# TECNOLÓGICO NACIONAL DE MÉXICO

## INSTITUTO TECNOLÓGICO DE OAXACA

**ASIGNATURA:** FUNDAMENTOS DE TELECOMUNICACIONES

**CATEDRÁTICO:** MCC. VALVERDE JARQUÍN REYNA

**ALUMNO:** GARCÍA GARCÍA JOSÉ ÁNGEL

**GRUPO:** ISB

**HORA:** 12:00 – 13:00

**CARRERA:** INGENIERÍA EN SISTEMAS COMPUTACIONALES

**REPORTE DEL TRABAJO DE UNIDAD 2**

OAXACA DE JUÁREZ, OAX, 01/NOVIEMBRE/2020

# ÍNDICE GENERAL

# ÍNDICE DE IMÁGENES

# PLANTEAMIENTO DEL PROBLEMA

Para realizar la práctica de esta unidad, el catedrático solicitó la practica de forma textual en la plataforma Moodle, pero también lo hizo mediante una sesión de Meet en la que explicó a detalle lo que deseaba. La practica es sencilla y sólo consiste en realizar un programa que convierta un tren de bits en un polinomio y viceversa, es decir, dado un polinomio mostrar un tren de bits.

# DISEÑO DE LA SOLUCIÓN

:

Para dar solución al problema tenemos muchos caminos o maneras de hacerlo, bueno, esto dependiente a cada función del programa. El problema lo partí en dos caminos, ya que obviamente hay dos conversiones por hacer, entonces de esta forma es cómo se divide el programa. Cada conversión corresponde a un método o función. Las explicaré a detalle cada una y por separado:

***Conversión de un tren de bits a un polinomio***

Lo primero a realizar fue plantear cuales eran los datos de entrada y cuales deberían ser su salida. Ya luego nos enfocaríamos en lograr esa parte que dada la entrada no de la salida esperada. Entonces

*Entrada:* Una cadena de caracteres que corresponde a unos y ceros. Por ejemplo: 1010

*Salida*: Una cadena de caracteres que corresponde a un polinomio. Por ejemplo: x^2+1

De lo anterior, podemos deducir que tanto la entrada cómo la salida es del tipo de dato abstracto (TDA) de nombre String, entonces esto nos lleva a pensar que podemos hacer uso de los métodos de dicha clase.  Además, que la declaración del método de inicio a nivel código es:

```
public String bits_polinomio(String entrada){
    String polinomio = ""; // Variable para almacenar el polinomio
    return  polinomio;
}
```

*Ilustración 1 - Declaración del método*

Donde *bits_polinomio* es el nombre de nuestro método, el parámetro de nombre *entrada* hace referencia al tren de bits y la variable local *polinomio* almacenará el polinomio que vamos a regresar.

Para empezar, al analizar el problema, me di cuenta que el número de bits menos 1 indica el grado del polinomio, esto es de gran ayuda porque entonces podemos decir que existe un limite de bits para retornar.

Es decir

$$grado \ = \ numero\ de\ bits - 1 \ \ grado\ del\ polinomio$$

Esto se puede visualizar con el siguiente ejemplo:

$$1010 \qquad 4\ números\ de\ bits$$

$$grado \ = \ 4 \ - \ 1 \ = \ 3$$

$$x^3 + x \qquad Polinomio\ con\ grado\ 3$$

Y a nivel de código, el número de bits lo podemos obtener gracias a un método de String que se llama **length()** y luego restar 1 para obtener el grado, de tal forma que a nivel código tenemos:

```
public String bits_polinomio(String entrada){
    String polinomio = ""; // Variable para almacenar el polinomio
    int grado = entrada.length() - 1; // Grado del polinomio
    return  polinomio;
}
```

*Ilustración 2 - Calculando el grado del polinomio*

Obtenido el límite del polinomio, lo que podemos hacer ahora, es recorrer todo el String correspondiente al tren de bits, esto quiere decir que vamos a iniciar desde 0 hasta el grado del polinomio a generar. Esto establece la siguiente parte del código:

```
public String bits_polinomio(String entrada){
    String polinomio = ""; // Variable para almacenar el polinomio
    int grado = entrada.length() - 1; // Grado del polinomio
    for(int i = 0; i < grado; i++){ // Iterar desde 0 con limite el grado
    }
    return  polinomio;
}
```

*Ilustración 3 - Iteración del tren de bits*

Y en cada iteración vamos a verificar que bit es 1, que es lo mismo a decir que sea diferente de 0. y en el caso de que esto se cumpla, es porque entonces ahí debemos colocar una parte del polinomio y su respectivo exponente. Para extraer un bit del tren de bits hacemos uso del método **charAt()** de la clase String.

Entonces a nivel de código sería:

```
public String bits_polinomio(String entrada){
    String polinomio = ""; // Variable para almacenar el polinomio
    int grado = entrada.length() - 1; // Grado del polinomio
    for(int i = 0; i < grado; i++){ // Iterar desde 0 con limite el grado
        if(entrada.charAt(i) != '0') // Si el bit es 1
    }
    return  polinomio;
}
```

*Ilustración 4 - Comprobando si el bit es 1*

Entonces, lo que pude ser constante que agregamos al momento de iterar es el **x^** ya que esto no cambiará, sólo cambia el exponente. Para calcular el exponente lo que debemos hacer es lo siguiente mientras estamos iterando el tren de bits. Donde *i* es la variable de iteración que irá incrementando: $exponente = grado - i$

Y además que cuando la variable de iteración y el exponente sean iguales, lo que haremos es colocar ***x+*** porque esa posición es la que corresponde a dicho valor y ya no sería ***x^***.
A nivel de código sería cómo la siguiente imagen y nótese que hago uso de if incrustado.

```
public String bits_polinomio(String entrada){
    String polinomio = ""; // Variable para almacenar el polinomio
    int grado = entrada.length() - 1; // Grado del polinomio
    for(int i = 0; i < grado; i++){ // Iterar desde 0 con limite el grado
        if(entrada.charAt(i) != '0') // Si el bit es 1
          //Agregamos al polinomio su valor correspondiente
          polinomio += (i != grado - 1) ? "x^" + ( grado - i ) + "+" : "x+";
    }
    return  polinomio;
}
```

*Ilustración 5 - Agregando exponentes al polinomio*

Ahora supongamos que nos introduce un tren de bits de un solo bit, entonces no se hace la iteración con el for y el paso anterior mencionado ya que el grado es 0. Para eso verificamos si nuestra variable de polinomio está vacía, y si lo está es porque realmente no se uso en el for y eso nos indica que la entrada es de un solo bit y para ello retornamos “1” y finalizamos al momento. Implementándolo en nuestro método a nivel de código sería fácil con un if y aplicando el método **isEmpty()** de la clase String para verificar si está vacía la cadena.

```
public String bits_polinomio(String entrada){
    String polinomio = ""; // Variable para almacenar el polinomio
    int grado = entrada.length() - 1; // Grado del polinomio
    for(int i = 0; i < grado; i++){ // Iterar desde 0 con limite el grado
        if(entrada.charAt(i) != '0') // Si el bit es 1
          //Agregamos al polinomio su valor correspondiente
          polinomio += (i != grado - 1) ? "x^" + ( grado - i ) + "+" : "x+";
    }
    // Si el polinomio no tiene nada es porque sólo tiene un bit y es 1
    if(polinomio.isEmpty()) return "1";
    return  polinomio;
}
```

