Actividad 3.1. El objetivo del ejercicio es crear una aplicación cliente/servidor que se comunique por el puerto 2000 y realice lo siguiente:

El servidor debe generar un número secreto de forma aleatoria entre el 0 al 100. El objetivo de cliente es solicitarle al usuario un número y enviarlo al servidor hasta que adivine el número secreto. Para ello, el servidor para cada número que le envía el cliente le indicará si es menor, mayor o es el número secreto del servidor.

Clase Servidor

```
/**
 * Clase Servidor que gestiona las conexiones de múltiples clientes mediante hilos.
 *
 * El servidor escucha en el puerto 2000 y crea una instancia de {@link HiloServidor}
 * para manejar cada cliente que se conecta. Permite la ejecución concurrente de múltiples
 * clientes.
 */
public class Servidor {

    /**
     * Método principal para iniciar el servidor.
     *
     * Este método crea un ServerSocket para escuchar conexiones en el puerto 2000,
     * y maneja cada cliente que se conecta creando un nuevo hilo de tipo HiloServidor.
     *
     */
    public static void main(String[] args) {

        try {
            // Crea un servidor que escucha en el puerto 2000
            ServerSocket servidor = new ServerSocket(2000);
            System.out.println("Servidor iniciado");

            // Bucle infinito para aceptar múltiples conexiones de clientes
            while (true) {
                // Acepta una conexión entrante
                Socket sc = servidor.accept();

                // Crea y lanza un nuevo hilo para manejar al cliente conectado
                HiloServidor s = new HiloServidor(sc);
                s.start();
            }

        } catch (IOException ex) {
            // Manejo de errores relacionados con entrada/salida
            Logger.getLogger(Servidor.class.getName()).log(Level.SEVERE, null, ex);
        }

    }
}
```

Clase HiloServidor

```java
package com.mycompany.psp03_ejercicio1;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.Socket;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 * Clase HiloServidor que extiende Thread para gestionar la conexión con un cliente.
 *
 * El servidor genera un número aleatorio y el cliente debe adivinarlo enviando números.
 * Proporciona retroalimentación sobre si el número enviado es mayor, menor o correcto.
 */
public class HiloServidor extends Thread {

    private Socket sc;

    /**
     * Constructor de la clase HiloServidor.
     *
     * @param sc Socket asociado a la conexión del cliente.
     */
    public HiloServidor(Socket sc) {
        this.sc = sc;
    }
```

```java
/**
 * Método ejecutado cuando el hilo comienza su ejecución.
 * Gestiona la interacción con el cliente, incluyendo el intercambio de mensajes
 * y la lógica para adivinar el número.
 */
@Override
public void run() {

    System.out.println("Cliente conectado");
    DataInputStream in = null;
    DataOutputStream out = null;

    try {
        // Inicializa los flujos de entrada y salida
        in = new DataInputStream(sc.getInputStream());
        out = new DataOutputStream(sc.getOutputStream());

        // Genera un número aleatorio entre 0 y 100
        int aleatorio = generarNumeroAleatorio(0, 100);
        int numeroCliente;

        System.out.println("El número generado es: " + aleatorio);

        do {
            // Envía una solicitud al cliente para que ingrese un número
            out.writeUTF("Escribe un número entre 1 y 100:");

            // Recibe el número enviado por el cliente
            numeroCliente = in.readInt();

            System.out.println("El número enviado por el cliente es: " + numeroCliente);

            // Proporciona retroalimentación sobre el número enviado
            if (numeroCliente == aleatorio) {
                out.writeUTF("Has acertado el número");
            } else if (numeroCliente > aleatorio) {
                out.writeUTF("El número es menor");
            } else {
                out.writeUTF("El número es mayor");
            }

            // Informa al cliente si ha acertado
            out.writeBoolean(numeroCliente == aleatorio);

        } while (numeroCliente != aleatorio); // Finaliza cuando el cliente acierta el número
```

```
            // Cierra el socket y los recursos asociados
            sc.close();
            System.out.println("Cliente desconectado");

        } catch (IOException ex) {
            // Manejo de errores de entrada/salida
            Logger.getLogger(HiloServidor.class.getName()).log(Level.SEVERE, null, ex);
        } finally {
            // Cierra los flujos de entrada y salida
            try {
                if (in != null) {
                    in.close();
                }
                if (out != null) {
                    out.close();
                }
            } catch (IOException ex) {
                Logger.getLogger(HiloServidor.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    }

    /**
     * Genera un número aleatorio dentro de un rango especificado.
     *
     * @param min Valor mínimo del rango (inclusive).
     * @param max Valor máximo del rango (inclusive).
     * @return Un número entero aleatorio entre min y max.
     */
    private int generarNumeroAleatorio(int min, int max) {
        int numAleatorio = (int) (Math.random() * (max - min + 1) + min);
        return numAleatorio;
    }
}
```

