PROGRAMACION DE SERVICIOS

TAREA I UD3

ALUMNO CESUR 25/26

Alejandro Muñoz de la Sierra

PROFESOR

Santiago Martin-palomo Garcia

I N T R O D U C C I O N

Este caso práctico plantea una situación muy cercana a un entorno real. Desarrollamos un sistema de alarmas capaz de enviar avisos desde viviendas particulares hasta una central receptora. Las empresas de seguridad utilizan estos sistemas habitualmente. Muchos dispositivos clientes deben comunicarse de forma rápida y continua con un servidor central para gestionar las alertas.

Nuestro contacto previo con la programación en red era bastante limitado. Tuvimos que familiarizarnos con conceptos nuevos antes de comenzar el desarrollo. Estudiamos la comunicación entre equipos, los protocolos de red y el funcionamiento básico de los sockets en Java.

El objetivo principal del ejercicio es simular este sistema de alertas mediante comunicación UDP. Diferenciamos varios tipos de mensajes según su importancia. También comprobamos que el servidor recibe, interpreta y muestra los datos correctamente.

02

INVESTIGACIÓN PREVIA Y ELECCIÓN DE LA TECNOLOGÍA

Realizamos una fase de investigación sobre los conceptos clave de la unidad antes de escribir código. Nos centramos en comprender las aplicaciones distribuidas. Analizamos cómo se comunican dos programas a través de una red. Vimos las diferencias entre los protocolos TCP y UDP. También estudiamos el papel del paquete java.net en el lenguaje Java.

Revisamos la documentación de la unidad y fuentes complementarias. Entendimos que UDP es un protocolo no orientado a la conexión. No se establece una comunicación previa entre cliente y servidor. Los mensajes se envían directamente y confiamos en que lleguen a su destino.

TCP ofrece mayor fiabilidad, pero entendimos que UDP es más adecuado para sistemas de alertas. Es más rápido y consume menos recursos. No bloquea el sistema esperando confirmaciones. Permite enviar mensajes cortos y frecuentes.

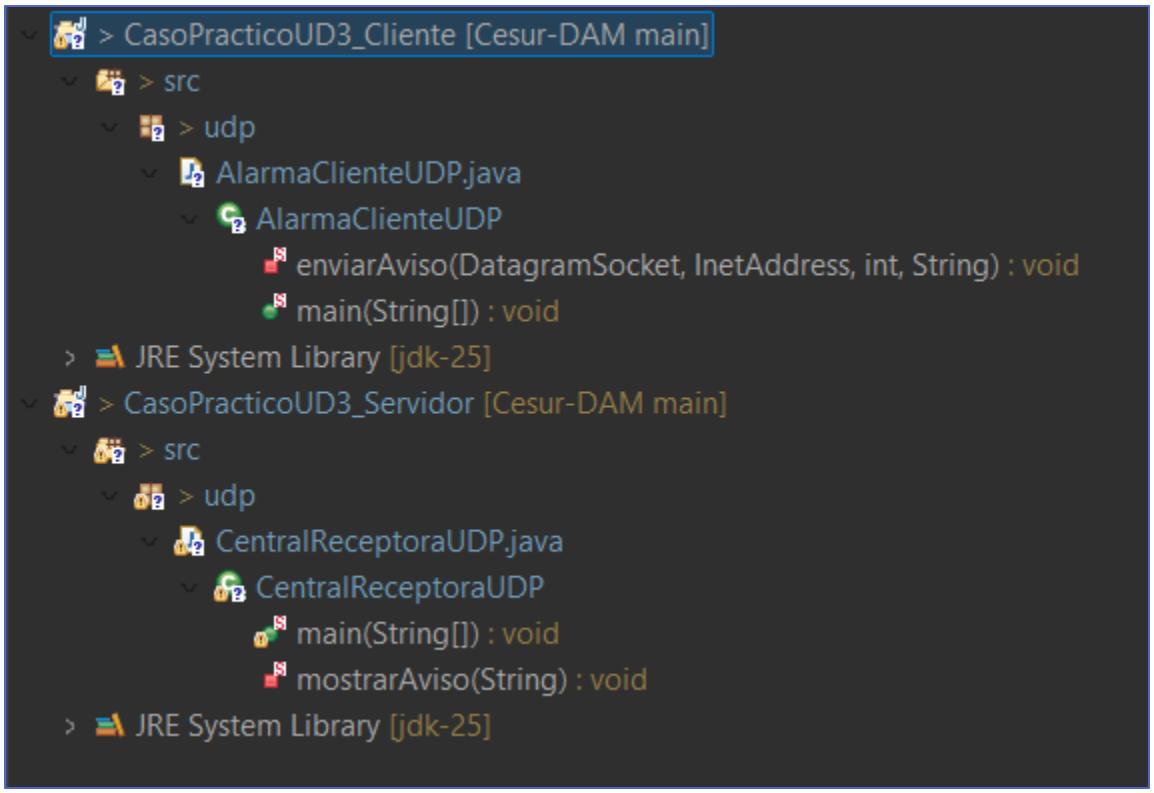
Decidimos implementar el sistema utilizando sockets UDP por estos motivos y por las indicaciones del enunciado.