*Ilustración 6 - Validando tren de un sólo bit*

Hasta ahora sólo hemos contemplado para rellenar con x y sus respectivos exponentes, falta verificar si se agrega el 1 al final o no. Entonces debemos comprobar si nuestro tren de bits termina con 1 si es así agregamos **1+** y si es falso, no agregamos nada. Para verificar si el tren de bits termina con 1, usamos el método **endWith()** de la clase String.
Agregando esta parte a nuestro método:

```
public String bits_polinomio(String entrada){
    String polinomio = ""; // Variable para almacenar el polinomio
    int grado = entrada.length() - 1; // Grado del polinomio
    for(int i = 0; i < grado; i++){ // Iterar desde 0 con limite el grado
        if(entrada.charAt(i) != '0') // Si el bit es 1
          //Agregamos al polinomio su valor correspondiente
          polinomio += (i != grado - 1) ? "x^" + ( grado - i ) + "+" : "x+";
    }
    // Si el polinomio no tiene nada es porque sólo tiene un bit y es 1
    if(polinomio.isEmpty()) return "1";
    // Agregamos el 1+ si al final el tren de bits termina con 1
    polinomio += entrada.endsWith("1") ? "1+" : "";
    return  polinomio;
}
```

*Ilustración 7 - Verificar si agregamos al final el 1*

Cómo hemos visto, al formar nuestro polinomio, ya sea si termina en **x+, x^1,1+** siempre lo hará con un **+** al final, es por eso que al retornar el polinomio, usando el método **substring()** de la clase String, podemos quitar ese último **+** y retornar el polinomio ya formado de manera correcta.

Y ya tendríamos nuestro método terminado

```
public String bits_polinomio(String entrada){
    String polinomio = ""; // Variable para almacenar el polinomio
    int grado = entrada.length() - 1; // Grado del polinomio
    for(int i = 0; i < grado; i++){ // Iterar desde 0 con limite el grado
        if(entrada.charAt(i) != '0') // Si el bit es 1
          //Agregamos al polinomio su valor correspondiente
          polinomio += (i != grado - 1) ? "x^" + ( grado - i ) + "+" : "x+";
    }
    // Si el polinomio no tiene nada es porque sólo tiene un bit y es 1
    if(polinomio.isEmpty()) return "1";
    // Agregamos el 1+ si al final el tren de bits termina con 1
    polinomio += entrada.endsWith("1") ? "1+" : "";
    // Retornamos el   polinomio
    return polinomio.substring(0, polinomio.length() - 1);
}
```

*Ilustración 8 - Método terminado*

***Conversión de un polinomio a tren de bits***

Para dar solución a esta parte del programa, analizaremos la entrada y salida.

*Entrada*: Una cadena de caracteres que corresponde a un polinomio. Por ejemplo: x^2+1

*Salida:* Una cadena de caracteres que corresponde a unos y ceros. Por ejemplo: 101

Al igual que el método anterior, la entrada y salida son String, por lo que manipular su variable será sencillo y tendremos ventajas para crear el método. Empezamos por la declaración del método y su respectiva variable local.

```
public String polinomio_bits(String entrada){
    String bits = "";
    return  bits;
}
```

*Ilustración 9 - Declaración del segundo método*

Donde *polinomio_bits* es el nombre de nuestro método, el parámetro de nombre *entrada*

hace referencia al polinomio y la variable local bits almacenará los bits que

vamos a regresar.

Para empezar, la primera condición es que, si nuestro polinomio es sólo 1, entonces el bit igual es 1, por lo que implementamos y verificamos esto antes de crear las variables y realizar lo demás.

```
public String polinomio_bits(String entrada){
    // Evitamos uso de recursos y si es 1 simplemente regresamos 1
    if(entrada.equals("1")) return "1";
    String bits = "";
    return  bits;
}
```

*Ilustración 10 - Verificación de polinomio 1*

Ahora, cómo sabemos que tiene más de uno, entonces nuestro polinomio a fuerzas tiene que tener el signo **+** para unir dos monomios, entonces para obtener dichos monomios, vamos a usar un método de String que nos permite separar por carácter y dicho carácter es ese signo. Y eso lo almacenamos en un arreglo de nombre data

```
public String polinomio_bits(String entrada){
    // Evitamos uso de recursos y si es 1 simplemente regresamos 1
    if(entrada.equals("1")) return "1";
    String bits = "";
    String data[] = entrada.split("\\+"); // Partimos el polinomio en cada una de sus partes
    return  bits;
}
```

*Ilustración 11 - Partiendo el polinomio en monomios*

Ahora, de los monomios, cómo sabemos que el polinomio está ordenado al entrar, nos damos cuenta que el primer monomio va a tener el exponente más grande y que el tipo de dato de este es String, es necesario obtener este exponente ya que así vamos a limitar nuestro número de bits. Similar a cómo hicimos en el método anterior:

Es decir

$$numero\ de\ bits = \ grado\ + 1$$

Esto se puede visualizar con el siguiente ejemplo:

$$x^3 + x \quad Polinomio\ con\ grado\ 3$$

$$numero\ de\ bits\ = \ grado\ + 1\ = \ 4$$

$$1010 \qquad 4\ números\ de\ bits$$

Luego de obtener el monomio también hay que verificar si el primer monomio es **x** o **x^1** ya que este tendrá un valor por defecto de 1, entonces lo excluimos de los demás, para el resto, extraer el exponente consiste en aplicar el método **substring()**, que va desde la posición 2 para que así quitemos el **x^** y hasta el final de ese monomio, entonces usamos **length()**. Agregamos esa parte a nuestro método y así obtendríamos al exponente mayor.

```
public String polinomio_bits(String entrada){
    // Evitamos uso de recursos y si es 1 simplemente regresamos 1
    if(entrada.equals("1")) return "1";
    String bits = "";
    String data[] = entrada.split("\\+"); // Partimos el polinomio en cada una de sus partes
    // Obtenemos el primer exponente
    String primerExponente = data[0].equals("x") || data[0].equals("x^1") ? "1" : "" +
                             data[0].substring(2,data[0].length());
    return  bits;
}
```

*Ilustración 12 - Obtener exponente mayor.*

Ahora con ese exponente mayor, obtenemos el número de bits. Que será convertir ese exponente de String a un tipo de dato entero (int) y luego sumar 1.

```
public String polinomio_bits(String entrada){
    // Evitamos uso de recursos y si es 1 simplemente regresamos 1
    if(entrada.equals("1")) return "1";
    String bits = "";
    String data[] = entrada.split("\\+"); // Partimos el polinomio en cada una de sus partes
    // Obtenemos el primer exponente
    String primerExponente = data[0].equals("x") || data[0].equals("x^1") ? "1" : "" +
                             data[0].substring(2,data[0].length());
    int numBits = Integer.parseInt(primerExponente) + 1; // Obtenemos el número de bits
    return  bits;
}
```

*Ilustración 13 - Obteniendo número de bits*

Ahora, vamos a crear un arreglo de booleanos, que estos nos servirán para saber en qué posición si colocar un 1 y en cual no. Y también preparamos un for para iterar nuestro String de entrada, ya que ahí va estar la otra parte.

Recorreremos suponiendo que están presente todos los bits en 1, entonces eso nos da cómo máximo el numero de bits obtenidos. Entonces vamos desde 0 hasta el número de bits.

```
public String polinomio_bits(String entrada){
    // Evitamos uso de recursos y si es 1 simplemente regresamos 1
    if(entrada.equals("1")) return "1";
    String bits = "";
    String data[] = entrada.split("\\+"); // Partimos el polinomio en cada una de sus partes
    // Obtenemos el primer exponente
    String primerExponente = data[0].equals("x") || data[0].equals("x^1") ? "1" : "" +
                             data[0].substring(2,data[0].length());
    int numBits = Integer.parseInt(primerExponente) + 1; // Obtenemos el número de bits
    boolean [] pos = new boolean[numBits]; // Arreglo para verificar donde colocar bits 1
    for(int i = 0; i < numBits; i++) {
    }
    return  bits;
}
```

*Ilustración 14 - Arreglo de soporte para bits*

Bien, cómo es una suposición lo que hacemos, debemos ser consciente de que, si no existe cierta posición en nuestro arreglo de monomios, entonces obtendríamos un error. Para eso usamos un bloque try-catch y en el realizamos las demás acciones.

