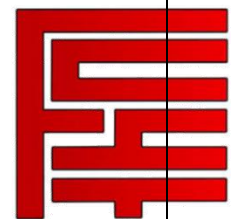


UNIVERSIDAD MAYOR DE SAN SIMÓN
FACULTAD DE CIENCIAS Y TECNOLOGÍA



TRABAJO FINAL

CALCULADORA IP

Universitarios(as):

- Fuentes Claros Leonardo.
- Garcia Chambilla Edwin Ramiro.
- Roman Torrico Elizabeth Andrea.

Carrera: Ing. Sistemas

Docente: Ing. Carlos Benito Manzur Soria

Cochabamba – Bolivia

ÍNDICE

Definición	1
Por Qué Usar La Calculadora De Subred Ip	1
1. Descripción del proyecto (su funcionamiento y/o variaciones, en caso de nuevas funcionalidades adicionales).	2
2. Objetivo General	3
3. Objetivos Específicos.....	3
3.5 Límites y alcances	3
4. Diagrama de clases (y su evolución del diagrama en diferentes versiones)	4
5. Presentación (videos, presentaciones, audios, etc.)	9
Evolución de su programa	9
La forma de trabajo	26
Los puntos más relevantes del programa	29
Mostrar el funcionamiento del programa	34
Uso de control de errores (try-catch).....	34
Seleccionar una clase y sus métodos para realizar y aplicar pruebas de Test (JUnit).	38

INTRODUCCIÓN

Definición

La Calculadora de Subred IP es una herramienta que realiza cálculos de los valores de red. Usa la clase de red, dirección IP y máscara de subred para calcular y devolver una lista de datos sobre las subredes.

La Calculadora de Subred IP es una herramienta útil que puede ayudar a todos los administradores de red, desde principiantes hasta profesionales de TI experimentados, a solucionar problemas y administrar sus redes.

Por Qué Usar La Calculadora De Subred Ip

Una calculadora de subred IP se puede usar para calcular la cantidad de subredes en una red más grande, la cantidad de hosts que pueden caber en una subred y el rango de todas las subredes.

Las calculadoras de IP nos permiten realizar todos los cálculos, como conversiones de decimal a binario, sin tener que hacerlo manualmente. La Calculadora de Subred IP hace que dividir una red en subredes sea mucho más rápido y fácil.

CONTENIDO

1. Descripción del proyecto (su funcionamiento y/o variaciones, en caso de nuevas funcionalidades adicionales).

Calculadora Ip es un proyecto que consiste en implementar una calculadora que dada:

La Dirección IP: La dirección IPv4 utiliza una dirección de 32 bits y puede admitir de 232 direcciones IP. La IPv4 tiene 4 grupos de números, llamados octetos. Cada octeto oscila entre 0 y 255 y está separado por puntos (.).

La Clase de red: La red se divide en Clase A, Clase B y Clase C. La red de Clase A usa los primeros 8 bits de la dirección IP como “parte de su red”, y el resto de los bits los usa como “parte de los hosts” (24 bits), mientras que la Clase B usa los primeros 16 bits para la red y el resto para los hosts (16 bits) y la Clase C usa los primeros 24 bits y el resto para los hosts (8 bits)

El prefijo de Red (/): Indica cuántas direcciones IPv4 hay disponibles para los hosts de su red.

La cantidad de subredes: Una subred es una subdivisión de una red IP. Las subredes se forman cuando una red se divide en dos o más redes vinculadas.

Obtenga los siguientes datos:

- IP BINARIO.
- RED DECIMAL.
- RED BINARIO.
- PRIMERA IP UTILIZABLE.
- ÚLTIMA IP UTILIZABLE.
- BROADCAST DECIMAL.
- BROADCAST BINARIO.
- MÁSCARA DE RED DECIMAL.
- MÁSCARA DE RED BINARIO.
- CLASE.
- MÁSCARA DE SUBRED DECIMAL.
- MÁSCARA DE SUBRED BINARIO.
- HOST POR SUBRED.
- SALTO DE RED.

Para utilizar la Calculadora de Subred IPv4:

1. Introduzca la dirección IP octeto por octeto
2. Elija la clase de Red preferida (lista desplegable), que está determinada por sus primeros bits, haciendo clic en cualquiera de las tres opciones: Clase A (8 bits), B (16 bits) y C (24 bits), existe un cuadro de referencia en el lado derecho en el que se puede guiar para elegir una clase en base al valor del primer octeto.
3. Seleccione el Prefijo de Red (lista desplegable) según la clase de red que eligió.
4. Seleccione la cantidad de subredes que necesite calcular
5. Pulse el botón Generar para visualizar los resultados de todos los valores de red y dirección IP.
6. Pulse el botón "Eliminar" en caso que usted desee resetear la calculadora para volver a utilizarla.

2. Objetivo General

Diseñar una aplicación de escritorio orientado al área de redes para simplificar el cálculo de subredes y brindar la información más relevante sobre la red, para concretar su diseño utilizamos diferentes herramientas como el lenguaje de programación "java v8.2", netbeans v13, JUnit y documentación de internet.

3. Objetivos Específicos

- Identificar las clases, atributos y métodos que se utilizarán a lo largo de la implementación del proyecto.
- Implementar dichas clases, atributos y métodos de manera correcta haciendo uso de control de errores (try -catch) en el proyecto.
- Demostrar el correcto funcionamiento del proyecto en base a los requerimientos sugeridos.
- Implementar pruebas unitarias sobre una clase seleccionada haciendo uso de la herramienta JUnit.

3.5 Límites y alcances

Alcances:

- **Calculadora trabaja con ips de clases A, B, C**

La implementación del proyecto solo contempla los tres tipos de clases más comerciales

- **Calculadora solo podrá realizar subnetting de la red**

La implementación del proyecto solo contempla el cálculo de subredes

- **Calculadora mostrará el salto y octeto de cada subred**

La implementación del proyecto mostrará la información básica y necesaria para realizar subnetting de la red en base al salto de red y en qué octeto se realiza dicho salto, por lo que no será necesario generar una tabla de cada subred.

Límites:

- **Calculadora no trabaja con ips de clases D y E**

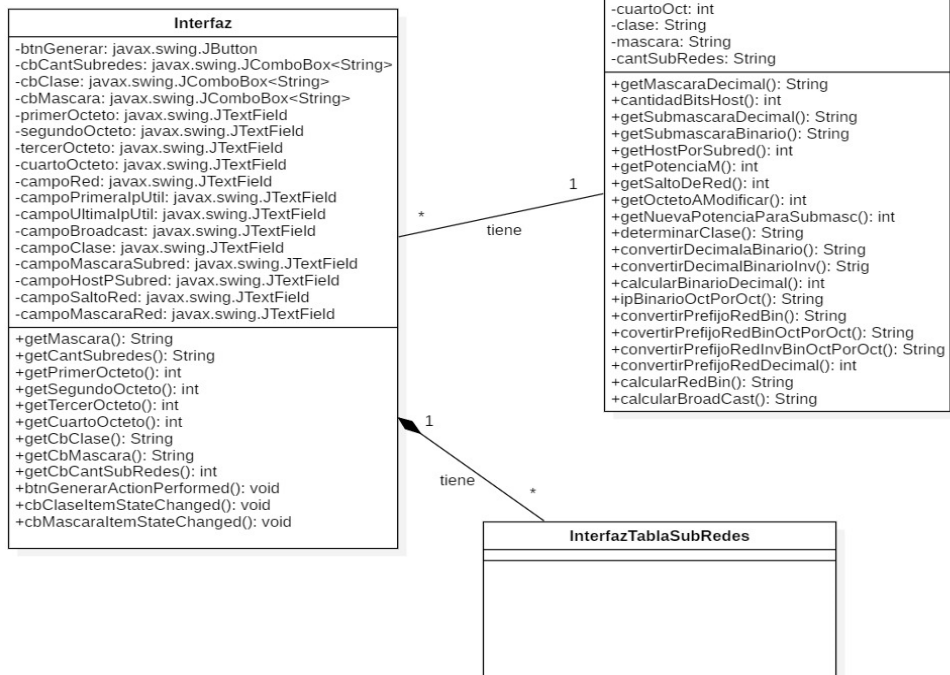
La implementación del proyecto no contempla a las clases D y E, debido a que clase D se utilizan para aplicaciones de multidifusión y la clase E, a pesar de que esta clase está reservada, nunca se definió su uso, por lo que la mayoría de las implementaciones de red descartan estas direcciones como ilegales o indefinidas.

- **Calculadora no podrá realizar supernetting de la red**

La implementación del proyecto no contempla el cálculo de superredes

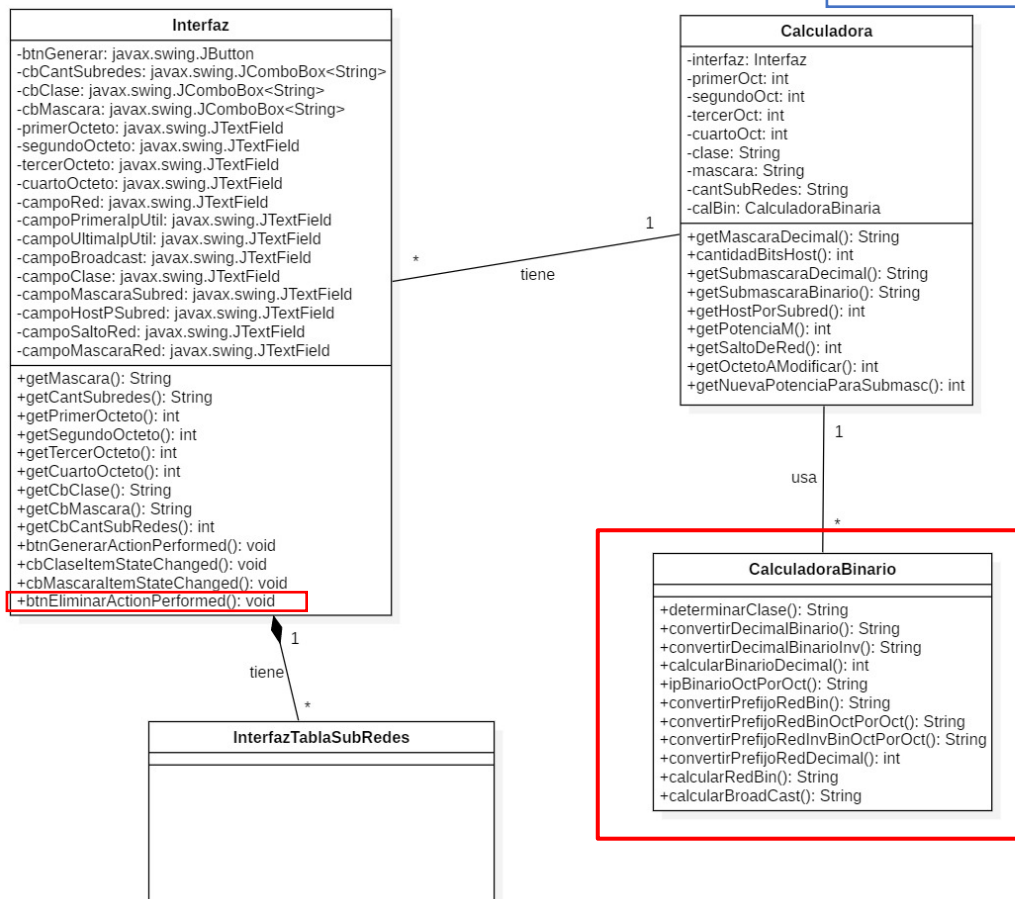
4. Diagrama de clases (y su evolución del diagrama en diferentes versiones)

Primera Versión:



Para esta primera versión se tenía planeado manejar toda la funcionalidad en una sola clase llamada “Calculadora”, esta se encargaría de realizar los cálculos de la red, primera ip utilizable, última ip utilizable, broadcast, máscara de red, host por subred y salto de red. Además contaría con las clases “Interfaz” e “InterfazTablaSubred” que se encargarían de manejar la interfaz de esta calculadora siendo Interfaz la que muestra los campos de Dirección ip, Clase, prefijo de red y Cantidad de subredes e InterfazTablaSubred la que muestra la información de la red y la subred.

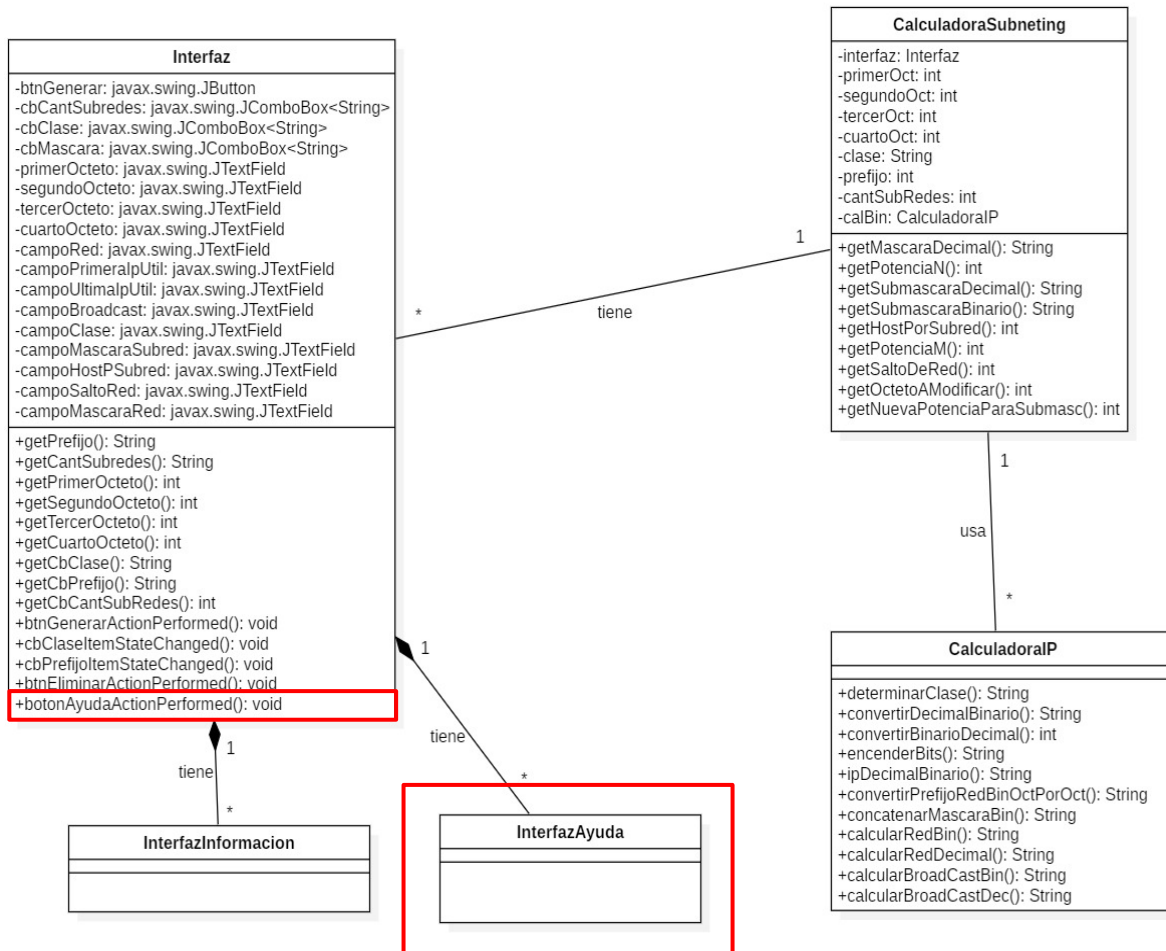
Segunda Versión:



Para esta Segunda versión se cambió la idea de mantener todas las funcionalidades en una sola clase y se agregó la clase “CalculadoraBinario”, esta se encargaría de realizar la mayoría de cálculos que se encontraran en Binario, mientras que la clase “Calculadora” aquellos que se encontraran en decimal.

