Escaner de Red

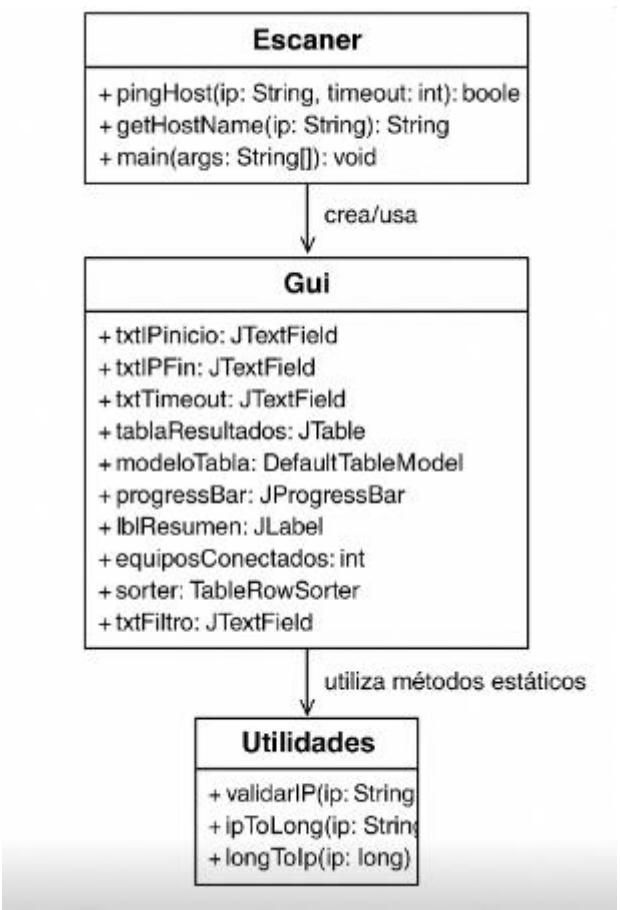
Para qué sirve el programa

El **Escáner de IPs** es una herramienta gráfica desarrollada en Java que permite analizar un rango de direcciones IP dentro de una red.
Su objetivo principal es detectar qué dispositivos están activos, conocer su nombre (si es posible resolverlo) y medir el tiempo de respuesta en milisegundos.

Casos de uso:

1) Administradores de red que necesitan verificar conectividad.
2) Estudiantes para practicar conceptos de redes.
3) Usuarios domésticos que quieran identificar dispositivos conectados en su LAN.

Cómo está hecho el programa (modelo de clases)



Qué métodos se usaron y por qué

División en 3 clases

El sistema se armó en **tres clases principales** (Escaner, Gui y Utilidades) porque cada una representa una **responsabilidad distinta** dentro del programa (principio de responsabilidad única).

Escaner

Es la parte lógica del sistema.

Se encarga de las operaciones de red: hacer un *ping* a una IP y obtener el nombre del host.

También contiene el main, que es el punto de entrada.

La idea es que toda la lógica relacionada con conectividad esté separada de la interfaz gráfica.

De esta forma, si mañana se quisiera usar el escáner en una aplicación de consola, bastaría con reutilizar esta clase sin necesidad de modificar la GUI.

Gui

Es la **interfaz gráfica del usuario** (la ventana que vos ves).

Maneja todo lo visual: los campos de texto, botones, tabla de resultados, barra de progreso.

También es la que coordina entre el usuario y el motor (Escaner), por ejemplo: cuando apretás "Escanear", llama a los métodos de Escaner y actualiza la tabla.

Aquí no se hace lógica de red, porque la idea es mantener el código ordenado: lo visual en la GUI, lo técnico en Escaner.

Utilidades

Contiene funciones auxiliares comunes, que no son ni visuales ni estrictamente de red, pero necesarias para el funcionamiento:

Validar que una IP sea válida.

Convertir IP en número largo (ipToLong) para poder recorrer un rango.

Pasar de número largo a IP (longToIp).

Esta clase existe para **no ensuciar ni la GUI ni el escáner con código repetitivo o secundario**.

Por qué hacerlo así

Separación de responsabilidades: cada clase hace una sola cosa bien definida.