Clase Cliente

```java
package com.mycompany.psp03_ejercicio1;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.Socket;
import java.util.Scanner;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 * Clase Cliente para gestionar la conexión con un servidor a través de sockets.
 * Este cliente envía y recibe datos utilizando flujos de entrada y salida de datos.
 *
 * Flujo básico:
 *     1.Establece conexión con el servidor en el puerto 2000.
 *     2.Intercambia mensajes y números enteros con el servidor.
 *     3.Termina la conexión una vez finalizada la comunicación.
 */
public class Cliente {

    /**
     * Método principal para ejecutar la lógica del cliente.
     *
     * @param args Argumentos de línea de comandos (no se utilizan en este programa).
     */
    public static void main(String[] args) {

        try {
            // Se conecta al servidor en el puerto 2000 en la máquina local
            Socket sc = new Socket("localhost", 2000);

            // Flujos para enviar y recibir datos
            DataInputStream in = new DataInputStream(sc.getInputStream());
            DataOutputStream out = new DataOutputStream(sc.getOutputStream());

            boolean salir = false;
            Scanner entrada = new Scanner(System.in);

            do {
                // Lee un mensaje del servidor y lo imprime en la consola
                String mensaje = in.readUTF();
                System.out.println(mensaje);

                // Captura un número desde la entrada del usuario y lo envía al servidor
                int num = entrada.nextInt();
                out.writeInt(num);

                // Recibe otro mensaje del servidor y lo imprime
                mensaje = in.readUTF();
                System.out.println(mensaje);

                // Lee un valor booleano del servidor (aunque no se usa aquí)
                in.readBoolean();

            } while (!salir); // El bucle continuará indefinidamente hasta que se cambie la condición

            // Cierra el socket y los recursos asociados
            sc.close();

        } catch (IOException ex) {
            // Manejo de excepciones en caso de errores de E/S
            Logger.getLogger(Cliente.class.getName()).log(Level.SEVERE, null, ex);
        }

    }
```

3.2. El objetivo del ejercicio es crear una aplicación cliente/servidor que permita el envío de ficheros al cliente. Para ello, el cliente se conectará al servidor por el puerto 1500 y le solicitará el nombre de un fichero del servidor. Si el fichero existe, el servidor, le enviará el fichero al cliente y éste lo mostrará por pantalla. Si el fichero no existe, el servidor le enviará al cliente un mensaje de error. Una vez que el cliente ha mostrado el fichero se finalizará la conexión.Clase Servidor

```java
package com.mycompany.psp03_ejercicio2;

import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 * Clase Servidor que gestiona solicitudes de múltiples clientes para obtener el contenido de archivos.
 *
 * El servidor escucha conexiones en el puerto 1500 y crea un hilo de tipo HiloServidor
 * para atender cada cliente de manera concurrente. Permite manejar múltiples conexiones simultáneamente.
 */
public class Servidor {

    /**
     * Método principal para iniciar el servidor.
     *
     * Este método configura un ServerSocket para escuchar en el puerto 1500
     * y entra en un bucle infinito aceptando conexiones de clientes. Cada conexión
     * es manejada en un hilo independiente usando la clase HiloServidor.
     */

    public static void main(String[] args) {

        try {
            // Crea un servidor que escucha en el puerto 1500
            ServerSocket servidor = new ServerSocket(1500);
            System.out.println("Servidor iniciado");

            // Bucle infinito para aceptar múltiples conexiones de clientes
            while (true) {
                // Acepta una conexión entrante
                Socket sc = servidor.accept();

                // Crea y lanza un nuevo hilo para manejar al cliente conectado
                HiloServidor s = new HiloServidor(sc);
                s.start();
            }

        } catch (IOException ex) {
            // Manejo de errores relacionados con entrada/salida
            Logger.getLogger(Servidor.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

Clase HiloServidor

```java
package com.mycompany.psp03_ejercicio2;

import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.net.Socket;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 * Clase HiloServidor que extiende Thread para gestionar la solicitud de archivos de un cliente.
 *
 * El hilo se encarga de recibir la ruta de un archivo desde el cliente, verificar si el archivo
 * existe y, en caso afirmativo, enviar su contenido. Si el archivo no existe, notifica al cliente.
 */
public class HiloServidor extends Thread {

    private Socket sc;