A la clase interfaz se agregó un nuevo método “btnEliminarActionPerformed()” este se encargaría de dar funcionalidad al botón eliminar que se decidió agregar en la interfaz gráfica.

Tercera Versión:



Para esta última versión se hicieron varios cambios, empezando por renombrar las clases. La clase “Calculadora” fue renombrada como “CalculadoraSubneting”, la clase “CalculadoraBinario” como “CalculadoraIP” y la clase “InterfazTablaSubRedes” como “InterfazInformación” esta decisión fue tomada ya que los nuevos nombres se adecuaban mejor al orden y planificación de las funcionalidades de cada clase a diferencia de los anteriores nombres que podían causar confusión.

A demás se agregó una nueva clase llamada “InterfazAyuda” esta se encargaría de mostrar la información que contenga los pasos que se deben seguir para la comprensión del uso de la calculadora.

Otros cambios significativos en la evolución del diagrama de clases fueron:

CAMBIO DE NOMBRE A LOS MÉTODOS:

Clase Interfaz:

Antes	Después
getMascara()	getPrefijo()
getCbMascara()	getCbPrefijo()
cbMascaraItemStateChanged()	cbPrefijoItemStateChanged()

Clase CalculadoraSubneting:

Antes	Después
cantidadBitsHost()	getPotenciaN()

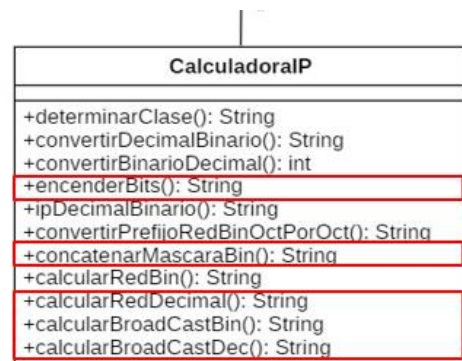
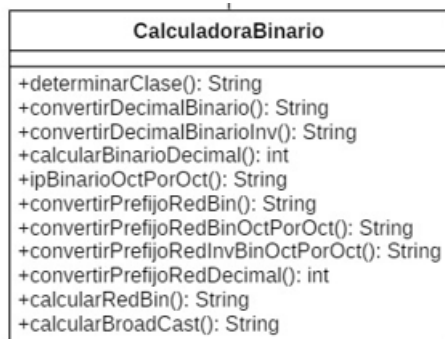
Clase CalculadoraIp:

Antes	Después
CalcularBinarioDecimal()	ConvertirBinarioDecimal()

AGREGACIÓN DE MÉTODOS:

Clase CalculadoraIp:

- encenderBits():String
- concatenarMascaraBin(): String
- calcularRedDecimal
- calcularBroadCastBin(): String
- calcularBroadCastDec(): String



Clase Interfaz:

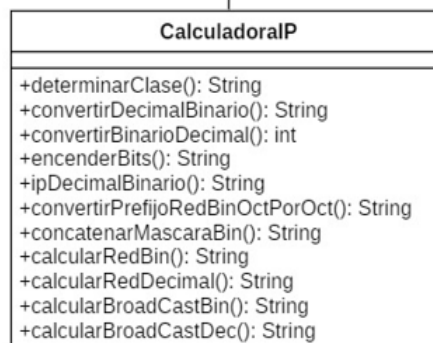
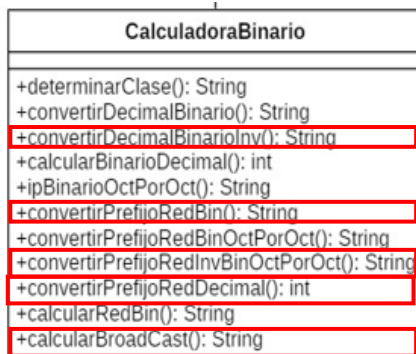
- botonAyudaActionPerformed(): void



ELIMINACIÓN DE MÉTODOS:

Clase CalculadoraIp:

- convertirDecimalBinarioInv():String
- convertirPrefijoRedBin(): String
- convertirPrefijoRedInvBinOctPorOct(): String
- convertirPrefijoRedDecimal(): int
- calcularBroadcast



5. Presentación (videos, presentaciones, audios, etc.)

Evolución de su programa

Se tuvo los siguientes avances de los cuales se hará un breve resumen

Este equipo > Escritorio > avancesTallerP >

Nombre	Fecha de modificación	Tipo	Tamaño
19deMayoSeparadoEnClases.zip	03/06/2022 16:39	Carpeta comprimi...	2.734 KB
Avance_28 de mayo v1.zip	30/05/2022 13:42	Carpeta comprimi...	1.372 KB
avance9mayo.zip	09/05/2022 21:13	Carpeta comprimi...	2.714 KB
avance12deMayo.zip	13/05/2022 0:35	Carpeta comprimi...	2.719 KB
avance14deMayo.zip	14/05/2022 16:32	Carpeta comprimi...	2.722 KB
avance16deMayo.zip	16/05/2022 16:12	Carpeta comprimi...	2.730 KB
avance16deMayoo.zip	16/05/2022 20:30	Carpeta comprimi...	2.731 KB
avance17deMayo.zip	17/05/2022 23:30	Carpeta comprimi...	2.736 KB
avance18deMayo1.0.zip	18/05/2022 17:52	Carpeta comprimi...	2.733 KB
avance20deMayo.zip	20/05/2022 22:28	Carpeta comprimi...	2.738 KB
avance22deMayo.zip	22/05/2022 12:50	Carpeta comprimi...	2.739 KB
avance22deMayoo.zip	22/05/2022 14:07	Carpeta comprimi...	2.741 KB
avance23deMayo(CalcRefactorizada).zip	23/05/2022 20:57	Carpeta comprimi...	2.745 KB
avance24deMayo.zip	24/05/2022 23:36	Carpeta comprimi...	2.745 KB
avance25deMayo.zip	25/05/2022 22:56	Carpeta comprimi...	2.744 KB
avance30deMayo.zip	30/05/2022 20:32	Carpeta comprimi...	1.372 KB
avances26deMayo.zip	27/05/2022 12:31	Carpeta comprimi...	1.369 KB
CalculadoraSubredes_V3.0.zip	03/06/2022 11:56	Carpeta comprimi...	1.364 KB
CalculadoraSubRedes	21/05/2022 15:18	Carpeta de archivos	

08/05/2022.- Antes de empezar a programar se hizo el diseño del mockup para la interfaz gráfica.

CALCULADORASUBREDES

DIRECCION IP:

MASCARA Decimal: a (/)

CANT DESUBREDES:

TIPO: CLASE A

GENERAR

Nº DE RED	SUBRED	RANGO IPS UTILIZABLES	BROADCAST

09/05/2022.- Para el primer avance se implementaron los bocetos de la primera interfaz gráfica que estaría destinada a recibir los datos.

The screenshot shows a Java Swing window titled "CALCULADORA SUBREDES". The window has a standard Mac OS X title bar with a red close button, a yellow maximize button, and a green window control button. The main content area is light gray and contains several input fields and a button. On the left, there are four input fields: "DIRECCIÓN IP:" (with four empty boxes for octets), "CLASE:" (a dropdown menu showing "Seleccione una clase"), "MÁSCARA:" (a dropdown menu showing "Seleccione una máscara"), and "CANT. DE SUBREDES:" (a dropdown menu showing "Seleccione la cantidad de subr..."). On the right, there is a list of class ranges: "CLASE A: 0 - 127", "CLASE B: 128 - 191", and "CLASE C: 192 - 223". At the bottom right, there is a button labeled "GENERAR".

CALCULADORA SUBREDES			
DIRECCIÓN IP:	<input type="text"/>	<input type="text"/>	<input type="text"/>
CLASE:	Seleccione una clase ▼		
MÁSCARA:	Seleccione una máscara ▼		
CANT. DE SUBREDES:	Seleccione la cantidad de subr... ▼		
CLASE A: 0 - 127 CLASE B: 128 - 191 CLASE C: 192 - 223			
GENERAR			

En este primer avance también se tenía planeado que se implemente una segunda interfaz que permita visualizar las redes ya subneteadas.

LISTADO DE SUBREDES			
Nº DE SUBRED	RED	RANGO IP'S UTILIZABLES	BROADCAST

14/05/2022.- Para este avance se empezó a tener avances significativos en cuanto al código puesto a que se obtuvo los primeros datos correctos derivados del código que se escribió, sin embargo, al ser un código muy primitivo todavía se optó por mostrar los resultados por consola.

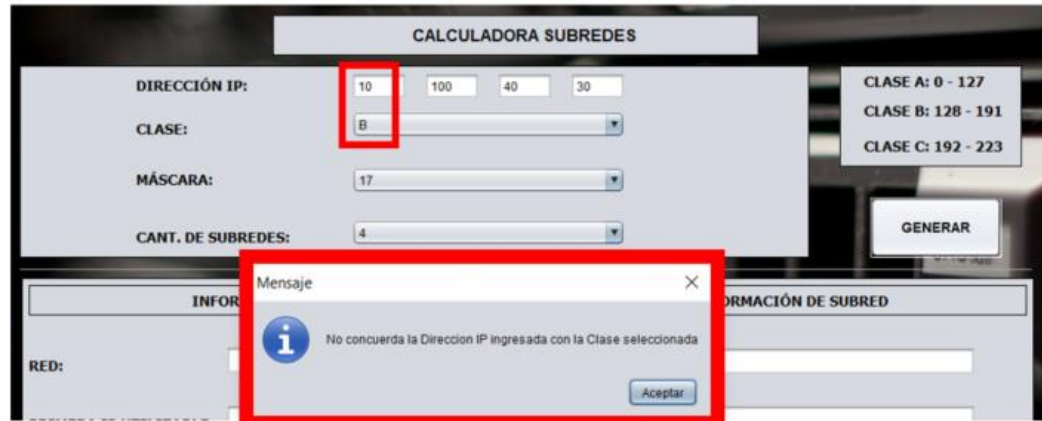
```

Output - CalculadoraSubRedes (run) X
run:
PotenciaBit: 4
Mascara: 255.255.240.0
SaltodeRed: 16
PotenciaMBitsRestantes:5
HostPorSubRed: 30
00001010010101100001111000101000 <---IPBIN
11111111100000000000000000000000 <---MASCARABIN
00001010010101100001111000101000 <---IPBIN
Exception in thread "AWT-EventQueue-0" java.lang.NumberFormatExcept
|   at java.lang.NumberFormatException.forInputString(NumberFor
|   at java.lang.Integer.parseInt(Integer.java:583)
|   at java.lang.Integer.parseInt(Integer.java:615)
|   at recursos.Calculadora.calcularRedBin(Calculadora.java:253)
|   at recursos.Calculadora.<init>(Calculadora.java:37)
|   at recursos.Interfaz.btnGenerarActionPerformed(Interfaz.jav

```

17/05/2022.- Para este avance se decidió hacer un cambio en cuanto a la interfaz gráfica que se implementaría y se optó por agregarle más espacios con el objetivo de que permita visualizar a más detalle los datos, también se empezó a hacer uso de métodos que permitieron mostrar los resultados directamente a dicha interfaz, sin embargo, todavía se tenía código que arreglar.

Un cambio a tomar en cuenta también fue la implementación de un método que limitaba e informaba en caso de ingresar una IP y una clase que no coincidieran.

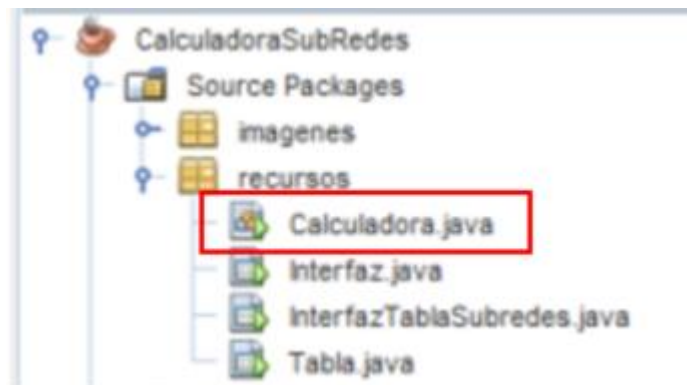


```

public String determinarClase() {
    String clase = "";
    String[] octetosIP = ipBinarioOctPorOct();
    if(octetosIP[0].charAt(0) == '0'){
        clase = "A";
    }else if(octetosIP[0].charAt(0) == '1' && octetosIP[0].charAt(1) == '0'){
        clase = "B";
    }else if(octetosIP[0].charAt(0) == '1' && octetosIP[0].charAt(1) == '1' && octetosIP[0].charAt(2) == '0'){
        clase = "C";
    }
    System.out.println(clase);
    return clase;
}

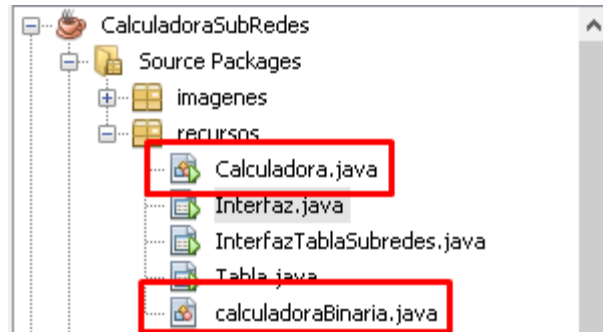
```

En este punto los métodos implementados que realizaban las operaciones estaban únicamente en la clase Calculadora lo que lo hacía desordenado y difícil de leer en algunos momentos.



