.

**Solucionario de ejercicios**

**3.1. Escribe una pareja de programas (A y B) que transfieran un fichero entre ellos. El programa A deberá leer un fichero de texto del disco y enviarlo a B. B recibirá el contenido del fichero y lo imprimirá por su salida estándar. Utiliza para ello *sockets* *stream*.**

**Solución**

La solución es similar a los ejemplos vistos en el libro. El programa A utiliza la clase *PrintWriter* para realizar el envío de la información de manera más cómoda.

Código del programa A:

**import** java.io.BufferedReader;   
**import** java.io.FileReader;   
**import** java.io.IOException;

66

**import** java.io.OutputStream;   
**import** java.io.PrintWriter;

**import** java.net.InetSocketAddress; **import** java.net.Socket;

**public** **class** Ej1A {

**3** n **PROGRAMACIÓN DE COMUNICACIONES EN RED**

**public** **static** **void** **main**(String[] args) {   
 **try** {

// Creando socket cliente

Socket clientSocket = **new** Socket(); // Estableciendo la conexión

InetSocketAddress addr = **new** InetSocketAddress(“localhost”, 5555); clientSocket.connect(addr);

OutputStream os = clientSocket.getOutputStream(); // Abriendo el fichero

BufferedReader br = **new** BufferedReader(**new** FileReader(“fichero\_prueba.txt”)); // Enviando el contenido del fichero

PrintWriter pw = **new** PrintWriter(os, **true**);

**while** (br.ready()) {

String line = br.readLine(); pw.println(line);

}

// Cerrando el socket cliente clientSocket.close();

// Terminado

} **catch** (IOException e) {   
 e.printStackTrace(); }

}   
}

67

**Programación de Servicios y Procesos**

Código del programa B:

**import** java.io.BufferedReader;

**import** java.io.IOException;

**import** java.io.InputStream;

**import** java.io.InputStreamReader; **import** java.net.InetSocketAddress; **import** java.net.ServerSocket;   
**import** java.net.Socket;

**public** **class** Ej1B {

**public** **static** **void** **main**(String[] args) {   
 **try** {

// Creando socket servidor

ServerSocket serverSocket = **new** ServerSocket();

// Realizando el bind

InetSocketAddress addr = **new** InetSocketAddress(“localhost”, 5555); serverSocket.bind(addr);

// Aceptando conexiones

Socket newSocket = serverSocket.accept();

System.out.println(“Conexión recibida”);

InputStream is = newSocket.getInputStream();

BufferedReader br = **new** BufferedReader(**new** InputStreamReader(is));

String line = br.readLine();

**while** (line != **null**) {

System.out.println(line);   
 line = br.readLine();   
 }

// Cerrando el nuevo socket newSocket.close();

//Cerrando el socket servidor

68

**3** n **PROGRAMACIÓN DE COMUNICACIONES EN RED**

serverSocket.close(); // Terminado

} **catch** (IOException e) {   
 e.printStackTrace(); }

}   
}

**3.2. Escribe un programa que conteste a preguntas. El programa creará un *socket* *stream* y aguardará conexiones. Cuando llegue una conexión, leerá los mensajes recibidos, byte a byte, hasta que encuentre el carácter ASCII “?” (signo de final de interrogación). Cuando esto ocurra, construirá una frase con todos los bytes recibidos y contestará con un mensaje. El contenido del mensaje dependerá de la frase recibida:**

n**n Si la frase es “¿Cómo te llamas?”, responderá con la cadena “Me llamo Ejercicio 2”.**

n**n Si la frase es “¿Cuántas líneas de código tienes?”, responderá con el número de líneas de**

**código que tenga.**

n**n Si la frase es cualquier otra cosa, responderá “No he entendido la pregunta”.**

**Solución**

A continuación se muestra la solución y dos programas de ejemplo. El primer ejemplo realiza una pregunta válida (“¿Cómo te llamas?”) y el segundo una inválida (“¿Cuántos años tienes?”). El programa principal del ejercicio responderá lo adecuado en cada caso.

