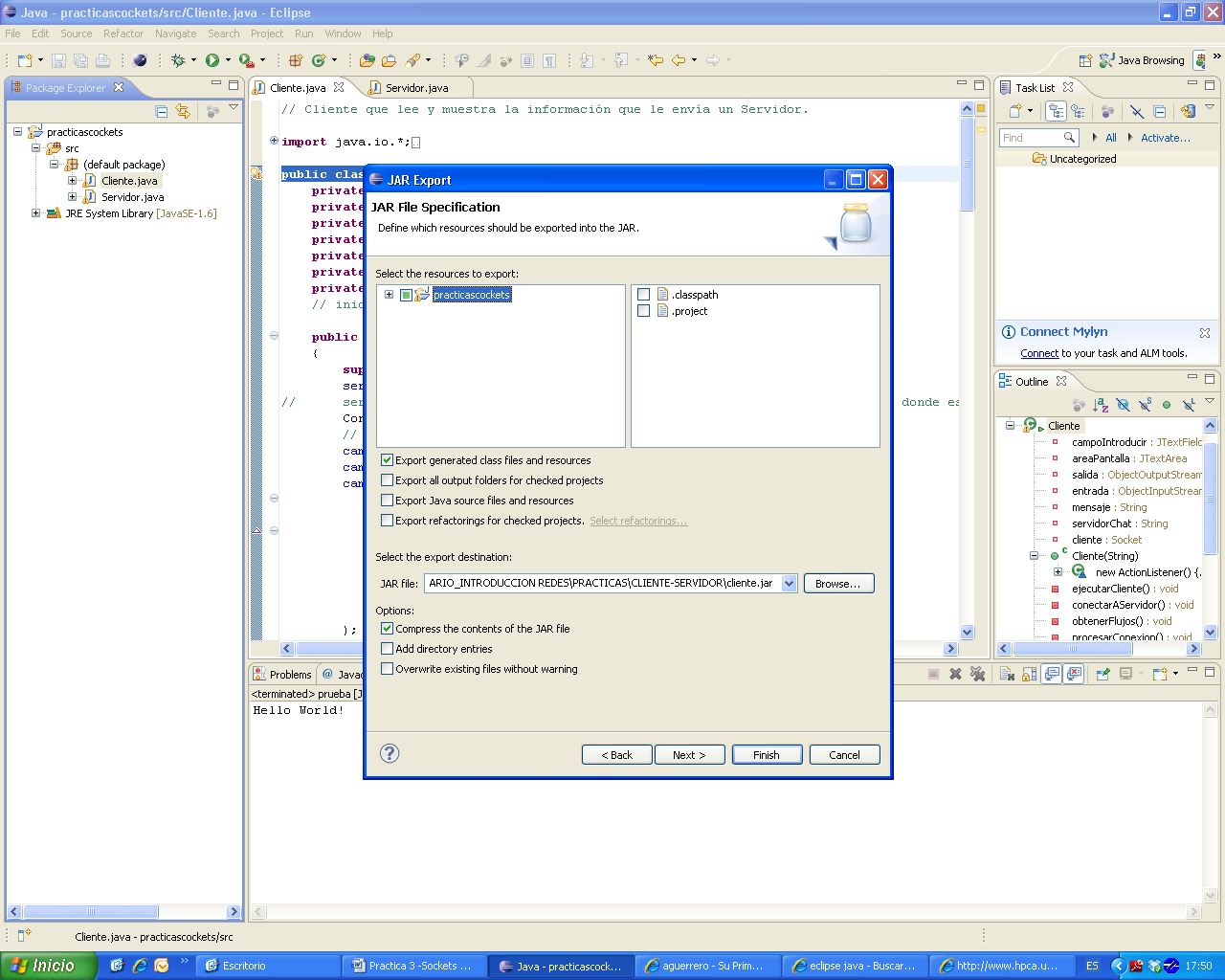


**Como crear un ejecutable**

Para generar un jar "ejecutable" desde eclipse, seleccionamos la clase y al presionar el botón derecho aparece el menú despegable en el que seleccionamos "**Export**".

|  |  |
| --- | --- |
|  |  |

En la ventana que aparece, se despliega el árbol de "java" y elegimos **"Runnable JAR file"**. Seleccionar "Next". Aparece el siguiente menú despegable donde se da nombre al fichero y se le asigna el directorio de salida. Seleccionando “Finish” creara el fichero ejecutable.



