UD3 - 04. Elementos para la programación de aplicaciones en red

En esta unidad desarrollaremos de manera didáctica y estructurada el tema de los elementos fundamentales para la programación de aplicaciones en red. Este contenido es esencial para comprender cómo funcionan las aplicaciones que se comunican mediante redes, desde el nivel de código hasta la interacción con las API y librerías específicas.

---

1. Introducción

La programación de aplicaciones en red requiere comprender y manejar diversos elementos que facilitan la comunicación entre dispositivos. Estos elementos son herramientas y conceptos básicos que permiten a los desarrolladores implementar soluciones eficientes, seguras y escalables.

---

2. Elementos clave

1. **Librerías y clases**
    - Funciones probadas específicas para la programación en red.
    - Permiten trabajar con sockets, protocolos de red y otros componentes fundamentales.

Ejemplo en Java:

```
import java.net.Socket; // Clase para crear conexiones cliente.
import java.net.ServerSocket; // Clase para gestionar un servidor.
```

    -
    - Librerías populares:
        - `java.net`: Para trabajar con enchufes y protocolos.
        - `java.io`: Para gestionar flujos de entrada y salida.
2. **API (Interfaces de Programación de Aplicaciones)**
    - Defina cómo interactuar con librerías y servicios en red.
    - Ejemplo:
        - La API de Sockets en Java proporciona métodos para crear conexiones cliente/servidor.
        - La API `JavaMail`permite trabajar con el envío de correos electrónicos.
3. **Zócalos**
    - Son puntos de comunicación entre dos nodos en una red.
    - Tipos principales:
        - **Socket orientado a conexión (TCP)** :
            - Garantiza entrega de datos.

        - Ejemplo: Transferencia de archivos.
    - **Socket no orientado a conexión (UDP)** :
        - No garantiza la entrega ni el pedido.
        - Ejemplo: Streaming en tiempo real.

Clases relacionadas:

```
ServerSocket servidor = new ServerSocket(8080); // Crear socket
servidor.
Socket cliente = new Socket("127.0.0.1", 8080); // Crear socket
cliente.
```

    -

4. **Flujos de entrada y salida (Streams)**
    - Utilizados para enviar y recibir datos a través de sockets.
    - Ejemplo:
        - `InputStream`y `OutputStream`en Java para leer y escribir datos.

```
OutputStream salida = cliente.getOutputStream();
salida.write("Hola, servidor".getBytes());
```

5. **Protocolo de comunicación**
    - Defina las reglas de intercambio de datos entre cliente y servidor.
    - Protocolos comunes:
        - TCP/IP: Confiable, orientado a conexión.
        - UDP: Rápido, no confiable.
        - HTTP/HTTPS: Para aplicaciones web.
    - Ejemplo:
        - Configuración de sockets para usar TCP.

```
ServerSocket servidor = new ServerSocket(8080);
Socket cliente = servidor.accept();
```

    -

---

3. Ejemplo práctico: Uso de elementos básicos

Fundamentos de Java

Este servidor escucha en el puerto 8080 y responde a mensajes de los clientes.

```
import java.net.ServerSocket;
import java.net.Socket;
import java.io.InputStream;
import java.io.OutputStream;

public class ServidorBasico {
    public static void main(String[] args) throws Exception {
        ServerSocket servidor = new ServerSocket(8080);
        System.out.println("Servidor en espera de conexiones...");

        Socket cliente = servidor.accept(); // Aceptar conexión.
        System.out.println("Cliente conectado.");

        InputStream entrada = cliente.getInputStream();
        byte[] buffer = new byte[1024];
        int bytesLeidos = entrada.read(buffer);
        System.out.println("Mensaje recibido: " + new String(buffer,
0, bytesLeidos));

        OutputStream salida = cliente.getOutputStream();
        salida.write("Respuesta desde el servidor".getBytes());

        cliente.close();
        servidor.close();
    }
}
```

Cliente básico en Java

Este cliente se conecta al servidor, envía un mensaje y recibe una respuesta.

```
import java.net.Socket;
import java.io.InputStream;
import java.io.OutputStream;

public class ClienteBasico {
    public static void main(String[] args) throws Exception {
        Socket cliente = new Socket("127.0.0.1", 8080);

        OutputStream salida = cliente.getOutputStream();
```

```
        salida.write("Hola, servidor".getBytes());

        InputStream entrada = cliente.getInputStream();
        byte[] buffer = new byte[1024];
        int bytesLeidos = entrada.read(buffer);
        System.out.println("Respuesta del servidor: " + new
String(buffer, 0, bytesLeidos));

        cliente.close();
    }
}
```

---

4. Aplicaciones prácticas

- **Chat en tiempo real:** Utilizando sockets y programación multihilo.
- **Transferencia de archivos:** Usando sockets orientados a conexión.
- **Sistema cliente-servidor en red local:** Para gestionar bases de datos remotas.

---

5. Conclusión

Dominar los elementos para la programación de aplicaciones en red es crucial para desarrollar soluciones robustas. Las herramientas como las librerías específicas, los sockets y las API facilitan la implementación de sistemas eficientes que responden a las demandas de comunicación modernas.