3.1. INTRODUCCIÓN

Antiguamente la programación de aplicaciones que comunican diferentes maquinas era difícil, compleja y fuente de muchos errores; el programador tenía que conocer detalles sobre las capas del protocolo de red incluso sobre el hardware de la máquina. Los diseñadores de las librerías Java han hecho que la programación en red para comunicar distintas maquinas no sea una tarea tan compleja.

Java dispone de clases para establecer conexiones, crear servidores, enviar y recibir datos, y para el resto de las operaciones utilizadas en las comunicaciones a través de redes de ordenadores. Además, el uso de hilos, que se trataron en el capítulo anterior, nos va a permitir la manipulación simultanea de múltiples conexiones.

En este capítulo usaremos Java para programar comunicaciones en red.

3.2. CLASES JAVA PARA COMUNICACIONES EN RED

TCP/IP es una familia de protocolos desarrollados para permitir la comunicación entre cualquier par de ordenadores de cualquier red o fabricante, respetando los protocolos de cada red individual. Tiene 4 capas o niveles de abstracción:

■ Capa de aplicación: en este nivel se encuentran las aplicaciones disponibles para los

usuarios. Por ejemplo, FTP, SMTP, Telnet, HTTP, etc.

■ Capa de transporte: suministra a las aplicaciones servicio de comunicaciones

extremo a extremo utilizando dos tipos de protocolos: TCP (Transmission Control Protocol) y UDP (User Datagram Protocol).

■ Capa de red: tiene como proposito seleccionar la mejor ruta para enviar paquetes por

la red. El protocolo principal que funciona en esta capa es el Protocolo de Internet

(IP).

■ Capa de enlace o interfaz de red: es la interfaz con la red real. Recibe los

datagramas de la capa de red y los transmite al hardware de la red.

Los equipos conectados a Internet se comunican entre si utilizando el protocolo TCP o UDP. Cuando se escriben programas Java que se comunican a través de la red, se está programando en la capa de aplicación. Normalmente, no es necesario preocuparse per las capas TCP y UDP; en su lugar, se pueden utilizar las clases del paquete java.net. Sin embargo, existen algunas diferencias entre una y otra que conviene saber para decidir qué clases usar en los programas:

■TCP: Protocolo basado en la conexión, garantiza que los datos enviados desde un

extremo de la conexión llegan al otro extremo y en el mismo orden en que fueron enviados. De lo contrario, se notifica un error.

■UDP: No está basado en la conexión como TCP. Envía paquetes de datos independientes, denominados datagramas, de una aplicación a otra; el orden de entrega no es importante y no se garantiza la recepción de los paquetes enviados.

El paquete java.net contiene clases e interfaces para la implementación de aplicaciones de red. Estas incluyen:

■La clase URL, Uniform Resource Locator (Localizador Uniforme de Recursos). Representa un puntero a un recurso en la Web.

■La clase URLConnection, que admite operaciones más complejas en las URL.

■Las clases ServerSocket y Socket, para dar soporte a sockets TCP.

ServerSocket: Utilizada por el programa servidor para crear un socket en el puerto en el que escucha las peticiones de conexion de los clientes.

Socket: utilizada tanto por el cliente como por el servidor para comunicarse entre si leyendo y escribiendo datos usando streams.

■ Las clases DatagramSocket, MulticastSocket y DatagramPacket para dar soporte

a la comunicacion via datagramas UDP.

■ La clase InetAddress, que representa las direcciones de Internet.

3.2.1. LOS PUERTOS

Los protocolos TCP y UDP usan puertos para asignar datos entrantes a un proceso en particular que se ejecuta en un ordenador.

En términos generales, un ordenador tiene una única conexión física a la red. Los datos destinados a este ordenador llegan a través de esa conexión. Sin embargo, los datos pueden estar destinados a diferentes aplicaciones que se ejecutan en el ordenador. Entonces, ¿cómo sabe el ordenador a que aplicación enviar los datos? Mediante el uso de puertos.

Los datos transmitidos a través de Internet van acompañados de información de direccionamiento que identifica la maquina y el puerto para el que está destinada. La máquina se identifica por su dirección IP de 32 bits, para entregar datos a una maquina concreta necesitaremos conocer su IP. Los puertos se identifican mediante un número de 16 bits, que TCP y UDP utilizan para entregar los datos a la aplicación correcta.

En la comunicación basada en TCP, una aplicación de servidor vincula un socket a un número de puerto especifico. Esto tiene el efecto de registrar el servidor en el sistema para recibir todos los datos destinados a ese puerto. Una aplicación cliente puede entonces comunicarse con el servidor enviándole peticiones a troves de ese puerto.

En la comunicación basada en datagramas, como UDP, el paquete de datagramas contiene el número de puerto de su destino y UDP enruta el paquete a la aplicación adecuada.

3.2.2. LA CLASE InetAddress

La clase InetAddress es la abstracción que representa una dirección IP (Internet Protocol). Tiene dos subclases: Inet4Address para direcciones IPv4 e Inet6Address para direcciones IPv6; pero en la mayoría de los casos InetAddress aporta la funcionalidad necesaria y no es necesario recurrir a ellas.

