UNIDAD 3

PROGRAMACIÓN DE COMUNICACIONES EN RED

Java dispone de clases especializadas en crear conexiones, servidores, envío y recepción de datos. Gracias a los hilos podremos realizar conexiones múltiples de forma simultánea.

Los sockets son un mecanismo que nos permite establecer un enlace entre dos programas que se ejecutan independientes el uno del otro (generalmente un programa cliente y un programa servidor) Java por medio de la librería java.net nos provee dos clases: Socket para implementar la conexión desde el lado del cliente y ServerSocket que nos permitirá manipular la conexión desde el lado del servidor.

Los sockets de Internet constituyen el mecanismo para la entrega de paquetes de datos provenientes de la tarjeta de red a los procesos o hilos apropiados. Un socket queda definido por un par de direcciones IP local y remota, un protocolo de transporte y un par de números de puerto local y remoto.

En las aplicaciones en red es muy común el paradigma cliente-servidor. El servidor es el que espera las conexiones del cliente y el cliente es el que lanza las peticiones a la máquina donde se está ejecutando el servidor, y al lugar donde está esperando el servidor (el puerto(s) específico que atiende). Una vez establecida la conexión, ésta es tratada como un stream (flujo) típico de entrada/salida.

Cuando se escriben programas Java que se comunican a través de la red, se está programando en la capa de aplicación y en su lugar se puede utilizar las clases del paquete java.net. Estas clases proporcionan comunicación de red independiente del sistema.

Las clases URL, URLConnection, Socket y SocketServer utilizan TCP para comunicarse a través de la red.

Las clases DatagramPacket y DatagramServer utilizan UDP.

Las clases Socket y ServerSocket del paquete java.net proporcionan un canal de comunicación independiente del sistema utilizando TCP, cada una de las cuales implementa el lado del cliente y el servidor respectivamente. TCP proporciona un canal de comunicación fiable punto a punto

Socket: Implementa un extremo cliente de la conexión TCP.
ServerSocket: Se encarga de implementar el extremo Servidor de la conexión en la que se esperarán las conexiones de los clientes.
DatagramSocket: Implementa tanto el servidor como el cliente cuando se utiliza UDP.
DatagramPacket: Implementa un datagram packet, que se utiliza para la creación de servicios de reparto de paquetes sin conexión
InetAddress: Se encarga de implementar la dirección IP.

El entorno de desarrollo de Java incluye un paquete, java.io, que contiene un juego de canales de entrada y salida que los programas pueden utilizar para leer y escribir datos. Las clases InputStream y OutputStream del paquete java.io son superclases abstractas que definen el comportamiento de los canales de I/O de tipo stream de Java. java.io también incluye muchas subclases de InputStream y OtputStream que implementan tipos especificos de canales de I/O.

El esquema básico en sockets TCP sería construir un único servidor con la clase ServerSocket e invocar al método accept() para esperar las peticiones de conexión de los clientes. Cuando un cliente se conecta, el método accept() devuelve un objeto Socket, éste se usará para crear un hilo cuya misión es atender a este cliente. Después se vuelve a invocar a accept() para esperar a un nuevo cliente; habitualmente la espera de conexiones se hace dentro de un bucle infinito

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

public class Servidor {
	public static void main(String args[]) throws IOException
		ServerSocket servidor;
		servidor = new ServerSocket(6000);
		System.out.println("Servidor iniciado ... ");
		while (true) {
			Socket cliente = new Socket();
			cliente = servidor.accept();//esperando cliente
			HiloServidor hilo = new HiloServidor(cliente);
			hilo.start(); //se atiende al cliente
		}
	}
}
```

CLASE InetAddress

Es la clase que nos representa una dirección IP. La forma más típica de crear instancias de InetAddress, es invocando al método 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.

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*.

```
import java.net.*;

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

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

      } catch (UnknownHostException e1) {e1.printStackTrace();}
    }// main

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

        // USAMOS METODOS DE LA CLASE
        System.out.println("\tMetodo getHostName(): "+dir.getHostName());
        System.out.println("\tMetodo getHostAddress(): "+
                                                dir.getHostAddress());
        System.out.println("\tMetodo getCanonicalHostName(): " +
                                            dir.getCanonicalHostName());
        }//pruebaMetodos

