Comunicaciones en Java
Sockets Cliente

Daniel Díaz Sánchez
Florina Almenárez
Andrés Marín
Departamento de Ingeniería Telemática
Universidad Carlos III de Madrid
dds@it.uc3m.es

Contexto

- **Objetivos**
  - Conocer las particularidades de las comunicaciones en Java, desarrollando pequeños programas que interactúen con protocolos conocidos

Índice

- Introducción a los sockets
- Conceptos básicos
- La clase Socket

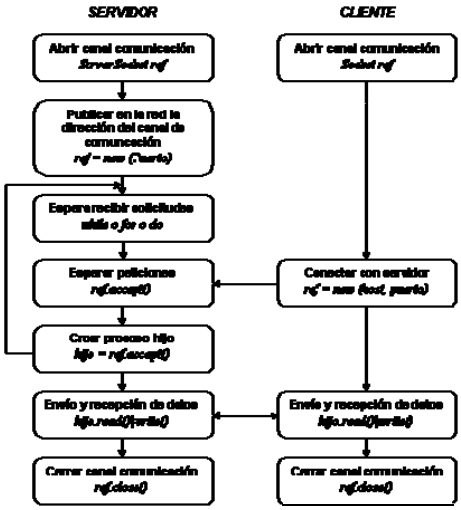
Introducción a los sockets

- Los sockets son un sistema de comunicación entre procesos. Un proceso, a través de un socket, puede recibir/enviar información de/a otro proceso localizado en otra máquina (o en la misma).

- Los sockets se hicieron populares gracias a las implementaciones de la Universidad de Berkeley de su sistema operativo tipo Unix llamado BSD.

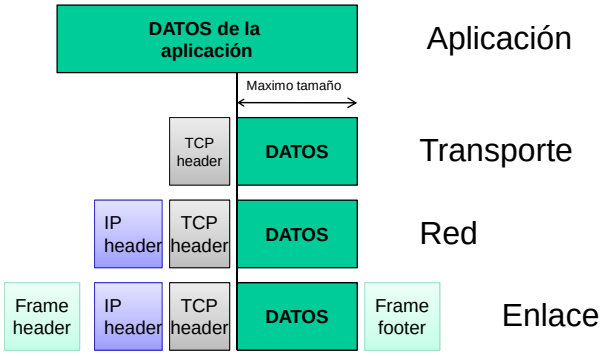
Introducción a los sockets

- El API de sockets define una serie de primitivas que combinadas dan lugar a este diálogo
- Durante esta sesión veremos como lo ha implementado Java.



Introducción a los sockets

- Los datos se transmiten en Internet usando paquetes.
- Los datos se manejan en capas, por ejemplo:



Introducción a los sockets

- Hay dos tipos de transporte básicos
  - Orientado a conexión
    - mantiene un canal de comunicación con el otro extremo de forma constante durante la conexión
    - La entrega de paquetes es ordenada
  - Orientado a datagrama
    - No hay canal de comunicación constante
    - La entrega es desordenada
- Hay varias formas de comunicarse
  - De uno a uno: Cliente-Servidor, Peer to Peer (P2P)
  - De uno a todos: broadcast
  - De uno a varios: multicast

Introducción a los sockets

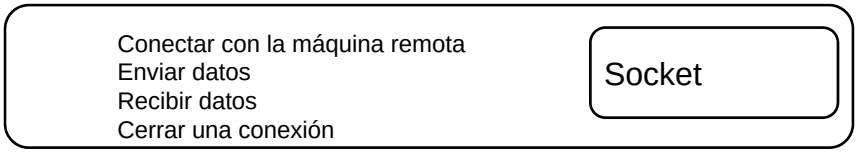
- Hay que realizar un gran número de operaciones con los datos:
  - Fraccionarlos
  - Etiquetarlos con las cabeceras de los diferentes protocolos
  - Controlar la congestión
  - Retransmitir paquetes que se hayan perdido
  - Administrar almacenamiento intermedio….
- Todas estas tareas son tediosas y complejas
  - Imaginad el coste de hacer todas estas operaciones en cada uno de las aplicaciones que se conectan a Internet