**2.- SOCKETS EN JAVA**

Un Socket es una interfaz de entrada-salida de datos que permite la intercomunicación entre procesos, en la capa de transporte. Se utiliza como direccionamiento, la combinación de una **dirección IP** + **número de puerto TCP**.

Los sockets no son más que puntos o mecanismos de comunicación entre procesos que permiten que un proceso hable (emita o reciba información) con otro proceso incluso estando estos procesos en distintas máquinas. Los sockets permiten la comunicación entre procesos, como los teléfonos permiten la comunicación entre las personas.

La forma de referenciar un socket por los procesos implicados es mediante un **descriptor** del mismo tipo que el utilizado para referenciar ficheros. Las operaciones de socket se parecen a las operaciones de archivo en muchos aspectos:

* Un socket **se crea** cuando se necesita, y el sistema operativo proporciona un entero de bajo valor (un descriptor de socket, exactamente análogo a un descriptor de archivo) que se usa para referirse después a él.
* Las operaciones de transferencia de datos en sockets simplemente trabajan como las operaciones de **lectura y escritura** de archivos.
* Cuando se terminan las comunicaciones un socket es **cerrado** como un archivo.

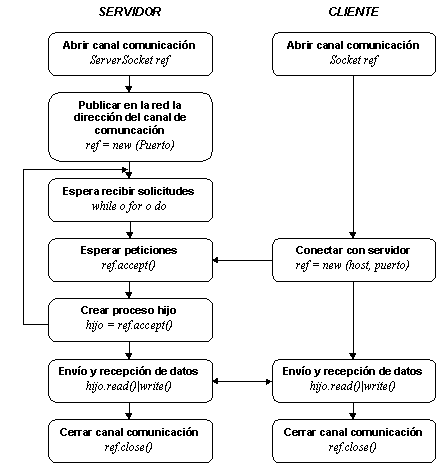
La comunicación entre procesos a través de sockets se basa en la filosofía **CLIENTE-SERVIDOR**: un proceso en esta comunicación actuará de **proceso servidor** **creando un socket cuyo nombre conocerá el proceso cliente**, el cual podrá "hablar" con el proceso servidor a través de la conexión con dicho socket nombrado.

El proceso **cliente crea un socket sin nombre** cuyo valor de vuelta es un descriptor sobre el que se leerá o escribirá, permitiéndose una comunicación bidireccional, característica propia de los sockets y que la diferencia de los pipes, o canales de comunicación unidireccional entre procesos de una misma máquina. El mecanismo de comunicación vía sockets tiene los siguientes pasos:

1. El proceso **servidor** crea un socket con nombre y espera la conexión.
2. El proceso **cliente crea** un socket sin nombre.
3. El proceso **cliente** realiza una **petición de conexión** **al** socket **servidor**.
4. El cliente realiza la conexión a través de su socket (enlaza con el socket del servidor) mientras el proceso servidor mantiene el socket servidor original con nombre.

Es muy común en este tipo de comunicación lanzar un proceso hijo, una vez realizada la conexión, que se ocupe del intercambio de información con el proceso cliente mientras el proceso padre servidor sigue aceptando conexiones.

El siguiente esquema muestra de manera simplificada el funcionamiento de los sockets.



**PROGRAMACIÓN DE LOS SOCKETS EN JAVA**

Java proporciona mecanismos potentes y a la vez sencillos para construir programas para redes tales como Internet.

Utilizan una serie de primitivas para establecer el punto de comunicación, para conectarse a una máquina remota en un determinado puerto que esté disponible*,* para escuchar en él, para leer o escribir y publicar información en él, y finalmente para desconectarse.

**Crear el socket servidor**

Para **crear** un servidor en java proporciona la clase **ServerSocket**. Al instanciarla se usa un constructor al que se le pasa un número de puerto. Este número de puerto puede ser cualquier entero entre 1 y 65535. Recuerda lo visto sobre los puertos en clase. No puede haber dos servidores atendiendo al mismo puerto/servicio en mismo equipo.

ServerSocket socket = new ServerSocket (xxxxxx);

**Aceptar un cliente**

Una vez creado el servidor, se le solicita que **comience a atender conexiones de clientes**. Para ello se llama al método **accept()**. Este método se queda esperando hasta que algún cliente se conecta y devuelve el **Socket**, que es la conexión con dicho cliente.

