UD4 - 02. Librerías de Clases y Componentes para Servicios en Red

Introducción

Las librerías de clases y componentes en Java permiten simplificar la creación de aplicaciones que se comunican a través de la red. Estas librerías ofrecen clases predefinidas para manejar conexiones, enviar y recibir información, gestionar protocolos y garantizar la seguridad de las comunicaciones.

Java incluye un potente conjunto de librerías para redes dentro del paquete `java.net`, además de librerías adicionales para tareas avanzadas, como el acceso a APIs REST, gestión de hilos en servidores, etc.

Clases Principales de `java.net`

1. **InetAddress**
    - Permite obtener información sobre direcciones IP y nombres de dominio.
    - **Principales métodos**:
        - `getByName(String host)`: Obtiene la dirección IP de un host.
        - `getLocalHost()`: Obtiene la dirección IP de la máquina local.

Ejemplo:

```
import java.net.InetAddress;

public class InetAddressExample {
    public static void main(String[] args) {
        try {
            InetAddress local = InetAddress.getLocalHost();
            InetAddress google =
InetAddress.getByName("www.google.com");

            System.out.println("Dirección local: " +
local.getHostAddress());
            System.out.println("Dirección de Google: " +
google.getHostAddress());
```

```
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

        ○

2. **Socket** (Cliente)
    - Permite crear conexiones orientadas a conexión utilizando el protocolo TCP.
    - **Principales métodos**:
        - `getInputStream()`: Obtiene un flujo de entrada desde el servidor.
        - `getOutputStream()`: Obtiene un flujo de salida hacia el servidor.

Ejemplo de cliente:

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

public class ClienteTCP {
    public static void main(String[] args) {
        try (Socket socket = new Socket("localhost", 5000);
             BufferedWriter writer = new BufferedWriter(new
OutputStreamWriter(socket.getOutputStream()));
             BufferedReader reader = new BufferedReader(new
InputStreamReader(socket.getInputStream()))) {

            writer.write("¡Hola, servidor!\n");
            writer.flush();

            String respuesta = reader.readLine();
            System.out.println("Respuesta del servidor: " +
respuesta);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

3. **ServerSocket** (Servidor)
    - Escucha conexiones entrantes de clientes TCP.
    - **Principales métodos**:
        - `accept()`: Acepta una conexión entrante y devuelve un `Socket` para comunicarse con el cliente.

Ejemplo de servidor:

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

public class ServidorTCP {
    public static void main(String[] args) {
        try (ServerSocket serverSocket = new ServerSocket(5000)) {
            System.out.println("Servidor escuchando en el puerto
5000...");
            while (true) {
                Socket cliente = serverSocket.accept();
                BufferedReader reader = new BufferedReader(new
InputStreamReader(cliente.getInputStream()));
                BufferedWriter writer = new BufferedWriter(new
OutputStreamWriter(cliente.getOutputStream()));

                String mensaje = reader.readLine();
                System.out.println("Cliente dice: " + mensaje);

                writer.write("¡Mensaje recibido!\n");
                writer.flush();
                cliente.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

4. **DatagramSocket** (UDP)
    - Permite enviar y recibir datagramas sin conexión utilizando el protocolo UDP.
    - **Principales métodos**:

- `send(DatagramPacket packet)`: Envía un paquete UDP.
- `receive(DatagramPacket packet)`: Recibe un paquete UDP.

Ejemplo de servidor UDP:

```
import java.net.DatagramPacket;
import java.net.DatagramSocket;

public class ServidorUDP {
    public static void main(String[] args) {
        try (DatagramSocket socket = new DatagramSocket(5001)) {
            byte[] buffer = new byte[1024];
            DatagramPacket packet = new DatagramPacket(buffer,
buffer.length);

            System.out.println("Servidor UDP esperando
mensajes...");
            socket.receive(packet);

            String mensaje = new String(packet.getData(), 0,
packet.getLength());
            System.out.println("Mensaje recibido: " + mensaje);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

-

Otras Librerías y Componentes Importantes

1. **JavaMail (SMTP, POP3, IMAP)**:
   - Librería que permite enviar y recibir correos electrónicos.

Ejemplo de uso básico con SMTP:

```
import javax.mail.*;
import javax.mail.internet.*;
import java.util.Properties;
```

```
public class EnviarCorreo {
    public static void main(String[] args) {
        String host = "smtp.ejemplo.com";
        String remitente = "usuario@ejemplo.com";
        String destinatario = "destinatario@ejemplo.com";

        Properties props = new Properties();
        props.put("mail.smtp.auth", "true");
        props.put("mail.smtp.starttls.enable", "true");
        props.put("mail.smtp.host", host);
        props.put("mail.smtp.port", "587");

        Session session = Session.getInstance(props, new
Authenticator() {
            protected PasswordAuthentication
getPasswordAuthentication() {
                return new PasswordAuthentication("usuario",
"contraseña");
            }
        });

        try {
            Message mensaje = new MimeMessage(session);
            mensaje.setFrom(new InternetAddress(remitente));
            mensaje.setRecipients(Message.RecipientType.TO,
InternetAddress.parse(destinatario));
            mensaje.setSubject("Prueba de correo");
            mensaje.setText("Este es un mensaje de prueba.");

            Transport.send(mensaje);
            System.out.println("Correo enviado con éxito.");
        } catch (MessagingException e) {
            e.printStackTrace();
        }
    }
}
```

        ○

2. **Librerías para Servicios REST (Spring Boot)**:

- Spring Boot permite crear servicios REST fácilmente.

Ejemplo de servicio REST simple:

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication
public class RestApplication {
    public static void main(String[] args) {
        SpringApplication.run(RestApplication.class, args);
    }
}

@RestController
@RequestMapping("/api")
class SaludoController {
    @GetMapping("/saludo")
    public String saludo() {
        return "¡Hola desde el servicio REST!";
    }
}
```

Conclusión

Las librerías de clases y componentes son esenciales para desarrollar aplicaciones de red eficientes y seguras en Java. Clases como `Socket`, `ServerSocket` y `DatagramSocket` permiten gestionar conexiones y transmisiones de datos en protocolos TCP y UDP. Además, librerías avanzadas como `JavaMail` y `Spring Boot` permiten desarrollar aplicaciones más complejas, como servicios de correo y APIs REST.