19/05/2022.- Para este avance se decidió dividir la clase “Calculadora” en dos clases: “Calculadora” y “Calculadora Binaria”.

Calculadora contendrá los métodos que no usaran operaciones con números binarios y Calculadora Binaria contendrá métodos que usarán operaciones con números binarios.



20/05/2022.- Para este avance se decidió cambiar el tipo de dato del método “getCb máscara” (String a int), esto para no utilizar el método ParseInt de la clase Integer en las distintas llamadas a dicho método, además se cambió el nombre “ máscara” por un nombre más correcto al combobox “prefijo de red” en la interfaz .

22/05/2022.- Para este avance se implementó el botón eliminar para tener la opción de restablecer todos los campos a sus valores predeterminados.

```
// OBTENER LA MASCARA (PREFIJO DE RED) QUE SE SELECCIONO EN EL COMBOBOX DE MASCARA
public int getCbMascara() {
    int mascara = 0;
    mascara = Integer.parseInt((String) cbMascara.getSelectedItem());

    return mascara;
}
```

23/05/2022.- A partir de este avance se empezó con el proceso de refactorización de los métodos, documentar ciertas partes y renombramiento de algunos métodos para hacerlo más entendible.

Código antes de refactorizar. -

```
// ***** PASO 2: RESOLVER LA FORMULA 2^N >= C (METODO OBTIENE N) *****
public int cantidadBitsHost(){
    int cantidadSubR = cantSubRedes;
    int bitsDesig = 0;
    int potenciaBit = 0;
    boolean encontrado = false;
    for(int i = 0; i <= 24; i++){
        if( ((int) Math.pow(2, i)) >= cantidadSubR){
            encontrado = true;
            if(encontrado = true){
                bitsDesig =(int) Math.pow(2, i);
                potenciaBit = i;
            }
            break;
        }
    }
    System.out.println("PotenciaBit: " + potenciaBit);
    //    getSubmascara(potenciaBit);
    //    getHostPorSubred(potenciaBit);
    return potenciaBit;
}
```


Código después de refactorizar. -

```
// ***** PASO 2: RESOLVER LA FORMULA 2^N >= C (METODO OBTIENE N) *****
public int getPotenciaN() {
    int potenciaN = 0;
    for (int i = 0; i <= 24; i++) {
        if (((int) Math.pow(2, i)) >= cantSubRedes) { // Busca un numero cuyo valor elevado a la N sea >= a la cant subR
            potenciaN = i;
            break;
        }
    }

    return potenciaN;
}
```

Se redujo considerablemente la cantidad de líneas de código implementadas para este método y se simplificó el proceso ya que existían datos que no se empleaban en el programa.

Código antes de refactorizar. -

```
//PASO 3

public String getSubmascaraDecimal(){
    int potenciaBit = cantidadBitsHost();
    String res = "";
    String octetosSubmasc = "";
    String octetosMasc = convertirPrefijoRedBin();
    int nuevaPotencia = potenciaBit + Integer.parseInt(mascara);
    String bitsEncendidos = octetosMasc.substring(0, nuevaPotencia);
    bitsEncendidos = bitsEncendidos.replace('0', '1');
    String bitsApagados = octetosMasc.substring(nuevaPotencia, 32);
    octetosSubmasc = bitsEncendidos + bitsApagados;
    int primerOct = calcularBinarioDecimal(octetosSubmasc.substring(0, 8));
    int segundoOct = calcularBinarioDecimal(octetosSubmasc.substring(8, 16));
    int tercerOct = calcularBinarioDecimal(octetosSubmasc.substring(16, 24));
    int cuartoOct = calcularBinarioDecimal(octetosSubmasc.substring(24, 32));

    res = primerOct + "." + segundoOct + "." + tercerOct + "." + cuartoOct;
    System.out.println(res + " <--- SUBMASCARA");
    return res;
}
```

Código después de refactorizar. -

Se redujo la cantidad de líneas de código y se comentó para simplificar el entendimiento

Código antes de refactorizar. -

```
public int getHostPorSubred() {
    int potenciaM = getPotenciaM();
    int HostPorSubred;
    HostPorSubred = (int) (Math.pow(2, potenciaM) - 2);
    System.out.println("PotenciaMBitsRestantes:" + potenciaM);
    System.out.println("HostPorSubRed: " + HostPorSubred);
    return HostPorSubred;
}
```

Código después de refactorizar. -

```
public int getHostPorSubred() {
    int HostPorSubred = (int) (Math.pow(2, getPotenciaM()) - 2); // Obtener Host Por Subred en base a potencia M
    return HostPorSubred;
}

// PASO 3: OBTENER LA NUEVA MASCARA (SUBMASCARA) EN DECIMAL
public String getSubmascaraDecimal() {
    String subMascaraDec = "";
    String subMascBin = getSubmascaraBinario();
    int primerOct = calcBin.calcularBinarioDecimal(subMascBin.substring(0, 8));
    int segundoOct = calcBin.calcularBinarioDecimal(subMascBin.substring(8, 16)); // Separar la submascara en binario
    int tercerOct = calcBin.calcularBinarioDecimal(subMascBin.substring(16, 24)); // luego, Convertir submascara a decimal
    int cuartoOct = calcBin.calcularBinarioDecimal(subMascBin.substring(24, 32));

    subMascaraDec = primerOct + "." + segundoOct + "." + tercerOct + "." + cuartoOct; // Concatenar submascara en decimal
    return subMascaraDec;
}
```

Se redujeron las líneas de código necesarias para este método y se simplificó.

24/05/2022.- Para este avance se continuo con el proceso de refactorización de los métodos de la clase Calculadora. Se decidió refactorizar el método getSubMascaraBinario() y para hacerlo se añadió el método encenderBits() en la clase calculadoraBinaria el cual ayudaría a reducir código para hacerlo más entendible, además que este nuevo método nos ayudaría en algunos otros.

Código antes de refactorizar. -

```
// SUBMASCARA EN BINARIO (PASO 3)
public String getSubmascaraBinario() {
    String subMascBin = "";
    String mascaraBin = calcBin.concatenarMascaraBin(prefijo); // Obtener mascara Binario
    int nuevaPotencia = getNuevaPotenciaParaSubmasc(); // Cantidad de bits a encender
    String bitsEncendidos = mascaraBin.substring(0, nuevaPotencia); // Obtener Bits que deben encenderse
    bitsEncendidos = bitsEncendidos.replace('0', '1'); // Encendiendo Bits
    String bitsApagados = mascaraBin.substring(nuevaPotencia, 32); // Obtener Bits que deben apagarse
    subMascBin = bitsEncendidos + bitsApagados; //Concatenar Bits de la submascara
    String primerOct = subMascBin.substring(0, 8);
    String segundoOct = subMascBin.substring(8, 16); // Separar la submascara en binario
    String tercerOct = subMascBin.substring(16, 24);
    String cuartoOct = subMascBin.substring(24, 32);

    subMascBin = primerOct + segundoOct + tercerOct + cuartoOct; //Concatenar submascara en binario
    return subMascBin;
}
```

Código después de refactorizar. -

```
public String encenderBits(int bitsAEncender, String mascaraBin){
    String bitsEncendidos = "";
    bitsEncendidos = mascaraBin.substring(0, bitsAEncender); // Obtener Bits que deben encenderse
    bitsEncendidos = bitsEncendidos.replace('0', '1'); // Encendiendo Bits
    if(bitsAEncender < mascaraBin.length()){
        String bitsApagados = mascaraBin.substring(bitsAEncender, mascaraBin.length()); // Obtener Bits que deben apagarse
        bitsEncendidos = bitsEncendidos + bitsApagados; //Concatenar Bits
    }

    return bitsEncendidos;
}
```

```
// SUBMASCARA EN BINARIO (PASO 3)
public String getSubmascaraBinario() {
    String subMascBin = "";
    String mascaraBin = calcBin.concatenarMascaraBin(prefijo); // Obtener mascara Binario

    String octetos = calcBin.encenderBits(getNuevaPotenciaParaSubmasc(), mascaraBin);
    String primerOct = octetos.substring(0, 8);
    String segundoOct = octetos.substring(8, 16); // Separar la submascara en binario
    String tercerOct = octetos.substring(16, 24);
    String cuartoOct = octetos.substring(24, 32);

    subMascBin = primerOct + segundoOct + tercerOct + cuartoOct; //Concatenar submascara en binario

    return subMascBin;
}
```

Otros cambios que se realizo fue refactorizar los métodos en CalculadoraBinaria:

Método convertirDecimalBinario():

Código antes de refactorizar. -

```
// CONVERTIR UN OCTETO EN FORMATO DECIMAL A BINARIO (RETURN PASO 1)
private String convertirDecimalBinario(int octeto) {
    String cadenaBinario = "";
    int[] octetoBin = new int[8];
    int nuevoValor = octeto;
    for (int i = 7; i >= 0; i--) {
        if (i != -1) {
            if (nuevoValor >= Math.pow(2, i)) {
                nuevoValor = nuevoValor - (int) (Math.pow(2, i));
                octetoBin[octetoBin.length - 1 - i] = 1;
            } else {
                octetoBin[octetoBin.length - 1 - i] = 0;
            }
        }
    }
    for (int i = 0; i < 8; i++) {
        cadenaBinario = cadenaBinario + String.valueOf(octetoBin[i]);
    }
    // System.out.print(cadenaBinario);
    return cadenaBinario;
}
```

Código después de refactorizar. -

```
// CONVERTIR UN OCTETO DE DECIMAL A BINARIO (RETURN PASO 1)
private String convertirDecimalBinario(int octetoDecimal) {
    String octetoBinario = "";
    for (int i = 7; i >= 0; i--) { //Recorrer la tabla de derecha a izquierda
        if (i != -1) {
            if (octetoDecimal >= (int) Math.pow(2, i)) { // Comparar el octeto Decimal con la tabla
                octetoBinario = octetoBinario + 1; // Concatenar 1's a nuestra respuesta
                octetoDecimal = octetoDecimal - (int) Math.pow(2, i); // Actualizar el nuevo Valor
            } else {
                octetoBinario = octetoBinario + 0; // Concatenar 0's a nuestra respuesta
            }
        }
    }
    return octetoBinario;
}
```

Método convertirDecimalBinarioInv():

Código antes de refactorizar. -

```
// CONVERTIR UN OCTETO EN FORMATO DECIMAL A BINARIO PARA LA MASCARA INVERTIDA
private String convertirDecimalBinarioInv(int octeto) {
    String cadenaBinario = "";
    int[] octetoBin = new int[8];
    int nuevoValor = octeto;
    for (int i = 7; i >= 0; i--) {
        if (i != -1) {
            if (nuevoValor >= Math.pow(2, i)) {
                nuevoValor = nuevoValor - (int) (Math.pow(2, i));
                octetoBin[(octetoBin.length - 1) - i] = 0;
            } else {
                octetoBin[(octetoBin.length - 1) - i] = 1;
            }
        }
    }
    for (int i = 0; i < 8; i++) {
        cadenaBinario = cadenaBinario + String.valueOf(octetoBin[i]);
    }
    // System.out.print(cadenaBinario);
    return cadenaBinario;
}
```

Código después de refactorizar. -

```
// CONVERTIR UN OCTETO EN FORMATO DECIMAL A BINARIO PARA LA MASCARA INVERTIDA
private String convertirDecimalBinarioInv(int octetoDecimal) {
    String octetoBinario = "";
    for (int i = 7; i >= 0; i--) { //Recorrer la tabla de derecha a izquierda
        if (i != -1) {
            if (octetoDecimal >= (int) Math.pow(2, i)) { // Comparar el octeto Decimal con la tabla
                octetoBinario = octetoBinario + 0; // Concatenar 1's a nuestra respuesta
                octetoDecimal = octetoDecimal - (int) Math.pow(2, i); // Actualizar el nuevo Valor
            } else {
                octetoBinario = octetoBinario + 1; // Concatenar 0's a nuestra respuesta
            }
        }
    }
    return octetoBinario;
}
```

Método ipBinarioOctPorOct():

```
// ***** PASO 1: CONVERTIR LA DIRECCION IP INGRESADA A BINARIO *****
// CONVERTIR LA IP A BINARIO OCTETO POR OCTETO (RETURN METODO determinarClase())
public String[] ipBinarioOctPorOct(int[] ip) {

    String[] octetosIPBin = new String[4];
    String cad = "";
    octetosIPBin[0] = convertirDecimalBinario(ip[0]);
    octetosIPBin[1] = convertirDecimalBinario(ip[1]);
    octetosIPBin[2] = convertirDecimalBinario(ip[2]);
    octetosIPBin[3] = convertirDecimalBinario(ip[3]);

    for (int i = 0; i < 4; i++) {
        if (i == 3) {
            cad = cad + octetosIPBin[i];
        } else {
            cad = cad + octetosIPBin[i] + ".";
        }
    }
    System.out.println(cad + " <---IPBIN");
    return octetosIPBin;
}
```

```
// ***** PASO 1: CONVERTIR LA DIRECCION IP INGRESADA A BINARIO *****
// CONVERTIR LA IP A BINARIO OCTETO POR OCTETO (RETURN METODO determinarClase())
public String[] ipDecimalBinario(int[] ipDecimal) {
    String[] octetosIPBin = new String[4];

    octetosIPBin[0] = convertirDecimalBinario(ipDecimal[0]);
    octetosIPBin[1] = convertirDecimalBinario(ipDecimal[1]); // Convertir ip Decimal a
    octetosIPBin[2] = convertirDecimalBinario(ipDecimal[2]); // Binario Octeto por Octeto
    octetosIPBin[3] = convertirDecimalBinario(ipDecimal[3]);

    return octetosIPBin;
}
```

Método covertirPrefijoRedBinOctPorOct():