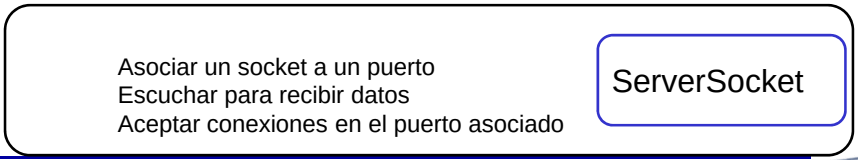
Introducción a los sockets

- Los sockets son una **abstracción** de esas tareas para facilitar la programación de aplicaciones con conectividad.
- Los sockets permiten:
  - A los clientes y servidores:
    - Conectar con la máquina remota (prot. orientados a conexión)
    - Enviar datos
    - Recibir datos
    - Cerrar una conexión (prot. orientados a conexión)
  - A los servidores:
    - Asociar un socket a un puerto
    - Escuchar para recibir datos
    - Aceptar conexiones en el puerto asociado

Introducción a los sockets

- Java separa en dos clases la funcionalidad de sockets orientados a conexión.
  - La clase Socket usada por clientes y servidores.
  - La clase ServerSocket usada únicamente por servidores.



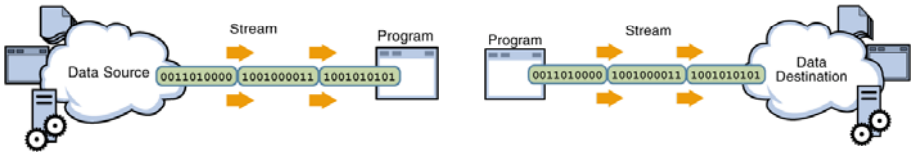


Conceptos básicos

- Java es un lenguaje orientado a la red
  - Todo programa en Java hace uso de la entrada y salida, el paquete IO y en el caso de las comunicaciones también
  - Vamos a repasar la entrada y salida en Java, es decir, el paquete java.io
- Los sockets utilizan unos identificadores de red como
  - Dirección IP y puerto
  - Familia de sockets
  - Vamos a repasar dichos identificadores

Conceptos básicos: java.io

- Cualquier operación de entrada y salida en Java utiliza un **stream** o flujo
  - Un stream es una abstracción de las operaciones de lectura y escritura reales
  - Permite usar el mismo modelo de entrada y salida (IO) con independencia del dispositivo que realiza tarea. Usaremos las mismas clases y métodos para interactuar con ficheros, pantalla, sockets…



Conceptos básicos: java.io

- Java define dos tipos de streams:
  - Byte streams: para el manejo de entrada y salida binaria.
    - La clase **abstracta** para lectura es `InputStream`.
    - La clase **abstracta** para escritura es `OutputStream`.
    - Ambas clases no se pueden usar directamente dado que son `abstract`. Estas clases serán extendidas por otras proporcionando la funcionalidad necesaria para realizar las operaciones de IO.
  - Character streams: para el manejo de entrada y salida de texto
    - La clase **abstracta** para lectura es `Reader`.
    - La clase **abstracta** para escritura es `Writer`.
    - Ambas clases no se pueden usar directamente dado que son `abstract`. Estas clases serán extendidas por otras proporcionando la funcionalidad necesaria para realizar las operaciones de IO.

Conceptos básicos: java.io

	Byte Based		Character Based	
	Input	**Output**	**Input**	**Output**
Basic	`InputStream`	`OutputStream`	`Reader` `InputStreamReader`	`Writer` `OutputStreamWriter`
Arrays	`ByteArrayInputStream`	`ByteArrayOutputStream`	`CharArrayReader`	`CharArrayWriter`
Files	`FileInputStream` `RandomAccessFile`	`FileOutputStream` `RandomAccessFile`	`FileReader`	`FileWriter`
Pipes	`PipedInputStream`	`PipedOutputStream`	`PipedReader`	`PipedWriter`
Buffering	`BufferedInputStream`	`BufferedOutputStream`	`BufferedReader`	`BufferedWriter`
Filtering	`FilterInputStream`	`FilterOutputStream`	`FilterReader`	`FilterWriter`
Parsing	`PushbackInputStream` `StreamTokenizer`		`PushbackReader` `LineNumberReader`	
Strings			`StringReader`	`StringWriter`
Data	`DataInputStream`	`DataOutputStream`		
Data - Formatted		`PrintStream`		`PrintWriter`
Objects	`ObjectInputStream`	`ObjectOutputStream`		
Utilities	`SequenceInputStream`			

Conceptos básicos: java.io

- Ejemplo de uso “Leer del teclado”
  - Java nos proporciona en su clase `System` acceso a lo que se conoce como entrada y salida estándar

Field Summary	
static PrintStream	err The "standard" error output stream.
static InputStream	in The "standard" input stream.
static PrintStream	out The "standard" output stream.