Código del programa:

**import** java.io.BufferedReader;

**import** java.io.IOException;

**import** java.io.InputStream;

**import** java.io.InputStreamReader;   
**import** java.io.OutputStream;   
**import** java.io.PrintWriter;   
**import** java.net.InetSocketAddress;   
**import** java.net.ServerSocket;   
**import** java.net.Socket;

69

**Programación de Servicios y Procesos**

**public** **class** Ej2 {

**public** **static** **void** **main**(String[] args) {   
 **try** {

// Creando socket servidor

ServerSocket serverSocket = **new** ServerSocket();

// Realizando el bind

InetSocketAddress addr = **new** InetSocketAddress(“localhost”, 5555); serverSocket.bind(addr);

// Aceptando conexiones

Socket newSocket = serverSocket.accept();   
System.out.println(“Conexión recibida”);

InputStream is = newSocket.getInputStream();

OutputStream os = newSocket.getOutputStream();

BufferedReader br = **new** BufferedReader(**new** InputStreamReader(is)); PrintWriter pw = **new** PrintWriter(os, **true**);

String mensaje = “”;

**char**[] buffer = **new** **char**[1]; **int** tam = 0;

**while** ((tam != -1) && (buffer[0] != ‘?’)) {   
 tam = br.read(buffer);

**if** (tam != -1)

mensaje = mensaje + buffer[0];   
}

String respuesta = **null**;

**if** (mensaje.equals(“¿Cómo te llamas?”))   
 respuesta = “Me llamo Ejercicio 2”;

**else** **if** (mensaje.equals(“¿Cuántas líneas de código tienes?”))   
 respuesta = “Tengo 74 líneas de código”;

**else**

respuesta = “No he entendido la pregunta”;

pw.println(respuesta);

70

// Cerrando el nuevo socket

newSocket.close();

//Cerrando el socket servidor

serverSocket.close();

// Terminado

} **catch** (IOException e) {   
 e.printStackTrace();

}

}

}

Código del ejemplo 1:

**import** java.io.BufferedReader; **import** java.io.InputStream;

**import** java.io.IOException;   
**import** java.io.InputStreamReader; **import** java.io.OutputStream;   
**import** java.io.PrintWriter;   
**import** java.net.InetSocketAddress; **import** java.net.Socket;

**public** **class** Ej2Prueba1 {

**3** n **PROGRAMACIÓN DE COMUNICACIONES EN RED**

**public** **static** **void** **main**(String[] args) {   
 **try** {

// Creando socket cliente

Socket clientSocket = **new** Socket(); // Estableciendo la conexión

InetSocketAddress addr = **new** InetSocketAddress(“localhost”, 5555); clientSocket.connect(addr);

InputStream is = clientSocket.getInputStream();

OutputStream os = clientSocket.getOutputStream();

71

**Programación de Servicios y Procesos**

BufferedReader br = **new** BufferedReader(**new** InputStreamReader(is)); PrintWriter pw = **new** PrintWriter(os, **true**);

String pregunta = “¿Cómo te llamas?”;

System.out.println(“Pregunta: “+pregunta); pw.println(pregunta);

// Leyendo la respuesta

String respuesta = br.readLine();

System.out.println(“Respuesta: “+respuesta); // Cerrando el socket cliente

clientSocket.close();

// Terminado

} **catch** (IOException e) {   
 e.printStackTrace(); }

}   
}

Código del ejemplo 2:

**import** java.io.BufferedReader;

**import** java.io.IOException;

**import** java.io.InputStream;

**import** java.io.InputStreamReader; **import** java.io.OutputStream;   
**import** java.io.PrintWriter;   
**import** java.net.InetSocketAddress; **import** java.net.Socket;

**public** **class** Ej2Prueba2 {

**public** **static** **void** **main**(String[] args) {   
 **try** {

// Creando socket cliente

72

**3** n **PROGRAMACIÓN DE COMUNICACIONES EN RED**

Socket clientSocket = **new** Socket(); // Estableciendo la conexión

InetSocketAddress addr = **new** InetSocketAddress(“localhost”, 5555); clientSocket.connect(addr);

InputStream is = clientSocket.getInputStream();

OutputStream os = clientSocket.getOutputStream();

BufferedReader br = **new** BufferedReader(**new** InputStreamReader(is)); PrintWriter pw = **new** PrintWriter(os, **true**);

String pregunta = “¿Cuántos años tienes?”; System.out.println(“Pregunta: “+pregunta); pw.println(pregunta);