Vamos a obtener el exponente, del monomio correspondiente al de la posición *i* y luego vamos a convertir a entero. Indicamos que ahí hay un exponente para luego colocar en esa posición un bit 1.

```
public String polinomio_bits(String entrada){
    // Evitamos uso de recursos y si es 1 simplemente regresamos 1
    if(entrada.equals("1")) return "1";
    String bits = "";
    String data[] = entrada.split("\\+"); // Partimos el polinomio en cada una de sus partes
    // Obtenemos el primer exponente
    String primerExponente = data[0].equals("x") || data[0].equals("x^1") ? "1" : "" +
                             data[0].substring(2,data[0].length());
    int numBits = Integer.parseInt(primerExponente) + 1; // Obtenemos el número de bits
    boolean [] pos = new boolean[numBits]; // Arreglo para verificar donde colocar bits 1
    for(int i = 0; i < numBits; i++) {
      try {
            String conv = data[i].equals("x") || data[i].equals("x^1") ? "1" : "" +
                          data[i].substring(2,data[i].length()); // Obtenemos el exponente
            int value = Integer.parseInt(conv); // Convertimos el exponente a un entero
            pos[value] = true; // Indicamos que aquí si hay un exponente
          } catch (Exception e){
      }
    }
    return  bits;
}
```

*Ilustración 15 - Obteniendo posición de bit 1*

Ahora, dentro del catch lo que haremos es aprovechar y verificar si nuestro último monomio termina con **x** o con **1** para indicar su respectiva posición en el arreglo de boléanos para luego colocar un bit 1 en su posición.

```
public String polinomio_bits(String entrada){
    // Evitamos uso de recursos y si es 1 simplemente regresamos 1
    if(entrada.equals("1")) return "1";
    String bits = "";
    String data[] = entrada.split("\\+"); // Partimos el polinomio en cada una de sus partes
    // Obtenemos el primer exponente
    String primerExponente = data[0].equals("x") || data[0].equals("x^1") ? "1" : "" +
                             data[0].substring(2,data[0].length());
    int numBits = Integer.parseInt(primerExponente) + 1; // Obtenemos el número de bits
    boolean [] pos = new boolean[numBits]; // Arreglo para verificar donde colocar bits 1
    for(int i = 0; i < numBits; i++) {
      try {
            String conv = data[i].equals("x") || data[i].equals("x^1") ? "1" : "" +
                          data[i].substring(2,data[i].length()); // Obtenemos el exponente
            int value = Integer.parseInt(conv); // Convertimos el exponente a un entero
            pos[value] = true; // Indicamos que aquí si hay un exponente
          } catch (Exception e){
              if(data[data.length - 1].equals("x")) // Comprobamos si termina con x
                  pos[1] = true;
              if(data[data.length - 1].equals("1")) // Comprobamos si termina con 1
                  pos[0] = true;
          }
    }
    return  bits;
}
```

*Ilustración 16 - Considerando x y 1*

Por último, recorremos todo nuestro arreglo de posiciones que son booleanos, en donde el valor sea true es porque vamos a colocar un bit 1 y un bit 0 si el valor es false, estos los iremos agregando a nuestra variable de bits para finalmente retornar y terminar el método.

```
public String polinomio_bits(String entrada){
    // Evitamos uso de recursos y si es 1 simplemente regresamos 1
    if(entrada.equals("1")) return "1";
    String bits = "";
    String data[] = entrada.split("\\+"); // Partimos el polinomio en cada una de sus partes
    // Obtenemos el primer exponente
    String primerExponente = data[0].equals("x") || data[0].equals("x^1") ? "1" : "" +
                             data[0].substring(2,data[0].length());
    int numBits = Integer.parseInt(primerExponente) + 1; // Obtenemos el número de bits
    boolean [] pos = new boolean[numBits]; // Arreglo para verificar donde colocar bits 1
    for(int i = 0; i < numBits; i++) {
      try {
            String conv = data[i].equals("x") || data[i].equals("x^1") ? "1" : "" +
                          data[i].substring(2,data[i].length()); // Obtenemos el exponente
            int value = Integer.parseInt(conv); // Convertimos el exponente a un entero
            pos[value] = true; // Indicamos que aquí si hay un exponente
          } catch (Exception e){
              if(data[data.length - 1].equals("x")) // Comprobamos si termina con x
                  pos[1] = true;
              if(data[data.length - 1].equals("1")) // Comprobamos si termina con 1
                  pos[0] = true;
          }
    }
    // Creamos el arreglo de bits respectoa al arreglo de booleanos
    for(int i = pos.length - 1; i >= 0; i--) bits += pos[i] ? "1" : "0";
   return  bits;
}
```

*Ilustración 17 - Método terminado*

## HERRAMIENTAS UTILIZADAS

- Computadora con Sistema Operativo Windows.
- JDK 1.8 para compilar y ejecutar programas .java
- Blue J