    /**
     * Constructor de la clase HiloServidor.
     *
     * @param sc Socket asociado a la conexión con el cliente.
     */
    public HiloServidor(Socket sc) {
        this.sc = sc;
    }

    /**
     * Método ejecutado cuando el hilo comienza su ejecución.
     *
     * Gestiona la interacción con el cliente:
     *     1.Recibe la ruta de un archivo.
     *     2.Verifica si el archivo existe.
     *     3.Si existe, lee y envía el contenido del archivo.
     *     4.Si no existe, informa al cliente.
     */

    @Override
    public void run() {
        DataInputStream in = null;
        DataOutputStream out = null;

        try {
            // Inicializa los flujos de entrada y salida
            in = new DataInputStream(sc.getInputStream());
            out = new DataOutputStream(sc.getOutputStream());

            // Lee la ruta del archivo enviada por el cliente
            String ruta = in.readUTF();

            // Verifica si el archivo existe
            File fichero = new File(ruta);

            if (fichero.exists()) {
                // Notifica al cliente que el archivo existe
                out.writeBoolean(true);

                // Lee el contenido del archivo línea por línea
                BufferedReader br = new BufferedReader(new FileReader(ruta));
                String linea;
                String contenido = "";

                while ((linea = br.readLine()) != null) {
                    contenido += linea + "\r\n";
                }

                br.close();
```

```
                // Convierte el contenido del archivo a un arreglo de bytes
                byte[] contenidoFichero = contenido.getBytes();

                // Envía el tamaño del archivo al cliente
                out.writeInt(contenidoFichero.length);

                // Envía el contenido del archivo byte por byte
                for (int i = 0; i < contenidoFichero.length; i++) {
                    out.writeByte(contenidoFichero[i]);
                }

                // Cierra la conexión con el cliente
                sc.close();
            } else {
                // Notifica al cliente que el archivo no existe
                out.writeBoolean(false);
            }

        } catch (IOException ex) {
            // Manejo de errores de entrada/salida
            Logger.getLogger(HiloServidor.class.getName()).log(Level.SEVERE, null, ex);
        } finally {
            // Cierra los flujos de entrada y salida
            try {
                if (in != null) {
                    in.close();
                }
                if (out != null) {
                    out.close();
                }
            } catch (IOException ex) {
                Logger.getLogger(HiloServidor.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    }
}
```

Clase Cliente

```java
package com.mycompany.psp03_ejercicio2;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.Socket;
import java.util.Scanner;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 * Clase Cliente para solicitar al servidor el contenido de un archivo.
 *
 * El cliente se conecta al servidor en el puerto 1500, envía la ruta de un archivo,
 * y espera recibir su contenido si el archivo existe. Si no existe, informa al usuario.
 */
public class Cliente {

    /**
     * Método principal que implementa la lógica del cliente.
     *
     * Se establece la conexión con el servidor, se envía la ruta del archivo solicitada
     * por el usuario, y se procesa la respuesta del servidor para mostrar el contenido del
     * archivo o un mensaje de error.
     */
    public static void main(String[] args) {

        try {
            // Se conecta al servidor en el puerto 1500
            Socket sc = new Socket("localhost", 1500);

            // Inicializa los flujos de entrada y salida
            DataInputStream in = new DataInputStream(sc.getInputStream());
            DataOutputStream out = new DataOutputStream(sc.getOutputStream());

            Scanner entrada = new Scanner(System.in);

            // Solicita al usuario la ruta del archivo
            System.out.println("Introduce la ruta donde se encuentra el archivo que quieres mostrar:");
            String ruta = entrada.next();

            // Envía la ruta del archivo al servidor
            out.writeUTF(ruta);

            // Verifica si el archivo existe en el servidor
            boolean existe = in.readBoolean();

            if (existe) {
                // Si existe, lee el tamaño del contenido del archivo
                int longitud = in.readInt();

                // Crea un arreglo de bytes para almacenar el contenido del archivo
                byte[] contenido = new byte[longitud];

                // Recibe el contenido del archivo byte por byte
                for (int i = 0; i < longitud; i++) {
                    contenido[i] = in.readByte();
                }

                // Convierte el contenido recibido a String y lo muestra
                String contenidoFichero = new String(contenido);
                System.out.println(contenidoFichero);

            } else {
                // Si el archivo no existe, informa al usuario
                out.writeBoolean(false);
                System.out.println("No existe el fichero especificado");
            }

            // Cierra el socket y los recursos asociados
            sc.close();

        } catch (IOException ex) {
            // Manejo de errores de entrada/salida
            Logger.getLogger(Cliente.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```