Código antes de refactorizar. -

```
// ***** PASO 2: CONVERTIR EL PREFIJO DE RED INGRESADO A BINARIO *****
// CONVERTIR PREFIJO DE RED A BINARIO OCTETO POR OCTETO (RETURN PASO 1 Y PASO DE OBTENER LA RED)
public String[] covertirPrefijoRedBinOctPorOct(int prefijo) {
    String[] octetosMasc = new String[4];
    String cad = "";
    String primerOctMasc;
    String segundoOctMasc;
    String tercerOctMasc;
    String cuartoOctMasc;
    int bitsAsumar;
    if (prefijo >= 8 && prefijo < 16) {
        bitsAsumar = prefijo - 8;
        primerOctMasc = convertirDecimalBinario(255);
        segundoOctMasc = convertirDecimalBinario(convertirPrefijoRedDecimal(bitsAsumar));
        tercerOctMasc = convertirDecimalBinario(0);
        cuartoOctMasc = convertirDecimalBinario(0);
        octetosMasc[0] = primerOctMasc;
        octetosMasc[1] = segundoOctMasc;
        octetosMasc[2] = tercerOctMasc;
        octetosMasc[3] = cuartoOctMasc;
    } else if (prefijo >= 16 && prefijo < 24) {
        bitsAsumar = prefijo - 16;
        primerOctMasc = convertirDecimalBinario(255);
        segundoOctMasc = convertirDecimalBinario(255);
        tercerOctMasc = convertirDecimalBinario(convertirPrefijoRedDecimal(bitsAsumar));
        cuartoOctMasc = convertirDecimalBinario(0);
        octetosMasc[0] = primerOctMasc;
        octetosMasc[1] = segundoOctMasc;
        octetosMasc[2] = tercerOctMasc;
        octetosMasc[3] = cuartoOctMasc;
    } else if (prefijo >= 24 && prefijo <= 32) {
        bitsAsumar = prefijo - 24;
        primerOctMasc = convertirDecimalBinario(255);
        segundoOctMasc = convertirDecimalBinario(255);
        tercerOctMasc = convertirDecimalBinario(255);
        cuartoOctMasc = convertirDecimalBinario(convertirPrefijoRedDecimal(bitsAsumar));
        octetosMasc[0] = primerOctMasc;
        octetosMasc[1] = segundoOctMasc;
        octetosMasc[2] = tercerOctMasc;
        octetosMasc[3] = cuartoOctMasc;
    }
    for (int i = 0; i < 4; i++) {
        if (i == 3) {
            cad = cad + octetosMasc[i];
        } else {
            cad = cad + octetosMasc[i] + ".";
        }
    }
    System.out.println(cad + " <--- MASCARABIN");
    // calcularRedBin(octetosMasc);

    return octetosMasc;
}
```

Código después de refactorizar. -

Para la refactorización de este método se decidió hacer uso de encenderBits() además de usar substring el cual ayudaría a eliminar los if's haciendo que el código reduzca notablemente.

```
// ***** PASO 2: CONVERTIR EL PREFIJO DE RED INGRESADO A BINARIO *****
// CONVERTIR PREFIJO DE RED A BINARIO OCTETO POR OCTETO (RETURN PASO 1 Y PASO DE OBTENER LA RED)
public String[] covertirPrefijoRedBinOctPorOct(int prefijo) {
    String[] octetosMasc = new String[4];

    String mascaraBin = "00000000000000000000000000000000";
    String res = encenderBits(prefijo, mascaraBin); // enciende bits
    octetosMasc[0] = res.substring(0, 8);
    octetosMasc[1] = res.substring(8, 16); // Dividir en octetos
    octetosMasc[2] = res.substring(16, 24);
    octetosMasc[3] = res.substring(24, 32);

    return octetosMasc;
}
```

25/05/2022.- Para este avance se continuó con la refactorización de los métodos de la clase “Calculadora Binaria”, el cual tuvo los siguientes cambios:

Se decidió eliminar los siguientes métodos que permitían convertir un número Decimal a Binario, `convertirPrefijoRedInvBinOctPorOct` para los cálculos del método `calcularBroadcast()` y `convertirPrefijoRedDecimal` que permite traducir los bits que se encienden a su valor decimal.

```
43 // CONVERTIR UN OCTETO EN FORMATO DECIMAL A BINARIO PARA LA MASCARA INVERTIDA
44 private String convertirDecimalBinarioInv(int octetoDecimal) {
45     String octetoBinario = "";
46     for (int i = 7; i >= 0; i--) { //Recorrer la tabla de derecha a izquierda
47         if (i != -1) {
48             if (octetoDecimal >= (int) Math.pow(2, i)) { // Comparar el octeto Decimal con la tabla
49                 octetoBinario = octetoBinario + 0; // Concatenar 1's a nuestra respuesta
50                 octetoDecimal = octetoDecimal - (int) Math.pow(2, i); // Actualizar el nuevo Valor
51             } else {
52                 octetoBinario = octetoBinario + 1; // Concatenar 0's a nuestra respuesta
53             }
54         }
55     }
56     return octetoBinario;
57 }

167 private int convertirPrefijoRedDecimal(int bitsAsumar) {
168     int prefijoDec = 0;
169     switch (bitsAsumar) {
170         case 0:
171             prefijoDec = 0;
172             break;
173         case 1:
174             prefijoDec = 128;
175             break;
176         case 2:
177             prefijoDec = 192;
178             break;
179         case 3:
180             prefijoDec = 224;
181             break;
182         case 4:
183             prefijoDec = 240;
184             break;
185         case 5:
186             prefijoDec = 248;
187             break;
188         case 6:
189             prefijoDec = 252;
190             break;
191         case 7:
192             prefijoDec = 254;
193             break;
194         case 8:
195             prefijoDec = 255;
196             break;
197         default:
198             break;
199     }
200     return prefijoDec;
201 }
```

```

121 public String[] convertirPrefijoRedInvBinOctPorOct(int prefijo) {
122     String[] octetosMasc = new String[4];
123     String primerOctMasc;
124     String segundoOctMasc;
125     String tercerOctMasc;
126     String cuartoOctMasc;
127     int bitsAsumar;
128     if (prefijo >= 8 && prefijo < 16) {
129         bitsAsumar = prefijo - 8;
130         primerOctMasc = convertirDecimalBinarioInv(255);
131         segundoOctMasc = convertirDecimalBinarioInv(convertirPrefijoRedDecimal(bitsAsumar));
132         tercerOctMasc = convertirDecimalBinarioInv(0);
133         cuartoOctMasc = convertirDecimalBinarioInv(0);
134         octetosMasc[0] = primerOctMasc;
135         octetosMasc[1] = segundoOctMasc;
136         octetosMasc[2] = tercerOctMasc;
137         octetosMasc[3] = cuartoOctMasc;
138
139     } else if (prefijo >= 16 && prefijo < 24) {
140         bitsAsumar = prefijo - 16;
141         primerOctMasc = convertirDecimalBinarioInv(255);
142         segundoOctMasc = convertirDecimalBinarioInv(255);
143         tercerOctMasc = convertirDecimalBinarioInv(convertirPrefijoRedDecimal(bitsAsumar));
144         cuartoOctMasc = convertirDecimalBinarioInv(0);
145         octetosMasc[0] = primerOctMasc;
146         octetosMasc[1] = segundoOctMasc;
147         octetosMasc[2] = tercerOctMasc;
148         octetosMasc[3] = cuartoOctMasc;
149
150     } else if (prefijo >= 24 && prefijo <= 32) {
151         bitsAsumar = prefijo - 24;
152         primerOctMasc = convertirDecimalBinarioInv(255);
153         segundoOctMasc = convertirDecimalBinarioInv(255);
154         tercerOctMasc = convertirDecimalBinarioInv(255);
155         cuartoOctMasc = convertirDecimalBinarioInv(convertirPrefijoRedDecimal(bitsAsumar));
156         octetosMasc[0] = primerOctMasc;
157         octetosMasc[1] = segundoOctMasc;
158         octetosMasc[2] = tercerOctMasc;
159         octetosMasc[3] = cuartoOctMasc;
160     }
161
162     return octetosMasc;
163 }

```


Se refactorizaron los siguientes métodos:

Código antes de refactorizar. –

```
204 public String[] calcularRedBin(int[] ip, int prefijo) {
205     String cad = "";
206     String[] octetosRed = new String[4];
207     String[] octetosIP = ipDecimalBinario(ip);
208     String[] octetosMasc = covertirPrefijoRedBinOctPorOct(prefijo);
209
210     int primerOctIPBin = Integer.valueOf(octetosIP[0], 2);
211     int segundoOctIPBin = Integer.valueOf(octetosIP[1], 2);
212     int tercerOctIPBin = Integer.valueOf(octetosIP[2], 2);
213     int cuartoOctIPBin = Integer.valueOf(octetosIP[3], 2);
214
215     int primerOctMascBin = Integer.valueOf(octetosMasc[0], 2);
216     int segundoOctMascBin = Integer.valueOf(octetosMasc[1], 2);
217     int tercerOctMascBin = Integer.valueOf(octetosMasc[2], 2);
218     int cuartoOctMascBin = Integer.valueOf(octetosMasc[3], 2);
219
220     int primerOctRed = primerOctIPBin & primerOctMascBin;
221     int segundoOctRed = segundoOctIPBin & segundoOctMascBin;
222     int tercerOctRed = tercerOctIPBin & tercerOctMascBin;
223     int cuartoOctRed = cuartoOctIPBin & cuartoOctMascBin;
224
225     octetosRed[0] = String.valueOf(primerOctRed);
226     octetosRed[1] = String.valueOf(segundoOctRed);
227     octetosRed[2] = String.valueOf(tercerOctRed);
228     octetosRed[3] = String.valueOf(cuartoOctRed);
229
230     return octetosRed;
231 }
```

Código después de refactorizar. –

Se decidió implementar un método adicional para obtener la red en Decimal(2º img)

```
104 public String[] calcularRedBin(int[] ipDec, int prefijo) {
105     String[] octetosRedBin = new String[4];
106     String redConcatenada = "";
107     String[] ipBin = ipDecimalBinario(ipDec); // Convertir IP Decimal a Binario
108     String ipBinConcatenada = ipBin[0] + ipBin[1] + ipBin[2] + ipBin[3]; // Concatenar ip Binario
109     String[] octetosMascBin = covertirPrefijoRedBinOctPorOct(prefijo); // Convertir Mascara de Red a Binario
110     String mascBinConcatenada = octetosMascBin[0] + octetosMascBin[1] +
111         octetosMascBin[2] + octetosMascBin[3]; // Concatenar mascara de Red Binaria
112     for (int i = 0; i < 32; i++) {
113         if (ipBinConcatenada.charAt(i) == '1' && mascBinConcatenada.charAt(i) == '1') { // Realiza la operacion logi
114             redConcatenada = redConcatenada + 1; // AND entre la IP y la masc
115         } else { // en Binario
116             redConcatenada = redConcatenada + 0;
117         }
118     }
119     octetosRedBin[0] = redConcatenada.substring(0, 8);
120     octetosRedBin[1] = redConcatenada.substring(8, 16); // Separar la Red Binaria
121     octetosRedBin[2] = redConcatenada.substring(16, 24); // en Octetos
122     octetosRedBin[3] = redConcatenada.substring(24, 32);
123
124     return octetosRedBin;
125 }
126 }
```

```

127 public String[] calcularRedDecimal(int[] ipDec, int prefijo) {
128     String[] octetosRedDec = new String[4];
129     String[] octetosRedBin = calcularRedBin(ipDec, prefijo); // Calcular la Red en Binario
130     octetosRedDec[0] = String.valueOf(convertirBinarioDecimal(octetosRedBin[0]));
131     octetosRedDec[1] = String.valueOf(convertirBinarioDecimal(octetosRedBin[1])); // Convertir y separar en
132     octetosRedDec[2] = String.valueOf(convertirBinarioDecimal(octetosRedBin[2])); // octetos la Red en Decimal
133     octetosRedDec[3] = String.valueOf(convertirBinarioDecimal(octetosRedBin[3]));
134
135     return octetosRedDec;
136 }

```

Código antes de refactorizar. –

```

233 // PASO 4: ENCONTRAR EL BROADCAST (DIRECCION DE RED OR MASCARA DE RED INVERTIDA);
234 public String[] calcularBroadcast(int[] ip, int prefijo) {
235
236     String[] octetosBroadCast = new String[4];
237     String[] octetosRed = calcularRedBin(ip, prefijo);
238     String octetosRedS = octetosRed[0] + octetosRed[1] + octetosRed[2] + octetosRed[3];
239     String cad = octetosRedS.substring(0, prefijo) + encenderBits(32 - prefijo, octetosRedS.substring(prefijo, 32));
240
241     // String[] octetosMascInvertida = covertirPrefijoRedInvBinOctPorOct(prefijo);
242
243     octetosBroadCast[0] = String.valueOf(convertirBinarioDecimal(cad.substring(0, 8)));
244     octetosBroadCast[1] = String.valueOf(convertirBinarioDecimal(cad.substring(8, 16)));
245     octetosBroadCast[2] = String.valueOf(convertirBinarioDecimal(cad.substring(16, 24)));
246     octetosBroadCast[3] = String.valueOf(convertirBinarioDecimal(cad.substring(24, 32)));
247
248
249     return octetosBroadCast;
250 }
251
252 )
253

```

Código después de refactorizar. -

Se decidió implementar un método adicional para obtener el Broadcast en Decimal (2ª imagen)

```

138 // PASO 4: ENCONTRAR EL BROADCAST (DIRECCION DE RED OR MASCARA DE RED INVERTIDA);
139 public String[] calcularBroadcastBin(int[] ipDec, int prefijo) {
140     String[] octetosBroadCast = new String[4];
141     String[] octetosRedBin = calcularRedBin(ipDec, prefijo); // Calcular la Red en Binario
142     String redBinConcatenada = octetosRedBin[0] + octetosRedBin[1] + octetosRedBin[2] + octetosRedBin[3]; // Concaten
143     String broadCastBin = redBinConcatenada.substring(0, prefijo) + encenderBits(32 - prefijo,
144     redBinConcatenada.substring(prefijo, 32)); //
145     // Concatena la red Binaria manteniendo los bits que pertenecen a la Red y enciende (1's) bits pertenecientes a l
146     octetosBroadCast[0] = broadCastBin.substring(0, 8);
147     octetosBroadCast[1] = broadCastBin.substring(8, 16); // Separar el Broadcast Binario
148     octetosBroadCast[2] = broadCastBin.substring(16, 24); // en Octetos
149     octetosBroadCast[3] = broadCastBin.substring(24, 32);
150
151     return octetosBroadCast;
152 }
153
154 public String[] calcularBroadcastDec(int[] ipDec, int prefijo) {
155     String[] octetosBroadCastDec = new String[4];
156     String[] octetosBroadCastBin = calcularBroadcastBin(ipDec, prefijo); //Calcular Broadcast en Binario
157     String broadCastBinConcatenado = octetosBroadCastBin[0] + octetosBroadCastBin[1] + octetosBroadCastBin[2]
158     + octetosBroadCastBin[3]; // Concatenar Broadcast en Binario
159     String broadCastBin = broadCastBinConcatenado.substring(0, prefijo) + encenderBits(32 - prefijo,
160     broadCastBinConcatenado.substring(prefijo, 32));
161     // Concatena la red Binaria manteniendo los bits que pertenecen a la Red y enciende (1's) bits pertenecientes a l
162     octetosBroadCastDec[0] = String.valueOf(convertirBinarioDecimal(broadCastBin.substring(0, 8)));
163     octetosBroadCastDec[1] = String.valueOf(convertirBinarioDecimal(broadCastBin.substring(8, 16))); // Separar en o
164     octetosBroadCastDec[2] = String.valueOf(convertirBinarioDecimal(broadCastBin.substring(16, 24))); // y convertir
165     octetosBroadCastDec[3] = String.valueOf(convertirBinarioDecimal(broadCastBin.substring(24, 32))); // broadcast a
166
167     return octetosBroadCastDec;
168 }

```

26/05/2022. - Para este avance se empezó a añadir comentarios a la clase “Interfaz” con el fin de una mejor comprensión del código para el programador, además se realizó la refactorización de los siguientes métodos.