  - El teclado se representa en Java como `System.in` que es de tipo `InputStream`. Si nos fijamos en la tabla anterior, se trata de una entrada orientada a byte.
  - Como queremos leer caracteres y no bytes, crearemos un objeto `InputStreamReader` a partir de `System.in`.

Conceptos básicos: java.io

- Dado que `InputStreamReader` hereda de `Reader`, creamos un objeto de tipo `InputStreamReader`
  - `InputStreamReader(InputStream in)`
- Ahora disponemos de un objeto Reader que podremos recubrir con otra clase para su uso. Usaremos, por ejemplo un BufferedReader. Esta clase tiene un buffer interno que almacena varios caracteres y por tanto es útil si no se va a leer continuamente.
  - Finalmente, el código sería:

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in);
```

Conceptos básicos: java.io

- En el caso de las clases de IO orientadas a byte, se procedería de la misma forma, desde el tipo básico hasta el recubrimiento o wraper que más nos convenga.
- **Aplicación en sockets:**
  - Si tenemos un protocolo que utiliza una cabecera tipo texto para mandar contenido binario, usaremos
    - un stream orientado a carácter para la cabecera
    - un stream orientado a byte para escribir el fichero
  - Veremos ejemplos más adelante

Conceptos básicos: identificadores

- Todo socket se identifica por tres parámetros básicos:
  - HOST: nombre o dirección de la máquina
    - Ej: www.uc3m.es
    - Ej. 163.117.141.114
  - PUERTO: identifica a un proceso dentro de la máquina, es decir, una máquina con una dirección puede tener varios procesos intercambiando tráfico. Con el puerto distinguimos unos de otros
  - PROTOCOLO: mecanismo de intercambio de datos que se usará
- Por otro lado hay otros parámetros como familias de sockets… que iremos viendo cuando sea necesario.

Conceptos básicos: IDE

- Se recomienda el uso de un IDE: “Integrated Development Environment” donde podais programar más cómodamente.
- Eclipse es sin duda el mejor
  - http://www.eclipse.org/
  - Es fácil de usar
  - Gran cantidad de tutoriales

La clase Socket: constructores

- La clase Socket está en el paquete java.net.socket
  - Proporciona soporte de TCP/IP
  - Los constructores toman como parámetros:
    - El HOST se puede especificar como InetAddress o String
    - El PUERTO consisten en un entero (int) entre 0-65535
    - Si la máquina donde se ejecuta el programa dispone de más de un interfaz de red (ej. está conectada a la vez por cable y wifi), se puede especificar el interfaz que se debe usar para enviar/recibir datos (multihoming)
  - Las excepciones pueden ser de dos tipos:
    - UnknownHostException: cuando el host se especifica como string y no es posible resolver la dirección IP con DNS
    - IOException: por errores de entrada y salida, falta de permisos, interfaz de red erroneo…

La clase Socket: constructores

Java.net.Socket

Constructor

public Socket(String host, int port) throws UnknownHostException, IOException

Descripción/Parámetros

Crea un socket TCP para comunicarse con la máquina descrita con los parámetros host/port y trata de abrirlo.

Ejemplo

```
try{
   Socket uc3m = new Socket("www.uc3m.es",80);
   // código para enviar y recibir datos
}catch(UnknownHostException uhe){
   uhe.printStackTrace();
}catch(IOException ex){
   ex.printStackTrace();
}
```

La clase Socket: constructores

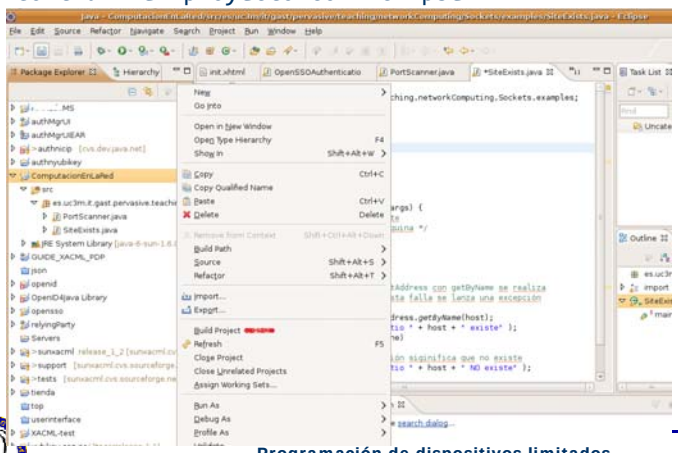
Ejemplo. Escaneando puertos