0 3
D I S E Ñ O D E L A S O L U C I Ó N

Dividimos la solución en dos proyectos Java independientes para abordar el problema con orden. El proyecto cliente simula un dispositivo de alarma en una vivienda que envía avisos. El proyecto servidor simula el sistema central que recibe los mensajes y los muestra por pantalla.

Esta separación refleja el funcionamiento de un sistema real. También facilita tanto el desarrollo como las pruebas del sistema.

Trabajamos con Eclipse y creamos las clases **main** de cada proyecto dentro del **paquete** personalizado **udp**, y añadimos la librería **jdk25** al classpath con la que trabajamos usualmente.



04

DESARROLLO DEL PROYECTO CLIENTE (SIMULACIÓN DE ALARMA)

AlarmaClienteUDP.java × CentralReceptoraUDP.java

```java
package udp;

import java.net.DatagramPacket;

public class AlarmaClienteUDP {

    public static void main(String[] args) {

        try {
            // Dirección IP del servidor.
            // En este caso usamos "localhost" porque estamos probando
            // el cliente y el servidor en el mismo ordenador.
            InetAddress direccionCentral = InetAddress.getByName("localhost");

            // Puerto por el que la central está escuchando los avisos
            int puertoCentral = 5000;

            // Creamos el socket UDP.
            // No se establece ninguna conexión previa, simplemente se crea
            // el canal por el que se enviarán los mensajes.
            DatagramSocket socket = new DatagramSocket();

            // Envío de un mensaje informativo
            enviarAviso(socket, direccionCentral, puertoCentral,
                    "INFO: Sistema de alarma activado correctamente en la vivienda");

            // Envío de un aviso de advertencia
            enviarAviso(socket, direccionCentral, puertoCentral,
                    "WARNING: Sensor de movimiento detecta actividad inusual");

            // Envío de un aviso crítico
            enviarAviso(socket, direccionCentral, puertoCentral,
                    "ALERT: Posible intrusión detectada. Avisar a las autoridades");

            // Cerramos el socket una vez enviados todos los avisos
            socket.close();

        } catch (Exception e) {
            // En caso de error mostramos un mensaje y la traza
            System.out.println("Se ha producido un error en el dispositivo de alarma");
            e.printStackTrace();
        }
    }
```

```java

    /**
     * Este método se encarga de enviar un aviso concreto a la central.
     * Recibe el mensaje como texto y lo envía mediante UDP.
     */
    private static void enviarAviso(DatagramSocket socket,
                                    InetAddress direccion,
                                    int puerto,
                                    String mensaje) throws Exception {

        // Convertimos el mensaje de texto en un array de bytes,
        // ya que UDP trabaja siempre a nivel de bytes
        byte[] datos = mensaje.getBytes();

        // Creamos el paquete UDP indicando:
        // - Los datos a enviar
        // - El tamaño del mensaje
        // - La dirección IP de destino
        // - El puerto de destino
        DatagramPacket paquete = new DatagramPacket(
                datos, datos.length, direccion, puerto);

        // Enviamos el paquete a la red
        socket.send(paquete);

        // Mostramos por consola el mensaje enviado (solo para pruebas)
        System.out.println("Alarma envía aviso -> " + mensaje);
    }
}

```

4.1 Función del cliente

El cliente representa un dispositivo de alarma doméstico. Su función principal es enviar distintos tipos de avisos a la central mediante comunicación UDP. Enviamos tres tipos de mensajes en este caso. **INFO** son mensajes informativos. **WARNING** son avisos de advertencia. **ALERT** son avisos críticos.

Cada mensaje se envía en formato de texto. El tipo de aviso precede al contenido para que la central lo interprete correctamente.

4.2 Explicación del código del cliente

La clase principal del cliente es **AlarmaClienteUDP**. Importamos primero las clases necesarias del paquete java.net. Estas nos permiten trabajar con direcciones **IP**, sockets y paquetes **UDP**.

Obtenemos la dirección IP del servidor en el método **main**. Utilizamos "**localhost**" para las pruebas. Esto indica que el servidor se ejecuta en el mismo ordenador.