Código antes de refactorizar. –

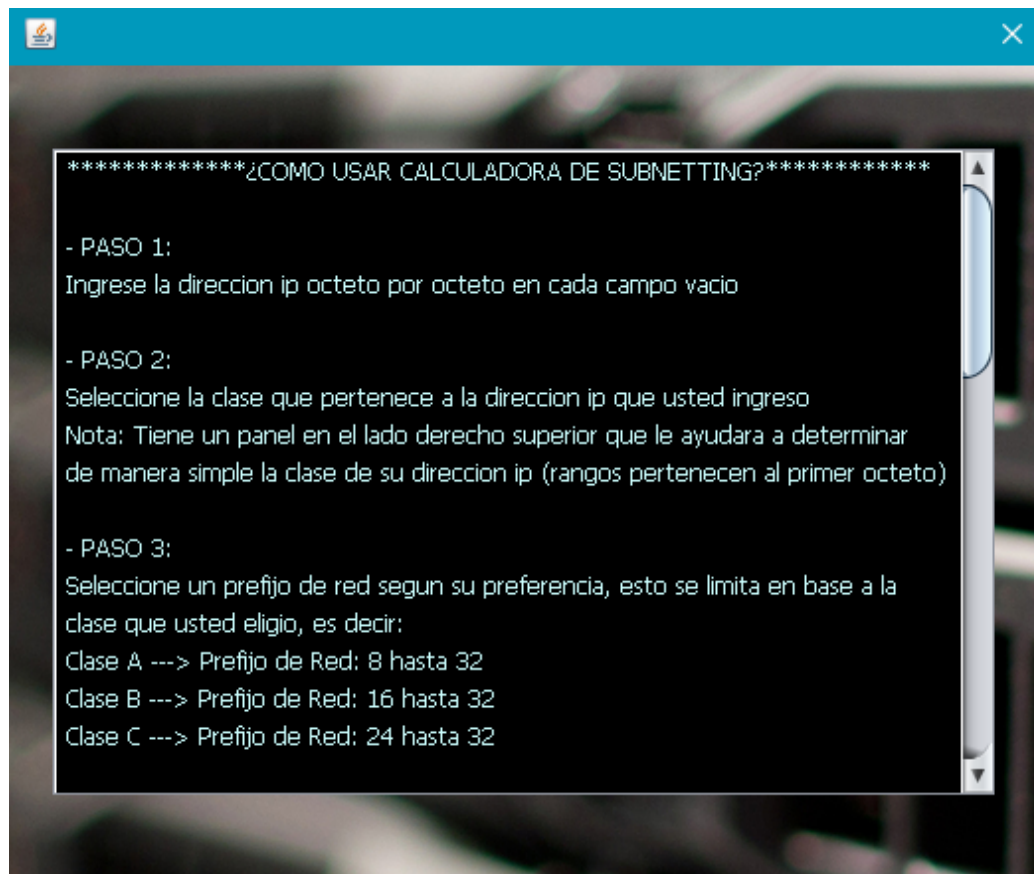
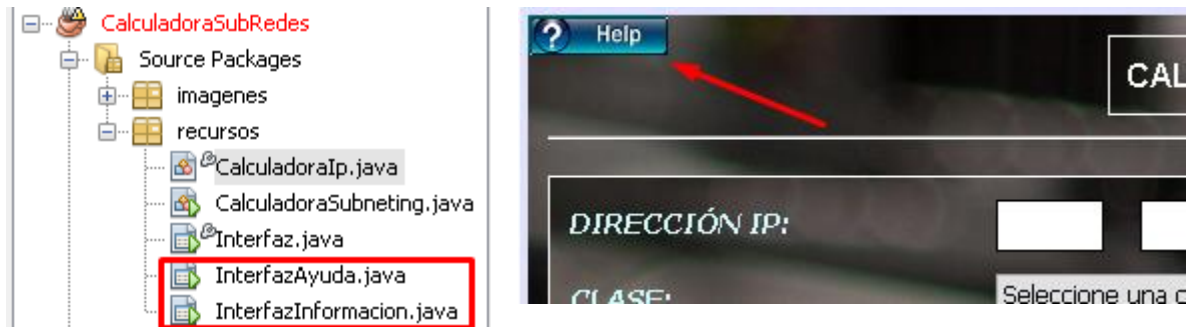
Switch contaba con 32 casos.

```
320 // OBTENER LA CANTIDAD DE SUBREDES DISPONIBLES EN BASE A LA MASCARA (PREFIXO DE RED) QUE SE SELECCIONE
321 public String[] getCantSubRedes(String mascara) {
322     String[] cantSubredes = null;
323
324     switch (Integer.parseInt(mascara)) {
325         case 8:
326             cantSubredes = new String[26];
327             cantSubredes[0] = "Seleccione la cantidad de subredes";
328             for (int i = 0; i <= 24; i++) {
329                 cantSubredes[i + 1] = String.valueOf((int) Math.pow(2, i));
330             }
331             break;
332         case 9:
333             cantSubredes = new String[25];
334             cantSubredes[0] = "Seleccione la cantidad de subredes";
335             for (int i = 0; i <= 23; i++) {
336                 cantSubredes[i + 1] = String.valueOf((int) Math.pow(2, i));
337             }
338             break;
339         case 10:
340             cantSubredes = new String[24];
341             cantSubredes[0] = "Seleccione la cantidad de subredes";
342             for (int i = 0; i <= 22; i++) {
343                 cantSubredes[i + 1] = String.valueOf((int) Math.pow(2, i));
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

Código después de refactorizar. –

```
302 // OBTENER LA CANTIDAD DE SUBREDES DISPONIBLES EN BASE A LA MASCARA (PREFIXO DE RED) QUE SE SELECCIONE
303 public String[] getCantSubRedes(String prefijo) {
304     String[] cantSubredes = null;
305     int prefijoEntero = Integer.parseInt(prefijo);
306     int tamListaSubredes = 32 - prefijoEntero; // Define el tamaño de la lista de subredes en base al prefijo de red
307     cantSubredes = new String[tamListaSubredes + 2]; // Asigna el tamaño al arreglo
308     cantSubredes[0] = "Seleccione la cantidad de subredes";
309     for (int i = 0; i <= tamListaSubredes; i++) { // Genera la lista de subredes dependiendo del prefijo de red
310         cantSubredes[i + 1] = String.valueOf((int) Math.pow(2, i)); // Añade cada una de las subredes creadas al arreglo
311     }
312
313     return cantSubredes;
314 }
```

Como último avance, se agregó el botón ayuda y su ventana correspondiente de información de ayuda para después mejorar la estética de nuestras interfaces.



La forma de trabajo

Se empezó el proyecto definiéndose un horario el cual debíamos seguir durante el tiempo que estaríamos trabajando

HORARIO DE REUNIONES

LUNES	MARTES	MIERCOLES	JUEVES	VIERNES
17:15 pm	18:45 pm	17:15 pm	16:00 pm	18:45 pm

Dedicamos los primeros días a estudiar para entender el tema y definir lo que se quería lograr con el proyecto, para ello se recurrió a diferentes fuentes de información como páginas web y videos entre los cuales se puede resaltar el canal de youtube “Eliezer de León” y las páginas web de ejemplo que se nos proporcionó:

- <https://youtu.be/rg8RwcQyPfs>
- https://youtu.be/sLWYpqjT0_Y
- https://www.iptp.net/es_ES/iptp-tools/ip-calculator/
- <https://www.calculadora-redes.com/>

Como guía de los videos se obtuvieron 5 pasos para obtener los datos necesarios, los cuales fueron implementados a el proyecto y pasaron a ser parte del algoritmo.

192.168.1.0 /24 4 subredes

1. [Identificar la mascara actual]: 11111111.11111111.11111111.00000000
255.255.255.0

2. [Resolver la fórmula]: $2^N \geq 4$; $2^2 = 4$

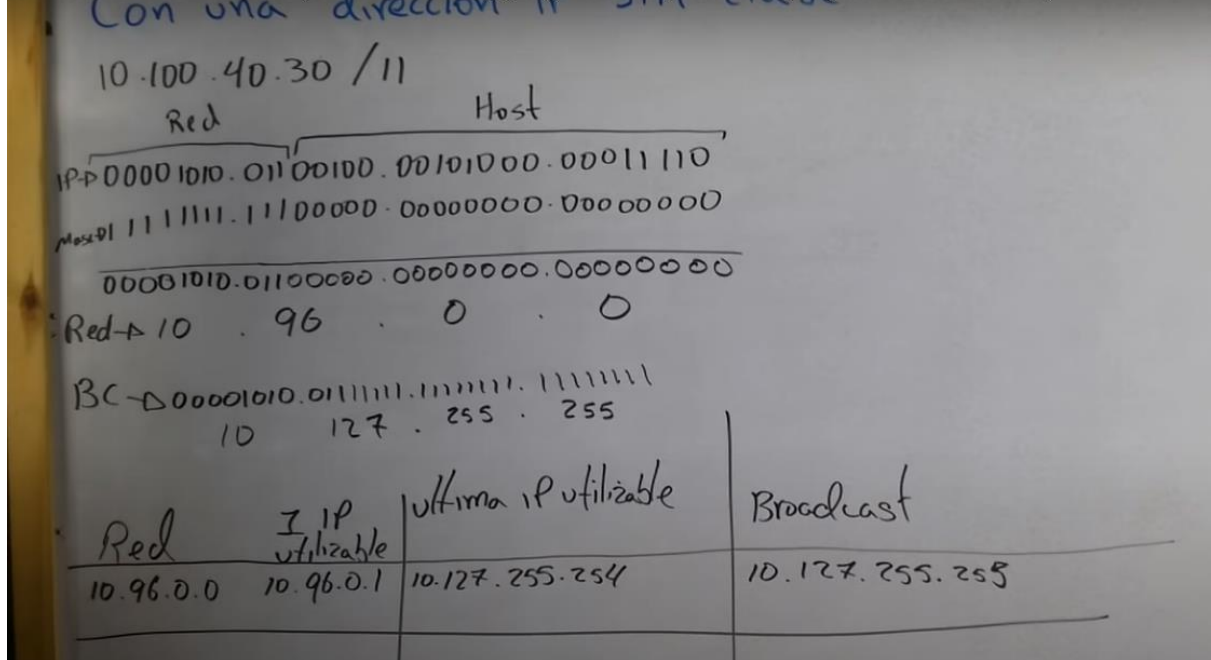
3. [Obtener nueva mascara]: 11111111.11111111.11111111.11000000
255.255.255.192

4. [Host por subred]: $2^m - 2 = H$; $2^6 - 2 = 62$

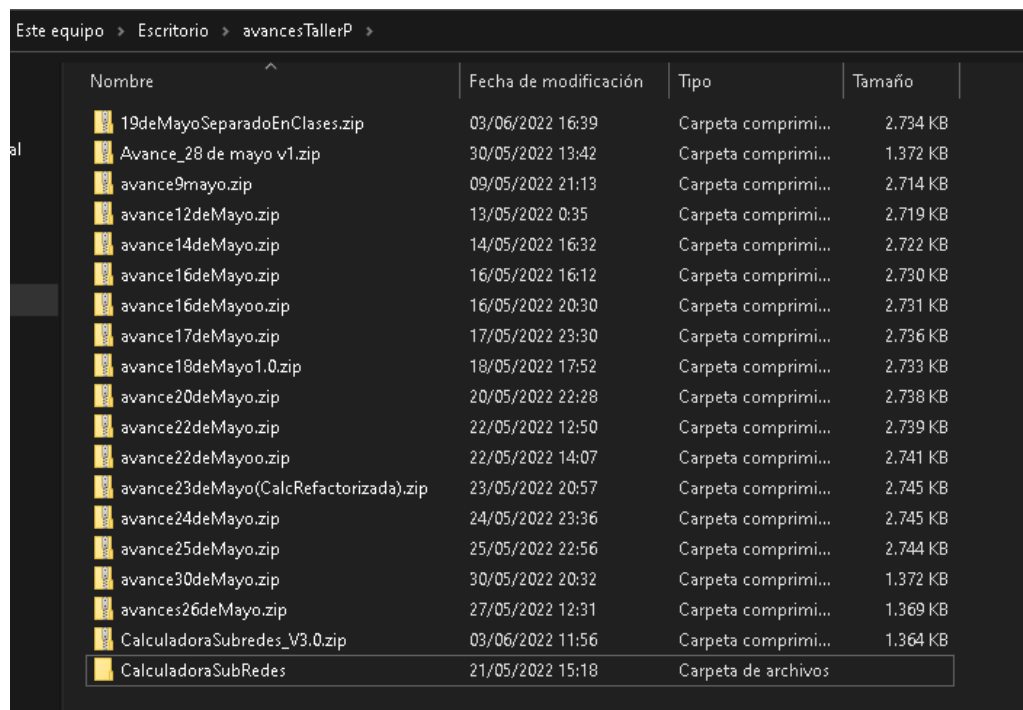
5. [Salto de Red]: $256 - 192 = 64$

NO	Subred	Primera IP utilizable	ultima IP utilizable	Broadcast
1.	192.168.1.0	192.168.1.1	192.168.1.62	192.168.1.63
2.	192.168.1.64	192.168.1.65	192.168.1.126	192.168.1.127
3.	192.168.1.128	192.168.1.129	192.168.1.190	192.168.1.191
4.	192.168.1.192	192.168.1.193	192.168.1.254	192.168.1.255

SUBNETEO DE REDES [CAPITULO VII] {Logica AND, Obtener propiedades de Red}



A medida que se fue obteniendo los avances estos fueron almacenados como diferentes sub versiones que contienen pequeños avances de una a otra los cuales se pueden observar en la siguiente imagen:



Los puntos más relevantes del programa

Líneas 428 - 431: Se implementaron para evitar que se muestre la información en la interfaz mientras no se llenen y seleccionen todos los campos.

Líneas 432-436: Se implementó para verificar si la dirección ip concuerda con el la clase que se seleccionó.

Línea 440: Se implementó para abrir la ventana de “información”.

Líneas 443-445: Introducimos la ip Decimal en el campo correspondiente de la interfaz llamando al método de la clase CalculadoraIp que nos permitirá obtener la ip en Decimal.