```
package es.uc3m.it.gast.pervasive.teaching.networkComputing.Sockets.examples;
import java.net.*;
import java.io.*;
public class PortScanner {
     public static void main(String[] args) {
            /* por defecto usamos esta máquina */
            String host = "localhost";
            if (args.length > 0) {
                        host = args[0];
            }
            System.out.println("Comprobando puertos en " + host);
            System.out.println("--------------------------------");
            for (int i = 1; i < 1024; i++) {
                        try {
                                    /* el constructor creará el socket y tratará de abrirlo */
                                    Socket s = new Socket(host, i);
                                    /* si logra abrir un socket en ese puerto significa que hay un proceso
                                     escuchando*/
                                    System.out.println("Puerto " + i + " : abierto ");
                        } catch (UnknownHostException ex) {
                                    /* no debería ocurrir */
                                    System.out.println("El host no existe, comprueba tus parámetros");
                                    break; //salir del bucle y del programa
                        } catch (IOException ex) {
                                    /* Si hay una excepción, significa que no se puede abrir el socket
                                     * en ese puerto y por tanto no hay ningún proceso escuchando ahí */
                                    System.out.println("Puerto " + i + " : cerrado ");
                        }
            }
     }
```

La clase Socket: constructores

Ejecutando el ejemplo (1/2)

Contruir el proyecto con Eclipse



La clase Socket: constructores

Ejecutando el ejemplo (2/2)

- Para ejecutar/depurar el ejemplo:
  - Puedes usar los comandos Run/Debug de Eclipse
  - Puedes ir al directorio donde está el workspace, entrar en el directorio bin y ejecutar:

```
~/workspace/ComputacionEnLaRed/bin$ java -cp . es.uc3m.it.gast.pervasive.teaching.networkComputing.Sockets.examples.PortScanner
Comprobando puertos en localhost
--------------------------------
Puerto 1 : cerrado
Puerto 2 : cerrado
Puerto 3 : cerrado

...
```

La clase Socket: construtores

Java.net.Socket

Constructor

public Socket(InetAddress host, int port) throws IOException

Descripción/Parámetros

Este constructor crea un socket TCP y trata de abrirlo. En este caso, se utiliza la clase InetAddress para identificar al host. Como puede comprobar, el constructor solo arroja la excepción IOException si no puede abrirlo. La excepción UnknownHostException no la lanza en este caso el constructor sino InetAddress si la dirección no existe.

Ejemplo

```
try {
    InetAddress uc3m = InetAddress.getByName("www.uc3m.es");
    Socket uc3mSocket = new Socket(uc3m , 80);  // send and receive data...
}catch (UnknownHostException ex) {  System.err.println(ex);
}catch (IOException ex) {  System.err.println(ex); }
```

La clase Socket: construtores

Ejemplo, ¿existe el sitio?

```
import java.net.*;
import java.io.*;

public class SiteExists {

    public static void main(String[] args) {
            // Comprueba si un sitio existe
            /* por defecto usamos esta máquina */
            String host = "localhost";
            if (args.length > 0) {
                        host = args[0];
            }
            try{
                        /* Al crear un objeto InetAddress con getByName se realiza
                         * una consulta DNS. Si esta falla se lanza una excepción
                         */
                        InetAddress test = InetAddress.getByName(host);
                        System.out.println("El sitio " + host + " existe" );
            }catch(UnknownHostException uhe)
            {
                        //Si se lanza esta excepción siginifica que no existe
                        System.out.println("El sitio " + host + " NO existe" );
            }
    }
}
```

La clase Socket: construtores

Ejemplo. Escaneando puertos. Versión 2.