Socket cliente = socket.accept();

Se pueden aceptar simultáneamente varios clientes (no se implementa en el ejemplo)

**Obtener el InputStream y/o OutputStream**

En el *cliente* se tiene el descriptor de la conexión. Hay que obtener de él los canales de entrada y salida de datos **OuputStream** o **InputStream** con los métodos **getOutputStream()** o **getInputStream()**. La clase **OutpuStream** sirve para enviarle datos al cliente. La clase **InputStream** sirve para leer datos del cliente.

InputStream entrada = cliente.getIntputStream();  
OutputStream salida = cliente.getOutputStream();

Se crea un **InputStream** y/o un **OutputStream en función de las n**ecesidades de la aplicación.

Los métodos de estas dos clases únicamente envían bytes. Suele ser habitual construir alguna otra clase de entrada/salida de datos que tenga métodos más adecuados:

* Si se quiere enviar o recibir tipos normales (enteros, flotantes, strings) se tienen las clases **DataInputStream** y **DataOutputStream**. Estas clases tienen un constructor que admite un **InputStream** y un **OutputStream** respectivamente.

*DataInputStream entradaDatos = new DataInputStream (entrada);  
DataOuputStream salidaDatos = new DataOutputStream (salida) ;*

* Si se quiere enviar o recibir clases enteras propias, se tienen las clases **ObjectInputStream** y **ObjectDataStream**. Al igual que las anteriores, se usa el constructor que admite un **InputStream** y un **OutputStream**

*ObjectInputStream entradaObjetos = new ObjectInputStream (entrada);  
ObjectOutputStream salidaObjetos = new ObjectOutputStream (salida);*

Estas nuevas clases tienen métodos más intuitivos de usar **readObject()** y **writeObject()**.

#### Cierre del socket

Para cerrar un socket hay que llamar a la función **close()**.

*cliente.close(); // Con esto se cierra la conexión con el cliente.   
socket.close(); // Con esto se cierra el socket servidor, ya no atendemos más conexiones.*

**Crear el socket Cliente**

Para crear un **Socket** cliente se usa la clase **Socket**. **Basta instanciar esta indicándole contra que máquina conectarse (IP) y el puerto con el que debe conectarse**. Debe ser el mismo que el puerto que está atendiendo el servidor.

*Socket socket = new Socket ("localhost", 35557);*

|  |  |
| --- | --- |
| El resto se realiza igual que en el servidor. |  |

**3.- PRACTICA: UN CHAT EN JAVA CON SOCKETS**

Junto con la practica tienes dos ficheros de texto **Cliente.txt** y **Servidor.txt** con las **clases Cliente** y S**ervidor** del chat escritas en Java.

1. Examina estos ficheros y **busca las clases java que se han mencionado.**
2. Cuestión 1: Comprueba y localiza en el código los pasos indicados en el diagrama de bloques para crear el sockets.

Crear el socket (new ServerSocket) en la clase Servidor en el método ejecutarServidor()….

Aceptar un cliente (socket.accept) en la clase Servidor en el método esperarConexion()

1. Cuestión 2: ¿Qué número de puerto utiliza el servidor?¿Qué número de puerto utiliza el cliente?

En cliente lo pone en cliente = new Socket(InetAddress.getByName(servidorChat), 12345);

El servidor 12345(se ve en servidor = new ServerSocket(12345, 100);)

1. Cuestión 3: Utilizando el entorno IDE de eclipse compila y ejecuta estas clases **en tu ordenador**. ¿Qué se debe ejecutar primero: el servidor o el cliente?

Primero el servidor y luego el cliente

1. Cuestión 4: Utiliza el comando ***netstat -n*** para ver las conexiones establecidas y si existe algún problema. Modifica el programa si es necesario. ¿Qué número de puerto utiliza nuestro servidor? ¿Qué número de puerto utiliza nuestro cliente? ¿En qué dirección IP esta nuestro servidor? ¿Y el cliente?

Servidor 12345 y cliente 50364.

Ambas están en la dirección ip 127.0.0.1

1. Cuestión 5: Modifica lo que sea necesario para conectarte con otro ordenador del aula. Analiza la conexión con **netstat**. Explica los cambios realizados y el motivo de los mismos. ¿Qué número de puerto utiliza nuestro servidor? ¿Qué número de puerto utiliza nuestro cliente? ¿En qué dirección IP esta nuestro servidor? ¿Y el cliente (mirado desde el servidor)? ¿Pueden conectarse más de un cliente al mismo servidor?

En servidor cambias al crear el cliente el parámetro de la izquierda por la ip del otro ordenador

puerto Nuestro servidor 12345

puerto Nuestro cliente 51050

ip servidor 10.227.76.39 y ip cliente 10.227.76.38

Si.