Escalabilidad: si mañana quisiera cambiar la GUI por JavaFX o convertirlo en una app web, el motor de escaneo (Escaner y Utilidades) se puede reutilizar sin problemas.

Mantenibilidad: el código es más fácil de leer y de arreglar porque está ordenado por roles.

Rendimiento: el escaneo se hace en un Thread separado para que la GUI no se congele, y se usa invokeLater para actualizar la interfaz sin romper el hilo gráfico de Swing.

Por qué se eligió java

A continuación se enumerarán las razones por las cuales se eligió Java:
1) Lenguaje fuertemente orientado a objetos: Java es un lenguaje fuertemente orientado a objetos, por lo que facilitaba el hecho de pensar en objetos.
2) Lenguaje aprendido: Java es el primer y único lenguaje orientado a objetos en el cual puedo programar, porque lo que aprender otras sintaxis como C# sería más tedioso.
3) Lenguaje recomendado por grandes empresas: empresas como Microsoft (creadora de C#) recomiendan y usan Java por lo que es un lenguaje muy versátil.

Problemas que surgieron y se solucionaron

Durante el desarrollo del programa, uno de los principales problemas fue el **rendimiento del escaneo de IPs**.

En un inicio, la aplicación intentaba realizar los *pings* de forma **secuencial dentro del hilo principal** (el mismo que maneja la interfaz gráfica). Esto generaba dos inconvenientes importantes:

> **La interfaz se congelaba** mientras se realizaba el escaneo, ya que Swing no podía refrescar la ventana hasta que terminara el proceso.
>
> **El escaneo resultaba muy lento**, especialmente en rangos grandes de direcciones IP, porque se procesaba una por una sin permitir que la aplicación siguiera respondiendo al usuario.

La solución fue implementar el escaneo dentro de un **hilo separado (Thread)**. De esta forma:

> La lógica de escaneo corre en segundo plano, sin bloquear la interfaz.
>
> Swing puede seguir actualizando la **barra de progreso**, la **tabla de resultados** y el **contador de equipos conectados** en tiempo real.
>
> Se logra una mejor **experiencia de usuario**, ya que la aplicación responde inmediatamente y muestra el avance del proceso.

En conclusión, el uso de **multihilo** permitió optimizar el rendimiento y hacer que la herramienta fuera realmente usable en rangos amplios de direcciones IP.

Qué se podría mejorar en el futuro

Si bien el programa cumple con la función de escanear un rango de direcciones IP y mostrar los equipos activos, existen varias mejoras que podrían incorporarse en futuras versiones para ampliar su utilidad y optimizar la experiencia del usuario:

Botón de detener el escaneo

Actualmente, una vez que el escaneo comienza, el usuario debe esperar a que termine el rango completo.

Sería útil implementar un botón **"Detener"**, que interrumpa el hilo de ejecución y permita cancelar el proceso en cualquier momento. Esto daría mayor control al usuario y evitaría esperas innecesarias en rangos muy grandes.

Obtención de más información de los equipos

Además de la IP y el nombre de host, se podría mostrar información como:

Dirección MAC de la tarjeta de red.

Sistema operativo detectado (mediante técnicas de fingerprinting).

Puertos abiertos y servicios disponibles.

Esto convertiría el programa en una herramienta más completa de diagnóstico de red.

Mejorar la visualización de resultados

Incorporar iconos o colores para distinguir equipos conectados y desconectados.

Añadir **exportación a diferentes formatos** (CSV, Excel, PDF).

Mostrar gráficos estadísticos simples (ejemplo: cantidad de dispositivos activos por subred).

Optimización del rendimiento

Implementar **uso de múltiples hilos en paralelo** para acelerar el escaneo de grandes bloques de direcciones.

Permitir ajustar el número de hilos desde la interfaz.

Compatibilidad y portabilidad

Crear una versión multiplataforma empaquetada, que funcione fácilmente en Windows, Linux y macOS sin necesidad de configurar manualmente el entorno de ejecución.

Interfaz más avanzada

Integrar un panel de configuración avanzado.

Guardar configuraciones de escaneo favoritas para reutilizarlas.

Añadir un **modo oscuro** para mejorar la experiencia de uso prolongado.