```
import java.net.*;
import java.io.*;
public class PortScanner2 {
    public static void main(String[] args) {
            String host = "localhost";
            if (args.length > 0) {
                        host = args[0];
            }
            InetAddress test;
            try{
                        test = InetAddress.getByName(host);
            }catch(UnknownHostException ex){
                        System.out.println("El host no existe, comprueba tus parámetros");
                        return;
            }
            System.out.println("Comprobando puertos en " + host);
            System.out.println("-------------------------------");
            for (int i = 1; i < 1024; i++) {
                        try {
                                    Socket s = new Socket(test, i);
                                    System.out.println("Puerto " + i + " : abierto ");
                        } catch (IOException ex) {
                                    System.out.println("Puerto " + i + " : cerrado ");
                        }
            }
    }
}
```

La clase Socket: constructores

Java.net.Socket

Constructor

public Socket(String host, int port, InetAddress interface, int localPort) throws IOException, UnknownHostException

Descripción/Parámetros

Si nuestro ordenador tiene dos conexiones a Internet (por ejemplo WiFI y Ethernet) con este constructor es posible seleccionar el interfaz por el que saldrán los paquetes

Ejemplo

El uso de este constructor es sencillo. Lo más complicado es saber cuantos interfaces de red están disponibles y conectados.
Veamos un ejemplo.
Usaremos la clase **java.net.NetworkInterface**

La clase Socket: constructores

Listar los interfaces de red disponibles en el ordenador

```
package es.uc3m.it.gast.pervasive.teaching.networkComputing.Sockets.examples;

import java.io.*;
import java.net.*;
import java.util.*;

/* tomado de un tutorial de "The Java Tutorials"
 * http://java.sun.com/docs/books/tutorial/networking/nifs/listing.html */
public class ListNetworkInterfaces {

        public static void main(String args[]) throws SocketException {
            Enumeration<NetworkInterface> nets = NetworkInterface.getNetworkInterfaces();
            for (NetworkInterface netint : Collections.list(nets))
                displayInterfaceInformation(netint);
        }

        static void displayInterfaceInformation(NetworkInterface netint) throws SocketException {
            System.out.printf("Display name: %s\n", netint.getDisplayName());
            System.out.printf("Name: %s\n", netint.getName());
            Enumeration<InetAddress> inetAddresses = netint.getInetAddresses();
            for (InetAddress inetAddress : Collections.list(inetAddresses)) {
                        System.out.printf("InetAddress: %s\n", inetAddress);
            }
            System.out.printf("\n");
         }
    }
```

La clase Socket: constructores

Listar los interfaces de red disponibles en el ordenador

Una vez sabemos cuantos interfaces disponibles hay, podemos usar el que más nos convenga.

```
user@host:~/ComputacionEnLaRed/bin$ java -cp .
   es.uc3m.it.gast.pervasive.teaching.networkComputing.Sockets.examples.ListNetw
   orkInterfaces
Display name: eth0
Name: eth0
InetAddress: /fe80:0:0:0:a00:27ff:feb3:e70f%2
InetAddress: /10.0.2.15

Display name: lo
Name: lo
InetAddress: /0:0:0:0:0:0:0:1%1
InetAddress: /127.0.0.1
```

La clase Socket: métodos

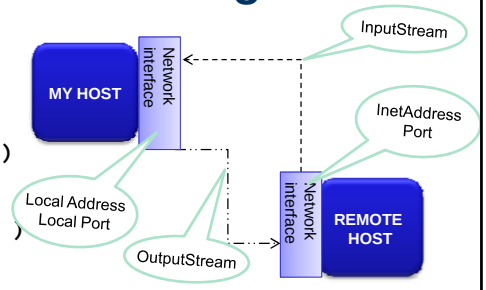
- **Getters**: obtener información sobre el socket
- **Entrada/Salida**: Obtener input y output stream
  - Input: mecanismo para obtener un *inputstream* para escribir en el socket
  - Output: mecanismo para obtener un *outputstream* para leer del socket
- **Cierre** del socket
- **Setters**: Opciones de sockets

La clase Socket: métodos: getters

Java.net.Socket

Métodos