// Leuyendo la respuesta

String respuesta = br.readLine();

System.out.println(“Respuesta: “+respuesta); // Cerrando el socket cliente

clientSocket.close();

// Terminado

} **catch** (IOException e) {   
 e.printStackTrace(); }

}   
}

73

**Programación de Servicios y Procesos**

**3.3. Escribe un programa que responda a saludos usando *sockets* *datagram*. El programa escuchará por el *socket* mensajes que contengan la cadena de texto “Hola”. Cuando reciba uno, responderá a su emisor con otro mensaje que contenga la cadena “¿Qué tal?”. Escribe además un programa adicional para probar el funcionamiento de este.**

**Solución**

La solución es muy similar a ejemplos vistos en el libro. Se muestra además un programa de prueba que realiza el saludo y muestra la respuesta obtenida.

Código del programa:

**import** java.io.IOException;

**import** java.net.DatagramPacket;

**import** java.net.DatagramSocket;

**import** java.net.InetSocketAddress;

**public** **class** Ej3 {

**public** **static** **void** **main**(String[] args) {   
 **try** {

// Creando socket datagrama

InetSocketAddress addr = **new** InetSocketAddress(“localhost”, 5555); DatagramSocket datagramSocket = **new** DatagramSocket(addr);

// Recibiendo mensaje

**while** (**true**) {

**byte**[] mensaje = **new** **byte**[4];

DatagramPacket datagrama1 = **new** DatagramPacket(mensaje, 4); datagramSocket.receive(datagrama1);

String saludo = **new** String(mensaje); **if** (saludo.equals(“Hola”)) {   
 // Enviando mensaje

String respuesta = “¿Qué tal?”;

DatagramPacket datagrama2 = **new** DatagramPacket(respuesta.getBytes(), respuesta. getBytes().length, datagrama1.getAddress(), datagrama1.getPort());   
datagramSocket.send(datagrama2);

}

}

74

} **catch** (IOException e) {   
 e.printStackTrace();

}

}

}

Código del programa de prueba:

**import** java.io.IOException;   
**import** java.net.DatagramPacket;   
**import** java.net.DatagramSocket;   
**import** java.net.InetAddress;   
**import** java.net.InetSocketAddress;

**public** **class** Ej3Prueba {

**3** n **PROGRAMACIÓN DE COMUNICACIONES EN RED**

**public** **static** **void** **main**(String[] args) {   
 **try** {

// Creando socket datagrama

InetSocketAddress addr = **new** InetSocketAddress(“localhost”, 5556); DatagramSocket datagramSocket = **new** DatagramSocket(addr);

// Enviando mensaje

String saludo = “Hola”;

InetAddress addr1 = InetAddress.*getByName*(“localhost”);

DatagramPacket datagrama1 = **new** DatagramPacket(saludo.getBytes(), saludo. getBytes().length, addr1, 5555);

datagramSocket.send(datagrama1);

// Recibiendo mensaje

**byte**[] mensaje = **new** **byte**[9];

DatagramPacket datagrama2 = **new** DatagramPacket(mensaje, 9); datagramSocket.receive(datagrama2);

String respuesta = **new** String(mensaje);

System.out.println(“Saludo: “+saludo);

System.out.println(“Respuesta: “+respuesta);

} **catch** (IOException e) {

e.printStackTrace();   
}

}   
}

75

**Programación de Servicios y Procesos**

**3.4. Escribe una pareja de programas (A y B) que usen *sockets* *datagram* para intercambiar un mensaje llamado *token*. Al arrancarse, el programa A enviará un mensaje al B con la palabra “token”. Cuando el B la reciba, enviará de vuelta a A un mensaje con la palabra “recibido”, y terminará. Cuando A reciba el mensaje de B, terminará también.**

**Solución**

Código del programa A:

**import** java.io.IOException;

**import** java.net.DatagramPacket;

**import** java.net.DatagramSocket;

**import** java.net.InetAddress;

**import** java.net.InetSocketAddress;

**public** **class** Ej4A {

**public** **static** **void** **main**(String[] args) {   
 **try** {

// Creando socket datagrama

InetSocketAddress addr = **new** InetSocketAddress(“localhost”, 5556); DatagramSocket datagramSocket = **new** DatagramSocket(addr);

