**1. ¿Para qué sirve el programa?**

**El programa es un "Escáner de Red" que permite a los usuarios escanear un rango de direcciones IP. Su función principal es determinar qué equipos están activos en una red, resolviendo sus nombres de host y midiendo el tiempo de respuesta. Los resultados del escaneo se muestran en una tabla, se puede filtrar por equipos activos e inactivos, y también guardar los resultados en un archivo CSV.**

**2. Cómo está armado el sistema (con diagramas)**

**El sistema está construido con una arquitectura de múltiples clases, cada una con una responsabilidad específica.**

**Diagrama de Clases**
**Fragmento de código**

```
classDiagram
  EscanerDeRed "1" -- "1" Escaneador : usa
  EscanerDeRed "1" -- "1" TablaResultados : usa
  EscanerDeRed "1" -- "1" AnalizadorResultados : usa
  EscanerDeRed "1" -- "1" ValidadorIP : usa
  Escaneador "1" -- "1" Escaneo : usa
  Escaneador "1" -- "1" ResultadoEscaneo : publica
  Escaneo "1" -- "1" ResultadoEscaneo : crea
  TablaResultados "1" -- "*" ResultadoEscaneo : contiene
  AnalizadorResultados "1" -- "*" ResultadoEscaneo : filtra y cuenta
  EscanerDeRed "1" -- "*" ResultadoEscaneo : contiene

  class EscanerDeRed {
    +EscanerDeRed()
    -iniciarEscaneo()
    -detenerEscaneo()
    -limpiarResultados()
    -guardarResultados()
    -actualizarContador()
    -validarCampos()
    -ipToLong(String ip)
  }


  class Escaneador {
    <<SwingWorker>>
    +Escaneador(String, String, int, int, JProgressBar, TablaResultados,
List<ResultadoEscaneo>)
    #doInBackground()
    #process(List<ResultadoEscaneo>)
    #done()
  }

  class Escaneo {
    +escanearIP(String ip, int espera, int reintentos)
```

```
    +obtenerNombreHost(String ip)
}

class ResultadoEscaneo {
    -String ip
    -String nombreEquipo
    -boolean conectado
    -long tiempoRespuesta
    +ResultadoEscaneo(String, String, boolean, long)
    +getters()
    +getEstado()
}

class TablaResultados {
    <<AbstractTableModel>>
    -List<ResultadoEscaneo> datos
    -String[] columnas
    +setDatos(List<ResultadoEscaneo>)
    +getRowCount()
    +getColumnCount()
    +getValueAt(int, int)
    +getColumnClass(int)
}
class ValidadorIP {
    +esIPValida(String ip)
}
class AnalizadorResultados {
    +contarActivos(List<ResultadoEscaneo>)
    +filtrarPorEstado(List<ResultadoEscaneo>, boolean)
    +ordenarPorTiempo(List<ResultadoEscaneo>, boolean)
}
```

**3. Qué métodos se usaron y por qué**

- **SwingWorker: Esta clase se utilizó en Escaneador.java para realizar el escaneo de red en un hilo separado del hilo de la interfaz gráfica de usuario (GUI). Esto evita que la aplicación se "congele" mientras se realiza el escaneo, manteniendo la GUI receptiva. El método doInBackground() realiza la tarea pesada, publish() envía resultados parciales a la GUI y process() los maneja en el hilo de la GUI.**
- **ProcessBuilder: Se usó en Escaneo.java para ejecutar comandos del sistema operativo, como ping y nslookup. Esto permite la interacción con el sistema para realizar las pruebas de conectividad y la resolución de nombres de host, adaptándose a los comandos de sistemas Windows y Unix.**
- **AbstractTableModel: En TablaResultados.java, esta clase fue extendida para crear un modelo de datos personalizado para la JTable. Esto desacopla la lógica de los datos de la presentación visual, permitiendo que la tabla se actualice dinámicamente a medida que se obtienen nuevos resultados.**

- **Streams (java.util.stream): Se utilizaron en AnalizadorResultados.java para realizar operaciones funcionales sobre las listas de resultados, como filtrar los equipos activos (filter) o contarlos (count). Esto permite que el código sea más legible y conciso para manipular colecciones de datos.**
- **Métodos de conversión ipToLong y longToIp: Se implementaron para convertir direcciones IP (en formato de cadena) a un número largo (long) y viceversa. Esta conversión facilita el manejo del rango de IP a escanear, ya que permite realizar operaciones matemáticas sencillas para iterar de una IP de inicio a una de fin.**

**4. Por qué se eligieron ciertas tecnologías**

- **Java Swing: Se eligió para la interfaz gráfica (GUI) debido a su simplicidad y a que es una tecnología nativa de Java, lo que facilita la creación de aplicaciones de escritorio multiplataforma sin dependencias adicionales.**
- **ping y nslookup (comandos del sistema): En lugar de implementar una lógica de ping o resolución de DNS nativa en Java, se optó por utilizar los comandos del sistema operativo (ping, nslookup) a través de ProcessBuilder. Esto se debe a que son herramientas robustas, confiables y ya optimizadas para su tarea. Permiten una solución más simple y directa para probar la conectividad y obtener nombres de host en diferentes sistemas operativos.**

**5. Qué problemas aparecieron y cómo se solucionaron**

- **Números incorrectos: tuve un problema a la hora de insertar valores <= 0 en los campos de velocidad ms y reintentos. Lo que hice fue hacer que marque incorrecto el poner valores incorrectos, valga la redundancia**

**6. Qué se podría mejorar en el futuro**

- **Paralelización de Escaneo: Actualmente, el escaneo se realiza de forma secuencial, IP por IP, dentro del SwingWorker. Una mejora significativa sería implementar un escaneo multihilo para escanear varias IPs simultáneamente, lo que reduciría drásticamente el tiempo de escaneo en rangos grandes. Esto podría lograrse con un ExecutorService o un ForkJoinPool que gestione un grupo de hilos de escaneo.**
- **Escaneo de Puertos: Además de la verificación de conectividad (ping), se podría añadir la funcionalidad de escanear puertos comunes (como 80, 443, 22) para detectar servicios activos en los equipos. Esto proporcionaría información más detallada sobre los dispositivos de la red.**
- **Interfaz de Usuario Mejorada: La interfaz podría ser más avanzada, incluyendo:**
    - **Una vista de árbol para mostrar la topología de la red.**
    - **Gráficos en tiempo real que muestren la evolución de los tiempos de respuesta.**
    - **Opciones para exportar a otros formatos además de CSV (ej. JSON, PDF).**
    - **Un sistema de logs para registrar los eventos del escaneo.**

- **Manejo de Errores Más Robusto: El programa podría gestionar excepciones de manera más específica, brindando al usuario mensajes de error más detallados en lugar de un simple "Error durante el escaneo".**
- **Autodetección de Rango de IP: Se podría añadir la capacidad de detectar automáticamente el rango de IP local del usuario (por ejemplo, 192.168.1.0/24) para facilitar el inicio rápido del escaneo.**
- **Rango de ips: También hay un error concreto que al final no pude resolver que no permite escanear más de 100**