Definimos el puerto de recepción de la central a continuación. El cliente y el servidor deben utilizar el mismo puerto. La comunicación es imposible sin este requisito.

Después creamos un **DatagramSocket**. **UDP** no establece una conexión persistente. Simplemente creamos el socket y lo utilizamos. enviar los mensajes.

Usamos un método auxiliar llamado **enviarAviso** para ordenar el código. Este método realiza las siguientes tareas:

Recibe el mensaje como texto.

Convierte el texto en un arreglo de **bytes**.

Crea un **DatagramPacket** con los datos necesarios.

Envía el paquete por el **socket**.

Cerramos el **socket** tras enviar los tres mensajes para liberar recursos. Todo el proceso ocurre dentro de un bloque **try-catch**. Esto captura errores de red. El programa evita cierres inesperados.

05

DESARROLLO DEL SERVIDOR (CENTRAL DE ALARMAS)

AlarmaClienteUDP.java CentralReceptoraUDP.java

```java
package udp;

import java.net.DatagramPacket;

public class CentralReceptoraUDP {

    public static void main(String[] args) {

        DatagramSocket socket = null; // declaramos la variable aquí para que exista fuera

        try {
            // Puerto en el que la central va a escuchar los avisos
            int puertoEscucha = 5000;

            // Creamos el socket UDP asociado a ese puerto
            socket = new DatagramSocket(puertoEscucha);

            System.out.println("Central de alarmas iniciada y a la espera de avisos...\n");

            // La central debe estar siempre activa, por eso usamos un bucle infinito
            while (true) {

                // Creamos un buffer donde se almacenará temporalmente el mensaje recibido
                byte[] buffer = new byte[1024];

                // Creamos el paquete UDP que recibirá la información
                DatagramPacket paquete = new DatagramPacket(buffer, buffer.length);

                // Este método se queda bloqueado hasta que llega un aviso
                socket.receive(paquete);

                // Convertimos los bytes recibidos en un String legible
                String mensajeRecibido = new String(
                        paquete.getData(), 0, paquete.getLength());

                // Procesamos el aviso recibido
                mostrarAviso(mensajeRecibido);

            }

        } catch (Exception e) {
            System.out.println("Error en la central de recepción de alarmas");
            e.printStackTrace();
        } finally {
            if (socket != null && !socket.isClosed()) {
                socket.close(); // libera el puerto
            }
        }
    }
```

```java

    /**
     * Este método analiza el tipo de aviso recibido y lo muestra
     * de forma clara en la consola de la central.
     */
    private static void mostrarAviso(String mensaje) {


        // Comprobamos el tipo de aviso según el prefijo del mensaje
        if (mensaje.startsWith("INFO")) {
            System.out.println("[INFORMACIÓN recibida] " + mensaje.substring(6));

        } else if (mensaje.startsWith("WARNING")) {
            System.out.println("[ADVERTENCIA recibida] " + mensaje.substring(9));

        } else if (mensaje.startsWith("ALERT")) {
            System.out.println("[ALERTA CRÍTICA recibida] " + mensaje.substring(7));

        } else {
            // Por si llega algún mensaje que no siga el formato esperado
            System.out.println("[MENSAJE DESCONOCIDO recibido] " + mensaje);
        }
    }
}

```

5.1 Función del servidor

El servidor actúa como central receptora de alarmas. Sus funciones principales son:

Escuchar en un puerto específico.

Recibir los mensajes de los clientes.

Analizar el tipo de aviso.

Mostrar el mensaje con claridad.

La central debe estar siempre disponible. El servidor usa un bucle infinito y espera nuevos avisos continuamente.

5.2 Explicación del código

La clase principal es **CentralReceptoraUDP**. Importamos las clases necesarias de **java.net** al inicio.

Creamos un **DatagramSocket** en un puerto específico dentro del método **main**. El **socket** espera los mensajes entrantes en ese puerto.

Usamos un bucle **while (true)**. El servidor sigue activo y recibe varios avisos seguidos.

Dentro del bucle ocurren estos pasos:

Creamos un búfer de **bytes** para guardar el mensaje.

Creamos un **DatagramPacket** vinculado a ese búfer.

Usamos el bloque **finally** para cerrar el socket ante un posible error o terminación del programa inesperado, para así no dejar el **puerto ocupado**.

El método **receive()** se bloquea hasta que llega un mensaje.

Convertimos los **bytes** a texto tras la recepción.