## CÓDIGO

```
/**
 * Proyecto de Fundamentos de Telecomunicaciones de la Unidad 2.
 *
 * @author García García José Ángel
 * @version 25/10/2020
 */

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;

public class ProyectoUnidad2 {

    public static void main(String[] args) {
        Ventana v = new Ventana();
    }

}
```

```
class Ventana extends JFrame{


    private JLabel etqEntrada, resultado, creditos;
    private JTextArea txtEntrada;
    private JButton btnCalcular, btnBorrar;


    public  Ventana(){
        super("Fundamentos de Telecomunicaciones - Proyecto de Unidad 2");
        setVisible(true);
        setResizable(false);
        setContentPane(getElements());
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        setSize(700, 500);
        setLocationRelativeTo(null);
    }


    public String polinomio_bits(String entrada){
        if(entrada.equals("1")) return "1"; // Evitamos uso de recursos y si es 1 simplemente
regresamos 1
        String bits = "";
        String data[] = entrada.split("\\+"); // Partimos el polinomio en cada una de sus partes
        String primerExponente = data[0].equals("x") || data[0].equals("x^1") ? "1" : "" +
data[0].substring(2,data[0].length()); // Obtenemos el primer exponente
        int numBits = Integer.parseInt(primerExponente) + 1; // Obtenemos el número de bits
        boolean [] pos = new boolean[numBits]; // Arreglo para verificar donde colocar bits 1
        for(int i = 0; i < numBits; i++) {
            try {
                String conv = data[i].equals("x") || data[i].equals("x^1") ? "1" : "" +
data[i].substring(2,data[i].length()); // Obtenemos el exponente
```

```
                int value = Integer.parseInt(conv); // Convertimos el exponente a un entero
                pos[value] = true; // Indicamos que aquí si hay un exponente
            }catch (Exception e){
                if(data[data.length - 1].equals("x")) // Comprobamos si termina con x
                    pos[1] = true;
                if(data[data.length - 1].equals("1")) // Comprobamos si termina con 1
                    pos[0] = true;
            }
        }
        for(int i = pos.length - 1; i >= 0; i--) bits += pos[i] ? "1" : "0"; // Creamos el arreglo de bits respectoa al arreglo de booleanos
        return  bits;
    }

    public String bits_polinomio(String entrada){
        String polinomio = ""; // Variable para almacenar el polinomio
        int grado = entrada.length() - 1; // Grado del polinomio
        for(int i = 0; i < grado; i++) // Iterar desde 0 con limite el grado
            if(entrada.charAt(i) != '0') // Si el bit es 1
                polinomio += (i != grado - 1) ? "x^" + ( grado - i ) + "+" : "x+"; // Agregamos al polinomio su valor correspondiente
        if(polinomio.isEmpty()) return "1"; // Si el polinomio no tiene nada es porque sólo tiene un bit y es 1
        polinomio += entrada.endsWith("1") ? "1+" : ""; // Agregamos el 1+ si al final el tren de bits termina con 1
        return polinomio.substring(0, polinomio.length() - 1); // Retornamos el polinomio
    }

    public JPanel getElements(){
```

```
        SpringLayout s = new SpringLayout();
        JPanel panel = new JPanel(s);
        panel.setBackground(Color.GRAY);
        etqEntrada = new JLabel("Introduce (tren de bits /
polinomio)",SwingConstants.CENTER);
        etqEntrada.setFont(new Font("Sylfaen", Font.BOLD, 20));
        txtEntrada = new JTextArea();
        txtEntrada.setAlignmentX(SwingConstants.CENTER);
        panel.add(etqEntrada);
        s.putConstraint(SpringLayout.WEST, etqEntrada, 70, SpringLayout.WEST, panel);
        s.putConstraint(SpringLayout.EAST, etqEntrada, -70, SpringLayout.EAST, panel);
        s.putConstraint(SpringLayout.NORTH, etqEntrada, 100, SpringLayout.NORTH, panel);

        panel.add(txtEntrada);
        s.putConstraint(SpringLayout.WEST, txtEntrada, 100, SpringLayout.WEST, panel);
        s.putConstraint(SpringLayout.EAST, txtEntrada, -100, SpringLayout.EAST, panel);
        s.putConstraint(SpringLayout.NORTH, txtEntrada, 20, SpringLayout.SOUTH,
etqEntrada);
        s.putConstraint(SpringLayout.SOUTH, txtEntrada, -290, SpringLayout.SOUTH, panel);

        btnCalcular = new JButton("Calcular");
        btnCalcular.setFont(new Font("Sylfaen", Font.BOLD, 20));
        panel.add(btnCalcular);
        s.putConstraint(SpringLayout.WEST, btnCalcular, 220, SpringLayout.WEST, panel);
        s.putConstraint(SpringLayout.NORTH, btnCalcular, 40, SpringLayout.SOUTH,
txtEntrada);

        btnBorrar = new JButton("Borrar");
        btnBorrar.setFont(new Font("Sylfaen", Font.BOLD, 20));
```

```
        btnBorrar.setActionCommand("Borrar");
        panel.add(btnBorrar);
        s.putConstraint(SpringLayout.WEST, btnBorrar, 20, SpringLayout.EAST, btnCalcular);
        s.putConstraint(SpringLayout.NORTH, btnBorrar, 40, SpringLayout.SOUTH, txtEntrada);

        resultado = new JLabel("------------------------------");
        resultado.setFont(new Font("Sylfaen", Font.PLAIN, 20));
        resultado.setHorizontalAlignment(SwingConstants.CENTER);
        resultado.setBackground(Color.RED);
        panel.add(resultado);
        s.putConstraint(SpringLayout.WEST, resultado, 20, SpringLayout.WEST, panel);
        s.putConstraint(SpringLayout.EAST, resultado, -20, SpringLayout.EAST, panel);
        s.putConstraint(SpringLayout.NORTH, resultado, 40, SpringLayout.SOUTH,
btnCalcular);
        s.putConstraint(SpringLayout.SOUTH, resultado, -150, SpringLayout.SOUTH, panel);

        creditos = new JLabel("García García José Ángel");
        creditos.setFont(new Font("Sylfaen", Font.PLAIN, 20));
        panel.add(creditos);
        s.putConstraint(SpringLayout.EAST, creditos, -20, SpringLayout.EAST, panel);
        s.putConstraint(SpringLayout.SOUTH, creditos, -30, SpringLayout.SOUTH, panel);

        Controlador c = new Controlador();
        txtEntrada.addKeyListener(c);
        btnCalcular.addActionListener(c);
        btnBorrar.addActionListener(c);
        return panel;
    }
```

```
class Controlador implements ActionListener, KeyListener {

    @Override
    public void actionPerformed(ActionEvent e) {

        if(e.getActionCommand().equals("Borrar")){
            resultado.setText("------------------------------");
            txtEntrada.setText("");
            return;
        }
        String entrada = txtEntrada.getText();
        if ( entrada.isEmpty() ||
             entrada.contains("x^0") || entrada.equals("0") ||
             (!entrada.contains("x^") && !entrada.replace("1","").replace("0","").isEmpty())
        ) return;

        String salida = (!entrada.contains("x^") && !entrada.equals("x+1") ) ?
                        bits_polinomio(entrada) :
                        polinomio_bits(entrada);
        resultado.setText(salida);
        resultado.setHorizontalAlignment(SwingConstants.CENTER);
    }

    @Override
    public void keyTyped(KeyEvent e) {
        char car = e.getKeyChar();
            if((car<'0' || car>'9') && (car<'x' || car>'x') && (car<'^' || car>'^') && (car<'+' || car>'+'))
```

```
                e.consume();
        }

        @Override
        public void keyPressed(KeyEvent e) {}

        @Override
        public void keyReleased(KeyEvent e) {}
    }
}
```

# PRUEBA DE ESCRITORIO

♦ Convertir el tren de bits **1010** a polinomio.

entrada = “1010”;

polinomio = “”;

entrada.length() = 4;

grado = 4 – 1 = 3

Entramos al for y lo simularemos mediante la siguiente tabla:

| i | charAt(i) | grado – i | ¿i != 2? | ¿charAt(i) != 0? | polinomio |
|---|---|---|---|---|---|
| **0** | 1 | 3 | Si | Si | “x^3+” |
| **1** | 0 | 2 | Si | No | “x^3+” |
| **2** | 1 | 1 | No | Si | “x^3+x+” |

Salimos del for.

¿polinomio está vacío? No.

¿entrada (“1010”) termina con 1? No

Retorna polinomio = “x^3+x”

♦ Convertir el polinomio **x^3+x** a tren de bits.

entrada = x^3+x

¿entrada es igual a 1? No

bits = “”

data = [“x^3”,”x”]

primerExponente = ¿x^3 es igual a x? ¿x^3 es igual a x^1? No, entonces es “3”

numBits = primerExponente + 1 = 3 + 1 = 4

pos <- de longitud 4

Ahora entramos al primer for.

| i | conv | value | ¿Error? | ¿x igual a x? | ¿x igual a 1? | pos |
|---|---|---|---|---|---|---|
| **0** | “3” | 3 | No | -- | -- | pos[3] = true |
| **1** | “1” | 1 | No | -- | -- | pos[1] = true |
| **2** | -- | ---- | Si | Si | No | pos[1] = true |
| **3** | -- | -- | Si | Si | No | pos[1] = true |

Realizamos el segundo for con pos = [false,true,false,true]

| i | pos[i] | ¿pos[i] == true? | bits |
|---|---|---|---|
| **3** | true | Si | “1” |
| **2** | false | No | “10” |
| **1** | true | Si | “101” |
| **0** | false | No | “1010” |

Finalmente se obtiene bits = “1010” y es lo que se retorna.

# RESULTADOS OBTENIDOS

La interfaz gráfica que presento es la siguiente, en la que sin importar que deseas convertir, el programa es capaz de saber cuando convertir un tren de bits o un polinomio.