```
418 // ACCIONES QUE SE REALIZARAN DESPUES DE PRESIONAR EL BOTON
419 // El try-catch se uso para controlar alguna excepcion tipo : NumberFormatException al momento que presionar el boton "Generar"
420 private void btnGenerarActionPerformed(java.awt.event.ActionEvent evt) {
421
422     try {
423         CalculadoraSubneting calculadora = new CalculadoraSubneting(this);
424         CalculadoraIp calcBin = new CalculadoraIp();
425         int[] ipDecimal = {Integer.parseInt(primerOcteto.getText()), Integer.parseInt(segundoOcteto.getText()),
426                             Integer.parseInt(tercerOcteto.getText()), Integer.parseInt(cuartoOcteto.getText())};
427         //Obtiene la direccion ip ingresada y lo guarda en un arreglo
428         if (!cbClase.getSelectedItem().equals("Seleccione una clase")) { //Mientras no se seleccione nada no se despliega la inter
429             if (!cbPrefijo.getSelectedItem().equals("Seleccione un Prefijo de Red")) {
430                 if (!cbCantSubredes.getSelectedItem().equals("Seleccione la cantidad de subredes")) { } {
431
432                     if (!calcBin.determinarClase(ipDecimal).equals(getCbClase())) { //Verifica que la clase seleccionada correspo
433                         JOptionPane.showMessageDialog(null, "No concuerda la Direccion IP ingresada con la Clase seleccionada");
434                         cbClase.setSelectedIndex(0); // Las listas vuelven a sus valores por defecto
435                         cbPrefijo.setSelectedIndex(0);
436                         cbCantSubredes.setSelectedIndex(0);
437                     } else if (calcBin.determinarClase(ipDecimal).equals(getCbClase())) { // Si todo es correcto la informacion
438
439                         // ABRIR VENTANA INFORMACION
440                         informacion.setVisible(true);
441
442                         // INTRODUCIR VALORES EN CAMPO IP BINARIO
443                         String[] ipBinario = calcBin.ipDecimalBinario(ipDecimal);
444                         informacion.campoIpBin.setText(ipBinario[0] + " . " + ipBinario[1] + " . "
445                                                         + ipBinario[2] + " . " + ipBinario[3]);
```

Líneas 448-453: Introducimos la red Decimal y la primera ip utilizable en sus campos correspondientes de la interfaz llamando a los métodos de la clase CalculadoraIp que nos permitirá obtener la red en Decimal y a partir de ello, deducir la primer ip utilizable.

Líneas 456-458: Introducimos la Red Binario en el campo correspondiente de la interfaz llamando al método de la clase CalculadoraIP que nos permitirá obtener dicho valor.

Líneas 461-466: Introducimos el Broadcast Decimal y la última ip utilizable en sus campos correspondientes de la interfaz llamando a los métodos de la clase CalculadoraIp que nos permitirá obtener el Broadcast en Decimal y a partir de ello, deducir la última ip utilizable.

Líneas 469-471: Introducimos la Broadcast Binario en el campo correspondiente de la interfaz llamando al método de la clase CalculadoraIP que nos permitirá obtener dicho valor.

Líneas 474-475: Introducimos la máscara de Red en Decimal en el campo correspondiente de la interfaz llamando al método de la clase CalculadoraSubnetting que nos permitirá obtener dicho valor.

```

447 // INTRODUCIR VALORES EN CAMPO RED Y PRIMERA IP UTILIZABLE
448 String[] redDecimal = calcBin.calcularRedDecimal(ipDecimal, getCbPrefijo());
449 int primeraIpU = Integer.parseInt(redDecimal[3]) + 1;
450 informacion.campoRed.setText(redDecimal[0] + "." + redDecimal[1] + "." +
451     + redDecimal[2] + "." + redDecimal[3]);
452 informacion.campoPrimeraIpU.setText(redDecimal[0] + "." + redDecimal[1] + "." +
453     + redDecimal[2] + "." + String.valueOf(primeraIpU));
454
455 // INTRODUCIR VALORES EN CAMPO RED BINARIO
456 String[] redBinaria = calcBin.calcularRedBin(ipDecimal, getCbPrefijo());
457 informacion.campoRedBin.setText(redBinaria[0] + "." + redBinaria[1] + "." +
458     + redBinaria[2] + "." + redBinaria[3]);
459
460 // INTRODUCIR VALORES EN CAMPO BROADCAST Y ULTIMA IP UTILIZABLE
461 String[] broadcastDecimal = calcBin.calcularBroadcastDec(ipDecimal, getCbPrefijo());
462 int ultimaIpU = Integer.parseInt(broadcastDecimal[3]) - 1;
463 informacion.campoBroadcast.setText(broadcastDecimal[0] + "." + broadcastDecimal[1] + "." +
464     + broadcastDecimal[2] + "." + broadcastDecimal[3]);
465 informacion.campoUltimaIpU.setText(broadcastDecimal[0] + "." + broadcastDecimal[1] + "." +
466     + broadcastDecimal[2] + "." + String.valueOf(ultimaIpU));
467
468 // INTRODUCIR VALORES EN CAMPO BROADCAST BINARIO
469 String[] broadcastBin = calcBin.calcularBroadcastBin(ipDecimal, getCbPrefijo());
470 informacion.campoBroadcastBin.setText(broadcastBin[0] + "." + broadcastBin[1] + "." +
471     + broadcastBin[2] + "." + broadcastBin[3]);
472
473 //INTRODUCIR VALORES EN CAMPO MÁSCARA DE RED
474 String[] mascaraRedDecimal = calculadora.getMascaraDecimal();
475 informacion.campoMascaraRed.setText(mascaraRedDecimal[0] + "." + mascaraRedDecimal[1] + "." +

```

Líneas 479-481: Introducimos la Máscara de Red en su campo correspondiente de la interfaz llamando al método de la clase CalculadoraIp para obtener dicho valor.

Líneas 484-485: Introducimos la clase de la dirección ip en su campo respectivo de la interfaz.

Líneas 488-489: Introducimos la máscara de subred Decimal en su campo respectivo de la interfaz.

Líneas 492-495: Introducimos la máscara de subred Binario en su campo respectivo de la interfaz.

Líneas 498-499: Introducimos los hosts por subred en su campo respectivo de la interfaz.

Líneas 502-517: Introducimos el salto de red y deducimos en qué octeto se realiza el salto.


```

478 // INTRODUCIR VALORES EN CAMPO MASCARA DE RED BIN
479 String[] mascaraRedBin = calcBin.convertirPrefijoRedBinOctPorOct(getCbPrefijo());
480 informacion.campoMascaraRedBin.setText(mascaraRedBin[0] + " . " + mascaraRedBin[1] + " . "
481 + mascaraRedBin[2] + " . " + mascaraRedBin[3]);
482
483 //INTRODUCIR VALORES EN CAMPO CLASE
484 String clase = calcBin.determinarClase(ipDecimal);
485 informacion.campoClase.setText(clase);
486
487 //INTRODUCIR VALORES CAMPO MASCARA SUBRED
488 String mascaraSubred = calculadora.getSubmascaraDecimal();
489 informacion.campoMascaraSubred.setText(mascaraSubred);
490
491 // INTRODUCIR VALORES EN CAMPO MASCARA SUBRED BIN
492 String mascaraSubredBin = calculadora.getSubmascaraBinario();
493 String MascaraSubRedBin =mascaraSubredBin.substring(0,8)+" . "+mascaraSubredBin.substring(8,16)+" . "
494 +mascaraSubredBin.substring(16,24)+" . "+mascaraSubredBin.substring(24,32);
495 informacion.campoMascaraSubredBin.setText(MascaraSubRedBin);
496
497 //INTRODUCIR VALORES EN CAMPO HOST POR SUBRED
498 int hostPorSubred = calculadora.getHostPorSubred();
499 informacion.campoHostPSubred.setText(String.valueOf(hostPorSubred + " Hosts por subred"));
500
501 //INTRODUCIR VALORES EN CAMPO SALTO DE RED
502 int saltoDeRed = calculadora.getSaltoDeRed();
503 int nuevaPotenciaSubmasc = calculadora.getNuevaPotenciaParaSubmasc();
504 if (nuevaPotenciaSubmasc >= 8 && nuevaPotenciaSubmasc <= 16) //Determina en que octeto se realiza el salto
505     informacion.campoSaltoRed.setText("De " + saltoDeRed + " en " + saltoDeRed
506 + " en el Segundo Octeto");
507 } else if (nuevaPotenciaSubmasc >= 17 && nuevaPotenciaSubmasc <= 24) {
508     informacion.campoSaltoRed.setText("De " + saltoDeRed + " en " + saltoDeRed
509 + " en el Tercer Octeto");
510 } else if (nuevaPotenciaSubmasc >= 25 && nuevaPotenciaSubmasc <= 32) {
511     informacion.campoSaltoRed.setText("De " + saltoDeRed + " en " + saltoDeRed
512 + " en el Cuarto Octeto");
513 }
514 }
515 }
516 }
517 }
518 } catch (NumberFormatException ex) {
519     System.out.println("Error" + ex);
520 }
521 }
522 }

```

Línea 71-74: Se accede a las posiciones de la variable octetosBin y llamamos al método `convertirDecimalBinario` el cual convertirá la ip que se encuentra en decimal a binario.

```

66 // ***** PASO 1: CONVERTIR LA DIRECCION IP INGRESADA A BINARIO *****
67 // CONVERTIR LA IP A BINARIO OCTETO POR OCTETO (RETURN METODO determinarClase())
68 public String[] ipDecimalBinario(int[] ipDecimal) {
69     String[] octetosIPBin = new String[4];
70
71     octetosIPBin[0] = convertirDecimalBinario(ipDecimal[0]);
72     octetosIPBin[1] = convertirDecimalBinario(ipDecimal[1]); // Convertir ip Decimal a
73     octetosIPBin[2] = convertirDecimalBinario(ipDecimal[2]); // Binario Octeto por Octeto
74     octetosIPBin[3] = convertirDecimalBinario(ipDecimal[3]);
75
76     return octetosIPBin;
77 }

```

Líneas 84-85: Se almacena una cadena de 32 ceros en la variable `mascaraBin` ya que nos ayudará como parámetro para el método `encenderBits` seguido llamamos al método `encenderBits` el cual convertirá los ceros en unos dependiendo el prefijo que se tenga.

Líneas 86-89: En cada posición de `octetosMasc` hacemos un `substring` el cual permitirá dividir los octetos en los rangos (0-8) (8-16) (16-24) (24-32).

```

79 // ***** PASO 2: CONVERTIR EL PREFIJO DE RED INGRESADO A BINARIO *****
80 // CONVERTIR PREFIJO DE RED A BINARIO OCTETO POR OCTETO
81 public String[] convertirPrefijoRedBinOctPorOct(int prefijo) {
82     String[] octetosMasc = new String[4];
83
84     String mascaraBin = "00000000000000000000000000000000";
85     String res = encenderBits(prefijo, mascaraBin); // enciende bits
86     octetosMasc[0] = res.substring(0, 8);
87     octetosMasc[1] = res.substring(8, 16); // Dividir en octetos
88     octetosMasc[2] = res.substring(16, 24);
89     octetosMasc[3] = res.substring(24, 32);
90
91     return octetosMasc;
92 }

```

Línea 106: Se obtiene la ip en formato binario haciendo un llamado al método ipDecimalBinario.

Línea 108: Se obtiene la máscara de red en formato binario.

Líneas 111-117: Se realiza la operación lógica AND entre estos 2 datos.

Línea 123: Se retorna la variable que contiene la red en binario.

```
102 // ***** PASO 3: ENCONTRAR LA DIRECCION DE RED EN DECIMAL (DIRECCION IP AND MASCARA DE RED) *****
103 public String[] calcularRedBin(int[] ipDec, int prefijo) {
104     String[] octetosRedBin = new String[4];
105     String redConcatenada = "";
106     String[] ipBin = ipDecimalBinario(ipDec); // Convertir IP Decimal a Binario
107     String ipBinConcatenada = ipBin[0] + ipBin[1] + ipBin[2] + ipBin[3]; // Concatenar ip Binario
108     String[] octetosMascBin = convertirPrefijoRedBinOctPorOct(prefijo); // Convertir Mascara de Red a Binario
109     String mascBinConcatenada = octetosMascBin[0] + octetosMascBin[1]
110         + octetosMascBin[2] + octetosMascBin[3]; // Concatenar mascara de Red Binaria
111     for (int i = 0; i < 32; i++) {
112         if (ipBinConcatenada.charAt(i) == '1' && mascBinConcatenada.charAt(i) == '1') { // Realiza la operacion logica
113             redConcatenada = redConcatenada + 1; // AND entre la IP y la mascara
114         } else { // en Binario
115             redConcatenada = redConcatenada + 0;
116         }
117     }
118     octetosRedBin[0] = redConcatenada.substring(0, 8);
119     octetosRedBin[1] = redConcatenada.substring(8, 16); // Separar la Red Binaria
120     octetosRedBin[2] = redConcatenada.substring(16, 24); // en Octetos
121     octetosRedBin[3] = redConcatenada.substring(24, 32);
122
123     return octetosRedBin;
124 }
```

Línea 128: Llamamos al método calcularRedBin para obtener la red en binario.

Línea 130-133: Se convierten los datos de binario a decimal con el método convertirBinarioDecimal.