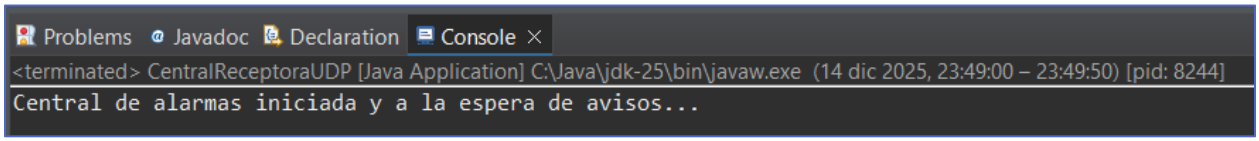
El método **mostrarAviso** procesa el mensaje para mayor claridad. Analiza el contenido. Usa condiciones **if** para ver si es **INFO**, **WARNING** o **ALERT**.

La consola muestra el mensaje con un formato distinto según el tipo. El operador lee la información fácilmente.

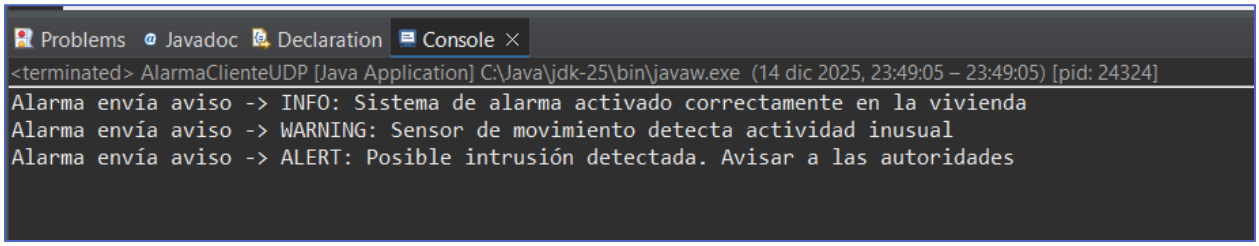
0 6
P R U E B A S Y F U N C I O N A M I E N T O

Ejecutamos ambos proyectos en el mismo equipo para probar el sistema:

Primero iniciamos el **servidor** (central).



Luego ejecutamos el **cliente** (alarma).

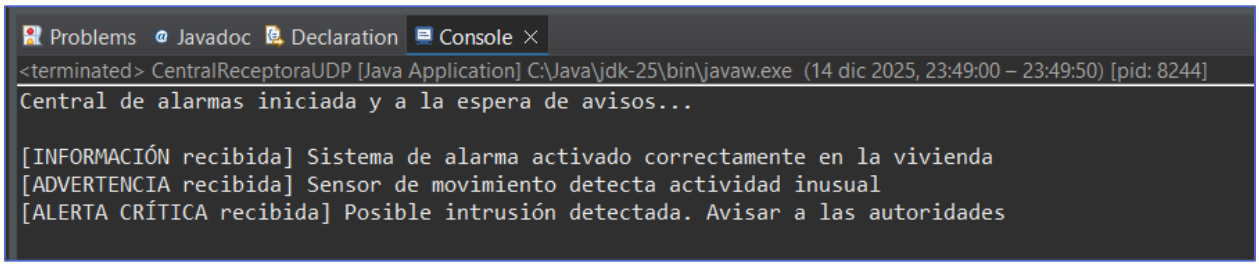


Vimos lo siguiente durante las pruebas:

El servidor espera mensajes correctamente.

El cliente envía los tres avisos sin problemas.

El servidor recibe cada mensaje y lo clasifica por tipo.



La comunicación UDP funciona bien. El sistema cumple con los requisitos del enunciado.

0 7
C O N C L U S I O N E S

Este caso práctico inició nuestra programación de aplicaciones distribuidas. También practicamos el uso de sockets en Java. Conceptos como UDP parecían abstractos al principio. El ejemplo práctico los hizo comprensibles.

Aprendimos varios puntos durante el desarrollo:

No todas las comunicaciones necesitan TCP.

UDP sirve para sistemas de avisos y notificaciones.

Java tiene herramientas de red potentes y simples.

Separar cliente y servidor ayuda al diseño y las pruebas.

Este ejercicio aclaró el funcionamiento de los sistemas en red. Estamos listos para desarrollos complejos en el futuro.

0 8
R E F E R E N C I A S

https://docs.oracle.com/en/java/javase/25/docs/api/java.base/java/net/DatagramPacket.html

https://www.geeksforgeeks.org/java/working-udp-datagramsockets-java/

https://docs.oracle.com/javase/tutorial/networking/datagrams/index.html

https://www.programacion.com.py/escritorio/java-escritorio/sockets-en-java-udp-y-tcp

https://www.youtube.com/watch?v=v20amqH9mqs

https://www.youtube.com/watch?v=fdjOcvDpSD0