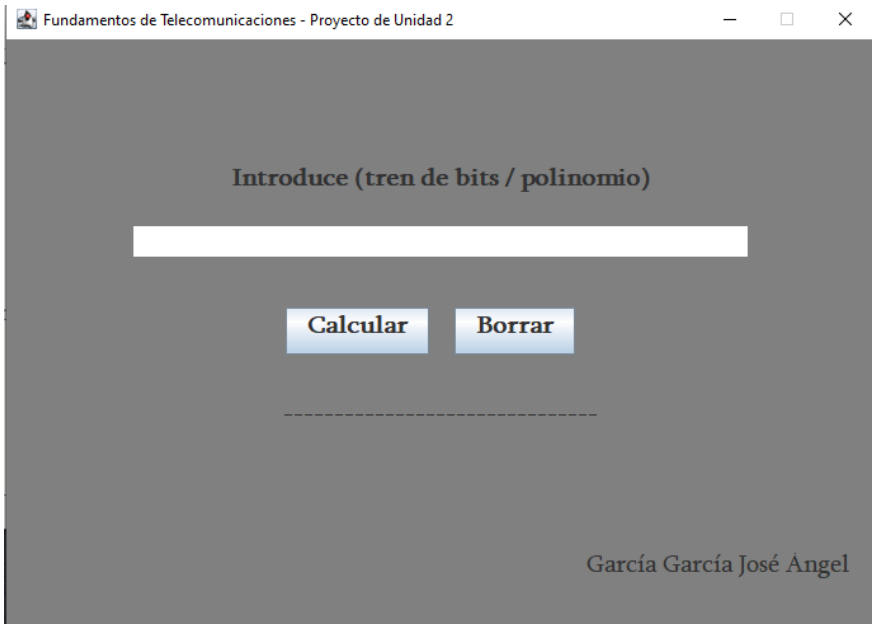


*Ilustración 18 - Interfaz gráfica*

Ahora voy a realizar unas pruebas convirtiendo dado un tren de bits.