}//fin
```

CLASE URL

Representa un puntero a un recurso en la web. Un recurso puede ser algo tan simple como un fichero o un directorio, o puede ser una referencia a un objeto más complicado, como una consulta a una base de datos o a un motor de búsqueda.

CONSTRUCTOR	MISIÓN
URL (String url)	Crea un objeto URL a partir del String indicado en *url*.
URL(String protocolo, String host, String fichero)	Crea un objeto URL a partir de los parámetros *protocolo, host* y *fichero* (o directorio).
URL(String protocolo, String host, int puerto, String fichero)	Crea un objeto URL en el que se especifica el *protocolo*, *host*, *puerto* y *fichero* (o directorio) representados mediante String.

MÉTODOS	MISIÓN
String getAuthority ()	Obtiene la autoridad del objeto URL.
int getDefaultPort()	Devuelve el puerto asociado por defecto al objeto URL.
int getPort()	Devuelve el número de puerto de la URL, -1 si no se indica.
String getHost()	Devuelve el nombre de la máquina.
String getQuery()	Devuelve la cadena que se envía a una página para ser procesada (es lo que sigue al signo? de una URL).
String getPath()	Devuelve una cadena con la ruta hacia el fichero desde el servidor y el nombre completo del fichero.
String getFile()	Devuelve lo mismo que *getPath ()*, además de la concatenación del valor de *getQuery()* si lo hubiese. Si no hay una porción consulta, este método y *getPath()* devolverán los mismos resultados.
String getProtocol()	Devuelve el nombre del protocol asociado al objeto URL.
String getUserInfo()	Devuelve la parte con los datos del usuario o nulo si no existe.
InputStream openStream()	Devuelve un **InputStream** del que podremos leer el contenido del recurso que identifica la URL.
URLConnection openConnection()	Devuelve un objeto **URLConnection** que nos permite abrir una conexión con el recurso y realizar operaciones de lectura y escritura sobre él.

Ejemplo de uso de varios métodos de la clase URL

```
import java.net.*;
public class Ejemplo1URL {
  public static void main(String[] args) {
    URL url;
    try {
      System.out.println("Constructor simple para una URL:");
      url = new URL("http://docs.oracle.com/");
      Visualizar(url);

      System.out.println("Otro constructor simple para una URL:");
      url = new URL("http://localhost/PFC/gest/cli_gestion.php?S=3");
        Visualizar(url);

      System.out.println("Const. para protocolo +URL + directorio:");
      url = new URL("http", "docs.oracle.com", "/javase/9");
      Visualizar(url);

      System.out.println("Constructor para protocolo + URL + puerto + directorio:");
        url = new URL("http", "localhost", 8084,
                      "/WebApp/Controlador?accion=modificar");
        Visualizar(url);

      System.out.println("Constructor para un objeto URL en un contexto:");
      URL urlBase = new URL("https://docs.oracle.com/");
      url = new URL(urlBase, "/javase/9/docs/api/java/net/URL.html");
      Visualizar(url);

    } catch (MalformedURLException e) { System.out.println(e);}
  }// main

  private static void Visualizar(URL url) {
    System.out.println("\tURL completa: " + url.toString());
    System.out.println("\tgetProtocol(): " + url.getProtocol());
    System.out.println("\tgetHost(): " + url.getHost());
    System.out.println("\tgetPort(): " + url.getPort());
    System.out.println("\tgetFile(): " + url.getFile());
    System.out.println("\tgetUserInfo(): " + url.getUserInfo());
    System.out.println("\tgetPath(): " + url.getPath());
    System.out.println("\tgetAuthority(): " + url.getAuthority());
    System.out.println("\tgetQuery(): " + url.getQuery());
    System.out.println("\tgetDefaultPort(): "+ url.getDefaultPort());
    System.out
       .println("=================================================");
  }
}
```

CLASE URLConnection

Una vez que tenemos un objeto de la clase URL, si se invoca al método openConnection() para realizar la comunicación con el objeto y la conexión se establece satisfactoriamente, entonces tenemos una instancia de un objeto de la clase URLConnection:

```
URL url = new URL("http://www.elaltozano.es");
URLConnection urlCon= url.openConnection();
```

La clase URLConnection contiene métodos que permiten la comunicación entre la aplicación y una URL.

MÉTODOS	MISIÓN
InputStream getInputStream()	Devuelve un objeto **InputStream** para leer datos de esta conexión.
OutputStream getOutputStream()	Devuelve un objeto **OutputStream** para escribir datos en esta conexión.
void setDoInput (boolean b)	Permite que el usuario reciba datos desde la URL si el parámetro *b* es *true* (por defecto está establecidoa *true*)
void setDoOutput((boolean b)	Permite que el usuario envíe datos si el parámetro *b* es *true* (no está establecido al principio)
void connect()	Abre una conexión al recurso remoto si tal conexión no se ha establecido ya.
URL getURL()	Devuelve la dirección URL.

SOCKETS

Los protocolos TCP y UDP utilizan el concepto de sockets para proporcionar los puntos extremos de la comunicación entre aplicaciones o procesos. La comunicación entre procesos consiste en la transmisión de un mensaje entre un conector de un proceso y un conector de otro proceso, a este conector es a lo que llamamos socket.

Para los procesos receptores de mensajes, su conector debe tener asociado dos campos:

- La dirección IP del host en el que la aplicación está corriendo.
- El puerto local a través del cual la aplicación se comunica.

Hay dos tipos básicos de sockets en redes IP:

- Los que utilizan el protocolo TCP:
  Orientados a conexión. se garantiza la entrega de los paquetes de datos y el orden en que fueron enviados
- Los que utilizan el protocolo UDP:
  No orientados a conexión. No es fiable y no se garantiza que la información enviada llegue a su destino. Los sockets UDP se usan cuando una entrega rápida es más importante que una entrega garantizada.

SOCKETS TCP

Clase **ServerSocket**:

La clase ServerSocket se utiliza para implementar el extremo de la conexión que corresponde al servidor

CONSTRUCTOR	MISIÓN
ServerSocket()	Crea un socket de servidor sin ningún puerto asociado.
ServerSocket(int port)	Crea un socket de servidor, que se enlaza al puerto especificado.
ServerSocket(int port, int máximo)	Crea un socket de servidor y lo enlaza con el número de puerto local especificado. El parámetro *máximo* especifica, el número máximo de peticiones de conexión que se pueden mantener en cola.

MÉTODOS	MISIÓN
Socket accept ()	El método **accept()** escucha una solicitud de conexión de un cliente y la acepta cuando se recibe. Una vez que se ha establecido la conexión con el cliente, devuelve un objeto de tipo **Socket**, a través del cual se establecerá la comunicación con el cliente. Tras esto, el **ServerSocket** sigue disponible para realizar nuevos ***accept()***. Puede lanzar ***IOException***.
close ()	Se encarga de cerrar el **ServerSocket**.
int getLocalPort ()	Devuelve el puerto local al que está enlazado el **ServerSocket**.

Clase **Socket**:

La clase Socket implementa el extremo cliente de la conexión TCP

CONSTRUCTOR	MISIÓN
Socket()	Crea un socket sin ningún puerto asociado.
Socket (InetAddress address, int port)	Crea un socket y lo conecta al puerto y dirección IP especificados.
Socket (String host, int port)	Crea un socket y lo conecta al número de puerto y al nombre de host especificados. Puede lanzar ***UnKnownHostException, IOException***

MÉTODOS	MISIÓN
InputStream getInputStream ()	Devuelve un **InputStream** que permite leer bytes desde el socket utilizando los mecanismos de streams, el socket debe estar conectado. Puede lanzar *IOException*.
OutputStream getOutputStream ()	Devuelve un **OutputStream** que permite escribir bytes sobre el socket utilizando los mecanismos de streams, el socket debe estar conectado. Puede lanzar ***IOException***.
close ()	Se encarga de cerrar el socket.
int getLocalPort ()	Devuelve el puerto local al que está enlazado el socket, -1 si no está enlazado a ningún puerto.
int getPort ()	Devuelve el puerto remoto al que está conectado el socket, 0 si no está conectado a ningún puerto.

Clase **ObjectInputStream / ObjectOutputStream**:

Nos permiten enviar objetos a través de sockets TCP

```java
import java.io.IOException;

public class TCPServidorBasico {

    public static void main(String[] args) throws IOException {

        int Puerto = 6000;// Puerto
        ServerSocket Servidor = new ServerSocket(Puerto);

        System.out.println("Escuchando en " + Servidor.getLocalPort());
        Socket cliente1= Servidor.accept();//esperando a un cliente
        //realizar acciones con cliente1

        Socket cliente2 = Servidor.accept();//esperando a otro cliente
        //realizar acciones con cliente2

        Servidor.close();    //cierro socket servidor
    }

}
```

```java
import java.io.IOException;

public class TCPClienteBasico {

    public static void main(String[] args) throws UnknownHostException, IOException {
        String Host = "localhost";
        int Puerto = 6000;//puerto remoto

        // ABRIR SOCKET
        Socket Cliente = new Socket(Host, Puerto);//conecta

        System.out.println ("Puerto local: "+ Cliente.getLocalPort());
        System.out.println ("Puerto Remoto: "+ Cliente.getPort());
        System.out.println ("Nombre Host/IP: "+ Cliente.getInetAddress());
        Cliente.close();// Cierra el socket
    }
}
```