```
public InetAddress getInetAddress( )
public int getPort( )
public int getLocalPort( )
public InetAddress getLocalAddress( )
```



Descripción/Parámetros

Un socket TCP se puede modelar con un par dirección-puerto origen y un par dirección-puerto destino:

- getInetAddress devuelve un objeto InetAddress indicando la dirección del host remoto.
- getPort devuelve un entero indicando el puerto del host remoto.
- getLocalAddress devuelve un objeto InetAddress que identificca cual de los interfaces de red de nuestro ordenador estamos usando.
- getLocaPort devuelve un entero indicando el puerto de nuestra máquina que estamos usando con el socket.

La clase Socket: métodos: getters

Obtener información de un socket abierto

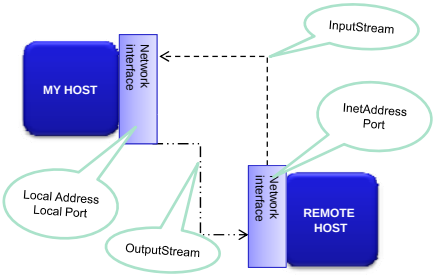
```
package es.uc3m.it.gast.pervasive.teaching.networkComputing.Sockets.examples;

import java.net.*;
import java.io.*;

public class GetInfoFromSocket {

    public static void main(String[] args) {

            try {
            Socket mySocket = new Socket(args[0], 80);
            System.out.println("Conectado a el host " + mySocket.getInetAddress()
                        + " con direccion IP " + mySocket.getInetAddress().getHostAddress()
                        + " en el puerto remoto " + mySocket.getPort()
                        + " desde el puerto local " + mySocket.getLocalPort()
                        + " de mi interfaz local " + mySocket.getLocalAddress());
            }
            catch (UnknownHostException ex) {
                        System.err.println("No puedo encontrar la direccion " + args[0]);
            } catch (SocketException ex) {
                        System.err.println("No puedo conectar con la direccion  " + args[0]);
            } catch (IOException ex) {
                        System.err.println(ex);
            }
    }
}
```

La clase Socket: métodos: E/S

Java.net.Socket

Métodos

```
public InputStream getInputStream( )
   throws IOException
public OutputStream getOutputStream( )
   throws IOException
```



Descripción/Parámetros

- getInputStream devuelve un objeto de tipo InputStream que habrá que recubrir con cualquier filter stream como DataInputStream o InputStreamReader. También es una buena idea recubrirlo con un BufferedReader or BufferedInputStream por motivos de rendimiento.
- getOutputStream devuelve un objeto de tipo OutputStream. Del mismo modo que con getInputStream habrá que recubrirlo para escribir texto o datos binarios o proporcionar un buffer.

La clase Socket: métodos: E/S

Solicitar una página web (1)

Vamos a solicitar una página web a un servidor. Para ello, debemos saber como funciona HTTP 1.1 (el protocolo que usaremos)

RFC 2616 (1999).
Métodos: GET, HEAD, POST, PUT, DELETE, OPTIONS, TRACE, CONNECT
Cabecera Date: obligatoria en respuestas.
Cabecera Host: obligatoria en peticiones.

El método que vamos a usar es GET.

¿Como sería la petición?

La clase Socket: métodos: E/S

Solicitar una página web (2)

Para solicitar la página http://www.google.es, necesitamos la petición HTTP (\n significa retorno de carro):

```
GET / HTTP/1.1\n
Host:www.google.es\n
\n
```

Obtendremos la respuesta

```
HTTP/1.1 200 OK
Date: Mon, 01 Mar 2010 16:16:35 GMT
Expires: -1
Cache-Control: private, max-age=0
--linea en blanco—
<html> …
```

La clase Socket: métodos: E/S

Solicitar una página web (3)

Para investigar un poco como funciona http, es posible usar el comando telnet, por ejemplo:

```
~>telnet www.uc3m.es 80
Trying 163.117.136.249...
Connected to turan.uc3m.es.
Escape character is '^]'.
GET / HTTP/1.1
Host: www.ietf.org:80

HTTP/1.1 200 OK
Date: Mon, 12 Feb 2007 21:56:03 GMT
Server:  Apache/1.3.33  (Debian  GNU/Linux)  PHP/4.3.10-18  mod_ssl/2.8.22
   OpenSSL/0.9.7e DAV/1.0.3
Last-Modified: Mon, 23 Jan 2006 17:40:26 GMT
ETag: "6417e-534-43d5150a"
Accept-Ranges: bytes
Content-Length: 1332
Content-Type: text/html; charset=iso-8859-1

<html>
[…]
</html>
[la conexión se mantiene abierta]
```

La clase Socket: métodos: E/S

Solicitar una página web (4)

Dado que HTTP puede usarse para transmitir texto (páginas web) y cualquier otro tipo de dato (imágenes, videos, flash…) es necesario fijarse en el **content-type**.