- Dado el tren de bits **[11]**

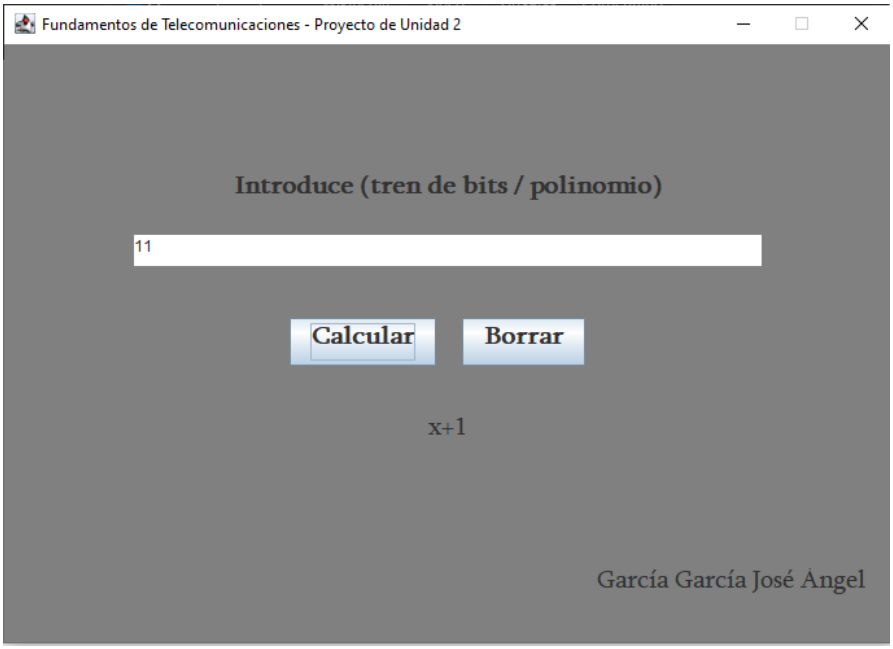


*Ilustración 19 - Conversión del tren de bits [11]*

- Dado el tren de bits **[1010]**

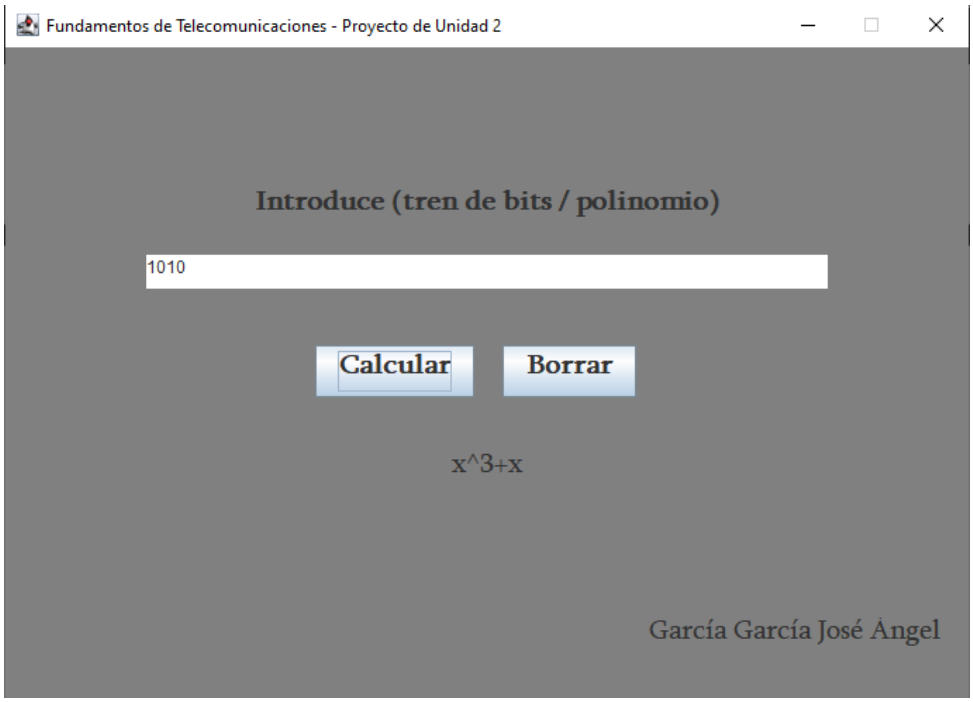


*Ilustración 20 - Conversión del tren de bits [1010]*

- Dado el tren de bits **[1000001]**

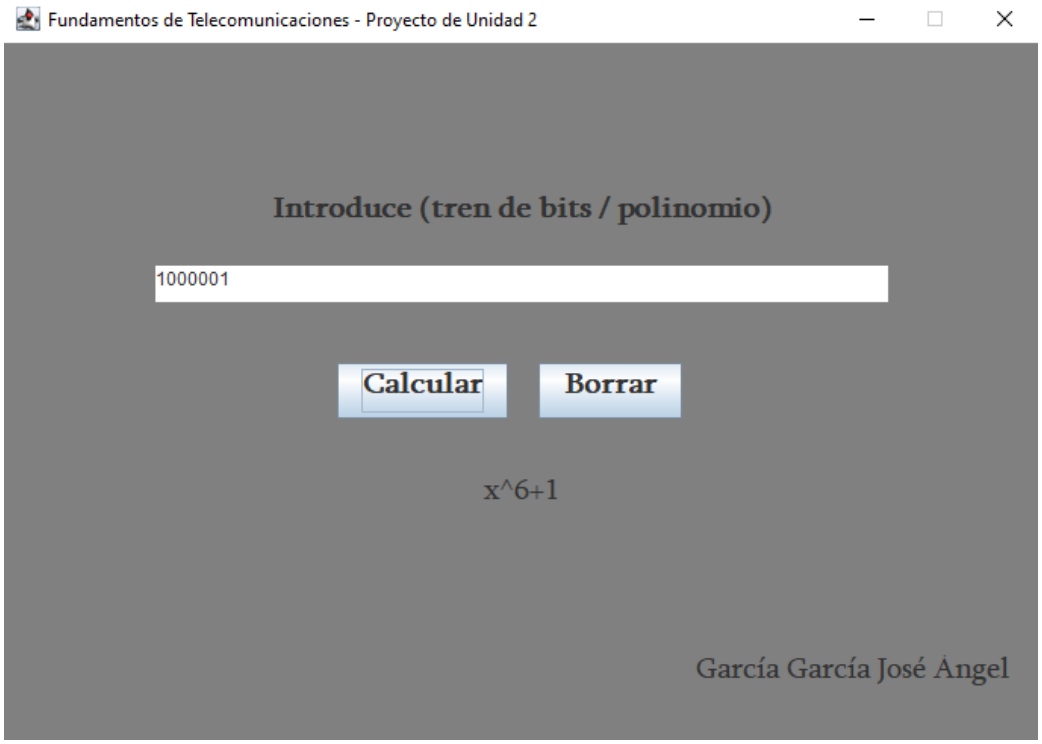


*Ilustración 21 - Conversión del tren de bits [1000001]*

- Dado el tren de bits **[11111111]**

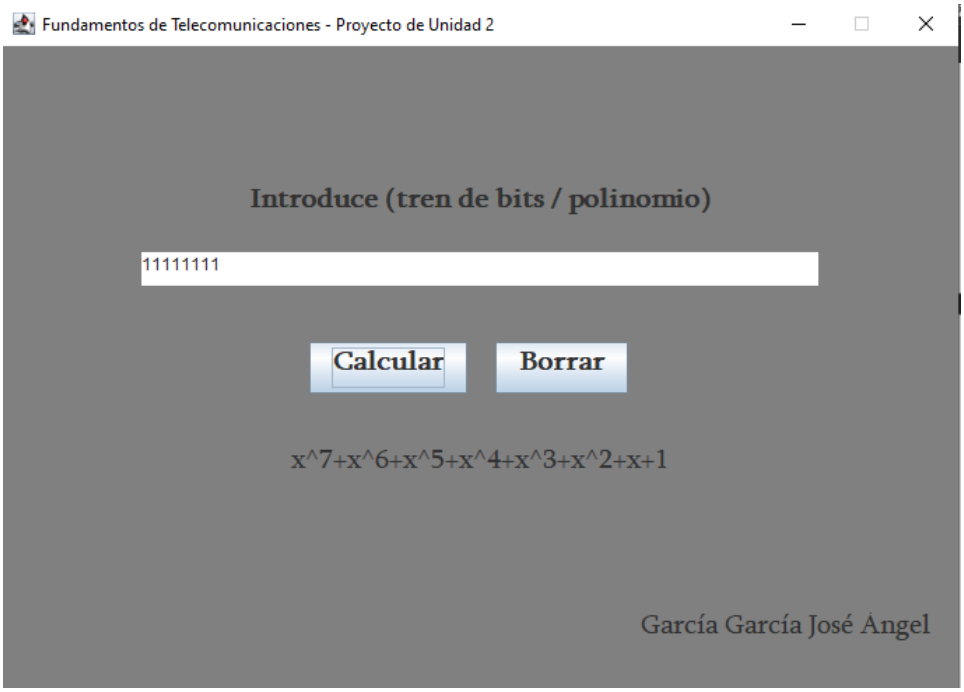


*Ilustración 22 - Conversión del tren de bits [11111111]*

- Dado el tren de bits **[11000000000000000001]**

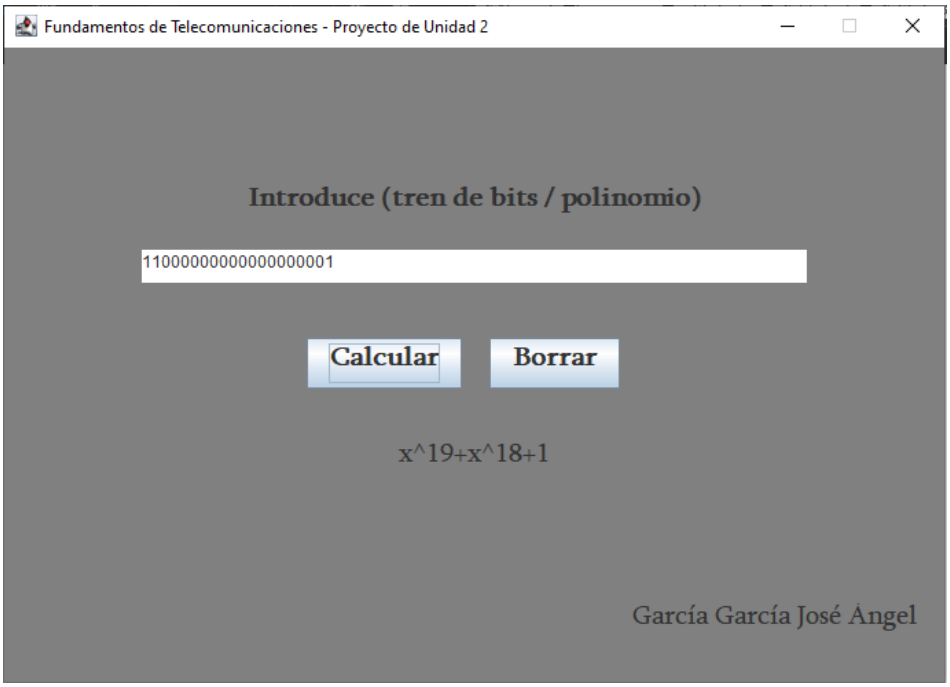


*Ilustración 23 - Conversión del tren de bits [11000000000000000001]*

Ahora vamos a realizar lo inverso, dado unos polinomios que nos muestre el tren de bits,

- Primero, voy a probarlo con el polinomio mostrado en el libro.
  El polinomio es ***x^7+x^5+x^2+x+1*** y su tren de bits debe ser **10100111**

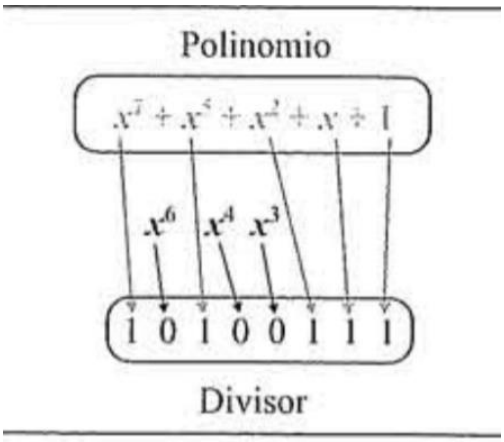


*Ilustración 24 - Polinomio tomado del libro*

Comprobamos esto con nuestro programa, verificando así que está bien elaborado:

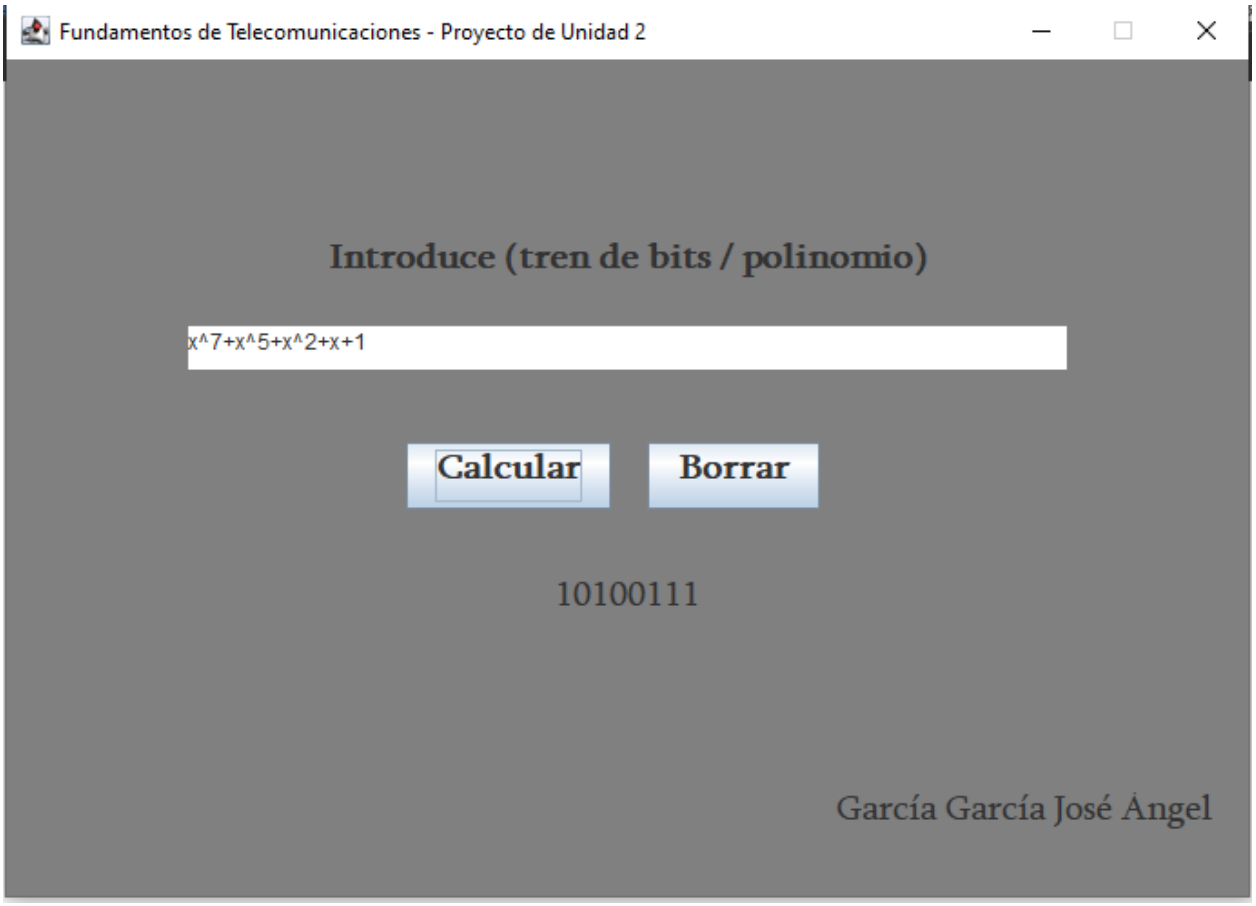


*Ilustración 25 - Conversión del polinomio x^7+x^5+x^2+x+1*

- Dado el polinomio **x^2+1**:

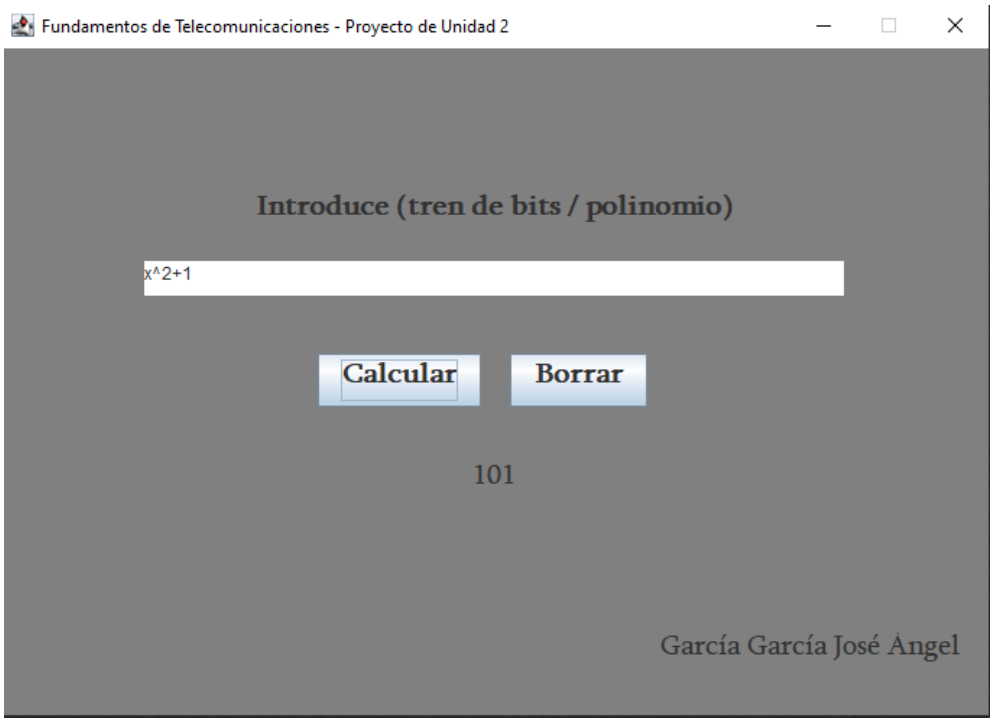


*Ilustración 26 - Conversión del polinomio x^2+1*

- Dado el polinomio **x^5+x^3+1**:

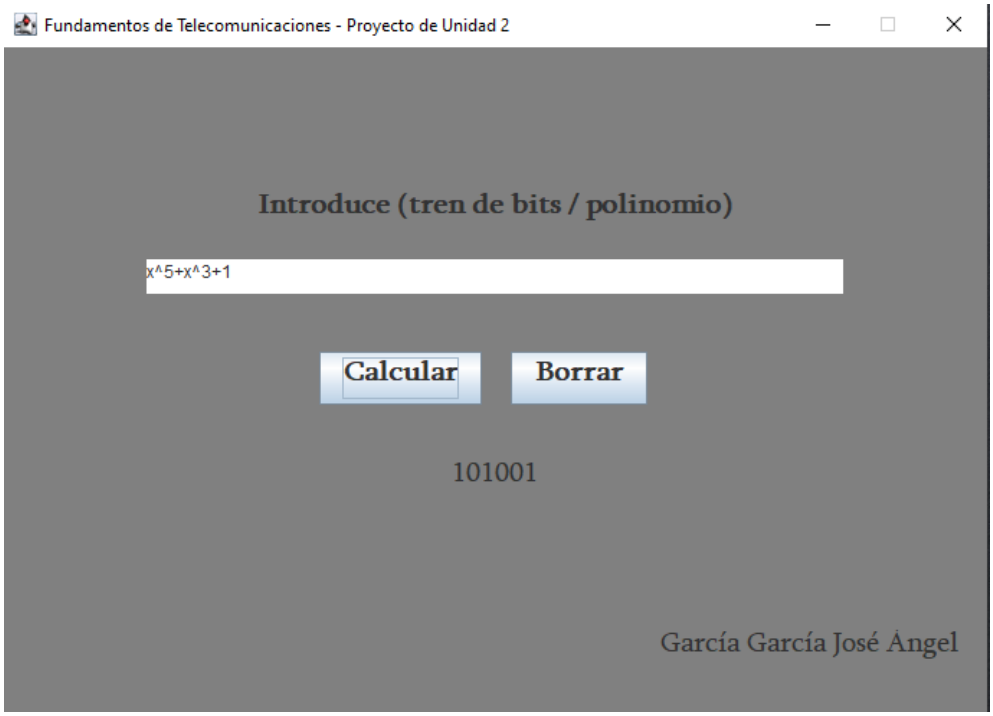


*Ilustración 27 - Conversión del polinomio x^5+x^3+1*

- Dado el polinomio **x^10+x^4+x^2+1**:

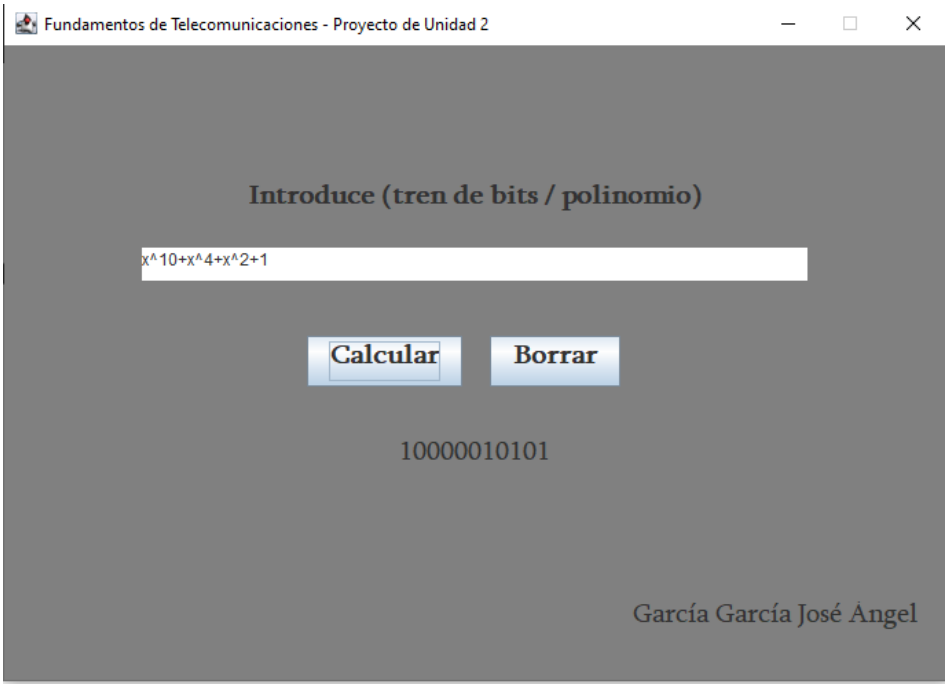


*Ilustración 28 - Conversión del polinomio x^10+x^4+x^2+1*

- Dado el polinomio **x^30+x^2**:

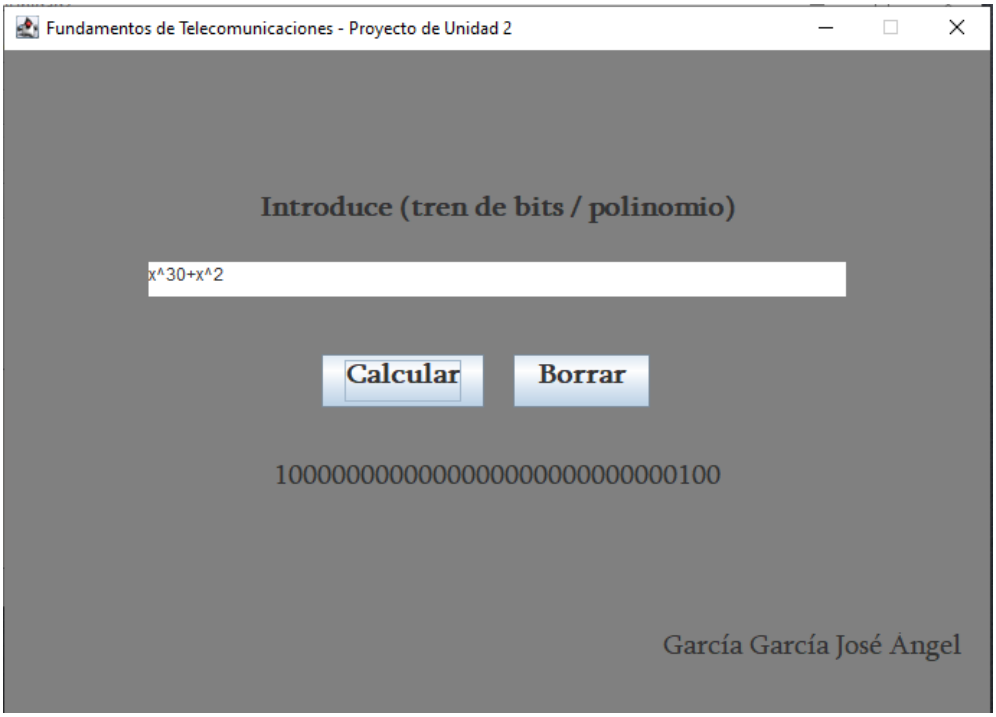


*Ilustración 29 - Conversión del polinomio x^30+x^2*

# ENLACE DEL VÍDEO

El vídeo en el que se explica a detalle cada una de las fases del programa desde su desarrollo hasta su implementación la puede visualizar en el siguiente enlace:

https://youtu.be/DbNpHVz4fII

Disculpe si el vídeo es muy largo, me emocioné al momento de explicar que no noté que había tardado tanto tiempo, ya que es una actividad que me agrada hacer entonces disfruto mucho hacer esto, trato de ser lo más claro al explicar, espero y si lo pueda entender.

Le dejo una tabla de las partes del vídeo para que le sea más fácil hacer la revisión.

| Minuto | Acción |
|---|---|
| **00:52** | Descripción del problema |
| **01:20** | Muestra de la interfaz |
| **01:30** | Pruebas |
| **06:25** | Explicación del código de forma general |
| **08:40** | Explicación de método de conversión de tren de bits a polinomio |
| **16:20** | Prueba de escritorio del primer método |
| **23:10** | Explicación del método de conversión de polinomio a tren de bits |
| **35:00** | Prueba de escritorio del segundo método |

# CONCLUSIÓN

Cómo mencioné al principio la practica no estaba complicada de realizar, básicamente se intenta hacer la simulación del CRC porque en realidad no se tomó en cuenta la parte de validación que hace el CRC, intenté realizar dichas validaciones, pero eran complicadas realizar a nivel código y al final decidí no agregarlas. Pero, por otra parte, se cumple de forma satisfactoria con lo solicitado que es lo planteado al inicio de este documento.

La parte tal vez un poco más compleja de hacer era la de dado un polinomio mostrar el tren de bits, esta parte fue la que me costo más analizar y a la que le dediqué la mayor parte del tiempo, ya que primero la había hecho, pero fallaba para polinomios de grado mayor a 9, después de realizar varios intentos lo conseguí y pude cumplir con esa parte del programa. Porque realizar la conversión de un tren de bits a polinomio fue muy sencilla de hacer, no había gran complejidad en su desarrollo.

La interfaz gráfica que presenté, es sencilla, no fue complicado de hacer porque ya he cursado una materia enfocada a eso además que el lenguaje en el que se desarrolla el programa es el que he usado durante todo lo que llevo de la carrera y se agradece que se siga usando el mismo, así se facilitan algunas partes, por ejemplo, esta de hacer interfaz gráfica.

Realizar esta practica me agradó mucho porque al inicio, supuse que iba ser muy sencilla y pensaba que la terminaría muy rápido, pero al desarrollarla note un poco la complejidad, entonces me llevó a pensar que problemas que parecen sencillos terminan siendo complejos. Claro, con esta práctica también me di a la tarea de investigar más sobre los CRC e intenté aplicar algunas propiedades de este en el programa.

Estoy satisfecho con lo que realicé y mostré en este trabajo, además que me recalcó mucho la importancia de los números binarios en cualquier parte de la tecnología, en este caso eran utilizados para indicar cuando posicionar una parte del polinomio, pero personalmente admiro mucho lo que se hace con sólo dos números, 1 y 0.