```
126 public String[] calcularRedDecimal(int[] ipDec, int prefijo) {
127     String[] octetosRedDec = new String[4];
128     String[] octetosRedBin = calcularRedBin(ipDec, prefijo); // Calcular la Red en Binario
129     octetosRedDec[0] = String.valueOf(convertirBinarioDecimal(octetosRedBin[0]));
130     octetosRedDec[1] = String.valueOf(convertirBinarioDecimal(octetosRedBin[1])); // Convertir y separar en
131     octetosRedDec[2] = String.valueOf(convertirBinarioDecimal(octetosRedBin[2])); // octetos la Red en Decimal
132     octetosRedDec[3] = String.valueOf(convertirBinarioDecimal(octetosRedBin[3]));
133
134     return octetosRedDec;
135 }
```

Línea 140: Se llama al método calcularRedBin para obtener la red en binario.

Línea 141: Se almacena el valor de la red en un String para agilizar su utilidad.

Línea 142: Se almacena un String con las posiciones correspondientes a la red y se concatena con el resto de los valores con bits encendidos (osea 1's).

```
137 // PASO 4: ENCONTRAR EL BROADCAST;
138 public String[] calcularBroadCastBin(int[] ipDec, int prefijo) {
139     String[] octetosBroadCast = new String[4];
140     String[] octetosRedBin = calcularRedBin(ipDec, prefijo); // Calcular la Red en Binario
141     String redBinConcatenada = octetosRedBin[0] + octetosRedBin[1] + octetosRedBin[2] + octetosRedBin[3]; // Concatenar la R
142     String broadCastBin = redBinConcatenada.substring(0, prefijo) + encenderBits(32 - prefijo,
143         redBinConcatenada.substring(prefijo, 32)); //
144     // Concatena la red Binaria manteniendo los bits que pertenecen a la Red y enciende (1's) bits pertenecientes a los host
145     octetosBroadCast[0] = broadCastBin.substring(0, 8);
146     octetosBroadCast[1] = broadCastBin.substring(8, 16); // Separar el BroadCast Binario
147     octetosBroadCast[2] = broadCastBin.substring(16, 24); // en Octetos
148     octetosBroadCast[3] = broadCastBin.substring(24, 32);
149
150     return octetosBroadCast;
151 }
```

Línea 155: Se llama al método para calcular broadcast en binario pasando como parámetro la ip en formato decimal y el prefijo de red.

Línea 156: Se almacena el broadcast en una variable String para agilizar su utilidad.

Línea 157: Se almacena un String con las posiciones correspondientes a la máscara y se concatena con el resto de los valores con bits encendidos (osea 1's).

```
153 public String[] calcularBroadcastDec(int[] ipDec, int prefijo) {
154     String[] octetosBroadCastDec = new String[4];
155     String[] octetosBroadCastBin = calcularBroadCastBin(ipDec, prefijo); //Calcular BroadCast en Binario
156     String broadcastBinConcatenado = octetosBroadCastBin[0] + octetosBroadCastBin[1] + octetosBroadCastBin[2] + octetosBroadCastBin[3]; // Concat
157     String broadcastBin = broadcastBinConcatenado.substring(0, prefijo) + encenderBits(32 - prefijo,
158         broadcastBinConcatenado.substring(prefijo, 32));
159     // Concatena la red Binaria manteniendo los bits que pertenecen a la Red y enciende (1's) bits pertenecientes a los hosts
160     octetosBroadCastDec[0] = String.valueOf(convertirBinarioDecimal(broadcastBin.substring(0, 8)));
161     octetosBroadCastDec[1] = String.valueOf(convertirBinarioDecimal(broadcastBin.substring(8, 16))); // Separar en octetos
162     octetosBroadCastDec[2] = String.valueOf(convertirBinarioDecimal(broadcastBin.substring(16, 24))); // y convertir
163     octetosBroadCastDec[3] = String.valueOf(convertirBinarioDecimal(broadcastBin.substring(24, 32))); // broadcast a Decimal
164
165     return octetosBroadCastDec;
166 }
```

COMBOBOX ANIDADOS

Línea 517: se verifica que el evento ha sido seleccionado.

Línea 518: se evade el elemento de la posición 0 en nuestro cb-Box el cual solo es una descripción de los datos a seleccionar.

Línea 519-520: tomando en cuenta el valor seleccionado se limita la cantidad de valores que el “cb-Box cantSubRedes” tendrá almacenado.

```
514 // COMBOBOX ANIDADADO A CANTIDAD DE SUBREDES (PADRE = MASCARA, HIJO = CANTIDAD DE SUBREDES)
515 private void cbPrefijoItemStateChanged(java.awt.event.ItemEvent evt) {
516     //controla los eventos del combo box prefijo
517     if (evt.getStateChange() == ItemEvent.SELECTED) {
518         if (this.cbPrefijo.getSelectedIndex() > 0) {
519             this.cbCantSubRedes.setModel(new DefaultComboBoxModel(
520                 getCantSubRedes(this.cbPrefijo.getSelectedItem().toString())));
521         }
522     }
523 }
524 }
```

Línea 506: se verifica que el evento ha sido seleccionado

Línea 507: se evade el elemento de la posición 0 en nuestro cb-Box el cual solo es una descripción de los datos a seleccionar

Línea 508: tomando en cuenta el valor seleccionado se limita la cantidad de valores que el “cb-Box Prefijo” tendrá almacenado

```
503 //COMBOBOX ANIDADADO A MASCARA (PADRE = CLASE, HIJO = MASCARA)
504 private void cbClaseItemStateChanged(java.awt.event.ItemEvent evt) {
505     //controla los eventos del combo box clase
506     if (evt.getStateChange() == ItemEvent.SELECTED) {
507         if (this.cbClase.getSelectedIndex() > 0) {
508             this.cbPrefijo.setModel(new DefaultComboBoxModel(getPrefijo(this.cbClase.getSelectedItem().toString())));
509         }
510     }
511 }
512 }
```

Mostrar el funcionamiento del programa

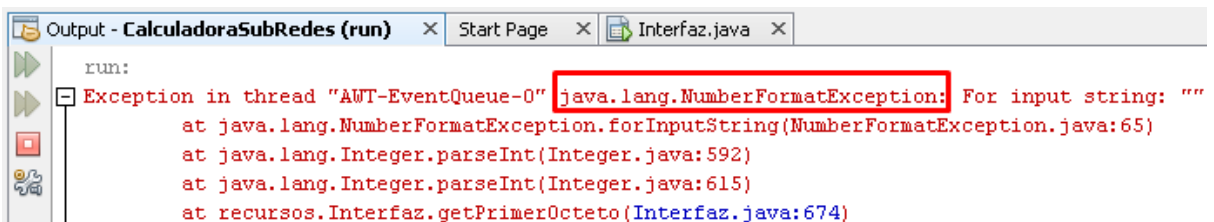
Para una mejor comprensión de este punto se decidió grabar el siguiente video:

<https://drive.google.com/file/d/188WwUi0wT7R3a-h-MdPD5ge7Bgzd1C9v/view>

Uso de control de errores (try-catch)

Se tienen las siguientes excepciones que posteriormente corregimos con el uso de try catch:

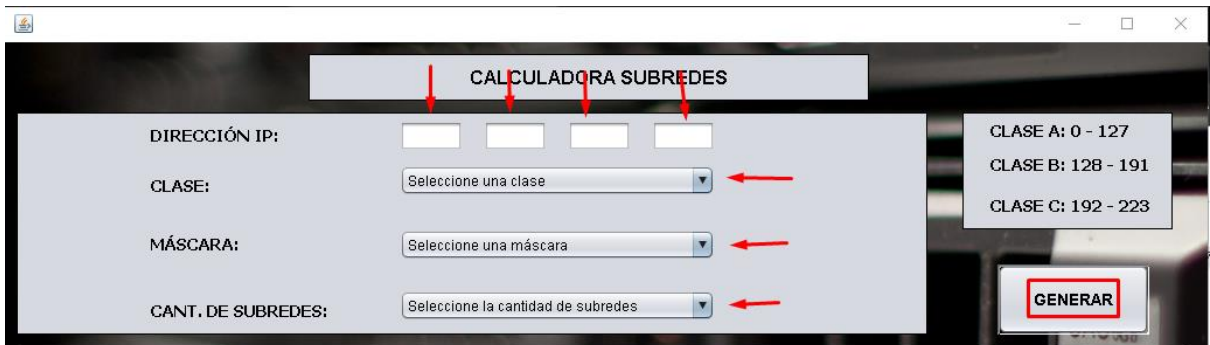
Se generaba la excepción “`java.lang.NumberFormatException`”: For input string: “” para los siguientes casos:



The screenshot shows an IDE window titled "Output - CalculadoraSubRedes (run)". The output text is as follows:

```
run:
Exception in thread "AWT-EventQueue-0" java.lang.NumberFormatException: For input string: ""
    at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
    at java.lang.Integer.parseInt(Integer.java:592)
    at java.lang.Integer.parseInt(Integer.java:615)
    at recursos.Interfaz.getPrimerOcteto(Interfaz.java:674)
```

- Dejar todos los campos y combobox vacíos y presionar el botón “Generar”



The screenshot shows the "CALCULADORA SUBREDES" application window. The fields are: "DIRECCIÓN IP:" (four empty text boxes), "CLASE:" (a dropdown menu with "Seleccione una clase"), "MÁSCARA:" (a dropdown menu with "Seleccione una máscara"), and "CANT. DE SUBREDES:" (a dropdown menu with "Seleccione la cantidad de subredes"). To the right, there is a box listing "CLASE A: 0 - 127", "CLASE B: 128 - 191", and "CLASE C: 192 - 223". At the bottom right is a "GENERAR" button. Red arrows point to each of the four input fields and the "GENERAR" button.

- Dejar todos los campos, combobox vacíos excepto el primer Octeto de la dirección Ip para después presionar el botón “Generar”



The screenshot shows the "CALCULADORA SUBREDES" application window. The "DIRECCIÓN IP:" field now has the first octet filled with "10". The other fields ("CLASE:", "MÁSCARA:", "CANT. DE SUBREDES:") and the "GENERAR" button remain the same as in the previous screenshot. Red arrows point to the first octet of the IP address and the "GENERAR" button.

- Dejar todos los campos, combobox vacíos excepto el Primer y Segundo Octeto de la dirección Ip para después presionar el botón “Generar”

- Dejar todos los campos, combo box vacíos excepto el Primer, Segundo y Tercer Octeto de la dirección Ip para después presionar el botón “Generar”

Se generaba excepción “`java.lang.NumberFormatException: For input string: “30abc”`” para el siguiente caso:

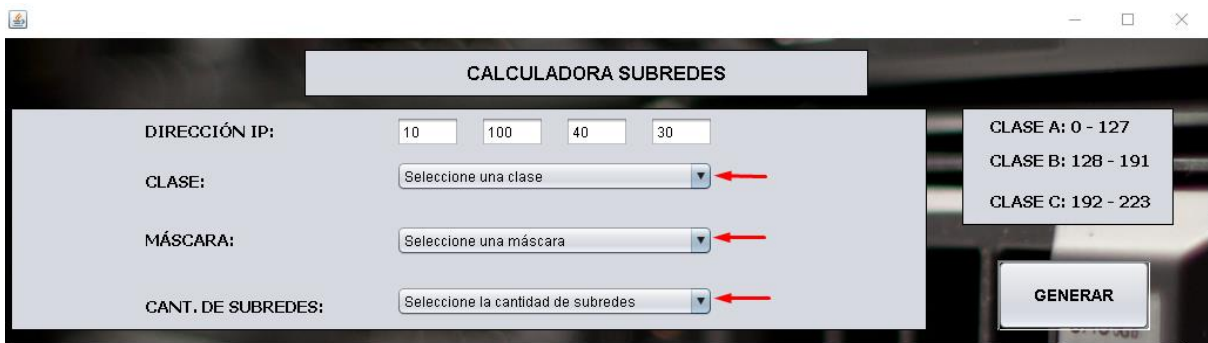
```
Output - CalculadoraSubRedes {run} x Start Page x Interfaz.java x
run:
Exception in thread "AWT-EventQueue-0" java.lang.NumberFormatException: For input string: "30abc"
    at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
```

- Introducir letras en cualquiera de los campos de la dirección IP y presionar el botón “Generar”

Se generaba la excepción “`java.lang.NumberFormatException: For input string: “Seleccione la cantidad de subredes”`” para el siguiente caso:

```
Exception in thread "AWT-EventQueue-0" java.lang.NumberFormatException: For input string: "Seleccione la cantidad de subredes"
at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
```

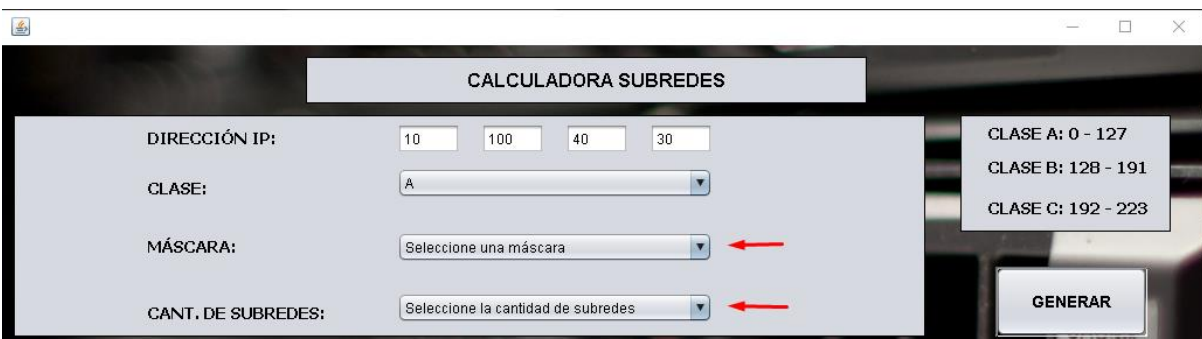
- Dejar los 3 combo box vacíos:



Se generaba la excepción “java.lang.NumberFormatException: For input String: “Seleccione una máscara”” para el siguiente caso:

```
Output - CalculadoraSubRedes (run) x Start Page x Interfaz.java x
Exception in thread "AWT-EventQueue-0" java.lang.NumberFormatException: For input string: "Seleccione una máscara"
at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
```

- Dejar vacíos los combobox “máscara” (prefijo de red) y “cantidad de subredes” para posteriormente presionar el botón “Generar”



Se generaba la excepción “java.lang.NumberFormatException: For input String: “Seleccione la cantidad de subredes”” para el siguiente caso:

```
Output - CalculadoraSubRedes (run) x Start Page x Interfaz.java x
run:
Exception in thread "AWT-EventQueue-0" java.lang.NumberFormatException: For input string: "Seleccione la cantidad de subredes"
at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
```

- Dejar vacío el combo box “cantidad de subredes” para posteriormente presionar el botón “Generar”

CALCULADORA SUBREDES

DIRECCIÓN IP:

CLASE:

MÁSCARA:

CANT. DE SUBREDES:

CLASE A: 0 - 127

CLASE B: 128 - 191

CLASE C: 192 - 223

Implementación de try catch (clase Interfaz)