```
GET / HTTP/1.1
Host: www.uc3m.es
Connection:close

HTTP/1.1 200 OK
Date: Mon, 12 Feb 2007 21:56:34 GMT
Server: Apache/1.3.33 (Debian GNU/Linux) PHP/4.3.10-18 mod_ssl/2.8.22
   OpenSSL/0.9.7e DAV/1.0.3
Last-Modified: Mon, 23 Jan 2006 17:40:26 GMT
ETag: "6417e-534-43d5150a"
Accept-Ranges: bytes
Content-Length: 1332
Connection: close
Content-Type: text/html; charset=iso-8859-1

<html>
[…]
</html>
Connection closed by foreign host.
```

La clase Socket: métodos: E/S

Solicitar una página web (5)

¿Qué debe hacer el programa?

1. Abrir un socket
2. Obtener el outputstream y recubrirlo para escribir texto (las peticiones http siempre son en texto)
3. Obtener el inputStream y recubrirlo para leer texto, de forma que podamos leer las cabeceras (que son texto) y el contenido (que lo limitaremos a texto)
4. Enviar petición HTTP
5. Comprobar si la respuesta es correcta (código 200)
6. Obtener las cabeceras HTTP y colocarlas en un HashMap
7. Obtener el tipo de contenido y su tamaño
8. Leer contenido calculando los bytes leidos y guardar el contenido en un fichero

La clase Socket: métodos: E/S

Solicitar una página web (6)

¿Qué debe hacer el programa?

1. Abrir un socket

dado el formato de entrada:

```
SimpleHTTPClient www.it.uc3m.es /dds/index.html
```

Preparamos la captura de parámetros y abrimos un socket en el puerto 80 (el de HTTP)

```
if (args.length < 2) {
   System.err.println("Debes proporcionar dos parámetros host y fichero");
   System.err.println("Por ejemplo: SimpleHTTPClient www.it.uc3m.es / ");
   System.err.println("o: SimpleHTTPClient www.it.uc3m.es /dds/index.html ");
   System.exit(-1);
}
/* Creamos el socket y lo abrimos */
Socket mySocket = new Socket(args[0], 80);
```

La clase Socket: métodos: E/S

Solicitar una página web (7)

¿Qué debe hacer el programa?

2. Obtener el outputstream y recubrirlo para escribir texto (las peticiones http siempre son en texto)

```
/* El output stream nos permite leer caracteres del socket */

OutputStream os = mySocket.getOutputStream();

/* Un outputStreamWriter convierte caracteres en bytes */

OutputStreamWriter osw = new OutputStreamWriter(os);

/*
 * como vamos a escribir líneas completas (las consultas en http
   terminan con retorno de carro). Usaremos un printWriter */

PrintWriter escritura = new PrintWriter(osw);
```

La clase Socket: métodos: E/S

Solicitar una página web (8)

¿Qué debe hacer el programa?

3. Obtener el inputStream y recubrirlo para leer texto, de forma que podamos leer las cabeceras (que son texto) y el contenido (que lo limitaremos a texto)

```
 /* Obtenemos el input stream que nos permite leer del socket */
InputStream is = mySocket.getInputStream();
/*
 * Un InputStreamReader es un puente entre byte y caracter
 * tomará todos los bytes y los convertirá en caracteres
 */

InputStreamReader isr = new InputStreamReader(is);

/* usamos un buffered reader */
BufferedReader lectura = new BufferedReader(isr);
```

La clase Socket: métodos: E/S

Solicitar una página web (9)

¿Qué debe hacer el programa?

4. Enviar petición HTTP

```
/* Realizando la petición */
/* Formato: GET fichero HTTP/1.1\n */

escritura.println("GET " + args[1] + " HTTP/1.0");

/* añadimos cabecera host: host: args[0]\n */

escritura.println("HOST:" + args[0]);

/* no haremos más peticiones. Solicitamos cierre. */
escritura.println("Connection: close");

/* retorno de carro para señalar que no hay más cabeceras */
escritura.println();

/* flush obliga a vaciar buffers y enviar todo al otro lado */
escritura.flush();
```

La clase Socket: métodos: E/S

Solicitar una página web (10)

¿Qué debe hacer el programa?

5. Comprobar si la respuesta es correcta (código 200)