En la siguiente tabla se muestran algunos métodos importantes de esta clase:

MÉTODOS	MISIÓN
InetAddress getLocalHost()	Devuelve un objeto *InetAddress* que representa la dirección IP de la máquina donde se está ejecutando el programa.
InetAddress getByName(String host)	Devuelve un objeto *InetAddress* que representa la dirección IP de la máquina que se especifica como parámetro (*host*). Este parámetro puede ser el nombre de la máquina, un nombre de dominio o una dirección IP.
InetAddress[] getAllByName(String host)	Devuelve un array de objetos de tipo *InetAddress*. Este método es útil para averiguar todas las direcciones IP que tenga asignada una máquina en particular.
String getHostAddress()	Devuelve la dirección IP de un objeto *InetAddress* en forma de cadena.
String getHostName()	Devuelve el nombre del host de un objeto *InetAddress*.
String getCanonicalHostName()	Obtiene el nombre canónico completo (suele ser la dirección real del host) de un objeto *InetAddress*.

Los 3 primeros métodos pueden lanzar la excepción UnknownHostException. La forma típica de crear instancias de InetAddress, es invocando al método estático getByName(String) pasándole el nombre DNS del host como parámetro. Este objeto representará la dirección IP de ese host, y se podrá utilizar para construir sockets

En el siguiente ejemplo se define un objeto InetAddress de nombre dir. En primer lugar lo utilizamos para obtener la dirección IP de la maquina local en la que se ejecuta el programa, en el ejemplo su nombre es localhost. A continuación llamarnos al método pruebaMetodos() llevando el objeto creado. En dicho método se prueban los métodos de la clase InetAddress. Después utilizamos el objeto para obtener la dirección IP de la URL www.google.es y volvemos a invocar a pruebaMetodos() (para que funcione en este segundo caso necesitamos estar conectados a Internet). Por último utilizamos el método getAllByName() para ver todas las direcciones IP asignadas a la maquina representada por www.google.es. Se encierra todo en un bloque try-catch:

```
import java.net.*;
public class TestInetAddress{
public static void main(String[] args{
            InetAddress dir = null;
  System.out.printin("=======================");
  System.out.println("SALIDA PARA LOCALHOST: ");
  try      {
   //LOCALHOST
   dir = InetAddress.getByName("localhost");
   pruebaMetodos(dir);//

//URLwww.google.es
    System.out.println("====================");
     System.out.println("SALIDA PARA UNA URL:");
     dir = InetAddress.getByName("www.google.es");
     pruebaMetodos(dir);

//Array de tipo InetAddress con todas las direcciones IP
//asignadas a google.es
```

```
            System.out.println("\tDIRECCIONES IP PARA: " + dir.getHostName());
             InetAddress[] direcciones =
            InetAddress.getAllByName(dir.getHostName());
          for(int i = 0;i < direcciones.length; i++)
                System.out.println("\t\t"+direcciones[i].toString());
          System.out.println("==================");
          }catch(UnknownHostException el){el.printStackIrace();}

        }// main

        private static void pruebaMetodos(InetAddress dir){
            System.out.println("\tMetodo getByName(): " + dir);
             InetAddress dir2;
            try{
                 dir2 = InetAddress.getLocalHost();
                System.out.println("\tMetodo getLocalHost(): " + dir2);
             }catch (UnknownHostException e) {e.printStackTrace();}

           //USAMOS MÉTODOS DE LA CLASS
        System.out.println("\tMetodo getHostName(): "+dir.getHostName()); System.out.println("\tMetodo
getHostAddress(): "+
        dir.getHostAddress());
        System.out.println("\tMetodo toString():"+ dir.toString()); System.out.println("\tMetodo
getCanonicalHostName(): " +
        dir.getCanonicalHostName());

        }//pruebaMetodos

        }//Fin
```

La salida generada es la siguiente:

```
============================

        SALIDA PARA LOCALHOST:
        Metodo getByName(): localhost/127.0.0.1
        Metodo getLocalHost(): PC-ASUS/192.168.56.1 Metodo getHostName(): localhost
        Metodo getHostAddress(): 127.0.0.1
        Metodo toStringO: localhost/127.0.0.1
        Metodo getCanonicalHostName(): 127.0.0.1
        ============================
        SALIDA PARA UNA URL:
        Metodo getByName(): www.google.es/216.58.210.131
        Metodo getLocalHost(): PC-ASUS/192.168.56.1
        Metodo getHostName(): www.google.es
        Metodo getHostAddress(): 216.58.210.131
        Metodo toString(): www.google.es/216.58.210.131
        Metodo getCanonicalHostName(): madO6s09-in-f3.1e100.net
        DIRECCIONES IP PARA: www.google.es
        www.google.es/216.58.210.131
        ============================
```

ACTIVIDAD 3.1

Realiza un programa Java que admita desde la línea de comandos un nombre de maquina o una dirección IP y visualice información sobre ella.