// Enviando token

String token = “token”;

InetAddress addr1 = InetAddress.*getByName*(“localhost”);

DatagramPacket datagrama1 = **new** DatagramPacket(token.getBytes(), token. getBytes().length, addr1, 5555);

datagramSocket.send(datagrama1);

// Recibiendo respuesta

**byte**[] mensaje = **new** **byte**[8];

DatagramPacket datagrama2 = **new** DatagramPacket(mensaje, 8); datagramSocket.receive(datagrama2);

String respuesta = **new** String(mensaje);

**if** (respuesta.equals(“recibido”))

System.out.println(“Respuesta recibida correctamente”);

} **catch** (IOException e) {

e.printStackTrace();   
}

}   
}

76

Código del programa B:

**import** java.io.IOException;   
**import** java.net.DatagramPacket;   
**import** java.net.DatagramSocket;   
**import** java.net.InetSocketAddress;

**public** **class** Ej4B {

**3** n **PROGRAMACIÓN DE COMUNICACIONES EN RED**

**public** **static** **void** **main**(String[] args) {   
 **try** {

// Creando socket datagrama

InetSocketAddress addr = **new** InetSocketAddress(“localhost”, 5555); DatagramSocket datagramSocket = **new** DatagramSocket(addr);

// Recibiendo token

**byte**[] mensaje = **new** **byte**[5];

DatagramPacket datagrama1 = **new** DatagramPacket(mensaje, 5); datagramSocket.receive(datagrama1);

String token = **new** String(mensaje); **if** (token.equals(“token”)) {

System.out.println(“Token recibido. Enviando respuesta”);

String respuesta = “recibido”;

DatagramPacket datagrama2 = **new** DatagramPacket(respuesta.getBytes(),

respuesta.getBytes().length, datagrama1.getAddress(), datagrama1.getPort()); datagramSocket.send(datagrama2);

}

} **catch** (IOException e) {   
 e.printStackTrace(); }

}   
}

77

**Programación de Servicios y Procesos**

**3.5. Escribe un programa que cuente el número de conexiones que vaya recibiendo. Este programa dispondrá de un *socket* *stream* servidor. Cada vez que un *socket* cliente se conecte, este le enviará un mensaje con el número de clientes conectados hasta ahora. Así pues, el primer cliente que se conecte recibirá un 1, el segundo un 2, el tercero un 3, etc.**

**Solución**

Además de la solución, se muestra un programa de prueba. Se puede ejecutar sucesivas veces dicho programa para observar cómo el conteo de conexiones aumenta.

Código del programa:

**import** java.io.IOException;

**import** java.net.InetSocketAddress; **import** java.net.ServerSocket;   
**import** java.net.Socket;

**public** **class** Ej5 {

**public** **static** **void** **main**(String[] args) {   
 **try** {

// Creando socket servidor

ServerSocket serverSocket = **new** ServerSocket();

// Realizando el bind

InetSocketAddress addr = **new** InetSocketAddress(“localhost”, 5555); serverSocket.bind(addr);

// Aceptando conexiones   
**int** num\_conexiones = 0;

**while** (**true**) {

Socket newSocket = serverSocket.accept(); num\_conexiones++;

System.out.println(“Conexión “+num\_conexiones+” recibida”);

// Cerrando el nuevo socket

78

newSocket.close();   
}

} **catch** (IOException e) {   
 e.printStackTrace();

}

}

}

Código del programa de prueba:

**import** java.io.IOException;   
**import** java.net.InetSocketAddress; **import** java.net.Socket;

**public** **class** Ej5Prueba {

**3** n **PROGRAMACIÓN DE COMUNICACIONES EN RED**

**public** **static** **void** **main**(String[] args) {   
 **try** {

// Creando socket cliente

Socket clientSocket = **new** Socket(); // Estableciendo la conexión

InetSocketAddress addr = **new** InetSocketAddress(“localhost”, 5555); clientSocket.connect(addr);

// Cerrando el socket cliente clientSocket.close();