```
/* Leemos la respuesta */
String respuesta = lectura.readLine();

/* Si todo ha ido bien, la respuesta debe ser HTTP/1.1 200 OK */
/* Partimos la respuesta en trozos separando en cada espacio */
StringTokenizer st = new StringTokenizer(respuesta, " ");
System.out.println("Versión " + st.nextToken());
String responseCode = st.nextToken();
int code = Integer.decode(responseCode);
System.out.println("Respuesta " + responseCode + st.nextToken());

if (code != 200) {
   System.err.println("Se ha producido un error");
   System.exit(-1);
}
```

La clase Socket: métodos: E/S

Solicitar una página web (11)

¿Qué debe hacer el programa?

6. Obtener las cabeceras HTTP y colocarlas en un HashMap

```
/*
 * Si todo ha ido bien, tendremos una respuesta con varias cabeceras
 * las cabeceras se terminan con una linea en blanco seguido de \n
 * sabemos que al menos debe aparecer la cabecera date segun HTTP
 * 1.1
 */
Map<String, String> cabeceras = new HashMap<String, String>();
String cabecera = "";

while (true) {
   cabecera = lectura.readLine();

   // la partimos en dos usando ":" como delimitador
   st = new StringTokenizer(cabecera, ":");
   if (st.countTokens() < 2)
        break; // el final de cabeceras es una linea en blanco

   // quitamos los posibles espacios
   cabeceras.put(st.nextToken().trim(), st.nextToken().trim());
}
```

La clase Socket: métodos: E/S

Solicitar una página web (12)

¿Qué debe hacer el programa?

7. Obtener el tipo de contenido y su tamaño

```
/*vamos a obtener las cabeceras de tipo de contenido y longitud */
String contentType = cabeceras.get("Content-Type");
//debugPrintHeaders(cabeceras);
/* vamos a descargar el contenido. De momento solo soporta text */
String tipo = "";
String subtipo = "";
int contentLength = 0;

if (contentType.contains("text")) {
    st = new StringTokenizer(contentType, "/");
    tipo = st.nextToken();
    subtipo = st.nextToken();
    System.out.print("El content-type es de tipo \"" + tipo …
    contentLength = Integer.parseInt(cabeceras.get("Content-Length"));
    System.out.println(" de longitud " + contentLength + ".");
} else {
    System.err.println("Contenido no soportado. SimpleHTTPClient solo soporta
    text");
    System.exit(-1);
}
```

La clase Socket: métodos: E/S

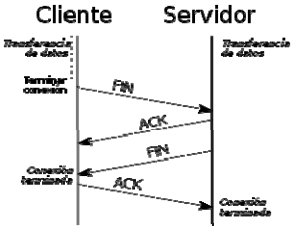
Solicitar una página web (13)

¿Qué debe hacer el programa?

8. Leer contenido calculando los bytes leídos y guardar el contenido en un fichero

```
int read = 0;
int restante = contentLength;
char[] cbuffer = new char[1024];
File file = new File(args[0] + args[1].replace('/', '.') + ".“+ subtipo);
FileWriter fr = new FileWriter(file);

while (restante>0) {
    read = lectura.read(cbuffer, 0, (restante > 1024) ? 1024: restante);

    if (read > 0) {
          //convertimos a String para calcular el número de bytes
          // solo soporta la codificación por defecto, por lo que es muy limitado
          String temp = new String(cbuffer,0,read);
          fr.write(temp);
          fr.flush();
          restante -= temp.getBytes().length;
    }

}

fr.flush();
fr.close();
lectura.close();
```

La clase Socket: métodos: cierre



Java.net.Socket

Método

public void close( ) throws IOException
public boolean isClosed( )

Descripción/Parámetros

Las conexiones TCP deben cerrarse. Hasta ahora no ha importado dado que el cierre de la aplicación cierra el socket. Pero si abrimos y cerramos varios en nuestra aplicación, hemos de cerrarlos para que TCP realice el cierre ordenado.

Para cerrar el socket simplemente se debe llamar a **close**.

Para determinar si un socket está abierto o cerrado se usaría **isClosed**

Ejemplo

```
connection = new Socket(host, i);
// leer – escribir
Connection.close();
```

Código de los ejemplos

- El código de los ejemplos está disponible en la sección de prácticas