```

319 public int getPrimerOcteto() {
320     int primerOct = 0;
321
322     try {
323         Integer.parseInt(primerOcteto.getText());
324         if (primerOct >= 255) {
325             primerOct = Integer.parseInt("Fuera de rango");
326         }
327     } catch (NumberFormatException ex) {
328         JOptionPane.showMessageDialog(null, "Error en el primer octeto de tipo:" + ex);
329     }
330
331     return primerOct;
332 }

```

Después de la implementación del uso de try catch, tenemos los siguientes mensajes de advertencia según se cumpla las excepciones mencionadas anteriormente:



Seleccionar una clase y sus métodos para realizar y aplicar pruebas de Test (JUnit).

Se tomó la decisión de aplicar las pruebas unitarias a la clase “CalculadoraIp”, ya que en dicha clase contamos con los métodos que consideramos son de vital importancia (manejo de operaciones con números binarios) para el proyecto y por lo tanto deseamos que se pueda verificar el correcto funcionamiento de nuestros métodos implementados en dicha clase.

Además, se aplicó Tests Parametrizados, en la cual contamos con una batería de 3 pruebas que consideramos las correctas para probar el correcto funcionamiento de nuestros métodos, que fueron las siguientes:

```
49  @Parameters
50  public static Iterable<Object[]> bateriaDePruebas(){
51
52      List<Object[]> bateriaP = Arrays.asList(new Object[][]{
53          //
54              PO SO TO CO CL BAE MASC SUBMASC ip
55              {10, 100, 40, 30, "A", 13, "111111111100000000000000000000", "111111111110000000000000000000", "000010
56              {128, 100, 40, 30, "B", 19, "111111111111110000000000000000", "111111111111110000000000000000", "100000
57              {192, 100, 40, 30, "C", 30, "11111111111111111111111100000000", "1111111111111111111111111100", "110000
58          });
59      return bateriaP;
60  }
```

```
49
50
51
52
53  .p bin P redBinario PORd SORd TORd CORd Broadcastbin
54  .010011001000010100000011110", 11, "00001010011000000000000000000000", "10", "96", "0", "0", "00001010011111111111
55  )000011001000010100000011110", 17, "10000000011001000000000000000000", "128", "100", "0", "0", "1000000001100100011111
56  )000011001000010100000011110", 25, "11000000011001000010100000000000", "192", "100", "40", "0", "1100000001100100001010
57
58
59
60
```

```
49
50
51
52
53  io PORd SORd TORd CORd Broadcastbin POBd SOBd TOBd COBd
54  11000000000000000000000000000000", "10", "96", "0", "0", "000010100111111111111111111111", "10", "127", "255", "255"},
55  01100100000000000000000000000000", "128", "100", "0", "0", "100000000110010001111111111111", "128", "100", "127", "255"},
56  11001000010100000000000000000000", "192", "100", "40", "0", "11000000011001000010100001111111", "192", "100", "40", "127"}
57
58
59
60
```


Se tiene un total de 10 métodos implementados en la clase CalculadoraIp, por lo que tenemos los siguientes test:

```
113 @Test
114 public void testDeterminarClase() {
115     System.out.println("determinarClase");
116
117     ➔ int[] ipDecimal = {primerOctIpDec, segundoOctIpDec, tercerOctIpDec, cuartoOctIpDec};
118     CalculadoraIp instance = new CalculadoraIp();
119     ➔ String expResult = "clase";
120     String result = instance.determinarClase(ipDecimal);
121     // TODO review the generated test code and remove the default call to fail.
122     if(!expResult.equals(result)){
123         fail("----- El resultado del metodo Determinar Clase no es correcto -----");
124     }
125     assertEquals(expResult, result);
126 }
```

```
131 @Test
132 public void testConvertirBinarioDecimal() {
133     System.out.println("convertirBinarioDecimal");
134     ➔ String octetoBin = IpBin.substring(0,8);
135     CalculadoraIp instance = new CalculadoraIp();
136     ➔ int expResult = primerOctIpDec;
137     int result = instance.convertirBinarioDecimal(octetoBin);
138     if(expResult!=(result)){
139         fail("----- El resultado del metodo Convertir Binario Decimal no es correcto -----");
140     }
141
142     assertEquals( expResult, result);
143     // TODO review the generated test code and remove the default call to fail.
144
145 }
```

```
150 @Test
151 public void testEncenderBits() {
152     System.out.println("encenderBits");
153     ➔ int bitsAEncender = this.bitsAEncender;
154     ➔ String mascaraBin = mascBin;//11
155     CalculadoraIp instance = new CalculadoraIp();
156     ➔ String expResult = mascBinEsperada;//13
157     String result = instance.encenderBits(bitsAEncender, mascaraBin);
158
159     // TODO review the generated test code and remove the default call to fail.
160     if(!expResult.equals(result)){
161         fail("----- El resultado del metodo Encender Bits no es correcto -----");
162     }
163
164     assertEquals(expResult, result);
165 }
```

```

170 @Test
171 public void testIpDecimalBinario() {
172     System.out.println("IpDecimalBinario");
173     int[] ipDecimal = {primerOctIpDec, segundoOctIpDec, tercerOctIpDec, cuartoOctIpDec};
174     CalculadoraIp instance = new CalculadoraIp();
175     String[] expResult = {IpBin.substring(0,8), IpBin.substring(8,16), IpBin.substring(16,24), IpBin.substring(24,32)};
176     String[] result = instance.ipDecimalBinario(ipDecimal);
177
178     // TODO review the generated test code and remove the default call to fail.
179
180     for (int i = 0; i < 3; i++) {
181         if(!expResult[i].equals(result[i])){
182             fail("----- El resultado del metodo Ip Decimal Binario no es correcto -----");
183         }
184     }
185     assertEquals(expResult, result);
186 }
187

```

```

192 @Test
193 public void testCovertirPrefijoRedBinOctPorOct() {
194     System.out.println("covertirPrefijoRedBinOctPorOct");
195     int prefijo = this.prefijo;
196     CalculadoraIp instance = new CalculadoraIp();
197     String[] expResult = {mascBin.substring(0,8), mascBin.substring(8,16), mascBin.substring(16,24),
198         mascBin.substring(24,32)};
199     String[] result = instance.covertirPrefijoRedBinOctPorOct(prefijo);
200     System.out.println("-----"+result[0]+"-----");
201     // TODO review the generated test code and remove the default call to fail.
202     for (int i = 0; i < 3; i++) {
203         if(!expResult[i].equals(result[i])){
204             fail("----- El resultado del metodo Convertir Prefijo Red Bin Oct por Oct no es correcto -----");
205         }
206     }
207 }
208
209     assertEquals(expResult, result);
210 }
211

```

```

215 @Test
216 public void testConcatenarMascaraBin() {
217     System.out.println("concatenarMascaraBin");
218     int prefijo = this.prefijo;
219     CalculadoraIp instance = new CalculadoraIp();
220     String expResult = mascBin;
221     String result = instance.concatenarMascaraBin(prefijo);
222     // TODO review the generated test code and remove the default call to fail.
223     if(!expResult.equals(result)){
224         fail("----- El resultado del metodo Concatenar Mascara Bin no es correcto -----");
225     }
226
227     assertEquals(expResult, result);
228 }
229

```

```

233 @Test
234 public void testCalcularRedBin() {
235     System.out.println("calcularRedBin");
236     int[] ipDec = {primerOctIpDec, segundoOctIpDec, tercerOctIpDec, cuartoOctIpDec};
237     int prefijo = this.prefijo;
238     CalculadoraIp instance = new CalculadoraIp();
239     String[] expResult = {RedBin.substring(0,8), RedBin.substring(8,16), RedBin.substring(16,24),
240         RedBin.substring(24,32)};
241     String[] result = instance.calcularRedBin(ipDec, prefijo);
242     // TODO review the generated test code and remove the default call to fail.
243     for (int i = 0; i < 3; i++) {
244         if(!expResult[i].equals(result[i])){
245             fail("----- El resultado del metodo Calcular Red Bin no es correcto -----");
246         }
247     }
248     assertEquals(expResult, result);
249 }
250

```

```

254 @Test
255 public void testCalcularRedDecimal() {
256     System.out.println("calcularRedDecimal");
257     int[] ipDec = {primerOctIpDec, segundoOctIpDec, tercerOctIpDec, cuartoOctIpDec};
258     int prefijo = this.prefijo;
259     CalculadoraIp instance = new CalculadoraIp();
260     String[] expResult = {primerOctRedDec, segundoOctRedDec, tercerOctRedDec, cuartoOctRedDec};
261     String[] result = instance.calcularRedDecimal(ipDec, prefijo);
262     // TODO review the generated test code and remove the default call to fail.
263     for (int i = 0; i < 3; i++) {
264         if(!expResult[i].equals(result[i])){
265             fail("----- El resultado del metodo Calcular Red Decimal no es correcto -----");
266         }
267     }
268     assertEquals(expResult, result);
269 }

```

```

274 @Test
275 public void testCalcularBroadCastBin() {
276
277     System.out.println("calcularBroadCastBin");
278     int[] ipDec = {primerOctIpDec, segundoOctIpDec, tercerOctIpDec, cuartoOctIpDec};
279     int prefijo = this.prefijo;
280     CalculadoraIp instance = new CalculadoraIp();
281     String[] expResult = {BroadCBin.substring(0,8), BroadCBin.substring(8,16), BroadCBin.substring(16,24),
282         BroadCBin.substring(24,32)};
283     String[] result = instance.calcularBroadCastBin(ipDec, prefijo);
284     // TODO review the generated test code and remove the default call to fail.
285     for (int i = 0; i < 3; i++) {
286         if(!expResult[i].equals(result[i])){
287             fail("----- El resultado del metodo Calcular BroadCast Bin no es correcto -----");
288         }
289     }
290
291     assertEquals(expResult, result);
292 }
293

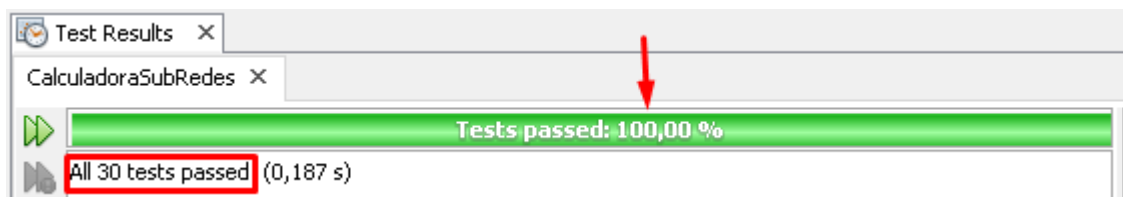
```

```

298 @Test
299 public void testCalcularBroadcastDec() {
300     System.out.println("calcularBroadcastDec");
301     int[] ipDec = {primerOctIpDec, segundoOctIpDec, tercerOctIpDec, cuartoOctIpDec};
302     int prefijo = this.prefijo;
303     CalculadoraIp instance = new CalculadoraIp();
304     String[] expResult = {primerOctBroadCDec, segundoOctBroadCDec, tercerOctBroadCDec, cuartoOctBroadCDec};
305     String[] result = instance.calcularBroadcastDec(ipDec, prefijo);
306
307     // TODO review the generated test code and remove the default call to fail.
308     for (int i = 0; i < 3; i++) {
309         if(!expResult[i].equals(result[i])){
310             fail("----- El resultado del metodo Calcular BroadCast Dec no es correcto -----");
311         }
312     }
313     assertEquals(expResult, result);
314 }
315
316 }

```

Ejecutando la clase CalculadoraIpTest.java y obtenemos los siguientes resultados:



Se puede observar que se ejecutaron 30 test, es decir, 10 test por cada prueba, y se obtiene un resultado del 100% de test que pasaron las pruebas.

Ahora, probemos con un pequeño cambio a nuestra batería de pruebas:

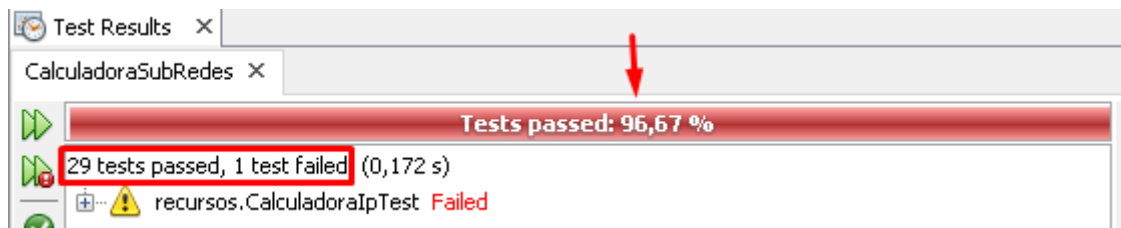
Antes:

```
POBd  SOBd  TOBd  COBd
"10", "127", "255", "255"},
"128", "100", "127", "255"},
"192", "100", "40", "127"
```

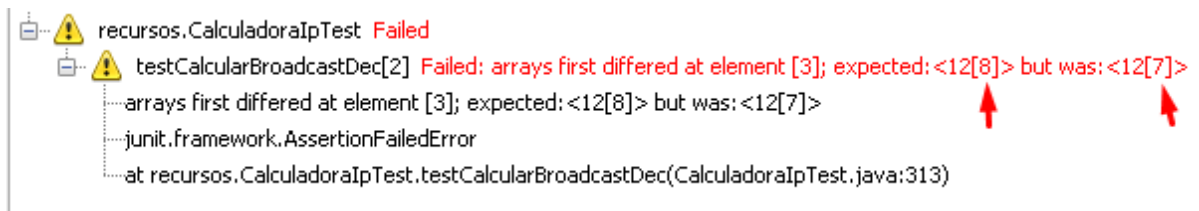
Después:

```
POBd  SOBd  TOBd  COBd
"10", "127", "255", "255"},
"128", "100", "127", "255"},
"192", "100", "40", "128"
```

Ejecutando nuevamente la clase CalculadoraIpTest.java se obtiene los siguientes resultados:



Se puede observar que 1 test no pasó las pruebas, esto se debe al cambio que se realizó a los test parametrizados como se ve a continuación:



Como se puede observar, la falla se encuentra en el parámetro que modificamos, con esto podemos concluir que, además de probar que nuestros métodos funcionan de manera correcta, podemos decir que nuestras pruebas parametrizadas (batería de pruebas) se adecua de buena manera a nuestros tests para la clase CalculadoraIP.