// Terminado

} **catch** (IOException e) {   
 e.printStackTrace(); }

}   
}

79

**Programación de Servicios y Procesos**

**3.6. Crea una versión generalizada de los programas del ejercicio 4, para hacer posible que   
el *token* se pase entre un grupo de 2 o más programas, en forma de anillo. Cada uno de los   
programas se debe arrancar indicando como argumentos de entrada su posición en el anillo   
y el tamaño del anillo. El programa que se encuentre en la posición número 1 (el primero),   
generará el mensaje “token” y se lo enviará al programa 2. Cuando este lo reciba se lo pasará   
al 3, y así sucesivamente. Cuando lo reciba el último programa, lo enviará de vuelta al número**

**1. Cuando el número 1 lo reciba, la secuencia se interrumpirá (el *token* habrá dado una vuelta completa al anillo). Se deben cumplir además las siguientes restricciones:**

n**n Todos los programas deben tener el mismo código fuente. Se trata, por tanto, del mismo**

**programa, pero ejecutado con distintos parámetros.**

n**n El programa debe permitir un número variable de elementos en el anillo. El tamaño del anillo**

**se especificará de antemano.**

**Se puede programar usando *sockets* *stream* o *sockets* *datagram*.**

**Solución**

La solución implementada utiliza *sockets* *datagram*, como la del ejercicio 4. Si se ejecuta por línea de mandatos sin parámetros, muestra un mensaje indicando los argumentos esperados (tamaño del anillo y posición en el anillo) y termina. En el código se distingue el caso de que el programa ejecutado sea el primero del anillo (posición 0) del resto. Si es el primero, debe enviar el *token*, esperar su recepción y terminar. Si no es el primero, le basta con esperar el *token*, reenviarlo y terminar. Los diferentes programas del anillo usan números de puerto consecutivos para poder localizarse.

Código del programa:

**import** java.io.IOException;

**import** java.net.DatagramPacket;

**import** java.net.DatagramSocket;

**import** java.net.InetAddress;

**import** java.net.InetSocketAddress;

**public** **class** Ej6 {

**public** **static** **void** **main**(String[] args) {

**if** (args.length != 2) {

System.out.println(“argumentos: <tamaño del anillo> <posición en el anillo>”); System.*exit*(1);

}

**int** tam\_anillo = Integer.*parseInt*(args[0]);

80

**3** n **PROGRAMACIÓN DE COMUNICACIONES EN RED**

**int** pos\_anillo = Integer.*parseInt*(args[1]);

System.out.println(“Uniéndose a un anillo de tamaño “+tam\_anillo+” en la posición “+pos\_anillo);

**try** {

// Creando socket datagrama

InetSocketAddress addr = **new** InetSocketAddress(“localhost”, 5550+(pos\_anillo % tam\_anillo));

DatagramSocket datagramSocket = **new** DatagramSocket(addr);

// Si se trata del primer elemento, enviando token **if** (pos\_anillo == 0) {

System.out.println(“Generando token”); String token = “token”;

InetAddress addr1 = InetAddress.*getByName*(“localhost”);

DatagramPacket datagrama1 = **new** DatagramPacket(token.getBytes(), token. getBytes().length, addr1, 5551);

datagramSocket.send(datagrama1);   
}

// Recibiendo token

**byte**[] mensaje = **new** **byte**[5];

DatagramPacket datagrama2 = **new** DatagramPacket(mensaje, 5); datagramSocket.receive(datagrama2);

System.out.println(“Token recibido desde “+datagrama2.getAddress()+”, puerto “+datagrama2.getPort());

// Si no se trata del primer elemento, enviando al siguiente **if** (pos\_anillo != 0) {

System.out.println(“Enviando el token al siguiente elemento del anillo”);

String token = “token”;

InetAddress addr1 = InetAddress.*getByName*(“localhost”);

DatagramPacket datagrama3 = **new** DatagramPacket(token.getBytes(), token. getBytes().length, addr1, 5550+((pos\_anillo+1) % tam\_anillo));   
 datagramSocket.send(datagrama3);

}

81

**Programación de Servicios y Procesos**

datagramSocket.close();

System.out.println(“Terminado”);

} **catch** (IOException e) {   
 e.printStackTrace(); }

}   
}

**Solución del test de conocimientos**

**Tabla 3.2** Soluciones del

test de conocimientos del   
capítulo 3

1: c 6: a,b,c

2: d 7: c

3: d 8: c

4: a,b,c 9: b

5: a 10: d

82